UNIVERSIDAD DE GRANADA

Departamento de Arquitectura y Tecnología de Computadores

ENTORNO PARALELO DE MODELADO Y SIMULACIÓN BAJO MATLAB. APLICACIÓN A SISTEMAS DE VISIÓN

TESIS DOCTORAL

Francisco Javier Fernández Baldomero

Granada, Septiembre 2001

A mi familia

Reconocimientos

Para realizar este trabajo se utilizó el cluster *oxígeno* adquirido por el proyecto CICYT TIC97-1149, y cuya instalación fue realizada por mi compañero Antonio Díaz, que además colabora en su mantenimiento.

La actualización del ordenador personal de trabajo fue asumida por el proyecto CICYT TAP97-1166.

El Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada proporcionó el software comercial necesario y cofinanció con la Junta de Andalucía la asistencia a los Congresos en donde se ha publicado parte del trabajo presentado en esta memoria.

Índice General

0.1 Resumen del trabajo presentado 2
0.2 Visión por Computador Bioinspirada 3
0.3 Computación Paralela y clusters de computadores 5
0.4 Estructura de los capítulos 7

1 Introducción **9**
1.1 El sistema PVM 10
1.1.1 Empaquetamiento 11
1.1.2 Encaminamiento 13
1.1.3 Grupos Dinámicos 13
1.2 El sistema LAM 14
1.2.1 Homogeneidad 15
1.2.2 Modo *daemon* o *cliente-a-cliente* 16
1.2.3 Grupos Estáticos 17
1.3 El entorno MATLAB 17
1.3.1 Oportunidad del trabajo presentado 19
1.3.2 Trabajos relacionados 22
1.4 PVMTB 24
1.5 MPITB 25
1.6 Comparación 26
1.6.1 Programas C 33
1.6.2 Programas MATLAB 43
1.7 Conclusiones 71

2 Análisis de Prestaciones **75**
2.1 Equipo utilizado 75
2.2 Eficiencia de un sistema de paso de mensajes 80
2.2.1 Obtención de mediciones reproducibles 82
2.2.2 Modelos lineal y afín 82
2.3 Modelos basados en la MTU 87
2.3.1 Interfaz gráfico 92
2.3.2 Ficheros de mínimos PVM 96
2.3.3 Ficheros de mínimos LAM 102
2.3.4 Modelo 0 108
2.3.5 Modelo 1 123

2.3.6 Determinación de los parámetros de cabecera 132
2.3.7 Modelo 2 144
2.3.8 Modelo 3 160
2.4 Conclusiones 175

3 Estudio de PVMTB 177
3.1 Elecciones de diseño 178
3.1.1 Compilación dinámica de PVM 179
3.2 Patrones de llamada PVM 181
3.3 Discusión y detalles 183
3.3.1 Patrones de llamada generales (P–*) 187
3.4 Eficiencia de PVMTB 192
3.4.1 Test *ping-pong* en C 192
3.4.2 Test *ping-pong* en MATLAB 204
3.5 Comparación 212
3.6 Conclusiones 216

4 Estudio de MPITB 219
4.1 Elecciones de diseño 220
4.2 Categorías de llamada MPI 222
4.3 Discusión y detalles 223
4.3.1 Categoría general MPI 230
4.3.2 Categoría punto a punto SndRecv 234
4.4 Eficiencia de MPITB 237
4.4.1 Test *ping-pong* en C 238
4.4.2 Test *ping-pong* en MATLAB 245
4.5 Comparación 250
4.6 Comparación entre PVMTB y MPITB 257
4.7 Conclusiones 258

5 Ejemplo de Aplicación: Análisis Wavelet para Sistemas de Visión 263
5.1 Análisis *wavelet* y su relación con Sistemas de Visión 264
5.2 Análisis *wavelet* secuencial 265
5.2.1 Código MATLAB secuencial 269
5.2.2 Código MATLAB secuencial para imagen en color 275
5.3 Análisis *wavelet* paralelo para imágenes B/W 279
5.3.1 Versión paralela B/W PVMTB 282
5.3.2 Versión paralela B/W MPITB 287
5.4 Algoritmo embarazosamente paralelo 291
5.4.1 Versión paralela RGB PVMTB 293
5.4.2 Versión paralela RGB MPITB 296
5.5 Discusión de los resultados 299
5.6 Conclusiones 300

6 Conclusiones **303**
6.1 Trabajo futuro 305
6.2 Méritos destacables de PVMTB y MPITB 305
6.3 Epílogo 307

A Sobre el cálculo paralelo de π **309**
A.1 Discusión 309
A.2 Listados 313

B Modelos para el tiempo de transmisión **341**
B.1 Introducción 341
B.2 Modelo 0 341
B.3 Modelo 1 343
B.4 Modelo 2 345
B.5 Modelo 3 351

C Programación MEX de PVMTB **355**
C.1 Introducción 355
C.2 Funciones de gestión de máscaras (MSK) 355
C.3 Funciones colectivas (COL) 357
C.4 Funciones de registro (REG) 359
C.5 Funciones simples (SMP) 359
C.6 Funciones (FUN, PCK, NRM) 361
C.7 Extensiones 364
C.8 Makefile 365

D Código para el estudio del *overhead* **369**
D.1 Introducción 369
D.2 Listados 369

E Código MATLAB para la transformada *wavelet* **389**
E.1 Introducción 389
E.2 Listados 389

F Glosario **429**

Prólogo

Motivación

Los clusters*[1] de computadores se han convertido en la opción más sencilla y popular para iniciarse en computación paralela. Ofrecen a *cualquier* institución educativa una atractiva oportunidad para utilizar y enseñar Computación de Altas Prestaciones (HPC*), sin requerir ahora acceso a equipamiento costoso [34]. La idea de explotar este significativo potencial computacional ha merecido una entusiasta aceptación en la comunidad HPC, y la tendencia actual favorece este tipo de *supercomputación* basada en elementos de uso común (*commodity supercomputing*) [64].

Sin embargo, el diseño de herramientas software que posibiliten o faciliten la programación de este tipo de sistemas es esencial. Como hacen notar Coulad y Dillon en el *Journal of Parallel and Distributed Computing* [14],

> "...eficiencia no sólo consiste en encontrar la arquitectura más rápida para ejecutar una aplicación concreta, sino también en reducir el tiempo de diseño de una aplicación paralela y facilitar que nuevos usuarios se inicien en programación paralela."

Chapin y Worringen (miembros de la IEEE TFCC*), refiriéndose al uso de clusters para computación paralela, comentan en el *Cluster Computing White Paper* [9] que

> "muchas aproximaciones de computación superiores tanto técnica como conceptualmente han fracasado debido a la carencia de soporte en sus diversos aspectos: qué herramientas, gestores hardware y entornos *middleware** hay disponibles. Este soporte depende fundamentalmente del número de usuarios."

El entorno MATLAB* es un estándar *de facto* para simulación y modelado de prototipos, ampliamente utilizado tanto académicamente como en la industria, para resolución de problemas en un amplísimo espectro de dominios de aplicación. No parece razonable cuestionarse el número de usuarios presentes o futuros de MATLAB. De hecho, no es descabellado suponer que muchos centros educativos y grupos de investigación que instalen un cluster contarán con tanto o más personal familiarizado con MATLAB que personal habituado a programar aplicaciones paralelas mediante paso de mensajes (el método natural de programar este tipo de sistemas).

La naturaleza interactiva del entorno MATLAB es ideal no sólo para aprender y habituarse rápidamente a un nuevo dominio de aplicación, sino también para ahorrar una cantidad ingente de esfuerzo de programación en el desarrollo de una aplicación o prototipo. Un entorno middleware

[1] Las palabras marcadas con * aparecen en el Glosario.

que permitiera aprender y utilizar desde MATLAB una biblioteca de paso de mensajes, facilitaría la iniciación de la amplia base de usuarios MATLAB en programación paralela. Las distintas posibilidades de diseño de una aplicación paralela podrían ser rápidamente exploradas, dada la naturaleza interactiva de MATLAB y sus facilidades para permitir un rápido prototipado de aplicaciones. Los dos factores de eficiencia mencionados por Coulad y Dillon (y frecuentemente infraestimados) quedarían así satisfechos.

Por otra parte, un entorno middleware semejante contribuiría al éxito de los clusters como plataforma para supercomputación en el sentido al que hacen referencia Chapin y Worringen, permitiendo a los usuarios MATLAB utilizar el cluster para aplicaciones HPC desde el mismo momento de su instalación.

Indudablemente existen y han existido plataformas más potentes, más costosas y más difícilmente accesibles (no se pueden encargar en cualquier comercio del ramo, ni los plazos de entrega son tan inmediatos) que los clusters de computadores. Durante su vida útil, antes de que los avances tecnológicos dejen desfasada su potencia de cálculo, el desarrollo de software para estas plataformas es costoso, dado que debe ser programado o comprado por su reducido número de usuarios. La vida útil suele acabar cuando aparece un nuevo tipo de supercomputador con arquitectura y procesadores superiores, lo cual obliga a iniciar de nuevo el costoso proceso de adquirir el hardware y el software.

En comparación, los clusters pueden ir incorporando computadores cada vez más potentes conforme vayan siendo disponibles comercialmente, equilibrando la carga según las prestaciones de cada computador, retirando eventualmente los computadores más antiguos si fuera conveniente, y reutilizando siempre no sólo el software previamente existente sino también el nuevo software que previsiblemente irá apareciendo, desarrollado por una cada vez más extensa base de usuarios (atraidos por el bajo costo del hardware y del software), con menor costo, menor esfuerzo y mayor utilización.

Desde este punto de vista, un entorno middleware como el descrito previamente podría considerarse como una contribución significativa.

0.1 Resumen del trabajo presentado

En esta memoria se presentan dos entornos de simulación paralelos, PVMTB y MPITB, basado el primero en el sistema de paso de mensajes PVM* y el segundo en la implementación LAM* del estándar de paso de mensajes MPI*. Ambos entornos toman la forma de *Toolboxes** del entorno de cálculo científico MATLAB.

El sistema utilizado durante el desarrollo del trabajo ha sido un cluster* de computadores personales con una distribución estándar de Linux (RedHat), interconectados mediante un *switch**. Los lenguajes de desarrollo han sido MATLAB y C. Se ha comprobado también el funcionamiento de ambas *Toolboxes* en estaciones de trabajo Sun con el Sistema Operativo Solaris.

La Figura 1 muestra la relación entre el diverso *software* mencionado. Un programa C básico realiza llamadas al Sistema Operativo y a la biblioteca C libc. Si se tienen instaladas las bibliotecas PVM o MPI, el programa C puede realizar llamadas a ellas, convirtiéndose en una aplicación PVM o MPI, respectivamente.

Si se dispone de una instalación de MATLAB, se pueden redactar programas fuentes en len-

guaje MATLAB (*M-files**, ficheros .m), que son interpretados y ejecutados por el entorno MATLAB. El lenguaje MATLAB tiene una notación vectorial de alto nivel que conviene utilizar al objeto de evitar la pérdida de prestaciones debida a bucles interpretados.

También se pueden redactar programas C o FORTRAN que accedan al entorno MATLAB (para consultar el valor de una variable, realizar cálculos específicos sobre ella, etc), usando la biblioteca API* proporcionada a tal efecto por el fabricante (*The MathWorks*). Estos programas son invocados desde el intérprete MATLAB, y se desarrollan típicamente al objeto de acelerar cálculos que no puedan ser redactados en forma vectorial. En nuestro caso, se han utilizado para acceder a las bibliotecas de paso de mensajes.

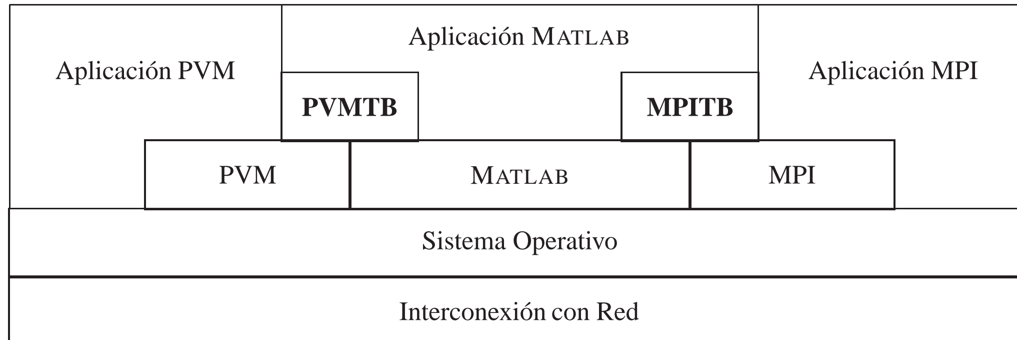


Figura 1: Diagrama general del uso de PVMTB y MPITB como entornos de simulación paralela. El código de la *Toolbox* realiza llamadas a las bibliotecas PVM o MPI para proporcionar funcionalidad de paso de mensajes, y realiza llamadas a la API MATLAB para tomar/devolver parámetros del/al proceso MATLAB. Varios de estos procesos se ejecutan en distintos computadores interconectados por la red.

Disponiendo de ambas instalaciones, MATLAB y PVM o MPI, es posible redactar ficheros-MEX (*MEX-files**) que utilicen paso de mensajes y se puedan invocar desde MATLAB. Las *Toolboxes* descritas en este trabajo "recubren"* las llamadas PVM y MPI, proporcionando al usuario MATLAB acceso a dichas bibliotecas, y permitiéndole por tanto desarrollar aplicaciones HPC, conservando no obstante todas las ventajas del entorno interpretado, incluyendo depuración, gráficos, comodidad de prototipado, etc, y añadiendo herramientas de depuración, visualización, etc, implementadas para PVM o MPI.

0.2 Visión por Computador Bioinspirada

El objetivo inicial del trabajo era proporcionar un entorno de desarrollo apropiado para la simulación de modelos de visión por computador, aunque finalmente ha derivado en un entorno de desarrollo más general. Estos modelos son computacionalmente costosos, presentando múltiples características que hacen interesante su paralelización. Muchos de los argumentos esgrimidos en la literatura se fundamentan o inspiran en el cada vez más detallado conocimiento que se tiene de la estructura y funcionamiento de los sistemas naturales de visión, dando lugar a la "Visión por Computador Bioinspirada" (*Bioinspired Computer Vision*). Los sistemas de visión natural presentan paralelismo de procesamiento a varios niveles:

Campo receptor: El propio sistema sensor (fotorreceptores) suele incorporar mecanismos de pro-

cesamiento local, probablemente al objeto de reducir el número de fibras del tracto visual y aprovechar mejor su ancho de banda. Las conexiones entre sensores no pueden propagarse ilimitadamente, lo cual implica que las primeras etapas de procesamiento han de estar restringidas a una forzosa localidad espacial, denominada “campo receptor” de un ganglio retinal. El procesamiento de un campo es relativamente independiente de los demás (salvo mecanismos de adaptación, corrección de contraste, etc), sugiriendo la posible simulación de distintas zonas receptivas en procesadores independientes.

Canales: La información de los sensores, una vez procesada localmente, se agrupa en múltiples canales de funcionamiento relativamente independiente. Existen canales de color (Blanco/Negro, Rojo/Verde, Amarillo/Azul), sistemas de procesamiento (Parvocelular, dedicado a la forma, y Magnocelular, dedicado al movimiento), y cada sistema tiene sus propias preferencias de localidad: así, el color se percibe más nítidamente en la fóvea*, donde hay mayor aglomeración de conos*, mientras que el movimiento se detecta mejor en la periferia retinal, donde predominan los bastones* y se emplean campos receptivos más extensos. La biología inspira pues la simulación paralela de dichos subsistemas, utilizando elementos de proceso separados, compartiendo la información preprocesada por los sensores.

Estructura en capas: La retina, el LGN* y el córtex visual presentan una estratificación en capas de neuronas. En la retina, los receptores (conos y bastones), las células bipolares, horizontales, amacrinas y los ganglios retinales son distinguibles morfológicamente, por su forma celular y árboles dendríticos. Esta característica permitió su temprana clasificación en capas, y hoy se conocen bastante bien las distintas funciones que desempeñan.

En el cisma óptico y LGN se segrega lateralmente el tracto óptico, dirigiendo a la mitad izquierda del cerebro las mitades izquierdas de ambas retinas, correspondientes a las mitades derechas del campo visual. El LGN presenta un sistema perfectamente ordenado de capas provenientes alternativamente de un ojo y del otro.

En el córtex visual primario, además de las mencionadas discriminaciones (lateralidad, canales de color, etc), las neuronas se estratifican en distintas capas relacionadas con el procesamiento realizado. Así, en cada capa se encuentran franjas de dominancia ocular, zonas selectivas a la orientación, al color, estructuras especializadas en análisis de forma (*blobs*) o movimiento (*stripes*), repartidas en capas (I-IV) con diversa morfología e interconexión.

Dado que cada capa transmite a la capa siguiente el resultado de su procesamiento, resulta natural (*bioinspirado*) simular el procesamiento de distintas capas en elementos de proceso separados.

La visión por computador es una de las *Grand Challenges** actuales, requiriendo capacidades de cómputo superiores a las comúnmente ofrecidas por la tecnología actual para un computador de propósito general. Los entornos de simulación presentados en este trabajo permiten a un usuario MATLAB prototipar aplicaciones HPC, como por ejemplo Visión por Computador Bioinspirada, en un cluster de computadores.

0.3 Computación Paralela y clusters de computadores

La Tabla 1 muestra las distintas clases de paralelismo disponibles bajo Linux que se comentan en la "Parallel Processing HOWTO" [17]. Se entiende por Procesamiento Paralelo el hecho de dividir un programa en fragmentos que puedan ser ejecutados simultáneamente, cada uno en un procesador distinto. Idealmente, un programa ejecutado en n procesadores debería ejecutarse n veces más rápido que su versión secuencial. La ganancia en velocidad (*speedup**) de un programa paralelo, definida como la razón entre el tiempo de ejecución del programa secuencial óptimo y el tiempo de ejecución del programa paralelo, sería entonces n.

Diversos factores impiden esta ganancia ideal, siendo uno de los motivos que siempre hay un tramo de código secuencial no susceptible de paralelización. En el artículo a partir del cual se acuñó el término *Ley de Amdahl** [3], el autor estima dicho tramo en un 15–20% del programa (denominándolo "overhead" o "data management housekeeping"), acotando cualquier posible ganancia en velocidad a un modesto 5–7. Adicionalmente, la versión paralela gastará un cierto tiempo comunicando resultados intermedios y sincronizando los distintos procesos. El costo de dichas operaciones, que podríamos denominar "*overhead* de paralelización" depende fuertemente de la alternativa de paralelismo adoptada.

Tabla 1: Distintas formas de paralelismo disponibles bajo Linux.
Prog: Paradigma de programación. **HL**ev: Métodos de programación de alto nivel. **LL**ev: Métodos de bajo nivel.

	1 computador	n computadores
1 CPU	Monoprocesador ILP* (instrucciones) SWAR* (datos)	Cluster (COTS*) **P**rog: Memoria Distribuida **HL**ev: Paso de Mensajes **LL**ev: sockets IP
+ copr.	Procesadores acoplados	
n CPUs	SMP* **P**rog: Memoria Compartida **HL**ev: S.O., Hebras, OpenMP **LL**ev: SysV shmem	Constelación **HL**ev: DSM* **LL**ev: Mensajes, shmem

A continuación se comentan brevemente las distintas alternativas de paralelismo mostradas en la Tabla 1:

Monoprocesador: Las CPUs actuales disponen de múltiples unidades funcionales. En las arquitecturas superescalares (Pentium, PowerPC 750, Alpha 21264. . .) el posible paralelismo a nivel de instrucción (ILP*) lo extrae principalmente el propio procesador, emitiendo las instrucciones de forma que se mantengan ocupadas el máximo número posible de unidades funcionales. En los diseños VLIW* (Itanium de Intel, Trimedia de Philips), el paralelismo se debe extraer al generar el código máquina, bien por parte del programador o del compilador, proporcionando a la CPU una secuencia de instrucciones que mantengan ocupadas el máximo número posible de unidades funcionales.

También existen ampliaciones multimedia (MMX* en los IA32* de Intel, Altivec de PowerPC, MVI en los Alpha de Digital/Compaq), que añaden al repertorio máquina instrucciones que operan con vectores. Esta forma de paralelismo de datos, principalmente accesible a nivel de lenguaje máquina o mediante bibliotecas de funciones, se denomina SWAR*.

Coprocesadores: Es también usual acoplar procesadores especializados a un computador de propósito general, como las tarjetas de audio, vídeo y DSPs*. Esta forma de coprocesamiento se verifica mediante el uso de lenguaje máquina o bibliotecas especializadas ("Kits de desarrollo") en el caso de coprocesadores especializados, o con entornos completos de desarrollo (compiladores cruzados, cargadores, programadores EPROM) en el caso de sistemas más generales (DSPs, arrays* de DSPs). Estos sistemas se denominan Procesadores Acoplados (*Attached Processors*).

Multiprocesadores: Cada vez son más accesibles placas madre de computadores multiprocesador (Bi-, Tetra-procesador. . .). En estas arquitecturas, todos los procesadores comparten la memoria principal y E/S, pudiendo desempeñar indistintamente las mismas funciones bajo el control del Sistema Operativo [86]. Debido e ello, esta forma de paralelismo recibe el nombre de Multiprocesamiento Simétrico (SMP*), y su paradigma de programación explota el hecho de que comparten la memoria principal de la placa madre (*Shared Memory*). Esta forma de paralelismo es mejor aprovechada por un Sistema Operativo multihebra (*MultiThreaded*) que explícitamente por el usuario.

Las aplicaciones del usuario en un sistema SMP también pueden utilizar un sistema de programación de alto nivel como las Hebras (*Threads*), o un estándar de Memoria Compartida como OpenMP, o incluso alguna biblioteca de bajo nivel como SVr4 IPC shm (estándar UNIX System V Release 4 para Comunicación entre Procesos usando memoria compartida) para aprovechar dicho paralelismo.

Clusters (Monoprocesadores con interconexión de altas prestaciones): También se están volviendo habituales las instalaciones de clusters de computadores, debido a su fácil disponibilidad (COTS*), precio asequible y facilidad de ampliación. Estos sistemas de Memoria Distribuida pueden utilizarse de diversas formas:

- **HTC* (High Throughput Computing)** : El estudio de la NHSE de 1996 ([72]) presenta una extensa revisión de los sistemas de alto rendimiento más populares, incluyendo PRM (Prospero Resource Manager), PBS (Portable Batch System) y Condor, entre otros. Son sistemas de colas y de equilibrado de carga que no requieren (aunque se pueden beneficiar de) la interconexión de altas prestaciones del cluster.
- **SSI* (Single System Image)** : Estas aproximaciones ofrecen al usuario una imagen unificada del sistema, como las mencionadas en la página del Area Técnica de Toni Cortés en la IEEE TFCC [85], incluyendo *Mosix* y *Solaris MC*.
- **DSM*, VSM* (Distributed/Virtual Shared Memory)** : También existe software para convertir un cluster en un sistema virtual distribuido de memoria compartida, bien sea a nivel de página, de objetos, de espacio único de direcciones, etc, permitiendo al usuario programar el cluster bajo el paradigma de memoria compartida. Una clasificación de estos sistemas puede consultarse en [38], incluyendo *TreadMarks* y *Opal*, por ejemplo.
- **HPC* (High Performance Computing)** : Un cluster es el entorno ideal para realizar paso de mensajes explícito, obteniendo provecho de la interconexión de altas prestaciones, permitiendo al usuario ejecutar aplicaciones HPC.

Constelaciones (SMPs con interconexión de altas prestaciones): La progresiva disminución de precio de los sistemas SMP hace que empiece a ser también atractivo instalar un cluster en el que cada computador es a su vez un Multiprocesador. Las mismas aproximaciones para utilizar un cluster (HTC, HPC, DSM, SSI) son aplicables a constelaciones.

Por ejemplo, un sistema de paso de mensajes podría implementar la comunicación entre CPUs del mismo SMP mediante memoria compartida (semáforos, spin-locks, zonas de exclusión mutua) y mediante TCP/IP cuando son de distintos SMPs, como hace el sistema LAM/MPI [41] bajo la opción de transporte sysv.

También se han modificado sistemas de memoria compartida para realizar el acceso a VSM fuera del SMP mediante paso de mensajes implícito, como hace por ejemplo *TreadMarks*.

La curva de aprendizaje para un método explícito de paso de mensajes es muy rápida. Un programador puede aprender rápidamente a paralelizar sus aplicaciones secuenciales, dividiendo las aplicaciones en tramos y programando explícitamente el envío de datos y recepción de resultados (o viceversa) en el tramo correspondiente.

Programando bajo memoria compartida, las aplicaciones de hebras o con semáforos pueden dar lugar fácilmente a errores difíciles de depurar (*starvation* o deprivación, acceso indebido a sección crítica, sobreescritura de buffers, código no reentrante...). Para hacer la biblioteca C *thread-safe* (susceptible de multihebra) se requiere, por ejemplo, convertir la variable global errno en una llamada a función errno(), ya que distintas hebras podrían sobreescribir su valor asíncronamente al generar un código de error.

La instalación del software y su administración también es una cuestión a considerar. Los sistemas de paso de mensajes (PVM, LAM/MPI) suelen consistir en una biblioteca de llamadas y una serie de programas de gestión y control para ejecutar y depurar las aplicaciones paralelas. Cada usuario puede instalar y configurar su propia versión del software, sin requerir privilegios especiales. En Sistemas Operativos con soporte multiprocesador (SMP), las bibliotecas de hebras o memoria compartida tampoco requieren una instalación o administración específica. Las restantes aproximaciones para clusters, particularmente SSI y HTC, requieren la intervención del administrador del sistema para instalar, configurar y mantener el cluster.

0.4 Estructura de los capítulos

Esta memoria describe las *Toolboxes* PVMTB y MPITB para prototipado de aplicaciones HPC bajo MATLAB en un cluster de computadores.

En el Capítulo 1 se introducen los diversos sistemas software utilizados. Se presentan brevemente PVM, LAM/MPI y sus opciones de codificación y encaminamiento, dada la influencia que tienen sobre sus prestaciones. También se presenta MATLAB y la actitud comercial del fabricante sobre los posibles usos del paralelismo bajo su entorno. Se introducen entonces las *Toolboxes* objeto de esta memoria y se comparan con trabajos previos, realizándose para ello un estudio de escalabilidad inspirado en una aplicación paralela frecuentemente encontrada en la bibliografía. También se realiza un test *ping-pong* para evaluar exclusivamente las prestaciones del paso de mensajes bajo cada *Toolbox*, sin contemplar la realización de ningún cálculo paralelo.

En el Capítulo 2 se presenta el equipo hardware utilizado y se describe brevemente el proceso de instalación y configuración. Se introduce también el tema del Modelado de Prestaciones, realizándose un estudio comparativo de las prestaciones de PVM y LAM/MPI en nuestro cluster.

En el Capítulo 3 se estudia con detalle el diseño de PVMTB, remarcando las decisiones de índole técnica que nos han permitido superar las prestaciones de trabajos anteriores y reducir el *overhead* del interfaz al mínimo posible. También se destaca la robustez del diseño escogido, con el que obtenemos una alta reutilización de código y por tanto un coste de desarrollo y mantenimiento mínimos. El capítulo concluye con un estudio comparativo con PVM al objeto de evaluar el *overhead* introducido por PVMTB en el paso de mensajes.

En el Capítulo 4 estudia con detalle el diseño de MPITB, con una estructura homóloga a la del capítulo anterior. Durante el estudio del *overhead* de MPITB frente a MPI se comentan las similitudes y diferencias más remarcadas con el ya estudiado *overhead* de PVMTB frente a PVM.

El Capítulo 5 se dedica a estudiar un ejemplo de aplicación real, basado en el análisis *wavelet* de imágenes. Tras presentar la aplicación y estudiar el código secuencial de referencia, se estudian dos posibles formulaciones del programa paralelo, con distintas granularidades al objeto de estimar las ganancias en velocidad que cabe esperar del uso de las *Toolboxes* presentadas en esta memoria.

En el Capítulo 6 se resumen las conclusiones y principales aportaciones de este trabajo. La memoria concluye con un glosario de términos frecuentemente utilizados y la bibliografía citada a lo largo de la exposición.

Capítulo 1

Introducción

Resumen del capítulo

En este Capítulo se introducen los diversos sistemas software utilizados para construir PVMTB y MPITB. Las características principales del sistema PVM, del sistema LAM/MPI, y del entorno MATLAB, bajo los cuales se ha desarrollado este trabajo, se resumen en los Apartados 1.1, 1.2 y 1.3, respectivamente.

Se han dedicado subsecciones específicas a los modos de codificación y encaminamiento de PVM y LAM, dada la influencia que tienen sobre las prestaciones del paso de mensajes. También se revisan los argumentos del fabricante (*The MathWorks*) sobre los posibles usos del paralelismo bajo MATLAB, y las características recientemente incorporadas a MATLAB que posibilitan o facilitan el desarrollo de un interfaz elegante con un sistema de paso de mensajes y la ejecución ventajosa de una aplicación HPC sobre un cluster de computadores. También se relacionan los trabajos previos (e incompletos) realizados en dicho sentido.

A continuación, en los Apartados 1.4 y 1.5, se presentan introductoriamente las *Toolboxes* PVMTB y MPITB. El Apartado 1.6 se dedica a comparar con cierto detalle el desarrollo de una aplicación paralela sencilla bajo algunas de las *Toolboxes* paralelas que se citaron en la relación anterior, incluyendo PVMTB y MPITB.

Al proponer el problema bajo la foma de un estudio de escalabilidad, en el cual se cronometra repetidamente la ejecución de la aplicación conforme se va aumentando el número de computadores que cooperan, se consigue comparar no sólo la relativa facilidad de programación de cada *Toolbox* manifiesta en el código fuente de la aplicación, sino también la controlabilidad y observabilidad del sistema, que afecta al código del *script** de automatización utilizado en cada caso. Estos detalles, que son frecuentemente infravalorados en la literatura (probablemente debido a su carácter técnico, en oposición al carácter *científico* del código de la aplicación) influyen más que la propia aplicación en el tiempo total de desarrollo y prueba/validación del prototipo, afectando significativamente a la relativa comodidad de uso de una *Toolbox* desde el punto de vista del usuario.

El Apartado 1.7 resume las conclusiones de este capítulo.

1.1 El sistema PVM

PVM (Parallel Virtual Machine, Máquina Paralela Virtual) es un conjunto integrado de bibliotecas y herramientas software que constituyen un entorno de computación concurrente heterogéneo, flexible y de propósito general, sobre computadores interconectados, de diversas arquitecturas [28, 27]. El objetivo global del sistema PVM es habilitar una colección heterogénea de computadores para ser usados cooperativamente en computación paralela o concurrente. Algunos principios en los que se basa PVM son:

Conjunto de computadores configurable: El usuario enumera en un fichero de configuración los computadores que formarán parte de la Máquina Paralela Virtual, cada uno de los cuales puede ser mono o multiprocesador de memoria compartida o distribuida. El usuario puede controlar cuáles de ellos cooperarán en cada ejecución de un programa PVM. El conjunto puede alterarse añadiendo o eliminando computadores, característica útil para tolerancia a fallos.

Soporte multiprocesador y heterogéneo: Los mensajes pueden contener diversos tipos de datos. Los computadores pueden tener diferentes representaciones internas de dichos tipos de datos. Entre los procesadores de un computador multiprocesador, PVM usa los mecanismos nativos de paso de mensajes para aprovechar el hardware subyacente.

Acceso "translúcido" al hardware: El usuario puede controlar la ejecución de determinadas tareas en los procesadores más apropiados, si conoce suficientemente el conjunto de computadores que usa, o tratar el entorno hardware como una colección de procesadores virtuales sin atributos especiales.

Computación paralela basada en procesos: La unidad de paralelismo en PVM es la *tarea*, hebra de control secuencial e independiente que alterna entre computación y comunicación. Usualmente cada tarea es un proceso UNIX.

Paso de mensajes explícito: Las tareas cooperan mediante envío y recepción explícitos de mensajes entre ellas.

El sistema PVM se compone de dos partes fundamentales: un *daemon** llamado *pvmd* y una biblioteca de rutinas de interfaz con PVM. También se dispone de un programa "consola" desde el cual se puede monitorizar y controlar la Máquina Virtual.

Al crear el usuario la máquina virtual, el *daemon* se ejecuta en cada computador de la misma. Se pueden iniciar entonces aplicaciones PVM usando cualesquiera de dichos computadores. Distintos usuarios pueden crear máquinas virtuales que se solapen. Cada usuario puede ejecutar simultáneamente varias aplicaciones PVM.

La *biblioteca* PVM contiene un repertorio funcionalmente completo de primitivas para la cooperación entre tareas de la aplicación PVM, como por ejemplo ejecución remota de procesos, sincronización, envío y recepción de datos y modificación de la máquina virtual.

El paradigma general para programación de aplicaciones PVM es como sigue. El usuario redacta uno o más programas secuenciales en C, C++ o FORTRAN que incluyen llamadas a la biblioteca PVM. Estos programas se recompilan para cada arquitectura en el conjunto de

computadores, y los programas de cada arquitectura se ubican en algún lugar accesible a los computadores de dicha arquitectura.

Típicamente, el usuario inicia desde la consola PVM o desde el intérprete de órdenes (*shell*) una copia de una tarea "maestra", que posteriormente inicia la ejecución remota de las diversas tareas "esclavas" cuya comunicación y computación constituye la aplicación paralela. Desde la consola también es posible arrancar simultáneamente varias copias (procesos) de un programa.

Los dos parámetros que influyen en las prestaciones del paso de mensajes en PVM son los modos de empaquetamiento y de encaminamiento.

1.1.1 Empaquetamiento

Cada tarea PVM tiene asociado un número identificador (TID), en analogía a los identificadores de proceso (PID) en el S.O. El TID se asigna secuencialmente por orden de creación, a partir del TID del *daemon* del que depende la tarea (DTID). Los DTID son números lo suficientemente separados (típicamente espaciados 0x40000==2^{18}).

Para enviar un mensaje, una tarea PVM debe crear un buffer* de transmisión utilizando la llamada:

```
int bufid = pvm_initsend ( int encoding )
```

La página de manual pvm_initsend (3PVM) explica que el argumento encoding puede tomar los valores:

PvmDataDefault (0): El mensaje se codifica a XDR* (eXternal Data Representation). Esto implica, por ejemplo, que las máquinas *little endian** deben transmitir en orden inverso los bytes que representan una variable entera.

PvmDataRaw (1): El mensaje se transmite "crudo", sin traducción a XDR. Cuando los computadores fuente y destino son de la misma arquitectura de datos, esta opción mejora las prestaciones, ya que se ahorra el paso intermedio de codificación/reordenación.

PvmDataInPlace (2): El mensaje se transmite sin copiarlo previamente al buffer de transmisión. En el buffer se almacena un puntero a los datos y el tamaño (e intercalado) a transmitir. Cuando la aplicación PVM redactada por el usuario puede garantizar que los datos no se alterarán hasta que acabe la transmisión, esta opción mejora las prestaciones, ya que se ahorra la copia de memoria intermedia.

A continuación el usuario empaqueta sucesivamente los datos deseados en el buffer, utilizando en cada caso la llamada correspondiente al tipo de datos a empaquetar:

```
int info = pvm_pk<type> ( type* data, int nitem, int stride )
```

El argumento data indica la dirección de comienzo de los datos, nitem es el número de elementos y stride el intercalado o separación entre elementos. El usuario dota a cada mensaje PVM de una etiqueta (*tag*) al enviarlo a otra tarea (tid) PVM, usando la llamada:

```
int info = pvm_send ( int tid , int msgtag )
```

El argumento tid identifica la tarea destino del mensaje, y msgtag identifica el propio mensaje.

La llamada pvm_send no es bloqueante, en el sentido de que el mensaje se envía al *daemon* del computador donde reside la tarea tid, independientemente de que ésta realice o no una llamada pvm_recv(). La tarea destino recibe y desempaqueta el mensaje usando las llamadas:

```
int bufid = pvm_recv ( int tid , int msgtag )
int info = pvm_upk<type> ( type* data, int nitem, int stride )
```

en donde tid identifica la tarea fuente deseada, y msgtag identifica el tipo de mensaje deseado. Ambos parámetros pueden tomar el valor -1 para indicar indiferencia. Típicamente, las aplicaciones mantienen un sistema coherente de etiquetado, de manera que la tarea destino puede deducir a partir de la etiqueta el contenido del mensaje, y realizar entonces llamadas unpack adecuadas al tipo y tamaño de los contenidos, que en principio serían desconocidos. Tras desempaquetar, la zona de almacenamiento proporcionada, *data, queda sobrescrita con nitem elementos del buffer distanciados stride posiciones entre sí.

El mensaje lleva una cabecera indicando la codificación utilizada, de manera que si se usó PvmDataDefault se deshace ahora la codificación XDR. En recepción no hay diferencia entre los modos PvmDataRaw y PvmDataInPlace, ya que de todos modos se deben copiar los datos desde el buffer de recepción del *daemon* a la zona de almacenamiento *data proporcionada por la aplicación PVM. En las referencias [28, §10.4.9, p.57] y [27, §7.2.2, p.98] se menciona explícitamente que no existen decodificadores *inplace*. Esto es coherente con la semántica no bloqueante de la llamada pvm_send(), ya que el *daemon* del host receptor siempre puede recibir el mensaje aunque el proceso receptor no esté ejecutando pvm_recv().

La llamada pvm_recv() es bloqueante, en el sentido de que si el *daemon* de la tarea destino no dispone de un mensaje cumpliendo las condiciones (tid ,msgtag) especificadas por el receptor, éste queda retenido en la llamada, retornando sólo cuando se haya recibido un mensaje que las cumpla.

Existen también una pareja de llamadas que simplifican el paso de un mensaje, a costa de una lista de parámetros mayor y la limitación a mensajes de una única variable (un único tipo de datos sin intercalado):

```
int info = pvm_psend ( int tid , int msgtag, char *buf, int len , int datatype )
int info = pvm_precv ( int tid , int msgtag, char *buf, int len , int datatype, int * atid , int *atag , int *alen )
```

Los dos primeros parámetros coinciden con los de pvm_send()/recv(), los tres siguientes definen la variable a enviar o zona de memoria reservada para recibir, y los tres últimos devuelven el *sobre* (*envelope*) del mensaje recibido (identificador de fuente, etiqueta y longitud del mensaje). PVM define constantes (PVM_STR, PVM_INT, PVM_DOUBLE...) para usar como argumento datatype, tantas como diversas llamadas pvm_[un]pack() existen. El análisis de prestaciones indica que estas llamadas utilizan ruta directa y codificación *DataInPlace*.

Debe recalcarse que son las rutinas básicas pvm_send()/recv() las que requieren una etapa previa de empaquetamiento. Aunque aparentemente sea una complicación innecesaria, se debe recordar que la *latencia* (o tiempo de inicialización de la comunicación) es muy alto en Ethernet (y también en otras implementaciones del nivel físico de red), de manera que si hay que transmitir varios (digamos n) datos, es muy ventajoso agruparlos y transmitirlos de una sola vez ahorrándose $n-1$ latencias. Las rutinas pvm_psend()/precv() sólo son ventajosas cuando se necesita transmitir un único array.

Existe una etiqueta implícita adicional asociada a cada mensaje, el *contexto*. Por defecto, todos los mensajes se envían y reciben bajo el contexto 0. Las llamadas pvm_newcontext() y pvm_freecontext() crean y liberan contextos dinámicamente. pvm_getcontext() y pvm_setcontext() permiten consultar y fijar el contexto actual. A diferencia de la etiqueta *tag*, que se menciona explícitamente en las llamadas de envío y recepción, el contexto es implícito, de manera que una recepción solicitada bajo contexto 0 no puede ser satisfecha por un envío realizado bajo un contexto distinto.

La utilización de diferentes contextos permite la programación de bibliotecas que utilicen PVM internamente, y que puedan ser invocadas desde aplicaciones PVM sin interferir con el paso de mensajes de la propia aplicación. Para ello, la biblioteca solicita al arrancar un número de contexto propio, asignado dinámicamente por el sistema PVM, con lo cual todas sus comunicaciones quedan aisladas del paso de mensajes de cualquier otra aplicación o biblioteca que pueda estar ejecutando el usuario.

1.1.2 Encaminamiento

De entre las opciones de la biblioteca PVM explicadas en la página de manual pvm_setopt (3PVM), la primera se denomina PvmRoute, y puede tomar los siguientes valores:

PvmDontRoute (1): Si una tarea PVM no desea que se establezcan enlaces TCP directos con ella, puede usar esta opción para impedirlo.

PvmAllowDirect (2): Valor por defecto, permite que se establezcan rutas TCP directas con esta tarea PVM.

PvmRouteDirect (3): Una vez establecida esta opción, al realizarse un envío entre dos tareas PVM, se crea un enlace TCP directo entre ambas que se reutiliza en subsecuentes envíos. Los datos no se copian al *daemon* del host emisor, sino que se transmiten por la ruta TCP establecida.

El impacto de esta opción en las prestaciones de PVM es significativo, sobre todo para los empaquetamientos PvmDataInPlace y PvmDataRaw. El empaquetamiento PvmDataDefault, al requerir un paso intermedio de codificación XDR, ha de pasar ineludiblemente a través del *daemon*,

1.1.3 Grupos Dinámicos

Se denominan *colectivas* aquellas operaciones de paso de mensajes en las que varias tareas han de comunicar datos, en oposición a las rutinas *punto a punto*, en la que una tarea envía y otra recibe datos. Por ejemplo, son operaciones colectivas la barrera, en la que varias tareas van

quedando bloqueadas hasta que un número suficiente de ellas ejecuta pvm_barrier(), o el *broadcast*, en el que una tarea envía los mismos datos a varias.

En PVM los grupos se referencian por nombre (un *string* identificador). En cualquier momento, cualquier tarea puede escoger un identificador cualquiera y crear un grupo, o unirse a él si ya existía. Para ello es necesario que exista un proceso *servidor* de grupos (pvmgs, de **G**roup **S**erver). Probablemente se ha preferido excluir esta funcionalidad del *daemon* PVM para reducir su tamaño, facilitando la ejecución de programas que no utilicen llamadas colectivas. El *daemon* pvmgs sólo se arranca cuando la aplicación PVM realiza la primera llamada colectiva.

También al objeto de no aumentar la complejidad de la biblioteca PVM, que por defecto es una biblioteca estática cuyo código se añade a cualquier programa enlazado con ella, la biblioteca de grupos libgpvm3 está separada de la biblioteca PVM libpvm3.

Las operaciones colectivas PVM son:

```
int info = pvm_barrier( char *group, int count )
int info = pvm_bcast ( char *group, int msgtag )
int info = pvm_reduce( void (*func)(), void *data, int count, int datatype, int msgtag, char *group, int rootginst)
int info = pvm_scatter ( void *result , void *data, int count, int datatype, int msgtag, char *group, int rootginst)
int info = pvm_gather ( void *result, void *data, int count, int datatype, int msgtag, char *group, int rootginst)
```

1.2 El sistema LAM

LAM (Local Area Multicomputer, Multicomputador de Area Local, [41, 70]) es un entorno de programación y desarrollo de aplicaciones MPI ([66, 67, 65]) para computadores heterogéneos conectados en red. Usando LAM, un cluster dedicado o la infraestructura de red que se tenga disponible puede actuar como un computador paralelo en la resolución de un problema.

LAM implementa el estándar MPI 1.2 completo salvo la cancelación de envíos, y gran parte del estándar MPI 2.0, incluyendo comunicación unilateral, interfaz C++, arranque dinámico de procesos, facilidades cliente/servidor, etc. Puede ser usado con soporte extensivo para depuración (modo *daemon*) o con máximas prestaciones para aplicaciones en fase de producción (modo *cliente a cliente*).

Las características del sistema LAM/MPI destacadas por los autores son:

Soporte MPI-1: Completo, salvo cancelación de envíos.

Soporte MPI-2: Creciente.

Heterogeneidad: Soporta multitud de plataformas UNIX, como Solaris, IRIX, AIX, Linux, HP-UX... e interoperación entre todas ellas.

Depuración: Se dispone de utilidades que, bajo el modo *daemon* (−lamd), permiten observar los procesos arrancados bajo LAM y su estado (mpitask), así como los mensajes pendientes de recepción (mpimsg).

Modo cliente a cliente: Se dispone de otro modo de funcionamiento (−c2c), para aplicaciones completamente desarrolladas y depuradas, con prestaciones superiores, a costa de la información de depuración.

Nodos dinámicos: Se pueden añadir o eliminar máquinas del Multicomputador, posibilitando así implementar un esquema de tolerancia a fallos.

Gratuito: Al igual que PVM, LAM no requiere ningún pago de licencias, alquiler o precio de compra.

Similarmente a PVM, LAM se ejecuta en cada computador como un *daemon* UNIX. Los autores destacan que está estructurado como un *nano-kernel* con la adición de varios procesos. El kernel proporciona un servicio básico de paso de mensajes y citas (*rendez-vous*, sincronización) a los restantes procesos del *daemon*.

Algunos de los procesos del *daemon* forman el subsistema de comunicación por red, encargado de transferir mensajes entre *daemons* LAM, a través de un protocolo basado en TCP/UDP. Esta "capa" añade al sistema básico proporcionado por el kernel capacidades como la fragmentación en paquetes al enviar y reconstrucción de mensajes al recibir (*packetization*) y almacenamiento temporal (*buffering*).

Otros procesos del *daemon* funcionan como servidores de capacidades remotas, tal como acceso a ficheros o ejecución de programas. Se pueden añadir o suprimir servicios según se desee, mediante un fichero de configuración en texto legible. En cualquier caso, el *daemon* aparece como un único proceso.

Al igual que PVM dispone de la consola para manejar la Máquina Virtual, LAM dispone de herramientas para monitorizar y controlar el Multicomputador (mpitask, mpimsg, mpirun, lamboot. . .). Asimismo, el usuario dispone de una biblioteca de rutinas con un conjunto estándar (MPI) de llamadas para realizar paso de mensajes explícito.

Similarmente a PVM, en donde la elección del modo de empaquetamiento o de encaminamiento se hace en tiempo de ejecución, LAM decide según unos indicadores (*runtime flags**) el uso del modo *daemon* o del modo *cliente-a-cliente*, y la homogeneidad o no del Multicomputador. Sin embargo, las herramientas proporcionadas sólo permiten el ajuste de dichos indicadores al arrancar la aplicación paralela (mpirun −O −c2c), no estando previsto el caso de alterar dicha elección una vez arrancada la aplicación como en PVM.

1.2.1 Homogeneidad

El estándar MPI contempla la posibilidad de que el usuario deseara empaquetar una colección de datos de diversos tipos para transmitirlos en un solo mensaje, aunque no es la forma de operación preferida. La operación más básica consiste en enviar una variable usando la llamada:

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

que en el destino puede ser recibida mediante la llamada:

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
```

En ambas llamadas, los tres primeros argumentos describen el dato a enviar o la zona de almacenamiento dispuesta para recibir. buf es la dirección de una zona de almacenamiento proporcionada por la aplicación, por ejemplo &var en el caso de una variable aislada, o &arr[0] o incluso

arr en el caso de un array*. A partir de dicha posición se encuentran almacenados (en envío) o se almacenarán (en recepción) count elementos de tipo datatype.

El estándar MPI proporciona constantes de tipo MPI_Datatype para muchos tipos de datos comunes (MPI_CHAR, MPI_INT, MPI_LONG, MPI_FLOAT, MPI_DOUBLE...). Las llamadas MPI_Type_* permiten crear variables de tipo MPI_Datatype describiendo tipos de datos arbitrariamente complejos, para ser usadas como argumento datatype en posteriores llamadas MPI.

La alternativa MPI al empaquetamiento PVM es por tanto definir un nuevo tipo de datos describiendo el contenido deseado del mensaje, asignar los valores de una variable con el tipo de datos definido, y transmitirla usando las llamadas MPI_Send()/Recv(). Si no se desean definir tipos de datos, es aún posible usar las llamadas MPI_[Un]Pack(), aunque no es el método preferido.

Los argumentos restantes indican el destino, etiqueta y contexto del mensaje enviado, o imponen los valores deseados de fuente, etiqueta y contexto en recepción. A diferencia de PVM, bajo MPI el contexto es un argumento explícito denominado *comunicador*, estando siempre disponible el comunicador por defecto MPI_COMM_WORLD, que aglutina a todas las tareas MPI de la sesión, y el comunicador individual MPI_COMM_SELF, que incluye únicamente a la tarea que lo referencia.

La recepción requiere un argumento de estado adicional en el que se refleja el *sobre* (*envelope*: fuente, etiqueta, longitud y código de error) del mensaje recibido. Esta información es útil cuando se usan las constantes predefinidas MPI_ANY_SOURCE o MPI_ANY_TAG como argumentos source o tag en recepción.

Al arrancar una aplicación MPI mediante el comando mpirun −O, se indica que el Multicomputador es homogéneo, activándose el *flag* RTF_HOMOG. En tiempo de ejecución, se comprueba el valor del *flag*, y si está activado las llamadas de paso de mensajes no realizan codificación XDR.

La presencia o no del modificador −O se corresponde con las opciones de empaquetamiento PvmDataRaw y PvmDataDefault, respectivamente.

1.2.2 Modo *daemon* o *cliente-a-cliente*

Al arrancar una aplicación MPI mediante el comando mpirun −lamd, se indica que se desea el modo de depuración, desactivándose el *flag* RTF_MPIC2C. Con mpirun −c2c se activaría. En tiempo de ejecución, se comprueba el valor del *flag*, y si está activado las llamadas de paso de mensajes no utilizan el *daemon* LAM como mecanismo de transmisión, sino que se usa directamente TCP/IP.

Los modificadores −c2c y −lamd se corresponden respectivamente con las opciones de encaminamiento PvmRouteDirect y PvmDontRoute. Sin embargo, conviene notar que el establecimiento de la ruta directa se realiza en distintas circunstancias:

PVM: si la opción PvmRoute de una tarea vale PvmRouteDirect y la tarea realiza un pvm_send() a otra tarea con la que aún no se tiene ruta directa, se procede a establecer la ruta directa en ese momento, antes de realizar el pvm_send(). Toda comunicación posterior entre dichas tareas (envío o recepción) se realiza a traves de la ruta directa.

MPI: al arrancar la aplicación con mpirun −c2c, se establecen rutas directas entre todos los procesos. Cuando retorna la llamada MPI_Init(), ya están establecidas todas las rutas directas.

Al realizar estudios de prestaciones comparativos con PVM (por ejemplo, un test *ping-pong*), es usual realizar un pvm_send() previo con el único objeto de establecer la ruta directa, y que el tiempo de establecimiento no sea contabilizado erróneamente como tiempo de transmisión.

1.2.3 Grupos Estáticos

El estándar MPI contempla una multitud de operaciones colectivas, entre ellas todas las ofrecidas por PVM:

```
int MPI_Barrier (MPI_Comm comm)
int MPI_Bcast  (void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
int MPI_Reduce (void *sendbuf,
                void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
int MPI_Scatter (void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)
int MPI_Gather (void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Los grupos MPI están asociados a comunicadores que no son alterables. El grupo del comunicador MPI_COMM_WORLD está formado por todos los procesos que fueron arrancados conjuntamente por el comando mpirun. Es posible arrancar nuevos procesos mediante MPI_Comm_spawn(), y los nuevos procesos están agrupados en un nuevo comunicador. También es posible formar un nuevo (intra)comunicador normal uniendo dos (inter)comunicadores obtenidos de una llamada spawn, usando MPI_Intercomm_merge(). Es posible obtener el grupo de un comunicador (MPI_Comm_group()), operar sobre grupos usando las llamadas MPI_Group_*(), y crear un comunicador a partir de un grupo modificado (MPI_Comm_create()).

En todos los casos citados, el comunicador contiene siempre un número fijo de procesos (rangos MPI), y las llamadas colectivas implican a todos los rangos dentro del comunicador especificado. MPI_Barrier() no incluye por tanto un argumento cnt, imprescindible en PVM para asegurarse de que el número esperado de tids haya tenido tiempo de unirse al grupo. Similarmente, ninguna llamada colectiva MPI tiene argumento msgtag. Otra diferencia notable con PVM es que el mensaje enviado mediante pvm_bcast() puede ser recibido con pvm_recv(), mientras que en MPI todos los rangos deben ejecutar la llamada MPI_Bcast().

Los grupos estáticos son citados frecuentemente como una ventaja de MPI sobre PVM, siendo su uso menos proclive a errores semánticos. Las páginas de manual de pvm_reduce() y pvm_gather() advierten explícitamente de los errores típicos que se pueden cometer, como por ejemplo iniciar una operación colectiva cuando aún no se han unido todos los tids del grupo, o cuando ya lo han abandonado algunos.

1.3 El entorno MATLAB

MATLAB es un entorno interactivo de simulación y prototipado. Construido en torno a las bibliotecas de Algebra Lineal LAPACK y BLAS, y de procesamiento de señal FFTW*, dispone de más de 600 rutinas de cálculo numérico aplicables a infinidad de problemas matemáticos, de estadística o ingeniería.

Las rutinas han sido optimizadas para operaciones matriciales, de manera que un programa MATLAB correctamente redactado, en el cual los datos son operados mediante notación matricial (no todas las aplicaciones son susceptibles de dicha formulación) iguala en prestaciones a las

más cuidadosas implementaciones del mismo algoritmo en lenguajes de más bajo nivel como C++ o C, con la ventaja adicional de requerir menor esfuerzo de programación. Esto permite al programador concentrarse en su aplicación, en lugar de consumir esfuerzo en buscar, estudiar, implementar, depurar y mantener código relativo a métodos numéricos. MATLAB permite reutilizar gran cantidad de métodos numéricos depurados, estables, catalogados como “state-of-the-art”, con un mínimo esfuerzo de programación.

MATLAB incluye, entre otras, funciones para:

Matrices y Algebra Lineal: aritmética matricial, ecuaciones lineales, valores propios, valores singulares, factorizaciones.

Polinomios e Interpolación: raíces, evaluación, diferenciación, aproximación de funciones, desarrollo en fracciones parciales.

Procesamiento de Señal: filtros digitales, transformada de Fourier, convolución.

Estadística y Análisis de Datos: regresión, aproximación, filtrado y preprocesamiento de datos.

Funcionales: utilidades para crear gráficas de una función, para problemas de optimización, localización de ceros y cuadratura.

Ecuaciones Diferenciales: resolución de problemas de valor inicial y de límite.

Matrices Dispersas: soportadas tanto en operaciones generales como especializadas, incluyendo métodos iterativos especiales para ecuaciones lineales dispersas.

MATLAB permite crear el tipo de gráficas especializadas requeridas en ingeniería y ciencia. Desde gráficas bidimensionales de funciones o datos experimentales hasta curvas de nivel etiquetadas o interfaces gráficos, un abanico de posibilidades está disponible para ayudar al usuario a visualizar, manejar, procesar y presentar datos, incluyendo:

Gráficas 2-D y 3-D comunes, como gráficas lineales o logarítmicas, histogramas, diagramas de barras, de tarta, gráficas de superficies, de alambres (*mesh*), etc.

Visualización de volúmenes.

Importación y exportación en formatos gráficos estándar.

Soporte Hardware y software de aceleración OpenGL.

Animación y sonido.

Diversos modelos de iluminación, incluyendo múltiples fuentes de diversos colores.

Control de la perspectiva, movimiento de cámara, etc.

Edición interactiva o control programado de atributos gráficos (grosor/color/estilo de línea, de ejes, leyenda, etc).

El lenguaje MATLAB puede ser ampliado por el usuario mediante la creación de *scripts** (guiones) y funciones propias, redactados en lenguaje MATLAB e interpretados por el entorno interactivo. Estos ficheros de texto conteniendo código MATLAB interpretable por el entorno se denominan *ficheros M* (M-files*, por MATLAB) y se identifican por la extensión `.m`.

También existe la posibilidad de redactar ficheros fuente en lenguaje C o FORTRAN usando llamadas a la API proporcionada por *The MathWorks*. Una vez compilados con un compilador C o FORTRAN y enlazados con la biblioteca API, estos *ficheros MEX* (MEX-files*, por MATLAB EXecutable) pueden ser invocados desde el intérprete MATLAB. Las llamadas de la API se clasifican en cinco grandes grupos:

MX: Las gran mayoría de funciones API tienen prefijo mx. Permiten leer los argumentos con los que se ha invocado al fichero MEX desde el intérprete MATLAB, crear y dar valor a los argumentos de retorno, etc. Por ejemplo, mxArrayToString() permite traducir un *string* MATLAB (de tipo mxArray) a lenguaje C (tipo **char***), y mxCreateDoubleMatrix() crea un array MATLAB (tipo mxArray) de elementos **double.**

MEX: Las funciones con prefijo mex permiten realizar desde el fichero MEX operaciones en el entorno MATLAB; por ejemplo, imprimir un mensaje de error usando mexErrMsgTxt() o encargar al entorno la evaluación de un comando mediante mexEvalString().

MAT: Las funciones mat facilitan el manejo de ficheros de datos MATLAB, caracterizados por la extensión .mat. Un fichero MEX podría exportar un fichero MAT a otro formato, usando funciones MAT para consultar el contenido, y llamadas libc para escribir un nuevo fichero en el formato deseado, o viceversa.

ENG: Las funciones con prefijo eng (*engine*, motor) permiten utilizar el MATLAB como si de un fichero se tratara. El MATLAB se convierte así en un “motor” de cálculo. Por ejemplo, engOpen() crea una nueva sesión MATLAB, devolviendo un *handle**. engEvalString() permite “escribir” un comando al motor

DDE: La versión Windows dispone de llamadas que permiten comunicar MATLAB con otras aplicaciones Windows mediante los protocolos ActiveX o DDE (Dynamic Data Exchange).

Una colección de *scripts*, funciones y posiblemente ficheros MEX relacionados con un ámbito concreto de aplicación suele agruparse bajo el nombre de *Toolbox* (caja de herramientas), existiendo gran diversidad de ellas disponibles tanto comercialmente, por parte de *The MathWorks* y otras empresas, como gratuitamente, por parte de Universidades e investigadores individuales.

Por ilustrar el concepto de *Toolbox* con un ejemplo relacionado con procesadores acoplados, recientemente *The MathWorks* ha lanzado un Kit de Desarrollo para la familia de DSPs TMS320 C5000/C6000 de Texas Instruments [56]. Este Kit establece una comunicación entre el software de *The MathWorks* (MATLAB-Simulink) y el software de Texas Instruments (Code Composer Studio/Real Time Workshop), de manera que se puede controlar totalmente el DSP desde MATLAB, diseñar filtros u otro tipo de sistema digital en Simulink, traducir el código Simulink a código DSP, cargar el programa en el DSP real, ejecutarlo, monitorizarlo, depurarlo, obtener los resultados en tiempo real y visualizarlos usando las capacidades gráficas de MATLAB. Un producto similar existe para la familia 56K de Motorola [54].

1.3.1 Oportunidad del trabajo presentado

Los nuevos tipos de datos introducidos con la versión 5.0, así como la mejorada capacidad de depuración, son muy apropiados para desarrollar un software como las *Toolboxes* objeto de esta

memoria, convirtiendo a MATLAB en un entorno capaz de reducir el tiempo de desarrollo y ejecución de aplicaciones paralelas y facilitar el acceso de nuevos usuarios a la programación paralela. En este sentido, un campus con licencia MATLAB sería el entorno natural para PVMTB y MPITB como herramientas docentes.

Debido a la reducida pérdida de prestaciones que su uso supone (frente al uso directo de PVM o LAM/MPI desde C), también son útiles para investigación e ingeniería (uso industrial). En este sentido, el entorno natural sería un cluster, en donde la interconexión de altas prestaciones es fundamental. El entorno MATLAB permite un rápido prototipado de aplicaciones, simplificando tareas como la implementación de métodos matemáticos, generación de gráficas, etc, mientras que el cluster permite obtener ganancia en prestaciones. La posibilidad de utilizar el compilador MATLAB para eliminar el *overhead* asociado al entorno interpretado, una vez depurado el prototipo, hace aún más interesante esta aproximación.

A partir de la versión 6.0, MATLAB incorpora LAPACK* [63], ofreciendo eventualmente la posibilidad de usar una biblioteca BLAS* multihebra (*multithreaded*) específicamente afinada al procesador concreto para obtener aún mayor *speedup* en clusters de SMPs (constelaciones). La decisión de incorporar LAPACK sugiere que *The MathWorks* contempla la posibilidad de que el usario de una constelación utilice paso de mensajes explícito entre nodos, pero no la de que programe explícitamente los múltiples procesadores de cada nodo. El paralelismo SMP debe ser extraido automáticamente por el Sistema Operativo y/o LAPACK.

Así pues, varios factores han concurrido desde 1995 para eludir el descartado uso de paralelismo en los propios comandos MATLAB, y reforzar la única posibilidad que Moler [62] consideraba provechosa para computación paralela bajo MATLAB: paralelismo explícito por parte del usuario, de granularidad media–alta.

Soporte SMP: La progresiva accesibilidad y reducción de precio de PCs y estaciones de trabajo multiprocesador ha provocado la adecuación y optimización de bibliotecas de Algebra Lineal (como ATLAS) para este tipo de sistemas. *The MathWorks* ha estado atento a esta evolución y ha incorporado LAPACK a su entorno MATLAB. Esto abre las puertas al uso de una biblioteca BLAS afinada a multiprocesador. En [62] se descartaba crear versiones paralelas de comandos MATLAB argumentando que es a nivel de S.O. donde debe decidirse la asignación óptima de procesos/hebras a procesadores.

Clusters: La progresiva accesibilidad y reducción de precio de *switches* y tarjetas de red de altas prestaciones ha provocado la aparición del cluster de computadores como plataforma alternativa de computación paralela. En [62] se descartaba realizar paso de mensajes en multiprocesadores de memoria distribuida debido a la escasa cantidad de memoria por nodo y el costoso tiempo de comunicación usual en aquella epoca.

También se destacaba que muchas aplicaciones MATLAB consisten en la repetición de cálculos con distintos datos, como por ejemplo en una simulación MonteCarlo, en donde cada iteración o experimento es independiente de los resultados de los demás, pudiendo ser realizada en un procesador distinto. Este “paralelismo de bucle externo”, la única forma de paralelismo que se consideraba viable bajo MATLAB en 1995, no requiere ningún cambio fundamental al entorno MATLAB, siendo el usuario quien explícitamente debería paralelizar su código.

Moler reflexionaba que aún si la tecnología permitiera realizar una versión paralela eficiente de MATLAB, comercialmente sería un desastre para ellos, ya que supondría un esfuerzo considerable

de diseño que no se podría amortizar con el previsiblemente reducido número de ventas. Así pues, era más razonable continuar optimizando y mejorando la versión secuencial.

Paso de mensajes bajo MATLAB

Algunas de las mejoras recientemente introducidas en MATLAB facilitan significativamente el desarrollo elegante y sistemático de un interfaz entre MATLAB y una biblioteca de paso de mensajes ([24], [53], [51]):

- Auto-diagnóstico en la API.
- Soporte para Depuración y Optimización.
- El tipo de datos “estructura”.
- El tipo de datos “cell-array”.
- Capacidad de un número variable de argumentos en llamada a función (varargin, varargout).

En el desarrollo de PVMTB y MPITB se han seguido una serie de criterios comunes para aprovechar estas características del entorno MATLAB:

Coerción, cambio o promoción de tipo: Se han aprovechado convenientemente los nuevos tipos de datos:

int↔double: Existen tipos de datos enteros en MATLAB, pero su utilidad fundamental es el ahorro de memoria en tratamiento de imágenes. No se puede operar (suma, producto, etc) con datos enteros. Se ha preferido usar *flints* (enteros almacenados en punto flotante), tansformando a **int** antes de invocar a PVM/MPI y a **double** al volver a MATLAB.

cell-array↔char:** El tipo “array de celdas” es el idóneo para almacenar *strings* de distintas longitudes bajo MATLAB. Todas las filas de un array de caracteres MATLAB deben tener el mismo número de columnas, siendo inconveniente para almacenar cadenas de caracteres de distintas longitudes.

estructura↔struct: Las estructuras C en PVM/MPI se traducen al tipo estructura MATLAB respetando la nomenclatura de los campos. Esto es particularmente interesante en un entorno interactivo en donde las variables quedan almacenadas en la zona de trabajo (*workspace*). Las estructuras permiten mantener agrupada toda la información relacionada con (o devuelta por) una llamada PVM/MPI. Esto no se podía hacer en versiones anteriores de MATLAB (p.ej. comando pvm_config en DP-TB, Tabla 1.3).

handle↔FILE:** MATLAB admite E/S orientada a descriptores de fichero (*handles*) en lugar de usar la biblioteca de *streams*. Los descriptores MATLAB se traducen a FILE** usando la llamada fdopen() de la biblioteca de *streams*.

Retorno de valores: El lenguaje C no permite que las funciones devuelvan varios valores. Para conseguir este efecto, se suele devolver un puntero a estructura conteniendo los datos deseados. También se recurre a pasar punteros a variables que pueden ser entonces modificadas por la función invocada (paso por referencia). MPI usa la segunda técnica,

siendo siempre el entero devuelto un código de error (salvo MPI_Wtick() y MPI_Wtime). PVM usa ambas técnicas.

El lenguaje MATLAB permite a las funciones devolver varios valores. Se ha seguido el criterio de reordenar el patrón de llamada pasando a la parte izquierda (*left-hand side*) de la llamada los parámetros pasados por referencia.

Argumentos inferibles: Una de las características más útiles de MATLAB como lenguaje de rápido prototipado es la ausencia de declaraciones de tipo. Las variables pueden alterar dinámicamente su tamaño y número de dimensiones, y estos atributos pueden ser consultados en cualquier instante.

El lenguaje C se indica explícitamente el tamaño de una matriz o vector en su declaración (matriz[M][N]). El final de un *string* (**char** ∗) se suele marcar mediante un carácter ASCII NULL. Sin una función puede manejar matrices de diversos tamaños, se suele indicar el tamaño mediante un parámetro adicional (tantos como dimensiones).

Ya que MATLAB almacena como atributo de cada variable su tamaño y dimensiones, haciendo superfluos los métodos expuestos, se ha seguido el criterio de simplificar el patrón de llamada, consultando la información requerida antes de invocar a PVM/MPI.

Multiplicidad de patrones: MATLAB permite ahora invocar una función con un número variable de argumentos de entrada y de salida. Esta capacidad se ha utilizado sólo ocasionalmente para utilizar valores por defecto en algunas de las llamadas más habituales, liberando al usuario de especificar algún argumento de entrada. En general, se ha preferido respetar el patrón de llamada original para conservar el valor didáctico de las *Toolboxes* y/o las prestaciones del sistema de paso de mensajes.

Si surgiera la necesidad, podría contemplarse la coexistencia de múltiples patrones para cada llamada, siendo MATLAB capaz de determinar el patrón deseado según el número y tipo de los argumentos de entrada y salida requeridos en el código fuente del usuario. Esta multiplicidad está limitada únicamente por ambigüedad en la llamada: dos patrones deben distinguirse al menos en un tipo de datos o en el número de argumentos E/S.

Por ilustrar el concepto, el comando PVMTB info=pvm_upkdouble(vnam [,nitems [,stride]]) (en el cual ya son opcionales los dos últimos parámetros) se podría alterar de manera que si no se especifica vnam, el array desempaquetado se devolviera como primer argumento de salida, precediendo a info. Esta modificación permitiría redactar código MATLAB más compacto a costa de peores prestaciones. Los criterios para diseñar los posibles nuevos patrones se pueden establecer cuando el número de usuarios de PVMTB y MPITB aumente.

1.3.2 Trabajos relacionados

Como se ha comentado, sólo el paralelismo de media-alta granularidad puede producir *speedups* significativos bajo un entorno de simulación interpretado como MATLAB. Cleve Moler, el autor de la primera versión de MATLAB, coautor de diversos paquetes matemáticos (LINPACK, EISPACK) y cofundador y vicedirector de *The MathWorks*, la empresa fabricante de MATLAB, demostró [62] que la ejecución de prototipos paralelos de MATLAB en el *iPSC* de Intel y el *Titan* de Ardent conseguían ganancia en prestaciones sólo para las rutinas matemáticas, siempre y cuando el

tamaño del problema fuera lo suficientemente grande. De hecho, en el *Titan*, una máquina de memoria distribuida, la configuración utilizada disponía de tan poca memoria local (0.5 MB) que la ejecución paralela era realmente *más lenta* que la secuencial.

Además, una fracción del tiempo de ejecución de cualquier sesión se consume en el intérprete de comandos y en las rutinas gráficas, en donde es difícil encontrar paralelismo alguno. Otra desventaja a considerar es el relativamente pequeño número de usuarios en posesión de un supercomputador como los mencionados, lo cual termina de anular el hipotético interés comercial en desarrollar una versión paralela de MATLAB.

Recientemente las placas madre duales o multiprocesador se han abaratado hasta hacerlas relativamente comunes, y MATLAB, que fue desarrollado sobre los paquetes LINPACK y EISPACK, ha incorporado LAPACK [63], que puede beneficiarse de una biblioteca BLAS afinada para SMP. El desarrollo comercial de MATLAB parece apuntar a que el paralelismo de baja granularidad encontrado en rutinas BLAS puede ejecutarse ventajosamente en las varias CPUs de un SMP, mientras que el paralelismo de máxima granularidad, visible al programador de la aplicación MATLAB, debe ser extraído explícitamente por éste, pudiendo ejecutarse ventajosamente en las distintas CPUs de un cluster. En su artículo de 1995, Moler [62] denomina a esta granularidad "outer loop parallelism", paralelismo de bucle externo. Si una aplicación presentara ambas granularidades sería ideal su ejecución en un cluster de SMPs (constelación) con BLAS optimizado para las máquinas concretas.

Diversas *Toolboxes* paralelas de dominio público (Tabla 1.1) han sido construidas para posibilitar esta paralelización de gruesa granularidad ("del bucle más externo") bajo MATLAB, diferenciándose entre otros aspectos en el mecanismo de comunicación subyacente, el nivel de integración con los tipos de datos MATLAB, y el número de llamadas de paso de mensajes que recubren, en su caso:

- **Paralize** usa el sistema de ficheros compartido para pasar datos a los "servidores de cálculo". Los datos deben estar organizados como matriz tridimensional, y cada servidor se encarga de una submatriz bidimensional.

- **TMath** [22] es un interfaz Tcl* entre Ptolemy y MATLAB o Mathematica, desarrollado en la Texas University. Los procesos MATLAB son arrancados y manipulados como "motores MATLAB" (*engines*), usando el *engine API* de MATLAB. Se pueden arrancar motores MATLAB,

Tabla 1.1: *Toolboxes* paralelas para MATLAB. **cmds** número de comandos. ‡ disponibles en MATLAB Downloads [52]. † *Toolboxes* presentadas en esta memoria.

Toolbox	método	cmds	fecha	WWW / FTP
Paralize	filesys.	2	ene-99	ftp://ftp.mathworks.com/pub/contrib/v5/tools/paralize ‡
TMath	e.iface.	9	ago-97	http://www.ece.utexas.edu/~bevans/projects/tmath.html ‡
TCPIP	TCP/IP	12	abr-99	ftp://ftp.mathworks.com/pub/contrib/v5/tools/tcpip ‡
PMI	MPI	19	mar-99	ftp://ftp.mathworks.com/pub/contrib/v5/tools/PMI ‡
PT	PVM	24	apr-95	ftp://web.mthcsc.wfu.edu/pub/pt/
MultiMATLAB	MPI	27	nov-97	http://www.tc.cornell.edu/~anne/projects/MM.html
DP-TB	PVM	56	mar-99	http://www-at.e-technik.uni-rostock.de/dp
PVMTB†	PVM	93	jun-99	http://atc.ugr.es/~javier/pvmlog-eng.html
MPITB†	MPI	153	feb-00	http://atc.ugr.es/~javier/mpilog-eng.html

hacerles evaluar un guión (*script*) y obtener las variables resultado como pares ordenados de números.

- **TCPIP** es un interfaz MATLAB básico a los *sockets* TCP/IP, permitiendo abrir, escribir, leer y cerrar *sockets*. El usuario debe especificar dirección IP (host) y puerto. Implementa un sencillo protocolo para transmitir ficheros, y las variables se transmiten salvándolas primero en un fichero `.mat`.

- **PMI** recubre 8 llamadas MPI. Los procesos MATLAB se manejan como motores, usando la *engine API*.

- **PT** [32] fue desarrollada en la Wake Forest University. Consta de 24 comandos, admite tipos de datos enteros y doble precisión, y los procesos MATLAB se manejan como motores.

- **MultiMATLAB** [90, 59] es una *Toolbox* de alto nivel que evita al usuario tener que aprender MPICH, el mecanismo subyacente de paso de mensajes que utiliza. Se pueden arrancar y terminar sesiones remotas, se pueden enviar comandos a evaluar, distribuir los datos, recolectar los resultados, etc. Fue desarrollado en el Cornell Theory Center [16] de la Cornell University, con la cooperación de *The MathWorks*. La versión en la que continúan trabajando se denomina Cornell Multitask Toolbox for MATLAB, CMTM [13].

- **DP-Toolbox** [19, 76] recubre 44 llamadas PVM. Fue desarrollada en la Universität Rostock. Incluye una *Toolbox* de alto nivel (DPMM) formada por 15 comandos, que libera al usuario de aprender PVM, el mecanismo de paso de mensajes subyacente.

- **PVMTB** [24, 23] es un recubrimiento completo de PVM bajo MATLAB, incluyendo por ejemplo las llamadas `pvm_spawn()` y `pvm_tickle()`. También recubre las macros de máscaras `TEV_MASK_*()` y permite programar los *ganchos* (hooks) PVM en lenguaje MATLAB y registrarlos como funciones de recepción, gestor de recursos, gestor de tareas (*tasker*) o de nodos (*hoster*). Soporta todos los tipos de datos MATLAB.

- **MPITB** [23] es un recubrimiento prácticamente total del estándar MPI 1.2 bajo MATLAB, excluyendo básicamente las rutinas de tipos de datos (`MPI_Type_*()`, `MPI_Op_*()`) y `MPI_Pcontrol()`. También recubre las llamadas MPI 2.0 necesarias para arrancar procesos MATLAB e intercomunicarlos: `MPI_Comm_spawn_*()`, `MPI_Comm_get_parent()`, `MPI_Comm_accept()`, `MPI_Comm_[dis]connect()`, `MPI_*_port()`, `MPI_*_name()`, `MPI_Info_*()`. También soporta todos los tipos de datos MATLAB.

A continuación se incluye una descripción más completa de las dos últimas *Toolboxes* citadas, que son el objeto de esta memoria.

1.4 PVMTB

PVMTB es un acrónimo para "PVM *Toolbox*". Esta *Toolbox* permite utilizar el sistema PVM desde MATLAB. A diferencia de todos los intentos anteriores (Tabla 1.1), la cobertura del sistema PVM es total, contando con 93 comandos, frente a los 56 de DP-TB o 24 de PT.

La integración es completa, siendo compatible la utilización de PVMTB con las herramientas proporcionadas por PVM. Así, es posible arrancar MATLAB desde la consola PVM, o visualizar la traza de llamadas PVMTB bajo MATLAB desde el depurador XPVM. El propio *daemon* PVM puede arrancarse desde MATLAB, sin necesidad de usar la consola para este fin. Un proceso MATLAB puede conectarse al *daemon* PVM arrancado desde la consola, no siendo obligatorio arrancar el *daemon* desde MATLAB.

Todos los tipos de datos MATLAB, incluyendo los nuevos tipos estructurados de la versión 5 (**struct**, **cell**, etc) pueden ser transmitidos y recibidos usando PVMTB. Aunque MATLAB no documenta la implementación de sus tipos de datos, se ha conseguido integrar la opción de empaquetamiento PvmDataInPlace para los tipos no estructurados, ya que la suposición de que estaban almacenados contiguamente ha resultado ser correcta. Al no disponerse de información sobre la disposición en memoria de los tipos estructurados, no se puede utilizar empaquetamiento PvmDataInPlace con ellos, ya que se desconoce dónde reside la información necesaria para reconstruir la variable estructurada en el receptor.

Se contempla la utilización de la biblioteca de grupos y de las llamadas colectivas, siendo éste un rasgo distintivo de esta *Toolbox*, no contemplado en ningún trabajo anterior.

Todos los "ganchos" (*hooks*) que permiten al usuario alterar el comportamiento del sistema PVM están integrados en PVMTB, siendo pues posible para el usuario de PVMTB programar *en lenguaje* MATLAB una función de recepción propia, o un equilibrador de carga (*hoster*), o un servidor remoto de ejecución (*tasker*). Este rasgo también es exclusivo de esta *Toolbox*, no siendo contemplado en ningún trabajo previo.

PVMTB puede ser descargada desde la propia página Web del sistema PVM ubidada en el *Oak Ridge National Laboratory* http://www.epm.ornl.gov/pvm/, en la sección "Noteable PVM related software", o desde la página del autor, http://atc.ugr.es/javier-bin/pvmtb.

1.5 MPITB

MPITB es un acrónimo para "MPI *Toolbox*". Esta *Toolbox* permite utilizar el sistema MPI desde MATLAB. A diferencia de los (limitados) intentos anteriores, la cobertura del sistema MPI es prácticamente total, contando con 153 comandos, frente a los 19 de PMI o 27 de MultiMATLAB.

MPITB recubre todas las llamadas del estándar MPI 1.2 salvo MPI_Pcontrol(), MPI_Op_create(), MPI_Op_free(), y las relacionadas con tipos de datos: MPI_Type_*(). El motivo es doble:

Opacidad: El tipo mxArray es un objeto opaco. El API MATLAB no proporciona información sobre qué campos contiene el tipo mxArray, ni cómo se almacena en memoria; sólo proporciona llamadas para manipular variables de dicho tipo (encapsulado de datos). El conocimiento de dicha información permitiría a MPITB definir un tipo MPI_MXARRAY y diversas variantes MPI_CELL, MPI_STRUCT, etc, mediante las cuales podría sistematizar y optimizar la transmisión de variables MATLAB.

Alto nivel: Siendo MATLAB un lenguaje de muy alto nivel, carece de soporte para manejo de memoria a bajo nivel. El usuario MATLAB podría tal vez definir tipos MPI personalizados, pero no podría crear variables MATLAB que respetaran el patrón de almacenamiento (*data layout*) definido por dichos tipos MPI.

MPITB también porporciona acceso a algunas llamadas del estándar MPI 2.0:

Spawn: al objeto de poder arrancar otros procesos MATLAB desde un proceso inicial, se han implementado MPI_Comm_spawn(), MPI_Comm_spawn_multiple() y MPI_Comm_get_parent().

Objeto Info: ya que la llamada spawn requiere un objeto Info, se decidió implementar todas las llamadas MPI_Info_*()

Servicios: las llamadas orientadas a puertos MPI_Open/Close_port(), MPI_Comm_accept/[dis]connect() y MPI_[Un]Publish/Lookup_name(), permiten al usuario MATLAB redactar una aplicación servidora a la que otros procesos se podrían conectar para requerir un servicio.

MPITB puede ser descargada desde la propia página Web del equipo LAM, ubicada en la Universidad de Notredame (IN, USA), en la sección “Related MPI Software”, http://www.lam-mpi.org/software/, o desde la página del autor, http://atc.ugr.es/~javier/mpitb.

1.6 Comparación

Para ilustrar las diferencias entre las *Toolboxes* paralelas mencionadas previamente, se ha utilizado el problema propuesto en la “Parallel Processing HOWTO” ([17] Apartado 1.3 “Example Algorithm”): calcular π sin usar funciones trigonométricas, integrando numéricamente la derivada del arco tangente en [0,1].

$$\arctan(0) = 0, \quad \arctan(1) = \frac{\pi}{4}, \quad \arctan'(x) = \frac{1}{1+x^2}, \quad \int_0^1 \frac{1}{1+x^2} = \arctan(x)\Big|_0^1 = \frac{\pi}{4} - 0$$

Este problema ha sido utilizado para comparar otros sistemas de programación paralela (Quinn [80], por ejemplo). Para calcular el *speedup* de una aplicación paralela, se debe dividir el tiempo de ejecución de la versión secuencial óptima entre el de la versión paralela. La versión secuencial propuesta es:

```
#include <stdlib.h>
#include <stdio.h>
main(int argc, char **argv)
{
  register double width, sum;                /* la versión secuencial óptima */
  register int intervals, i;
  /* get the number of intervals */
  intervals = atoi(argv[1]);                 /* indicar #subdiv. como argumento */
  width = 1.0 / intervals;                   /* anchura de cada rectángulo */

  /* do the computation */
  sum = 0;
  for (i=0; i<intervals; ++i) {
    register double x = (i + 0.5) * width;   /* abscisa punto medio */
    sum += 4.0 / (1.0 + x * x);              /* ordenada/acum. 4* no hace perder */
  }                           /* ... prestaciones y mejora precisión acumulación */
  sum *= width;                              /* 4* estaría aquí y width en bucle */
  printf("Estimation of pi is %f\n", sum);
  return(0);
}
```

Listado 1.1: Versión secuencial utilizada para el cálculo de *pi*.

en donde se observa que se usa el método del punto medio, evaluando la función $\frac{1}{1+x^2}$ en los puntos centrales $x = \frac{i+0.5}{intervals}$ ($i = 0..$intervals-1), de las subdivisiones del intervalo [0,1]. Otros métodos, como la regla del trapecio, que interpola linealmente entre los puntos izquierdo y derecho, $\frac{f(i)+f(d)}{2}$, o la de Simpson, que utiliza los puntos izquierdo, medio y derecho, $\frac{f(i)+4f(c)+f(d)}{6}$, presentan una cota de error más baja (ver Figura 1.1 y Tabla 1.2).

Tabla 1.2: Fórmulas sencillas de cuadratura

Método	Fórmula	Error en tramo simple anchura $h = b - a$	Error compuesto N tramos de anchura $h = \frac{b-a}{N}$
punto izq.	$\int_a^b f(x)\,dx = f(a)\,(b-a)$	$R(f) = \frac{f'(\xi)}{2}\,(b-a)^2$	$R(f) = \frac{f'(\xi)}{2}\,(b-a)\,h$
punto der.	$\int_a^b f(x)\,dx = f(b)\,(b-a)$	$R(f) = \frac{-f'(\xi)}{2}\,(b-a)^2$	$R(f) = \frac{-f'(\xi)}{2}\,(b-a)\,h$
p. medio	$\int_a^b f(x)\,dx = f(c)\,(b-a)$	$R(f) = \frac{f''(\xi)}{24}\,(b-a)^3$	$R(f) = \frac{f''(\xi)}{24}\,(b-a)\,h^2$
trapecio	$\int_a^b f(x)\,dx = \frac{f(a)+f(b)}{2}\,(b-a)$	$R(f) = \frac{-f''(\xi)}{12}\,(b-a)^3$	$R(f) = \frac{-f''(\xi)}{12}\,(b-a)\,h^2$
Simpson	$\int_a^b f(x)\,dx = \frac{f(a)+4f(c)+f(b)}{6}\,(b-a)$	$R(f) = \frac{-f^{iv)}(\xi)}{2880}\,(b-a)^5$	$R(f) = \frac{-f^{iv)}(\xi)}{2880}\,(b-a)\,h^4$

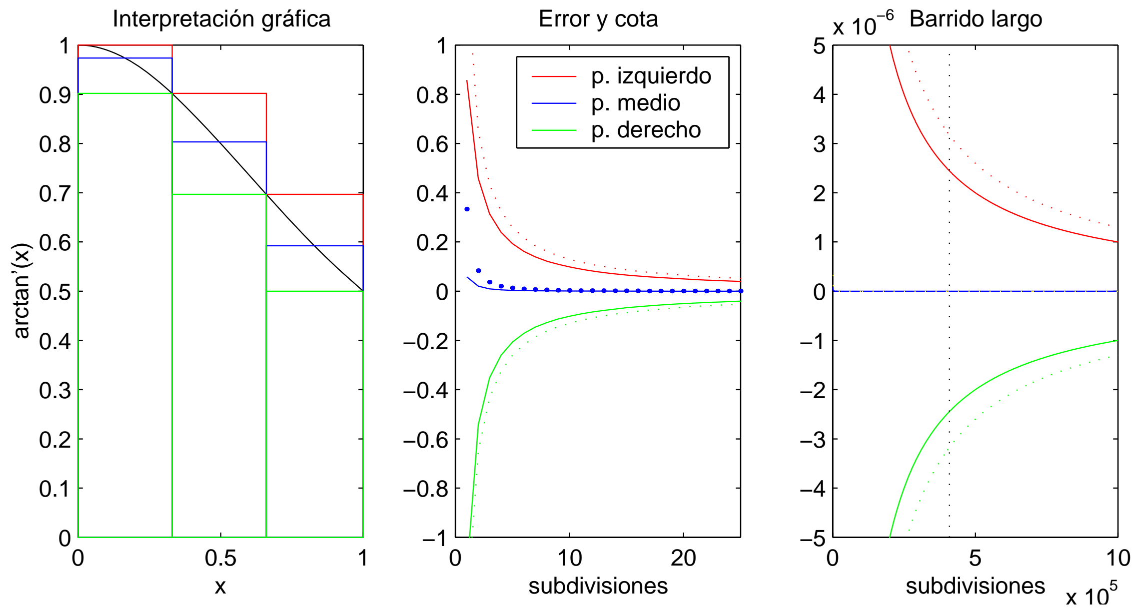
Aparentemente se ha escogido el método del punto medio considerando que sólo el método de Simpson converge más rápido, y dicho método implica dos productos y dos sumas adicionales, y en principio tres evaluaciones de la función en lugar de una. Sin embargo, el método de Simpson halla la aproximación **double** correcta de π con 101 subdivisiones del intervalo [0,1] en tan sólo 48μs (en un Pentium II 400MHz, Fig. 1.2(b)), tiempo imposible de mejorar mediante paso de mensajes. La latencia de un *switch* Ethernet 100Mbps es del mismo orden de magnitud (80μs [17]), de manera que la simple transmisión de un resultado parcial es más costosa que el cálculo completo. Cabe imaginar que la relación (latencia de interconexión/tiempo de cálculo) no era tan elevada cuando se propuso el problema por primera vez. Aún así usaremos este ejemplo como comparación, dada su popularidad.

El número escogido de subdivisiones del intervalo [0, 1] (línea vertical punteada en Fig. 1.1) es un compromiso entre la precisión deseada (**double**) y el error de acumulación (que también crece con las subdivisiones) conveniente a nuestro objetivo de incrementar el tiempo de cálculo hasta que supere con creces la latencia mínima Ethernet.

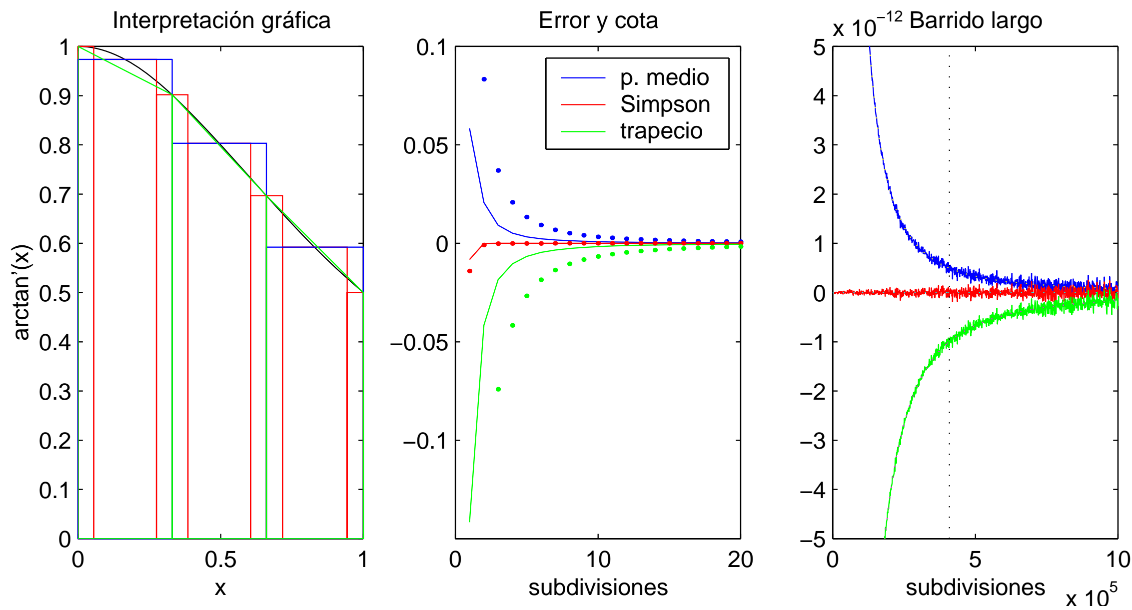
Se decidió que merecía la pena llegar a 13 dígitos significativos ($R(f) < 5\ 10^{-13}$), escogiendo un mínimo local alrededor de $4\ 10^5$ subdivisiones, introduciéndose ligeramente en la zona de rizado (Fig. 1.1(b) derecha). El rizado se debe a la acumulación de errores de ecualización (ver la discusión en Apéndice A). En la Figura 1.2(a) se muestra una ampliación de dicha zona, mientras que la Figura 1.2(b) muestra el reducido número de subdivisiones requeridas por el método de Simpson.

Los productos pierden bits de precisión por redondeo y la acumulación pierde cada vez más bits por ecualización, conforme la cantidad acumulada y la cantidad por acumular son progresivamente más dispares. El algoritmo propuesto se beneficiaría significativamente de un esquema de suma en árbol, en el cual las áreas se acumularan de dos en dos, ambas de similar orden de magnitud. La Figura 1.3 explica gráficamente la idea, y las Figuras 1.4 y 1.5 muestran el efecto de dicha mejora sobre los diversos métodos de integración.

En la comparación de las distintas *Toolboxes* paralelas utilizaremos el ejemplo original tal y

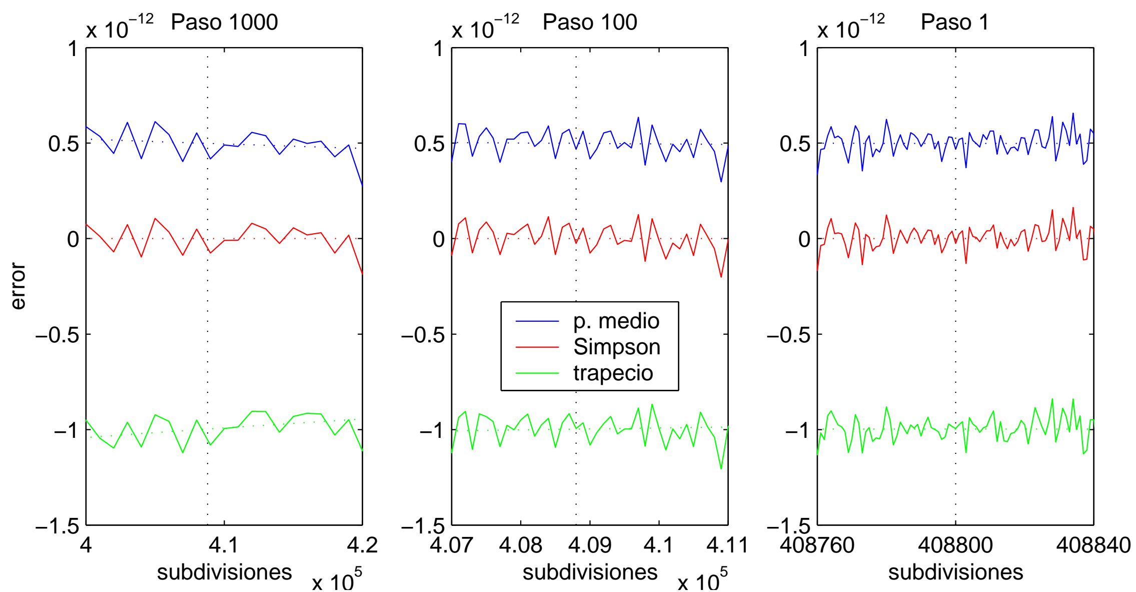


(a) Comparación de los métodos del punto medio, izquierdo y derecho.

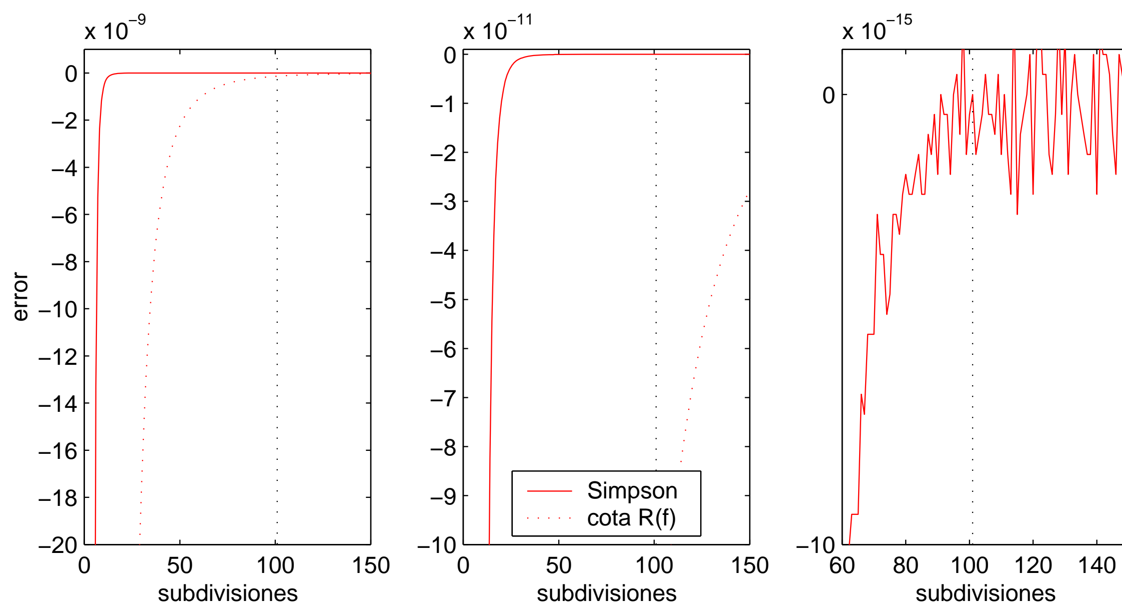


(b) Comparación de los métodos del punto medio, trapecio y Simpson.

Figura 1.1: Varios métodos de integración numérica aplicados a la derivada de $\arctan(x)$, $\frac{1}{1+x^2}$.



(a) Obtención de 13 dígitos significativos. Se escoge un mínimo local pasado el punto en el que la cota teórica de error baja de $5\ 10^{-13}$, en este caso $N = 408800$. Se muestran barridos con distintos incrementos de subdivisiones: cada 1000 subdivisiones, cada 100, e incrementando de 1 en 1. Obsérvese que la forma del error en función del número de sudvisiones prácticamente no depende del método, sino del orden de las operaciones (multiplicar por 4 antes que por **width**).



(b) Error 0. El método de Simpson obtiene la aproximación **double** perfecta de π con 101 subdivisiones, en tan sólo 48μs.

Figura 1.2: Elección del número de subdivisiones para integración numérica.

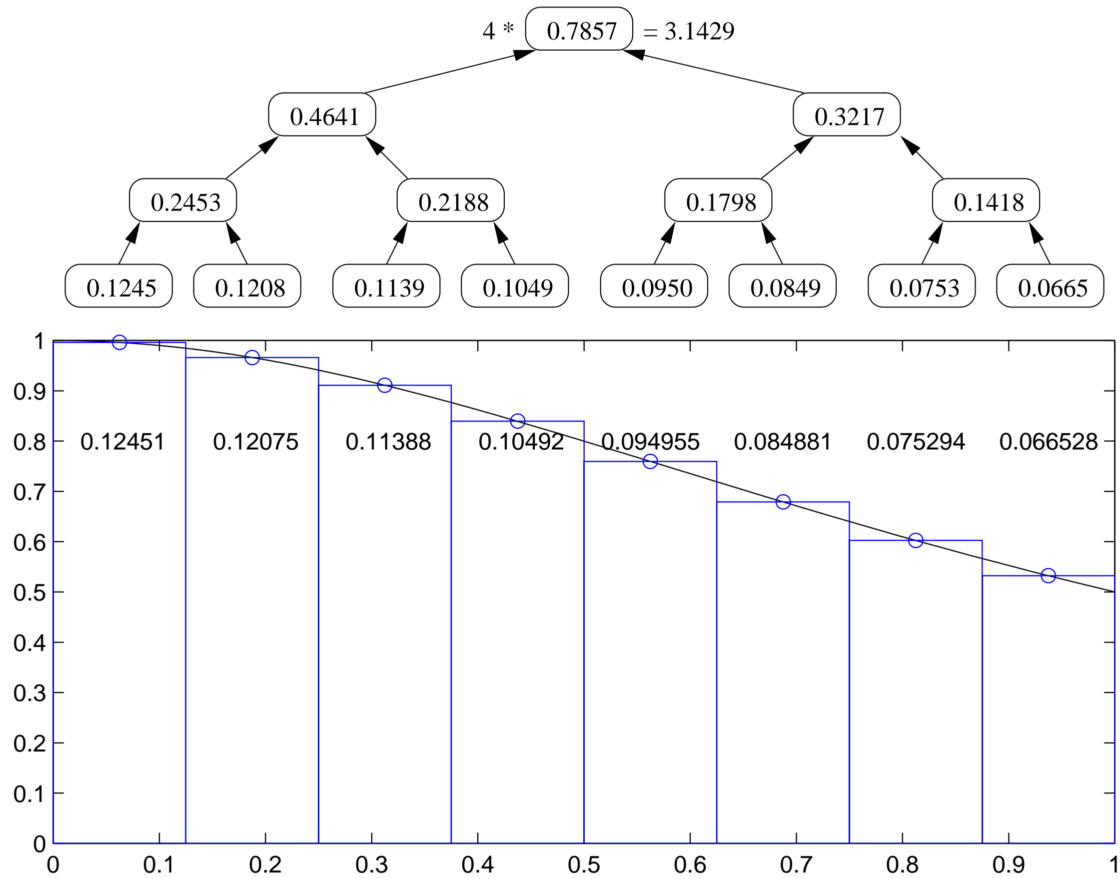
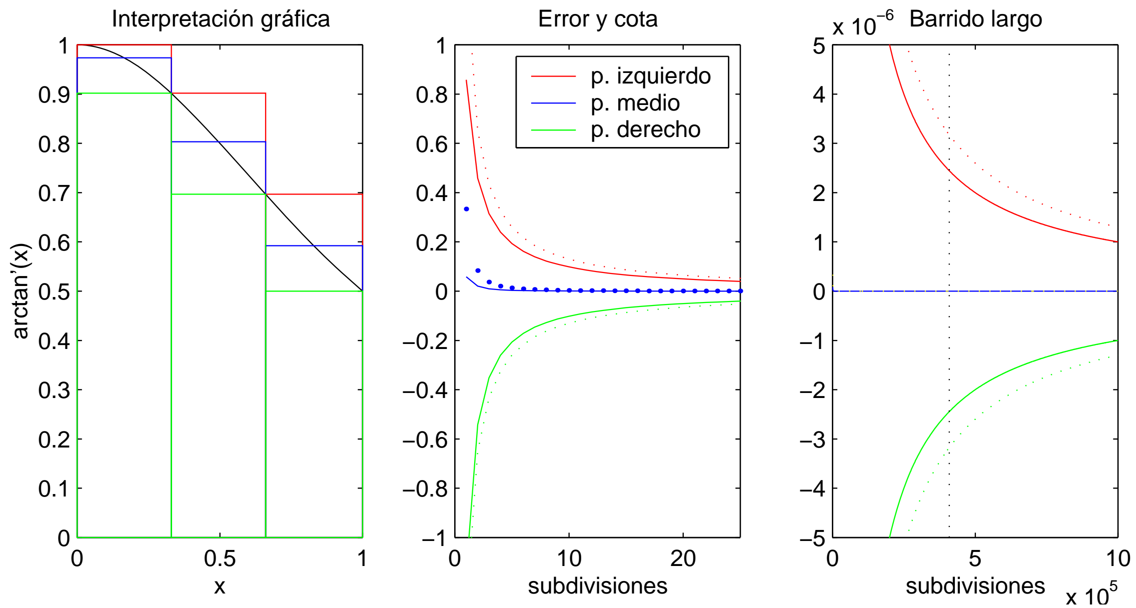
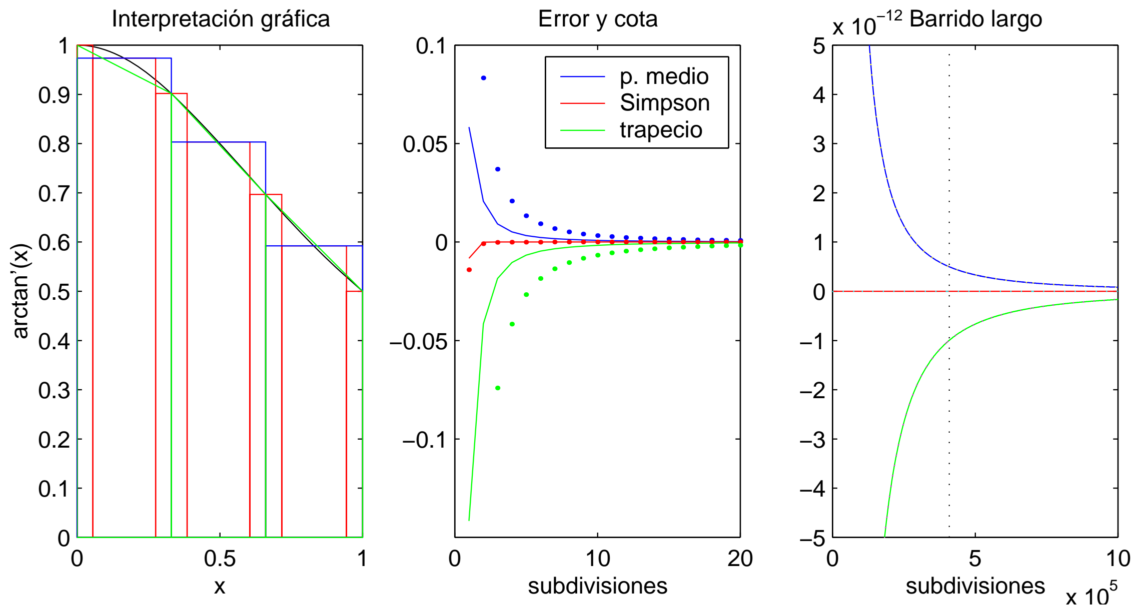


Figura 1.3: Al acumular mediante un esquema de sumas en árbol, las cantidades a sumar son similares y se elimina el rizado de ecualización. Si el número de subdivisiones es potencia de dos ($N = 2^n$) todas las cantidades a sumar son similares. En el caso general, la última suma de algunos niveles sufre ecualización. En el peor caso $N = 2^n + 1$, la última suma del árbol sufre toda la ecualización.

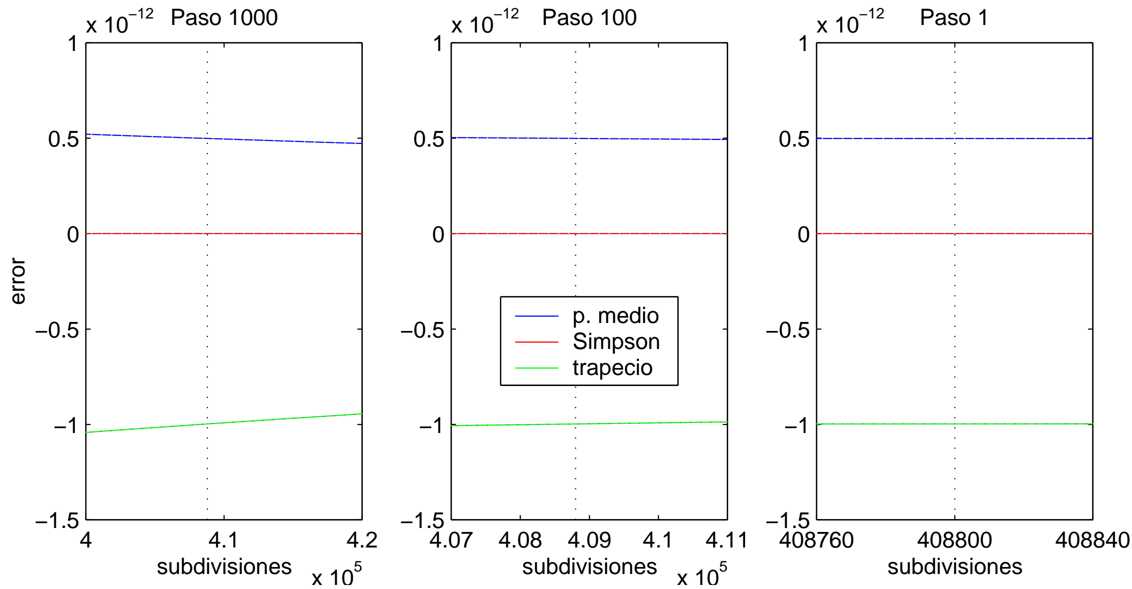


(a) Comparación de los métodos del punto medio, izquierdo y derecho.

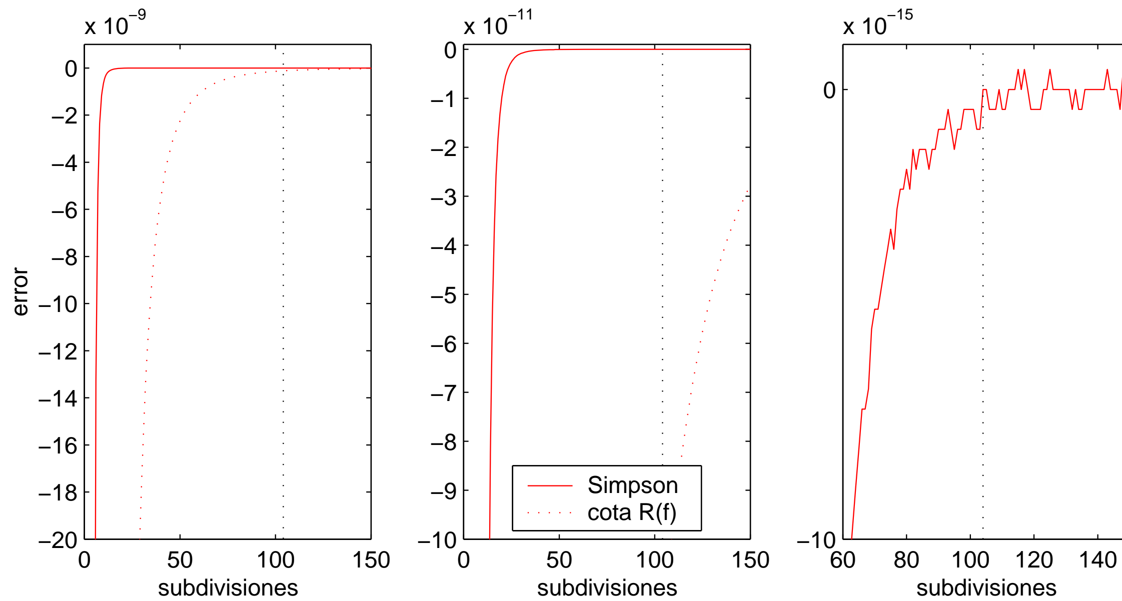


(b) Comparación de los métodos del punto medio, trapecio y Simpson. Obsérvese la eliminación del rizado comparando con la Figura 1.1(b)

Figura 1.4: Integración numérica de $\arctan'(x)$ usando sumas en árbol.



(a) Obtención de 13 dígitos significativos. Compárese con el rizado de la Figura 1.2(a).



(b) Error 0. El método de Simpson obtiene la mejor aproximación **double** de π con 104 subdivisiones, en tan sólo 63μs. A partir de 104, los errores son $\pm 4.44\ 10^{-16}$, al límite de la resolución **double**. Las 101 subdivisiones de la Figura 1.2(b) se deben a un afortunado error.

Figura 1.5: Número de subdivisiones requeridas usando sumas en árbol.

como se encuentra en la literatura, permitiéndonos esta decisión escoger un número de subdivisiones tal que se obtengan *speedups* superiores a la unidad.

Para programación por paso explícito de mensajes, se debe descomponer el código en tramos secuenciales a ejecutar en cada procesador. El problema que nos ocupa es *embarazosamente paralelo**, admitiendo una paralelización trivial. Si C procesos (numerados de 0 a $C-1$, por fijar conceptos) cooperan para calcular la integral subdividiendo el intervalo [0,1] en $N \gg C$ subdivisiones (numeradas de 0 a $N-1$), cada uno acumula las áreas de las subdivisiones a distancia C empezando por la subdivisión que coincide con su índice de proceso. De esta manera, el proceso p (con $0 \leq p < C$) calcula las subdivisiones de índice i congruente p módulo C, $p \equiv i \pmod{C}$. Si N no es múltiplo de C ($N \not\equiv 0 \pmod{C}$), los primeros $(N \bmod C)$ procesos calculan una subdivisión más.

En los Apartados siguientes se programa este algoritmo paralelo usando PVM y MPI en lenguaje C, y las distintas *Toolboxes* en lenguaje MATLAB. Al objeto de ponderar la ganancia en velocidad se realiza un estudio de escalabilidad, consistente en determinar la evolución del *speedup* conforme se aumenta el número de procesadores que cooperan. También se realiza un test *ping-pong* destinado a evaluar las prestaciones del sistema de paso de mensajes subyacente.

1.6.1 Programas C

En la "Linux Parallel Processing HOWTO" [17] se muestran versiones paralelas del problema propuesto en lenguaje C usando PVM (Sección 3.4 de la HOWTO) y MPI (Sección 3.5) al objeto de comparar el código paralelo con la versión secuencial. En nuestro estudio compararemos también con las versiones MATLAB usando algunas de las *Toolboxes* paralelas mencionadas, y mediremos las ganancias en prestaciones obtenidas bajo cada sistema en nuestro cluster.

Al objeto de comparar todos los sistemas en unas condiciones lo más similares y equitativas posibles, se añadieron las indispensables instrucciones para medir el tiempo transcurrido desde que se emiten los argumentos hasta que se obtiene el valor de π. Cada sistema emplea métodos distintos para preparar el cluster y arrancar los procesos paralelos, siendo el único divisor común que los procesos paralelos empiezan a calcular en cuanto disponen de los datos requeridos.

En una aplicación real se debería considerar el tiempo total de ejecución de la aplicación, incluyendo los tiempos de carga en memoria e inicialización, tanto en la versión secuencial como en la paralela. Para nuestro objetivo resulta más ilustrativo considerar únicamente el tiempo de cálculo y de comunicación, ignorando los tiempos de arranque e inicialización. Se trata de una aplicación muy simple (*toy example*) con un tiempo de computación muy reducido, frente al cual los tiempos de inicialización del sistema de paso de mensajes (PVM o MPI) y sobre todo del entorno MATLAB resultan onerosos.

Dado que nuestro cluster (ver el Apartado 2.1 más adelante) dispone de un servidor a 400MHz y 8 clientes a 333MHz, se ha optado por medir el tiempo de cálculo del programa secuencial tanto en el servidor como en uno de los clientes. Respecto al programa paralelo, se ha preferido usar el servidor sólo para arrancar los esclavos y acumular el resultado, sin colaborar en su cálculo. Como se ha mencionado previamente, se mide el tiempo desde que se emiten los argumentos a los procesos esclavos hasta que se obtiene el valor de π.

Se calculan entonces dos *speedups*: el relativo al tiempo secuencial en el servidor ilustra hasta qué punto el uso conjunto de varias máquinas puede superar a otra máquina de mayores

prestaciones; el relativo al tiempo secuencial en un cliente ilustra la escalabilidad casi lineal del programa. Aunque no todas las aplicaciones paralelas escalan bien, se asume que las embarazosamente paralelas deben aproximarse mucho a escalabilidad lineal.

Usando esta segunda métrica, si no existiera ninguna comunicación entre los procesos la aplicación exhibiría un *speedup* igual al número de computadores usados. Así pues, este *speedup* permite evaluar cómodamente el coste relativo de las comunicaciones en función del número de computadores usados, es decir, cómo de bien escala un algoritmo o programa concreto. Para ello basta con comparar el *speedup* con el número de computadores usados. En este Capítulo lo usaremos para comparar los méritos relativos de las distintas *Toolboxes*.

La primera métrica también tiene un interés estratégico. Periódicamente podría probarse un computador de manufactura reciente en lugar del servidor, para realizar el estudio de escalabilidad de esta y otras aplicaciones. Un bajo valor de esta segunda métrica indicaría cuándo ha llegado el momento de ampliar el cluster, añadiendo ordenadores del tipo del servidor utilizado. Un valor extremadamente bajo sugeriría retirar del cluster los esclavos utilizados en el estudio. En un cluster que ya haya sido previamente ampliado, contando por tanto con computadores de distintas prestaciones, el estudio podría hacerse separadamente para cada grupo de computadores de las mismas prestaciones.

Programa PVM

Es posible diseñar aplicaciones SPMD* en PVM. Por lo general su redacción es más sencilla y compacta que la correspondiente versión Master-Slave*: el código queda simplificado al eliminarse la necesidad de arrancar el daemon PVM (usando pvm_start_pvmd()) y las copias adicionales (usando pvm_spawn()) desde la copia inicial. Las diversas copias del programa SPMD pueden ser lanzadas usando el comando spawn –(count) de la consola PVM. Sin embargo, la consola PVM no tiene forma de detectar programadamente la finalización de una aplicación SPMD. En función de los posibles mensajes por pantalla, o usando el comando ps, el usuario detecta interactivamente que la aplicación ha terminado.

La versión del cálculo de π ofrecida en la “HOWTO” es SPMD. Para poder automatizar el estudio de escalabilidad se rediseñó la aplicación, dividiéndola en un programa maestro encargado de arrancar los esclavos y acumular sus resultados, y un programa esclavo que realiza el cálculo (Apéndice A, Listados A.4 y A.5). Esta división permite arrancar el programa maestro desde el *shell* sin usar la consola, y esperar a su terminación programadamente.

Un *script* (Listado A.3) permite automatizar las mediciones de tiempo. Se anota en un fichero de resultados el número de computadores clientes solicitado para el estudio de escalabilidad. El *script* ejecuta primero el programa secuencial en el servidor y, usando rsh, en un cliente, anotando en ambos casos el valor de π obtenido y el tiempo requerido en el fichero de resultados. Se crea entonces un fichero de hosts PVM al que se va añadiendo en cada iteración un computador cliente adicional, proporcionándoselo posteriormente a la consola PVM. En cada iteración se arranca el programa maestro en el servidor, quien a su vez arranca los esclavos, acumula el resultado, mide el tiempo transcurrido y lo anota en el fichero de resultados.

Se prefiere parar y rearrancar el *daemon* PVM en cada iteración para no perturbar el algoritmo *round-robin* seguido por pvm_spawn(). El sistema PVM memoriza a cuál procesador se le asignó la última tarea, y el siguiente pvm_spawn() comenzaría por el siguiente computador de la lista.

En esta aplicación, parte de la información requerida por los esclavos (número de esclavos C, número de subdivisiones N) es común y puede proporcionarse como argumentos al propio programa esclavo. Para asignar el índice i de cada tarea esclava y acumular en el maestro las sumas parciales se pueden utilizar tres alternativas de paso de mensajes:

send-recv: Las llamadas básicas punto a punto. Requieren inicialización del buffer de envío pvm_initsend() y (des)empaquetamiento pvm_[u]pk<type>(). Se puede utilizar la codificación PvmDataInPlace, ya que no se realiza ninguna operación posterior sobre la suma local en los esclavos.

psend-precv: Como ya se comentó, estas llamadas evitan el uso de pvm_initsend() y pvm_[un]pack(). Podemos usarlas dado que sólo devolvemos una variable (de un único tipo).

reduce: Al objeto de utilizar operaciones colectivas, el *script* arranca el *daemon* servidor de grupos previamente. De no hacerlo así, el *daemon* pvmgs sería asignado a un procesador siguiendo el algoritmo *round-robin* citado previamente. Para nuestra aplicación resulta interesante ejecutar el *daemon* en el computador servidor, que no coopera en el cálculo. El uso de grupos también evita tener que asignar el índice de tarea i, pudiéndose utilizar en su lugar el número de instancia que cada tarea obtiene al unirse al grupo usando pvm_joingroup().

Para cada uno de los tres métodos, se ha probado tanto el encaminamiento directo como a través del *daemon*. El código paralelo más compacto corresponde a la operación colectiva pvm_reduce(). El fragmento relevante del programa maestro (Listado A.4) es:

```
inum=pvm_joingroup("pi");                              /* Garantizar que somos inum==0 */
numt=pvm_spawn("pvmWork", argv, PvmTaskHost+PvmHostCompl, ".", C, NULL);
pvm_freezegroup("pi", C+1);                            /* Esperar hasta que haya C esclavos */

PT = Wtime();                         /* Medir tras sincroniz. inicial (equivale a MPI_Init) */
  Psum=0;
  pvm_reduce(PvmSum, &Psum, 1, PVM_DOUBLE, TAG, "pi", 0);
  Psum/=N;                                             /* width=1/N */
PT = Wtime()-PT ;                                      /* Medir en cuanto pi esté calculado */

pvm_barrier("pi", C+1);                                /* ver Caveat en man pvm_reduce */
pvm_lvgroup("pi");                                     /* Salir */
pvm_exit();
pvm_halt();
```

Listado 1.2: Versión paralela PVM: código maestro.

Se arrancan los esclavos pvmWork pasándoles los propios argumentos del maestro: el número de computadores esclavos C y el de subdivisiones N. Se especifica que se deben arrancar C copias en computadores distintos (PvmHostCompl) del actual ("."). Se realiza entonces una sincronización (pvm_freezegroup()) entre los $1 + C$ procesos (maestro+esclavos) antes de empezar a medir el tiempo del bucle de cálculo, para garantizar que todos los procesos están cargados en memoria y listos para ejecutar el bucle. El maestro sólo se ocupa de acumular en Psum los resultados parciales, medir el tiempo transcurrido y salir de PVM. La posterior barrera, recomendada por la página de manual, es para evitar el hipotético caso de que un esclavo acabara su cálculo y saliera del grupo antes de que el maestro iniciara la operación colectiva.

El código relevante del programa esclavo (Listado A.5) se reduce a:

```
inum=pvm_joingroup("pi");                       /* índice de tarea >= 1 */
pvm_freezegroup("pi",C+1);                      /* Sincronización inicial */

  width = 1.0/N; lsum = 0;
  for (i=inum-1; i<N; i+=C) {
    register double x = (i + 0.5) * width;
    lsum += 4.0 / (1.0 + x * x);
  }
sum = lsum;                                     /* lsum es register */
pvm_reduce(PvmSum, &sum,1,PVM_DOUBLE, TAG,"pi",0);
pvm_barrier("pi",C+1);                          /* ver Caveat en man pvm_reduce */
pvm_lvgroup("pi");
pvm_exit();
```

Listado 1.3: Versión paralela PVM: código esclavo.

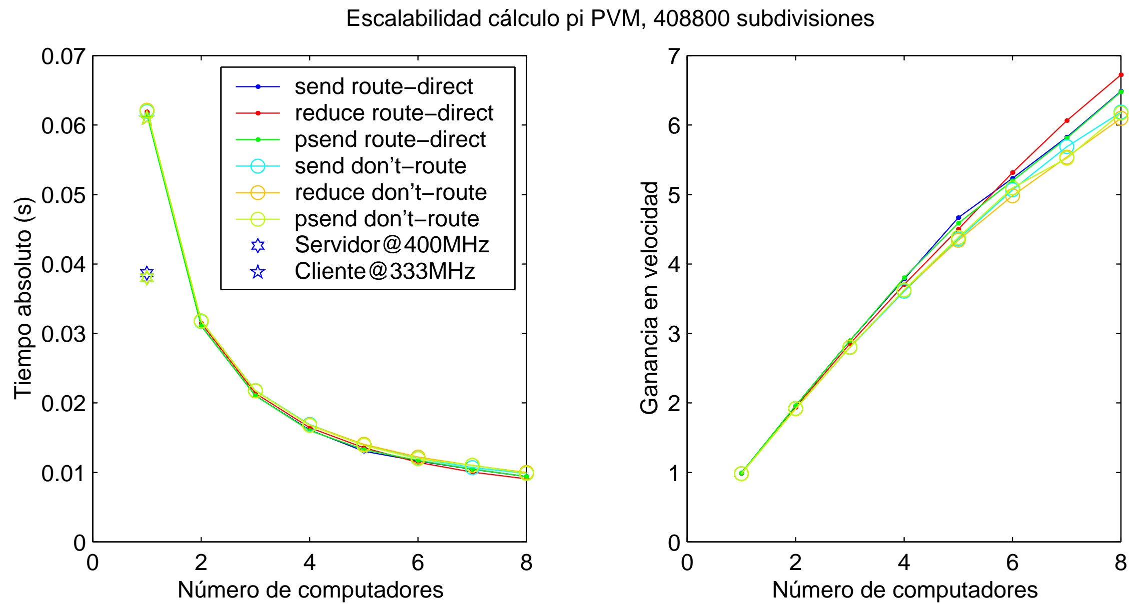
Tras sincronizarse con el maestro, que empieza a contabilizar tiempo, el bucle de cálculo acumula las subdivisiones correspondientes a este proceso inum, que, como se comentó, son todas las congruentes $inum-1 \equiv i \pmod{C}$, esto es, $inum-1$, $inum-1+C$, $inum-1+2C\ldots$

La operación de reducción requiere especificar la posición de memoria local a reducir con las otras instancias (y el tipo de datos, número de elementos y operación a usar en la reducción). Como se usó una variable **register** para acelerar el cálculo, se debe copiar ahora el resultado a memoria (sum=lsum;). Se indica también la etiqueta común de mensaje, el grupo y la instancia maestra que recibirá el resultado de la reducción.

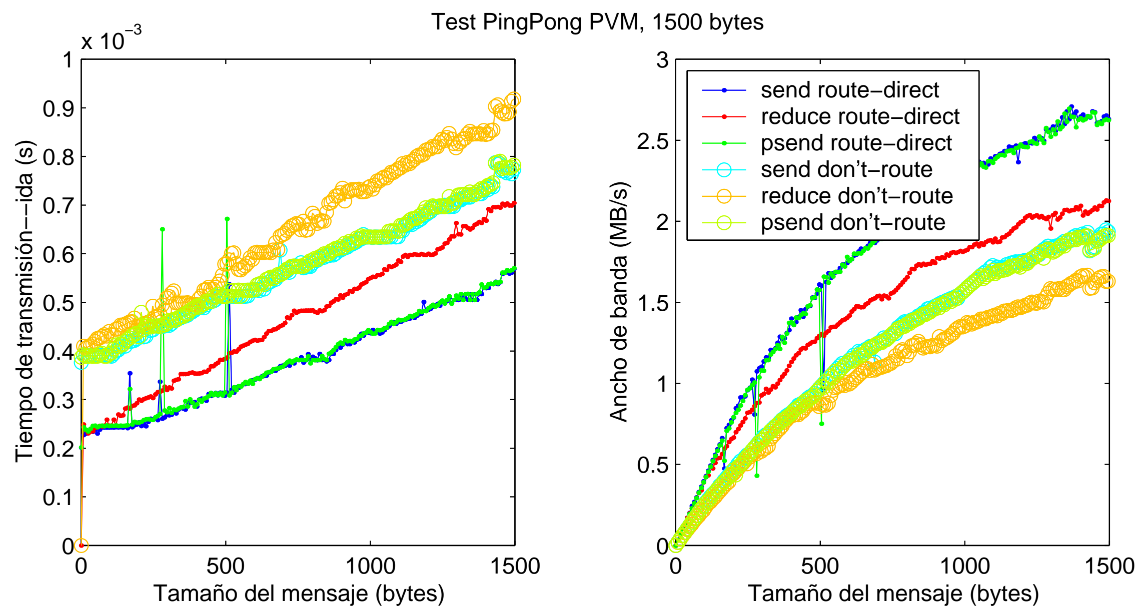
Esta aplicación concreta ha admitido una paralelización trivial, como se observa al comparar el código esclavo con el secuencial. La escalabilidad, o ganancia en velocidad conforme aumenta el número de procesadores, depende del coste *relativo* del paso de mensajes respecto al tiempo de cálculo, y por tanto de las distintas alternativas de comunicación y encaminamiento probadas. Siendo ésta una aplicación embarazosamente paralela, con muy poca comunicación en comparación con el tiempo de cálculo, no cabe esperar diferencias muy significativas entre las distintas alternativas. Con todas ellas debe obtenerse una escalabilidad casi lineal, como queda reflejado en la Figura 1.6(a). A la izquierda se muestran los tiempos de la aplicación paralela en función del número de computadores. Cada trazo corresponde a una alternativa distinta. Se muestran también los tiempos del programa secuencial ejecutado en el computador servidor (etiquetado $*$) y cliente ($\star$). Aunque son bastante reproducibles, las mediciones secuenciales se han repetido con cada alternativa, utilizándose para calcular el *speedup* la media de los tiempos secuenciales en el cliente. Dicho *speedup* se muestra en la gráfica a la derecha.

El encaminamiento a través del *daemon* permite solapar comunicación y cómputo a costa de una copia de memoria adicional, siendo esto beneficioso para algunas aplicaciones, aunque no sea el caso para ésta. En la Figura 1.6(a) derecha se observa mayor *speedup* con encaminamiento directo (superior a 6.5) que a través de *daemon* (superior a 6). Notablemente, la mejor opción es reducción con ruta directa, y la peor es reducción con encaminamiento a través del *daemon*, al menos a partir de 4 computadores. Más adelante se ofrece una explicación de esta peculiaridad.

Para comparar entre sí los diversos modos de comunicación y encaminamiento resulta más apropiado un test *ping-pong*, en donde se muestra el tiempo de transmisión (coste *absoluto* del paso de mensajes) en función del tamaño del mensaje (Figura 1.6(b)). Un mensaje de tamaño creciente se envía y recibe múltiples veces (en este caso usamos $N = 10$ repeticiones) entre un proceso iniciador o "fuente" y otro que responde, o "eco". Se cronometra la duración del bucle,



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.6: Tests realizados bajo PVM.

confiando en que al quedar el coste de inicialización del bucle repartido entre las N iteraciones resulte insignificante en comparación con el costo de la transmisión/recepción. El tiempo cronometrado se divide entre $2N$, contabilizando por tanto el tiempo medio que se tarda en transmitir un mensaje entre dos computadores, incluyendo la recepción y desempaquetamiento en el computador destino. La aplicación del estudio de escalabilidad transmite muy poca información, por lo cual se ha reducido el barrido a un tamaño de 1500 bytes. En el Capítulo 2 (Análisis de Prestaciones) se realizan tests más exhaustivos.

Con el modo reduce se ha utilizado la operación de suma, igual que en la aplicación del estudio de escalabilidad. Se ha tenido cuidado en arrancar siempre el proceso servidor de grupos pvmgs en el computador servidor del cluster. Se realizan alternativamente reducciones sobre el proceso fuente y eco.

El principio del barrido (Figura 1.6(b) izquierda) muestra que el tiempo de transmitir un **double** (8 bytes) es prácticamente indéntico al coste de inicialización de la comunicación (*setup*) independientemente de la modalidad y encaminamiento escogidos. El tiempo de comunicación en el estudio de escalabilidad queda por tanto dominado por el tiempo de *setup*, ya que los mensajes de vuelta son **double**s aislados. Los modos [p]send requieren además enviar un entero (índice de proceso) a cada uno de los procesos esclavos (transmisión etiquetada como /* *Asignar inum* */ en el Listado A.4). En el modo reduce los procesos esclavos disponen de esa información desde que se unen al grupo con pvm_joingroup(). Esta diferencia contribuye a la superioridad de la reducción, al menos con encaminamiento directo.

Al utilizar el encaminamiento a través del *daemon* el coste del *setup* casi se duplica, lo cual indica que el costo de la copia de memoria adicional es similar (ligeramente inferior) al de inicialización de la comunicación con encaminamiento directo. Estudiando la progesión hasta 1500 bytes, se observa que el coste de dicha copia es aproximadamente constante para el rango de tamaños barrido (crece, pero muy gradualmente). Los trazos [p]send (verdes/azules) son paralelos entre sí, al igual que los trazos reduce (rojo-naranja). La distancia entre ellos es el costo de la copia de memoria adicional debida al encaminamiento a través del *daemon*.

Los modos [p]send del test *ping-pong* son indistinguibles entre sí tanto con encaminamiento directo como a través del *daemon*. El modo reduce presenta un costo adicional en ambos casos. Volviendo a consultar la Figura 1.6(b) izquierda al principio del barrido, se puede observar la progresiva distancia entre el trazo verde oscuro y el rojo, similar a la distancia entre el trazo verde claro y el naranja.

Estas peores prestaciones del modo de reducción son básicamente atribuibles a la sentencia sbuf=pvm_mkbuf(PvmDataDefault) en el código fuente de la rutina pvm_reduce, que se puede encontrar en el fichero <PVMROOT>/pvmgs/pvmgsu_aux.c. Esto fuerza una (de)codificación XDR en (recepción y) envío para todas las operaciones de reducción. Comparando el modo reduce con los [p]send, se comprueba que el tiempo empleado para realizar esta conversión es (casi) lineal con el tamaño, y en concreto para 1500 bytes casi iguala el costo de la copia de memoria. La tendencia invita a pensar que sigue creciendo linealmente, pasando pronto a dominar en el tiempo total de la reducción. Para el encaminamiento a través de *daemon* la linealidad falla cerca del tamaño 0 (el coste adicional no es 0). Realizar la reducción sobre más procesos (se llega hasta 8 en el estudio de escalabilidad) sólo empeora la diferencia, ya que se ha de realizar la misma (de)codificación XDR sobre más buffers.

Esta diferencia entre los costes adicionales de reduce sobre [p]send cerca del tamaño 0 con

encaminamiento a través del *daemon* colabora en la explicación de los aparentemente contradictorios resultados en el estudio de escalabilidad, donde reduce era la mejor opción usando encaminamiento directo y la peor a través del *daemon*. El otro factor ya mencionado es que al utilizar en la aplicación la operación de reducción nos ahorramos el bucle de envío de los índices de proceso y la recepción por parte de los esclavos de dicho índice.

Las llamadas [p]send no son bloqueantes, en el sentido de que su retorno no garantiza que el dato haya sido desempaquetado en el receptor, sino que el buffer de envío puede ser reutilizado sin afectar a la transmisión. De esta forma, algo del coste de las C transmisiones de índice queda oculto, siendo el total menor que C tiempos de *setup*. Esto no queda de manifiesto en el test *ping-pong* porque allí se realizan transmisiones completas alternativamente en cada sentido, incluyendo los obligados pasos de (des)empaquetamiento. Al usar encaminamiento directo, cada envío retorna en cuanto ha sido recibido por el proceso esclavo, aunque no haya sido desempaquetado aún. Con encaminamiento a través del *daemon* el [p]send puede retornar en cuanto se realiza la copia de memoria al *daemon*, siendo responsabilidad de éste hacer progresar la transmisión. En ambos casos, se pasa inmediatamente a bloquearse en el [p]recv, de manera que lo que influye realmente es lo temprano que puedan empezar los esclavos a calcular su suma parcial. Con el uso de grupos, los esclavos disponen de toda la información antes incluso de que el maestro empiece a cronometrar el tiempo. Correspondientemente el *speedup* es mayor, al menos con encaminamiento directo.

A pesar de que al usar grupos no se requiera la recepción inicial del índice de proceso, ya que éste se obtiene al unirse al grupo y el tiempo empieza a cronometrarse después, el coste adicional de reduce a través del *daemon* va creciendo conforme aumenta el número de procesos sobre los que se reduce, llegando a compensar la recepción inicial de índices por parte de los esclavos a partir de unos 4 procesos, según se observa en la Figura 1.6(a) derecha.

Programa MPI

En lugar de consola como la del sistema PVM, las implementaciones MPI proporcionan una serie de herramientas para controlar y monitorizar el entorno de ejecución. Así, LAM proporciona lamboot para arrancar los daemon, mpitask y mpimsg para listar las tareas MPI en ejecución y los mensajes en curso, el acostumbrado mpirun para arrancar procesos desde el *shell*, etc. En concreto, la versión LAM de mpirun soporta los llamados “esquemas de aplicación” (*application schema*), ficheros de texto legible en donde se especifica la asignación de los procesos de la aplicación a los procesadores de los computadores disponibles (tal vez SMPs). Con esta sintaxis es posible arrancar cualquier aplicación MPMD*. Naturalmente LAM también soporta la habitual sintaxis SPMD de mpirun, en la que sólo se especifica el programa y número de procesos a arrancar.

Esta característica de LAM hace igualmente cómodos los estilos de programación SPMD y Master-Slave, aunque en general el comando mpirun de otras implementaciones MPI tienden a favorecer (o incluso imponer) el estilo SPMD. La ventaja de LAM para el ejemplo que nos ocupa, en comparación con la consola PVM, es que por defecto mpirun espera hasta la finalización de la aplicación arrancada por él (el modificador −nw permite retornar en cuanto arranca la aplicación). La aplicación se ha redactado en estilo SPMD, aunque los tramos maestro y esclavo están claramente diferenciados en el Listado A.7.

Se desarrolló por tanto un *script* (ver Listado A.6) similar al del Apartado previo, para automa-

tizar las mediciones de tiempo y generar un fichero con el mismo formato, conteniendo el número de computadores usados (C), el valor de π y tiempo de cálculo requerido por el programa secuencial en el computador servidor y en un cliente, y los valores de π y tiempo requeridos por el programa paralelo usando un número creciente ($1 \ldots C$) de clientes.

LAM reinicia el algoritmo *round-robin* de asignación de procesos a procesadores con cada invocación de mpirun y cada llamada a MPI_Spawn(). En comparación, PVM recuerda dónde se lanzó el último ejecutable, y la siguiente llamada pvm_spawn comienza asignando procesos por el siguiente procesador de la lista, motivo por el cual se prefirió reiniciar PVM tras cada ejecución en el apartado anterior, garantizando que pvmMast se ejecutara siempre en el ordenador servidor. Bajo LAM, el proceso con rango 0 (correspondiente al tramo maestro en nuestra aplicación) se ejecutará siempre en el primer ordenador listado en el fichero de configuración lamboot (el servidor, en nuestro caso).

Esto permite arrancar con lamboot el MultiComputador LAM conteniendo todos los computadores requeridos (servidor y clientes) y ejecutar tanto el programa secuencial en servidor y un cliente como el programa paralelo en un progresivo número de clientes usando en todos los casos la sintaxis SPMD de mpirun.

Dos de los datos requeridos por los esclavos en esta aplicación pueden obtenerse mediante llamadas MPI (índice de proceso con MPI_Comm_rank(), número de esclavos+1 con MPI_Comm_size()), mientras que el número de subdivisiones N puede especificarse como argumento al programa. El rango MPI (MPI_Comm_rank()) es equivalente al nº de instancia PVM (inum), tomando valores desde 0 a C. Si el rango no es nulo, se trata de un proceso "esclavo" que debe colaborar en el cálculo. El tramo maestro sólo necesita conocer su rango MPI (no necesita C ni N) para detectar su condición de maestro y acumular el resultado usando:

send-recv: Las llamadas básicas punto a punto. En este caso sí se necesita C para iterar el bucle de acumulación.

reduce: Operación colectiva sincronizante. Consultando el código fuente para esta llamada MPI en el fichero <LAMHOME>/share/mpi/reduce.c se observa que LAM pasa de un algoritmo lineal a uno logarítmico usando sumas en árbol a partir de LAM_COLLMAXLIN==4 procesos. Esta constante se define en <LAMHOME>/h/mpisys.h.

Para ambos métodos, se ha probado tanto el encaminamiento cliente a cliente como a través del *daemon*. El programa SPMD completo se muestra en el Listado A.7. Contiene una ramificación condicional separando el código maestro del esclavo. El fragmento relevante del código maestro se muestra en el Listado 1.4.

En comparación con el código PVM, no hemos necesitado arrancar los procesos esclavos programadamente, siendo realizada esta tarea por el comando mpirun en el *script* de automatización. Los procesos arrancados forman un grupo estático accesible mediante el comunicador MPI_COMM_WORLD, no siendo necesario crear o unirse a un grupo arbitrario.

A diferencia de la reducción PVM, en MPI no se requieren sincronizaciones adicionales, como una barrera anterior o posterior, o una etiqueta de mensaje, pero se requiere un buffer destino separado, distinto del fuente. Existe un código de error MPI_ERR_BUFFER reservado para indicar el no cumplimiento de esta condición.

El Listado muestra también la alternativa punto a punto, en la que se acumulan los resultados

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size); C=size-1; /* Master no trabaja */

if (rank){                                  /* tramo esclavo */
  ...
} else {                                    /* tramo maestro */
  PT = MPI_Wtime();                         /* MPI_Init ya sincronizó */
  P2=Psum=0;                                /* fuente/destino reducción */
  switch (MOD){
  case 'r': MPI_Reduce(&P2,&Psum,1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
            break;
  case 's': for (i=1;i<size;i++){
              MPI_Recv(&P2,1, MPI_DOUBLE, MPI_ANY_SOURCE, TAG, MPI_COMM_WORLD,&st);
              Psum+=P2;                     /* acumular */
            }
            break;
  }
  Psum/=N;                                  /* width=1/N */
  PT = MPI_Wtime()-PT;                      /* Medir en cuanto pi esté calculado */
} /* else rank */

MPI_Finalize();
```

Listado 1.4: Versión paralela MPI: código maestro.

parciales de los procesos esclavos conforme se van recibiendo (MPI_ANY_SOURCE). El correspondiente código esclavo se ubica en la otra rama condicional (Listado 1.5).

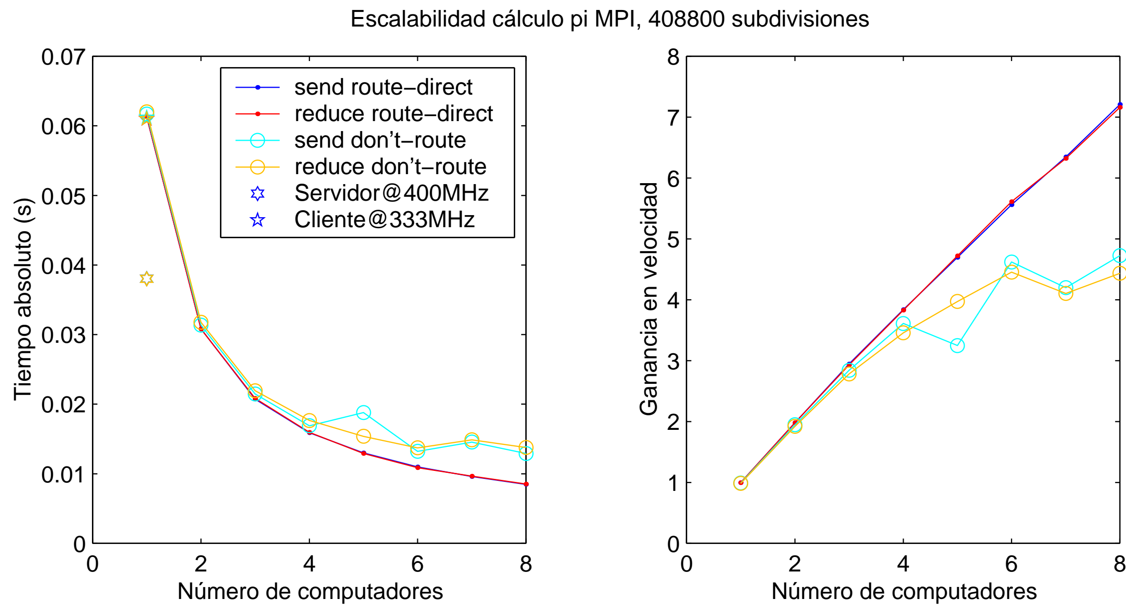
Los procesos esclavos no necesitan especificar buffer de destino para la operación de reducción. La variable **register** usada para acelerar los cálculos debe copiarse a memoria, igual que se hizo en PVM. En el caso de comunicación punto a punto, los distintos procesos MPI envían su resultado parcial a maestro conforme completan su bucle de cálculo.

La escalabilidad de la aplicación bajo MPI (Figura 1.7(a)) da una idea del coste relativo comunicación/cálculo. Para comparar entre sí las distintas alternativas de paso de mensajes resulta más conveniente medir el coste absoluto de la comunicación mediante un test *ping-pong* (Figura 1.7(b)).

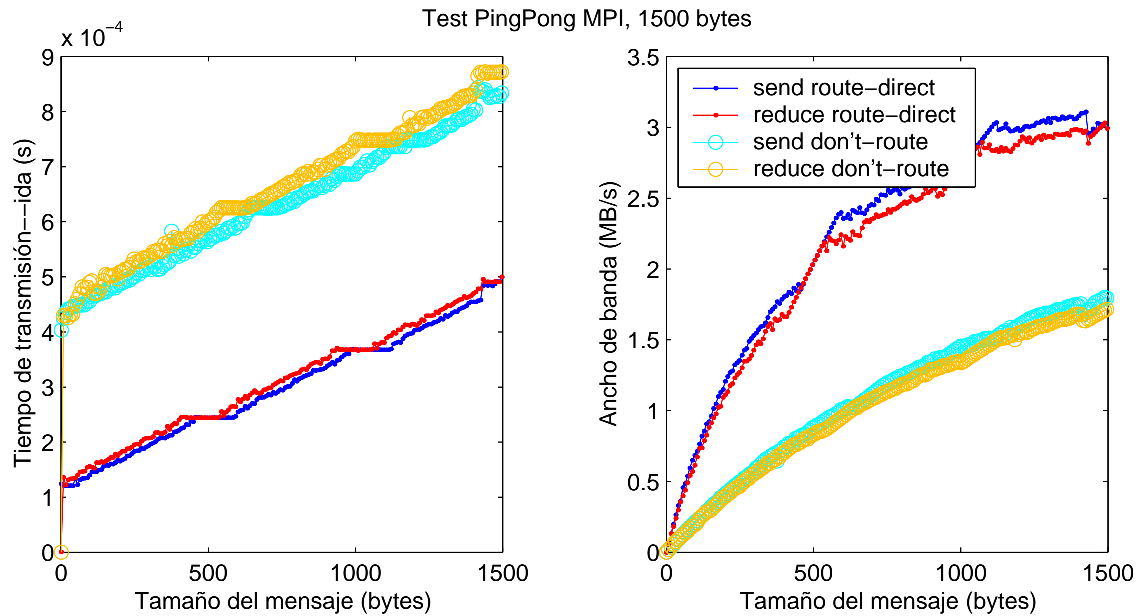
Se observan las superiores prestaciones del modo cliente-a-cliente LAM, con un *speedup* superior a 7, donde PVM obtenía algo más de 6.5. Ambas modalidades, colectiva y punto a punto, obtienen el mismo *speedup*. En PVM el modo [p]send requería una recepción de índice por

```
if (rank){                                  /* tramo esclavo */
    width = 1.0/N; sum = 0;
    for (i=rank-1; i<N; i+=C) {
      register double x = (i + 0.5) * width;
      sum += 4.0 / (1.0 + x * x);
    }
    Psum = sum;                             /* sum es register */
    switch (MOD){
    case 'r': MPI_Reduce(&Psum,NULL ,1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
              break;
    case 's': MPI_Send  (&Psum,      1, MPI_DOUBLE,        0, TAG, MPI_COMM_WORLD);
              break;
    }
} else {                                    /* tramo maestro */
  ...
} /* else rank */
```

Listado 1.5: Versión paralela MPI: código esclavo.



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.7: Tests realizados bajo MPI.

parte de los procesos esclavos que reducía sus prestaciones respecto a reduce. Bajo MPI esto no es necesario, pudiéndose controlar que el proceso maestro sea el rango 0, de manera que los esclavos puedan usar su propio rango MPI como índice.

Por contra, el modo *daemon* es muy inferior al de PVM (que obtenía *speedup* superior a 6, Figura 1.6(a) derecha), particularmente a partir de 4 computadores. La degradación del modo de reducción es peor, como pasó con PVM: el algoritmo en árbol coopera con el *daemon* LAM para reducir sus prestaciones. El comportamiento errático observado en la Figura es típico; eventualmente se puede obtener una gráfica de evolución suave, pero se ha preferido dejar constancia de que ocasionalmente se obtienen tiempos mejores que lo habitual.

El test *ping-pong* corrobora la similitud de los tiempos de transmisión para las dos modalidades, incluso con encaminamiento a través del *daemon*. No hay (de)codificación XDR si el usuario usa el modo homogéneo. Conviene notar el reducidísimo tiempo de *setup* con encaminamiento cliente a cliente, casi la mitad que con PVM. El encaminamiento a través del *daemon* es más parecido al de PVM, con un tiempo de *setup* ligeramente superior. Si el *daemon* no presentara un comportamiento tan errático cuando crece el número de procesos, se hubieran obtenido unos *speedups* que coincidirían comparativamente con los de PVM. En el test *ping-pong*, donde sólo hay implicados dos computadores, no se manifiesta dicho comportamiento errático.

1.6.2 Programas MATLAB

A continuación se muestran las soluciones del problema propuesto en lenguaje MATLAB bajo algunas de las *Toolboxes* paralelas anteriormente citadas: nuestras PVMTB y MPITB, las dos siguientes en número de comandos, DP-TB (basada en PVM) y MultiMATLAB (basada en MPI), y otras dos de entre las más sencillas, TCPIP (basada en *sockets* TCP/IP) y Paralize (basada en el sistema de ficheros compartido).

Programa PVMTB

Siguiendo el esquema de la aplicación PVM en lenguaje C, se automatizó el estudio de escalabilidad mediante un *script* MATLAB que ejecuta el programa secuencial en el servidor y un cliente, y el programa paralelo en un número creciente de clientes. Se contemplan los mismos modos de comunicación (send, reduce, psend) y encaminamiento (directo o no). El programa maestro (Listado A.8) realiza tareas muy similares a la versión C. El Listado 1.6 muestra un primer fragmento del mismo.

El comando de extensión PVMTB pvme_start_pvmd permite al usuario MATLAB especificar la configuración PVM en la propia llamada, generándose automáticamente un fichero de configuración temporal. El *cell-array* HN mostrado en el código fuente proporciona a los *daemons* remotos el mismo camino de búsqueda ($PATH) y directorio de trabajo (**pwd**) que usa el proceso MATLAB local. Del conjunto de clientes listados se usarán sólo los C primeros.

En la aplicación C vista anteriormente la tarea de reconfigurar PVM recaía sobre el *script* de automatización, ya que era más cómodo invocar la consola PVM desde el *script* que utilizar la llamada pvm_start_pvmd() en el programa C. Bajo MATLAB se invierten los papeles y es más cómodo usar la extensión pvme_start_pvmd que invocar a la consola PVM.

A continuación se arranca el servidor de grupos pvmgs, se crea el grupo **pi** usando pvm_joingroup

```
HN={['*␣ep=$PATH␣wd=' pwd], 'ox1', 'ox2', 'ox3', 'ox4', 'ox5', 'ox6', 'ox7', 'ox8'};
pvme_start_pvmd(HN{1:C+1});                              % * ep, wd + C clientes
pvm_spawn('pvmgs',{},1,'.',1);                           % groupserver siempre en servidor
inum=pvm_joingroup('pi');                                % y antes que Spawn esclavos

CC=int2str(C); NN=int2str(N);
cmd=['Work(' CC ',' NN ')'];
putenv(['cmd=' cmd]); pvm_export('cmd');                 % Pasar comando por entorno
if DEBUG
  DISP=getenv('DISPLAY');
  [numt tids]=pvm_spawn('xterm',{'-display' DISP '-e' 'matlab'},33,'.',C);
else
  [numt tids]=pvm_spawn('matlab',{'-nosplash'},            33,'.',C);
end
```

Listado 1.6: Configuración del *daemon* PVM y arranque de esclavos desde MATLAB.

y se arrancan los procesos MATLAB esclavos, bien directamente en el *background* o como comando a ejecutar dentro de un terminal xterm. Esta última opción es utilísima para depurar el código usando el *debugger* integrado en el propio MATLAB. El método usado para pasar el comando deseado a los procesos MATLAB esclavos requiere una explicación adicional.

Al ser MATLAB una aplicación interactiva, no habría en principio otro modo de hacerse con el control de un proceso MATLAB más que mediante el teclado. Afortunadamente, MATLAB ejecuta varios *scripts* de inicio bajo Unix, uno global a la instalación (**matlabrc**.m) y otro dependiente del usuario (**startup**.m) antes de mostrar el *prompt* y esperar comandos del usuario. Un uso típico de dicho *script* es añadir subdirectorios al *path* (camino de búsqueda de *scripts*) MATLAB. Está tan extendida esta costumbre que los creadores de *Toolboxes* proporcionan las líneas que se deberían añadir al *script* de inicio (global o personal) para poder usar sus *Toolboxes*. El usuario que se instale un par de *Toolboxes* personales no necesita alterar la configuración global (afectando al resto de usuarios) ni cambiarse al subdirectorio de instalación de cada una; le basta con añadir las líneas sugeridas a su **startup**.m para poder utilizar ambas sin cambiar de subdirectorio.

El **startup** sugerido para PVMTB, tras añadir subdirectorios al *path* MATLAB, incluye las líneas siguientes:

```
if ~isempty(getenv('PVMEPID')),  startup_Slv;
else,                             disp('Help␣on␣PVM:␣help␣pvm')
end
```

Un proceso MATLAB arrancado mediante pvm_spawn() tiene una variable de entorno PVMEPID (ver la página de manual pvm_intro) de la cual carece un proceso MATLAB normal arrancado por el usuario desde el *shell*. Por lo tanto se ha usado la presencia de dicha variable para discriminar los procesos arrancados por el daemon PVM, a los cuales se les hace ejecutar otro *script* (startup_Slv.m, en este caso). El usuario puede editar este *script* o dejarlo en blanco si lo desea. El *script* esclavo sugerido para esta aplicación contiene:

```
hostname                                          % Mostrar nombre host en xterm
cmd=getenv('cmd'); eval(cmd); quit                % evaluar y salir
```

Naturalmente, el comando hostname permite ver el nombre del host remoto sólo si se arranca MATLAB mediante un xterm. Los MATLAB esclavos esperan disponer de la variable de entorno cmd exportada por el maestro mediante el comando pvm_export previo al pvm_spawn (Listado 1.6). El

programa maestro (Listado 1.7) empieza entonces a medir el tiempo del bucle:

```
pvm_freezegroup('pi',C+1);                          % ver Caveat man pvm_reduce
    tic                                         % Medir tras sincr. inicial como MPI_Init
    Psum=0; Sum=0;
    pvm_reduce('Sum',Psum,TAG,'pi',0);
    Psum      =Psum/N;                          % width = 1/N
    Data.pi   =Psum;
    Data.err  =Psum-pi;
    Data.time =toc;                             % tic-toc
pvm_barrier('pi',C+1);                          % ver Caveat en man pvm_reduce
pvm_lvgroup('pi');                              % salir
pvm_exit;
pvm_halt;
```

Listado 1.7: Versión paralela PVMTB: código maestro.

La operación es idéntica a la de PVM en lenguaje C. Las barreras garantizan que todos los procesos se hayan unido al grupo y ninguno haya salido antes de realizar la reducción.

El comando invocado en los procesos esclavos, Work(C,N), consiste sencillamente en:

```
function Work(C,N)

inum=pvm_joingroup('pi');                       % argumento implícito
pvm_freezegroup('pi',C+1);                      % sincronización inicial

  width=1/N; lsum=0;                            % código vectorizado, equivale a
  i=inum-1:C:N-1;                               % for i=inum-1:C:N-1
  x=(i+0.5)*width;                              %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                        %   lsum=lsum+4/(1+x^2);
  pvm_reduce('Sum',lsum,TAG,'pi',0);            % end

pvm_barrier('pi',C+1);                          % ver Caveat en man pvm_reduce
pvm_lvgroup('pi');                              % salir
pvm_exit;
```

Listado 1.8: Versión paralela PVMTB: código esclavo.

La versión completa se recoge en el Listado A.9. El número de instancia en el grupo PVM sirve como argumento i de la aplicación. Los restantes argumentos C y N son comunes, lo cual nos ha permitido exportarlos textualmente mediante la variable de entorno cmd. Otras aplicaciones más complejas pueden requerir un método de paso de comandos distinto. Soportando el caso más general, PVMTB proporciona un startup_Slv en el cual se espera que el proceso maestro envíe explícitamente una variable NUMCMDS y a continuación envíe explícitamente NUMCMDS veces una variable cmd conteniendo la expresión a evaluar, y posiblemente variables adicionales que puedan ser referenciadas como argumentos por dicho comando cmd.

Para medir el tiempo en igualdad de condiciones con MPI es necesario realizar una sincronización inicial, equivalente a MPI_Init(). En nuestro caso, congelamos el grupo con pvm_freezegroup, que además sirve de barrera. También ejercitamos la ruta directa si se escoge dicho encaminamiento, aunque no se muestre en el código. Tras satisfacer la barrera con los procesos esclavos, el proceso maestro se bloquea en la reducción. El proceso esclavo calcula entonces las subdivisiones que le corresponden, y reduce sobre el proceso maestro (realizando una barrera posterior como aconseja el manual PVM), tras lo cual sólo queda abandonar el grupo, el sistema PVM y el propio comando Work. El proceso esclavo vuelve entonces al *script* startup_Slv, que como vimos ejecuta **quit** tras evaluar cmd.

Por su parte, el código maestro también abandona el grupo **pi** y para el sistema PVM tras superar la barrera posterior a la reducción (Listado 1.7).

Todas las llamadas PVM están disponibles bajo PVMTB, de manera que es posible redactar programas MATLAB igual de sencillos o complejos que usando PVM bajo C. El patrón de llamada bajo PVMTB es algo más sencillo ya que MATLAB permite a una función devolver varios argumentos de retorno y no requiere declarar previamente ninguna variable. Compárense las respectivas llamadas pvm_spawn, por ejemplo.

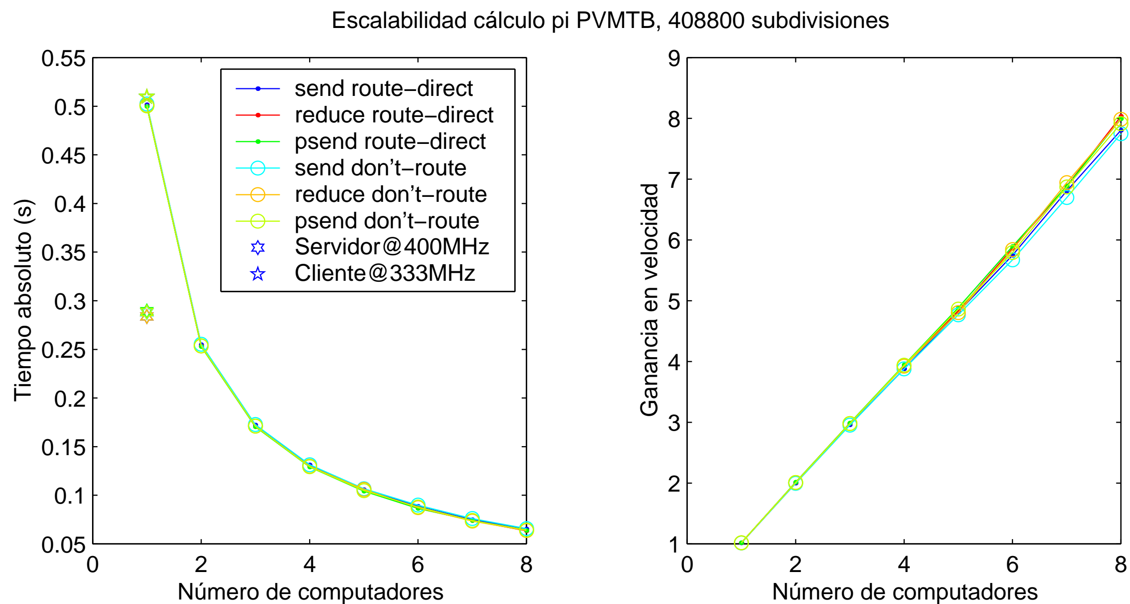
La escalabilidad de la aplicación bajo PVMTB (Figura 1.8(a)) da una idea del coste relativo comunicación/cálculo. Comparando con la Figura 1.6(a) izquierda, notamos que bajo MATLAB el tiempo de cálculo es un orden de magnitud superior, a pesar de haber vectorizado cuidadosamente el bucle de cálculo de la aplicación (Listado 1.8). Obtenemos ahora unos tiempos secuenciales de 0.3/0.5s (en servidor y cliente) frente a 40/60ms en lenguaje C (Figura 1.6(a)).

La recepción de mensajes, naturalmente, no es susceptible de vectorización. Comparando sin embargo los resultados del test *ping-pong* (Figura 1.8(b) izquierda), se observa que el *overhead* introducido por nuestra PVMTB es menor que el de la copia de memoria adicional o que el tiempo de *setup*. Los tiempos de *setup* pasan de 0.2/0.4ms bajo PVM (Figura 1.6(b)) a 0.4/0.6ms en PVMTB, y los tiempos de transmisión para 1500 bytes pasan de 0.55/0.9ms a 0.7/1.0ms, reflejando que nuestra *Toolbox* no realiza ninguna copia de memoria adicional.

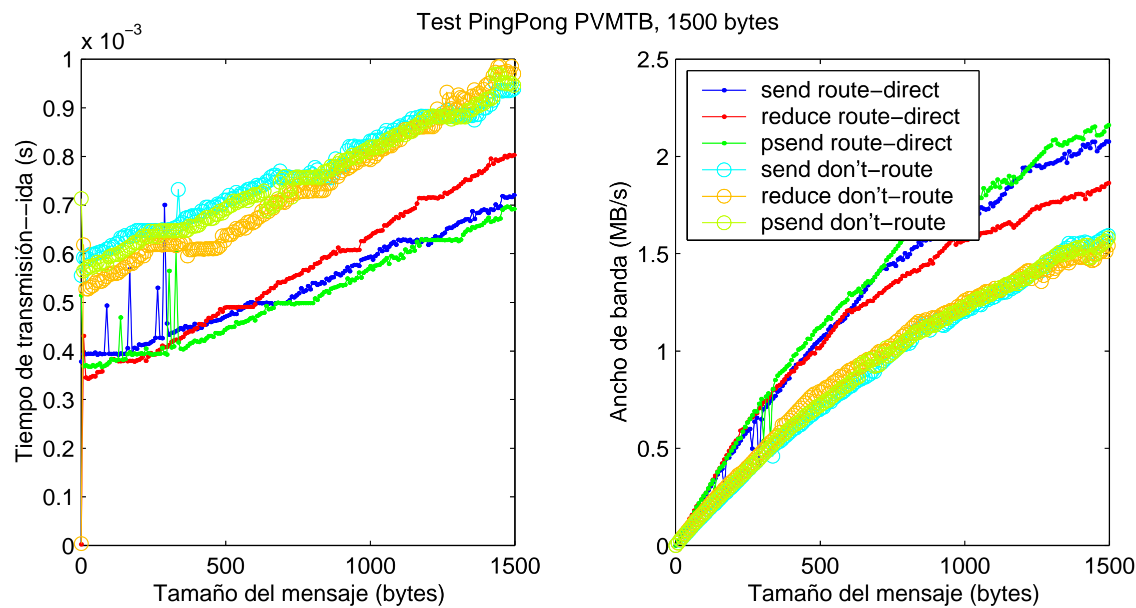
Al ser MATLAB un lenguaje interpretado, el número de comandos dentro del bucle *ping-pong* influye considerablemente en el tiempo de ejecución, pudiéndose comprobar que la modalidad send tarda más que psend mientras que eran prácticamente indistinguibles bajo PVM. Recordemos que usar send implica 5 comandos en el bucle (initsend, pack, send, recv, unpack), mientras que psend y reduce sólo 2. Los costes de la copia de memoria del *daemon* PVM y de la codificación XDR con reduce se mantienen bajo PVMTB.

No está claro que los paradójicamente excelentes resultados de reduce para tamaños de mensaje pequeños se deban al solapamiento de procesamiento (bucle *ping-pong* bajo MATLAB) y comunicación (servidor de grupos pvmgs), superando a psend a pesar de tener el mismo número de instrucciones en el bucle. Este argumento también explicaría que la ventaja (relativa) sea mayor con encaminamiento a través del *daemon* PVM. Comprobar objetivamente la hipótesis requiere usar el mecanismo de trazado PVM o el propio visualizador XPVM, lo cual altera la propia medición arruinando el fenómeno que se deseaba observar. En cualquier caso, el costo de la codificación XDR se impone rápidamente, eliminando la ventaja de reduce sobre psend con ruta directa a partir de unos 250 bytes, y sobre send a partir de unos 500 bytes.

En el estudio de escalabilidad se refleja el incremento en un orden de magnitud en el tiempo de cálculo de MATLAB en comparación con el moderado *overhead* de nuestra *Toolbox* (similar al tiempo de *setup*), resultando en una escalabilidad lineal. La modalidad de reducción obtiene el mejor *speedup*, existiendo una desventaja pequeña para el modo psend, y ligeramente mayor para send. Las excelentes prestaciones de reduce anteriormente comentadas no guardan relación con este resultado. De hecho, apenas hay diferencia entre usar el encaminamiento directo o a través del *daemon*. Los gastos del paso de mensajes, incluyendo el pequeño *overhead* añadido por PVMTB, se vuelven inapreciables frente al orden de magnitud de *overhead* de cálculo en MATLAB. Es más ventajoso ahorrarse el bucle de recepción de resultados (Listado A.8). Este bucle es inexistente para reduce, consta de 2 instrucciones para psend (un precv y la propia acumulación), y de 3 para send (recv, unpack y la acumulación).



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.8: Tests realizados bajo PVMTB.

Programa MPITB

La aplicación MPITB es prácticamente una traducción a MPI de la aplicación PVMTB recien vista. Está redactada por tanto en estilo Master-Slave, en lugar del estilo SPMD seguido por la versión MPI bajo C. Esto se debe a que no se desea implicar al usuario MATLAB en el uso de la herramienta mpirun. Arrancar una aplicación MATLAB SPMD mediante mpirun supondría que la aplicación es invocable desde el *shell*, es decir, se trata de un programa MEX de estilo *engine*. Esto arruinaría el objetivo de poder desarrollar aplicaciones paralelas en lenguaje MATLAB mediante ficheros .m.

Al carecer el estándar MPI de una llamada que arranque los *daemon* (el estándar ni siquiera establece que las implementaciones deban funcionar mediante la ejecución de *daemons*), es obligatorio que el usuario MATLAB arranque con lamboot el sistema LAM.

El *script* para el estudio de la escalabilidad es idéntico al de PVMTB, cambiando únicamente el contenido de los *scripts* Mast.m y Work.m, que han sido traducidos de PVM a MPI. El código completo de ambos programas se ofrece en los Listados A.10 y A.11, respectivamente. Un primer fragmento del maestro muestra el procedimiento para arrancar procesos MPI dinámicamente:

```
[info flag]=MPI_Initialized;
if info | ~flag, error('MPI no iniciado, ejecute MPI_Init'), end
if DEBUG
  DISP=getenv('DISPLAY');
  [info children errs]=MPI_Comm_spawn('xterm',{'-display' DISP '-e' 'matlab'},
                                                              C,'NULL',0,'SELF');
else
  [info children errs]=MPI_Comm_spawn('matlab',{'-nosplash'},    C,'NULL',0,'SELF');
end
[info NEWORLD]=MPI_Intercomm_merge(children,0);
```

Listado 1.9: Comprobación de la configuración MPI y arranque de esclavos desde MATLAB.

El estándar MPI [66, pág.199, líneas 46–48] establece que tras llamar a MPI_Finalize() una aplicación MPI no debe realizar posteriores llamadas MPI, ni siquiera puede volver a invocar MPI_Init(). Esto nos impide usar MPI_Finalize en nuestras aplicaciones interactivas, ya que nos obligaría a salir y volver a entrar en MATLAB para poder ejecutar otra aplicación MPITB. También se establece [66, pág.199, líneas 12–16] que se debe llamar a MPI_Init() antes de poder invocar cualquier otra rutina MPI, y que posteriores llamadas a MPI_Init() son erróneas. Todo ello obliga al usuario a usar interactivamente el comando MPITB MPI_Init, y obliga a las aplicaciones MPITB a comprobar tal condición usando MPI_Initialized. En [66, pág.200, líneas 10–12] se rectifica indicando que esta llamada es la única que se puede usar sin haber llamado previamente a MPI_Init().

A diferencia de los grupos dinámicos en PVM, los comunicadores MPI se crean en tiempo de arranque y son estáticos. El *inter*comunicador creado para los procesos esclavos y devuelto en la llamada MPI_Comm_spawn (children en el Listado 1.9) no incluye al proceso maestro. Es necesario construir un *intra*comunicador que agrupe al proceso maestro inicial (comunicador MPI_COMM_SELF) y a los esclavos arrancados posteriormente (children) usando MPI_Intercomm_merge. Naturalmente, esta llamada también es colectiva. El segundo parámetro indica si los procesos del comunicador local (en este caso sólo el maestro) aparecen en el intracomunicador resultante antes (0) o después (1) que los del comunicador remoto. Por uniformidad con las aplicaciones anteriores, preferimos asignar al proceso maestro el rango 0 en el nuevo comunicador. Con la otra elección, ubicando el proceso maestro como último rango del nuevo intracomunicador, el

código esclavo se ahorraría una resta en la inicialización del bucle, y el programador ganaría la preocupación adicional de tener que recordar dónde era la resta, si en PVMTB o en MPITB.

El fichero **startup**.m sugerido para esta *Toolbox*, tras ajustar el *path*, acaba con:

```
if ~isempty(getenv('LAMPARENT')),                    startup_Slv;
else,
  disp('Ajustar␣opciones␣-O␣-c2c␣usando␣el␣comando:')
  disp('␣␣putenv([''TROLLIUSRTF=''␣int2str(RTF_HOMOG+RTF_MPIC2C)]),␣ MPI_Init')
  disp('Help␣on␣MPI:␣ help␣mpi')
end
```

en donde la variable de entorno LAMPARENT juega un papel homólogo al de PVMEPID bajo PVM. Se recomienda al usuario del proceso MATLAB arrancado desde el *shell* que ajuste los modos homogéneo y cliente-a-cliente antes de invocar el obligatorio MPI_Init.

Los procesos MATLAB arrancados mediante el *daemon* no ejecutan esa rama del condicional sino el *script* startup_Slv, que para esta aplicación puede reducirse a:

```
info=MPI_Init;                                   % Para el acostumbrado merge
[info parent] = MPI_Comm_get_parent;
global NEWORLD                                   % luego lo heredan los esclavos
[info NEWORLD]=MPI_Intercomm_merge(parent,1);    % ponerse detrás en comunicador
hostname                                         % Mostrar nombre del host

cmd='␣'; cmd=repmat(cmd,1,100);                  % Sitio para recibir string cmd
MPI_Bcast(cmd,0,NEWORLD); eval(cmd);             % Recibir y evaluar comando
MPI_Finalize; quit                               % Salir
```

en donde sí se invoca MPI_Init (y consecuentemente se acaba con MPI_Finalize) para poder utilizar inmediatamente la rutina MPI_Intercomm_merge y desbloquear al proceso padre. Esta última llamada es colectiva, bloqueando todos los procesos implicados hasta obtener el intercomunicador resultado. Tras crear el nuevo intercomunicador, en el cual el proceso padre es el rango 0 (comparar el 2º argumento de la rutina merge en ambos listados), se recibe de él el comando a ejecutar, se evalúa y se finaliza.

Se ha optado por enviar explícitamente el comando porque el estándar MPI no dispone de llamada para exportar variables de entorno programadamente (sí lo contempla el comando LAM mpirun, que no deseamos utilizar). Para medir el tiempo en condiciones equivalentes a PVMTB, se realiza una sincronización justo después de transmitir explícitamente el comando mediante MPI_Bcast. El proceso padre puede arrancar el cronómetro tras superar la barrera:

Como se puede observar, hemos usado una barrera lineal alternativa. MPI_Barrier(), al igual que vimos con MPI_Reduce(), usa también un algoritmo en árbol a partir de LAM_COLLMAXLIN==4 procesos, como se puede comprobar en el fichero <LAMROOT>/share/mpi/barrier.c. Esto nos hacía obtener unos *speedups* inferiores en medio punto. Para la aplicación MPI no se nos presentó el problema porque el propio MPI_Init() servía de sincronización, pasando el maestro a cronometrar inmediatamente. Nuestra barrera alternativa usa Recv ya que acabamos de superar un Bcast en el sentido opuesto.

La operación MPI_Bcast es también colectiva, funcionando el *string* cmd como fuente en el proceso maestro (el rango 0 indicado en la llamada) y como destino en el resto de procesos en el intercomunicador.

El proceso maestro queda entonces bloqueado en la colectiva MPI_Reduce (Listado 1.10) mien-

```
 CC=int2str (C); NN=int2str (N); cmd=['Work(' CC ',' NN ',''' MOD ''')'];
 MPI_Bcast (cmd,0,NEWORLD);                                   % Pasar comando
%MPI_Barrier(NEWORLD);                                        % Synch inicial
 NULL=[];
 for numt=1:C,    MPI_Recv(NULL,MPI_ANY_SOURCE,TAG,NEWORLD); end

    tic
    P2=0; Psum=0;                                             % fuente/destino reduce separados
    switch (MOD)
    case 'r',     MPI_Reduce(P2,Psum,'SUM',0,NEWORLD);
    case 's'
    for numt=1:C, MPI_Recv(P2,MPI_ANY_SOURCE,TAG,NEWORLD); Psum=Psum+P2; end
    end

    Psum      =Psum/N;                                        % width = 1/N
    Data.pi   =Psum;
    Data.err  =Psum-pi;
    Data.time =toc;                                           % tic-toc
```

Listado 1.10: Versión paralela MPITB: código maestro.

tras los esclavos calculan. Tras verificarse la reducción, los procesos esclavos acaban y el maestro mide el tiempo empleado. Usando la alternativa punto a punto, se acumulan los resultados parciales de los esclavos conforme se van recibiendo (MPI_ANY_SOURCE). Con un bucle MATLAB de dos instrucciones, se pueden prever prestaciones similares a las de psend bajo PVM.

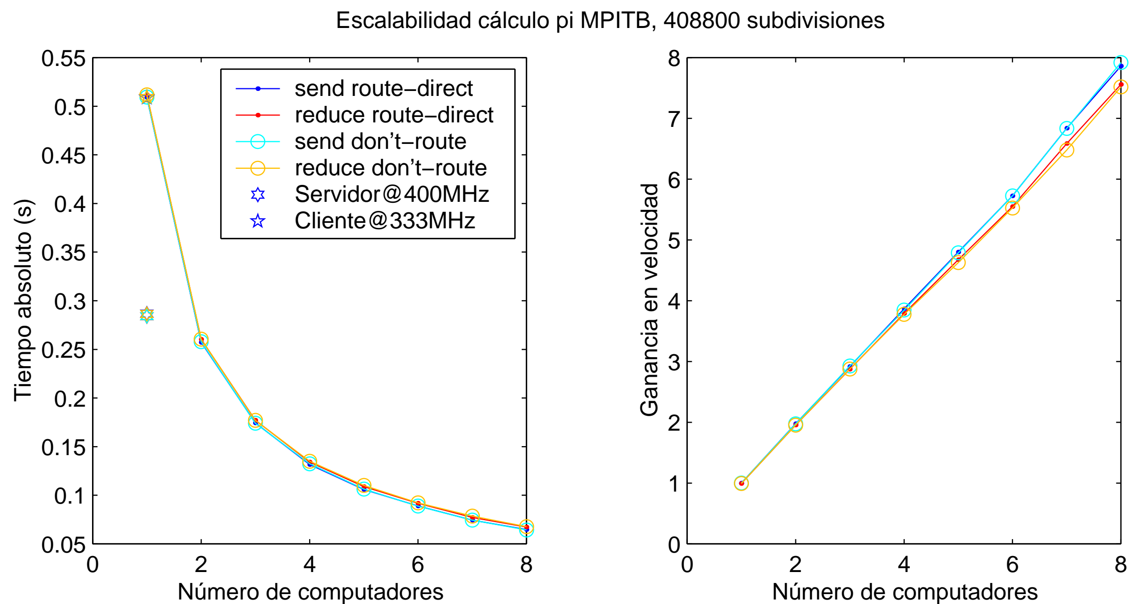
El código esclavo MPITB (también mostrado en el Listado A.10) es:

```
function Work(C,N,MOD)

TAG=7; global NEWORLD                                  % heredado de startup_Slv, incluye padre
%MPI_Barrier(NEWORLD);                                 % Synch inicial
 NULL=[];
 MPI_Send(NULL,0,TAG,NEWORLD);
[info rank]=MPI_Comm_rank(NEWORLD);                    % rango en intracom = 1..C

  width=1/N; lsum=0;                                   % código vectoriza, equivale a
  i=rank-1:C:N-1;                                      % for i=rank-1:C:N-1
  x=(i+0.5)*width;                                     %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                               %   lsum=lsum+4/(1+x^2);
                                                       % end
switch MOD
  case 'r',            dum=0;
         MPI_Reduce(lsum,dum,'SUM',0,NEWORLD);         % master = (0 en NEWORLD)
  case 's'
         MPI_Send(lsum,0,TAG,NEWORLD);
end
```

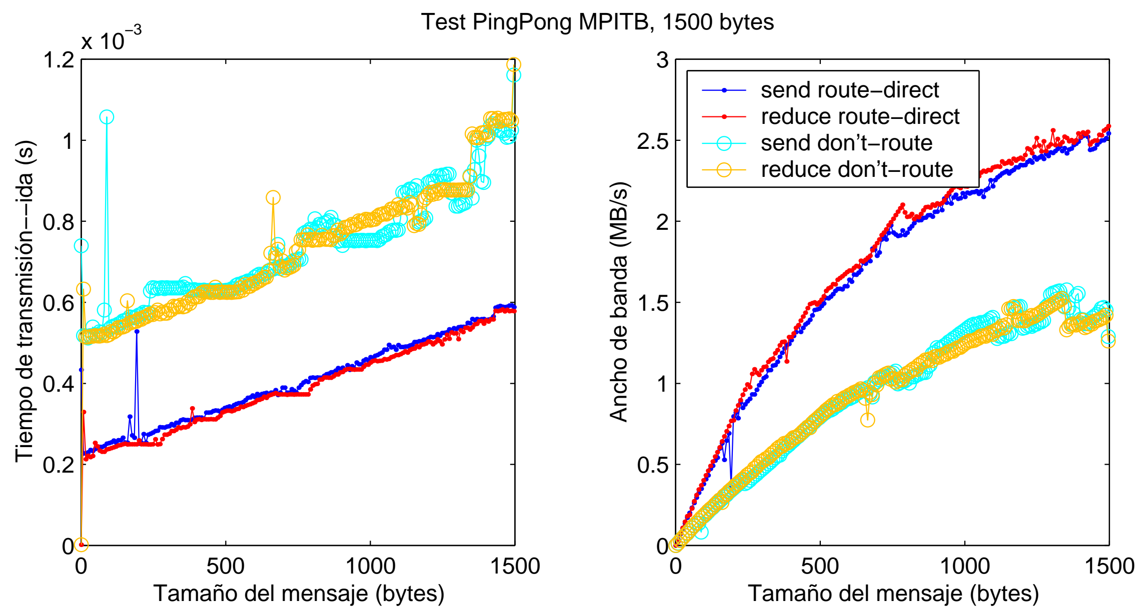
Listado 1.11: Versión paralela MPITB: código esclavo.

Nada más empezar se satisface la sincronización con el maestro, que empieza a cronometrar el tiempo empleado. Como vemos, el índice del proceso es un argumento implícito que se puede obtener del intracomunicador NEWORLD creado en startup_Slv. Esto es ventajoso para la modalidad punto a punto MPI en comparación con PVM, en la cual se debía transmitir dicho índice explícitamente. El resto de argumentos son comunes, lo cual nos ha permitido transmitir el comando en una única llamada colectiva MPI_Bcast. Cada rango MPI calcula su resultado parcial y lo envía al maestro (modalidad punto a punto) o coopera en la reducción (modalidad colectiva).

Tras finalizar la evaluación de este comando, los procesos MATLAB esclavos retornan al *script* startup_Slv en donde ejecutan MPI_Finalize y terminan (**quit**). Las gráficas del estudio de escalabilidad y del test *ping-pong* se muestran en las Figuras 1.9(a) y 1.9(b), respectivamente.



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.9: Tests realizados bajo MPITB.

La modalidad punto a punto presenta un *speedup* ligeramente inferior a 8, como sucedía con psend bajo PVM; las dos instrucciones en el bucle de acumulación impiden la linealidad perfecta. Por contra, la operación de reducción no tiene bucle de acumulación, pero usa el algoritmo de reducción en árbol, lo cual nos hace perder casi medio punto de ganancia para 8 computadores. La pérdida empieza a notarse a partir de LAM_COLLMAXLIN==4 procesos. No está claro por qué no se manifestaba este hecho en el programa MPI bajo C (Figura 1.7(a)), en donde lo que quedaba discriminado era el encaminamiento a través del *daemon*, mientras que aquí queda discriminada la operación de reducción colectiva.

En el test *ping-pong* se observa que MPITB introduce un *overhead* cercano a 0.1ms, subiendo el tiempo de *setup* de los 0.1/0.4ms de MPI en la Figura 1.7(b) a los 0.2/0.5ms de MPITB en la Figura 1.9(b). Al igual que bajo PVMTB, no está claro cómo se obtienen unos tiempos de *ping-pong* menores para la reducción que para las primitivas punto a punto. Incluso hemos probado a alterar el orden normal del test, realizando la prueba de reducción antes que la de punto a punto (por si el sistema LAM afinara algún parámetro de espera en función de estadísticas adaptativas) obteniendo siempre los mismos resultados reproducibles.

Programa DP-TB

La *Distributed-Parallel Toolbox* [19, 76] realizada por el equipo de Sven Pawletta de la Universität Rostock (Alemania) es el trabajo más parecido a PVMTB que se puede encontrar en la literatura. En comparación sólo ofrece 56 comandos siendo 47 de ellos rutinas PVM, frente a los 93 comandos de PVMTB que incluyen 86 rutinas PVM. La Tabla 1.3 relaciona los comandos disponibles bajo DP-TB.

En particular, DP-TB no contempla las llamadas relacionadas con grupos, lo cual incluye a todas las llamadas colectivas PVM. También es interesante notar que aunque existe interfaz para la llamada pvm_notify(), no lo hay para pvm_upkint(). No queda claro cómo podría el usuario DP-TB desempaquetar los mensajes de notificación enviados por el *daemon* PVM a la tarea que solicitó notificaciones. Los mensajes de notificación consisten en *tids* (números de tarea PVM) y contadores, ilegibles si sólo se dispone del comando DP-TB pvm_upkdouble.

Respecto a éste último comando, y en general todos los de desempaquetamiento, nótese que se devuelve el dato desempaquetado como argumento de retorno (a la izquierda, *lefthand side*). Esto tiene implicaciones en cuanto al número de copias adicionales en el paso de mensajes. Los argumentos de retorno deben ser creados en el fichero MEX (ó M) que los devuelve, no pudiéndose reutilizar una variable previamente existente. Por ejemplo, en la sentencia d=pvm_upkdouble, es responsabilidad del fichero MEX pvm_upkdouble reservar espacio para la variable de retorno. MATLAB destruye la variable d (si existía previamente) antes de almacenar la variable devuelta en el espacio de trabajo con el nombre d. En comparación con DP-TB, PVMTB no incurre en ninguna copia adicional, lo cual se refleja en las prestaciones del paso de mensajes.

Otra diferencia técnica es el método de enlazado. DP-TB, al igual que otras *Toolboxes* paralelas previas, está programada usando un sistema de directorio. Los comandos de paso de mensajes son ficheros M que invocan a un fichero MEX "directorio" intermedio enlazado estáticamente con la biblioteca PVM. Así por ejemplo, el comando pvm_upkdouble consiste en la llamada MEX [info,data] = m2pvm(703,nitem,stride). El fichero MEX invocado, m2pvm, es el directorio. El primer argumento de la llamada determina la función a invocar; en este caso se ha asignado a la rutina

pvm_upkdouble() el código 703. El código fuente m2pvm.c contiene una sentencia switch, uno de cuyos casos es case 703: m2pvm_upkdouble(nlhs,plhs,nrhs,prhs); **break**;.

Aparte del consumo de disco y memoria adicionales, la interpretación del fichero M y la selección del caso presentan un coste de tiempo adicional en el que no incurre PVMTB. También se requiere un cuidadoso sistema de bloqueo en memoria para garantizar que el directorio está permanentemente cargado en memoria. En las pruebas realizadas se ha recompilado DP-TB enlazándola con nuestra biblioteca dinámica PVM, al objeto de considerar únicamente las diferencias de prestaciones debidas a las propias *Toolboxes*, no a los tiempos de carga de las respectivas bibliotecas. No se han utilizado los comandos pvme_[un]link.

Tabla 1.3: Comandos DP-TB y descripción de su finalidad o lista de argumentos.
∗ diferencias notables con la rutina PVM correspondiente o sus argumentos. † comandos de extensión.

	Comando	Descripción / argumentos
	Enlace con libpvm	
†	pvme_link	bloquea libpvm en memoria
†	pvme_unlink	desbloquea libpvm de mem.
	Control de PVM	
†	pvme_is	test tarea PVM
†	pvme_default_config	crea hostfile PVM
†	pvme_start_pvmd	(conf, block)
	pvm_addhosts	(hosts)
	pvm_delhosts	(hosts)
∗	pvm_config	[nhost,narch,dtids, hosts,archs,speeds,info]=...
	pvm_mstat	(host)
∗†	pvme_halt	protege sesión que llama
	Opciones PVM	
	pvm_getopt	val=...(what)
	pvm_setopt	old=...(what,val)
	Control de procesos	
	pvm_spawn	[tids,numt]=..(tsk, args,flg,whr,ntsk)
	pvm_export	(name)
	pvm_unexport	(name)
	pvm_kill	(tid)
	pvm_exit	
	(Des)empaquetamiento de datos	
	pvm_pkdouble	(data,nitem,stride)
	pvm_upkdouble	[data,info]=... (nitem,stride)
†	pvme_pkarray	array y un nombre
†	pvme_upkarray	[arr,name,info]=...
†	pvme_upkarray_name	[name,info]=...
†	pvme_upkarray_rest	[arr,info]=...
	Mailbox PVM	
	pvm_putinfo	idx=..(nam,buf,flg)
	pvm_recvinfo	buf=..(nam,idx,flg)
	pvm_delinfo	inf=..(nam,idx,flg)
∗	pvm_getmboxinfo	[nclass,nams,nentr, idxs,owns,flgs,info]=...(patt)

	Comando	Descripción / argumentos
	Señales	
	pvm_sendsig	(tid,sig)
	pvm_notify	(wht,tag,cnt,tids)
	Información	
	pvm_mytid	tid=...
	pvm_parent	tid=...
	pvm_pstat	stat=...(tid)
∗	pvm_tasks	[ntask,tids,ptids, dtids,sts,tsks,inf]=..(which)
	pvm_tidtohost	dtid=...(tid)
	pvm_perror	
	pvm_archcode	cod=...(name)
	pvm_getfds	[fds,nfds]=...
	pvm_version	ver=...
	Buffers de mensaje	
	pvm_initsend	buf=...(enc)
	pvm_mkbuf	buf=...(enc)
	pvm_getrbuf	buf=...
	pvm_getsbuf	buf=...
	pvm_setrbuf	old=...(buf)
	pvm_setsbuf	old=...(buf)
	pvm_bufinfo	[bytes,tag,tid, info]=...(buf)
	pvm_freebuf	(buf)
	Envío y Recepción	
	pvm_send	(tid,tag)
	pvm_mcast	(tids,tag)
	pvm_probe	buf=..(tid,tag)
	pvm_recv	buf=..(tid,tag)
∗	pvm_trecv	buf=..(td,tg,s,us)
	pvm_nrecv	buf=..(tid,tag)
	Extensiones MATLAB	
†	persistent2	crea/borra vars. pers.
†	putenv	llamada putenv(3)
†	unsetenv	llamada unsetenv()
∗†	selectstdin	llamada select(2)

La ausencia de estructuras en versiones anteriores de MATLAB forzó a los autores de la DP-TB a utilizar multitud de argumentos de retorno en algunos comandos, como en pvm_config, pvm_getmboxinfo, pvm_tasks o pvm_bufinfo. La ausencia de *cell-arrays* les impide expresar cómodamente el tipo C char** (array de *strings*), como se comenta más abajo en el ejemplo de aplicación respecto al comando pvm_spawn.

El software, tal y como se ofrece en la página Web [19] presenta un problema en el *script* **startup**.m, provocando un error relacionado con variables persistentes al salir de MATLAB. Se optó por sustituir el **startup**.m ofrecido por uno similar al utilizado con nuestra PVMTB.

El *script* para el estudio de la escalabilidad es idéntico al de PVMTB, cambiando únicamente el contenido de los *scripts* Mast.m y Work.m, eliminando las modalidades no disponibles bajo DP-TB. El primer fragmento del maestro (ver también Listado A.12) sólo presenta sutiles diferencias:

```
HN={['*␣ep=$PATH␣wd=' pwd],'ox1','ox2','ox3','ox4','ox5','ox6','ox7','ox8'};
pvme_default_config(str2mat(HN{1:C+1}));
pvme_start_pvmd('d',1);                                 % * ep, wd + C clientes

CC=int2str(C); NN=int2str(N); cmd=['Work(' CC ',' NN ')'];
putenv(['cmd=' cmd]); pvm_export('cmd');                % Pasar comando por entorno
if DEBUG
  DISP=getenv('DISPLAY');
  [tids numt]=pvm_spawn('xterm',str2mat({'-display' DISP '-e' 'matlab'}),33,'.',C);
else
  [tids numt]=pvm_spawn('matlab',         '-nosplash',                   33,'.',C);
end

for numt=1:C                                            % Anotar tids conforme respondan
    pvm_recv(-1,TAG); tids(numt)=pvm_upkdouble(1,1);
end
```

Listado 1.12: Configuración del *daemon* PVM y arranque de esclavos con DP-TB.

La secuencia de arrancar el *daemon* requiere dos instrucciones separadas y el uso de strmat. DP-TB no aprovecha los nuevos tipos de datos MATLAB, mientras que PVMTB aprovechó aquí ventajosamente los *cell-arrays*. También se usaron para expresar más cómodamente los argumentos del comando pvm_spawn (Listado 1.6). Para pasar el comando se utiliza el mismo método basado en variable de entorno, reutilizándose los mismos *scripts* **startup** mostrados para PVMTB.

Al no disponer de grupos no se puede usar el número de instancia devuelto por pvm_joingroup como argumento implícito en los esclavos. El programa maestro debe asignar los "números de instancia", enviándolos explícitamente a los diversos procesos. En principio el programa maestro dispone de los números de tarea tids arrancados, y podría asignar y enviar a cada tids(i) el argumento i, pero eso supondría un problema para la medición de tiempo que deseamos iniciar justo después de emitir todos los argumentos. Al no ser pvm_send bloqueante, el maestro podría comenzar a cronometrar mucho antes de que los procesos esclavos lleguen al pvm_recv correspondiente (incluso antes de que termine de arrancar el MATLAB esclavo). Se ha preferido esperar a que todos los esclavos respondan con un mensaje (el propio tid, por ejemplo) como método explícito de sincronización, para garantizar que todos están listos para empezar el cálculo en cuanto reciban el último argumento que necesitan (el índice del proceso).

Los esclavos deben por tanto enviar su tid al maestro, procediendo con el cálculo en cuanto se les comunique su índice de proceso (ver Listado 1.13).

El resultado parcial se envía al maestro para su acumulación. Las diferencias más notables con el modo punto a punto del programa PVMTB equivalente (Listado A.9) son las relaciona-

```
function Work(C,N)

me=pvm_mytid;
parent=pvm_parent;                                    % obtener inum, de paso Synch
pvm_initsend(PLACE); pvm_pkdouble(me,1,1); pvm_send(parent,TAG);
pvm_recv(parent,TAG); inum=pvm_upkdouble(1,1);

  width=1/N; lsum=0;                                  % código vectorizado, equivale a
  i=inum-1:C:N-1;                                     % for i=inum-1:C:N-1
  x=(i+0.5)*width;                                    %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                              %   lsum=lsum+4/(1+x^2);
                                                      % end
pvm_initsend(PLACE); pvm_pkdouble(lsum,1,1); pvm_send(parent,TAG);
pvm_exit;
```

Listado 1.13: Versión paralela DP-TB: código esclavo.

das con el empaquetamiento y desempaquetamiento. Conviene considerar que una llamada de la forma inum=pvm_upkdouble(1,1) crea la variable inum independientemente de que existiera previamente. Esto supone una copia adicional de memoria en el sistema global de paso de mensajes, reduciendo las prestaciones conforme crece el tamaño del array. En PVMTB el comando toma la forma pvm_upkdouble(inum). Una versión alternativa, pvm_upkdouble('inum'), permite usar el espacio de la variable como buffer si existe, o crearla si no.

Dejamos al proceso maestro esperando a recibir de los esclavos los mensajes de sincronización. Tras recibirlos se arranca el cronómetro, se asigna el índice de proceso y se esperan a los resultados parciales:

```
tic                                           % Medir tras sincr. inicial como MPI_Init
  for numt=1:C                                % Asignar inum (equivale a emitir args)
    pvm_initsend(PLACE); pvm_pkdouble(numt,1,1); pvm_send(tids(numt),TAG);
  end

  Psum=0;                                             % Acumulación
  for numt=1:C, pvm_recv(-1,TAG); Psum=Psum+pvm_upkdouble(1,1); end

  Psum       =Psum/N;                                 % width = 1/N
  Data.pi    =Psum;
  Data.err   =Psum-pi;
  Data.time=toc;                                      % tic-toc

pvm_exit;                                             % Salir
pvme_halt;
```

Listado 1.14: Versión paralela DP-TB: código maestro.

La sintaxis Psum=Psum+pvm_upkdouble(1,1) es atractiva para esta aplicación, simplificando la redacción del bucle de acumulación. Como se comentó al final del Apartado 1.3.1, contemplar esta sintaxis *adicional* en el respectivo comando PVMTB sería cuestión de añadir líneas de código comprobando el número de argumentos de salida requeridos por el usuario, similares a las que comprueban si la variable existía previamente o no en la versión alternativa de pvm_upkdouble para PVMTB. Como ya se dijo, la única limitación estriba en que los diversos patrones contemplados por una función no pueden ser ambiguos, debiendo distinguirse entre ellos por el tipo o número de argumentos. No contemplar el patrón pvm_upkdouble(vnam) limita las prestaciones de DP-TB en comparación con PVMTB.

En esta aplicación ha sido relativamente sencillo obviar la necesidad de operaciones colectivas. Aparte de su evidente utilidad y comodidad de uso, son operaciones que se presentan

frecuentemente en el desarrollo de aplicaciones paralelas. Por ejemplo, desde su primera versión el estándar MPI define una mayor variedad de operaciones colectivas que PVM. Se podría argumentar que siempre es posible implementar las funciones colectivas a partir de las primitivas punto a punto, pero siempre es conveniente disponer de una implementación al más bajo nivel posible. Si un fabricante crea una biblioteca PVM afinada a su arquitectura específica con soporte hardware para las llamadas colectivas, carecer del interfaz a la biblioteca de grupos impide aprovechar estas características de la arquitectura.

Usando todo el cluster, la (embarazosamente paralela) aplicación consigue ganancias entre 7 y 7.5 (Figura 1.10(a) derecha) a pesar del severo *overhead* introducido por DP-TB en el paso de mensajes (Figura 1.10(b)). Los tiempos de *setup* se incrementan de 0.2/0.4ms en PVM (Figura 1.6(b) izquierda) a 0.7/0.9ms en DP-TB, frente a los 0.4/0.6ms de PVMTB (Figura 1.8(b)). Dado que se ha tenido cuidado en usar la misma biblioteca PVM dinámicamente enlazada para probar la DP-TB, cabe atribuir un *overhead* de 0.3ms exclusivamente a la copia adicional de memoria en desempaquetado (unos 0.2ms) y al sistema de directorio empleados en la programación de DP-TB ($\simeq$0.1ms). Recordemos que PVMTB obtenía *speedups* prácticamente lineales, muy cercanos a 8 cuando se usa todo el cluster (Figura 1.8(a)).

Programa MultiMATLAB

El MultiMATLAB [59, 90] realizado por el antiguo equipo de Anne Trefethen en el Cornell Theory Center (Ithaca, NY, EEUU) es el trabajo más parecido a MPITB que se puede encontrar en la literatura. En comparación sólo ofrece 27 comandos siendo unos 20 de ellos rutinas MPI (ver Tabla 1.4), frente a los 153 comandos de MPITB que incluye 135 rutinas MPI y diversas constantes e identificadores definidos por el estándar. En particular, la creación dinámica de procesos (MPI_Comm_spawn()) no está contemplada en MultiMATLAB. Esto es consistente con la intención original de los autores, que no pretendían crear un entorno con toda la expresividad y control de un estándar de paso de mensajes como MPI, sino simplificar al máximo el acceso a la programación paralela bajo MATLAB. La primera versión de MultiMATLAB utilizaba la implementación MPICH como mecanismo subyacente de paso de mensajes.

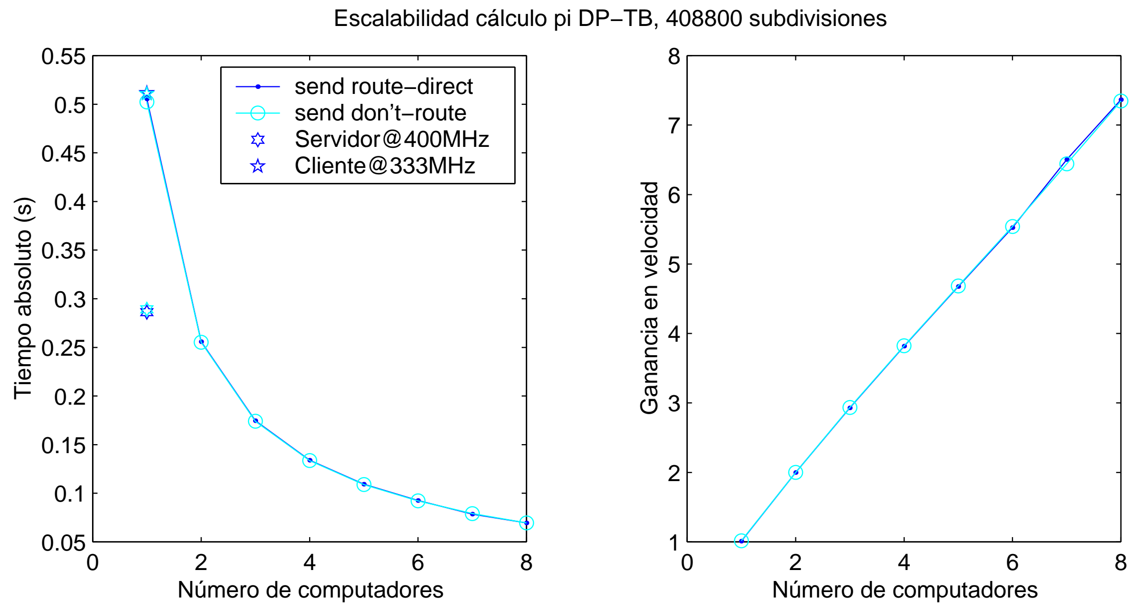
El software no se ofrece en las páginas Web relacionadas con [90], a pesar de que la oferta gratuita está documentada en Menon [59]. Gracias a Mr. Menon, disponemos de una copia de evaluación que ha sido necesario retocar para conseguir hacerla funcionar en nuestro cluster. En particular, no hemos conseguido que el comando Start (el primero de la Tabla 1.4) funcione como aparece documentado.

Hemos optado por la solución de arrancar los múltiples procesos MATLAB mediante mpirun, ofreciendo al hipotético usuario un *script* llamado MultiMATLAB para liberarle de implicarse en el control de LAM:

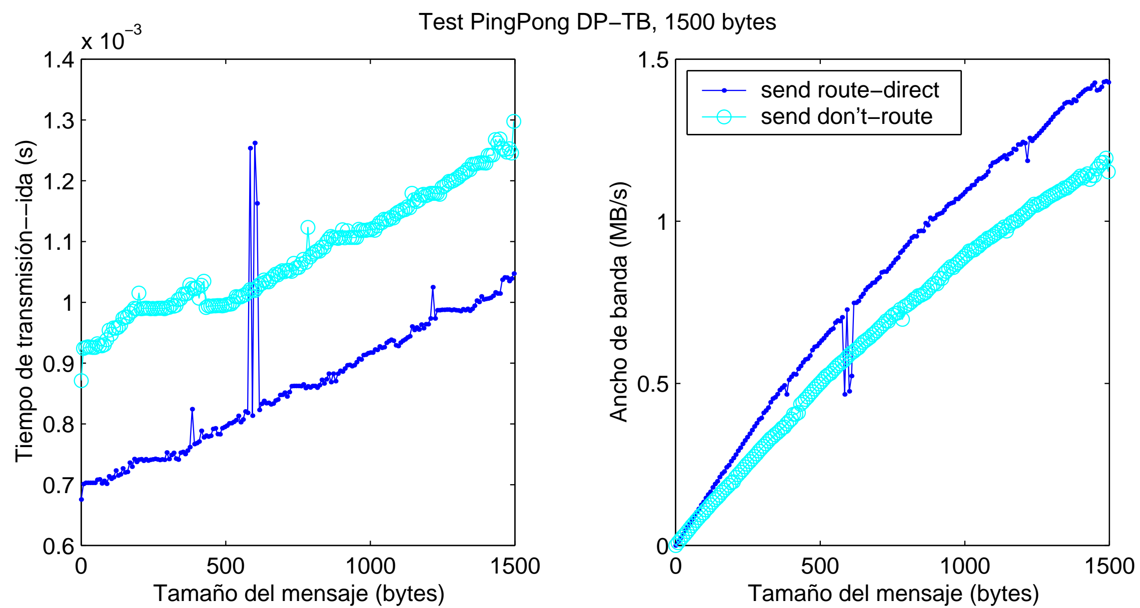
```
#!/bin/bash
lamboot
mpirun -w -O $OPTS -x DISPLAY N xterm --- -e matlab
wipe
```

Listado 1.15: Comando MultiMATLAB.

donde naturalmente no se usa la opción -nw de mpirun, para que wipe elimine los *daemons* sólo tras acabar los procesos MATLAB. También se declara que nuestro cluster es homogéneo



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.10: Tests realizados bajo DP-TB.

(-O), se permite seleccionar entre modo *daemon* o cliente-a-cliente según indique $OPTS, y se exporta la variable de entorno DISPLAY en donde deben visualizarse los terminales xterm en los que se ejecutan los procesos MATLAB. El Listado A.14 muestra la versión completa del comando MultiMATLAB.

Para arrancar una aplicación MultiMATLAB ya depurada podría prescindirse de los terminales, aunque la intención original de los autores era que sirviera de entorno interactivo, para lo cual es útil poder ver los posibles mensajes de error de cada proceso MATLAB.

El *script* **startup**.m que hemos creado para MultiMATLAB ajusta el *path* y contiene la llamada al programa MEX Init (proporcionado por MultiMATLAB, de uso interno, no para el usuario) que en condiciones normales se hubiera invocado desde el comando Start (el primer comando de la Tabla 1.4) para todos los procesos, tanto el arrancado por el usuario desde el *shell* como los arrancados por la propia orden Start. No hay pues necesidad de discriminar entre procesos ni de crear un startup_Slv.m auxiliar. El *script* para el estudio de escalabilidad puede ser reutilizado, con pequeñas modificaciones para contemplar las diferentes modalidades disponibles.

En principio se podría pensar en utilizar el comando de reducción Sum para el problema que estamos estudiando. El estudio de escalabilidad podría realizarse usando el comando Start(C+1) para arrancar desde el MATLAB inicial un MultiMATLAB con C computadores esclavos. Tras ejecutar el programa, se utilizaría Quit para acabar la sesión y de nuevo Start con un número diferente de procesadores. Desafortunadamente, no se consiguió que Start funcionara como está documentado. Sum reduce pues sobre todos los procesos MATLAB arrancados con nuestro *script* MultiMATLAB. Si se dispusiera de un interfaz a las rutinas de manejo de grupos y comunicadores (bastarían MPI_Comm_group, MPI_Group_incl y MPI_Comm_create), o mejor sólo usando MPI_Comm_split), podríamos ir creando comunicadores que progresivamente incluyeran más rangos miembros del MultiMATLAB inicial. Esta carencia impide el uso efectivo de las operaciones de reducción para

Tabla 1.4: Comandos MultiMATLAB y descripción de la finalidad del comando, o rutina MPI correspondiente.
* indica una correspondencia muy forzada, ** aún más. † pueden ejecutarse en cualquier proceso (0–Nproc-1).

	Arrancar y parar MultiMATLAB	
	Start(n)	`MPI_Init`*
	Interrupt(pid)	durante un cálculo
	Abort	`MPI_Abort`*
	Quit	`MPI_Finalize`*
	Organización e identificación de Procesos	
†	pid = ID	`MPI_Comm_rank`
†	n = Nproc	`MPI_Comm_size`
	Grid(m,n,p)	`MPI_Cart_create`*
†	[m,n,p] = Gridsize	`MPI_Cart_get`*
†	[i,j,k] = Coord	`MPI_Cart_get`*
	Ejecución sobre diversos Procesos	
	Eval(pid,'cmd')	ejecutar cmd en rangos
	Graficos	
	Window(m,n)	retícula de figuras
	ResetWindow	posición por defecto
	Refresh(ids)	actualizar gráficas

	Comunicación	
†	Send(pid,data,tag)	`MPI_Send`
†	[var,src,tag] = Recv(pid)	`MPI_Recv`
†	[flg,src] = Probe(pid)	`MPI_Iprobe`
†	Barrier	`MPI_Barrier`
	Put(pid,'vnam')	copiar datos a un rango
	var=Get(pid,'vnam')	traer datos de un rango
	Bcast('vnam',root)	`MPI_Bcast`
	Distribución de datos	
	Distribute(data)	`MPI_Scatter`**
	Collect(data)	`MPI_Gather`**
	Shift(data,axis,cnt)	`MPI_Cart_Shift/` `MPI_Sendrecv`*
	Reducción	
	m=Max('vnam')	`MPI_Reduce`
	m=Min('vnam')	`MPI_Reduce`
	s=Sum('vnam')	`MPI_Reduce`
	p=Prod('vnam')	`MPI_Reduce`

nuestro estudio de escalabilidad. Recuérdese que bajo MPITB se dispone de MPI_Comm_spawn para arrancar procesos dinámicamente, y de MPI_Intercomm_merge para reunirlos con el proceso maestro, por lo que no se presentó este problema. Excluyendo por tanto el uso de Sum, quedan disponibles dos modalidades de transmisión bajo MultiMATLAB:

Send/Recv: Equivalente a comunicación punto a punto MPI.

Put/Get: No requiere programación del usuario en los procesos remotos. Conceptualmente equivalente a RMA (Remote Memory Access) en MPI 2.0. Están implementadas basándose en Eval, Send y Recv. Por ejemplo, Put(i,'A') consiste en Send(i,A) y Eval(i,'A=Recv(0)').

La sintaxis escogida (ver Tabla 1.4) implica una copia de memoria adicional en Recv (y por tanto en Get), igual que sucedía con DP-TB. También MultiMATLAB está programado mediante el sistema de "directorio", y se enlazaba originalmente con la biblioteca MPICH estática. Por ejemplo, el comando Recv es un fichero M conteniendo la sentencia [a1,a2,a3]=MMAT(4,b,c,d), en donde MMAT es el fichero MEX directorio, cuyo código fuente MMAT.c contiene la sentencia (*(mmat[opcode]))(nlhs, plhs, nrhs, prhs). En concreto, mmat[4]==mm_receive es la función C que termina invocando a MPI_Recv().

El directorio presenta una peculiaridad importante en MultiMATLAB: el fichero MEX MMAT almacena información de estado en variables globales: el rango MPI_Comm_rank y el tamaño MPI_Comm_size se almacenan en tiempo de inicialización, y posteriores invocaciones a los comandos ID y Nproc devuelven el valor memorizado, en lugar de utilizar la llamada MPI correspondiente. El comando Grid redefine la topología malla de los procesos MATLAB, almacenándose en variables globales el tamaño de la malla y las coordenadas del proceso. Posteriores invocaciones a Gridsize y Coord devuelven los valores memorizados, en lugar de utilizar MPI_Cart_get. Esto obliga a bloquear el directorio en memoria, puesto que su borrado provocaría el mal funcionamiento de los comandos mencionados.

Volviendo a la discusión de la aplicación, sólo se utilizarán las modalidades Recv y Get, dado que el uso de Sum implica a todo el comunicador, dificultando considerablemente el estudio de escalabilidad: habría que salir y volver a entrar a MATLAB para poder añadir un computador. El programa maestro se muestra completo en el Listado A.15. El fragmento relevante del mismo es:

```
tic                                                       % Medir tiempo
                 Psum=0;                                  % evitar warning "uninitialized"
  Eval([1:C],['Psum=Work(' CC ',' NN ',''' MOD ''')'])    % emitir comando/argumentos
  switch MOD                                              % acumular
    case 's', for i=1:C, Psum=Psum + Recv; end
    case 'p', for i=1:C, Psum=Psum + Get(i,'Psum'); end
  end

  Psum      =Psum/N;                                      % width = 1/N
  Data.pi   =Psum;
  Data.err  =Psum-pi;
  Data.time=toc;                                          % tic-toc

Eval([1:C],'clear␣Psum␣functions')                        % limpiar
```

Listado 1.16: Versión paralela MultiMATLAB: código maestro.

Proporcionando valores por defecto para el comunicador, etiqueta, etc, MPITB podría conseguir un código igual de compacto. Tampoco habría mayor dificultad en contemplar el patrón var=MPI_Recv permitiendo expresiones como Psum=Psum+MPI_Recv a costa de copias de memoria

adicionales. Se podrían mencionar consideraciones didácticas (enseñar MPI usando MPITB) y de investigación (evitar que el usuario realice copias adicionales inopinadamente) en favor del diseño actual de MPITB, siendo también inmediato extenderlo si futuros usuarios lo desearan.

El programa maestro (Listado 1.16) activa el cronómetro y encarga la evaluación del comando esclavo en los C primeros clientes MultiMATLAB. Por razones de eficiencia se prefiere incluir los argumentos como texto en el comando (igual que se hizo con las *Toolboxes* anteriores) en lugar de transmitirlos previamente mediante Put. El comando Get (o Recv) se bloquea entonces hasta recibir el primer resultado parcial, y posteriormente acumula los restantes. Tras parar el cronómetro se limpian las funciones del espacio de trabajo de los esclavos (**clear** functions). Esto requiere una explicación adicional.

Cuando el entorno MATLAB interpreta por primera vez un *script* lo analiza transformándolo a un formato interno denominado "código-P" (*P-code*, por "pre-parsed code", código pre-analizado). De esta manera, la primera ejecución de un *script* es más lenta de lo que debería por incluir operaciones para conservar el resultado del pre-análisis, y las ejecuciones posteriores son más rápidas de lo que deberían por ahorrarse el preanálisis entero. El comando pcode permite salvar dicho código preanalizado a disco de manera que no sea necesario ningún preanálisis en futuras sesiones. El entorno también vigila las posibles modificaciones a los ficheros .m originales, de manera que el código-P en memoria se corresponda siempre con el *script* en disco.

En el estudio de escalabilidad, las *Toolboxes* anteriores repiten la aplicación sobre procesos MATLAB arrancados dinámicamente en cada iteración. Estos procesos no han ejecutado previamente ningún *script* MATLAB. Permitir que los MultiMATLAB esclavos recuerden el preanálisis del programa Work mejora artificialmente las prestaciones de esta *Toolbox*. El código esclavo se muestra completo en el Listado A.16. El fragmento relevante es:

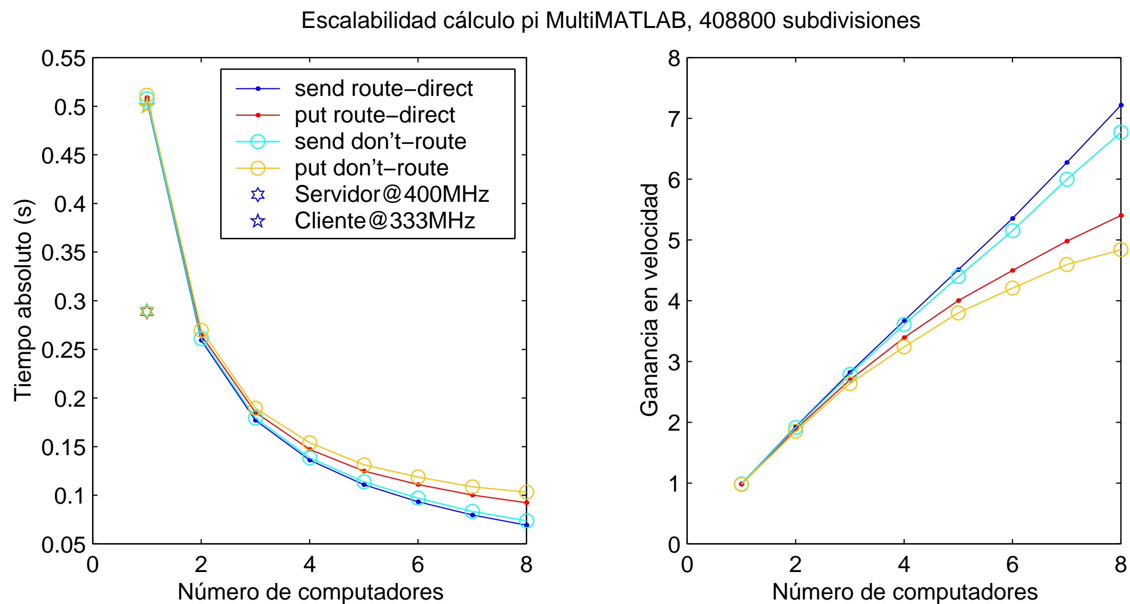
```
function lsum=Work(C,N,MOD)

rank=ID;
  width=1/N; lsum=0;                     % código vectorizado, equivale a
  i=rank-1:C:N-1;                        % for i=rank-1:C:N-1
  x=(i+0.5)*width;                       %    x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                 %    lsum=lsum+4/(1+x^2);
                                         % end
if strcmp(MOD,'s'), Send(0,lsum); end    % Put/Get es one-sided
```

Listado 1.17: Versión paralela MultiMATLAB: código esclavo.

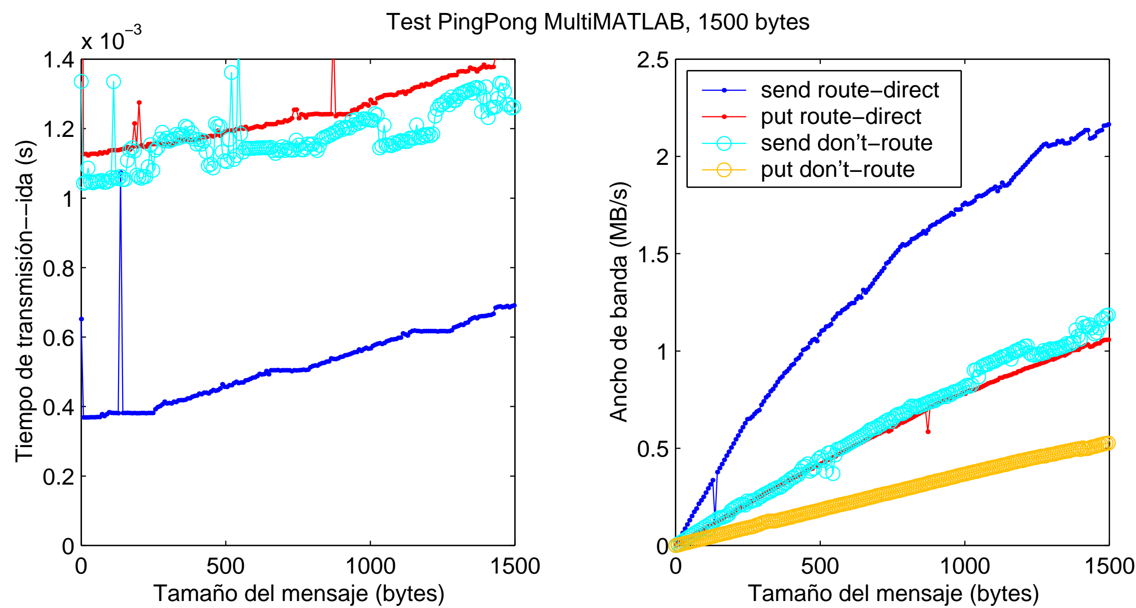
Para la modalidad Put/Get no se requiere programación por parte del usuario en el proceso esclavo, como se comentó anteriormente. Las gráficas del estudio de escalabilidad y del test *ping-pong* se muestran en las Figuras 1.11(a) y 1.11(b), respectivamente.

MultiMATLAB se separa de la escalabilidad lineal más que las *Toolboxes* previas. Recordemos que PVMTB y MPITB llegaban a un *speedup* de 8, y DP-TB superaba 7, ganancia apenas alcanzada por MultiMATLAB cuando utiliza todo el cluster. Las prestaciones se pueden subir artificialmente permitiendo que los procesos MultiMATLAB esclavos retengan el código pre-analizado. La misma subida artificial de prestaciones se puede aplicar a las *Toolboxes* anteriores, modificando el *script* de escalabilidad de manera que no se arranque un nuevo *daemon* ni nuevos procesos MATLAB a cada iteración, sino que se reutilicen los existentes. Otra posibilidad hubiera sido salvar a disco la versión *P-code* de todos los *scripts* implicados.

La modalidad Put/Get tiene un innegable atractivo desde el punto de vista del programador



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.11: Tests realizados bajo MultiMATLAB.

MATLAB. Desafortunadamente no presenta unos resultados convincentes. Incluso a pesar de ser la aplicación embarazosamente paralela, la ganancia se separa alarmantemente de la linealidad (5/8 usando todo el cluster). Nuestra opinión es que la forma más efectiva de atraer usuarios a la Computación Paralela pasa necesariamente por ofrecer ganancias significativas, y, dentro de lo posible, comodidad de uso, pero nunca sacrificando lo primero por lo último.

El test *ping-pong* revela que las rutinas de paso de mensajes son sólo parcialmente responsables de esta separación de la linealidad. Recuérdese que los tiempos de *setup* rondaban los 0.4/0.6ms en PVMTB (Figura 1.8(b)), 0.2/0.5ms en MPITB (Figura 1.9(b)), y 0.7/0.9ms en DP-TB (Figura 1.10(b)), a pesar de lo cual DP-TB conseguía una aceptable ganancia de casi 7.5 en el estudio de escalabilidad. En el test *ping-pong* MultiMATLAB, la Figura 1.11(b) derecha muestra que el modo send con ruta directa presenta unas prestaciones similares a las de PVMTB, y aún así la ganancia es bastante inferior. En concreto, los modos send a través del *daemon* y Put/Get con ruta directa presentan similares prestaciones y ganancias muy dispares. El modo Put/Get tiene un *setup* de 2.7ms que lo excluye de la gráfica.

La codificación de los comandos MultiMATLAB presenta multitud de comprobaciones intermedias para soportar los distintos patrones aceptados (valores por defecto de etiquetas de mensaje, comunicador, etc) tanto en los ficheros M de cada comando como en la subrutina de directorio que recubre a cada llamada MPI. Aparentemente gran parte de este costo queda oculto en el test *ping-pong*, progresando el proceso fuente en las comprobaciones de la siguiente llamada de recepción, mientras que el proceso eco termina la recepción del mensaje anterior y prepara el envío correspondiente. Estos costes vuelven a quedar de manifiesto en el bucle de acumulación de la aplicación, al ser la llamada Recv bloqueante y no permitir progresar en las comprobaciones de la siguiente llamada. De nuevo, comprobar objetivamente esta suposición implicaría usar el mecanismo de traza MPI, alterando el propio fenómeno bajo estudio.

Programa TCPIP

La *Toolbox* TCPIP desarrollada por Peter Rydesäter de la Mitthögskolan Östersund (Suecia) es un interfaz básico para comunicación TCP/IP*. Ya que la capa de transporte tanto de PVM como de LAM/MPI se basa en TCP/IP, pareció conveniente estudiar las características de esta *Toolbox*. Desafortunadamente no contempla ningún mecanismo de ejecución remota. El objetivo del autor no es utilizar paso de mensajes para programación paralela, sino posibilitar la comunicación entre procesos de computadores conectados por TCP/IP. Así lo atestiguan los ejemplos de demostración incluidos:

popmail: Se conecta al servidor POP (*Post-Office Protocol*) de una máquina remota y devuelve las cartas del usuario. Se debe especificar nombre del servidor (el puerto POP3 es el 100), nombre del usuario y contraseña.

webserver: Se deja un proceso MATLAB ofreciendo servicio en un puerto arbitrario (el autor escoge el puerto 2000). Especificando la URL http :// localhost:2000/<comando MATLAB> en un *browser* www, el proceso servidor MATLAB evalúa el comando y genera una página web mostrando el resultado.

La Tabla 1.5 es una relación de los comandos disponibles. Bajo TCP/IP un proceso (cliente) puede establecer una conexión con otro proceso (servidor) abriendo un *socket*, y escribir o

leer datos del mismo, como si de un fichero se tratara. El proceso servidor se identifica mediante dirección IP y nº de *puerto*. El servidor usualmente se bloquea esperando (*escuchando*) conexiones en el puerto de servicio abierto a tal fin. Cuando el proceso cliente se conecta al servicio, se crea el *socket* a través del cual se realiza el diálogo cliente-servidor de acuerdo con el protocolo establecido para este servicio o número de puerto. El *socket* de diálogo se destruye tras prestar el servicio al cliente. Usualmente el proceso servidor vuelve a *escuchar* en el puerto asociado al servicio, esperando conexiones de nuevos clientes. También es usual que haya varios procesos servidores, o un proceso servidor multihebra (*multithreaded*), de manera que varios clientes puedan abrir su *socket* de diálogo para recibir servicio concurrentemente.

Al no ofrecer la *Toolbox* TCPIP un método para el arranque remoto de aplicaciones, tuvimos que desarrollar a tal efecto nuestro propio código MATLAB basado en rsh (Listado A.17). Las líneas relevantes se muestran en el Listado 1.18.

Se usa el comando MATLAB **unix** para ejecutar un *shell* remoto en el ordenador indicado como primer argumento. En dicho computador se ejecuta un proceso MATLAB, usando una ventana xterm propia si se indicó el modo de depuración. El segundo argumento es un *cell-array* conteniendo las líneas de texto que se desea que ejecute el proceso MATLAB. Esto se consigue incluyéndolas en el fichero **startup**.m. El modo de depuración prepende al **startup** líneas de código para poner el proceso MATLAB bajo el control del *debugger*. Aunque el sistema de ficheros del servidor está compartido con los clientes (y un nuevo fichero en disco es eventualmente visible en todos ellos), se prefiere copiar explícitamente el fichero **startup** al cliente para evitar el mecanismo de cache NFS para atributos de directorios. Bajo Linux, por defecto acdirmin=30s y acdirmax=60s, indicando que un cliente NFS actualiza la visión que tiene de los directorios (*Attribute Cache for DIRectories*) entre una y dos veces por minuto.

Tabla 1.5: Comandos TCPIP y descripción de su finalidad, o llamada TCP/IP correspondiente.

Creación de *sockets* en servidor	
sck=tcpip_servsocket(port)	`socket()`, `bind()`, `listen()`, `accept()`
sck=tcpip_servopen(port)	(obsoleta) `socket()`, `bind()`
sck=tcpip_listen(sck)	(obsoleta) `accept`
Creación de *socket* en cliente	
sck=tcpip_open(host,sck)	`socket()`, `connect()`
tcpip_close(sck)	`close()`
Estado, Lectura y Escritura	
st=tcpip_status(sck)	Conectado / no conectado / cliente / servidor
var=tcpip_read(sck,len,typ)	`read()`, `/recv()`
str=tcpip_readln(sck,len)	`recv()` y consume hasta <CR>/<LF>
str=tcpip_viewbuff(sck,len)	`recv()` y no consume
unsent=tcpip_write(sck,arg...)	`write()`, `send()`
Utilidades	
len=tcpip_getfile(sck,fname)	Recibe fichero (usa `tcpip_readln`)
err=tcpip_sendfile(sck,fname)	Envía fichero (usa `tcpip_write`)
var=tcpip_getvar(sck)	Recibe y carga fichero `.mat` (usa `tcpip_getfile`)
err=tcpip_sendvar(sck,var)	Salva y envía fichero `.mat` (usa `tcpip_sendfile`)
Demos	
popmail_demo(host,user,pass)	Muestra correo obtenido de servidor POP
webserver_demo	Ejecuta comandos MATLAB en URL browser

```
function Spawner(host,lines,debug)                 % Arrancar MATLAB en host
                                                   % poniendo "lines" en startup
fid=fopen('startup.m','w');       if debug         % Si se desea depuración
  fprintf(fid,'dbtype\n');                         % se visualiza startup
  fprintf(fid,'keyboard\n');      end              % (y se entra en el debugger)
  fprintf(fid,'delete startup.m\n');               % antes de borrarlo
  fprintf(fid,'%s\n',lines{:});   if ~debug        % Si no se desea depuración
  fprintf(fid,'quit\n');          end              % se sale de MATLAB al acabar
fclose(fid);
unix_cmd=['rcp startup.m ' host ':`pwd`'];         % Copiar startup a cliente remoto
unix(unix_cmd);                                    % Asunto acdirmin cache NFS

if debug                                           % El modo de depuración
  DISP=getenv('DISPLAY');                          % ejecuta MATLAB en xterm
        unix_cmd=['xterm -display $DISPLAY -e matlab']; % para poder teclear/ver errores
else ,  unix_cmd=['matlab > /dev/null'];           % El modo normal no muestra nada
end
        unix_cmd=['rsh -n ' host ' "cd ' pwd '; ' unix_cmd '" &'];
        unix(unix_cmd);
```

Listado 1.18: Comando Spawner para arranque remoto de MATLAB.

Al *script* para estudio de escalabilidad se le añadió código para arrancar en cada iteración un MATLAB adicional en otro cliente del cluster. El Listado A.18 muestra el *script* completo. Las líneas relevantes son las que aparecen en el Listado 1.19.

Así pues, en la primera iteración se le encarga al primer cliente (ox1 en el Listado 1.19) repetir C veces el comando Work, y en la última iteración se le encarga al último cliente una única ejecución (con argumento $i = C = 8$ en el caso de nuestro cluster). En la llamada a Work se especifican los argumentos comunes: número de esclavos, número de subdivisiones y modo de comunicación. Queda por especificar un argumento variable, el "índice de proceso", que se enviará explícitamente a cada esclavo.

Para establecer las conexiones de diálogo con el servidor, los clientes deben realizar un bucle de espera ocupada. El código maestro (ver la versión completa en el Listado A.19) debe ejecutar varios bucles, uno por cliente. Una vez establecidas las conexiones de diálogo, se arranca el cronómetro y se emite el argumento "índice de proceso" a los clientes, como se muestra en el Listado 1.20.

El índice de proceso se ha asignado en el orden en que se establecieron las conexiones TCP/IP. Respecto al modo de comunicación, de entre los comandos de la Tabla 1.5 hemos utilizado:

tcpip_write/read: basadas en las llamadas send(2) y recv(2) de la biblioteca C estándar libc, respec-

```
startup={'start=00;',...                           % esta línea es la que cambia
        ['C=' CC ';'],...                          % todos cooperan en última iter.
         'for i=start:C',...                       % cada uno hace Work varias veces
         '  i',...                                 % (informativo para modo DEBUG)
        ['  Work(i,' NN ',''' MOD ''')'],...       % ox1 participa 8 veces (todas)
         '  clear Work',...                        % (borrar el preanálisis)
         'end'};                                   % ox8 participa 1 vez (la última)

for i=1:C,            ii=int2str(i);               % en cada iteración
  startup{1}=['start=' ii ';'];                    % se modifica 1ª línea startup
  Spawner(['ox' ii], startup, DEBUG)               % nuevo MATLAB hace C-i+1 Works
  Par(i)=Mast( i,N,MOD);                           % repetir con i clientes
end
```

Listado 1.19: Script Speedup para estudio de escalabilidad en TCPIP.

```
PORT=2000; sock=tcpip_servsocket(PORT);                       % Abrir puerto servicio
fids=-1*ones(1,C);                                            % Abrir conexiones diálogo
for i=1:C                                                     % Esperar C conexiones
  while fids(i)<0                                             % No hay solicitud (fid==-1)?
    fids(i)=tcpip_listen(sock);                               % reintentar
end, end                                                      % C bucles de espera ocupada
tcpip_close(sock)                                             % servicio cumplido, cerrar

tic                                                           % Medir tiempo
switch MOD                                                    % emitir argumentos
  case 'w', for i=1:C, tcpip_write   (fids(i), i-1); end      % el índice va de 0 a C-1
  case 's', for i=1:C, tcpip_sendvar(fids(i), i-1); end
end
```

Listado 1.20: Apertura del servicio, conexiones de los clientes y cierre del servicio.

tivamente. En el interfaz proporcionado, `recv()` devuelve el array vacío si no se han recibido datos aún. Está por tanto orientado a recepción mediante espera ocupada.

tcpip_sendvar/getvar: construidas sobre `tcpip_send/getfile`, que a su vez se basan en `tcpip_write/read`. La variable se salva previamente a un fichero que es posteriormente transmitido. Al seguir un protocolo interno para la transmisión de ficheros, resuelve internamente la espera ocupada, simplificando el código del usuario a costa de reducir prestaciones.

El Listado A.20 ofrece el código esclavo completo. Tras establecer conexión con el servidor, se recibe el índice emitido y se procede con el cálculo, como refleja el Listado 1.21.

El resultado parcial se devuelve usando la misma modalidad que en recepción. Este envío señala de paso al servidor el inminente cierre de la conexión. Para garantizar que el proceso maestro obtenga el resultado antes de que el esclavo cierre la conexión (eventualidad no soportada por TCPIP), se completa el *handshake** obligando al esclavo a esperar una confirmación del maestro, que tras emitir el índice de proceso al final del Listado 1.20, comienza a esperar resultados parciales como se muestra al principio del Listado 1.22.

Se debe considerar que cuando hay un número elevado de esclavos puede transcurrir un tiempo significativo (a escala TCPIP) desde que el maestro lee del primer esclavo hasta que lee del último: el tiempo empleado por el maestro en realizar las acumulaciones. Al no ser bloqueante el envío (la recepción tampoco lo es), el último esclavo podría estar cerrando la conexión mientras el maestro está acumulando todavía resultados de esclavos anteriores. El *handshake* a cuatro bandas propuesto evita este problema.

```
rank=[]; switch MOD                                           % recepción del índice
  case 'w', while isempty(rank), rank=tcpip_read(fid,1,'double'); end
  case 's', rank=tcpip_getvar(fid);
end

  width=1/N; lsum=0;                                          % código vectorizado, equivale a
  i=rank:C:N-1;                                               % for i=rank:C:N-1
  x=(i+0.5)*width;                                            %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                                      %   lsum=lsum+4/(1+x^2);
  switch MOD                                                  % end
    case 'w', tcpip_write   (fid,lsum)                        % devolver resultado
    case 's', tcpip_sendvar(fid,lsum)
  end
tcpip_getvar(fid);                                            % Handshake avisando de cierre
tcpip_close (fid);                                            % Cerrar
```

Listado 1.21: Versión paralela TCPIP: código esclavo.

```
Sum=0;                                              % acumular resultados
switch MOD
  case 'w', for i=1:C                               % modalidad write/read
    Psum=[]; while isempty(Psum)                    % espera ocupada en recepción
      Psum=tcpip_read(fids(i),1,'double');
    end
    Sum=Sum+Psum;
  end
  case 's', for i=1:C                               % modalidad send/getvar
    Sum=Sum+tcpip_getvar(fids(i));
  end
end % switch

Sum       =Sum/N;                                   % width = 1/N
Data.pi   =Sum;
Data.err  =Sum-pi;
Data.time =toc;                                     % tic-toc

for i=1:C,     tcpip_sendvar(fids(i),i);            % Handshake avisando de cierre
pause(0.1),    tcpip_close  (fids(i));              % Cerrar
end
```

Listado 1.22: Versión paralela TCPIP: código maestro.

Incluso usando este *handshake*, se presenta el problema de que el maestro cierra demasiado pronto en relación con el comando tcpip_getvar en el esclavo. No pudiendo evitarse el problema con bandas adicionales (que irían trasladando el problema del esclavo al maestro y viceversa) se ha optado por eliminar el problema con un retraso suficiente (de hecho excesivo a escala TCPIP) antes de cerrar. El retraso, como se muestra al final del Listado 1.22, está por supuesto fuera de la medición de tiempo.

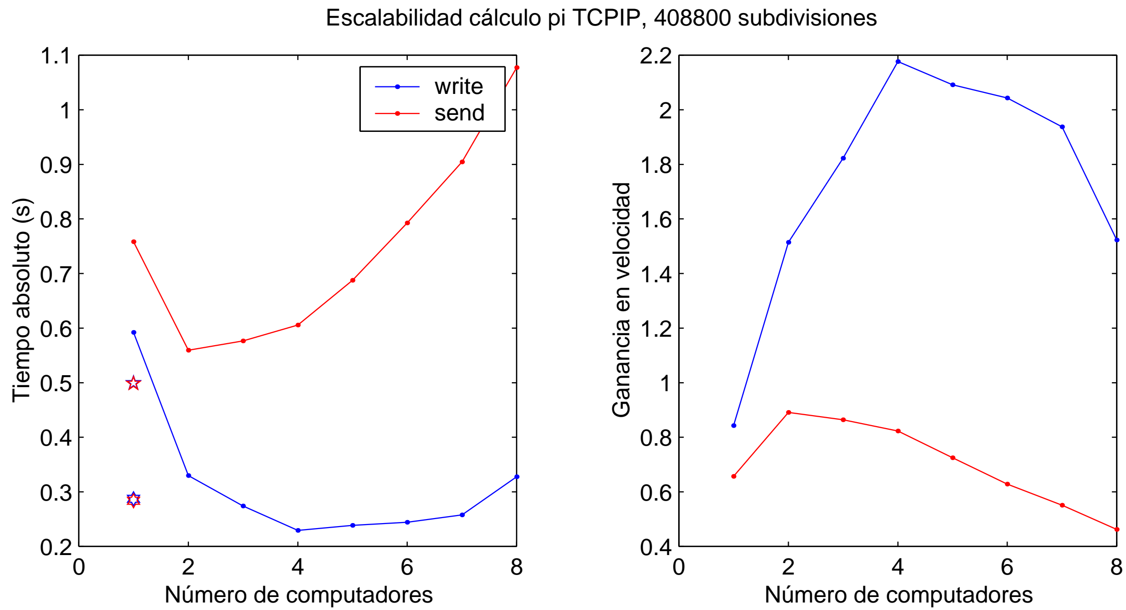
Esta aplicación es tan sencilla que puede ser programada sin excesiva dificultad con el interfaz TCP/IP proporcionado. Aunque la intención original del autor era proporcionar conectividad TCP/IP entre MATLAB y otros procesos ya arrancados, la ausencia de un mecanismo de ejecución remota se puede resolver utilizando los mecanismos contemplados por el Sistema Operativo y el comando MATLAB de interfaz con el *shell* **unix**. También se podrían arrancar manualmente los procesos MATLAB requeridos, pero para aplicaciones reales en etapa de producción dicho mecanismo es vital. Incluso para una aplicación pequeña como el ejemplo que nos ocupa, nos ha merecido la pena desarrollar nuestro propio mecanismo de arranque remoto. Así el estudio de escalabilidad se puede repetir automáticamente sin esfuerzo.

La delicada temporización exigida para la operación de cierre de conexión tampoco es excesivamente problemática, especialmente en un cluster con conexiones dedicadas, en donde la técnica de retardo mostrada resulta suficientemente robusta.

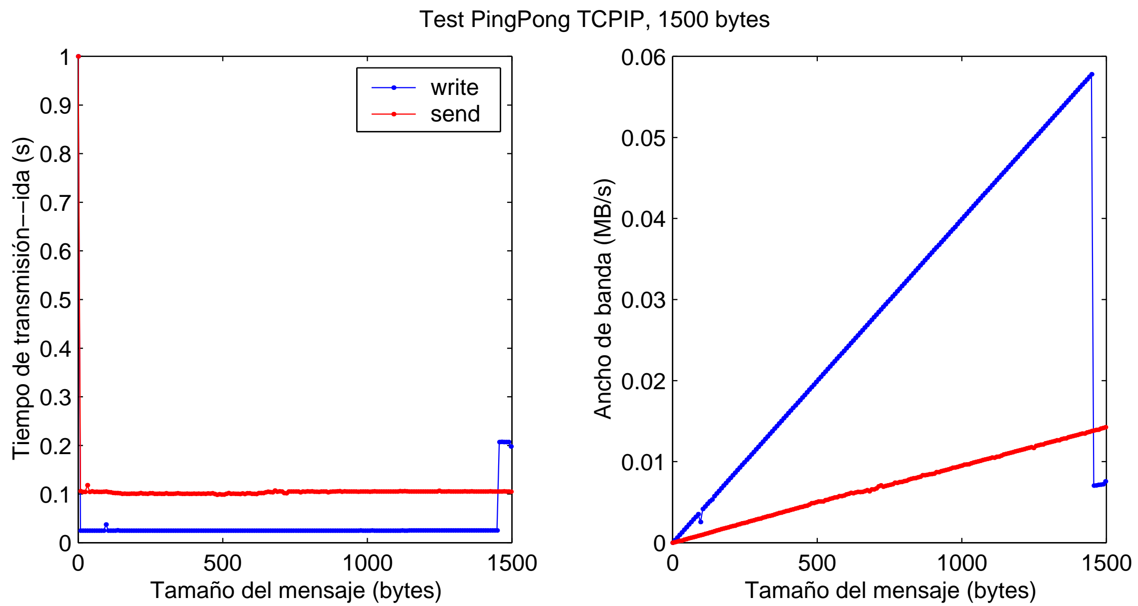
En cualquier caso, el autor de la *Toolbox* TCPIP no reivindica su uso para HPC, habiendo sido incluida en la exposición por motivos ilustrativos. Las gráficas del estudio de escalabilidad y del test *ping-pong* se muestran en las Figuras 1.12(a) y 1.12(b), respectivamente.

El *overhead* asociado con la *Toolbox* TCPIP es a todas luces excesivo en comparación con las *Toolboxes* anteriores, basadas en paso de mensajes. El protocolo utilizado para la modalidad send es muy ineficiente debido al paso intermedio de escritura en disco y posterior transmisión del fichero resultante, no obteniéndose *speedup* superiores a la unidad.

Un detalle técnico de la modalidad write no documentado (por la *Toolbox* TCPIP) es que el asociado tcpip_read devuelve fragmentos de tamaño MTU* como máximo. El resto de la información transmitida se obtiene con posteriores comandos tcpip_read. Esta peculiaridad queda de mani-



(a) Estudio de escalabilidad del cálculo de π.



(b) Test *ping-pong* hasta 1500 bytes.

Figura 1.12: Tests realizados bajo TCPIP.

fiesto en el test *ping-pong*. En nuestro sistema, la Unidad de Transferencia Máxima es de 1500 bytes. Se observa el retraso debido a la segunda lectura unos pocos bytes (correspondientes a cabeceras Ethernet y TCP/IP) antes de llegar a dicho tamaño.

Programa Paralize

La *Toolbox* Paralize, disponible libremente en *The MathWorks Downloads* [52], explora la posibilidad de usar un sistema de ficheros compartido (en red) como mecanismo de paso de mensajes para computación paralela. Al igual que la *Toolbox* anterior, carece de mecanismo de ejecución remota. A diferencia de ella, el autor reivindica su uso para HPC, y proporciona en la documentación del software una gráfica de un experimento en el que 9 computadores consiguen un tiempo de ejecución menor que el de un computador escalar lento (sic.) para algún tamaño de matriz problema entre 75x75 y 100x100.

No se indica en la documentación si es conveniente configurar parámetros específicos del sistema de ficheros (al estilo de acdirmin/accdirmax en NFS). Habiendo estado expuestos al problema de actualización NFS en alguna ocasión (recordar el comando rcp en el Spawner TCPIP, Listado 1.18), no queda claro si el autor usó un sistema de ficheros compartido distinto (no estando informado por tanto de qué parámetros NFS serían de interés), si olvidó comentar en la documentación cómo configurar NFS para usar provechosamente su *Toolbox*, o si simplemente considera aceptable el eventual retraso de 30–60 segundos entre actualizaciones de subdirectorios NFS. Este valor por defecto ha sido cuidadosamente ajustado tras años de experiencia con instalaciones NFS bajo Linux, y no parece sensato reducirlo, ya que cargaría excesivamente al servidor.

La *Toolbox* consiste en dos comandos: serve, que consulta periódicamente el sistema de ficheros compartido en busca de anotaciones que satisfacer, y paralize, que realiza las anotaciones de comandos a ejecutar. No se considera por tanto la posibilidad de comunicación entre los procesos esclavos serve. La anotación especial paralize(' kill ') indica a todos los procesos serve que deben dejar de ejecutarse.

Al menos un argumento de la aplicación paralela debe ser una matriz 3D, cuya tercera dimensión se utiliza como índice de paralelización. Si hay varios, sus terceras dimensiones deben tener el mismo tamaño. Los argumentos de menos de 3 dimensiones se consideran fijos y se repiten para cada valor del índice de paralelización. El comando a ejecutar en paralelo debe ser SPMD y no producir dependencias (comunicación) entre los diversos procesos serve que realizan el cálculo. El comando paralize selecciona consecutivamente cada sección 2D de los argumentos 3D a lo largo de la tercera dimensión, y anota en disco el comando a ejecutar y los argumentos que se le han asignado, donde algún proceso serve libre los captará. Los resultados son devueltos de la misma manera.

Las anotaciones en disco son ficheros siguiendo la nomenclatura process<to>_<idx>.mat, en donde <to>=in para anotaciones paralize y <to>=out para anotaciones serve. Los ficheros contienen las siguientes variables:

process_no: (in/out) al igual que el campo idx en el nombre del fichero, refleja el índice en la tercera dimensión correspondiente a esta anotación.

whattodo: (in) *string* conteniendo el texto a ejecutar, como por ejemplo '[q1]=Mast(p1,p2);'. Los

argumentos son renombrados sistemáticamente, p<i> los de entrada y q<i> los de salida.

p1, p2...: (in) argumentos obtenidos de la interpretación del comando paralize. Al menos uno de ellos proviene del process_no-ésimo elemento de un array 3D.

q1, q2...: (out) valores devueltos por el proceso serve que atendió la anotación.

Siendo excesivamente laborioso y propenso a errores ir arrancando procesos MATLAB adicionales en distintos computadores clientes para realizar el estudio de escalabilidad, se prefirió reutilizar la automatización desarrollada para la *Toolbox* TCPIP, modificándola ligeramente. El comando Spawner (Listado 1.18) se simplifica eliminando el segundo argumento (que especificaba las líneas a incluír en el **startup**.m) puesto que los computadores clientes siempre ejecutarán el comando serve sin argumentos.

El *script* Speedup también resulta simplificado. Bajo TCPIP era necesario especificar cuántas veces se ejecutaba el comando Work y los parámetros para cada ejecución. Bajo Paralize, el proceso servidor de cálculo serve (ejecutado en los computadores clientes del cluster) realiza un bucle esperando anotaciones y evaluándolas hasta encontrar una anotación paralize(' kill '), haciendo innecesario especificar el número de anotaciones que se atenderán. En nuestro caso, el lugar apropiado para la anotación ' kill ' es al final del estudio de escalabilidad (Listado A.21). La sección relevante es:

```
for  i=1:C,        ii=int2str(i);                          % en cada iteración
  Spawner(['ox' ii], DEBUG)                                % añadir un serve
  if DEBUG, input(['Pulsar␣<Enter>␣tras␣asegurarse␣de␣que␣hay␣',...
                  ii '␣esclavos␣ejecutando␣serve:␣']);
  else , fprintf('Esperando␣unos␣segundos␣para␣que␣arranque␣serve...');
                  pause(10); fprintf('\n');
  end                                                      % dejarle arrancar
  Par(i)=Mast( i,N);                                       % y realizar anotaciones
end

save(FN, 'C', 'Sec', 'Par')                                % salvar mediciones
paralize('kill')                                           % y limpiar cluster
```

Listado 1.23: Script Speedup simplificado para la *Toolbox* Paralize.

El código relativo al modo de depuración que hemos incluido el *script* nos ha sido de enorme ayuda para detectar los puntos clave en el código del proceso serve en donde se requeriría una actualización de la cache NFS. Se ha insertado en dichos puntos el comando **unix**('touch␣.') que provoca la actualización del subdirectorio de trabajo. Es nuestra impresión que ofrecer al público en general esta *Toolbox* sin avisar de que puede requerir conocimientos detallados del funcionamiento del sistema de ficheros compartido que se use puede resultar contraproducente para atraer a los usuarios de MATLAB al campo de la Computación Paralela.

Incluso con dichas mejoras, no es posible garantizar que en cada iteración las anotaciones se repartan equitativamente, siendo muy frecuente que algunos procesos serve atiendan más de una anotación (sobre todo el más recientemente arrancado) y por tanto otros tantos permanezcan inactivos durante esa iteración.

Otro problema insoslayable es la medición de tiempo. Nuestro código Speedup ofrece dos alternativas:

debug: el usuario, que puede ver las respectivas ventanas xterm y los mensajes de arranque de

los respectivos procesos MATLAB, indica por teclado cuándo ha terminado de arrancar el proceso serve adicional.

pause: la experiencia con nuestro cluster nos indica que el arranque de un proceso MATLAB remoto consume unos 5s. Se espera programadamente una cantidad de tiempo muy superior para garantizar sobradamente que, para cuando se inicie el cronómetro, los procesos serve estén ejecutándose.

Según el criterio que seguimos en este estudio comparativo, el lugar apropiado para iniciar la medición de tiempo sería el momento de creación del fichero de anotación, dentro del comando paralize. Ese instante es el equivalente al punto en que se emiten los argumentos en las restantes *Toolboxes*. Dado que esta *Toolbox* consta de 2 comandos y uno de ellos ha requerido modificaciones para no obtener resultados disparatados, se ha preferido no correr el riesgo de acabar desarrollando nuestra propia *Toolbox*, dejando al autor la tarea de realizar mejoras adicionales.

El programa maestro, que aparece completo en el Listado A.22, inicia el cronómetro, realiza las anotaciones y acumula los resultados (Listado 1.24):

```
i=0:C-1;                                   % generar equivalente a rank/inum
if C==1, i=[0  N]; end                     % C==1 -> i==[0] (singleton)
I=shiftdim(i,-1);                          % pasar (1xC) -> (1x1xC) C>1

  tic
  O=paralize('Work',I,C,N);                % arg. paralelización (I) es 3D
  Sum      =sum(O(:))/N;                   % si C==1, 2ºproc for i=N:C:N-1
  Data.pi  =Sum;                           %           dummy -> lsum==0
  Data.err =Sum-pi;
  Data.time=toc;                           % tic-toc
```

Listado 1.24: Versión paralela Paralize: código maestro.

El usuario observa en pantalla mensajes internos del comando paralize indicando el orden en que se hacen las anotaciones y el orden en que aparecen las anotaciones de respuesta de los procesos serve. No es posible garantizar que cada proceso se encarga de una anotación. El sistema no es tan controlable como con otras *Toolboxes*.

El formato de los ficheros de anotación tampoco informa de cuál computador cliente atendió cada anotación. Nuestro modo de depuración, al ejecutar los MATLAB esclavos en un xterm, permite al usuario observar los mensajes de los procesos serve y deducir cuál computador cliente atendió cada anotación, pero tendría que anotar manualmente dicha información, lo cual es propenso a errores durante un estudio de escalabilidad (es básicamente agotador, motivo por el cual se automatizan siempre los estudios de escalabilidad). La *Toolbox* no permite colectar dicha información programadamente. El sistema no es tan observable como con otras *Toolboxes*.

Para nuestra aplicación debemos pasar los argumentos rank/inum, C y N. Estos dos últimos son fijos, y Paralize requiere expresar los argumentos variables como matrices 3D, lo cual se consigue fácilmente generando un vector fila (1xC) con los índices de los C procesos y añadiendo una primera dimensión singleton con shiftdim.

En MATLAB se le aplica el epíteto *singleton* a las dimensiones de una matriz que contienen un único valor de índice. Así, la segunda dimensión de un vector columna (nx1) es *singleton*, igual que la primera dimensión de un vector fila (1xn), o ambas de un escalar (1x1). El tipo básico MATLAB es la matriz bidimensional, siendo considerados los vectores y escalares como matrices

2D en las que una o ambas dimensiones son *singleton*. En este último caso, el calificativo se aplica también a la variable en sí, pudiéndose decir que un escalar *es* un singleton 2D.

No es posible crear bajo MATLAB un singleton 3D (1x1x1), es decir, un escalar indexado en tres dimensiones siendo todas ellas singleton. MATLAB elimina cuidadosamente la tercera dimensión si no es utilizada, reduciéndolo a la representación canónica de un escalar como singleton 2D. En general, en una matriz resultado (mx...xn x1x...1) se eliminan automáticamente las dimensiones singleton de cola (consultar los comandos **cat**, **reshape**, **squeeze**, **shiftdim**, etc).

Tenemos pues un problema técnico para expresar el caso en que sólo un procesador esclavo realiza cálculos (C==1), dado que la forma de nuestro "dato de paralelización" debería tomar la forma de singleton 3D. El autor de Paralize no contempla este caso especial. Se ha optado por anotar en dicho caso dos comandos para el único servidor, siendo el primero Work(0,C,N) y el segundo Work(N,C,N), que es la forma más rápida de no afectar el resultado, como veremos.

Tras acumular los resultados parciales se calcula el tiempo empleado por los procesos esclavos. Estos ejecutan el código usual (Listado 1.25):

```
function lsum=Work(offs,C,N)

  width=1/N; lsum=0;                 % código vectorizado equivalente a
  i=offs:C:N-1;                      % for i=offs:C:N-1
  x=(i+0.5)*width;                   %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));             %   lsum=lsum+4/(1+x^2);
                                     % end
```

Listado 1.25: Versión paralela Paralize: código esclavo.

en donde se comprueba que la segunda orden ficticia para el caso C==1 (Work(N,C,N)) genera un vector fila vacío y un resultado lsum==0, no afectando al posterior paso de acumulación en el maestro.

La gráfica del estudio de escalabilidad se muestra en la Figura 1.13. No es viable hacer un test *ping-pong* con resultados razonables bajo Paralize.

1.7 Conclusiones

En este capítulo se han introducido las *Toolboxes* PVMTB y MPITB junto con los restantes trabajos previos en la misma línea, realizándose un estudio comparativo basado en un ejemplo sencillo de paralelización encontrado frecuentemente en la literatura. El estudio considera no sólamente las ganancias en velocidad obtenidas con el estudio de escalabilidad de la aplicación, y la evaluación de prestaciones del paso de mensajes mediante el habitual test *ping-pong*, sino que también contempla el esfuerzo de prototipado asociado con la automatización de las mediciones y la controlabilidad y observabilidad de la aplicación al ser ejecutada en paralelo en un cluster de computadores. Las *Toolboxes* paralelas más significativas siguen el paradigma del paso de mensajes, requiriendo una instalación de PVM o MPI.

Los dos sistemas de paso de mensajes más utilizados, PVM y MPI, se basan en el uso de bibliotecas específicas bajo un compilador C o FORTRAN. El interfaz MEX disponible en el entorno MATLAB permite crear una *Toolbox* que realize llamadas a una biblioteca de paso de mensajes. De este modo, varios procesos MATLAB ejecutándose en diferentes computadores de

un cluster pueden realizar paso de mensajes entre ellos, posibilitando al usuario prototipar una aplicación HPC bajo MATLAB.

La evolución comercial de MATLAB refleja la opinión del fabricante respecto a que el paralelismo interno de baja granularidad debe ser extraido o explotado por el Sistema Operativo u otro software (biblioteca matemática, etc); en cualquier caso, nunca por el propio MATLAB. La subdivisión interna en hebras lógicas de procesamiento (intérprete, gráficos y cálculo), así como la incorporación de LAPACK (susceptible de ser enlazado con ATLAS en un SMP) son los dos puntos fundamentales que revelan el excepticismo de *The MathWorks* acerca de la conveniencia de ejecutar varios procesos MATLAB en el mismo computador.

Muchas aplicaciones MATLAB son de granularidad alta, incluso embarazosamente paralelas, con lo cual se pueden esperar *speedups* aceptables con el uso de una *Toolbox* de paso de mensajes como PVMTB o MPITB. Una vez depurado, el prototipo podría ser compilado para evitar el *overhead* asociado con el entorno interpretado. Las *Toolboxes* también pueden ser utilizadas simplemente para reducir el tiempo de ejecución bajo el entorno interpretado.

De entre los trabajos previos presentados en este Capítulo cabe destacar las *Toolboxes* DP-TB y MultiMATLAB, aunque ambas son trabajos preliminares: DP-TB consta de 56 comandos (44 llamadas PVM) frente a los 93 de PVMTB (incluyendo las 86 llamadas PVM), y MultiMATLAB se reduce a unas 20 llamadas MPI frente a los 153 comandos MPITB. Otras *Toolboxes* pueden parecer más sencillas de programar o instalar, pero no obtienen unas ganancias en velocidad comparables, ni siquiera para aplicaciones embarazosamente paralelas. De entre los paradigmas explorados por las *Toolboxes* paralelas disponibles, el paso de mensajes es el que mejores resultados proporciona bajo MATLAB.

La comparación de las *Toolboxes* basadas en paso de mensajes arroja la conclusión de que PVMTB y MPITB, las propuestas en la presente memoria, son las que menos *overhead* introducen en el mecanismo de paso de mensajes, lo cual se manifiesta incluso en los *speedups*

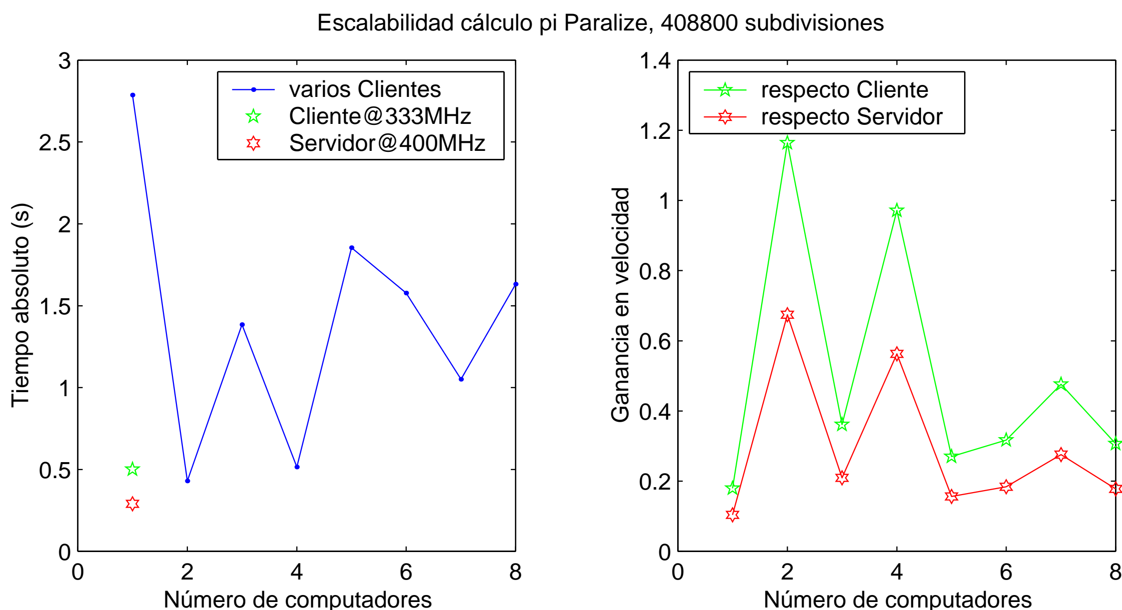


Figura 1.13: Estudio de escalabilidad del cálculo de π usando Paralize.

obtenidos con aplicaciones embarazosamente paralelas, a pesar de la elevada relación cálculo/comunicación de dichas aplicaciones.

Como rasgos distintivos de PVMTB cabe citar que permite el uso de grupos y llamadas colectivas PVM, soporta las máscaras de depuración, e incluso contempla la posibilidad de programar los manejadores PVM (*hoster*, *tasker*, etc) en lenguaje MATLAB. No incurre en copias de memoria adicionales ni se enlaza estáticamente a la biblioteca PVM usando el sistema de directorio.

Como rasgos distintivos de MPITB cabe citar que contempla prácticamente la totalidad del estándar MPI-1.2 y parte del MPI-2.0, incluyendo arranque dinámico de procesos (MPI_Comm_spawn) y conexiones cliente-servidor (MPI_Lookup_name, etc). Tampoco incurre en copias de memoria adicionales ni usa el sistema de directorio.

El estudio de escalabilidad muestra que el tiempo de cómputo de una aplicación programada en lenguaje C puede incrementarse significativamente cuando se recodifica en lenguaje MATLAB (un orden de magnitud en nuestro ejemplo), lo cual hace más interesante aún la utilización de nuestras *Toolboxes* simplemente para acelerar el cálculo bajo el entorno interpretado. Con aplicaciones embarazosamente paralelas como la estudiada en este capítulo hemos obtenido *speedups* prácticamente lineales.

Los tests *ping-pong* realizados en este capítulo demuestran que el *overhead* introducido por nuestras *Toolboxes* es en general inferior a una copia de memoria adicional, siendo apropiadas para prototipar aplicaciones HPC bajo MATLAB. Ningún trabajo previo consigue nuestras prestaciones.

Capítulo 2

Análisis de Prestaciones

Resumen del capítulo

En este capítulo se realiza una introducción a la problemática de analizar las prestaciones de un sistema de paso de mensajes, se describen y justifican los modelos desarrollados para estudiar las prestaciones del equipo utilizado (el cluster "oxígeno"), y se usan dichos modelos para establecer comparaciones entre las dos alternativas contempladas, PVM y LAM/MPI.

En el Apartado 2.1 se presenta el equipo *hardware* utilizado durante la realización de este trabajo. También se comentan brevemente algunos detalles significativos del proceso de instalación y configuración del equipo.

El Apartado 2.2 revisa los distintos aspectos a considerar en la evaluación de prestaciones de un sistema de paso de mensajes. Se describen asimismo las métricas más comúnmente utilizadas en los ámbitos de investigación y publicación científica.

En el Apartado 2.3 se proponen 4 alternativas de modelado (Modelos 0. . . 3) y se muestra su creciente capacidad predictiva. Los parámetros extraidos se utilizan para comparar entre sí los distintos modos de empaquetamiento y encaminamiento de PVM, así como las opciones homólogas bajo LAM/MPI (−O −c2c −lamd). Siendo el objetivo del presente capítulo evaluar las prestaciones del sistema de paso de mensajes en nuestro cluster, en este apartado no se utilizan las *Toolboxes* PVMTB y MPITB, sino PVM o LAM/MPI llamados directamente desde lenguaje C.

Aunque los programas de medición están naturalmente redactados en lenguaje C, el entorno MATLAB se ha utilizado intensivamente para visualizar y estudiar los datos obtenidos, desarrollar y afinar los distintos modelos, y automatizar el proceso de extracción de parámetros. El Apéndice B explica con mayor detalle las funciones MATLAB utilizadas para evaluar los distintos modelos.

2.1 Equipo utilizado

El equipo utilizado, *oxígeno*, es un *"cluster Beowulf"*, consistente básicamente en una serie de computadores personales PCs conectados mediante un *switch* Fast Ethernet, a los que se les instaló el Sistema Operativo Linux y los sistemas de paso de mensajes PVM y MPI. Se puede obtener información más detallada sobre el diseño e instalación de un cluster Beowulf, así como de la filosofía y objetivos perseguidos por el Proyecto Beowulf, en su página Web ([5]).

El Proyecto de Documentación Linux también mantiene tradicionalmente HOWTOs (información de "cómo hacer para…") sobre Beowulf (*Beowulf HOWTO* [6]), y sobre las distintas formas de procesamiento paralelo disponibles bajo Linux, (*Parallel Processing HOWTO* [17]). Las cuatro posibilidades de paralelismo mencionadas en esta última HOWTO son:

SMP: Symmetric MultiProcessing (Multiprocesamiento Simétrico), se refiere a computadores con varias CPUs entre las que el Sistema Operativo puede repartir la carga del sistema. El kernel Linux puede ser recompilado para soportar CPUs conforme a la MPS (MultiProcessing Specification) de Intel. El programador puede recurrir al modelo de *threads* (hebras) o a las llamadas de memoria compartida System V IPC (InterProcess Communication) para programar este tipo de sistemas.

Clusters: Un conjunto de computadores conectados por una red dedicada de altas prestaciones puede ser programado mediante un paradigma de programación en memoria distribuida, como por ejemplo el paso de mensajes.

MMX: Las operaciones multimedia, al operar con múltiples datos en paralelo, pueden ser interpretadas como operaciones vectoriales (SWAR: SIMD Within a Register).

Procesador paralelo: Muchos procesadores de señal, como la familia TMS320 de Texas Instruments o la SHARC DSP de Analog Devices, están diseñados para permitir intercomunicar varios de ellos formando un computador paralelo alojado en un PC anfitrión (*host*).

Como argumento a favor de los clusters Beowulf, se suele citar la rápida y fácil disponibilidad de componentes, y su alta relación prestaciones/precio.

El cluster *oxígeno* (Figura 2.1), además del servidor, consta de 8 Pentium II *Deschutes* a 333MHz con 512KB de cache y 128MB de RAM. El subsistema de memoria se conecta mediante un bus de 64 bits a 66MHz. El computador servidor funciona a 400MHz y su bus de memoria a 100MHz. Cada uno está equipado con su tarjeta de red y una BootPROM de arranque. Las placas madre son Intel SE440BX con bus PCI de 32 bits a 33MHz.

La tarjeta de red (NIC*) es una 3Com 3c905 *Boomerang* con 8Kpalabras de RAM (partidas en 3K:5K recepción:transmisión) y transceptor MII 10/100BaseT con autoselección y con capacidad *bus-master* para el DMA. La capacidad de autoselección permitiría conectar la tarjeta de red a un *switch* 10BaseT o 100BaseT, detectando el transceptor el tipo de enlace y adaptándose al mismo (10 o 100Mbps). La modalidad *full-dúplex*, sin embargo, sólo se activa bajo 100BaseT. En cualquier caso, las conexiones del cluster son dedicadas y la posibilidad de funcionar a 10Mbps no se ejercita. La capacidad *bus-master* permite a la tarjeta convertirse en maestro del bus PCI para acceder al bloque de memoria a transmitir o recibir, liberando a la CPU y/o al DMA de la placa madre de dicha tarea.

La BootPROM insertada en el correspondiente zócalo de la 3c905 se programó mediante un programador EPROM universal (Sunshine EXPRO-80), usando el código ofrecido con el paquete NetBoot [69]. Este sistema permite al computador ejecutar el código contenido en la BootPROM para realizar el *bootstrap* del Sistema Operativo en ausencia de otros medios de arranque como diskettes, discos, CD-ROM, etc. La EPROM de arranque utiliza el protocolo BOOTP para obtener del servidor la información sobre su dirección IP, subdirectorio raíz y fichero de imagen a cargar y ejecutar, el protocolo TFTP para obtener el fichero de imagen, y el protocolo NFS para acceder a su subdirectorio raíz en el servidor.

Figura 2.1: Fotografía del cluster utilizado para esta memoria, "oxígeno". Consta de 8 clientes Pentium II (ox5–8 apilados, ox1–4 en la base) y un servidor de disco (ox0) con ratón, teclado y monitor, accesible desde el exterior (a la derecha, con el calendario de las copias de seguridad que se ven encima). El monitor apilado está conectado a ox5, encima del cual se ve el switch BayStack 350T.

Bajo Linux, esta información se proporciona mediante el fichero /etc/bootptab, que en el servidor *oxígeno* contiene:

```
.default:\
        :sm=255.255.255.0:\
        :ds=192.168.1.9:\
        :gw=192.168.1.9:\
        :ht=ethernet:\
        :dl=0xFFFFFFFF:\
        :vm=auto:\
        :hd=/tftpboot:\
        :bf=bootImage:

    ox1:ha=00600846464C:ip=192.168.1.1:tc=.default:rp=/export/root/ox1:
    ox2:ha=0060084645A6:ip=192.168.1.2:tc=.default:rp=/export/root/ox2:
    ox3:ha=0060084645A4:ip=192.168.1.3:tc=.default:rp=/export/root/ox3:
    ox4:ha=00600846464F:ip=192.168.1.4:tc=.default:rp=/export/root/ox4:
    ox5:ha=006008138AB0:ip=192.168.1.5:tc=.default:rp=/export/root/ox5:
    ox6:ha=00600820702D:ip=192.168.1.6:tc=.default:rp=/export/root/ox6:
    ox7:ha=006008464671:ip=192.168.1.7:tc=.default:rp=/export/root/ox7:
    ox8:ha=006008464594:ip=192.168.1.8:tc=.default:rp=/export/root/ox8:

    bay:ha=0000A269D090:ip=192.168.1.250:tc=.default:
```

Listado 2.1: Fichero de configuración del protocolo BOOTP /etc/bootptab.

Cada uno de los 8 clientes del cluster dispone de una conexión dedicada a un puerto del *switch* BayStack 350T, de 16 puertos. El *switch* permite establecer cualesquiera interconexiones simultáneas entre sus puertos siempre que no coincida el receptor. Tiene capacidad de autonegociación para conmutar entre 10/100Mbps, capacidad de aprendizaje de las direcciones Ethernet conectadas a cada puerto (al objeto de encaminar la información en lugar de repetirla, *router vs. repeater*), y capacidad de separar puertos en dominios de colisión separados, pudiendo conectarse en cascada con otros *switches*.

Otro puerto del BayStack se dedica a conectar el servidor, un Pentium II a 400MHz, con 512KB de cache, 128MB de RAM y 14GB de disco. De las dos tarjetas 3Com 3c905 de que dispone, una se dedica a la mencionada conexión con el *switch* y el resto del cluster, y otra se conecta a 10Mbps con el tramo Internet de la Universidad, permitiendo acceso remoto al cluster.

Tal y como recomienda la Beowulf HOWTO [6], sólo el servidor es visible desde el exterior, quedando el resto del cluster oculto mediante el uso de la dirección privada 192.168.0.0 (RFC-1918). Esto impide la propagación de tráfico desde la red compartida hacia la red privada, garantizando la disponibilidad del ancho de banda para las aplicaciones del cluster. Bajo Linux, esta información se proporciona mediante el fichero /etc/hosts, que en el servidor *oxígeno* contiene:

```
127.0.0.1       localhost.localdomain   localhost   loghost
150.214.60.67   oxigeno.ugr.es   oxigeno
192.168.1.9     ox0
192.168.1.1     ox1
192.168.1.2     ox2
192.168.1.3     ox3
192.168.1.4     ox4
192.168.1.5     ox5
192.168.1.6     ox6
192.168.1.7     ox7
192.168.1.8     ox8
192.168.1.250   bay
```

Listado 2.2: Fichero de direcciones IP /etc/hosts.

El servidor también dispone de teclado, ratón y monitor, permitiendo usarlo como consola del sistema. Sólo uno de los computadores sin disco (*diskless*) tiene tarjeta de video y monitor, lo que permite obtener información y mensajes de error sobre los posibles problemas de arranque o funcionamiento del resto de computadores sin monitor (*headless*), ya que sus configuraciones son idénticas (salvo video). También es común calificar de *headed* al servidor, ya que suele ser el único en disponer de monitor ("head", cabeza).

Otra HOWTO del LDP (Linux Documentation Project), la DiskLess HOWTO [18], proporciona instrucciones detalladas para configurar el arranque de computadores sin disco. El paquete utilizado, NetBoot, se menciona como una de las alternativas.

En el servidor se recompiló el kernel Linux para los clientes, incluyendo soporte TCP/IP, cliente NFS, y la opción de "subdirectorio raíz / sobre NFS" . Estas capacidades del S.O. son requeridas durante el arranque de los computadores clientes, de manera que no pueden configurarse como módulos del kernel. Lejos de ser opciones anecdóticas del kernel, estas capacidades son frecuentemente usadas, como queda de manifiesto por la existencia de sus correspondientes HOWTO's [18, 71].

Utilizando entonces la utilidad NetBoot mknbi−linux sobre el kernel recompilado, se genera la imagen /tftpboot/bootImage mencionada en el fichero /etc/bootptab (Listado 2.1). Al arrancar un cliente sin disco, la EPROM de arranque hace una petición BOOTP atendida por el proceso bootpd en el servidor. Los servicios bootpd y tftpd se activan en el fichero de configuración /etc/inetd.conf del servidor, que incluye las líneas:

```
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers." Do not uncomment
# this unless you *need* it.
#
tftp     dgram   udp    wait    root    /usr/sbin/tcpd   in.tftpd
bootps   dgram   udp    wait    root    /usr/sbin/tcpd   bootpd
```

Listado 2.3: Fichero de servicios dependientes del *super-daemon* inetd.

La EPROM de arranque consulta la dirección Ethernet de la tarjeta y emite una petición BOOTP etiquetada con dicha dirección. El servidor BOOTP identifica la línea de /etc/bootptab correspondiente (campo **ha**, *hardware address*) y responde la información asociada a dicho computador, incluyendo la imagen a cargar y el subdirectorio raíz (campos **rp/hd/bf**, *root partition*, *home dir*, *boot file*).

Con esta información, la EPROM de arranque puede emitir una petición TFTP para cargar la imagen y ejecutarla (con lo cual arranca el sistema operativo), y pasarle a la imagen parámetros suficientes como para que ésta construya una petición NFS para montar el subdirectorio raíz. Dado que el subdirectorio raíz se monta durante el arranque, no puede accederse a los programas clientes NFS (están en una partición NFS), por lo cual el kernel requiere soporte para partición raíz sobre NFS.

La partición raíz de los clientes *diskless* debe contener todo lo necesario para arrancar hasta el punto en que se monta el subdirectorio /usr, momento en el cual todo el software del servidor está disponible a los clientes. Típicamente, esto exige determinados programas en /bin, /sbin y /lib, ficheros de dispositivo en /dev, ficheros de configuración en /etc, etc.

2.2 Eficiencia de un sistema de paso de mensajes

El modelo más básico para el estudio de prestaciones de un sistema de paso de mensajes es una ecuación afín $T = L + S/B$ [7, 81], que predice el tiempo de transmisión T para un tamaño de mensaje dado S. El estudio del sistema se reduce a la determinación y discusión de los dos parámetros del modelo, la latencia L y el ancho de banda B. El primer parámetro representa una cantidad de tiempo mínima gastada al transmitir cualquier mensaje, independientemente de su tamaño. El segundo parámetro modela un incremento lineal del tiempo de transmisión respecto al tamaño del mensaje.

En un entorno habitual de red de área local (LAN, *Local Area Network*), en el cual los computadores compiten por el ancho de banda del sistema provocando colisiones de datagramas en el medio de transmisión, este modelo es de dudosa reproducibilidad. Los parámetros dependen fuertemente del tráfico LAN en el momento de la medición. Incluso en un entorno controlado como un cluster, los tiempos de transmisión pueden verse afectados por tareas iniciadas o mantenidas por el propio sistema, como muestra la Figura 2.2.

En un entorno de computadores en cluster, algunos de estos fenómenos "aleatorios" o irreproducibles pueden ser controlados. Los usuarios pueden establecer un horario de reservas para la realización de mediciones de tiempo precisas, durante el cual el cluster puede estar desconectado del mundo exterior, dedicado únicamente a ejecutar la medición deseada.

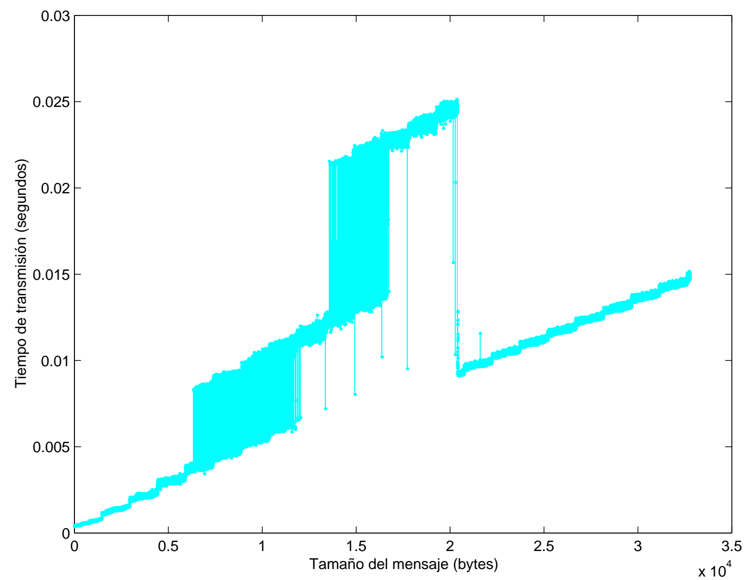
El medio de transmisión, típicamente un *switch*, se dedica únicamente a interconectar los computadores del cluster, proporcionando conectividad *todos-a-todos*, de manera que la única colisión posible consiste en dos fuentes intentando transmitir al mismo destino.

Los mecanismos de ahorro de energía de la placa madre pueden ser desactivados mediante el *CMOS-setup* o ajustados a un modo compatible con el método de medición planeado. Algunos de estos mecanismos están típicamente basados en el tiempo transcurrido sin actividad en los respectivos periféricos (disco duro, consola, modem, tarjeta de red), y sólo afectan a dichos periféricos. Otros, basados en un sensor de temperatura, intentan controlarla reduciendo la frecuencia de reloj del sistema, afectando por tanto a todo el computador.

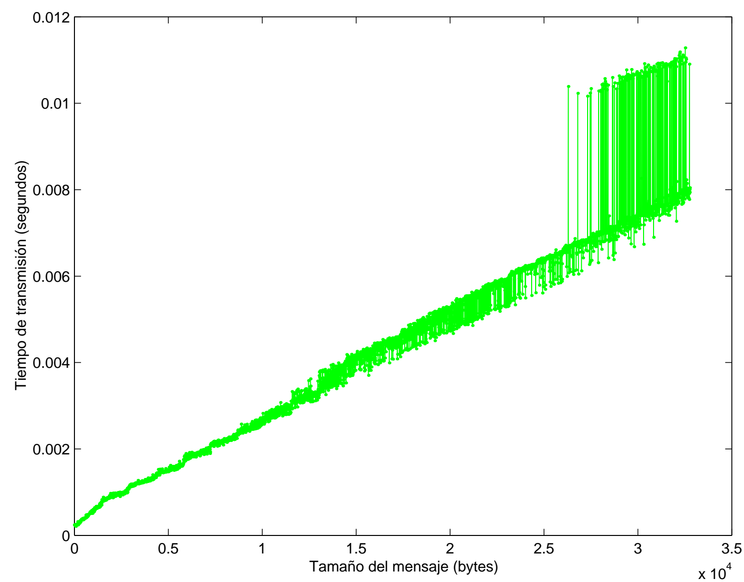
El sistema operativo puede ser afinado para prestar los servicios mínimos compatibles con la ejecución de la medición: entorno multiusuario, login remoto con rsh, servidor NFS. . . Otros servicios, como el entorno de ventanas, servidor de fuentes X, trabajos cron, etc, pueden ser anulados durante la medición. El sistema de niveles de ejecución (*runlevels*) seguido por Linux y otros UNIX SystemV es particularmente adecuado para esta necesidad. Se puede ajustar el nivel 5 para proporcionar la gama completa de servicios, incluyendo login gráfico, por ejemplo, y reservar el nivel 3 para mediciones, excluyendo todos los servicios no imprescindibles. Un simple comando telinit permite conmutar automáticamente desde el entorno "abierto" al entorno de medición.

La latencia y ancho de banda observables entonces resultan del uso conjunto del *switch* con el sistema *software* de paso de mensajes; la latencia es siempre mayor que la del *switch* y el ancho de banda siempre menor, ya que el *software* también consume, tanto un tiempo mínimo equiparable a una latencia (como la requerida por el *switch*), como otro tiempo dependiente del tamaño del mensaje, aunque la dependencia no sea exactamente lineal.

Por ejemplo, con un *switch* de 100Mbps se debería esperar un ancho de banda máximo de $100/8 = 12.5$MB/s, y una latencia mínima de $24/12.5 = 1.92 \simeq 2\mu$s considerando una hipotética



(a) Se pueden obtener alteraciones de diversa magnitud. Obsérvese también que en algunos tramos la alteración es tan regular que podría inducir a pensar que la medición es correcta (si sólo se dispusiera de ese tramo), como por ejemplo la parte final de los escalones mostrados.



(b) La entrada en standby sigue un patrón reconocible.

Figura 2.2: Incluso en un cluster los tiempos pueden ser no reproducibles, si el sistema arranca un trabajo cron o entra en o sale de *standby*, por ejemplo. Según los tests que hemos realizado, sólo los tiempos mínimos son reproducibles.

cabecera de 24 bytes: cada dirección *Ethernet* consume 6 bytes, cada dirección *Internet* consume 4 bytes, y se debe indicar la longitud del datagrama e información de control. Cada protocolo (UDP, TCP, ...) añade información adicional (puertos fuente y destino, índice de secuencia...), de manera que 2μs es una estimación definitivamente optimista.

2.2.1 Obtención de mediciones reproducibles

Siendo posible cuestionar el título de este apartado por una posible redundancia ("*si no es reproducible, no es una medición*"), resulta paradójico comprobar que algunas técnicas frecuentemente utilizadas para determinar el tiempo de transmisión son sólo aproximadamente reproducibles.

Incluso controlando y *declarando* todos los detalles de configuración, el tiempo empleado en transmitir un mensaje presenta una cierta variabilidad. En la referencia [29] se mencionan algunos de los motivos más frecuentemente ignorados, y se caracteriza explícitamente el tiempo de transmisión observable como una cantidad de tiempo mínima reproducible y una serie de perturbaciones *positivas* aleatorias con una distribución desconocida.

También se menciona una posible perturbación *negativa*, la asociada con la resolución finita del reloj. La transmisión de mensajes pequeños puede emplear un tiempo tan reducido que se acerque peligrosamente a la precisión del sistema de medida. En nuestro cluster, usando gettimeofday() en lenguaje C obtenemos una resolución de 3–4μs, mientras que invocando dos veces consecutivas clock en MATLAB se obtiene un lapso de 17–20μs. El problema se puede soslayar cronometrando una serie de transmisiones y dividiendo por el número de repeticiones.

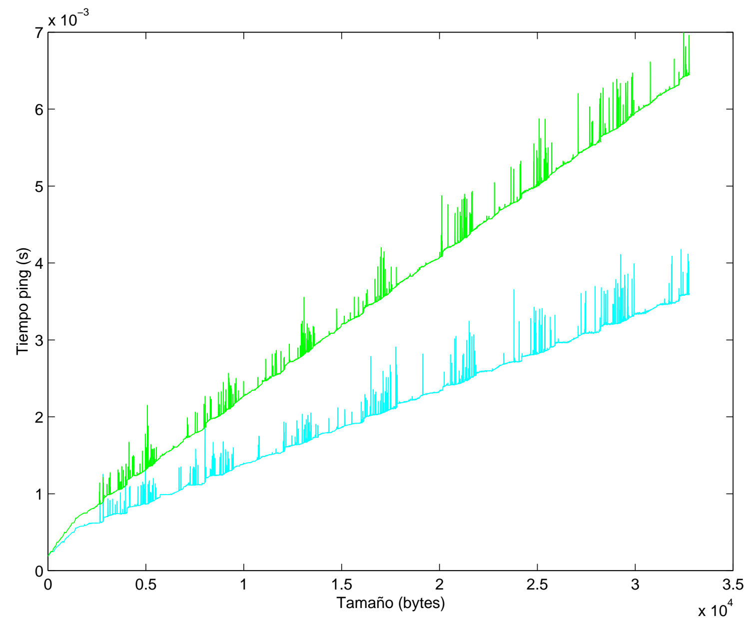
Por ilustrar lo acertado de ésta caracterización, la Figura 2.3(a) muestra los resultados de un test *ping-pong* típico. Un bucle exterior barre el tamaño del mensaje a enviar (abscisas). Otro bucle interno repite el envío y recepción un cierto número de veces, mayor cuanto más pequeño es el tamaño del mensaje. Se ejecutan en sucesión dos versiones del bucle interno, una utilizando mensajes de tipo **double** (color verde) y otra de tipo **char** (azul), de manera que las perturbaciones están correlacionadas en ambos trazos.

La extensión y frecuencia de las perturbaciones varía según el modo de encaminamiento, siendo más frecuentes con ruta TCP directa entre tareas. Sin embargo, la propia impredictibilidad de las mismas garantiza que un tramo de tamaños de mensaje afectado por una perturbación aparecerá sin perturbación en alguna otra ejecución del test. Dado que el tiempo mínimo es el único eventualmente reproducible, es el único que interesa estimar, como se propone en [29].

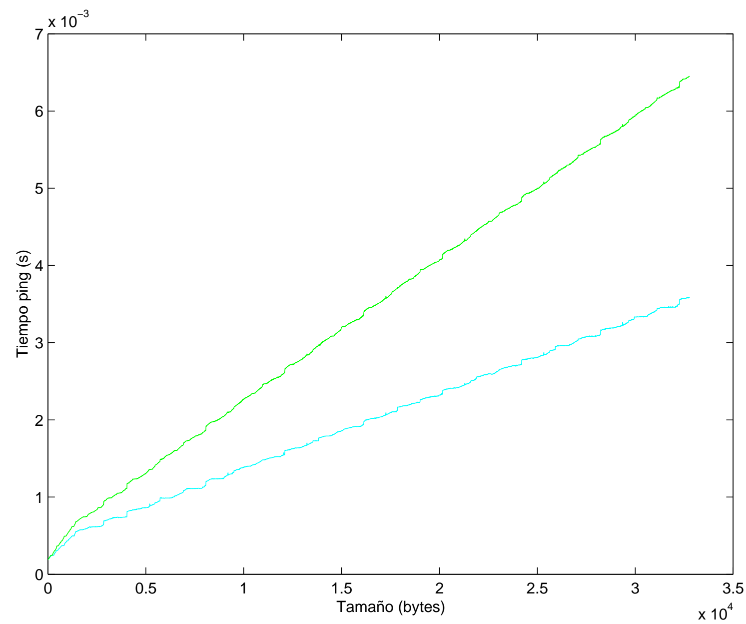
Dicha estimación se muestra en la Figura2.3(b). Repetir el proceso con distintos ficheros de mediciones vuelve a producir el mismo resultado (dentro de la precisión del reloj), lo cual es virtualmente imposible calculando sólo la media del bucle interno en un test, o sólo el mínimo (lo cual requeriría invocar la rutina de medición de tiempo en cada iteración, dentro del bucle), o incluso la media de los mínimos de diferentes tests. Se podrían calcular dichas estimaciones, pero difícilmente se podrían reproducir con precisión cercana a la de la función gettimeofday().

2.2.2 Modelos lineal y afín

En la Figura 2.4 se muestra el tiempo de transmisión para un barrido exponencial sobre el tamaño del mensaje. Superponiendo una recta, la gráfica demuestra que la relación es prácticamente lineal si se consideran tamaños de mensaje lo suficientemente grandes.

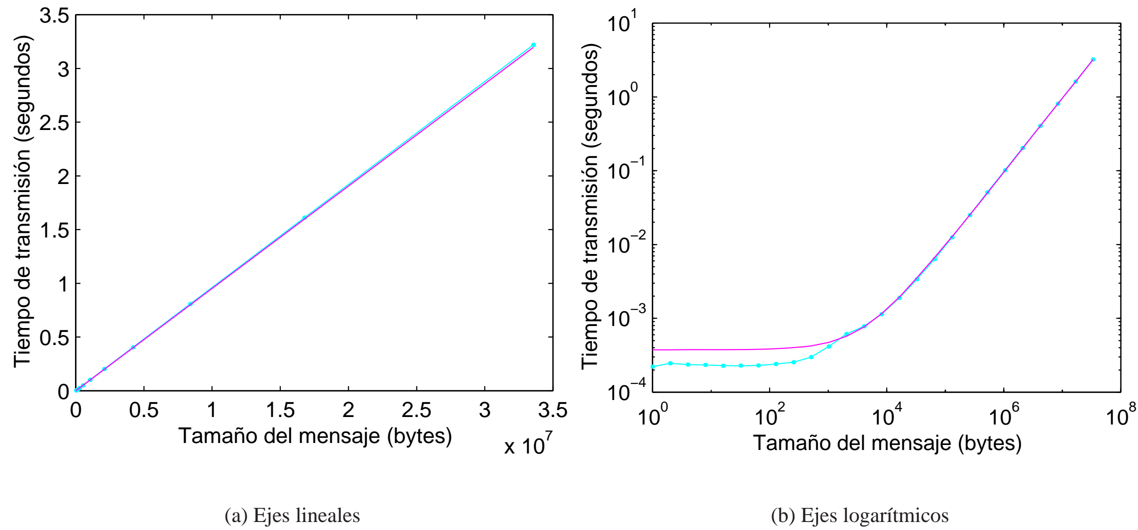


(a) Fichero de resultados, mostrando el tiempo de transmisión medio sobre un número de repeticiones decreciente con el tamaño del mensaje.



(b) Mínimo de medias calculado sobre 10 ficheros de mediciones.

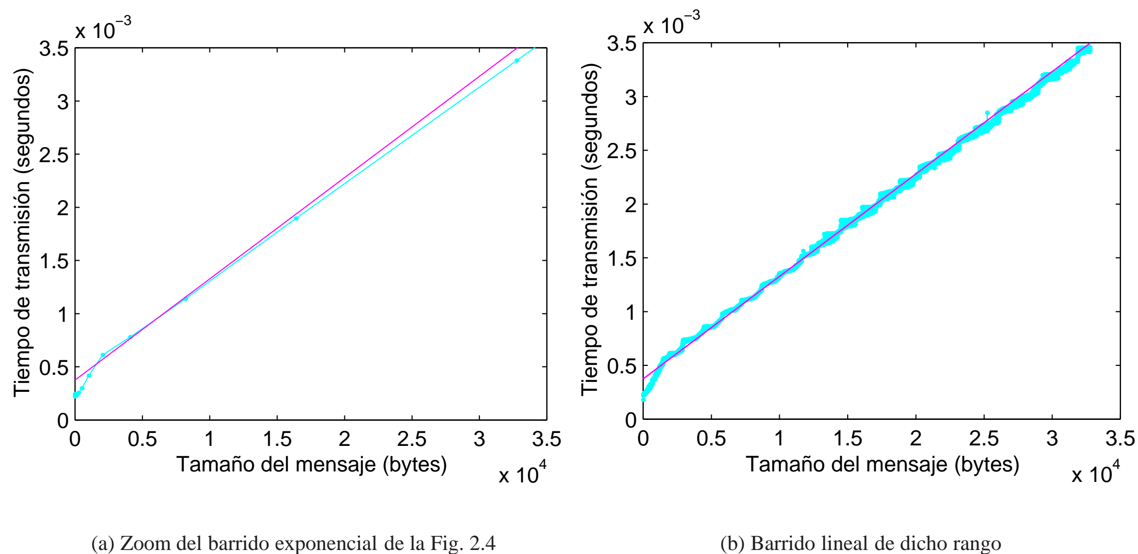
Figura 2.3: Resultados de un test *ping-pong* típico. Al no ser reproducibles las perturbaciones, manifestándose en instantes de tiempo distintos, es fácil eliminarlas cotejando varias ejecuciones del test.



(a) Ejes lineales

(b) Ejes logarítmicos

Figura 2.4: Tiempo de transmisión para tamaños de mensaje en progresión exponencial. Se ha superpuesto en color morado la recta afín $T = 375\mu s + S/10.5MB/s$.



(a) Zoom del barrido exponencial de la Fig. 2.4

(b) Barrido lineal de dicho rango

Figura 2.5: Detalle para tamaños de mensaje menores. Se ha superpuesto la misma recta $T = 375\mu s + S/10.5MB/s$.

La gráfica logarítmica a la derecha permite observar simultáneamente todas las mediciones realizadas sin necesidad de ampliar detalles, e identificar el tamaño de mensaje mínimo por debajo del cual se empieza a perder la linealidad, en este caso del orden de $10^3 \simeq$ 1KB. Se requeriría una ampliación de la gráfica lineal (izq.) para poder apreciar este límite sobre ella.

En la Figura 2.5 se ofrece dicha ampliación, junto con una medición más detallada consistente en un barrido lineal del mismo rango de tamaños. Se puede observar el punto de medición 16KB aislado en la gráfica de la izquierda y la densa agrupación de puntos de medición cada 8 bytes a la derecha. El rápido crecimiento del tiempo de transmisión al inicio del barrido hace que se sobreestime el parámetro L, perdiendo su interpretación física como *"tiempo para transmitir el mensaje vacío"*. Como veremos posteriormente, un ajuste de los parámetros mediante minimización del error cuadrático resulta en una infraestimación del parámetro B, debido también a la fuerte pendiente al inicio de la gráfica.

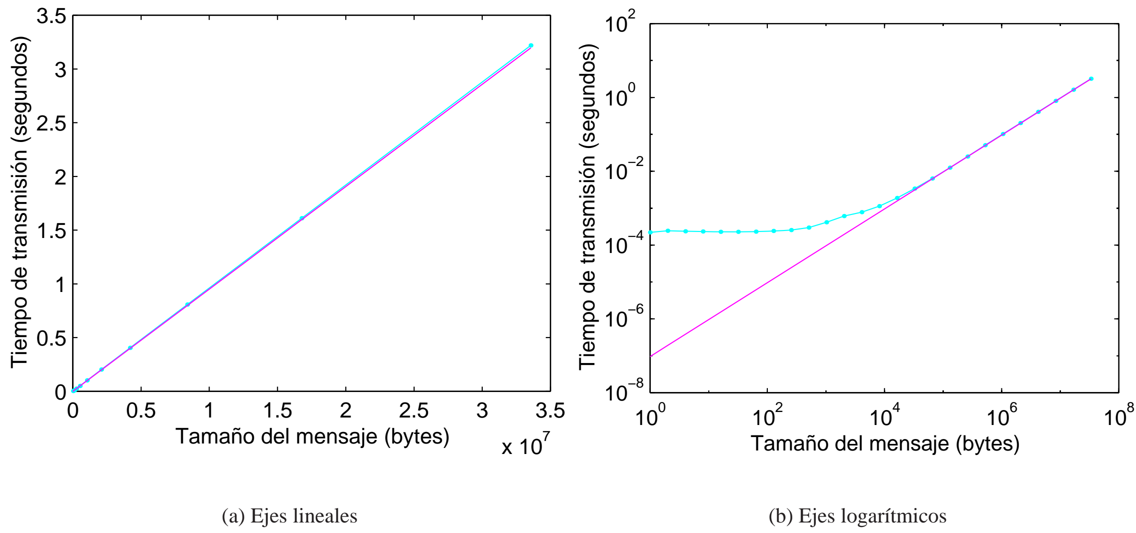
Aunque no suele ser éste el caso usando Fast Ethernet, puede suceder que la latencia sea relativamente pequeña en comparación con el tiempo de transmisión. El tamaño de mensaje utilizado por algunas aplicaciones HPC es tal que, con otros soportes de interconexión (como por ejemplo Myrinet), el término S/B domina claramente en el coste total de comunicación. El modelo se linealiza entonces a $T = S/B$ y el usuario sólo necesita recordar el parámetro B, ancho de banda. Las Figuras 2.6 y 2.7 ilustran esta linealización del modelo anterior.

La gráfica lineal (Figura 2.6(a)) pone de manifiesto que para tamaños de mensaje grandes la diferencia entre ambos modelos no es apreciable. La gráfica logarítmica (Figura 2.6(b)) permite estimar en unos $10^4 \simeq$10KB el tamaño por debajo del cual la diferencia puede apreciarse, resaltando la creciente importancia relativa de ignorar la latencia conforme se disminuye el tamaño del mensaje. Para no invitar a cometer el error intuitivo de conceder excesiva importancia al tramo inicial de la gráfica logarítmica, se muestran las ampliaciones de dicho tramo (Figura 2.7) como también se hizo con el modelo anterior.

Comparando las gráficas logarítmicas para ambos modelos, se concluye que considerar el parámetro L permite predecir el tiempo de transmisión en el orden de 10^3–10^4 bytes, y reducir el error de predicción para tamaños menores. Para tamaños superiores no supone una ventaja significativa. El límite 10^3 proviene del primer tramo de la gráfica (la primera MTU*, 1500 bytes por defecto bajo Linux), y su fuerte pendiente está motivada por la fragmentación del mensaje y las copias adicionales de memoria durante el proceso de transmisión.

Ambos modelos, el lineal $T = S/B$ y el afín $T = L + S/B$, son apropiados para un sistema de transmisión ideal sin copia de memoria (*zero copy*). Por ejemplo, para caracterizar las prestaciones de un *switch*, usualmente sólo se indica el ancho de banda B (10Mbps, 100Mbps...); raramente puede encontrarse la latencia entre las especificaciones técnicas del producto. Tampoco es usual encontrar detalles técnicos acerca del funcionamiento del mismo (método de encaminamiento, interconexionado, etc).

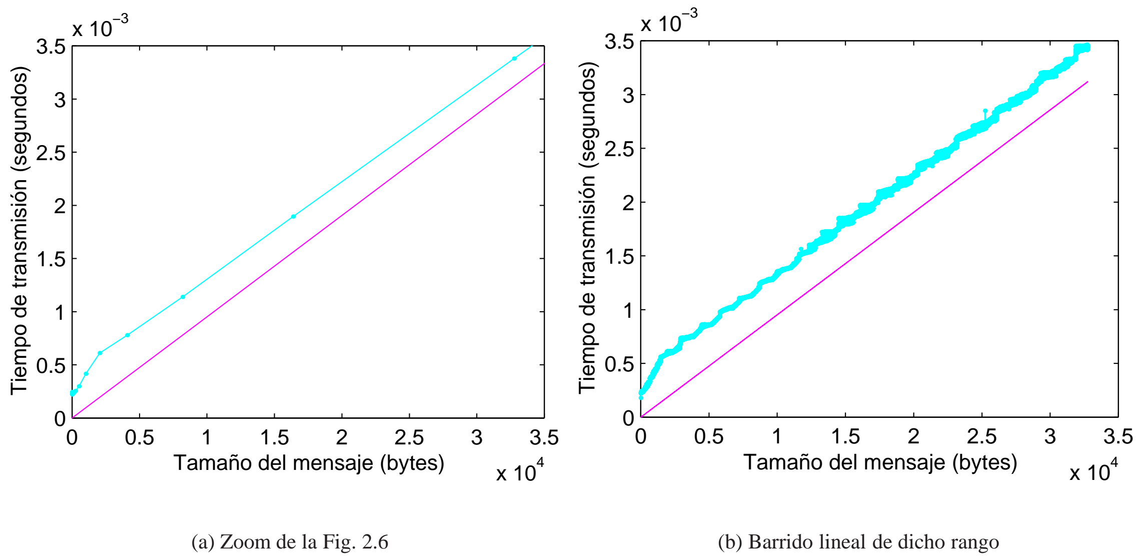
En sistemas completos en los cuales el subsistema de transmisión coopera con un subsistema de almacenamiento que copia los datos del mensaje al adaptador de red, y de allí a la memoria del sistema, la desviación de la linealidad observada para mensajes pequeños se debe fundamentalmente a las diversas copias de memoria implícitas que se realizan sobre el mensaje, a la fragmentación del mismo para su transmisión por un medio orientado a datagramas, y a la imposibilidad de funcionamiento paralelo de los subsistemas de almacenamiento y transmisión para el primer fragmento del mensaje.



(a) Ejes lineales

(b) Ejes logarítmicos

Figura 2.6: Repetición de la Fig. 2.4. Se ha superpuesto en morado la ecuación lineal $T = S/10.5MB/s$.



(a) Zoom de la Fig. 2.6

(b) Barrido lineal de dicho rango

Figura 2.7: Detalle. Se ha superpuesto la misma recta $T = S/10.5MB/s$.

Dado que el parámetro *latencia* tiene una clara interpretación como el tiempo necesario para transmitir el mensaje vacío ($T_0 = L + 0/B = L$), se acepta tácitamente que sea el parámetro B quien absorba las desviaciones, y debe tabularse para distintos tamaños de mensaje, siendo usual calcularlo para cada punto de la medición. Esto equivale a usar varias instancias del modelo, cada una restringida a un punto o un pequeño tramo de tamaños de mensaje durante el cual tienen validez los parámetros. La Figura 2.8 y Tabla 2.1 ilustran esta técnica de tabulación.

También es frecuente linealizar el modelo al usar esta técnica, siendo por tanto difícil predecir correctamente el tiempo de transmisión para tamaños pequeños si no se tabula B para cada punto de la medición. Con ambos modelos, lineal o afín, las estimaciones de ancho de banda obtenidas crecen asintóticamente hacia un máximo que caracteriza el sistema global hardware y software de paso de mensajes. Entre las figuras de mérito asociadas con esta técnica de modelado resulta obligado citar el *"Punto de Potencia Media"* (*Half-Power Point*), definido como el tamaño de mensaje para el cual se obtiene (según el modelo lineal $T = S/B$ usualmente) un ancho de banda mitad del asintótico.

Se pueden consultar en la referencia [11] otros métodos frecuentemente usados para evaluar las prestaciones de comunicación, como el test *ping-pong* usado en esta memoria (*end-to-end*), el test *ping* unilateral (*one-sided*), la función característica de prestaciones (*throughput curve*, como la Figura 2.8), los modelos LogP, BSP, etc.

2.3 Modelos basados en la MTU

Los gestores de dispositivo para adaptadores de red (*drivers*) suelen subdividir la secuencia de datos a transmitir en *paquetes* de tamaño limitado. Este límite se denomina Unidad Máxima de Transferencia (Maximum Transfer Unit, MTU). Los mensajes mayores que 1 MTU se dividen en fragmentos (MTUs) que son individualmente ubicados en memoria y transmitidos, pudiendo ser el último fragmento menor que la MTU.

Al transmitir una única MTU no hay oportunidad de explotar el potencial paralelismo del diverso *hardware* que opera en secuencia sobre ella: CPU/DMA fuente, transceptores de red (*transceivers*) y DMA/CPU destino. Cada subsistema debe esperar a que llegue la MTU completa antes de realizar su cometido sobre ella.

Al transmitir varias MTUs, mientras el adaptador de red transmite una la CPU fuente puede preparar la siguiente, posiblemente copiando de los buffers del S.O. al buffer de transmisión de la tarjeta mediante *Bus-Master* DMA. También simultáneamente, salvo para el caso de la primera MTU, la CPU destino puede copiar a buffers del S.O. y a memoria de usuario una MTU anteriormente recibida en los buffers de su tarjeta de red. El hardware implicado puede por tanto trabajar en paralelo, de forma similar al paralelismo obtenido por segmentación de cauce (*pipeline*) en una CPU.

Dicho paralelismo es más apreciable en los modos de transmisión que realizan más copias de memoria, observándose una fuerte pendiente en el primer tramo de la característica de prestaciones (como en la Figura 2.8). Otros modos realizan menos copias intermedias del mensaje, resultando en un primer tramo de pendiente más similar al resto de la característica. Conforme un sistema realiza menor número de copias de memoria ([1, 25, 92]) presenta una ancho de banda cada vez más próximo al del *switch*.

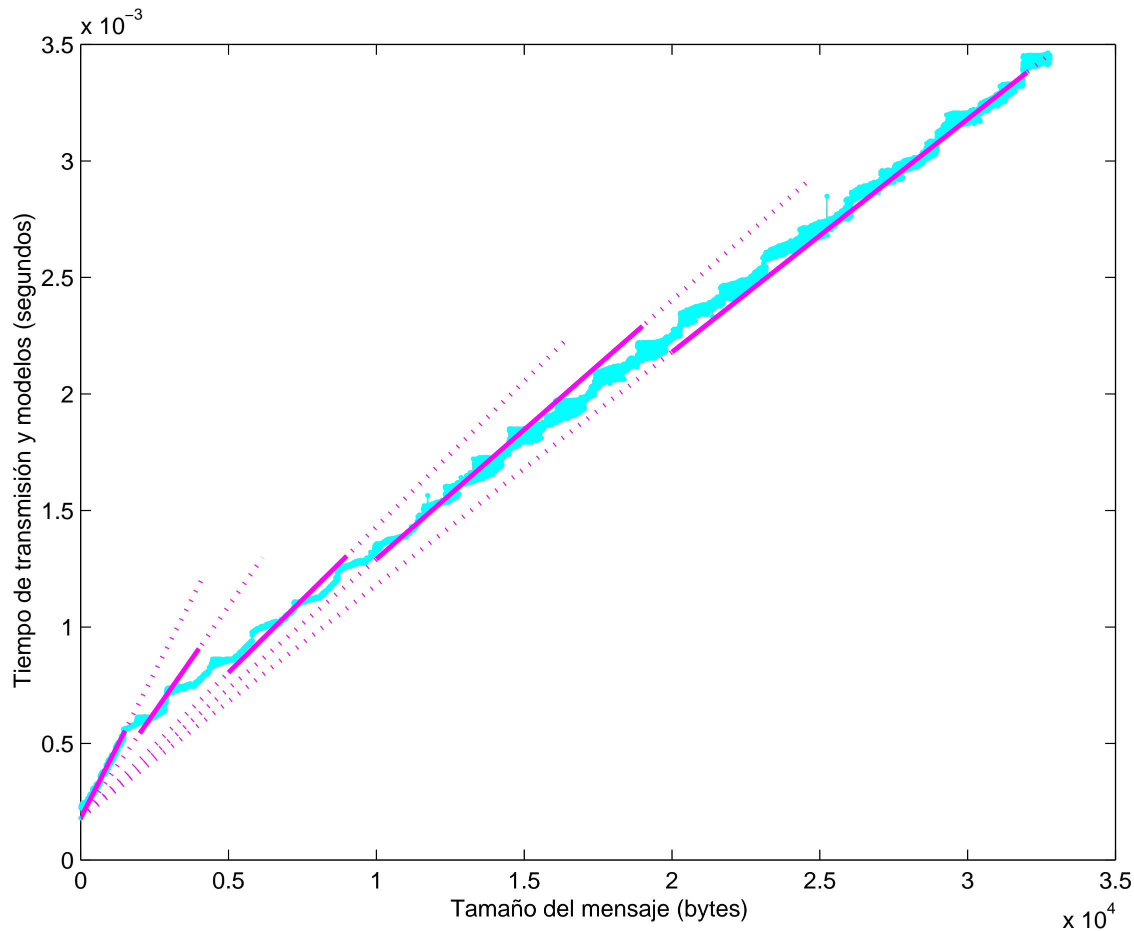


Figura 2.8: Tramos de validez. El parámetro B absorbe las desviaciones de la linealidad.

L	180μs		
B	valor	desde S	hasta S
	4.0 MB/s	0 bytes	1500 bytes
	5.5 MB/s	2000 bytes	4000 bytes
	8.0 MB/s	5000 bytes	9000 bytes
	9.0 MB/s	10000 bytes	19000 bytes
	10.0 MB/s	20000 bytes	32000 bytes

Tabla 2.1: Parámetros del modelo $T = L + S/B$. En el modelado a tramos, la latencia se obtiene por medición directa del tiempo de transmisión del mensaje nulo. También es habitual forzar $L = 0$, dificultándose el ajuste del modelo (lineal) al tramo inicial de las mediciones. Este problema no se presenta si se calcula B para cada tamaño de mensaje. Por economía se ha preferido reducir la tabla a 5 tramos en lugar de proporcionar la lista completa de anchos de banda.

Adicionalmente a la fragmentación en MTUs, tanto PVM como LAM/MPI definen un sistema de fragmentación propio de tamaño superior a la MTU (UDPMAXLEN, MAXNMSGLEN) como muestran los listados 2.4 y 2.5. Estos sistemas de fragmentación son usados por los respectivos *daemons* PVM y LAM. El efecto de esta fragmentación adicional sobre las prestaciones se manifiesta al utilizar el encaminamiento a través del *daemon*.

```
/* UDPMAXLEN should be set to the largest UDP message length your system can handle. */

#define KEEPONPAGE 16

#ifndef UDPMAXLEN
#define UDPMAXLEN (4096 - KEEPONPAGE)          /* generic max fragment length */
#endif

/* Message header */

#define MSGHDRLEN        32
```

Listado 2.4: Fragmento del fichero $PVM_ROOT/src/global.h.

```
/*
 * Revision 5.2.1.1  94/08/25  15:23:38  vaigl-j
 * Increase MAXNMSGLEN to 8K.
 */

/*
 * network constants
 */
#define MAXNMSGLEN      8192            /* maximum network frame size */
#define NOTNODEID       0x80000000      /* not a valid node ID */
#define NOTLINKID       0x80000000      /* not a valid logical link ID */
```

Listado 2.5: Fragmento del fichero $LAMHOME/include/net.h.

PVM dispone también de un símbolo **#define** CLUMP_ALLOC, mediante el cual se puede alterar la estrategia de reserva de memoria de PVM. El Listado 2.6 muestra que esta opción está incluida entre las opciones por defecto usadas en la compilación de PVM.

```
#
# $Id: Makefile.aimk,v 1.34 1998/10/01 21:10:25 pvmsrc Exp $
#
# Generic Makefile body to be concatenated to config header.
#
# Imports:
#   PVM_ARCH     = the official pvm-name of your processor
...
# Compatibility defines (usually in conf/*.def):
#   UDPMAXLEN=        for alternate max udp packet size
...
# Options defines:
#   CLUMP_ALLOC       allocates several data structures in big chunks
#   MCHECKSUM         to enable crc checksums on messages
#   RSHNPLL=          for number of parallel rsh startups (default is 5)
#   RSHTIMEOUT=       for rsh timeout other than default (60 sec)
#   STATISTICS        to enable collection of statistics in pvmd
#   TIMESTAMPLOG      to enable timestamps in pvmd log file
#   USE_PVM_ALLOC     to enable instrumented malloc functs (for pvm debug)
#
...
OPTIONS          =        -DCLUMP_ALLOC -DSTATISTICS -DTIMESTAMPLOG -DSANITY
```

Listado 2.6: Fragmento del fichero $PVM_ROOT/src/Makefile.

El tamaño de MTU típico para los adaptadores Ethernet es de 1500 bytes. Esta fragmentación se manifiesta en todas las modalidades de transmisión. Los modelos presentados en este Apartado han sido desarrollados basándose en dos configuraciones distintas de PVM y LAM/MPI:

- Estándar PVM: UDPMAXLEN==4K–16, MAXNMSGLEN==4440 para LAM.
- Estándar LAM: UDPMAXLEN==8K para PVM, MAXNMSGLEN==8K.

Pruebas preliminares y consultas con los autores de LAM/MPI (ver la pregunta msg02059.php y la respuesta msg02060.php en http://www.lam-mpi.org/MailArchives/lam/, el archivo de la lista de distribución LAM) recomendaron no rebajar de 4436 el tamaño de fragmentación del *daemon* LAM, dado que éste emite mensajes de petición de dicho tamaño que conviene que no sean fragmentados.

El tiempo de transmisión se mide mediante un test *ping-pong* programado en C con llamadas a la respectiva biblioteca, PVM o LAM/MPI. El test se repite un cierto número de veces (concretamente 10) generándose en cada repetición un fichero de mediciones en el que quedan anotados los tamaños de array transmitidos y el tiempo medio de transmisión, tanto para los arrays de tipo **char** como para los de tipo **double**.

Como ya se mostró en el Apartado 2.2.1, sólo los tiempos mínimos de una serie de mediciones son reproducibles, mientras que los tiempos máximos caracterizan el tipo de dificultades que ha sufrido el sistema de paso de mensajes durante la medición —tráfico de red, colisiones, trabajos cron u otras cargas en los computadores fuente y destino, entrada a ó salida de standby... dificultades que no son el objeto de nuestras mediciones. Siguiendo lo propuesto en [29] para obtener medidas reproducibles, el tiempo de transmisión para cada tamaño de mensaje se estima como el mínimo de las 10 medias correspondientes.

Para ejercitar las diversas opciones del sistema PVM, se repitieron los tests bajo los tres modos de empaquetamiento, *PvmDataDefault*, *PvmDataRaw* y *PvmDataInplace*, combinándolos o no con la opción de encaminamiento directo, *PvmRouteDirect*. Se probaron por tanto 6 combinaciones de opciones PVM. Bajo LAM/MPI se pueden activar o no las opciones de homogeneidad –O y ruta directa –c2c, resultando en 4 combinaciones de opciones.

Para sistematizar el proceso de medición y facilitar la identificación de mediciones concretas, los ficheros de mediciones, tanto los generados bajo PVM como los de LAM, siguen la nomenclatura <prg>–<src>–<dst>–<pck>–<rou>–<num>.dat, en donde:

prg nombre del programa que generó el fichero, en este caso MTU.

src-dst ordenadores usados para el test *ping-pong*.

pck abreviatura de la opción de empaquetamiento PVM (def, raw, pla) o de homogeneidad LAM (O, _) utilizada. El subrayado indica la ausencia de la opción LAM.

rou abreviatura de la opción de encaminamiento usada: di/no bajo PVM, c2c/___ bajo LAM.

num número de secuencia, del 0 al 9. Los mínimos de las 10 medias se salvan en otro fichero, etiquetado con la abreviatura *min* en lugar de número de secuencia.

Por ejemplo, la Figura 2.3 se obtuvo del fichero de medición MTU–ox0–ox1–def–di–0.dat y del fichero de mínimos MTU–ox0–ox1–def–di–min.mat. El fichero de mínimos se genera desde MATLAB, lo cual explica su extensión.

Los ficheros se clasifican en directorios nombrados <prg>graph–<udp>–<max>, en donde:

udp abreviatura de la configuración PVM UDPMAXLEN (udp4K, udp8K) o de la configuración LAM MAXNMSGLEN (maxn4K, maxn8K) utilizada.

max tamaño de mensaje al que llega el barrido. Aunque se contemplaron varias posibilidades durante el estudio, en la exposición final mostraremos los resultados sobre un barrido lineal de 32KB, identificado por la abreviatura lin32K.

Por ejemplo, los ficheros de la Figura 2.3 se hallan en un subdirectorio MTUgraph–udp4K–lin32K.

En el barrido de tamaños se escogió un incremento de 8 para poder realizar todas las iteraciones con ambos tipos de datos. Se utilizan distintos tipos de datos para revelar el distinto costo adicional de la codificación XDR, siendo **sizeof**(**char**)==1 y **sizeof**(**double**)==8 bajo la arquitectura y compilador escogidos.

El tamaño máximo de 32KB es un compromiso entre el ancho de banda alcanzado y la facilidad de visualización. Interesaría en principio obtener gráficas que abarcaran no sólo varias MTUs sino varios fragmentos UDP, para observar el efecto de los distintos mecanismos de fragmentación en las prestaciones del paso de mensajes. Teniendo algunas configuraciones un tamaño UDP de UDPMAXLEN=MAXNMSGLEN=8KB, parece razonable llegar a 3 o 4 fragmentos UDP. Con 32KB no se alcanza el máximo ancho de banda, pero aumentar el número de fragmentos dificulta la visualización de la característica de prestaciones, volviéndose indetectable la mayor pendiente de la primera MTU.

El número de repeticiones del bucle *ping-pong* (bucle interno) para cada tamaño de mensaje barrido por el bucle más externo del test, sigue una función decreciente con el tamaño, por diversos motivos:

- Se barren 4 órdenes de magnitud en el tamaño del mensaje.
- 100Mbps implica tiempos de milisegundos para transmitir decenas de KBs, e inferior a la latencia (centenares de μs) para decenas de bytes, acercándose a la resolución de la rutina de medición usada, gettimeofday().
- La variabilidad de las mediciones aumenta sólo muy gradualmente con el tamaño del mensaje transmitido, por lo cual va perdiendo importancia relativa en comparación al propio tiempo de transmisión.

No parece razonable estimar el tiempo de transmisión de 32KB con la misma precisión de μs empleada para estimar la latencia. Para tamaños pequeños, repetir la transmisión numerosas veces no alarga demasiado el tiempo global del test, mejora la precisión del resultado al dividirse el lapso medido entre el número de repeticiones, y reparte entre las múltiples transmisiones el coste de inicialización del bucle y de ejecución de la propia llamada gettimeofday().

En concreto, la función escogida asigna 256 repeticiones al mensaje nulo y una única repetición para transmitir 32MB. El tiempo de transmisión de 32MB es del orden de segundos, siendo en comparación insignificantes la resolución del reloj, los costes de inicialización del bucle y de llamada a gettimeofday(), y la propia magnitud de las perturbaciones (alrededor de 0.6ms). Además, cada repetición alargaría en segundos (ida y vuelta) el tiempo global del test, que debe repetirse 10 veces (10 ficheros de medias) para cada combinación de opciones (6 de PVM, 4 de LAM) con cada configuración (2 de PVM, 2 de LAM).

Entre ambos extremos, 256 repeticiones para el mensaje nulo y 1 para 32MB, se sigue un decaimiento exponencial con el tamaño del mensaje que asigna 8 repeticiones para un tamaño de 32KB. El tiempo de transmisión para 32KB es del orden de milisegundos.

Un *script* bash de automatización que barre las opciones de empaquetamiento PVM u homogeneidad LAM, encaminamiento y número de secuencia, facilita la generación de los 10 ficheros de medias de cada combinación de opciones con menor esfuerzo y propensión a errores. El *script* admite como argumentos la configuración PVM o LAM a utilizar y el tamaño del barrido deseado, usando la versión correcta del correspondiente software de paso de mensajes.

Con todas estas elecciones de tamaño máximo, paso de barrido, número de repeticiones de cada transmisión y número de ficheros generados para cada combinación de opciones, un test típico de una configuración PVM puede llegar a tardar unas 6 horas aproximadamente, debiéndose repetir para ambas configuraciones UDPMAXLEN. Los tests LAM tardan algo menos de 4 horas, al haber sólo 4 combinaciones de opciones. Dejando el cluster desconectado del exterior a las 22:00 y tecleando las órdenes para ejecutar en secuencia el *script* de automatización sobre una configuración PVM y otra LAM, es posible volver a las 8:00 y generar acto seguido los ficheros de mínimos utilizando MATLAB. Otra noche bastaría para completar la generación de ficheros de medición.

Como ya se comentó, los test fueron ejecutados en un cluster de 8 Pentium II 333MHz, 128MB RAM, con servidor de disco Pentium II 400MHz, 128MB RAM, 14GB disco, interconectados mediante un conmutador (*switch*) de 100Mbps BayStack 350T, con 16 puertos.

En el servidor de disco se inicia el programa de medición (MTU, ya mencionado), y en otro computador del cluster el programa de eco o respuesta. Los argumentos del programa indican la opción de empaquetamiento u homogeneidad y de encaminamiento, y el rango y paso del barrido.

El *script* de automatización consta de tres bucles anidados en los que se barren el número de ficheros deseados (10), las opciones de encaminamiento (*PvmRouteDirect/DontRoute* para PVM, −O/_ para LAM) y las de empaquetamiento u homogeneidad (*PvmDataDefault/DataRaw/InPlace* para PVM, −c2c/−lamd para LAM).

2.3.1 Interfaz gráfico

Evaluar el modelo para todas las combinaciones de opciones (6 en PVM, 4 en LAM) bajo las dos configuraciones sugeridas de UDPMAXLEN–MAXNMSGLEN puede resultar tedioso y es propenso a errores. Además, durante la etapa de estudio los modelos han ido variando hasta llegar a su forma definitiva, en un proceso iterativo en el que el estudio de los valores de parámetros extraidos y del error del modelo conduce a modificaciones del propio código del modelo para obtener un mejor ajuste.

En este caso, no interesa desarrollar un *script* de automatización, sino un interfaz gráfico (*GUI*) que permita iterar rápidamente el ciclo *extracción* de parámetros–*modificación* del modelo. En la Figura 2.9 se muestra el aspecto de la herramienta GUI desarrollada a tal efecto en MATLAB, que permite:

- seleccionar el directorio y fichero concretos de la medición (columna izquierda)
- observar gráficamente las mediciones y el modelo (gráfica central)

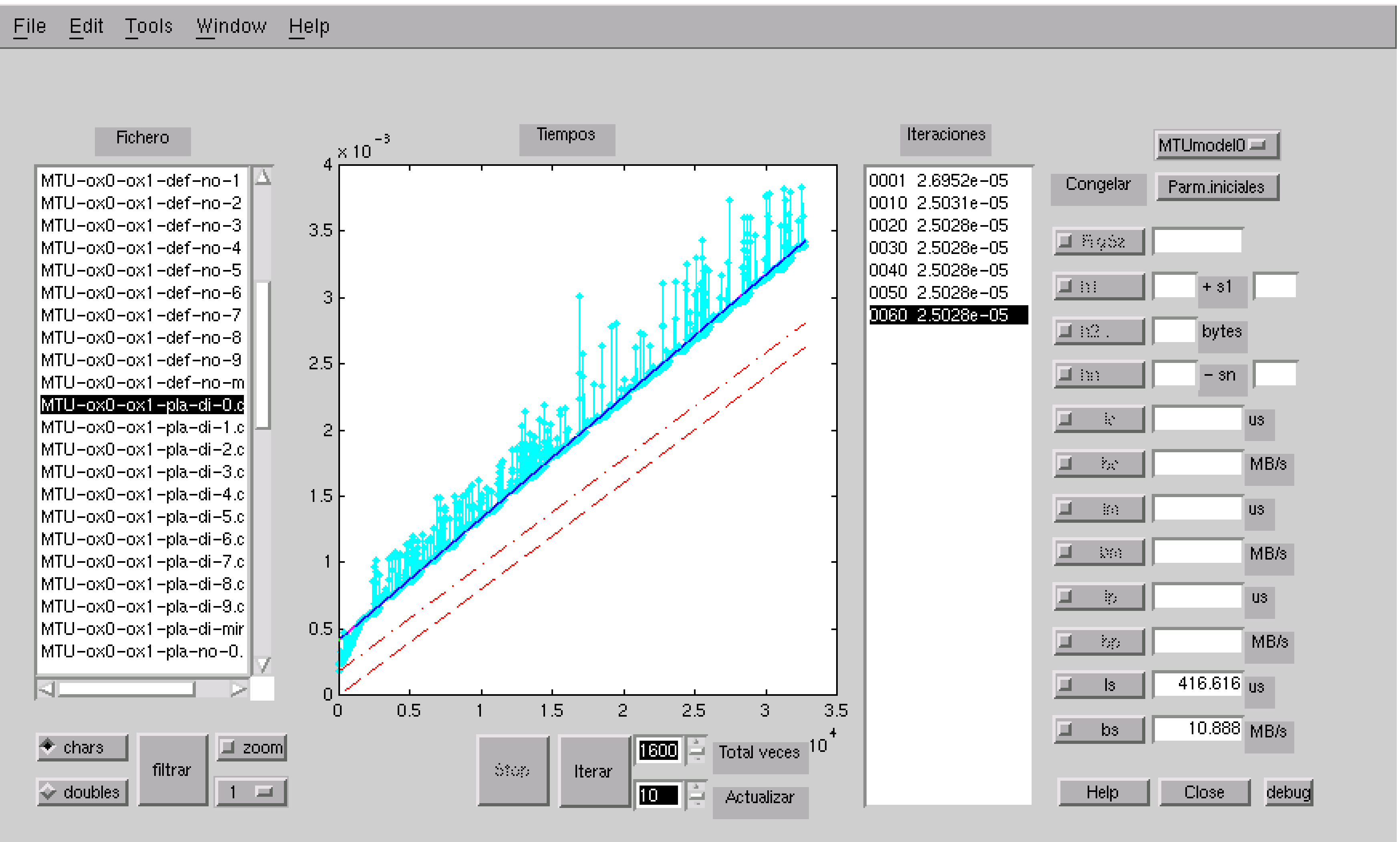


Figura 2.9: GUI desarrollada para automatizar el test de los modelos. Se ha escogido el primer fichero con las opciones PvmDataInplace y PvmRouteDirect. En la gráfica se han seleccionado los datos char. El Modelo 0 contempla los parámetros ls y bs, los demás parámetros corresponden a otros modelos y están desactivados. "Congelando" a 0 el parámetro ls se obtiene el modelo lineal.

- seleccionar los datos referidos a **char**s o **double**s (casillas de chequeo a la izquierda)
- filtrar picos máximos no reproducibles sustituyéndolos por la siguiente medición. Este botón inicia un proceso interactivo en el que el usuario debe confirmar el filtrado de cada máximo propuesto por el GUI, o finalizar el proceso. El punto escogido es remarcado visualmente y se puede hacer *zoom* del mismo para ayudar en la decisión (ver Figura 2.10). También se puede filtrar varios puntos simultáneamente, para acelerar el proceso.
- ajustar los parámetros del modelo a los datos de la medición minimizando el error cuadrático mediante el método iterativo simplex de Nelder-Mead. Por defecto se permite un máximo de 1600 iteraciones, anotándose la evolución de los parámetros cada 10 iteraciones. Ambas constantes pueden alterarse, y la aproximación puede detenerse en cualquier momento mediante un botón “Stop”.
- seleccionar una anotación de la trayectoria simplex (columna derecha). El método iterativo sigue una trayectoria descendente sobre la superficie de error en el espacio de parámetros. Dicha trayectoria es anotada cada cierto número de iteraciones. Concluido el proceso iterativo, al seleccionar una anotación se actualizan las casillas de parámetros y la gráfica del modelo, con lo cual se puede evaluar visualmente la evolución del proceso.
- cambiar el modelo utilizado (casilla de selección superior derecha). Cuatro modelos, numerados del 0 al 3, han superado la etapa de estudio.
- ajustar los parámetros del modelo seleccionado a unos valores típicos (botón “Parámetros Iniciales”). El GUI proporciona los valores más apropiados a la configuración y combinación de opciones utilizada, las cuales se deducen del nombre del fichero y del directorio de medición.
- modificar los parámetros de un modelo (casillas de edición a la derecha). Al cambiar de modelo se desactivan las casillas de los parámetros no relacionados con dicho modelo. También se puede “congelar” un parámetro del modelo seleccionado a un valor concreto, con lo cual no es modificado por el método simplex. Naturalmente, esto restringe la evolución de la trayectoria sobre la superficie de error a la sección correspondiente al valor congelado del parámetro.

El fichero de mediciones visualizado en la Figura 2.9 se denomina MTU−ox0−ox1−pla−di−1.dat (columna izquierda). Se obtuvo por tanto entre los computadores ox0 y ox1 usando las opciones PvmDataInPlace y PvmRouteDirect. El fichero está en el subdirectorio MTUgraph−udp4K−lin32K, de manera que se usó la configuración estándar de PVM (UDPMAXLEN=4K−16), siguiendo una progresión lineal del tamaño del mensaje hasta 32KB. Se han seleccionado las mediciones para datos **char** y el Modelo 0, por lo cual sólo están activos los parámetros contemplados por este modelo, ls y bs. El algoritmo simplex hace evolucionar los parámetros iniciales sugeridos por el GUI (401.6μs, 10.934MB/s) hasta conseguir un error cuadrático de 2.5028 10^{-5} tras 60 iteraciones en el punto (416.616μs, 10.888MB/s).

En la Figura 2.10 se muestra una versión antigua del GUI operando sobre ficheros de mediciones mínimas. El tercer fichero de la secuencia *pla-di* tiene una medición fácilmente indentificable como no mínima. En lugar de desecharlo, desperdiciando el tiempo empleado en generarlo, podría “falsificarse” la medición copiándola del siguiente tamaño de mensaje.

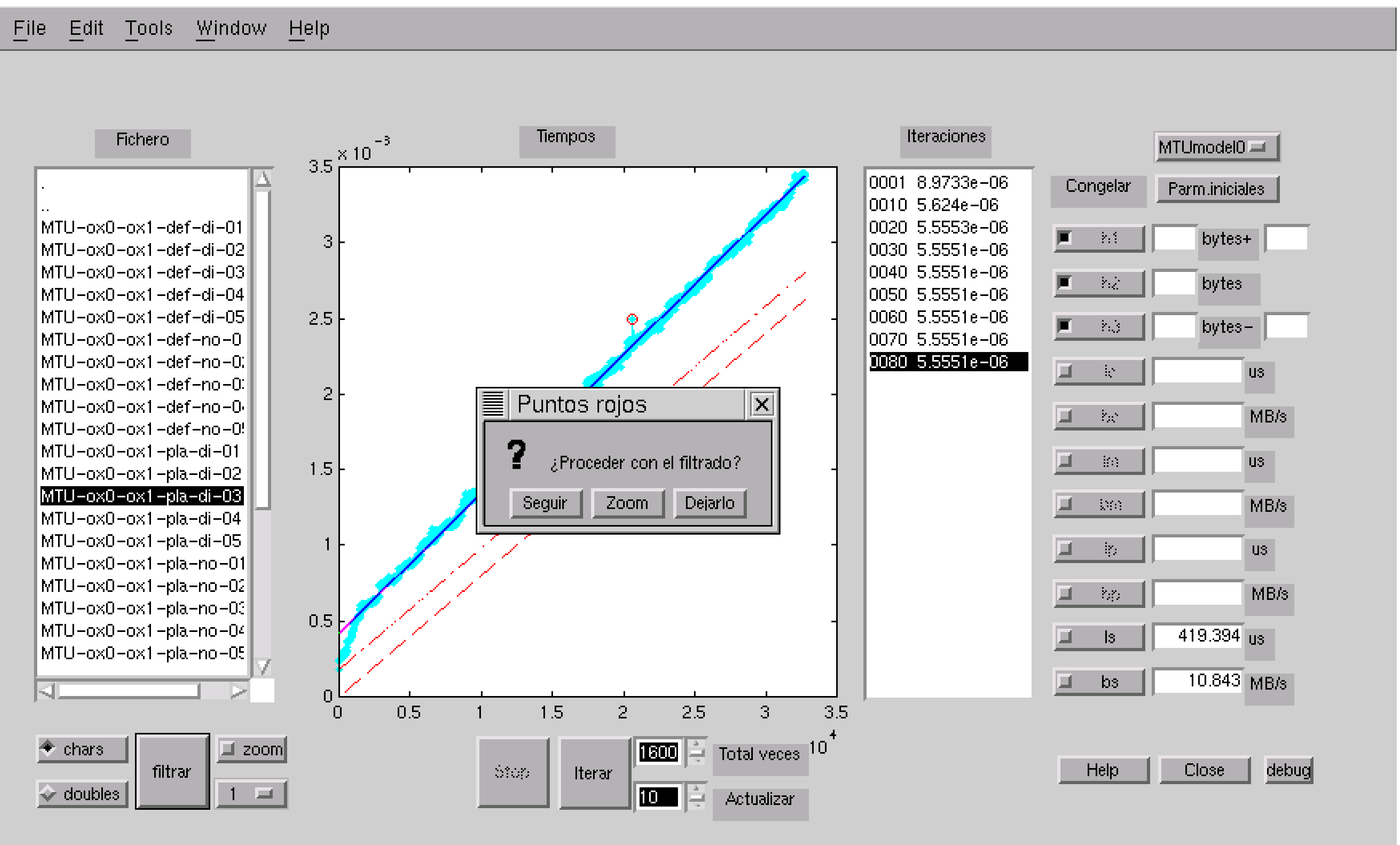


Figura 2.10: El mecanismo de filtrado muestra mediciones susceptibles de ser corregidas y espera confirmación del usuario.

En realidad, el mecanismo de filtrado es innecesario. Estaba destinado a operar sobre ficheros de datos conteniendo tiempos medios y/o mínimos, al objeto de ayudar al desarrollo del código definitivo para los distintos modelos. Esta funcionalidad se incluyó en las primeras etapas del estudio, cuando aún no se tenía excesiva familiaridad con los resultados de las mediciones. La posterior comprobación de la reproducibilidad de los mínimos de medias ha convertido en obsoleta esta capacidad del GUI, habiéndose realizado la extracción de parámetros definitiva sobre los mínimos de las medias mediante un *script* MATLAB, sin utilizar el entorno gráfico.

Los ficheros de mínimos también se obtienen ejecutando otro *script* MATLAB sobre el directorio correspondiente. El proceso completo, desde la obtención de mediciones hasta la extracción de los parámetros, puede ser realizado sin intervención del usuario. Tras haber servido como versátil herramienta durante la etapa de estudio y desarrollo de los modelos, el GUI aún es útil para comprobar visualmente la correción de los resultados obtenidos, al permitir superponer los trazos de los modelos sobre los ficheros de mínimos.

2.3.2 Ficheros de mínimos PVM

Las Figuras 2.11 y 2.12 muestran los ficheros PVM de mínimos de medias de los cuales se han extraido los valores definitivos de parámetros de los modelos.

En los apartados posteriores volverán a aparecer repetidamente, siempre con la misma organización, superponiendo en cada caso los trazos correspondientes a la evaluación del modelo en cuestión sobre los tiempos de transmisión de cada característica de prestaciones. Por este motivo se ha preferido presentarlos con antelación y comentar ahora los detalles relevantes no relacionados con ningún modelo, en lugar de posponer dichos comentarios al apartado dedicado al Modelo 0.

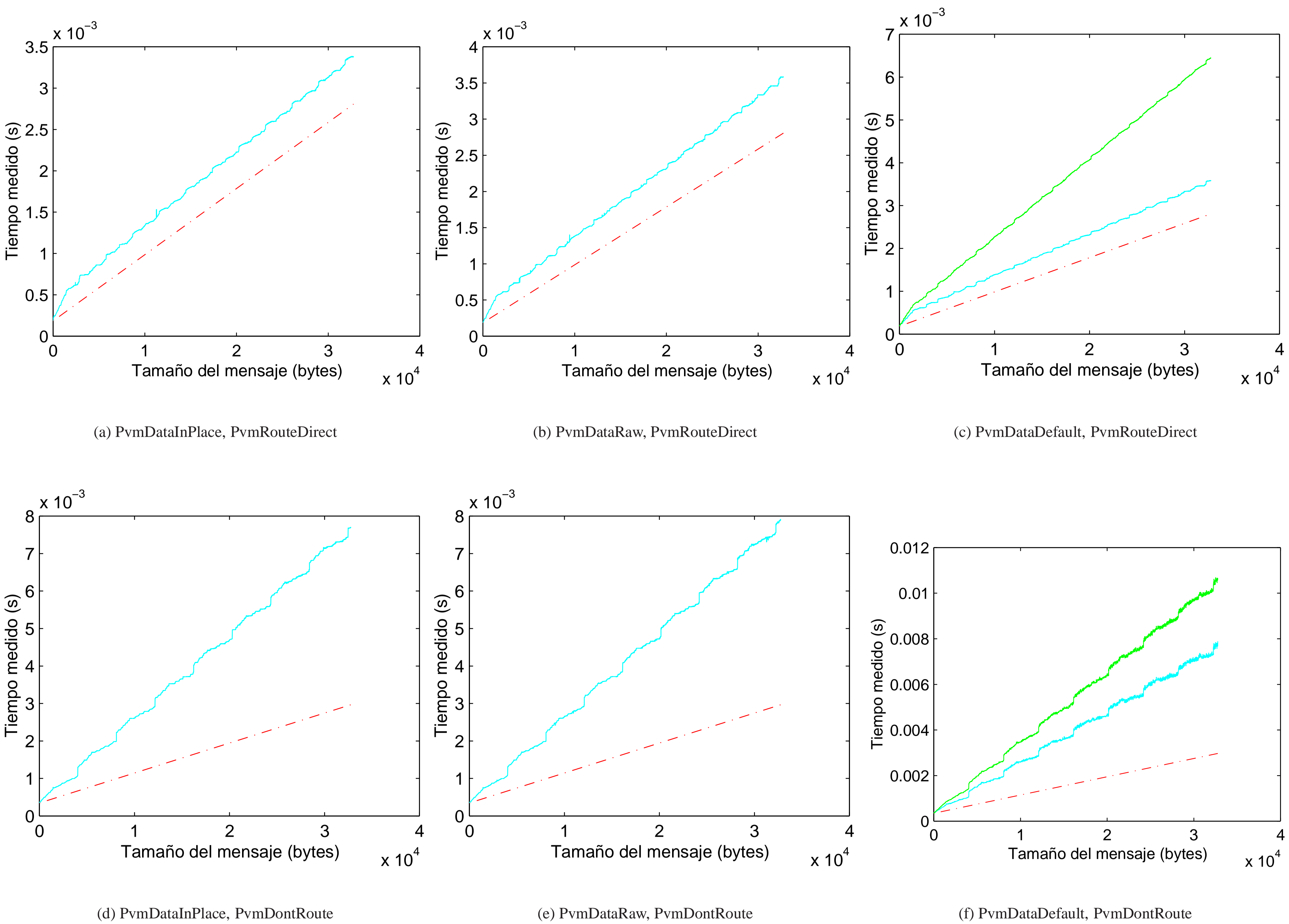
Las gráficas se presentan en dos páginas, la primera dedicada a la configuración estándar, UDPMAXLEN=4K, y la segunda a la coincidente con LAM, UDPMAXLEN=8K.

Cada página se organiza en dos filas de tres columnas. La fila superior corresponde a la ruta directa entre tareas, *PvmRouteDirect*, y la inferior al encaminamiento a través del *daemon* PVM, *PvmDontRoute*. La columna izquierda se dedica al empaquetamiento *PvmDataInPlace*, la central a *PvmDataRaw* y la derecha a *PvmDataDefault*.

Las gráficas son curvas características de prestaciones, mostrando en abscisas el tamaño del mensaje transmitido y en ordenadas el tiempo de transmisión. Se ha seguido la norma de representar en azul el tiempo de transmisión para datos de tipo **char**, y en verde el de **double**s. Este último sólo se representa en la última columna (*PvmDataDefault*), ya que coincide con el de **char**s en los restantes empaquetamientos.

Se añade un trazo rojo de referencia correspondiente a T_0+12.5MB/s, en donde T_0 es la latencia correspondiente a la configuración y combinación de opciones estudiada, y 12.5MB/s es la velocidad del *switch* (100Mbps). Naturalmente, ninguna configuración conseguirá superar esta referencia, que también sirve para recordar que los ejes de ordenadas no son uniformes. En el orden de lectura (de izquierda a derecha, y de arriba hacia abajo) las gráficas aparecen en orden decreciente de prestaciones, con un eje de ordenadas progresivamente más extenso.

No existe una solución perfecta para la presentación comparativa de las 6 gráficas. La aglomeración de los trazos es inevitable. Más adelante se comentan con más detalle las dificultades encontradas.



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

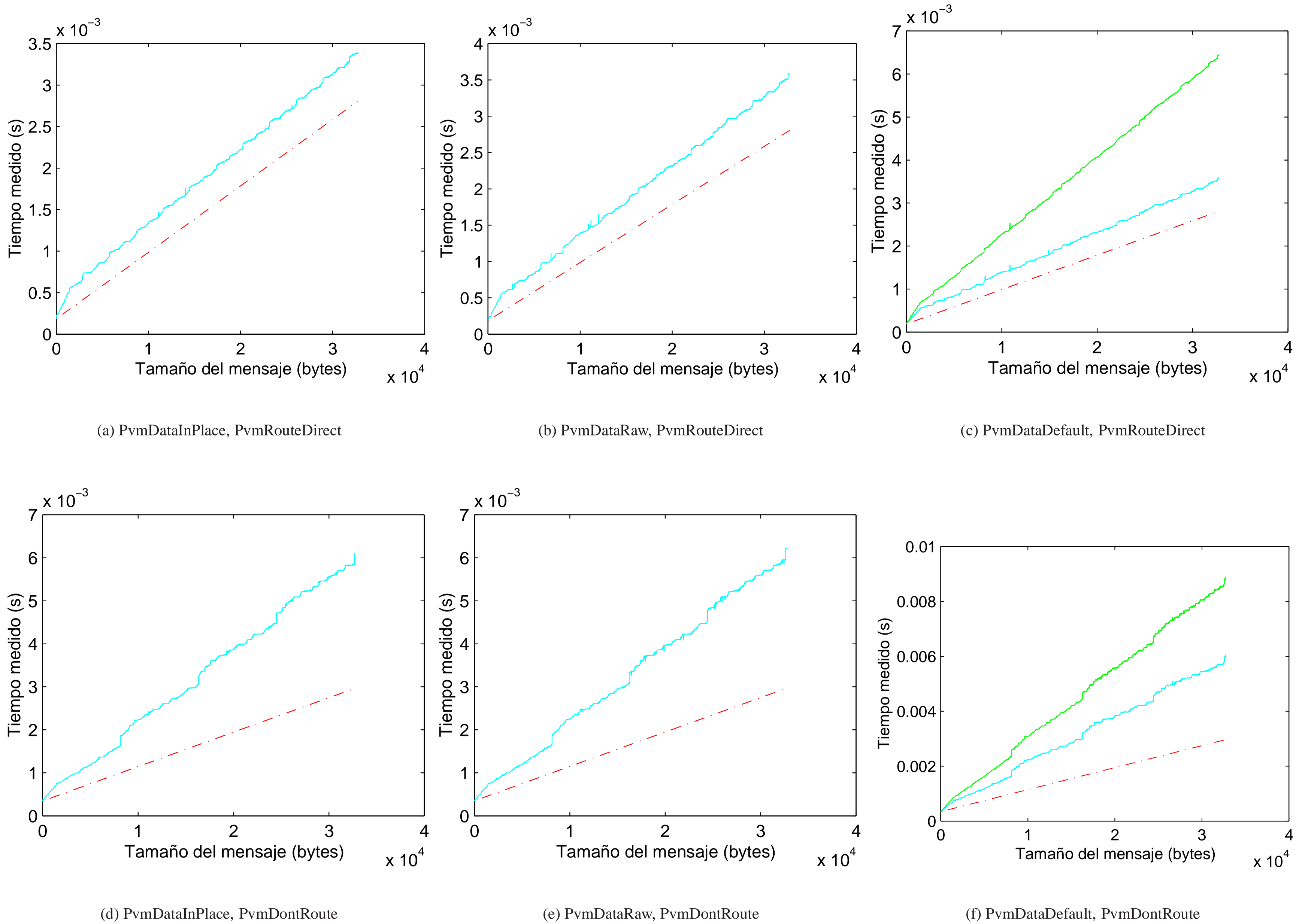
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.11: Mínimos de medias del tiempo de transmisión. Configuración PVM estándar UDPMAXLEN=4K.



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.12: Mínimos de medias del tiempo de transmisión. Configuración PVM coincidente con LAM UDPMAXLEN=8K.

Se pueden destacar de antemano algunas características de los distintos empaquetamientos, encaminamientos y configuraciones UDPMAXLEN, sin necesidad de introducir ningún modelo. Por ejemplo, el encaminamiento a través del *daemon* PVM revela la configuración UDPMAXLEN, ya que las características presentan pequeños escalones al principio de cada unidad de fragmentación PVM. En la fila inferior de la primera página se observan a simple vista 8 fragmentos de 4KB, mientras que en la de la segunda hay 4 fragmentos de 8KB. Naturalmente, una menor fragmentación mejora las prestaciones, como muestran las respectivas escalas de tiempo.

La ruta directa entre tareas no utiliza el mecanismo de fragmentación PVM, por lo cual no aparecen estos tramos en las filas superiores de ambas páginas. El tiempo de transmisión se reduce a la mitad aproximadamente, de donde se deduce que el tráfico entre *daemons* y tareas PVM consume prácticamente el mismo tiempo que la propia transmisión de datos por la red.

La pendiente de la primera MTU es superior a la del resto de la característica, como comentamos anteriormente. Con ruta directa, el resto de la característica es de menor pendiente, mientras que a través del *daemon* se observa que cada fragmento UDP repite el esquema del primero, con una primera MTU de mayor pendiente y las restantes aprovechando el funcionamiento paralelo de los subsistemas de almacenamiento y transmisión implicados.

Como ya se ha comentado, no hay solución perfecta para comparar visualmente las 6 gráficas de cada página. Reunir los trazos en una única gráfica produce la aglomeración observada en las Figuras 2.13(a) y 2.14(a). Afortunadamente, el encaminamiento influye poderosamente en el tiempo de transmisión, separando los trazos en dos grupos bien diferenciados y permitiendo reutilizar los colores de la leyenda.

Un rango de visualización más reducido, como los 5KB de las Figuras 2.13(b) y 2.14(b), puede ser útil para identificar los puntos de media potencia (*Half-Power Point*) presentados resumidamente en la Tabla 2.2. No disponiendo aún de una expresión algebraica para el tiempo de transmisión en función del tamaño de mensaje, hemos optado por la solución informal de sustituir el ancho de banda asintótico por el máximo obtenido en un barrido de 32MB (ver Capítulo 3).

Encaminamiento directo

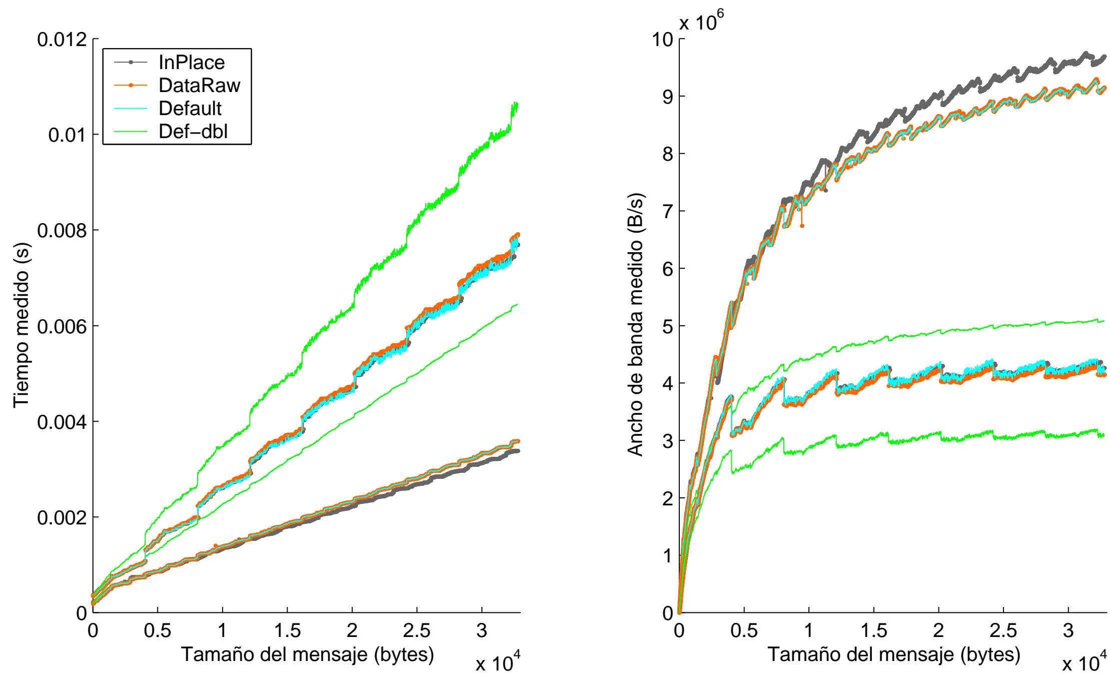
bytes	ms	MB/s	%	Máx.		bytes	ms	MB/s	%	Máx.
4064	0.777	5.232	50.01%	10.462	**InPlace**	**4080**	0.780	5.234	50.03%	10.462
3544	0.738	4.805	49.98%	9.614	**DataRaw**	**3544**	0.737	4.807	50.00%	9.614
3552	0.739	4.806	49.97%	9.617	**Default**	**3544**	0.736	4.817	50.09%	9.617
1968	0.749	2.628	50.01%	5.255	**Def/dbl**	**1968**	0.749	2.628	50.02%	5.255

Encaminamiento PVM

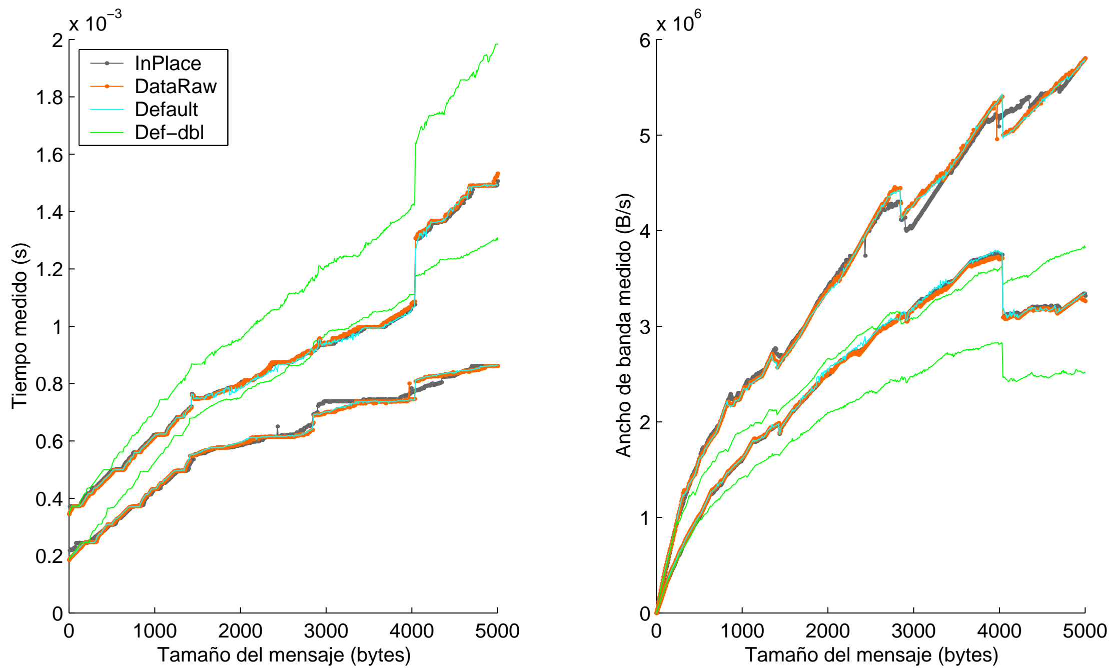
bytes	ms	MB/s	%	Máx.		bytes	ms	MB/s	%	Máx.
1664	0.762	2.185	50.02%	4.368	**InPlace**	**2440**	0.874	2.792	49.97%	5.588
1616	0.750	2.156	50.33%	4.284	**DataRaw**	**2424**	0.874	2.774	49.98%	5.551
1680	0.752	2.234	50.50%	4.424	**Default**	**2472**	0.869	2.845	50.07%	5.682
1280	0.800	1.600	50.06%	3.196	**Def/dbl**	**1776**	0.931	1.907	50.13%	3.804

(a) Configuración estándar UDPMAXLEN=4K. (b) Configuración std. LAM UDPMAXLEN=8K.

Tabla 2.2: Puntos de media potencia para las diversas configuraciones y opciones de PVM. Los anchos de banda se comparan con el máximo obtenido en el barrido exponencial de 32MB del Capítulo 3 (columna etiquetada **Máx**). Ver también las Figuras 2.13(b) y 2.14(b).

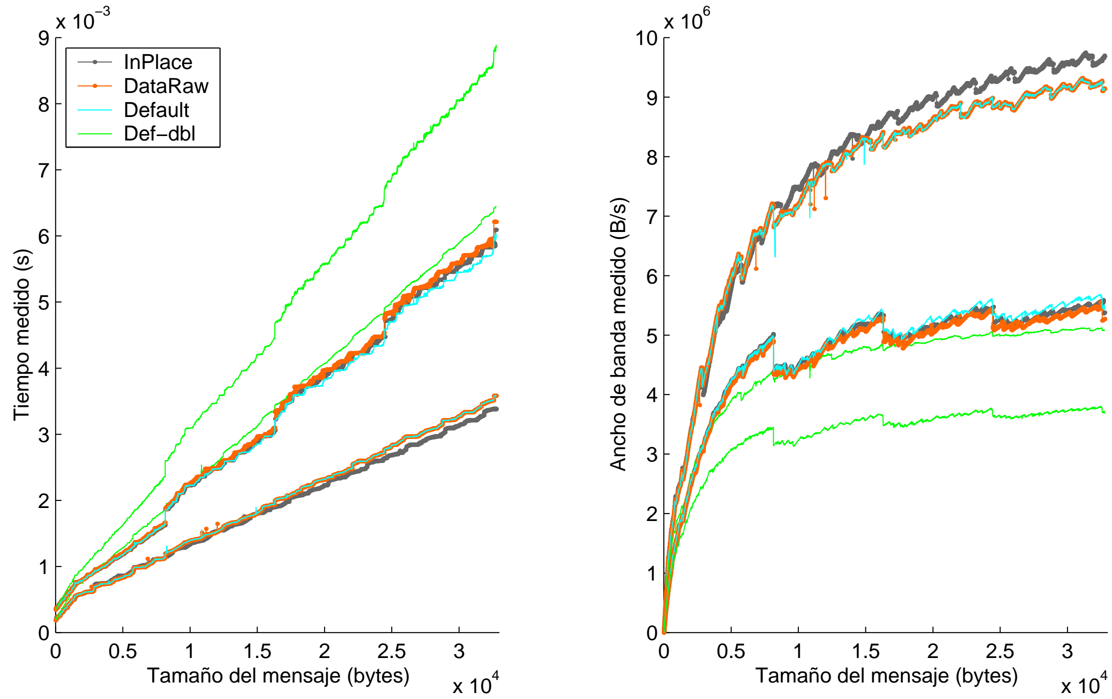


(a) Curva característica de prestaciones para las 6 combinaciones de opciones PVM.

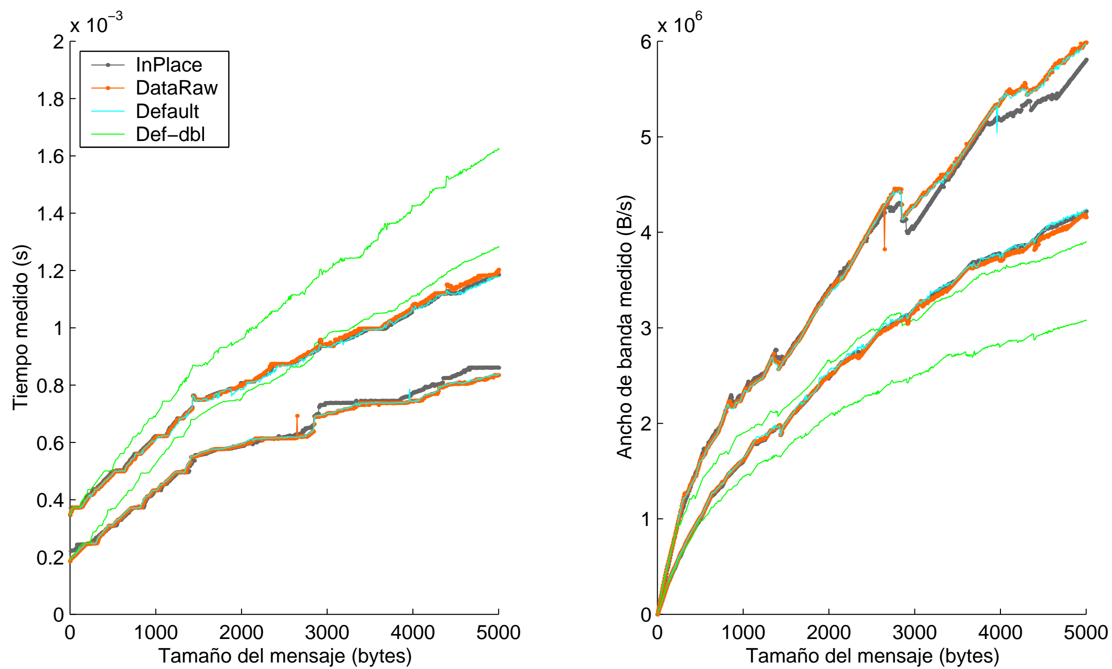


(b) Ampliación al rango de 5KB.

Figura 2.13: Agrupación comparativa de las 6 gráficas de la Figura 2.11. Configuración PVM estándar UDPMAXLEN=4K.



(a) Curva característica de prestaciones para las 6 combinaciones de opciones PVM.



(b) Ampliación al rango de 5KB.

Figura 2.14: Agrupación comparativa de las 6 gráficas de la Figura 2.12. Configuración PVM coincidente con LAM UDPMAXLEN=8K.

El rango de 5KB separa un poco los trazos y permite distinguir mejor la acusada pendiente del tramo inicial, pero no muestra de una forma clara el efecto de la fragmentación UDP. Además, la combinación de opciones *DataDefault-RouteDirect* para **double**s (trazo verde del grupo inferior en las Figuras 2.13(b) y 2.14(b) izquierda) aparece como muy similar a las combinaciones *DontRoute* (trazos del grupo superior, salvo el verde), ya que empiezan a diferenciarse con un tamaño de mensaje superior. En los primeros 5KBs apenas da tiempo a superar el punto de media potencia.

Se escogió 32KB como rango de barrido en un intento de compaginar un número suficiente de fragmentos UDP y un ancho de banda más próximo al asintótico con una escala lo suficientemente pequeña como para poder seguir observando los pequeños tramos de que se componen las curvas características.

2.3.3 Ficheros de mínimos LAM

La misma estructura utilizada para presentar los datos obtenidos bajo PVM se puede aplicar a los ficheros de mínimos generados con LAM/MPI, salvo que las combinaciones de opciones se reducen a 4, según se active o no la opción −O (cluster homogéneo vs. heterogéneo) y se escoja entre −c2c o −lamd (ruta cliente-a-cliente vs. encaminamiento a través del *daemon* LAM). Las Figuras 2.11 y 2.12 muestran los ficheros MPI de mínimos de medias de los cuales se han extraido los valores definitivos de parámetros de los modelos.

De nuevo se organizan las gráficas en dos páginas, la primera dedicada a la configuración con menor tamaño UDP, MAXNMSGLEN==4440, y la segunda a la configuración estándar LAM, MAXNMSGLEN==8K. La fila superior corresponde a la ruta TCP directa entre tareas (*cliente-a-cliente*, en terminología LAM), y la inferior al encaminamiento a través del *daemon* LAM. La columna izquierda se dedica a la opción de cluster homogéneo y la derecha a la de heterogéneo.

Se vuelve a seguir la norma de representar en azul el tiempo de transmisión para datos **char**, y en verde el de **double**s. Este último sólo difiere para la columna derecha, cluster heterogéneo.

En el orden de lectura (de izquierda a derecha, y de arriba hacia abajo) las gráficas aparecen en orden decreciente de prestaciones, con un eje de ordenadas progresivamente más extenso. El trazo rojo de referencia T_0+12.5MB/s se vuelve a usar como recordatorio de las diferentes escalas utilizadas.

El reducido número de combinaciones de opciones hace posible reunir las 4 gráficas de cada configuración (4K y 8K) en las Figuras 2.17(a) y 2.18(a), respectivamente, sin una excesiva aglomeración de trazos. Esto permite una comparación visual entre ellas, más fácilmente asimilable que la consulta a los ejes de ordenadas de las gráficas aisladas.

Las mismas peculiaridades notadas para PVM se mantienen cualitativamente en LAM/MPI, aunque difieren cuantitativamente. Así, la configuración MAXNMSGLEN queda delatada en las filas inferiores de las Figuras 2.15 y 2.16, correspondientes al modo *daemon* LAM, observándose a simple vista los mismos 8 fragmentos de 4KB y 4 fragmentos de 8KB que se observaron bajo PVM, así como las superiores prestaciones de la configuración estándar de LAM.

Recordemos que la configuración coincidente con PVM ajusta en realidad MAXNMSGLEN=4440, por lo cual el 8º fragmento sólo aparece parcialmente. Otra diferencia notable respecto a PVM es que sólo el segundo fragmento UDP repite la estructura del primero, observándose un salto importante entre ellos. Los restantes fragmentos UDP presentan un ancho de banda mayor (menor pendiente que los dos primeros) y constante.

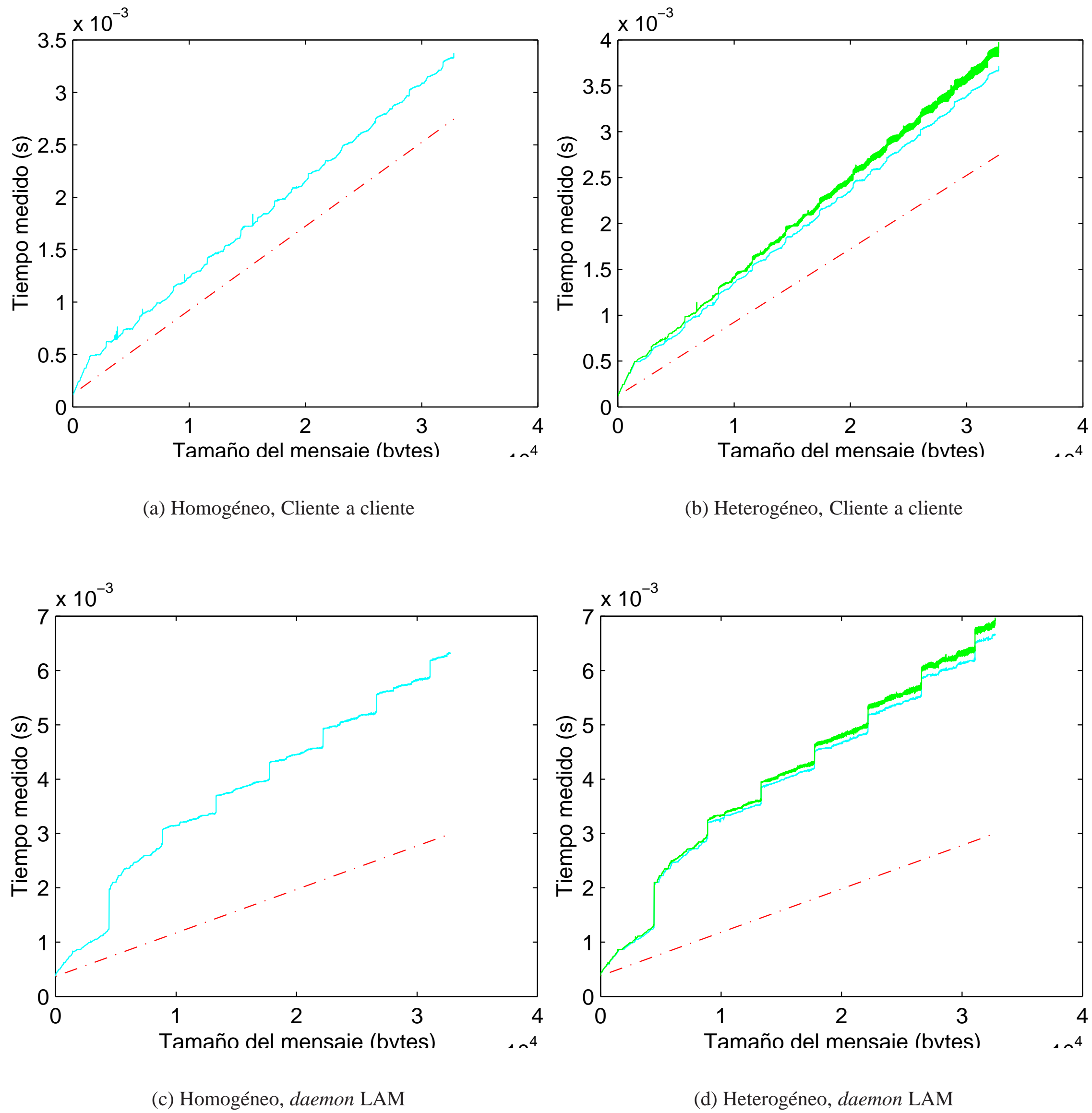
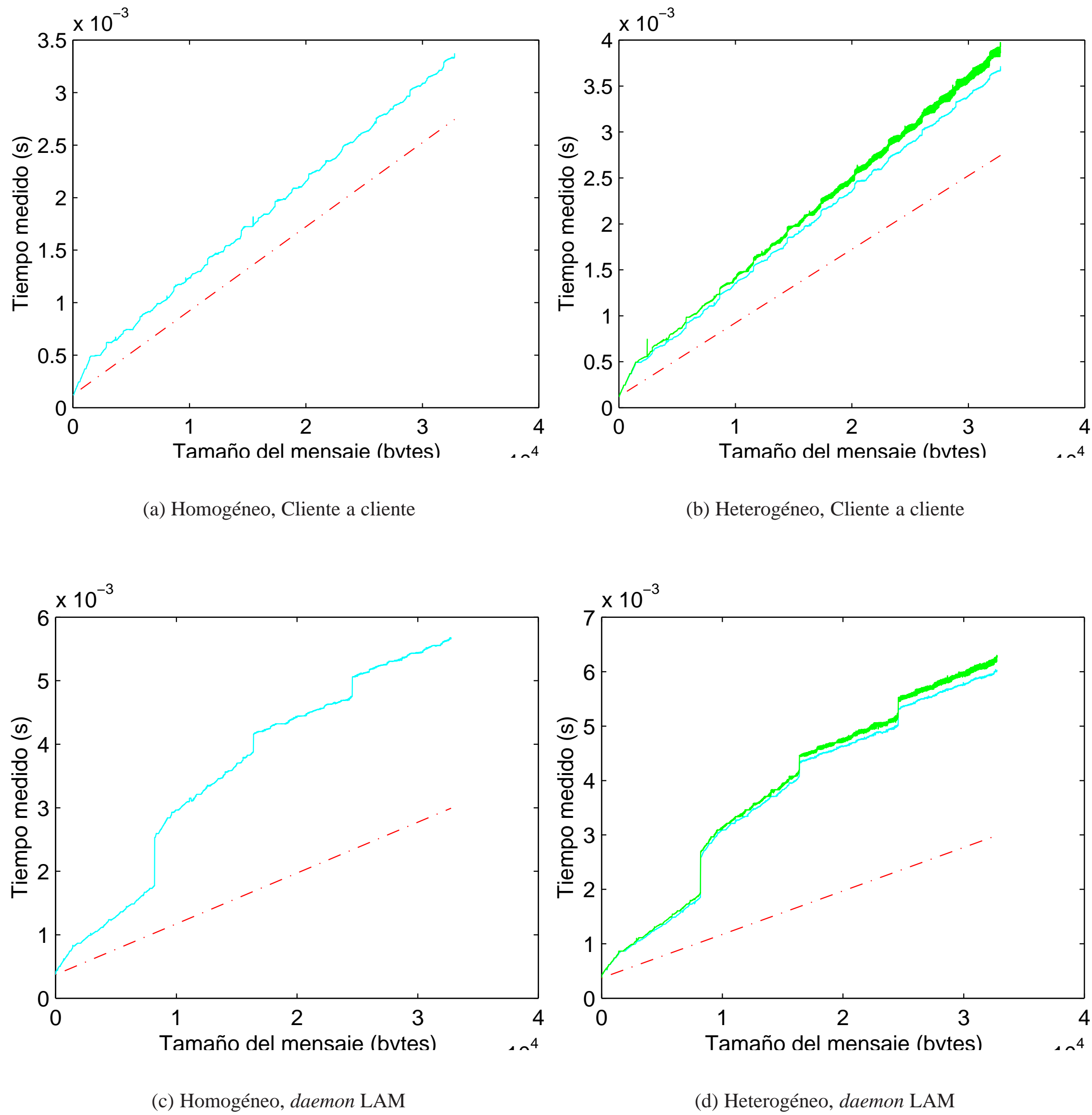


Figura 2.15: Mínimos de medias del tiempo de transmisión. Configuración LAM coincidente con PVM MAXNMSGLEN=4K.



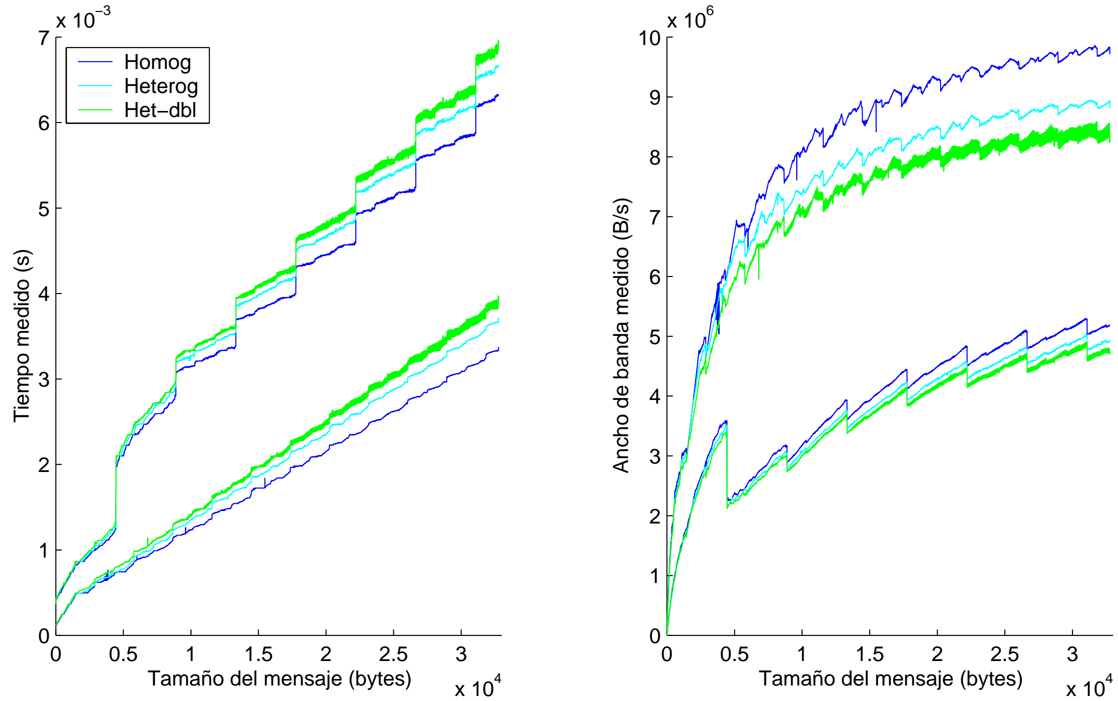
(a) Homogéneo, Cliente a cliente

(b) Heterogéneo, Cliente a cliente

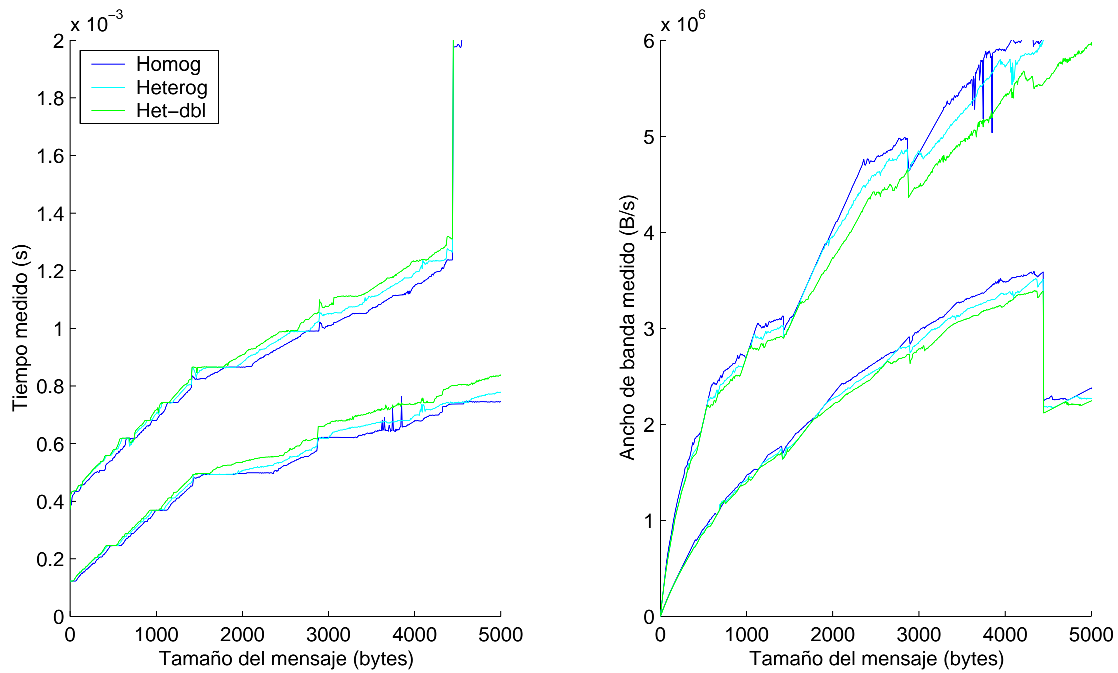
(c) Homogéneo, *daemon* LAM

(d) Heterogéneo, *daemon* LAM

Figura 2.16: Mínimos de medias del tiempo de transmisión. Configuración LAM estándar, fragmentos UDP de MAXNMSGLEN=8K.

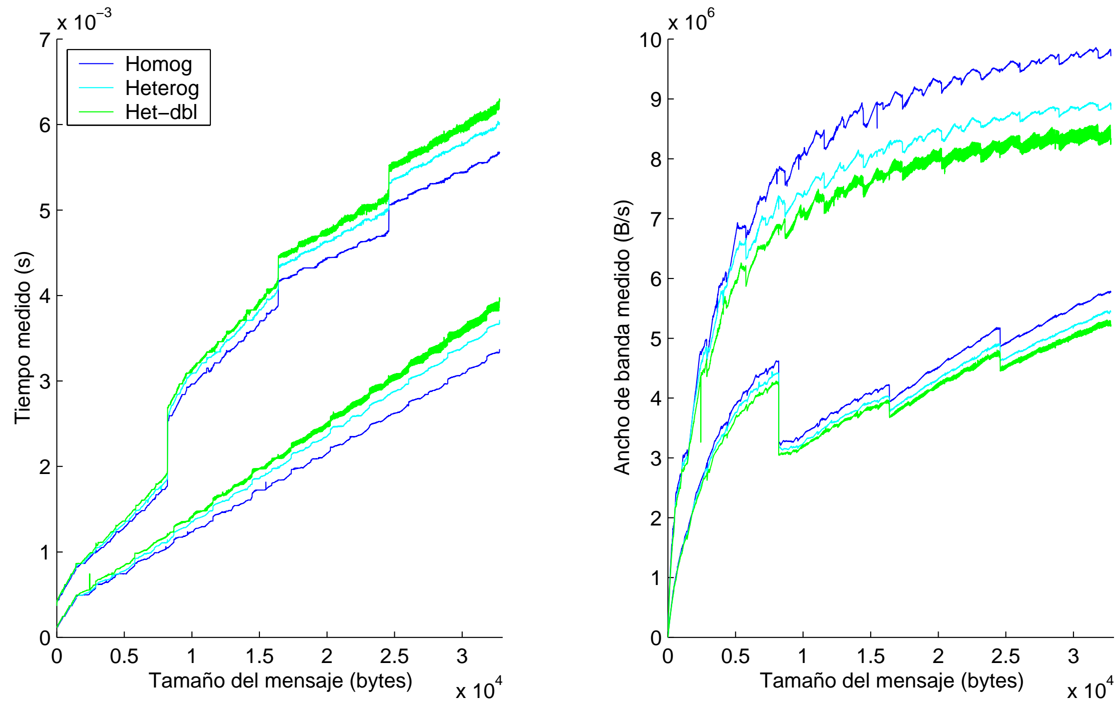


(a) Curva característica de prestaciones para las 4 combinaciones de opciones LAM.

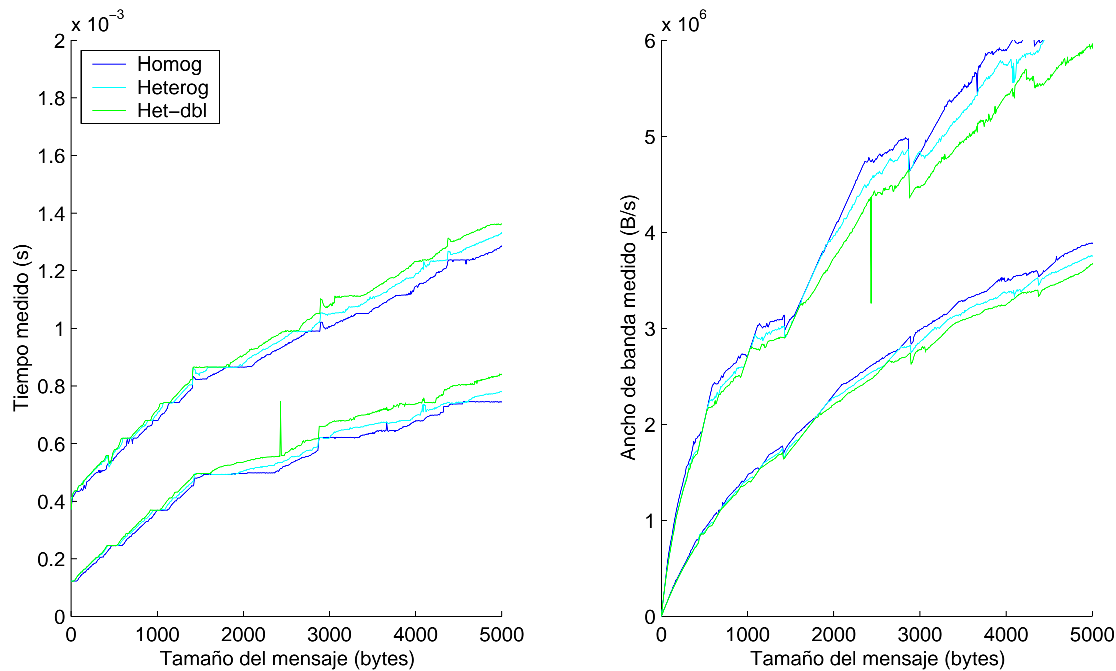


(b) Ampliación al rango de 5KB.

Figura 2.17: Agrupación comparativa de las 4 gráficas de la Figura 2.15. Configuración LAM coincidente con PVM MAXNMSGLEN=4K.



(a) Curva característica de prestaciones para las 4 combinaciones de opciones LAM.



(b) Ampliación al rango de 5KB.

Figura 2.18: Agrupación comparativa de las 4 gráficas de la Figura 2.16. Configuración LAM estándar MAXNMSGLEN=8K.

Naturalmente estas peculiaridades desaparecen en el modo cliente-a-cliente, en donde el mecanismo de fragmentación del *daemon* LAM no resulta implicado. Igual que en PVM, el tiempo de transmisión se reduce a la mitad; para estos tamaños de mensaje, el tráfico entre *daemons* y tareas LAM consume prácticamente el mismo tiempo que la propia transmisión de datos por la red.

Independientemente del modo *daemon* o cliente-a-cliente, la pendiente de la primera MTU es superior a la del resto de la característica. Con ruta directa, el resto de la característica es de menor pendiente (mayor ancho de banda). A través del *daemon*, como ya se ha comentado, los dos primeros fragmentos UDP siguen una estructura similar y los restantes presentan menor pendiente que ellos. De hecho, presentan menor pendiente que el propio encaminamiento directo (ver Figuras 2.17(a) y 2.18(a), izquierda), revelando que parte del tiempo de procesamiento del *daemon* queda oculto por el tiempo de transmisión, quedando descubierto al principio de cada nuevo fragmento UDP, cuando el tiempo de transmisión adicional es muy pequeño.

Al igual que bajo PVM, la diferencia entre los modos directo y *daemon* es tan grande que permite reutilizar los colores de leyenda en las gráficas comparativas de las Figuras 2.17(a) y 2.18(a). A la izquierda se muestran los tiempos de transmisión agrupados, y a la derecha el correspondiente ancho de banda "lineal" despejado del correspondiente modelo $T = S/B$. En ambas gráficas quedan los trazos separados en dos grupos correspondientes a los modos directo y *daemon*: en las de tiempo (izquierda) el modo directo presenta menor tiempo de transmisión y queda por debajo del modo *daemon*, mientras que a la derecha presenta mayor ancho de banda, quedando por encima.

Por permitir una comparación más detallada con PVM, se ofrecen las mismas ampliaciones al rango de 5KB con la misma escala en los ejes de coordenadas (Figuras 2.17(b) y 2.18(b)). Se aprovecha la ocasión para presentar el equivalente LAM de la Tabla 2.2 con la relación de los puntos de media potencia bajo las diversas configuraciones y combinaciones de opciones. Los anchos de banda máximos de la Tabla 2.3 corresponden naturalmente al barrido de 32MB realizado en el Capítulo 4.

Modo cliente-a-cliente

bytes	ms	MB/s	%	Máx.		bytes	ms	MB/s	%	Máx.
3776	0.642	5.884	50.04%	11.758	**Homogéneo**	**3832**	0.652	5.880	50.00%	11.758
2568	0.554	4.639	49.99%	9.281	**Heterogéneo**	**2576**	0.554	4.647	50.07%	9.281
2384	0.555	4.298	50.06%	8.586	**Het/double**	**2384**	0.555	4.299	50.12%	8.579

Modo *daemon*

bytes	ms	MB/s	%	Máx.		bytes	ms	MB/s	%	Máx.
16856	3.948	4.269	50.01%	8.536	**Homogéneo**	**6336**	1.484	4.270	50.02%	8.536
4336	1.234	3.513	50.08%	7.015	**Heterogéneo**	**4336**	1.235	3.512	50.06%	7.015
3752	1.177	3.187	50.08%	6.364	**Het/double**	**3744**	1.175	3.186	50.07%	6.364

(a) Configuración std.PVM MAXNMSGLEN=4K.

(b) Configuración std. MAXNMSGLEN=8K.

Tabla 2.3: Puntos de media potencia para las diversas configuraciones y opciones de LAM. Los anchos de banda se comparan con el máximo obtenido en el barrido exponencial de 32MB del Capítulo 4 (columna etiquetada **Máx**). Ver también las Figuras 2.17(b) y 2.18(b).

Merece la pena volver a comentar que en un rango de barrido tan reducido como 5KB apenas da tiempo a superar el punto de media potencia, ni queda claramente manifiesto el mecanismo de fragmentación del modo *daemon*. El rango de 32KB reúne los suficientes fragmentos UDP como para observar más claramente dicho mecanismo y acercarse bastante al ancho de banda asintótico, mientras que aún permite distinguir a simple vista los pequeños tramos de que se componen las curvas características y las distintas pendientes de los mismos, en particular la elevada pendiente de la primera MTU.

Los modelos que se presentan a continuación intentan, con un progresivo grado de refinamiento, ajustar sobre estas curvas características los correspondientes trazos predictivos, llegando a requerir como parámetro el tamaño de fragmento UDP utilizado.

2.3.4 Modelo 0

Este modelo es el previamente comentado $T = L + S/B$. Congelando $L = 0$ queda reducido al modelo lineal puro. No está basado en MTUs. Se incluye en esta sección simplemente como referencia. Todos los modelos que propongamos deberán superar la capacidad de predicción de este modelo. El código MATLAB utilizado para evaluarlo se muestra en el Apéndice B.2.

En este modelo existe una diferencia notable entre el parámetro latencia, L, y el *tiempo de transmisión del mensaje nulo*, T_0. En la Figura 2.19 se ilustra esta distinción. Debido al reducido número de parámetros, el modelo sólo puede trazar una recta sobre la gráfica. En principio, se desearía que el modelo se superpusiera a las mediciones lo máximo posible (trazo negro en Figura 2.19(a)). El tramo inicial de mayor pendiente no puede ser modelado.

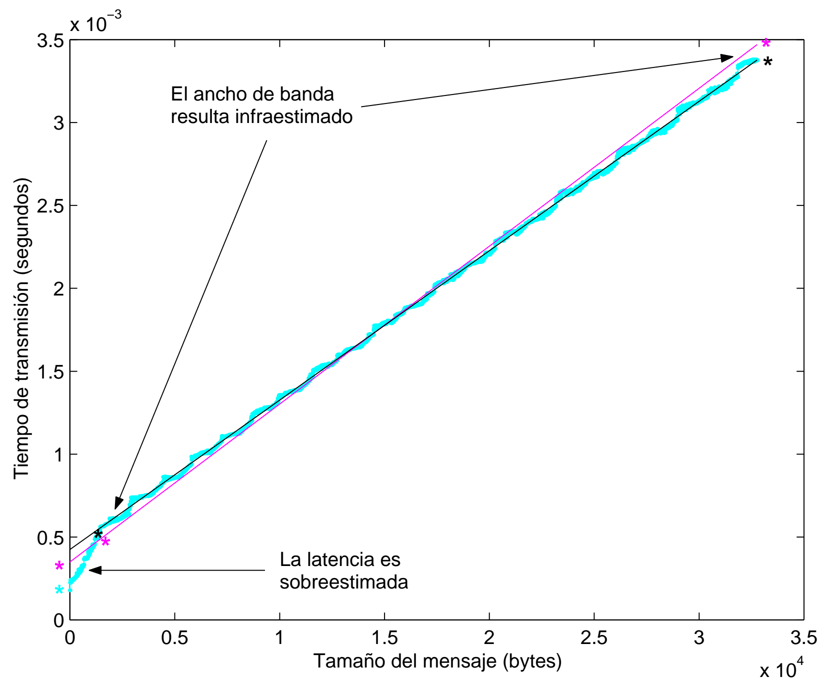
El método simplex minimiza el error cuadrático centrando la recta del modelo sobre la gráfica de las mediciones. En la Figura 2.19(a) el trazo morado exagera este comportamiento; en la Figura 2.19(b) refleja fielmente los parámetros extraidos. Dado que la gráfica tiene un tramo inicial de pendiente más pronunciada, este centrado deja los tramos inicial y final del modelo por encima de las mediciones reales.

El parámetro L resulta sobreestimado, ya que el tramo inicial del modelo queda por encima de las mediciones, y por tanto por encima de T_0. Dicha sobreestimación es menor que la que hubiera resultado de ignorar el tramo inicial durante la minimización (trazo negro).

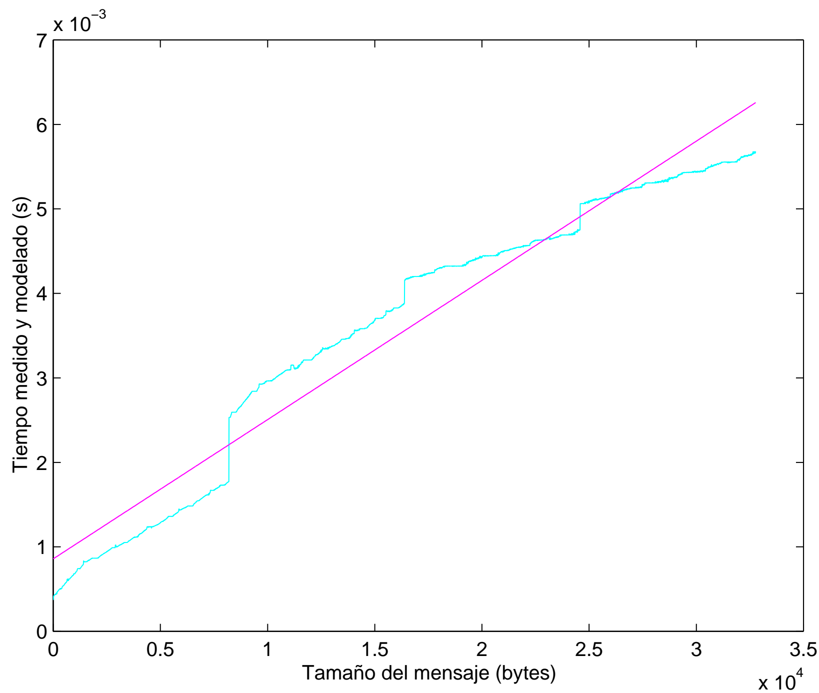
El parámetro B resulta infraestimado, ya que el tramo final del modelo también queda por encima de las mediciones mientras que el tramo central queda por debajo. El modelo predice una progresión del tiempo de transmisión más acentuada que la que se produce en las mediciones, es decir, un ancho de banda menor.

Como se comentó al comenzar este apartado, bajo PVM se ha evaluado el modelo variando el modo de empaquetamiento de datos (PvmDataInPlace, PvmDataRaw, PvmDataDefault) y de encaminamiento del mensaje (PvmRouteDirect, PvmDontRoute), contemplándose 6 combinaciones de opciones en total. El estudio se repite para una configuración alternativa de UDPMAXLEN=8K, coincidente con el tamaño UDP por defecto de LAM.

Bajo LAM, las combinaciones de opciones se reducen a 4, dado que la opción homóloga al empaquetamiento PVM (-O) sólo toma dos posibles valores: cluster homogéneo o heterogéneo.



(a) El efecto es reducido en los modos con ruta TCP directa.



(b) El modelo carece de capacidad de extrapolación para los modos *daemon*, particularmente para el *daemon* LAM. La recta del modelo queda lo más centrada posible sobre las mediciones. Resulta evidente que el ancho de banda extraido depende fuertemente del rango de medición considerado.

Figura 2.19: El modelo afín sobreestima L e infraestima B.

Parámetros extraidos bajo PVM

La Figura 2.20 muestra el fichero de mínimos de cada una de las seis combinaciones de opciones PVM bajo la configuración estándar, y la Figura 2.21 bajo la configuración coincidente, con la misma disposición en que se presentaron en el Apartado 2.3.2, superponiendo esta vez las respectivas predicciones del Modelo 0, en color morado.

La referencia T_0+12.5MB/s (en rojo) ayuda a identificar visualmente la escala del eje de ordenadas, tiempo T. Representa un sistema "ideal" con latencia igual al tiempo de transmisión del mensaje nulo, y ancho de banda igual al del *switch* usado. Se debe observar que dicha latencia varía según la combinación de opciones PVM utilizada. Naturalmente, ningún punto de la medición conseguirá rebajar el mínimo dado por la referencia.

Manteniendo la estructura del Apartado 2.3.2, la Figura 2.22 agrupa las 6 subgráficas anteriores para la configuración estándar, y la Figura 2.23 para la configuración coincidente con LAM. Esto permite comparar entre sí las distintas opciones de empaquetamiento y encaminamiento sin necesidad de recurrir a consultar el eje de ordenadas o estimar la relación con el trazo rojo de referencia. Se incluyen también las predicciones del modelo, cada trazo en un tono algo más oscuro del mismo color que la correspondiente opción PVM.

Si se ha de presentar dicha comparación gráficamente, es inevitable incurrir en la aglomeración de los trazos. Por otro lado, separar los trazos en gráficas disjuntas impide, o al menos dificulta considerablemente, la comparación de las diversas opciones PVM.

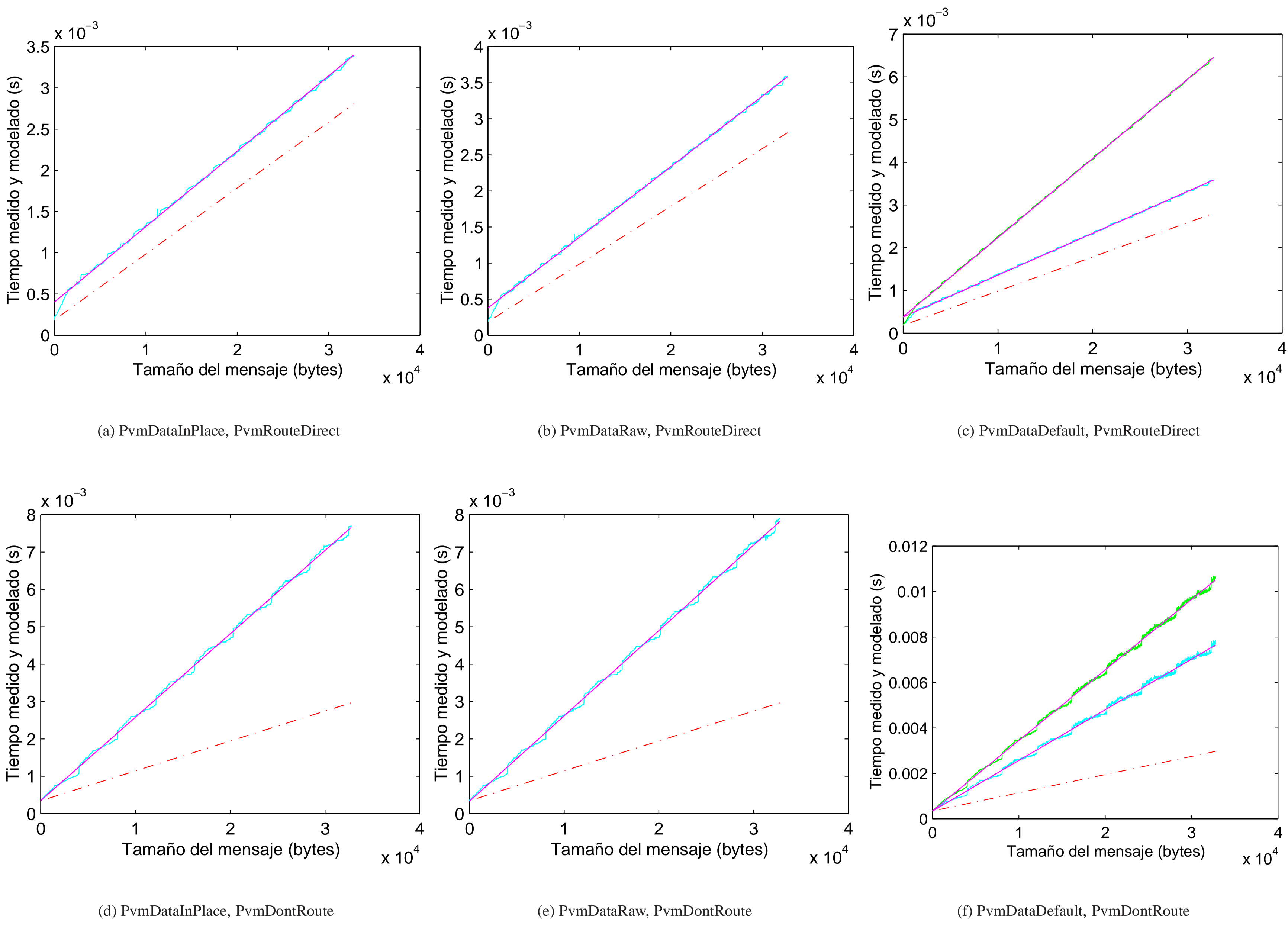
La Tabla 2.4 resume los valores de parámetros extraidos de los correspondientes 12 ficheros PVM de mínimos de medias, habiéndose utilizando para ello el mencionado *script* de automatización. Dado que con el empaquetamiento PvmDataDefault se debe aplicar el modelo tanto a datos **char** como **double**, la tabla presenta un total de 16 entradas.

Al objeto de resumir la información y permitir obtener conclusiones de ella, la Tabla 2.5 vuelve a presentar los anchos de banda de la Tabla 2.4 (parámetro B) normalizados respecto al mínimo. Esta medida, asimilable a una ganancia en velocidad (*speedup*), ignora premeditadamente el parámetro L.

Para analizar las gráficas tal vez resulte más cómoda la medida inversa, una "pérdida en tiempo" (Tabla 2.6) resultado de normalizar los inversos de dichos anchos de banda respecto a su mínimo (máximo ancho de banda).

Repasando la información obtenida, lo que más puede extrañar inicialmente es el comportamiento de las latencias (parámetro L) en la Tabla 2.4. Los anchos de banda en general decrecen conforme se van usando empaquetamientos menos eficientes, pero las latencias presentan el mayor valor justamente para RouteDirect, DataInPlace. Como ya se remarcó en la Figura 2.19, esto es un problema previsible del modelo, que sobreestima L. Probablemente las Figuras 2.22(b) y 2.23(b) sean las más apropiadas para reconocer de un rápido vistazo el problema. Los trazos con menor pendiente intersectan con el eje de ordenadas por encima de donde lo hacen los trazos de mayor pendiente.

Centrándonos en los anchos de banda, la opción PvmDataInplace sólo es apreciablemente superior a los restantes empaquetamientos en conjunción con el encaminamiento directo. En la Tabla 2.4 se observa una ventaja de 0.7MB/s sobre DataRaw con ruta directa, reducida a 0.1MB/s con DontRoute. En la Tabla 2.5, la ventaja de 0.2 puntos de ganancia queda reducida a 0.03 puntos con DontRoute.



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

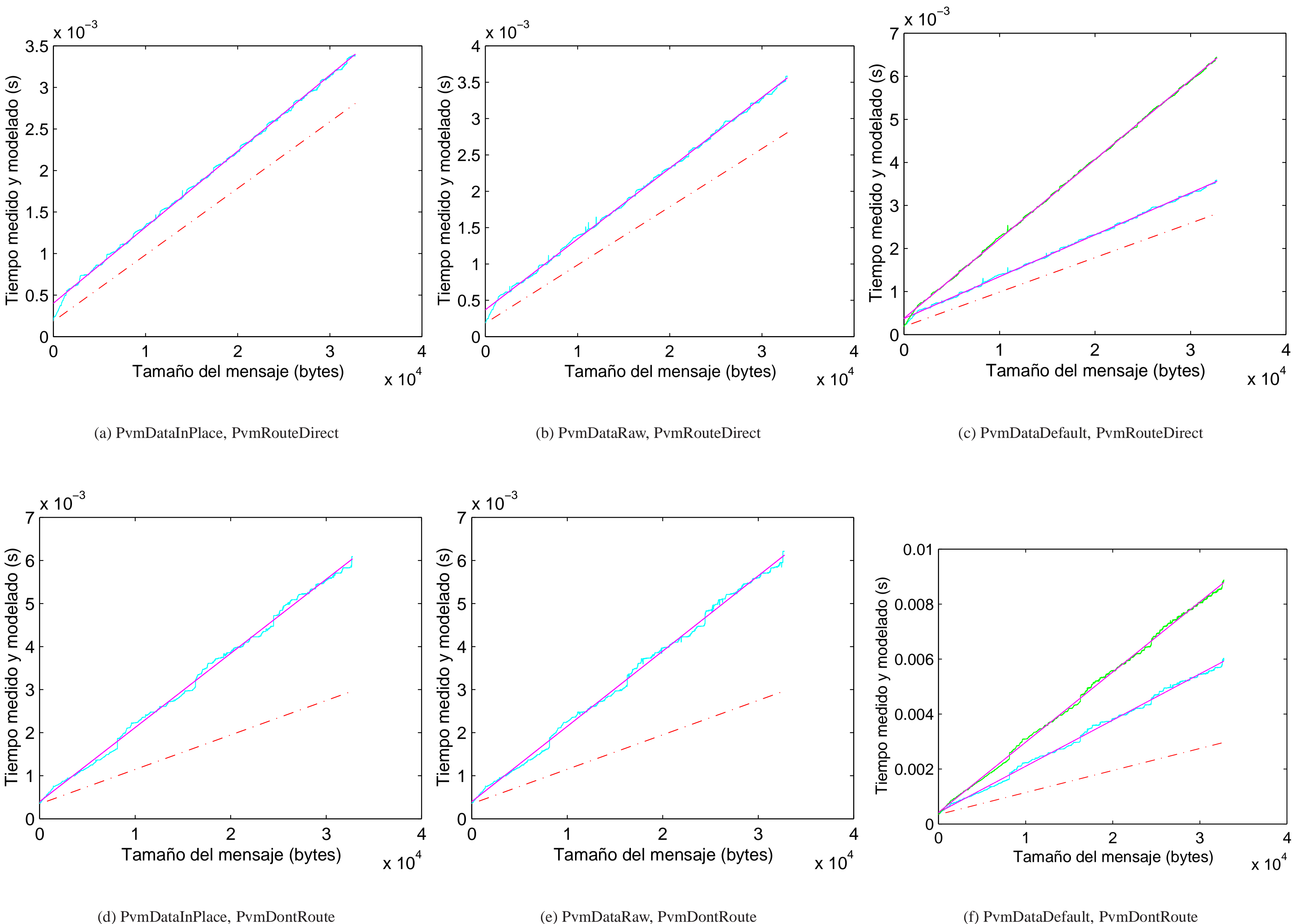
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.20: Modelo 0, configuración estándar PVM (UDPMAXLEN 4K).



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

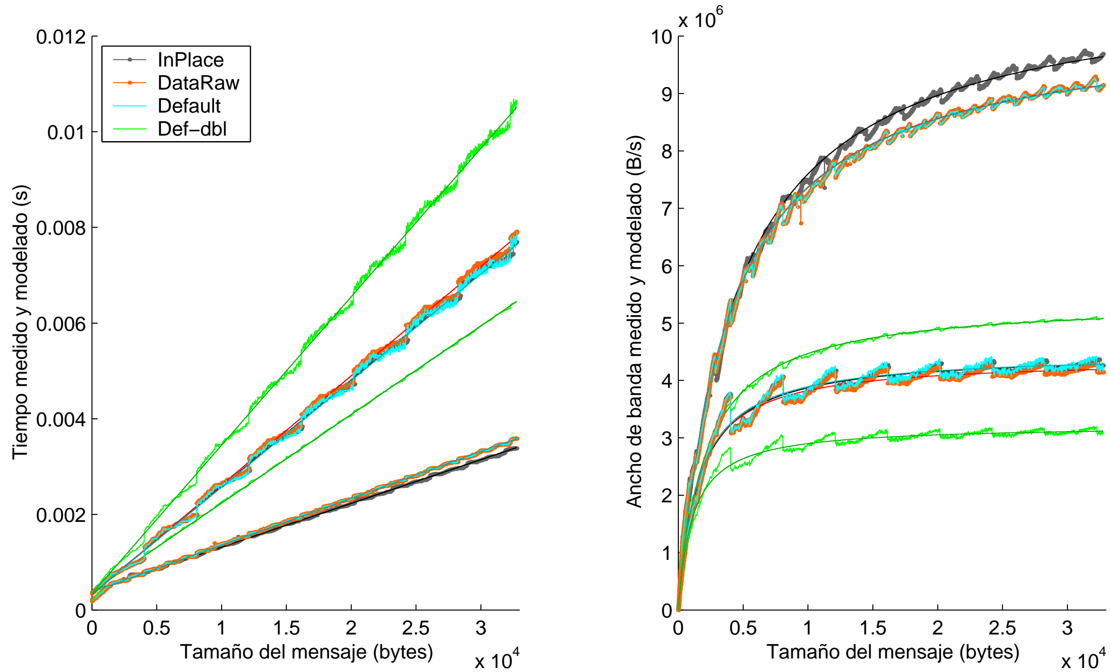
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

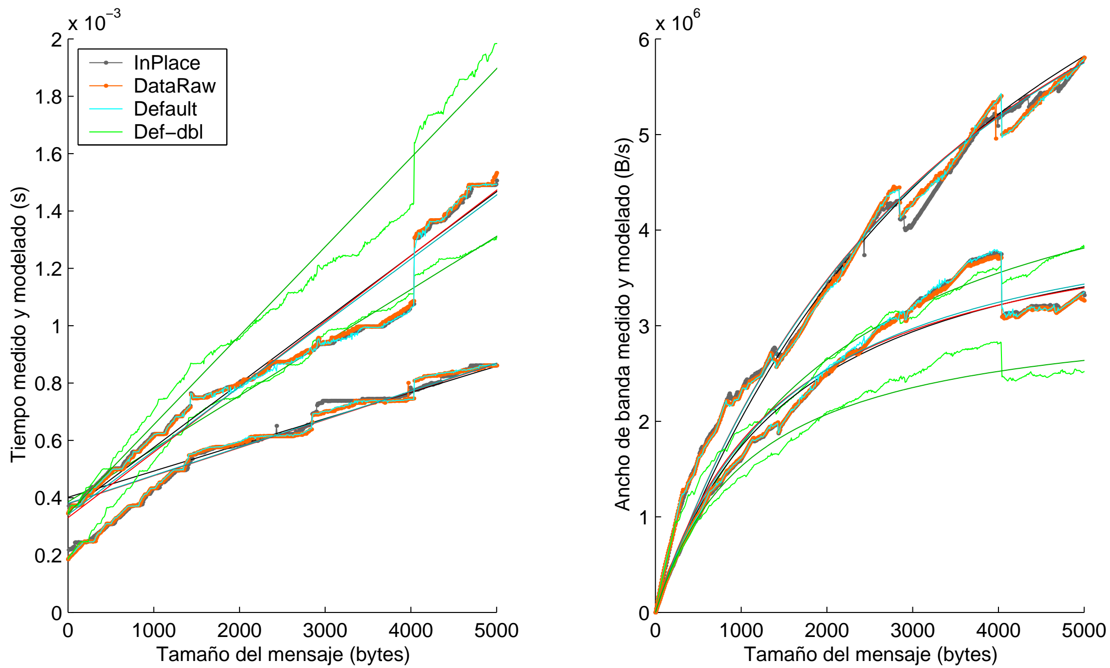
(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.21: Modelo 0, configuración PVM coincidente con LAM (UDPMAXLEN 8K).

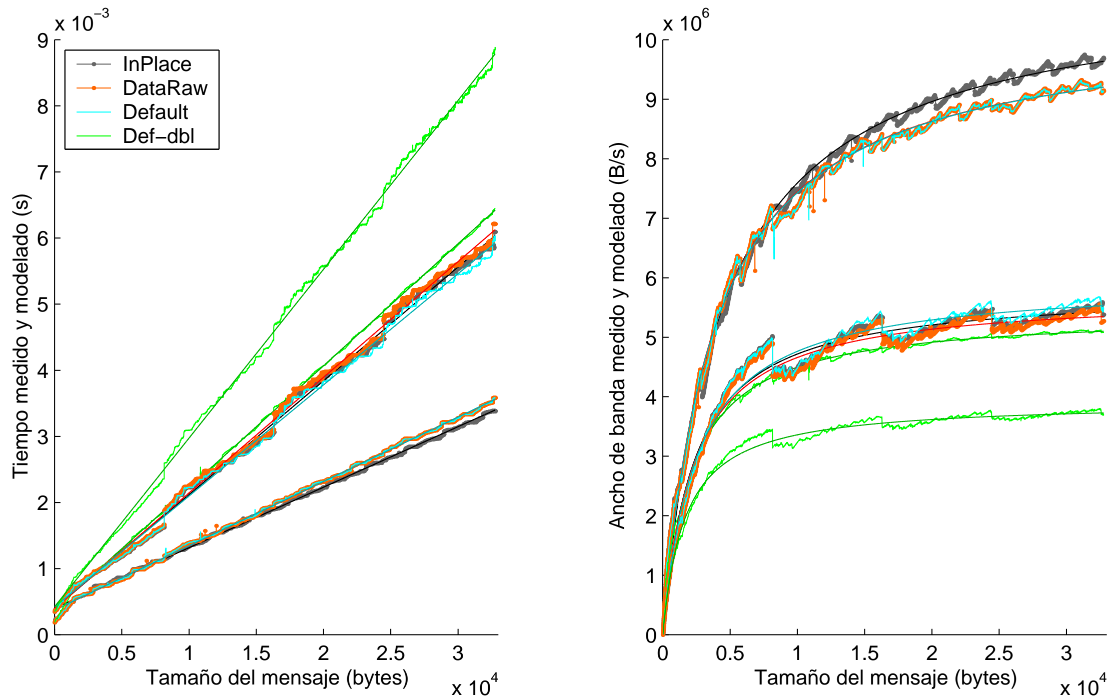


(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.

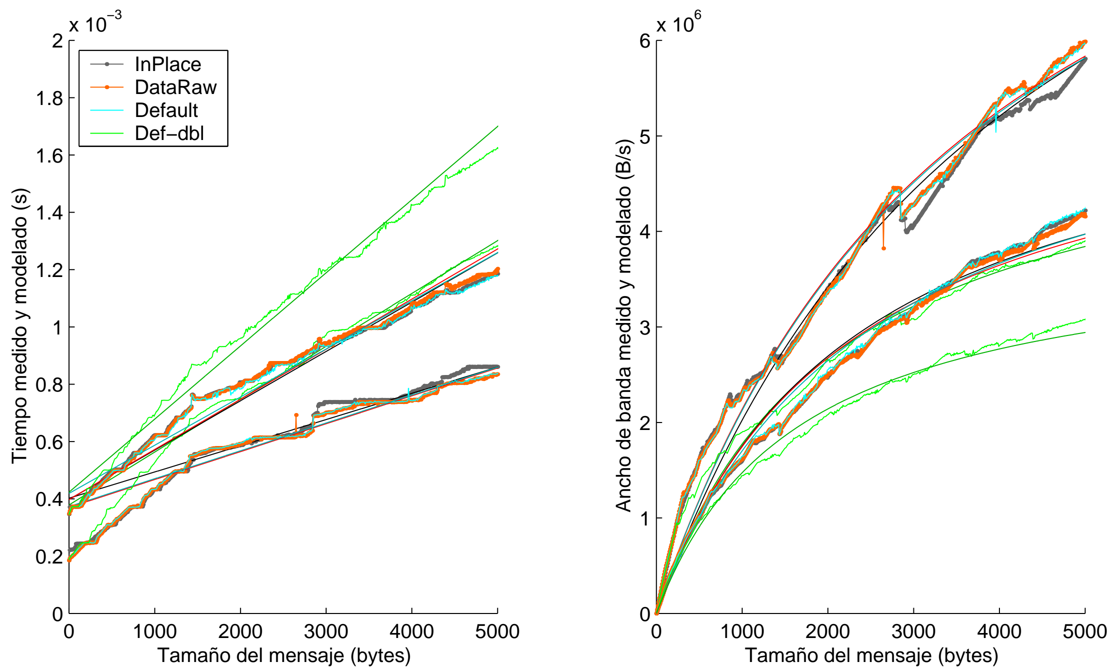


(b) Ampliación al rango de 5KB.

Figura 2.22: Modelo 0, Configuración estándar PVM UDPMAXLEN=4K: Agrupación comparativa de las 6 gráficas de la Figura 2.20.



(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.



(b) Ampliación al rango de 5KB.

Figura 2.23: Modelo 0, Configuración PVM coincidente con LAM UDPMAXLEN=8K: Agrupación comparativa de las 6 gráficas de la Figura 2.21.

Encaminamiento →	**PvmRouteDirect**			**PvmDontRoute**		
Empaquetamiento ↓	L (μs)	B (MB/s)	err	L (μs)	B (MB/s)	err
PvmDataInPlace	401.598	10.934	4.1123E-06	352.563	4.485	2.6297E-05
PvmDataRaw	378.916	10.225	3.2794E-06	330.271	4.378	2.8827E-05
PvmDataDefault	380.272	10.218	3.2880E-06	341.186	4.487	3.9582E-05
Default, double	385.476	5.400	3.1871E-06	344.283	3.222	3.9605E-05

(a) Configuración estándar (UDPMAXLEN 4K)

Encaminamiento →	**PvmRouteDirect**			**PvmDontRoute**		
Empaquetamiento ↓	L (μs)	B (MB/s)	err	L (μs)	B (MB/s)	err
PvmDataInPlace	402.431	10.931	4.1020E-06	397.915	5.808	2.9108E-05
PvmDataRaw	370.192	10.273	3.8665E-06	397.710	5.719	3.1528E-05
PvmDataDefault	373.162	10.280	3.9305E-06	419.213	5.951	3.0174E-05
Default, double	378.836	5.420	3.7315E-06	423.072	3.917	2.8365E-05

(b) Configuración coincidente LAM (UDPMAXLEN 8K)

Tabla 2.4: Parámetros del Modelo 0 ajustados a las diversas configuraciones de PVM.

	RouteDirect	**DontRoute**
DataInPlace	3.394	1.392
DataRaw	3.173	1.359
DataDefault	3.171	1.393
ídem, double	1.676	1.000

(a) Configuración estándar (UDPMAXLEN 4K)

	RouteDirect	**DontRoute**
DataInPlace	3.393	1.803
DataRaw	3.188	1.775
DataDefault	3.191	1.847
ídem, double	1.682	1.216

(b) Configuración coincidente (UDPMAXLEN 8K)

Tabla 2.5: Modelo 0: Anchos de Banda de las diversas configuraciones PVM, normalizados respecto al mínimo alcanzado con la configuración udp4K, PvmDataDefault, PvmDontRoute, **double**s.

	RouteDirect	**DontRoute**
DataInPlace	1.000	2.438
DataRaw	1.069	2.497
DataDefault	1.070	2.437
ídem, double	2.025	3.394

(a) Configuración estándar (UDPMAXLEN 4K)

	RouteDirect	**DontRoute**
DataInPlace	1.000	1.883
DataRaw	1.064	1.912
DataDefault	1.064	1.837
ídem, double	2.017	2.791

(b) Configuración coincidente (UDPMAXLEN 8K)

Tabla 2.6: Modelo 0: Inversos de los Anchos de Banda normalizados respecto al mínimo inverso de la configuración udp4K, PvmDataInPlace, PvmRouteDirect.

Con ruta directa, el empaquetamiento Default para **char** es casi idéntico a DataRaw, y para **double** el ancho de banda se reduce casi a la mitad: de 10 a 5MB/s en la Tabla 2.4 y de 3.2 a 1.7 (0.5 puntos) en la Tabla 2.5, relación prácticamente independiente de la configuración UDPMAXLEN. Con ruta a través del *daemon* la situación para **double** apenas cambia, bajando de 4.49 a 3.22MB/s con udp4K ($1.4 - 1.0 = 0.4$ puntos), y de 5.95 a 3.92MB/s con udp8K ($1.8 - 1.2 = 0.6$ puntos). Para Default/**char** cambia drásticamente: el ancho de banda supera incluso al de InPlace en ambas configuraciones UDPMAXLEN.

Todas las relaciones comentadas pueden observarse en la gráfica superior derecha de la Figura 2.22. Con ruta directa, el empaquetamiento InPlace (negro) es óptimo, seguido de cerca por Raw (rojo) y Default/**char** (azul). A mitad de altura está Default/**double** (verde). Con ruta *daemon* PVM, Default/**char** se adelanta ligerísimamente a InPlace y Raw (azul, negro, rojo, respectivamente), siendo las tres opciones muy similares. Default/**double** (verde) queda a 3/4 de altura.

En la Figura 2.22(b) se observa una inversión: por un momento (rango 3–4KB) los encaminamientos *daemon* consiguen superar al empaquetamiento Default/**double** directo. Tal vez los trazos del modelo colaboren en abarrotar la gráfica, siendo un motivo adicional para preferir el barrido de 32KB escogido.

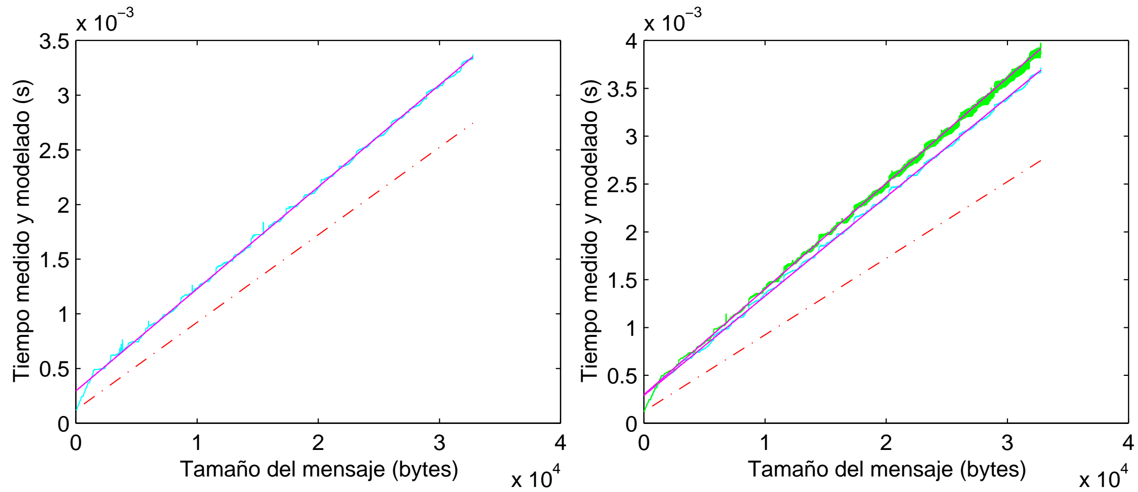
En la Figura 2.23 se produce una inversión más significativa, siendo los encaminamientos *daemon* consistentemente superiores al empaquetamiento Default/**double** directo (a partir de unos 2500 bytes, como se observa en la ampliación a 5KB). La pérdida de prestaciones por codificación XDR **double** es superior a la pérdida por fragmentación con UDPMAXLEN=8K, pero cuando se usan fragmentos de 4KB son los costes de fragmentación los que superan a la codificación XDR **double**. Como veremos en el Capítulo 4, LAM fragmenta por defecto con un tamaño MAXNMSGLEN=8K, presentando esta misma inversión (modo homogéneo −O más eficiente que modo cliente-a-cliente −c2c) para tamaños de mensaje a partir de 200KB.

Resumiendo, el encaminamiento directo es el factor que más afecta a las prestaciones de PVM. A través del *daemon*, el ancho de banda baja a menos de la mitad, efecto que puede ser paliado incrementando el tamaño del fragmento. El coste de codificación XDR depende del tamaño del tipo básico transmitido; en nuestro cluster, para **double** el ancho de banda baja a la mitad con ruta directa y a 2/3–3/4 con *daemon* PVM, según el tamaño del fragmento. Si el fragmento UDP es lo suficientemente extenso, el coste del encaminamiento *daemon* puede volverse inferior al de la codificación XDR.

Parámetros extraidos bajo LAM/MPI

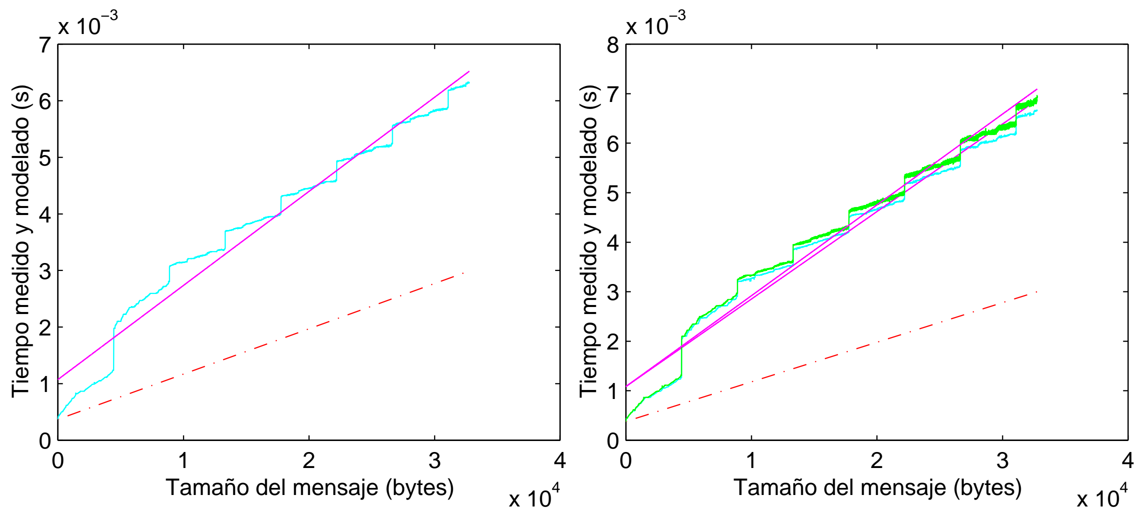
El seguimiento sistemático del mismo esquema de presentación nos ahorrará en lo sucesivo las prolijas explicaciones requeridas para presentar los datos. Los ficheros de mínimos para cada una de las 4 combinaciones de opciones LAM, junto con los trazos ajustados por el Modelo 0, se presentan en las Figuras 2.24 (configuración coincidente con PVM) y 2.25 (configuración estándar LAM). Las gráficas agrupadas se muestran en las Figuras 2.26 y 2.27.

La Tabla 2.7 resume los valores de parámetros extraidos de los correspondientes 8 ficheros LAM de mínimos de medias. Dado que con la opción de cluster heterogéneo se debe aplicar el modelo tanto a datos **char** como **double**, la tabla presenta un total de 12 entradas. Los anchos de banda normalizados respecto al mínimo se relacionan en la Tabla 2.8, y sus inversos normalizados respecto al máximo en la 2.9.



(a) Homogéneo, Cliente a cliente

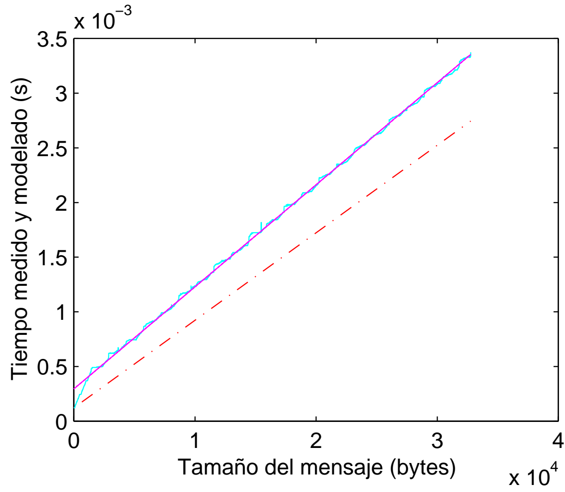
(b) Heterogéneo, Cliente a cliente



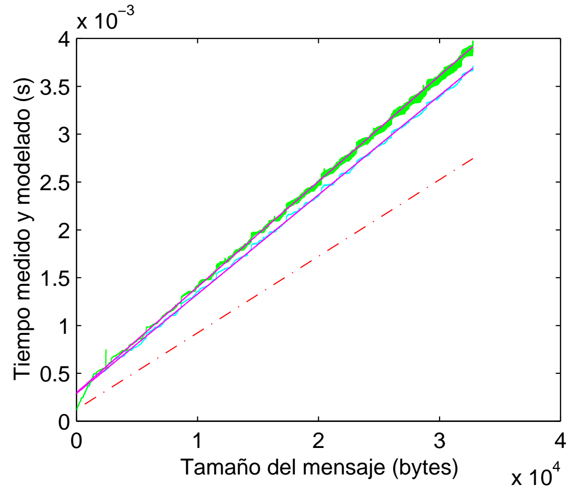
(c) Homogéneo, *daemon* LAM

(d) Heterogéneo, *daemon* LAM

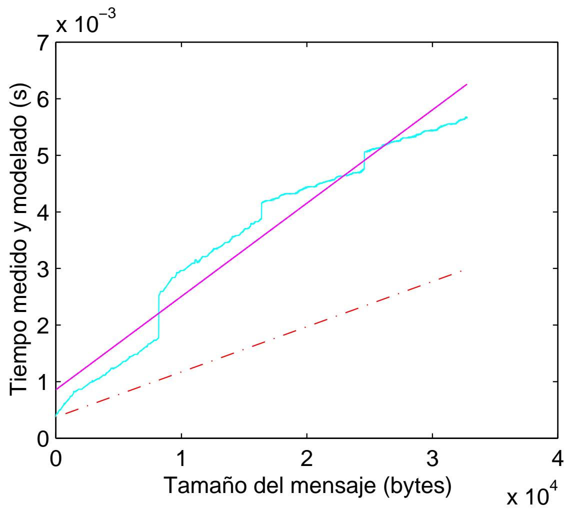
Figura 2.24: Modelo 0, configuración LAM coincidente con PVM MAXNMSGLEN=4K.



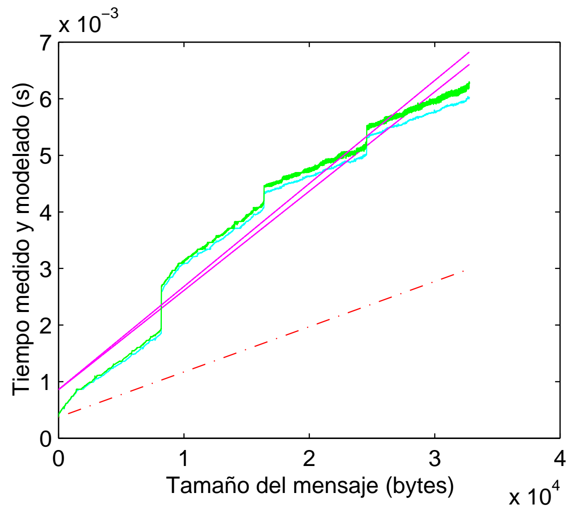
(a) Homogéneo, Cliente a cliente



(b) Heterogéneo, Cliente a cliente

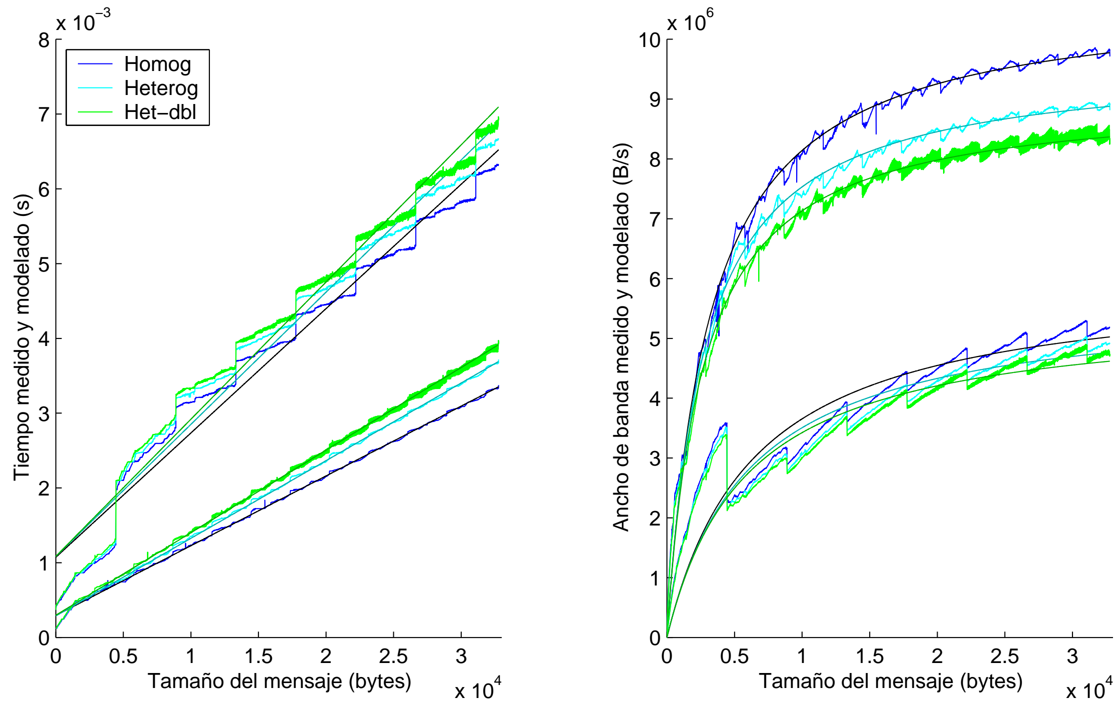


(c) Homogéneo, *daemon* LAM

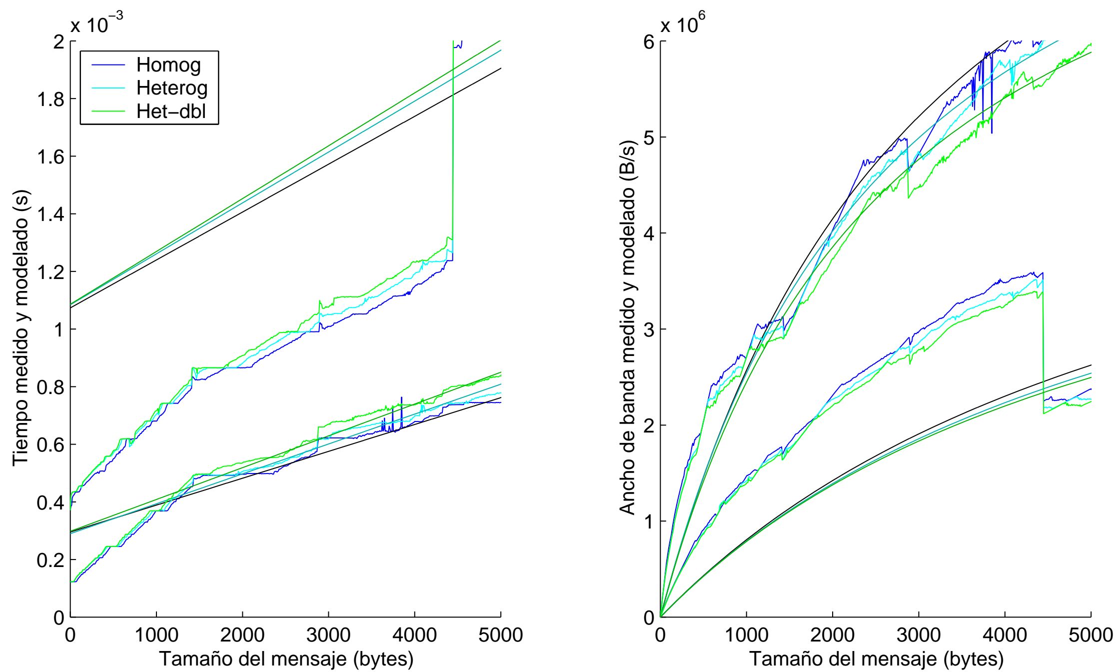


(d) Heterogéneo, *daemon* LAM

Figura 2.25: Modelo 0, configuración estándar LAM MAXNMSGLEN=8K.

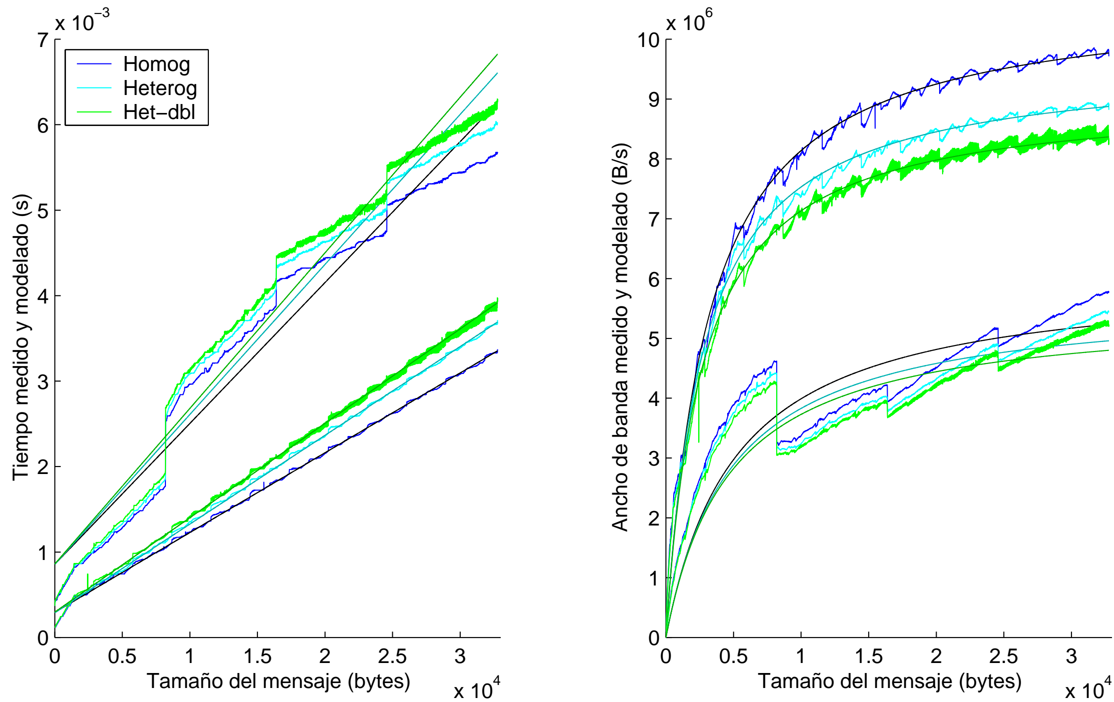


(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.

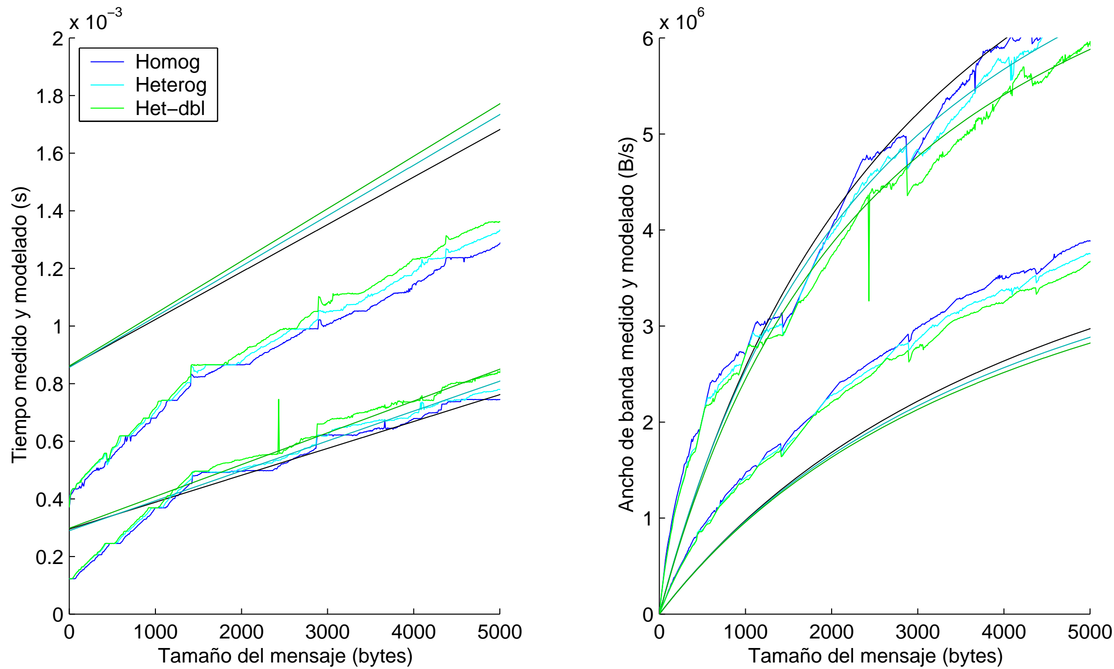


(b) Ampliación al rango de 5KB.

Figura 2.26: Modelo 0, Configuración LAM coincidente con PVM MAXNMSGLEN=4K: Agrupación comparativa de las 4 gráficas de la Figura 2.24.



(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.



(b) Ampliación al rango de 5KB.

Figura 2.27: Modelo 0, Configuración LAM estándar MAXNMSGLEN=8K: Agrupación comparativa de las 4 gráficas de la Figura 2.25.

Encaminamiento →	cliente-a-cliente			*daemon* LAM		
Cluster ↓	L (μs)	B (MB/s)	err	L (μs)	B (MB/s)	err
Homogéneo	295.636	10.720	3.0023E-06	1073.537	6.015	3.5327E-04
Heterogéneo	289.768	9.635	2.9506E-06	1084.362	5.659	3.6478E-04
Het, double	298.053	9.057	4.7717E-06	1085.876	5.453	3.6345E-04

(a) Configuración coincidente PVM (MAXNMSGLEN 4K)

Encaminamiento →	cliente-a-cliente			*daemon* LAM		
Cluster ↓	L (μs)	B (MB/s)	err	L (μs)	B (MB/s)	err
Homogéneo	295.333	10.713	2.9890E-06	857.969	6.068	5.0394E-04
Heterogéneo	289.751	9.632	2.8805E-06	856.604	5.699	5.0780E-04
Het, double	298.146	9.057	4.7240E-06	861.245	5.494	4.9916E-04

(b) Configuración estándar LAM (MAXNMSGLEN 8K)

Tabla 2.7: Parámetros del Modelo 0 ajustados a las diversas configuraciones de LAM.

	-c2c	**-lamd**
Homogéneo	1.966	1.103
Heterogéneo	1.767	1.038
ídem, double	1.661	1.000

(a) Configuración coincidente (MAXNMSGLEN 4K)

	-c2c	**-lamd**
Homogéneo	1.965	1.113
Heterogéneo	1.766	1.045
ídem, double	1.661	1.008

(b) Configuración estándar (MAXNMSGLEN 8K)

Tabla 2.8: Modelo 0: Anchos de Banda de las diversas configuraciones LAM, normalizados respecto al mínimo alcanzado con la configuración maxn4K, $-$lamd, Heterogéno, **double**s.

	-c2c	**-lamd**
Homogéneo	1.000	1.782
Heterogéneo	1.113	1.894
ídem, double	1.184	1.966

(a) Configuración coincidente (MAXNMSGLEN 4K)

	-c2c	**-lamd**
Homogéneo	1.001	1.767
Heterogéneo	1.113	1.881
ídem, double	1.184	1.951

(b) Configuración estándar (MAXNMSGLEN 8K)

Tabla 2.9: Modelo 0: Inversos de los Anchos de Banda normalizados respecto al mínimo inverso de la configuración udp4K, $-$c2c, $-$O.

Se repite para LAM el previsible comportamiento de las latencias, ya comentado al aplicar el Modelo 0 a PVM, aunque en mucha menor medida: las latencias estimadas para cluster homogéneo son ligerísimamente mayores que para heterogéneo (Tabla 2.7). Bajo PVM (Tabla 2.4) la diferencia era mucho mayor, haciendo aparecer incluso las latencias para ruta directa como mayores que las del *daemon* PVM. Esto no sucede para LAM debido al fuerte escalón entre los dos primeros fragmentos UDP: comparar las Figuras 2.26 y 2.22 izquierda (o la 2.27 con la 2.23).

El máximo ancho de banda alcanzado en los primeros 32KB es algo menor que el alcanzado con PVM. En las decenas de KBs "PVM gana a LAM/MPI", como veremos de nuevo en los barridos de 32MB de los Capítulos 3 y 4. La frase la entrecomillamos porque la correspondiente pregunta (*"¿cuál es mejor?"*) es ávidamente repetida y no tiene respuesta única: depende no sólo del tamaño de mensaje utilizado sino de multitud de detalles de la codificación concreta de la aplicación paralela.

El modo cliente-a-cliente presenta una independencia impecable respecto al tamaño de fragmento UDP, mostrando una variabilidad en los respectivos parámetros perfectamente atribuible a la propia variabilidad de las mediciones. Paradójicamente, el ancho de banda en modo *daemon* también varía muy poco, información difícilmente previsible observando las Figuras 2.26 y 2.27, en donde tal vez destaca más la diferencia de latencias. El modo *daemon* PVM (Tabla 2.4) presentaba una fuerte dependencia respecto al tamaño de fragmento UDP.

El modo *daemon* LAM reduce el ancho de banda a algo más de la mitad. La diferencia entre cluster homogéneo/heterogéneo mantiene la misma proporción: alrededor de 1MB/s cliente-a-cliente, reducido a unos 0.5MB/s en modo *daemon*. La penalización por datos **double** es mucho más reducida que bajo PVM (0.6MB/s cliente-a-cliente, 0.2MB/s en modo *daemon*), lo cual indica una conversión XDR mucho más eficiente en LAM.

Trasladando a las gráficas estos comentarios realizados sobre las tablas, lo más destacado visualmente en la parte izquierda de las Figuras 2.26 y 2.22 es la elevada sobreestimación de la latencia para el modo *daemon*. El pronunciado salto entre los dos primeros fragmentos UDP hace el Modelo 0 muy inapropiado para este modalidad LAM, produciendo un error cuadrático medio del orden de 10^{-4} (Tabla 2.7), un orden de magnitud superior al obtenido bajo PVM (Tabla 2.4).

Las partes derechas de las Figuras 2.26 y 2.22 quedan más claramente separadas en dos grupos, el modo *daemon* por debajo. La reducida penalización por codificación XDR coopera en el ordenado aspecto de las gráficas en ambas configuraciones MAXNMSGLEN. La opción de cluster homogéneo es sistemáticamente superior (trazo azul oscuro), y la diferencia entre datos **char** y **double** (azul/verde) en modo *daemon* es reducida. Como comentamos anteriormente, en el barrido a 32MB del Capítulo 4 veremos que a partir de unos 200KB se produce una inversión de costes, resultando el modo homogéneo *daemon* (–O –lamd) más eficiente que el modo heterogéneo cliente-a-cliente (–c2c). Esto indica que gran parte del coste del *daemon* puede quedar oculto por el tiempo de transmisión, aunque naturalmente los dos primeros fragmentos UDP retrasan bastante la recuperación del ancho de banda en comparación con la codificación XDR de datos **double** en modo cliente-a-cliente.

El siguiente modelo intentará reducir el error cuadrático medio añadiendo parámetros para ajustarse mejor a la primera MTU, corrigiendo por tanto la sobrestimación de la latencia y la infraestimación del ancho de banda, al menos para las modalidades de encaminamiento directo.

2.3.5 Modelo 1

Este modelo intenta mejorar el anterior ajustando con mayor precisión el primer tramo de la gráfica, correspondiente a la primera MTU. Se introduce para ello un parámetro H que indica el tamaño de cabecera (*header*) de la primera MTU. El tamaño de datos de la primera MTU es por tanto $\|MTU\| = 1500 - H$. Dicho parámetro varía según la combinación de opciones PVM. La conveniencia de expresar los diferentes tamaños de MTU mediante el tamaño de cabecera, en lugar de directamente, así como el motivo de descomponer $H = h1 + s1$ en dos parámetros, se explicarán más adelante. El modelo distingue por tanto dos casos en el cálculo del tiempo de transmisión T_S para un mensaje de tamaño S:

$$T_S = \begin{cases} lc + \frac{S}{bc} & S \leq \|MTU\| \\ lc + \frac{\|MTU\|}{bc} + ls + \frac{S - \|MTU\|}{bs} & S > \|MTU\| \end{cases}$$

Ambos tramos de la gráfica se aproximan como en el Modelo 0, mediante una latencia y un ancho de banda. La pronunciada pendiente inicial se modela con los parámetros lc y bc (la letra c es por *call*, llamada), hasta llegar a un tamaño de mensaje de $\|MTU\|$ =MTU1HDR= $1500 - H$ bytes, que tarda en transmitirse $T_{\|MTU\|} = lc + \|MTU\|/bc$. El tramo restante se modela con otros dos parámetros ls y bs. La letra s es por *switch*, aunque los parámetros del modelo son sintéticos, careciendo en principio de interpretación física.

Se pretende que el tiempo de transmisión de las dos primeras cabeceras quede incluido en las latencias lc y ls, respectivamente, mientras que las demás vayan absorbidas por el ancho de banda bs. En este modelo se puede identificar uno de los parámetros latencia, lc, con el *tiempo de transmisión de mensaje nulo*, T_0. El parámetro ls permite variar la altura desde donde arranca el segundo tramo de la gráfica, solucionando el comentado problema de infraestimación de bs (Figura 2.19). Como veremos, ls no representa fiablemente el tiempo de transmisión de la segunda cabecera, quedando reducido a un mero parámetro *sintético*.

Para un mensaje de tamaño $1500 - H + s$, el tiempo de transmisión sería $T_{\|MTU\|+s} = T_{\|MTU\|} + ls + s/bs$. En MATLAB esto se expresa compactamente como:

```
MTU       = 1500;
MTU1HDR = MTU-h1-s1 ;           %% MTU - cabecera : tamaño datos MTUs
  below  = ( s<=MTU1HDR);
  above  =       ~below ;
t ( below )=  lc +s ( below )/ bc ;
t ( above )=  lc + MTU1HDR/ bc +  ls +( s ( above )-MTU1HDR)/ bs ;
```

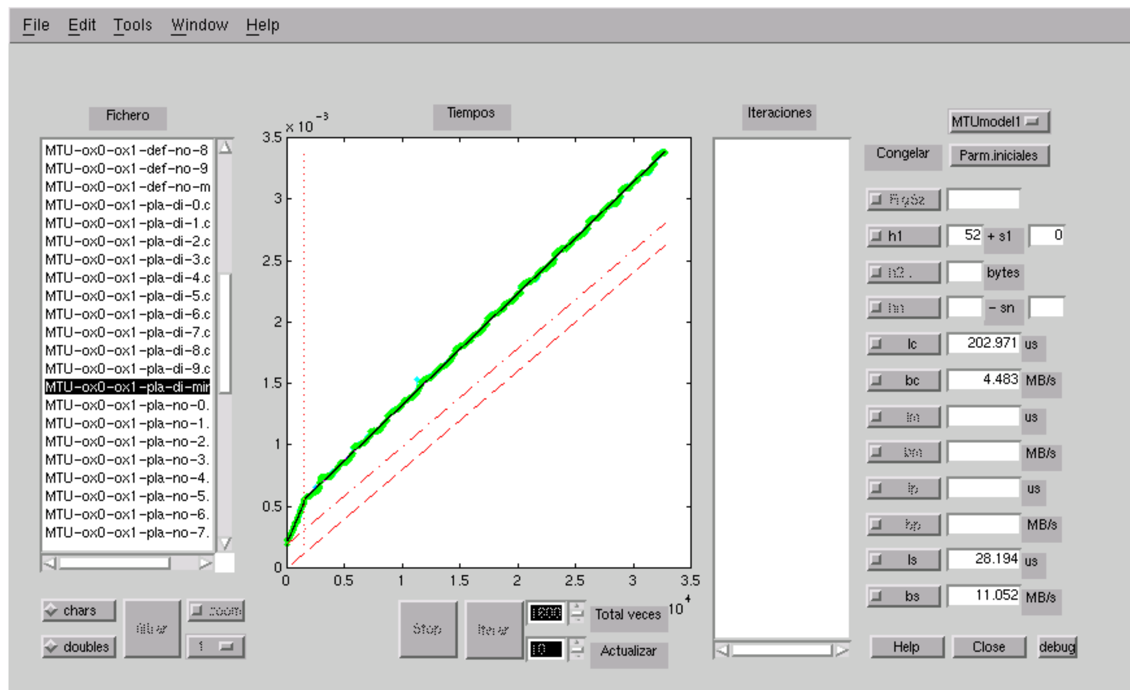
Listado 2.7: Código MATLAB para evaluar el Modelo 1 sobre un vector *s* de tamaños de mensaje.

El código MATLAB completo utilizado para evaluar el modelo se presenta en el Apéndice B.3.

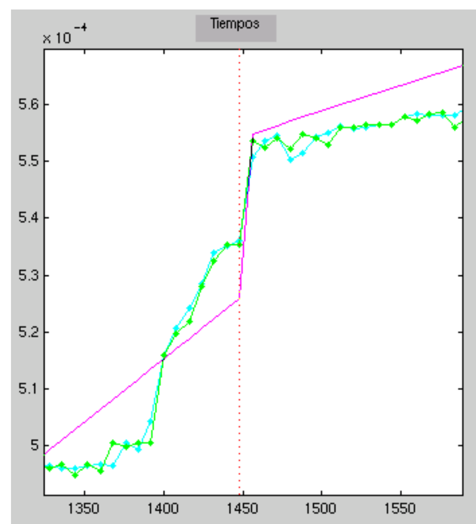
Para determinar el valor de H se utilizaron las casillas de edición h1 y s1 del GUI MATLAB, haciendo *zoom* alrededor de la región de 1500 bytes, como se muestra en la Figura 2.28. Una vez determinado dicho valor, las casillas se "congelan" para que su valor no varíe durante la evolución del método simplex.

Parámetros extraidos bajo PVM

Los ficheros de mínimos para cada una de las 6 combinaciones de opciones PVM, junto con los trazos ajustados por el Modelo 1, se presentan en las Figuras 2.29 (configuración estándar PVM) y 2.30 (configuración coincidente con LAM).

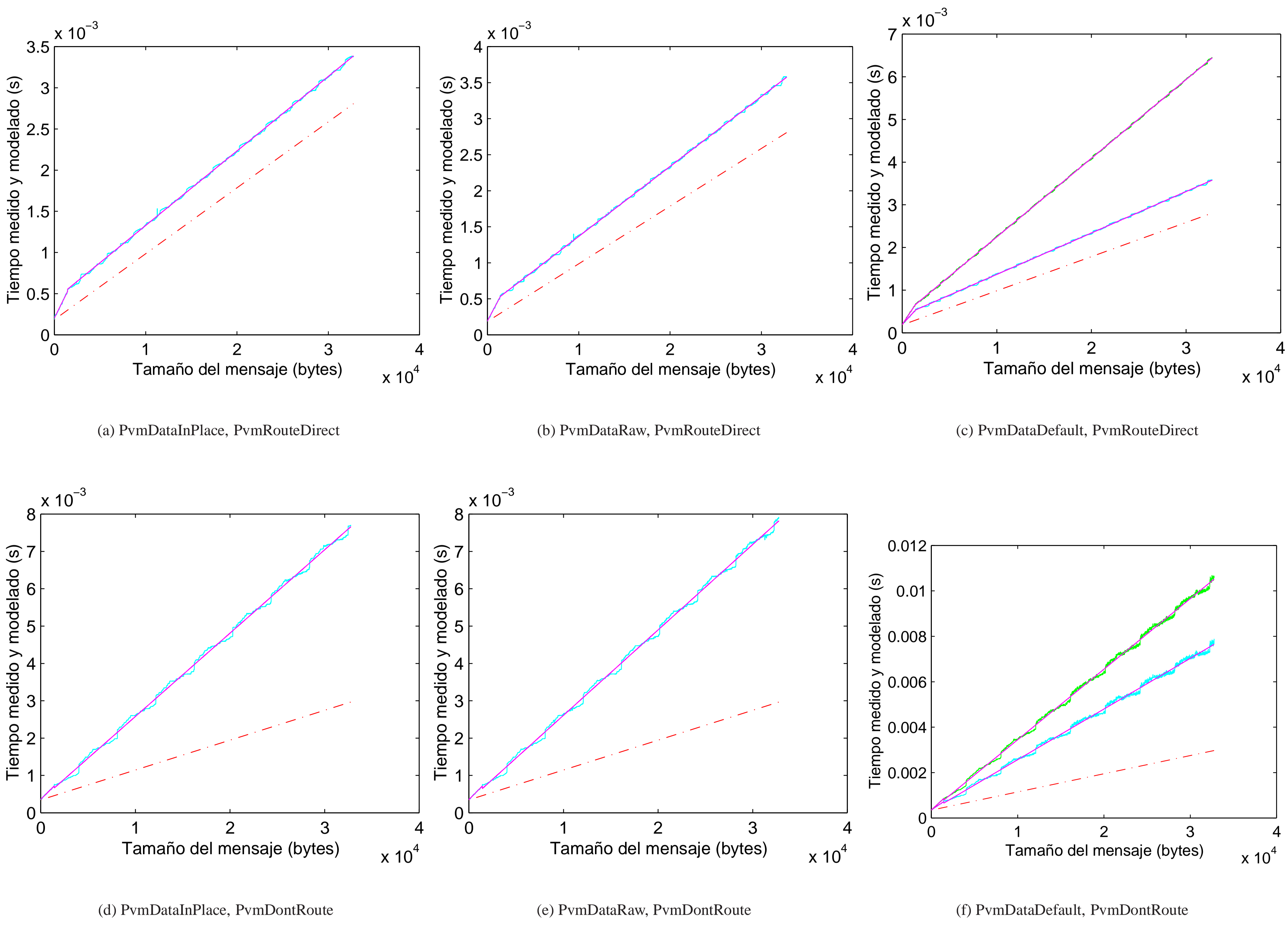


(a) Observar las casillas de parámetros h1 y s1 activadas, así como la marca vertical indicando la frontera entre tramos. Los parámetros pueden editarse hasta encontrar el valor exacto.



(b) El GUI permite también realizar *zoom* para afinar el valor hasta el byte exacto.

Figura 2.28: El modelo a dos tramos (Modelo 1) requiere estimar la frontera entre ambos tramos (parámetros h1/s1). Esto se consigue gráficamente mediante el GUI desarrollado.



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

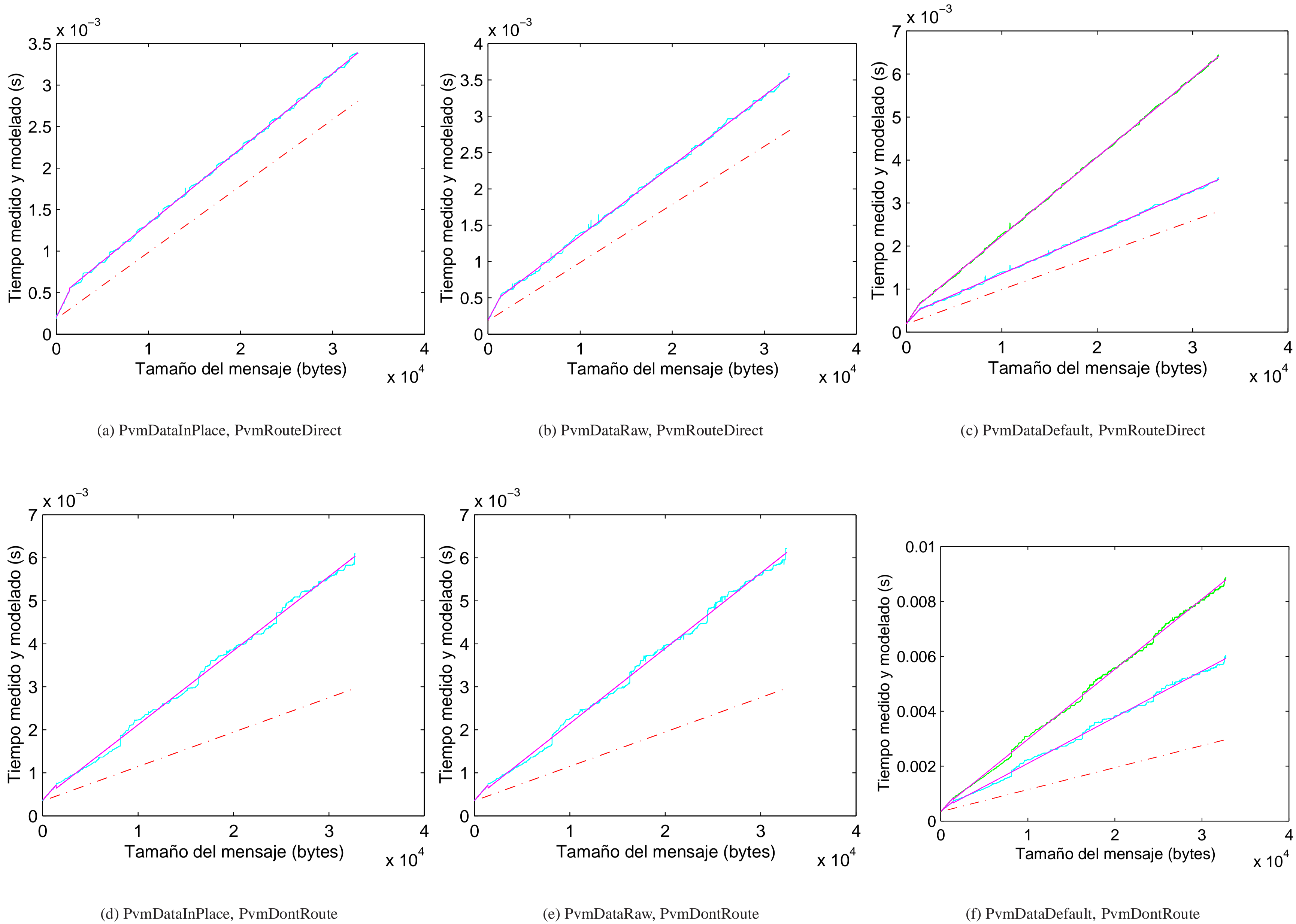
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.29: Modelo 1, configuración estándar PVM (UDPMAXLEN 4K).



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

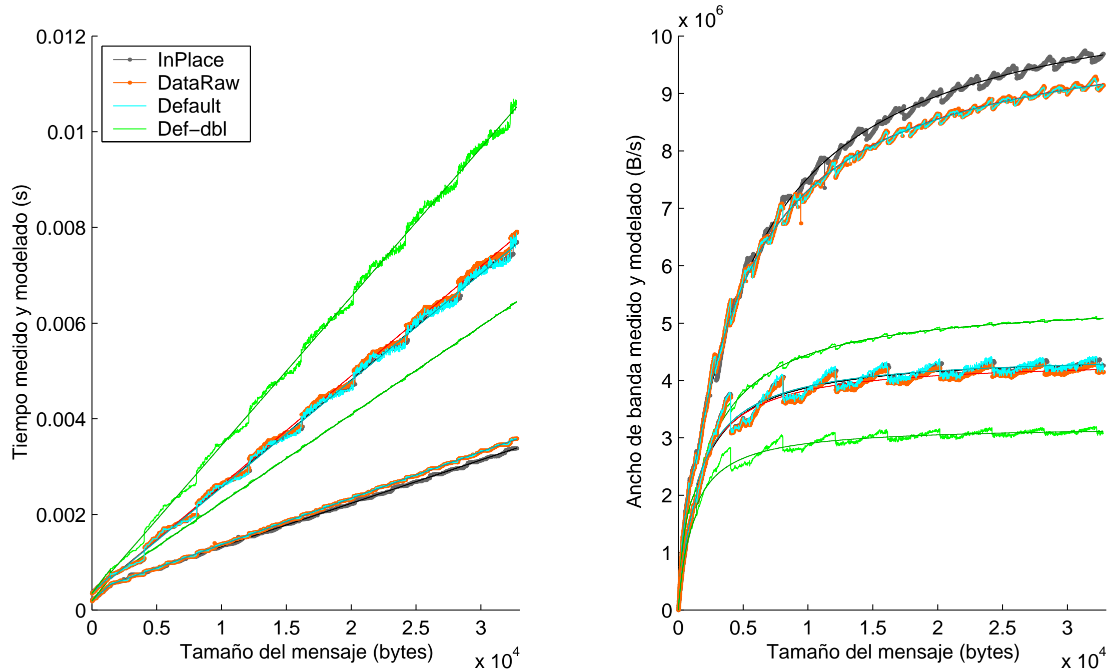
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

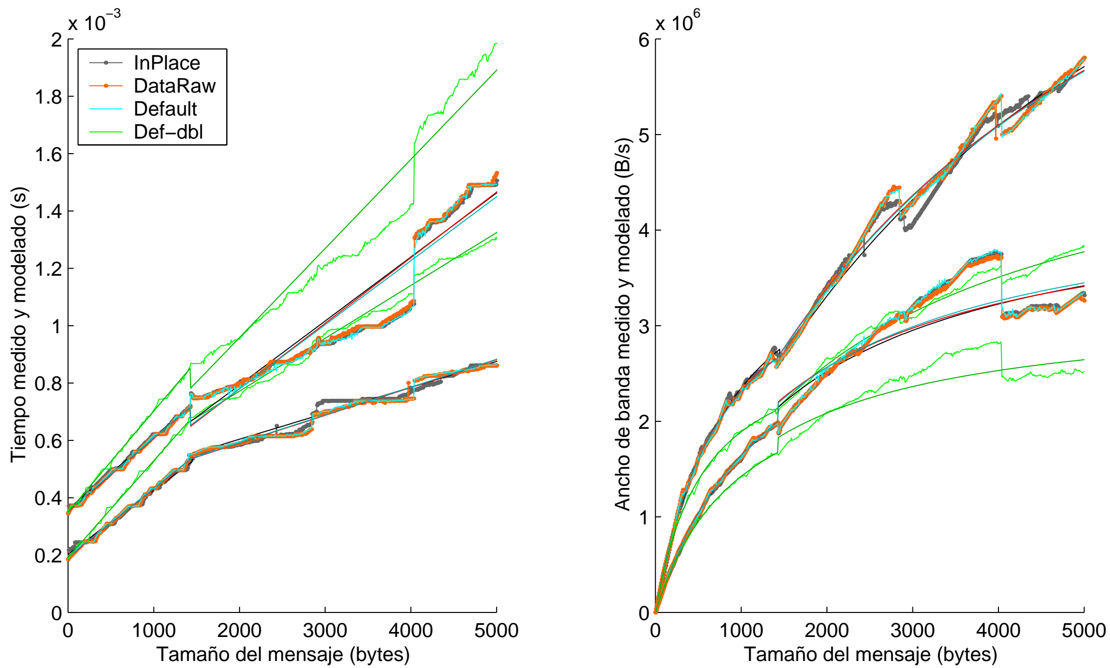
(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.30: Modelo 1, configuración coincidente con LAM (UDPMAXLEN 8K).

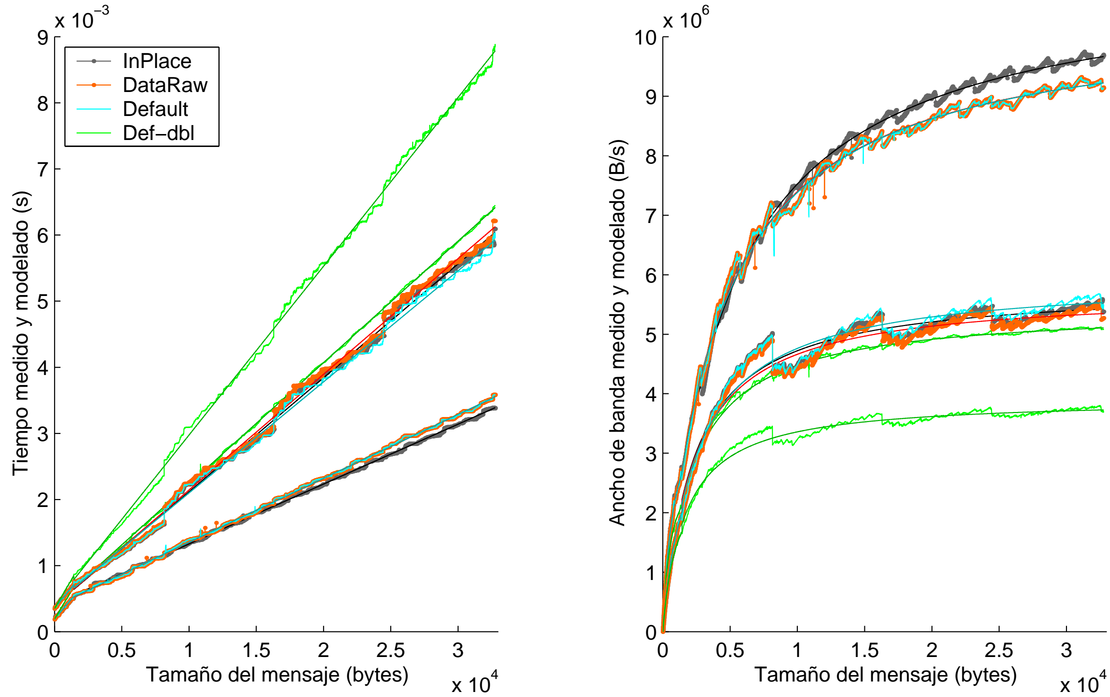


(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.

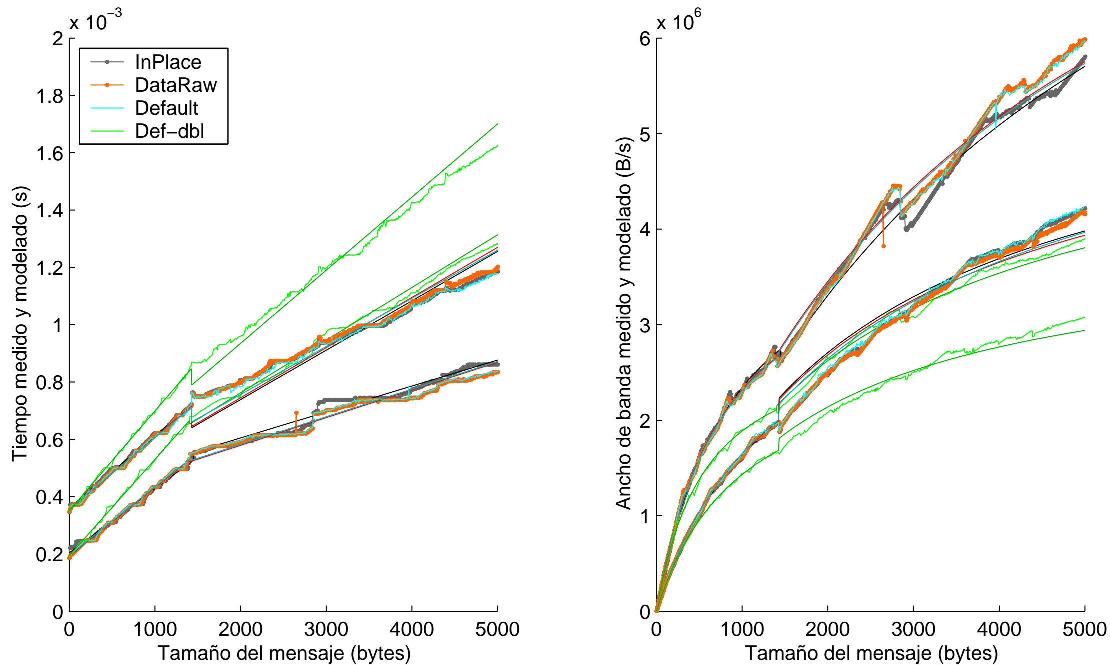


(b) Ampliación al rango de 5KB.

Figura 2.31: Modelo 1, Configuración estándar PVM UDPMAXLEN=4K: Agrupación comparativa de las 6 gráficas de la Figura 2.29.



(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.



(b) Ampliación al rango de 5KB.

Figura 2.32: Modelo 1, Configuración PVM coincidente con LAM UDPMAXLEN=8K: Agrupación comparativa de las 6 gráficas de la Figura 2.30.

			PvmRouteDirect			
Empaquetamiento	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
DataInPlace	52	202.971	4.483	28.194	11.052	1.2157E-06
DataRaw	100	191.926	4.172	4.792	10.309	1.0756E-06
DataDefault	100	193.614	4.173	4.643	10.302	1.0935E-06
ídem, double	100	192.395	2.970	-2.663	5.424	9.2408E-07
			PvmDontRoute			
Empaquetamiento	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
DataInPlace	76	360.608	3.979	-53.786	4.480	2.6102E-05
DataRaw	76	351.573	3.919	-67.021	4.371	2.8457E-05
DataDefault	76	355.052	3.934	-65.248	4.480	3.9272E-05
ídem, double	76	350.050	2.839	-72.002	3.218	3.9282E-05

(a) Configuración PVM estándar (UDPMAXLEN 4K)

			PvmRouteDirect			
Empaquetamiento	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
DataInPlace	52	203.604	4.473	27.621	11.049	1.2087E-06
DataRaw	100	192.718	4.184	-6.128	10.349	1.9909E-06
DataDefault	100	195.103	4.191	-4.901	10.357	2.0303E-06
ídem, double	100	194.244	2.978	-12.300	5.441	1.7600E-06
			PvmDontRoute			
Empaquetamiento	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
DataInPlace	76	361.086	3.972	-80.117	5.802	2.8831E-05
DataRaw	76	353.720	3.931	-71.520	5.716	3.1301E-05
DataDefault	76	356.711	4.017	-51.632	5.953	2.9968E-05
ídem, double	76	351.211	2.877	-58.181	3.918	2.8097E-05

(b) Configuración PVM coincidente con LAM (UDPMAXLEN 8K)

Tabla 2.10: Parámetros del Modelo 1 ajustados a las diversas configuraciones de PVM.

La Figura 2.31 agrupa las 6 subgráficas de la configuración estándar, y la Figura 2.32 las de la coincidente con LAM. Las predicciones del modelo aparecen en un tono algo más oscuro del mismo color que la correspondiente opción PVM.

La Tabla 2.10 resume los valores de parámetros extraidos. Se proporcionan también los valores fijos del parámetro de cabecera H.

Con ruta TCP directa, se observa una fuerte disminución del antiguo parámetro L al nuevo lc, que ahora sí representa fiablemente T_0. A través del *daemon* se observa en general un ligerísimo incremento, apenas observable comparando las ampliaciones a 5KB del Modelo 1 (Figs. 2.31 y 2.32) con las del Modelo 0 (Figs. 2.22 y 2.23).

bc es un parámetro nuevo que no admite comparación con el modelo anterior, y que ajusta la fuerte pendiente del primer tramo según lo explicado. El parámetro bc es mucho menos sensible que bs al encaminamiento y es casi independiente del tamaño UDP.

Conviene notar que el parámetro ls juega en este modelo un papel “bisagra”, al permitir que el segundo tramo de la gráfica comience a cualquier altura, en lugar de forzarlo a coincidir con el tiempo de transmisión de la primera MTU. El método simplex tiene así el suficiente grado de libertad para minimizar el error en ambos tramos independientemente.

ls no representa por tanto ningún valor con sentido físico; no es de extrañar que llegue a tomar valores negativos con encaminamiento a través del *daemon*, cuando el primer tramo de la gráfica está siendo correctamente modelado para acabar a mayor altura mientras que la pendiente general seguida por el segundo tramo de la gráfica apunta a T_0. Los pequeños valores negativos de ls con ruta directa son perfectamente asumibles, e indican que el parámetro es superfluo en esta modalidad y se podría congelar a 0; su magnitud es a menudo inferior a la propia resolución de gettimeofday().

bs crece algo respecto al antiguo B con ruta directa, tal y como se pretendía, corrigiendo la infraestimación de B. También disminuye ligeramente para casi todos los empaquetamientos a través del *daemon*, dado que la elevada pendiente de la primera MTU, que se repite a cada fragmento UDP, está siendo separadamente modelada para el primer fragmento, subiendo el punto del cual arranca la recta modelada por ls y bs. Para el modo *daemon*, el Modelo 1 es muy poco superior al Modelo 0, como queda reflejado en los respectivos errores cuadráticos de las Tablas 2.10 y 2.4.

Salvo en las ampliaciones a 5KB, es difícil observar gráficamente las mejoras del modelado. La diferencia más visible es el mejor ajuste de la primera MTU (comparar Figuras 2.29(a) y 2.20(a)). La superioridad del modelo sólo puede manifestarse objetivamente mediante una comparación de los errores cuadráticos medios, ofrecida en la Tabla 2.11. Este modelo ha conseguido reducir a más de la tercera parte (29–33.3%) la magnitud del error cuadrático para el encaminamiento directo, debido a la mejor aproximación del primer tramo.

Bajo encaminamiento PVM el esfuerzo adicional de modelado produce una mejora puramente anecdótica alrededor del 1%. La estructura mostrada por esta modalidad es más compleja que los dos simples tramos del Modelo 1. El siguiente modelo deberá incorporar un mayor número de parámetros para poder adaptarse a esta complejidad y reducir el error en el segundo tramo.

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	$1.2157\ 10^{-6}$	$2.6102\ 10^{-5}$	$1.2087\ 10^{-6}$	$2.8831\ 10^{-5}$
PvmDataRaw	$1.0756\ 10^{-6}$	$2.8457\ 10^{-5}$	$1.9909\ 10^{-6}$	$3.1301\ 10^{-5}$
PvmDataDefault	$1.0935\ 10^{-6}$	$3.9272\ 10^{-5}$	$2.0303\ 10^{-6}$	$2.9968\ 10^{-5}$
ídem, double	$9.2408\ 10^{-7}$	$3.9282\ 10^{-5}$	$1.7600\ 10^{-6}$	$2.8097\ 10^{-5}$

(a) Modelo 1

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	$4.1123\ 10^{-6}$	$2.6297\ 10^{-5}$	$4.1020\ 10^{-6}$	$2.9108\ 10^{-5}$
PvmDataRaw	$3.2794\ 10^{-6}$	$2.8827\ 10^{-5}$	$3.8665\ 10^{-6}$	$3.1528\ 10^{-5}$
PvmDataDefault	$3.2880\ 10^{-6}$	$3.9582\ 10^{-5}$	$3.9305\ 10^{-6}$	$3.0174\ 10^{-5}$
ídem, double	$3.1871\ 10^{-6}$	$3.9605\ 10^{-5}$	$3.7315\ 10^{-6}$	$2.8365\ 10^{-5}$

(b) Modelo 0

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	3.383	1.007	3.394	1.010
PvmDataRaw	3.049	1.013	1.942	1.007
PvmDataDefault	3.007	1.008	1.936	1.007
ídem, double	3.449	1.008	2.120	1.010

(c) Mejora relativa: error del Modelo 0 dividido entre el del Modelo 1

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	0.296	0.993	0.295	0.990
PvmDataRaw	0.328	0.987	0.515	0.993
PvmDataDefault	0.333	0.992	0.517	0.993
ídem, double	0.290	0.992	0.472	0.991

(d) Reducción relativa: error del Modelo 1 dividido entre el del Modelo 0

Tabla 2.11: Errores cuadráticos medios de los Modelos 0 y 1 bajo las diversas configuraciones PVM.

Parámetros extraidos bajo LAM/MPI

Los ficheros de mínimos para cada una de las 4 combinaciones de opciones LAM, junto con los trazos ajustados por el Modelo 1, se presentan en las Figuras 2.33 (configuración coincidente con PVM) y 2.34 (configuración estándar LAM). Las gráficas agrupadas se muestran en las Figuras 2.35 y 2.36. La Tabla 2.12 resume los valores de parámetros extraidos. Se proporcionan también los valores fijos del parámetro de cabecera H. La comparación de errores cuadráticos con el Modelo 0 se realiza en la Tabla 2.13.

Los mismos comentarios realizados para PVM son aplicables a LAM/MPI. Se observa una fuerte disminución del antiguo parámetro L al nuevo lc, que ahora sí representa fiablemente T_0. A través del *daemon* la reducción es más significativa, dado que la sobreestimación de L por el Modelo 0 estaba agravada por el fuerte escalón entre los dos primeros fragmentos UDP (Figs. 2.26 y 2.27).

El parámetro bc es mucho menos sensible que bs al encaminamiento. Ambos son casi independientes del tamaño UDP.

ls sigue jugando el mismo papel "bisagra", al permitir que el segundo tramo de la gráfica comience a cualquier altura, en lugar de forzarlo a coincidir con el tiempo de transmisión de la primera MTU. El método simplex tiene así el suficiente grado de libertad para minimizar el error en ambos tramos independientemente. Su elevadísimo valor para el modo *daemon*, claramente comprensible en vista a las Figuras 2.36(a) y 2.36(a), tan sólo abunda en la poca adecuación entre la compleja estructura del modo *daemon* y los simples dos tramos que puede producir este modelo, como queda también reflejado en los órdenes de magnitud para el error cuadrático en la Tabla 2.12.

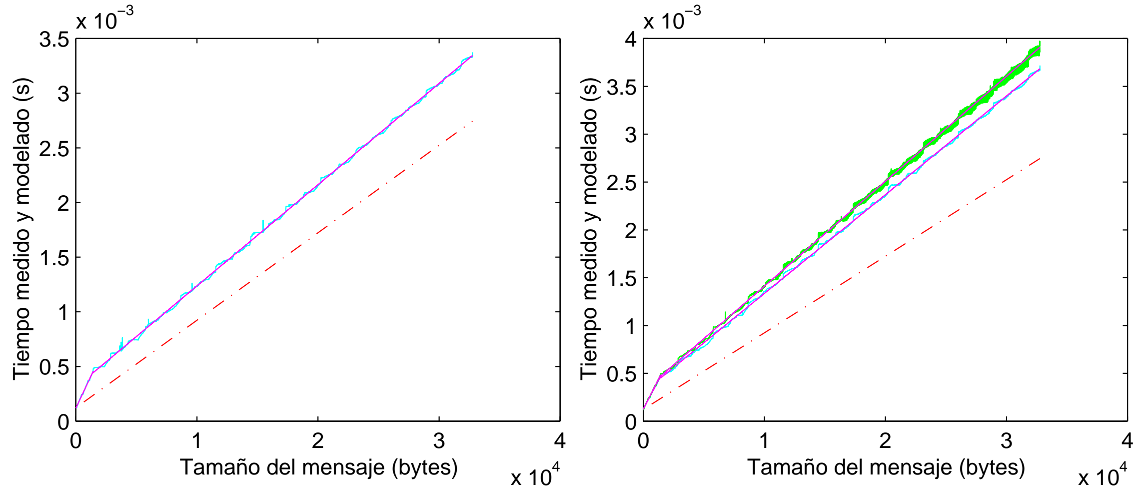
bs crece algo respecto al antiguo B con ruta directa, tal y como se pretendía, corrigiendo la infraestimación de B. A través del *daemon* el crecimiento no se debe a una mejor estimación de la pendiente, sino a la exclusión de la primera MTU (modelada ahora por lc y bc), que eleva el punto de partida del segundo tramo modelado por bs.

Salvo en las ampliaciones a 5KB, es difícil observar gráficamente las mejoras del modelado. La diferencia más visible es el mejor ajuste de la primera MTU en las Figuras 2.35(b) y 2.36(b) (comparar con Figs. 2.26(b) y 2.27(b)). La superioridad del modelo sólo puede manifestarse objetivamente mediante una comparación de los errores cuadráticos medios, ofrecida en la Tabla 2.13. Este modelo consigue reducir más o menos a la mitad (38–63%) la magnitud del error cuadrático para la modalidad cliente-a-cliente, debido a la mejor aproximación del primer tramo.

Bajo el *daemon* LAM la mejora ronda un 20% para MAXNMSGLEN 4K, bajando a un modestísimo 6% para MAXNMSGLEN 8K. La estructura mostrada por esta modalidad es más compleja que los dos simples tramos del Modelo 1. El siguiente modelo deberá incorporar un mayor número de parámetros para poder adaptarse a esta complejidad y reducir el error en el segundo tramo.

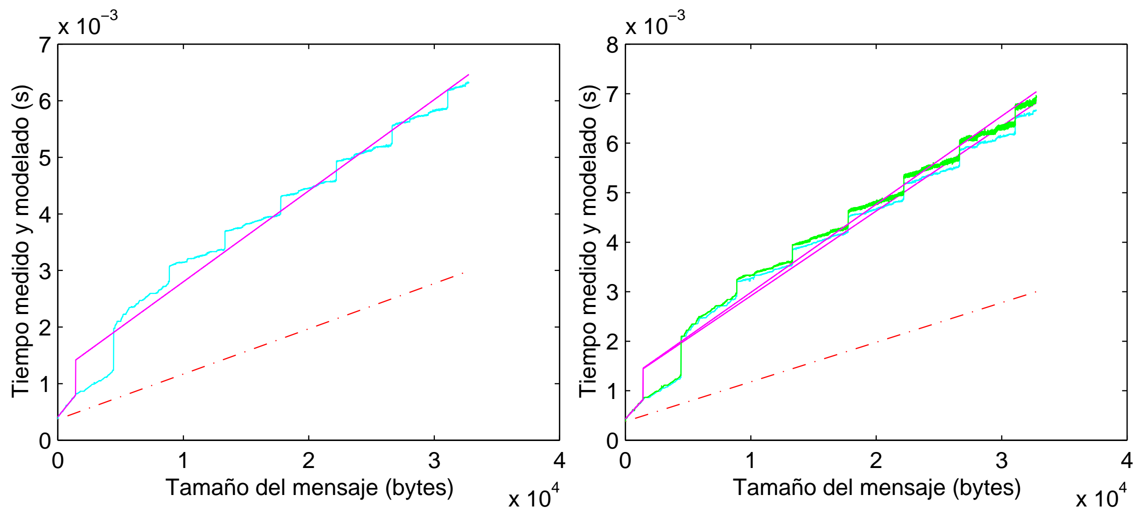
2.3.6 Determinación de los parámetros de cabecera

El primer tramo de las gráficas de tiempo no presenta ninguna característica adicional susceptible de ser modelada con mayor precisión. Además, su dominio es muy pequeño, con lo cual las mejoras en este tramo producirían suboptimización del modelo. Para reducir aún más el error cuadrático es necesario modelar con mayor precisión el segundo tramo.



(a) Homogéneo, Cliente a cliente

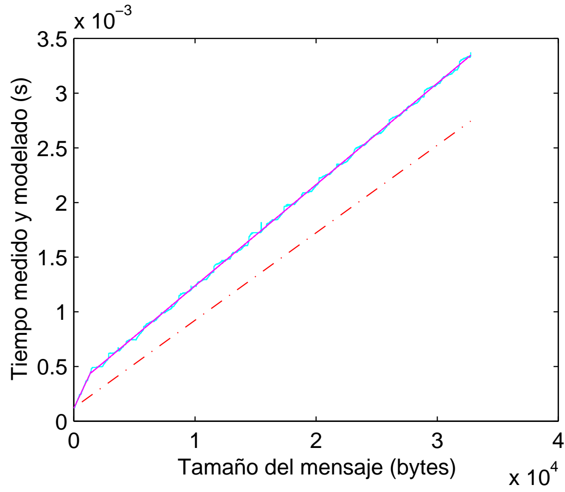
(b) Heterogéneo, Cliente a cliente



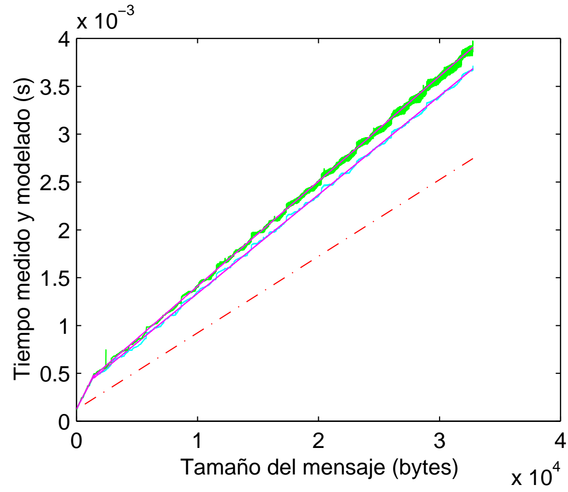
(c) Homogéneo, *daemon* LAM

(d) Heterogéneo, *daemon* LAM

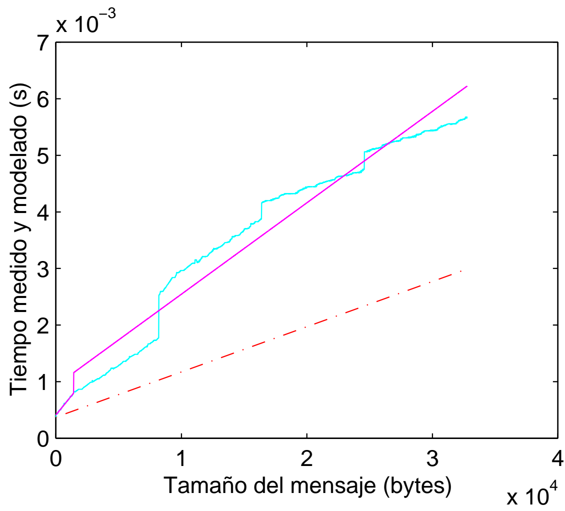
Figura 2.33: Modelo 1, configuración LAM coincidente con PVM MAXNMSGLEN=4K.



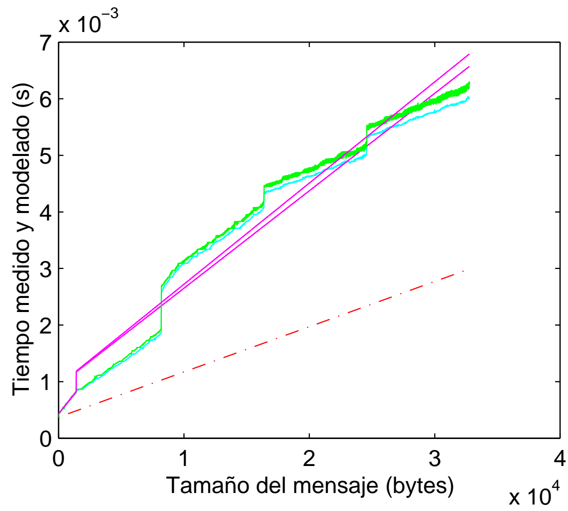
(a) Homogéneo, Cliente a cliente



(b) Heterogéneo, Cliente a cliente

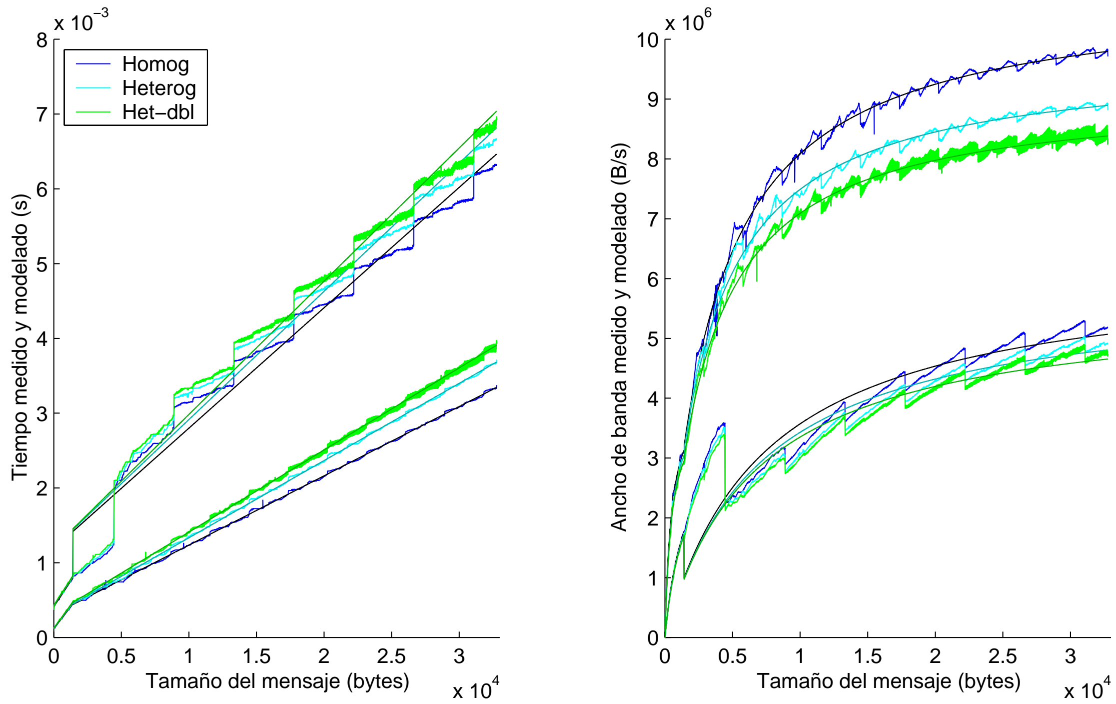


(c) Homogéneo, *daemon* LAM

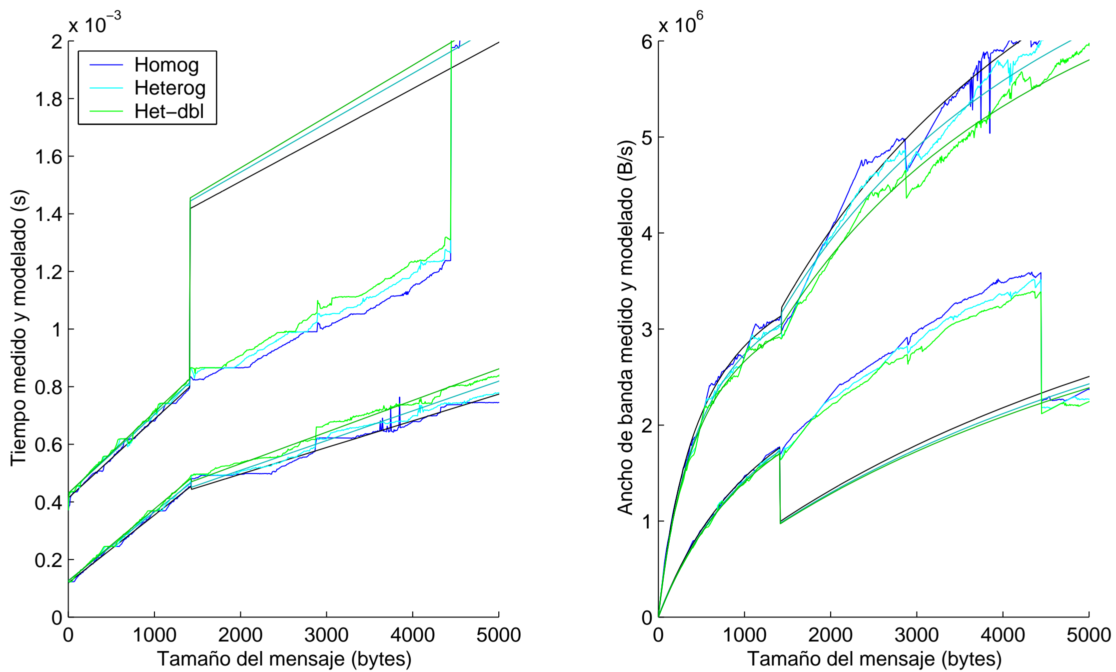


(d) Heterogéneo, *daemon* LAM

Figura 2.34: Modelo 1, configuración estándar LAM MAXNMSGLEN=8K.

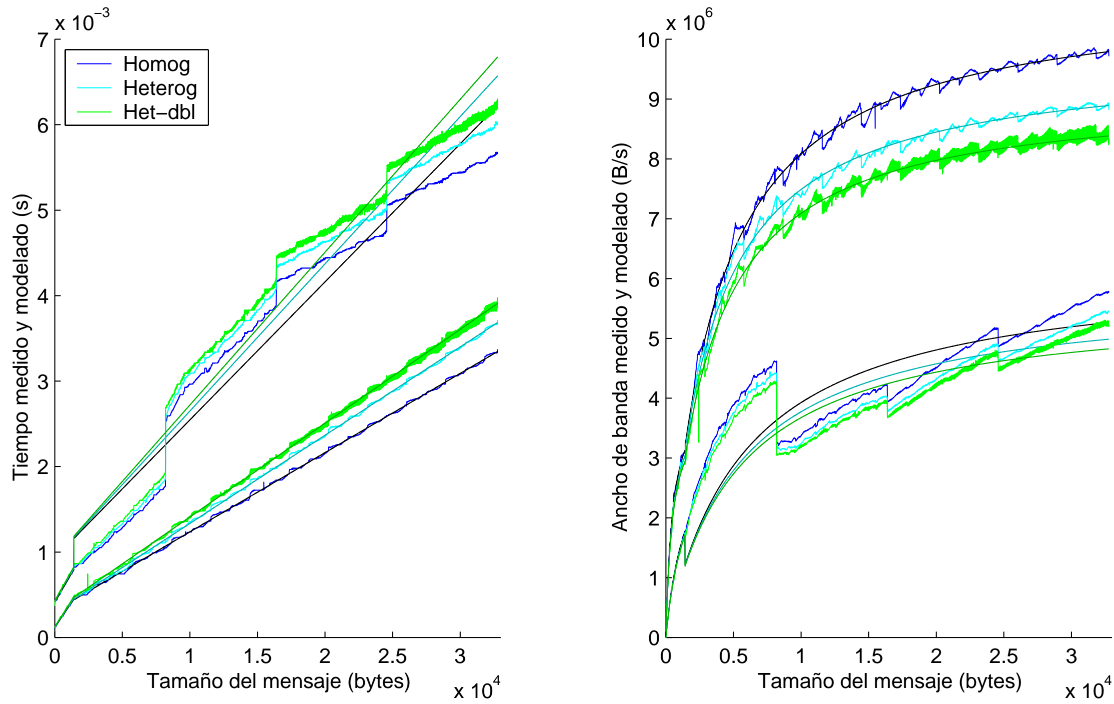


(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.

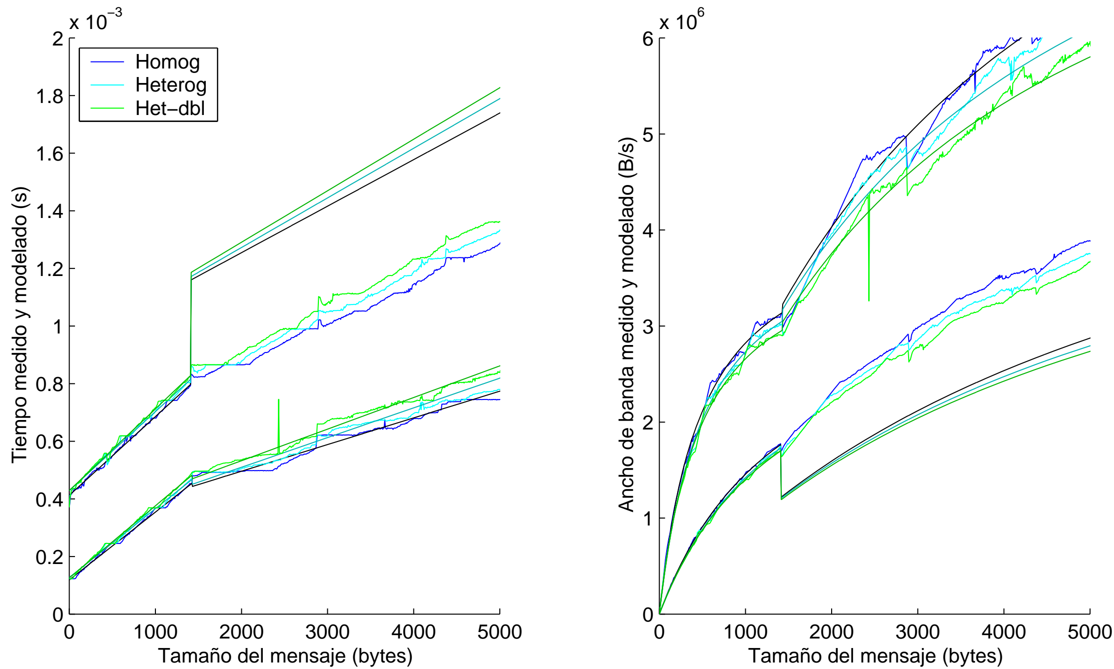


(b) Ampliación al rango de 5KB.

Figura 2.35: Modelo 1, Configuración LAM coincidente con PVM MAXNMSGLEN=4K: Agrupación comparativa de las 4 gráficas de la Figura 2.33.



(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.



(b) Ampliación al rango de 5KB.

Figura 2.36: Modelo 1, Configuración LAM estándar MAXNMSGLEN=8K: Agrupación comparativa de las 4 gráficas de la Figura 2.34.

	cliente-a-cliente					
Cluster	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
Homogéneo	76	119.017	4.251	-11.026	10.802	1.1584E-06
Heterogéneo	76	126.951	4.196	-15.919	9.693	1.4307E-06
Het, double	76	125.414	3.999	-12.159	9.113	3.0223E-06
	***daemon* LAM**					
Cluster	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
Homogéneo	92	413.483	3.655	618.258	6.213	2.8075E-04
Heterogéneo	92	430.148	3.670	629.096	5.835	2.9177E-04
Het, double	92	430.017	3.552	627.428	5.615	2.9042E-04

(a) Configuración LAM coincidente con PVM (MAXNMSGLEN 4K)

	cliente-a-cliente					
Cluster	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
Homogéneo	76	118.827	4.249	-11.266	10.795	1.1497E-06
Heterogéneo	76	127.275	4.189	-16.883	9.690	1.3746E-06
Het, double	76	125.772	3.996	-12.799	9.113	2.9858E-06
	***daemon* LAM**					
Cluster	H (B)	lc (μs)	bc (MB/s)	ls (μs)	bs (MB/s)	err
Homogéneo	92	412.462	3.658	361.761	6.193	4.7481E-04
Heterogéneo	92	428.768	3.647	355.898	5.807	4.8037E-04
Het, double	92	429.131	3.528	356.871	5.594	4.7135E-04

(b) Configuración LAM estándar (MAXNMSGLEN 8K)

Tabla 2.12: Parámetros del Modelo 1 ajustados a las diversas configuraciones de LAM.

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	$1.1584\ 10^{-6}$	$2.8075\ 10^{-4}$	$1.1497\ 10^{-6}$	$4.7481\ 10^{-4}$
Heterogéneo	$1.4307\ 10^{-6}$	$2.9177\ 10^{-4}$	$1.3746\ 10^{-6}$	$4.8037\ 10^{-4}$
Het, double	$3.0223\ 10^{-6}$	$2.9042\ 10^{-4}$	$2.9858\ 10^{-6}$	$4.7135\ 10^{-4}$

(a) Modelo 1

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	$3.0023\ 10^{-6}$	$3.5327\ 10^{-4}$	$2.9890\ 10^{-6}$	$5.0394\ 10^{-4}$
Heterogéneo	$2.9506\ 10^{-6}$	$3.6478\ 10^{-4}$	$2.8805\ 10^{-6}$	$5.0780\ 10^{-4}$
Het, double	$4.7717\ 10^{-6}$	$3.6345\ 10^{-4}$	$4.7240\ 10^{-6}$	$4.9916\ 10^{-4}$

(b) Modelo 0

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	2.592	1.258	2.600	1.061
Heterogéneo	2.062	1.250	2.096	1.057
Het, double	1.579	1.251	1.582	1.059

(c) Mejora relativa: error del Modelo 0 dividido entre el del Modelo 1

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	0.386	0.795	0.385	0.942
Heterogéneo	0.485	0.800	0.477	0.946
Het, double	0.633	0.799	0.632	0.944

(d) Reducción relativa: error del Modelo 1 dividido entre el del Modelo 0

Tabla 2.13: Errores cuadráticos medios de los Modelos 0 y 1 bajo las diversas configuraciones LAM.

Es inmediato observar que cada MTU produce un pequeño salto en el tiempo de transmisión, que puede ser modelado reutilizando el parámetro ls, sumándolo por cada MTU que se añada al mensaje.

Resulta aún más evidente que las modalidades *daemon* tanto de PVM como de LAM presentan "escalones" en el tiempo de transmisión cada 3 o 6 MTUs, según la configuración (longitud de fragmento UDP, 4K u 8K), escalones susceptibles de ser modelados mediante un parámetro latencia adicional, lm (la letra m es por *memory*). Conviene volver a matizar que los parámetros carecen realmente de interpretación física, y los nombres asignados a los mismos tienen la categoría de simples ayudas mnemotécnicas.

Con ruta TCP directa no se observan dichos escalones, por lo cual se puede congelar el parámetro lm a 0 o permitir que la evolución del método simplex lo reduzca a un valor prácticamente nulo.

Este tipo de modelado requiere conocer con exactitud tanto el tamaño de datos de la MTU como el del fragmento UDP, a fin de determinar con exactitud los tamaños de mensaje en los que se deben sumar las latencias ls y lm, respectivamente.

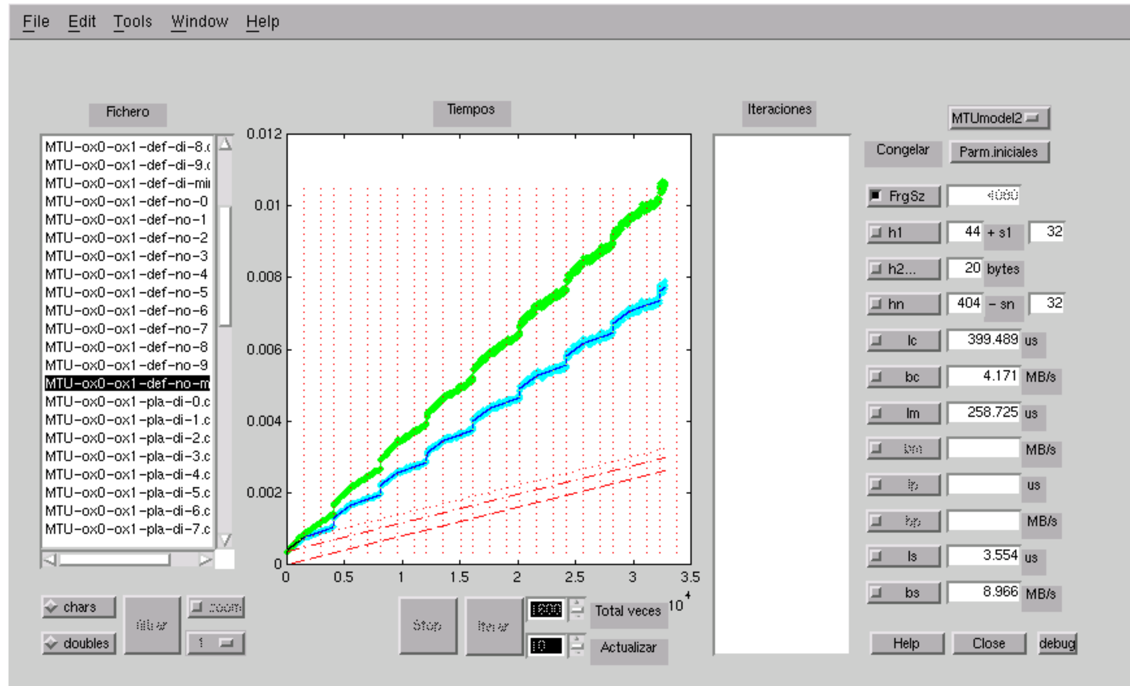
El tamaño de fragmento UDP se fija en tiempo de compilación de PVM o LAM. El tamaño de MTU no sólo varía según la combinación de opciones, sino que para una combinación dada, varía según el número de orden de la MTU dentro del mensaje. Cada una de entre la 1ª, 2ª, 3ª... MTUs puede tener un tamaño de datos distinto.

Afortunadamente, el tamaño de las cabeceras sigue un patrón cíclico predecible en función del tamaño del fragmento UDP. La primera MTU de cada fragmento UDP suele exhibir un tamaño de datos algo menor que el resto debido a una cabecera adicional que se añade a cada fragmento. La última MTU de cada fragmento también suele tener un tamaño menor debido a que el tamaño de fragmento UDP no suele ser múltiplo exacto del tamaño de MTU. Por último, el primer fragmento UDP del mensaje puede presentar excepciones, que usualmente se manifiestan como un incremento de la cabecera de la 1ª MTU y a veces también como un decremento de la cabecera de la última MTU.

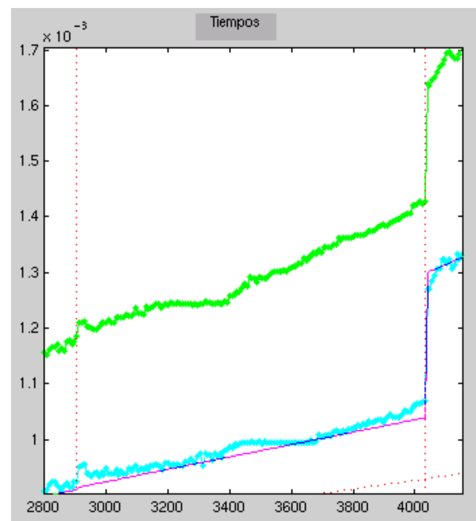
Estas excepciones son el motivo de que definiéramos en el Modelo 1 el tamaño de cabecera $H = h1 + s1$ como un tamaño fijo $h1$ (*header*) más un tamaño excepcional $s1$ (*special*). Incluso estos parámetros son sintéticos, como veremos inmediatamente. El único parámetro de cabecera con realidad física es fs (*fragment size*), el tamaño del fragmento UDP.

Por ejemplo, la configuración estándar de PVM es UDPMAXLEN=4K−MSGHDRLEN. Dicho tamaño ($4096 - 16 = 4080$) no es múltiplo del tamaño de MTU estándar para Ethernet $\|MTU\| = 1500$, por lo cual la tercera MTU será fragmentaria ($4080 = 1500 * 2 + 1080$). Las cabeceras determinadas mediante el GUI (Figura 2.37) para la modalidad *daemon* (44–20–404 en la Tabla 2.14) sólo justifican 4032 bytes. Además, la primera MTU presenta 32 bytes de cabecera adicionales, recuperados con 32 bytes de defecto en la última MTU del primer fragmento UDP. Por concluir la descripción de este caso, la modalidad de empaquetamiento *InPlace* sólo presenta la primera excepción $s1 = 32$, mientras que $sn = 0$, y hn es 32 bytes menor.

Para el objetivo de este trabajo es suficiente poder predecir con exactitud el tamaño de mensaje para el que se produce el escalón en la característica de prestaciones, objetivo satisfecho a la perfección mediante la lista de parámetros de cabecera relacionados en la Tabla 2.14. No obstante, es natural cuestionarse la posibilidad de explicar todas y cada una de las cabeceras en función de los distintos mecanismos implicados en cada modalidad de transmisión.



(a) Observar las casillas de parámetros $h1 \ldots sn$ activadas (fs no se activa porque se deduce del nombre del directorio), así como las marcas verticales indicando la frontera entre MTUs. Los parámetros pueden editarse hasta que todas las fronteras queden correctamente determinadas. Un error de 1 byte podría permanecer oculto durante 7 fronteras, dado que los barridos se realizaron con un paso de 8 bytes. El rango escogido, 32KB, abarca más de 20 MTUs (20*1500=30000<32KB).



(b) El GUI permite también realizar *zoom* para afinar el valor hasta el byte exacto.

Figura 2.37: Los Modelos 2 y 3 necesitan conocer los tamaños exactos de mensaje que son inicio/fin de una MTU o de un fragmento UDP. Para ello bastan 6 parámetros: fs; $h1$, $h2$ y hn; $s1$ y sn. Los parámetros se obtienen gráficamente mediante el GUI.

Configuración PVM estándar UDPMAXLEN=4K−16

	PvmRouteDirect						PvmDontRoute.					
	fs	h1	h2	hn	s1	sn	fs	h1	h2	hn	s1	sn
InPlace	4080	52	52	52	0	0	4080	44	20	372	32	0
DataRaw	4080	68	52	348	32	32	4080	44	20	404	32	32
Default	4080	68	52	348	32	32	4080	44	20	404	32	32

Configuración PVM coincidente con LAM UDPMAXLEN=8K

	PvmRouteDirect						PvmDontRoute.					
	fs	h1	h2	hn	s1	sn	fs	h1	h2	hn	s1	sn
InPlace	8192	52	52	52	0	0	8192	44	20	700	32	0
DataRaw	8192	68	52	580	32	32	8192	44	20	732	32	32
Default	8192	68	52	580	32	32	8192	44	20	732	32	32

(a) Parámetros de cabecera PVM.

Configuración LAM coincidente con PVM MAXNMSGLEN=4440

	cliente-a-cliente						*daemon* LAM.					
	fs	h1	h2	hn	s1	sn	fs	h1	h2	hn	s1	sn
Homogéneo	4440	52	52	52	24	0	4440	92	20	1428	0	0
Heterogéneo	4440	52	52	52	24	0	4440	92	20	1428	0	0

Configuración LAM estándar MAXNMSGLEN=8K

	cliente-a-cliente						*daemon* LAM.					
	fs	h1	h2	hn	s1	sn	fs	h1	h2	hn	s1	sn
Homogéneo	8192	52	52	52	24	0	8192	92	20	636	0	0
Heterogéneo	8192	52	52	52	24	0	8192	92	20	636	0	0

(b) Parámetros de cabecera LAM.

Tabla 2.14: Parámetros de cabecera para las distintas configuraciones y combinaciones de opciones de PVM y LAM/MPI. En general, el tamaño de datos de la i-ésima MTU, $||MTU_i|| = 1500 - h_{(i \bmod n)+1}$, salvo para la primera y última del primer fragmento UDP (1^{a} y n^{a}), que tienen cabeceras de tamaño $h1 + s1$ y $hn - sn$, respectivamente. n es el número de MTUs que componen un fragmento UDP, y se deduce de fs.

Se proporciona a continuación información relacionada con el tema, suficiente para hacer comprender la dificultad de semejante explicación, que requeriría localizar sobre el código fuente de PVM o LAM/MPI, o del propio Sistema Operativo, o incluso en las especificaciones técnicas de la tarjeta de red, el lugar exacto en que se añaden las diversas cabeceras.

El propio sistema PVM añade a cada mensaje una cabecera de 32 bytes (Figura 2.38). Existe también una cabecera para paquetes entre *daemons* PVM (Figura 2.39(a)), y otra para paquetes entre *daemons* y tareas PVM (Figura 2.39(b)), que aunque no formen parte del mensaje, consumen tiempo de transmisión si la combinación de opciones PVM utilizada los requiere.

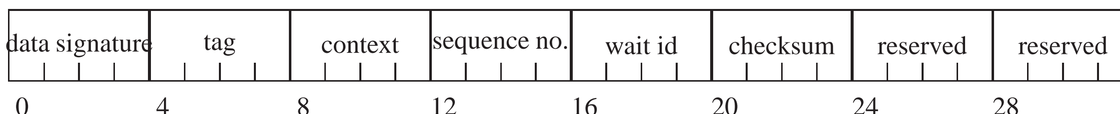


Figura 2.38: Formato de cabecera de mensaje PVM. La información ha sido obtenida del fichero fuente **global**.h. El libro PVM [27, pág.113, fig.7.8] y la Guía del Usuario [28, pág.51, fig.14] proporcionan aproximadamente la misma información.

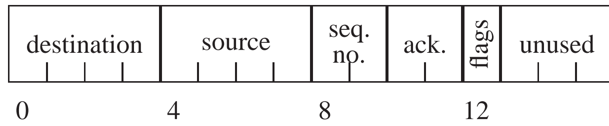
La documentación del sistema LAM no es tan detallada, careciendo de figuras equivalentes a las mostradas para PVM. Afortunadamente, la modalidad *daemon* de LAM está segregada en un único fichero fuente $LAMHOME/share/mpi/rpi.lamd.c de tamaño abordable (unos 35KB de texto), y el sistema de Interfaz para Progreso de Peticiones (Request Progression Interface, RPI) está documentado en [42], pudiendo ser un punto de partida para realizar un estudio que conduzca a obtener dicha información.

Adicionalmente, los protocolos UDP/IP y TCP/IP añaden a cada segmento de datos enviado su propia cabecera indicando diversa información propia al protocolo, como por ejemplo la dirección de los puertos fuente y destino, el tamaño de datos y de cabecera, códigos de detección de error, etc (Figuras 2.39(e) y 2.39(f)).

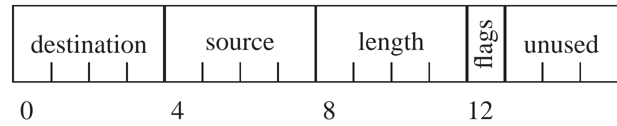
Ambos protocolos se basan en el protocolo de red IP, que añade a cada datagrama enviado su propia cabecera indicando información propia adicional, como dirección IP fuente y destino, tamaño, *checksum*... (Figura 2.39(d)). Bajo RedHat, un buen lugar para empezar a buscar dónde se añaden estas cabeceras en el código fuente del Sistema Operativo podría ser el subdirectorio /usr/src/linux/net/ipv4.

Si el datagrama IP se envía mediante una red Ethernet, se debe refragmentar y empaquetar en marcos de paquete Ethernet, indicando esta vez información propia del nivel físico, como dirección Ethernet fuente y destino, longitud, checksum... (Figura 2.39(c)). Si no se dispone del texto del estándar, se pueden deducir algunos de los campos consultando el subdirectorio /usr/src/linux/net/ethernet, o más recientemente (desde Mayo de 2001), consultando las referencias [21] y [20].

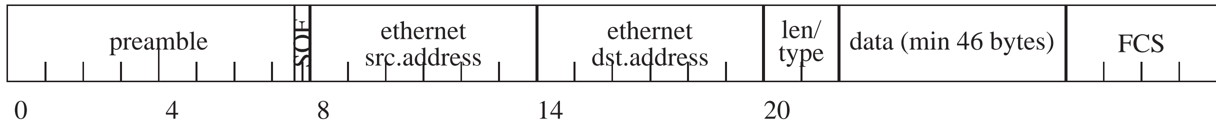
Habiendo tantas capas de protocolos y sistemas añadiendo cada una su propia cabecera a la cantidad total de datos enviados, resulta más fiable y rápido utilizar el GUI para determinar los tamaños de datos de cada MTU bajo cada combinación de opciones PVM y LAM, como se ha mostrado en la Figura 2.37. Activando las casillas de edición correspondientes, se procedió a alterar los valores hasta conseguir que todos los saltos entre MTUs quedaran reflejados correctamente por las líneas rojas verticales. Una vez determinadas las cabeceras, las casillas se "congelan" para que su valor no varíe durante la evolución del método simplex.



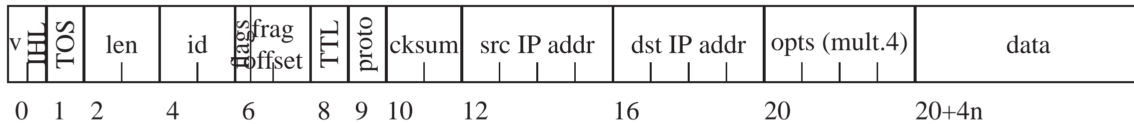
(a) Cabecera PVM de paquete pvmd–pvmd. Información obtenida del fichero fuente pvmproto.h. También disponible en [28, pág.50, fig.13] y en [27, pág.113, fig.7.9].



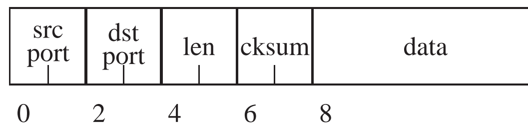
(b) Cabecera PVM de paquete pvmd–task. Información obtenida del fichero fuente pvmproto.h. También disponible en [28, pág.53, fig.15] y en [27, pág.116, fig.7.11].



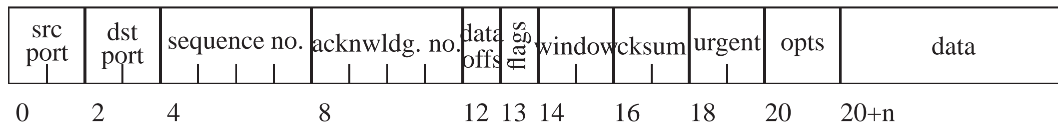
(c) Formato de marco Ethernet (frame). Se puede consultar por ejemplo en http://netman.cit.buffalo.edu/FAQs/ethernet.faq. También disponible como estándar IEEE802.3 previo pago. El preámbulo sirve para sincronizar los relojes de los transceptores fuente y destino. La dirección destino indica cuál transceptor debe continuar escuchando el paquete. El campo de chequeo ayuda a detectar una posible colisión. Los 62bits de preámbulo, 2bits de *Start Of Frame* y 4bytes de *Frame Check Sequence* son añadidos por el chip transceptor.



(d) Cabecera de datagrama IP. Información obtenida del estándar http://www.rfc-editor.org/rfc/std/std5.txt. También disponible como estándar RFC791. El objetivo de IP es encaminar datagramas, fragmentándolos si fuera preciso para poder transmitirlos por subredes con menor tamaño máximo de paquete (MTU). Para ello sirven los campos length, id y offset.



(e) Cabecera de datagrama UDP/IP. Información obtenida del estándar http://www.rfc-editor.org/rfc/std/std6.txt. También disponible como estándar RFC768. El objetivo de UDP/IP es transmitir datagramas entre aplicaciones del usuario. Para ello sirven los campos src/dst port.



(f) Cabecera de datagrama TCP/IP. Información obtenida del estándar http://www.rfc-editor.org/rfc/std/std7.txt. También disponible como RFC793. El objetivo de TCP/IP es proporcionar una conexión fiable entre procesos de usuario. Para ello sirven los campos adicionales frente a UDP.

Figura 2.39: Formatos de cabeceras PVM, Ethernet e Internet.

Los tamaños de cabeceras obtenidos se mostraron en la Tabla 2.14. Como se anticipó al principio de este apartado, es posible reducir toda la información sobre cabeceras a 6 parámetros: el tamaño de fragmento UDP fs, las cabeceras repetitivas $h1$, $h2$ y hn, y las excepcionales $s1$ y sn. A partir de fs es posible deducir el número n de MTUs que componen un fragmento UDP, y por tanto las MTUs a la que se aplican $h1$ y hn (y sn, que sólo se aplica a la n^a). $s1$ siempre se aplica a la 1ª MTU, no se requiere conocer fs. A las restantes MTUs se les aplica $h2$.

Respecto a la conveniencia de expresar el tamaño de datos de cada MTU mediante el tamaño de cabecera en vez de directamente, baste con hacer notar que su magnitud es menor y son más fáciles de recordar. Esto es particularmente cierto para LAM: sólo hn cambia con fs, estando los demás parámetros predeterminados por la modalidad de encaminamiento. Concebiblemente, la magnitud de las cabeceras podría mantenerse constante tras un cambio en el tamaño de MTU soportado por el Sistema Operativo, argumento que terminó de decidirnos por esta opción.

2.3.7 Modelo 2

En este modelo, cada MTU produce un nuevo tramo en la gráfica. La primera se modela con los parámetros lc y bc, hasta llegar a un tamaño de mensaje de $\text{MTU1HDR} = 1500 - h1 - s1$ bytes, que tarda en transmitirse $T_{M1H} = lc + \text{MTU1HDR}/bc$. De nuevo se puede identificar el parámetro latencia lc con el *tiempo de transmisión de mensaje nulo*, T_0. bc también tiene sentido físico, representando el ancho de banda efectivo para la primera MTU.

Los siguientes tramos de menor pendiente se modelan mediante los parámetros ls y bs: cada nueva MTU o fracción añade un tiempo de transmisión de $ls + S/bs$, en donde S es el tamaño de datos de la correspondiente MTU o el tamaño restante del mensaje (última MTU fraccionaria). Cada n MTUs se utiliza la latencia lm en lugar de ls. El código MATLAB es algo más elaborado, y puede consultarse en el Apéndice B.4, junto con la correspondiente explicación detallada.

El modelo no se presta a ser reducido a una simple expresión algebraica, ni siquiera como disyunción de casos al estilo de la utilizada en la presentación del Modelo 1. Puede ser ilustrativo en cambio resumir los parámetros que modelan cada tramo de la característica de prestaciones:

$$\text{Modelo 2} \begin{cases} \quad lc/bc \quad ls/bs \quad \ldots \quad ls/bs & \text{1ª UDP, n tramos, } n = \frac{\|UDP\|}{\|MTU\|} \\ [\quad lc + lm + n.ls/bc \quad ls/bs \quad \ldots \quad ls/bs \quad] & \text{2ª UDP para daemon LAM} \\ \quad lm/bc, bs \quad ls/bs \quad \ldots \quad ls/bs & \text{restantes UDPs, bc para daemon PVM} \end{cases}$$

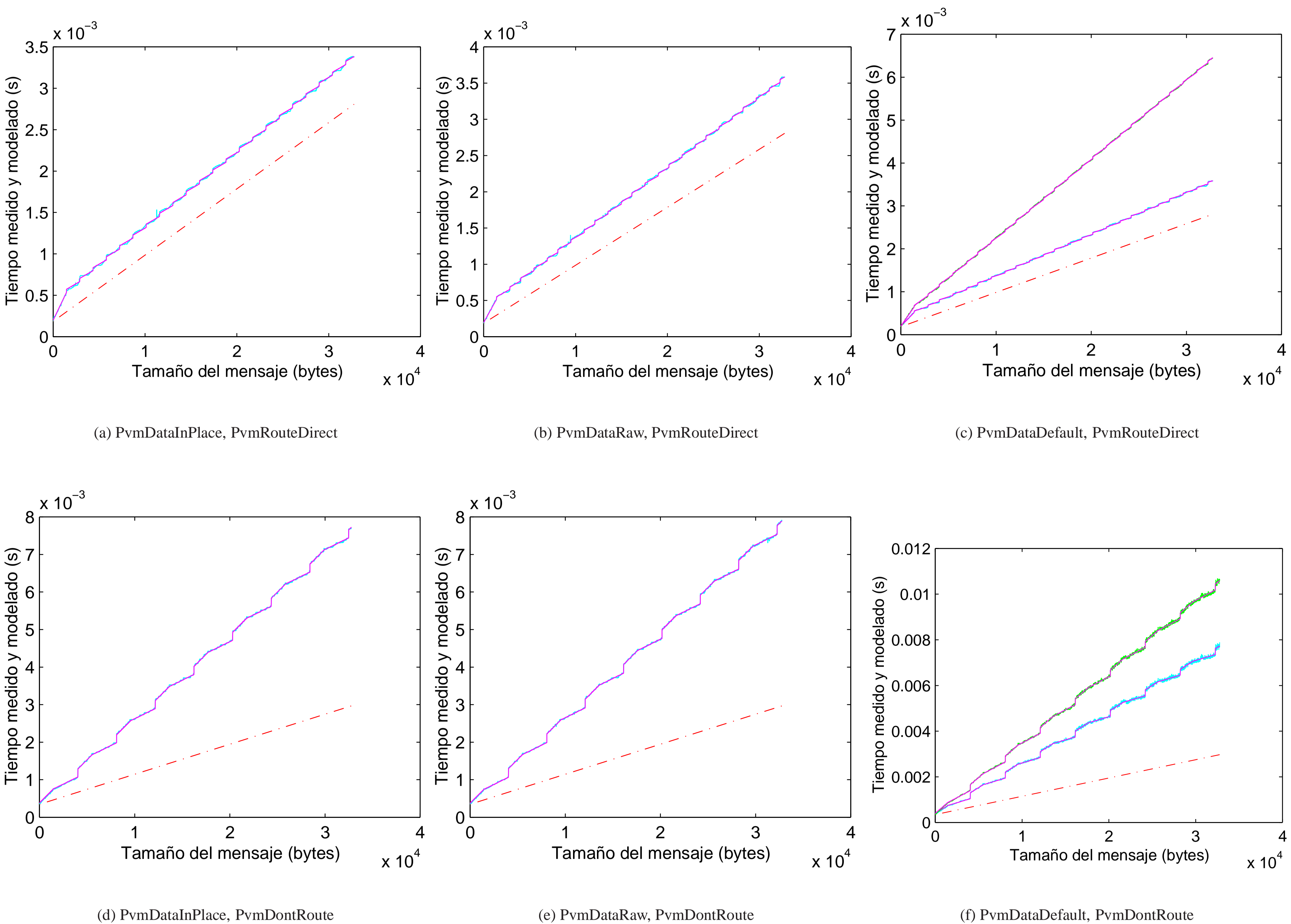
El segundo fragmento UDP especial sólo se considera para el caso del *daemon* LAM, habiéndose encontrado tras varios ajustes una estimación aceptable del salto entre la 1ª y 2ª UDPs.

Como ya vimos, el *daemon* PVM repite la estructura de la primera UDP en las restantes, reutilizando bc en cada 1ª MTU. Las demás modalidades sólo reutilizan bs.

Resulta interesante destacar que el código del modelo puede detectar las respectivas modalidades *daemon* mediante el valor de las cabeceras. Un rápido vistazo a la Tabla 2.14 permite comprobar que el *daemon* LAM queda delatado por $h1 == 92$, y el *daemon* PVM por $h1 == 44$.

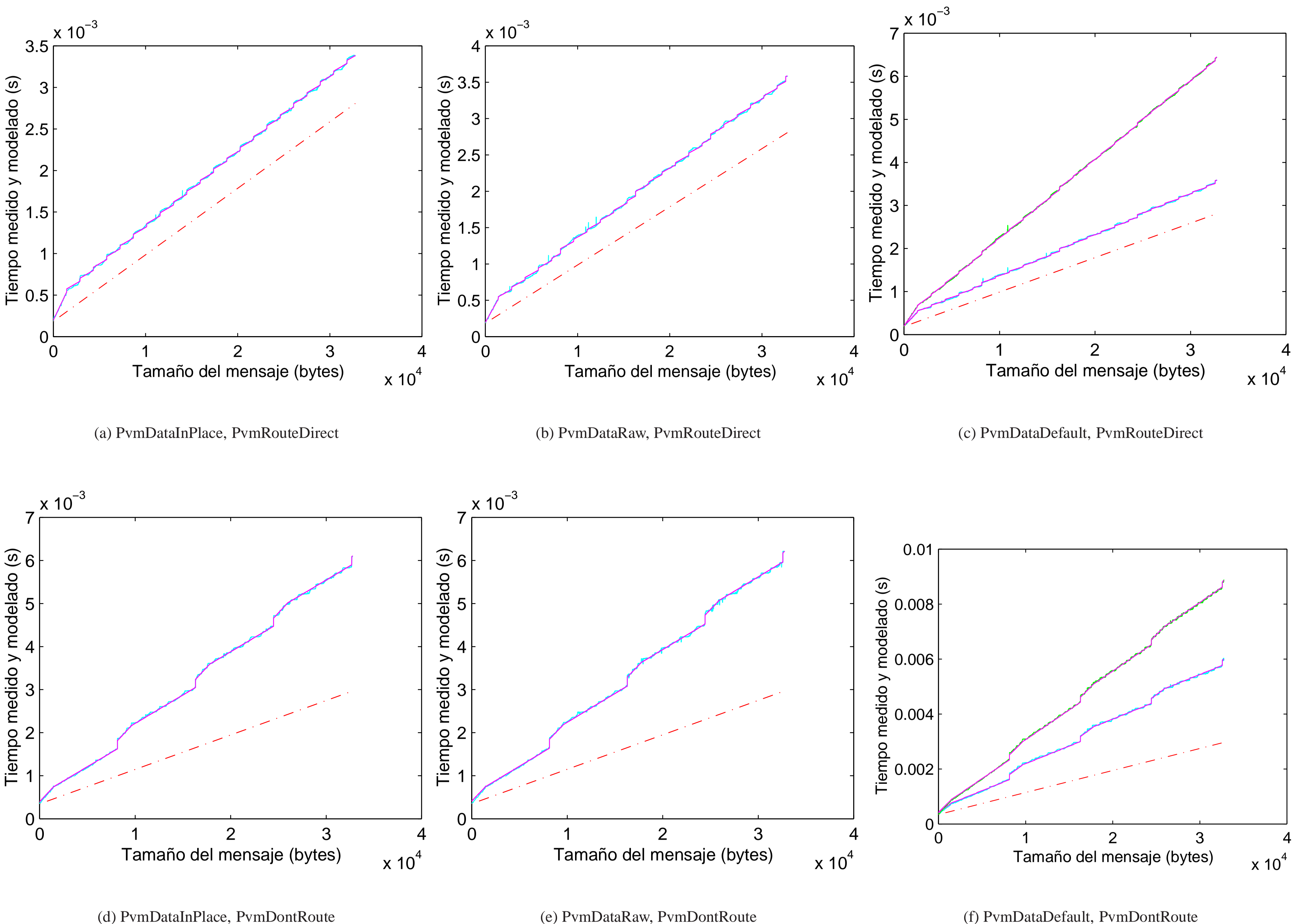
Parámetros extraidos bajo PVM

Los ficheros de mínimos para las 6 combinaciones de opciones PVM, junto con los trazos ajustados por el Modelo 2, se presentan en las Figuras 2.40 (configuración estándar PVM) y 2.41 (configuración coincidente con LAM).



(a) PvmDataInPlace, PvmRouteDirect
(b) PvmDataRaw, PvmRouteDirect
(c) PvmDataDefault, PvmRouteDirect
(d) PvmDataInPlace, PvmDontRoute
(e) PvmDataRaw, PvmDontRoute
(f) PvmDataDefault, PvmDontRoute

Figura 2.40: Modelo 2, configuración estándar PVM (UDPMAXLEN 4K).



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

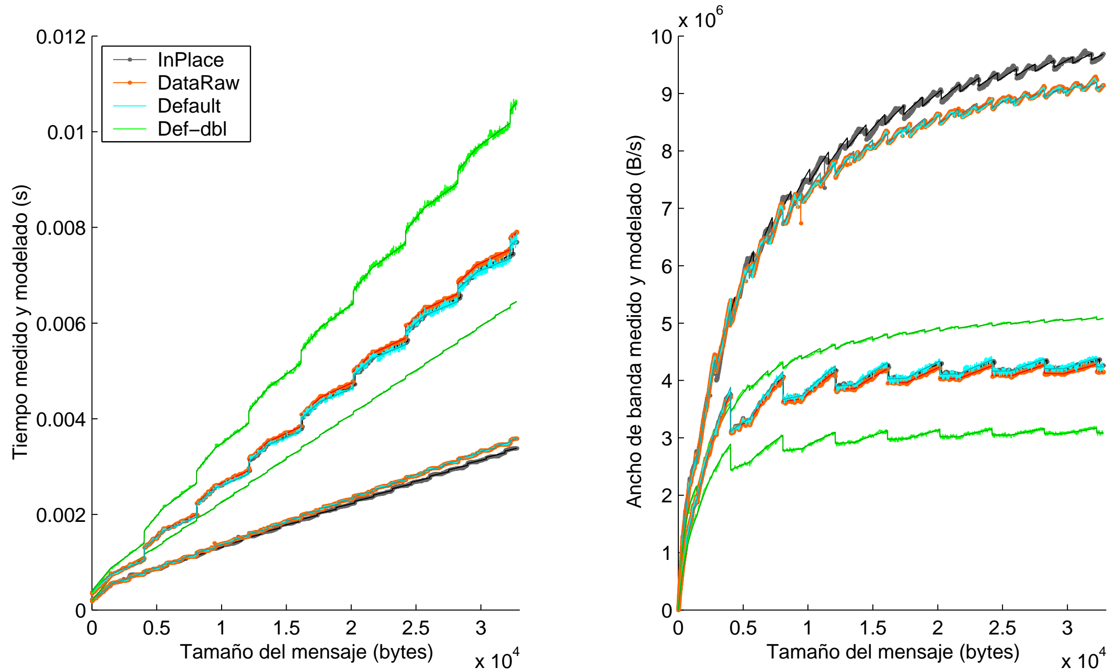
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

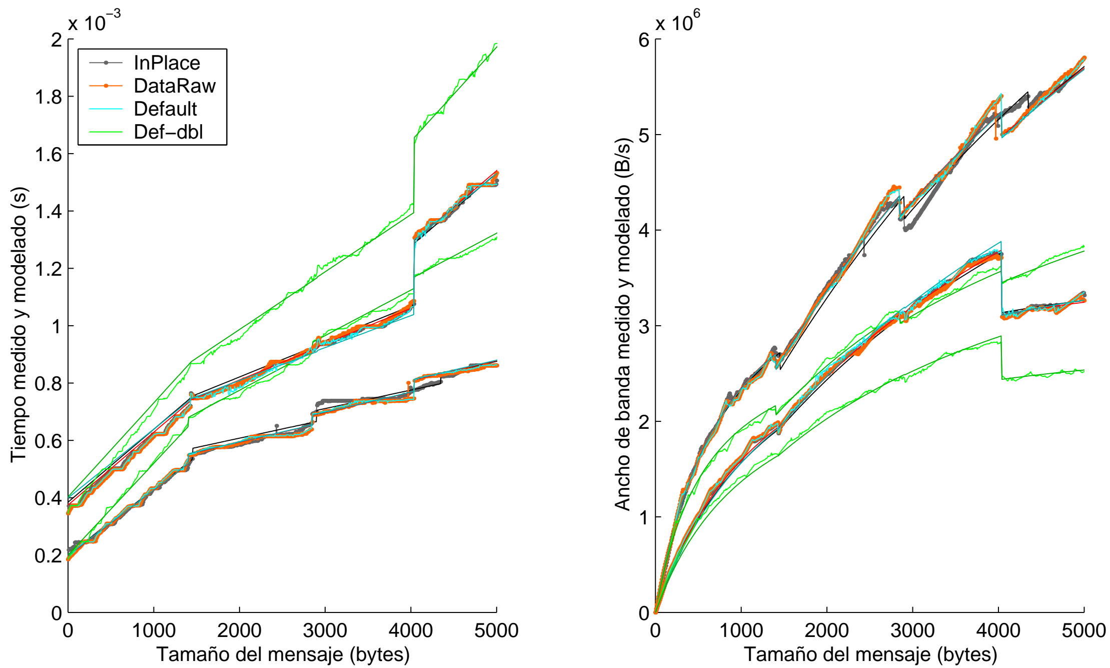
(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.41: Modelo 2, configuración coincidente con LAM (UDPMAXLEN 8K).

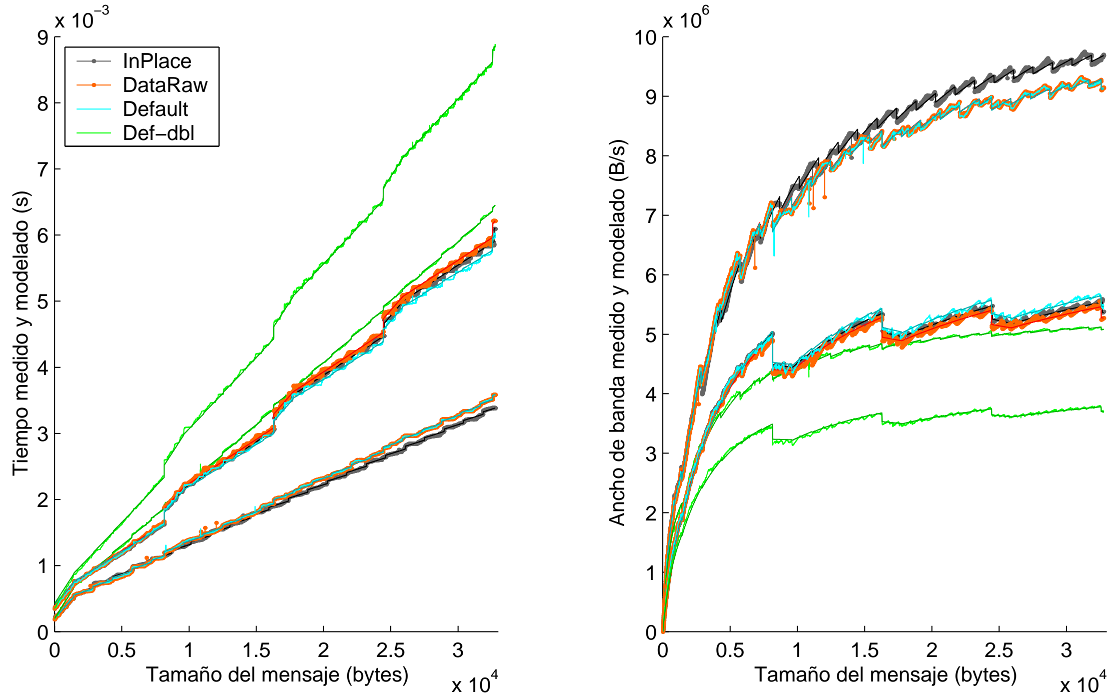


(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.

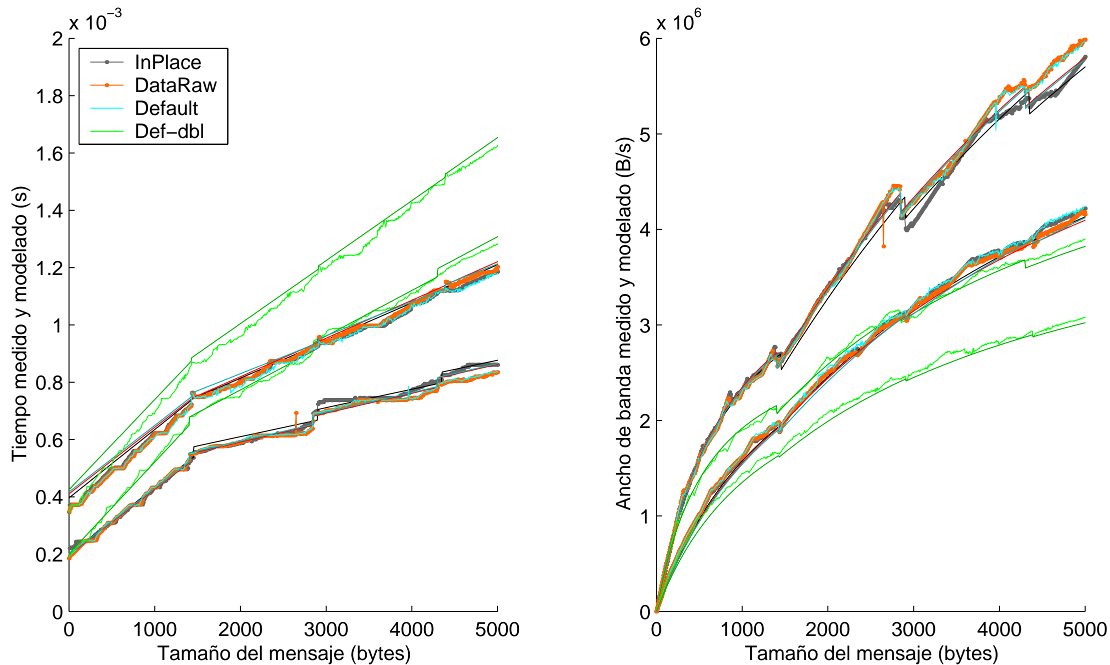


(b) Ampliación al rango de 5KB.

Figura 2.42: Modelo 2, Configuración estándar PVM UDPMAXLEN=4K: Agrupación comparativa de las 6 gráficas de la Figura 2.40.



(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.



(b) Ampliación al rango de 5KB.

Figura 2.43: Modelo 2, Configuración PVM coincidente con LAM UDPMAXLEN=8K: Agrupación comparativa de las 6 gráficas de la Figura 2.41.

	PvmRouteDirect					
Empaquetamiento	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
InPlace	199.300	4.335	35.613	38.875	15.521	7.6634E-07
DataRaw	196.922	4.368	41.069	34.152	14.313	5.8595E-07
Default	198.761	4.376	42.437	34.758	14.432	5.7723E-07
íd, dbl	200.437	3.131	39.751	32.214	6.308	4.7444E-07
	PvmDontRoute					
Empaquetamiento	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
InPlace	384.710	3.937	213.464	5.072	8.251	1.2179E-06
DataRaw	374.600	3.940	230.506	3.734	8.132	9.9444E-07
Default	399.489	4.171	258.725	3.554	8.966	6.7030E-06
íd, dbl	402.940	3.051	260.174	2.221	5.019	6.2309E-06

(a) Configuración PVM estándar (UDPMAXLEN 4K)

	PvmRouteDirect					
Empaquetamiento	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
InPlace	198.628	4.274	43.103	37.211	15.582	7.4893E-07
DataRaw	197.461	4.371	65.344	31.849	14.440	9.7720E-07
Default	199.216	4.353	66.291	32.081	14.509	9.8732E-07
íd, dbl	201.590	3.126	59.293	28.426	6.280	9.3206E-07
	PvmDontRoute					
Empaquetamiento	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
InPlace	396.481	4.192	188.521	3.209	7.702	2.2668E-06
DataRaw	410.678	4.311	207.634	2.143	7.557	2.9399E-06
Default	416.877	4.302	184.832	13.729	8.434	3.9256E-06
íd, dbl	425.147	3.179	183.472	11.068	4.783	3.7009E-06

(b) Configuración PVM coincidente con LAM (UDPMAXLEN 8K)

Tabla 2.15: Parámetros del Modelo 2 ajustados a las diversas configuraciones de PVM.

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	$7.6634\ 10^{-7}$	$1.2179\ 10^{-6}$	$7.4893\ 10^{-7}$	$2.2668\ 10^{-6}$
PvmDataRaw	$5.8595\ 10^{-7}$	$9.9444\ 10^{-7}$	$9.7720\ 10^{-7}$	$2.9399\ 10^{-6}$
PvmDataDefault	$5.7723\ 10^{-7}$	$6.7030\ 10^{-6}$	$9.8732\ 10^{-7}$	$3.9256\ 10^{-6}$
ídem, double	$4.7444\ 10^{-7}$	$6.2309\ 10^{-6}$	$9.3206\ 10^{-7}$	$3.7009\ 10^{-6}$

(a) Modelo 2

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	$1.2157\ 10^{-6}$	$2.6102\ 10^{-5}$	$1.2087\ 10^{-6}$	$2.8831\ 10^{-5}$
PvmDataRaw	$1.0756\ 10^{-6}$	$2.8457\ 10^{-5}$	$1.9909\ 10^{-6}$	$3.1301\ 10^{-5}$
PvmDataDefault	$1.0935\ 10^{-6}$	$3.9272\ 10^{-5}$	$2.0303\ 10^{-6}$	$2.9968\ 10^{-5}$
ídem, double	$9.2408\ 10^{-7}$	$3.9282\ 10^{-5}$	$1.7600\ 10^{-6}$	$2.8097\ 10^{-5}$

(b) Modelo 1

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	1.586	21.432	1.614	12.719
PvmDataRaw	1.836	28.616	2.037	10.647
PvmDataDefault	1.894	5.859	2.056	7.634
ídem, double	1.948	6.304	1.888	7.592

(c) Mejora relativa: error del Modelo 1 dividido entre el del Modelo 2

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	0.630	0.047	0.620	0.079
PvmDataRaw	0.545	0.035	0.491	0.094
PvmDataDefault	0.528	0.171	0.486	0.131
ídem, double	0.513	0.159	0.530	0.132

(d) Reducción relativa: error del Modelo 2 dividido entre el del Modelo 1

Tabla 2.16: Errores cuadráticos medios de los Modelos 1 y 2 bajo las diversas configuraciones PVM.

La Figura 2.42 agrupa las 6 subgráficas de la configuración estándar, y la Figura 2.43 las de la coincidente con LAM. Las predicciones del modelo aparecen en un tono algo más oscuro del mismo color que la correspondiente opción PVM.

La Tabla 2.15 resume los valores de parámetros extraidos, mientras que la comparación de los errores cuadráticos medios con los del modelo anterior se realiza en la Tabla 2.16.

Se debe recordar que los parámetros $h1 \ldots sn$ (y por supuesto fs) están "congelados", permaneciendo constantes mientras evoluciona el método simplex. Se ha preferido referir al lector a los valores que ya se mostraron en la Tabla 2.14, en lugar de abarrotar la tabla de parámetros obtenidos.

Comparando con la correspondiente Tabla 2.10 del modelo anterior, tal vez sorprenda comprobar que lc y bc varían un poco. lc crece unos poco significativos 5μs para ruta directa, y del orden de 25–50μs a través del *daemon* PVM, mientras que bc crece unos 0.1–0.3MB/s.

Los valores del Modelo 1 eran más "físicos" para estos parámetros, acercándose más a la latencia T_0 y al ancho de banda de la primera MTU. Un motivo de variación de bc es que se reutiliza en la modalidad *daemon*, pero eso no justificaría la variación observada incluso con ruta directa, donde tanto lc como bc están igual de libres que en el Modelo 1.

La ligazón es indirecta, ya que el único parámetro conectando el final de la 1ª MTU con el principio de la 2ª es siempre ls, parámetro fuertemente reutilizado (igual que bs). El incremento de lc en este modelo se debe por tanto a que el método simplex encuentra más provechoso reducir el error en el resto de la gráfica (a partir de la 2ª MTU) con un centrado impecable del modelo sobre las mediciones, a costa incluso de subir ligeramente el punto de arranque de la 2ª MTU. Tal vez una aguda comparación de las Figuras 2.42(b) y 2.43(b) con las 2.31(b) y 2.32(b) permita comprobar dicho extremo.

El crecimiento de bc es más complicado de analizar al tener también una ligazón directa en modo *daemon* debido a su reutilización como 1ª MTU de cada fragmento UDP. De hecho, con ruta directa no siempre crece (ver *DataInPlace*). En definitiva, el modelo presenta ligazones que directa o indirectamente restan sentido físico a los parámetros que modelan la primera MTU, lc y bc. En cualquier caso, el error global del modelo es menor.

Como se esperaba, lm se adapta a los pronunciados saltos de cada UDP a través del *daemon*. Con ruta directa, su valor depende de la configuración, siendo muy similar a ls en la estándar (4K) y casi el doble en la coincidente con LAM (8K). Un vistazo más detallado a las características de prestaciones con el GUI revela que con ruta directa hay alternancia de dos latencias, una menor (modelada con bastante precisión por ls) y otra algo mayor. lm intenta corregir el desfase cada $n = \frac{\|UDP\|}{\|MTU\|}$ MTUs, así que no es de extrañar que su valor bajo UDPMAXLEN=4K sea el doble que bajo UDPMAXLEN=8K. El siguiente modelo corrige esta deficiencia.

ls pierde el estigma de "parámetro bisagra" al modelar con éxito el pequeño salto entre MTUs. Con ruta directa se estabiliza en torno a unos 20–30μs con una envidiable independencia del tamaño UDP, y a través del *daemon* desaparece, tomando frecuentemente valores inferiores a la resolución de la rutina gettimeofday() utilizada para medir tiempos.

Podría argumentarse que bs empieza también a perder sentido físico, al superar repetidamente el máximo ancho de banda que puede proporcionar el *switch*, 12.5MB/s.

Sucede sin embargo lo contrario que con bc, ya que las mediciones presentan realmente esas pendientes (15.582MB/s, etc), y bs las refleja correctamente, lo cual no sucedía en el modelo anterior.

Las filas superiores (encaminamientos *PvmRouteDirect*) de las Figuras 2.40 y 2.41 son el lugar apropiado para comprobar que las MTUs presentan individualmente pendientes inferiores al mínimo esperado 12.5MB/s de la línea roja de referencia.

Indudablemente, otros fenómenos no considerados en estos modelos, como por ejemplo los paquetes de reconocimiento del sistema PVM o del propio protocolo TCP, están enmascarando el tiempo real de transmisión para el caso de MTUs fragmentarias de tamaño muy reducido. Dado un tamaño de mensaje S muy poco superior a un número entero N de MTUs, con un tiempo de transmisión T_S, es concebible que un tamaño algo superior $S+R$ (dentro del mismo número $N+1$ de MTUs) tarde significativamente menos que el esperado $T_S + R/bs$, produciendo las bajas pendiente observadas, menores que la esperada de 12.5MB/s.

Por ejemplo, siendo las conexiones del *switch* *Full-Duplex*, se podría suponer que los paquetes de reconocimiento de las sucesivas MTUs se reciben en paralelo a la transmisión de la siguiente MTU, quedando por tanto ocultos. Sólo el último paquete de reconocimiento PVM se recibe en solitario, sin poder realizar el transmisor ninguna tarea útil aparte de esperar, añadiendo un tiempo adicional que quedaría modelado por ls o lm, ya que sólo se añade con cada nueva MTU, no con cada byte transmitido.

En el caso de un mensaje PVM con una última MTU fragmentaria muy pequeña (el mencionado tamaño S, ligeramente superior a un número entero de MTUs), podría suceder que la transmisión fuera tan rápida que acabara antes incluso de haber recibido el reconocimiento de la penúltima MTU (la anterior), quedando este tiempo de recepción desenmascarado.

Un mecanismo como el explicado justificaría que para un tamaño de mensaje ligeramente superior ($S+R$), el tiempo de transmisión adicional R/bs queda ocultado por (en realidad sería él quien, si fuera mayor, enmascararía a) el tiempo de recepción de la confirmación de la penúltima MTU, tardando en total la transmisión menos que el esperado $T_S + R/bs$.

Aunque los parámetros lc y bc (que no bs) pierdan algo de sentido físico, el modelo predice con mayor precisión el tiempo de transmisión. Se puede apreciar a simple vista, comparando las Figuras 2.42(a) y 2.43(a) con las 2.31(a) y 2.32(a) del modelo anterior, una superposición más centrada de los trazos del modelo en el segundo tramo, tapando el trazo morado casi por completo a las mediciones reales. Es necesario reducirse al rango de 5KB / 2ms en las Figuras 2.42(b) y 2.43(b) para poder observar alguna divergencia entre el tiempo modelado y el medido. indicando (al menos subjetivamente) que el esfuerzo adicional de modelado ha conseguido su objetivo.

En cualquier caso, la superioridad del modelo sólo puede manifestarse objetivamente mediante la comparación de errores cuadráticos medios mostrada en la Tabla 2.16. Esta comparación demuestra que el Modelo 2 ha conseguido reducir significativamente el error para ruta directa (a un 50–65% del error del Modelo 1). Para encaminamiento a través del *daemon*, la mejora es espectacular, de un orden de magnitud, reduciéndose el error a un 3.5–17% del presentado por el Modelo 1.

Aún así, la magnitud absoluta de los errores en este modelo (Tabla 2.16(a)) sigue siendo un orden de magnitud superior para la modalidad *daemon*. El siguiente modelo intentará centrar lo más perfectamente posible cada segmento rectilíneo del modelo sobre el trazo de las mediciones. Usará para ello parámetros adicionales, llevando al límite la capacidad de convergencia del método simplex utilizado.

Parámetros extraidos bajo LAM/MPI

Los ficheros de mínimos para cada una de las 4 combinaciones de opciones LAM, junto con los trazos ajustados por el Modelo 2, se presentan en las Figuras 2.44 (configuración coincidente con PVM) y 2.45 (configuración estándar LAM). Las gráficas agrupadas se muestran en las Figuras 2.46 y 2.47. La Tabla 2.17 resume los valores de parámetros extraidos. La comparación de errores cuadráticos con el Modelo 1 se realiza en la Tabla 2.18.

Se recuerda de nuevo que los parámetros $h1 \ldots sn$ (y por supuesto fs) están "congelados", no siendo modificados por el método simplex. Se han utilizado los valores que ya se mostraron en la Tabla 2.14.

Son ahora aplicables a LAM/MPI similares comentarios a los realizados para PVM. lc crece unos 12μs con ruta directa, y decrece unos 75–80μs a través del *daemon*, mientras que bs sube 0.5MB/s y baja 0.8–1.3MB/s en cada caso. El motivo vuelve a ser las ligazones directas e indirectas de ambos parámetros, mucho más exigentes bajo LAM en modo *daemon*, con una 2ª UDP que liga directamente ambos, como se indicó en el resumen de parámetros al principio del Apartado 2.3.7. No se requiere en este caso muy fino ojo para percibir el resultado de estas ligazones comparando las ampliaciones en las Figuras 2.46(b) y 2.47(b) con las 2.35(b) y 2.36(b), resultando abrumadoramente evidente que lc y bc tenían una mejor interpretación física en el Modelo 1.

lm detecta bastante bien el pronunciado salto entre UDPs en modo *daemon*, y tiene una interesante independencia (o casi independencia) del tamaño UDP que no se observaba bajo PVM.

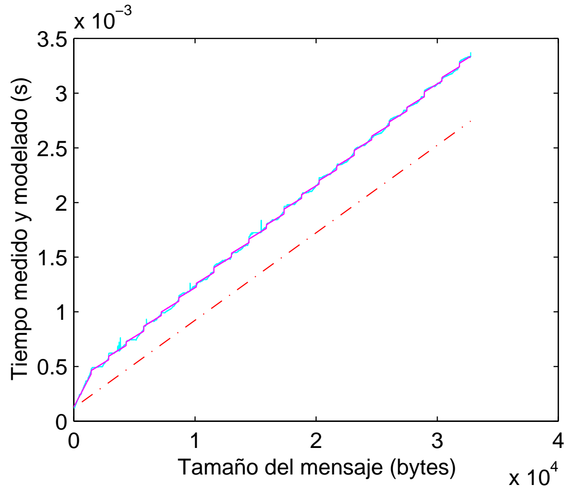
ls pierde el papel "bisagra" del Modelo 1. Se estabiliza a unos valores muy sensatos de 30–50μs, muy similares a los de PVM con ruta directa.

bs crece hasta por encima de los 12.5MB/s del *switch*, incluso en modalidad *daemon*. No se debe pensar sin embargo que el modo *daemon* LAM sea superior al PVM. bs tan sólo indica que las pendientes de los tramos modelados por él son inferiores a las de los correspondientes tramos PVM. La rápida acumulación de tiempo de transmisión en las dos primeras UDPs del *daemon* LAM contribuyen ciertamente a que el método simplex eleve el valor de bs.

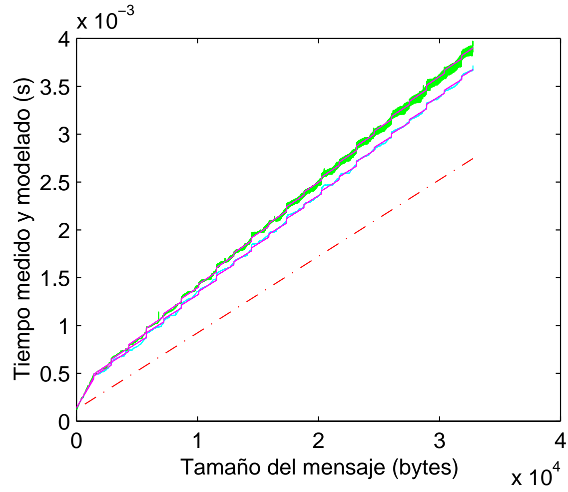
El Modelo 2 no es tan satisfactorio para LAM como lo fue para PVM, y en las Figuras 2.46(a) y 2.47(a) se puede comprender rápidamente la razón: hay tres tipos de pendientes claramente distintas en las mediciones, mientras que el modelo sólo puede contemplar dos. El siguiente modelo resolverá esta deficiencia añadiendo más parámetros, aún a riesgo de comprometer la convergencia del método simplex utilizado.

A pesar de ello, el Modelo 2 supera objetivamente al Modelo 1 también bajo LAM/MPI. Naturalmente, no se obtienen unas relaciones tan espectaculares como bajo PVM. En la Tabla 2.18 se cuantifica en un 33–40% la reducción del error al usar este modelo.

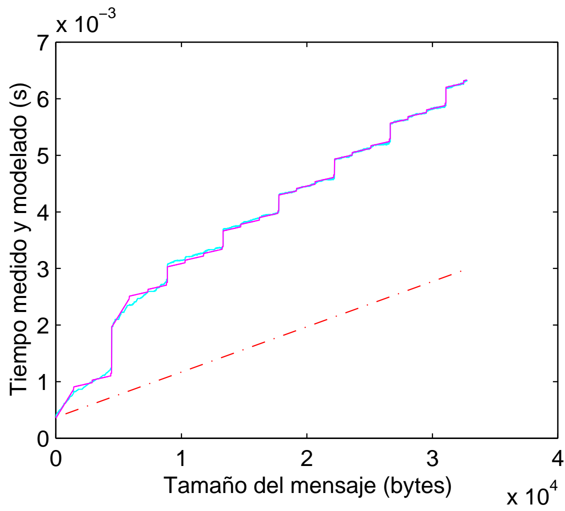
En cualquier caso el modelado está desequilibrado. El error de modelado para la modalidad *daemon* presenta una diferencia de casi *dos* órdenes de magnitud en comparación con la modalidad cliente-a-cliente, lo cual no es de extrañar recordando que en PVM el desequilibrio era de un orden de magnitud *después* de la espectacular mejora (de un orden de magnitud) en el error para la modalidad *daemon*.



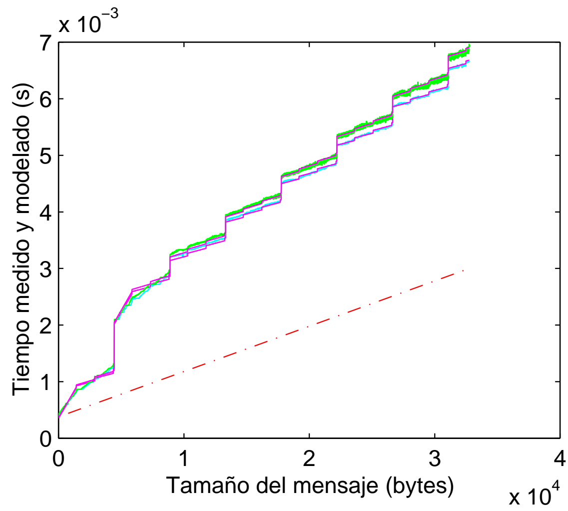
(a) Homogéneo, Cliente a cliente



(b) Heterogéneo, Cliente a cliente

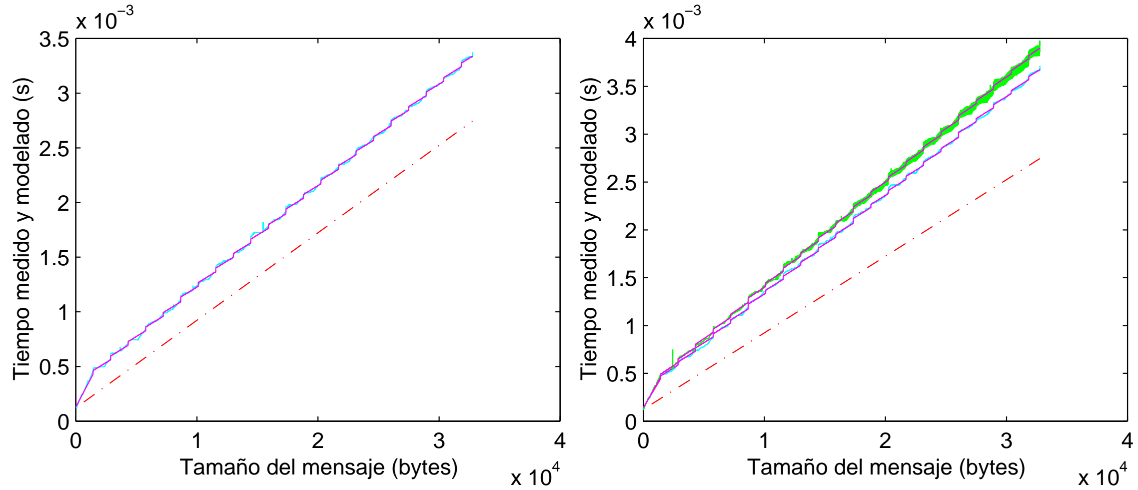


(c) Homogéneo, *daemon* LAM



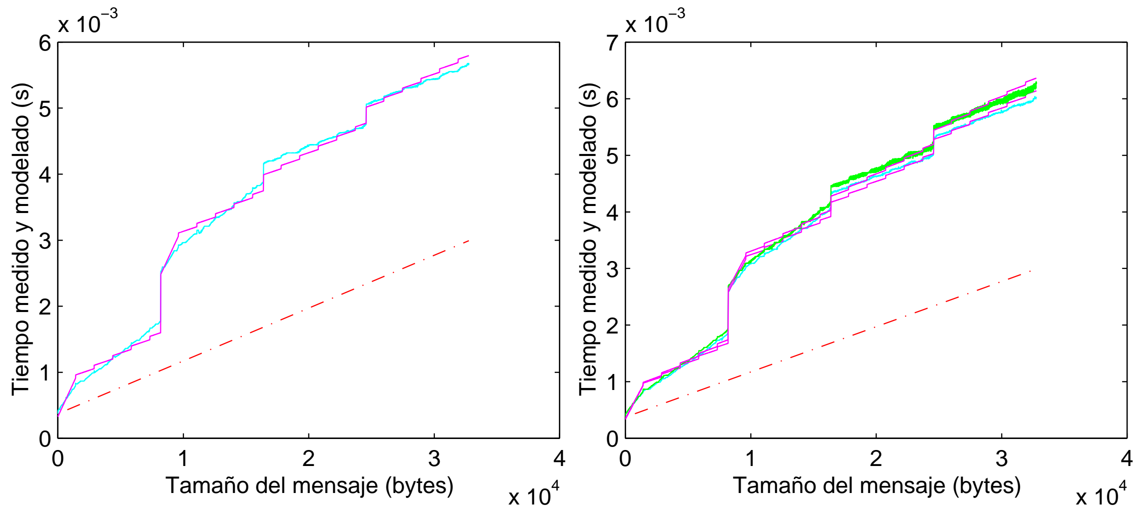
(d) Heterogéneo, *daemon* LAM

Figura 2.44: Modelo 2, configuración LAM coincidente con PVM MAXNMSGLEN=4K.



(a) Homogéneo, Cliente a cliente

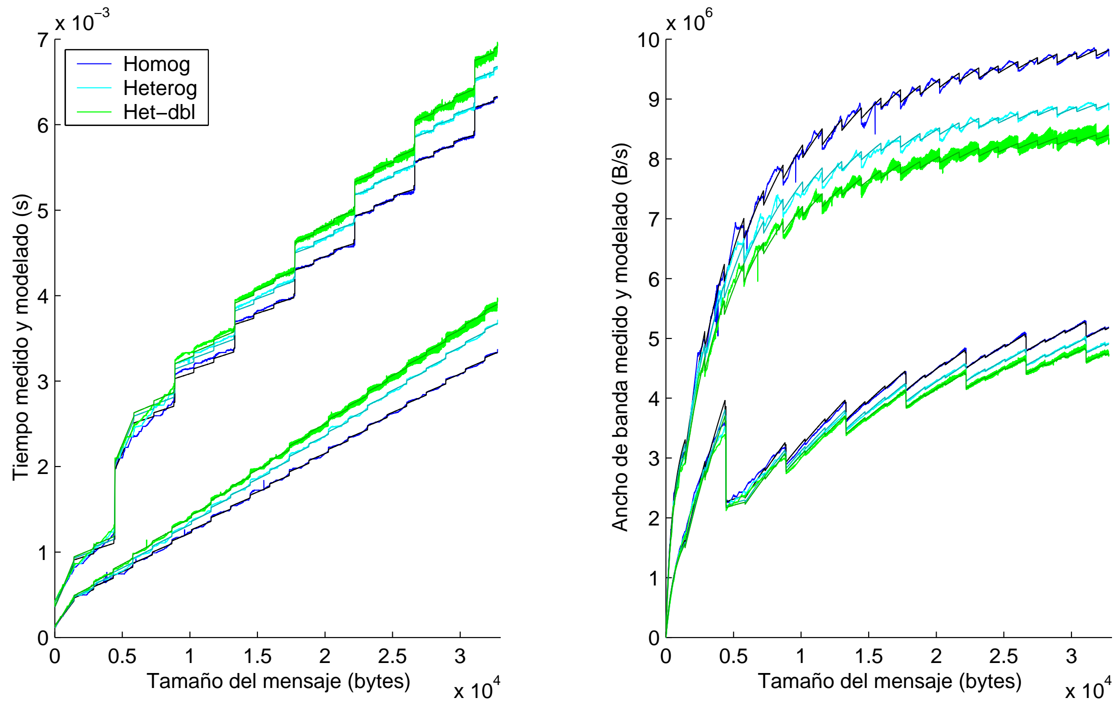
(b) Heterogéneo, Cliente a cliente



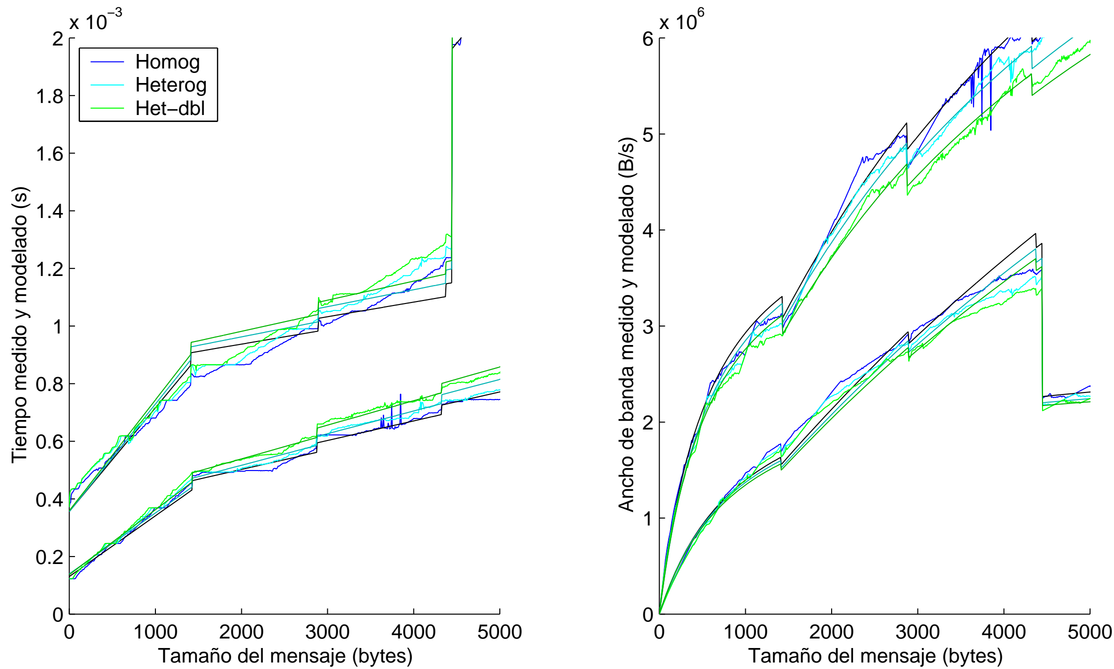
(c) Homogéneo, *daemon* LAM

(d) Heterogéneo, *daemon* LAM

Figura 2.45: Modelo 2, configuración estándar LAM MAXNMSGLEN=8K.

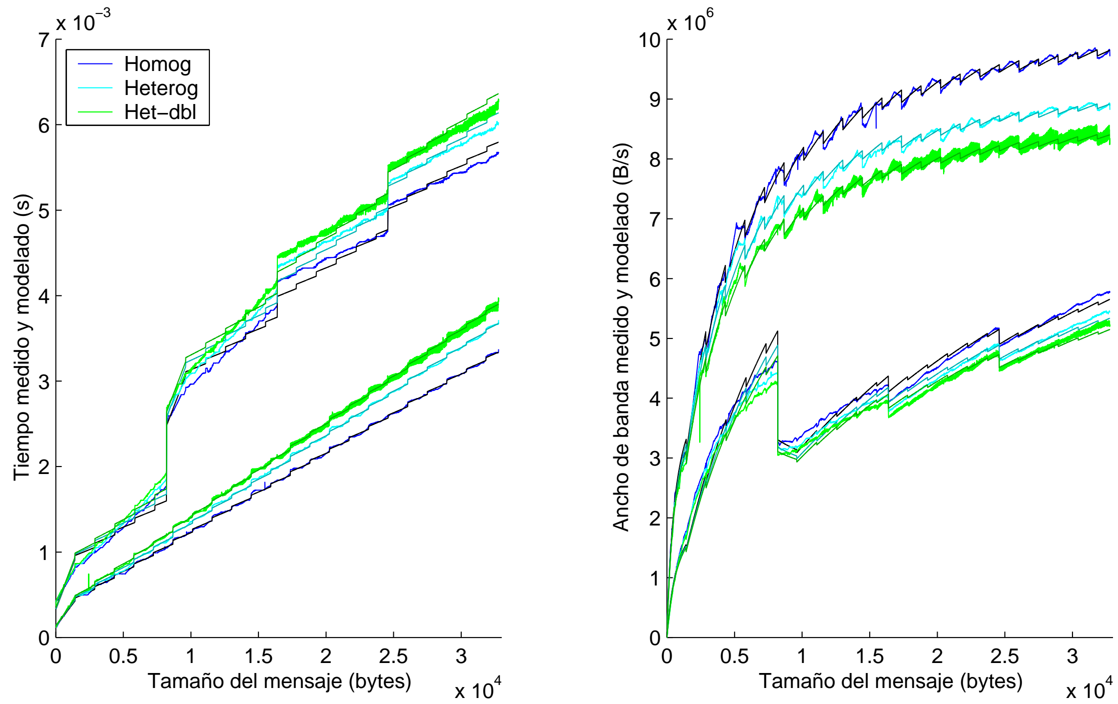


(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.

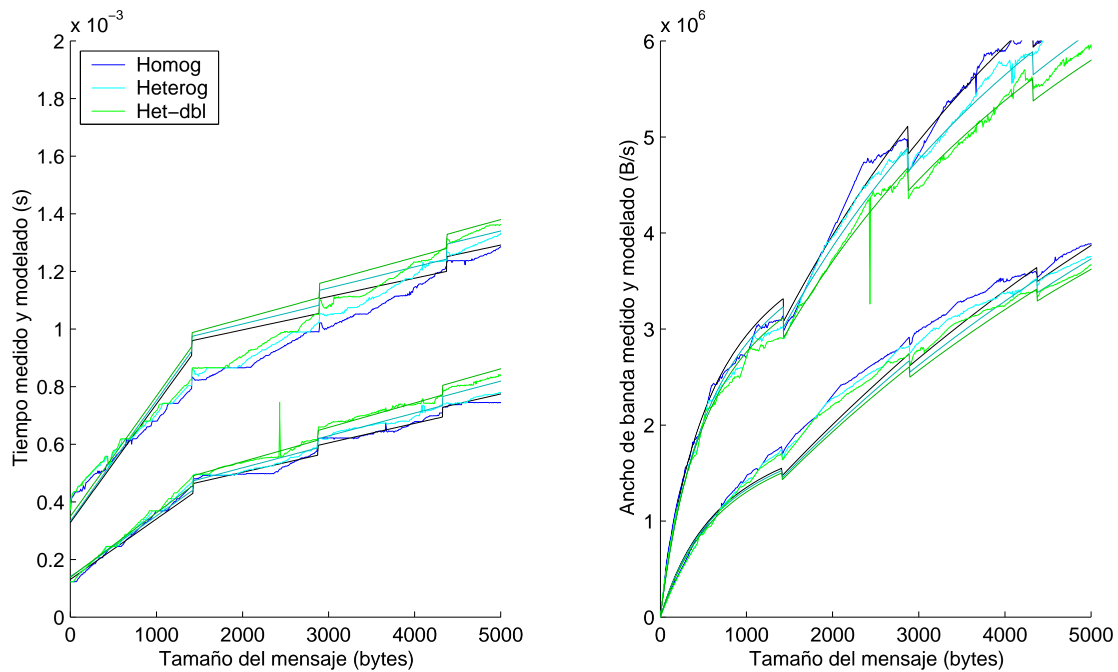


(b) Ampliación al rango de 5KB.

Figura 2.46: Modelo 2, Configuración LAM coincidente con PVM MAXNMSGLEN=4K: Agrupación comparativa de las 4 gráficas de la Figura 2.44.



(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.



(b) Ampliación al rango de 5KB.

Figura 2.47: Modelo 2, Configuración LAM estándar MAXNMSGLEN=8K: Agrupación comparativa de las 4 gráficas de la Figura 2.45.

	cliente-a-cliente					
Cluster	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
Homogéneo	130.633	4.748	44.698	32.707	14.735	7.0319E-07
Heterogéneo	139.777	4.735	48.040	30.836	12.684	9.5388E-07
Het, double	137.149	4.441	47.205	32.425	11.802	2.5387E-06
	***daemon* LAM**					
Cluster	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
Homogéneo	356.357	2.780	275.884	44.494	19.734	6.4094E-06
Heterogéneo	355.637	2.678	279.545	46.236	17.016	7.5077E-06
Het, double	359.310	2.602	289.986	42.311	15.178	8.7539E-06

(a) Configuración LAM coincidente con PVM (MAXNMSGLEN 4K)

	cliente-a-cliente					
Cluster	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
Homogéneo	130.940	4.769	44.210	34.191	14.739	7.0463E-07
Heterogéneo	140.304	4.736	50.011	31.923	12.651	8.9984E-07
Het, double	138.149	4.464	45.778	33.600	11.752	2.5360E-06
	***daemon* LAM**					
Cluster	lc (μs)	bc (MB/s)	lm (μs)	ls (μs)	bs (MB/s)	err
Homogéneo	328.096	2.428	243.412	51.278	15.629	3.4474E-05
Heterogéneo	333.461	2.390	253.039	51.683	13.670	3.1567E-05
Het, double	349.140	2.391	250.918	49.214	12.199	3.3371E-05

(b) Configuración LAM estándar (MAXNMSGLEN 8K)

Tabla 2.17: Parámetros del Modelo 2 ajustados a las diversas configuraciones de LAM.

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	$7.0319\ 10^{-7}$	$6.4094\ 10^{-6}$	$7.0463\ 10^{-7}$	$3.4474\ 10^{-5}$
Heterogéneo	$9.5388\ 10^{-7}$	$7.5077\ 10^{-6}$	$8.9984\ 10^{-7}$	$3.1567\ 10^{-5}$
Het, double	$2.5387\ 10^{-6}$	$8.7539\ 10^{-6}$	$2.5360\ 10^{-6}$	$3.3371\ 10^{-5}$

(a) Modelo 2

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	$1.1584\ 10^{-6}$	$2.8075\ 10^{-4}$	$1.1497\ 10^{-6}$	$4.7481\ 10^{-4}$
Heterogéneo	$1.4307\ 10^{-6}$	$2.9177\ 10^{-4}$	$1.3746\ 10^{-6}$	$4.8037\ 10^{-4}$
Het, double	$3.0223\ 10^{-6}$	$2.9042\ 10^{-4}$	$2.9858\ 10^{-6}$	$4.7135\ 10^{-4}$

(b) Modelo 1

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	1.647	43.803	1.632	13.773
Heterogéneo	1.500	38.863	1.528	15.217
Het, double	1.190	33.176	1.177	14.125

(c) Mejora relativa: error del Modelo 1 dividido entre el del Modelo 2

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	0.607	0.023	0.613	0.073
Heterogéneo	0.667	0.026	0.655	0.066
Het, double	0.840	0.030	0.849	0.071

(d) Reducción relativa: error del Modelo 2 dividido entre el del Modelo 1

Tabla 2.18: Errores cuadráticos medios de los Modelos 1 y 2 bajo las diversas configuraciones LAM.

2.3.8 Modelo 3

La más grave deficiencia detectada en el modelo anterior es que el modo *daemon* LAM necesita un tercer parámetro de ancho de banda, al que llamaremos bm (por no dejar desemparejado a lm). Se pueden consultar los (relativamente) elevados errores del modelo en la Tabla 2.17 o 2.18(a), mientras que un rápido vistazo a las filas inferiores de las Figuras 2.44 y 2.45 permite identificar la necesidad de este bm adicional. Las primera MTU quedaría modelada por la pareja lc/bc, las restantes MTUs de la primera y segunda UDPs por ls/bm, y las restantes UDPs por ls/bs (lm/bs para la 1ª MTU).

Para desligar directamente lc/bc es necesario añadir otra latencia lp que modele el pronunciado escalón entre 1ª y 2ª UDP. Se decidió añadir el correspondiente bp para comprobar si realmente la 1ª MTU de la 2ª UDP tiene la misma pendiente que el de la 1ª UDP.

El modo *daemon* PVM también se podría beneficiar del mayor número de parámetros disponibles. lc/bc quedarían desligados modelando sólo la 1ª MTU y recuperarían la interpretación física que tenían en el Modelo 1. La pareja lm/bm podría modelar la 1ª MTU en sucesivas UDPs, por comprobar si tienen la misma pendiente que la primera; con esta elección (en lugar de lp/bp) lm conserva su interpretación de "salto entre UDPs" que tenía en el Modelo 2. Disponiendo de dos parejas adicionales, se puede dedicar ls/bs a la última MTU de cada UDP, mientras que lp/bp modelarían las MTUs intermedias.

Un posible argumento para segregar la última MTU en lugar de otra cualquiera es que la última suele ser fragmentaria: es más probable que muestre un ancho de banda distinto debido al anteriormente comentado mecanismo de reconocimiento. No es tan plausible que una MTU intermedia presente un ancho de banda significativamente distinto del de otra MTU intermedia en la configuración UDPMAXLEN=8K (en 4K sólo hay una MTU intermedia).

Como se comentó en la discusión de parámetros del modelo anterior, los encaminamientos directos presentan más bien una alternancia de latencias (y de pendientes) que lm no era capaz de corregir, al ser aplicada cada $n = \frac{\|UDP\|}{\|MTU\|}$ MTUs. Correspondientemente, el nuevo modelo aprovecha sólamente el nuevo bm al objeto de desligar completamente la primera MTU (modelada por lc/bc) de las primeras MTUs en sucesivas UDPs (modeladas por lm/bm), quedando cada segunda MTU modelada por ls/bs.

El siguiente cuadro resume esquemáticamente la utilización de parámetros en cada modalidad.

$$\text{Modelo 3}\left\{\begin{array}{cccl} \overbrace{lcbc/lsbs}^{\text{ruta directa}} & \overbrace{lcbc/lpbp/\ldots/lsbs}^{\text{daemon PVM}} & \overbrace{lcbc/lsbm/\ldots}^{\text{daemon LAM}} & \text{1ª UDP}\\ lmbm/lsbs & lmbm/lpbp/\ldots/lsbs & lpbp/lsbm/\ldots & \text{2ª UDP}\\ \ldots & \ldots & lmbs/lsbs/\ldots & \text{otras}\end{array}\right.$$

Ya vimos en la Tabla 2.14 que los *daemons* quedan delatados por $h1 == 92$ (LAM) y $h1 == 44$ (PVM). Las modalidades directas que presentan alternancia de 2 tipos de MTU son justamente las delatadas por $h1 == 52$, esto es, LAM −c2c y *PvmDataInPlace/RouteDirect*.

Los restantes empaquetamientos con *PvmRouteDirect* presentan una cabecera $h1 == 68$ y se adecúan al mismo esquema del modo *daemon* PVM. De hecho, los valores $hn == 348, 580$ revelan que el mecanismo de fragmentación del *daemon* PVM está activo. Una comprobación visual en el GUI ratifica que estos modos no presentan alternancia de dos tipos de MTU, sino el habitual esquema del *daemon* PVM de periodicidad $n = \frac{\|UDP\|}{\|MTU\|}$.

El código MATLAB para evaluar este modelo, de una complejidad similar al anterior, puede consultarse en el Apéndice B.5.

Conviene notar que al aumentar la dimensionalidad del espacio de parámetros, el número de iteraciones para la convergencia del método crece significativamente. Las propiedades de convergencia del método simplex han sido estudiadas (ver [40]) sólo para un número reducido de dimensiones. Con 8 parámetros como los utilizados en este modelo es aún posible hacer converger el método a valores estables de los mismos.

Parámetros extraidos bajo PVM

Los ficheros de mínimos para las 6 combinaciones de opciones PVM, junto con los trazos ajustados por el Modelo 3, se presentan en las Figuras 2.48 (configuración estándar PVM) y 2.49 (configuración coincidente con LAM).

La Figura 2.50 agrupa las 6 subgráficas de la configuración estándar, y la Figura 2.51 las de la coincidente con LAM. Las predicciones del modelo aparecen en un tono algo más oscuro del mismo color que la correspondiente opción PVM.

La Tabla 2.19 resume los valores de parámetros extraidos, mientras que la comparación de los errores cuadráticos medios con los del modelo anterior se realiza en la Tabla 2.20. Los parámetros de cabecera son los mismos que en el Modelo 2.

Los parámetros lc y bc vuelven a variar algo, pudiéndose comprobar en las Figuras 2.50(b) y 2.51(b) izquierda el mejor ajuste conseguido al eliminar las ligaduras directas que sufrían.

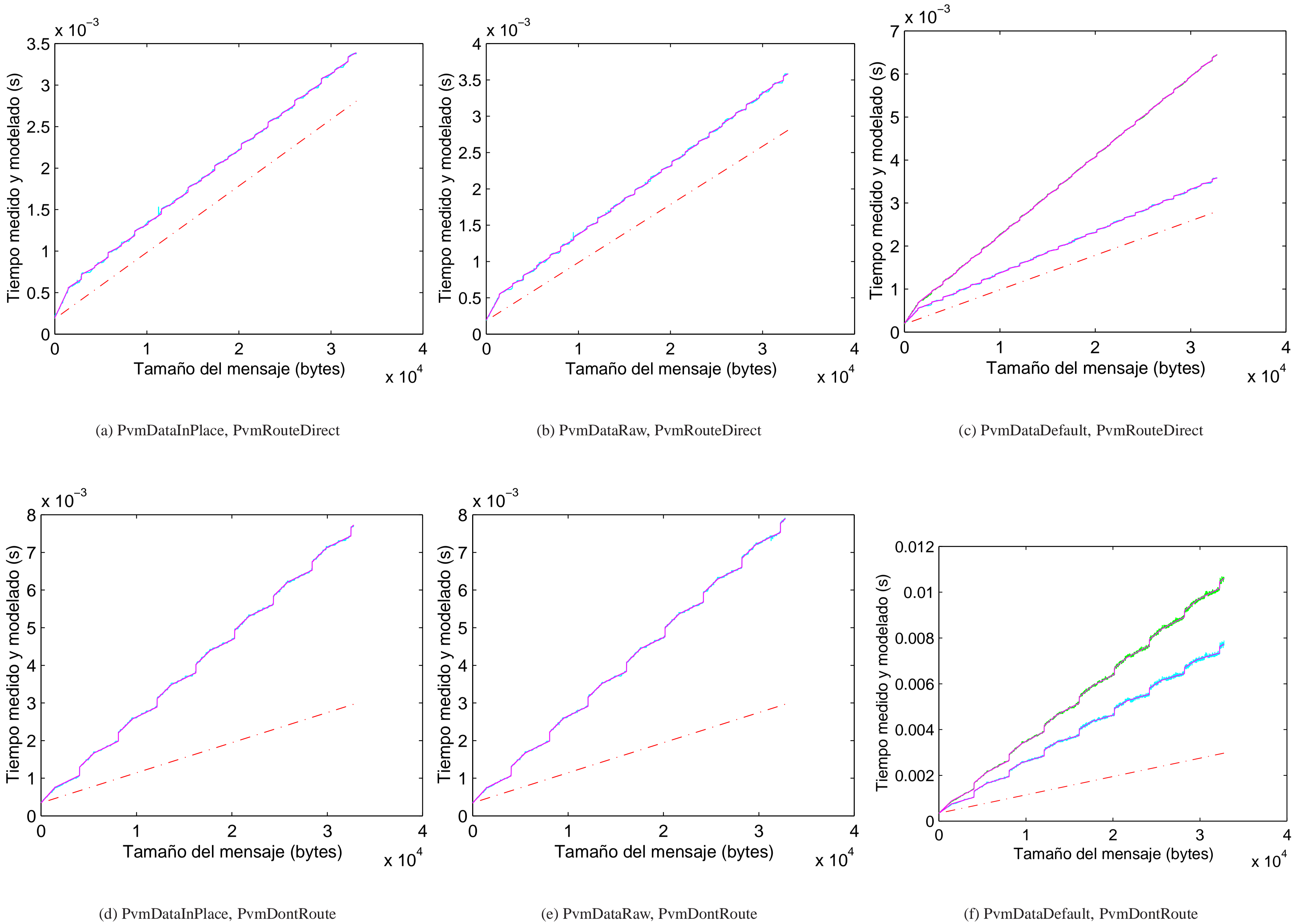
El salto entre UDPs en modo *daemon*, lm, es bastante menor (hasta 65μs) con UDPMAXLEN=8K. El correspondiente ancho de banda (bm, asociado a la 1ª MTU de cada UDP) crece 0.2–0.5MB/s al cambiar de 4K a 8K. Esto indica que los costos adicionales por cada nueva UDP son menores en la configuración coincidente con LAM.

Con ruta directa, la pareja lm/bm modela la primera de cada dos MTUs. lm es bastante independiente del tamaño de fragmento UDP y ronda los 45–55μs. bm muestra una tendencia inversa a lo visto en modo *daemon*, modelando primeras MTUs más lentas bajo la configuración coincidente.

Respecto a la pareja lp/bp, que modela las MTUs intermedias en modo *daemon*, el parámetro de latencia podría haberse ignorado para esta modalidad, ya que toma valores próximos a la resolución de gettimeofday(), a veces incluso negativos. Esto indica un esfuerzo conducente a reducirlo a 0 por parte del método simplex. Los anchos de banda son casi el doble que bm, indicando de nuevo que la copia de memoria y paso por el *daemon* PVM consumen casi tanto tiempo como la transmisión de datos, o algo más para el caso de doubles.

lp/bp también eran aplicables a ruta directa salvo para el empaquetamiento *InPlace*, modelado con la alternancia ya comentada. lp es bastante menor que lm como cabía esperar. bp presenta un aparente intercambio con bm en la configuración 4K con ruta directa, siendo el único caso en que es menor que bm. Con UDP de 8K vuelve a comportarse como en el modo *daemon*. Una consulta al error cuadrático para UDPMAXLEN=4K, *RouteDirect*, *DataRaw/Default*) aconseja no dedicar mayor esfuerzo de refinamiento para esta combinación de opciones.

La pareja ls/bs modela las últimas MTUs de cada UDP en modo *daemon*, resultando de nuevo que la latencia debería haberse ignorado, bajo 4K porque su valor correcto es 0, y bajo 8K porque no se estabiliza, mostrando valores muy dispares según el empaquetamiento utilizado.



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

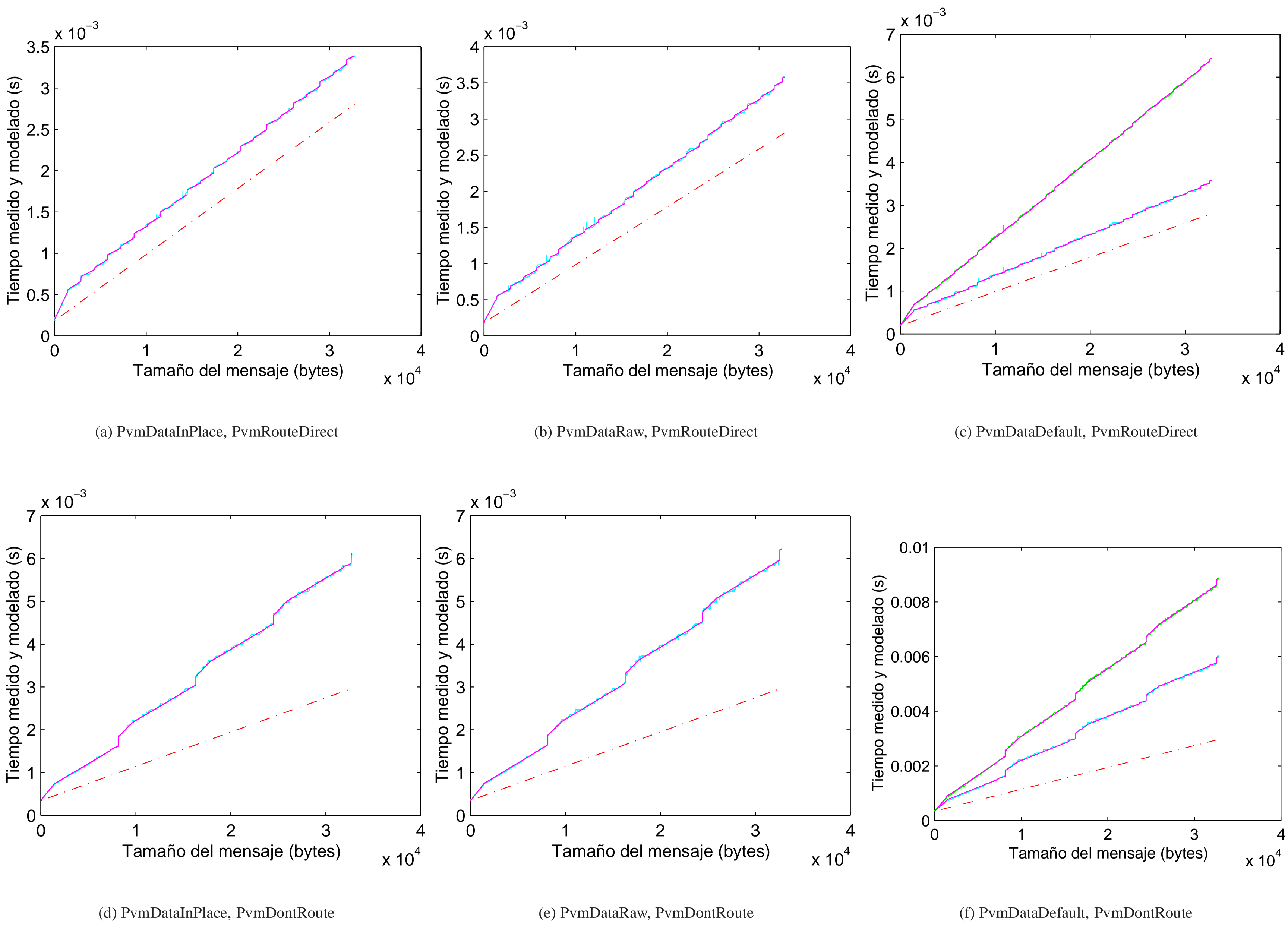
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.48: Modelo 3, configuración estándar PVM (UDPMAXLEN 4K).



(a) PvmDataInPlace, PvmRouteDirect

(b) PvmDataRaw, PvmRouteDirect

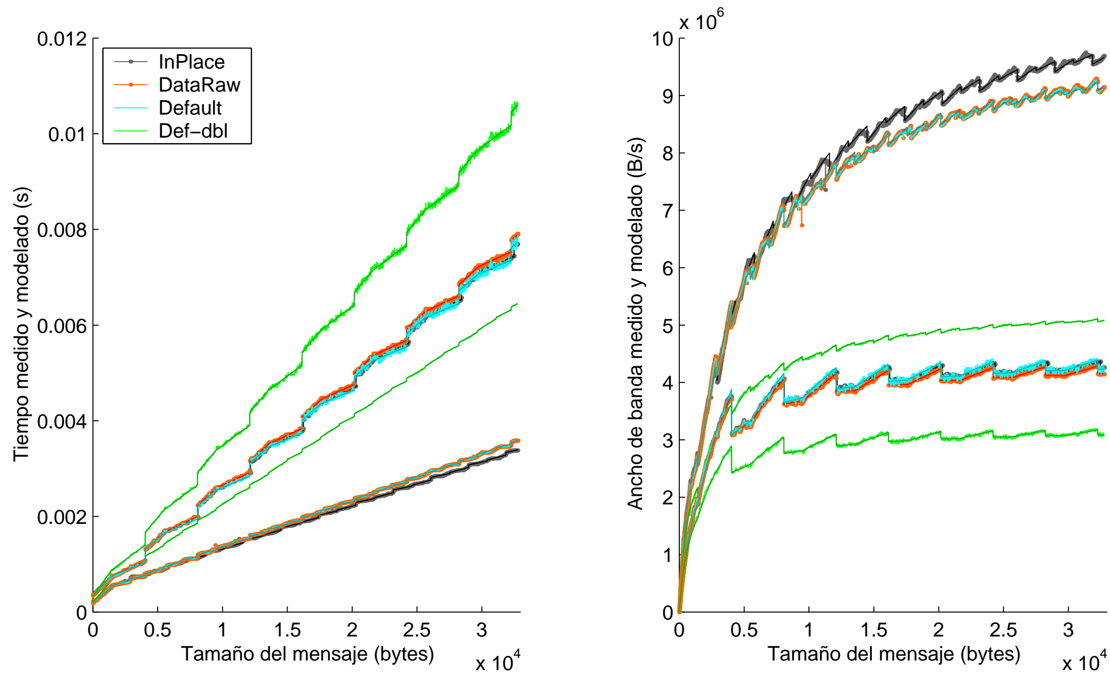
(c) PvmDataDefault, PvmRouteDirect

(d) PvmDataInPlace, PvmDontRoute

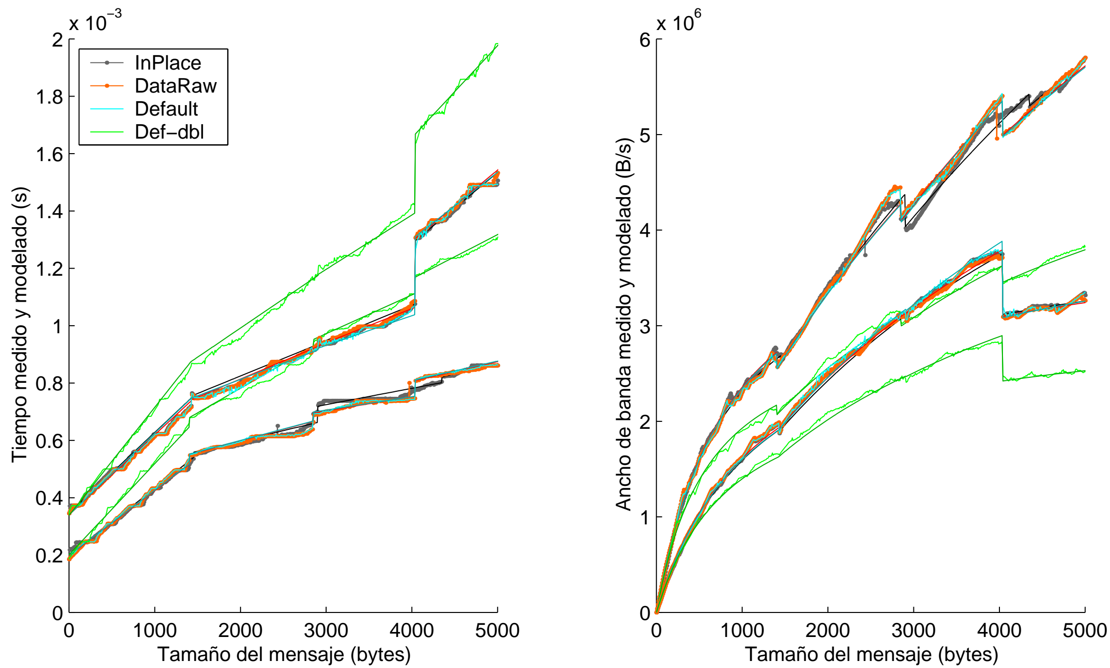
(e) PvmDataRaw, PvmDontRoute

(f) PvmDataDefault, PvmDontRoute

Figura 2.49: Modelo 3, configuración coincidente con LAM (UDPMAXLEN 8K).

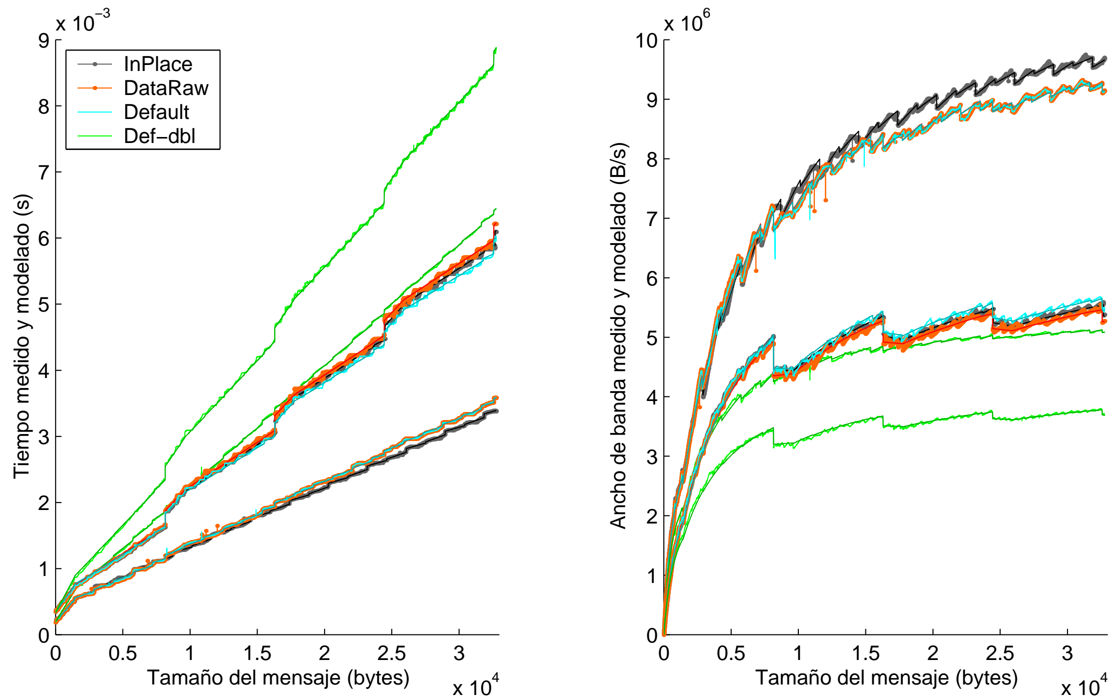


(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.

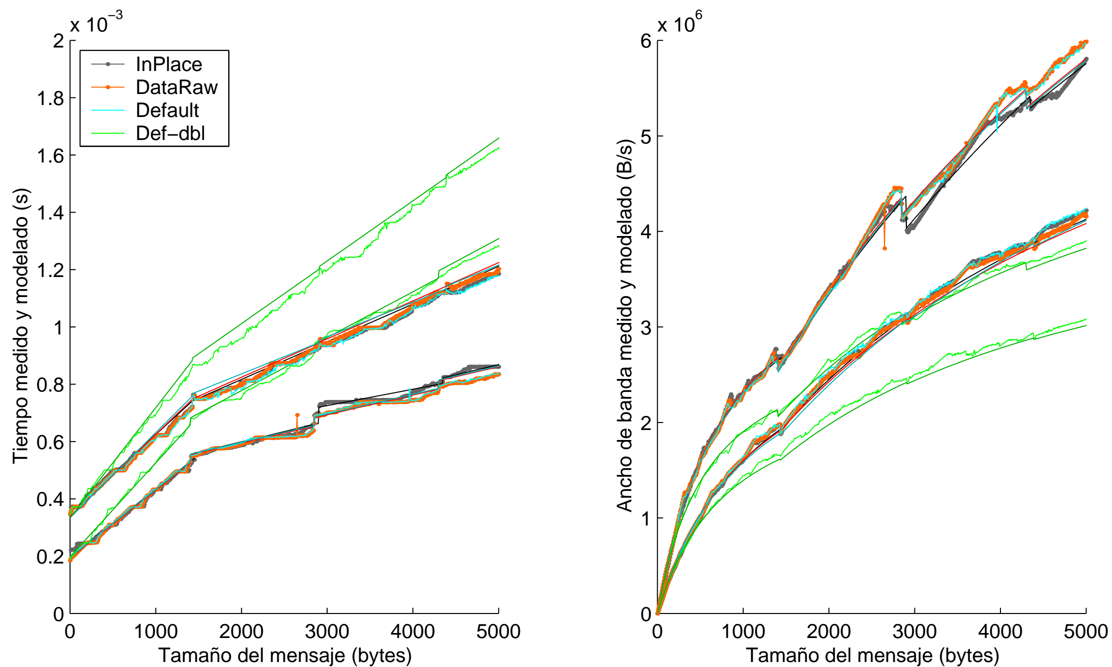


(b) Ampliación al rango de 5KB.

Figura 2.50: Modelo 3, Configuración estándar PVM UDPMAXLEN=4K: Agrupación comparativa de las 6 gráficas de la Figura 2.48.



(a) Característica de prestaciones y predicción del modelo para las 6 combinaciones de opciones PVM.



(b) Ampliación al rango de 5KB.

Figura 2.51: Modelo 3, Configuración PVM coincidente con LAM UDPMAXLEN=8K: Agrupación comparativa de las 6 gráficas de la Figura 2.49.

	PvmRouteDirect								
Empaq.	lc (μs)	bc (MB/s)	lm (μs)	bm (MB/s)	lp (μs)	bp (MB/s)	ls (μs)	bs (MB/s)	err
InPlace	184.053	4.223	53.426	17.560	—	—	15.521	13.937	3.9288E-07
DataRaw	175.540	4.420	49.983	14.672	31.781	12.532	28.239	20.297	4.5114E-07
Default	177.079	4.416	52.139	14.658	31.054	12.522	30.253	21.520	4.3209E-07
íd, dbl	169.642	3.151	46.078	6.478	24.172	5.817	21.161	7.396	3.1503E-07

	PvmDontRoute								
Empaq.	lc (μs)	bc (MB/s)	lm (μs)	bm (MB/s)	lp (μs)	bp (MB/s)	ls (μs)	bs (MB/s)	err
InPlace	325.060	3.554	208.092	4.028	4.782	8.195	-0.545	8.144	1.1074E-06
DataRaw	319.113	3.584	227.647	4.018	1.379	7.954	0.622	8.444	9.0919E-07
Default	319.094	3.514	259.666	4.266	-1.149	8.769	4.098	9.627	6.4647E-06
íd, dbl	309.748	2.653	259.647	3.113	-4.380	4.927	0.935	5.278	5.9446E-06

(a) Configuración PVM estándar (UDPMAXLEN 4K)

	PvmRouteDirect								
Empaq.	lc (μs)	bc (MB/s)	lm (μs)	bm (MB/s)	lp (μs)	bp (MB/s)	ls (μs)	bs (MB/s)	err
InPlace	184.908	4.222	53.417	17.492	—	—	15.693	13.992	3.8783E-07
DataRaw	170.849	4.248	48.762	12.554	26.085	14.520	45.609	16.767	9.2255E-07
Default	172.504	4.231	51.913	12.733	25.757	14.496	47.585	17.811	9.3303E-07
íd, dbl	165.413	3.055	46.171	5.924	17.261	6.264	43.246	7.468	9.0487E-07

	PvmDontRoute								
Empaq.	lc (μs)	bc (MB/s)	lm (μs)	bm (MB/s)	lp (μs)	bp (MB/s)	ls (μs)	bs (MB/s)	err
InPlace	329.851	3.652	200.053	4.342	-2.065	7.617	15.701	10.009	2.0808E-06
DataRaw	316.641	3.487	212.907	4.597	-1.216	7.553	0.680	7.215	2.5530E-06
Default	311.096	3.360	197.940	4.564	6.893	8.307	32.913	10.819	3.3829E-06
íd, dbl	304.328	2.598	193.151	3.363	4.606	4.765	16.988	4.977	3.1001E-06

(b) Configuración PVM coincidente con LAM (UDPMAXLEN 8K)

Tabla 2.19: Parámetros del Modelo 3 ajustados a las diversas configuraciones de PVM.

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	$3.9288\ 10^{-7}$	$1.1074\ 10^{-6}$	$3.8783\ 10^{-7}$	$2.0808\ 10^{-6}$
PvmDataRaw	$4.5114\ 10^{-7}$	$9.0919\ 10^{-7}$	$9.2255\ 10^{-7}$	$2.5530\ 10^{-6}$
PvmDataDefault	$4.3209\ 10^{-7}$	$6.4647\ 10^{-6}$	$9.3303\ 10^{-7}$	$3.3829\ 10^{-6}$
ídem, double	$3.1503\ 10^{-7}$	$5.9446\ 10^{-6}$	$9.0487\ 10^{-7}$	$3.1001\ 10^{-6}$

(a) Modelo 3

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	$7.6634\ 10^{-7}$	$1.2179\ 10^{-6}$	$7.4893\ 10^{-7}$	$2.2668\ 10^{-6}$
PvmDataRaw	$5.8595\ 10^{-7}$	$9.9444\ 10^{-7}$	$9.7720\ 10^{-7}$	$2.9399\ 10^{-6}$
PvmDataDefault	$5.7723\ 10^{-7}$	$6.7030\ 10^{-6}$	$9.8732\ 10^{-7}$	$3.9256\ 10^{-6}$
ídem, double	$4.7444\ 10^{-7}$	$6.2309\ 10^{-6}$	$9.3206\ 10^{-7}$	$3.7009\ 10^{-6}$

(b) Modelo 2

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	1.951	1.100	1.931	1.089
PvmDataRaw	1.299	1.094	1.059	1.152
PvmDataDefault	1.336	1.037	1.058	1.160
ídem, double	1.506	1.048	1.030	1.194

(c) Mejora relativa: error del Modelo 2 dividido entre el del Modelo 3

	UDPMAXLEN 4K		UDPMAXLEN 8K	
	PvmRouteDirect	**PvmDontRoute**	**PvmRouteDirect**	**PvmDontRoute**
PvmDataInPlace	0.513	0.909	0.518	0.918
PvmDataRaw	0.770	0.914	0.944	0.868
PvmDataDefault	0.749	0.964	0.945	0.862
ídem, double	0.664	0.954	0.971	0.838

(d) Reducción relativa: error del Modelo 3 dividido entre el del Modelo 2

Tabla 2.20: Errores cuadráticos medios de los Modelos 2 y 3 bajo las diversas configuraciones PVM.

Esto sí es una indicación de que el modelo es susceptible de mayor refinamiento, en vista de los errores cuadráticos de la modalidad *daemon* en general. bs es mucho mayor que bm bajo 4K (rondando el doble), lo cual se anticipaba, y sufre la misma inestabilidad que ls bajo 8K, aunque el error cuadrático es "paradójicamente" menor. (Con tantos parámetros, el método simplex tiene multitud de opciones para corregir un parámetro no adecuado).

ls/bs con *RouteDirect*/*InPlace* modelaban las segundas MTUs alternantes. Al igual que sucedía con lm/bm (primeras MTUs alternantes), son casi independientes del tamaño UDP y modelan una 2ª MTU más lenta (menor ancho de banda). Con los otros empaquetamientos tienen la misma interpretación que en modo *daemon*, modelando una última MTU más rápida, como se esperaba y como queda confirmado en los correspondientes errores cuadráticos.

Por fijar ideas, el modelo podría refinarse en los casos en que no se cumple la esperada relación $lc \geq lm \geq lp \geq ls$, que básicamente es el modo *daemon* a causa de $lp = 0$ y ls inestable. Los errores cuadráticos, de 9 10^{-7} a 6.5 10^{-6}, delatan esta condición.

A pesar de los poco halagüeños comentarios, el modelo es ampliamente superior, como un vistazo a las Figuras 2.50 y 2.51 revela. Consultando la Tabla 2.20, este modelo ha conseguido reducir el error cuadrático medio de *RouteDirect/InPlace* al 51–52% (debido al esquema de MTUs alternantes), y el de *RouteDirect/Raw/Default* con UDPMAXLEN=4K al 65–75% (debido al esquema *daemon*).

Las demás combinaciones sólo mejoran en un 5–15%, siendo el objetivo natural de posibles refinamientos del modelo. Es prudente advertir sin embargo que con un espacio de parámetros tan versátil el método simplex resulta estorbado en su tarea de escoger una trayectoria descendente en la superficie de error. 8 parámetros variables supone un elevado grado de libertad, y añadir parámetros sólo complicaría aún más la situación.

Parámetros extraidos bajo LAM/MPI

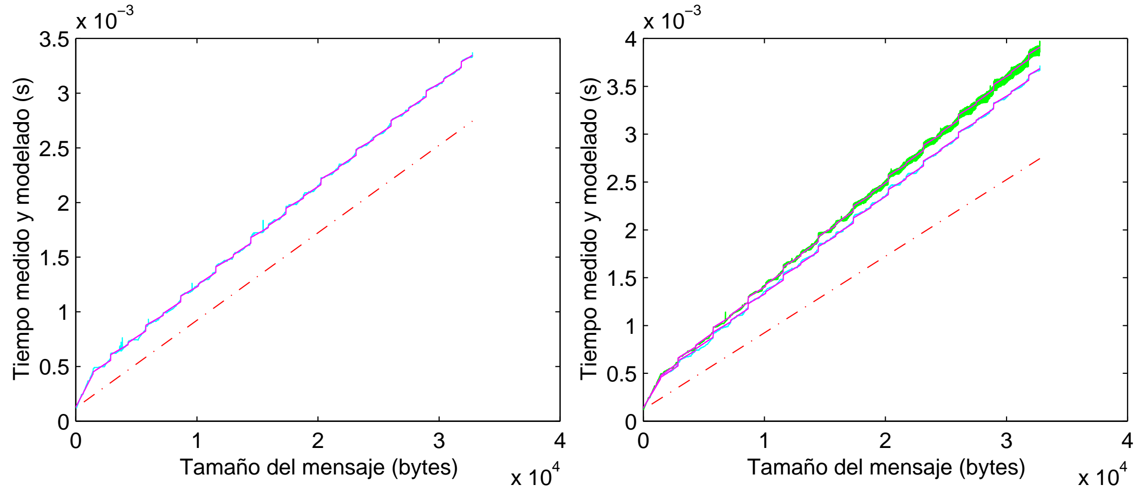
Los ficheros de mínimos para cada una de las 4 combinaciones de opciones LAM, junto con los trazos ajustados por el Modelo 3, se presentan en las Figuras 2.52 (configuración coincidente con PVM) y 2.53 (configuración estándar LAM). Las gráficas agrupadas se muestran en las Figuras 2.54 y 2.55. La Tabla 2.21 resume los valores de parámetros extraidos. La comparación de errores cuadráticos con el Modelo 2 se realiza en la Tabla 2.22.

Un vistazo preliminar a los nuevos errores cuadráticos en la Tabla 2.21 nos anticipa que el resultado va a ser de similar calidad al obtenido con PVM, tal vez ligeramente superior, con errores mínimos de 4.3 10^{-7} y máximos de 2.6 10^{-6}.

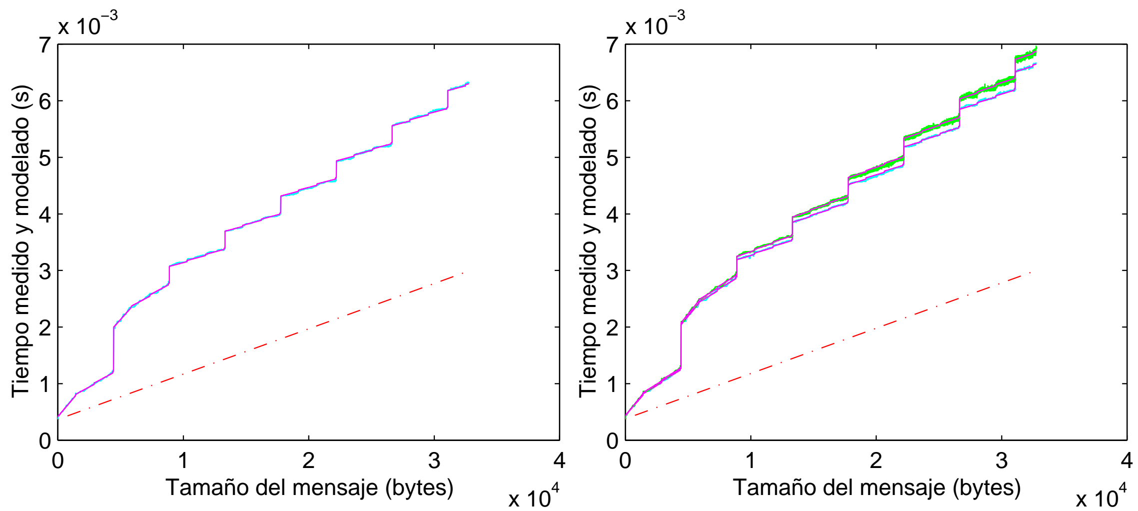
lb/bc se independizan prácticamente de la configuración MAXNMSGLEN, como sucedió bajo PVM. Muy notablemente, lo mismo sucede con *todos* los parámetros bajo ruta directa $-$c2c.

Una propiedad notable del modelado del *daemon* LAM es que *todos* los parámetros corresponden a prestaciones muy poco inferiores bajo la configuración estándar: las latencias se incrementan entre 4 y 15μs (salvo la inestabilidad de ls bajo MAXNMSGLEN=8K que también exhibía PVM) y los anchos de banda se decrementan 0.1–0.6MB/s con algunas excepciones (bp doubles aumenta 0.2, bs llega a bajar 2.1MB/s, y por supuesto bc oscila $\pm$0.1MB/s)

Son cantidades muy modestas, y se puede concluir que LAM es muy adecuado al tipo de modelado que hemos realizado, mostrando en general valores de los parámetros muy estabilizados al variar la combinación de opciones.

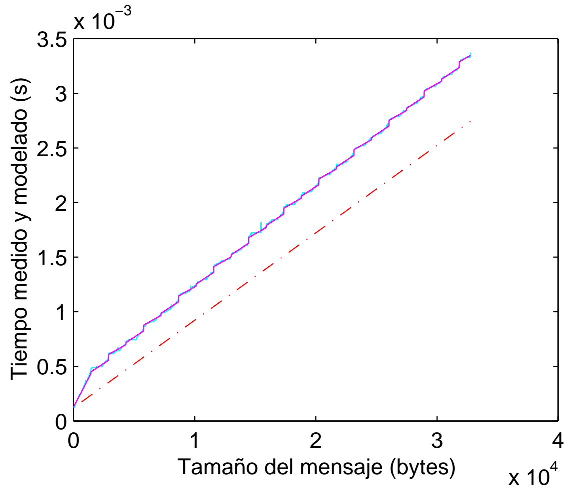


(a) Homogéneo, Cliente a cliente (b) Heterogéneo, Cliente a cliente

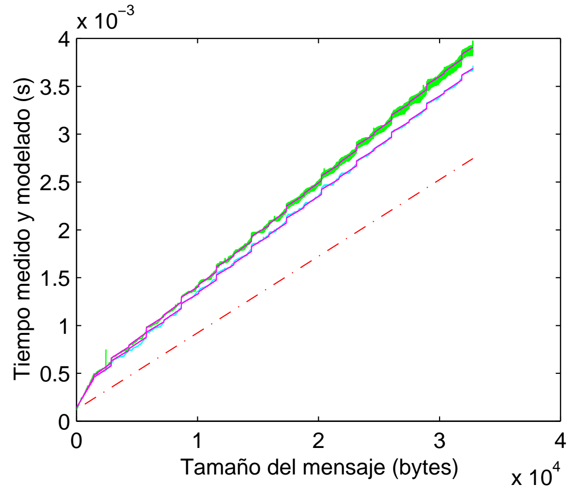


(c) Homogéneo, *daemon* LAM (d) Heterogéneo, *daemon* LAM

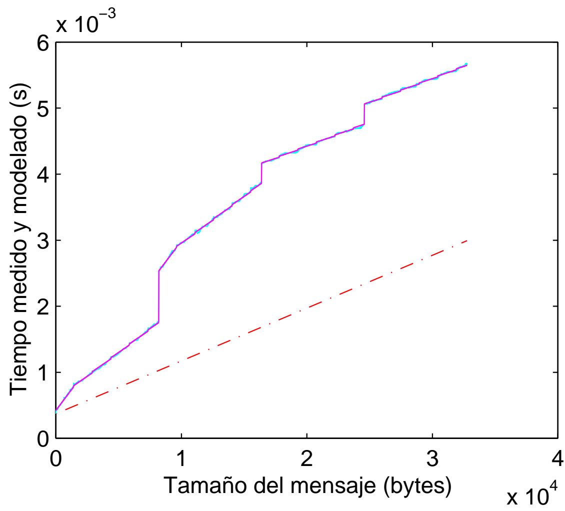
Figura 2.52: Modelo 3, configuración LAM coincidente con PVM MAXNMSGLEN=4K.



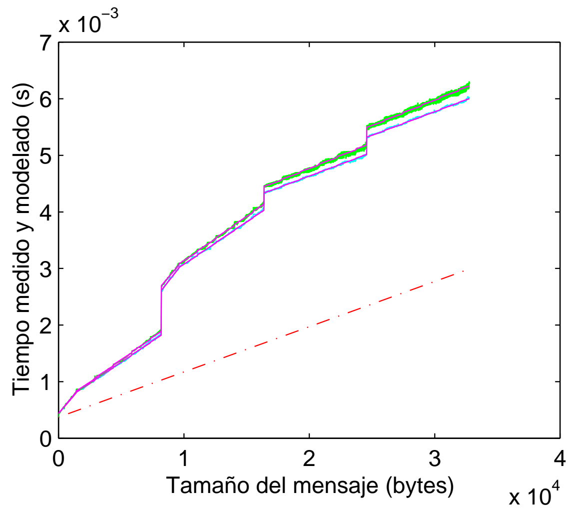
(a) Homogéneo, Cliente a cliente



(b) Heterogéneo, Cliente a cliente

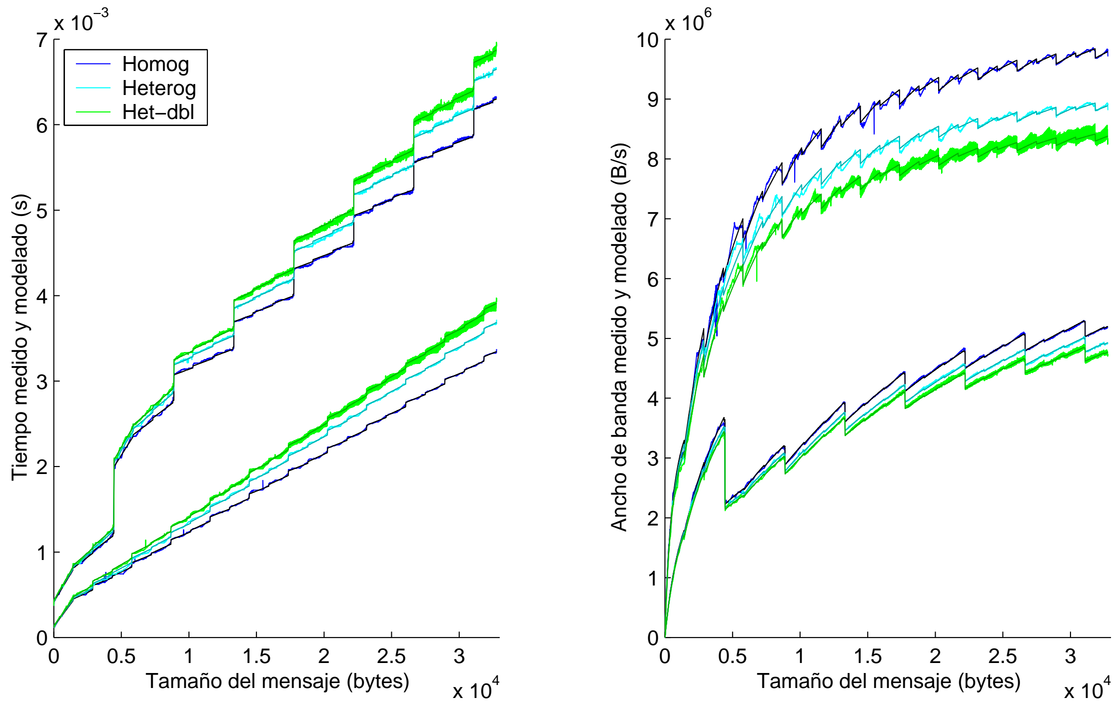


(c) Homogéneo, *daemon* LAM

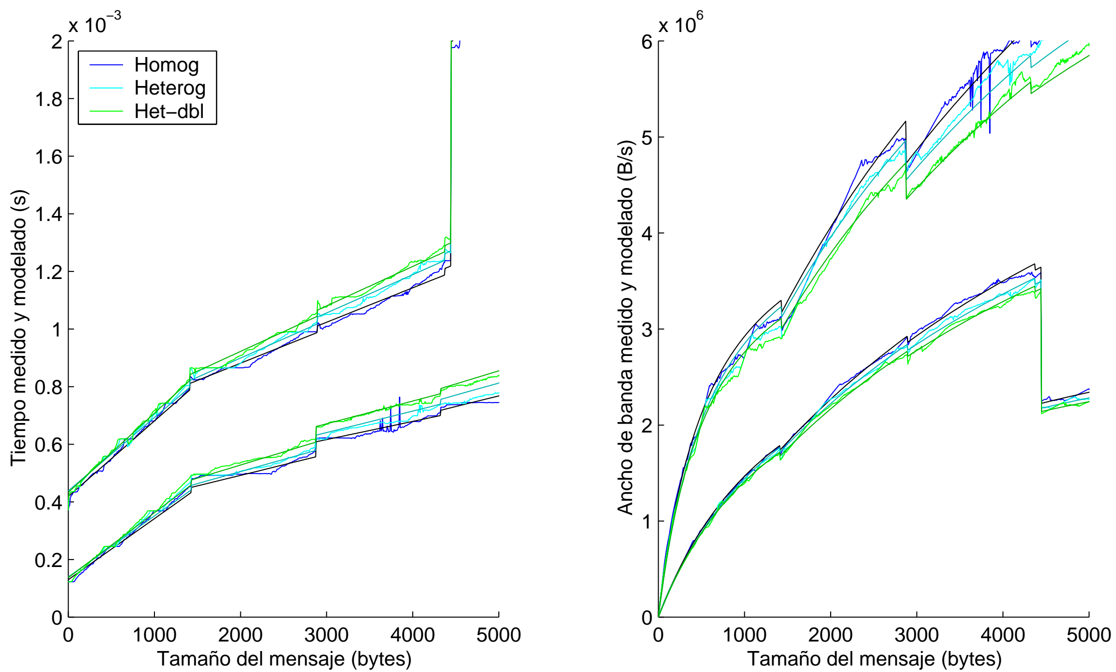


(d) Heterogéneo, *daemon* LAM

Figura 2.53: Modelo 3, configuración estándar LAM MAXNMSGLEN=8K.

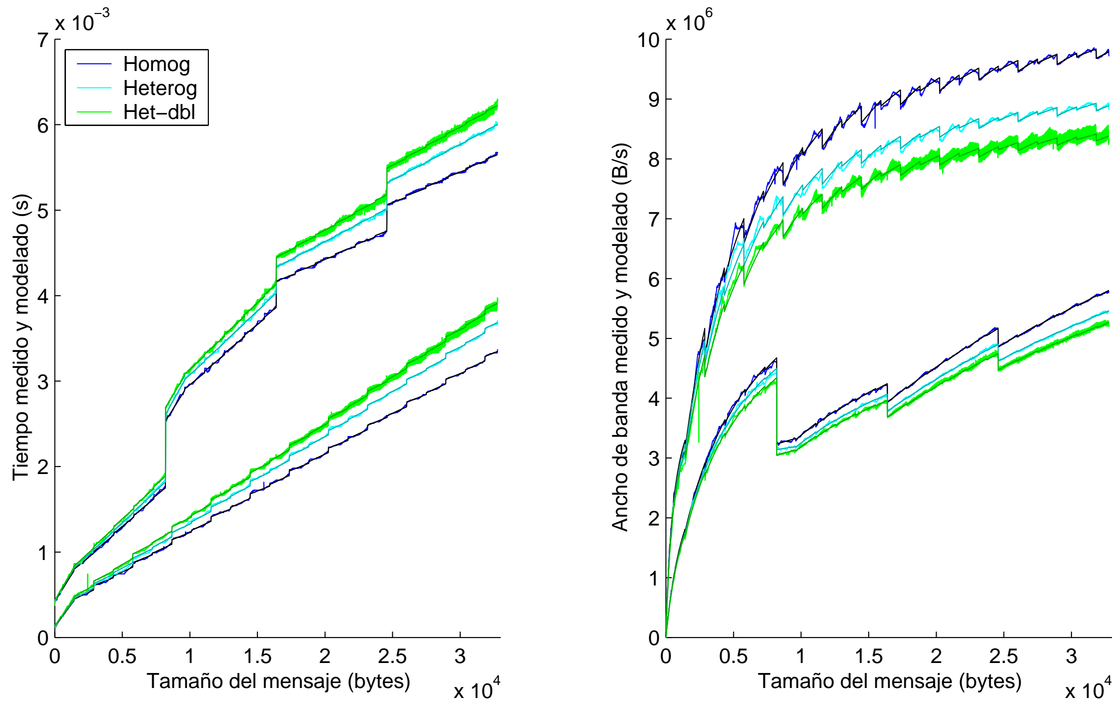


(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.

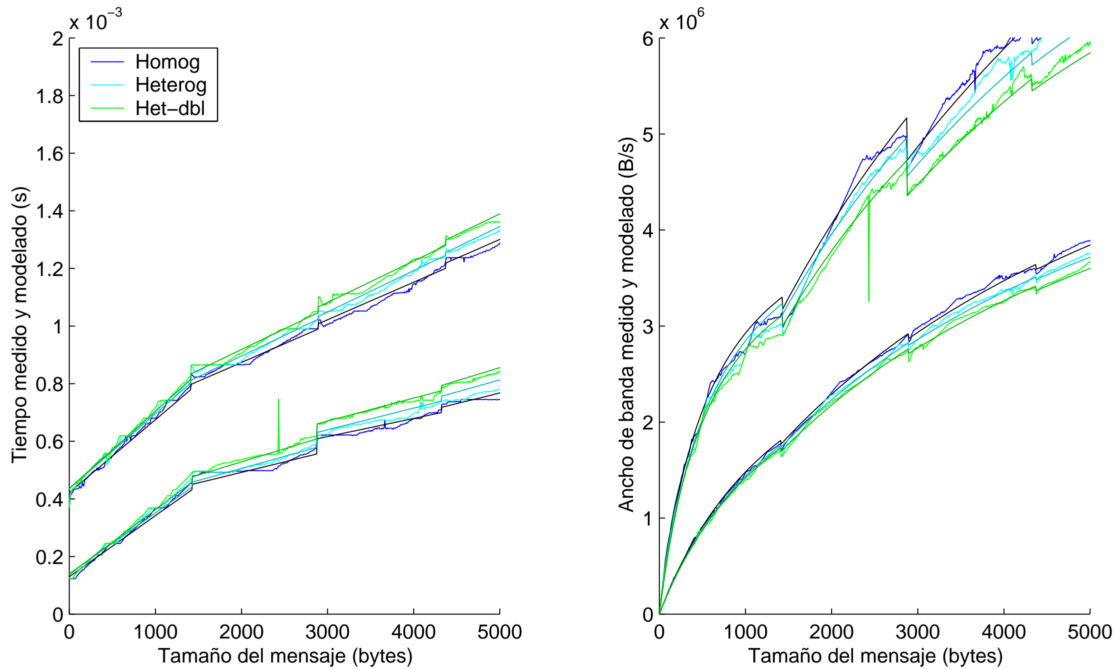


(b) Ampliación al rango de 5KB.

Figura 2.54: Modelo 3, Configuración LAM coincidente con PVM MAXNMSGLEN=4K: Agrupación comparativa de las 4 gráficas de la Figura 2.52.



(a) Característica de prestaciones y predicción del modelo para las 4 combinaciones de opciones LAM.



(b) Ampliación al rango de 5KB.

Figura 2.55: Modelo 3, Configuración LAM estándar MAXNMSGLEN=8K: Agrupación comparativa de las 4 gráficas de la Figura 2.53.

Cluster	lc (μs)	bc (MB/s)	lm (μs)	bm (MB/s)	lp (μs)	bp (MB/s)	ls (μs)	bs (MB/s)	err
	cliente-a-cliente								
Homog.	113.926	4.719	48.937	15.860	—	—	14.572	13.667	4.4595E-07
Heterog.	123.609	4.731	48.769	13.419	—	—	12.128	11.923	7.0374E-07
Het, dbl	119.950	4.438	49.478	12.650	—	—	12.897	10.985	2.2600E-06
	***daemon* LAM**								
Homog.	394.690	3.814	290.831	8.369	753.672	4.008	20.348	17.038	9.4541E-07
Heterog.	414.919	3.924	295.012	7.518	747.355	3.784	20.737	14.928	9.2361E-07
Het, dbl	407.841	3.638	304.243	7.304	750.784	3.650	18.394	13.654	2.7113E-06

(a) Configuración LAM coincidente con PVM (MAXNMSGLEN 4K)

Cluster	lc (μs)	bc (MB/s)	lm (μs)	bm (MB/s)	lp (μs)	bp (MB/s)	ls (μs)	bs (MB/s)	err
	cliente-a-cliente								
Homog.	113.881	4.723	49.191	15.882	—	—	14.579	13.661	4.2999E-07
Heterog.	124.392	4.742	47.662	13.301	—	—	12.422	11.931	6.8412E-07
Het, dbl	120.723	4.449	47.928	12.560	—	—	13.379	10.968	2.2727E-06
	***daemon* LAM**								
Homog.	398.635	3.951	303.101	7.691	758.111	3.981	15.791	16.382	7.1340E-07
Heterog.	411.383	3.844	299.547	6.975	761.059	3.531	6.693	12.838	8.3592E-07
Het, dbl	411.622	3.730	307.535	6.953	780.764	3.823	16.728	12.834	2.5985E-06

(b) Configuración LAM estándar (MAXNMSGLEN 8K)

Tabla 2.21: Parámetros del Modelo 3 ajustados a las diversas configuraciones de LAM.

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	$4.4595\ 10^{-7}$	$9.4541\ 10^{-7}$	$4.2999\ 10^{-7}$	$7.1340\ 10^{-7}$
Heterogéneo	$7.0374\ 10^{-7}$	$9.2361\ 10^{-7}$	$6.8412\ 10^{-7}$	$8.3592\ 10^{-7}$
Het, double	$2.2600\ 10^{-6}$	$2.7113\ 10^{-6}$	$2.2727\ 10^{-6}$	$2.5985\ 10^{-6}$

(a) Modelo 3

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	$7.0319\ 10^{-7}$	$6.4094\ 10^{-6}$	$7.0463\ 10^{-7}$	$3.4474\ 10^{-5}$
Heterogéneo	$9.5388\ 10^{-7}$	$7.5077\ 10^{-6}$	$8.9984\ 10^{-7}$	$3.1567\ 10^{-5}$
Het, double	$2.5387\ 10^{-6}$	$8.7539\ 10^{-6}$	$2.5360\ 10^{-6}$	$3.3371\ 10^{-5}$

(b) Modelo 2

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	1.577	6.779	1.639	48.324
Heterogéneo	1.355	8.129	1.315	37.763
Het, double	1.123	3.229	1.116	12.842

(c) Mejora relativa: error del Modelo 2 dividido entre el del Modelo 3

	MAXNMSGLEN 4K		MAXNMSGLEN 8K	
	cliente-a-cliente	***daemon* LAM**	**cliente-a-cliente**	***daemon* LAM**
Homogéneo	0.634	0.148	0.610	0.021
Heterogéneo	0.738	0.123	0.760	0.026
Het, double	0.890	0.310	0.896	0.078

(d) Reducción relativa: error del Modelo 3 dividido entre el del Modelo 2

Tabla 2.22: Errores cuadráticos medios de los Modelos 2 y 3 bajo las diversas configuraciones LAM.

Los parámetros para la modalidad cliente-a-cliente cumplen todas las relaciones esperadas salvo $bm < bs$, igual que sucedió con PVM. En la alternancia de MTUs, la segunda no es de mayor ancho de banda como sucedía en la primera UDP ($bc \ll bs$) sino menor ($bm \leq bs$). Afortunadamente sí se cumple $lc \gg lm \gg ls$, lo cual posibilitó la detección y modelado de dicha alternancia de MTUs.

Los parámetros para el *daemon* tienen una primera MTU idéntica a la de ruta directa pero elevada 280μs, como muestran lc/bc en la Tabla 2.21 y las ampliaciones de las Figuras 2.54(b) y 2.55(b). Aunque los valores de parámetros indiquen (muy ligeramente) inferiores prestaciones para la configuración estándar, no hay que olvidar que las UDPs son mayores, siendo menos frecuente el costoso salto lm entre UDPs. La gráfica izquierda de las Figuras 2.54(a) y 2.55(a) muestra cómo la configuración estándar evita llegar a los 7ms para 32KB alcanzados por la configuración coincidente con PVM.

Resulta manifiesto el oneroso salto entre 1ª y 2ª UDPs, lp, que casi duplica el valor de lc. La latencia por cada UDP, lm, es aproximadamente 3/4 de bc, y ls es sistemáticamente menor, aunque no esté muy estabilizada. Siempre se cumple la esperada relación $lp \gg lc > lm \gg ls$.

El ancho de banda para la 1ª MTU de la segunda UDP, bp, ha resultado muy similar al de la 1ª UDP, por lo cual se podría considerar identificar ambos parámetros. Los anchos de banda para las restantes MTUs de las dos primeras UDPs, bm, cumplen efectivamente la anticipada propiedad $bc \simeq bp < bm < bs$. De hecho, cada uno es aproximadamente el doble que el anterior.

El modelo sólo presenta errores superiores a 10^{-6} para las mediciones con datos double. El modo *daemon* oscila entre 7–9.5 10^{-7} y la ruta directa entre 4–7 10^{-7}. Los cocientes en la Tabla 2.22 (d) muestran una significativa reducción del error (al 60–75%) y una reducción de casi dos órdenes de magnitud (al 1–2%) con el *daemon*. Al igual que bajo PVM, la reducción para datos double es menor (a 90% cliente-a-cliente y a 10–30% con el *daemon*).

No se puede reducir más el error utilizando segmentos afines para cada MTU. Un refinamiento ulterior del modelo requeriría considerar otro tipo de tramos, no basados en MTUs, o no lineales.

2.4 Conclusiones

En este capítulo se ha presentado el equipamiento utilizado para desarrollar, depurar y comprobar el funcionamiento de las *Toolboxes* paralelas presentadas en este trabajo: PVMTB y MPITB para MATLAB. Se describe brevemente el proceso de instalación y configuración del cluster y los pasos fundamentales de configuración del Sistema Operativo utilizado (Linux), incluyendo referencias a la documentación relevante en donde es posible encontrar instrucciones mucho más detalladas. Esta información, así como el propio Sistema Operativo, son de público dominio, redundando en el argumento de que los clusters de computadores proporcionan una plataforma eficaz para Computación Paralela *asequible para cualquier institución*.

Se ha introducido el problema de evaluación de prestaciones de un sistema de paso de mensajes. Se mencionan los métodos más comúnmente usados a tal fin. Hemos desarrollado una serie de modelos numerados del 0 al 3 en orden de complejidad, costes de evaluación y méritos predictivos crecientes. Estos modelos condensan en unos pocos parámetros una mayor información que la proporcionada por los habituales ancho de banda y tiempo de *setup*, también denominado *latencia* en esta memoria.

Se han extraido los parámetros resultado de aplicar los distintos modelos a PVM y LAM/MPI. Esto permite comparar entre sí ambos sistemas de paso de mensajes con mayor precisión que la visualización directa de las funciones características de prestaciones, que simplemente informan del tiempo empleado para transmitir en función del tamaño del mensaje.

Capítulo 3

Estudio de PVMTB

Resumen del capítulo

PVMTB son las siglas de “PVM *Toolbox*”. Como ya se mencionó previamente, se suelen agrupar bajo el nombre genérico de “caja de herramientas” (*Toolbox*) una colección de comandos MATLAB para la resolución de problemas en una determinada materia. Esta *Toolbox* permite utilizar el sistema PVM desde MATLAB.

En el Apartado 3.1 se detallan algunos criterios de índole técnica que se han observado en el diseño de este interfaz entre MATLAB y PVM. En concreto, ningún trabajo anterior utiliza una biblioteca PVM dinámicamente enlazada, teniendo este detalle técnico implicaciones en el consumo de disco y memoria por parte de los comandos de la *Toolbox*. El Apartado 3.1.1 explica las modificaciones requeridas en el proceso de compilación de PVM para producir una biblioteca PVM compartida.

El Apartado 3.2 muestra la clasificación de las llamadas PVM según su patrón de llamada, clasificación realizada al efecto de sistematizar su interfaz. Ningún trabajo previo hace un estudio parecido, lo cual les conduce a un diseño basado en una rutina “directorio”. El *overhead* debido a esta rutina directorio queda reflejado en el estudio comparativo que se realizó en el Capítulo 1, en donde todas las *Toolboxes*, incluyendo por supuesto nuestras PVMTB y MPITB, usaban las mismas bibliotecas PVM y LAM/MPI compartidas.

La conjunción del diseño basado en directorio con el mencionado enlace estático invariablemente utilizado por los trabajos anteriores tiene también profundas implicaciones en el *overhead* introducido por la *Toolbox*. Esto no queda de manifiesto en los resultados de esta memoria, al no haberse utilizado ninguna biblioteca estática.

En el Apartado 3.3 se explica detalladamente la implementación de la *Toolbox* usando el citado método de clasificar previamente las llamadas PVM según su patrón de llamada.

El Apartado 3.4 estudia la eficiencia de PVMTB en un test *ping-pong*, similar al realizado en el capítulo anterior para PVM. Los resultados son útiles para evaluar la pérdida de prestaciones introducida por la *Toolbox*. Esta comparación con PVM se realiza en el Apartado 3.5. Finalmente se resumen las conclusiones de este capítulo en el Apartado 3.6.

3.1 Elecciones de diseño

Para respetar el principio de paso de mensajes explícito mantenido por PVM, se optó por redactar un comando MATLAB para cada llamada PVM. Estos comandos toman la forma de un fichero MEX (MATLAB EXecutable, MEX-file) redactado en C, lenguaje desde el cual se puede llamar a la biblioteca PVM. Estos programas actúan como recubrimientos (*wrappers*) para las rutinas de paso de mensajes, realizando las siguientes funciones:

- traducción de los argumentos MATLAB al tipo de datos requerido bajo C.
- traducción inversa para pasar los argumentos de retorno devueltos por PVM a MATLAB.
- anotación de información de estado coherente con PVM, en los casos en que sea necesario.

El patrón de llamada de las rutinas PVM (número y tipo de argumentos de cada llamada) es bastante sistemático, a veces repetitivo, lo cual se ha aprovechado para reutilizar código intensivamente, mediante sustitución de macros **#define** en C.

La interfaz de programas de aplicación (API) MATLAB requiere que los ficheros MEX sean enlazados dinámicamente, para que MATLAB los pueda cargar en tiempo de ejecución [51]. Para evitar excesivo gasto de disco y memoria, se optó por enlazar dinámicamente también la propia biblioteca PVM, lo cual reduce drásticamente su tamaño y el de los ficheros MEX enlazados con ella.

Los motivos expuestos demuestran que la línea de razonamiento seguida surge de manera natural, y las elecciones de diseño parecen muy lógicas. En realidad no resulta tan inmediato seleccionar los criterios de diseño expuestos, como lo demuestra el hecho de que ningún trabajo anterior (DP-TB [19], PT [32]) utilice una biblioteca PVM dinámicamente enlazada, teniendo este detalle técnico implicaciones en el consumo de disco y memoria por parte de los comandos de la *Toolbox*.

DP-TB (Apartado 1.6.2) opta por el diseño basado en directorio, en el que un fichero MEX auxiliar se enlaza estáticamente con las bibliotecas PVM, pudiendo seleccionar la rutina PVM a invocar según un argumento de la llamada MEX. DP-TB separa la funcionalidad de directorio (fichero MEX m2pvm) de los recubrimientos (*wrappers*) propiamente dichos, segregados en otro fichero MEX auxiliar m2libpvm. Adicionalmente, un tercer fichero MEX (m2libpvme) implementa las extensiones y variantes de las llamadas PVM contempladas en el diseño de la *Toolbox*.

No obstante, se observa un esfuerzo denodado por ahorrar espacio en memoria: la segregación de los *wrappers* consigue no aumentar el tamaño del propio directorio, aunque introduce una llamada intermedia adicional al sistema global de paso de mensajes. En cualquier caso, la utilización de una única llamada PVM fuerza la carga en memoria de los *wrappers*. Tal vez el motivo por el que DP-TB no incluye un interfaz con las llamadas colectivas PVM sea para evitar enlazar también estáticamente la biblioteca de grupos (libgpvm3) con el ya abultado fichero MEX de directorio m2libpvm.

PT, la otra *Toolbox* paralela MATLAB basada en PVM, contiene alrededor de 25 comandos en total, de los cuales 14 son llamadas PVM, y desarrolla un sistema de 2 *wrappers* aislados para las llamadas PVM. También tiene un directorio pt, en el que la selección del comando se realiza con una cascada **if – else if** en lugar del habitual **switch**. El número de pasos intermedios llega por tanto a 5. Por ejemplo, pt_mywid.m finaliza con una llamada a directorio wid=pt('mywid'), el cual

termina encontrando la rama **else** correspondiente a 'mywid' y ejecuta el primer *wrapper* intermedio pt_mywid(nlhs,plhs,nrhs−1,prhs+1), quien a su vez invoca al segundo *wrapper* ∗wid = (**double**)mywid(), que es el que finalmente llama a la rutina PVM tid = pvm_mytid(). Aparte de la pérdida de prestaciones debida a las numerosas llamadas intermedias, enlazar estáticamente los *wrappers* de último nivel (como mywid()) con la biblioteca PVM conduce a gastos de disco prohibitivos (y de memoria, al cargarlos en MATLAB).

Adicionalmente, PT define un sistema de "Entornos Paralelos" (*Parallel Environments, PEs*) extraño a PVM, requiriendo un estudio adicional al de la propia biblioteca PVM para el usuario no familiarizado con la materia.

Se pueden valorar ahora los méritos del diseño propuesto para PVMTB en cuanto a robustez, coste de desarrollo y mantenimiento, reutilización del código, ocupación en disco y eficiencia de llamada. Los trabajos previos se basan en una "rutina directorio", elección que termina traduciéndose en una cierta (a veces significativa) pérdida de prestaciones debida a los pasos intermedios de interpretación del fichero M y selección del caso en el directorio.

El sistema de patrones adoptado en PVMTB reduce el esfuerzo de programación para desarrollar la *Toolbox*, ya que el patrón desarrollado para una llamada se reutiliza, una vez depurado, para añadir *todas* las llamadas PVM que tienen el mismo patrón. No se introducen errores de codificación en este paso, lo cual redunda en la robustez del diseño, limitándose la tarea del diseñador a ejecutar las nuevas llamadas para constatar su funcionamiento correcto. El mantenimiento del software es mínimo, pudiéndose dedicar el esfuerzo a realizar patrones óptimos. Cada mejora introducida en un patrón se refleja automáticamente en todos los *wrappers* que lo reutilizan.

Con este diseño también se introduce el mínimo gasto de disco y memoria posible, ya que los *wrappers* no incluyen el código PVM, presente en la biblioteca compartida. Incluso varios usuarios podrían utilizar simultáneamente la *Toolbox*, y usuarios adicionales podrían usar PVM desde C simultáneamente, habiendo una única copia de la biblioteca PVM compartida en memoria.

Una aplicación MATLAB que necesitara por ejemplo sólo 6 llamadas PVM cargaría únicamente los 6 *wrappers* correspondientes, que además son de tamaño más reducido que un directorio, especialmente si el directorio está enlazado estáticamente con la biblioteca PVM. La primera llamada carga la biblioteca PVM dinámica si no estaba cargada aún, quedando disponible para eventuales usuarios PVM o PVMTB adicionales.

La propia ejecución de un *wrapper* PVMTB carece de llamadas previas a ficheros M y posteriores a directorios MEX: se invoca a la llamada PVM directamente. El único coste adicional es la traducción de parámetros de entrada MATLAB a lenguaje C, y de valores de retorno PVM a lenguaje MATLAB. La superioridad en prestaciones del diseño queda manifiesta en el estudio comparativo realizado en el Capítulo 1.

3.1.1 Compilación dinámica de PVM

La biblioteca PVM se ofrece enlazada estáticamente. Desde que se inició el desarrollo de este trabajo (usando RedHat 6.1) se han popularizado las versiones dinámicas de la misma, y versiones más recientes del propio Sistema Operativo utilizado (RedHat 7.1) incluyen ahora su propia versión dinámica tanto de PVM como de LAM. Mr. Trond Eivind Glomsrød, miembro del equipo de desarrollo de RedHat, es también colaborador habitual de la lista de distribución de LAM, y

artífice de la inclusión de PVM y LAM/MPI en RedHat 7.1. La popularización y fácil acceso a este tipo de software en distribuciones estándar de Linux como RedHat redunda aún más en el argumento de que los clusters de computadores son una plataforma efectiva para Computación Paralela, en donde no sólo el hardware, sino también el software, se puede obtener y actualizar más fácilmente.

En su momento fue necesario modificar el proceso de compilación de PVM bajo RedHat 6.1 para producir bibliotecas dinámicas. La página Web de PVMTB [79] incluye instrucciones, ficheros de parches e incluso la especificación RPM para producir bibliotecas PVM compartidas. Estas instrucciones son ahora redundantes, pudiendo los usuarios instalar directamente el paquete RPM o versión PVM correspondiente en su Sistema Operativo. Como se ve, todo apunta a facilitar la introducción de nuevos usuarios en el campo de Computación Paralela, lo cual en último término sólo puede resultar en mayores posibilidades de éxito para los clusters de computadores como plataforma paralela.

Los criterios seguidos para modificar los Makefile PVM son:

Creación de ficheros objeto: El compilador gcc requiere el modificador –fPIC para generar código independiente de la posición, que es el apropiado para enlace dinámico, y por lo tanto también para bibliotecas compartidas.

PVM utiliza sólo los *flags* –O/–g para producir código optimizado o con información para depuración.

Creación de bibliotecas dinámicas: Para producir objetos compartidos, gcc requiere el modificador –shared. Las bibliotecas dinámicas suelen denominarse lib<nombre>.so (*shared object*) bajo UNIX, en oposición a las estáticas lib<nombre>.a (*archive*).

Es costumbre usar la opción de enlazado –soname para indicar el nombre *oficial* de la biblioteca compartida; nombre bajo el cual la buscarán en tiempo de ejecución los programas que la usen.

PVM crea bibliotecas estáticas, usando la utilidad de archivado ar sobre los objetos estáticos. También es costumbre aplicar la utilidad ranlib para producir un índice que acelera el enlace (estático) con la biblioteca.

Referencia a bibliotecas dinámicas: La única forma de referencia válida para bibliotecas dinámicas es el modificador –l<nombre>. Las referencias a biblioteca estática pueden indicarse también usando el nombre de fichero lib<nombre>.a en la línea de comandos. PVM usa ocasionalmente dicha forma, que debe cambiarse a –l<nombre>.

En alguna ocasión el proceso de compilación del sistema PVM aprovecha que alguna biblioteca estática se halla en el subdirectorio actual, haciendo referencia a ella aunque no haya sido copiada a un subdirectorio tal que se le podría hacer referencia usando –l<nombre>. En dichos casos, se ha adelantado la dependencia que copia la biblioteca entre los objetivos (*targets*) del Makefile, de manera que la biblioteca esté en su sitio antes de referenciarla.

La *Toolbox* requiere imprescindiblemente la biblioteca PVM (libpvm3.so) y el programa *daemon* pvmd3. Si se usan grupos, se requiere también la correspondiente biblioteca libgpvm3.so y el programa servidor de grupos pvmgs. Durante la etapa de depuración también fue útil disponer de la consola pvm, la cual a su vez requiere también la biblioteca de trazado libpvmtrc.so. Se decidió por

lo tanto aplicar enlace dinámico a todos los componentes del sistema PVM. A título ilustrativo, el Listado 3.1 muestra algunas líneas características del fichero de parches generado tras aplicar dichos criterios de modificación.

```
-CFLOPTS          =     -g
+CFLOPTS          =     -g -fPIC
...
-$(LIBPVM).a:   regexconfig $(LOBJ) $(LPVMSOCK) $(SOCKLOBJ)
-       rm -f $(LIBPVM).a
-       $(AR) cr $(LIBPVM).a $(LOBJ) $(LPVMSOCK) $(SOCKLOBJ)
-       case x$(HASRANLIB) in \
-            xt ) echo ranlib; ranlib $(LIBPVM).a ;; esac
+$(LIBPVM).so:  regexconfig $(LOBJ) $(LPVMSOCK) $(SOCKLOBJ)
+       rm -f $(LIBPVM).so
+       $(CC) -shared -Wl,-soname,$(LIBPVM).so -o $(LIBPVM).so \
+                         $(LOBJ) $(LPVMSOCK) $(SOCKLOBJ)
...
 pvmgroups$(EXESFX):   pvmgroups.o $(LIBGPVM) $(PVMLIBDEP)
-       $(CC) $(CFLAGS) -o $@ pvmgroups.o $(LIBGPVM) $(LIBS)
+       $(CC) $(CFLAGS) -o $@ pvmgroups.o $(LIBS)  -lgpvm3
...
 $(LIBGPVM):    $(LOBJ)
        rm -f $(LIBGPVM)
-       $(AR) cr $(LIBGPVM) $(LOBJ)
-       case x$(HASRANLIB) in \
-            xt ) echo ranlib; ranlib $(LIBGPVM) ;; esac
+       $(CC) -shared -Wl,-soname,$(LIBGPVM) -o $(LIBGPVM) $(LOBJ)
...
 $(LIBPVMTRC):  $(TRCOBJS) $(PVMLDIR)
-       $(AR) rcv $(LIBPVMTRC) $(TRCOBJS)
-       @ case x$(HASRANLIB) in \
-         xt ) echo "ranlib␣$(LIBPVMTRC)"; ranlib $(LIBPVMTRC) ;; \
-         xf ) echo "No␣Ranlib␣Necessary." ;; esac
+       $(CC) -shared -Wl,-soname,$(LIBPVMTRC) -o $(LIBPVMTRC) $(TRCOBJS)
```

Listado 3.1: Aspecto de los parches aplicados a los ficheros Makefile de PVM para producir bibliotecas compartidas. El signo – indica eliminación de una línea, + indica inserción.

3.2 Patrones de llamada PVM

Como se mencionó previamente, la regularidad de los patrones de llamada de las rutinas PVM favorece la reutilización de código, a la vez que invita a realizar un interfaz que respete escrupulosamente el tipo y número de argumentos de las llamadas PVM originales.

Ningún trabajo previo hace un estudio parecido, lo cual les conduce a un diseño basado en una rutina "directorio". La conjunción de esta elección de diseño con el mencionado enlace estático afecta severamente al *overhead* introducido por la *Toolbox*.

Además, el esfuerzo de codificación y mantenimiento se multiplica, no sorprendiendo que ninguna implementación anterior contemplara todas las llamadas PVM. Cada nueva llamada debe ser codificada independientemente, pudiéndose cometer nuevos errores con el costo de depuración asociado. El mantenimiento del software se impone rápidamente, dominando en el coste global de diseño en cuanto las llamadas implementadas superan un cierto número.

Se ha mencionado previamente alguna situación en la que, a pesar de ser atractiva alguna modificación del patrón, se ha preferido retrasar la implementación del nuevo interfaz hasta que un número suficiente de usuarios lo reclame. Recuérdese por ejemplo la comparación entre el comando PVMTB pvm_upkdouble(var) y el comando DP-TB var=pvm_upkdouble(1,1) (Apartado 1.3.1).

Motivos didácticos y de eficiencia respaldan tal decisión, a pesar del innegable atractivo de patrones alternativos para producir un código MATLAB más compacto. Sin embargo, el patrón de algunas llamadas ha sido modificado bajo PVMTB. En los casos en que esto sucede (consultar Tabla 3.1) se debe a alguno de los motivos siguientes:

Coerción, cambio o promoción de tipo: Bajo MATLAB no se puede operar con datos *int*, por lo cual se optó por promocionarlos a *double*. En la llamada a PVM se coerce *double*→**int** y a la vuelta se promociona **int**→*double*. Similar trato reciben los *cell-arrays* MATLAB, cambiados a **char**∗∗ bajo C, como en las llamadas pvm_spawn y pvm_start_pvmd, por ejemplo.

Las *struct* MATLAB se traducen a **struct** C, como por ejemplo en pvm_config, _tasks, _reg_rm, _getminfo, _setminfo, _getmboxinfo. También se agrupan en una *struct* MATLAB los **int**s devueltos por pvm_bufinfo y pvm_precv.

La **struct** timeval se cambia a *double* en pvm_trecv y select, por ser más cómodo para el usuario MATLAB especificar un tiempo como *double*, pero se respeta bajo pvm_hostsync, por la semántica especial que tiene para el caso de segundos negativos.

En pvm_catchout resulta imprescindible cambiar el argumento FILE∗ a **int** (handle C) y promocionarlo a *double* para su uso bajo MATLAB. Similar promoción se realiza bajo pvm_getfds y select.

En la gestión de máscaras (pvm_[get,set]tmask, TEV_MASK_∗) se ha cambiado el tipo pvmtmask de las máscaras de traza PVM al tipo *string* MATLAB. Los argumentos who y kind se han cambiado de **int** a *string* para que el usuario no necesite recordar los valores numéricos concretos y pueda usar los símbolos PVM.

Varios argumentos de retorno: Las funciones C sólo devuelven un argumento de retorno. Para devolver varios argumentos, se suele recurrir a:

- devolver un puntero a array o estructura conteniendo dichos valores.
- pasar punteros a argumentos adicionales (paso por referencia), de manera que a la vuelta contengan los valores de retorno deseados.

PVM usa ambas técnicas. Las funciones MATLAB pueden devolver varios argumentos de retorno. Se ha seguido el criterio de simplificar la llamada, trasladando a la parte izquierda (*left-hand side*, plhs[]) los argumentos por referencia. Esta técnica se usa en pvm_addhosts, _delhosts, _gettmask, _config, _tasks, _getfds, _reg_rm, _hostsync, _siblings, _unpack, _bufinfo, _getminfo, _getmboxinfo, _psend, _precv, _tickle.

Argumentos inferibles: Una de las características más útiles de MATLAB como lenguaje de rápido prototipado de sistemas es la ausencia de declaraciones de tipo. Las variables pueden alterar dinámicamente su tamaño y número de dimensiones, y estos atributos pueden ser consultados en cualquier instante.

El lenguaje C no fija ningún método para indicar el tamaño de un array:

- el carácter ASCII NULL marca el fin de un *string* (**char**∗).
- se indica explícitamente el tamaño de una matriz o vector en su definición (por ejemplo, **double** vector[M]).

- si una función trata matrices de diversos tamaños, se suele indicar éste como parámetro adicional (tantos como dimensiones).

PVM usa todas las técnicas citadas. MATLAB almacena como atributo de cada variable su tamaño y dimensionado actual, de manera que puede ser consultado, haciendo superfluos cualesquiera de los métodos anteriores. Esto se ha aplicado en pvm_start_pvmd, _addhosts, _delhosts, _spawn, _notify, _mcast, _[u]pk*, _psend, _reduce, _scatter, _gather.

A menudo PVM devuelve un **int** indicando el tamaño de un array. Este argumento se suele respetar (promocionándolo a **double**), ya que suele ser de interés para el usuario. Como ejemplos podemos citar pvm_getfds, _tasks, _getmboxinfo, _config, _siblings.

Tras realizar los cambios mencionados, los resultantes patrones de llamada de los comandos PVMTB se han clasificado según el número y tipo de los argumentos, al objeto de reutilizar código y simplificar la implementación y estudio de la *Toolbox*. La Tabla 3.1 enumera las llamadas PVM originales, junto con los comandos PVMTB correspondientes y el nombre dado al patrón que sigue cada comando, según la nomenclatura usada en el siguiente apartado.

3.3 Discusión y detalles

El interfaz procura ser lo más transparente y sistemático posible. A dicho efecto se ha redactado un fichero **#include** con los patrones de llamada más comunes. El Listado 3.2 muestra la estructura del mismo.

```
/* header.h */
#include <mex.h>          /* Matlab */
#include <pvm3.h>         /* pvm_* */

/* Patrones de llamada */
...

/* Bloque constructivo para retornar un escalar */
#define FUNCALL                                                  \
        *mxGetPr(plhs[0]= mxCreateDoubleMatrix (1,1,mxREAL)) =

/* Punto de entrada MEX */
/*------------------------------------------------------------------------*/
void mexFunction(int nlhs,mxArray*plhs[], int nrhs, const mxArray*prhs[])
/*------------------------------------------------------------------------*/
```

Listado 3.2: Primeras líneas y final del fichero header.h, en donde se incluyen los patrones más comúnmente usados en el desarrollo de PVMTB.

La parte final de header.h probablemente requiere una explicación adicional: el punto de entrada de los ficheros MEX es la función mexFunction(), en lugar de la habitual main() de un programa C independiente. En tiempo de ejecución, MATLAB buscará en disco y cargará en memoria el fichero MEX que desee ejecutar el usuario, e intentará pasar el control a la función mexFunction(). Si el fichero MEX requiere llamadas enlazadas dinámicamente que aún no han sido cargadas, la biblioteca dinámica que las contiene también se busca y se carga en memoria.

Incorporar el prototipo de función MEX al final del fichero **#include** evita duplicarlo en cada fichero fuente de la *Toolbox*. Su reutilización ahorra recodificación innecesaria y evita una fuente potencial de errores. El prototipo de función mexFunction muestra que el intérprete MATLAB pasa al

Tabla 3.1: Llamadas PVM bajo MATLAB.

PAT.	PVMTB	PVM
	Control de la Máquina Paralela Virtual	
SMP	info = pvm_start_pvmd ({'arg'...}, block)	**int** info = pvm_start_pvmd (**int** argc , **char****argv, **int** block)
P–9	[numh, infos] = pvm_addhosts ('host'...)	**int** info = pvm_addhosts (**char****hosts, **int** nhost , **int***infos)
P–9	[numh, infos] = pvm_delhosts ('host'...)	**int** info = pvm_delhosts (**char****hosts, **int** nhost , **int***infos)
SMP	[numt, tids] = pvm_spawn ('task',{'arg'...}, flag, 'where', ntask)	**int** numt = pvm_spawn (**char***task, **char****argv, **int** flag , **char***where, **int** ntask , **int***tids)
P–2	info = pvm_kill (tid)	**int** info = pvm_kill (**int** tid)
P–4	info = pvm_sendsig (tid, signum)	**int** info = pvm_sendsig (**int** tid , **int** signum)
SMP	info = pvm_notify (what, msgtag, (cnt|tids))	**int** info = pvm_notify (**int** what, **int** msgtag, **int** cnt , **int***tids)
P–1	info = pvm_exit	**int** info = pvm_exit (**void**)
P–1	info = pvm_halt	**int** info = pvm_halt (**void**)
SMP	info = pvm_catchout [(fildes)]	**int** info = pvm_catchout (FILE*ff)
P–2	val = pvm_getopt (what)	**int** val = pvm_getopt (**int** what)
P–4	oldval = pvm_setopt (what, val)	**int** oldval = pvm_setopt (**int** what, **int** val)
MSK	[info, mask] = pvm_gettmask ('who')	**int** info = pvm_gettmask (**int** who, pvmtmask mask)
MSK	info = pvm_settmask ('who', mask)	**int** info = pvm_settmask (**int** who, pvmtmask mask)
MSK	TEV_MASK_INIT (mask)	TEV_MASK_INIT (mask)
MSK	TEV_MASK_SET (mask,'kind')	TEV_MASK_SET (mask, kind)
MSK	TEV_MASK_UNSET (mask,'kind')	TEV_MASK_UNSET (mask, kind)
MSK	TEV_MASK_CHECK (mask,'kind')	TEV_MASK_CHECK (mask, kind)
	Información de la Máquina Paralela Virtual	
SMP	[nhost, narch, hostinfo] = pvm_config	**int** info = pvm_config (**int***nhost , **int***narch, **struct** pvmhostinfo**hostp)
P–6	mstat = pvm_mstat ('host')	**int** mstat = pvm_mstat (**char***host)
SMP	[ntask, tinfo] = pvm_tasks ('where')	**int** info = pvm_tasks (**int** where, **int***ntask, **struct** pvmtaskinfo**taskp)
P–2	status = pvm_pstat (tid)	**int** status = pvm_pstat (**int** tid)
P–1	tid = pvm_mytid	**int** tid = pvm_mytid (**void**)
P–1	tid = pvm_parent	**int** tid = pvm_parent (**void**)
P–2	dtid = pvm_tidtohost (tid)	**int** dtid = pvm_tidtohost (**int** tid)
P–6	info = pvm_perror ('msg')	**int** info = pvm_perror (**char***msg)
SMP	'version' = pvm_version	**char***version = pvm_version (**void**)
P–6	cod = pvm_archcode ('arch')	**int** cod = pvm_archcode (**char***arch)
P–5	[nfds, fds] = pvm_getfds	**int** nfds = pvm_getfds (**int****fds)
P–5	[ntids, tids] = pvm_siblings	**int** ntids = pvm_siblings (**int****tids)
SMP	[info, clk, dlt] = pvm_hostsync (htid)	**int** info = pvm_hostsync (**int** host , **struct** timeval*clk, **struct** timeval*dlt)
	Buffers de Envío y Recepción	
P–3	bufid = pvm_mkbuf [(encoding)]	**int** bufid = pvm_mkbuf (**int** encoding)
P–2	info = pvm_freebuf (bufid)	**int** info = pvm_freebuf (**int** bufid)
SMP	[info, bufinfo] = pvm_bufinfo (bufid)	**int** info = pvm_bufinfo (**int** bufid , **int***bytes , **int***tag , **int***tid)
SMP	[info, msginfo] = pvm_getminfo (bufid)	**int** info = pvm_getminfo (**int** bufid , **struct** pvmminfo*minfo)
SMP	info = pvm_setminfo (bufid, msginfo)	**int** info = pvm_setminfo (**int** bufid , **struct** pvmminfo*minfo)
P–1	bufid = pvm_getrbuf	**int** bufid = pvm_getrbuf (**void**)
P–1	bufid = pvm_getsbuf	**int** bufid = pvm_getsbuf (**void**)
P–2	oldbuf = pvm_setrbuf (bufid)	**int** oldbuf = pvm_setrbuf (**int** bufid)
P–2	oldbuf = pvm_setsbuf (bufid)	**int** oldbuf = pvm_setsbuf (**int** bufid)

Tabla 3.1: Llamadas PVM bajo MATLAB (continúa).

PAT.	PVMTB	PVM
	Envío y Recepción de Mensajes	
P–3	bufid = pvm_initsend [(encoding)]	**int** bufid = pvm_initsend (**int** encoding)
PCK	info = pvm_pack (var...)	**int** info = pvm_packf (**const char***fmt ,...)
P–4	info = pvm_send (tid, msgtag)	**int** info = pvm_send (**int** tid , **int** msgtag)
PCK	[info, names] = pvm_unpack [('vnam'...)]	**int** info = pvm_unpackf (**const char***fmt ,...)
P–4	bufid = pvm_recv (tid, msgtag)	**int** bufid = pvm_recv (**int** tid , **int** msgtag)
SMP	bufid = pvm_trecv (tid, msgtag, tmout)	**int** bufid = pvm_trecv (**int** tid , **int** msgtag,
P–4	bufid = pvm_nrecv (tid, msgtag)	**int** bufid = pvm_nrecv (**int** tid , **int** msgtag)
P–4	bufid = pvm_probe (tid, msgtag)	**int** bufid = pvm_probe (**int** tid , **int** msgtag)
SMP	info = pvm_mcast (tids, msgtag)	**int** info = pvm_mcast (**int*** tids , **int** ntask , **int** msgtag)
SMP	info = pvm_pkmesg (bufid)	**int** info = pvm_pkmesg (**int** bufid)
P–1	bufid = pvm_upkmesg	**int** bufid = pvm_upkmesg (**void**)
P–2	info = pvm_pkmesgbody (bufid)	**int** info = pvm_pkmesgbody (**int** bufid)
SMP	[info, len] = pvm_psend (tid, msgtag, data)	**int** info = pvm_psend (**int** tid , **int** msgtag, **char***buf, **int** len , **int** datatype)
SMP	[info, bufinfo] = pvm_precv (tid, msgtag, 'vnam', len)	**int** info = pvm_precv (**int** tid , **int** msgtag, **char***buf, **int** len , **int** datatype , **int*** atid , **int***atag , **int***alen)
	Empaquetamiento	
UPK	info = pvm_[u]pkdouble (var [,n [,str]])	**int** info = pvm_[u]pkdouble (**double***dp, **int** nitem , **int** stride)
UPK	info = pvm_[u]pkfloat (var [,n [,str]])	**int** info = pvm_[u]pkfloat (**float** *fp , **int** nitem , **int** stride)
UPK	info = pvm_[u]pk[u]long (var [,n [,str]])	**int** info = pvm_[u]pk[u]**long** ([**unsigned**] **long***ip , **int** nitem , **int** stride)
UPK	info = pvm_[u]pk[u]int (var [,n [,str]])	**int** info = pvm_[u]pk[u]**int** ([**unsigned**] **int***ip , **int** nitem , **int** stride)
UPK	info = pvm_[u]pk[u]short (var [,n [,str]])	**int** info = pvm_[u]pk[u]**short** ([**unsigned**] **short***ip , **int** nitem , **int** stride)
UPK	info = pvm_[u]pkbyte (var [,n [,str]])	**int** info = pvm_[u]pkbyte (**char***xp, **int** nitem , **int** stride)
UPK	info = pvm_[u]pkstr (var)	**int** info = pvm_[u]pkstr (**char***sp)
	Grupos	
P–6	inum = pvm_joingroup ('group')	**int** inum = pvm_joingroup (**char***group)
P–7	info = pvm_freezegroup ('group', size)	**int** info = pvm_freezegroup (**char***group, **int** size)
P–6	info = pvm_lvgroup ('group')	**int** info = pvm_lvgroup (**char***group)
P–7	info = pvm_barrier ('group', count)	**int** info = pvm_barrier (**char***group, **int** count)
P–7	inum = pvm_getinst ('group', tid)	**int** inum = pvm_getinst (**char***group, **int** tid)
P–7	tid = pvm_gettid ('group', inum)	**int** tid = pvm_gettid (**char***group, **int** inum)
P–6	size = pvm_gsize ('group')	**int** size = pvm_gsize (**char***group)
P–7	info = pvm_bcast ('group', msgtag)	**int** info = pvm_bcast (**char***group, **int** msgtag)
COL	info = pvm_reduce ('op', var, msgtag, 'group', inst)	**int** info = pvm_reduce (**void**(*func)(), **void***data , **int** count , **int** datatype , **int** msgtag , **char***group, **int** rootginst)
COL	info = pvm_scatter (porcion, var, msgtag, 'group', inst)	**int** info = pvm_scatter (**void*** result , **void***data , **int** count , **int** datatype , **int** msgtag , **char***group, **int** rootginst)
COL	info = pvm_gather (var, porcion, msgtag, 'group', inst)	**int** info = pvm_gather (**void*** result , **void***data , **int** count , **int** datatype , **int** msgtag , **char***group, **int** rootginst)

Tabla 3.1: Llamadas PVM bajo MATLAB (continúa).

PATRON	PVMTB	PVM
	Entorno	
P–6	cc = pvm_export ('name')	**int** cc = pvm_export (**char**∗name)
P–6	cc = pvm_unexport ('name')	**int** cc = pvm_unexport (**char**∗name)
	Mailbox global	
P–8	index = pvm_putinfo ('name', bid, flags)	**int** index = pvm_putinfo (**char**∗name, **int** bufid , **int** flags)
P–8	bufid = pvm_recvinfo ('name', idx, flags)	**int** bufid = pvm_recvinfo (**char**∗name, **int** index , **int** flags)
P–8	info = pvm_delinfo ('name', idx, flags)	**int** info = pvm_delinfo (**char**∗name, **int** index , **int** flags)
SMP	[info, mbinfo] = pvm_getmboxinfo ('patrón')	**int** info = pvm_getmboxinfo (**char**∗pattern , **int** nclasses , **struct** pvmmboxinfo∗∗classes)
	Contexto	
P–1	ctx = pvm_getcontext	**int** ctx = pvm_getcontext (**void**)
P–1	ctx = pvm_newcontext	**int** ctx = pvm_newcontext (**void**)
P–2	old_ctx = pvm_setcontext (new_ctx)	**int** old_ctx = pvm_setcontext (**int** new_ctx)
P–2	info = pvm_freecontext (ctx)	**int** info = pvm_freecontext (**int** ctx)
	Registro de Gestores	
REG	info = pvm_reg_hoster	**int** cc = pvm_reg_hoster (**void**)
REG	info = pvm_reg_tasker	**int** cc = pvm_reg_tasker (**void**)
REG	[info, hostinfo] = pvm_reg_rm	**int** cc = pvm_reg_rm (**struct** pvmhostinfo∗∗hip)
FUN	mhid = pvm_addmhf (src, tag, ctx, 'cmd')	**int** mhid = pvm_addmhf (**int** src , **int** tag , **int** ctx , **int**(∗func)(**int** bufid))
FUN	info = pvm_delmhf (mhid)	**int** info = pvm_delmhf (**int** mhid)
FUN	pvm_recvf [('cmd')]	**int**(∗old)(**void**) = pvm_recvf (**int**(∗newfun)(**int** bufid , **int** tid , **int** tag))
	Extensiones	
M	info = pvme_default_config [('hostfile')]	crea fichero de hosts
M	info = pvme_start_pvmd [('arg'...)]	arranca PVM con opciones e información hosts
M	info = pvme_is	comprueba si se está ejecutando el *daemon*
M	info = pvme_kill (tids)	varios tids
M	info = pvme_pstat (tids)	ídem
M	info = pvme_sendsig (tids)	ídem
M	info = pvme_gids ('group')	tids en grupo
M	[info, msg] = pvme_upkntfy	desempaqueta mensaje notificación *daemon* PVM
M	info = pvme_pack (var...)	empaqueta como cell-array evitando *End Of Buffer*
	Utilidades	
M	'str' = hostname	nombre del host
M	'str' = uid	número de usuario
NRM	putenv ('var=valor')	**int** putenv (**const** **char**∗string)
NRM	unsetenv ('var')	**int** unsetenv (**const** **char**∗string)
NRM	info = select [(fildes [, timeout])]	**int** select (**int** n , fd_set ∗readfds , fd_set ∗ writefds , fd_set ∗ exceptfs , **struct** timeval ∗timeout)
	No documentadas	
NRM	[info, resp] = pvm_tickle (how [, arg])	tickle how [arg ...]

fichero MEX el array de argumentos de entrada prhs[] (*Pointers to Right-Hand-Side*), y espera que se devuelva un array de nlhs valores de retorno, plhs[] (*Pointers to Left-Hand-Side*). El tamaño de ambos arrays se indica con los argumentos formales nrhs y nlhs, que son ajustados para coincidir con el comando tecleado por el usuario en el entorno interpretado (o redactado en un *script*).

Se muestra también el bloque constructivo FUNCALL, en el que se observa el uso de la llamada API MATLAB mxCreateDoubleMatrix() para crear una variable escalar MATLAB y devolverla como primer valor de retorno, tras referenciar su contenido usando otra llamada API MATLAB, mxGetPr() (*Get Pointer*). Este bloque es sistemáticamente utilizado para retornar códigos de error PVM, por lo cual se redacta de forma que baste escribir a continuación la llamada a la rutina PVM deseada.

En el listado anterior se han suprimido los patrones (línea de puntos suspensivos ...), al objeto de explicarlos a continuación por separado, junto con algún ejemplo de su uso. La nomenclatura usada para referirse a los patrones es:

Patrones generales: P–1, ..., P–9

Patrón de empaquetado: UPK

Patrón colectivo: COL

Patrón de máscaras: MSK

Patrón de registro: REG

Patrón simple: SMP

Funciones: FUN, PCK, NRM

La Tabla 3.1 sigue esta nomenclatura. En este apartado se describen los patrones generales, dado que su explicación es breve y permiten obtener una idea detallada de cómo se programó la *Toolbox*. El Apéndice C describe el resto de patrones, las extensiones añadidas a la *Toolbox* y el método de compilación de la misma, que ha sido automatizado mediante el uso de la utilidad make y el correspondiente fichero Makefile.

3.3.1 Patrones de llamada generales (P–*)

Estos patrones son aplicables a la mayoría de las llamadas PVM. Los comandos PVMTB que se ajustan a ellos se clasifican según los parámetros que aceptan y valores de retorno que producen. La definición del patrón se redacta en el fichero **#include** mostrado previamente (línea ...), y se invoca en los ficheros fuente de los comandos PVMTB que siguen dicho patrón. Los ficheros MEX fuente quedan reducidos pues a la directiva **#include** y uso del patrón en cuestión.

P–1: int = f (void) El patrón queda redactado en header.h como:

```
#define  INT_F_VOID(NAME)                                    \
         FUNCALL NAME();
```

Listado 3.3: Definición del patrón P–1 en el fichero header.h.

En principio una definición de macro en lenguaje C debe expresarse en una única línea. La barra hacia atrás (*backslash*) impide que se interprete el cambio de línea, permitiendo redactar el patrón de una forma más legible.

La definición se realiza en función del símbolo *macro* NAME que se usará en la invocación del patrón para indicar el nombre de la llamada PVM deseada. Al tratarse de una función sin argumentos (*void*) devolviendo un entero, basta con usar el bloque constructivo FUNCALL previamente comentado e invocar a la llamada PVM agregando los paréntesis al nombre de la rutina.

Volviendo a consultar FUNCALL (Listado 3.2) se observa que el primer (y único) argumento de retorno se crea como una matriz 1x1 *double* (plhs[0]), y en su primer (y único) elemento (*mxGetPr()) se almacena el **int** devuelto por la función PVM.

Un ejemplo de comando PVMTB con este patrón se puede observar en el fichero MEX fuente para el comando *pvm_exit*:

```
/* info = pvm_exit --- Abandona PVM */
#include "header.h"
{INT_F_VOID(pvm_exit)}
```

Listado 3.4: Ejemplo de uso del patrón P–1: comando PVMTB *pvm_exit*.

Un total de 9 rutinas PVMTB aprovechan este patrón: pvm_exit, _halt, _mytid, _parent, _getcontext, _newcontext, _getrbuf, _getsbuf, _upkmsg.

P–2: int = f (int) El patrón en header.h es:

```
#define INT_F_INT(NAME)                                          \
        if ((nrhs!=1) || (!mxIsNumeric(prhs[0])))                \
                mexErrMsgTxt("se requiere 1 arg numérico");      \
        FUNCALL NAME((int)mxGetScalar(prhs[0]));
```

Listado 3.5: Definición del patrón P–2 en el fichero header.h.

Tras comprobar que el único argumento es numérico, se coerce a **int** y se procede a la llamada. Como ejemplo se muestra el fichero MEX fuente para *pvm_kill*:

```
/* info = pvm_kill(tid) --- Termina tarea PVM */
#include "header.h"
{INT_F_INT(pvm_kill)}
```

Listado 3.6: Ejemplo de uso del patrón P–2: comando PVMTB *pvm_kill*.

Un total de 11 rutinas PVMTB aprovechan este patrón: pvm_kill, _pstat, _tidtohost, _freebuf, _freecontext, _setcontext, _setrbuf, _setsbuf, _getopt, _pkmesg, _pkmesgbody.

P–3: int = f [(int)] Patrón:

```
#define INT_F_INTOPT(NAME,DEFAULT)                               \
        int argin = DEFAULT;                                     \
        if ((nrhs==1) && (mxIsNumeric(prhs[0])))                 \
                argin = mxGetScalar(prhs[0]);                    \
        FUNCALL NAME(argin);
```

Listado 3.7: Definición del patrón P–3 en el fichero header.h.

Si hay un argumento numérico, se coerce a **int**. Si no, se usa el valor por defecto. En realidad, las dos llamadas que reutilizan este patrón deberían seguir el anterior, **int** = f(**int**). Se decidió ofrecer esta funcionalidad por comodidad al usar pvm_initsend y pvm_mkbuf. Por ejemplo, el Listado 3.8 muestra el valor por defecto escogido para pvm_initsend.

```
/* bufid = pvm_initsend [ ( encoding ) ] --- Inicia buffer envío */
#include "header.h"
{INT_F_INTOPT(pvm_initsend , PvmDataDefault)}
```

Listado 3.8: Ejemplo de uso del patrón P–3: comando PVMTB *pvm_initsend*.

P–4: int = f (int, int) Patrón:

```
#define INT_F_INT_INT(NAME)                                         \
        if ( ( nrhs!=2)   || (! mxIsNumeric(prhs[0]))               \
                          || (! mxIsNumeric(prhs[1])) )             \
                mexErrMsgTxt("se requieren 2 args numéricos");      \
        FUNCALL NAME( mxGetScalar(prhs[0]),                         \
                      mxGetScalar(prhs[1]) );
```

Listado 3.9: Definición del patrón P–4 en el fichero header.h.

Tras comprobar que los dos argumentos son numéricos se procede a la llamada. Ejemplo:

```
/* info = pvm_send(tid , msgtag) --- Envía buffer de mensajes */
#include "header.h"
{INT_F_INT_INT(pvm_send)}
```

Listado 3.10: Ejemplo de uso del patrón P–4: comando PVMTB *pvm_send*.

Las seis rutinas que siguen este patrón son: pvm_send, _recv, _nrecv, _probe, _setopt, _sendsig.

P–5: [int, int[]] = f (void) Patrón:

```
#define INT_INTARR_F_VOID(NAME)                                     \
        int i, n, *arr;                                             \
        FUNCALL n = NAME(&arr);                                     \
        if(n>0) plhs[1]= mxCreateDoubleMatrix(n,1,mxREAL);          \
        for (i=0; i<n; i++) mxGetPr(plhs[1])[i] = arr[i];
```

Listado 3.11: Definición del patrón P–5 en el fichero header.h.

Se devuelve como primer argumento de retorno n, el tamaño de arr. Si no hay errores, se crea y devuelve como segundo argumento el propio arr, previa coerción a **double**. Este patrón se utiliza para pvm_getfds (Listado 3.12) y pvm_siblings.

```
/* [nfds fds] = pvm_getfds --- sockets abiertos bajo PVM */
#include "header.h"
{INT_INTARR_F_VOID(pvm_getfds)}
```

Listado 3.12: Ejemplo de uso del patrón P–5: comando PVMTB *pvm_getfds*.

P–6: int = f ('str') El patrón se muestra en el Listado 3.13:

```
#define INT_F_STR(NAME)                                             \
        char*str;                                                   \
        if ((nrhs!=1) || (! mxIsChar(prhs[0])))                     \
                mexErrMsgTxt("se requiere 1 arg string");           \
        str   =   mxArrayToString     (prhs[0]);                    \
        FUNCALL NAME(str);                                          \
        mxFree        (str);
```

Listado 3.13: Definición del patrón P–6 en el fichero header.h.

Tras comprobar que el único argumento es *string*, se procede a la llamada. Se libera entonces la memoria reservada por la traducción a string C. Ejemplo:

```
/* mstat = pvm_mstat ('host') --- Estado de máquina PVM */
#include "header.h"
{INT_F_STR(pvm_mstat)}
```

Listado 3.14: Ejemplo de uso del patrón P–6: comando PVMTB *pvm_mstat*.

Este patrón lo reutilizan las siguientes 8 rutinas PVMTB: pvm_mstat, _perror, _archcode, _gsize, _joingroup, _lvgroup, _export, _unexport.

P–7: int = f ('str', int) Patrón:

```
#define INT_F_STR_INT(NAME)                                     \
        char    *str;                                           \
        if ((nrhs!=2)    || (! mxIsChar    (prhs[0]))           \
                         || (! mxIsNumeric(prhs[1])))           \
                mexErrMsgTxt("requiere 1 arg string y 1 arg numérico"); \
        str  =  mxArrayToString         (prhs[0]);              \
        FUNCALL NAME( str,     mxGetScalar(prhs[1]));           \
        mxFree       (str);
```

Listado 3.15: Definición del patrón P–2 en el fichero header.h.

Sólo se añade 1 argumento **int** al patrón P–6. Ejemplo:

```
/* info = pvm_barrier('group', count) --- Sincronización grupo */
#include "header.h"
{INT_F_STR_INT( pvm_barrier )}
```

Listado 3.16: Ejemplo de uso del patrón P–7: comando PVMTB *pvm_barrier*.

Se reutiliza en estas cinco rutinas PVMTB: pvm_getinst, _gettid, _barrier, _bcast, _freezegroup.

P–8: int = f ('str', int, int) El patrón se muestra en el Listado 3.17.

```
#define INT_F_STR_INT_INT(NAME)                                 \
        char*str;                                               \
        if ( (nrhs!=3)   || (! mxIsChar    (prhs[0]))           \
                         || (! mxIsNumeric(prhs[1]))            \
                         || (! mxIsNumeric(prhs[2])) )          \
                mexErrMsgTxt("requiere 1 arg string y 2 args numéricos"); \
        str  =  mxArrayToString         (prhs[0]);              \
        FUNCALL NAME( str,     mxGetScalar(prhs[1]),            \
                               mxGetScalar(prhs[2]));           \
        mxFree       (str);
```

Listado 3.17: Definición del patrón P–8 en el fichero header.h.

Se añaden 2 argumentos **int** al patrón P–6. Este patrón se utiliza en las rutinas de *mailbox* pvm_putinfo (Listado 3.18), _recvinfo y _delinfo.

```
/* index = pvm_putinfo('name', bufid, flags) --- Msg a Mailbox */
#include "header.h"
{INT_F_STR_INT_INT( pvm_putinfo )}
```

Listado 3.18: Ejemplo de uso del patrón P–8: comando PVMTB *pvm_putinfo*.

P–9: [int, int[]] = f (str[]) El patrón se muestra en el Listado 3.19.

Tras comprobar cuántos argumentos hay y que son *strings*, se reserva memoria, se realiza la traducción a string C, y se procede a la llamada. También se reserva memoria para los códigos de retorno, que se devuelven como segundo argumento de retorno. Se libera entonces la memoria reservada.

```
#define INT_INTARR_F_STRARR(NAME)                              \
        char** hosts;                                          \
        int    *infos, i;                                      \
                                                               \
        if (nrhs==0)                                           \
                mexErrMsgTxt("se requiere al menos 1 arg");    \
        for (i=0; i<nrhs; i++)                                 \
          if (! mxIsChar(prhs[i]))                             \
                mexErrMsgTxt("args deben ser strings");        \
                                                               \
        infos = mxCalloc(nrhs, sizeof(int  ));                 \
        hosts = mxCalloc(nrhs, sizeof(char*));                 \
        for (i=0; i<nrhs; i++)                                 \
                hosts[i] = mxArrayToString(prhs[i]);           \
                                                               \
        FUNCALL NAME(hosts, nrhs, infos);                      \
        plhs[1]= mxCreateDoubleMatrix(nrhs,1,mxREAL);          \
        for (i=0; i<nrhs; i++){                                \
                mxGetPr(plhs[1])[i] = infos[i];                \
                mxFree (hosts[i]);                             \
        }                                                      \
        mxFree(hosts);                                         \
        mxFree(infos);
```

Listado 3.19: Definición del patrón P–9 en el fichero header.h.

El patrón se utiliza en las rutinas pvm_addhosts (Listado 3.20) y pvm_delhosts.

```
/* [numh infos] = pvm_addhosts(hosts) --- Añade hosts a PVM */
#include "header.h"
{INT_INTARR_F_STRARR(pvm_addhosts)}
```

Listado 3.20: Ejemplo de uso del patrón P–9: comando PVMTB *pvm_addhosts*.

Aunque aparentemente sea poco rentable codificar un patrón para sólo dos rutinas PVMTB, se ha de tener en cuenta que esta metodología permite codificar una única vez la funcionalidad utilizada, evitando errores de codificación adicionales para la segunda rutina.

El código queda además anotado junto a los demás patrones, donde es fácil detectar si una mejora a algún patrón (al anterior, por ejemplo) es susceptible de ser aplicada a éste también.

Cada vez que se reutiliza un patrón se anota junto al mismo el nombre de la rutina, convirtiéndose esta relación en una poderosa forma de autodocumentación de la *Toolbox*: la Tabla 3.1 se ha construido a partir de dicha relación de rutinas. Si hubiera que repasar la adscripción de rutinas a patrones (al objeto de añadir patrones alternativos a alguna rutina, por ejemplo), esta relación facilita enormemente la revisión, sugiriendo qué otras rutinas y patrones podrían ser susceptibles de admitir la misma extensión.

3.4 Eficiencia de PVMTB

En el Capítulo 1 ya se abordó la tarea de comparar las distintas *Toolboxes* paralelas MATLAB entre sí, y con los propios sistemas PVM y LAM/MPI programados directamente en lenguaje C. Las herramientas usadas a tal fin fueron un estudio de escalabilidad de una aplicación sencilla, y un test *ping-pong* de barrido corto, hasta 1500 bytes.

En el Capítulo 2 se abordó un estudio mucho más ambicioso de los sistemas PVM y LAM/MPI, usando para ello unos tests *ping-pong* con barridos mucho más extensos (32KB, 32MB) e introduciendo unos modelos específicamente diseñados para predecir el tiempo de transmisión de un mensaje en función de su tamaño.

Aunque las prestaciones de PVMTB y MPITB ya han sido identificadas como superiores a las de las otras *Toolboxes* paralelas, la comparación directa con los sistemas PVM y LAM/MPI no es tan inmediata. Los resultados del estudio de escalabilidad no son equiparables debido al fuerte *overhead* de cómputo bajo MATLAB, que altera la relación comunicación/cómputo en perjuicio del lenguaje C. La granularidad de la aplicación es más elevada bajo MATLAB, obteniéndose *speedups* superiores.

En este apartado nos centraremos pues en tests *ping-pong* de barrido más extenso usando PVM y PVMTB. Estos tests proporcionan los datos requeridos para una evaluación más detallada del *overhead* introducido por PVMTB en el paso de mensajes usando el sistema PVM.

3.4.1 Test *ping-pong* en C

Para estimar la pérdida de prestaciones introducida por PVMTB, se programó el mismo test *ping-pong* tanto en C como en MATLAB. Los argumentos del programa permiten especificar la modalidad de empaquetamiento (encode) y encaminamiento (droute), al objeto de ejercitar las diversas opciones del sistema PVM. Se usó la configuración estándar de PVM con UDPMAXLEN==4K−16.

Para poner de manifiesto la diferencia entre las opciones de empaquetamiento PvmDataDefault y PvmDataRaw, se utilizaron tanto mensajes conteniendo **double**s como mensajes conteniendo **char**s. Es de esperar que el empaquetado de arrays **double** sea menos eficiente que el de **char**s bajo la opción PvmDataDefault, debido a la codificación XDR adicional, y que sea igual de eficiente bajo la opción PvmDataRaw: a igualdad de tamaño en bytes (no en dimensionado), el tiempo de empaquetamiento y transmisión de ambos arrays debería coincidir.

Los test fueron ejecutados en el cluster “oxígeno”. Como se comentó en el Apartado 2.1, consta de 8 Pentium II 333MHz, 128MB RAM, con servidor de disco Pentium II 400MHz, 128MB RAM, 14GB disco, interconectados mediante un conmutador (*switch*) de 100Mbps BayStack 350T, con 16 puertos.

En el servidor de disco se ejecuta un *script* de automatización (ver Listado D.1 en Apéndice D) el cual arranca el programa *fuente* en el servidor y el programa *eco* en otro computador del cluster. Para evitar realizar varios barridos de diferente extensión y paso, los tamaños de mensaje se van incrementando casi exponencialmente, según el bucle mostrado en el Listado 3.21. El tamaño del array crece linealmente hasta el triple (1x1, 1x2, 1x3), y entonces se produce el paso exponencial (2x2, 2x4, 2x6...). Se lleva una contabilidad indirecta del dimensionado a través de un índice auxiliar (indx en el Listado 3.21). Este tipo de barrido permite alcanzar tamaños grandes con un tiempo de medición razonable y sin sacrificar el detalle en los tamaños pequeños.

```
/* BW.c: Medición de latencia (ping-pong) y ancho de banda */
#define SWAPLIM 30
char lat=' ';                                   /* pvm_pkbyte para latencia */
...
  for (ExpAncho=0; ExpAncho<13; ExpAncho++){
    Ancho=1<<ExpAncho;
    for (Alto=Ancho; Alto<4*Ancho; Alto+=Ancho){
      indx++; nelem=Ancho*Alto;

      if (((indx>=Dstr)&&(indx<=Dend))||        /* algo que hacer */
         ((indx>=Ustr)&&(indx<=Uend))){

        NTIMES=ntimes*reps(0);
        pvm_barrier("BW",2);                    /* hacer parte latencia */

        T=Wtime();
        for (i=0; i<NTIMES; i++){
          pvm_initsend(encode); pvm_pkbyte(&lat,0,1); pvm_send(other,TAG);
          pvm_recv(other,TAG);  pvm_upkbyte(&lat,0,1);
        }
        T=Wtime()-T;            l[indx]=T/2/NTIMES;
      }
      ...                                       /* hacer parte double */
    }
    ...
  }
...
```

Listado 3.21: Programa *ping-pong* para PVM: bucle de tamaños y sección dedicada a la latencia. La dedicada a transmitir **double**s se ofrece en el Listado 3.22. El Listado D.2 muestra el programa completo.

Un argumento adicional del programa de medición permite especificar los valores máximos Uend−Dend deseados para indx, y por tanto el tamaño máximo de barrido. Éste se limita a tamaños que no produzcan *swapping*; el límite de dimensionado para arrays **double** (Dend) es por tanto menor que para **char**s (Uend). Teniendo bajo esta arquitectura y compilador tamaños **sizeof**(**char**)=1 y **sizeof**(**double**)=8, el valor de indx para alcanzar (2^{25}=) 32MB es Uend=37 para **char**s (4096x8192 x 1B = 2^{12}x2^{13}) y Dend=33 para **double**s (2048x2048 x 8B = 2^{11}x2^{11}x2^{3}). Tamaños superiores provocan acceso a la partición de *swap* ("intercambio"), lo cual no es objeto de nuestra medición. Debido a la copia a memoria adicional bajo las opciones de empaquetamiento PvmDataDefault y PvmDataRaw, se produce bajo las mismas un pequeño *swapping*, apenas detectable mediante inspección visual de las gráficas.

Así pues, el bucle de tamaños realiza una barrido casi exponencial sobre las dimensiones del array, y si éstas son apropiadas para el tipo **double**, Dstr $\leq$ indx $\leq$ Dend, se realizan la transmisión y medición correspondientes. Lo mismo sucede con **char** cuando Ustr $\leq$ indx $\leq$ Uend)

En cada iteración del barrido de tamaños, como se observa en la condición **if** del Listado 3.21, se vuelve a estimar la latencia como el tiempo de ida del mensaje vacío. Se utiliza a dicho fin un *string* nulo, lat en el Listado 3.21. Se ha definido una función auxiliar reps() para repetir más veces los tamaños menores; así, reps(0)==256 y reps(32MB)==1. Un argumento adicional del programa, ntimes, se usa como factor para el número de repeticiones de cada medición. En concreto, el bucle *ping-pong* para la latencia se repite ntimes∗reps(0)==256 ntimes veces.

El programa continúa realizando la medición para datos **double**, como se muestra en el Listado 3.22. Se decidió medir también el tiempo de reserva de memoria para poder comparar con MATLAB, y también para tener una idea más exacta del coste relativo de las operaciones de paso de mensajes frente a la reserva de memoria.

```
if  ((indx>=Dstr)&&(indx<=Dend)){                    /* hacer parte double */

        NTIMES=ntimes*reps(nelem*sizeof(double));   arrayDouble=NULL;
        T=Wtime();
        for (i=0; i<NTIMES; i++){                    /* medir reserva memoria */
          free(arrayDouble);
          arrayDouble=calloc(nelem, sizeof(double));
        }
        T=Wtime()-T;        d[indx][Size]=nelem*sizeof(double);
                            d[indx][ Mtm]=T/NTIMES;

        pvm_barrier("BW",2);                         /* ping-pong parte double */
        T=Wtime();
        for (i=0; i<NTIMES; i++){
          pvm_initsend(encode);  pvm_pkdouble(arrayDouble, nelem,1);
          pvm_send(other,TAG);
            if (indx>=SWAPLIM)   pvm_freebuf   (pvm_getsbuf()       );
          pvm_recv(other,TAG);   pvm_upkdouble(arrayDouble, nelem,1);
            if (indx>=SWAPLIM)   pvm_freebuf   (pvm_getrbuf()       );
        }
        T=Wtime()-T;        d[indx][TXtm]=T/2/NTIMES;

        free(arrayDouble);        pvm_freebuf   (pvm_getsbuf());
                                  pvm_freebuf   (pvm_getrbuf());
}
```

Listado 3.22: Programa *ping-pong* para PVM (continuación). Sección dedicada al tiempo de transmisión de **double**s.

La sección para **char**s que sigue a ésta es idéntica, salvo que se usan los límites Ustr y Uend en el **if**, el array arrayUint8 en calloc() y free(), el tipo **unsigned char** en **sizeof**(), y la llamada PVM [u]pkbyte() en lugar de [u]pkdouble(). Las mediciones se anotan en el array u en lugar de d.

Las Figuras de la 3.1 a la 3.6 muestran las mediciones correspondientes a la ejecución de este código C con las seis combinaciones de opciones PVM contempladas. Las tres primeras corresponden a las tres opciones de empaquetado PvmDataDefault, PvmDataRaw y PvmDataInPlace con encaminamiento a través del *daemon* PVM. Las tres siguientes, con ruta TCP directa entre las tareas PVM. Cada figura muestra dos hileras de tres gráficas.

En el pie de la primera figura se explica la estructura común a todas ellas: las gráficas a la derecha muestran el barrido logarítmico completo, mientras que a la izquierda se hace un *zoom* para el rango de decenas de KBs, y en el centro para decenas de MBs. La parte superior se dedica a las mediciones de tiempo, y la inferior muestra el ancho de banda calculado con el modelo lineal $T = S/B$. El modelo afín sólo es significativamente diferente para tamaños más pequeños que los mostrados en las ampliaciones.

Como muestra la leyenda en la gráfica central, el color verde se dedica a los datos **double**, y el azul a los **char**. Los colores más claros se refieren al paso de mensajes, y los más oscuros a la reserva de memoria, que sólo se muestra al objeto de hacerse una idea intuitiva del coste relativo de ambas operaciones. En la parte inferior se observa que el *ancho de banda de memoria* es muy superior al de paso de mensajes, saliéndose de la gráfica en cuanto progresa un poco el barrido.

La expresión *ancho de banda* es desafortunada en este caso, ya que sólo estamos reservando el bloque, no copiando datos a él. El tiempo de reserva depende del gestor de memoria Linux, que tiene clasificada la memoria libre en bloques de distintos tamaños (potencias de dos), añadiendo un número variable de ellos a la tabla de páginas de un proceso según el tamaño que éste le

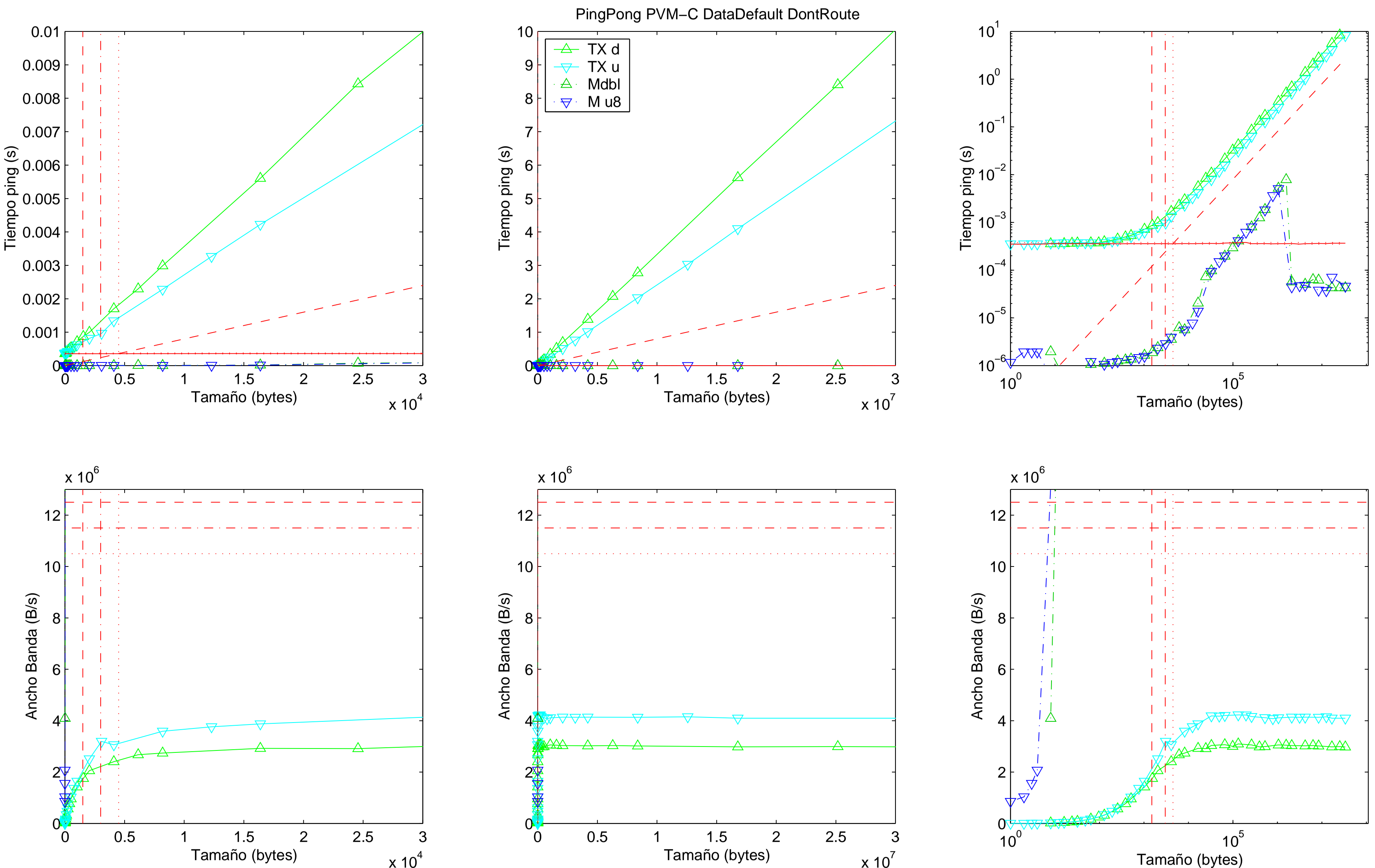


Figura 3.1: Test *ping-pong* bajo PVM, con opción PvmDataDefault. **Derecha**: gráfica logarítmica incluyendo todas las mediciones. **Izquierda**: detalles con abscisa lineal en los rangos de decenas de KB y MB. **Abajo**: anchos de banda correspondientes. **Leyenda**: **TX d**: tiempo de transmisión (ida) para array **double**. **TX u**: para array **char**. **Mdbl**: tiempo para reserva de memoria de array **double**. **M u8**: para array **char**.

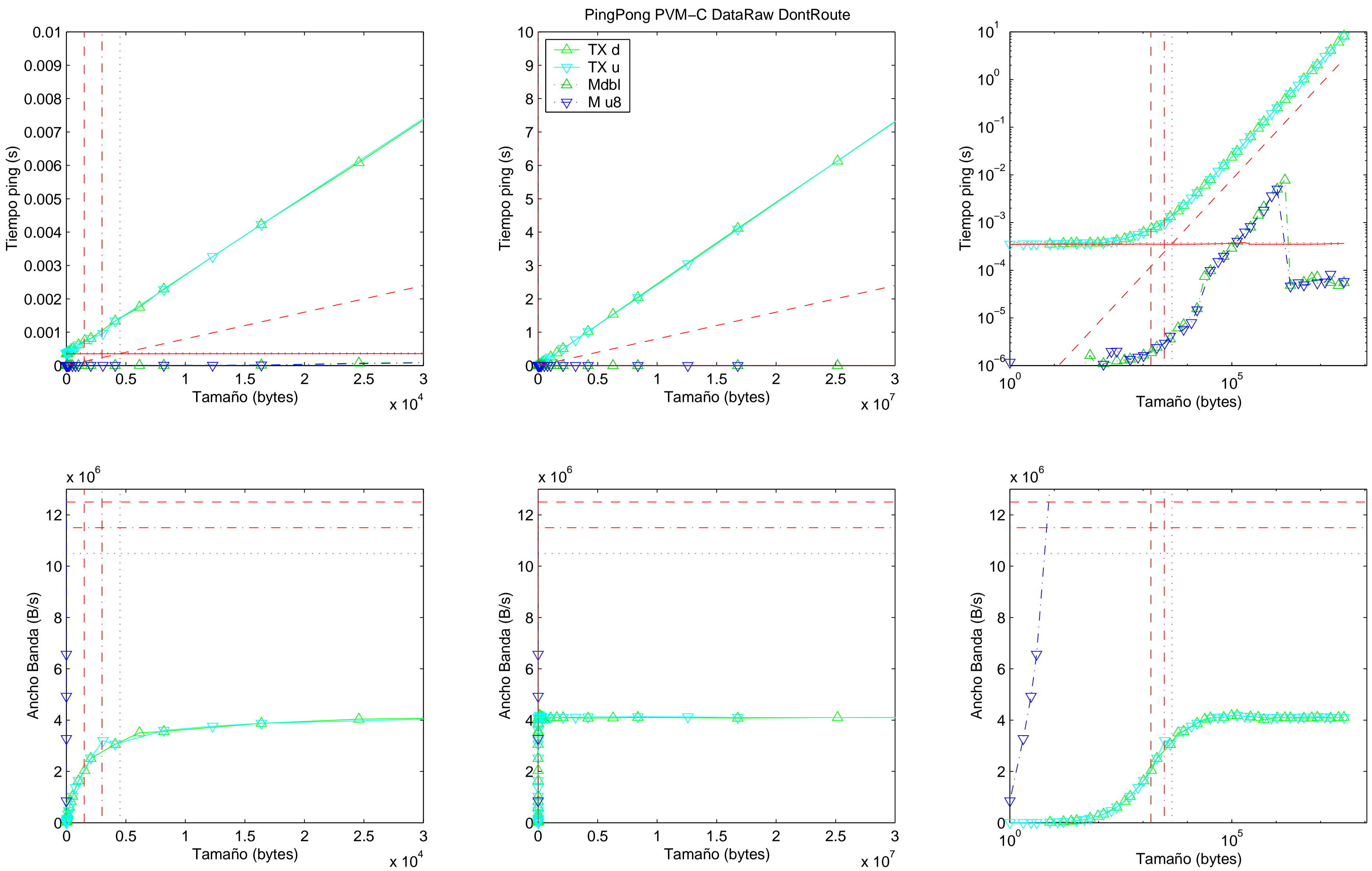


Figura 3.2: Test *ping-pong* bajo PVM, con opción PvmDataRaw. Consultar pie de Figura 3.1 para más detalles.

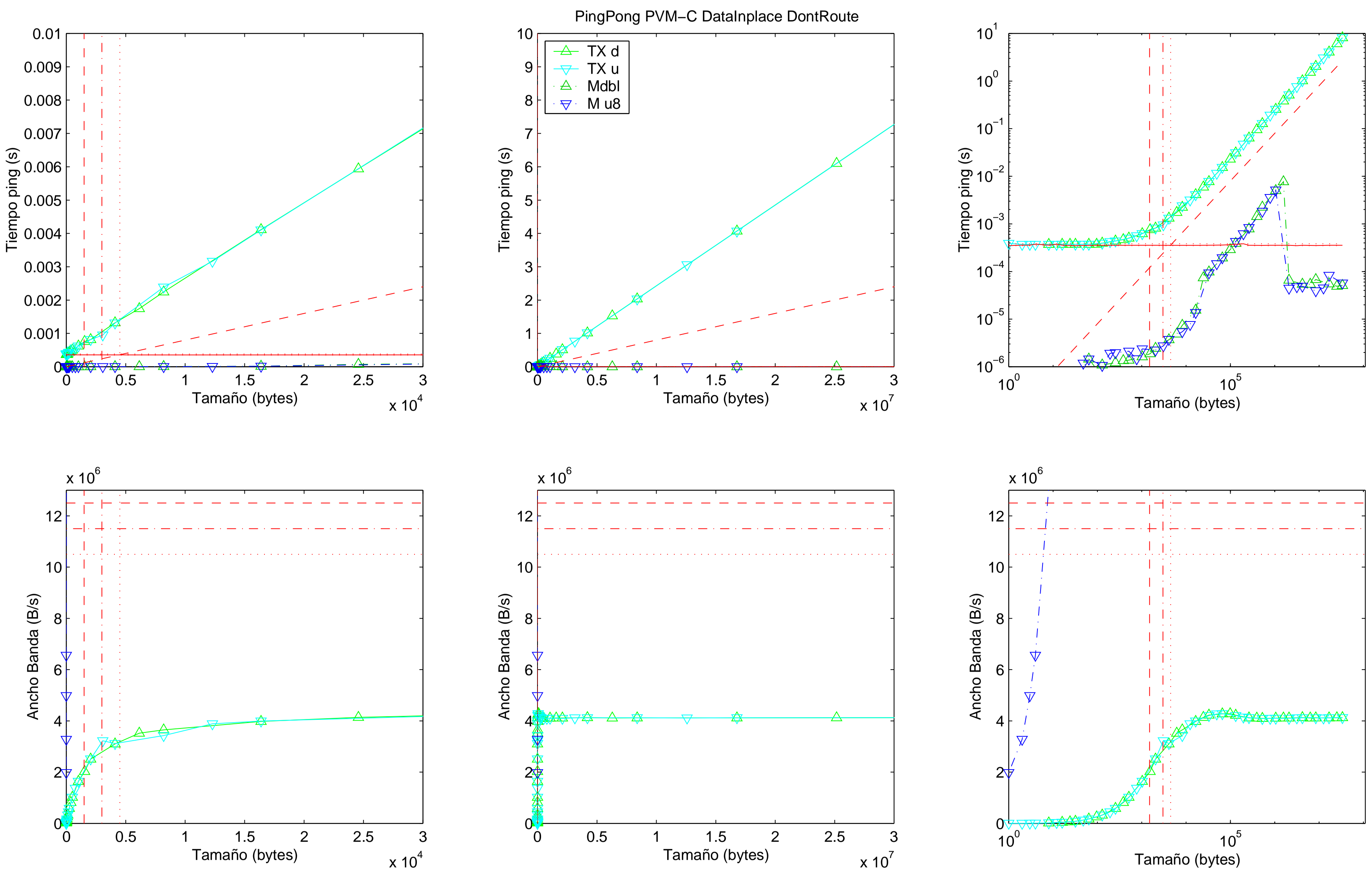


Figura 3.3: Test *ping-pong* bajo PVM, con opción PvmDataInPlace. Consultar pie de Figura 3.1 para más detalles.

pida. La forma de campana de la gráfica superior derecha, algo deformada por el eje de abscisas logarítmico, muestra el relativo trabajo que le cuesta al Sistema Operativo proporcionar bloques en el rango de decenas de KBs. Por encima de MBs, el tiempo de reserva se estabiliza alrededor de 40μs. Por debajo de decenas de KBs, se estabiliza en μsegundos o menos. El pico de casi una centésima de segundo para 1–2MB es un artificio debido a las repetidas liberaciones de los buffers de envío y recepción (sentencias **if** (indx>=SWAPLIM) en Listado 3.22) necesarias para no producir *swapping* bajo algunos modos de transmisión. Estas operaciones complican el normal funcionamiento del gestor de memoria.

En la Figura 3.6, correspondiente al empaquetamiento *InPlace* con ruta TCP directa, se observa que el pico no es tan pronunciado, llegando sólo a 0.2–0.3ms en 64–96KBs, y el tiempo de reserva se estabiliza fuera del rango de decenas de KBs. Este es un comportamiento más normal del gestor de memoria Linux, y lo veremos reproducido en el siguiente capítulo con MPI. Bajo esta modalidad (ruta directa, empaquetamiento *InPlace*) el buffer de transmisión consiste únicamente en punteros a los datos y tamaños, no existiendo copia adicional de memoria en transmisión, de manera que la repetida liberación de los buffers de transmisión no afecta tanto al gestor de memoria Linux. Aún se nota una ligera pérdida de prestaciones, achacable a la liberación de los buffers de recepción.

Las gráficas incluyen trazos rojos punteados de referencia: verticales para los tamaños de 1, 2 y 3 MTUs* (*Maximum Transfer Unit*, 1500 bytes por defecto bajo Linux), y horizontales para 10.5, 11.5 y 12.5MB/s. Éste último serían 100Mbps, la velocidad del propio *switch* BayStack. Naturalmente, en la ampliación central (decenas de MBs) las referencias verticales de 1-3 MTU están apelmazadas en el origen de abscisas. La gráfica logarítmica (derecha) muestra que la referencia 3MTU divide aproximadamente por la mitad el número de mediciones realizadas. Las mediciones se aglomeran a la izquierda de las referencias en la ampliación izquierda (decenas de KBs). El trazo diagonal en las gráficas de la mitad superior (mediciones de tiempo) corresponde a la referencia 12.5MB/s. En la mitad inferior (anchos de banda) las referencias son horizontales, naturalmente.

También aparecen en color rojo las repetidas estimaciones de la latencia realizadas en cada iteración del bucle que barre los tamaños de array (Listado 3.21). Bajo algunas combinaciones de opciones en PVMTB se ha observado una variabilidad importante de la latencia según el tamaño de la transmisión anterior a su estimación, por lo que se encierra el trazo entre dos líneas punteadas correspondientes a las estimaciones máxima y mínima. En PVM bajo C no se observa este efecto, siendo la latencia perfectamente reproducible desde la Figura 3.1 a la 3.6.

Quedan por comentar los trazos del tiempo de transmisión (colores verde y azul claro), que son el principal motivo del estudio. Usando empaquetamiento PvmDataDefault a través del *daemon* (Figura 3.1) los **double**s deben ser codificados XDR, operación que se añade al tiempo de transmisión (separación azul-verde en ampliaciones arriba izquierda/centro) disminuyendo el ancho de banda efectivo en comparación con **char**s (de 4 a 3MB/s abajo derecha). Bajo PvmDataRaw se ignora la codificación XDR, igualándose el ancho de banda para ambos tipos de datos en 4MB/s (Fig. 3.2 abajo derecha). La opción PvmDataInPlace no supone una ventaja significativa a través del *daemon*, ya que el mensaje se copia de todas maneras. Se aprecia tal vez una ligerísima ventaja en el rango de decenas de KBs (cresta en Fig. 3.3 abajo derecha, ampliación arriba izquierda). Los valores exactos de anchos de banda se ofrecen más adelante en la Tabla 3.2, junto con los del encaminamiento directo (Figuras de la 3.4 a la 3.6) para permitir una fácil comparación.

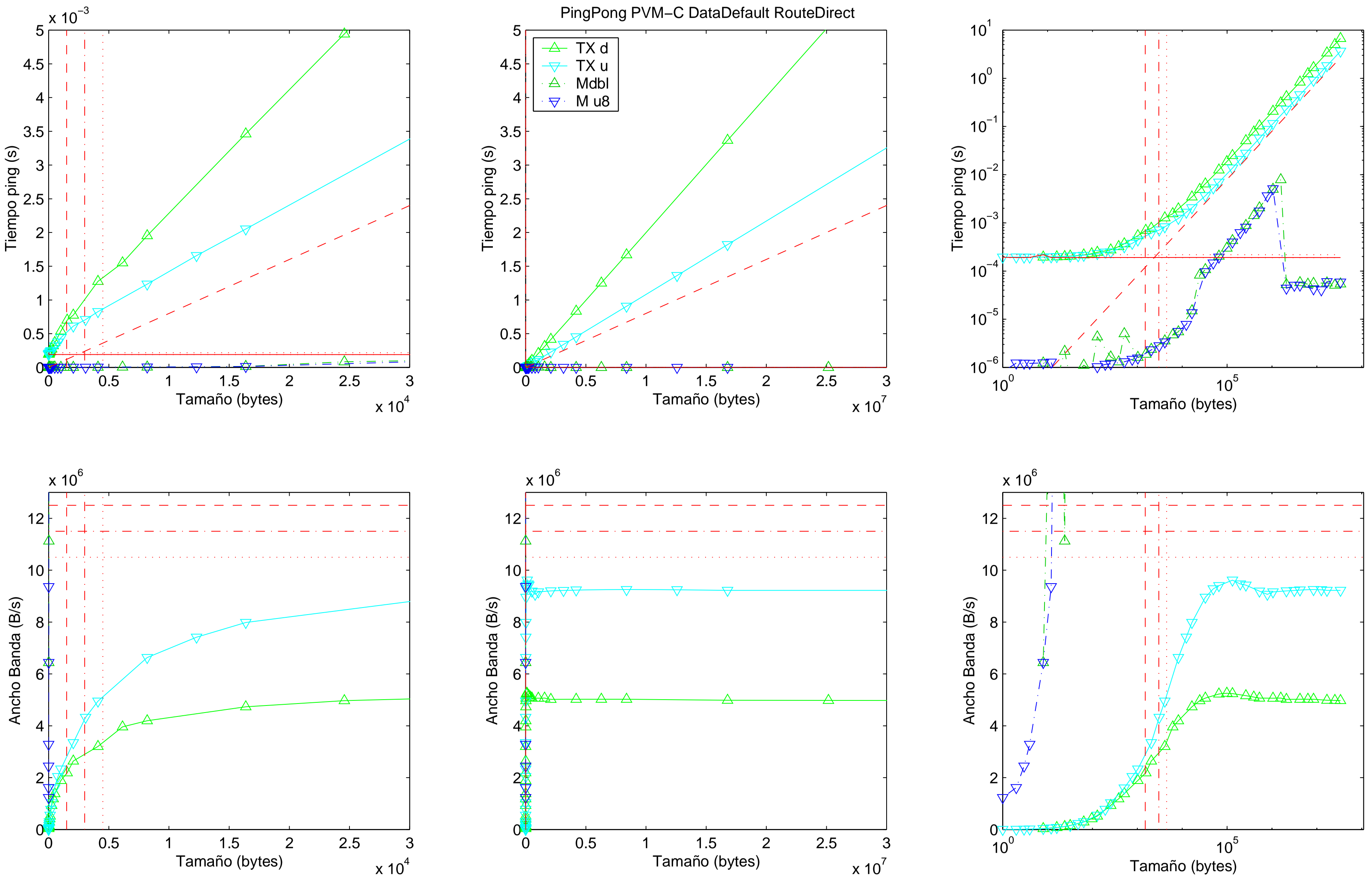


Figura 3.4: Test *ping-pong* bajo PVM, con opciones PvmDataDefault y PvmRouteDirect. Consultar pie de Figura 3.1 para más detalles.

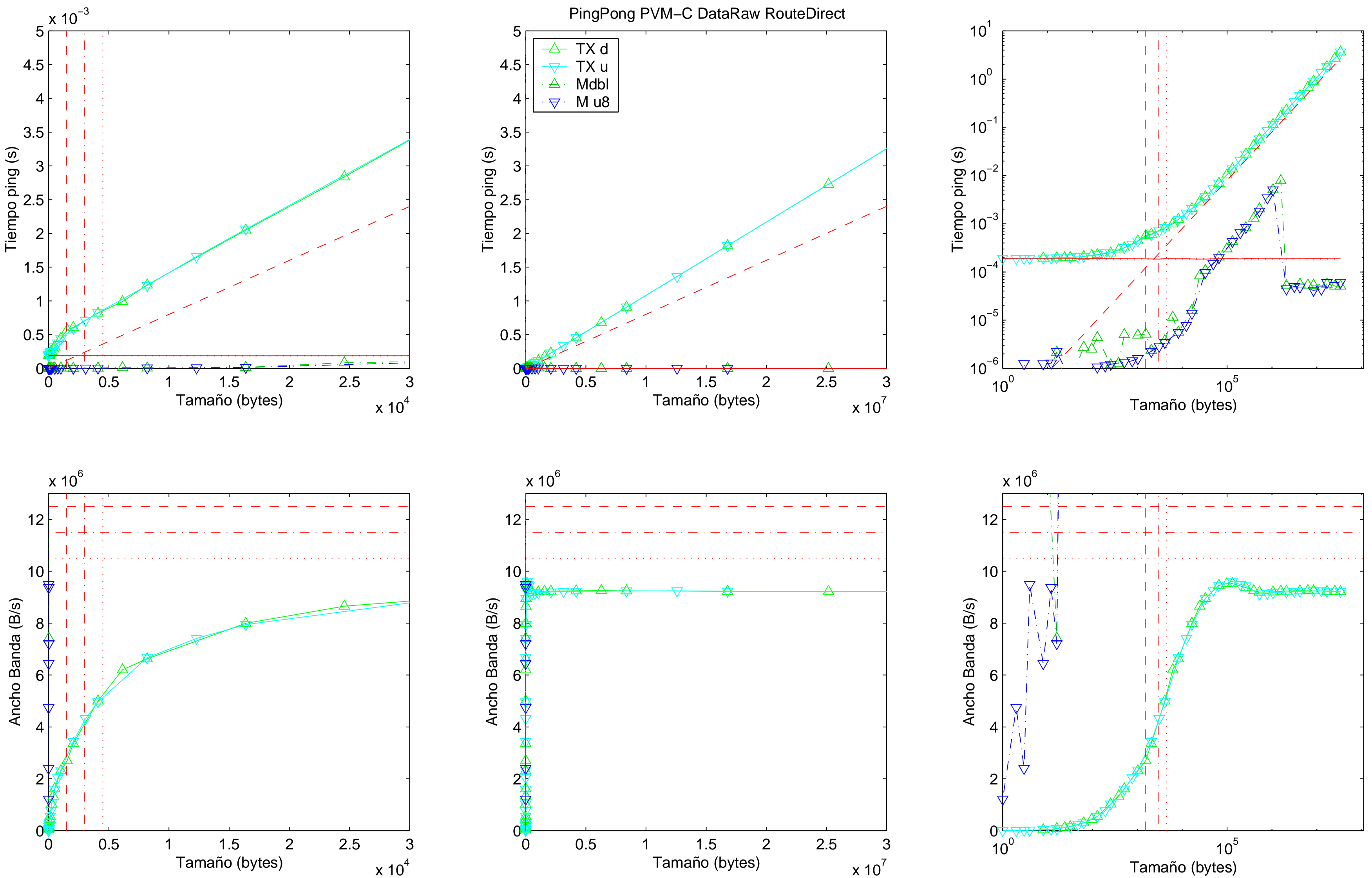


Figura 3.5: Test *ping-pong* bajo PVM, con opciones PvmDataRaw y PvmRouteDirect. Consultar pie de Figura 3.1 para más detalles.

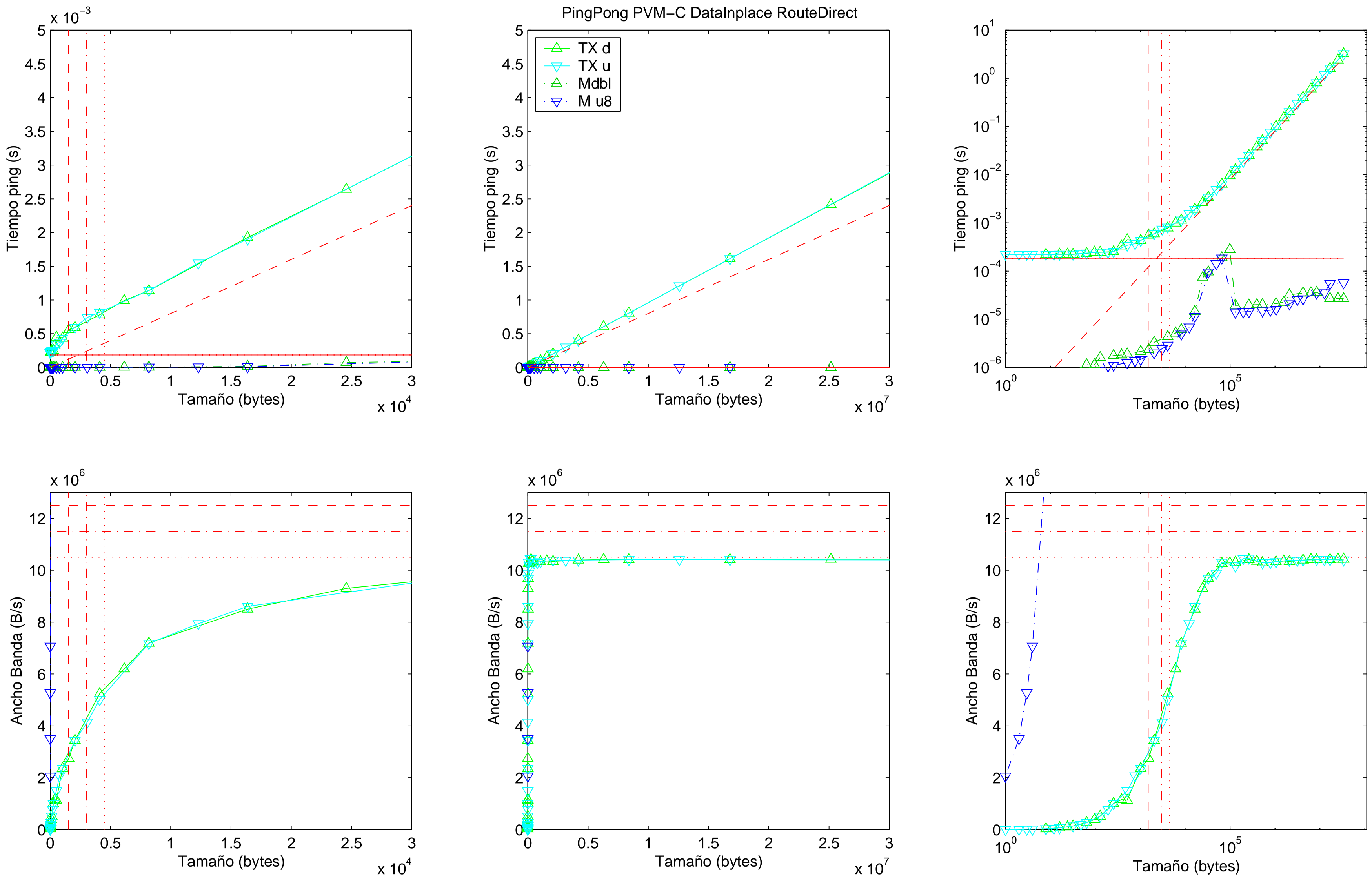


Figura 3.6: Test *ping-pong* bajo PVM, con opciones PvmDataInPlace y PvmRouteDirect. Consultar pie de Figura 3.1 para más detalles.

Las tres siguientes figuras añaden la opción PvmRouteDirect, creándose pues una ruta TCP directa entre las tareas fuente y eco en lugar de usar la ruta por defecto a través de los *daemons* PVM. Los anchos de banda se duplican.

Para empaquetamiento DataDefault con codificación XDR, la diferencia 3–4MB/s entre **char-double** en la Figura 3.1 abajo derecha pasa a 5–9.5MB/s en la Figura 3.4 con ruta directa. Con codificación DataRaw se pasa de 4MB/s en la Figura 3.2 a 9.5MB/s en la Figura 3.5. La mejor combinación de opciones para el test *ping-pong* es la última, DataInPlace con ruta directa, que ahora sí presenta diferencia notable con DataRaw. Ya se comentó previamente la escasa diferencia entre las gráficas inferiores derecha de las Figuras 3.2 y 3.3. En las Figuras 3.5 y 3.6 se pasa de 9.5–9.25MB/s a 10.5MB/s. El incremento respecto a no usar ruta directa (4.1MB/s en la Figura 3.3) es más del doble.

Merece la pena resaltar el mejor comportamiento del gestor de memoria con la última combinación de opciones. En la Figura 3.6 superior derecha se observa que el pico del tiempo de reserva se restringe al rango de decenas de KBs y es más reducido, hasta 200–300μs. Al usar la opción DataInPlace con ruta directa, el buffer de envío consiste únicamente en contadores de tamaño y punteros a los datos del usuario, de manera que su liberación no afecta a posteriores reservas de memoria (de mucho mayor tamaño) para datos del usuario.

Se ofrece ahora la Tabla 3.2, en la cual se presentan resumidamente los detalles ya destacados sobre el test *ping-pong* en PVM.

	BW (MB/s)		Latencia	Reserva Memoria (μs)	
	char	**double**	(μs)	extremos	pico
def-no	4.2– 4.1	3.1–3.0	359	1–48	7.9ms@1.5MB
raw-no	4.2– 4.1		356	1–58	7.8ms@1.5MB
pla-no	4.3– 4.1		356	1–57	7.7ms@1.5MB
def-di	9.6– 9.2	5.3–5.0	192	1–59	7.9ms@1.5MB
raw-di	9.6– 9.2		187	1–61	7.8ms@1.5MB
pla-di	10.5–10.4		185	1–27	281μs@ 96KB

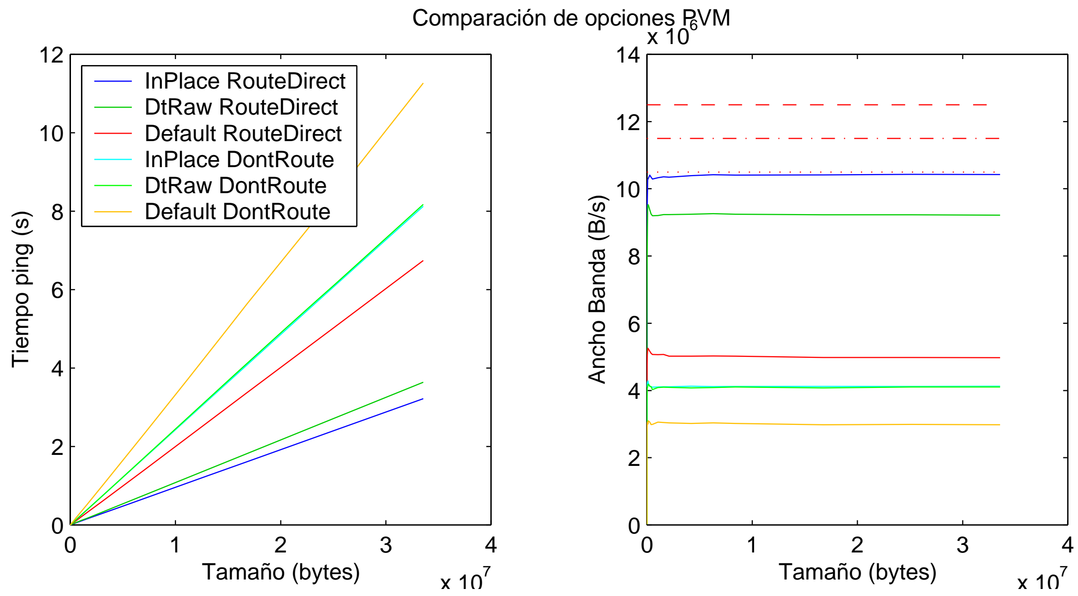
Tabla 3.2: Detalles remarcables del test *ping-pong* en PVM.

Se indican los tiempos típicos para reserva de memoria al principio y final del barrido, así como la posición del máximo tiempo de reserva. La variabilidad observada es propia del gestor de memoria Linux, no estando asociada a la combinación de opciones PVM (salvo para el citado caso *PvmDataInPlace-RouteDirect*). Éstas se han abreviado a la nomenclatura utilizada con los nombres de fichero en esta memoria.

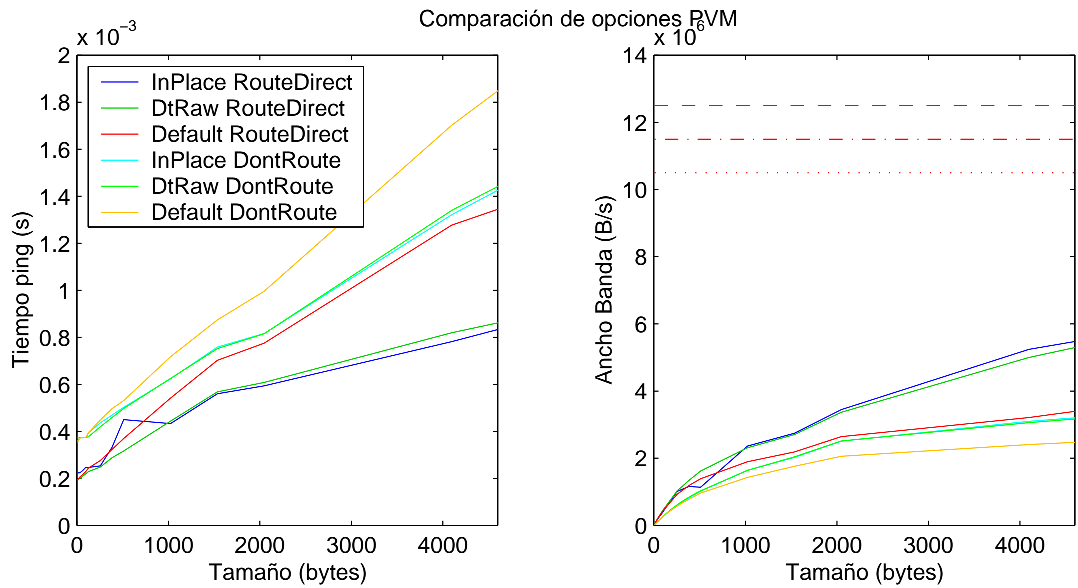
Respecto al paso de mensajes, la ruta directa incrementa el ancho de banda 4–9.5MB/s y disminuye la latencia 360–190μs. La codificación XDR añade *overhead* para datos **double** (4–3MB/s, 9.5–5.25MB/s), y el empaquetamiento *InPlace* sólo supone una mejora con ruta directa (9.5–10.5MB/s).

En la Figura 3.7 se presenta una gráfica adicional comparando los tiempos de transmisión de datos **double** bajo las distintas combinaciones de opciones PVM. Se ha escogido **double** para que quede evidenciada la diferencia entre los empaquetamientos DataDefault y DataRaw. La Figura 3.7(b) muestra una ampliación de la misma gráfica en las primeras 3 MTUs, en donde se puede observar una latencia de 0.2ms para ruta directa y 0.4ms a través del *daemon* PVM.

Más adelante (Tabla 3.4 izquierda) se presentan algunos puntos de la mejor opción PVM ("*pla-di*") con datos **double**, al objeto de posibilitar una comparación objetiva con PVMTB.



(a) Barrido completo.



(b) Detalle de las primeras 3 MTUs.

Figura 3.7: Test *ping-pong* bajo PVM, comparación de opciones con datos **double**.

3.4.2 Test *ping-pong* en MATLAB

El mismo test *ping-pong* programado en PVMTB bajo MATLAB toma la forma que se muestra en el Listado 3.23.

Se puede apreciar la similitud con la versión C. La notación MATLAB es más compacta, sobre todo comparando el código (no mostrado aquí) de inicialización de variables. Los argumentos del programa PVMTB controlan los mismos aspectos que el programa C equivalente del Listado 3.21: codificación, encaminamiento, tamaño del barrido y nº de repeticiones. La sección mostrada reestima la latencia en cada iteración del barrido. Como se comentó en el apartado anterior, bajo PVMTB se ha observado una mayor variabilidad de la latencia.

El bucle barre los mismos tamaños que se consideraron en el programa C. En la variable indx se lleva cuenta del índice de l []/ u []/ d[] en donde se almacenará el tiempo cronometrado. De nuevo se controla mediante variables auxiliares Dstr-Dend y Ustr-Uend los índices válidos para transmitir un array *double* o *uint8* de las dimensiones correspondientes sin producir excesivo *swapping*. Estos índices se calculan en función del tercer argumento, row.

El Listado 3.24 muestra la sección dedicada a cronometrar la transmisión de datos.

Al igual que se hizo en lenguaje C, además del tiempo de transmisión se mide también el tiempo de reserva de memoria bajo MATLAB (primer bucle del Listado 3.24). El siguiente bucle mide el costo de la conversión *double*→*uint8*. Como veremos en el ejemplo de aplicación del Capítulo 5, ésta es una operación frecuentemente usada en tratamiento de señal y en concreto de imágenes, que es uno de los campos ideales para la utilización de Computación Paralela. Es por tanto de vital importancia tener una noción bien fundamentada del coste relativo de dicha operación respecto a las de reserva de memoria y paso de mensajes. El tiempo de transmisión se estima finalmente en el último bucle del Listado 3.24. Al igual que se hizo en el apartado anterior bajo C, se tiene especial cuidado en liberar los buffers de envío y recepción para reducir el

```
function [l,u,d]=BW(encode,droute,row,ntimes)
%BW: Medición de latencia (ping-pong) y ancho de banda
...
 SWAPLIM=30; arrayNULL=[];

 for ExpAncho=0:12
   Ancho=2^ExpAncho;

   for Alto=Ancho:Ancho:3*Ancho
     indx=indx+1;

     if (indx>=Dstr & indx<=Dend) |...             % algo que hacer
        (indx>=Ustr & indx<=Uend)

       NTIMES=ntimes*reps(0);                       % hacer parte latencia
       pvm_barrier('BW',2); T=clock; for i=1:NTIMES
         pvm_initsend(encode); pvm_pkdouble( arrayNULL ); pvm_send(tids,TAG);
         pvm_recv(tids,TAG);   pvm_upkdouble( arrayNULL );
       end, T=etime(clock,T);
       l(indx)=T/2/NTIMES;
     end
     ...                                            % hacer parte double
   end
   ...
 end
```

Listado 3.23: Programa *ping-pong* para PVMTB: sección dedicada a la latencia. La dedicada a transmitir **double**s aparece en el Listado 3.24. El programa aparece completo en el Listado D.4.

```
if indx>=Dstr & indx<=Dend                          % hacer parte double

  NTIMES = ntimes*reps(Alto*Ancho*sizeofDbl);       % medir reserva memoria
  T=clock; for i=1:NTIMES
    clear arrayDouble
    arrayDouble(Alto,Ancho)=0;                      % liberar al final
  end, T=etime(clock,T); s=whos('arrayDouble');
  d(indx,Size)=s.bytes;
  d(indx, Mtm)=T/NTIMES;

  T=clock; for i=1:NTIMES                           % medir cambio de tipo
    arrayUint8 =uint8(arrayDouble);
    clear arrayUint8
  end, T=etime(clock,T);
  d(indx, Ctm)=T/NTIMES;

  pvm_barrier('BW',2); T=clock; for i=1:NTIMES      % ping-pong parte double
    pvm_initsend(encode); pvm_pkdouble( arrayDouble ); pvm_send(tids,TAG);
        if indx>SWAPLIM,  pvm_freebuf(pvm_getsbuf ); end
    pvm_recv(tids,TAG);  pvm_upkdouble( arrayDouble );
        if indx>SWAPLIM,  pvm_freebuf(pvm_getrbuf ); end
  end, T=etime(clock,T);
  d(indx,TXtm)=T/2/NTIMES;

  clear arrayDouble,        pvm_freebuf(pvm_getsbuf);
end
```

Listado 3.24: Programa *ping-pong* para PVMTB (continuación). Sección dedicada al tiempo de transmisión de *double*s.

swapping al mínimo posible. Estas liberaciones afectan a las prestaciones del gestor de memoria.

Las Figuras de la 3.8 a la 3.13 muestran los resultados de este código MATLAB con las seis combinaciones de opciones PVM. Se usa la misma organización que para las figuras de PVM, con la adición de las medidas de conversión de tipo *double→uint8*, en color morado. La reserva de memoria empeora espectacularmente, siendo éste una característica bien conocida del entorno MATLAB. No sólo el tiempo absoluto es mayor, sino que a partir de decenas de KBs aumenta proporcionalmente con el tamaño del array. Para tamaños menores consume alrededor de 0.1ms. La conversión de tipo consume algo menos hasta aproximadamente 1MTU (coincidencia no relacionada con el paso de mensajes), y un orden de magnitud más que la reserva de memoria para tamaños superiores a 3MTUs.

Las tres primeras Figuras, de la 3.8 a la 3.10, corresponden a las tres opciones de empaquetado *PvmDataDefault*, *PvmDataRaw* y *PvmDataInPlace* con encaminamiento a través del *daemon* PVM, obteniéndose las mismas conclusiones que en el test en lenguaje C. Con *DataDefault* la codificación XDR añade mayor *overhead* para datos *double* (4–3MB/s en Figura 3.8 inferior derecha). Con *PvmDataRaw* se evita la codificación XDR, igualando el ancho de banda *double* con el *uint8* (4MB/s). *PvmDataInPlace* a través del *daemon* sólo incrementa muy ligeramente el ancho de banda para ambos tipos de datos en el rango de decenas de KBs.

Comparando con las Figuras de la 3.1 a la 3.3 se puede comprobar que el *overhead* de PVMTB es mínimo, apenas discernible mediante inspección visual de las gráficas.

Las mediciones de latencia tienen mayor desviación bajo PVMTB (gráficas superiores izquierda), presentando picos característicos cuando se alterna su estimación con la transmisión de datos a partir del centenar de KBs. Este efecto desaparece al utilizar encaminamiento *PvmRouteDirect* en las siguientes tres gráficas (Figuras de la 3.11 a la 3.13). Al igual que bajo C, los anchos de banda se duplican.

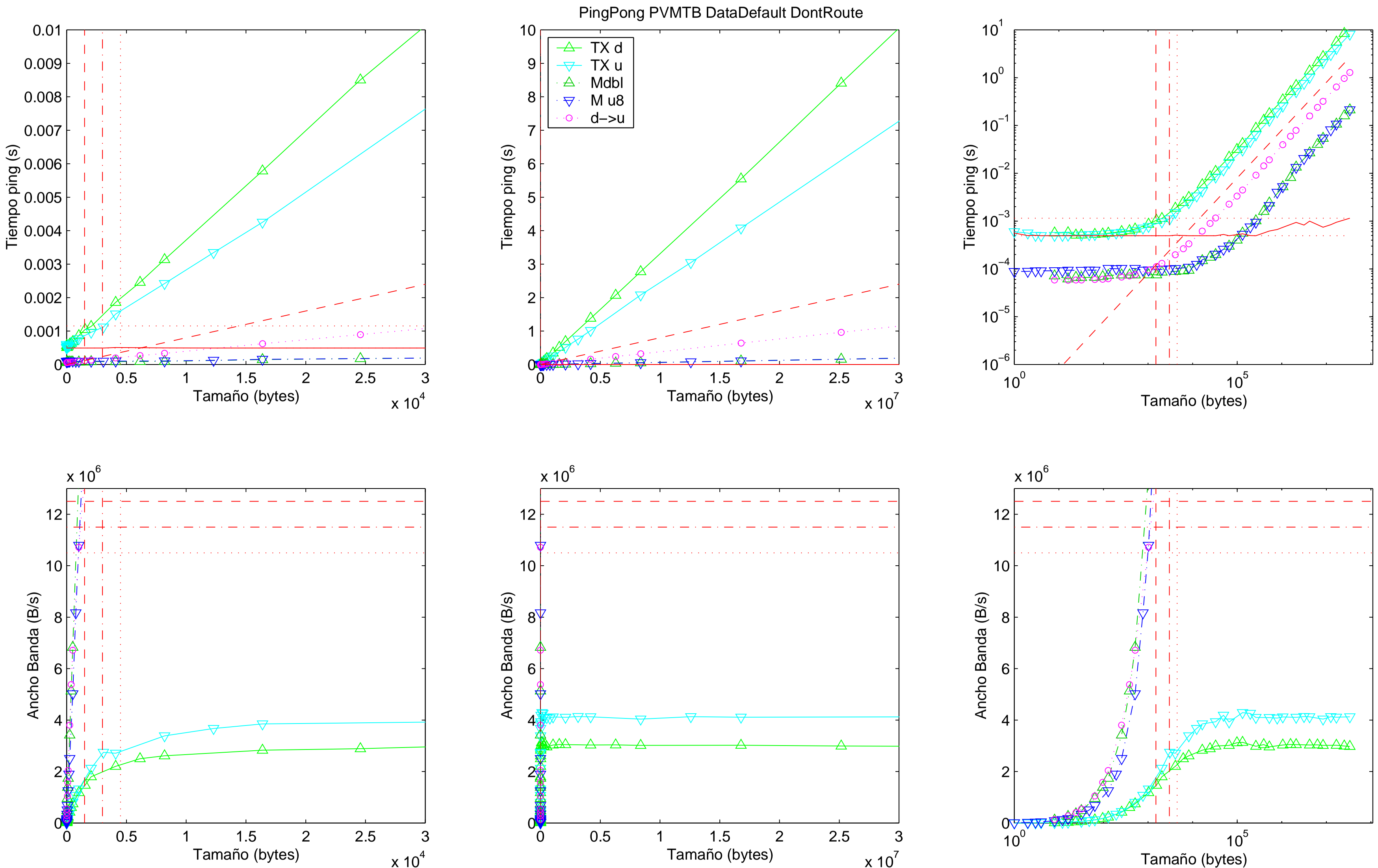


Figura 3.8: Test *ping-pong* bajo PVMTB, con opción **PvmDataDefault**. **Derecha**: gráfica logarítmica incluyendo todas las mediciones. **Izquierda**: detalles con abscisa lineal en los rangos de decenas de KB y MB. **Abajo**: anchos de banda correspondientes. **Leyenda**: **TX d**: tiempo de transmisión (ida) para array *double*. **TX u**: para array *uint8*. **Mdbl**: tiempo para reserva de memoria de array *double*. **M u8**: para array *uint8*. **d→u**: tiempo de conversión *double*→*uint8*.

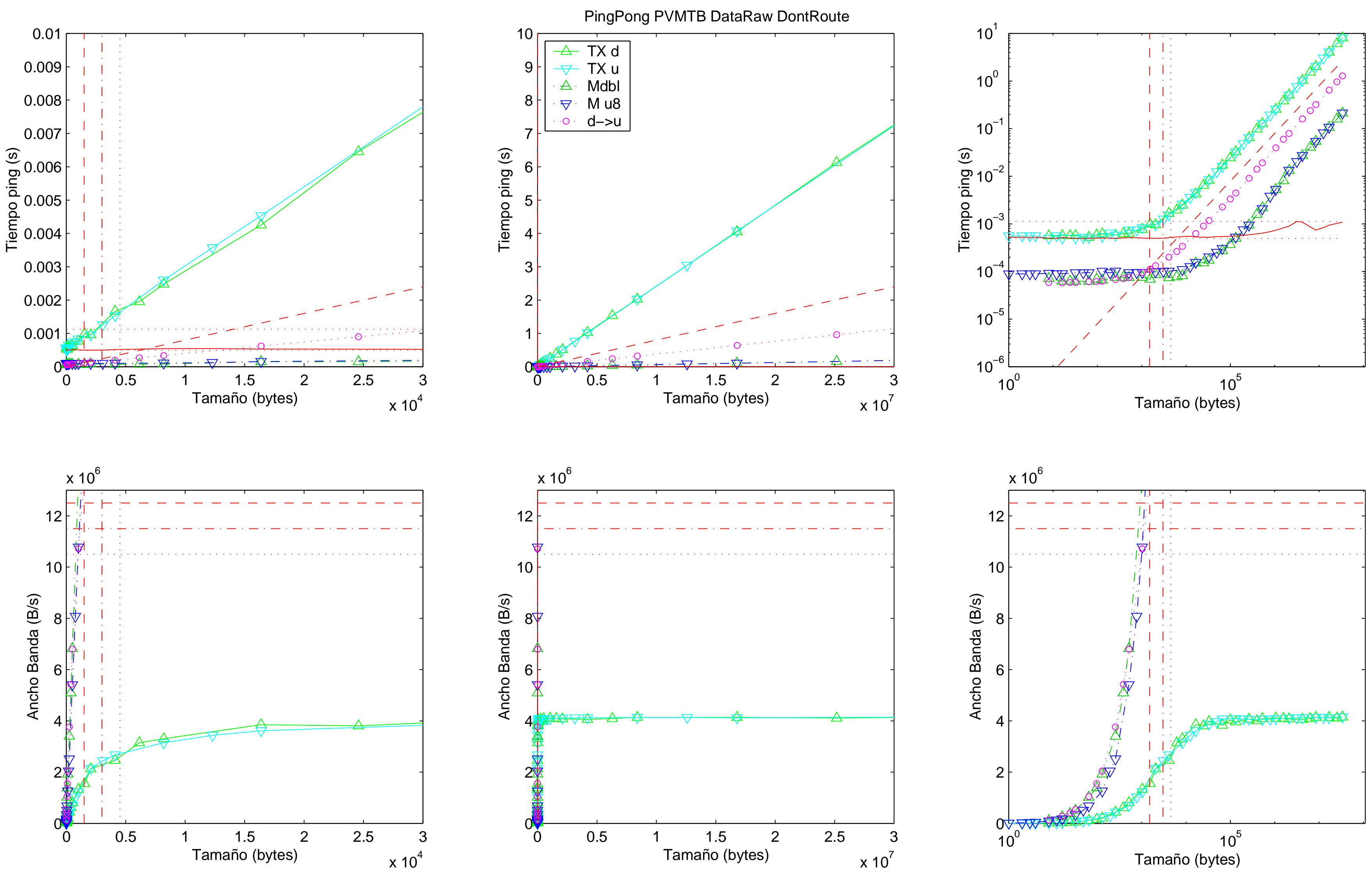


Figura 3.9: Test *ping-pong* bajo PVMTB, con opción **PvmDataRaw**. Consultar pie de Figura 3.8 para más detalles.

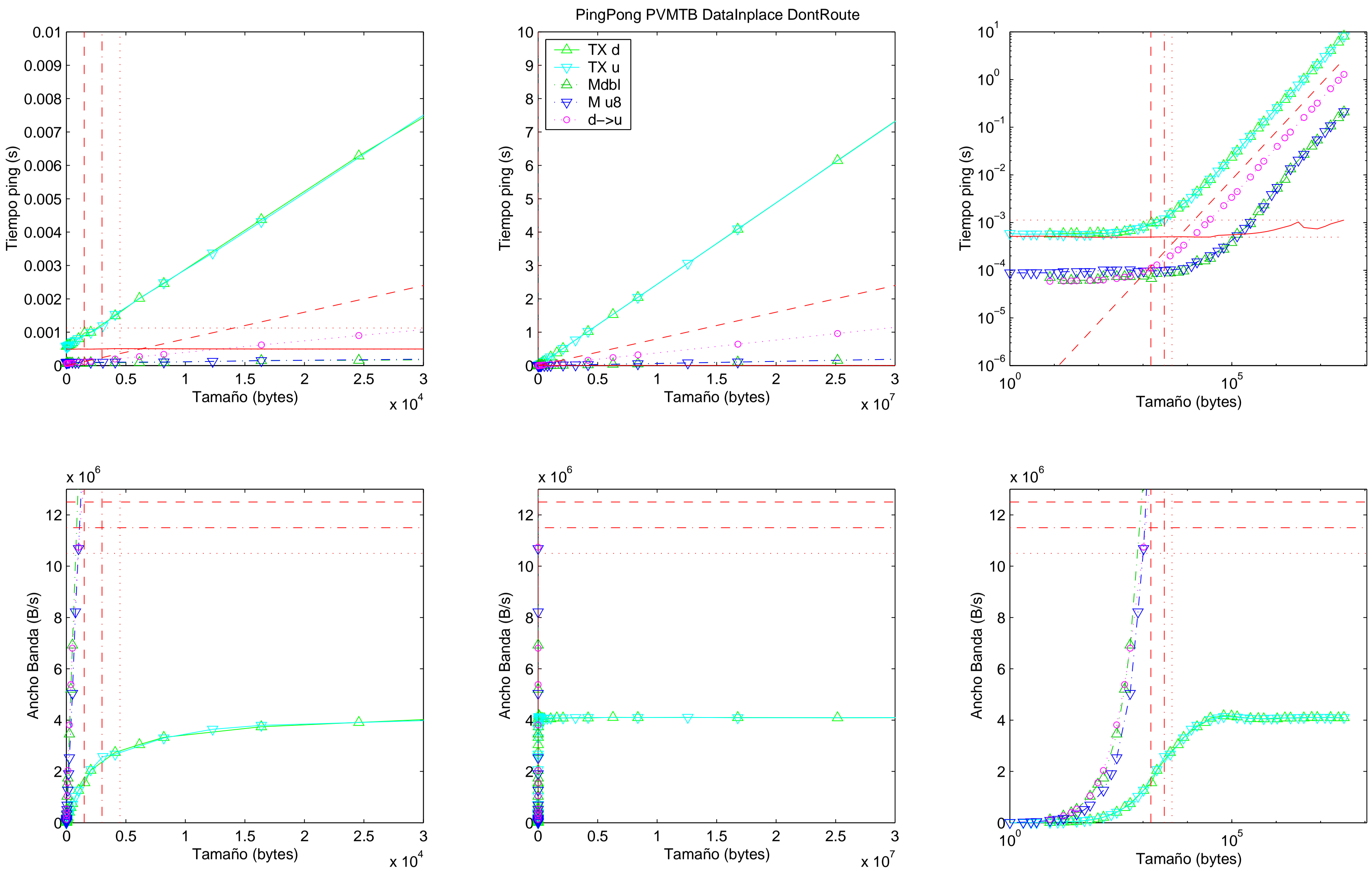


Figura 3.10: Test *ping-pong* bajo PVMTB, con opción **PvmDataInPlace**. Consultar pie de Figura 3.8 para más detalles.

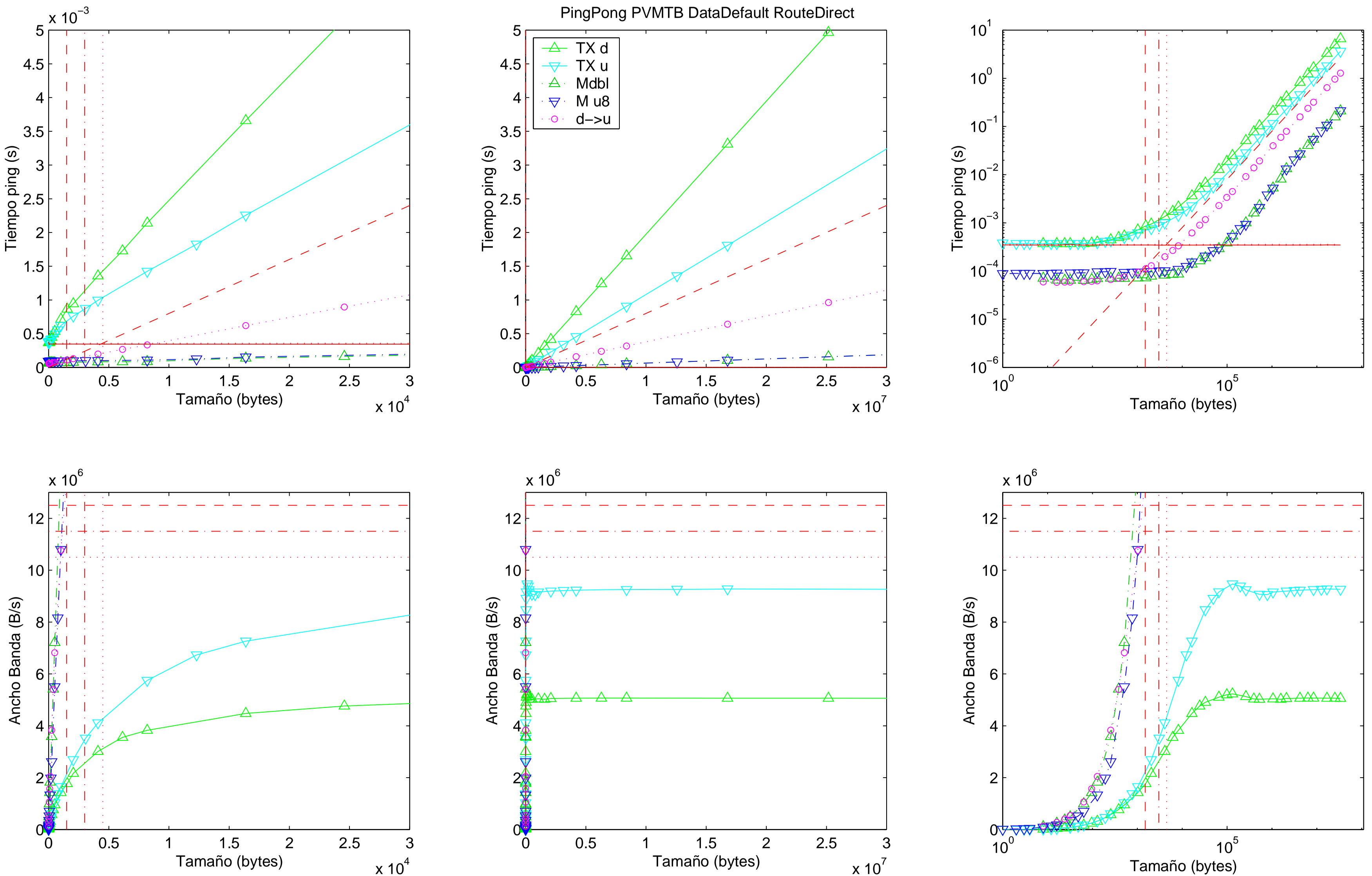


Figura 3.11: Test *ping-pong* bajo PVMTB, con opciones **PvmDataDefault** y **PvmRouteDirect**. Consultar pie de Figura 3.8 para más detalles.

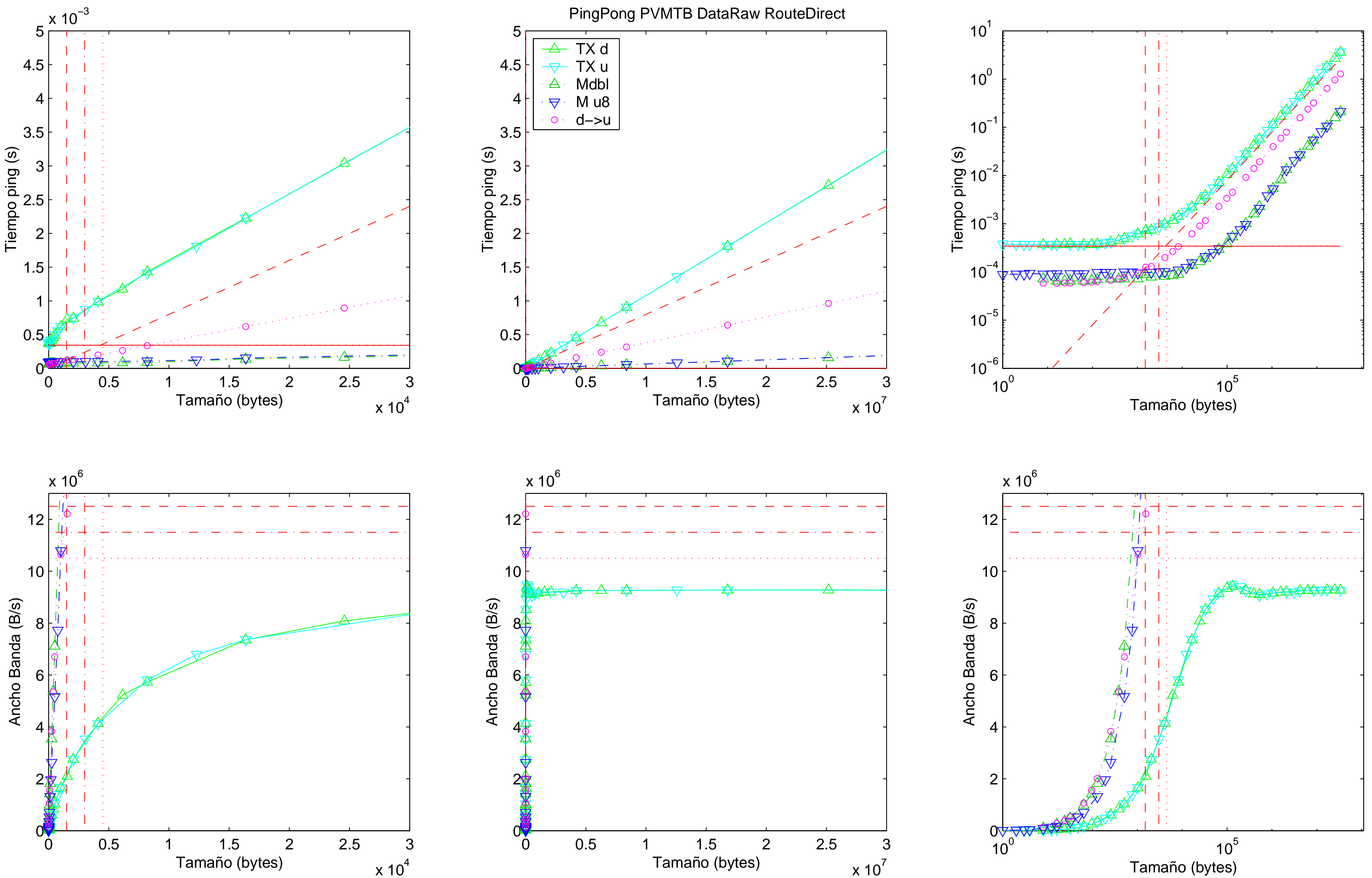


Figura 3.12: Test *ping-pong* bajo PVMTB, con opciones **PvmDataRaw** y **PvmRouteDirect**. Consultar pie de Figura 3.8 para más detalles.

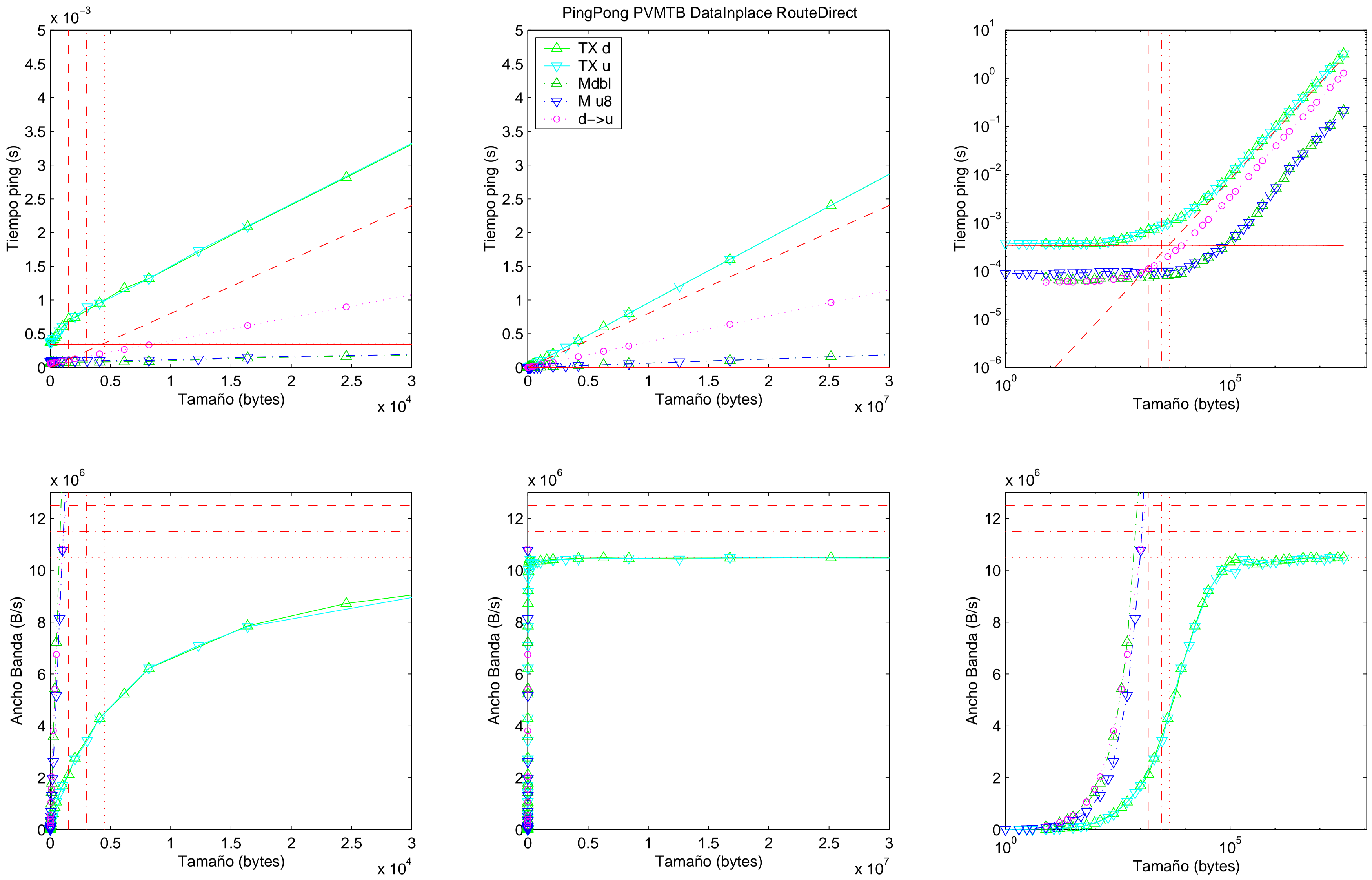


Figura 3.13: Test *ping-pong* bajo PVMTB, con opciones **PvmDataInPlace** y **PvmRouteDirect**. Consultar pie de Figura 3.8 para más detalles.

De nuevo aumenta la diferencia *uint8/double* con codificación XDR, de 3–4MB/s en la Figura 3.8 abajo derecha a 5–9.5MB/s en la Figura 3.11 con ruta directa. Con codificación *DataRaw* se pasa de 4 a 9.5MB/s en la Figura 3.12. *DataInPlace* sí presenta diferencia notable con *DataRaw* bajo ruta directa, pasando de los citados 9.5 a 10.5MB/s (Fig. 3.13). Las cantidades y conclusiones mencionadas son casi idénticas a las del apartado anterior bajo PVM, como queda resumido en la Tabla 3.3.

	BW (MB/s)		Latencia (μs)	
	uint8	**double**	media	pico
def-no	4.2– 4.1	3.1–3.0	592	1153
raw-no	4.1		615	1126
pla-no	4.1		598	1128
def-di	9.5– 9.2	5.2–5.0	347	
raw-di	9.5– 9.2		343	
pla-di	10.5–10.4		343	

Tabla 3.3: Detalles remarcables del test *ping-pong* en PVMTB.

No es aplicable sin embargo el comentario sobre el mejor comportamiento de la reserva de memoria bajo la última opción ("*pla-di*"). El gestor de memoria MATLAB presenta unos costes muy superiores al del Sistema Operativo, resultando menos afectado (relativamente) por las repetidas liberaciones de los buffers de envío y recepción.

En la Figura 3.14 se presenta la misma gráfica adicional comparando los tiempos de transmisión de datos *double* bajo las distintas combinaciones de opciones PVM. La Figura 3.14(b) muestra la ampliación de las primeras 3 MTUs, en donde se puede observar unas latencias alrededor de 0.4ms para ruta directa y de 0.6ms a través del *daemon* PVM. La diferencia entre transmitir 0 bytes y un mensaje pequeño es mayor bajo PVMTB que bajo PVM.

Más adelante, en la Tabla 3.4 derecha, se presentan algunos puntos de la mejor opción PVMTB (*DataInPlace-RouteDirect*) con datos *double*, al objeto de posibilitar una comparación objetiva con PVM.

3.5 Comparación

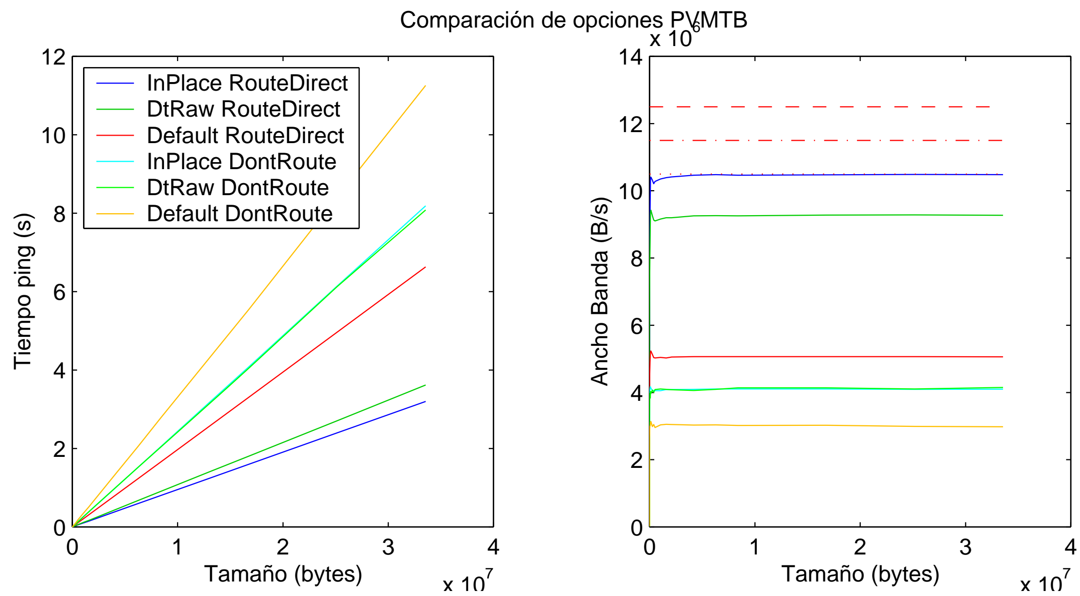
La información presentada en las gráficas anteriores se muestra ahora en forma tabulada, más apropiada para una evaluación objetiva del *overhead* introducido por PVMTB en el paso de mensajes PVM. Para realizar la presentación en una extensión aceptable, se han seleccionado sólo las mediciones para datos **double** con la combinación de opciones PvmDataInPlace/PvmRouteDirect.

La información ha sido estructurada en la Tabla 3.4 de la siguiente manera: a la izquierda se presentan los datos relativos a PVM, a la derecha las mediciones bajo PVMTB, y en el centro se muestra el tamaño de mensaje considerado. Cada fila recoge la información asociada a un tamaño de mensaje. Los tamaños se dan en orden creciente, naturalmente.

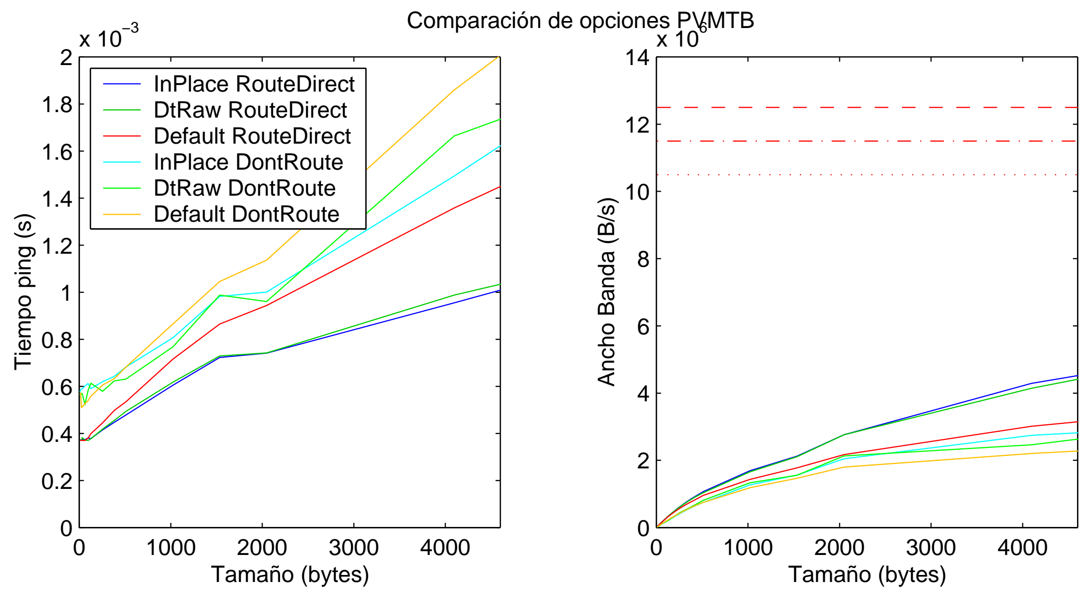
Tanto para PVM como para PVMTB, cada fila de la tabla proporciona:

S: el tamaño de mensaje considerado en la fila (bytes).

M: el tiempo requerido para ubicar un bloque de memoria del tamaño requerido (ms).



(a) Barrido completo.



(b) Detalle de las primeras 3 MTUs.

Figura 3.14: Test *ping-pong* bajo PVMTB, comparación de opciones con datos *double*.

T: el tiempo de transmisión para ese tamaño (ms), también denominado a veces tiempo de *ping*. Esta cantidad incluye también el tiempo empleado en el receptor para recibir y desempaquetar el mensaje.

B lineal: el cociente entre tamaño y tiempo de *ping* (Apartado 2.3.4). Denominaremos a esta cantidad "ancho de banda lineal" para no confundirlo con el "ancho de banda afín". Según el modelo lineal, $T = S/B$, excepto para tamaño $S = 0$ (debería ser $T_0 = 0/B = 0$) pudiéndose emplear el término "latencia" para el tiempo de *ping* del mensaje nulo, T_0.

B afín: despejando B del modelo afín $T = L + S/B$ se obtiene $B = S/(T - L)$. Naturalmente, se

M (ms)	T ping (ms)	B lineal	B afín	**S (bytes)**	M (ms)	T ping (ms)	B lineal	B afín
	0.185	(MB/s)	(MB/s)	**0**		0.343	(MB/s)	(MB/s)
0.001	0.223	0.036	0.213	**8**	0.072	0.372	0.022	0.278
0.001	0.223	0.072	0.425	**16**	0.065	0.373	0.043	0.538
0.001	0.224	0.107	0.625	**24**	0.065	0.373	0.064	0.795
0.001	0.223	0.143	0.841	**32**	0.065	0.374	0.086	1.024
0.001	0.234	0.274	1.321	**64**	0.065	0.373	0.172	2.108
0.001	0.247	0.389	1.569	**96**	0.067	0.381	0.252	2.508
0.002	0.246	0.520	2.102	**128**	0.071	0.378	0.338	3.619
0.002	0.253	1.012	3.788	**256**	0.071	0.415	0.617	3.548
0.002	0.332	1.158	2.624	**384**	0.071	0.448	0.858	3.659
0.002	0.450	1.138	1.935	**512**	0.071	0.479	1.069	3.761
0.002	0.433	2.363	4.130	**1024**	0.072	0.605	1.692	3.904
0.002	0.560	2.745	4.105	**1536**	0.072	0.723	2.125	4.041
0.003	0.594	3.449	5.013	**2048**	0.081	0.742	2.761	5.134
0.004	0.782	5.240	6.869	**4096**	0.081	0.955	4.290	6.693
0.005	0.991	6.198	7.623	**6144**	0.086	1.174	5.232	7.388
0.006	1.140	7.189	8.586	**8192**	0.089	1.317	6.220	8.409
0.015	1.927	8.505	9.410	**16384**	0.140	2.086	7.856	9.401
0.073	2.642	9.302	10.004	**24576**	0.165	2.818	8.720	9.927
0.096	3.382	9.688	10.249	**32768**	0.189	3.557	9.211	10.193
0.189	6.383	10.267	10.574	**65536**	0.293	6.574	9.969	10.517
0.281	9.560	10.283	10.486	**98304**	0.392	9.524	10.322	10.707
0.020	12.717	10.307	10.459	**131072**	0.531	12.592	10.409	10.700
0.020	25.184	10.409	10.486	**262144**	1.013	25.402	10.320	10.461
0.020	38.002	10.347	10.398	**393216**	1.579	38.520	10.208	10.300
0.020	50.964	10.287	10.325	**524288**	2.317	51.022	10.276	10.345
0.021	101.534	10.327	10.346	**1048576**	5.272	101.307	10.350	10.386
0.023	151.894	10.355	10.368	**1572864**	8.209	151.396	10.389	10.413
0.032	202.747	10.344	10.353	**2097152**	13.343	201.390	10.413	10.431
0.035	403.587	10.393	10.397	**4194304**	27.040	400.874	10.463	10.472
0.036	604.030	10.416	10.419	**6291456**	40.334	600.396	10.479	10.485
0.036	806.144	10.406	10.408	**8388608**	54.086	801.764	10.463	10.467
0.027	1611.165	10.413	10.414	**16777216**	105.988	1601.649	10.475	10.477
0.027	2413.357	10.428	10.429	**25165824**	158.438	2400.079	10.485	10.487
0.027	3218.677	10.425	10.426	**33554432**	212.356	3201.912	10.480	10.481

(a) Datos para PVM. (b) Tam (c) Datos para PVMTB.

Tabla 3.4: Resultados del test *ping-pong* para datos **double** con opciones PvmDataInPlace/PvmRouteDirect. Los datos han sido exportados desde la hoja de cálculo con tres cifras decimales.

debe cumplir que B afín > B lineal ya que $0 < L < T$.

La comparación se realiza dividiendo las columnas de tiempos MATLAB entre las correspondientes columnas C (ubicación de memoria y transmisión), para obtener una idea de cuánto más lento es MATLAB que C reservando memoria, y cuánto más lento es PVMTB que PVM transmitiendo mensajes. Cabe destacar que el primer cociente no está relacionado con PVMTB, dependiendo exclusivamente de los gestores de memoria de MATLAB y del S.O.

Con los anchos de banda se procede de forma inversa, dividiendo las columnas PVM entre las PVMTB, para obtener cantidades con la misma interpretación intuitiva. La comparación de anchos de banda lineales da por tanto el mismo resultado que la de tiempos de transmisión, al cancelarse el factor del tamaño de mensaje. Estas cantidades proporcionan un método objetivo de comparación, preferible a la simple inspección de gráficas, para evaluar el *overhead* de PVMTB. Los cocientes se muestran en la Tabla 3.5.

S (bytes)	M	T ping	B lineal	B afín
0		1.850		
8	125.736	1.667	1.667	0.766
16	78.426	1.671	1.671	0.790
24	79.322	1.667	1.667	0.786
32	77.509	1.674	1.674	0.821
64	56.989	1.596	1.596	0.627
96	58.522	1.546	1.546	0.626
128	44.200	1.536	1.536	0.581
256	40.418	1.641	1.641	1.068
384	38.716	1.350	1.350	0.717
512	37.082	1.065	1.065	0.515
1024	33.582	1.397	1.397	1.058
1536	30.850	1.292	1.292	1.016
2048	24.135	1.249	1.249	0.976
4096	18.614	1.221	1.221	1.026
6144	16.516	1.185	1.185	1.032
8192	14.602	1.156	1.156	1.021
16384	9.605	1.083	1.083	1.001
24576	2.249	1.067	1.067	1.008
32768	1.961	1.052	1.052	1.005
65536	1.549	1.030	1.030	1.005
98304	1.396	0.996	0.996	0.979
131072	27.188	0.990	0.990	0.978
262144	51.296	1.009	1.009	1.002
393216	78.541	1.014	1.014	1.010
524288	115.030	1.001	1.001	0.998
1048576	250.416	0.998	0.998	0.996
1572864	363.256	0.997	0.997	0.996
2097152	414.302	0.993	0.993	0.993
4194304	772.469	0.993	0.993	0.993
6291456	1133.055	0.994	0.994	0.994
8388608	1498.186	0.995	0.995	0.994
16777216	3897.477	0.994	0.994	0.994
25165824	5824.284	0.994	0.994	0.994
33554432	7926.393	0.995	0.995	0.995

Tabla 3.5: Cocientes PVMTB/PVM. Los anchos de banda son PVM/PVMTB.

Como se puede comprobar, la reserva de memoria es una costosa operación en MATLAB. Repasando la Tabla 3.4, se observa que la reserva de memoria bajo C presenta una cresta en decenas de KBs con un pico de 281μs en 96KB y un costo muy reducido para tamaños menores (del orden del μs), estabilizándose en unos 20–30μs para tamaños mayores. Se debe recordar que se realizan repetidas liberaciones de buffers para reducir al mínimo el posible *swapping*. Incluso bajo esta opción PVM, la liberación del buffer de recepción afecta a las prestaciones del gestor de memoria Linux. En el capítulo siguiente se observa un mejor comportamiento del gestor de memoria al no ser necesarias estas liberaciones de memoria bajo MPI.

En comparación, el tiempo de reserva de memoria bajo MATLAB cuesta como mínimo 65μs (más del doble que reservar 32MBs bajo C) y crece proporcionalmente con el tamaño del array a partir de decenas de KBs. Queda así patente el mérito de PVMTB al evitar una copia adicional de memoria, copia en la que incurren los trabajos anteriores (DP-TB y PT). Los cocientes (Tabla 3.5) para la columna etiquetada **M** toman valores realmente altos, salvo en el comentado rango de decenas de KBs donde el *overhead* del gestor de memoria MATLAB baja hasta un 40% para el pico en 96KB.

En comparación, el tiempo de transmisión tiene un comportamiento excelente. El *overhead* baja rápidamente desde 1.85 hasta 1.2 en las primeras 3MTUs, y pronto se estabiliza alrededor de la unidad en el rango de decenas de KBs, obteniéndose un *overhead* inferior al 1% a partir del centenar de KBs. El hecho de que no sea difícil obtener mediciones con *overhead* ligeramente inferior a la unidad revela que la variabilidad de las mediciones es de una magnitud superior al propio *overhead*. Se puede afirmar sin temor que PVMTB no presenta *overhead* sobre el uso directo de PVM para tamaños de mensaje por encima del centenar de KBs.

La comparación de las columnas de anchos de banda lineal y afín revela que a partir de 1MB la latencia (tiempo de *setup*, transmisión del mensaje vacío) se disimula en el tiempo de transmisión total, afectando sólo en una milésima de punto al *overhead* estabilizado alrededor de la unidad. Repasando de nuevo la Tabla 3.4, se observa que estamos hablando de 100ms de tiempo de transmisión frente a 0.2–0.3ms de latencia. Desde el centenar de KBs la diferencia entre los *overheads* de los modelos lineal y afín es menor que un 1%, siendo prácticamente indiferente escoger entre ambos salvo para tamaños menores.

Resumidamente, PVTMB no presenta *overhead* significativo sobre PVM para mensajes a partir de centenares de KBs; para decenas de KBs es inferior al 10%, en el rango de KBs crece del 10% al 40%, y para tamaños menores puede llegar al 65%. La latencia es un 85% mayor.

3.6 Conclusiones

Resumimos a continuación los detalles más significativos del trabajo presentado en este capítulo:

Complitud: Se ha implementado un interfaz completo con PVM, respetando los patrones de llamada de las rutinas PVM originales, salvo cuando esto entra en conflicto con el propio entorno MATLAB. Para tratar dicho caso se han establecido unos criterios razonados de modificación del patrón de llamada.

Otras *Toolboxes* implementan sólo un subconjunto de rutinas PVM (particularmente la biblioteca de grupos libgpmv queda siempre discriminada), o incluso optan por ofrecer una

serie de comandos de alto nivel que utilizan internamente las llamadas PVM, impidiendo al usuario MATLAB el acceso directo a ellas.

También se ha resuelto el conflicto producido por la *opacidad* de los tipos de datos MATLAB, proporcionándose un par de comandos pvm_[un]pack que soportan variables MATLAB de cualquier tipo, manteniendo así la coherencia con la filosofía MATLAB que promueve la ausencia (de hecho carece) de declaraciones para favorecer el rápido y fácil prototipado de aplicaciones. Para los tipos básicos compatibles con PVM se pueden utilizar las llamadas PVM originales pvm_[u]pkdouble, pvm_[u]pkint, etc.

Enlace dinámico: El interfaz se ha realizado sobre una versión dinámicamente enlazada de la biblioteca PVM. Se ha mostrado la forma de modificar la distribución original de PVM bajo sistemas UNIX para producir dicha versión. Se consigue con ello ahorro de espacio en disco, con el beneficio adicional de que múltiples usuarios pueden compartir el código de la biblioteca una vez cargado en memoria.

Otras *Toolboxes*, en lugar de modificar los Makefile del sistema PVM, prefieren complicar el diseño de la interfaz usando un fichero MEX “distribuidor” o “directorio” que lleva estáticamente enlazada toda la biblioteca PVM. Todos los comandos de la *Toolbox* terminan pasando el control a dicha rutina, identificando mediante un argumento adicional la llamada que desean invocar. Esto provoca un *overhead* en el tiempo de ejecución de los comandos de la *Toolbox* así como un gasto adicional de espacio en disco y memoria.

Reutilización de código: El método de sustitución de macros **#define** es particularmente apropiado para el interfaz con PVM, debido a la gran regularidad de sus patrones de llamada. Esto reduce el esfuerzo de diseño, depuración y mantenimiento de la *Toolbox*, y facilita enormemente su estudio y comprensión.

Basta comparar el Makefile o el código fuente de los ficheros MEX de esta *Toolbox* con los de cualquier *Toolbox* similar para estimar la importancia de este mecanismo de reutilización.

Rápido prototipado: Esta *Toolbox* permite desarrollar prototipos de aplicaciones de altas prestaciones (HPC, *High Performance Computing*) mediante PVM, en un entorno de rápido prototipado como es MATLAB. Esto evita la redacción de Makefiles, uso del compilador y estudio de sus modificadores (switches, flags, como –L, –shared, etc). Una vez instalada correctamente la *Toolbox*, el usuario puede olvidarse de dónde y cómo está instalado el sistema PVM, concentrándose en el desarrollo de su prototipo, pudiendo utilizar para ello todas las facilidades disponibles bajo MATLAB, como por ejemplo las herramientas de creación de GUIs, generación de gráficos y visualización 3D, así como herramientas diseñadas para facilitar la programación, depuración y temporización de programas PVM, como por ejemplo XPVM.

Hay que tener en cuenta (ver por ejemplo [14], [39]) que la mayor parte del tiempo de desarrollo de un prototipo de cierta complejidad en un lenguaje compilado se invierte en el diseño del interfaz (posiblemente gráfico) con el usuario y la E/S en general, incluyendo el manejo de ficheros y directorios, visualización de los resultados, gráficas, imágenes, etc.

Según muestra la experiencia docente, la redacción de Makefiles es la tarea que más ingrata resulta a los estudiantes de un sistema de paso de mensajes como PVM. Esta herramienta tiene pues un gran potencial para su aplicación en docencia. También resulta de interés

destacar de nuevo bajo este contexto la transparencia conseguida por la interfaz al mantener reducidas al mínimo posible las modificaciones a los patrones de las llamadas PVM originales.

Pérdida de prestaciones mínima: La *Toolbox* también tiene potencial para su aplicación en investigación, tanto por el ahorro en esfuerzo de diseño de aplicaciones HPC como por la aceptable pérdida de prestaciones que causa (comparada con el uso directo del sistema PVM bajo lenguaje C), especialmente para mensajes a partir de la decena de KBs.

Las operaciones de reserva de memoria y cambio de tipo **double↔uint8** son frecuentes en aplicaciones MATLAB típicas como procesamiento de señal e imagen, pudiendo incluso incurrir en ellas involuntariamente al programar expresiones complejas. Una despreocupada programación bajo MATLAB, ignorando el costo de las operaciones de reserva de memoria al crear variables temporales de gran tamaño, o realizando costosas operaciones de cambio de tipo, provoca mayor pérdida de prestaciones que las manifestadas en la comparación de PVMTB bajo MATLAB con PVM bajo C.

Cabe manifestar a este respecto el hecho de que tanto DP-TB como PT incurren en una copia de memoria adicional en el mecanismo global de paso de mensajes. Las variables contenidas en los mensajes recibidos se copian a zonas de memoria explícitamente reservadas por dichas *Toolboxes* mediante las llamadas API MATLAB mxCalloc() o mxCreateDoubleMatrix(), implicando por tanto al gestor de memoria MATLAB con su elevado *overhead*.

Opciones de empaquetamiento y encaminamiento: Se ha demostrado la importancia de la opción *PvmRouteDirect* en la reducción de la latencia y maximización del ancho de banda, evitando la ruta normal a través del *daemon* PVM y usando una ruta TCP directa entre tareas.

También se ha cuantificado la importancia de la opción *PvmDataInPlace*; al evitar la copia adicional de memoria, ofrece prestaciones superiores, al tiempo que puede evitar el costoso mecanismo de *swapping* asociado a arrays de tamaño comparable al de la memoria libre. Sin embargo, esta opción sólo debe usarse para comunicación entre computadores con representación de datos homogénea.

Capítulo 4

Estudio de MPITB

Resumen del capítulo

MPITB son las siglas de “MPI *Toolbox*”. Esta *Toolbox* permite utilizar el sistema LAM/MPI desde MATLAB. La estructura de este capítulo coincide con la del anterior, dedicado al estudio de PVMTB.

En el Apartado 4.1 se detallan algunos criterios de índole técnica que se han observado en el diseño de este interfaz entre MATLAB y MPI. De nuevo, ningún trabajo anterior utiliza una biblioteca MPI dinámicamente enlazada, teniendo este detalle técnico implicaciones en el consumo de disco y memoria por parte de los comandos de la *Toolbox*. La implementación LAM del estándar MPI está construida con la herramienta GNU libtool, de manera que un simple comando ./configure −−with−shared permite generar la versión compartida de las bibliotecas LAM.

El Apartado 4.2 muestra la clasificación en categorías de las llamadas MPI en función de los bloques constructivos que pueden compartir, clasificación realizada al efecto de sistematizar su interfaz. Al igual que sucedió con PVMTB, ningún trabajo previo hace un estudio parecido, lo cual les conduce a un diseño basado en una rutina “directorio”. La conjunción de esta elección de diseño con el mencionado enlace estático invariablemente utilizado por los trabajos anteriores tiene profundas implicaciones en el *overhead* introducido por la *Toolbox*. En el Apartado 4.3 se explica detalladamente la implementación de MPITB usando el citado método de clasificar previamente en categorías las llamadas MPI según los bloques constructivos compartidos. Estos bloques son tramos de código encargados de manipular los argumentos de entrada y valores de retorno. La técnica de clasificar por patrones de llamada no se puede utilizar tan provechosamente como con PVMTB, al no presentar MPI unos patrones de llamada tan repetitivos como PVM.

El Apartado 4.4 estudia la eficiencia de MPITB en un test *ping-pong*, similar al realizado en el capítulo anterior para PVMTB. Los resultados son útiles para evaluar la pérdida de prestaciones introducida por la *Toolbox*, frente al uso directo de MPI en lenguaje C. Esta comparación con MPI se realiza en el Apartado 4.5. Habiéndose realizado un estudio similar en el capítulo anterior para PVMTB, se cotejan ahora en el Apartado 4.6 los resultados de ambos estudios para comparar la relativa eficiencia de ambas *Toolboxes*.

Finalmente se resumen las conclusiones de este capítulo en el Apartado 4.7.

4.1 Elecciones de diseño

En el diseño de MPITB se mantienen los mismos criterios que para PVMTB:

Paso de mensajes explícito: Se redacta un comando MPITB para cada llamada MPI, usando el mismo nombre con vistas a la utilidad docente de la *Toolbox*. Otras *Toolboxes* basadas en MPI, como por ejemplo MultiMATLAB, ofrecen comandos de alto nivel con nombres diferentes a los del estándar, que internamente realizan varias llamadas MPI o parte de las funciones de una llamada MPI. Por ejemplo, Gridsize y Coord devuelven parte de la información recolectada con MPI_Cart_get(), mientras que Shift aglomera la funcionalidad de MPI_Cart_shift() y MPI_Sendrecv().

Los comandos MultiMATLAB se pueden consultar en la Tabla 1.4 del Apartado 1.6.2.

Implementación MEX: Un fichero MEX es la forma contemplada por MATLAB para llamar a una biblioteca C desde el entorno interpretado. Cada comando MPITB es un fichero MEX separado. Estos comandos recubren las llamadas MPI originales (son *wrappers*). No existen ficheros M intermedios con su asociado coste de interpretación.

Transparencia en la llamada: Se procura respetar al máximo el patrón de llamada MPI original. Las funciones que deben realizar los comandos MPITB son

- traducción MATLAB→C de los argumentos de entrada.
- traducción C→MATLAB de los valores de retorno.
- anotación de información de estado coherente con MPI, en los casos en que sea necesario.

Enlace dinámico: Ya que cada fichero MEX se compila separadamente, se evita un gasto excesivo de disco y memoria usando enlace dinámico con una biblioteca MPI compartida.

Reutilización de código: Los bloques constructivos se implementan como *macro*-instrucciones **#define**. Se agrupan en ficheros **#include** los bloques constructivos asociados con una determinada categoría de llamadas MPI. Como veremos, estas categorías vienen determinadas principalmente por la funcionalidad de las llamadas.

Tras haber estudiado el diseño de PVMTB, los criterios expuestos para el diseño de MPITB surgen de manera natural. Sin embargo, ningún trabajo anterior (MultiMATLAB [90, 59], PMI [52]) utiliza una biblioteca MPI dinámicamente enlazada, teniendo este detalle técnico implicaciones en el consumo de disco y memoria por parte de los comandos de la *Toolbox*.

Particularmente, PMI está pensado para su ejecución en un único computador, arrancándose los diversos procesos MATLAB mediante el uso de llamadas *engine* de la API MATLAB. Esta elección de diseño excluye su uso en clusters de computadores, siendo un computador SMP la única plataforma paralela viable para obtener *speedup* con PMI. Ya se comentó en el Apartado 1.3.1 la opinión al respecto del fabricante de MATLAB, *The MathWorks*, que ha incorporado recientemente LAPACK en el diseño de MATLAB con vistas a obtener ganancias de velocidad en computadores SMP disponiendo de una biblioteca BLAS afinada contra la cual se pueda enlazar la biblioteca LAPACK utilizada por MATLAB.

MultiMATLAB opta por un diseño mixto basado en último término en directorio. Recordemos que en este tipo de diseño un fichero MEX auxiliar se enlaza estáticamente con la biblioteca MPI, pudiéndose seleccionar la rutina MPI a invocar según un argumento de la llamada MEX. El fichero directorio en MultiMATLAB es MMAT, que contempla las llamadas MPI Init, Comm_rank, Comm_size, Send, Recv, Iprobe, Barrier, Bcast, Reduce, Cart_create, Cart_sub y Cart_get.

Sin embargo, el directorio MultiMATLAB no proporciona interfaz directo desde MATLAB a todas estas llamadas MPI. Ya se comentaron los casos de Gridsize y Coord, que devuelven parte de la información recolectada con MPI_Cart_get(), mientras que Shift aglomera la funcionalidad de MPI_Cart_shift() y MPI_Sendrecv().

El complejo diseño mixto de MultiMATLAB se revela en el variable número de pasos intermedios que presentan los distintos comandos. Así, el fichero M de algunos comandos consiste sencillamente en la llamada a directorio correspondiente, MMAT(x). Esto se verifica para Init (0), ID (1), Nproc (2), Send (3), Recv (4), Coord (7), Gridsize (8), Quit (12), Interrupt (13), Abort (14), Allreduce (15) y Barrier (16). Debe notarse que algunas de estas entradas realizan no sólo la llamada MPI deseada sino también una cantidad variable de procesamiento adicional.

Otros comandos MultiMATLAB tienen por contra procesamiento adicional en el propio fichero M. Esto sucede con Bcast, Collect, Distribute, Get, Grid, GridWindow, Probe y Start. El fichero M de algunos de estos comandos es realmente extenso, como Bcast, Start o Shift, por ejemplo.

Por último, es común en MultiMATLAB que algunos ficheros M estén programados en función de adicionales ficheros M "de uso interno", que añaden un paso de interpretación intermedio. Particularmente, se han seleccionado 4 ejemplos:

Broadcast: El fichero M Bcast.m es uno de los más extensos. Incluye llamadas reiteradas a Eval, Put, y al fichero de uso interno MMbcast.m que a su vez invoca a MMAT(9).

Reducción: Los comandos MultiMATLAB Max, Min, Sum y Prod llaman a los correspondientes comandos intermedios MMmax.m, MMmin.m, etc, quienes a su vez son pasos intermedios para invocar a Reduce.m, quien a su vez invoca a MMAT(10). Contabilizamos un total de 5 llamadas partiendo del entorno MATLAB.

Topología: El comando MultiMATLAB Grid realiza repetidas llamadas a Nproc, Put y Eval, al objeto de ejecutar en los esclavos MMgrid.m quien termina por invocar a MMAT(6).

Por su parte, Shift, el comando MultiMATLAB más extenso, realiza diversas llamadas a Gridsize, Put, Eval, Send, Recv, al fichero M interno DistributeMap, realiza una cantidad de cálculos matemáticos y movimiento (copia) de datos, y finalmente produce la evaluación remota del comando interno PerformShift, quien tras diversos cálculos adicionales invoca repetidamente a Send y Recv. Se puede comprender que este comando aparezca marcado como sólo ligeramente relacionado con las correspondientes llamadas MPI_Cart_Shift y MPI_Sendrecv en la Tabla 1.4 del Apartado 1.6.2.

Se pueden valorar ahora los méritos del diseño propuesto para MPITB en cuanto a robustez, coste de desarrollo y mantenimiento, reutilización del código, ocupación en disco y eficiencia de llamada. Los trabajos previos se basan en una "rutina directorio", elección que termina traduciéndose en una cierta (a veces significativa) pérdida de prestaciones debida a los pasos intermedios de interpretación del fichero M y selección del caso en el directorio. Los mismos comentarios realizados a propósito de PVMTB en el capítulo anterior son ahora aplicables a MPITB, incluyendo:

Reutilización: los bloques constructivos son programados y depurados una vez, pasando a formar parte del fichero **#include** correspondiente a la categoría de llamadas MPI en cuestión. Estos bloques son reutilizados múltiples veces en los ficheros MEX de la categoría, ahorrando costos de desarrollo y mantenimiento. La reutilización no introduce errores de programación nuevos, conduciendo a un diseño robusto. El código fuente es más fácil de leer, entender, depurar, documentar y explicar.

Eficiencia espacial y temporal: Los *wrappers* MEX no incluyen código enlazado estáticamente de la biblioteca MPI. Si otro usuario C o MATLAB está usando la biblioteca compartida, el uso de MPITB sólo carga en memoria los *wrappers* utilizados. Esta carga es también más rápida, dado el menor tamaño de los objetos compartidos.

Ausencia de directorio: Los *wrappers* carecen de prólogos o *stubs*, bien sean en la forma de ficheros M interpretados o como selección de rama en una sentencia **switch** o cascada **if** – **else if** de un directorio. La llamada MPI se invoca inmediatamente después de obtener los argumentos de entrada, y los valores de retorno son inmediatamente devueltos al entorno interpretado MATLAB.

La superioridad en prestaciones del diseño queda manifiesta en el estudio comparativo realizado en el Capítulo 1.

4.2 Categorías de llamada MPI

Como se mencionó previamente, los argumentos de las llamadas MPI se presentan en agrupaciones sistemáticas relacionadas con la funcionalidad de la llamada, aunque no presentan el grado de repetición encontrado en el repertorio de llamadas PVM. Esta repetición sistemática de grupos de argumentos invita a realizar un interfaz que respete escrupulosamente el tipo y número de argumentos de las llamadas MPI originales.

Ningún trabajo previo hace un estudio parecido, lo cual les conduce a un diseño basado en una rutina “directorio”. Los mismos comentarios realizados en este sentido para PVMTB son ahora aplicables a MPITB, con el agravante de que el repertorio de llamadas MPI es auténticamente masivo. MPITB dispone de 153 comandos, contemplando casi en su totalidad el estándar MPI 1.2 y parte del MPI 2.0.

Enlazar estáticamente 153 comandos está directamente fuera de lugar. El diseño ha de contemplar forzosamente alguna forma de reutilización de código, no sorprendiendo el bajo número de llamadas MPI recubiertas en los trabajos anteriores: 20 en MultiMATLAB, 8 en PMI. Con un diseño basado en directorio, contemplar 153 llamadas hace el coste de depuración prohibitivo. El coste de mantenimiento sólo habría que considerarlo si se llega a superar la etapa de depuración.

De vuelta a la discusión del diseño de MPITB, motivos didácticos y de eficiencia respaldan, al igual que para PVMTB, el criterio de respetar escrupulosamente los patrones de llamada de las rutinas MPI, con el argumento adicional de que MPI es un *estándar* que ha sido *formalmente especificado* (razones #9 y #10 en la lista [89]; también conviene leer la referencia [26]). Pero de nuevo algunos patrones han sido alterados, por los mismos motivos enumerados para PVMTB. Al haberse seguido exactamente los mismos criterios, se prefiere referir al lector al Apartado 3.2 en lugar de reiterarlos aquí de nuevo.

Tras realizar los cambios según dichos criterios, los resultantes patrones de llamada de los comandos MPITB se han clasificado en categorías según el conjunto de bloques constructivos que pueden compartir, al objeto de reutilizar código y simplificar la implementación y estudio de la *Toolbox*. La Tabla 4.1 enumera las llamadas MPI originales, junto con los comandos MPITB correspondientes y el nombre dado a la categoría a la cual corresponden (y al fichero **#include** que reutilizan). Esta nomenclatura de categorías se utilizará en el siguiente apartado para estructurar la explicación de la *Toolbox*.

4.3 Discusión y detalles

El interfaz procura ser lo más transparente y sistemático posible. Algunos bloques constructivos son compartidos por varias categorías de comandos MPITB, por lo cual se han reunido en un fichero **#include** general (hMPI.h). Más que ser una categoría, hMPI reúne las rutinas de más alta reutilización, siendo compartidas por varias categorías de comandos MPITB.

Cada grupo de comandos MPITB que, debido principalmente a su funcionalidad, comparten en exclusiva un conjunto de bloques constructivos, son considerados como una "categoría" MPITB y se crea para ellos un nuevo fichero **#include** en el que se redactan dichos bloques para su posterior reutilización. Las categorías a las que ha conducido el estudio previo de las llamadas MPI son las siguientes:

MPI: Bloques MEX_HDR, FUNCALL, etc. Rutinas API para tratamiento bajo MATLAB de los tipos MPI_Comm, MPI_Request, MPI_Status, etc.

SndRecv: Comunicación punto a punto: bloques SEND, RECV, SNDRECV, etc.

TstWait: Rutinas API para el tipo MPI_Request, bloques para múltiples MPI_Stat y MPI_Request, patrones WAIT, TEST, START.

Topo: Topología: rutinas API y bloques para arrays de enteros.

Prob: Patrones para llamadas MPI_[I]Probe.

Info: Procesos dinámicos: rutinas API para tipo MPI_Info, patrones COMM_F_PORT, etc.

Err: Errores: rutinas API para gestores MPI_Errhandler.

Attr: Manejo de Atributos: *wrappers* de copia y borrado, rutinas API para manejar los *wrappers* como argumentos MATLAB.

Coll: Operaciones colectivas: rutinas API para tipo MPI_Op, bloques para el tratamiento de múltiples tripletes (buff,type,size), para el doblete (root,comm), etc.

Grp: Grupos: rutinas API para tipo MPI_Group, bloques para dobletes de grupos y múltiples rangos, patrones INT_F_COM, GRP_F_2GRP, etc.

La Tabla 4.1 sigue esta nomenclatura. En este apartado se describen la categoría general MPI y la categoría punto a punto SndRecv, dado que su explicación es breve y permiten obtener una idea detallada de cómo se programó la *Toolbox*. Habida cuenta del tamaño de MPITB, 153

Tabla 4.1: Llamadas MPI bajo MATLAB.

CAT.	MPITB	MPI
		Comunicación punto a punto
		Modos
SndRecv	info = MPI_Send (buf, dest, tag, comm)	**int** MPI_Send(**void** *buf, **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm)
SndRecv	info = MPI_Bsend (buf, dest, tag, comm)	**int** MPI_Bsend(**void** *buf, **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm)
SndRecv	info = MPI_Ssend (buf, dest, tag, comm)	**int** MPI_Ssend(**void** *buf, **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm)
SndRecv	info = MPI_Rsend (buf, dest, tag, comm)	**int** MPI_Rsend(**void** *buf, **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm)
SndRecv	[info stat] = MPI_Recv (buf, src, tag, comm)	**int** MPI_Recv(**void** *buf, **int** count , MPI_Datatype datatype , **int** source , **int** tag , MPI_Comm comm, MPI_Status * status)
MPI	info = MPI_Buffer_attach (buf)	**int** MPI_Buffer_attach (**void** * buffer , **int** size)
MPI	info = MPI_Buffer_detach	**int** MPI_Buffer_detach(**void** * bufferptr , **int** * size)
SndRecv	[info stat] = MPI_Sendrecv (sbuf, dest, stag, rbuf, src , rtag, comm)	**int** MPI_Sendrecv(**void** *sendbuf, **int** sendcount , MPI_Datatype sendtype, **int** dest , **int** sendtag , **void** * recvbuf , **int** recvcount , MPI_Datatype recvtype , **int** source , **int** recvtag , MPI_Comm comm, MPI_Status *status)
SndRecv	[info stat] = MPI_Sendrecv_replace (buf, dest, stag, src , rtag, comm)	**int** MPI_Sendrecv_replace(**void** *buf , **int** count , MPI_Datatype datatype , **int** dest , **int** sendtag , **int** source , **int** recvtag , MPI_Comm comm, MPI_Status *status)
MPI	info = MPI_Pack (var, buf, pos, comm)	**int** MPI_Pack (**void** *inbuf , **int** incount , MPI_Datatype datatype, **void** * outbuf , **int** outcount , **int** * position , MPI_Comm comm)
MPI	[info retnam] = MPI_Unpack (buf, pos, comm [,'vname']...)	**int** MPI_Unpack (**void** *inbuf , **int** insize , **int** * position , **void** * outbuf , **int** outcount , MPI_Datatype datatype , MPI_Comm comm)
MPI	[info size] = MPI_Pack_size (var, comm)	**int** MPI_Pack_size (**int** incount , MPI_Datatype datatype , MPI_Comm comm, **int** *size)
MPI	[info count] = MPI_Get_count (stat)	**int** MPI_Get_count(MPI_Status *status , MPI_Datatype datatype , **int** *count)
MPI	[info elems] = MPI_Get_elements (stat, type)	**int** MPI_Get_elements (MPI_Status * status , MPI_Datatype datatype , **int** *elements)
		Comunicación no bloqueante
SndRecv	[info req] = MPI_Isend (buf, dest, tag, comm)	**int** MPI_Isend(**void** *buf , **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Ibsend (buf, dest, tag, comm)	**int** MPI_Ibsend(**void** *buf , **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm, MPI_Request *request)
SndRecv	info = MPI_Issend (buf, dest, tag, comm)	**int** MPI_Issend(**void** *buf , **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Irsend (buf, dest, tag, comm)	**int** MPI_Irsend(**void** *buf , **int** count , MPI_Datatype datatype , **int** dest , **int** tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Irecv (buf, src, tag, comm)	**int** MPI_Irecv(**void** *buf , **int** count , MPI_Datatype datatype , **int** source , **int** tag , MPI_Comm comm, MPI_Request *request)

Tabla 4.1: Llamadas MPI bajo MATLAB (continúa).

CAT.	MPITB	MPI
TstWait	[info stat] = MPI_Wait (req)	int MPI_Wait (MPI_Request *request , MPI_Status *status)
TstWait	[info idx stat] = MPI_Waitany (req [,req]...)	int MPI_Waitany(int count , MPI_Request array_of_requests [], int *index , MPI_Status *status)
TstWait	[info stats] = MPI_Waitall (req [,req]...)	int MPI_Waitall(int count , MPI_Request array_of_requests [], MPI_Status array_of_statuses [])
TstWait	[info idxs stats] = MPI_Waitsome (req [,req]...)	int MPI_Waitsome(int incount , MPI_Request array_of_requests [], int *outcount , int array_of_indices [], MPI_Status array_of_statuses [])
TstWait	[info flag stat] = MPI_Test (req)	int MPI_Test (MPI_Request *request , int *flag , MPI_Status *status)
TstWait	[info idx flag stat] = MPI_Testany (req [,req]...)	int MPI_Testany(int count , MPI_Request array_of_requests [], int *index , int *flag , MPI_Status *status)
TstWait	[info flag stats] = MPI_Testall (req [,req]...)	int MPI_Testall (int count , MPI_Request array_of_requests [], int *flag , MPI_Status array_of_statuses [])
TstWait	[info idxs stats] = MPI_Testsome (req [,req]...)	int MPI_Testsome(int incount , MPI_Request array_of_requests [], int *outcount , int array_of_indices [], MPI_Status array_of_statuses [])
		Cancelación
Prob	[info stat] = MPI_Probe (src, tag, comm)	int MPI_Probe(int source , int tag , MPI_Comm comm, MPI_Status *status)
Prob	[info flag stat] = MPI_Iprobe (src, tag, comm)	int MPI_Iprobe(int source , int tag , MPI_Comm comm, int *flag, MPI_Status *status)
TstWait	info = MPI_Cancel (req)	int MPI_Cancel(MPI_Request *request)
TstWait	[info flag stat] = MPI_Test_cancelled (stat)	int MPI_Test_cancelled (MPI_Status *status , int *flag)
		Comunicación Persistente
SndRecv	[info req] = MPI_Send_init (buf, dest, tag, comm)	int MPI_Send_init(void *buf , int count , MPI_Datatype datatype , int dest , int tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Bsend_init (buf, dest, tag, comm)	int MPI_Bsend_init(void *buf , int count , MPI_Datatype datatype , int dest , int tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Ssend_init (buf, dest, tag, comm)	int MPI_Ssend_init(void *buf , int count , MPI_Datatype datatype , int dest , int tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Rsend_init (buf, dest, tag, comm)	int MPI_Rsend_init(void *buf , int count , MPI_Datatype datatype , int dest , int tag , MPI_Comm comm, MPI_Request *request)
SndRecv	[info req] = MPI_Recv_init (buf, src, tag, comm)	int MPI_Recv_init(void *buf , int count , MPI_Datatype datatype , int source , int tag , MPI_Comm comm, MPI_Request *request)
TstWait	info = MPI_Start (req)	int MPI_Start(MPI_Request *request)
TstWait	info = MPI_Startall (req [,req]...)	int MPI_Startall (int count , MPI_Request array_of_requests [])
TstWait	info = MPI_Request_free (req)	int MPI_Request_free(MPI_Request *request)
		Comunicación Colectiva
Grp	info = MPI_Barrier(comm)	int MPI_Barrier (MPI_Comm comm)
Coll	info = MPI_Bcast (var, root, comm)	int MPI_Bcast (void *buffer , int count , MPI_Datatype datatype , int root , MPI_Comm comm)
Coll	info = MPI_Gather (svar, rvar, root, comm)	int MPI_Gather (void *sendbuf , int sendcnt , MPI_Datatype sendtype, void *recvbuf , int recvcount , MPI_Datatype recvtype , int root , MPI_Comm comm)
Coll	info = MPI_Gatherv (svar, rvar, cnts, disps, root, comm)	int MPI_Gatherv (void *sendbuf , int sendcnt , MPI_Datatype sendtype, void *recvbuf , int *recvcnts , int *displs , MPI_Datatype recvtype , int root , MPI_Comm comm)

Tabla 4.1: Llamadas MPI bajo MATLAB (continúa).

CAT.	MPITB	MPI
Coll	info = MPI_Scatter (svar, rvar, root, comm)	**int** MPI_Scatter (**void** *sendbuf , **int** sendcnt , MPI_Datatype sendtype, **void** *recvbuf , **int** recvcnt , MPI_Datatype recvtype , **int** root , MPI_Comm comm)
Coll	info = MPI_Scatterv (svar, cnts, disps, rvar, root, comm)	**int** MPI_Scatterv (**void** *sendbuf , **int** *sendcnts , **int** * displs , MPI_Datatype sendtype, **void** *recvbuf , **int** recvcnt , MPI_Datatype recvtype , **int** root , MPI_Comm comm)
Coll	info = MPI_Allgather (svar, rvar, comm)	**int** MPI_Allgather (**void** *sendbuf , **int** sendcount , MPI_Datatype sendtype, **void** *recvbuf , **int** recvcount , MPI_Datatype recvtype , MPI_Comm comm)
Coll	info = MPI_Allgatherv (svar, rvar, cnts, disps, comm)	**int** MPI_Allgatherv (**void** *sendbuf , **int** sendcount , MPI_Datatype sendtype, **void** *recvbuf , **int** *recvcounts , **int** * displs , MPI_Datatype recvtype , MPI_Comm comm)
Coll	info = MPI_Alltoall (svar, rvar, srcnt, comm)	**int** MPI_Alltoall (**void** *sendbuf , **int** sendcount , MPI_Datatype sendtype, **void** *recvbuf , **int** recvcnt , MPI_Datatype recvtype , MPI_Comm comm)
Coll	info = MPI_Alltoallv (svar,scnts,sdisps, rvar,rcnts,rdisps, comm)	**int** MPI_Alltoallv (**void** *sendbuf , **int** *sendcnts , **int** * sdispls , MPI_Datatype sendtype, **void** *recvbuf , **int** * recvcnts , **int** * rdispls , MPI_Datatype recvtype , MPI_Comm comm)
Coll	info = MPI_Reduce (svar, rvar, op, root, comm)	**int** MPI_Reduce (**void** *sendbuf , **void** *recvbuf , **int** count , MPI_Datatype datatype, MPI_Op op, **int** root , MPI_Comm comm)
Coll	info = MPI_Allreduce (svar, rvar, op, comm)	**int** MPI_Allreduce (**void** *sendbuf , **void** *recvbuf , **int** count , MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
Coll	info = MPI_Reduce_scatter (svar, rvar, cnts, op, comm)	**int** MPI_Reduce_scatter (**void** *sendbuf , **void** *recvbuf , **int** * recvcnts , MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
Coll	info = MPI_Scan (svar, rvar, op, comm)	**int** MPI_Scan (**void** *sendbuf , **void** *recvbuf , **int** count , MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)
		Comunicadores, Grupos y Contextos
		Grupos
Grp	[info size] = MPI_Group_size (grp)	**int** MPI_Group_size (MPI_Group group, **int** *size)
Grp	[info rank] = MPI_Group_rank (grp)	**int** MPI_Group_rank (MPI_Group group, **int** *rank)
Grp	[info ranks2] = MPI_Group_translate_ranks (grp1, ranks1, grp2)	**int** MPI_Group_translate_ranks (MPI_Group group_a, **int** n , **int** *ranks_a , MPI_Group group_b, **int** *ranks_b)
Grp	[info result] = MPI_Group_compare (grp1, grp2)	**int** MPI_Group_compare (MPI_Group group1, MPI_Group group2, **int** *result)
Grp	[info grp] = MPI_Comm_group (comm)	**int** MPI_Comm_group (MPI_Comm comm, MPI_Group *group)
Grp	[info grp] = MPI_Group_union (grp1, grp2)	**int** MPI_Group_union (MPI_Group group1, MPI_Group group2, MPI_Group *group_out)
Grp	[info grp] = MPI_Group_intersection (grp1, grp2)	**int** MPI_Group_intersection (MPI_Group group1, MPI_Group group2, MPI_Group *group_out)
Grp	[info grp] = MPI_Group_difference (grp1, grp2)	**int** MPI_Group_difference (MPI_Group group1, MPI_Group group2, MPI_Group *group_out)
Grp	[info grp2] = MPI_Group_incl (grp, ranks)	**int** MPI_Group_incl (MPI_Group group, **int** n , **int** *ranks , MPI_Group *group_out)
Grp	[info grp2] = MPI_Group_excl (grp, ranks)	**int** MPI_Group_excl (MPI_Group group, **int** n , **int** *ranks , MPI_Group *newgroup)
Grp	[info grp2] = MPI_Group_range_incl (grp, ranges)	**int** MPI_Group_range_incl (MPI_Group group, **int** n , **int** ranges [][3], MPI_Group *newgroup)
Grp	[info grp2] = MPI_Group_range_excl (grp, ranges)	**int** MPI_Group_range_excl (MPI_Group group, **int** n , **int** ranges [][3], MPI_Group *newgroup)
Grp	info = MPI_Group_free (grp)	**int** MPI_Group_free (MPI_Group *group)

Tabla 4.1: Llamadas MPI bajo MATLAB (continúa).

CAT.	MPITB	MPI
		Comunicadores
Grp	[info size] = MPI_Comm_size (comm)	**int** MPI_Comm_size (MPI_Comm comm, **int** *size)
Grp	[info rank] = MPI_Comm_rank (comm)	**int** MPI_Comm_rank (MPI_Comm comm, **int** *rank)
MPI	[info result] = MPI_Comm_compare (comm1, comm2)	**int** MPI_Comm_compare (MPI_Comm comm1, MPI_Comm comm2, **int** *result)
Grp	[info comm2] = MPI_Comm_dup (comm1)	**int** MPI_Comm_dup (MPI_Comm comm, MPI_Comm *comm_out)
Grp	[info comm2] = MPI_Comm_create (comm1, grp)	**int** MPI_Comm_create (MPI_Comm comm, MPI_Group group, MPI_Comm *comm_out)
MPI	[info comm2] = MPI_Comm_split (comm1, color, key)	**int** MPI_Comm_create (MPI_Comm comm, MPI_Group group, MPI_Comm *comm_out)
Grp	info = MPI_Comm_free (comm)	**int** MPI_Comm_free (MPI_Comm *commp)
Grp	[info flag] = MPI_Comm_test_inter (comm)	**int** MPI_Comm_test_inter (MPI_Comm comm, **int** *flag)
Grp	[info size] = MPI_Comm_remote_size (comm)	**int** MPI_Comm_remote_size (MPI_Comm comm, **int** *size)
Grp	[info grp] = MPI_Comm_remote_group (comm)	**int** MPI_Comm_remote_group (MPI_Comm comm, MPI_Group *group)
MPI	[info comm] = MPI_Intercomm_create (l_comm, l_leader, p_comm, r_leader, tag)	**int** MPI_Intercomm_create (MPI_Comm local_comm, **int** local_leader, MPI_Comm peer_comm, **int** remote_leader, **int** tag , MPI_Comm *comm_out)
MPI	[info intracomm] = MPI_Intercomm_merge (intercom, high)	**int** MPI_Intercomm_merge (MPI_Comm comm, **int** high, MPI_Comm *comm_out)
		Cache de Atributos
MPI	[info kv] = MPI_Keyval_create (copy_fn, del_fn, extra_st)	**int** MPI_Keyval_create (MPI_Copy_function *copy_fn, MPI_Delete_function *delete_fn , **int** *keyval , **void** * extra_state)
MPI	info = MPI_Keyval_free (kv)	**int** MPI_Keyval_free (**int** *keyval)
MPI	info = MPI_Attr_put (comm, kv, attr)	**int** MPI_Attr_put (MPI_Comm comm, **int** keyval, **void** *attr_value)
MPI	[info attr flag] = MPI_Attr_get (comm, kv)	**int** MPI_Attr_get (MPI_Comm comm, **int** keyval, **void** *attr_value , **int** * flag)
MPI	info = MPI_Attr_delete (comm, kv)	**int** MPI_Attr_delete (MPI_Comm comm, **int** keyval)
		Topologías de Procesos
Topo	[info comm_cart] = MPI_Cart_create (comm, dims, periods, reorder)	**int** MPI_Cart_create (MPI_Comm comm_old, **int** ndims, **int** *dims, **int** *periods , **int** reorder , MPI_Comm *comm_cart)
Topo	[info dims] = MPI_Dims_create (nnodes, ndims)	**int** MPI_Dims_create(**int** nnodes , **int** ndims , **int** *dims)
Topo	[info comm_graph] = MPI_Graph_create (comm, index, edges, reorder)	**int** MPI_Graph_create (MPI_Comm comm_old, **int** nnodes, **int** *index, **int** *edges , **int** reorder , MPI_Comm *comm_graph)
MPI	[info topo] = MPI_Topo_test (comm)	**int** MPI_Topo_test (MPI_Comm comm, **int** *top_type)
MPI	[info nnodes nedges] = MPI_Graphdims_get (comm)	**int** MPI_Graphdims_get (MPI_Comm comm, **int** *nnodes, **int** *nedges)
Topo	[info index edges] = MPI_Graph_get (comm)	**int** MPI_Graph_get (MPI_Comm comm, **int** maxindex, **int** maxedges, **int** *index, **int** *edges)
MPI	[info nneighs] = MPI_Graph_neighbors_count (comm, rank)	**int** MPI_Graph_neighbors_count (MPI_Comm comm, **int** rank, **int** *nneighbors)
Topo	[info neighs] = MPI_Graph_neighbors (comm, rank)	**int** MPI_Graph_neighbors (MPI_Comm comm, **int** rank, **int** maxneighbors, **int** *neighbors)

Tabla 4.1: Llamadas MPI bajo MATLAB (continúa).

CAT.	MPITB	MPI
MPI	[info ndims] = MPI_Cartdim_get (comm)	**int** MPI_Cartdim_get (MPI_Comm comm, **int** *ndims)
Topo	[info dims periods coords] = MPI_Cart_get (comm)	**int** MPI_Cart_get (MPI_Comm comm, **int** maxdims, **int** *dims, **int** *periods , **int** *coords)
Topo	[info rank] = MPI_Cart_rank (comm, coords)	**int** MPI_Cart_rank (MPI_Comm comm, **int** *coords, **int** *rank)
Topo	[info coords] = MPI_Cart_coords (comm, rank)	**int** MPI_Cart_coords (MPI_Comm comm, **int** rank, **int** maxdims, **int** *coords)
MPI	[info rsrc rdst] = MPI_Cart_shift (comm, dir, disp)	**int** MPI_Cart_shift (MPI_Comm comm, **int** direction, **int** displ , **int** *source , **int** *dest)
Topo	[info comm_sub] = MPI_Cart_sub (comm, dims)	**int** MPI_Cart_sub (MPI_Comm comm, **int** *remain_dims, MPI_Comm *comm_new)
Topo	[info nrank] = MPI_Cart_map (comm, dims, periods)	**int** MPI_Cart_map (MPI_Comm comm_old, **int** ndims, **int** *dims, **int** *periods , **int** *newrank)
Topo	[info nrank] = MPI_Graph_map (comm, index, edges)	**int** MPI_Graph_map (MPI_Comm comm_old, **int** nnodes, **int** *index, **int** *edges, **int** *newrank)
		Gestión del Entorno
		Entorno
MPI	[info name] = MPI_Get_processor_name	**int** MPI_Get_processor_name(**char** *name, **int** * resultlen)
		Errores
Err	[info eh] = MPI_Errhandler_create ('MatlabCMD')	**int** MPI_Errhandler_create (MPI_Handler_function *function , MPI_Errhandler *errhandler)
Err	info = MPI_Errhandler_set (comm, eh)	**int** MPI_Errhandler_set (MPI_Comm comm, MPI_Errhandler errhandler)
Err	[info eh] = MPI_Errhandler_get (comm)	**int** MPI_Errhandler_get (MPI_Comm comm, MPI_Errhandler *errhandler)
Err	info = MPI_Errhandler_free (eh)	**int** MPI_Errhandler_free (MPI_Errhandler *errhandler)
MPI	[info msg] = MPI_Error_string (errcode)	**int** MPI_Error_string (**int** errorcode , **char** * string , **int** * resultlen)
MPI	[info class] = MPI_Error_class (errcode)	**int** MPI_Error_class (**int** errorcode , **int** * errorclass)
		Miscelánea
MPI	info = MPI_Init [('arg' [, 'arg']...)]	**int** MPI_Init(**int** *argc , **char** ***argv)
MPI	[info flag] = MPI_Initialized	**int** MPI_Initialized (**int** * flag)
MPI	info = MPI_Finalize	**int** MPI_Finalize()
MPI	info = MPI_Abort (comm, errcode)	**int** MPI_Abort(MPI_Comm comm, **int** errorcode)
MPI	secs = MPI_Wtime	**double** MPI_Wtime()
MPI	res = MPI_Wtick	**double** MPI_Wtick()
MPI	[info addr] = MPI_Address (var)	**int** MPI_Address(**void** *location , MPI_Aint *address)
MPI	[info major minor] = MPI_Get_version	**int** MPI_Get_version(**int** * version , **int** * subversion)
		Llamadas MPI 2.0 contempladas
		Información
Info	[info inf] = MPI_Info_create	**int** MPI_Info_create(MPI_Info *info)
Info	info = MPI_Info_delete (inf, key)	**int** MPI_Info_delete(MPI_Info info , **char** *key)
Info	info = MPI_Info_set (inf, key, val)	**int** MPI_Info_set(MPI_Info info , **char** *key , **char** *value)
Info	[info flag val] = MPI_Info_get (inf, key)	**int** MPI_Info_get(MPI_Info info , **char** *key , **int** valuelen , **char** *value , **int** * flag)

Tabla 4.1: Llamadas MPI bajo MATLAB (continúa).

CAT.	MPITB	MPI
Info	[info len flag] = MPI_Info_get_valuelen (inf, key)	**int** MPI_Info_get_valuelen(MPI_Info info , **char** *key , **int** * valuelen , **int** * flag)
Info	[info nkeys] = MPI_Info_get_nkeys (inf)	**int** MPI_Info_get_nkeys(MPI_Info info, **int** *nkeys)
Info	[info key] = MPI_Info_get_nthkey (inf, n)	**int** MPI_Info_get_nthkey(MPI_Info info , **int** n , **char** *key)
Info	[info newinf] = MPI_Info_dup (inf)	**int** MPI_Info_dup(MPI_Info info, MPI_Info *newinfo)
Info	info = MPI_Info_free (inf)	**int** MPI_Info_free(MPI_Info *info)
		Procesos Dinámicos
Info	[info children errs] = MPI_Comm_spawn (prog, args, nproc, inf, root, parents)	**int** MPI_Comm_spawn(**char** *command, **char** *argv[], **int** maxprocs, MPI_Info info, **int** root , MPI_Comm comm, MPI_Comm *intercomm, **int** array_of_errcodes[])
MPI	[info parents] = MPI_Comm_get_parent	**int** MPI_Comm_get_parent (MPI_Comm *parent)
Info	[info children errs] = MPI_Comm_spawn_multiple (cprg,carg, cnprc,cinf, root, parents)	**int** MPI_Comm_spawn_multiple(**int** count, **char** *array_of_commands[], **char** **array_of_argv[], **int** array_of_maxprocs [], MPI_Info array_of_info [], **int** root , MPI_Comm comm, MPI_Comm *intercomm, **int** array_of_errcodes[])
Info	[info port] = MPI_Open_port (inf)	**int** MPI_Open_port (MPI_Info info , **char** *port_name)
MPI	info = MPI_Close_port (port)	**int** MPI_Close_port (**char** *port_name)
Info	[info clcomm] = MPI_Comm_accept (port,inf,root,comm)	**int** MPI_Comm_accept (**char***port_name,MPI_Info info,**int** root,MPI_Comm comm,MPI_Comm*newcomm)
Info	[info svcomm] = MPI_Comm_connect(port,inf,root,comm)	**int** MPI_Comm_connect(**char***port_name,MPI_Info info,**int** root,MPI_Comm comm,MPI_Comm*newcomm)
Info	info = MPI_Publish_name (svc, inf, port)	**int** MPI_Publish_name (**char** *service_name , MPI_Info info , **char** *port_name)
Info	info = MPI_Unpublish_name (svc, inf, port)	**int** MPI_Unpublish_name (**char** *service_name, MPI_Info info , **char** *port_name)
Info	[info port] = MPI_Lookup_name (svc, inf)	**int** MPI_Lookup_name (**char** *service_name, MPI_Info info, **char** *port_name)
Grp	info = MPI_Comm_disconnect (comm)	**int** MPI_Comm_disconnect (MPI_Comm *comm)
		Extensiones y utilidades MPITB
M-file	info = MPE_Pack (var, [var,]... buf, pos, comm)	Empaqueta variables agrupadas en *cell-array*
M-file	[info size] = MPE_Pack_size (var, [var,]... comm)	Tamaño para empaquetar con MPE_Pack
MPI	info = MPE_Attr_put (comm, kv, attr)	Para usar en conjunción con MATLAB_*_FN
M-file	[info attr_out flag] = MATLAB_COPY_FN (OLDcomm, keyval, extra_st, attr_in)	Esqueleto de función de copia de atributos
M-file	info = MATLAB_DEL_FN (comm, keyval, attr, extra_st)	Esqueleto de función de borrado de atributos
M-file	MATLAB_ERRORS_RETURN	MPI_Errhandler MPI_ERRORS_RETURN
M-file	STARTUP_MM	Script de inicialización procesos MATLAB hijos
M-file	’str’ = hostname	Interfaz con el comando UNIX hostname
M-file	[HN1 HN2...] = parsebhost	Devuelve nombres de computadores configuración LAM

comandos frente a los 93 de PVMTB, y que la forma de reutilización de código es más elaborada que la simple aplicación repetitiva de patrones a llamadas, no pareció prudente abrumar al lector con detallados listados en un estudio similar al realizado para PVMTB en el Apéndice C.

4.3.1 Categoría general MPI

El Listado 4.1 muestra un primer fragmento del fichero **#include** general, hMPI.h. Este fichero incluye los bloques constructivos más frecuentemente utilizados y "rutinas API propias" de interfaz con el sistema LAM/MPI para facilitar el manejo de tipos de datos MPI desde el entorno MATLAB, las cuales, debido a que son compartidas por varias categorías, resulta provechoso definir globalmente al objeto de ahorrar esfuerzo de mantenimiento del software.

```
/* hMPI.h */
#include <mex.h>          /* Matlab */
#include <mpi.h>          /* MPI_* */

#define MEX_HDR                                                          \
/*-----------------------------------------------------------------------*/
void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[]){ \
/*-----------------------------------------------------------------------*/

/*-----------------------------------------------------------------------
 *---------Bloques constructivos------------------------------------------
 *-----------------------------------------------------------------------*/

#define FUNCALL                                                     \
        *mxGetPr(plhs[0]=mxCreateDoubleMatrix (1,1,mxREAL)) =

#define PUTSCALAR(WHAT,WHERE)                                       \
        *mxGetPr(plhs[WHERE]=mxCreateDoubleMatrix (1,1,mxREAL)) = WHAT;
```

Listado 4.1: Parte inicial del fichero hMPI.h, en donde se incluyen los bloques constructivos más comúnmente usados para el desarrollo de MPITB.

Como ya se comentó en el desarrollo de PVMTB, el punto de entrada de los ficheros MEX es la función mexFunction(), en lugar de la habitual main() de un programa C independiente. Definir el bloque constructivo MEX_HDR ahorra teclear el prototipo para cada uno de los comandos MPITB. Recordemos de nuevo que el intérprete MATLAB pasa al fichero MEX el array de argumentos de entrada prhs[] (*Pointers to Right-Hand-Side*), y espera que se devuelva un array de nlhs valores de retorno, plhs[] (*Pointers to Left-Hand-Side*). El tamaño de ambos arrays se indica con los argumentos formales nrhs y nlhs, que son ajustados para coincidir con el comando tecleado por el usuario en el entorno interpretado (o redactado en un *script*).

También se definen las macros FUNCALL (llamada a función) y PUTSCALAR (retorno de valor). La primera está redactada de manera que escribiendo a continuación la llamada MPI correspondiente, el código de error resultante se devuelva como primer valor de retorno plhs[0]. La segunda devuelve como WHERE-ésimo valor de retorno el escalar WHAT, siendo ambos símbolos argumentos de la macroinstrucción. Ambos bloques usan las llamadas API MATLAB mxCreateDoubleMatrix(), que crea una matriz del tamaño y tipo indicado, y mxGetPr(), que devuelve un puntero (**double***) a la zona de almacenamiento de dicha matriz.

Uno de los tipos omnipresentes en la biblioteca MPI es el comunicador. Dado que LAM lo implementa como un puntero (**typedef struct** _comm *MPI_Comm), tipo no disponible bajo MATLAB, se ha optado por almacenar el valor en un *string* MATLAB, como se muestra en el Listado 4.2.

```
MPI_Comm mxGetMPIComm(const mxArray*arr){
MPI_Comm COM;
char*str = mxArrayToString(arr);
        if (!strcmp(str,"NULL" )) COM=MPI_COMM_NULL ;
   else if (!strcmp(str,"WORLD")) COM=MPI_COMM_WORLD ;
   else if (!strcmp(str,"SELF" )) COM=MPI_COMM_SELF ;
   else                 COM=    *(MPI_Comm*)mxGetData(arr);
        mxFree(str);                    /* No es error, se devuelve *str */
        return(COM);
}

mxArray * mxPutMPIComm(MPI_Comm COM){                   /* sizeof(mxChar)==2 */
const int dims[] = {1, sizeof(MPI_Comm)/sizeof(mxChar)};
mxArray * arr    = mxCreateCharArray(2,dims);
                 *(MPI_Comm*)mxGetData(arr) = COM;
        return(arr);
}
```

Listado 4.2: Fichero hMPI.h (continuación). Tratamiento de comunicadores.

MPITB define también las constantes MPI_COMM_WORLD, _NULL y _SELF (de tipo *string* MATLAB), pero se ofrece al usuario la alternativa de especificar el comunicador como *string* legible, pudiendo teclearse 'WORLD' como argumento comunicador de un comando MPITB. Este comportamiento está implementado en las primeras líneas de la función mxGetMPIComm(). También se pueden proporcionar como argumento comunicador las constantes definidas en MPITB, o alguna variable *string* que se haya usado para almacenar el comunicador devuelto por otro comando MPITB.

A diferencia de mxGetPr(), que devuelve **double***, la rutina API MATLAB mxGetData() devuelve un puntero **void*** a la zona de almacenamiento de la variable MATLAB. Cambiando el tipo mediante un *typecast* (MPI_Comm*) y desreferenciando el puntero resultante se accede a 4 bytes del *string* que son considerados como comunicador MPI. Esta conversión de tipo se realiza tanto en la obtención mxGetMPIComm() como en la devolución mxPutMPIComm() de comunicadores al entorno MATLAB.

Se ha extendido por tanto la API MATLAB con la creación de nuevas rutinas para tratamiento de los tipos específicos de MPI, convirtiéndolos a tipos que puedan ser almacenados en variables MATLAB. Así, el fichero global hMPI.h incluye definiciones para:

mxGetMPIReq(): recupera punteros de tipo **typedef struct** _req *MPI_Request almacenados en *strings* MATLAB. La correspondiente rutina mxPutMPIReq sólo se usa en la categoría TstWait, incluyéndose su código por tanto en hTstWait.h.

mx[Get/Put]MPIStat(): traducen a/de *structs* MATLAB el tipo C **typedef struct** _status MPI_Status.

mxGetMPIType(): escoge el tipo MPI apropiado a una variable MATLAB no estructurada. La correspondencia realizada es:

mxCHAR_CLASS	↔	MPI_UNSIGNED_SHORT
mxDOUBLE_CLASS	↔	MPI_DOUBLE
mxSINGLE_CLASS	↔	MPI_FLOAT
mx[U]INT8_CLASS	↔	MPI_CHAR/BYTE
mx[U]INT16_CLASS	↔	MPI_[UNSIGNED]_SHORT
mx[U]INT32_CLASS	↔	MPI_INT/UNSIGNED

Los restantes tipos MATLAB (mxCELL_CLASS, mxSTRUCT_CLASS, etc) no tienen una correspondencia directa con tipos MPI predefinidos, por lo que es necesario empaquetarlos y transmitirlos con tipo MPI_PACKED usando el comando MPITB MPI_Pack.

mxGetMPIBuff: devuelve un puntero a la zona de almacenamiento de una variable MATLAB.

Esta última rutina es utilizada asiduamente como bloque constructivo para obtener el triplete de argumentos (buff,type,size) presente en muchas llamadas MPI, por lo que mostramos el código en el Listado 4.3.

```
void * mxGetMPIBuff(const mxArray*arr, int argno, int*SIZ, MPI_Datatype *TYPE){
        *TYPE=mxGetMPIType(arr, argno);
         if (mxIsComplex(arr)){
                mexWarnMsgTxt("***␣variable␣compleja,␣usar␣MPI_Pack␣***");
                mexPrintf("***␣trabajando␣sólo␣con␣parte␣real␣***\n");}
        *SIZ=mxGetNumberOfElements(arr);
         return(mxGetData(arr));
}
```

Listado 4.3: Fichero hMPI.h (final). Obtención de buffer, tipo y tamaño.

A la rutina mxGetMPIBuff() de “nuestra API” se le proporciona una variable MATLAB (de tipo mxArray) como argumento. argno se utiliza sólo para proporcionar mensajes de error útiles desde mxGetType(), señalando al usuario el argumento cuyo tipo causó problemas. La rutina mxGetMPIBuff() devuelve un puntero (**void***) a la zona de almacenamiento, que como vimos se obtiene usando la llamada mxGetData(). También se devuelven el tipo (obtenido con nuestra rutina mxGetMPIType()) y tamaño (con la rutina API mxGetNumberOfElements()) de la variable MATLAB. Los tipos complejos se deberían empaquetar con MPI_Pack, aunque se proporciona la posibilidad de trabajar directamente sólo con la parte real.

Como vemos, se respetan los mismos criterios seguidos para PVMTB. El tamaño y tipo de una variable están almacenados en su descripción mxArray, pudiendo obtenerse muy rápidamente en tiempo de ejecución. Las llamadas API simplemente consultan los campos respectivos de la estructura mxArray. Se ahorra por tanto al usuario la redundante (bajo MATLAB) especificación de parámetros en la llamada MPITB.

En relación con este tema, y recordando lo comentado sobre correspondencia de tipos MATLAB↔C durante la descripción de MPI_GetMPIType(), conviene resaltar el motivo por el cual MPITB no recubre el estándar MPI 1.2 completo, dejando algunas llamadas MPI sin interfaz. Estas llamadas son:

MPI_Pcontrol(): La funcionalidad de perfilado (*profiling*) depende de la biblioteca concreta, no estando definida por el estándar. En cualquier caso, es más probable localizar los tramos de código más costosos usando el comando MATLAB profile que recurriendo al perfilado MPI. El entorno MATLAB es interpretado, y MPITB ha sido diseñada para prototipar aplicaciones HPC en forma de ficheros M.

MPI_Op_*(): Las llamadas MPI_Op_create() y MPI_Op_free() permitirían definir operadores para las llamadas colectivas de reducción. Se podrían redactar *wrappers* C para recubrir la ejecución de ficheros M que implementaran el operador, pero estos ficheros M no tendrían forma de operar sobre los datos pasados por MPI. El prototipo con el que MPI invoca al operador es

```
typedef void (MPI_User_function)(void *, void *, int *, MPI_Datatype *);
```

Está fuera de lugar considerar la posibilidad de que los *wrappers* realizaran conversión de tipos MPI↔MATLAB en entrada y salida del operador M.

MPI_Type_*(): Las llamadas de tipos permiten definir nuevos tipos MPI a partir de los tipos básicos, y utilizarlos en las posteriores llamadas MPI que transmitan o manipulen datos de dichos tipos. Sin embargo, el usuario MATLAB no puede ni conocer la disposición en memoria de sus variables, ni crear variables con una disposición predeterminada. Dicho de otra forma, ni se pueden crear variables MATLAB que sigan la disposición (*layout*) de una definición de tipo MPI, ni se dispone de información suficiente como para definir en MPI el *layout* de las variables MATLAB.

Esto hace inservibles las llamadas de tipos bajo MATLAB. Hay bastantes: MPI_Type_extent, _size, _lb, _ub, _free, _commit, _contiguous, _vector, _hvector, _indexed, _hindexed y _struct.

Todos los comandos MPITB incluyen el fichero hMPI.h. Por estudiar un par de ejemplos de distinta complejidad, en el Listado 4.5 aparece el código fuente del comando MPITB MPI_Init, y en el Listado 4.4 el de MPI_Finalize.

```
/*
 * Finalización de MPI, termina entorno ejecución MPI
 * info = MPI_Finalize
 */
#include "hMPI.h"
        MEX_HDR
        FUNCALL MPI_Finalize ();
}
```

Listado 4.4: Código fuente del comando MPITB MPI_Finalize.

La redacción del último es más sencilla, por lo que lo estudiaremos primero. Se trata de una función que sólo devuelve el obligatorio código de retorno MPI, por lo cual tras incluir el prototipo mexFunction() mediante la macro MEX_HDR que definimos en hMPI.h, se usa la macro de llamada FUNCALL pasándole el propio código de retorno de MPI_Finalize(). Se entiende ahora por qué se definió esta macro acabando en asignación:

```
*mxGetPr( plhs [0]= mxCreateDoubleMatrix (1,1, mxREAL)) =
```

El código devuelto se asigna a la zona de datos (referenciada usando *mxGetPr()) del escalar real MATLAB (creado con mxCreateDoubleMatrix()) que se devuelve como primer valor de retorno (plhs[0]).

Los argumentos de MPI_Init() son peculiares, y reflejan la suposición de que existirá un comando mpirun. Por respetar el estándar, el comando MPITB MPI_Init contempla la posibilidad de un número indefinido de argumentos *string* (Listado 4.5).

Se usan de nuevo las macros MEX_HDR para empezar y FUNCALL para devolver el código de retorno, pero antes de la llamada a MPI se deben traducir los *strings* MATLAB a lenguaje C usando la llamada API mxArrayToString(), y la memoria reservada para dichas traducciones se debe liberar después de la llamada MPI usando mxFree(). La rutina MPI espera que los *strings* se le pasen en un array C (indicando también el número de elementos). El array C se reserva con mxCalloc y también se debe liberar tras la llamada.

```
/*
 * Arranque de MPI, inicializa entorno ejecución MPI
 * info = MPI_Init [ ( 'arg' [, 'arg']... ) ]
 */
#include "hMPI.h"
        MEX_HDR
        int i;
        char **args;
        for (i=0; i<nrhs; i++) if (!mxIsChar(prhs[i]))
                mexErrMsgTxt("args␣deben␣ser␣strings");
        args = mxCalloc(nrhs, sizeof(char*));
        for (i=0; i<nrhs; i++) args[i] = mxArrayToString(prhs[i]);

        FUNCALL MPI_Init(&nrhs, &args);
        for (i=0; i<nrhs; i++) mxFree(args[i]); mxFree(args);
}
```

Listado 4.5: Código fuente del comando MPITB MPI_Init.

4.3.2 Categoría punto a punto SndRecv

Esta categoría da una buena idea del tipo de estudio realizado sobre las llamadas MPI, que no parece especialmente complicado atendiendo al resultado final en la Tabla 4.1. El estudio es del mismo detalle que el realizado para PVMTB, agravado por el hecho de que las llamadas MPI son mucho más numerosas y los patrones de llamada se repiten menos. Afortunadamente, los patrones presentan secuencias repetitivas que hemos denominado informalmente “dobletes”, “tripletes”, e incluso “cuádruplas”, mostrando esta categoría un buen número de ellas.

La categoría hSndRecv contiene casi todas las llamadas MPI punto a punto. Los respectivos comandos MPITB, tras aplicar los criterios de diseño mencionados, se agrupan por similitud de procesamiento:

SND:

```
           info = MPI_ Send       (buf, dest, tag, comm)
           info = MPI_Bsend       (buf, dest, tag, comm)
           info = MPI_Ssend       (buf, dest, tag, comm)
           info = MPI_Rsend       (buf, dest, tag, comm)
```

RCV:

```
     [info stat] = MPI_ Recv      (buf, src,  tag, comm)
```

ISND:

```
     [info req ] = MPI_I send     (buf, dest, tag, comm)
     [info req ] = MPI_Ibsend     (buf, dest, tag, comm)
     [info req ] = MPI_Issend     (buf, dest, tag, comm)
     [info req ] = MPI_Irsend     (buf, dest, tag, comm)
     [info req ] = MPI_I recv     (buf, src,  tag, comm)
     [info req ] = MPI_ Send_init (buf, dest, tag, comm)
     [info req ] = MPI_Bsend_init (buf, dest, tag, comm)
     [info req ] = MPI_Ssend_init (buf, dest, tag, comm)
     [info req ] = MPI_Rsend_init (buf, dest, tag, comm)
     [info req ] = MPI_ Recv_init (buf, src,  tag, comm)
```

SNDRCV:

```
     [info stat] = MPI_Sendrecv  (sbuf, dest, stag, ...
                                  rbuf, src , rtag, comm)
```

SNDRCVREP:

```
[info stat] = MPI_Sendrecv_replace (buf, dest, stag, ...
                                    src , rtag, comm)
```

Los cuatro comandos del primer grupo comparten el patrón PATN_SEND. El código fuente de MPI_Send consiste sencillamente en

```
#include "hMPI.h"
#include "hSndRecv.h"
        PATN_SEND(MPI_Send)
```

variando para los demás comandos del grupo únicamente el nombre de la llamada MPI usada como argumento al patrón.

La definición de PATN_SEND realizada en hSndRecv.h se muestra en el Listado 4.6. Consta del prototipo MEX_HDR y la reutilización del bloque constructivo PATN_SND_RCV, que se ha extraido de la definición dado que es posible compartirlo con el grupo RCV.

Este tipo de estudio es la firme base que permite a MPITB reutilizar efectivamente el código a pesar de no ser demasiado repetitivos los patrones de llamada MPI. Los grupos [I]SND son tal vez la más notable excepción, siendo totalmente repetitivos, e incluso en tal caso es provechoso utilizar el método de los bloques constructivos para producir un diseño robusto con bajo coste de mantenimiento.

```
#define PATN_SND_RCV              /* bloque constructivo previo */        \
        MPI_Datatype TYPE;                                                 \
        MPI_Comm COM;                                                      \
        void * BUF;                                                        \
        int    SIZ, srcdst , tag;                                          \
        if ((nrhs!=4)     ||(!* mxGetName   (prhs[0]))                     \
                          || (! mxIsNumeric(prhs[1]))                      \
                          || (! mxIsNumeric(prhs[2]))                      \
                          || (! mxIsChar    (prhs[3])))                    \
        mexErrMsgTxt("se requiere buff(var), src/dst, tag(int), comm(str)");\
        BUF    = mxGetMPIBuff(prhs[0],0, & SIZ,&TYPE);                     \
        srcdst = mxGetScalar  (prhs[1]);                                   \
        tag    = mxGetScalar  (prhs[2]);                                   \
        COM    = mxGetMPIComm(prhs[3]);                                    \
        FUNCALL

#define PATN_SEND(NAME)           /* patrón de las variantes MPI_Send */  \
        MEX_HDR                                                            \
        PATN_SND_RCV      NAME(BUF, SIZ, TYPE, srcdst, tag, COM);          \
}
```

Listado 4.6: Definición del patrón PATN_SEND en función del bloque constructivo PATN_SND_RCV.

El bloque PATN_SND_RCV realiza el procesamiento de la cuádrupla (buf, dest, tag, comm) por la que se caracterizan los grupos [I]SND/RCV, usando llamadas API para comprobar que los tipos de datos son correctos, y una mezcla de llamadas API MATLAB (mxGetScalar()) y de rutinas de hMPI.h (nuestras mxGetMPIComm(), etc) para traducir los argumentos de MATLAB a C.

Conviene destacar el uso de mxGetMPIBuff(), que devuelve un puntero a la zona de datos de la variable así como su tamaño y tipo MPI (triplete (buff,size,type)). Nótese que se exige que sea una variable (mxGetName()), no pudiéndose usar como buffer MPI un valor literal indicado en la llamada al comando MPITB MPI_Send.

El patrón PATN_RECV se define también en hSndRecv por comodidad de programación. Igualmente podría codificarse en el fichero fuente del comando MPITB MPI_Recv, ya que es el único que lo utiliza. Sin embargo, esto dificultaría la posterior revisión o mantenimiento, habiéndose preferido codificar MPI_Recv como

```
#include "hMPI.h"
#include "hSndRecv.h"
        PATN_RECV
```

e incluyendo en hSndRecv la definición del patrón PATN_RECV.

```
#define PATN_RECV                                                    \
        MEX_HDR                                                      \
        MPI_Status      ST;                                          \
        PATN_SND_RCV MPI_Recv(BUF, SIZ, TYPE, srcdst, tag, COM, &ST);  \
        plhs[1]         = mxPutMPIStats(1,&ST);                      \
}
```

La diferencia con el grupo SND estriba en que hay que devolver otro valor de retorno, la estructura de estado de recepción, usando "nuestra llamada API" mxPutMPIStats() definida en hMPI.h. Ese único detalle impide reutilizar el patrón PATN_SND_RCV, motivo por el cual se extrajo la funcionalidad para obtener la cuádrupla de argumentos a un bloque separado, que ahora se entiende que se denominara PATN_SND_RCV. Naturalmente, el patrón PATN_ISEND utilizado por todo el grupo ISND reutiliza también este bloque, así como nuestra rutina API mxPutMPIReq (incluida en hMPI.h) para devolver req.

```
#define PATN_ISEND(NAME)                                             \
        MEX_HDR                                                      \
        MPI_Request     REQ;                                         \
        PATN_SND_RCV    NAME(BUF, SIZ, TYPE, srcdst, tag, COM,&REQ);   \
        plhs[1]         = mxPutMPIReq(REQ);                          \
}
```

La reutilización es máxima, quedando reducida la codificación de los 10 comandos del grupo ISND a tres líneas, como por ejemplo para MPI_Irecv:

```
#include "hMPI.h"
#include "hSndRecv.h"
        PATN_ISEND( MPI_Irecv )
```

Por no abrumar al lector con infinidad de detalles que son más materia de la propia autodocumentación del código fuente de MPITB que objeto de esta memoria, se prefiere obviar el resto de la explicación. Las técnicas usadas para analizar los patrones de los comandos MPITB han sido presentadas con suficiente detalle como para tener una idea precisa de cómo se realiza el resto de la *Toolbox*.

Los componentes clave del diseño MPITB son por tanto:

Rutinas de traducción: informalmente descritas como "nuestras rutinas API" en las explicaciones precedentes, permiten el almacenamiento de entidades MPI en variables MATLAB. Así, los comunicadores (**typedef struct** _comm *MPI_Comm) se almacenan en *strings* MATLAB de 2 caracteres multilinguales (4 bytes). Similarmente se hace con otros objetos MPI cuyo tipo es un puntero. Los objetos MPI así almacenados requieren también otra rutina para volverlos a traducir a C.

No todos los objetos MPI se almacenan en *strings* MATLAB. Por ejemplo, el tipo de estado para recepción es una estructura C (**typedef struct** _status MPI_Status), que puede ser convertida de/a MATLAB, pudiendo el usuario consultar y manipular los campos aislados con instrucciones MATLAB.

Categorías: La propia funcionalidad de las llamadas MPI las clasifica en categorías, según el tipo de datos de sus argumentos. Así, está claro que sólo las llamadas relacionadas con arranque dinámico de procesos se verán envueltas en manipulaciones del objeto MPI_Info.

Se redacta por tanto una serie de rutinas de traducción para los tipos de datos usados por una categoría, agrupándolas en un fichero **#include** con el nombre de la categoría. Este fichero contiene también patrones y bloques constructivos.

Algunas rutinas son tan frecuentes que se usan en casi todas las categorías. El criterio de no repetir funcionalidad ya programada nos ha llevado a mover rutinas de traducción de una categoría **A** al fichero global hMPI.h en cuanto otra categoría **B** puede reutilizarla.

Patrones: El estudio de los patrones persigue localizar procesamiento común de argumentos y valores de retorno, al objeto de codificar una única vez la diversa funcionalidad. Idealmente, un patrón debería ser reutilizable para muchos comandos MPITB. Eso sucedía con PVMTB, pero en MPITB es común que una rutina tenga su propio patrón (los grupos [I]SND son la más notable excepción).

Aún así, es conveniente diseñar patrón incluso para una única rutina, quedando el código programado en el mismo fichero **#include**, en donde es fácil de revisar y mantener (junto a sus bloques).

Además, esta técnica permite repasar cuáles fueron las decisiones que llevaron a considerar un comando MPITB como parte de un grupo u otro dentro de su categoría, siendo una poderosa forma de autodocumentación del diseño.

Bloques constructivos: Son el auténtico vehículo que soporta la reutilización de código. Idealmente deberían ser identificados observando los patrones de llamada de los comandos MPITB que forman el grupo. Muchos bloques se han creado así, pero un buen número de ellos sólo han sido detectados al redactar un patrón nuevo, momento en el que se recuerda haber programado ya esa funcionalidad. En cuanto dos patrones han mostrado algún tramo parecido, la funcionalidad asociada se ha extraido a un bloque constructivo.

4.4 Eficiencia de MPITB

El estudio de escalabilidad del Capítulo 1 estaba más orientado a comparar las distintas *Toolboxes* paralelas que a estimar el *overhead* de las mismas. De hecho, tanto PVM como MPI obtenían menor *speedup* que PVMTB y MPITB. Vimos que esto se debía a que el tiempo de cálculo se incrementaba en un orden de magnitud bajo MATLAB, alterando la relación comunicación/cálculo.

El test *ping-pong* realizado en el mismo capítulo tampoco evalúa correctamente el *overhead* de MPITB respecto a MPI, siendo el barrido de tamaños de mensaje realizado muy corto para obtener una idea precisa de la aportación de la *Toolbox* al tiempo total de transmisión.

Los tests *ping-pong* del Capítulo 2 son mucho más detallados, pero están orientados a comparar PVM y MPI. En el capítulo anterior se realizó el estudio comparativo PVM/PVMTB, y ahora procedemos al estudio que queda por realizar, en donde estimaremos el *overhead* introducido por MPITB en el paso de mensajes usando el sistema LAM/MPI.

4.4.1 Test *ping-pong* en C

Al igual que en el capítulo anterior, programamos ahora el mismo test *ping-pong* tanto en C usando MPI como en MATLAB usando MPITB. Se repiten los tests utilizando o no la opción de cluster homogéneo (opción equivalente a PvmDataRaw frente a PvmDataDefault), combinándola o no con la opción de transmisión cliente a cliente (equivalente al encaminamiento PvmRouteDirect/DontRoute en PVM). Se usó la configuración estándar de LAM/MPI con MAXNMSGLEN==8KB. El equipo utilizado es el mismo cluster "oxígeno".

En oposición a PVM, donde las opciones de empaquetamiento y encaminamiento se especifican en tiempo de ejecución mediante las llamadas pvm_setopt() y pvm_initsend(), bajo MPI es más apropiado especificar o no los modificadores al comando mpirun –O –c2c utilizado para ejecutar el test (ver Listado D.6 en Apéndice D). Se realiza el mismo barrido quasi-exponencial de tamaños de mensaje que se hizo bajo PVM, tanto para datos **double** como para **char**. La latencia o tiempo de *setup* empleado en transmitir el mensaje nulo se vuelve a medir en cada iteración del barrido, como muestra el Listado 4.7.

```
/* BW.c: Medición de latencia (ping-pong) y ancho de banda */
char lat=' ';                                      /* pvm_pkbyte para latencia */
...
  for (ExpAncho=0; ExpAncho<13; ExpAncho++){
    Ancho=1<<ExpAncho;

    for (Alto=Ancho; Alto<4*Ancho; Alto+=Ancho){
      indx++; nelem=Alto*Ancho;

      if (((indx>=Dstr)&&(indx<=Dend))||           /* algo que hacer */
          ((indx>=Ustr)&&(indx<=Uend))){

        NTIMES=ntimes*reps(0);
        MPI_Barrier(MPI_COMM_WORLD);               /* hacer parte latencia */

        T=MPI_Wtime();
        for (i=0; i<NTIMES; i++){
          MPI_Send(&lat,0,MPI_CHAR, 1,TAG,MPI_COMM_WORLD     );
          MPI_Recv(&lat,0,MPI_CHAR, 1,TAG,MPI_COMM_WORLD, &ST);
        }
        T=MPI_Wtime()-T;                 l[indx]=T/2/NTIMES;
      }
      ...                                          /* hacer parte double */
    }
    ...
  }
...
```

Listado 4.7: Programa *ping-pong* para MPI: bucle de tamaños y sección dedicada a la latencia. La dedicada a transmitir **double**s se ofrece en el Listado 4.8. El Listado D.7 muestra el programa completo.

Los tamaños de array crecen linealmente hasta el triple (1x1, 1x2, 1x3), y entonces se produce el paso exponencial (2x2, 2x4, 2x6...). Dado que se realizarán mediciones usando arrays tanto de **double**s como de **char**s al objeto de poder detectar el coste de la codificación XDR, y que

los bucles barren las dimensiones del array, resultando en mensajes **char** con la octava parte (**sizeof**(**double**)==8 en nuestro código) del mensaje **double**, se sigue el mismo sistema empleado bajo PVM de identificar mediante variables Ustr/Uend y Dstr/Dend los valores de índice indx en los que se debe empezar y dejar de realizar mediciones para cada tipo. Un argumento del programa de medición permite alterar estas variables auxiliares.

Siempre que se vaya a realizar una medición, sea **double** o **char**, se mide primero la latencia con el bucle del Listado 4.7. Se utiliza esta vez la llamada MPI_Barrier() para empezar a cronometrar con MPI_Wtime(). El bucle usa las rutinas básicas punto a punto MPI_Send/Recv(), utilizándose un segundo argumento (tamaño) nulo para medir la latencia. El tiempo transcurrido MPI_Wtime()−T se divide por NTIMES confiando en que los costes de inicialización del bucle queden disimulados, y por 2 para obtener el tiempo de ida (*ping*) en lugar del de ida y vuelta.

Se procede entonces a medir tanto el tiempo de reserva de memoria como el tiempo de transmisión para un mensaje de las dimensiones especificadas. El Listado 4.8 muestra el tramo de código correspondiente a mensajes **double**. Sólo se ejecuta si el índice indx está entre los límites apropiados Dstr/Dend.

```
if ((indx>=Dstr)&&(indx<=Dend)){             /* hacer parte double */

  NTIMES=ntimes*reps(nelem*sizeof(double));  arrayDouble=NULL;
  T=MPI_Wtime();
  for (i=0; i<NTIMES; i++){                  /* medir reserva memoria */
    free(arrayDouble);
    arrayDouble=calloc(nelem, sizeof(double));
  }
  T=MPI_Wtime()-T;          d[indx][Size]=nelem*sizeof(double);
                            d[indx][ Mtm]=T/NTIMES;

  MPI_Barrier(MPI_COMM_WORLD);                /* ping-pong parte double */

  T=MPI_Wtime();
  for (i=0; i<NTIMES; i++){
    MPI_Send(arrayDouble, nelem, MPI_DOUBLE, 1, TAG, MPI_COMM_WORLD     );
    MPI_Recv(arrayDouble, nelem, MPI_DOUBLE, 1, TAG, MPI_COMM_WORLD, &ST);
  }
  T=MPI_Wtime()-T;          d[indx][TXtm]=T/2/NTIMES;

  free(arrayDouble);
}
```

Listado 4.8: Programa *ping-pong* para MPI (continuación). Sección dedicada al tiempo de transmisión de **double**s.

En el primer bucle se reservan repetidamente nelem **double**s. Tras esta medición viene un bucle similar al usado para la latencia (Listado 4.7), salvo que cambian el tamaño nelem y tipo MPI_DOUBLE de los datos transmitidos. Se usa la misma función reps() que se introdujo en el capítulo anterior para repetir mayor número de veces los tamaños más pequeños.

La sección para **char**s es idéntica, salvo que se usan los límites Ustr y Uend en el **if**, arrayUint8 en calloc() y free(), **unsigned char** en **sizeof**(), MPI_BYTE en lugar de MPI_DOUBLE, y se anotan las mediciones en el array u en lugar de d. La latencia se anotó en el array l.

Al igual que con PVM, el objeto de repetir la medición de la latencia es para obtener posteriormente la media de todas sus mediciones. Una incorrecta estimación de la latencia podría traducirse en anchos de banda (según el modelo afín) superiores a los físicamente realizables por el *switch*, o incluso negativos. En efecto, si se obtuviera por azar una latencia $L = T_0$ mayor

que el tiempo de transmisión de un **double** $T_8 < T_0 = L$, usando el modelo afín $T = L + S/B$, el ancho de banda $B = S/(T - L)$ que se obtendría para **sizeof**(**double**)==8 bytes sería negativo: $B_8 = 8/(T_8 - L) < 0 \Leftrightarrow (T_8 - L) < 0 \Leftrightarrow T_8 < T_0 = L$.

Las Figuras de la 4.1 a la 4.4 muestran las mediciones correspondientes a la ejecución de este código MPI. Las dos primeras corresponden al modo *daemon* LAM, con o sin la opción de cluster homogéneo. Las dos siguientes corresponden al modo cliente-a-cliente. La estructura de las figuras es idéntica a la utilizada en el capítulo anterior, volviendo a explicarse en el pie de la primera figura.

Comparando el tiempo de reserva de memoria (trazos oscuros) con los obtenidos bajo PVM en las Figuras de la 3.1 a la 3.6, se observa que el pico en decenas de KBs es mucho más reducido, 0.1–0.3ms, más similar al obtenido con PvmDataInPlace/RouteDirect (Figura 3.6). Con las otras combinaciones de opciones PVM se obtenía un pico de 5–8ms en los centenares de KBs (7.9ms @ 1.5MB en Tabla 3.2). Bajo MPI no ha sido preciso tomar medidas especiales para evitar el *swapping* con tamaños de mensaje de 32MB, siendo ésta la razón que explica el artificio observado bajo PVM.

La latencia tiene un comportamiento igual de reproducible al observado bajo PVM, alrededor de 400μs para el modo *daemon* LAM (360μs para el *daemon* PVM, Tabla 3.2) y próxima a 125μs para ruta directa frente a los 185–190μs de PVM.

Respecto al tiempo de transmisión en modo *daemon* LAM (Figuras 4.1 y 4.2), sin la opción homogénea (–O) la codificación XDR para **double**s se hace notar (7–6MB/s) igual que sucedió con PVM (4–3MB/s, Figuras de la 3.1 a la 3.3). Con cluster homogéneo se igualan los anchos de banda, pero no a 7MB/s, sino a 8.5MB/s. A diferencia de PVM, la opción homogénea también mejora el tiempo de transmisión de **char**s. PvmDataInplace sólo presentaba este comportamiento con ruta directa.

Las dos siguientes gráficas (Figuras 4.3 y 4.4) añaden la opción cliente a cliente, creándose por tanto una ruta TCP directa entre tareas, en lugar de usar la ruta por defecto a través de los *daemon* LAM. Al igual que bajo PVM, los anchos de banda aumentan, aunque no al doble, naturalmente. Los valores se resumen en la Tabla 4.2.

Para cluster heterogéneo, la diferencia 6–7MB/s entre **char**-**double** en la Figura 4.1 abajo derecha pasa a 8–9.5MB/s en la Figura 4.3 con ruta directa. En PVM se pasaba de 3–4MB/s a 5–9.5MB/s. Con cluster homogéneo se pasa de 8.5MB/s en la Figura 4.2 a 11.75MB/s en la Figura 4.4. En PVM se pasaba de 4MB/s a 9.5MB/s con DataRaw y a 10.5MB/s con DataInPlace. La Tabla 4.2 presenta sumariamente los detalles ya destacados sobre el test *ping-pong* en MPI.

	BW (MB/s)		Latencia	Reserva Memoria (μs)	
	char	**double**	(μs)	extremos	pico
_-lamd	7.0	6.4–6.2	399	1–50	273μs@96KB
O-lamd	8.6–8.3		401	1–52	269μs
_-c2c	9.3	8.5–7.9	124	1–51	271μs
O-c2c	11.75		123	1–58	293μs

Tabla 4.2: Detalles remarcables del test *ping-pong* en MPI.

En la tabla se muestran los tiempos típicos para reserva de memoria al principio y final del barrido, así como la posición del máximo tiempo de reserva. La variabilidad observada es propia del gestor de memoria Linux, no estando asociada a la combinación de opciones MPI. Éstas

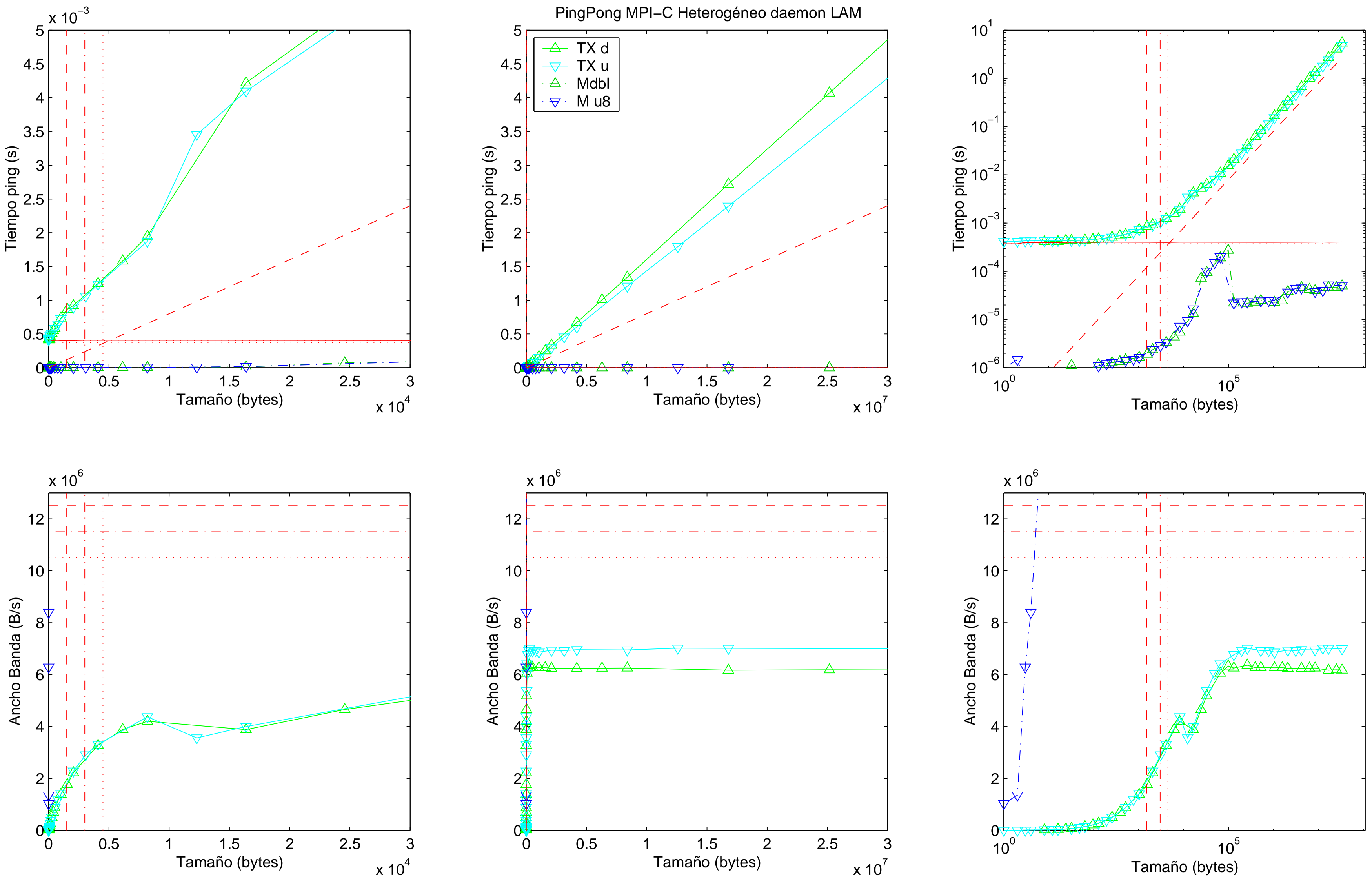


Figura 4.1: Test *ping-pong* bajo MPI, sin opciones −O −c2c. **Derecha**: gráfica logarítmica incluyendo todas las mediciones. **Izquierda**: detalles con abscisa lineal en los rangos de decenas de KB y MB. **Abajo**: anchos de banda correspondientes. **Leyenda**: **TX d**: tiempo de transmisión (ida) para array **double**. **TX u**: para array **char**. **Mdbl**: tiempo para reserva de memoria de array **double**. **M u8**: para array **char**.

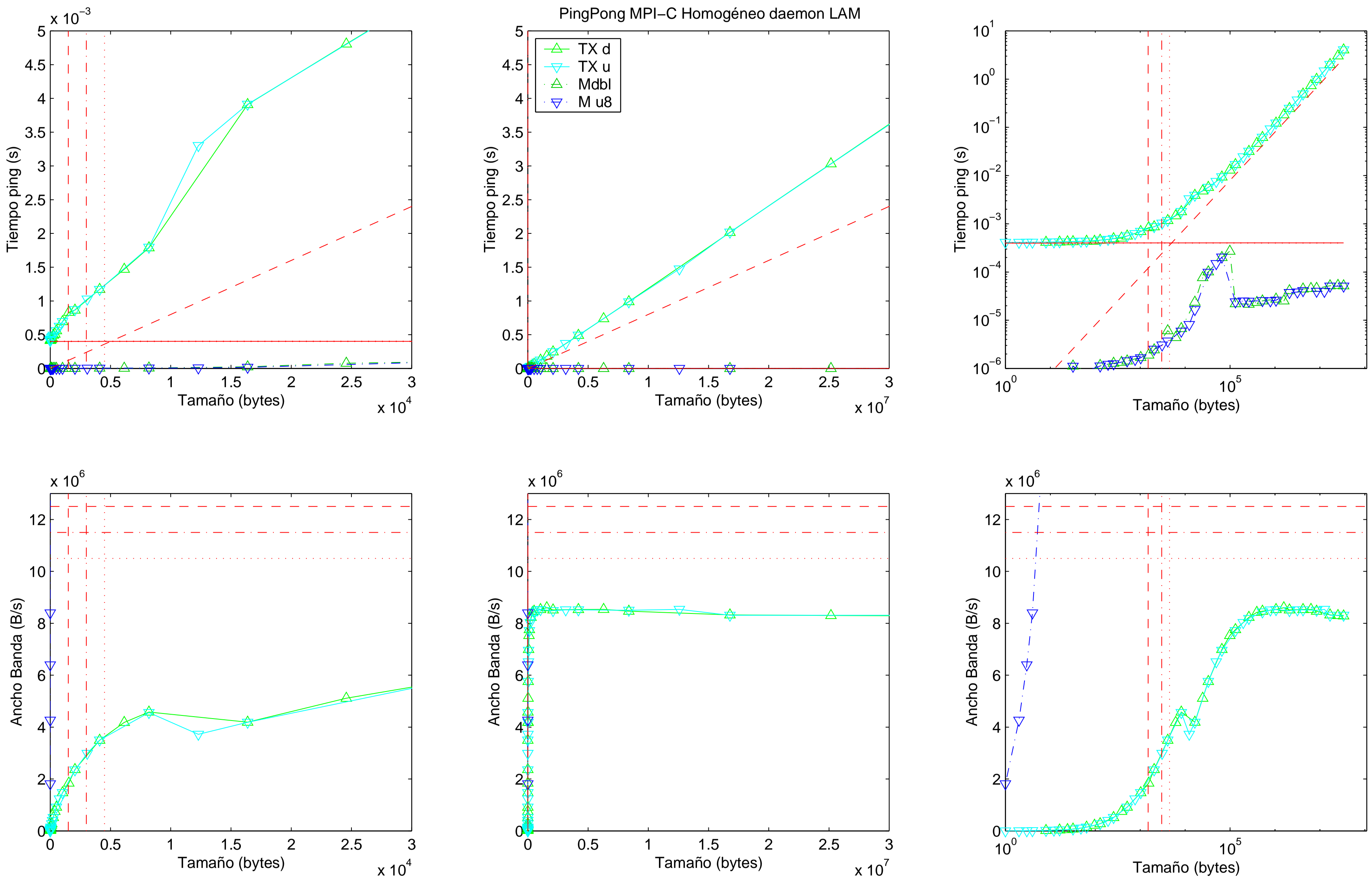


Figura 4.2: Test *ping-pong* bajo MPI, con opciones −O −lamd. Consultar pie de Figura 4.1 para más detalles.

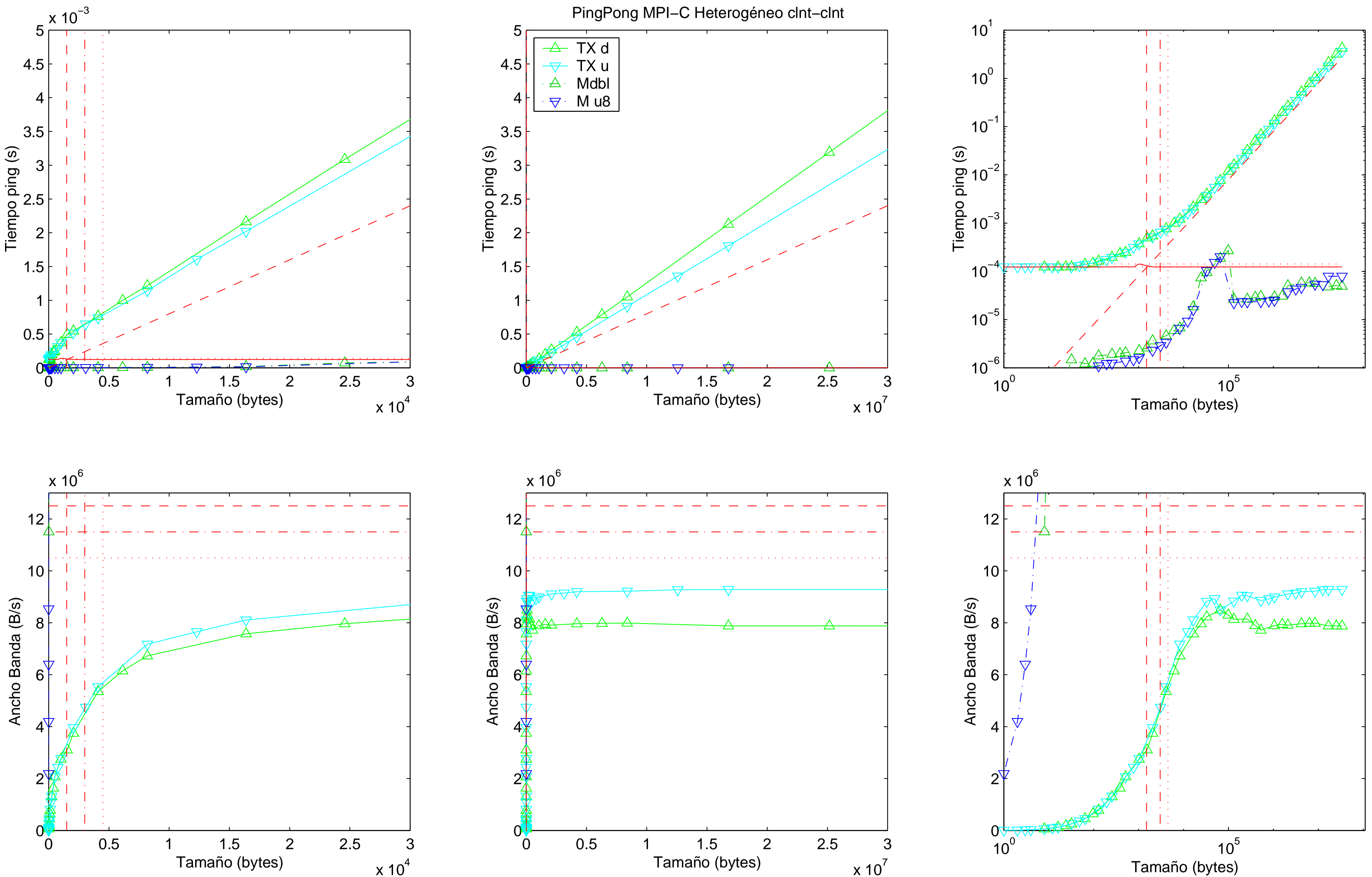


Figura 4.3: Test *ping-pong* bajo MPI, con la opción −c2c. Consultar pie de Figura 4.1 para más detalles.

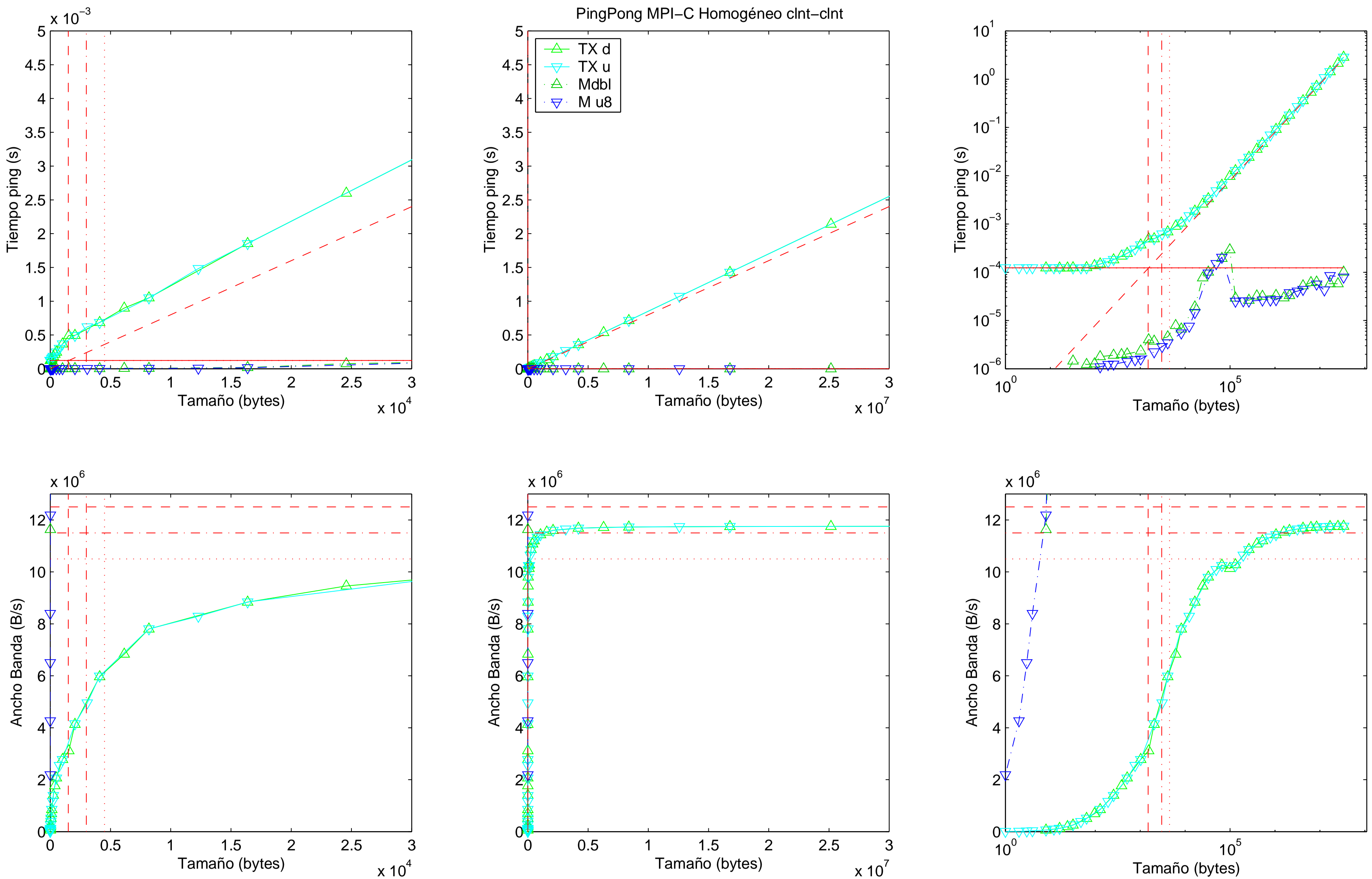


Figura 4.4: Test *ping-pong* bajo MPI, con opciones −O −c2c. Consultar pie de Figura 4.1 para más detalles.

se han abreviado a la nomenclatura utilizada con los nombres de fichero. Comparando con la Tabla 3.2 se comprueba hasta qué punto afectaba la liberación del buffer de envío al gestor de memoria Linux.

Resumiendo lo observado respecto al paso de mensajes, la ruta directa incrementa el ancho de banda (8.5–11.75MB/s) y disminuye la latencia (400–125μs). La opción de cluster homogéneo evita el *overhead* de codificación XDR para datos **double** (7–6MB/s, 9–8MB/s). Es de destacar que para tamaños pequeños la importancia de esta opción es mayor que la propia ruta directa, como se aprecia en el detalle de la siguiente figura.

En la Figura 4.5 se presenta una gráfica adicional comparando los tiempos de transmisión de datos **double** bajo las distintas combinaciones de opciones MPI. Se ha escogido **double** para que quede evidenciada la diferencia entre las opciones de cluster homogéneo y heterogéneo. La Figura 4.5(b) muestra una ampliación de la misma gráfica en las primeras 3 MTUs, en donde se puede observar una latencia cercana al citado 0.1ms para ruta directa y a 0.4ms a través del *daemon* LAM. Las gráficas mantienen la misma escala de la Figura 3.7 para permitir una fácil comparación visual.

Más adelante, en la Figura 4.11, se mostrará el tamaño de mensaje para el cual se produce el cruce entre el trazo del modo *daemon* LAM homogéneo y el del modo cliente a cliente heterogéneo. También más adelante, en la Tabla 4.4 izquierda, se presentan algunos puntos de medición de la mejor opción MPI, cliente a cliente homogéneo con datos **double**, al objeto de posibilitar una comparación objetiva con MPITB.

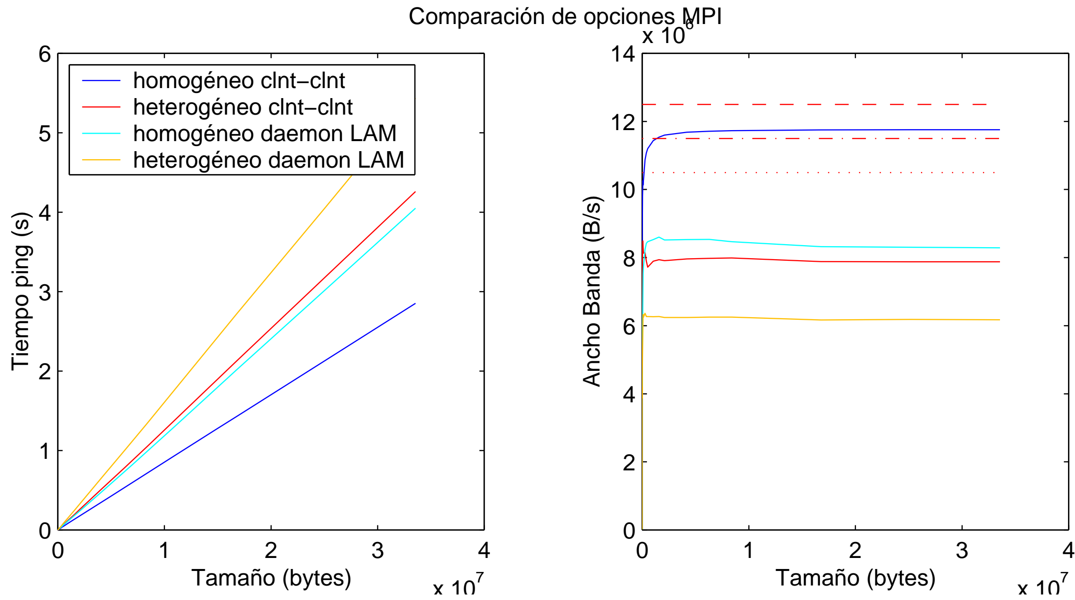
4.4.2 Test *ping-pong* en MATLAB

El mismo test *ping-pong* bajo MATLAB toma la forma que se muestra en el Listado 4.9:

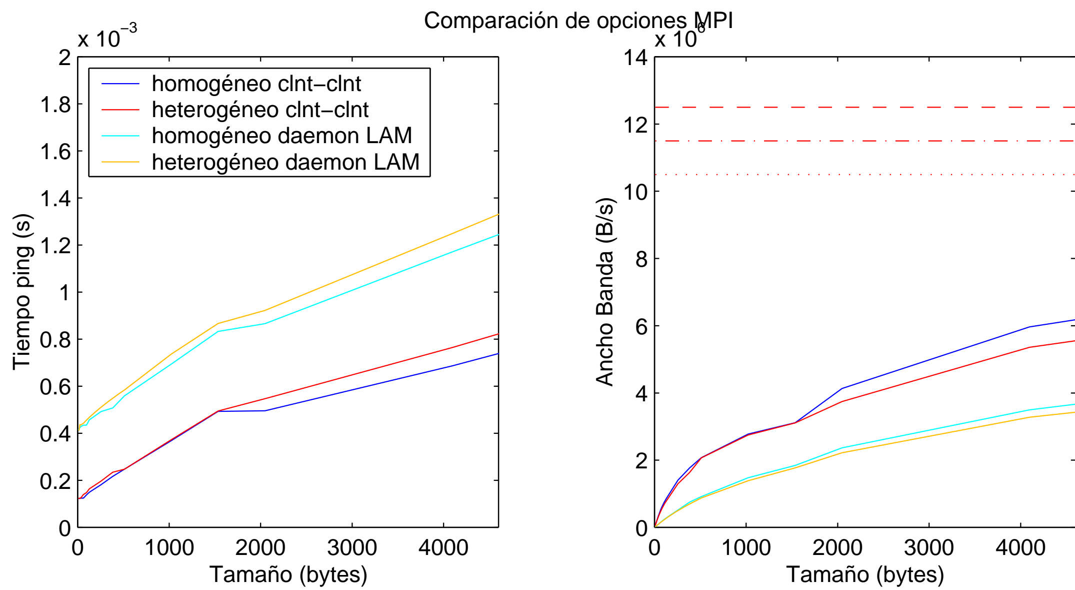
```
function [l u d]=BW(hmg, c2c, row, ntimes)
%BW: Medición de latencia (ping-pong) y ancho de banda
...
  arrayNULL=[];

  for ExpAncho=0:12
    Ancho=2^ExpAncho;

    for Alto=Ancho:Ancho:3*Ancho
      indx=indx+1;

      if (indx>=Dstr & indx<=Dend) |...                    % algo que hacer
         (indx>=Ustr & indx<=Uend)

        NTIMES=ntimes*reps(0);                             % hacer parte latencia
        MPI_Barrier(NEWORLD); T=clock; for i=1:NTIMES
          MPI_Send(arrayNULL,1,TAG,NEWORLD);
          MPI_Recv(arrayNULL,1,TAG,NEWORLD);
        end, T=etime(clock,T);
        l(indx)=T/2/NTIMES;
      end
      ...                                                  % hacer parte double
    end
    ...
  end
```

Listado 4.9: Programa *ping-pong* para MPITB: sección dedicada a la latencia. La dedicada a transmitir **double**s aparece en el Listado 4.10. El programa aparece completo en el Listado D.9.

Se puede apreciar la similitud con la versión C. La notación MATLAB es más compacta, sobre



(a) Barrido completo.



(b) Detalle de las primeras 3 MTUs.

Figura 4.5: Test *ping-pong* bajo MPI, comparación de opciones con datos **double**.

todo comparando el código (no mostrado aquí) de inicialización de variables. Los comandos MPITB MPI_Send y MPI_Recv resultan también algo más breves.

Los argumentos del programa MPITB controlan los mismos aspectos que la versión C: homogeneidad, encaminamiento, tamaño del barrido y nº de repeticiones. La sección mostrada reestima la latencia en cada iteración del barrido. Como sucedió con PVMTB frente a PVM, bajo MPITB se ha observado una mayor variabilidad de la latencia que bajo MPI.

De nuevo el bucle barre dimensiones de la matriz mensaje, llevándose una contabilidad paralela de índice para arrays de anotaciones (indx). Se controla mediante variables auxiliares Dstr-Dend y Ustr-Uend los índices válidos para transmitir un array *double* o *uint8* de las dimensiones correspondientes.

El código MPITB para el test sobre mensajes *double*s se muestra en el Listado 4.10:

```
if indx>=Dstr & indx<=Dend                           % hacer parte double

  NTIMES = ntimes*reps(Alto*Ancho*sizeofDbl);        % medir reserva memoria
  T=clock; for i=1:NTIMES
    clear arrayDouble
    arrayDouble=ones(Alto,Ancho);                    % liberar al final
  end, T=etime(clock,T); s=whos('arrayDouble');
  d(indx,Size)=s.bytes ;
  d(indx, Mtm)=T/NTIMES;

  T=clock; for i=1:NTIMES                            % medir cambio de tipo
    arrayUint8=uint8(arrayDouble);
    clear arrayUint8
  end, T=etime(clock,T);
  d(indx, Ctm)=T/NTIMES;

  MPI_Barrier(NEWORLD); T=clock; for i=1:NTIMES      % ping-pong parte double
    MPI_Send(arrayDouble,1,TAG,NEWORLD);
    MPI_Recv(arrayDouble,1,TAG,NEWORLD);
  end, T=etime(clock,T);
  d(indx,TXtm)=T/2/NTIMES;

  clear arrayDouble
end
```

Listado 4.10: Programa *ping-pong* para MPITB (continuación). Sección dedicada al tiempo de transmisión de *double*s.

Al igual que bajo PVMTB, además del tiempo de transmisión se mide previamente el tiempo de reserva de memoria (primer bucle) y de cambio de tipo (segundo bucle) bajo MATLAB. Naturalmente, la sección de código dedicada a datos *uint8* no realiza el bucle de cambio de tipo.

Las Figuras de la 4.6 a la 4.9 muestran las mediciones correspondientes a la ejecución de este código MATLAB. Las dos primeras corresponden al modo *daemon* LAM, con o sin la opción de cluster homogéneo. Las dos siguientes añaden la opción cliente a cliente.

En color morado aparece un nuevo trazo mostrando el coste de la conversión de tipo *double*→*uint8*. Revisando el conjunto de figuras se observa el mismo comportamiento del gestor de memoria MATLAB que ya se destacó en el capítulo anterior bajo PVMTB (trazos azul y verde oscuros). Se obtiene el mismo mínimo alrededor de 0.1ms y la misma progresión a partir de decenas de KBs, mientras que la conversión de tipo consume un tiempo un orden de magnitud por encima de la reserva de memoria a partir de 3MTUs, comportamientos éstos que se han reproducido en todas las mediciones MATLAB, tanto las de PVMTB (Figuras de la 3.8 a la 3.13) como las de MPITB (Figuras de la 4.6 a la 4.9).

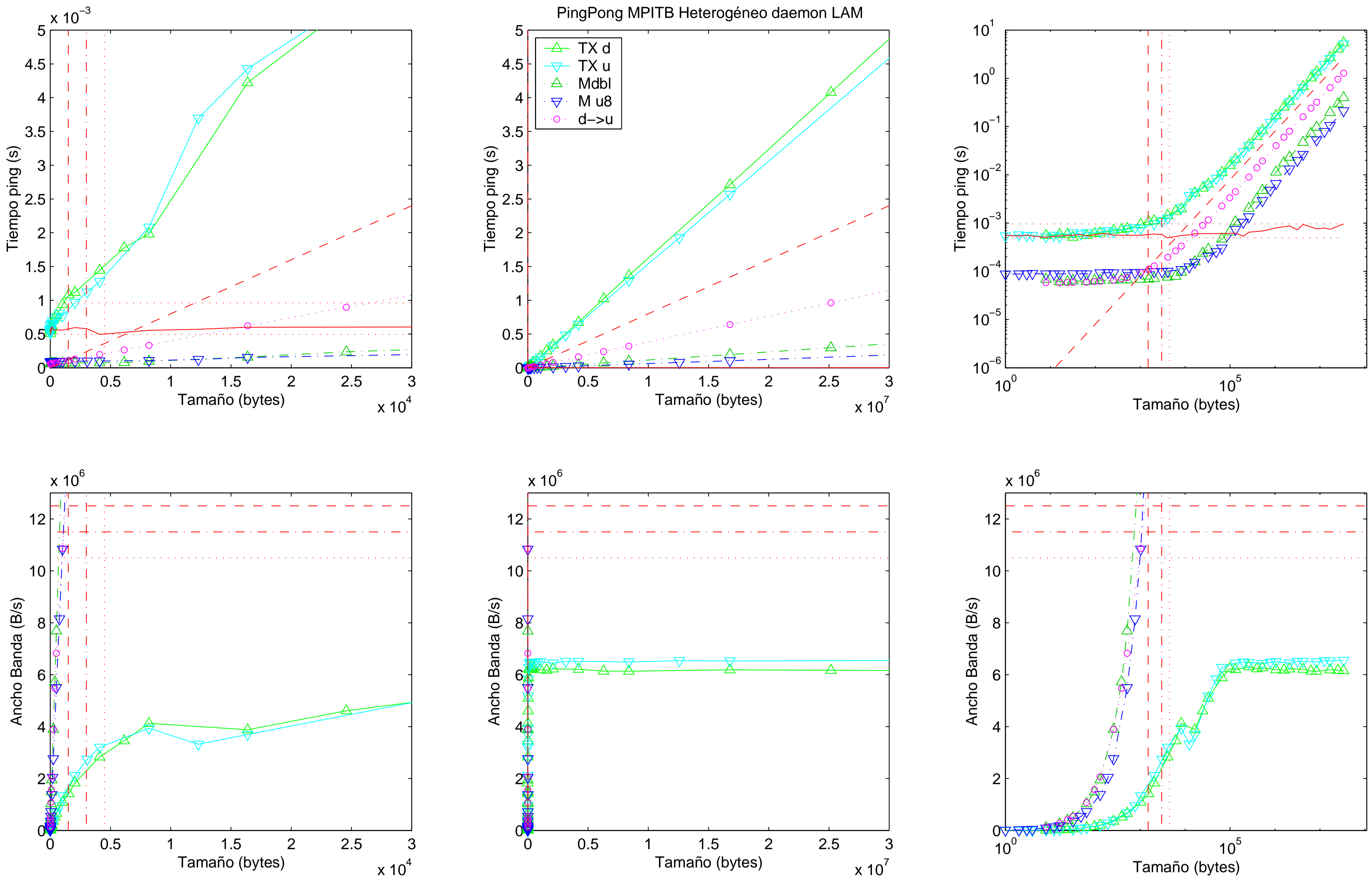


Figura 4.6: Test *ping-pong* bajo MPITB, sin opciones −O −c2c. **Derecha**: gráfica logarítmica incluyendo todas las mediciones. **Izquierda**: detalles con abscisa lineal en los rangos de decenas de KB y MB. **Abajo**: anchos de banda correspondientes. **Leyenda**: **TX d**: tiempo de transmisión (ida) para array *double*. **TX u**: para array *uint8*. **Mdbl**: tiempo para reserva de memoria de array *double*. **M u8**: para array *uint8*. **d→u**: tiempo de conversión *double*→*uint8*.

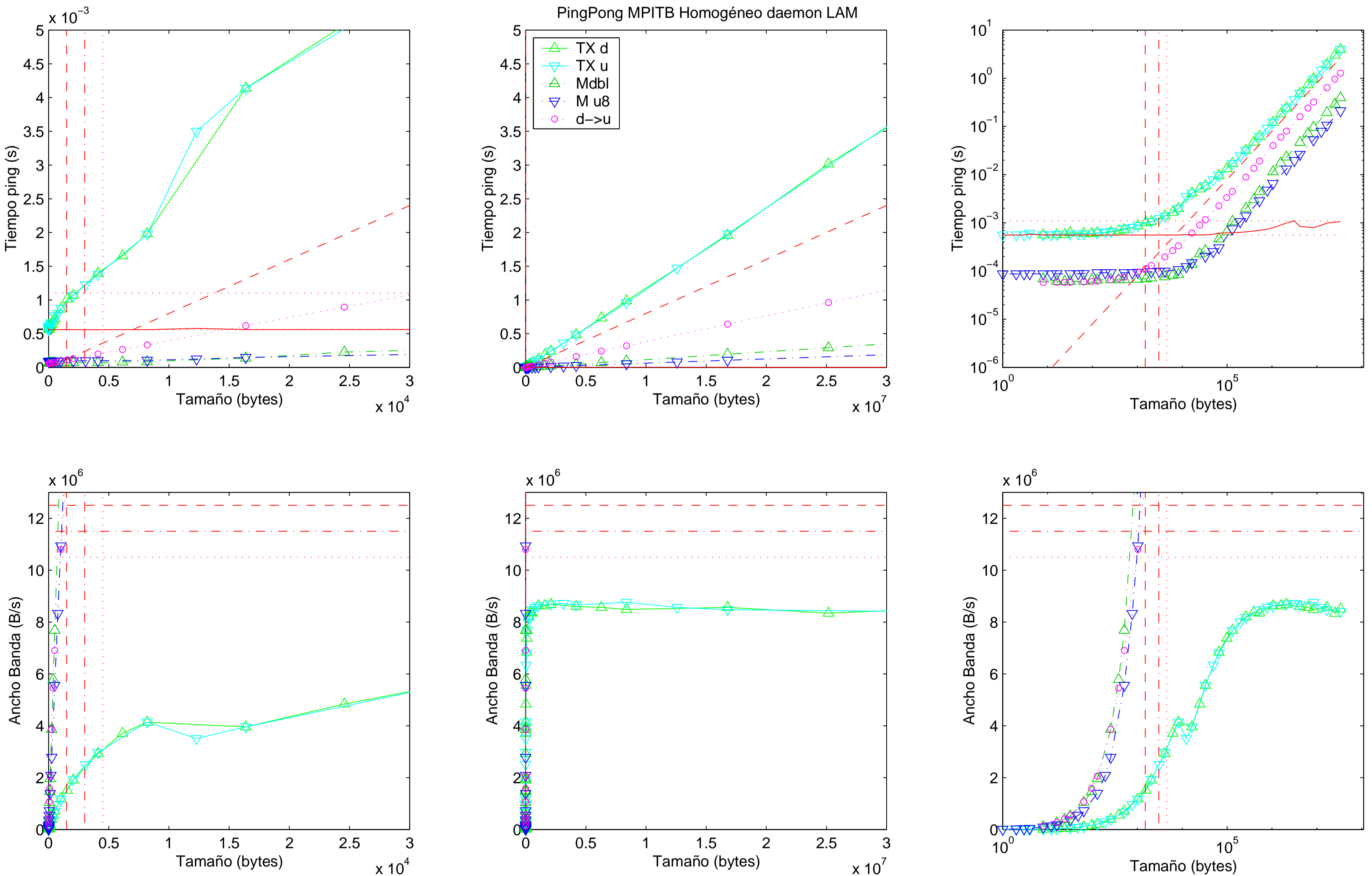


Figura 4.7: Test *ping-pong* bajo MPITB, con opciones −O −lamd. Consultar pie de Figura 4.6 para más detalles.

Con MPITB en modo *daemon* (Figuras 4.6 y 4.7) el ancho de banda crece de 6–6.5MB/s (6 para *double*s) a 8.5MB/s al añadir la opción homogénea –O. Al igual que bajo MPI, la opción homogénea también mejora el ancho de banda para datos *uint8*. La opción PvmDataInplace sólo presentaba este comportamiento con ruta directa.

Comparando con MPI en modo *daemon* (Figuras 4.1 y 4.2) se puede comprobar que el *overhead* de MPITB es mínimo, apenas discernible mediante inspección visual de las gráficas.

Lo que sí es notable por inspección visual son los característicos picos en la estimación de la latencia durante los últimos tamaños del barrido (gráficas superiores derecha) también obtenidos con PVMTB en modo *daemon*. Este efecto desaparece al utilizar ruta TCP directa en las siguientes dos Figuras (4.8 y 4.9). Al igual que con MPI bajo C, los anchos de banda aumentan.

Al añadir la ruta directa a MPITB, aumentan tanto los anchos de banda como la diferencia entre *uint8* y *double* con codificación XDR, de 6–6.5MB/s en la Figura 4.6 abajo derecha a 8–9MB/s en la Figura 4.8.

Con cluster homogéneo se pasa de 8.5MB/s en la Figura 4.7 a 11.75MB/s en la Figura 4.9. Las cantidades y conclusiones mencionadas son muy similares a las del apartado anterior bajo MPI, como queda resumido en la Tabla 4.3.

	BW (MB/s)		Latencia (μs)	
	uint8	**double**	media	pico
_-lamd	6.6	6.3–6.2	630	962
O-lamd	8.6–8.3		656	1102
_-c2c	9.0–8.7	8.4–8.0	214	
O-c2c	11.75		211	

Tabla 4.3: Detalles remarcables del test *ping-pong* en MPITB.

En la Figura 4.10 se presenta la habitual gráfica adicional comparando los tiempos de transmisión de datos *double* bajo las distintas combinaciones de opciones MPI. La Figura 4.10(b) muestra la ampliación de las primeras 3 MTUs, en donde se puede observar una latencia de 0.2ms para ruta directa y 0.6ms a través del *daemon* LAM, en concordancia con la Tabla 4.3. Con MPI se obtenían 0.1–0.4ms (Figura 4.5(b), Tabla 4.2), con PVMTB 0.4–0.6ms (Figura 3.14(b), Tabla 3.3), y con PVM 0.2–0.4ms (Figura 3.7(b), Tabla 3.3).

Más adelante, en la Tabla 4.4 derecha, se presentan algunos puntos de la mejor opción MPITB (–O –c2c) con datos *double*, al objeto de posibilitar una comparación objetiva con MPI. Ahora es sin embargo el momento apropiado para presentar la prometida Figura 4.11, que no es más que una ampliación de las respectivas Figuras 4.5 y 4.10 en el rango de centenares de KBs, donde es posible apreciar que a partir de unos 200KB la opción de cluster homogéneo afecta más a las prestaciones del paso de mensajes que la opción cliente a cliente.

4.5 Comparación

La misma comparación realizada en el capítulo anterior entre PVM y PVMTB se realiza ahora entre MPI y MPITB. Recordemos que la información se presentaba de forma tabulada al ser ésta más apropiada para una evaluación objetiva del *overhead* introducido por la *Toolbox*.

La Tabla 4.4 se presenta con la misma estructura que en aquella ocasión: a la izquierda se presentan los datos relativos a MPI, a la derecha las mediciones bajo MPITB, y en el centro

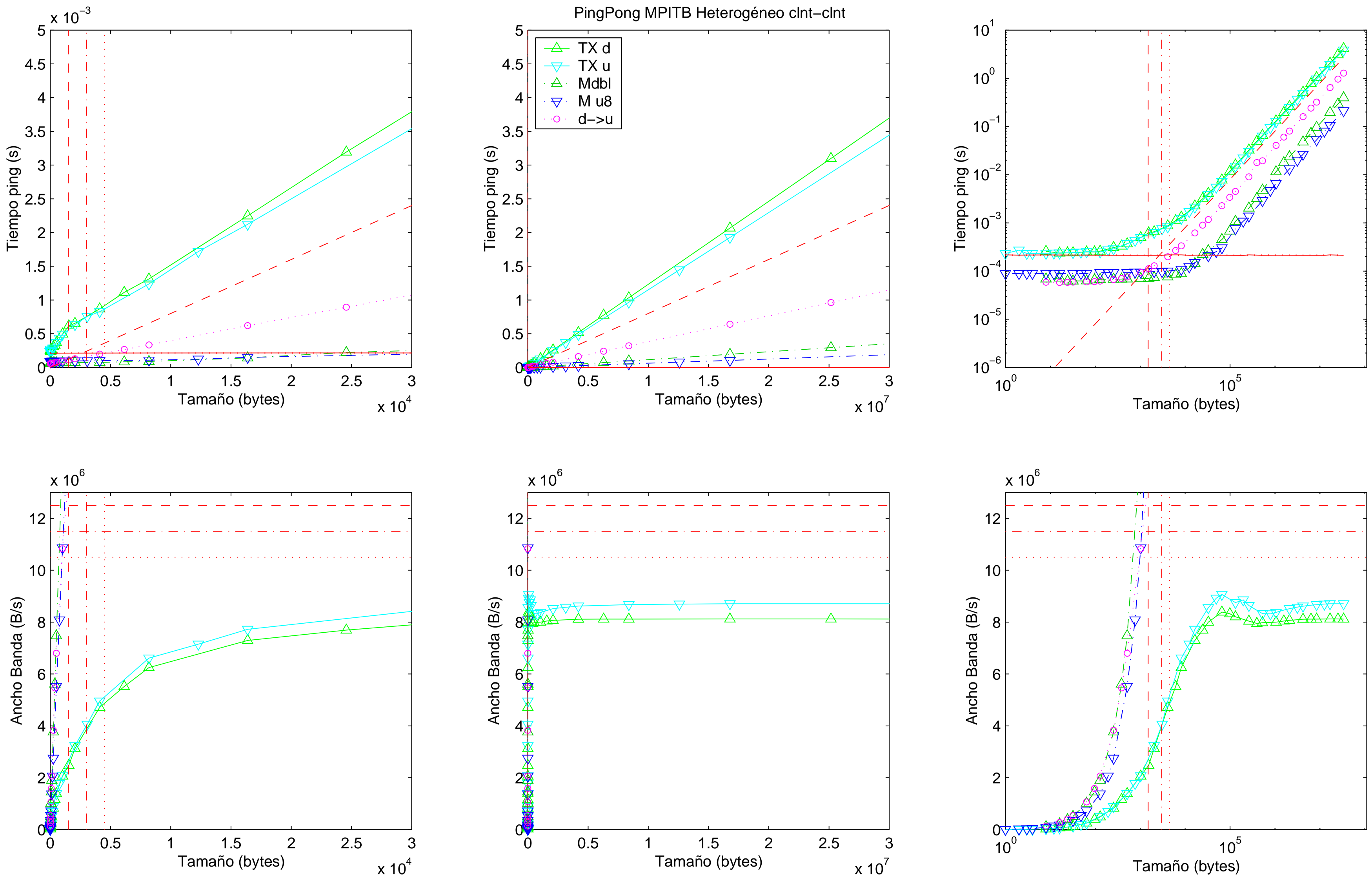


Figura 4.8: Test *ping-pong* bajo MPITB, con la opción −c2c. Consultar pie de Figura 4.6 para más detalles.

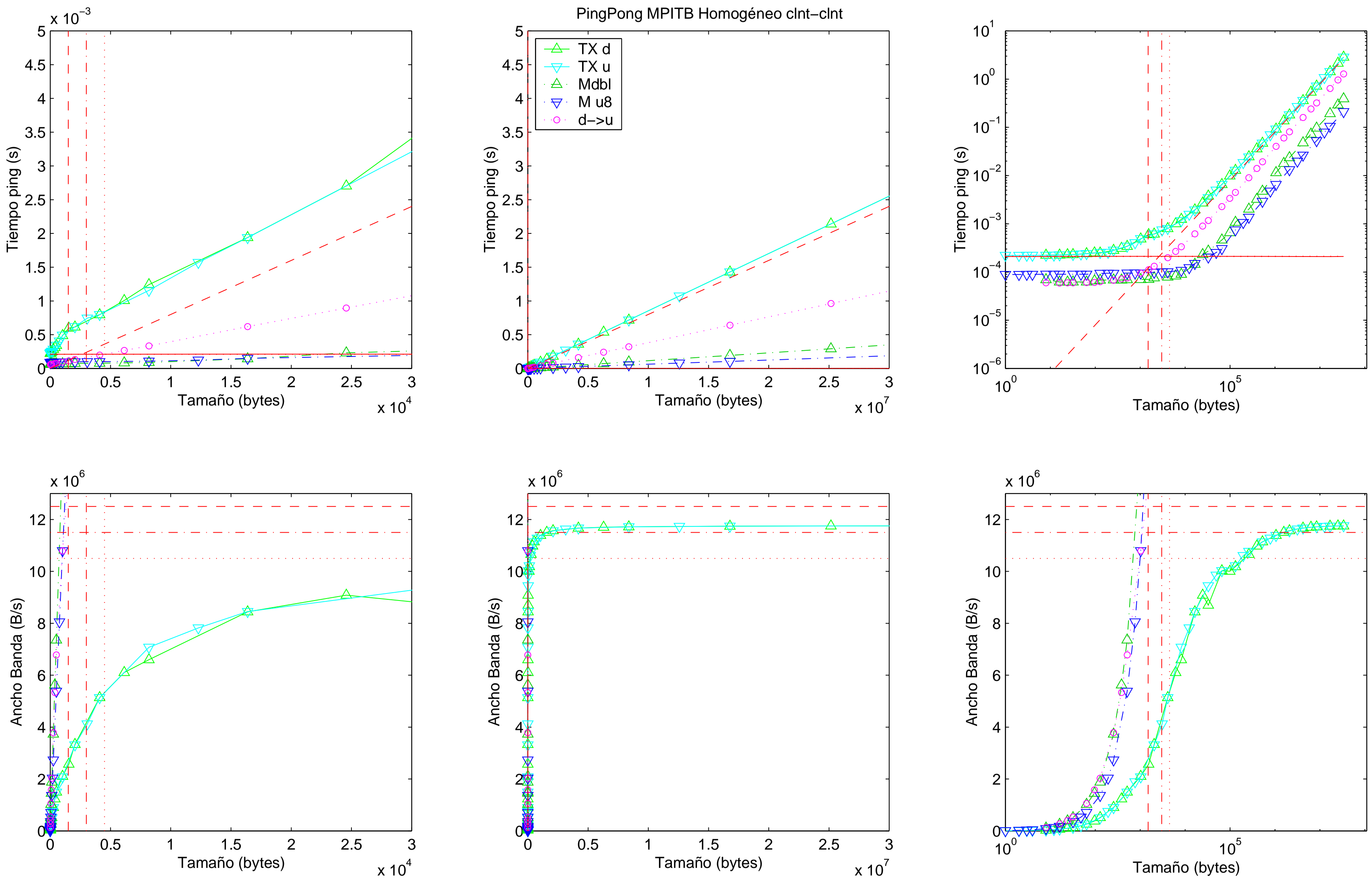
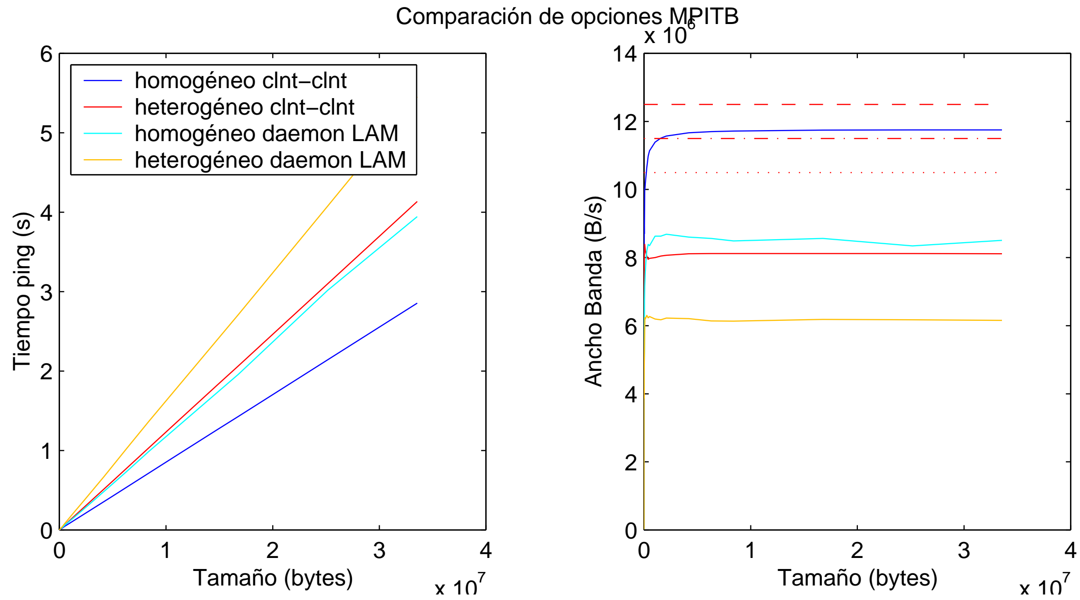
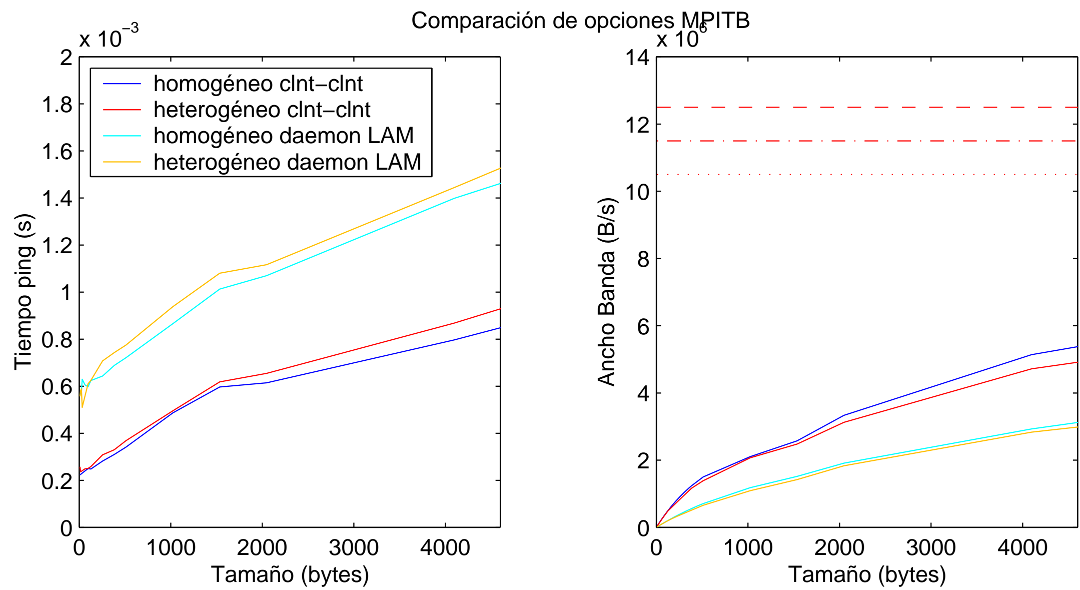


Figura 4.9: Test *ping-pong* bajo MPITB, con opciones −O −c2c. Consultar pie de Figura 4.6 para más detalles.

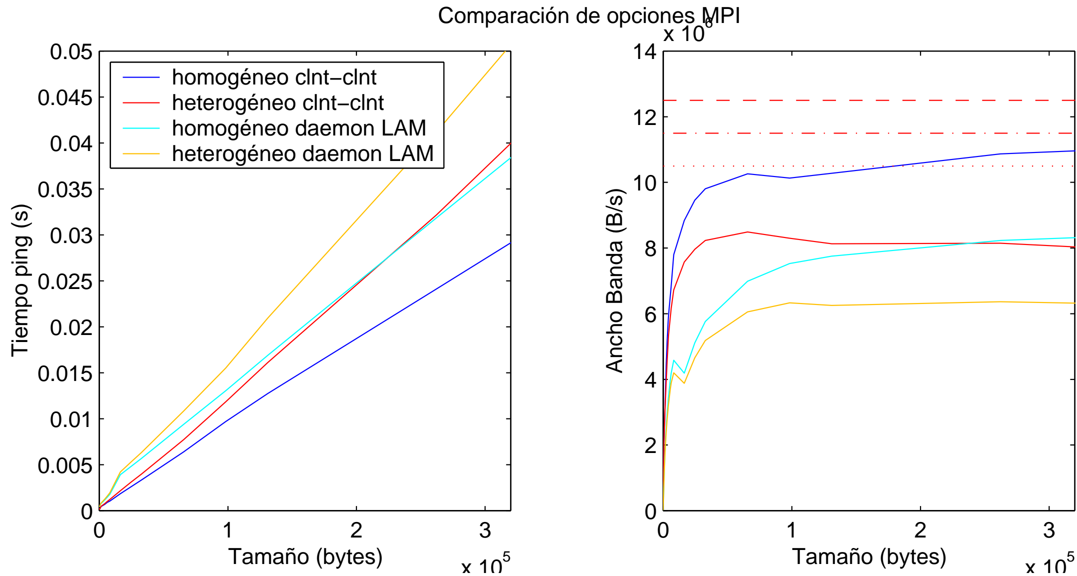


(a) Barrido completo.

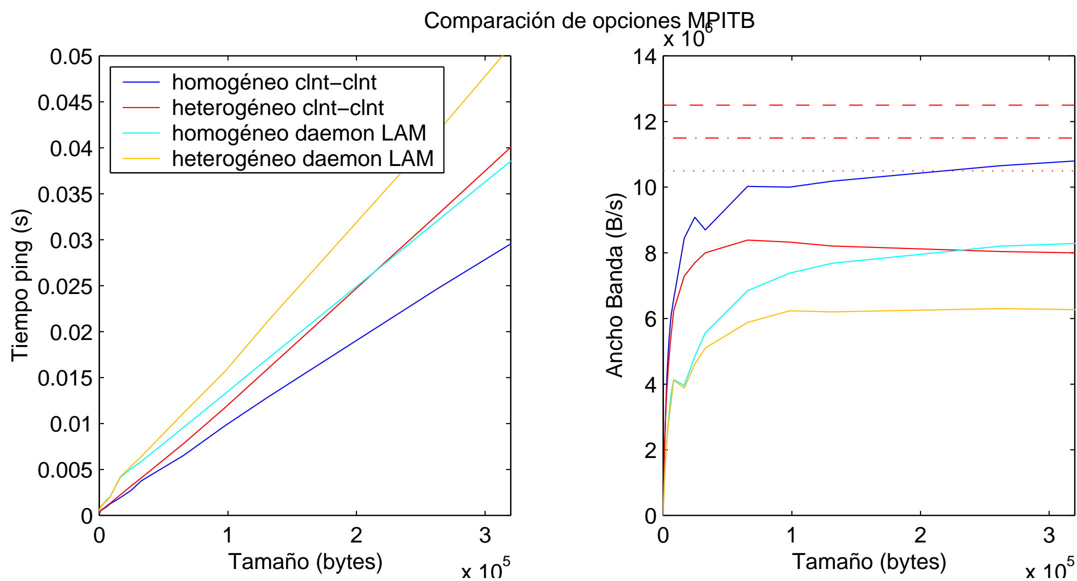


(b) Detalle de las primeras 3 MTUs.

Figura 4.10: Test *ping-pong* bajo MPITB, comparación de opciones con datos *double*.



(a) Detalle de la Figura 4.5 hasta 320KB.



(b) Detalle de la Figura 4.10 hasta 320KB.

Figura 4.11: Tanto con MPI como con MPITB, a partir de unos 200KB la influencia de la opción −O se vuelve más importante que la de −c2c en las prestaciones del paso de mensajes.

se muestra el tamaño de mensaje considerado. Cada fila recoge la información asociada a un tamaño de mensaje. Los tamaños se dan en orden creciente, como es habitual.

Tanto para MPI como para MPITB, cada fila de la tabla proporciona:

S: el tamaño de mensaje considerado en la fila (bytes).

M: el tiempo requerido para ubicar un bloque de memoria del tamaño requerido (ms).

T: el tiempo de transmisión (*ping*) para ese tamaño (ms). Recordemos que este tiempo incluye también la recepción del mensaje por el proceso *eco*.

M (ms)	T ping (ms)	B lineal	B afín	**S (bytes)**	M (ms)	T ping (ms)	B lineal	B afín
	0.123	(MB/s)	(MB/s)	**0**		0.211	(MB/s)	(MB/s)
0.001	0.124	0.065	19.048	**8**	0.069	0.223	0.036	0.621
0.001	0.123	0.130	-23.188	**16**	0.062	0.225	0.071	1.121
0.001	0.123	0.196	-37.325	**24**	0.062	0.229	0.105	1.331
0.001	0.124	0.257	33.368	**32**	0.062	0.230	0.139	1.631
0.001	0.125	0.512	42.300	**64**	0.062	0.240	0.267	2.176
0.001	0.138	0.695	6.486	**96**	0.066	0.250	0.384	2.453
0.002	0.150	0.855	4.859	**128**	0.068	0.248	0.516	3.424
0.002	0.182	1.408	4.377	**256**	0.069	0.282	0.907	3.578
0.002	0.217	1.770	4.103	**384**	0.068	0.310	1.237	3.847
0.002	0.247	2.073	4.141	**512**	0.070	0.341	1.501	3.925
0.002	0.368	2.779	4.179	**1024**	0.070	0.487	2.103	3.707
0.004	0.493	3.116	4.157	**1536**	0.071	0.597	2.574	3.979
0.004	0.495	4.133	5.504	**2048**	0.076	0.615	3.333	5.071
0.005	0.687	5.966	7.273	**4096**	0.078	0.797	5.141	6.989
0.008	0.899	6.834	7.921	**6144**	0.083	1.006	6.107	7.723
0.007	1.050	7.801	8.840	**8192**	0.090	1.241	6.600	7.949
0.019	1.854	8.837	9.466	**16384**	0.145	1.940	8.444	9.472
0.076	2.599	9.457	9.929	**24576**	0.231	2.705	9.086	9.853
0.101	3.342	9.805	10.181	**32768**	0.272	3.767	8.699	9.215
0.194	6.386	10.262	10.464	**65536**	0.464	6.536	10.027	10.361
0.293	9.702	10.132	10.263	**98304**	0.655	9.826	10.004	10.224
0.029	12.752	10.279	10.379	**131072**	1.077	12.871	10.184	10.353
0.026	24.126	10.866	10.921	**262144**	2.016	24.603	10.655	10.747
0.032	35.499	11.077	11.115	**393216**	3.235	35.814	10.979	11.044
0.032	46.816	11.199	11.229	**524288**	4.738	47.065	11.140	11.190
0.033	91.708	11.434	11.449	**1048576**	11.535	92.071	11.389	11.415
0.029	136.353	11.535	11.546	**1572864**	17.865	136.678	11.508	11.526
0.033	180.797	11.600	11.607	**2097152**	23.613	181.243	11.571	11.584
0.053	359.005	11.683	11.687	**4194304**	47.827	359.463	11.668	11.675
0.061	537.191	11.712	11.714	**6291456**	72.718	537.650	11.702	11.706
0.063	715.397	11.726	11.728	**8388608**	96.418	715.826	11.719	11.722
0.058	1428.046	11.748	11.749	**16777216**	193.190	1428.721	11.743	11.745
0.058	2140.955	11.754	11.755	**25165824**	291.950	2141.437	11.752	11.753
0.101	2853.849	11.758	11.758	**33554432**	392.128	2854.786	11.754	11.755

(a) Datos para MPI. (b) Tam (c) Datos para MPITB.

Tabla 4.4: Resultados del test *ping-pong* para datos **double** con opciones –O/–c2c. Los datos han sido exportados desde la hoja de cálculo con tres cifras decimales.

B lineal: Ancho de banda lineal, despejado del modelo $T = S/B$. Para tamaño $S = 0$, al tiempo de *ping* lo denominamos Latencia.

B afín: Ancho de banda despejado del modelo $T = L + S/B$.

Se puede consultar el Apartado 3.5 en donde se presentaron estas cantidades con las explicaciones correspondientes.

La comparación se establece dividiendo las respectivas columnas de ambas subtablas. En la Tabla 4.5 se muestran los cocientes MPITB/MPI para los tiempos de transmisión y reserva de memoria, y los cocientes inversos MPI/MPITB para los anchos de banda, al objeto de que las cantidades conserven la interpretación intuitiva de "cuánto más lento" es MPITB que MPI.

En particular, se debe notar que los tiempos de reserva de memoria dependen del gestor de memoria MATLAB, no de MPITB. Se pueden repetir por tanto las conclusiones del capítulo

S (bytes)	M	T ping	B lineal	B afín
0		1.707		
8	100.972	1.805	1.805	30.669
16	113.982	1.833	1.833	-20.694
24	71.948	1.863	1.863	-28.039
32	42.767	1.852	1.852	20.462
64	50.496	1.922	1.922	19.440
96	52.133	1.807	1.807	2.644
128	38.159	1.656	1.656	1.419
256	35.953	1.551	1.551	1.223
384	34.647	1.431	1.431	1.066
512	34.232	1.381	1.381	1.055
1024	29.837	1.321	1.321	1.127
1536	17.636	1.210	1.210	1.045
2048	20.250	1.240	1.240	1.085
4096	16.957	1.160	1.160	1.041
6144	10.544	1.119	1.119	1.026
8192	13.416	1.182	1.182	1.112
16384	7.582	1.046	1.046	0.999
24576	3.015	1.041	1.041	1.008
32768	2.697	1.127	1.127	1.105
65536	2.393	1.023	1.023	1.010
98304	2.236	1.013	1.013	1.004
131072	36.834	1.009	1.009	1.003
262144	76.786	1.020	1.020	1.016
393216	99.561	1.009	1.009	1.006
524288	149.239	1.005	1.005	1.003
1048576	346.894	1.004	1.004	1.003
1572864	605.622	1.002	1.002	1.002
2097152	715.214	1.002	1.002	1.002
4194304	910.860	1.001	1.001	1.001
6291456	1191.504	1.001	1.001	1.001
8388608	1542.483	1.001	1.001	1.000
16777216	3331.954	1.000	1.000	1.000
25165824	5037.268	1.000	1.000	1.000
33554432	3883.147	1.000	1.000	1.000

Tabla 4.5: Cocientes MPITB/MPI. Los anchos de banda son MPI/MPITB.

anterior, en donde se destacaba que bajo C el tiempo de reserva de memoria se estabiliza a unos 40μs mostrando un pico en las decenas de KBs, mientras que en MATLAB se emplea un mínimo de 60μs y crece con el tamaño del array. Se observa el mismo pico de alrededor de 290μs que vimos en el capítulo anterior (Tabla 3.4). Queda por tanto confirmado que los picos de hasta 0.01s en reserva de memoria observados con las restantes combinaciones de opciones PVM fueron un artificio provocado por la necesidad de liberar los buffers de envío y recepción bajo PVM para evitar *swapping* con mensajes de tamaño 32MB. Tanto con MPI como con la opción DataInPlace–RouteDirect de PVM desaparece dicho efecto. Los cocientes en la columna etiquetada **M** de la Tabla 4.5 son igual de descriptivos que en aquella ocasión acerca del *overhead* del gestor de memoria MATLAB.

Particularmente, la Tabla 4.4 refleja unos tiempos de 60–100μs para las últimas reservas de memoria (24–32MB) en la medición MPI, también observables en la correspondiente Figura 4.4. Comparando con los tiempos de reserva medidos bajo otras opciones en las Figuras de la 4.1 a la 4.3, se comprueba que cabe esperar dicha variabilidad en el comportamiento del gestor de memoria Linux.

El tiempo de transmisión en MPITB tiene un comportamiento igual de impecable que bajo PVMTB. Recordemos que el *overhead* de PVMTB bajaba rápidamente desde 1.85 hasta 1.2 en las primeras 3MTUs, y pronto se estabiliza alrededor de la unidad en el rango de decenas de KBs, obteniéndose un *overhead* inferior al 1% a partir del centenar de KBs. El *overhead* de MPITB es muy similar, bajando de 1.8 a 1.15 en las 3 primeras MTUs e inferior al 1% a partir del centenar de KBs.

La comparación entre anchos de banda lineales y afines arroja el mismo resultado que bajo PVMTB, a saber, que a partir de decenas de KBs la diferencia es del orden de centésimas de punto, y a partir de centenares de KBs es de milésimas de punto. Por encima de 1MB ambos anchos de banda son idénticos. De nuevo, sólo merece la pena distinguir entre ambos para tamaños de mensaje inferiores a decenas de KBs.

Resumidamente, MPITB no presenta *overhead* significativo sobre MPI para mensajes a partir de centenares de KBs; para decenas de KBs ronda el 5–10%, en el rango de KBs crece del 10% al 30%, y para tamaños menores puede llegar al 90%. La latencia de MPITB es 1.7 veces superior a la de MPI.

4.6 Comparación entre PVMTB y MPITB

Al objeto de servir como material de referencia, y para disponer en forma centralizada de toda la información acumulada, se repiten a continuación las Tablas 3.2, 3.3, 4.2 y 4.3, en donde se resumían los detalles más remarcables de los tests *ping-pong* de los dos últimos capítulos.

No se insiste en la información ya comentada. Las tablas de los tests en lenguaje C han sido simplificados eliminando las columnas relacionadas con la reserva de memoria, reflejándose sólo las magnitudes relacionadas con el paso de mensajes.

Las tablas se han dispuesto en dos filas de dos columnas. La primera fila corresponde a los tests del capítulo anterior y la segunda a los de éste. La primera columna contiene los tests C y la segunda los tests MATLAB.

Con esta organización, los datos aparecen en el orden en que han sido presentados previa-

	BW (MB/s)		Latencia
	char	**double**	(μs)
def-no	4.2– 4.1	3.1–3.0	359
raw-no	4.2– 4.1		356
pla-no	4.3– 4.1		356
def-di	9.6– 9.2	5.3–5.0	192
raw-di	9.6– 9.2		187
pla-di	10.5–10.4		185

(a) PVM

	BW (MB/s)		Latencia (μs)	
	uint8	**double**	media	pico
def-no	4.2– 4.1	3.1–3.0	592	1153
raw-no	4.1		615	1126
pla-no	4.1		598	1128
def-di	9.5– 9.2	5.2–5.0	347	
raw-di	9.5– 9.2		343	
pla-di	10.5–10.4		343	

(b) PVMTB

	BW (MB/s)		Latencia
	char	**double**	(μs)
-lamd	7.0	6.4–6.2	399
O-lamd	8.6–8.3		401
-c2c	9.3	8.5–7.9	124
O-c2c	11.75		123

(c) MPI

	BW (MB/s)		Latencia (μs)	
	uint8	**double**	media	pico
-lamd	6.6	6.3–6.2	630	962
O-lamd	8.6–8.3		656	1102
-c2c	9.0–8.7	8.4–8.0	214	
O-c2c	11.75		211	

(d) MPITB

Tabla 4.6: Detalles remarcables de los tests *ping-pong* de los Capítulos 3 y 4.

mente, permitiendo una fácil comparación horizontal para observar el escaso *overhead* de las *Toolboxes*, y vertical para estudiar las diferentes prestaciones bajo ambos sistemas de paso de mensajes.

4.7 Conclusiones

Las mismas conclusiones obtenidas anteriormente para PVMTB se mantienen ahora para MPITB:

Complitud: Se ha implementado un interfaz completo con MPI, respetando los patrones de llamada de las rutinas MPI originales, salvo cuando esto entra en conflicto con el propio entorno MATLAB. Para tratar dicho caso se han mantenido los mismos criterios razonados que se observaron en el diseño de PVMTB.

Se ha resuelto hasta donde es posible el conflicto producido por la *opacidad* de los tipos de datos MATLAB, estableciendo una correspondencia entre tipos MPI básicos y los tipos MATLAB compatibles. *The MathWorks* no proporciona información suficiente al programador de aplicaciones sobre sus tipos estructurados (mxCELL_CLASS, mxSTRUCT_CLASS, etc), siendo obligatorio transmitir las variables de dichos tipos por el procedimiento de empaquetar toda la información necesaria para su reconstrucción. Los comandos MPITB MPI_Pack y MPI_Unpack realizan las operaciones de empaquetamiento (en envío) y reconstrucción (en recepción), respectivamente.

Otras *Toolboxes* implementan sólo un reducidísimo subconjunto de llamadas MPI: MultiMATLAB se queda en unas 20, mientras que PMI sólo implementa 8. MultiMATLAB ofrece también comandos de alto nivel que utilizan internamente las llamadas MPI, con el consiguiente paso intermedio de interpretación y pérdida de interfaz directo con las llamadas

MPI involucradas.

Enlace dinámico: El interfaz se ha realizado sobre una versión dinámicamente enlazada de la biblioteca MPI. Se consigue con ello ahorro de espacio en disco, con el beneficio adicional de que múltiples usuarios pueden compartir el código de la biblioteca una vez cargado en memoria.

Otras *Toolboxes* optan por complicar el diseño de la interfaz usando un fichero MEX "distribuidor" o "directorio" que lleva estáticamente enlazada toda la biblioteca MPI, y una serie de ficheros M intermedios que se van llamando unos a otros. Todos los comandos de la *Toolbox* terminan pasando el control a dicha rutina, identificando mediante un argumento adicional la llamada que desean invocar. Esto provoca un *overhead* en el tiempo de ejecución de los comandos de la *Toolbox* así como un gasto adicional de espacio en disco y memoria.

Reutilización de código: El método de sustitución de macros **#define** es de vital importancia para simplificar el interfaz con MPI, habiéndose recurrido a un sistema de sustitución a varios niveles para evitar implementar varias veces la misma funcionalidad. Esto no fue necesario con PVMTB debido a la gran regularidad de sus patrones de llamada.

El sistema de sustitución seguido se estructura en cuatro conceptos básicos:

Rutinas de traducción: Soportan la representación de objetos MPI en el entorno MATLAB.

Categorías: Clases en que se segregan las llamadas MPITB según su funcionalidad. Esta es una clasificación inicial, que se refina posteriormente en *grupos* de comandos que siguen el mismo patrón de llamada.

Patrones: Secuencia reutilizable de tratamiento de argumentos y retorno de valores. Se construyen a partir de bloques.

Bloques constructivos: Funcionalidad de alta reutilización, común a varios patrones de llamada, que se extrae por tanto de los patrones que la requieren, al objeto de codificarla una única vez.

De la aplicación de estos conceptos durante el análisis de los comandos MPITB emerge una clasificación en 10 categorías, cada una con su correspondiente fichero **#include**:

MPI: Rutinas y bloques comunes a toda la MPITB.

SndRecv: Comunicación punto a punto.

TstWait: Grupos Test, Wait y Start.

Topo: Topologías de procesos.

Prob: Grupo [I]Probe.

Info: Arranque dinámico de procesos.

Err: Gestores de error.

Attr: Manipulación de atributos y *wrappers* asociados.

Coll: Operaciones colectivas.

Grp: Operaciones con grupos MPI.

El estricto seguimiento de esta normativa no sólo ha reducido el esfuerzo de diseño, depuración y mantenimiento de la *Toolbox*, además de facilitar enormemente su estudio y comprensión. En realidad, ha *hecho posible* la existencia de MPITB. Basta comparar el Makefile o el código fuente de los ficheros MEX de esta *Toolbox* con los de cualquier *Toolbox* “similar” para estimar la importancia de este elaborado sistema de reutilización. Una lectura del extenso código de los ficheros de categoría h<CAT>.h hace comprensible la inviabilidad de implementar uno a uno los 153 ficheros de interfaz presentes en MPITB. El esfuerzo de depuración sería prohibitivo. Se comprende que ningún trabajo anterior haya llegado a recubrir un número significativo de llamadas MPI.

Rápido prototipado: Al igual que PVMTB, MPITB permite desarrollar aplicaciones HPC, basadas en este caso en MPI, como ficheros M del entorno de rápido prototipado MATLAB. El usuario sólo debe recordar arrancar previamente LAM/MPI usando el comando lamboot, pudiendo olvidarse entonces de la tediosa redacción de Makefiles, de las opciones del compilador gcc, etc. El usuario puede concentrar su esfuerzo en el prototipo a desarrollar, teniendo a su disposición el arsenal de herramientas de depuración, perfilado, generación de GUIs, gráficos y visualización 3D disponible bajo MATLAB.

Hay que tener en cuenta (ver por ejemplo [14], [39]) que la mayor parte del tiempo de desarrollo de un prototipo de cierta complejidad en un lenguaje compilado se invierte en el diseño del interfaz (posiblemente gráfico) con el usuario y la E/S en general, incluyendo el manejo de ficheros y directorios, visualización de los resultados, gráficos, imágenes, etc.

Esta facilitación, junto con la mínima modificación de los patrones de las llamadas MPI originales, hacen de MPITB una herramienta ideal para docencia del estándar de paso de mensajes MPI.

Pérdida de prestaciones mínima: El *overhead* de MPITB es también inapreciable a partir de decenas de KBs (Tabla 4.5). Esto la convierte en una herramienta ideal para investigación, tanto por el ahorro en esfuerzo de diseño de aplicaciones HPC como por la aceptable pérdida de prestaciones que causa (comparando con el uso directo del sistema LAM/MPI bajo lenguaje C), especialmente para mensajes a partir de la decena de KBs.

Una despreocupada programación bajo MATLAB, incurriendo por ejemplo en copias de memoria o cambios de tipo no imprescindibles, provoca mayor pérdida de prestaciones que las manifestadas en la comparación de MPITB con MPI bajo C.

Cabe manifestar a este respecto el hecho de que tanto MultiMATLAB como PMI incurren en una copia de memoria adicional en el mecanismo global de paso de mensajes. Las variables recibidas en los mensajes se copian a zonas de memoria reservadas explícitamente por estas *Toolboxes*, usando la llamada API MATLAB mxCreateNumericArray en el caso de MultiMATLAB, o engGetArray en el caso de PMI, implicando por tanto al gestor de memoria MATLAB con su elevado *overhead*.

Opciones de homogeneidad y ruta cliente a cliente: Se ha demostrado la importancia de ambas opciones –O y –c2c en la reducción de la latencia y maximización del ancho de banda, evitando la ruta normal a través del *daemon* LAM y la codificación XDR de los mensajes. Se ha detectado la inversión a partir de 200KB de la importancia relativa de ambas opciones,

siendo más importante para tamaños mayores de 200KB la opción de cluster homogéneo que la ruta directa entre tareas. Naturalmente, esta opción sólo puede usarse cuando todos los computadores implicados tengan una representación de datos homogénea.

Capítulo 5

Ejemplo de Aplicación: Análisis Wavelet para Sistemas de Visión

Resumen del capítulo

En este capítulo se presenta un ejemplo de aplicación basado en el análisis *wavelet* de imágenes. Esta técnica se ha establecido en la última década como un poderoso método de análisis de señal, siendo su incorporación al estándar JPEG 2000 sólo una constatación de su capacidad para comprimir imágenes con una ínfima pérdida de calidad. Las *wavelets* son aplicables no obstante a multitud de problemas, y a diferencia de otras técnicas tiene una rigurosa fundamentación matemática, contando ya con una profusa bibliografía de orientación tanto tecnológica como matemática ([88, 10]) para ayudar al usuario a escoger la *wavelet* madre de la que dependerán las características del análisis realizado.

En el Apartado 5.1 se hace una brevísima introducción a la técnica de análisis *wavelet*, concentrándose particularmente en el artículo seminal de Mallat [46] en el que se inspira la aplicación presentada en el Apartado 5.2. Msr. Mallat trabaja actualmente en temas relacionados con Sistemas de Visión y Visión por Computador (flujo óptico, estimación de forma basada en textura, etc. [44]).

Tras estudiar el código secuencial MATLAB de referencia y mostrar los resultados del análisis *wavelet* y su tiempo de ejecución, se estudian dos posibles formulaciones del programa paralelo con distintas granularidades, al objeto de estimar las ganancias en velocidad que cabe esperar del uso de nuestras *Toolboxes* paralelas. Ambas aproximaciones se han programado tanto en PVMTB como en MPITB.

La primera propuesta es una variante paralela muy sencilla del algoritmo secuencial, exhibiendo dependencias entre los procesos involucrados. Puede por tanto aplicarse a imágenes en niveles de gris ([43]) como las utilizadas por Mallat en su artículo. Este algoritmo paralelo se estudia con detalle en el Apartado 5.3, dedicándose el siguiente Apartado 5.4 a la segunda propuesta, de mayor granularidad, en la que el propio algoritmo secuencial originalmente propuesto por Mallat se aplica separadamente a cada uno de los canales (rojo, verde, azul) de la versión en color de la misma imagen. Este algoritmo, embarazosamente paralelo, obtiene naturalmente un *speedup* casi lineal.

En el Apartado 5.5 se discuten los resultados obtenidos con cada propuesta bajo ambas *Toolboxes*, PVMTB y MPITB. Las conclusiones del estudio se resumen en el Apartado 5.6.

5.1 Análisis *wavelet* y su relación con Sistemas de Visión

En su artículo seminal, Mallat [46] encontró una conexión entre el trabajo intuitivamente desarrollado en el campo de Visión por Computador respecto a Pirámides Laplacianas (Burt y Adelson [8], Marr [49]) y los esfuerzos matemáticos (Meyer [60]) por encontrar una base de $\mathcal{L}^2(\mathcal{R})$, las funciones de energía finita. La conexión se realizó a lo largo de tres días en los que Msr. Yves Meyer, que estaba impartiendo un curso sobre *wavelets* en los Estados Unidos, no pudo visitar el *Arts Institute* de Chicago debido a la asiduidad de un por entonces estudiante Mallat. Meyer insistió en que Mallat publicara en solitario su descubrimiento y ha colaborado activamente en propagar la técnica (en concreto, el curso [60] fue dictado en Madrid) y estudiar la cronología de los diversos intentos anteriores. La anécdota, que puede adivinarse en la introducción al Capítulo 8 del citado curso de Meyer y en el Prefacio del libro de Mallat [47], aparece documentada en Hubbard [33, Capítulo 3, pág.39].

La expectación que ha generado esta nueva técnica de procesamiento de señal se puede valorar observando la proliferación de sitios Web dedicados a *clasificar* la ingente cantidad de información disponible sobre el tema en forma de tutoriales, software, bibliografía y publicaciones científicas. Ejemplos de tales directorios se proporcionan en la Bibliografía [2, 4, 94, 50]. Por ejemplo, un tutorial sobre *wavelets* de la Sociedad Internacional de Ingeniería Optica [84] motiva al lector con el siguiente párrafo:

> La transformada *wavelet* ha sido reconocida como herramienta preferida para compresión de vídeo e imagen y ha sido seleccionada como el bloque constructivo fundamental del emergente estándar JPEG-2000.

Bilgin y Marcellin tienen un muy asequible texto introductorio sobre JPEG-2000 y publicaciones relacionadas con el análisis *wavelet* [48]. Es común encontrar investigadores con intereses en ambos temas. En una conferencia impartida en la ETSII de la Universidad de Granada, el Editor Jefe del *IEEE Signal Processing Magazine*, Aggelos Katsaggelos, mencionó que superar la inclusión en JPEG-2000 sería la prueba de fuego para el análisis *wavelet*, como en efecto se ha verificado.

MATLAB naturalmente ha creado una *Toolbox* específica para análisis *wavelet* [57], basada en la de Procesamiento de Señal [55]. Explicado desde el punto de vista de procesamiento de señal, es posible diseñar un tipo especial de filtros paso-baja/paso-alta emparejados (QMF, *Quadrature Mirror Filters*) tales que la información se separa en dos canales, denominados “aproximación” y “detalle”, a partir de los cuales se puede reconstruir perfectamente la señal original. La explicación matemática usa por supuesto terminología y contexto completamente distintos [45].

La respuesta impulsiva del filtro paso-alta original es la función *wavelet madre*, que determina de forma única el QMF paso-baja correspondiente (función de *escala*) y la familia de filtros de frecuencias progresivamente menores que realizan un análisis multiresolución asociado a dicha *wavelet* madre. En la versión digital del análisis, una de cada dos muestras filtradas puede descartarse (*decimación*) sin pérdida de información, y el análisis multiresolución se realiza en pasos diádicos que duplican la escala, reduciendo cada vez la frecuencia a la mitad. Una buena

elección de *wavelet* madre para compresión de imagen debería concentrar la mayor parte de la información en la aproximación, de manera que los coeficientes detalle se anularan en su mayor parte, pudiendo ser transmitidos con muy poco gasto. El análisis puede ser repetido sobre la aproximación, produciendo adicionales ahorros, hasta que el detalle comienza a absorber parte importante de la energía de la señal, siendo muchos de sus coeficientes no nulos.

Existe bibliografía documentando tanto los diversos métodos ideados para construir dicho tipo de filtros ([96, 61, 77], por ejemplo) como las propiedades matemáticas que debe cumplir la pareja de filtros QMF para su uso en un análisis multiresolución. En realidad, el artículo de Mallat [46] demuestra, mediante un expeditivo método constructivo, que las condiciones que intuitivamente se deberían exigir a un análisis multiresolución de señal, traducidas a forma matemática, caracterizan perfectamente a la pareja de filtros requerida. Mallat propone un análisis multiresolución basado en *splines cúbicos*, muy frecuentemente usados en procesamiento de imagen (ver por ejemplo la revisión [91]) a partir de los cuales deduce los correspondientes filtros.

A continuación se estudian distintas versiones secuenciales y paralelas de dicho análisis. La Figura 5.1 proporciona una idea global de los diversos programas realizados, al mostrar esquemáticamente el resultado del análisis y los computadores en que se ejecuta cada uno.

5.2 Análisis *wavelet* secuencial

Mallat realiza un análisis bidimensional separable, en el que los pasos de filtrado y decimación se aplican sucesiva y separadamente a cada dimensión. La Figura 5.2 muestra el proceso completo a un nivel. El proceso se puede repetir anidadamente, realizando primero varios pasos de análisis (cada uno sobre la aproximación del paso anterior) y posteriormente las correspondientes etapas de reconstrucción.

Entre ambas etapas, la de análisis y la de reconstrucción (esto es, sobre los coeficientes de la transformada), se realizaría el procesamiento deseado: transmisión, eliminación de ruido, suavizado o resaltado de bordes, etc. En nuestro caso no realizaremos ningún procesamiento, sino que pasaremos directamente a la etapa de reconstrucción para comprobar que no se perdió información en el proceso, es decir, que el algoritmo secuencial (y más adelante el paralelo también) está correctamente programado.

Volviendo a la Figura 5.2, primero se filtra horizontalmente la imagen mediante la convolución con las respuestas impulsivas de los filtros de análisis $\tilde{H}$ (aproximación) y $\tilde{G}$ (detalle). Los coeficientes obtenidos se deciman, manteniéndose una de cada dos columnas. El simbolismo $2{\downarrow}1$ se interpreta como “decimación horizontal por 2, decimación vertical por 1”. Estos coeficientes ah y dh son sólo el resultado intermedio de la primera parte separable del análisis, siendo necesario a continuación filtrar verticalmente con los mismos filtros y decimar verticalmente.

En la Figura 5.2 hemos recuadrado no sólo los operadores de filtrado y decimación/expansión, sino también las señales (imágenes), al objeto de que quede patente la progresiva reducción o ampliación del tamaño de la imagen con cada etapa del análisis o reconstrucción separable, respectivamente. Adicionalmente, esta convención permite superponer las imágenes sobre el esquema, como se ha hecho en las Figuras 5.3 y 5.4. Se confía en que tanto el etiquetado como los relativos tamaños de los distintos recuadros permitan distinguir fácilmente cuáles son las imágenes y cuáles los operadores o procesos aplicados a ellas.

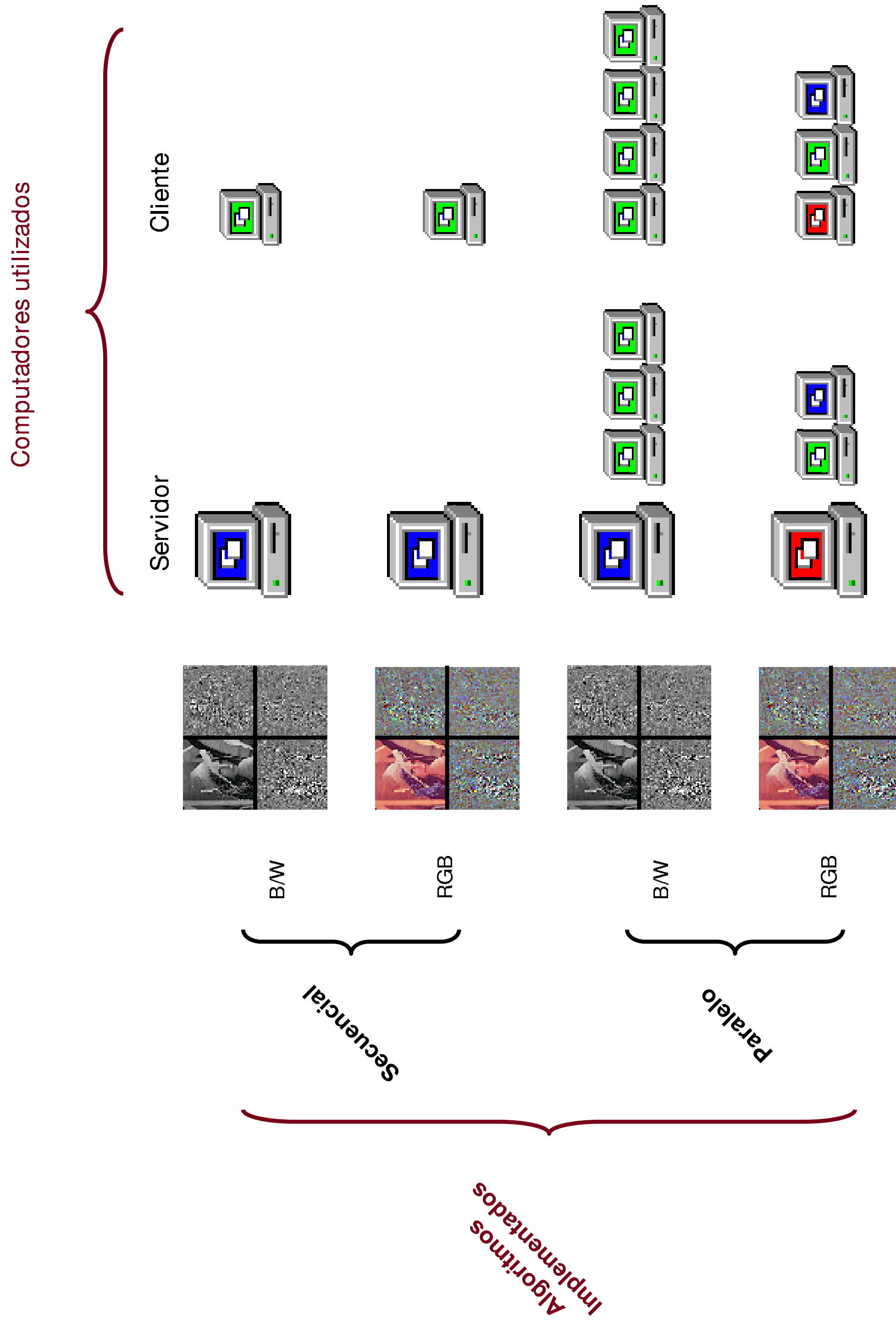


Figura 5.1: Algoritmos implementados y número de computadores que los ejecutan. Los programas secuenciales se ejecutan tanto en el servidor a 400MHz como en los clientes a 333MHz. Los programas paralelos se ejecutan usando y sin usar el servidor. La transformada en niveles de gris se ejecuta sobre 4 computadores. La transformada en color se paraleliza sobre 3.

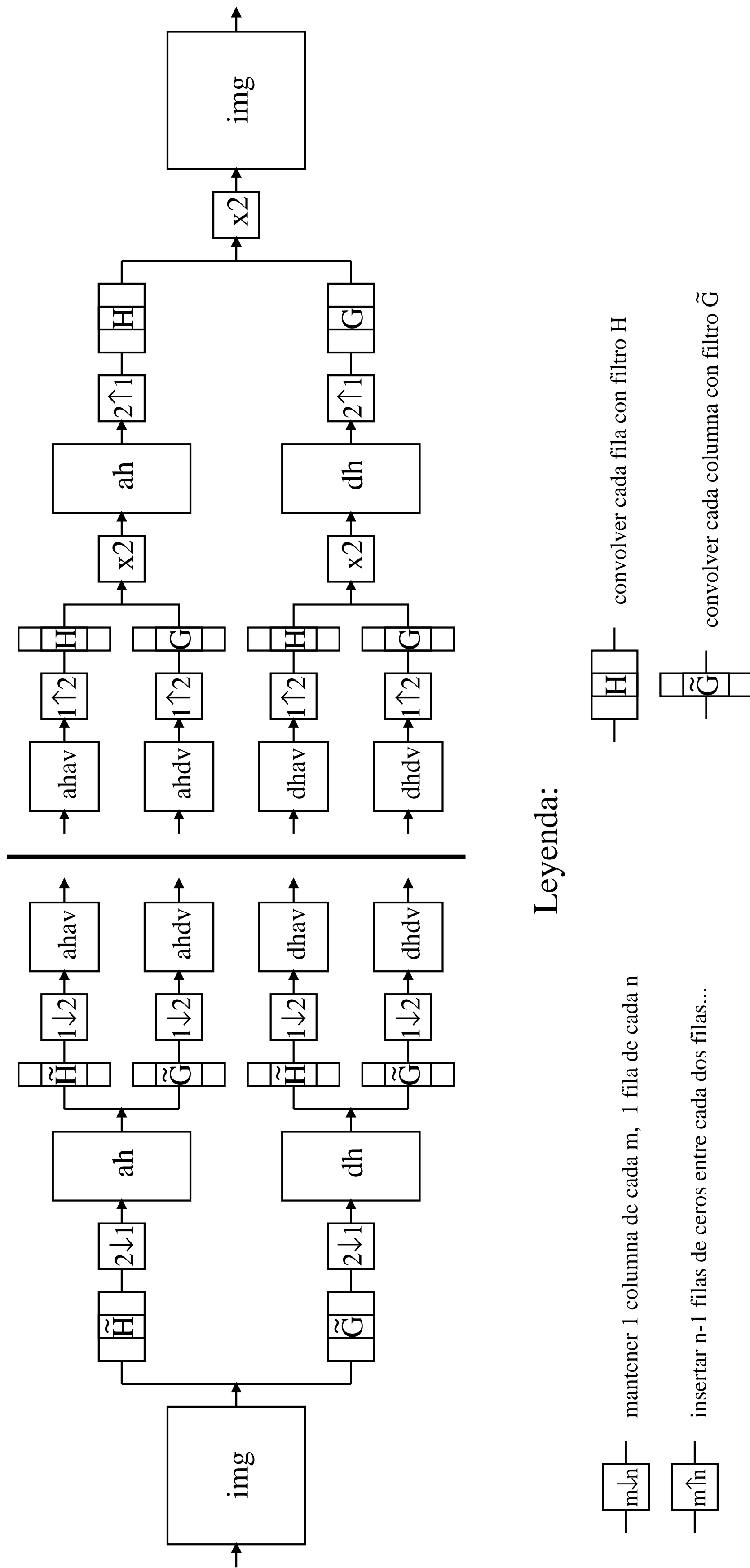


Figura 5.2: Análisis y reconstrucción *wavelet* de una imagen digitalizada. Según Mallat [46].

Se realizan por tanto 4 combinaciones de filtrado, cuyos resultados hemos etiquetado siguiendo la nomenclatura <tipo>h<tipo>v, en donde tipo es a/d según se trate de coeficientes aproximación o detalle, y h/v hacen referencia a la orientación horizontal o vertical. Así, ahdv es el subconjunto de coeficientes de la transformada *wavelet* bidimensional separable correspondientes a un primer filtrado horizontal de aproximación y un filtrado vertical de detalle. Se puede confiar en que dichos coeficientes delaten las líneas horizontales en la imagen original, por lo que se le suele llamar "detalle horizontal".

Las cuatro partes del análisis *wavelet* bidimensional separable, aproximación ahav, detalle horizontal ahdv, detalle vertical dhav y detalle diagonal dhdv, ocupan la misma cantidad de memoria que la señal original, y dadas las propiedades de los filtros de análisis, transportan la misma información. A diferencia de la señal original, muchos de los coeficientes obtenidos son nulos (para el tipo de imágenes que usualmente se desean transformar) o de pequeña magnitud. Se suelen presentar organizados como en la Figura 5.3: en la parte superior la aproximación y detalle horizontal, y en la mitad inferior los detalles vertical y diagonal.

En el análisis se puede intercambiar el orden de los filtrados horizontal y vertical, obteniéndose el mismo resultado. Naturalmente, las operaciones de decimación deben coincidir con el filtrado previo. No es necesario realizar la reconstrucción en el orden inverso a la descomposición, aunque en la Figura 5.2 se ha seguido ese criterio.

En la reconstrucción se expanden los coeficientes de la transformada antes de convolver con los filtros de reconstrucción H y G. De nuevo, el simbolismo $1\uparrow2$ se interpreta como "expansión horizontal por 1, expansión vertical por 2", implicando la inserción de una fila de ceros entre cada dos filas. Los filtros funcionan como interpoladores, siendo necesario un producto por 2 tras sumar la aproximación y el detalle para recuperar la energía removida de la señal mediante decimación en la etapa de análisis. Otra forma de considerar el problema consiste en constatar que la interpolación promedia la señal con 0.

Matemáticamente planteado, los filtros deberían acarrear un factor de normalización respecto a $\mathcal{L}^2(\mathcal{R})$ por cada reducción de escala para que la base fuera ortonormal: al contraerse a la mitad el dominio de los vectores se debería multiplicar por $\sqrt{2}$ su magnitud. De esta manera la norma de los vectores de la base mantendría valor unitario (el producto interno en $\mathcal{L}^2(\mathcal{R})$ es la integral del producto). En $\mathcal{L}^2(\mathcal{R}^2)$ (análisis 2D) una contracción a la mitad en cada dimensión requeriría un factor de 2 al reducir cada vector de la base. En un análisis separable se puede seguir utilizando el factor $\sqrt{2}$ con cada filtro 1D ya que la sucesiva aplicación horizontal y vertical del mismo produce el deseado factor global de 2.

Pero los filtros de la Figura 5.2 deberían poderse aplicar a cualquier imagen, incluso sucesivamente a la aproximación resultado del análisis anterior, sin conocer su escala real. Se prescinde por tanto de la normalización $\sqrt{2}$ en $\mathcal{L}^2(\mathcal{R})$, obteniéndose transformadas 1D de una magnitud $\sqrt{2}$ veces inferior, y transformadas 2D de magnitud mitad de la que se obtendría con una base ortonormal. Al volver a aplicar otras dos etapas de filtrado en la reconstrucción, la magnitud de la imagen reconstruida sería 1/4 de la original, lo cual se puede corregir añadiendo un producto final por 4. En el caso de un análisis separable, incorporar un producto por 2 al final de la rutina de reconstrucción 1D permite reutilizar los mismos filtros para transformada 1D y 2D.

A partir de las cuatro fracciones del análisis, aproximación ahav y detalles ahdv, dhav y dhdv, se obtienen por transformación vertical dos resultados intermedios ah y dh, que se combinan tras el homólogo filtrado horizontal para recuperar la imagen original sin pérdida de información.

En las Figuras 5.3 y 5.4 se muestran todos los resultados intermedios de aplicar el análisis y reconstrucción *wavelet* a la imagen de prueba. Obsérvese la correspondencia del esquema seguido en cada figura con la respectiva mitad de la Figura 5.2.

5.2.1 Código MATLAB secuencial

En el Listado 5.1 se muestra la parte más relevante el código MATLAB utilizado para el análisis y reconstrucción *wavelet*. Unas pocas líneas adicionales (no mostradas en este listado) permiten instrumentar el código para recolectar datos sobre el tiempo de ejecución y generar las gráficas correspondientes; el código completo aparece en el Apéndice E, Listado E.1.

```
%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_funet','tif'); load h, load g
gd=[  g(2:length(g));0];                          % G para descomposición
gr=[0;g(1:length(g)-1)];                          % G para reconstrucción

img=double(lena);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Descomposición 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[a1 dh1 dv1 dd1]=dwt2(img,h,gd);
[a2 dh2 dv2 dd2]=dwt2( a1,h,gd);
[a3 dh3 dv3 dd3]=dwt2( a2,h,gd);

%%%%%%%%%%%%%%%%%
% Recomposición %
%%%%%%%%%%%%%%%%%
r2=idwt2(a3,dh3,dv3,dd3 ,h,gr);
r1=idwt2(r2,dh2,dv2,dd2 ,h,gr);
r0=idwt2(r1,dh1,dv1,dd1 ,h,gr);
```

Listado 5.1: Sec.m: Programa MATLAB secuencial para transformada *wavelet* 2D B/W.

Al cargar la imagen, MATLAB la almacena en un array de tipo **uint8**. En versiones anteriores de MATLAB se debía almacenar en un array **double**, con un gasto de memoria sizeof(**double**)=8 veces mayor. Se introdujo esta modificación porque muchos usuarios deseaban sencillamente visualizar imágenes, sin operar con ellas. Las dimensiones de la imagen de prueba son 512x512=2^{18}=256KB (y los filtros H y G usados por Mallat son de longitud 23). En cualquier caso, para operar con la imagen es necesario transformarla a **double**, pasando a ocupar 2^9x2^9x2^3=2^{21} (2MB). Al ser simétrico el filtro de escala $\tilde{H}$ ($\tilde{G}$ no lo es) se puede reutilizar tanto para análisis como para reconstrucción ($\tilde{H} = H$).

Se realiza el análisis hasta el tercer nivel, conservando todas las transformadas, y posteriormente se reconstruye la imagen desde la transformada de nivel 3 (a3,dh3,dv3,dd3) sin utilizar las aproximaciones intermedias. Siguiendo una variante abreviada de la nomenclatura anterior, dh3 significa "detalle horizontal a nivel 3" (ahdv), dv3 es el detalle vertical (dhav), y dd3 es el detalle diagonal, equivalente al dhdv de la Figura 5.2.

A la rutina de análisis 2D dwt2 (por **d***iscrete* **w***avelet* **t***ransform* **2***D*) se le proporciona la imagen y los coeficientes de los filtros, devolviendo la aproximación y detalles de la transformada separadamente. A partir de la aproximación, los detalles y los coeficientes de los filtros de reconstrucción (el de escala coincide), la rutina inversa idwt2 (**i***nverse* **dwt 2***D*) reconstruye la imagen original.

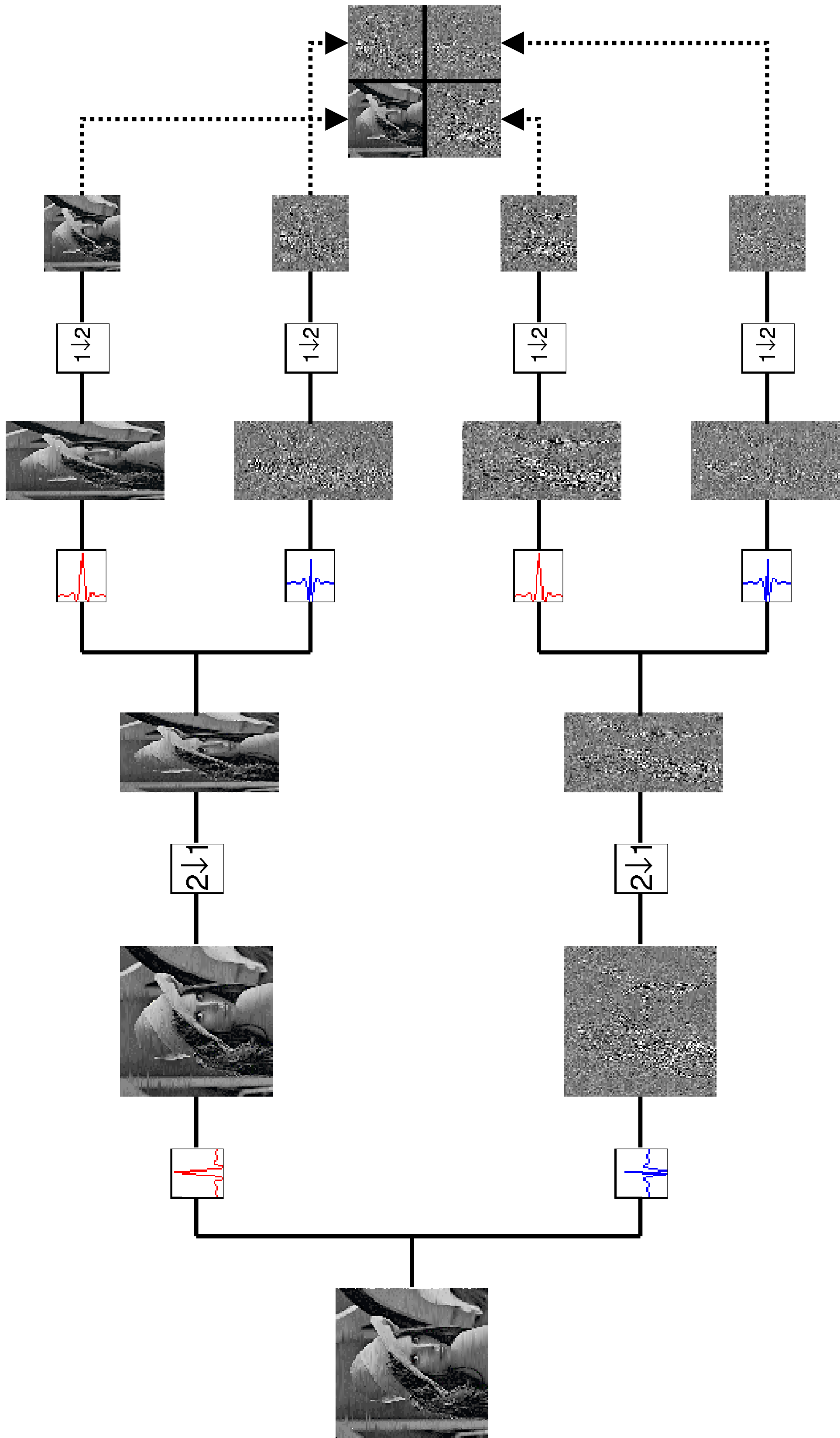


Figura 5.3: Esquema del proceso de descomposición *wavelet* de la imagen de prueba.

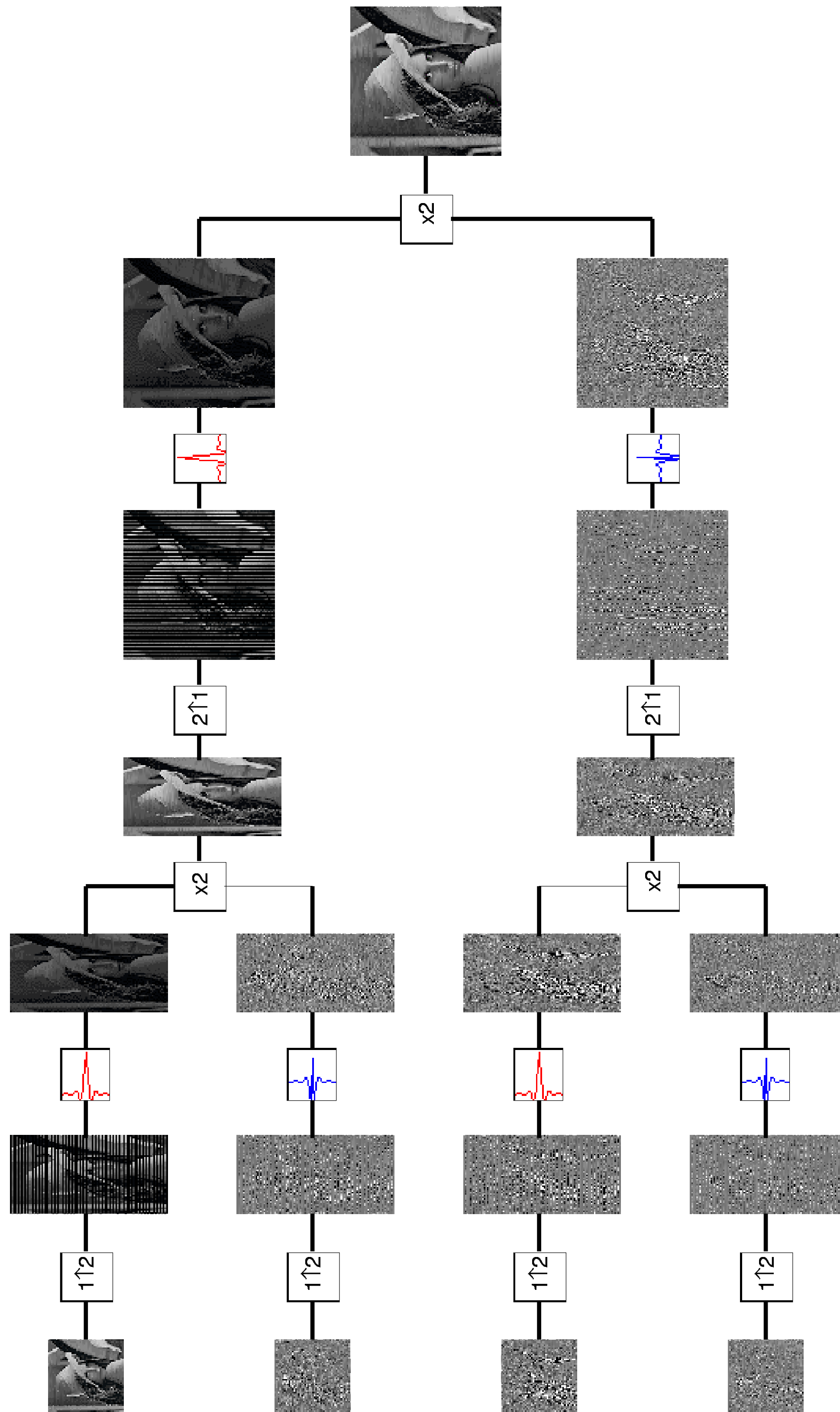


Figura 5.4: Esquema del proceso de reconstrucción *wavelet* de la imagen de prueba.

El Listado 5.2 es la parte más relevante de la rutina de análisis 2D (la mencionada dwt2), mostrada al completo en el Listado E.2.

```
function [ahav, ahdv, dhav, dhdv]=dwt2(img, h,g)

lih  =size(img,2);  liv  =size(img,1);                % Length image horz/vert
lih2 =ceil(lih/2);  liv2 =ceil(liv/2);                % Al decimar puede salir 1 de más

for row=1:liv                                         %%%%%%%%%%%%%%%%%%%
   r       = img(row,:);                              % Obtener img->ah, dh
  [ra rd] = dwt(r',h,g);                              %%%%%%%%%%%%%%%%%%%
   ah(row,:) = ra';                                   % Pasar vectores columna siempre
   dh(row,:) = rd';
end

for col=1:lih2
   c       = ah(:,col);                               %%%%%%%%%%%%%%%%%%%%%%
  [ca cd] = dwt(c ,h,g);                              % Obtener ah->ahav, ahdv
   ahav(:,col)= ca;                                   %%%%%%%%%%%%%%%%%%%%%%
   ahdv(:,col)= cd;

   c       = dh(:,col);                               %%%%%%%%%%%%%%%%%%%%%%
  [ca cd] = dwt(c ,h,g);                              % Obtener dh->dhav, dhdv
   dhav(:,col)= ca;                                   %%%%%%%%%%%%%%%%%%%%%%
   dhdv(:,col)= cd;
end
```

Listado 5.2: dwt2.m: Rutina para análisis *wavelet* 2D.

MATLAB puede consultar en tiempo de ejecución la longitud de la imagen (horizontal lih y vertical liv, en la nomenclatura usada en el listado), no siendo necesario indicarla como argumento a dwt2.

Tal y como se indicó en la Figura 5.2, se realiza primero un barrido por filas, almacenando en r la fila en cuestión. La notación (:) selecciona todo el rango de índices, en este caso de columnas. Al vector fila r se le aplica el operador de transposición (') ya que es habitual redactar los comandos MATLAB para que operen con vectores columna, y se ha seguido la costumbre con la rutina de análisis 1D dwt. Se invoca por tanto a dwt con la columna r' y los mismos filtros de análisis especificados en la llamada a dwt2 ($\tilde{H}$ y $\tilde{G}$, parámetros formales h y g).

Los coeficientes devueltos, aproximación ra y detalle rd, son almacenados en la correspondiente fila de los arrays intermedios, que se han denominado ah y dh siguiendo la misma notación de la Figura 5.2. dwt calcula tanto la aproximación ra como el detalle rd para reutilizar la fila original r y no incurrir de nuevo en el coste de extracción (operador (:)).

Se aplica entonces el filtrado vertical a las lih2 columnas de los resultados intermedios ah/dh. Tras la decimación, los resultados intermedios tenían la mitad de columnas que la imagen original. Siguiendo el mismo esquema que para el filtrado horizontal, se extrae primero la columna en cuestión c usando la notación (:) (no es necesario trasponer esta vez). Los coeficientes de aproximación y detalle ca/cd se almacenan en las correspondientes columnas del resultado final, sea éste la aproximación ahav o los detalles ahdv, dhav, dhdv.

Del listado completo del análisis unidimensional (E.3, dwt.m) se ha extraido el fragmento mostrado en el Listado 5.3.

Siendo nuestros filtros de longitud impar, se calcula primero la longitud de una "cola" del filtro, quitando el coeficiente central. La señal 1D a tratar (x en el listado) se extiende especularmente en una cola a ambos extremos para evitar efectos de borde, y tras realizar el filtrado-convolución

```
function [a, d]=dwt(x,h,g)

lx=length(x); lh=length(h); lg=length(g);           % Longitudes
lh2=floor(lh/2);                                     % Si lf impar, lf2 es longitud cola
lg2=floor(lg/2);                                     % (quitando la muestra central)
                                                     %% Aproximacion %%%%%%%%%%
y=[x(lh2+1:-1:2); x ; x(lx-1:-1:lx-lh2)];           % Colas para efecto borde
a=conv(y,h);                                         % Filtrado
a=a(lh:lx+lh-1);                                     % Corte correspondiente
a=a(1:2:lx);                    % Downsample         % Puede salir ceil(lx/2) una de más
                                                     %% Detalle %%%%%%%%%%%%%%%
y=[x(lg2+1:-1:2); x ; x(lx-1:-1:lx-lg2)];           % Colas
d=conv(y,g);                                         % Filtrado
d=d(lg:lx+lg-1);                                     % Corte
d=d(1:2:lx);                    % Downsample         % Puede salir ceil(lx/2) una de más
```

Listado 5.3: dwt.m: Rutina para análisis *wavelet* 1D.

se conserva el tramo central libre de efectos de borde. Como vimos, se pasan ambos filtros al objeto de reutilizar la señal x, calculando tanto la aproximación a como el detalle d.

Por fijar ideas, si la imagen era de 512x512 y los filtros de longitud 23, una cola del filtro de aproximación h mide lh2=11 muestras. La convolución directa con una señal x de longitud lx=512 daría como resultado una aproximación a de longitud la=lx+lh−1=534, y deberíamos descartar las primeras y últimas lh2=11 muestras "de cola" por no estar centrado el filtro sobre muestras de la señal, conservando sólo a(12:523), es decir, a(lh2+1:lh2+lx).

Las colas añadidas antes de la convolución son una mera reflexión especular de la señal, para evitar que el filtro quede "al aire" al aproximarse a los bordes, y son también eliminadas del resultado final (que ahora es dos colas más largo, 556 muestras) comenzando la indexación una cola más tarde y acabando una cola antes, a(23:534), es decir, a(lh:lx+lh−1), cuando el filtro está centrado sobre el inicio y final de la señal original y cubre también las colas añadidas, evitando los efectos de borde.

Similarmente, la transformada inversa 2D en el Listado 5.4 (completo en el E.4) se basa en la inversa 1D. Proporcionando en este caso los filtros de reconstrucción H y G, a partir de la aproximación y detalles horizontal, vertical y diagonal, se reconstruye la señal original img.

Se siguen nuevamente los pasos y nomenclatura de la Figura 5.2. Obsérvese que se realizan los filtrados de escala y de *wavelet* en la misma iteración del bucle, para evitar los costes de un bucle adicional.

En el primer paso se extraen, usando la notación (:), las columnas aproximación y detalle ca/cd que se obtuvieron por convolución vertical del resultado intermedio ah, se les aplica la reconstrucción 1D con los filtros de síntesis (el filtro de escala coincide, H=$\tilde{H}$) y se almacena la columna resultado c en la correspondiente columna de un nuevo resultado intermedio, que volvemos a denominar ah. Tras la interpolación se duplica el número de filas, y la longitud vertical de ah es liv, como se observa en la pre-ubicación de dicho array. En el mismo bucle se obtiene también el resultado intermedio dh.

La siguiente etapa se realiza sobre un número de filas igual al de la imagen original, ya que las columnas obtenidas en la etapa anterior eran el doble de largas. Las filas ra/rd de los resultados intermedios ah/dh (de longitud lih2) se deben trasponer, ya que la rutina de síntesis 1D idwt también opera sobre vectores columna. La columna resultado r (longitud doble lih) se traspone antes de almacenarla en la fila correspondiente de la imagen reconstruida img.

```
function img=idwt2(ahav, ahdv, dhav, dhdv, h,g)

l2=size(ahav);                                          % Longitudes
lih2=l2(2);   liv2=l2(1);
lih =lih2*2;  liv =liv2*2;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Interpolar por columnas, cada columna se filtra a lo largo de las filas
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ah=zeros(liv,lih2); dh=zeros(liv,lih2);                 % Pre-allocation
for col=1:lih2
   ca=ahav(:,col); cd=ahdv(:,col);                      %%%%%%%%%%%%%%%%%%%%%%%%%
   c =idwt(ca ,cd ,h,g);                                % Obtener ahav,ahdv->ah
   ah(:,col)= c ;                                       %%%%%%%%%%%%%%%%%%%%%%%%%

   ca=dhav(:,col); cd=dhdv(:,col);                      %%%%%%%%%%%%%%%%%%%%%%%%%
   c =idwt(ca ,cd ,h,g);                                % Obtener dhav,dhdv->dh
   dh(:,col)= c ;                                       %%%%%%%%%%%%%%%%%%%%%%%%%
end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Interpolar por filas, cada fila se filtra a lo largo de las columnas
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
img=zeros(liv,lih);                                     % Pre-allocation
for row=1:liv
   ra=ah(row,:); rd=dh(row,:);                          %%%%%%%%%%%%%%%%%%%%%
   r =idwt(ra',rd',h,g);                                % Obtener ah,dh->img
   img(row,:)= r';                                      %%%%%%%%%%%%%%%%%%%%%
end                                                     % Pasar siempre vectores columna
```

Listado 5.4: idwt2.m: Rutina para reconstrucción *wavelet* 2D.

Así pues, la reconstrucción 2D se basa en la rutina de reconstrucción 1D. Ésta realiza la expansión (*upsample*) e interpolación de cada señal (aproximación o detalle) con el filtro correspondiente (H o G), sumando posteriormente ambas para obtener la deseada reconstrucción. El Listado 5.5 es la porción relevante del E.5, la versión completa.

Se aplican las mismas "colas de convolución" que en el filtrado de análisis, descartando de nuevo la parte marginal y conservando la porción sobre la que el filtro ha estado centrado. Recuérdese que se debe recuperar la energía perdida en la decimación del análisis, multiplicando la señal por 2. Esto permite utilizar este código tanto para análisis unidimensional como bidimensional sin tener que realizar adicionales correcciones de normalización.

```
function x=idwt(a,d,h,g)

la=length(a); ld=length(d);                          % Longitudes
lh=length(h); lg=length(g);
lh2=floor(lh/2);                                     % Si lf impar, lf2 es longitud cola
lg2=floor(lg/2);                                     % (quitando muestra central)
                                                     %% Aproximacion %%%%%%%%
y(1:2:2*la)=a;                                       % Upsample. a(1),0,a(2),0... a(la),0
y=[y(lh2+1:-1:2); y ; y(2*la-1:-1:2*la-lh2)];        % Colas para efecto borde
za=conv(y,h);                                        % Filtrado
za=za(lh:2*la+lh-1);                                 % Corte correspondiente
                                                     %% Detalle %%%%%%%%%%%%%
y(1:2:2*ld)=d;                                       % Upsample. d(1),0,d(2),0... d(ld),0
y=[y(lg2+1:-1:2); y ; y(2*ld-1:-1:2*ld-lg2)];        % Colas
zd=conv(y,g);                                        % Filtrado
zd=zd(lg:2*ld+lg-1);                                 % Corte

x=2*(za+zd);                                         % Resultado, recuperar normalización R^2
```

Listado 5.5: idwt.m: Rutina para reconstrucción *wavelet* 1D.

Para tratamiento de imágenes por computador es muy conveniente que la transformada no esté normalizada en el espacio $\mathcal{L}^2(\mathcal{Z}^2)$ ya que las sucesivas aproximaciones serían cada vez más brillantes (para compensar la progresivamente reducida escala), saturándose pronto a una imagen completamente blanca. Incorporar la recuperación de la normalización en la reconstrucción 1D permite, además de obtener unas aproximaciones con la misma *densidad* de energía (brillo) que la imagen original, olvidarse del asunto de las normalizaciones. De hecho, la rutina `idwt2` no necesita ninguna normalización, al estar programada a base de repetidas invocaciones a `idwt`.

En la Figura 5.5 se muestran la imagen original, su transformada y la reconstrucción, así como las gráficas obtenidas mediante el mencionado código de instrumentación, que se proporciona como material de referencia en los Listados del E.6 al E.11.

El coste computacional mostrado en la Figura 5.5(d) incluye la visualización de las imágenes. Aunque no se haya mostrado en los listados de este capítulo, el programa admite argumentos para especificar el número de veces a repetir el experimento y si se desea visualizar las imágenes y/u obtener estadísticas sobre el tiempo de cálculo (consultar Listado E.1).

Los resultados de ejecutar el programa con el cluster controlado, sin visualizar las imágenes, se resumen en la Tabla 5.1 junto con los de la versión en color, para permitir una más fácil comparación.

5.2.2 Código MATLAB secuencial para imagen en color

Una forma sencilla de conseguir paralelismo consistiría en repetir en distintos computadores el mismo algoritmo secuencial, pero con distintos datos, como por ejemplo los distintos canales de color de una imagen RGB. Esta aproximación se corresponde a la cuarta línea en la Figura 5.1, ejecutándose en 3 computadores.

Para calcular el *speedup* a que conduce dicha aproximación es necesario conocer el tiempo (secuencial) que emplearía un computador aislado en completar el mismo cálculo. Este caso aparece reflejado en la segunda línea de la Figura 5.1, y se ejecutaría en un único computador, como conviene al epígrafe de este apartado.

Al no aportar nada nuevo el código fuente, nos limitamos a mostrar esquemáticamente el cálculo global en la Figura 5.6. Como material de referencia se ofrece el código completo en el Listado E.12. En la Figura 5.7 se muestran la imagen original, transformada y reconstrucción, así como las gráficas obtenidas con el habitual código de instrumentación.

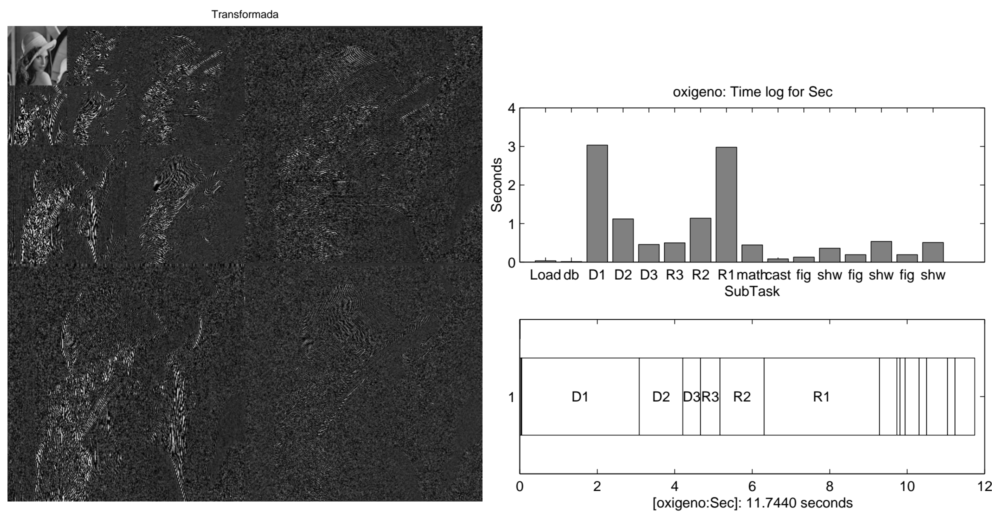
Ejecutando el programa con el cluster controlado y sin visualización se obtuvieron las estadísticas que se resumen en la Tabla 5.1 junto con las de la versión B/W. El tiempo para el análisis RGB es consistentemente el triple que para el B/W, y el computador servidor es consistentemente 1.6 veces más rápido que el computador cliente.

Relativo (normalizado a B/W Servidor)			**Absoluto** (segundos)			
			imagen B/W		imagen RGB	
imagen B/W	imagen RGB	**Computador**	media	desv.std.	media	desv.std.
1.0000	2.9945	**Servidor**	9.1606	0.0152	27.4318	0.0236
1.5937	4.8092	**Cliente**	14.5991	0.0175	44.0553	0.1102

Tabla 5.1: Resultados Secuenciales (obtenidos con el cluster desconectado del exterior).

(a) Imagen original.

(b) Reconstrucción. El análisis multiresolución conserva la información.



(c) Transformada a tercer nivel. Se sigue la disposición tradicional, aproximación y detalle horizontal arriba, detalles vertical y diagonal abajo.

(d) Gasto computacional. El código de instrumentación elaborado para generar estas gráficas se muestra en el Apéndice E, Listados del E.6 al E.11.

Figura 5.5: Transformada *wavelet* 2D B/W secuencial.

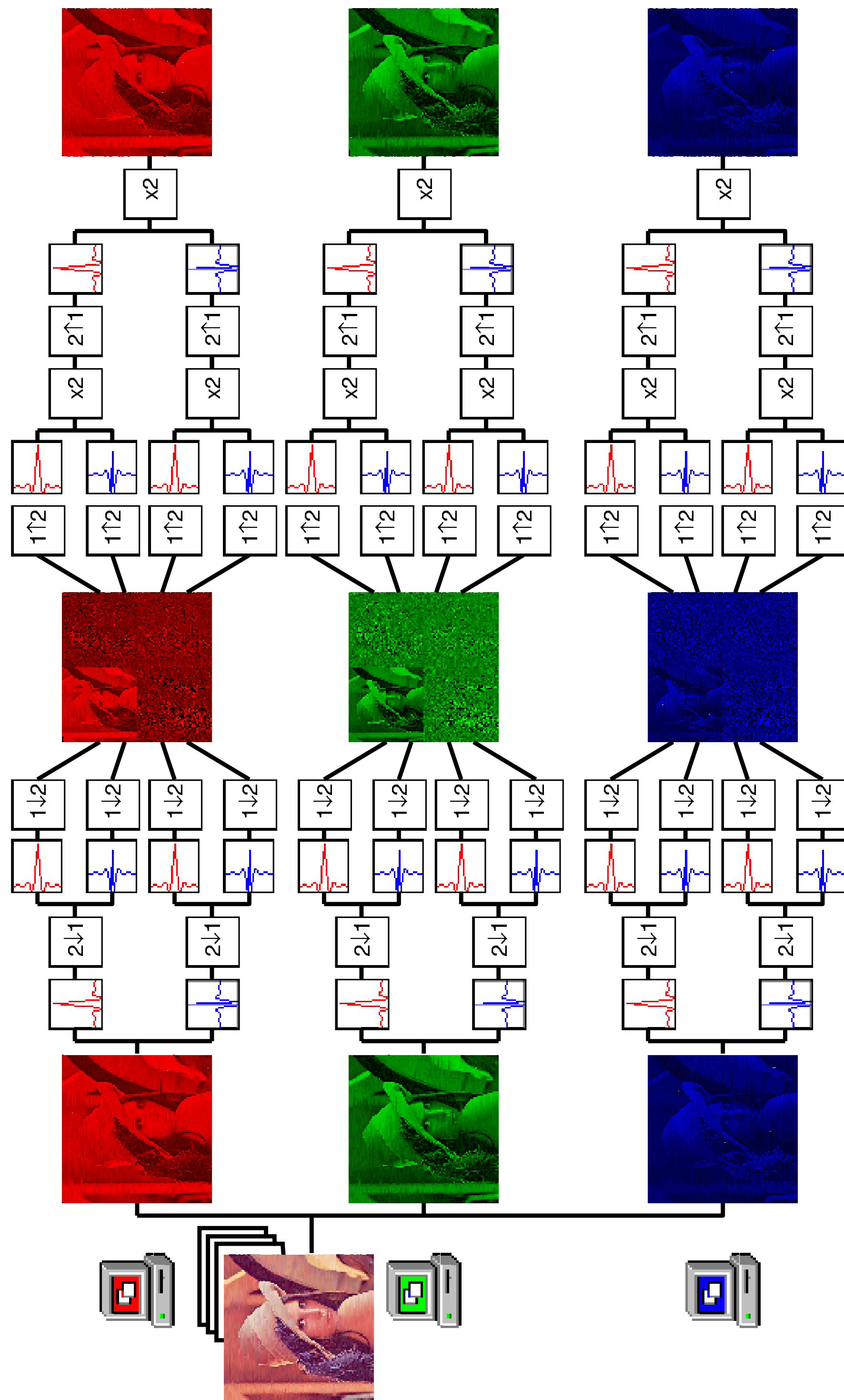
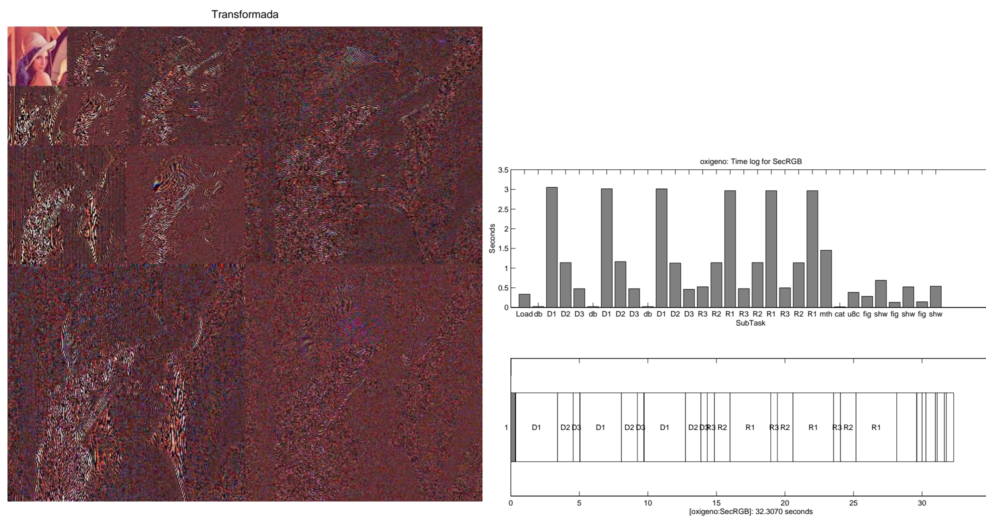


Figura 5.6: Análisis y reconstrucción *wavelet* de una imagen RGB. Se aplica separadamente el mismo algoritmo (Mallat [46]) a cada canal de color. La tarea es directamente paralelizable, asignando a un computador distinto cada canal de color.

(a) Imagen original.

(b) Reconstrucción. El análisis multiresolución conserva la información.



(c) Transformada a tercer nivel. Se sigue la disposición tradicional, aproximación y detalle horizontal arriba, detalles vertical y diagonal abajo.

(d) Gasto computacional. El tiempo de cálculo se triplica en comparación con el análisis B/W (Figura 5.5(d)), pero las tareas asociadas de carga de imagen, conversión a **double**, normalización de la transformada y creación de figuras rebajan el factor.

Figura 5.7: Transformada *wavelet* 2D RGB secuencial.

5.3 Análisis *wavelet* paralelo para imágenes B/W

La paralelización ideada para el algoritmo básico B/W se basa en la presencia de 4 operaciones independientes de filtrado al final de la descomposición y al principio de la reconstrucción (ver Figura 5.2). Por lo tanto, cuatro computadores podrían repartirse el trabajo de forma equilibrada. El objetivo es realizar un *test ácido* sobre las *Toolboxes*, no un estudio de escalabilidad de un algoritmo *wavelet*. Sólo se considera un particionamiento estático con 4 computadores.

La primera etapa en la descomposición (Figura 5.3) y la segunda en la reconstrucción (Figura 5.4) sólo constan de dos operaciones independientes de filtrado. El algoritmo paralelo que hemos diseñado reparte equilibradamente esta carga entre cada dos procesos, como se representa esquemáticamente en las Figuras 5.8 y 5.9.

En la primera etapa de la descomposición (Figura 5.8), los cuatro procesos involucrados colaboran por parejas. La primera pareja coopera en el cálculo del resultado intermedio de aproximación horizontal ah, y la segunda se dedica al detalle horizontal dh. De cada pareja, el primer proceso recibe la mitad superior de la imagen y calcula la mitad superior del resultado intermedio. El segundo proceso de la pareja hace lo propio con la mitad inferior. Entonces cada proceso comparte su resultado con el otro, de manera que ambos disponen del resultado intermedio completo y pueden proseguir independientemente con la segunda etapa de filtrado.

Uno de los procesos debe distribuir inicialmente los datos, leyendo de disco la imagen, convirtiéndola a **double** para poder operar matemáticamente sobre ella, e identificando a los restantes procesos para enviarles la mitad que les corresponda. La transmisión se realiza con datos **double**, puesto que la transformada *wavelet* se desea repetir hasta el tercer nivel, aplicando el mismo algoritmo a las sucesivas aproximaciones (**double**) obtenidas con cada etapa de descomposición. Realizar la primera transmisión con datos **uint8** requeriría considerar como caso especial la descomposición de primer nivel.

Tras el reparto inicial de datos se realiza el filtrado y decimación horizontal en paralelo, compartiéndose los resultados intermedios con el proceso emparejado. Esta transmisión es obligatoriamente **double**, ya que los coeficientes de la transformada no son cantidades enteras como los niveles de gris originales. Afortunadamente, las dimensiones del array de coeficientes son una cuarta parte de las originales: mitad por reparto de carga y mitad por decimación *wavelet*; el tamaño es sólo el doble, en lugar de sizeof(**double**)=8 veces el de la imagen original.

Tras realizar la segunda etapa de filtrado y decimación (vertical), se devuelven al proceso inicial los arrays de coeficientes calculados (aproximación y detalles). Esta transmisión también debe ser **double**, naturalmente; de hecho, la mayoría de los coeficientes detalle serán valores de magnitud muy baja, positivos o negativos.

En la reconstrucción (Figura 5.9), los procesos pueden calcular sin comunicación el filtrado y decimación verticales, obteniendo un array de altura doble que los coeficientes de los que parten. Como el primer proceso de cada pareja se dedicará en la siguiente etapa sólo a la mitad superior de la imagen (y el segundo a la inferior), no es necesario realizar la suma completa de los arrays de coeficientes en ambos procesos. El primer proceso transmite al segundo la mitad inferior de su array (y éste su mitad superior al primero), en donde se realiza la suma y se prosigue independientemente con la segunda etapa de filtrado horizontal.

Aunque la suma final se hubiera debido realizar en árbol, sumando los segundos procesos de cada grupo sus resultados parciales antes de devolverlos al coordinador (y los primeros también)

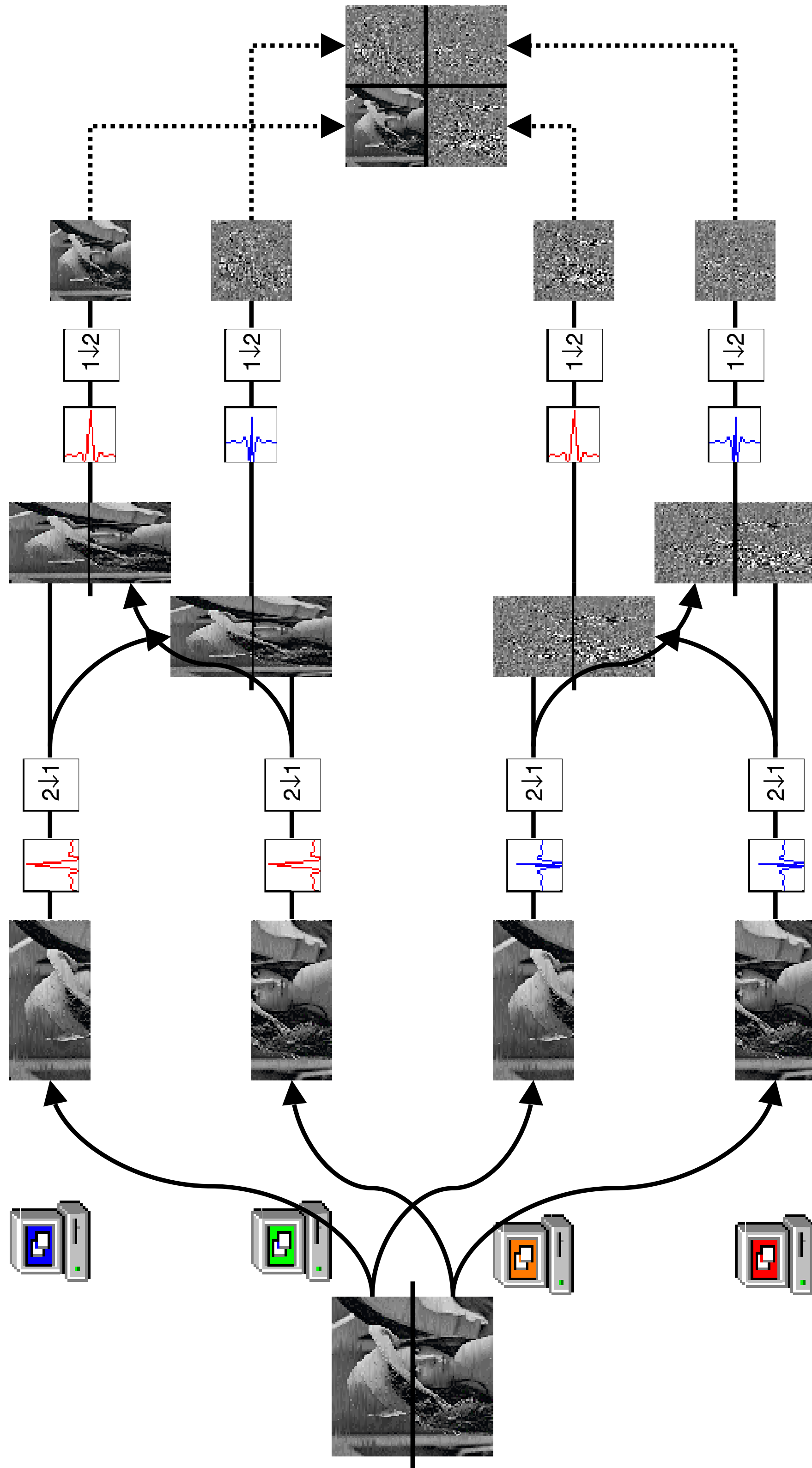


Figura 5.8: Esquema del proceso de descomposición *wavelet* de la imagen de prueba.

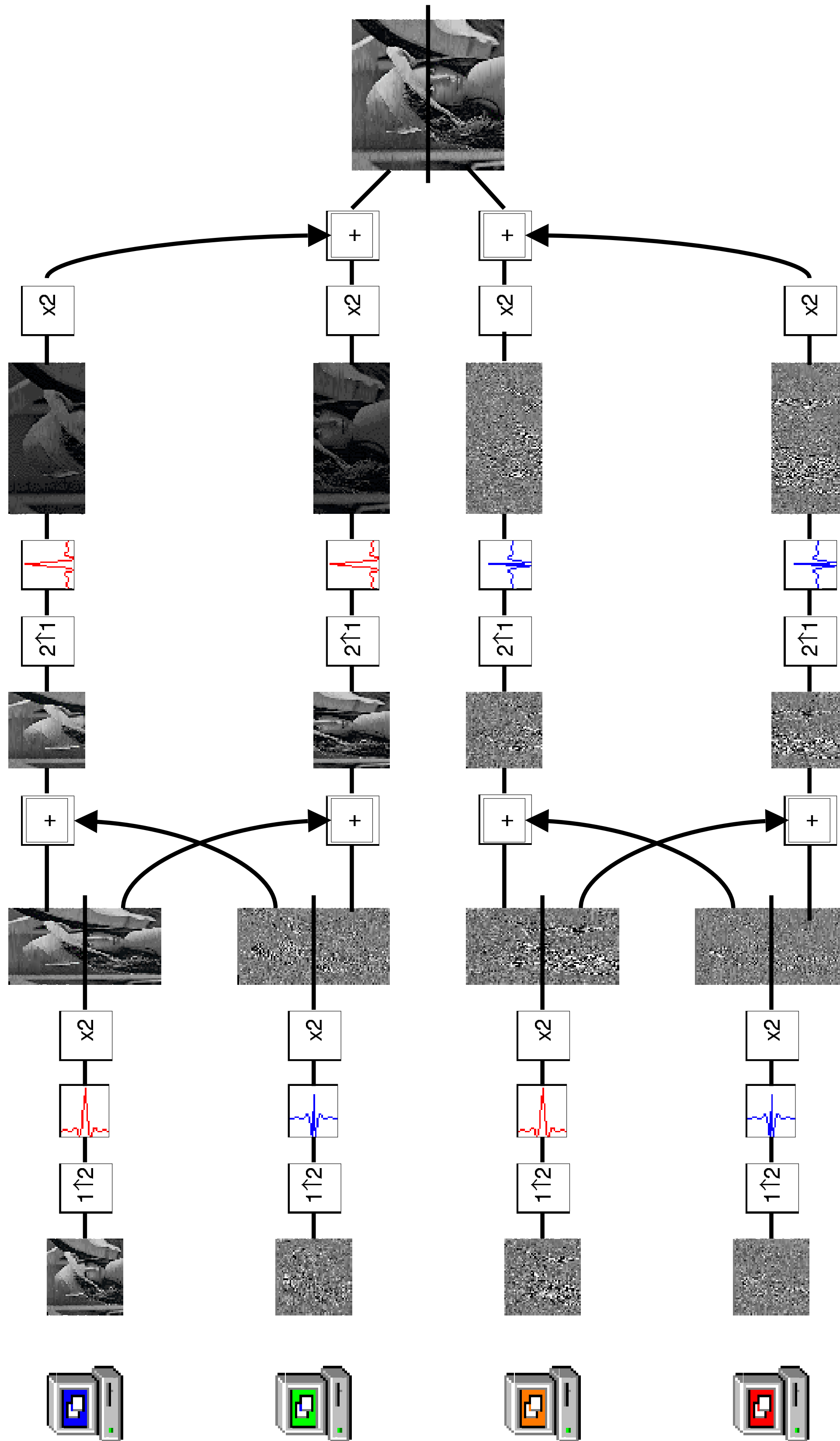


Figura 5.9: Esquema del proceso de reconstrucción *wavelet* de la imagen de prueba.

se ha preferido devolverlo todo al coordinador en donde se realiza la suma final, por un argumento de simetría. Una persecución exigente del máximo *speedup* posible también hubiera debido considerar un reparto inicial de datos en árbol: mientras el proceso distribuidor reparte la segunda semi-imagen, el cliente que acaba de recibir la primera puede transmitirla a un tercer cliente.

Para poder ejercitar el algoritmo con o sin la colaboración del computador servidor (tercera línea de la Figura 5.1), la citada primera semi-imagen es en el primer caso la inferior, dejándose de realizar la última transmisión, y en el segundo caso la superior, pudiendo evitarse la penúltima transmisión. La lógica para excluir dichas transmisiones no es muy complicada, ni tampoco lo es la de la suma en árbol o la transmisión inicial de tipo **uint8**, pero complicaría la explicación de los listados, y tampoco favorece nuestra intención de que este test fuera *ácido*.

El algoritmo completo se esquematiza en la Figura 5.10.

5.3.1 Versión paralela B/W PVMTB

Se ha preferido segregar las distintas funcionalidades requeridas en distintos ficheros, para facilitar el mantenimiento de la aplicación. Puede ser de utilidad conocer de antemano las dependencias entre las distintas rutinas, que se pueden resumir esquemáticamente como sigue:

```
Par -> dwt2ParHost -> dwt2Par -> dwtflt
   -> idwt2ParHost -> idwt2Par -> idwtflt
```

La aplicación sigue el paradigma maestro-esclavo. Para no hacer la exposición inútilmente árida, toda vez que los detalles del algoritmo han sido ya expuestos, se ha relegado el código completo de dichos programas MATLAB al Apéndice E (Listados del E.14 al E.20), mostrándose en la exposición que sigue sólo el uso de algunos comandos PVMTB destacados.

Puede ser interesante mostrar con detalle el código de inicialización, con el que se adquiere control de los procesos MATLAB esclavos. Siendo esta una aplicación un tanto compleja, se ha utilizado el sistema **startup** ya comentado en el Apartado 1.6.2. Los procesos MATLAB esclavos en los computadores clientes serán discriminados por la existencia de la variable de entorno PVMEPID, y forzados a ejecutar un *script* adicional startup_Slv, conteniendo básicamente lo mostrado en el Listado 5.6.

```
pvm_recv(pvm_parent,TAG); pvm_unpack;                % Se espera variable NUMCMDS
                                                     % y flag QUIT=0/1
for indiceBuclePVM=1:NUMCMDS                         % Bucle controlado por NUMCMDS
    indiceBuclePVM                                   % Para ver progreso
    pvm_recv(pvm_parent,TAG); pvm_unpack; eval(cmd);
end

if QUIT, quit, end                                   % Si se indicó salir, hacerlo
```

Listado 5.6: startup_Slv.m: *script* esclavo que establece el protocolo NUMCMDS/QUIT con el maestro.

El Listado E.13 proporciona el código completo. Este *script* espera recibir una variable llamada NUMCMDS, que le indica cuántos comandos se desea que ejecute, y un *flag* QUIT señalando si debe terminar la sesión MATLAB tras evaluar los NUMCMDS comandos. Tras recibir dichas variables, entra en un bucle de recepción y evaluación de comandos. Al terminar el bucle, abandona el entorno MATLAB si se le indicó que así lo hiciera. Aunque no se muestre en este fragmento, cuando se indican NUMCMDS=0 comandos, el bucle está controlado por el envío de un comando **quit** o **exit**, que provoca la salida del bucle de recepción y evaluación.

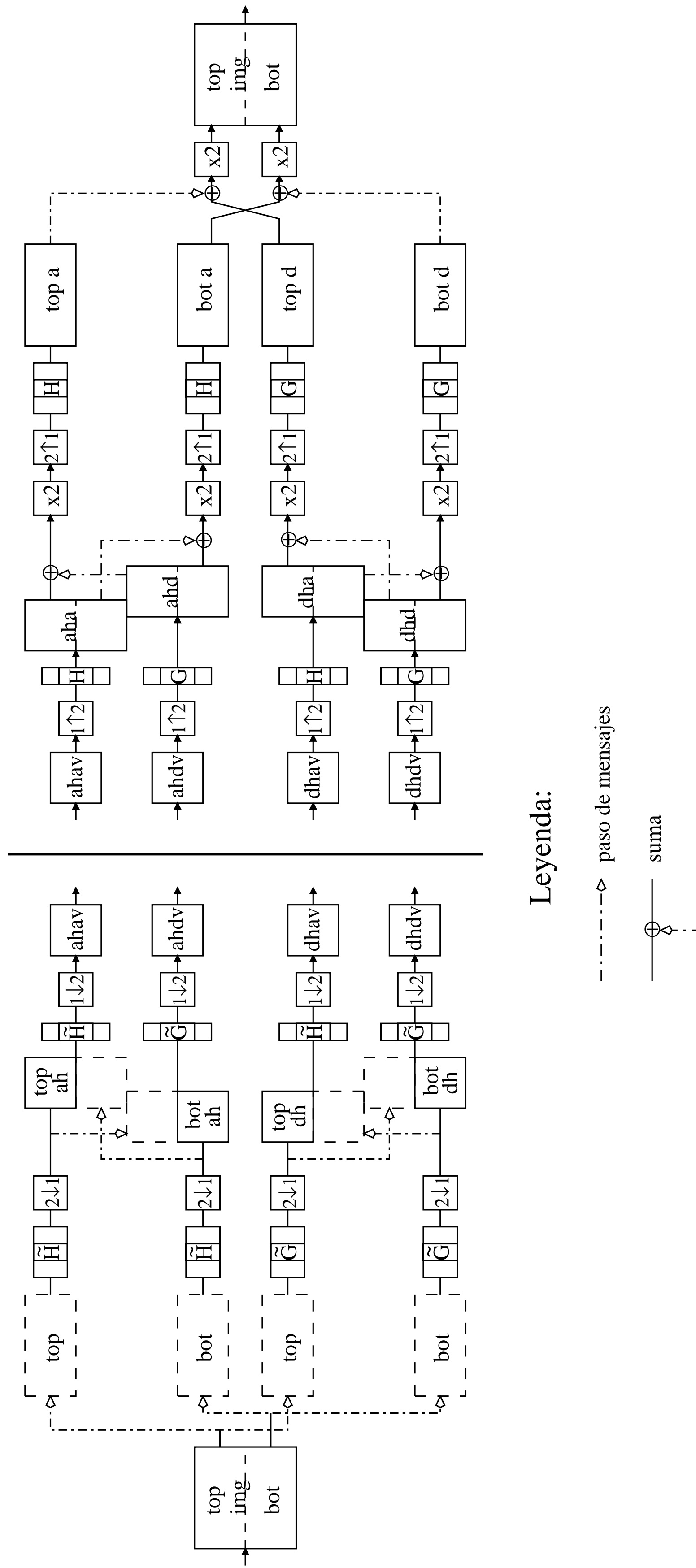


Figura 5.10: Análisis y reconstrucción *wavelet* de una imagen digitalizada. Versión paralela del algoritmo representado en la Figura 5.2.

El programa maestro Par, mostrado en el Listado 5.7, organiza los mismos tres niveles de análisis y reconstrucciones que se realizaron en el programa secuencial.

```
% Descomposición 3 niveles, Dependencia secuencial
[a1 dh1 dv1 dd1]=dwt2ParHost(lena,h,gd,tids);
[a2 dh2 dv2 dd2]=dwt2ParHost(  a1,h,gd,tids);
[a3 dh3 dv3 dd3]=dwt2ParHost(  a2,h,gd,tids);

% Recomposición, Dependencia secuencial
r2=idwt2ParHost(a3,dh3,dv3,dd3, h,gr, tids);
r1=idwt2ParHost(r2,dh2,dv2,dd2, h,gr, tids);
r0=idwt2ParHost(r1,dh1,dv1,dd1, h,gr, tids);
```

Listado 5.7: Par.m: Programa PVMTB maestro para transformada *wavelet* 2D B/W.

Este algoritmo pretende ser un *test ácido* al esfuerzo de paralelización, dificultando la obtención de *speedups* elevados por diversos motivos:

- El reparto de datos inicial se realiza linealmente y con datos **double**. La suma final tampoco se realiza en árbol.
- La aplicación presenta varias etapas de paso de mensajes y sincronización (4 en total).
- Se reduce progresivamente la granularidad (en un factor de 4). Cada aproximación ahav es 4 veces menor que la imagen de la que proviene, de manera que después de 3 descomposiciones se manejan imágenes 4^3=64 veces más pequeñas que la original. Al disminuír el tamaño de los mensajes la latencia va cobrando importancia.

Aunque no se muestre en el Listado 5.7, Par también se encarga de arrancar los procesos esclavos, establecer el protocolo comentado y generar las gráficas de resultados y estadísticas de ejecución. El código completo está en el Listado E.14.

Obsérvese que se pasa el array tids de identificadores de tarea PVM a los *script* de distribución. Estos *scripts* realizan el envío de datos iniciales y recuperación final de resultados. En la etapa de análisis, dwt2ParHost (Listado 5.8, completo en E.15) debe mandar la mitad superior de la imagen al primer proceso de ambos grupos y la inferior al segundo.

Se contempla la posibilidad de que el computador servidor colabore en el cálculo como primer proceso del primer grupo, o de que mande la tarea a otro cliente. Se observa que la transmisión inicial de datos es no sólo con tipo de datos **double**, sino que además es por duplicado, para que puedan realizarse en paralelo los filtrados de aproximación (primer grupo) y detalle (2º grupo) de la primera etapa. Los esclavos ejecutan entonces el *script* dwt2Par, correspondiente a una línea horizontal del esquema en la Figura 5.8, del cual la parte más relevante se muestra en el Listado 5.9 (completo en E.16).

De esta última rutina, que incluye todo el código de la transformada responsable de la mayor parte del tiempo de ejecución, se muestra tan sólo la estructura del código y las llamadas de paso de mensajes. Obsérvese el *flag* slv identificando si este proceso es segundo en el grupo, en cuyo caso la media imagen que debe mandar a la pareja es la superior (filas desde la 1 hasta liv2). Para el otro proceso del grupo, el intercambio se realiza con la mitad inferior (filas desde liv2+1 hasta liv). La expresión `tmp((1-slv)*liv2+1:(2-slv)*liv2 ,:)` contempla ambos casos.

Las rutinas de apoyo [i]dwtflt no se muestran por no contener paso de mensajes, aunque como referencia se incluyen en los Listados E.17 y E.20.

```
function [ahav, ahdv, dhav, dhdv]=dwt2ParHost(img, h,g, tids)

img1=double(img(      1:liv2 ,:));  siz1=size(img1);       % Mitad sup. a primeros
img2=double(img(liv2+1:liv  ,:));  siz2=size(img2);       % Mitad inf. a segundos
                                                          % Enviar mitades imagen
if tids(1)~=pvm_mytid                                     % Puede que host trabaje también
cmd=['pvm_recv(pvm_parent,TAG);␣pvm_unpack(''img1'');' ,...
     'dwt2Par(img1,␣h,h,''T2'',␣0,␣''DL'');' ];
        pvm_initsend(RAW);  pvme_pack(h,  cmd);  pvm_send(tids(1),TAG);
        pvm_initsend(PLACE); pvm_pack(img1);     pvm_send(tids(1),TAG);
end

cmd=['pvm_recv(pvm_parent,TAG);␣pvm_unpack(''img2'');' ,...
     'dwt2Par(img2,␣h,g,''T1'',␣1,␣''DL'');' ];
        pvm_initsend(RAW);  pvme_pack(h,g,cmd);  pvm_send(tids(2),TAG);
        pvm_initsend(PLACE); pvm_pack(img2);     pvm_send(tids(2),TAG);
...
                                                          % Tiempo muerto o trabajamos?
if tids(1)==pvm_mytid                                     % Puede que host trabaje también
ahav=dwt2Par(img1, h,h,tids(2), 0, rf,dlbl);
else                                                      % Si tiempo muerto, recibir ahav
                                                          % Recibir Transformada
pvm_recv(tids(1),TAG);  pvm_unpack('ahav');  end
pvm_recv(tids(2),TAG);  pvm_unpack('ahdv');
pvm_recv(tids(3),TAG);  pvm_unpack('dhav');
pvm_recv(tids(4),TAG);  pvm_unpack('dhdv');
```

Listado 5.8: dwt2ParHost.m: Distribución de datos y recolección de resultados para análisis *wavelet* 2D B/W.

```
function res=dwt2Par(img, f1,f2, tid,slv)
                                                          % Longitudes, Constantes, etc
...
                                                          % Filtrado por filas, 1ª etapa
...
                                                          % Envío/Recepción intermedio
img2=tmp(slv*liv2+1:(slv+1)*liv2, : );                    % mitad no interesante
pvm_initsend(PLACE); pvm_pack(img2); pvm_send(tid,TAG);   % mandarla al otro

pvm_recv(tid,TAG); pvm_unpack('img2');                    % mitad del compañero
tmp((1-slv)*liv2+1:(2-slv)*liv2, : )=img2;                % pegarla antes/después

                                                          % Filtrado por columnas, 2ª etapa
...
                                                          % Envío Transformada al padre
if pvm_parent>0                                           % Por si el host también trabaja
  pvm_initsend(PLACE); pvm_pack(res);
  pvm_send(pvm_parent,TAG);
end
```

Listado 5.9: dwt2Par.m: Esclavo para análisis *wavelet* 2D B/W.

La subrutina de distribución y recolección inversa (Listado 5.10, completo en E.18) proporciona a cada proceso la aproximación o detalle correspondiente, sumando los resultados de los procesos homólogos de cada grupo.

```
function r=idwt2ParHost(ahav,ahdv,dhav,dhdv, h,g, tids, dlbl)

if tids(1)~=pvm_mytid                                    % Puede que host trabaje también
cmd=['pvm_recv(pvm_parent,TAG);␣pvm_unpack(''ahav'');',...
     'idwt2Par(ahav,␣h,h,' int2str(tids(2)) ',␣0,␣' DL ');'];
        pvm_initsend(RAW); pvme_pack(h, cmd); pvm_send(tids(1),TAG);
        pvm_initsend(PLACE); pvm_pack(ahav);   pvm_send(tids(1),TAG);
end
cmd=['pvm_recv(pvm_parent,TAG);␣pvm_unpack(''ahdv'');',...
     'idwt2Par(ahdv,␣h,g,' int2str(tids(1)) ',␣1,␣' DL ');'];
        pvm_initsend(RAW); pvme_pack(h,g,cmd); pvm_send(tids(2),TAG);
        pvm_initsend(PLACE); pvm_pack(ahdv);   pvm_send(tids(2),TAG);
...
                                                         % Tiempo muerto o trabajamos?
if tids(1)==pvm_mytid                                    % Puede que host trabaje también
tmp=idwt2Par(ahav, h,h, tids(2), 0, rf,dlbl);
else                                                     % Si tiempo muerto recibir ah sup
pvm_recv(tids(1),TAG); pvm_unpack('tmp'); end                     % ah    sup
pvm_recv(tids(3),TAG); pvm_unpack('res'); r(     1:liv2,:)=tmp+res;  % ah+dh sup

pvm_recv(tids(2),TAG); pvm_unpack('tmp');                          % ah    inf
pvm_recv(tids(4),TAG); pvm_unpack('res'); r(liv2+1:liv ,:)=tmp+res;  % ah+dh inf
```

Listado 5.10: idwt2ParHost.m: Distribución de datos y recolección de resultados para reconstrucción *wavelet* 2D B/W.

De nuevo se observa la posibilidad de que el computador servidor colabore en el cálculo como primer proceso del primer grupo, o de que mande la tarea a otro cliente. Los esclavos ejecutan el *script* inverso idwt2Par, correspondiente a una línea horizontal del esquema en la Figura 5.9 (Listado 5.11, completo en E.19).

```
function res=idwt2Par(img, f1,f2, tid,slv)
                                                         % Longitudes, Constantes, etc
...
                                                         % Deshacer filtr.cols. 2ª etapa
...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                       Sumar una mitad con mitad del otro
% Envío/Recepción  intermedio %           Ej.: tids(1) ahav->aha- mitades superiores ah
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                tids(2) ahdv->ahd/          inferiores ah
 una=tmp(     slv *liv2+1:( slv+1)*liv2 , : );            % mitad    interesante
otra=tmp((1-slv)*liv2+1:(2-slv)*liv2 , : );               % mitad no interesante
pvm_initsend(PLACE); pvm_pack(otra); pvm_send(tid,TAG);   % mandarla al otro

pvm_recv(tid,TAG); pvm_unpack('otra');                    % Recibir sobre otra
tmp = una+otra;                                           % y sumarla
                                                          % Deshacer filtr.filas, 1ª etapa
...
                                                          % Envío Transformada al padre
if pvm_parent>0                                           % Por si el host trabaja
  pvm_initsend(PLACE); pvm_pack(res);
  pvm_send(pvm_parent,TAG);
end
```

Listado 5.11: idwt2Par.m: Esclavo para reconstrucción *wavelet* 2D B/W.

De nuevo se muestran sólo las llamadas de paso de mensajes. Se vuelve a utilizar el *flag* slv identificando si este proceso es segundo en el grupo, en cuyo caso la media imagen que debe intercambiar con su pareja es la superior (calificada como *"no interesante"* en el código fuente).

En la Figura 5.11 se muestra la visualización XPVM de una ejecución típica de la aplicación bajo el entorno XWindows. El cluster estaba en *runlevel 5*, no controlado. Las imágenes original, transformada y reconstruida de Lena no se vuelven a presentar, siendo el resultado indistinguible del secuencial.

Más adelante, en la Tabla 5.2, se muestran las estadísticas de ejecución del programa en el cluster controlado, presentadas comparativamente con las de la aplicación MPITB para imagen en niveles de gris.

5.3.2 Versión paralela B/W MPITB

La aplicación MPITB sigue exactamente el mismo esquema que la PVMTB, reflejando por tanto el mismo paradigma maestro-esclavo. Recordamos brevemente los nombres de los *scripts* involucrados y la jerarquía de llamada que siguen:

```
Par -> dwt2ParHost -> dwt2Par -> dwtflt
  -> idwt2ParHost -> idwt2Par -> idwtflt
```

Las rutinas de análisis y reconstrucción 1D, [i]dwtflt, son idénticas a las usadas con PVMTB, al no incluir paso de mensajes. Del resto de las funciones se ofrece el código completo en los Listados del E.22 al E.26, mostrándose en la exposición que sigue únicamente algunos comandos MPITB destacados.

De nuevo, los procesos MATLAB esclavos en los computadores clientes son discriminados por la existencia de una variable de entorno, LAMPARENT en este caso, y forzados a ejecutar el habitual startup_Slv, conteniendo básicamente lo mostrado en el Listado 5.12.

```
info=MPI_Init;
[info parent] = MPI_Comm_get_parent;
[info NEWORLD]=MPI_Intercomm_merge(parent,1);           % ponerse detrás en comunicador
...
NUMCMDS=0; QUIT=0;                                       % Usar variables directamente
MPI_Recv(NUMCMDS,0,TAG,NEWORLD);                         % se espera NUMCMDS
MPI_Recv(QUIT,   0,TAG,NEWORLD);                         % y QUIT
...
cmd=' '; cmd=repmat(cmd,1,100);                          % Sitio para recibir string cmd
for indiceBucleMPI=1:NUMCMDS                             % Bucle de respuesta
  MPI_Recv(cmd,0,TAG,NEWORLD); eval(cmd);                % Recibir/evaluar comando
end
if QUIT, quit, end                                       % Si se indicó salir, hacerlo
```

Listado 5.12: startup_Slv.m: *script* de arranque para esclavos. Establece un protocolo con el maestro, el cual debe enviar inicialmente las variables NUMCMDS y QUIT, y posteriormente las NUMCMDS variables cmd anunciadas.

El Listado E.21 proporciona el código completo. Como vemos, se sigue el mismo protocolo NUMCMDS / QUIT. El código maestro Par (Listado E.22) también se dedica a organizar los tres niveles deseados de transformación. No lo mostramos por no contener paso de mensajes; recuérdese que invoca repetidamente a [i]dwt2ParHost, igual que en la versión PVM (Listado 5.7).

La distribución de análisis, dwt2ParHost (Listado 5.13, completo en E.23) debe mandar la mitad superior de la imagen al primer proceso de ambos grupos y la inferior al segundo.

De nuevo se observa la posibilidad de que el computador servidor colabore en el cálculo como primer proceso del primer grupo, o de que mande la tarea a otro cliente. Los esclavos ejecutan entonces el *script* dwt2Par, correspondiente a una línea horizontal del esquema en la Figura 5.8 (Listado 5.14, completo en E.24).

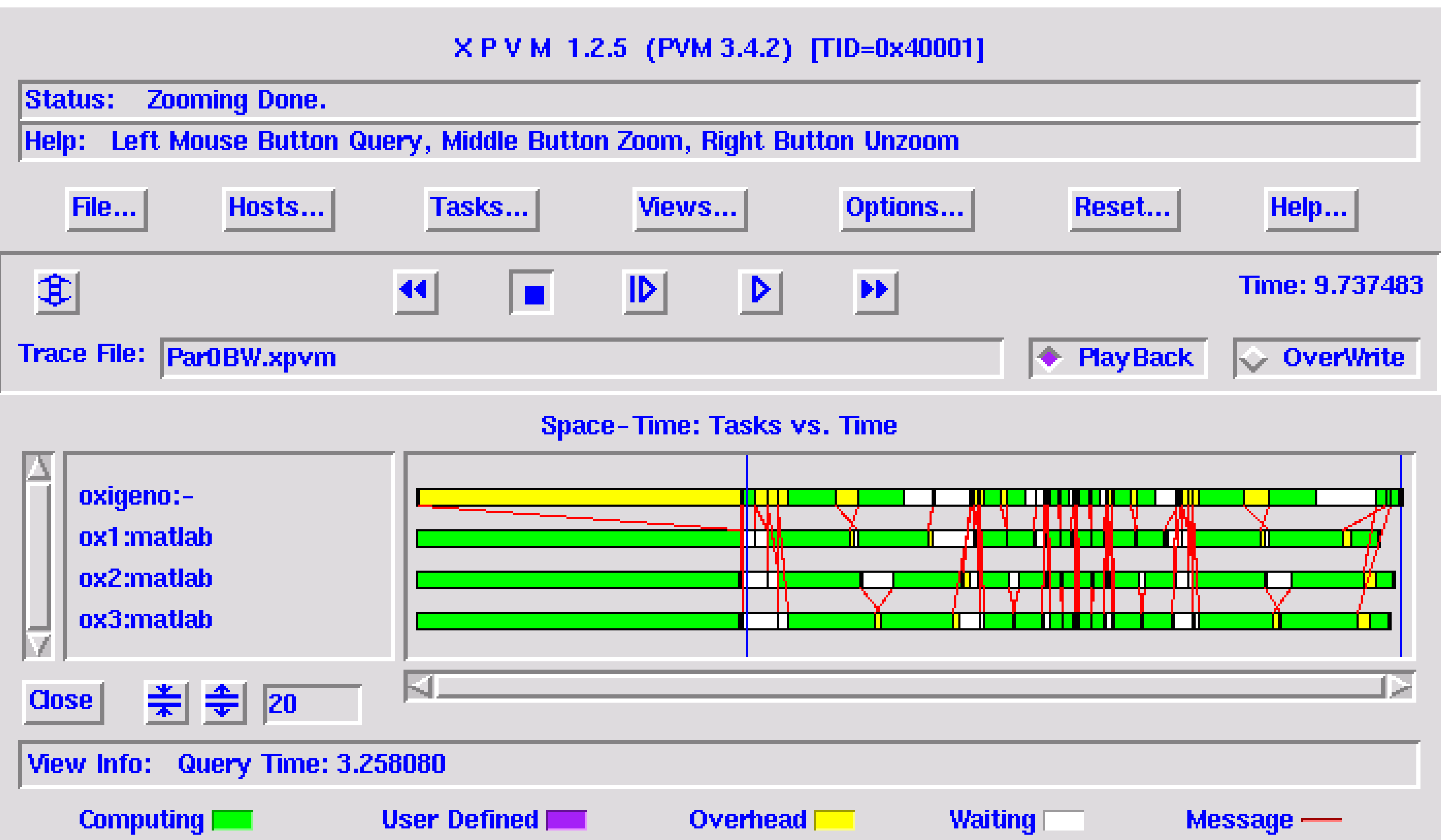


Figura 5.11: Visualización XPVM de una ejecución típica de la aplicación de transformada *wavelet* 2D B/W paralela (Par.m) usando PVMTB.

```
function [ahav, ahdv, dhav, dhdv]=dwt2ParHost(img, h,g, rnks)

img1=double(img(       1:liv2 ,:)); siz1=size(img1);          % Mitad sup. a primeros
img2=double(img(liv2+1:liv   ,:)); siz2=size(img2);          % Mitad inf. a segundos

if rnks(1)                                             % Puede que host trabaje también
cmd=['siz1=[0␣0];␣␣␣␣␣␣␣MPI_Recv(siz1,␣0,TAG,NEWORLD);' ,...
      'img1=zeros(siz1);␣MPI_Recv(img1,␣0,TAG,NEWORLD);' ,...
      'dwt2Par(img1,␣h,h,' T2 ',␣0,␣' DL ');' ];
                    MPE_Pack(h, cmd, buffer, 0, NEWORLD);      % sizeof(mxDOUBLE)==8 B
                    MPI_Send(buffer, rnks(1), TAG,NEWORLD);
                    MPI_Send(siz1 , rnks(1), TAG,NEWORLD);
                    MPI_Send(img1 , rnks(1), TAG,NEWORLD);
end
...
                                                       % Tiempo muerto o trabajamos ?
if ~rnks(1)                                            % Puede que host trabaje también
ahav=dwt2Par(img1, h,h,rnks(2), 0, rf, dlbl);
else                                                   % Si tiempo muerto, recibir ahav
                                                       % Recibir Transformada
ahav=zeros(liv2, lih2); MPI_Recv(ahav, rnks(1), TAG,NEWORLD); end
ahdv=zeros(liv2, lih2); MPI_Recv(ahdv, rnks(2), TAG,NEWORLD);
dhav=zeros(liv2, lih2); MPI_Recv(dhav, rnks(3), TAG,NEWORLD);
dhdv=zeros(liv2, lih2); MPI_Recv(dhdv, rnks(4), TAG,NEWORLD);
```

Listado 5.13: dwt2ParHost.m: Distribución de datos y recolección de resultados para análisis *wavelet* 2D B/W.

Nótese que ha sido posible usar una única llamada MPI_Sendrecv_replace en lugar de la pareja básica MPI_Send/MPI_Recv. Aparte de hacer más cómoda la codificación y lectura del programa, esta llamada MPI está implementada de manera que garantiza el no bloqueo de la operación (de intercambio, en este caso). Usando las rutinas básicas MPI_Send/MPI_Recv, es más fácil cometer errores de codificación que bloqueen algún proceso en recepción, o incluso en envío. La semántica PVM es algo más intuitiva, debido fundamentalmente a su llamada pvm_send() *asíncrona* (en terminología PVM).

Aunque un usuario podría no conocer en principio esta llamada, o no identificarla como apropiada en esta situación, incluir este tipo de llamadas de "más alto nivel" que el paso de mensajes básico (tras superar la curva de aprendizaje para MPI) produce en general aplicaciones que son más eficientes.

```
function res=dwt2Par(img, f1,f2, rnk,slv, wd,dlbl)
                                                   % Longitudes, Constantes, etc
...
                                                   % Filtrado por filas, 1ª etapa
...
                                                   % Envío/Recepción intermedio
img2=tmp(slv*liv2+1:(slv+1)*liv2, :);              % mitad no interesante
% MPI_Send          (img2,      rnk,TAG,NEWORLD);  % mandarla al otro
  MPI_Sendrecv_replace(img2,rnk,TAG,rnk,TAG,NEWORLD);
% MPI_Recv          (img2,      rnk,TAG,NEWORLD);  % mitad del compañero
tmp((1-slv)*liv2+1:(2-slv)*liv2, :)=img2;          % pegarla antes/después
                                                   % Filtrado por columnas, 2ª etapa
...
[info rank]=MPI_Comm_rank(NEWORLD);                % Envío Transformada al padre
if rank                                            % Por si el host también trabaja
  MPI_Send(res,0,TAG,NEWORLD);
end
```

Listado 5.14: dwt2Par.m: Esclavo para análisis *wavelet* 2D B/W.

La distribución inversa (Listado 5.15, completo en E.25) consiste en proporcionar a cada proceso esclavo la aproximación o detalle correspondiente, sumando los resultados de los procesos homólogos de cada grupo.

```
function r=idwt2ParHost(ahav,ahdv,dhav,dhdv, h,g, rnks, dlbl)

if rnks(1)                                          % Puede que host trabaje también
siz=size(ahav);
cmd=['siz␣=[0␣0];␣␣␣␣␣␣␣MPI_Recv(siz␣,␣0,TAG,NEWORLD);' ,...
     'ahav=zeros(siz);␣␣MPI_Recv(ahav,␣0,TAG,NEWORLD);' ,...
     'idwt2Par(ahav,␣h,h,' int2str(rnks(2)) ',␣0,␣' DL ');' ];
                 MPE_Pack(h,   cmd, buffer, 0 , NEWORLD);
                 MPI_Send(buffer, rnks(1), TAG,NEWORLD);
                 MPI_Send(siz,    rnks(1), TAG,NEWORLD);
                 MPI_Send(ahav,   rnks(1), TAG,NEWORLD);
end
...
                                                    % Tiempo muerto o trabajamos?
if ~rnks(1)                                         % Puede que host trabaje también
tmp=idwt2Par(ahav, h,h, rnks(2), 0, rf,dlbl);
else                                                % Si tiempo muerto recibir ah sup
MPI_Recv(tmp, rnks(1), TAG,NEWORLD); end                            % ah     sup
MPI_Recv(res, rnks(3), TAG,NEWORLD); r(      1:liv2 ,:)= tmp+res;   % ah+dh  sup

MPI_Recv(tmp, rnks(2), TAG,NEWORLD);                                % ah     inf
MPI_Recv(res, rnks(4), TAG,NEWORLD); r(liv2 +1:liv  ,:)= tmp+res;   % ah+dh  inf
```

Listado 5.15: idwt2ParHost.m: Distribución de datos y recolección de resultados para reconstrucción *wavelet* 2D B/W.

De nuevo se observa la posibilidad de que el computador servidor colabore en el cálculo como primer proceso del primer grupo, o de que mande la tarea a otro cliente. Los esclavos ejecutan el *script* inverso idwt2Par, correspondiente a una línea horizontal del esquema en la Figura 5.9 (Listado 5.16, completo en E.26).

```
function res=idwt2Par(img, f1,f2, rnk,slv, wd,dlbl)
                                                    % Longitudes, Constantes, etc
...
                                                    % Deshacer filtr.cols, 2ª etapa
...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%      Sumar una mitad con mitad del otro
% Envío/Recepción  intermedio %         Ej.: rnks(1) ahav->aha- mitades superiores ah
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%           rnks(2) ahdv->ahd/          inferiores ah
 una=tmp(    slv *liv2 +1:(slv+1)*liv2 , : );           % mitad     interesante
otra=tmp((1-slv)*liv2 +1:(2-slv)*liv2 , : );            % mitad  no interesante
% MPI_Send           (otra,        rnk, TAG,NEWORLD);   % mandarla al otro
  MPI_Sendrecv_replace(otra, rnk,TAG, rnk,TAG,NEWORLD);
% MPI_Recv           (otra,        rnk, TAG,NEWORLD);   % Recibir sobre otra
tmp = una+otra;                                         % y sumarla
                                                    % Deshacer filtr.filas, 1ª etapa
...
                                                    % Enviar Transformada
[info rank]=MPI_Comm_rank(NEWORLD);
if rank                                             % Por si el host trabaja
  MPI_Send(res,0,TAG,NEWORLD);
end
```

Listado 5.16: idwt2Par.m: Esclavo para reconstrucción *wavelet* 2D B/W.

Se vuelve a utilizar el *flag* slv identificando si este proceso es segundo en el grupo, en cuyo caso la media imagen que debe mandar a la pareja es la superior. El intercambio se realiza con la llamada MPI_Sendrecv_replace, evitando una pareja Send/Recv.

En la Figura 5.12 se muestran las gráficas de instrumentación de una ejecución típica de la aplicación bajo el entorno XWindows. El cluster estaba en *runlevel 5*, no controlado. Aunque LAM/MPI dispone de un visualizador XMPI similar a XPVM, no se ha podido ofrecer la visualización correspondiente porque no se soporta el trazado de llamadas MPI_Spawn actualmente. Cuando la característica sea implementada, se podrán obtener figuras similares a la 5.11 bajo MPITB.

La Tabla 5.2 muestra las estadísticas de ejecución (en el cluster controlado, *runlevel 3*) de esta aplicación MPITB para imagen en niveles de gris, presentadas junto con las de PVMTB para facilitar la comparación.

Speedups respecto a					**Absoluto** (segundos)			
9.16s Servidor		14.60s Cliente			PVMTB		MPITB	
PVMTB	MPITB	PVMTB	MPITB	**Computadores**	media	stdev	media	stdev
1.4620	1.5509	2.3300	2.4717	**servidor + 3 cl.**	6.2657	0.1788	5.9066	0.0491
1.3778	1.4732	2.1957	2.3478	**4 clientes**	6.6489	0.1554	6.2182	0.0629

Tabla 5.2: Resultados Paralelos para imagen B/W (obtenidos con el cluster desconectado del exterior).

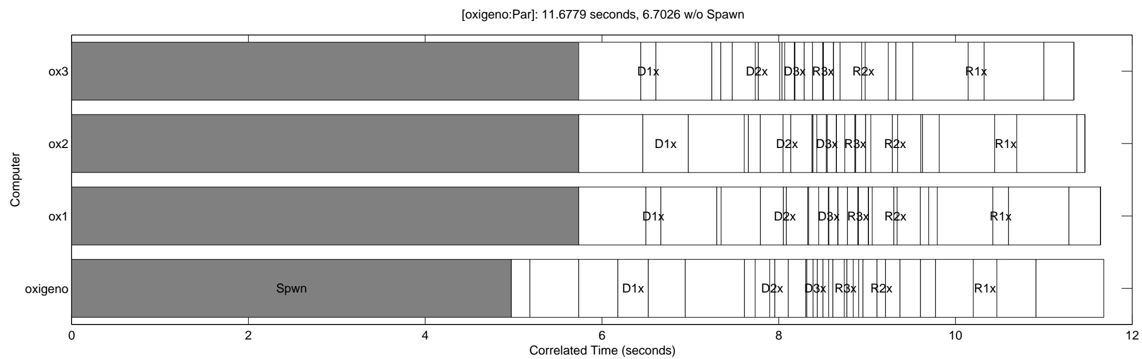
5.4 Algoritmo embarazosamente paralelo

La paralelización sugerida en la Figura 5.6 no implica ninguna modificación del algoritmo secuencial original, sino que incrementa la granularidad de la aplicación. El cálculo a realizar se triplica, repitiéndose para cada uno de los canales de color de la versión RGB de la imagen.

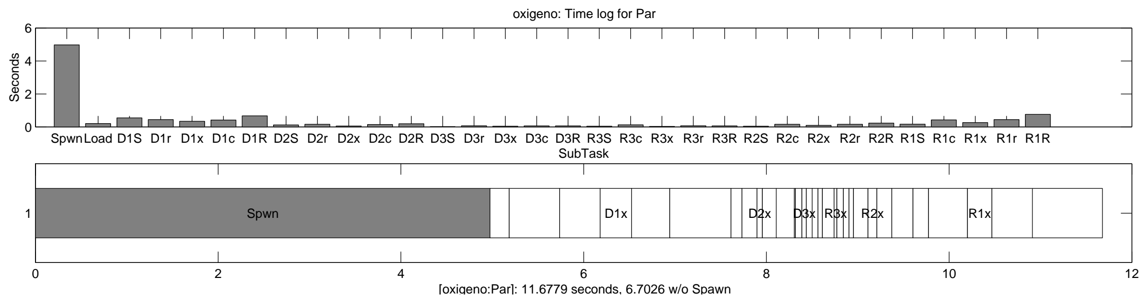
En contraposición a la alternativa estudiada en el apartado anterior, que pretendía servir de *test ácido* a los esfuerzos de paralelización e ilustrar hasta qué punto pueden degradarse las prestaciones si se necesitan transmitir mensajes relativamente grandes o se requieren muchos puntos de sincronización, ésta pretende servir de test neutro e ilustrar los *speedups* que pueden alcanzarse con un uso normal de las *Toolboxes*. La aplicación relativa al cálculo de π en el Capítulo 1 admitía una paralelización aún más ventajosa, al tener unos requisitos de comunicación mínimos. Muchas aplicaciones MATLAB son embarazosamente paralelas en concepto, consistiendo en un bucle que repite una simulación o cálculo sobre datos distintos.

Las necesidades de comunicación de la aplicación RGB se reducen a la distribución inicial de los canales de la imagen (*bit-planes* rojo, verde y azul) a los procesos esclavos, y la recolección de las transformadas y reconstrucciones correspondientes. Un argumento del programa, NUMSLV, permite determinar el número de computadores esclavos implicados. El código programado contempla la posibilidad de que un proceso maestro se limite a distribuir y recolectar los mensajes, involucrando un total de 4 procesos (1 maestro y 3 esclavos); aunque esto afecta naturalmente al *speedup* obtenido, prefiriéndose que coopere con otros 2 esclavos en el cálculo.

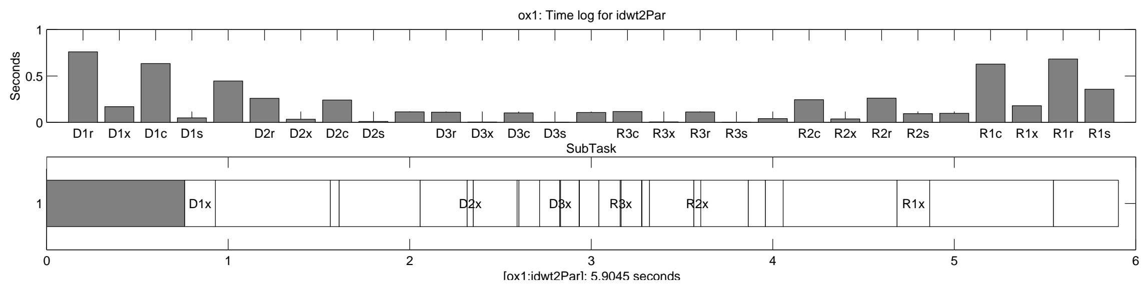
Restringiéndonos a esta última opción, cuando el proceso maestro está en el servidor a 400MHz (primer caso en la 4ª línea de la Figura 5.1) suele terminar sus etapas de análisis y reconstrucción antes que los esclavos, debiendo esperar sin realizar tarea útil. Cuando el maestro está en un computador cliente a 333MHz (segundo caso en la 4ª línea de la misma Figura) termina poco después de que lo hagan los esclavos, ya que empieza a calcular una vez que han sido transmitidos todos los datos, retrasando ligeramente la finalización de la aplicación.



(a) Tiempos correlacionados. Careciendo de visualización XMPI para MPI_Spawn, esta gráfica es un sucedáneo MPITB de la Figura 5.11 de PVMTB.



(b) Anotaciones de tiempo para el proceso maestro. Corresponde a la barra inferior de los tiempos correlacionados. Las etiquetas indican etapa y operación. Así, D1x es el intercambio intermedio en el primer nivel de descomposición. Ver los comandos record en los Listados del E.23 a E.26, (i)dwt2Par(Host).m.



(c) Anotaciones de tiempo para un proceso esclavo (ox1). Corresponde a las barras superiores de los tiempos correlacionados.

Figura 5.12: Datos recolectados mediante instrumentación de la transformada *wavelet* 2D B/W paralela usando MPITB.

Al igual que con la aplicación en niveles de gris, tanto la versión PVMTB como la MPITB de la transformada en color siguen la misma estructura maestro-esclavo, utilizando los mismos nombres para la rutinas de apoyo homólogas. Puede ser de utilidad conocer de antemano las dependencias entre las distintas rutinas, que se pueden resumir esquemáticamente como sigue:

```
ParRGB -> dwt2D3L -> dwt2 -> dwt
                  -> idwt2 -> idwt
```

Nótese que se reutilizan las rutinas de análisis y reconstrucción secuenciales 1D y 2D [i]dwt[2]. El programa principal (maestro) arranca los procesos esclavos, invoca repetidamente a la rutina esclava dwt2D3L para realizar el análisis bidimensional (2D) a 3 niveles (3L) deseado, y muestra la imagen original, su transformada y la reconstrucción junto con gráficas de las anotaciones de tiempo realizadas por el código de instrumentación.

5.4.1 Versión paralela RGB PVMTB

El Listado 5.17 (completo en el E.27) contempla la posibilidad de mandar el canal azul a otro proceso esclavo, aunque se prefiere ejecutar la otra rama condicional y colaborar en el cálculo. Por abreviar el listado, no se muestran las transmisiones previas a los otros dos esclavos, canales rojo lena (:,:,1) y verde lena (:,:,2).

```
lena=imread('lena_chuck','tif'); load h, load g                    % Imagen y Filtros
...

img=lena(:,:,3); if NUMSLV==3
  cmd=['pvm_recv(pvm_parent,TAG);␣ pvm_unpack(''img'');␣ img=double(img);' ,...
       'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');' ];
  pvm_initsend(RAW); pvme_pack(h,gd,gr,cmd); pvm_send(tids(3),TAG);
  pvm_initsend(PLACE); pvm_pack(img);        pvm_send(tids(3),TAG);
else
              img=double(img);
[trBLUE, rBLUE]=dwt2D3L(img,h,gd,gr);
end
                                                                   % Esperar Descomposición
pvm_recv(tids(1),TAG); pvm_unpack('trRED'  );
pvm_recv(tids(2),TAG); pvm_unpack('trGREEN'); if NUMSLV==3
pvm_recv(tids(3),TAG); pvm_unpack('trBLUE' ); end

pvm_recv(tids(1),TAG); pvm_unpack('rRED'  );                       % Recibir Re-composición
pvm_recv(tids(2),TAG); pvm_unpack('rGREEN'); if NUMSLV==3
pvm_recv(tids(3),TAG); pvm_unpack('rBLUE' ); end
```

Listado 5.17: ParRGB.m: Programa PVMTB maestro para transformada *wavelet* 2D RGB.

Obsérvese que el comando cmd especificado en el código incluye la recepción y desempaquetamiento previos a la invocación de dwt2D3L, y que la transmisión se realiza con el tipo de datos de la propia imagen, **uint8**, realizando los esclavos en paralelo la conversión a **double**.

Si sólo hay dos esclavos, el maestro ejecuta también el código esclavo dwt2D3L del Listado 5.18 (completo en E.28), en donde se reutilizan las rutinas secuenciales [i]dwt[2].

Obsérvese la similitud de dicho código esclavo para cada canal de color con el programa secuencial original para niveles de gris del Listado 5.1. La única diferencia estriba en la sección intermedia de recepción de la transformada (por parte del maestro) o agrupamiento y envío de la misma (por parte de los esclavos). Cuando se solicita visualización de imágenes, la transformada se ecualiza (ver math en Listado E.11) igual que se hizo en las versiones secuenciales B/W (Listado E.1) y RGB (Listado E.12), al objeto de evitar una poco informativa imagen negra.

```
function [transform,reconstr]=dwt2D3L(img,h,gd,gr)
...
[a1 dh1 dv1 dd1]=dwt2(img,h,gd);                          % Descomposición 3 niveles
[a2 dh2 dv2 dd2]=dwt2( a1,h,gd);
[a3 dh3 dv3 dd3]=dwt2( a2,h,gd);

if withmath,      transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);
else,             transform=     {a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1};
end

if ~parent                                                % Host: recibir ya
  pvm_recv(tids(1),TAG); pvm_unpack('trRED'   );
  pvm_recv(tids(2),TAG); pvm_unpack('trGREEN');
else                                                      % Esclavos: enviar transformada
  if withmath,    pvm_initsend(PLACE); pvm_pack(transform);
  else,           pvm_initsend(RAW);   pvme_pack(transform);
  end,            pvm_send(parent,TAG);
end

r2=idwt2(a3,dh3,dv3,dd3, h,gr);                           % Reconstrucción 3 niveles
r1=idwt2(r2,dh2,dv2,dd2, h,gr);
r0=idwt2(r1,dh1,dv1,dd1, h,gr);

if parent                                                 % Esclavos: enviar reconstrucción
  pvm_initsend(PLACE); pvm_pack(reconstr); pvm_send(parent,TAG);
end
```

Listado 5.18: dwt2D3L: Esclavo para análisis *wavelet* 2D a 3 niveles de un *bitplane* de imagen RGB.

En cualquier caso, se visualice o no la transformada, resulta cómodo agruparla para realizar un único envío, evitando la codificación de múltiples transmisiones: en este caso 1 aproximación más 3x3 detalles de tamaños progresivamente mayores (dh3...dd1). Como ya se comentó en el Capítulo 1 (acabando el Apartado 1.6.1), estas aplicaciones embarazosamente paralelas están dominadas por el tiempo de *setup* si los tamaños de los mensajes son lo suficientemente pequeños. En el tercer nivel, la aproximación y detalles son de 64x64 **double**s, o $2^6 2^6 2^3 = 2^{15} = 32$KB, con un tiempo de transmisión del orden de 3ms frente a una latencia sólo un orden de magnitud inferior. Realizar 10 envíos de ese tamaño casi equivaldría a un mensaje adicional.

Tal vez extrañe que se haya separado en el código fuente la agrupación (o llamada a math) de la transmisión, repitiendo la comprobación condicional withmath. Cuando el maestro coopera en el cálculo de la transformada, acaba la descomposición antes que los esclavos y se quedaría bloqueado en el pvm_recv de sus transformadas (condicional **if** ~parent). En esta situación es conveniente realizar la tarea de agrupar o ecualizar la transformada antes de bloquearse en el pvm_recv, pudiendo quedar este tiempo de cálculo parcialmente oculto.

En la Figura 5.13 se muestra la visualización XPVM de una ejecución típica de la aplicación bajo el entorno XWindows, comprobándose que dicho tiempo queda completamente oculto: de hecho el servidor sigue quedándose bloqueado al recibir tanto las transformadas como las reconstrucciones (segmentos blancos en la línea superior). El cluster estaba en *runlevel 5*, el entorno "no controlado", como venimos denominándolo (*runlevel 3* no ofrece servicio X11). Las imágenes original, transformada y reconstruida de Lena no se vuelven a presentar, ya que el resultado paralelo es indistinguible del secuencial.

Más adelante, en la Tabla 5.3, se muestran las estadísticas de ejecución del programa en el cluster controlado, presentadas comparativamente con las de la aplicación MPITB para imagen en color.

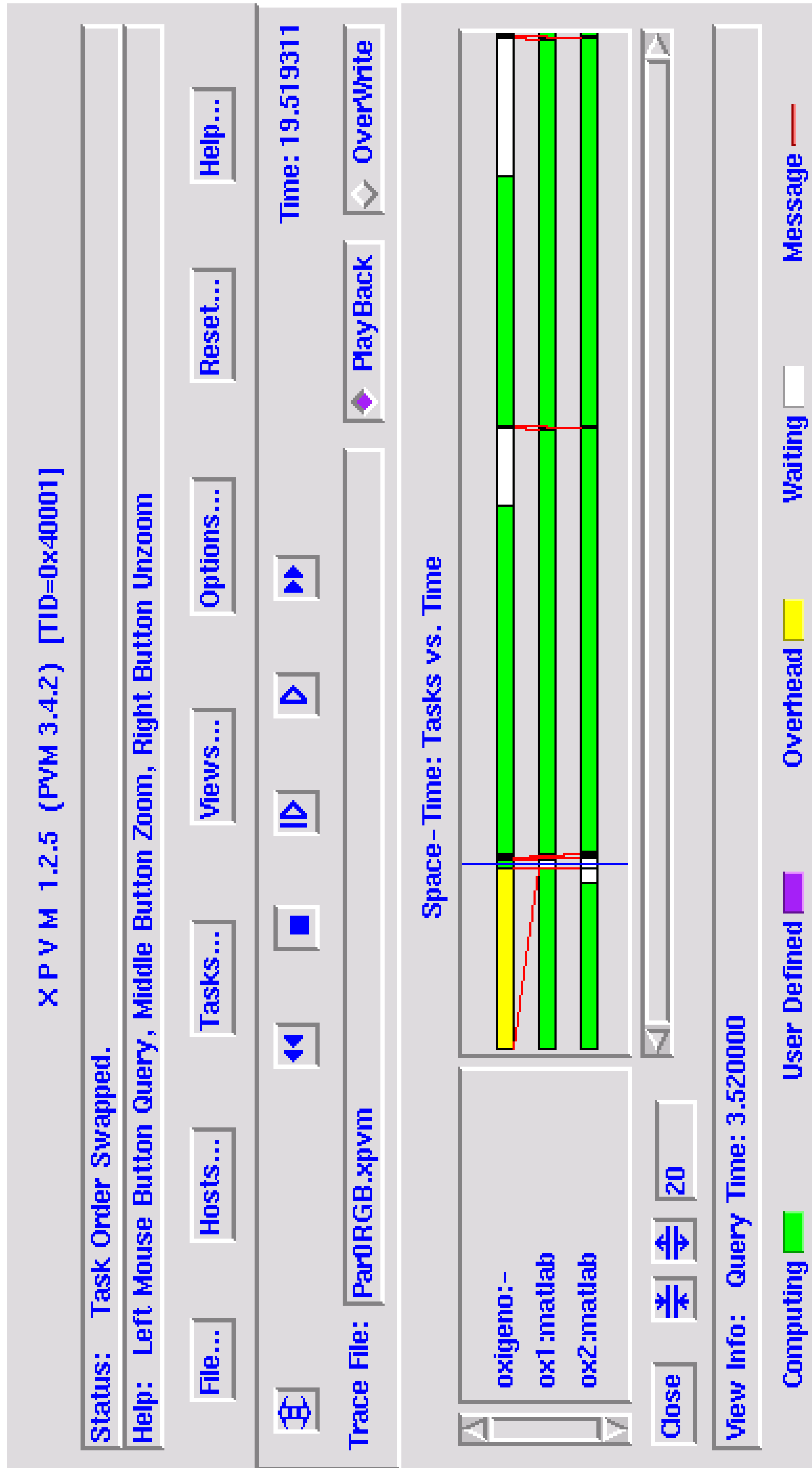


Figura 5.13: Visualización XPVM de una ejecución típica de la aplicación de transformada *wavelet* 2D RGB paralela (ParRGB.m) usando PVMTB.

5.4.2 Versión paralela RGB MPITB

El Listado 5.19 (completo en el E.29), es una traducción a MPI del programa PVM recién estudiado. Se emplea una versión ligeramente distinta del *script* **startup** que permite recibir otras variables además de cmd, por el método de empaquetarlas previamente usando MPI_Pack.

```
lena=imread('lena_chuck','tif'); load h, load g          % Imagen y Filtros
...
img=lena(:,:,3);  siz=size(img);  if NUMSLV==3
cmd=['siz=[0␣0];␣MPI_Recv(siz,␣␣0,TAG,NEWORLD);␣img(siz(1),siz(2))=uint8(0);',...
                 'MPI_Recv(img,␣␣0,TAG,NEWORLD);␣img=double(img);',...
      'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');'  ];
                 MPE_Pack(h,gd,gr,cmd,pbuf,0,NEWORLD);
                 MPI_Send(pbuf,        3,TAG,NEWORLD);
                 MPI_Send(siz,         3,TAG,NEWORLD);
                 MPI_Send(img,         3,TAG,NEWORLD);
end
                                                    % Dejar preparado Irecv Descomp.
[i Rreq]=MPI_Irecv(trRED  ,1,TAG,NEWORLD);
[i Greq]=MPI_Irecv(trGREEN,2,TAG,NEWORLD);  if NUMSLV==3
[i Breq]=MPI_Irecv(trBLUE ,3,TAG,NEWORLD);  end

if NUMSLV==2                                        % Trabajar durante análisis
                     img=double(img);               % Se calcula descomp. y reconstr.
[trBLUE, rBLUE]=dwt2D3L(img,h,gd,gr);               % MPI_Test? no! MPI_Wait mejor
end

MPI_Wait(Rreq);                                     % Esperar Descomposición
MPI_Wait(Greq);  if NUMSLV==3
MPI_Wait(Breq);  end

MPI_Recv(rRED,   1,TAG,NEWORLD);                    % Recibir Reconstrucción
MPI_Recv(rGREEN,2,TAG,NEWORLD);  if NUMSLV==3
MPI_Recv(rBLUE, 3,TAG,NEWORLD);  end
```

Listado 5.19: ParRGB.m: Programa MPITB maestro para transformada *wavelet* 2D RGB.

Al estar la rutina MPI_Recv orientada a su uso directo sin paso de desempaquetamiento previo que permita a la *Toolbox* crear el array de recepción si no existe (como se hace en PVMTB), se debe enviar previamente información sobre el tamaño. El *string* cmd especificado resuelve esta diferencia con PVM. El reparto de datos inicial se realiza con tipo de datos **uint8**, y la conversión a **double** se realiza en paralelo por los esclavos.

Otra diferencia notable de LAM/MPI respecto a PVM es que con ruta directa un envío no progresa si el receptor no ejecuta MPI_Recv, ya que el *daemon* no interviene en absoluto (en PVM el progreso depende del empaquetamiento y del tamaño del array). En el diseño inicial de esta aplicación bajo MPITB intentamos que el servidor continuara con la reconstrucción, dejando pendientes las correspondientes recepciones no bloqueantes MPI_Irecv.

Desafortunadamente, con ruta directa MPI_Irecv no progresa si no se chequea con MPI_Test o MPI_Iprobe, básicamente por el mismo motivo. Para esta aplicación es fundamental que los esclavos no se queden bloqueados al devolver sus respectivas transformadas, puesto que aún les queda por calcular las reconstrucciones y son más lentos que el servidor. Correspondientemente, se programó que el servidor realizara tras cada etapa de reconstrucción (a niveles 3, 2 y 1) una llamada MPI_Test sobre ambas peticiones Rreq/Greq.

Pruebas preliminares mostraron que el servidor termina bloqueándose en los MPI_Wait tras concluir todos los cálculos. Además, después de recibir las transformadas debe esperar aún a que los esclavos calculen la reconstrucción y la envíen de vuelta.

Tras consultar con los autores de LAM/MPI (ver la pregunta msg01428.php y la respuesta msg01436.php en el archivo de la lista de distribución LAM http://www.lam-mpi.org/MailArchives/lam/) se nos recomendó no ejercitar la cola de mensajes inesperados de LAM, motivo por el cual el código esclavo del Listado 5.20 (completo en E.30) es conceptualmente idéntico al de PVM.

```
function [transform,reconstr]=dwt2D3L(img,h,gd,gr)
...
[a1 dh1 dv1 dd1]=dwt2(img,h,gd);                    % Descomposición 3 niveles
[a2 dh2 dv2 dd2]=dwt2( a1,h,gd);
[a3 dh3 dv3 dd3]=dwt2( a2,h,gd);

if withmath,    transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);
else,           transform=    {a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1};
end

if ~rank                                            % Host: recibir ya
  MPI_Wait(Rreq);                 % Salvo motivos de ámbito MATLAB, hubiera resultado
  MPI_Wait(Greq);                 % ... igual de eficiente y más simple usar MPI_Recv
else                                                % Esclavos: enviar transformada
  if withmath,  MPI_Send(transform, 0,TAG,NEWORLD);
  else,         MPI_Pack(transform,pbuf,0,NEWORLD);
                MPI_Send        (pbuf,0,TAG,NEWORLD);
  end
end

r2=idwt2(a3,dh3,dv3,dd3, h,gr);                     % Reconstrucción 3 niveles
r1=idwt2(r2,dh2,dv2,dd2, h,gr);
r0=idwt2(r1,dh1,dv1,dd1, h,gr);

if rank                                             % Esclavos: enviar reconstrucción
  MPI_Send(reconstr,0,TAG,NEWORLD);
end
```

Listado 5.20: dwt2D3L: Esclavo para análisis *wavelet* 2D a 3 niveles de un *bitplane* de imagen RGB.

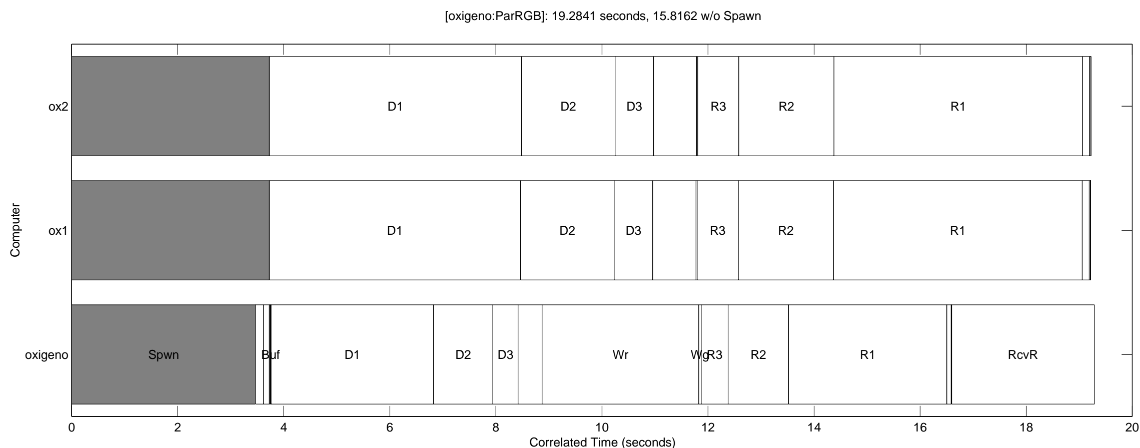
De nuevo se devuelve la transformada en una única transmisión, y si se solicitó visualización se normaliza previamente para no realizar secuencialmente dicha tarea en el código maestro.

En la Figura 5.12 se muestran las gráficas de instrumentación de una ejecución típica de la aplicación bajo el entorno XWindows. El cluster estaba en *runlevel 5*, no controlado. De nuevo no se ha podido ofrecer la visualización XMPI correspondiente porque dicha herramienta no soporta el trazado de llamadas MPI_Spawn aún. Cuando la característica sea implementada, se podrán obtener bajo MPITB figuras similares a las mostradas para PVMTB.

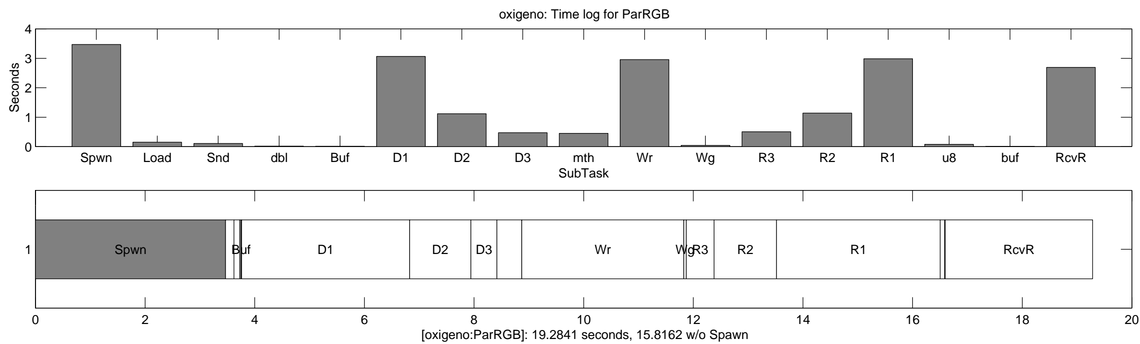
La Tabla 5.3 muestra las estadísticas de ejecución (en el cluster controlado, *runlevel 3*) de esta aplicación MPITB para imagen en color, presentadas junto con las de PVMTB para facilitar la comparación.

Speedups respecto a					**Absoluto** (segundos)			
27.43s Servidor		44.06s Cliente			PVMTB		MPITB	
PVMTB	MPITB	PVMTB	MPITB	**Computadores**	media	stdev	media	stdev
1.7402	1.7421	2.7948	2.7978	**servidor + 2 cl.**	15.7635	0.1615	15.7466	0.0448
1.7187	1.7223	2.7601	2.7659	**3 clientes**	15.9612	0.0380	15.9278	0.0356

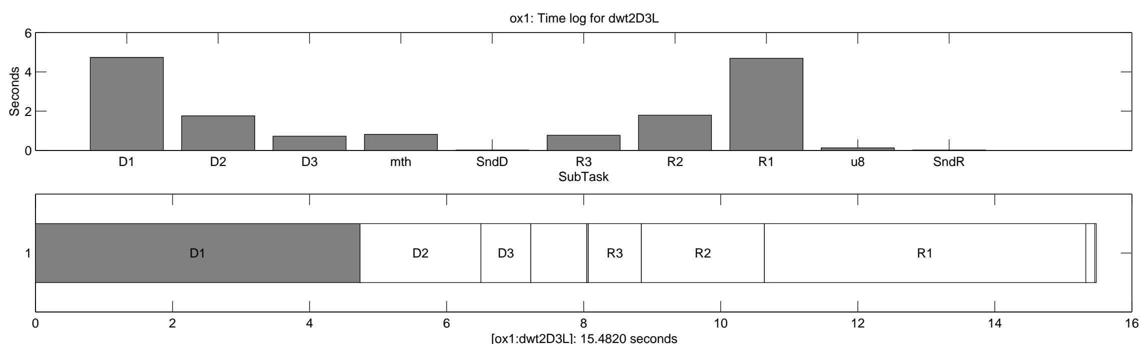
Tabla 5.3: Resultados Paralelos para imagen RGB (obtenidos con el cluster desconectado del exterior).



(a) Tiempos correlacionados. Careciendo de visualización XMPI para MPI_Spawn, esta gráfica es un sucedáneo MPITB de la Figura 5.13 de PVMTB.



(b) Anotaciones de tiempo para el proceso maestro. Corresponde a la barra inferior de los tiempos correlacionados. Las etiquetas indican la operación realizada. Así, Wr es la espera MPI_Wait(Rreq). Ver los comandos record en el Listado E.30, dwt2D3L.m.



(c) Anotaciones de tiempo para un proceso esclavo (ox1). Corresponde a las barras superiores de los tiempos correlacionados.

Figura 5.14: Datos recolectados mediante instrumentación de la transformada *wavelet* 2D RGB paralela usando MPITB.

5.5 Discusión de los resultados

Para facilitar la comparación, la misma información de las Tablas 5.2 y 5.3, se presenta ahora reorganizada en la Tabla 5.4 para separar todas las mediciones realizadas con PVMTB de las homólogas MPITB. Recordemos que los algoritmos B/W y RGB habían sido diseñados para utilizar 4 y 3 computadores, respectivamente. La Figura 5.1 resumía esquemáticamente las distintas mediciones realizadas.

Tabla 5.4: *Speedups* correspondientes a las Tablas 5.2 y 5.3.

PVMTB					MPITB			
Par respecto a		**ParRGB** respecto a			**Par** respecto a		**ParRGB** respecto a	
servidor	cliente	servidor	cliente	**Computadores**	servidor	cliente	servidor	cliente
1.46	2.33	1.74	2.80	**servidor+clientes**	1.55	2.47	1.74	2.80
1.38	2.20	1.72	2.76	**todos clientes**	1.47	2.35	1.72	2.77

Los *speedups* son naturalmente menores que dicho número de computadores, aproximándose a la linealidad cuando se reduce la comunicación y los procesos están calculando más tiempo. Esto favorece a la aproximación RGB, y a las aplicaciones de alta granularidad en general.

Salvo los *speedups* entre clientes (los que aparecen remarcados en la tabla), resultado de dividir el tiempo secuencial en un computador cliente (columnas *"respecto a cliente"*) entre el de la ejecución paralela sin usar el computador servidor (fila *"todos clientes"*), todos los demás *speedups* son objetables, ya que se está usando como maestro un computador más rápido (fila *"servidor+clientes"*), o como clientes computadores más lentos que el de referencia (columnas *"respecto a servidor"*). También se podría argumentar que en nuestro caso el servidor soporta el servicio NFS a los clientes, aunque intentar estimar objetivamente dicha carga durante las mediciones sólo conduciría a alterar la propia medición del *speedup*.

El *speedup* de la alternativa RGB está limitado por los 14.60s que tarda en secuencial la carga de cálculo encomendada a los procesos esclavos ejecutándose en los computadores clientes, y no por la relativamente pequeña diferencia de prestaciones entre PVMTB y MPITB. Este hecho queda claramente manifiesto en la visualización XPVM de la Figura 5.13 y delatado en la subtabla *"Speedups"* de la Tabla 5.3, en la que MPITB no consigue superar significativamente a PVMTB. Esta información se repite en las columnas ParRGB (a la derecha) para PVMTB y MPITB en la Tabla 5.4.

Comparando las columnas Par (izquierda) se observa mayor diferencia, ya que la aplicación para imágenes en niveles de gris tiene mayor proporción comunicación/cálculo que la de color ParRGB, influyendo por tanto las mejores prestaciones de MPITB en el rango de tamaños de mensaje utilizados. Se debe recordar que Par utilizaba 4 computadores, y el máximo *speedup* obtenido es cercano a 2.5, lo cual reduce la eficiencia a un discreto 0.62.

LAM/MPI puede ahorrarse una copia de memoria inevitable bajo PVM en recepción, debido al distinto funcionamiento de las respectivas llamadas:

```
bufid=pvm_recv(tid, tag);                    MPI_Recv(data, cnt, type, src, tag, comm, stat)
pvm_upk<type>(data, nitem, stride)
```

en donde el usuario puede ofrecer directamente el almacenamiento (data) a LAM/MPI, mientras que con PVM debe desempaquetar el mensaje desde el buffer de recepción (bufid).

Volviendo a la aproximación RGB, hay que destacar que se probó una versión tentativa que no bloquea al maestro en la recepción de la transformada, dejándole progresar en su etapa de reconstrucción (etapas etiquetadas como R3/R2/R1 en las Figuras 5.14(a) y 5.14(b)). Sin embargo, al usar ruta directa la recepción no progresa, obligando al proceso maestro a comprobar con MPI_Test o MPI_Iprobe la cola LAM de mensajes inesperados. Los propios autores de LAM desaconsejan la técnica, como se comentó en el Apartado 5.4.2. De hecho, se obtuvieron *speedups* peores que con la aproximación B/W.

Para esta aplicación era mucho más importante evitar la espera de los esclavos en el MPI_Send de la transformada. Una vez conseguido esto, la limitación de los 14.6s de tiempo de ejecución en los esclavos es obviamente insalvable. Dicho de otra manera, no se pierde nada por tener al maestro esperando en la recepción intermedia de la transformada.

Cuando se consigue esta situación con una aplicación embarazosamente paralela, prácticamente no hay diferencia entre escoger PVMTB y MPITB.

5.6 Conclusiones

En este capítulo se ha estudiado un ejemplo de aplicación real de las *Toolboxes* paralelas PVMTB y MPITB presentadas en esta memoria. La aplicación está basada en el análisis *wavelet* de imágenes, técnica que ha sido reconocida como herramienta preferida para compresión de vídeo y de imagen, y de la cual se están explorando aún sus múltiples aplicaciones para investigación en el campo de Visión por Computador ([44]).

Con tareas de alta granularidad obviamente paralelizables como el análisis *wavelet* RGB aquí estudiado se consiguen *speedups* casi lineales. Para este tipo de tareas, el tiempo de comunicación es muy reducido en comparación con el tiempo de cálculo. En esta situación, el *speedup* viene determinado mayormente por la carga de trabajo asignada a cada esclavo. Se debe procurar que la carga esté equilibrada, de manera que ningún proceso esclavo quede inactivo mientras que otro sigue calculando. Usando un servidor algo más rápido que los clientes para el proceso maestro, el reparto escalonado de los datos iniciales consigue que los resultados se obtengan con el mismo escalonamiento, estando el servidor listo para la recepción con suficiente antelación debido a su mayor velocidad, incluso cuando coopera en el cálculo. En estas condiciones no tiene mucha importancia escoger entre PVMTB o MPITB, obteniéndose con ambas muy buenos resultados.

Con tareas de menor granularidad decrece la eficiencia, y se empiezan a manifestar las diferencias entre usar PVM y MPI como mecanismo subyacente de paso de mensajes. Sin embargo, no hay que sobrevalorar esta pérdida de eficiencia. La conversión de tipo **double↔uint8**, y por consiguiente también la reserva de memoria de tipo **uint8**, son operaciones frecuentemente usadas en aplicaciones de procesamiento de señal e imagen, como es el caso de los análisis *wavelet* estudiados en este capítulo. Al crear expresiones complejas con arrays de gran tamaño bajo MATLAB es relativamente sencillo incurrir en costes adicionales debidos a una involuntaria reserva de memoria provocada por un resultado intermedio que se hubiera podido evitar expresando el cálculo de una forma alternativa. Si la aplicación opera con arrays de gran tamaño, repasar cuidadosamente estas operaciones en busca de expresiones alternativas y por supuesto vectorizadas conduce a ahorros de tiempo muy superiores a la relativamente pequeña diferencia de

prestaciones entre PVMTB y MPITB.

Como trabajo futuro se considera el estudio de algoritmos tipo “bolsa de trabajo” (una forma de paralelismo funcional de bucle), en los que el servidor se dedica únicamente a repartir carga computacional y recolectar resultados.

En el caso del análisis *wavelet* resulta obvio definir la línea de imagen como unidad de carga computacional, enviando primero a los N esclavos en la “granja de computadores” ambos filtros de análisis (H y G) para evitar su retransmisión, y repartiendo nueva línea a cada proceso conforme devuelva el resultado anterior.

Tras un periodo de sincronización en el que se pierden como máximo N-1 *slots* de trabajo esperando la línea filtrada del último cliente ocupado, se sustituyen los filtros de análisis por los de reconstrucción y se redefine la unidad de carga como la línea vertical de los dos resultados intermedios recolectados, ah/dh. El proceso de reconstrucción se resolvería homólogamente.

Otras aproximaciones [97] consideran segmentar la imagen original y repartir los segmentos, pudiendo presentarse los conocidos efectos de borde a menos de que se envíe también información de solapamiento con los segmentos vecinos.

Las aproximaciones estudiadas en este capítulo fueron diseñadas con el propósito de que resultaran en un código fuente relativamente simple e ilustraran el uso de PVMTB y MPITB, al mismo tiempo que mostraran los *speedups* que se pueden esperar bajo distintas situaciones típicas. Existen aplicaciones que admiten paralelizaciones más eficientes que el algoritmo *wavelet* RGB debido a sus menores requisitos de comunicación (como el cálculo de π en el Capítulo 1), y por supuesto otras con paralelizaciones menos eficientes que el algoritmo B/W. El algoritmo de *bolsa de trabajo* estaría en esta última categoría (con una eficiencia seguramente mucho menor que 0.6), aunque su mejor escalabilidad permitiría probablemente superar el *speedup* de 2.5 utilizando todos los computadores del cluster.

Aunque la eficiencia de aproximaciones como la “bolsa de trabajo” o la “segmentación con solapamiento” recién descritas se alejarían más de la unidad, por presentar no sólo mayor comunicación sino también mayor relación comunicación/cómputo, este tipo de aproximaciones que escalan bien, admitiendo paralelismo de un número arbitrario de procesadores, permitirían superar los *speedups* obtenidos con las aproximaciones estudiadas en este capítulo, incluso contando con tan sólo 8 computadores clientes como sucede en nuestro cluster *oxígeno*.

En este tipo de aplicaciones que requieren mayor comunicación con mensajes de menor tamaño, cabe esperar que se manifieste más claramente la menor latencia de MPITB como sucede con el análisis B/W; comentario hecho con la reserva de que también existen tamaños de mensaje en las dos primeras UDPs para los cuales PVM es más eficiente que MPI, dependiendo de la configuración UDPMAXLEN/MAXNMSGLEN utilizada, como se vio en el Capítulo 2.

Capítulo 6

Conclusiones

Conclusiones y principales aportaciones

El objetivo inicial del trabajo era sólo proporcionar un entorno apropiado para la **simulación de modelos de Visión por Computador**, aunque finalmente ha derivado en un entorno de desarrollo más general. En el último capítulo se ha mostrado la utilización de **PVMTB y MPITB** en Visión por Computador, programando varias aplicaciones de análisis *wavelet*, con las cuales se han podido observar las ganancias en prestaciones que se pueden esperar de su uso. Se resumen a continuación las principales aportaciones y las conclusiones obtenidas del estudio realizado.

- Se han desarrollado dos entornos aptos para prototipar aplicaciones paralelas bajo MATLAB, basados ambos en paso de mensajes, uno (PVMTB) en el popular sistema PVM y el otro (MPITB) en el estándar MPI. La contribución es relevante para el campo de Computación Paralela, dada la creciente popularidad de los clusters como plataforma asequible y viable para dicho campo y la amplia extensión del entorno MATLAB en docencia, investigación e industria.

  Una vez se establezca el proceso automantenido según el cual el software y hardware son elementos de uso común (COTS) dada la gran cantidad de usuarios que cooperan en desarrollar, utilizar, depurar y popularizar el primero y adquirir instalaciones del segundo, toda vez que el número de usuarios se incrementa dada la accesibilidad del software y hardware, sólo cabe esperar un progresivo refinamiento de la plataforma con el tiempo. Nuestra contribución es de ayuda en este sentido, al atraer usuarios MATLAB al campo de la Computación Paralela, proporcionándoles una herramienta adecuada a su entorno.

- Las *Toolboxes* desarrolladas, PVMTB y MPITB, presentan una pérdida de prestaciones (*overhead*) aceptable frente a la utilizacion directa de llamadas PVM y MPI bajo C. La pérdida puede llegar a ser inapreciable, dependiendo del tamaño del mensaje.

  También respetan el patrón de llamada de las rutinas originales, salvo cuando se entra en conflicto o incompatibilidad con el entorno MATLAB. El soporte de las respectivas bibliotecas es total en PVMTB y casi total bajo MPITB. Estas dos últimas características las hacen ideales como herramientas docentes para explicar PVM y MPI, pudiéndose utilizar en la infraestructura LAN de un campus universitario.

La conjunción de las tres características, bajo *overhead*, patrón de llamada inalterado y cobertura completa, las hace ideales para investigación, pudiéndose utilizar en un cluster dedicado. Los investigadores podrían así reducir tiempo de desarrollo bajo el entorno de rápido prototipado disponible bajo MATLAB, que incluye herramientas para desarrollo de GUIs, visualización de datos, generación de gráficas, etc.

- El diseño de PVMTB y MPITB es compatible con la opinión del propio fabricante de MATLAB sobre el posible uso de paralelismo en su entorno. No se intenta ejecutar varios procesos MATLAB en las múltiples CPUs de un SMP, sino en distintos computadores de un cluster.

- La comparación con los trabajos previos arroja la conclusión de que ninguno de ellos presenta una funcionalidad tan completa ni un *overhead* tan reducido. Estas diferencias se manifiestan incluso en los más sencillos ejemplos de aplicaciones embarazosamente paralelas, a pesar de la poca comunicación (y por tanto poca oportunidad de revelar diferencias) que estas aplicaciones presentan.

  El bajo *overhead* de nuestas *Toolboxes* las hace apropiadas para prototipar aplicaciones HPC. El prototipo podría ser posteriormente compilado para evitar la pérdida de prestaciones debida a la naturaleza interactiva de MATLAB. PVMTB y MPITB también se pueden utilizar simplemente para reducir el tiempo de ejecución en el entorno interpretado.

  Las *Toolboxes* proporcionan interfaz MATLAB directo a las diversas llamadas de la correspondiente biblioteca, en lugar de ofrecer comandos de alto nivel y uso específico. Cualquier aplicación HPC que se pueda diseñar con PVM o MPI también se puede prototipar con PVMTB o MPITB.

- Se han evaluado las prestaciones de PVM y MPI en nuestro cluster de PCs. Hemos desarrollado una serie de modelos numerados del 0 al 3 en orden de complejidad, costes de evaluación y méritos predictivos crecientes. Estos modelos condensan en unos pocos parámetros una mayor información que la proporcionada por los habitualmente ofrecidos, ancho de banda y latencia.

- El diseño de PVMTB y MPITB se basa en una clasificación y estudio previo de las distintas llamadas en la respectiva biblioteca. Esto permite una efectiva reutilización de código, resultando en un diseño robusto y fácilmente depurable y mantenible. Los trabajos previos se basan invariablemente en un sistema de “directorio” que añade un paso intermedio adicional de llamada a la rutina directorio. Nuestas *Toolboxes* presentan un fichero MEX independiente para cada llamada de paso de mensajes.

  Un detalle técnico de nuestras *Toolboxes* es que ambas se enlazan dinámicamente con la respectiva biblioteca. Ningún trabajo anterior exige esta condición, siendo evidente en algunos casos (DP-TB) que el objetivo del directorio es evitar una ocupación en disco excesiva.

  Nuestro diseño no incurre en copia de memoria adicional para los tipos de datos MATLAB compatibles con PVM o MPI. De nuevo, ningún trabajo anterior presenta esta característica.

  La conjunción de estas tres propiedades explica las superiores prestaciones de nuestras *Toolboxes* en comparación con todos los (parciales) trabajos anteriores, y sugiere un posible motivo del bajo nivel de cobertura de las respectivas bibliotecas que exhiben.

Dada su extensión, se ha preferido posponer a un apartado posterior (6.2) una relación detallada de las características de PVMTB y MPITB más notables.

- Con tareas de alta granularidad obviamente paralelizables, el uso de nuestras *Toolboxes* produce *speedups* prácticamente lineales, limitados básicamente por la carga de trabajo asignada a cada esclavo. Se debe procurar que la carga esté equilibrada. En estas condiciones no tiene mucha importancia escoger entre PVMTB o MPITB, obteniéndose con ambas muy buenos resultados.

  Un mérito particular de nuestras *Toolboxes* es que incluso con aplicaciones de menor granularidad se obtienen ganancias en velocidad, aunque se aparten de la linealidad. Incluso con un reducidísimo número de computadores (4 para el análisis *wavelet* B/W) se obtiene *speedup*. Esto es posible debido al bajo *overhead* que presentan ambas, lo cual las hace apropiadas para prototipar aplicaciones HPC bajo MATLAB.

6.1 Trabajo futuro

Se pueden anticipar las siguientes actividades en relación con el futuro uso y mantenimiento de las *Toolboxes* objeto de esta memoria:

- Incorporar a la página Web el código fuente de las *Toolboxes* al objeto de posibilitar su utilización por usuarios que no dispongan de alguna de las dos plataformas para las que existe actualmente versión precompilada: Linux y Solaris.

  Incorporar asimismo el código para el estudio de prestaciones y las aplicaciones desarrolladas en este trabajo, permitiendo al usuario comparar su cluster usando el nuestro como referencia. En particular, el estudio de escalabilidad del Capítulo 1 (Apartado 1.6) es de reconocido valor didáctico.

- Creación de una lista de distribución *e-mail* de usuarios, para soporte y mantenimiento de las *Toolboxes*, resolución de dudas y recepción de eventuales informes de error. También podría ser un medio efectivo para popularizar su uso, pudiendo servir asimismo como punto central para recolectar software adicional realizado por los usuarios en base a las *Toolboxes*. La página Web podría alojar un repositorio de utilidades desarrollado y mantenido por los propios usuarios, haciendo aún más atractivas las *Toolboxes* para nuevos usuarios recién introducidos en el campo de Computación Paralela.

- Utilización de las herramientas propuestas, PVMTB y MPITB, para el desarrollo de aplicaciones en proyectos de Visión Artificial en los que está involucrado el Departamento, concretamente los proyectos Europeos CORTIVIS y ECOVISION.

6.2 Méritos destacables de PVMTB y MPITB

Se relacionan resumidamente las características que más significativamente contribuyen a hacer de las *Toolboxes* presentadas en este trabajo un entorno válido para prototipado de aplicaciones HPC bajo MATLAB. Todas ellas son comunes a las dos, PVMTB y MPITB.

Complitud: Se ha implementado un interfaz completo con PVM y MPI, respetando los patrones de llamada de las rutinas PVM originales, salvo conflicto con el propio entorno MATLAB. Para tratar dicho caso se han establecido unos criterios razonados de modificación del patrón de llamada. También se ha resuelto el conflicto producido por la *opacidad* de los tipos de datos MATLAB, encapsulando la funcionalidad requerida en los comandos PVMTB pvm_[un]pack y MPITB MPI_[Un]Pack.

Incluso las características más exóticas de PVM (funciones de recepción, gestor de recursos, máscaras de trazado) y MPI (gestores de error, funciones de copia y borrado de atributos, etc) son accesibles desde la *Toolbox* correspondiente.

Enlace dinámico: Cada llamada de paso de mensajes tiene su correspondiente fichero MEX. Esto no provoca gasto excesivo de disco ya que se utiliza enlace dinámico con la correspondiente biblioteca.

No se utiliza ninguna rutina "directorio" que añada un paso intermedio de llamada. No se requieren *wrappers* M previos que añadan un paso previo de interpretación.

Reutilización de código: El diseño de ambas *Toolboxes* se basa en un estudio previo de las llamadas de paso de mensajes, siendo este estudio una característica distintiva de nuestras *Toolboxes* que no aparece en ningún trabajo anterior.

PVMTB aprovecha la repetitividad de los patrones de llamada PVM para su sistema de patrones de llamada reutilizables. MPITB recurre a bloques constructivos reutilizables, que junto con las rutinas de gestión de objetos MPI desarrolladas permiten programar los patrones para los distintos grupos de cada categoría de llamadas MPI.

Rápido prototipado: Estas *Toolboxes* permiten desarrollar prototipos de aplicaciones HPC bajo MATLAB. pudiendo utilizar para ello tanto las facilidades disponibles bajo MATLAB, (GUIs, gráficas, etc) como las de PVM y MPI (XPVM, XMPI, mpitask...) Es un hecho reconocido ([14], [39]) que la mayor parte del tiempo de desarrollo de un prototipo de cierta complejidad en un lenguaje compilado se invierte en el diseño del interfaz con el usuario y la E/S en general.

Pérdida de prestaciones mínima: Las *Toolboxes* también tienen potencial para su aplicación en investigación, tanto por el ahorro en esfuerzo de diseño de aplicaciones HPC como por la mínima pérdida de prestaciones que causan, comparándolas con el uso directo del sistema de paso de mensajes en lenguaje C.

Una despreocupada programación bajo MATLAB, ignorando el costo de las operaciones de reserva de memoria o de cambio de tipo, provoca mayor pérdida de prestaciones que las atribuibles a las *Toolboxes*. Estas operaciones son muy frecuentes en aplicaciones MATLAB típicas como procesamiento de señal e imagen, pudiéndose incluso incurrir en ellas involuntariamente al programar expresiones complejas.

Cabe manifestar a este respecto el hecho de que todos los trabajos anteriores (tanto los basados en PVM como los que utilizan MPI) incurren en una copia de memoria adicional en el mecanismo global de paso de mensajes. Las variables recibidas en los mensajes se copian a zonas de memoria reservadas a tal fin, implicando por tanto al gestor de memoria MATLAB con su elevado *overhead*.

6.3 Epílogo

En definitiva, consideramos que el presente trabajo constituye una aportación de gran utilidad en el ámbito de Computación Paralela, al posibilitar tanto la paralelización (y consiguiente mejora de prestaciones) de aplicaciones ya existentes, como el desarrollo de nuevas aplicaciones paralelas, utilizando una herramienta de gran aceptación y utilidad en el campo científico-técnico como es el entorno MATLAB.

Apéndice A

Sobre el cálculo paralelo de π

A.1 Discusión

En el Apartado 1.6 se ha usado el popular problema de calcular π mediante integración numérica de $\arctan'(x)$ para comparar la relativa comodidad de programación, capacidad expresiva y prestaciones de las distintas *Toolboxes* paralelas.

El problema aparece citado profusamente en la bibliografía, aunque es más raro encontrar una discusión acerca de los posibles méritos que cupiera atribuirle como ejemplo significativo para computación paralela.

Como propiedades atractivas del problema, cabe citar:

Sencillez del algoritmo secuencial: El método del punto central puede comprenderse prácticamente sin explicaciones adicionales. Esto permite concentrar la exposición en el método paralelo.

Sencillez del algoritmo paralelo: Los únicos datos requeridos por los procesos que cooperan son el propio índice del proceso (inum, rank), el número total de procesos C y el número total de subdivisiones N. La aplicación es *embarazosamente paralela**, requiriendo tan sólo que los procesos devuelvan como resultado parcial un único número que debe acumularse para obtener el resultado final.

Como factores negativos del problema se deben mencionar:

Uso del punto medio: El método de Simpson tiene una cota de error menor y converge más rápidamente, a pesar de requerir tres evaluaciones de la función cuando el punto medio requiere sólo una. Se puede reorganizar la expresión del método de Simpson para aprovechar la evaluación del punto derecho de una subdivisión como punto izquierdo de la siguiente, reduciendo el número de evaluaciones al doble en lugar del triple.

Uso de la acumulación: La suma en árbol es claramente superior.

Uso de optimizaciones: Tras ver el uso de la palabra clave **register** y del intercambio de los factores width y 4 (que normalmente aparecerían dentro y fuera del bucle **for**, respectivamente), el lector obtiene la impresión de que todos los detalles técnicos han sido tenidos en cuenta.

Todos estos factores intentan dirigir la atención al problema paralelo evitando perder la atención del lector con minuciosos detalles que aparentemente no guardan relación con la paralelización de algoritmos. Sin embargo, la definición de *speedup* exige que el programa secuencial con el que se compara el programa paralelo sea el óptimo. Ni siquiera se resalta, asumiéndose por entendido e inexcusable, el hecho de que el resultado del programa paralelo debe ser correcto, igual al del programa secuencial.

En el caso de integración numérica que nos ocupa, orientado a un cluster de computadores programado por paso de mensajes, aún aceptando usar el método del punto medio con acumulación, se debería resaltar que:

Homogeneidad: si las máquinas que cooperan tienen distintas arquitecturas pueden obtenerse diferencias sutiles en los resultados, debidas tanto a los distintos compiladores como a la implementación hardware concreta de las operaciones de punto flotante, incluso si siguen un estándar como el IEEE-754. Se ha incluido en la bibliografía una referencia (antigua) del manual de programación en punto flotante de Sun [87], en donde se comenta una anécdota que provocó la modificación de los tests de corrección exigidos por el FCVS (*Federal Compiler Validation Service*) para certificar la conformidad al estándar FORTRAN ANSI X3.9-1978.

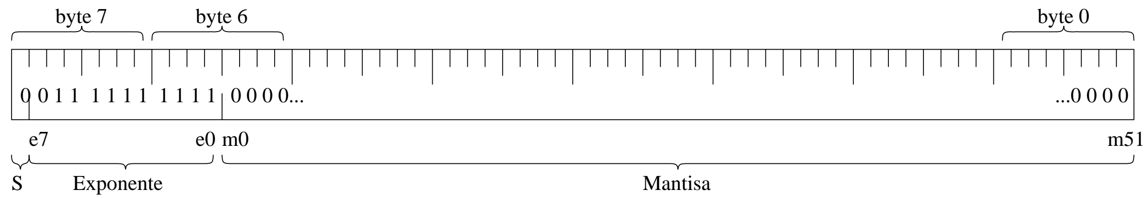
Orden de acumulación: al repartir el trabajo se altera el orden en que se acumulan las áreas individuales. El resultado *cambia*, aunque sólo ligeramente. En nuestro caso, con 408800 subdivisiones, obtenemos siempre 13 dígitos de precisión, aunque el error varía entre 4.68 y $4.99\ 10^{-13}$ según el número de computadores usados. Al ser nuestro cluster homogéneo, esta variación es reproducible secuencialmente sin más que realizar la acumulación en el mismo orden que el programa paralelo.

Los métodos de integración numérica requieren sumar multitud de pequeñas áreas. Si las operaciones se realizaran con infinita resolución, aumentar el número de subdivisiones disminuiría el error del resultado. Sin embargo, la resolución de un computador operando en punto flotante es finita y dependiente de la magnitud. Ir sumando pequeñas áreas a una única variable acumulador como se hace en el código propuesto, provoca no sólo errores de representación y redondeo, sino también errores de ecualización.

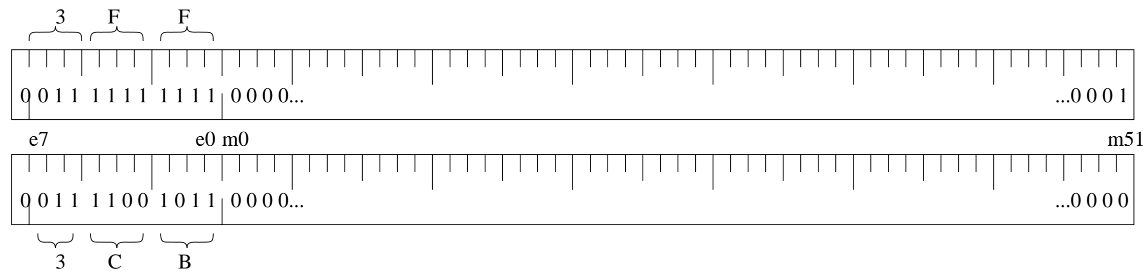
Las variables **double** bajo gcc (formato ISO C 9X / IEEE-754) ocupan 8 bytes, desglosados en 1 bit para el signo, 11 para el exponente y 52 para la mantisa (ver los ficheros **#include** math.h y ieee754.h, disponibles en cualquier instalación Linux/Unix, y la Figura A.1(a)). Esto proporciona un sesgo (*offset*) de $2^{11-1} - 1 = 1023$ para los exponentes, y una mantisa mínima de 2^{-52}, de manera que el mínimo **double** representable (**resolución**) es el número denormalizado $2^{-52}\ 2^{-1022} = 2^{-1074} \simeq 4.94\ 10^{-324}$ (Fig. A.1(c)). Se muestra también el paso de representación normalizada ($1\ 2^{-1022}$) a denormalizada ($0.5\ 2^{-1022}$). El tipo de datos escogido imposibilita representar variables con decimales menos significativos que 10^{-323}.

El Listado A.1 muestra el resultado de ejecutar un programa que asigna inicialmente la unidad a una variable **double** y divide por 2 sucesivamente hasta llegar a 0.

El mínimo número **double** que puede sumarse a la unidad para producir un resultado distinto de la unidad es la mantisa mínima, $2^{-52} \simeq 2.22\ 10^{-16}$ (Fig A.1(b)), de manera que en general sólo se puede confiar en los primeros 14–16 dígitos significativos en la expresión decimal de una

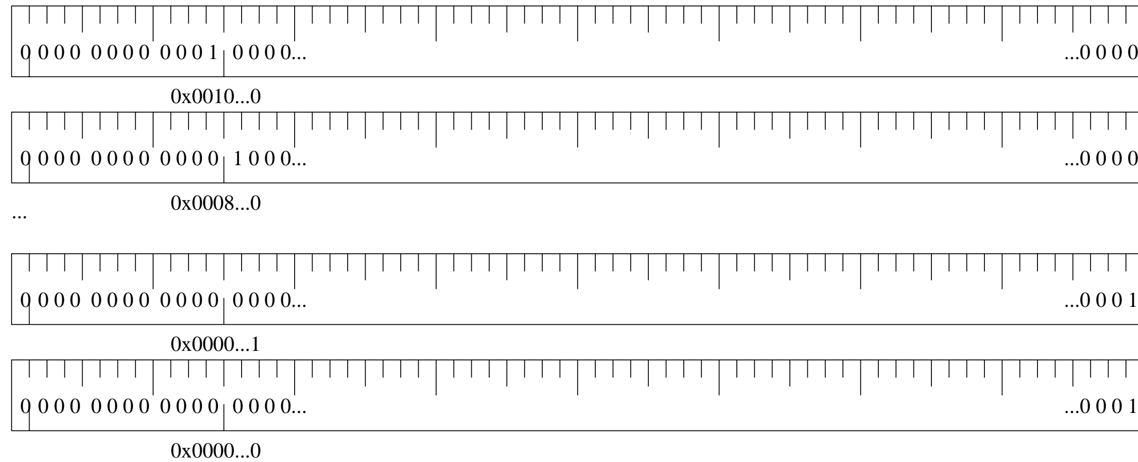


(a) Formato ISO C 9X / IEEE-754 para doble precisión. Aparece representada la unidad, con mantisa limpia y sesgo $2^{11-1}-1 = 1023$.



0 0 0 0 0 1 1 0 1 0 0 = 32+16+4 = 52

(b) El menor número acumulable a la unidad es $2^{-52} \simeq 2.22\ 10^{-16}$. La primera representación mostrada no es válida, por no estar normalizada. Se debe desplazar el bit **m51** 52 posiciones a la izquierda, para ocupar la posición del 1 implícito, y decrementar el exponente acordemente. El menor bit acumulable a 2 (o a π) sería pues $2^{-51} \simeq 4.44\ 10^{-16}$.



(c) El menor número normalizado es $1\ 2^{-1022} \simeq 2.23\ 10^{-308}$. Si se mantuviera la unidad implícita en la mantisa cuando el campo Exponente llegara a 0x000, la mínima cantidad representable sería 2^{-1023}, que sería la mejor representación positiva de 0. El estándar establece que 0x000 sigue representando 2^{-1022} pero sin suponer la unidad implícita (números denormalizados), de manera que 0x0001...0 representa la mitad del número anterior, $0.5\ 2^{-1022}$. Esta decisión subdivide linealmente el intervalo denormalizado en múltiplos del mínimo denormalizado representable, $2^{-52}\ 2^{-1022} = 2^{-1074} \simeq 4.94\ 10^{-324}$. También permite la representación del *cero limpio*, $0\ 2^{-1022}$.

Figura A.1: Diversos valores en punto flotante, doble precisión.

```
Resolución de double:
==========
2^    +0 == 1.000000000000000000000000000000000000000000000000000000000000 e+00
          == 0X3FF0000000000000
2^    -1 == 5.000000000000000000000000000000000000000000000000000000000000e-01
          == 0X3FE0000000000000
2^    -2 == 2.500000000000000000000000000000000000000000000000000000000000e-01
          == 0X3FD0000000000000
...
2^-1021 == 4.450147717014402766180465434664808128438431960924663661106655e-308
          == 0X0020000000000000
2^-1022 == 2.225073858507201383090232717332404064219215980462331830553327e-308
          == 0X0010000000000000
2^-1023 == 1.112536929253600691545116358666202032109607990231165915276664e-308
          == 0X0008000000000000
2^-1024 == 5.562684646268003457725581793331010160548039951155829576383319e-309
          == 0X0004000000000000
...
2^-1072 == 1.976262583364986176706275171472885489460239210457299057702343e-323
          == 0X0000000000000004
2^-1073 == 9.881312916824930883531375857364427447301196052286495288511714e-324
          == 0X0000000000000002
2^-1074 == 4.940656458412465441765687928682213723650598026143247644255857e-324
          == 0X0000000000000001
2^-1075 == 0.000000000000000000000000000000000000000000000000000000000000e+00
          == 0000000000000000
==========
```

Listado A.1: Resolución de **double**.

variable **double**. En concreto, el menor número que se puede sumar a 2 (o 3, o π) para producir un número distinto es $2^{-51} \simeq 4.44\ 10^{-16}$.

El Listado A.2 muestra el resultado de ejecutar un programa que comprueba estos cálculos acerca de la resolución de nuestro tipo de datos **double**.

En el ejemplo de integración numérica que nos ocupa, los productos pierden bits de precisión por redondeo y la acumulación pierde cada vez más bits por ecualización, conforme la cantidad acumulada y la cantidad por acumular son progresivamente más dispares. El esquema de suma en árbol evitaría gran parte de la ecualización, como ya se mostró en las Figuras 1.3, 1.4 y 1.5.

```
Precisión de double:
==========
1+2^ +0 == 2.000000000000000000000000000000000000000000000000000000000000 e+00
          == 0X4000000000000000
1+2^ -1 == 1.500000000000000000000000000000000000000000000000000000000000 e+00
          == 0X3FF8000000000000
1+2^ -2 == 1.250000000000000000000000000000000000000000000000000000000000 e+00
          == 0X3FF4000000000000
...
1+2^-50 == 1.000000000000000888178419700125232338905334472656250000000000e+00
          == 0X3FF0000000000004
1+2^-51 == 1.000000000000000444089209850062616169452667236328125000000000e+00
          == 0X3FF0000000000002
1+2^-52 == 1.000000000000000222044604925031308084726333618164062500000000e+00
          == 0X3FF0000000000001
1+2^-53 == 1.000000000000000000000000000000000000000000000000000000000000e+00
          == 0X3FF0000000000000
==========
  2^-52 ==                  2.220446049250313080847263336181640625000000000e-16
          == 0X3CB0000000000000
==========
```

Listado A.2: Precisión de **double**.

Al código original para el cálculo de π se le añadieron por tanto instrucciones para realizar la acumulación directamente o por árbol, siendo el método seleccionable mediante un argumento del programa. También se añadió código para implementar las fórmulas de cuadratura sencillas (punto izquierdo, derecho, medio, trapecio y Simpson), seleccionables mediante otro argumento. Por último, se añadió código para realizar un barrido sobre el número de subdivisiones del intervalo [0,1], al objeto de generar las gráficas mostradas en el Apartado 1.6. Argumentos adicionales controlan el inicio, paso y final del barrido. Se mide también el tiempo empleado en el bucle de cálculo para cada paso del barrido, como se refleja en los comentarios al pie de las Figuras 1.2(b) y 1.5(b).

A.2 Listados

A continuación se ofrecen los listados referenciados en el Apartado 1.6. El directorio que se ofrece a continuación facilita la localización de los mismos.

	Apartado	Listado	página
PVM	1.6.1 p.34		
pvmGenFiles: *script* de automatización		A.3	314
pvmMast.c: programa maestro		A.4	315
pvmWork.c: programa esclavo		A.5	317
MPI	1.6.1 p.39		
mpiGenFiles: *script* de automatización		A.6	318
mpiWork: programa SPMD		A.7	319
PVMTB	1.6.2 p.43		
pvmtb/Mast.m: programa maestro		A.8	321
pvmtb/Work.m: programa esclavo		A.9	323
MPITB	1.6.2 p.48		
mpitb/Mast.m: programa maestro		A.10	324
mpitb/Work.m: programa esclavo		A.11	326
DP-TB	1.6.2 p.52		
DPTB/Mast.m: programa maestro		A.12	327
DPTB/Work.m: programa esclavo		A.13	329
MultiMATLAB	1.6.2 p.56		
MultiMATLAB: comando MultiMATLAB		A.14	330
MM/Mast.m: programa maestro		A.15	331
MM/Work.m: programa esclavo		A.16	332
TCPIP	1.6.2 p.62		
Spawner: arranque remoto de procesos		A.17	333
Speedup: *script* de automatización		A.18	334
tcpip/Mast.m: programa maestro		A.19	335
tcpip/Work.m: programa esclavo		A.20	336
Paralize	1.6.2 p.68		
Speedup: *script* de automatización		A.21	337
para/Mast.m: programa maestro		A.22	338
para/Work.m: programa esclavo		A.23	339

06/26/01 20:16:08 pvmGenFiles 1

```
#!/bin/bash

##########################################################################
if [ $# -eq 0 ]; then
echo
echo Usage: ./pvmGenFiles SUB [ C [ N [ MOD [ ENC ]]]]
echo SUB••••••••: subdirectorio en donde guardar fichero mediciones
echo •C•••••\<=8 : número computadores esclavos estudio escalabilidad pi
echo •••••[C==0]: se usan dos computadores para test ping-pong
echo •N [408800]: número subdivisiones para escalabilidad integración pi
echo •••••[1500]: tamaño máximo paquete ping-pong
echo MOD•••••[s]: modo comunicación \(s\)end-recv \(r\)educe \(p\)send-precv
echo ENC•••••[d]: flag encaminamiento \(d\)irecto / \(n\)o
echo
exit
fi
##########################################################################
  PRG=s
SUB=${1:-.}     # el valor por defecto no se usa, se ejecutaría el Usage antes
  C=${2:-0}; if [ ${C} -eq 0 ]; then C=1; PRG=p; fi
  N=${3:-0}; if [ ${N} -eq 0 ]; then
                if [ ${PRG} = s ]; then N=408800; fi
                if [ ${PRG} = p ]; then N=1500; fi
  fi
MOD=${4:-s}
ENC=${5:-d}
#                       MasterName, SlaveName,  FileName
if [ ${PRG} = s ]; then MN=pvmMast; SN=pvmWork; FN=pvmSpeedup; fi
if [ ${PRG} = p ]; then MN=pvmPing; SN=pvmPong; FN=pvmPingPong; fi
FN=‘pwd‘/${SUB}/${FN}.${C}-${N}-${MOD}-${ENC}.dat

##########################################################################
# Ejecuciones secuenciales para Speedup, en servidor (oxigeno) y cliente (ox1)
##########################################################################
# Usage: ./pvmMast C N MOD ENC [FN [FC]]
#     C: # computadores esclavos / 0 para algoritmo secuencial
#     N: # subdivisiones para integración numérica pi
#   MOD:  (s)end-recv / (r)educe / (p)send-precv
#   ENC:   Route (d)irect / (n)o
#    FN:  (opcional): FileName donde salvar datos
#    FC:   0 añadir (default) / !=0 crear fichero y anotar FC
##########################################################################

if [ ${PRG} = s ]; then
           ./$MN 0 $N - - $FN $C       # Crear fichero, anotar # hosts escalab
rsh ox1 ‘pwd‘/$MN 0 $N - - $FN          # Datos de oxígeno, añadir de ox1
fi

##########################################################################
# Crear hostfile conteniendo PATH. Matar posible PVM anterior
##########################################################################

pvm  <<-@
        halt
        @
cat >   hostfile <<-@
        * ep=‘pwd‘:\$PATH
        @

##########################################################################
# Siguientes ejecuciones Speedup usan algoritmo paralelo, añaden a fichero.
# ir añadiendo hosts a hostfile, arrancar PVM (y groupserver para Reduce?)
# ejecutar pvmMast desde shell (desde consola no se queda bloqueado
# y seguiríamos haciendo spawn 2, 3... procesos)
# pvmMast hace pvm_halt, no necesitamos parar PVM nosotros
##########################################################################
# El caso PingPong cuadra con C=1. También usa grupos, y arranca el esclavo.
# SN (Slave Name) sólo hacía falta para ese caso.
##########################################################################
# Usage: ./pvmPing N MOD ENC [FN]
#     N:  tamaño máximo mensaje
#   MOD: (s)end-recv / (r)educe / (p)send-precv
#   ENC:  Route (d)irect / (n)o
#    FN: (opcional) FileName donde salvar datos
#       pvmPong debe ejecutarse también, ver pvmGenFiles
##########################################################################
                                            ############
if [ $PRG = p ]; then                       # PingPong #
    pvm -nox0 hostfile <<-@                 ############
        add ox1
        spawn -ox0 pvmgs
        spawn -ox1 pvmPong $N $MOD $ENC
        @
    ./$MN    $N $MOD $ENC $FN               # los $MN hacen ya pvm_halt
                                            ###########
else                                        # Speedup #
 i=0                                        ###########
 while [ $[i+=1] -le $C ]; do               # incrementar i y comparar después
  cat >>hostfile <<-@                       # añadir ox<i> desde ox1 hasta ox<C>
        ox$i
        @
  if [ $MOD = r ]; then                     # arrancar pvmgs si MOD==’r’
    pvm -nox0 hostfile <<-@
        spawn -ox0 pvmgs
        @
  else
    pvm -nox0 hostfile <<-@                 # si no reduce, nada de grupos
        @
  fi
    ./$MN $i $N $MOD $ENC $FN               # los $PN hacen ya pvm_halt
 done
fi

rm hostfile
```

Listado A.3: *script* pvmGenFiles para la automatización de mediciones PVM. Permite realizar el estudio de escalabilidad controlando el número máximo de computadores y subdivisiones. También permite realizar el test *ping-pong* controlando el tamaño máximo del mensaje. Contempla un total de 6 combinaciones variando el modo de comunicación (punto a punto send/recv, psend/precv, o reducción colectiva) y encaminamiento (directo o no).

06/22/01 18:36:00 **pvmMast.c** 1

```c
/*
 * pvmMast C N MOD ENC [FN [FC]]
 *   C: # computadores esclavos / 0 para algoritmo secuencial
 *   N: # subdivisiones para integración numérica pi
 * MOD:  (s)end-recv / (r)educe / (p)send-precv
 * ENC:   Route (d)irect / (n)o
 *  FN:  (opcional): FileName donde salvar datos -> 3xdouble (pi,err,time)
 *  FC:   0 añadir (default) / !=0 crear fichero y anotar FC (int) delante
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>   /* para gettimeofday() */
#include <sys/stat.h>   /* para creat, open */
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>     /* para write, close */
#include <math.h>       /* fabs */

#include <pvm3.h>
#define TAG 7
#define MAXHOSTS 8

/***************/
/* MEDIR TIEMPO */
double Wtime(){
  struct timeval now; gettimeofday(&now, NULL);
  return((double)now.tv_sec + (double)now.tv_usec/1E6);
}

/************/
/* PROGRAMA */
/************/
main(int argc, char **argv)             /* pvmMast C N MOD ENC [FN [FC]] ) */
{
  int  C,FC,fd;                         /* args */
  char MOD,ENC;                         /* N es register, ver abajo */
  char*FN;                              /* fd file descriptor para FileName */

  int    nh,numt,inum,tids[MAXHOSTS];   /* tids no usado sii MOD=='r' */
  double Sum,Psum,T,PT,err,Perr,PI25DIG = 3.141592653589793238462643 ;
  register  double width, sum;
  register  int N, i;

  /**************/
  /* Argumentos */
  /**************/
 if (argc<5) {
    printf("\n"
        "Usage: ./pvmMast C N MOD ENC [FN [FC]]\n"
        "    C: # computadores esclavos / 0 para algoritmo secuencial\n"
        "    N: # subdivisiones para integración numérica pi\n"
        "  MOD:  (s)end-recv / (r)educe / (p)send-precv\n"
        "  ENC:   Route (d)irect / (n)o\n"
        "   FN:  (opcional): FileName donde salvar datos"
                        " -> 3xdouble (pi,err,time)\n"
        "   FC:   0 añadir (default) / !=0 crear fichero y anotar FC (int)\n"
        "\n");
    exit(1);
  }
            C = atoi(argv[1]);                  /* Computadores */
            N = atoi(argv[2]);                  /* Subdivisiones */
          MOD=    *(argv[3]);                   /* Modo comunicación */
          ENC=    *(argv[4]);                   /* Ruta Directa */
if (argc>5) FN=      argv[5] ;                  /* Salvar a fichero */
if (argc>6) FC= atoi(argv[6]); else FC=0;       /* Creat flag / tam.barrido */

    argv++[4]=NULL;                             /* para spawn pvmWork */

  if (C>MAXHOSTS) {
    printf("%d > %d: subir MAXHOSTS en código fuente\n",C,MAXHOSTS);
    exit(1);
  }

if (C){
  /**************/
  /* Paralelo   */
  /**************/
  /* Test daemon PVM */
  pvm_config(&nh,NULL,NULL);
  if (nh<C+1) {
    printf("%d<%d computadores solicitados en PVM\n",nh,C+1);
    exit(1);
  }
  if (ENC=='d') pvm_setopt(PvmRoute, PvmRouteDirect);

  if (MOD=='r')                           /* Reduce usa grupos. Crear grupo */
    if (inum=pvm_joingroup("pi")){        /* antes de spawn -> somos master */
      printf("no puedo crear grupo\n");
      exit(2);
    }

  /* SPAWN */
    numt=pvm_spawn("pvmWork",argv,PvmTaskHost+PvmHostCompl,".",C,NULL);
    if (numt!=C) {
      printf("no puedo arrancar pvmWorks\n");
      exit(3);
    }

  /* Synch */
  switch (MOD){
  case 'r':                               /* ver Caveat man pvm_reduce */
        pvm_freezegroup("pi",C+1);                              break;

  case 's':                /* Anotar tids en el orden en que van respondiendo */
    for (numt=1; numt<=C; numt++){        /* establecer de paso ruta directa */
        pvm_recv(-1,TAG); pvm_upkint(&tids[numt],1,1);  /* antes de Wtime */
        pvm_initsend(PvmDataInPlace); pvm_pkint(&tids[numt],1,1);
        pvm_send(tids[numt],TAG);}                               break;

  case 'p':
    for (numt=1; numt<=C; numt++){        /* el psend es para ruta directa */
        pvm_precv(-1,TAG,&tids[numt],1,PVM_INT,NULL,NULL,NULL);
        pvm_psend(tids[numt],TAG,&tids[numt],1,PVM_INT);}        break;
  }

    PT = Wtime();       /* Medir tras Synch inicial (equivale a MPI_Init) */

  switch (MOD){
  case 's':                                              /* Asignar inum */
    for (numt=1; numt<=C; numt++){
        pvm_initsend(PvmDataInPlace); pvm_pkint(&numt,1,1);
        pvm_send(tids[numt],TAG);}                               break;
  case 'p':
    for (numt=1; numt<=C; numt++){
        pvm_psend(tids[numt],TAG,&numt,1,PVM_INT);}              break;
```

Listado A.4: pvmMast.c: Programa maestro PVM para el estudio de escalabilidad basado en el cálculo de π. Fragmentos correspondientes al modo de reducción se mostraron en el Listado 1.2.

06/22/01 18:36:00 — pvmMast.c — 2

```
  }

  /* Resultados */
  Psum=0;
  switch (MOD){
  case 'r':
        pvm_reduce(PvmSum, &Psum,1,PVM_DOUBLE, TAG,"pi",0);     break;
  case 's':
    for (numt=1; numt<=C; numt++){
        pvm_recv(-1,TAG); pvm_upkdouble(&Sum,1,1); Psum+=Sum;}  break;
  case 'p':
    for (numt=1; numt<=C; numt++){
        pvm_precv(-1,TAG,&Sum,1,PVM_DOUBLE,NULL,NULL,NULL);
                                                Psum+=Sum;}     break;
  }

    Psum/=N;            /* mejor hacerlo en master: no perder prec. */

    PT = Wtime()-PT ;           /* Medir en cuanto pi esté calculado */

  if (MOD=='r'){pvm_barrier("pi",C+1);  /* ver Caveat en man pvm_reduce */
                pvm_lvgroup("pi");}
                pvm_exit();
/*              pvm_halt();*/           /* nos mataría a nosotros también */


} else { /* if (C) */
  /**************/
  /* Secuencial */
  /**************/
    T   = Wtime();              /* Speedup lo usa para indicar C total */

    width = 1.0/N; sum = 0;
    for (i=0; i<N; ++i) {
      register double x = (i + 0.5) * width;
      sum += 4.0 / (1.0 + x * x);
    }
    sum *= width;

    T = Wtime()-T;
    Sum = sum;

    printf("===========\n");
    printf("Secuencial: (%d subdivisiones)\n", N);
    printf("===========\n");
    printf("pi es  %.16f\n",sum); err = sum-PI25DIG;
    printf("Error  %.2e\n", err);
    printf("Tiempo %.6f\n", T  );
if (fabs(err)>5E-10)
    printf("menudo valor de pi!\n");

} /* else (C) */

if (C){
    printf("===========\n");
    printf("Paralelo  : (%d CPUs, %d subd. %c %c)\n", C,N,MOD,ENC);
    printf("===========\n");
    printf("pi es  %.16f\n",Psum); Perr = Psum-PI25DIG;
    printf("Error  %.2e\n", Perr);
    printf("Tiempo %.6f\n", PT  );
if (fabs(Perr)>5E-10)
    printf("menudo valor de pi!\n");
}

  /***********/
  /* Fichero */
  /***********/

  if (argc>5){                          /* Se menciona fichero: crear? */

    if (FC){                            /* Crear y anotar número FC */
      if ((fd=creat(FN,S_IRWXU))<0){
        printf("no puedo crear fichero %s\n",FN);
        exit(15);
      }
      write(fd,&FC,sizeof(FC));         /* En realidad expresa C==8 */


    } else /* !FC */                    /* No crear, añadir */
      if ((fd=open(FN,O_WRONLY|O_APPEND))<0){
        printf("no puedo añadir a fichero %s\n",FN);
        exit(16);
      }

    if (C){
      write(fd,&Psum,sizeof(Psum));     /* Paralelo   */
      write(fd,&Perr,sizeof(Perr));
      write(fd,&PT  ,sizeof(PT  ));
    } else {
      write(fd,&Sum,sizeof(Sum));       /* Secuencial */
      write(fd,&err,sizeof(err));
      write(fd,&T  ,sizeof(T  ));
    }

    if (close(fd)){
      printf("no puedo cerrar fichero %s\n",FN);
      exit(17);
    }
  }  /* argc>6 */

  if (C) pvm_halt();
  exit(0);

}
```

Listado A.4: Continuación.

07/01/01 11:16:51 — pvmWork.c — 1

```c
#include <stdlib.h>
#include <pvm3.h>
#define TAG 7

int main(int argc, char **argv)         /* pvmWork C N MOD ENC */
{
  int  C;                               /* args, N es register */
  char MOD,ENC;
  int  inum,me,parent;          /* me,parent, no se usan sii MOD=='r' */
                                        /* inum se simula si MOD!='r' */
          double         sum;           /* lsum es register */
  register double width, lsum;
  register int N, i;                    /* bucles */

  /**************/
  /* Argumentos */
  /**************/
  if (argc<4) exit(1);
   C = atoi(argv[1]);                   /* Computadores */
   N = atoi(argv[2]);                   /* Subdivisiones */
  MOD=    *(argv[3]);                   /* (s)end (r)educe (p)send */
  ENC=    *(argv[4]);                   /* ruta directa */

if (ENC=='d') pvm_setopt(PvmRoute, PvmRouteDirect);

  /**************/
  /* Synch Mast */
  /**************/
  me    =pvm_mytid();
  parent=pvm_parent();
  switch (MOD){
  case 'r':
      pvm_psend(parent,TAG,&me,1,PVM_INT);      /* esto es para ruta directa */
      pvm_precv(parent,TAG,&me,1,PVM_INT,NULL,NULL,NULL); /* antes de Wtime */
      inum=pvm_joingroup("pi");         /* ver Caveat man pvm_reduce */
      pvm_freezegroup("pi",C+1);        /* hace de barrera también */
      break;
  case 's':
      pvm_initsend(PvmDataInPlace);/* de las dos recepciones, la 1° es para */
      pvm_pkint(&me,1,1); pvm_send(parent,TAG); /* establecer ruta directa */
      pvm_recv(parent,TAG); pvm_upkint(&me,1,1);        /* antes de Wtime */
      pvm_recv(parent,TAG); pvm_upkint(&inum,1,1);      /* después de Wtime */
      break;
  case 'p':                                     /* Enviar tid, recibir inum */
      pvm_psend(parent,TAG,&me,1,PVM_INT);      /* antes de cronómetro Wtime */
      pvm_precv(parent,TAG,&me,1,PVM_INT,NULL,NULL,NULL);       /* antes */
      pvm_precv(parent,TAG,&inum,1,PVM_INT,NULL,NULL,NULL);     /* después */
      break;
  }

  /**************/
  /* Cálculo    */
  /**************/
  width = 1.0/N; lsum = 0;
  for (i=inum-1; i<N; i+=C) {
    register double x = (i + 0.5) * width;
    lsum += 4.0 / (1.0 + x * x);
  }
  sum = lsum;                           /* lsum es register */

  switch (MOD){                         /* Enviar parcial */
  case 'r': pvm_reduce(PvmSum, &sum,1,PVM_DOUBLE, TAG,"pi",0);
            pvm_barrier("pi",C+1);      /* ver Caveat en man pvm_reduce */
            pvm_lvgroup("pi");                                   break;
  case 's': pvm_initsend(PvmDataInPlace); pvm_pkdouble(&sum,1,1);
            pvm_send(parent,TAG);                                break;
  case 'p': pvm_psend(parent,TAG,&sum,1,PVM_DOUBLE);             break;
  }

  pvm_exit();
  exit(0);
}
```

Listado A.5: pvmWork.c: Programa esclavo PVM para el estudio de escalabilidad basado en el cálculo de π. Fragmentos correspondientes al modo de reducción se mostraron en el Listado 1.3.

07/02/01
10:57:14

mpiGenFiles

1

```
#!/bin/bash

###########################################################################
if [ $# -eq 0 ]; then
echo
echo Usage: ./mpiGenFiles SUB [ C [ N [ MOD [ ENC ]]]]
echo SUB••••••••: subdirectorio en donde guardar fichero mediciones
echo •C•••••\<=8 : número computadores estudio escalabilidad pi
echo •••••[C==0]: se usan dos computadores para test ping-pong
echo •N [408800]: número subdivisiones para escalabilidad integración pi
echo •••••[1500]: tamaño máximo paquete ping-pong
echo MOD•••••[s]: modo comunicación \(s\)end-recv \(r\)educe
echo ENC•••••[d]: flag encaminamiento \(d\)irecto / \(n\)o
echo
exit
fi
###########################################################################
  PRG=s
SUB=${1:-.}     # el valor por defecto no se usa, se ejecutaría el Usage antes
  C=${2:-0}; if [ ${C} -eq 0 ]; then C=1; PRG=p; fi
  N=${3:-0}; if [ ${N} -eq 0 ]; then
                if [ ${PRG} = s ]; then N=408800; fi
                if [ ${PRG} = p ]; then N=  1500; fi
  fi
MOD=${4:-s}
ENC=${5:-d}
#                       ProgramName FileName
if [ ${PRG} = s ]; then PN=mpiWork; FN=mpiSpeedup; fi
if [ ${PRG} = p ]; then PN=mpiPong; FN=mpiPingPong; fi
FN=`pwd`/${SUB}/${FN}.${C}-${N}-${MOD}-${ENC}.dat

HMG="-O -nger -toff -nsigs"
C2C="-lamd"; if [ ${ENC} = d ]; then C2C="-c2c" ; fi
#
# mpirun activa por defecto los siguientes RunTimeFlags
# TROLLIUSRTF == 0x139292 == RTF_KENYA | RTF_WAIT | RTF_MPIC2C | RTF_TRSWITCH |
#                 RTF_MPIRUN | RTF_IO | RTF_PFDIO | RTF_TTYOUT | RTF_MPISIGS
# -nw desactiva RTF_WAIT
# -lamd desactiva RTF_MPIC2C y activa RTF_MPIGER --> muy malo prestaciones GER
# -toff desactiva RTF_TRSWITCH pero activa RTF_TRACE (-ton activa los dos)
# -f creo que desactiva RTF_[PFD]IO y/o RTF_TTYOUT ???
# -nsigs desactiva RTF_MPISIGS
# RTF_MPIGER se desactiva en mpiGenFiles para comparar prestaciones con MPITB
#
###########################################################################
# Crear hostfile. Matar posible LAM anterior
###########################################################################
rm -f   bhost.def
i=-1
while [ $[i+=1] -le $C ]; do
 cat >> bhost.def <<-@
        ox$i
        @
done

lamboot bhost.def

###########################################################################
# Ejecuciones secuenciales para Speedup, en servidor (oxigeno) y cliente (ox1)
###########################################################################
# Usage: mpirun -O [-c2c] -c C mpiWork -- N MOD [FN [FC]]
#  -c2c:   client-to-client
#     C: # computadores total / 1 para algoritmo secuencial
#     N: # subdivisiones para integración numérica pi
#   MOD:  (s)end-recv / (r)educe
#    FN:  (opcional): FileName donde salvar datos
#    FC:   0 añadir (default) / !=0 crear fichero y anotar FC
###########################################################################

if [ ${PRG} = s ]; then
mpirun -w $HMG $C2C n0 $PN -- $N - $FN $C       # Crear fichero, anotar # hosts
mpirun -w $HMG $C2C n1 $PN -- $N - $FN          # Datos de oxígeno y añadir ox1
fi

###########################################################################
# Siguientes ejecuciones Speedup usan algoritmo paralelo, añaden a fichero.
# si se pide estudio C=8 esclavos hay que llegar hasta -c 9 copias
# se empieza por -c 2 copias (rank 0 y 1 esclavo. oxígeno no trabaja)
###########################################################################
# El caso PingPong cuadra con C=1
###########################################################################
# Usage: mpirun -O [-c2c] -c 2 mpiPingPong -- N MOD [FN]
#  -c2c:   client-to-client
#     N:   tamaño máximo mensaje
#   MOD:  (s)end-recv / (r)educe
#    FN: (opcional): FileName donde salvar datos
###########################################################################

i=1
while [ $i -le ${C} ]; do
mpirun -w $HMG $C2C -c $[i+=1] $PN -- $N $MOD $FN
done

wipe bhost.def
rm bhost.def
```

Listado A.6: *script* mpiGenFiles para la automatización de mediciones MPI. Permite realizar el estudio de escalabilidad controlando el número máximo de computadores y subdivisiones. También permite realizar el test *ping-pong* controlando el tamaño máximo del mensaje. Contempla un total de 4 combinaciones variando el modo de comunicación (punto a punto o reducción colectiva) y encaminamiento (cliente a cliente o *daemon* LAM).

06/22/01
18:36:00 mpiWork.c 1

```c
/*
 * mpirun -O [-c2c] -c C mpiWork -- N MOD [FN [FC]]
 *-c2c:   client-to-client
 *   N: # subdivisiones para integración numérica pi
 * MOD:  (s)end-recv / (r)educe\n"
 *  FN:  (opcional): FileName donde salvar datos -> 3xdouble (pi,err,time)
 *  FC    0 añadir (default) / !=0 crear fichero y anotar FC (int) delante
 *
 *   C: # computadores total / 1 para algoritmo secuencial
 *             se deduce de MPI_Comm_size()
 * ENC: (d)irect / (n)o -> se controla mediante -c2c
 *             se deduce de TROLLISRTF (runtime flags)
 */

#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>   /* para creat, open */
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>     /* para write, close */
#include <math.h>       /* fabs */

#include <mpi.h>
#define TAG 7
#define RTFHMG 0x2000
#define RTFC2C 0x0080


/************/
/* PROGRAMA */  /* mpirun -O [-c2c] -c C mpiWork -- N MOD [FN [FC]] */
/************/
main(int argc, char **argv)
{
  int  C,FC,fd;         /* C se controla desde mpirun -c C, aquí se lee size */
  char MOD,ENC;         /* ENCaminamiento desde mpirun -c2c, aquí se detecta */
  int TRTF;             /* leyendo TROLLIUS RunTimeFlags */
  char*FN;              /* fd file descriptor para FileName */

  int    rank,size; MPI_Status st;
  double Sum,Psum,P2,T,PT,err,Perr,PI25DIG = 3.141592653589793238462643 ;
  register double width, sum;
  register int N, i;

  /* Test daemon */ /* para obtener rank y que sólo master imprima errores */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
                              C=size-1; /* Master no trabaja */
  /**************/
  /* Argumentos */
  /**************/
 if ( (!rank) && (argc<3)) {
    printf("\n"
        "Usage: mpirun -O [-c2c] -c C mpiWork -- N MOD [FN [FC]]\n"
        " -c2c:   client-to-client\n"
        "    C: # computadores total / 1 para algoritmo secuencial\n"
        "    N: # subdivisiones para integración numérica pi\n"
        "  MOD:  (s)end-recv / (r)educe\n"
        "   FN:  (opcional): FileName donde salvar datos"
        "              " -> 3xdouble (pi,err,time)\n"
        "   FC:   0 añadir (default) / !=0 crear fichero y anotar FC (int)\n"
        "\n");
    exit(1);
  }
           N = atoi(argv[1]);                  /* Subdivisiones */
         MOD=    *(argv[2]);                   /* Modo comunicación */
if (argc>3) FN=      argv[3] ;                 /* Salvar a fichero */
if (argc>4) FC= atoi(argv[4]); else FC=0;      /* Creat flag / tam.barrido */

  TRTF=atoi(getenv("TROLLIUSRTF"));
  if (! ( TRTF & RTFHMG )){
    printf("usar mpirun -O para cluster homogéneo!!!\n");
  }
  if (! ( TRTF & RTFC2C )) ENC='n';            /* Modo Daemon */
  else                     ENC='d';            /* Ruta Directa */


if (C){
  /**************/
  /* Paralelo   */
  /**************/
  if (rank){
  /**************/
  /* Cálculo    */
  /**************/
    width = 1.0/N; sum = 0;
    for (i=rank-1; i<N; i+=C) {
      register double x = (i + 0.5) * width;
      sum += 4.0 / (1.0 + x * x);
    }
    Psum = sum;                         /* sum es register */
    switch (MOD){
    case 'r': MPI_Reduce(&Psum,NULL ,1,MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
              break;
    case 's': MPI_Send  (&Psum,      1,MPI_DOUBLE,       0,TAG,MPI_COMM_WORLD);
              break;
    }

  } else { /* if (rank) */
  /**************/
  /* Master     */
  /**************/
    PT = MPI_Wtime();                   /* MPI_Init ya sincronizó */
    P2= Psum=0;
    switch (MOD){
    case 'r':
              MPI_Reduce(&P2,&Psum,1,MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
              break;
    case 's': for (i=1;i<size;i++){
              MPI_Recv(&P2,1,MPI_DOUBLE,MPI_ANY_SOURCE,TAG,MPI_COMM_WORLD,&st);
                Psum+=P2;
              }
              break;
    }
    Psum/=N;              /* mejor hacerlo en master: no perder prec. */

    PT = MPI_Wtime()-PT;        /* Medir en cuanto pi esté calculado */

  } /* else rank */

  MPI_Finalize();


}else{ /* if (C) */
    /**************/
    /* Secuencial */
```

Listado A.7: mpiWork.c: Programa SPMD MPI para el estudio de escalabilidad basado en el cálculo de π. Se mostraron fragmentos en los Listados 1.4 y 1.5.

06/22/01 18:36:00 — mpiWork.c — 2

```
    /**************/
      T   = MPI_Wtime();        /* Speedup lo usa para indicar C total */

      width = 1.0/N; sum = 0;
      for (i=0; i<N; ++i) {
        register double x = (i + 0.5) * width;
        sum += 4.0 / (1.0 + x * x);
      }
      sum *= width;

      T = MPI_Wtime()-T;
      Sum = sum;

      printf("===========\n");
      printf("Secuencial: (%d subdivisiones)\n", N);
      printf("===========\n");
      printf("pi es  %.16f\n",sum); err = sum-PI25DIG;
      printf("Error  %.2e\n", err);
      printf("Tiempo %.6f\n", T  );
if (fabs(err)>5E-10)
    printf("menudo valor de pi!\n");

} /* else (C) */

if (C && !rank){
    printf("==========\n");
    printf("Paralelo  : (%d CPUs, %d subd. %c %c)\n", C,N,MOD,ENC);
    printf("==========\n");
    printf("pi es  %.16f\n",Psum); Perr = Psum-PI25DIG;
    printf("Error  %.2e\n", Perr);
    printf("Tiempo %.6f\n", PT  );
if (fabs(Perr)>5E-10)
    printf("menudo valor de pi!\n");
}


    /***********/
    /* Fichero */
    /***********/

  if ( (!rank) && (argc>3)){            /* Se menciona fichero: crear? */

    if (FC){                            /* Crear y anotar número FC */
      if ((fd=creat(FN,S_IRWXU))<0){
        printf("no puedo crear fichero %s\n",FN);
        exit(15);
      }
        write(fd,&FC,sizeof(FC));       /* En realidad expresa C==8 */

    } else /* !FC */                    /* No crear, añadir */
      if ((fd=open(FN,O_WRONLY|O_APPEND))<0){
        printf("no puedo añadir a fichero %s\n",FN);
        exit(16);
      }

    if (C){
        write(fd,&Psum,sizeof(Psum));   /* Paralelo   */
        write(fd,&Perr,sizeof(Perr));
        write(fd,&PT  ,sizeof(PT  ));
    } else {
        write(fd,&Sum,sizeof(Sum));     /* Secuencial */
        write(fd,&err,sizeof(err));
        write(fd,&T  ,sizeof(T  ));
      }

      if (close(fd)){
          printf("no puedo cerrar fichero %s\n",FN);
          exit(17);
      }
    } /* argc>3 */

      exit(0);

} /* main() */
```

Listado A.7: Continuación.

06/22/01 20:22:05 ../PVMTB/Mast.m **1**

```
function Data=Mast(C,N,MOD,ENC)
% MAST: Master para cálculo pi PVMTB
%
%       Data = Mast ( C, N, MOD, ENC )
%
%  C    número de computadores usados (NO incluyendo éste)
%       Si C= 0, se hace el algoritmo en este ordenador
%       Si C=-1, se hace en el primer esclavo
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación:  (s)end (r)educe (p)send
%  ENC  modalidad de encaminamiento: (d)irecto (n)o
%
%  Data estructura conteniendo:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%
% pi se calcula integrando numéricamente arctg'(x) = 1/(1+x^2) entre 0 y 1
% el resultado es pi/4, multiplicado por 4 da pi
% MATLAB proporciona una función pi, que se usa para calcular el error
%
% Se configura y arranca PVM
% Si estaba previamente arrancado, se comprueba que haya C esclavos
% Pensado para llamar iterativamente desde "Todos"
%

HN={['* ep=$PATH wd=' pwd],'ox1','ox2','ox3','ox4','ox5','ox6','ox7','ox8'};
Data=[]; global TAG PLACE ROU DONT DIR, DEBUG=0;

                         MastFLG=0;      % Para ejecutar Mast(0) en ox1
if nargin>1 & C<0, C=1; MastFLG=1; end  % en cuyo caso se devuelve por disco
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<4, help(mfilename), return, end
flag=0;                                 CC=int2str(C);
flag=flag | fix(C)~=C | C<0;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;            MOD=lower(MOD);
flag=flag | isempty(findstr(MOD,'srp'));ENC=lower(ENC);
flag=flag | isempty(findstr(ENC,'dn'));
if flag, help(mfilename), return, end

if C>=length(HN), error('añadir ordenadores en código fuente Mast.m'), end

if C
%%%%%%%%%%%%%%%%%%%
% Test daemon PVM % Número de hosts>C
%%%%%%%%%%%%%%%%%%%
  pvm_setopt(3,0);                      % AutoErr Ignore
  pvme_start_pvmd(HN{1:C+1});           % * ep, wd + C ordenadores
  pvm_setopt(3,1);                      % AutoErr Warn
  if pvm_config<=C,error(['necesito ' int2str(C+1) ' computadores en PVM']),end
  if ENC=='d', pvm_setopt(ROU,DIR); end % PvmRouteDirect

%%%%%%%%%%%
% COMANDO %
%%%%%%%%%%%
  if MastFLG                            % Caso Mast(0) en ox1
    FN=[ pwd '/tmp.mat'];
    cmd=['Data=Mast(0,' NN ',''' MOD ''',''' ENC '''); save ' FN ' Data'];
  else
    if MOD=='r'                         % garantizar que Mast crea grupo
      pvm_spawn('pvmgs',{},1,'.',1);    % groupserver siempre en el servidor
      inum=pvm_joingroup('pi');         % y antes que Spawn esclavos
      if inum~=0, error('no puedo crear grupo'), end
    end
    cmd=['Work(' CC ',' NN ',''' MOD ''')'];
  end

  putenv(['cmd=' cmd]); pvm_export('cmd');

%%%%%%%%%
% SPAWN %
%%%%%%%%%
  if DEBUG
    DISP=getenv('DISPLAY');
    [numt tids]=pvm_spawn('xterm',...
                        {'-display' DISP '-e' 'matlab'},33,'.',C);
  else
    [numt tids]=pvm_spawn('matlab',{'-nosplash'},       33,'.',C);
  end
  if numt~=C, error('no puedo arrancar MATLABs'), end

%%%%%%%%%%%%%
% Resultado Secuencial caso Mast(0) en ox1
%%%%%%%%%%%%%
  if MastFLG
    while ~exist(FN,'file'), pause(1), end
    load(FN), delete(FN)
  else

%%%%%%%%%%%%
% PARALELO %
%%%%%%%%%%%%
    tid=0;
    switch(MOD)
    case 's' %%%%%%%%%%%%%%%            % Anotar tids conforme respondan
        for numt=1:C, pvm_recv(-1,TAG);pvm_upkdouble('tid');tids(numt)=tid; end
    case 'r' %%%%%%%%%%%%%%%            % se ejercita ruta directa de paso
        pvm_barrier('pi',C+1);          % ver Caveat man pvm_reduce
    case 'p' %%%%%%%%%%%%%%%
        for numt=1:C, pvm_precv(-1,TAG,'tid',1); tids(numt)=tid; end
    end

%%%%%%%%%
    tic %               % Medir tras Synch inicial (equivale a MPI_Init)
%%%%%%%%%
    switch(MOD)                         % Asignar inum (equivale a emitir args)
    case 's' %%%%%%%%%%%%%%%
    for numt=1:C
        pvm_initsend(PLACE); pvm_pkdouble('numt'); pvm_send(tids(numt),TAG); end
    case 'p' %%%%%%%%%%%%%%%
        for numt=1:C, pvm_psend(tids(numt),TAG,numt); end
    end

%%%%%%%%%%%%%
% Resultado Paralelo
%%%%%%%%%%%%%
    Psum=0; Sum=0;      % no usar [Psum Sum]=deal(0) -> aliasing en upkdouble
    switch(MOD)
    case 's' %%%%%%%%%%%%%%%
    for numt=1:C, pvm_recv(-1,TAG); pvm_upkdouble('Sum'); Psum=Psum+Sum; end
    case 'r' %%%%%%%%%%%%%%%
        pvm_reduce('Sum',Psum,TAG,'pi',0);
    case 'p' %%%%%%%%%%%%%%%
        for numt=1:C, pvm_precv(-1,TAG,'Sum',1);          Psum=Psum+Sum; end
```

Listado A.8: pvmtb/Mast.m: Programa maestro PVMTB para el estudio de escalabilidad basado en el cálculo de π. Fragmentos correspondientes al modo de reducción se mostraron en los Listados 1.6 y 1.7.

06/22/01 20:22:05 ../PVMTB/Mast.m 2

```
    end

    Psum     =Psum/N;           % mejor hacerlo en master: no perder precisión
    Data.pi  =Psum;
    Data.err =Psum-pi;
%%%%%%%%%%%%%%%%%%%
    Data.time=toc;%
%%%%%%%%%%%%%%%%%%%

    if MOD=='r'
      pvm_barrier('pi',C+1);            % Barrera Synch, psé, ya es lento
      pvm_lvgroup('pi');                % ver Caveat en man pvm_reduce
    end
  end  % MastFLG
      pvm_exit;
      pvm_setopt(3,0);                  % AutoErr Ignore
      pvm_halt;

else % if C
%%%%%%%%%%%%%%
% SECUENCIAL %
%%%%%%%%%%%%%%
  tic %
%%%%%%%
  width=1/N; Sum=0;
  i=0:N-1;              % for i=0:N-1
  x=(i+0.5)*width;      %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2)); %   Sum=Sum+4/(1+x^2);
                        % end
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%
  Data.time=toc;%
%%%%%%%%%%%%%%%%%

end % if C

if abs(Data.err)>5E-10, warning('menudo valor de pi!'), end
```

Listado A.8: Continuación.

06/29/01 15:36:27 — ../PVMTB/Work.m — 1

```
function Work(C,N,MOD,ENC)
% WORK: Esclavo para cálculo pi PVMTB
%
%       Work ( C, N, MOD, ENC )
%
%  C    número de computadores esclavos que cooperan
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación: (s)end (r)educe (p)send
%  ENC  modalidad de encaminamiento: (d)irecto (n)o
%
% el área calculada se envía al maestro
% hay un argumento implícito, inum, que se asigna por paso de mensajes
% el número de instancia vale de 1 (primera instancia) a C (última)
% inum-1 equivale al rango MPI en el comunicador de los esclavos
% se divide [0, 1] en N subdivisiones
% la instancia inum calcula las áreas de índice congruente inum-1 módulo C
% inum -> subdivisiones calculadas
% 1       0  C   2C   3C  ...
% 2       1  C+1 2C+1 3C+1...
% ...     . .... .... ....
% C     C-1 2C-1 3C-1 4C-1...
%

global TAG PLACE ROU DONT DIR
if ENC=='d', pvm_setopt(ROU,DIR);       % PvmRouteDirect
        else,pvm_setopt(ROU,DONT);
end


%%%%%%%%%%%%%%
% Synch Mast %
%%%%%%%%%%%%%%
  inum  =0;
  me    =pvm_mytid;
  parent=pvm_parent;                    % obtener inum, de paso Synch
  switch MOD
  case 'r'
      pvm_psend(parent,TAG, me);        % esto es para ruta directa
      pvm_precv(parent,TAG,'me',1);
      inum=pvm_joingroup('pi');         % obtener inum
      pvm_freezegroup('pi',C+1);        % barrera Synch
  case 's'                              % de paso se ejercita ruta directa
      pvm_initsend(PLACE);  pvm_pkdouble (me); pvm_send(parent,TAG);
      pvm_recv(parent,TAG); pvm_upkdouble(me);          % antes de clock
      pvm_recv(parent,TAG); pvm_upkdouble(inum);        % después de clock
  case 'p'
      pvm_psend(parent,TAG, me);
      pvm_precv(parent,TAG,'me',1);                     % antes
      pvm_precv(parent,TAG,'inum',1);                   % después
  end

%%%%%%%%%%%
% Cálculo %
%%%%%%%%%%%
  width=1/N; lsum=0;                    % código vectorizado equivalente a
  i=inum-1:C:N-1;                       % for i=inum-1:C:N-1
  x=(i+0.5)*width;                      %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                %   lsum=lsum+4/(1+x^2);
                                        % end
switch MOD
  case 'r', pvm_reduce('Sum',lsum,TAG,'pi',0);
            pvm_barrier('pi',C+1);      % ver Caveat en man pvm_reduce
            pvm_lvgroup('pi');
  case 's', pvm_initsend(PLACE); pvm_pkdouble(lsum); pvm_send(parent,TAG);
  case 'p', pvm_psend(parent,TAG,lsum);
end

  pvm_exit;
```

Listado A.9: pvmtb/Work.m: Programa esclavo PVMTB para el estudio de escalabilidad basado en el cálculo de π. Fragmentos correspondientes al modo de reducción se mostraron en el Listado 1.8.

06/22/01 20:22:22 ../MPITB/Mast.m 1

```
function Data=Mast(C,N,MOD,ENC)
% MAST: Master para cálculo pi MPITB
%
%       Data = Mast ( C, N, MOD, ENC )
%
%  C    número de computadores usados (NO incluyendo éste)
%       Si C= 0, se hace el algoritmo en este ordenador
%       Si C=-1, se hace en el primer esclavo
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación:  (s)end (r)educe
%  ENC  modalidad de encaminamiento: (d)irecto (n)o
%
%  Data estructura conteniendo:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%
% pi se calcula integrando numéricamente arctg'(x) = 1/(1+x^2) entre 0 y 1
% el resultado es pi/4, multiplicado por 4 da pi
% MATLAB proporciona una función pi, que se usa para calcular el error
%
% Se espera que lamboot haya arrancado al menos C+1 computadores
% Se puede hacer MPI_Init: hacer coincidir ENC / TROLLIUSRTF (RTF_MPIC2C)
% - si no cuadra TROLLIUSRTF, el test se para
% - si no se hace MPI_Init, el test se encarga de hacerlo cuadrando ENC
% Se pueden hacer los dos modos (s,r) de un ENC (d?) usando "Todos"
% Para usar el otro ENC (n?), salir, volver a entrar, seguir con "Todos"
% "Todos" también se encarga de lamboot
%

Data=[]; TAG=7; DEBUG=0;
                         MastFLG=0;      % Para ejecutar Mast(0) en ox1
if nargin>1 & C<0, C=1; MastFLG=1; end  % en cuyo caso se devuelve por disco
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<4, help(mfilename), return, end
flag=0;                                 CC=int2str(C);
flag=flag | fix(C)~=C | C<0;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;            MOD=lower(MOD);
flag=flag | isempty(findstr(MOD,'sr')); ENC=lower(ENC);
flag=flag | isempty(findstr(ENC,'dn'));
if flag, help(mfilename), return, end

if C
%%%%%%%%%%%%%%%%%%%
% Test daemon LAM % Número de hosts>C
%%%%%%%%%%%%%%%%%%%
  [info flag]=MPI_Initialized;
  if info | ~flag
    RTFLAGS=RTF_HOMOG+RTF_MPIC2C*(ENC=='d');
    putenv(['TROLLIUSRTF=' int2str(RTFLAGS)]), MPI_Init;
  end
  RTFLAGS=str2num(getenv('TROLLIUSRTF'));
  if ~bitand(RTFLAGS,RTF_HOMOG)
        warning('usar mpirun -O para cluster homogéneo!!!');
  end
  if ~bitand(RTFLAGS,RTF_MPIC2C)
    if ENC~='n', error('parámetro ENC no coincide'); end
  else
    if ENC~='d', error('parámetro ENC no coincide'); end
  end
  [info attr flag]=MPI_Attr_get(MPI_COMM_WORLD,MPI_UNIVERSE_SIZE);
  if info | ~flag, error('atributo MPI_UNIVERSE_SIZE no existe?'), end
  if attr<=C, error(['necesito ' int2str(C+1) ' computadores en LAM']), end

%%%%%%%%%
% SPAWN %
%%%%%%%%%
  if DEBUG
    DISP=getenv('DISPLAY');
    [info children errs]=MPI_Comm_spawn('xterm',...
                {'-display' DISP '-e' 'matlab'},C,'NULL',0,'SELF');
  else
    [info children errs]=MPI_Comm_spawn('matlab',...
                                   {'-nosplash'},C,'NULL',0,'SELF');
  end
  if info | any(errs),  error('no puedo arrancar MATLABs'), end
  [info numt] = MPI_Comm_remote_size(children);
  if info | numt~=C,    error('no puedo arrancar MATLABs'), end
  [info NEWORLD]=MPI_Intercomm_merge(children,0);
  if info,              error('no puedo arrancar MATLABs'), end

  if DEBUG
    MPI_Errhandler_set(NEWORLD,MPI_ERRORS_RETURN);
  end

%%%%%%%%%%%
% COMANDO %
%%%%%%%%%%%
  if MastFLG                              % Caso Mast(0) en ox1
    FN=[ pwd '/tmp.mat'];
    cmd=['Data=Mast(0,' NN ',''' MOD ''',''' ENC '''); save ' FN ' Data'];
    MPI_Barrier(NEWORLD); MPI_Bcast(cmd,0,NEWORLD);
    while ~exist(FN,'file'), pause(1), end
    load(FN), delete(FN)
  else

%%%%%%%%%%%%
% PARALELO %
%%%%%%%%%%%%
    cmd=['Work(' CC ',' NN ',''' MOD ''')'];
    MPI_Barrier(NEWORLD);                 % Synch para cronómetro
%%%%%%%%%
    tic %
%%%%%%%%%
    MPI_Bcast(cmd,0,NEWORLD);             % emitir argumentos
    P2=0; Psum=0;
    switch(MOD)
    case 'r' %%%%%%%%%%%%%
        MPI_Reduce(P2,Psum,'SUM',0,NEWORLD);
    case 's' %%%%%%%%%%%%%
    for numt=1:C,  MPI_Recv(P2,MPI_ANY_SOURCE,TAG,NEWORLD); Psum=Psum+P2; end
    end

    Psum     =Psum/N;           % mejor hacerlo en master: no perder precisión
    Data.pi  =Psum;
    Data.err =Psum-pi;
%%%%%%%%%%%%%%%%%%%
    Data.time=toc;%
%%%%%%%%%%%%%%%%%%%
%   MPI_Finalize;               % No, no, por si queremos seguir
  end  % MastFLG

else % if C
%%%%%%%%%%%%%%
```

Listado A.10: mpitb/Mast.m: Programa maestro MPITB para el estudio de escalabilidad basado en el cálculo de π. Se mostraron fragmentos en los Listados 1.9 y 1.10.

06/22/01 20:22:22 ../MPITB/Mast.m **2**

```
% SECUENCIAL %
%%%%%%%%%%%%%%
  tic %
%%%%%%%
  width=1/N; Sum=0;
  i=0:N-1;                % for i=0:N-1
  x=(i+0.5)*width;        %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2)); %   Sum=Sum+4/(1+x^2);
                          % end
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%
  Data.time=toc;%
%%%%%%%%%%%%%%%%%

end % if C

if abs(Data.err)>5E-10, warning('menudo valor de pi!'), end
```

Listado A.10: Continuación.

07/02/01
11:39:31

../MPITB/Work.m

1

```
function Work(C,N,MOD)
% WORK: Esclavo para cálculo pi MPITB
%
%       Work ( C, N, MOD )
%
%  C    número de computadores esclavos que cooperan
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación: (s)end (r)educe
%
% el área calculada se envía al maestro
% hay un parámetro implícito, el rango (MPI rank) del proceso
% el rango vale de 1 (primer esclavo) a C (último)
% se divide [0, 1] en N subdivisiones
% el rango rank calcula las áreas de índice congruente rank-1 módulo C
% rank -> subdivisiones calculadas
% 1       0  C   2C   3C  ...
% 2       1  C+1 2C+1 3C+1...
% ...     . .... .... ....
% C     C-1 2C-1 3C-1 4C-1...
%

TAG=7;
global NEWORLD  % intracomunicador con padre heredado de startup_slave
%MPI_Barrier(NEWORLD);                  % Synch inicial
NULL=[];
MPI_Send(NULL,0,TAG,NEWORLD);
[info rank]=MPI_Comm_rank(NEWORLD);     % rango en intracomunicador = 1..C

%%%%%%%%%%%
% Cálculo %
%%%%%%%%%%%
  width=1/N; lsum=0;                    % código vectorizado equivalente a
  i=rank-1:C:N-1;                       % for i=rank-1:C:N-1
  x=(i+0.5)*width;                      %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                %   lsum=lsum+4/(1+x^2);
                                        % end
switch MOD
  case 'r',             dum=0;
        MPI_Reduce(lsum,dum,'SUM',0,NEWORLD);   % master = (0 en NEWORLD)
  case 's'
        MPI_Send(lsum,0,TAG,NEWORLD);
end
```

Listado A.11: mpitb/Work.m: Programa esclavo MPITB para el estudio de escalabilidad basado en el cálculo de π. Aparece también en el Listado 1.11.

06/22/01 20:22:50 ../DPTB/Mast.m 1

```
function Data=Mast(C,N,ENC)
% MAST: Master para cálculo pi DP-TB
%
%       Data = Mast ( C, N, ENC )
%
%  C    número de computadores usados (NO incluyendo éste)
%       Si C= 0, se hace el algoritmo en este ordenador
%       Si C=-1, se hace en el primer esclavo
%  N    número de subdivisiones del intervalo [0, 1]
%  ENC  modalidad de encaminamiento: (d)irecto (n)o
%
%  Data estructura conteniendo:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%
% pi se calcula integrando numéricamente arctg'(x) = 1/(1+x^2) entre 0 y 1
% el resultado es pi/4, multiplicado por 4 da pi
% MATLAB proporciona una función pi, que se usa para calcular el error
%
% Se configura y arranca PVM
% Si estaba previamente arrancado, se comprueba que haya C esclavos
% Pensado para llamar iterativamente desde "Todos"
%

HN={['* ep=$PATH wd=' pwd],'ox1','ox2','ox3','ox4','ox5','ox6','ox7','ox8'};
Data=[]; global TAG RAW PLACE ROU DONT DIR; DEBUG=0;

                        MastFLG=0;      % Para ejecutar Mast(0) en ox1
if nargin>1 & C<0, C=1; MastFLG=1; end  % en cuyo caso se devuelve por disco
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<3, help(mfilename), return, end
flag=0;                                 CC=int2str(C);
flag=flag | fix(C)~=C | C<0;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;            ENC=lower(ENC);
flag=flag | isempty(findstr(ENC,'dn'));
if flag, help(mfilename), return, end

if C>=length(HN), error('añadir ordenadores en código fuente Mast.m'), end

if C
%%%%%%%%%%%%%%%%%%%
% Test daemon PVM % Número de hosts>C
%%%%%%%%%%%%%%%%%%%
  pvm_setopt(3,0);                      % AutoErr Ignore
  pvme_default_config(str2mat(HN{1:C+1}));
  pvme_start_pvmd('d',1);               % * ep, wd + C ordenadores
  pvm_setopt(3,1);                      % AutoErr Warn
  if pvm_config<=C,error(['necesito ' int2str(C+1) ' computadores en PVM']),end
  if ENC=='d', pvm_setopt(ROU,DIR); end % PvmRouteDirect

%%%%%%%%%%%
% COMANDO %
%%%%%%%%%%%
  if MastFLG                            % Caso Mast(0) en ox1
    FN=[ pwd '/tmp.mat'];
    cmd=['Data=Mast(0,' NN ',''' ENC '''); save ' FN ' Data'];
  else
    cmd=['Work(' CC ',' NN ')'];
  end
  putenv(['cmd=' cmd]); pvm_export('cmd');

%%%%%%%%%
% SPAWN %
%%%%%%%%%
  if DEBUG
    DISP=getenv('DISPLAY');
    [tids numt]=pvm_spawn('xterm',...
               str2mat({'-display' DISP '-e' 'matlab'}),33,'.',C);
  else
    [tids numt]=pvm_spawn('matlab','-nosplash',         33,'.',C);
  end
  if numt~=C, error('no puedo arrancar MATLABs'), end

%%%%%%%%%%%%%
% Resultado Secuencial caso Mast(0) en ox1
%%%%%%%%%%%%%
  if MastFLG
    while ~exist(FN,'file'), pause(1), end
    load(FN), delete(FN)
  else

%%%%%%%%%%%%
% PARALELO %
%%%%%%%%%%%%                            % Anotar tids conforme respondan
        for numt=1:C, pvm_recv(-1,TAG); tids(numt)=pvm_upkdouble(1,1); end

%%%%%%%%%
    tic %               % Medir tras Synch inicial (equivale a MPI_Init)
%%%%%%%%%
    for numt=1:C                        % Asignar inum (equivale a emitir args)
        pvm_initsend(PLACE); pvm_pkdouble(numt,1,1); pvm_send(tids(numt),TAG);
    end

%%%%%%%%%%%%%
% Resultado Paralelo
%%%%%%%%%%%%%
    Psum=0;
    for numt=1:C, pvm_recv(-1,TAG); Psum=Psum+pvm_upkdouble(1,1); end

    Psum     =Psum/N;           % mejor hacerlo en master: no perder precisión
    Data.pi  =Psum;
    Data.err =Psum-pi;
%%%%%%%%%%%%%%%%%%%
    Data.time=toc;%
%%%%%%%%%%%%%%%%%%%

  end  % MastFLG
      pvm_exit;
      pvm_setopt(3,0);                  % AutoErr Ignore
      pvme_halt;

else % if C
%%%%%%%%%%%%%%
% SECUENCIAL %
%%%%%%%%%%%%%%
  tic %
%%%%%%%
  width=1/N; Sum=0;
  i=0:N-1;               % for i=0:N-1
  x=(i+0.5)*width;       %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2)); %   Sum=Sum+4/(1+x^2);
                         % end
```

Listado A.12: DPTB/Mast.m: Programa maestro DP-TB para el estudio de escalabilidad basado en el cálculo de π. Se mostraron fragmentos en los Listados 1.12 y 1.14.

06/22/01 20:22:50 ../DPTB/Mast.m 2

```
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%
  Data.time=toc;%
%%%%%%%%%%%%%%%%%

end % if C

if abs(Data.err)>5E-10, warning('menudo valor de pi!'), end
```

Listado A.12: Continuación.

06/22/01 20:22:48 ../DPTB/Work.m **1**

```
function Work(C,N)
% WORK: Esclavo para cálculo pi DP-TB
%
%        Work ( C, N )
%
%  C     número de computadores esclavos que cooperan
%  N     número de subdivisiones del intervalo [0, 1]
%
% el área calculada se envía al maestro
% hay un argumento implícito, inum, que se asigna por paso de mensajes
% el número de instancia vale de 1 (primera instancia) a C (última)
% inum-1 equivale al rango MPI en el comunicador de los esclavos
% se divide [0, 1] en N subdivisiones
% la instancia inum calcula las áreas de índice congruente inum-1 módulo C
% inum -> subdivisiones calculadas
% 1        0  C   2C   3C  ...
% 2        1  C+1 2C+1 3C+1...
% ...      . .... .... ....
% C      C-1 2C-1 3C-1 4C-1...
%

global TAG PLACE

%%%%%%%%%%%%%%
% Synch Mast %
%%%%%%%%%%%%%%
    me=pvm_mytid;
    parent=pvm_parent;                  % obtener inum, de paso Synch
  pvm_initsend(PLACE); pvm_pkdouble(me,1,1); pvm_send(parent,TAG);
  pvm_recv(parent,TAG); inum=pvm_upkdouble(1,1);

%%%%%%%%%%%
% Cálculo %
%%%%%%%%%%%
  width=1/N; lsum=0;                    % código vectorizado equivalente a
  i=inum-1:C:N-1;                       % for i=inum-1:C:N-1
  x=(i+0.5)*width;                      %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                %   lsum=lsum+4/(1+x^2);
                                        % end

  pvm_initsend(PLACE); pvm_pkdouble(lsum,1,1); pvm_send(parent,TAG);
  pvm_exit;
```

Listado A.13: DPTB/Work.m: Programa esclavo DP-TB para el estudio de escalabilidad basado en el cálculo de π. Aparece también en el Listado 1.13.

06/23/01 10:52:24 ../MM/MultiMATLAB **1**

```
#!/bin/bash

############################################################################
# Usage: ./MultiMATLAB  [-c | -l]  [-d]  [C]
# -c -l: [-c2c] / -lamd options for mpirun
# -d   : debug flag. MATLABs will be run on xterms
# C [8]: # Computers in MultiMATLAB
############################################################################
DEBUG=no
OPTS='-c2c'
while getopts dcl opt; do
  case $opt in
    d)
        DEBUG=yes
        ;;
    c)
        OPTS=-c2c
        ;;
    l)
        OPTS=-lamd
        ;;
    *)
echo
echo Usage: ./MultiMATLAB [-c \| -l] [ C  [ X ] ]
echo -c -l: [-c2c] / -lamd options for mpirun
echo -d   : debug flag. MATLABs will be run on xterms
echo C [8]: \# slave Computers in MultiMATLAB
echo
exit
        ;;
  esac
done
if [ $[OPTIND-=1] ]; then shift $OPTIND; fi

############################################################################
# Crear hostfile. Listar ox0 primero. Matar posible LAM anterior
############################################################################
rm -f   bhost.def
i=-1
while [ $[i+=1] -le ${1:-8} ]; do
 cat >> bhost.def <<-@
        ox$i
        @
done

############################################################################
echo Starting LAM. Please wait
lamboot bhost.def


############################################################################
echo Starting MultiMATLAB. Please wait

XARG="-f"
XCMD="matlab"
if [ $DEBUG = yes ]; then
  if [ -z $DISPLAY ]; then
    echo no DISPLAY envvar, cannot run in debug \(xterm\) mode
  else
    if [ $DISPLAY = :0 ]; then
      xhost +
      export DISPLAY=ox0:0
    fi
    XARG="-x DISPLAY"
    XCMD="xterm -- -e matlab"
    echo "####  Use proxymngr when remote DISPLAY ####"
  fi
fi

echo mpirun $OPTS -w -O $XARG N $XCMD
mpirun $OPTS -w -O $XARG N $XCMD

############################################################################
echo Wiping LAM. Please wait
wipe bhost.def
rm   bhost.def
```

Listado A.14: Comando MultiMATLAB. Contempla las opciones de encaminamiento mpirun −O|−c2c, una opción de depuración −d en la cual los procesos MATLAB se ejecutan bajo xterm, y un argumento indicando el número deseado de procesos esclavos. Se mostró una versión reducida en el Listado 1.15.

06/23/01
10:52:44

../MM/Mast.m

1

```
function Data=Mast(C,N,MOD,ENC)
% MAST: Master para cálculo pi MultiMATLAB
%
%       Data = Mast ( C, N, MOD, ENC )
%
%  C    número de computadores usados (NO incluyendo éste)
%       Si C= 0, se hace el algoritmo en este ordenador
%       Si C=-1, se hace en el primer esclavo
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación:  (s)end (p)ut
%  ENC  modalidad de encaminamiento: (d)irecto (n)o
%
%  Data estructura conteniendo:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%
% pi se calcula integrando numéricamente arctg'(x) = 1/(1+x^2) entre 0 y 1
% el resultado es pi/4, multiplicado por 4 da pi
% MATLAB proporciona una función pi, que se usa para calcular el error
%
% Se habrá arrancado con ./MultiMATLAB, que ya fija -c2c|-lamd (ENC='d'|'n')
% Sólo (s)end y (p)ut, (r)educe es global, implica a todos los MATLAB, y
%       habría que rearrancar ./MultiMATLAB cada vez que se
%       añadiera un host al estudio de escalabilidad -> prohibitivo
% Se pueden hacer los dos modos (s,p) de un ENC (d?) usando "Todos"
% Para usar el otro ENC (n?), salir, volver a entrar, seguir con "Todos"
%

Data=[];
RTF_HOMOG=hex2dec('002000');
RTF_MPIC2C=hex2dec('000080');
                            MastFLG=0;      % Para ejecutar Mast(0) en ox1
if nargin>1 & C<0, C=1; MastFLG=1; end  % se puede hacer con Eval también
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<4, help(mfilename), return, end
flag=0;                                 CC=int2str(C);
flag=flag | fix(C)~=C | C<0;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;            MOD=lower(MOD);
flag=flag | isempty(findstr(MOD,'sp')); ENC=lower(ENC);
flag=flag | isempty(findstr(ENC,'dn'));
if flag, help(mfilename), return, end

if C
%%%%%%%%%%%%%%%%%%%
% Test daemon LAM % Número de hosts>C
%%%%%%%%%%%%%%%%%%%
  RTFLAGS=str2num(getenv('TROLLIUSRTF'));
  if ~bitand(RTFLAGS,RTF_HOMOG)
        warning('usar mpirun -O para cluster homogéneo!!!');
  end
  if ~bitand(RTFLAGS,RTF_MPIC2C)
    if ENC~='n', error('parámetro ENC no coincide'); end
  else
    if ENC~='d', error('parámetro ENC no coincide'); end
  end
  if Nproc<=C, error([ int2str(C+1) ' MATLABs required on MultiMATLAB']), end

% SPAWN % no hay, ya se hizo

%%%%%%%%%%%
% COMANDO %
%%%%%%%%%%%
  if MastFLG                            % Caso Mast(0) en ox1
    Eval(1,['Data=Mast(0,' int2str(N) ',''' MOD ''',''' ENC ''')']);
    Data=Get(1,'Data')
    Eval(1,'clear Data functions')
  else

%%%%%%%%%%%%
% PARALELO %
%%%%%%%%%%%%
    tic %                               % medir antes de emitir argumentos
%%%%%%%%%
                Psum=0;                 % evitar warning "uninitialized"
    Eval([1:C],['Psum=Work(' CC ',' NN ',''' MOD ''')'])
    switch MOD
      case 's', for i=1:C, Psum=Psum + Recv; end
      case 'p', for i=1:C, Psum=Psum + Get(i,'Psum'); end
    end

    Psum     =Psum/N;
    Data.pi  =Psum;
    Data.err =Psum-pi;
%%%%%%%%%%%%%%%%%%%
    Data.time=toc;%
%%%%%%%%%%%%%%%%%%%
    Eval([1:C],'clear Psum functions')
  end  % MastFLG

else % if C
%%%%%%%%%%%%%%
% SECUENCIAL %
%%%%%%%%%%%%%%
%%%%%%%
  tic %
%%%%%%%
  width=1/N; Sum=0;
  i=0:N-1;                  % for i=0:N-1
  x=(i+0.5)*width;          %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2)); %   Sum=Sum+4/(1+x^2);
                            % end
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%
  Data.time=toc;%
%%%%%%%%%%%%%%%%%

end % if C

if abs(Data.err)>5E-10, warning('menudo valor de pi!'), end
```

Listado A.15: MM/Mast.m: Programa maestro MultiMATLAB para el estudio de escalabilidad basado en el cálculo de π. Se mostró un fragmento en el Listado 1.16.

06/23/01 10:52:47 ../MM/Work.m 1

```
function lsum=Work(C,N,MOD)
% WORK: Esclavo para cálculo pi MultiMATLAB
%
%       lsum = Work ( C, N, MOD )
%
%  C    número de computadores esclavos que cooperan
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación: (s)end (p)ut
%
%  lsum área calculada (se envía al maestro también cuando MOD==’s’)
%
% hay un parámetro implícito, el rango (MPI rank) del proceso
% el rango vale de 1 (primer esclavo) a C (último)
% se divide [0, 1] en N subdivisiones
% el rango rank calcula las áreas de índice congruente rank-1 módulo C
% rank -> subdivisiones calculadas
% 1       0  C   2C   3C  ...
% 2       1  C+1 2C+1 3C+1...
% ...     . .... .... ....
% C     C-1 2C-1 3C-1 4C-1...
%

rank=ID;

%%%%%%%%%%%
% Cálculo %
%%%%%%%%%%%
  width=1/N; lsum=0;                    % código vectorizado equivalente a
  i=rank-1:C:N-1;                       % for i=rank-1:C:N-1
  x=(i+0.5)*width;                      %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                %   lsum=lsum+4/(1+x^2);
                                        % end

if strcmp(MOD,’s’), Send(0,lsum); end           % Put/Get es one-sided
```

Listado A.16: MM/Work.m: Programa esclavo MultiMATLAB para el estudio de escalabilidad basado en el cálculo de π. Aparece también en el Listado 1.17.

06/23/01 10:53:28 ../TCPIP/Spawner.m **1**

```
function Spawner(host,lines,debug)
% SPAWNER:      Arrancador de MATLABs remotos para TCPIP
%
%       Spawner ( host, lines [,debug] )
%
%  host         nombre del ordenador remoto
%  lines        cellstr conteniendo comandos MATLAB a ejecutar
%  debug [0]    0: se ejecuta MATLAB a secas sin xterm y hace quit
%               1: se ejecuta bajo xterm y no hace quit
%               2: además entra en modo debug
%
%  La idea es meter lines en startup.m (ya que no hay startup para TCPIP)
%  El propio startup se borra a sí mismo
%  Ver código "lines" usado en Speedup
%

if nargin<3, debug=0; end

fid=fopen('startup.m','w');     if debug>1
  fprintf(fid,'dbtype\n');
  fprintf(fid,'keyboard\n');    end
  fprintf(fid,'delete startup.m\n');
  fprintf(fid,'%s\n',lines{:}); if ~debug
  fprintf(fid,'quit\n');        end
fclose(fid);

  unix_cmd=['rcp startup.m ' host ':`pwd`'];    % Asunto actualización NFS
  unix(unix_cmd);

if debug
  DISP=getenv('DISPLAY'); errmsg=[];
  if isempty(DISP),     errmsg='se indicó debug sin tener DISPLAY'; end
  if isempty(findstr('ox',DISP))
                        errmsg='**** recordar usar . proxy ****'; end
  if ~isempty(errmsg), delete startup.m, error(errmsg), end
        unix_cmd=['xterm -display $DISPLAY -e matlab'];
else,   unix_cmd=['matlab > /dev/null'];
end
        unix_cmd=['rsh -n ' host ' "cd ' pwd '; ' unix_cmd '" &'];
        unix(unix_cmd);
```

Listado A.17: Comando Spawner. Se mostró una versión reducida en el Listado 1.18.

06/23/01 10:53:33 ../TCPIP/Speedup.m 1

```
function [Par,Sec]=Speedup(SUB,C,N,MOD)
% SPEEDUP:      Generación fichero y/o gráfica de speedup cálculo pi TCPIP
%
%       [ Par Sec ] = Speedup ( SUB, C, N, MOD )
%
%  SUB  subdirectorio fichero mediciones
%  C    número computadores estudio escalabilidad
%  N    número subdivisiones para cálculo pi
%  MOD  modalidad de comunicación:  (w)rite (s)end
%
%  Par  Par(i) contiene resultados sobre i ordenadores esclavos ox1-ox<i>:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%  Sec  la misma estructura para algoritmo secuencial
%       en oxigeno: Sec(1) y en ox1: Sec(2)
%
%       Se dibuja la gráfica de (tiempo/speedup vs. #C) a partir del fichero
%               <SUB>/Speedup.<C>-<N>-<MOD>-d.mat
%       Si no existe, se genera
%

[Par,Sec]=deal([]); DEBUG=0;
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<4, help(mfilename), return, end
flag=0;
flag=flag | ~exist(SUB,'dir');          CC=int2str(C);
flag=flag | fix(C)~=C | C<1;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;            MOD=lower(MOD);
flag=flag | isempty(findstr(MOD,'ws')); ENC='d';
if flag, help(mfilename), return, end

%%%%%%%%%%%%%%%%%%%
% Generar Fichero % o Cargar Datos directamente
%%%%%%%%%%%%%%%%%%%
FN=[SUB '/Speedup.' CC '-' NN '-' MOD '-' ENC '.mat'];
if exist(FN,'file'), load(FN)
else
        disp(['Mast( 0,' NN ',''' MOD ''')'])
        fprintf('secuencial en maestro...')
        Sec   =Mast( 0,N,MOD); fprintf('\n'); disp(Sec   )

        disp(['Mast(-1,' NN ',''' MOD ''')'])
        Spawner('ox1',...
            {['Work( 0,' NN ',''' MOD ''')']}, DEBUG)
        if DEBUG, input(['Pulsar <Enter> tras asegurarse de que hay ',...
                    '1 esclavo ejecutando Work: ']); end
        fprintf('secuencial en esclavo...')
        Sec(2)=Mast(-1,N,MOD); fprintf('\n'); disp(Sec(2))

        % Spawner arranca MATLABs con este startup -> Works responden a Mast
        startup={'start=00;',...        % esta línea es la que cambia
                ['C=' CC ';'],...       % todos cooperan en la última iteración
                 'for i=start:C',...    % cada uno hace Work varias veces
                 '  i',...              % (informativo para modo DEBUG)
                ['  Work(i,' NN ',''' MOD ''')'],...
                 '  clear Work',...     % ox1 participa 8 veces (todas)
                 'end'};                % ox8 participa 1 vez (la última)

        Par=Sec(1); Par(C)=Sec(1);      % Pre-allocation, luego se machaca

    for i=1:C,        ii=int2str(i);
        startup{1}=['start=' ii ';'];
        disp(['Mast(' ii ',' NN ',''' MOD ''')'])
        Spawner(['ox' ii], startup, DEBUG)
        if DEBUG, input(['Pulsar <Enter> tras asegurarse de que hay ',...
                      ii ' esclavos ejecutando Work: ']); end
        fprintf('paralelo en %d esclavos...',i);
        Par(i)=Mast( i,N,MOD); fprintf('\n'); disp(Par(i))
    end

    save(FN,'C','Sec','Par')

end % exist(FN)

%%%%%%%%%%%%%%%%%%%
% Colores Speedup %
%%%%%%%%%%%%%%%%%%%
CPAR=[0 0 1];   SPAR='.-';              % Color/Símbolo paralelos (clientes)
CSLV=[0 1 0];   SSLV='p';               % Color/Símbolo 1 cliente
CSRV=[1 0 0];   SSRV='h';               % Color/Símbolo servidor

%%%%%%%%%%
% Título %
%%%%%%%%%%
mods='ws'; MODS={'write' 'send'};

 XL='Número de computadores';
YL1='Tiempo absoluto (s)';
YL2='Ganancia en velocidad';
TIT=['Escalabilidad cálculo pi TCPIP, ' NN ' subd, ',...
        MODS{findstr(mods,MOD)}];

%%%%%%%%%%%
% Gráfica %
%%%%%%%%%%%
figure
pp=get(gcf,'PaperPosition'); pp(4)=pp(4)*3/5;
   set(gcf,'PaperPosition', pp)

subplot 121, xlabel(XL), ylabel(YL1)
plot([Par.time], SPAR ,'Color',CPAR)
hold on
plot( Sec(2).time, SSLV ,'Color',CSLV)
plot( Sec(1).time, SSRV ,'Color',CSRV)
legend('varios Clientes','Cliente@333MHz','Servidor@400MHz',1)

text(1+3/16,1.05,TIT,'Units','Normalized','Horiz','Center','Vert','Bottom');

subplot 122, xlabel(XL), ylabel(YL2)
plot(Sec(2).time./[Par.time],[SSLV '-'],'Color',CSLV)
hold on
plot(Sec(1).time./[Par.time],[SSRV '-'],'Color',CSRV)
legend('respecto Cliente','respecto Servidor',2)
```

Listado A.18: Script Speedup. Se mostró una versión reducida en el Listado 1.19.

06/23/01 10:53:39 ../TCPIP/Mast.m 1

```
function Data=Mast(C,N,MOD)
% MAST: Master para cálculo pi TCPIP
%
%       Data = Mast ( C, N, MOD )
%
%  C    número de computadores usados (NO incluyendo éste)
%       Si C= 0, se hace el algoritmo en este ordenador
%       Si C=-1, se hace en el primer esclavo
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación:  (w)rite (s)end
%
%  Data estructura conteniendo:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%
% pi se calcula integrando numéricamente arctg'(x) = 1/(1+x^2) entre 0 y 1
% el resultado es pi/4, multiplicado por 4 da pi
% MATLAB proporciona una función pi, que se usa para calcular el error
%
% Se debe arrancar previamente matlab en los ordenadores esclavos
% y ejecutar en ellos "Work". "Mast" espera hasta que haya C esclavos.
% También se puede automatizar el proceso con Spawner, invocando "Todos"
%


Data=[]; PORT=2000;     MastFLG=0;      % Para ejecutar Mast(0) en esclavo
if nargin>0 & C<0, C=1; MastFLG=1; end  % en cuyo caso Work llamará a Mast
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<3, help(mfilename), return, end
flag=0;                                 CC=int2str(C);
flag=flag | fix(C)~=C | C<0;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;            MOD=lower(MOD);
flag=flag | isempty(findstr(MOD,'ws'));
if flag, help(mfilename), return, end

if C
%%%%%%%%%%%
% Connect %
%%%%%%%%%%%
  sock=tcpip_servsocket(PORT);                  % Abrir puerto servicio

  fids=-1*ones(1,C);                            % Abrir conexiones diálogo
  for i=1:C
    while fids(i)<0
      fids(i)=tcpip_listen(sock);
    end
  end

  tcpip_close(sock)                             % servicio cumplido, cerrar

%%%%%%%%%%%%%%%
% SECUENCIAL % REMOTO
%%%%%%%%%%%%%%%
  if MastFLG                                    % Caso Mast(0) en esclavo
      Data=tcpip_getvar(fids(1));               % Work(0,0,N,MOD)->Mast(0,N,MOD)

  else
%%%%%%%%%%%%
% PARALELO %
%%%%%%%%%%%%
    tic %
%%%%%%%%%
    switch MOD                                  % emitir argumentos
      case 'w', for i=1:C, tcpip_write  (fids(i),i-1); end
      case 's', for i=1:C, tcpip_sendvar(fids(i),i-1); end
    end
    Sum=0;                                      % acumular resultados
    switch MOD
      case 'w', for i=1:C, Psum=[];
                  while isempty(Psum)
                    Psum=tcpip_read(fids(i),1,'double');
                  end
                  Sum=Sum+Psum;
                end
      case 's', for i=1:C
                    Sum=Sum+tcpip_getvar(fids(i));
                end
    end % switch

    Sum      =Sum/N;
    Data.pi  =Sum;
    Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%%%
    Data.time=toc;%
%%%%%%%%%%%%%%%%%%%
  end % MastFLG

  for i=1:C,    tcpip_sendvar(fids(i),i);       % Handshake avisando de cierre
  pause(0.1),   tcpip_close  (fids(i));         % Cerrar
  end

else % if C
%%%%%%%%%%%%%%
% SECUENCIAL % LOCAL
%%%%%%%%%%%%%%
  tic %
%%%%%%
  width=1/N; Sum=0;
  i=0:N-1;              % for i=0:N-1
  x=(i+0.5)*width;      %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2)); %   Sum=Sum+4/(1+x^2);
                        % end
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%
  Data.time=toc;%
%%%%%%%%%%%%%%%%%

end % if C

if abs(Data.err)>5E-10, warning('menudo valor de pi!'), end
```

Listado A.19: tcpip/Mast.m: Programa maestro TCPIP para el estudio de escalabilidad basado en el cálculo de π. Se mostraron fragmentos en los Listados 1.20 y 1.22.

06/23/01 10:53:47 ../TCPIP/Work.m 1

```
function Work(C,N,MOD)
% WORK: Esclavo para cálculo pi TCPIP
%
%       Work ( C, N, MOD )
%
%  C    número de computadores esclavos que cooperan (0 para invocar Mast)
%  N    número de subdivisiones del intervalo [0, 1]
%  MOD  modalidad de comunicación:  (w)rite (s)end
%
% hay un parámetro implícito: rank (equivale a MPI rank del proceso)
% el rango vale de 0 (primer esclavo) a C-1 (último)
% se divide [0, 1] en N subdivisiones
% el rango rank calcula las áreas de índice congruente rank módulo C
% rank -> subdivisiones calculadas
% 0       0  C   2C   3C  ...
% 1       1  C+1 2C+1 3C+1...
% ...     . .... .... ....
% C-1   C-1 2C-1 3C-1 4C-1...
%

%%%%%%%%%%%
% Connect %
%%%%%%%%%%%
HOST='ox0';
PORT=2000;
fid=-1;
while fid<0
  fid=tcpip_open(HOST,PORT);            % Abrir conexión diálogo
end

if ~C
%%%%%%%%%%%%%%
% Secuencial %                          % Work(0,N,MOD)->Mast(0,N,MOD)
%%%%%%%%%%%%%%
  Data=Mast(C,N,MOD);                   % calcular y
  tcpip_sendvar(fid,Data)               % devolver resultado

else
%%%%%%%%%%
% ArgChk %                              % un argumento implícito adicional
%%%%%%%%%%
rank=[];                                % recepción argumento
switch MOD
  case 'w', while isempty(rank), rank=tcpip_read(fid,1,'double'); end
  case 's', rank=tcpip_getvar(fid);
end

%%%%%%%%%%%%
% Paralelo %
%%%%%%%%%%%%
  width=1/N; lsum=0;                    % código vectorizado equivalente a
  i=rank:C:N-1;                         % for i=rank:C:N-1
  x=(i+0.5)*width;                      %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));                %   lsum=lsum+4/(1+x^2);
                                        % end
  switch MOD
    case 'w',   tcpip_write  (fid,lsum) % devolver resultado
    case 's',   tcpip_sendvar(fid,lsum)
  end

end % ~C

  tcpip_getvar(fid);                    % Handshake avisando de cierre
tcpip_close (fid);                      % Cerrar
```

Listado A.20: tcpip /Work.m: Programa esclavo TCPIP para el estudio de escalabilidad basado en el cálculo de π. Se mostró un fragmento en el Listado 1.21.

06/23/01 10:54:05 ../PARA/Speedup.m **1**

```
function [Par,Sec]=Speedup(SUB,C,N)
% SPEEDUP:      Generación fichero y/o gráfica de speedup cálculo pi Paralize
%
%       [ Par Sec ] = Speedup ( SUB, C, N )
%
%  SUB  subdirectorio fichero mediciones                (sugerido: '.')
%  C    número computadores estudio escalabilidad       (sugerido:  8 )
%  N    número subdivisiones para cálculo pi            (sugerido: 408800)
%
%  Par  Par(i) contiene resultados sobre i ordenadores esclavos ox1-ox<i>:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%  Sec  la misma estructura para algoritmo secuencial en oxigeno
%
%       Se dibuja la gráfica de (tiempo/speedup vs. #C) a partir del fichero
%               <SUB>/Speedup.<C>-<N>.mat
%       Si no existe, se genera
%

[Par,Sec]=deal([]); DEBUG=0;
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<3, help(mfilename), return, end
flag=0;
flag=flag | ~exist(SUB,'dir');          CC=int2str(C);
flag=flag | fix(C)~=C | C<1;            NN=int2str(N);
flag=flag | fix(N)~=N | N<1;
if flag, help(mfilename), return, end

%%%%%%%%%%%%%%%%%%%
% Generar Fichero % o Cargar Datos directamente
%%%%%%%%%%%%%%%%%%%
FN=[SUB '/Speedup.' CC '-' NN '.mat'];
if exist(FN,'file'), load(FN)
else
        disp(['Mast( 0,' NN ')'])
        fprintf('secuencial en maestro...')
        Sec   =Mast( 0,N); fprintf('\n'); disp(Sec)

        disp(['Mast(-1,' NN ')'])
        Spawner('ox1', DEBUG)
        if DEBUG, input(['Pulsar <Enter> tras asegurarse de que hay ',...
                '1 esclavo ejecutando serve: ']); end
        fprintf('secuencial en esclavo...')
        Sec(2)=Mast(-1,N); fprintf('\n'); disp(Sec(2))

        Par=Sec(1); Par(C)=Sec(1);      % Pre-allocation, luego se machaca
        unix('rsh -n ox1 killall matlab');      % para simplificar for i=1:C

    for i=1:C,          ii=int2str(i);
        disp(['Mast('   ii ',' NN ')'])
        Spawner(['ox'   ii], DEBUG)
        if DEBUG, input(['Pulsar <Enter> tras asegurarse de que hay ',...
                        ii ' esclavos ejecutando serve: ']);
        else,   fprintf('Esperando unos segundos para que arranque serve...');
                pause(10); fprintf('\n');
        end
        fprintf('paralelo en %d esclavos...\n',i);
        Par(i)=Mast( i,N); fprintf('\n'); disp(Par(i))
    end
    save(FN,'C','Sec','Par')
    paralize('kill')

end % exist(FN)

%%%%%%%%%%%%%%%%%%%
% Colores Speedup %
%%%%%%%%%%%%%%%%%%%
CPAR=[0 0 1];   SPAR='.-';              % Color/Símbolo paralelos (clientes)
CSLV=[0 1 0];   SSLV='p';               % Color/Símbolo 1 cliente
CSRV=[1 0 0];   SSRV='h';               % Color/Símbolo servidor

%%%%%%%%%%
% Título %
%%%%%%%%%%
 XL='Número de computadores';
YL1='Tiempo absoluto (s)';
YL2='Ganancia en velocidad';
TIT=['Escalabilidad cálculo pi Paralize, ' NN ' subdivisiones'];

%%%%%%%%%%%
% Gráfica %
%%%%%%%%%%%
figure
pp=get(gcf,'PaperPosition'); pp(4)=pp(4)*3/5;
   set(gcf,'PaperPosition', pp)

subplot 121, xlabel(XL), ylabel(YL1)
plot([Par.time], SPAR ,'Color',CPAR)
hold on
plot( Sec(2).time,  SSLV ,'Color',CSLV)
plot( Sec(1).time,  SSRV ,'Color',CSRV)
legend('varios Clientes','Cliente@333MHz','Servidor@400MHz',1)

text(1+3/16,1.05,TIT,'Units','Normalized','Horiz','Center','Vert','Bottom');

subplot 122, xlabel(XL), ylabel(YL2)
plot(Sec(2).time./[Par.time],[SSLV '-'],'Color',CSLV)
hold on
plot(Sec(1).time./[Par.time],[SSRV '-'],'Color',CSRV)
legend('respecto Cliente','respecto Servidor',2)
```

Listado A.21: Script Speedup. Se mostró una versión reducida en el Listado 1.23.

06/23/01
10:54:08

../PARA/Mast.m

1

```
function Data=Mast(C,N)
% MAST: Master para cálculo pi Paralize
%
%       Data = Mast ( C, N )
%
%  C    número de computadores usados (NO incluyendo éste)
%       Si C= 0, se hace el algoritmo en este ordenador
%       Si C=-1, se hace en el primer esclavo
%       Si C=-2, se vuelve inmediatamente con Data=[0 0 0] (proceso dummy)
%  N    número de subdivisiones del intervalo [0, 1]
%
%  Data estructura conteniendo:
%       pi      valor estimado de pi
%       err     error
%       time    tiempo empleado (desde emisión argumentos hasta obtener pi)
%
% pi se calcula integrando numéricamente arctg'(x) = 1/(1+x^2) entre 0 y 1
% el resultado es pi/4, multiplicado por 4 da pi
% MATLAB proporciona una función pi, que se usa para calcular el error
%
% Se debe arrancar previamente matlab en los ordenadores esclavos
% y ejecutar en ellos "serve". "Mast" no puede comprobar que haya C esclavos.
% También se puede automatizar el proceso con Spawner, invocando "Todos"
%


Data=[];                MastFLG=0;      % Mast(0,N) y Work(0,1,N) en esclavo
if nargin>0                             % necesitarán 2° proceso dummy
  if C==-1,     C=1;    MastFLG=1; end
  if C==-2,     Data.pi  =0;            % Esto es necesario porque
                Data.err =0;            % paralize entra en debug si
                Data.time=0;            % se intenta devolver Data=[]
                        return; end     % éste es el Mast dummy
end

% Speedup llama a Mast pretendiendo ejecutar Mast/Work en distintos ordenadores
%       en servidor     en cliente(s)
% Speedup->Mast( 0,N)                   algoritmo secuencial en oxígeno
% Speedup->Mast(-1,N)-> Mast( 0,N)      alg. secuencial en ox1 (MastFLG)
%                       Mast(-2,N)      proceso dummy en ox1
% Speedup->Mast( 1,N)-> Work( 0,C==1,N) algoritmo paralelo en ox1
%                       Work( N,   1,N) proceso dummy for i=N:C:N-1
% Speedup->Mast( 2,N)-> Work( 0,C==2,N) algoritmo paralelo en ox1 y ox2
%                       Work( 1,C==2,N) no se necesitan más dummies

%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<2, help(mfilename), return, end
flag=0;
flag=flag | fix(C)~=C | C<0;
flag=flag | fix(N)~=N | N<1;
if flag, help(mfilename), return, end

if C
%%%%%%%%%%%%
% PARALELO %
%%%%%%%%%%%%                           % MATLAB no soporta singleton 3D
    i=0:C-1;                            % generar equivalente a rank/inum
    if C==1                             % C==1 -> i==[0] (singleton)
      if MastFLG, i=[0 -2];             % 1er caso: Mast(-1,N)-> seq. y dummy
      else,       i=[0  N]; end         % 2°  caso: Mast( 1,N)-> par. y dummy
    end
    I=shiftdim(i,-1);                   % pasar (1xC) -> (1x1xC) C>1 siempre
%%%%%%%%%%%%%%
% SECUENCIAL % REMOTO
%%%%%%%%%%%%%%
  if MastFLG                            % Caso Mast(0) en esclavo
      Data=paralize('Mast',I,N);        % si C==-1, 2° proceso es dummy
      Data=Data(1);                     % devolvió Data=[0 0 0] que eliminamos
  else

%%%%%%%
    tic%
%%%%%%%
    O=paralize('Work',I,C,N);           % argumento paralelización (I) es 3D

    Sum      =sum(O(:))/N;              % si C==1, 2° proceso for i=N:C:N-1
    Data.pi  =Sum;                      %                dummy ->  lsum==0
    Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%%%
    Data.time=toc;%
%%%%%%%%%%%%%%%%%%%
  end % MastFLG

else % if C
%%%%%%%%%%%%%%
% SECUENCIAL %
%%%%%%%%%%%%%%
  tic %
%%%%%%
  width=1/N; Sum=0;
  i=0:N-1;                % for i=0:N-1
  x=(i+0.5)*width;        %   x=(i+0.5)*width;
  Sum=sum(4./(1+x.^2)); %   Sum=Sum+4/(1+x^2);
                          % end
  Sum      =Sum*width;
  Data.pi  =Sum;
  Data.err =Sum-pi;
%%%%%%%%%%%%%%%%%
  Data.time=toc;%
%%%%%%%%%%%%%%%%%

end % if C

if abs(Data.err)>5E-10, warning('menudo valor de pi!'), end
```

Listado A.22: para/Mast.m: Programa maestro Paralize para el estudio de escalabilidad basado en el cálculo de π. Se mostró un fragmento en el Listado 1.24.

06/23/01 10:54:12 ../PARA/Work.m 1

```
function lsum=Work(offs,C,N)

  width=1/N; lsum=0;                % código vectorizado equivalente a
  i=offs:C:N-1;                     % for i=offs:C:N-1
  x=(i+0.5)*width;                  %   x=(i+0.5)*width;
  lsum=sum(4./(1+x.^2));            %   lsum=lsum+4/(1+x^2);
                                    % end
```

Listado A.23: para/Work.m: Programa esclavo Paralize para el estudio de escalabilidad basado en el cálculo de π. Se mostró también en el Listado 1.25.

Apéndice B

Modelos para el tiempo de transmisión

B.1 Introducción

En el Apartado 2.3 se discutieron los distintos modelos desarrollados para estimar el tiempo de transmisión de un mensaje bajo los sistemas PVM y LAM/MPI. En orden creciente de complejidad, costes de evaluación y capacidad predictiva, son:

Modelo 0: apartado 2.3.4, un tramo lineal, parámetros ls y bs.

Modelo 1: apartado 2.3.5, dos tramos lineales, parámetro de cabecera $H = h1 + s1$, tramos modelados por lc/bc, ls/bs.

Modelo 2: apartado 2.3.7, un tramo por MTU, parámetros de cabecera fs, $h1/h2/hn$, $s1/sn$, tramos lc/bc, lm y ls/bs.

Modelo 3: apartado 2.3.8, un tramo lineal por cada MTU, cada uno con su propia latencia y ancho de banda, aplicados con periodicidad 2 o $n = \frac{\|UDP\|}{\|MTU\|}$ según la combinación de opciones delatada por $h1$. Cabeceras como el Modelo 2. Parámetros de tramos: lc/bc, lm/bm, lp/bp y ls/bs.

A continuación se lista el código MATLAB usado para evaluar cada modelo. Su diseño está fuertemente influenciado por el entorno GUI y los *scripts* de automatización destinados a facilitar la extracción de parámetros y su posterior estudio y comprensión. Dado el número de diversas combinaciones de opciones bajo las distintas configuraciones con que se han recompilado ambos sistemas de paso de mensajes, PVM y LAM/MPI, era de vital importancia agilizar el tratamiento de tal cantidad de datos.

Por supuesto, el usuario siempre puede invocar el *script* de un modelo, proporcionando los tamaños de mensaje y parámetros deseados. Una sencilla orden **plot** le permitiría crear la gráfica mostrando el tiempo de transmisión predicho en función de dichos argumentos.

B.2 Modelo 0

Como ya se discutió en el Apartado 2.3.4, es el modelo afín básico, con un término independiente ls y un coeficiente lineal bs. Puede reducirse al modelo lineal sin más que ajustar $ls = 0$.

El Listado B.1 se muestra como ejemplo de la autodocumentación incluida con los *scripts* MATLAB correspondientes a cada modelo. Todos los modelos tienen una lista de argumentos y un código de inicialización similar, así que esta explicación sólo se realizará para el Modelo 0.

```
function t = MTUmodel0( s, p, varargin )
%
%       t = MTUmodel0 ( s, p [, P]... )
%
% s     array de tamaños a enviar
% p     array de parámetros variables del modelo
%         debe tener al menos tantos parámetros como falten en P
%         está hecho así para permitir buscar con fminsearch
%               congelando algunos parámetros (de P) y variando otros (de p)
%         si se desea descongelar alguno intermedio, ponerlo a empty [] en P
% P (opcional) parámetros fijos del modelo, como lista de argumentos sueltos
%       LS      latencia transmisión 0 bytes            (segundos)
%       BS      ancho de banda                          (bytes/segundo)
%         se consultan primero los valores de LS, BS
%         si alguno no está presente (por consiguiente tampoco los siguientes)
%                    o está empty    (vale [] / '' / {})
%            se toma el primer valor de p como sustituto
%            y así sucesivamente
%
% t     array de tiempos de transmisión predichos por el modelo
```

Listado B.1: Autodocumentación en el *script* MATLAB para evaluación del Modelo 0.

El primer argumento de los *scripts* de evaluación de modelos desarrollados es un vector conteniendo los tamaños de mensaje a considerar. A continuación se indican los parámetros variables del modelo, agrupados en un vector, en el orden establecido por el modelo. Por último se proporciona un vector con los parámetros fijos del modelo. El resultado debe ser un vector conteniendo los tiempos de transmisión estimados para dichos tamaños de mensaje según los parámetros indicados. Esta distinción entre parámetros fijos y variables permite evitar que el método simplex altere los tamaños de cabecera de las MTUs, y en general cualquier parámetro cuyo valor se desee establecer manualmente (por ejemplo, $lp = bp = 0$ en el Modelo 2).

El Listado B.2 es el programa MATLAB para evaluar el Modelo 0.

```
ARGS={'ls' 'bs'};
NARG=length(ARGS);
STRG=int2str(NARG);
if nargin<2 | ~isnumeric(s) |...
              ~isnumeric(p) | length(p)<[1 2+NARG-nargin]
  disp ('argumentos: vector tamaños, vector parámetros variables,')
  error(['          y parámetros fijos sueltos (total ' STRG ' parámetros)'])
end

for i=ARGS
    if  isempty(varargin) | isempty(varargin{1})
    if ~isempty(varargin),          varargin(1)=[];                 end
        if isempty(p),   error(['faltan parámetros (total ' STRG ')'])
        else,            eval([i{:} '=p(1);']), p(1)=[];            end
elseif ~isnumeric(varargin{1})
                         error('parámetro fijo no numérico')
    else,                eval([i{:} '=varargin{1};']), varargin(1)=[];    end
end
ls=ls; bs=bs;                       % parece que eval no los marca como escalares

  t = ls + s/bs;
```

Listado B.2: MTUmodel0.m: Código MATLAB para evaluación del Modelo 0.

La sección específica a este modelo es la última línea de código. Todas las líneas anteriores corresponden al código de inicialización. En él se comprueba que al menos se han especificado los dos primeros argumentos obligatorios, s (tamaños) y p (parámetros), y que ambos son numéricos. El vector de parámetros p debe contener 2 como máximo, ls y bs para el caso del Modelo 0. La expresión 2+NARG sería el máximo número de argumentos a la función, en el caso de que se especificaran todos los parámetros como fijos (NARG), además de los vectores s y p (2+).

Esta codificación permite la reutilización sistemática del código de inicialización en los restantes modelos sin más que alterar el valor inicial de la variable ARGS. Obsérvese que el número de argumentos NARG se calcula en función de ARGS.

Se van consumiendo entonces los parámetros fijos del tercer argumento P, referenciado en el código fuente como el *cell-array* varargin. Esta sintaxis permite a MATLAB invocar esta función con un número variable de argumentos. En la autodocumentación, naturalmente, se usa un nombre más razonable para el usuario, P. Si P estuviera ya vacío o el parámetro fijo en cuestión fuera el vector vacío [], el valor se obtiene del vector de parámetros variables p. Naturalmente, se comprueba que el parámetro es numérico, señalando el error en caso de que no lo sea o de que falte. La posibilidad de indicar un parámetro vacío es necesaria para contemplar la eventualidad de que los parámetros fijos no sean consecutivos, como sucede en el Modelo 2 donde algunas modalidades de transmisión fijan $lp = bp = 0$.

El modelo en sí se expresa en una única línea MATLAB: t = ls + s/bs;. Utilizar la notación vectorial supone un incremento significativo de prestaciones sobre un bucle **for** secuencial.

Los principales méritos de este modelo son su sencillez y eficiencia, la posibilidad de despejar algebraicamente el tamaño correspondiente a un tiempo de transmisión dado (invertir el modelo), y la propiedad natural de servir como referencia con la cual comparar los méritos de otros modelos más desarrollados.

B.3 Modelo 1

Este modelo, presentado en el Apartado 2.3.5, aproxima las mediciones mediante dos tramos lineales, cada uno con un término independiente de latencia (lc, ls) y un ancho de banda (bc, bs). La separación entre tramos viene dada por dos parámetros adicionales ($h1$, $s1$). La autodocumentación se muestra en el Listado B.3.

```
%       t = MTUmodel1 ( s, p [, P]... )
%
% s     array de tamaños a enviar
% p     array de parámetros variables del modelo
% P (opcional) parámetros fijos del modelo, como lista de argumentos sueltos
%       H1      tamaño cabecera 1ª MTU              (header) (bytes)
%       S1      tamaño cabecera 1ª MTU adicional    (special)(bytes)
%       LC      latencia transmisión 0 bytes                 (segundos)
%       BC      ancho de banda para 1ª MTU                   (bytes/segundo)
%       LS      latencia segundo tramo MTU1-2                (segundos)
%       BS      ancho de banda segundo tramo                 (bytes/segundo)
%
% t     array de tiempos de transmisión predichos por el modelo
```

Listado B.3: Autodocumentación en el *script* MATLAB para evaluación del Modelo 1.

El código de inicialización contempla esta vez un total de 6 parámetros. En principio este

modelo debería presentar un único parámetro de cabecera H indicando el tamaño de datos de la primera MTU, para localizar la frontera entre los dos tramos modelados. Por consistencia con los otros modelos desarrollados, se prefirió indicar en su lugar los parámetros $h1$ y $s1$, cuya suma reemplaza al deseado H. El Listado B.4 es el código específico para evaluar el Modelo 1.

```
ARGS={'h1' 's1' 'lc' 'bc' 'ls' 'bs'};
...
h1=h1; s1=s1;
lc=lc; bc=bc;                     % parece que eval no los marca como escalares
ls=ls; bs=bs;

% [stat,msg]=unix('ifconfig eth0');
% MTU=str2num(strtok(msg(findstr('MTU:',msg)+4:end),' '));
  MTU      = 1500;
  MTU1HDR  = MTU-h1-s1;                  % MTU - cabecera: tamaño datos MTUs

    below  = (s<=MTU1HDR);   above=~below;

  t        = zeros(size(s));             % preallocation
  t(below)= lc+s(below)/bc;
  t(above)= lc+ MTU1HDR/bc+ ls+(s(above)-MTU1HDR)/bs;
```

Listado B.4: MTUmodel1.m: Código MATLAB para evaluación del Modelo 1.

La sección de inicialización es idéntica a la mostrada anteriormente, sin más que cambiar la definición de los parámetros, motivo por el cual se ha suprimido del listado. Aparece comentado el código para obtener del propio Sistema Operativo el tamaño de MTU de la interfaz de red, ya que está prefijado a 1500. Se calcula entonces el tamaño de datos de la primera MTU (MTU1HDR), y se discriminan los tamaños de mensaje superiores (s(above)) de los que no lo son (s(below)). Ya que se calcularán vectorizadamente los tiempos correspondientes a cada tramo por separado, es preferible reservar toda la memoria requerida por el vector resultado t anticipadamente (*pre-allocation*), para evitar una posible copia de memoria posterior.

Los mensajes que se pueden enviar en una única MTU, s(below), son modelados con un primer tramo de menor ancho de banda, bc. El término independiente lc refleja el tiempo de transmisión del mensaje vacío. Ya que dicho parámetro incluye el tiempo de transmisión de la primera cabecera, se debe usar como tamaño discriminante MTU1HDR=MTU−h1−s1 en lugar de MTU=1500.

Para tamaños superiores s(above), se añaden una latencia y un término lineal adicionales (ls, $1/bs$) al tiempo de transmisión acumulado por la primera MTU (lc+MTU1HDR/bc). Se debe observar que el nuevo término lineal ($1/bs$) va en función del tamaño sobrante s(above−MTU1HDR).

En teoría, ls debiera reflejar el tiempo de transmisión de la cabecera de la segunda MTU. El de las restantes cabeceras quedaría absorbido por el parámetro bs, manifestándose como una pequeña disminución del mismo, ya que las cabeceras aumentan el tiempo de transmisión, incrementando la pendiente del tramo. En realidad, la verdadera utilidad del parámetro ls radica en permitir un grado de libertad necesario al método simplex utilizado para el ajuste del modelo a las mediciones reales, de manera que la aproximación del segundo tramo no se vea afectada por el punto exacto de finalización del primer tramo.

Representando por $t = m1(s)$ el tiempo predicho por el Modelo 1 para un tamaño de mensaje determinado s, el primer tramo empieza en el punto $(0, m1(0) = lc = T_0)$ y acaba en el punto $(\text{MTU1HDR}, m1(\text{MTU1HDR}) = lc + \text{MTU1HDR}/bc)$. El parámetro ls permite que el segundo tramo no tenga que empezar obligatoriamente en dicho punto $(\text{MTU1HDR}, m1(\text{MTU1HDR}))$, lo cual afectaría a la determinación del parámetro bs.

Los principales méritos de este modelo son su relativa sencillez y eficiencia, y la reducción del error cuadrático frente al del modelo anterior (ver Tabla 2.11).

B.4 Modelo 2

El modelo presentado en el Apartado 2.3.7 persigue reducir aún más el error cuadrático dedicando un tramo lineal de gráfica a cada MTU. Dicho modelado requiere el conocimiento exacto del dominio de cada tramo, y por consiguiente del tamaño exacto de cada MTU. Los problemas asociados a dicha determinación y la solución propuesta, consistente en un modelado mediante cinco parámetros $h1/h2/hn$ y $s1/sn$, se detallan en el Apartado 2.3.6.

Los tramos modelados en esta ocasión son:

$$\text{Modelo 2}\left\{\begin{array}{lrccccl} & lc/bc & ls/bs & \ldots & ls/bs & & \text{1ª UDP, n tramos, } n=\frac{\|UDP\|}{\|MTU\|} \\ [& lc+lm+n.ls/bc & ls/bs & \ldots & ls/bs &] & \text{2ª UDP para daemon LAM} \\ & lm/bc,bs & ls/bs & \ldots & ls/bs & & \text{restantes UDPs, bc para daemon PVM} \end{array}\right.$$

En el primer fragmento UDP, el primer tramo sigue modelándose mediante una latencia lc y un ancho de banda bc. Posteriores tramos acumulan una latencia ls y un segmento lineal de pendiente $1/bs$.

La modalidad *daemon* LAM, fácilmente delatada por el parámetro de cabecera $h1 = 92$, incorpora un segundo fragmento UDP excepcional, que se modela como el primero. Como se podría esperar, el salto entre 1ª y 2ª UDP no coincide con lc; tras algunos intentos se ha encontrado que el valor $lc + lm + n.ls$ es una estimación aceptable de dicho salto.

Salvo esta(s) UDP(s) excepcional(es), el resto de UDPs se modelan con tramos MTU ls/bs, salvo la 1ª MTU que presenta una latencia lm. La modalidad *daemon* PVM, delatada también por el parámetro $h1 == 44$, repite además la elevada pendiente $1/bc$ para cada primera MTU de estas UDPs restantes.

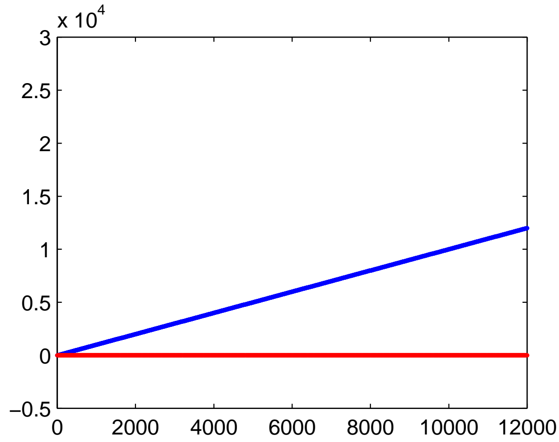
El código MATLAB puede parecer excesivamente extenso a primera vista, por lo que se ofrece fraccionado, intercalando comentarios. Para simplificar la redacción, se ha programado una subrutina de apoyo account() al final del fichero M que implementa el modelo (Listado B.5). Un vistazo a las Figuras B.1 y B.2 puede ayudar a entender más rápidamente el método de acumulación utilizado así como el propósito de esta subrutina de apoyo, consistente en contabilizar fraccionadamente el tiempo total de transmisión correspondiente a cada MTU del mensaje.

Como se observa en el código, los argumentos de account especifican, respectivamente:

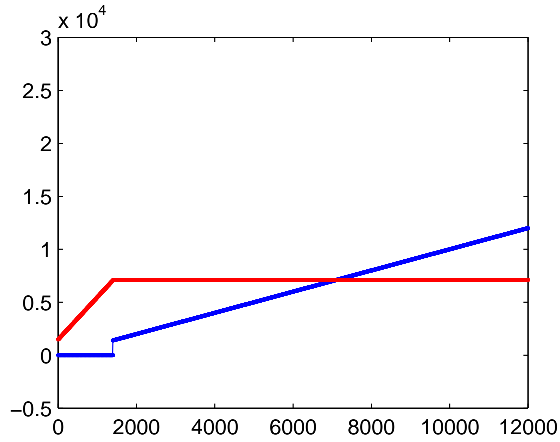
reman las posiciones del vector de tamaños S que aún quedan por contabilizar

```
function [above, accum]= account (reman, accnt, datasz, L, B)
global S t                                  % NO SE CONSIDERA TIEMPO TRANSM. HEADER
     accum =   accnt   +       datasz;      % nuevo tamaño total considerado
     below =   reman   &  (S<=accum);       % de los que quedan, los tamaños <=
     above =              (S> accum);       % los mayores quedan para después
   S(below)=S(below) -         accnt ;      % tamaño parcial por acumular
   t(below)=t(below) + L+S(below)/B;        % añadir tiempo del tamaño parcial
   t(above)=t(above) + L+  datasz/B;        % añadir también a los superiores
   S(below)=       -1  ;                    % marcar como considerado
```

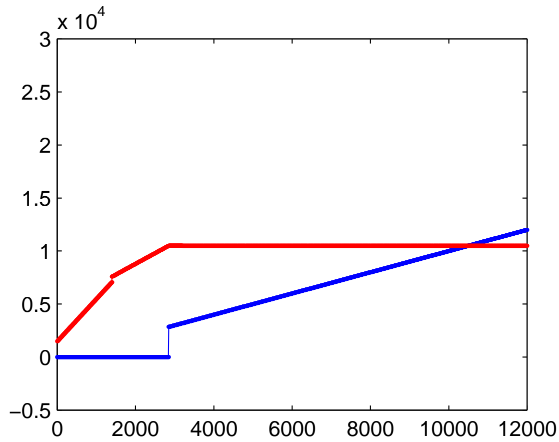
Listado B.5: Función de apoyo account.



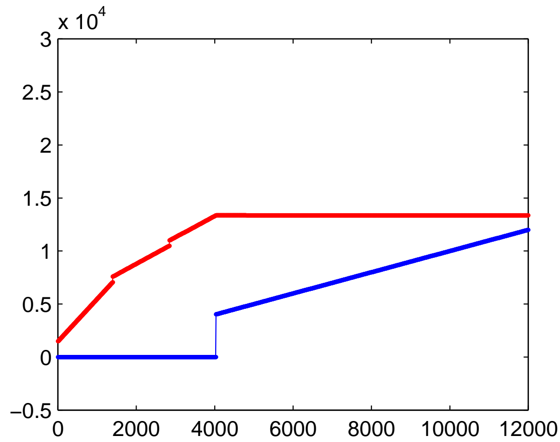
(a) Paso 0: el vector de tamaños original se ha trazado en azul; vector de tiempos a cero, trazado en rojo.



(b) Paso 1: tamaños dentro de 1ª MTU se convierten a tiempos, y se descuentan del vector tamaños. Se utilizan los parámetros lc, bc.

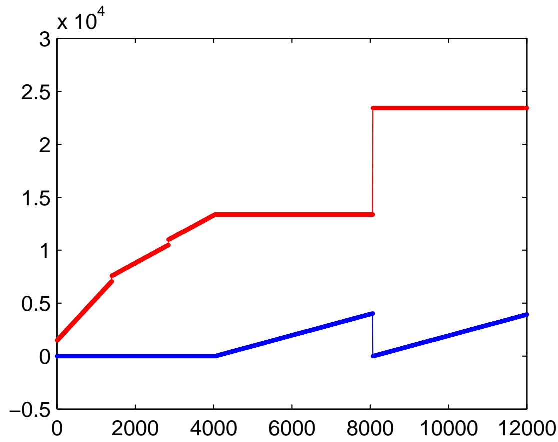


(c) Paso 2: tamaños dentro de 2ª MTU se convierten a tiempos. Se utilizan los parámetros ls, bs.

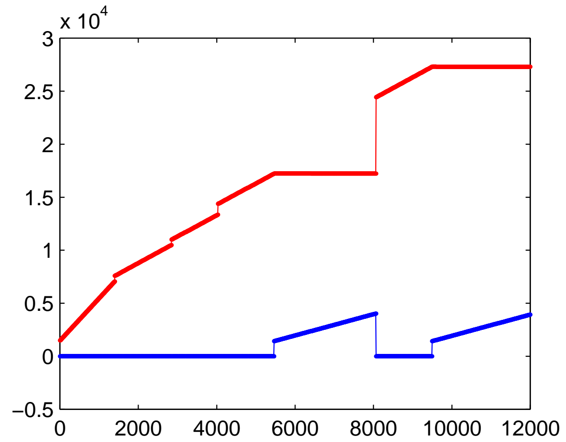


(d) Paso 3: tamaños dentro de 3ª MTU se convierten a tiempos. Se utilizan ls, bs. La 1ª y 3ª MTUs pueden ser irregulares.

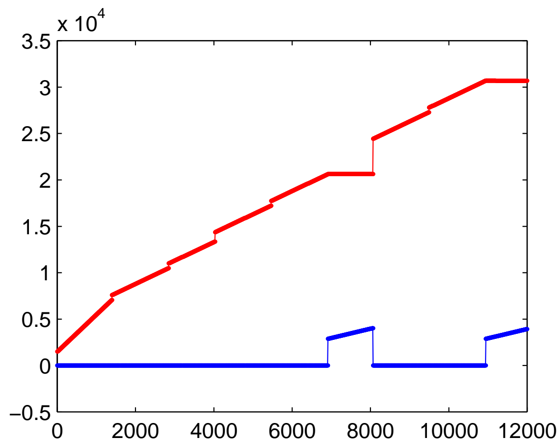
Figura B.1: Pasos del Modelo 2 para contabilizar tiempos de transmisión. Vector tamaños trazado en azul, tiempos en rojo. La irregularidad del primer fragmento UDP obliga a procesarlo previamente. El *daemon* LAM presenta 2 UDPs irregulares.



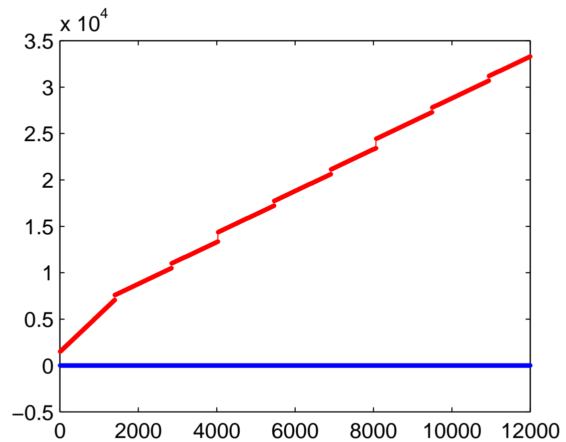
(a) Paso 4: Contabilización de fragmentos UDP completos.



(b) Paso 5: restos de 1ª MTU regular convertidos a tiempos. Ahora se utilizan los parámetros lm, bc/bs.



(c) Paso 6: restos de 2ª MTU regular.



(d) Paso 7: restos de 3ª MTU regular.

Figura B.2: Pasos del Modelo 2 para contabilizar tiempos de transmisión. La regularidad del resto de fragmentos UDP permite procesarlos conjuntamente, en 4 pasos.

accnt el máximo tamaño de mensaje que ya ha sido contabilizado

datasz el tamaño adicional de mensaje que se ha de contabilizar. Se debe observar que este modelo absorbe el tiempo de transmisión de la cabecera en el parámetro de latencia L, ya que el ancho de banda B sólo se aplica al tamaño de datos.

L, B latencia y ancho de banda aplicables a este tramo

La subrutina devuelve las posiciones del vector de tamaños S que quedan aún por contabilizar y el máximo tamaño contabilizado tras su ejecución. Estos datos pueden usarse como primer y segundo argumentos en una posterior llamada a account(). El código comienza incrementando el tamaño contabilizado accnt en datasz unidades. El nuevo tamaño contabilizado se usa para discriminar las posiciones del vector S que deben ser contabilizadas (below) y las que quedarán por calcular (above).

Pudiendo no ser ésta la primera invocación a la subrutina, se ha de restar a las posiciones que van a ser consideradas (S(below)) el tamaño contabilizado desde la primera ejecución (accnt) antes de aplicar al tamaño restante la latencia y ancho de banda para acumularlo al tiempo de transmisión t(below). A los tamaños superiores t(above) se les acumula el tiempo máximo debido a este tramo, pero sin descontarles el tamaño de datos causante de este tiempo de transmisión. Este descuento se realizará en alguna llamada posterior.

Los tamaños considerados se marcan a -1. Al final de la evaluación, el código del Modelo 2 comprueba que todos los tamaños han sido marcados. Dicho código comienza con la habitual autodocumentación que el usuario puede consultar mediante el comando **help** (Listado B.6).

```
function t = MTUmodel2( s, p, varargin )
%
%       t = MTUmodel2 ( s, p [, P]... )
%
% s     array de tamaños a enviar
% p     array de parámetros variables del modelo
% P (opcional) parámetros fijos del modelo, como lista de argumentos sueltos
%       FS      Fragment Size, N MTUs forman 1 fragm.UDP  (bytes)
%       H1      tamaño cabecera 1ª MTU de cada N (header) (bytes)
%       H2      tamaño cabecera 2ª..(N-1)ª                (bytes)
%       HN      tamaño cabecera Nª MTU de cada N          (bytes)
%       S1      tamaño cabecera 1ª MTU en exceso (special)(bytes)
%       SN      tamaño cabecera Nª MTU en defecto         (bytes)
%       LC      latencia de llamada 1ª MTU                (segundos)
%       BC      ancho de banda para 1ª MTU                (bytes/segundo)
%       LM      latencia por cada UDP                     (segundos)
%       LS      latencia por cada MTU                     (segundos)
%       BS      ancho de banda                            (bytes/segundo)
%
% t     array de tiempos de transmisión predichos por el modelo
```

Listado B.6: Autodocumentación en el *script* MATLAB para evaluación del Modelo 2.

El código de inicialización, suprimido del Listado B.7, contempla 11 parámetros: 6 de cabecera, 3 latencias y 2 anchos de banda.

En el caso general, cada MTU presenta un tamaño de datos distinto (MTU?HDR), que se repite con periodicidad $n = \left\lceil \frac{fs}{\text{MTU}} \right\rceil$. Como con cada n MTUs se completa el envío de un fragmento UDP de tamaño fs, se calcula también el tamaño de datos correspondiente (UDP_HDR). Un caso especial es el primer fragmento (UDP1HDR), que suele presentar excepciones en las MTUs 1ª y Nª, modeladas por los parámetros $s1$ y sn respectivamente.

```
ARGS={'fs' 'h1' 'h2' 'hn' 's1' 'sn' 'lc' 'bc' 'lm' 'ls' 'bs'};
...
    MTU      =1500;

    MTU1HDR=MTU-h1;                                      % MTU - cabecera: tamaño datos MTUs
    MTU2HDR=MTU-h2;
    MTUNHDR=MTU-hn;                                      % recordar excepciones especiales s1 y sn
                                                         % caso 4440 (N==3->4) corregir en general
    N=ceil(fs/MTU);                                      % pre-estimación de N (sin cabeceras)
    N=ceil(( fs -48 + h1 + (N>1)*((N-2)*h2 + hn)) / MTU );       % 48 para udp4K/8K
    UDP      = N*MTU;
    UDP_HDR=UDP - (h1 + (N>1)*((N-2)*h2 + hn));          % UDP-cabecera = tam.datos UDP
    UDP1HDR=UDP_HDR-s1+sn;                               % Tamaño datos 1ª UDP

    %------------------------------------------------------------------

    global       S t                                     % no se puede hacer global s para account
    S=           s ;
    t= zeros(size(S));                                   % pre-allocation

    LAMDFLG=(h1==92);                                    % modo LAM daemon (-lamd)
if  LAMDFLG, UDP1HDR = 2*UDP1HDR;                        % para evitar cuentas mal (1)
             lx = lc+lm+N*ls; end

    PVMDFLG=(h1==44);
if  PVMDFLG, bx=bc; else, bx=bs; end                     % periodicidad modo daemon PVM
```

Listado B.7: MTUmodel2.m: Cálculos previos en el Modelo 2.

Para acelerar los cálculos posteriores se utiliza la recomendada técnica de preubicación (*pre-allocation*), reservando memoria anticipadamente para el vector de tiempos de transmisión. El vector de tamaños de mensaje s se copia a una variable global S para poder acceder a él desde la subrutina de apoyo account().

Un último detalle consiste en definir los parámetros auxiliares lx/bx para considerar los casos especiales del *daemon* LAM y PVM, respectivamente. Con estos cálculos previos, el modelo puede evaluarse más fácilmente. El primer fragmento se muestra en el Listado B.8.

```
                              above=1;
                                     MTUacc=0;
for  i=1:1+LAMDFLG, if i>1, lc=lx; end              %%-lamd: lc/bc - lx/bc
[above,MTUacc]=account(above,MTUacc,MTU1HDR-s1, lc,bc); if N>1, for i=1:N-2
[above,MTUacc]=account(above,MTUacc,MTU2HDR   , ls,bs);          end
[above,MTUacc]=account(above,MTUacc,MTUNHDR+sn, ls,bs); end
end
     if MTUacc~=UDP1HDR, error('cuentas␣mal␣(1)'), end
```

Listado B.8: MTUmodel2.m: Código MATLAB para evaluación del Modelo 2.

Se discriminan en tres etapas sucesivas la primera y la última MTUs, ya que presentan las irregularidades $s1$ y sn. En la primera invocación a account(), todo el vector s está por calcular (reman=above=1), el tamaño de mensaje contabilizado es accnt=0, el tamaño de datos es el de la primera MTU (datasz=MTU1HDR−s1), y se le aplica la latencia lc y el ancho de banda bc.

En las siguientes llamadas, se usan como primeros argumentos los devueltos por la llamada anterior, como se sugirió en la explicación de account(). La última MTU también presenta irregularidad de tamaño sn y la latencia y ancho de banda aplicables son ls y bs.

Se recomienda de nuevo consultar la Figura B.1 para entender más rápidamente el método de contabilidad seguido. Tras el bucle de acumulación se realiza la mencionada comprobación acerca del tamaño acumulado. El caso del *daemon* LAM se resuelve redefiniendo UDP1HDR.

```
    UDPs    =         zeros ( size (S));                 % pre-allocation
  S(above)=S(above)    -UDP1HDR ;                        % Quitar excepción para dividir
UDPs(above)= fix (S(above)/UDP_HDR);
  S(above)=rem(S(above), UDP_HDR);                       % Quitar nº entero UDPs
    chng    =    ~S;                                     % múltiplo (no nulo) de UDP
UDPs(chng) = UDPs(chng)-1;                               %  quitar 1 UDP y
               S(chng)= UDP_HDR;                         %  poner UDP como resto

     TUDP =      lm+MTU1HDR/bx  + (N>1)*...
        (       (1s +MTU2HDR/bs)* (N-2)+...
                (1s +MTUNHDR/bs)        );
  t(above)=t(above)+UDPs(above)*TUDP;
```

Listado B.9: MTUmodel2.m: (continuación).

El resto de tamaños requiere un cálculo previo, mostrado en el Listado B.9.

Para realizar más eficientemente el cálculo de los tiempos restantes, una vez contabilizado el primer fragmento UDP irregular (y también el segundo bajo *daemon* LAM), se resta su tamaño de datos UDP1HDR a los tamaños superiores S(above).

El tamaño restante se divide por defecto entre el tamaño de datos regular del fragmento UDP (UDP_HDR, sin las excepciones asociadas al primer fragmento). El tiempo de transmisión asociado a dichas UDPs completas (TUDP) considera la latencia especial lm para la primera MTU. La Figura B.2(a) expresa gráficamente el descuento de tamaño e incremento de tiempo realizado en este paso.

Queda por contabilizar el tiempo de las MTUs posteriores a estos fragmentos PVM completos. El código MATLAB (Listado B.10) es muy similar al empleado para la primera UDP irregular, usando en este caso los tamaños discriminantes MTU?HDR sin excepciones $s1/sn$.

```
[above,MTUacc]=account(above,      0,MTU1HDR, lm,bx);    if N>1, for i=1:N-2
[above,MTUacc]=account(above,MTUacc,MTU2HDR, 1s,bs);              end
[above,MTUacc]=account(above,MTUacc,MTUNHDR, 1s,bs);     end

    if MTUacc~=UDP_HDR, error('cuentas mal (2)'), end
    if any(above),      error('cuentas mal (3)'), end
    if any(S~=-1),      error('cuentas mal (4)'), end
```

Listado B.10: MTUmodel2.m: (continuación).

De nuevo, la Figura B.2 expresa gráficamente las operaciones realizadas. Se restringe el procesamiento al subvector de tamaños que queda por contabilizar (above).

Se debe notar también que la primera MTU utiliza el parámetro auxiliar bx, con lo que el caso del *daemon* PVM queda contemplado. Tras el bucle de acumulación se realiza la comentada comprobación, no debiendo quedar ningún tamaño sin contabilizar.

Se podría argumentar indefinidamente sobre la aparente aparatosidad y artificio del código mostrado. Las principales ventajas que posee son que ya está depurado, que contiene multitud de comprobaciones de corrección, que es *más fácil de entender* que las alternativas concebibles, y sobre todo, es *mucho más breve* de explicar. Las alternativas que suelen acudir a la imaginación intentan contabilizar primero el tamaño regular y contemplar posteriormente las excepciones. El código resultante es más complejo.

Respecto al modelo en sí, presenta las ventajas de admitir aún un método relativamente eficiente para su evaluación, y de reducir aún más el error cuadrático, hasta el punto de que visualmente se aprecia una superposición contínua con las gráficas de las mediciones de tiempo.

B.5 Modelo 3

El modelo presentado en el Apartado 2.3.8 consigue reducir aún más el error cuadrático dedicando un par de parámetros (latencia y ancho de banda) a cada tramo. Recordemos la funcionalidad de los distintos parámetros:

$$\text{Modelo 3}\left\{\begin{array}{cccc} \overbrace{lcbc/lsbs}^{\text{ruta directa}} & \overbrace{lcbc/lpbp/\ldots/lsbs}^{\text{daemon PVM}} & \overbrace{lcbc/lsbm/\ldots}^{\text{daemon LAM}} & 1^{a}\text{ UDP}\\ lmbm/lsbs & lmbm/lpbp/\ldots/lsbs & lpbp/lsbm/\ldots & 2^{a}\text{ UDP}\\ \ldots & \ldots & lmbs/lsbs/\ldots & \text{otras}\end{array}\right.$$

Se ha observado que con ruta TCP directa una periodicidad 2 predice con mayor precisión los tiempos de transmisión, por lo que se ha optado por modelar con lm/bm las MTUs impares (salvo la primera, que continúa con lc/bc), y utilizar la pareja ls/bs para las pares. La ruta directa se detecta por la cabecera $h1 = 52$, tanto bajo PVM como bajo LAM.

El complejo comportamiento de los modos *daemon* tiene fiel reflejo en la secuencia periódica de parejas de parámetros utilizada en su modelado. Las modalidades *daemon* PVM y LAM son fácilmente detectadas por la cabecera $h1 = 44/92$, respectivamente.

Las opciones *PvmDataRaw/Default* no siguen una alternancia de periodicidad 2 con ruta directa, sino que presentan cabeceras propias del mecanismo de fragmentación del *daemon* PVM. Se les aplica por tanto el mismo esquema que a la modalidad *daemon*. Consultando la Tabla 2.14, se observa que este caso queda de nuevo delatado por la cabecera $h1 = 68$.

El código MATLAB para el Modelo 3 (Listado B.11) detecta y comprueba las condiciones mencionadas en los párrafos anteriores. Se calculan los *flags* C2CDI, PVMDFLG y LAMDFLG, que discriminan la citada periodicidad de los tramos de la gráfica.

Para automatizar mediante un bucle **for** la aplicación de parámetros a los distintos tramos del modelo, se recurre a usar un vector índice **I** y enumerar explícitamente los parámetros involucrados en cada tramo.

Así, por ejemplo, la alternancia de parámetros con ruta directa se modela con In=[1 2]. Las MTUs regulares se reflejan en Hn. Los vectores Ln y Bn enumeran las latencias y anchos de banda aplicables, en este caso lm/bm y ls/bs alternadamente.

Los vectores I1/H1/L1/B1 contemplan el caso de la primera UDP, que usa como parámetros lc/bc y puede presentar cabeceras especiales. Se calculan los habituales tamaños de comprobación que se introdujeron con el Modelo 2, UDP_HDR y UDP1HDR.

Esta técnica permite expresar mucho más cómodamente toda la complejidad de las modalidades *daemon* en las breves líneas de código MATLAB mostradas. Tal vez lo que más pueda extrañar de estas modalidades sea el inusual valor para H(**end**) (el último parámetro de cabecera) que probablemente requiera una explicación más detallada.

En modo *daemon*, la última MTU de cada UDP es fragmentaria. El tamaño indicado como cabecera, hn, no es una cabecera real que se transmita, sino el valor que hay que restar al tamaño de MTU=1500 para obtener el tamaño de datos real.

Para superar al Modelo 2 se ha procurado contabilizar el tiempo de transmisión de las cabeceras: 50 bytes, o un tamaño algo superior como 500 bits, transmitidos a 100Mbps tardarían unos 5μs, y el Modelo 2 ya se desenvolvía en ese rango de precisión. No considerar los diferentes tiempos de transmisión de las distintas cabeceras arruinaría el esfuerzo adicional de modelado.

```
ARGS={'fs' 'h1' 'h2' 'hn' 's1' 'sn' 'lc' 'bc' 'lm' 'bm' 'lp' 'bp' 'ls' 'bs'};
...
global MTU
   MTU      = 1500;
   N=ceil(fs/MTU);                                  % estimación de n (sin cabeceras)
   N=ceil( ( fs -48 + h1 + (N>1)*((N-2)*h2 + hn) ) / MTU );      % 48 para udp4K/8K

   C2CDI   =(h1==52);                               % DtInPlace RtDirect/Client2C
   PVMDFLG=(h1==44);                                % modo PVM daemon (DontRoute)
   LAMDFLG=(h1==92);                                % modo LAM daemon (-lamd)

switch h1
case 52                  %%%C2CDI%%%                % h1=h2=hn==52 s1=24/0 sn=0
   N=2;
        Ln=[lm    ls];    Bn=[bm    bs];            % repetir tramos lmbm/lsbs
        L1=[lc    ls];    B1=[bc    bs];            % salvo 1ª ronda lcbc/lsbs
        In=[ 1     2];    Hn=[h1    h1];            % índices y cabeceras
        I1=In;          H1=[s1+h1   h1-sn];         % h1==h2==hn, s1==24/0, sn==0

        MTU_HDR =    MTU-h1;                        % tamaño datos MTUs
        UDP_HDR = 2*MTU_HDR;                        % tamaño datos UDPs (se usa...)
        UDP1HDR =UDP_HDR-s1+sn;                     % tam datos 1ª UDP (.. en tests)

case {44,68}             %%%PVMDFLG%%%              % modo PVM daemon (DontRoute)
        Ln=lp*ones(1,N);  Ln([1,end])=[lm ls];      % repetir  lmbm/lpbp/.../lsbs
        Bn=bp*ones(1,N);  Bn([1,end])=[bm bs];
        L1=Ln;            L1(1)=lc;                 % salvo 1ª lcbc/lpbp/.../lsbs
        B1=Bn;            B1(1)=bc;
        In=1:N;                                     % índices y cabeceras
        I1=In;                                      % h2 sustituto de hn
        Hn=h2*ones(1,N);  Hn([1,end])=[h1    (hn    +MTU* h2    )];
        H1=Hn;  if  h2>=sn,H1([1,end])=[h1+s1 (hn-sn+MTU*(h2-sn))];
                    else ,H1([1,end])=[h1+s1 (hn-sn+MTU*(h1-sn))];  end
                                                    % h2-sn negativo para raw/def-no
        MTUn     = MTU-hn+h2;                       % tam. nª MTU frgmntaria UDP4K/8K
        UDP      = (N-1)*MTU+MTUn;                  % tam.UDP con cbceras (no se usa)
        UDP_HDR = N*MTU - (h1 + (N>1)*( (N-2)*h2 + hn ));       % tam. datos UDP
        UDP1HDR = UDP_HDR-s1+sn;                    % t. datos 1ª UDP (se usa->tests)

case 92                  %%%LAMDFLG%%%              % modo LAM daemon (-lamd)
        Ln=ls*ones(1,N);  Ln(1)=lm;                 % repetir  lmbs/lsbs/.../lsbs
        Bn=bs*ones(1,N);
        L1=ls*ones(1,N);  L1(1)=lc;                 % salvo 1ª lcbc/lsbm/.../lsbm
        B1=bm*ones(1,N);  B1(1)=bc;
        In=1:N;                                     % índices y cabeceras
        I1=[In In+N];                               % h2 sustituto de hn
        Hn=h2*ones(1,N);  Hn([1,end])=[h1    (hn    +MTU* h2    )];
        H1=Hn;  if  h2>=sn,H1([1,end])=[h1+s1 (hn-sn+MTU*(h2-sn))];
                    else ,H1([1,end])=[h1+s1 (hn-sn+MTU*(h1-sn))];  end

        H1=[H1 H1];                                 % ojo!! 2ª lpbp/lsbm/.../lsbm
        L1=[L1 L1]; L1(end/2+1)=lp;
        B1=[B1 B1]; B1(end/2+1)=bp;

        MTUn     = MTU-hn+h2;                       % idéntico a PVMDFLG
        UDP      = (N-1)*MTU+MTUn;
        UDP_HDR = N*MTU - (h1 + (N>1)*( (N-2)*h2 + hn ));
        UDP1HDR = UDP_HDR-s1+sn;
        UDP1HDR=2*UDP1HDR;                          % para evitar cuentas mal (1)

otherwise
        error('configuración desconocida, h1 extraño')
end
```

Listado B.11: MTUmodel3.m: Cálculos previos en el Modelo 3.

Para MTUs intermedias, la suma del tamaño de datos y la cabecera es obviamente el tamaño total MTU=1500, pero la tercera MTU fragmentaria presenta el citado problema.

Ya que a partir de la segunda MTU todas presentan la misma cabecera, se podría suponer que la última MTU presenta la misma cabecera $h2$ en realidad. Esta suposición presenta un par de fallos, concretamente la modalidad *daemon* con los empaquetamientos *PvmDataRaw/Default*, en donde $h2 < sn$ impidiendo aplicar la excepción a la última MTU de la 1ª UDP. Para estos dos únicos casos se ha preferido imaginar que $hn = h1$ en lugar del más razonable $hn = h2$.

Estamos por tanto diferenciando la cabecera habitual (que se resta a la MTU) de la cabecera *real* que se transmite. Esta distinción sólo afecta a la última MTU de cada UDP en modo *daemon*. Al objeto de alterar lo menos posible el código para la función auxiliar account() (mostrado más adelante), la cabecera *real* se pasa junto con la habitual, multiplicada por un valor que permita su separación. Esto explica los extraños valores de H1/Hn en el Listado B.11.

En el Listado B.12 se utilizan estos vectores I/H/L/B en un bucle que selecciona los parámetros aplicables en cada ocasión, simplificando notabilísimamente la redacción del código MATLAB.

```
  global          S t                                    % no se puede hacer global s
  S=              s   ;
  t= zeros(size(S));                                     % pre-allocation

                             above=1; MTUacc=0;
for  i=I1, [above,MTUacc]=account(above,  MTUacc,  H1(i), L1(i),B1(i));  end

  if MTUacc~=UDP1HDR,      error('cuentas mal (1)'),  end
```

Listado B.12: MTUmodel3.m: (continuación).

Igual que con el modelo anterior, para una mayor eficiencia, una vez eliminadas las irregularidades de la 1ª (y tal vez 2ª) UDP, se contabilizan agrupadamente el mayor número posible de UDPs utilizando el mismo tipo de cálculos previos que ya presentamos en el modelo anterior.

```
    UDPs   =          zeros(size(S));                    % pre-allocation
  S(above)=S(above)      -UDP1HDR ;                      % Quitar excepción para dividir
UDPs(above)= fix (S(above)/UDP_HDR);
  S(above)=rem(S(above),UDP_HDR);                        % Quitar nº entero UDPs
    chng   =    ~S;                                      % múltiplo (no nulo) de UDP
UDPs(chng) =  UDPs(chng)-1;                              %  quitar 1 UDP y
                   S(chng)= UDP_HDR;                     %  poner UDP como resto

      TUDP =  sum(Ln+MTU./Bn);
      TUDP =  TUDP-(hn-h2)/Bn(end);                      % corregir cbcera nª MTU fragm.

  t(above)=t(above)+UDPs(above)*TUDP;

                                   MTUacc=0;
for  i=In, [above,MTUacc]=account(above,  MTUacc,  Hn(i), Ln(i),Bn(i));  end

  if MTUacc~=UDP_HDR,      error('cuentas mal (2)'),  end
  if any(above),           error('cuentas mal (3)'),  end
  if any(S~=-1),           error('cuentas mal (4)'),  end
```

Listado B.13: MTUmodel3.m: (continuación).

Agrupar los parámetros en vectores ha permitido expresar compactamente el tiempo de transmisión de una UDP usando notación vectorial y el comando **sum**(), que acumula los elementos de un vector. También ha simplificado el bucle de acumulación final en el Listado B.13.

El código de la subrutina de apoyo account (Listado B.14) es una pequeña variación del mostrado previamente bajo el Modelo 2. Como se ha comentado, el parámetro de cabecera transporta tanto el parámetro habitual como la cabecera *real*, multiplicada ésta por MTU para garantizar su recuperación independientemente de los tamaños de cabeceras. Si se detecta la presencia de esta cabecera *real*, se tiene en cuenta al modelar el tiempo de transmisión.

```
function [above,accum]=account(reman,accnt,hdrsz,L,B)
global MTU S t                                   % SÍ SE CONSIDERA TIEMPO TRANSM. HEADER
     if hdrsz>MTU                                % truco para pasar cabecera y sustituto
        subst=fix(hdrsz/MTU);                    % truco sustitución se pasa*MTU
        hdrsz=rem(hdrsz,MTU);                    % h1/h2/hn/s1/sn son todas <MTU
     end
     datasz = MTU    -     hdrsz;                % el equivalente a MTU?HDR de antes
     accum =  accnt  +     datasz;               % nuevo tam.total de datos considerado
     below =  reman  & (S<=accum);               % de los que quedan, los tamaños <=
     above =           (S> accum);               % los mayores quedan para después
   S(below)=S(below) -     accnt ;               % tamaño parcial por acumular
     if hdrsz>300, hdrsz=subst; end              % última MTU fragmentaria
   t(below)=t(below) + L+(S(below)+hdrsz)/B;
   t(above)=t(above) + L+(  datasz+hdrsz)/B;     % datasz+hdrsz==MTU (ó MTUn)
   S(below)=       -1  ;
```

Listado B.14: MTUmodel3.m: (continuación).

Este modelo es el más preciso que se ha podido desarrollar. El error cuadrático restante sólo es achacable a la propia variabilidad de las mediciones y a la no linealidad de los tramos. En efecto, sobre todo en las modalidades *daemon* se observa bajo el GUI que los tramos correspondientes a cada MTU no son segmentos rectilíneos sino que muestran una pequeña concavidad.

Sin embargo, no es factible añadir más parámetros para intentar modelar esta separación de la linealidad. El método simplex encuentra grandes dificultades para converger a un valor estable de parámetros sobre una superficie de error 8-dimensional (4 parejas de parámetros variables L, B). Con tantos grados de libertad, una desviación en un parámetro tiene muchas posibilidades de ser corregida mediante otra pequeña alteración de algunos de los parámetros restantes. La probabilidad de que existan trayectorias sobre el espacio de parámetros para las cuales el error sea constante y mínimo, crece conforme aumenta la dimensionalidad del espacio. También aumenta la probabilidad de quedar atrapado en un mínimo local de la superficie de error.

En cualquier caso, el número de iteraciones en que converge el método simplex aumenta tanto que, incluso automatizando el proceso, hace cuestionable la viabilidad de un modelo con mayor número de parámetros.

Apéndice C

Programación MEX de PVMTB

C.1 Introducción

En el Apartado 3.3 se enumeraron los distintos patrones bajo los que se han clasificado las llamadas PVM:

Patrones generales: P–1, . . . , P–9

Patrón de máscaras: MSK

Patrón colectivo: COL

Patrón de registro: REG

Patrón simple: SMP

Funciones: FUN, PCK, NRM

La Tabla 3.1 relacionaba cada llamada PVM con su correspondiente patrón, siguiendo esta nomenclatura. También se explicaban los patrones generales, dado que su sencillez y brevedad no entorpecían la línea discursiva del capítulo, toda vez que permitían obtener una idea detallada del método seguido para programar la *Toolbox*.

Este apéndice describe brevemente el resto de patrones, las extensiones añadidas a la *Toolbox* y el método de compilación de la misma, el cual ha sido automatizado mediante el uso de la utilidad make dirigida por un fichero Makefile.

C.2 Funciones de gestión de máscaras (MSK)

Las funciones y macros de máscara de trazado admiten un parámetro de máscara (definido en lenguaje C como char[TEV_MASK_LENGTH]) y otro entero (**who**/kind). Se ha preferido tratar este segundo parámetro como *string*. Las funciones de apoyo se han agrupado en header_msk.h. Las más significativas son **who**() y kind().

```
/* header_msk.h
 * Devuelve/Ajusta/Manipula máscara de traza de una tarea o sus hijas
```

```
 *
 * Patrones de llamada MATLAB
 * [ info mask]= pvm_gettmask (who)
 *   info      =pvm_settmask(who, mask)
 *               TEV_MASK_INIT ( mask)
 *               TEV_MASK_SET   (mask, kind )
 *               TEV_MASK_UNSET(mask, kind )
 *   info      =   TEV_MASK_CHECK(mask, kind )
 */
...
#define argchkmask(argn)      /* argn podría almacenar una máscara */ \
  if (mxGetNumberOfElements(prhs[argn]) < TEV_MASK_LENGTH){           \
        mexPrintf("se requiere   arg#%d tipo mask "                    \
                  "(string[%d])", argn+1, TEV_MASK_LENGTH);            \
        mexErrMsgTxt("");                                     }

#define argchk1str            /* comprueba 1 arg string */            \
  if ((nrhs!=1) || (!mxIsChar(prhs[0])))                              \
        mexErrMsgTxt("se requiere 1 arg  string");

#define argchk2str            /* comprueba 2 arg string */            \
  if ((nrhs!=2) || (!mxIsChar(prhs[0]))                               \
                || (!mxIsChar(prhs[1])))                              \
        mexErrMsgTxt("se requieren 2 args string");

#define argout          /* deja creado plhs listo para asignar */     \
  *mxGetPr(plhs[0]=mxCreateDoubleMatrix(1,1,mxREAL)) =
/*
 * Funciones de apoyo: traducción de 'who', 'mask', 'kind'
 */
int who(const mxArray*arr){
        int  ret;
        char*str=mxArrayToString(arr);

        if (!strcmp(str,"Self" ))         ret=PvmTaskSelf ;
   else if (!strcmp(str,"Child"))         ret=PvmTaskChild;
   else { mexWarnMsgTxt("'who' debe ser 'Self' ó 'Child'");
                                          ret=PvmBadParam ;}
        mxFree(str); return(ret);
}
```

La función **who**() evita al usuario tener que recordar los valores concretos de los símbolos PVM asociados (PvmTaskSelf, PvmTaskChild). Si se comete error se devuelve PvmBadParam, lo cual es consistente con el funcionamiento de las llamadas PVM originales para manejo de máscaras.

Las restantes macros (argchkmsk, ...) sirven como bloques constructivos para este grupo de llamadas PVM. El fichero header_msk.h continúa con:

```
#define knd(NAME) \
else if (!strcmp(str, #NAME ))           ret=TEV_##NAME;

int kind(const mxArray*arr){
        int  ret;
        char*str=mxArrayToString(arr);
```

```
     if  (!strcmp(str,"FIRST"))          ret=TEV_FIRST;
else if  (!strcmp(str,"ADDHOSTS"))       ret=TEV_ADDHOSTS;
knd(BARRIER) knd(BCAST)    knd(BUFINFO)     knd(CONFIG) knd(DELETE)
...
knd(TIMING)  knd(PROFILING) knd(USER_DEFINED) knd(MAX)

else {
        mexWarnMsgTxt("'kind' debe estar entre 'FIRST' y 'MAX'");
        ret=PvmBadParam ;
     }
        mxFree(str); return(ret);
}

/*-----------------------------------------------------------------------*/
void mexFunction(int nlhs,mxArray*plhs[], int nrhs,const mxArray*prhs[])
/*-----------------------------------------------------------------------*/
```

Se deja redactado el prototipo de función MEX al final del #include por los mismos motivos que con los patrones generales.

La función kind() traduce el identificador escrito por el usuario a un código de máscara PVM (por ejemplo, ADDHOSTS→TEV_ADDHOSTS). Esto evita al usuario tener que recordar los valores concretos de los códigos. kind() implica demasiados símbolos PVM como para ser redactada explícitamente. Por ello se recurre a la macro auxiliar knd(). De nuevo, se devuelve PvmBadParam si se comete error.

Un ejemplo del uso de **who** es pvm_settmask:

```
/* info = pvm_settmask(who, mask) --- Fija máscara de tarea o hijas */
#include "header_msk.h"
{
  argchk2str
  argchkmask(1)
  argout
  pvm_settmask(who(prhs[0]), (char*)mxGetPr(prhs[1]));
}
```

mientras que un ejemplo del uso de kind() podría ser TEV_MASK_SET:

```
/* TEV_MASK_SET(mask, kind) --- Activa bits en string máscara */
#include "header_msk.h"
{
  argchk2str
  argchkmask(0)
  TEV_MASK_SET(((char*)mxGetPr(prhs[0])), kind(prhs[1]));
}
```

C.3 Funciones colectivas (COL)

Las funciones de grupo colectivas comparten la comprobación de parámetros y la traducción de tipos, así como el tratamiento de los argumentos msgtag, group y rootginst, para lo cual ha sido redactado header_col.h.

```
/* header_col.h
 * Funciones de utilidad para operaciones PVM colectivas
 */
int GetClass(const mxArray*arr){
  switch (mxGetClassID(arr)){
    case mxCHAR_CLASS:   return(PVM_SHORT);
...
    otherwise: mexErrMsgTxt("arg#2: tipo incompatible PVM");
  }
}
/*
 * Patrón de llamada Entrada MATLAB->PVM Salida PVM->MATLAB
 * int int[] = f ((var|'str'), var, int, 'str', int)
 *       pvm_scatter( var , var...
 *       pvm_gather ( var , var...
 *       pvm_reduce ('str', var...
 */
#define INT_F_COLLECTIVE                                              \
        void (*func)(),*result, *data;                                \
        int      count, datatype, msgtag, rootg, info;                \
        char    *op, *group;                                          \
  const mxArray*arr;                                                  \
       *mxGetPr(plhs[0]= mxCreateDoubleMatrix(1,1,mxREAL))=PvmBadParam;\
                                                                      \
        if (nrhs!=5)                                                  \
                mexErrMsgTxt("se requieren 5 args");                  \
        if (     (!mxIsNumeric(prhs[2])) ||                           \
                 (!mxIsNumeric(prhs[4])) )                            \
                mexErrMsgTxt("args#3,5 deben ser enteros");           \
        msgtag = mxGetScalar (prhs[2]);                               \
        rootg  = mxGetScalar (prhs[4]);                               \
                                                                      \
        if (     (!mxIsChar(prhs[3])) )                               \
                mexErrMsgTxt("arg#4 debe ser string");                \
        group  = mxArrayToString(prhs[3]);       /* -> mxFree */
```

En principio se devuelve PvmBadParam por si fallara alguna de las comprobaciones. El resto de código es específico a cada llamada. Un ejemplo de uso de GetClass es:

```
/* info = pvm_scatter(porcion, var, msgtag, 'group', inst)
 * Reparte una variable Matlab entre varias instancias */
#include "header_col.h"
/*-------------------------------------------------------------------*/
void mexFunction(int nlhs,mxArray*plhs[], int nrhs,const mxArray*prhs[])
/*-------------------------------------------------------------------*/
{       INT_F_COLLECTIVE                /* Toma msgtag, group, rootg */
        arr = prhs[0];
        if (!*mxGetName(arr)){
                mxFree(group);
                mexErrMsgTxt("arg#1 debe ser una variable");}
        result = (void*) mxGetPr        (arr);
        count = mxGetNumberOfElements(arr);
        datatype = GetClass             (arr);
...
        info=pvm_scatter(result, data,  count, datatype,
```

```
                                msgtag, group, rootg);
    ...
            mxFree(group);
           *mxGetPr(plhs[0]) = info;
    }
```

C.4 Funciones de registro (REG)

Las tres funciones de registro usan pvmproto.h.

```
/* header_reg.h
 * Las llamadas de registro necesitan protocolo <pvmproto.h>
 */
#include <mex.h>        /* Matlab */
#include <pvm3.h>       /* pvm_* */
#include <pvmproto.h>   /* SM_* */

#define INT_F_VOID(NAME)                                    \
        *mxGetPr(plhs[0]=mxCreateDoubleMatrix(1,1,mxREAL)) = NAME();

/*-----------------------------------------------------------------------*/
void mexFunction(int nlhs,mxArray*plhs[], int nrhs,const mxArray*prhs[])
/*-----------------------------------------------------------------------*/
```

Las funciones pvm_reg_hoster y pvm_reg_tasker siguen el patrón mostrado:

```
/* info = pvm_reg_hoster ---
 * Registra la tarea que llama como responsable de añadir hosts PVM
 */
#include "header_reg.h"
{INT_F_VOID(pvm_reg_hoster)}
```

La función pvm_reg_rm no sigue patrón, ya que devuelve struct pvmhostinfo.

C.5 Funciones simples (SMP)

El resto de llamadas sólo tienen en común los #include MATLAB y PVM, y el prototipo **mexFunction.** Esto se redacta una sola vez en header_smp.h y se incluye en cada fichero fuente.

```
/* header_smp.h
 * Llamadas que no siguen un patron de llamada fijo
 *
 * [info bfinfo] = pvm_bufinfo(bufid)             * [bytes,msgtag,tid]
 * info = pvm_catchout [ (fildes) ]               * stream -> fildes
 * [nhost narch hostinfo] = pvm_config            * struct pvmhostinfo
 * [info mbinfo] = pvm_getmboxinfo('pattern')     * struct pvmmboxinfo
 * [info msginfo] = pvm_getminfo(bufid)           * struct pvmminfo
 * [info clk dlt] = pvm_hostsync(hTID)            * struct timeval
 * info = pvm_mcast(tids, msgtag)                 * tids: int/array
 * info = pvm_notify(what, msgtag, (cnt | tids))* 3er arg según 1º
 * [info len] = pvm_psend(tid, msgtag, data)
 * [info bfinfo] = pvm_precv(tid, msgtag, 'vnam', len) * [b,tag,tid]
```

```
 * bufid = pvm_trecv(tid, msgtag, tmout)          * tmout : real
 * [ntask tinfo] = pvm_tasks(where)               * struct pvmtaskinfo
 * info = pvm_setminfo(bufid, msginfo)            * struct pvmminfo
 * info = pvm_start_pvmd({'arg' ...}, block)
 * [numt tids] = pvm_spawn('task', {'arg' ...}, flag, 'where', ntask)
 * [info resp]= pvm_tickle(args)
 * info = pvm_pkint(n)             * [info  n   ] = pvm_upkint
 * info = pvm_pkdouble(d)          * [info  d   ] = pvm_upkdouble
 * info = pvm_pkstr('str')         * [info 'str'] = pvm_upkstr
 * 'version' = pvm_version
 */
#include <mex.h>        /* Matlab */
#include <pvm3.h>       /* pvm_* */
/*---------------------------------------------------------------------*/
void mexFunction(int nlhs, mxArray*plhs[], int nrhs, const mxArray*prhs[])
/*---------------------------------------------------------------------*/
```

Un ejemplo de su uso es:

```
/* 'version' = pvm_version --- Versión de PVM */
#include "header_smp.h"
{ plhs[0] = mxCreateString(pvm_version()); }
```

Otro ejemplo, mostrando el tratamiento dado a las estructuras MATLAB, es:

```
/* [nhost narch hostinfo] = pvm_config     * struct pvmhostinfo:
 * Configuración Parallel Virtual Machine * [tid, name, arch, speed]
 */
#include "header_smp.h"
{
      int       nhost, narch, i;
      struct    pvmhostinfo *hostp;
      mxArray*arr;
const char    *FNAME[]={"tid","name","arch","speed"};
#define NFIELDS 4

  plhs[0] =   mxCreateDoubleMatrix(1,1,mxREAL);     /* argout 1: error? */
  if (    ((*mxGetPr(plhs[0])) = pvm_config(&nhost,&narch,&hostp)
          ) == PvmOk ){
           (*mxGetPr(plhs[0])) = nhost;             /* argout 1: # hosts */

    plhs[1]= mxCreateDoubleMatrix(1,1,mxREAL);      /* argout 2: # archs */
           (*mxGetPr(plhs[1])) = narch;
                                                     /* argout 3:  hinfo */
    plhs[2]= mxCreateStructMatrix(nhost,1,NFIELDS,FNAME);

    for (i=0; i<nhost; i++){
     *mxGetPr(arr=mxCreateDoubleMatrix(1,1,mxREAL))= hostp[i].hi_tid;
      mxSetField(plhs[2],i,FNAME[0] , arr);
      mxSetField(plhs[2],i,FNAME[1] , mxCreateString(hostp[i].hi_name));
      mxSetField(plhs[2],i,FNAME[2] , mxCreateString(hostp[i].hi_arch));
     *mxGetPr(arr=mxCreateDoubleMatrix(1,1,mxREAL))= hostp[i].hi_speed;
      mxSetField(plhs[2],i,FNAME[3] , arr);
    }
  }
}
```

```
#undef NFIELDS
```

Si PVM no produce error, se devuelven los dos primeros argumentos de retorno en la forma usual. El tercer argumento de retorno es una estructura, que requiere un tratamiento más elaborado.

Los nombres de campos se definen en un array (FNAME). Se crea un *struct-array* MATLAB para contener el número de structs C devueltas por PVM, mediante la llamada API mxCreateStructMatrix. Cada elemento de este array es una estructura con los campos mencionados en FNAME. Los elementos se recorren mediante un bucle **for**.

Los campos de cada elemento se asignan usando la llamada API mxSetField. **mxCreateString** devuelve un *string* MATLAB que se puede usar directamente como argumento de mxSetField. Para los campos int debe crearse primero un array con mxCreateDoubleMatrix y asignarse su valor con **mxGetPr**. Entonces puede asignarse dicho array como valor del campo en el elemento del *struct-array*.

C.6 Funciones (FUN, PCK, NRM)

Tampoco siguen patrón. No comparten #includes. Se han clasificado en categorías para facilitar su comprensión.

```
# Fuentes a mano, sin sistema
FUNSRCS = pvm_mhf.c      pvm_recvf.c
PCKSRCS = pvm_pack.c     pvm_unpack.c
NRMSRCS = putenv.c       select.c       unsetenv.c
```

Las funciones "normales" (NRM) son funciones de apoyo que no utilizan PVM. Aunque MATLAB dispone de una función *getenv* para consultar el valor de variables de entorno, no permite crearlas, modificarlas o eliminarlas. Tampoco permite comprobar si hay caracteres pendientes de lectura, para lo cual se añade la función de apoyo select, por si el usuario deseara utilizarla como indica la página de manual para pvm_getfds. Se muestra como ejemplo el fichero-MEX fuente para unsetenv.

```
/* unsetenv('var') --- Borra variable de entorno */
#include <stdlib.h>
#include <mex.h>
/*------------------------------------------------------------------*/
void mexFunction(int nlhs,mxArray*plhs[], int nrhs,const mxArray*prhs[])
/*------------------------------------------------------------------*/
{       char *str;
        if ((nrhs!=1) || (!mxIsChar(prhs[0])))          /* argchk */
                mexErrMsgTxt("se requiere 1 arg string");
        unsetenv(str=mxArrayToString(prhs[0]));
        mxFree(str);
}
```

Las funciones de empaquetamiento (PCK) son la clave de la integración PVM–MATLAB. Cada variable es empaquetada con una cabecera indicando su clase, nombre y dimensiones, de manera que a la recepción se pueda reconstruir una variable idéntica a la original.

```
/* info = pvm_pack(var [, var]...) --- Empaqueta variables Matlab */
...
int pack_class(const mxArray *arr){
  mxClassID class;
  const char*vname;
        int  ndim, info, lname;
  const int *dims;

 class = mxGetClassID(arr);  if (info=pvm_packf("%d", class)) return (info);
 vname = mxGetName   (arr);  lname = strlen(vname)+1;
         if (info=pvm_packf("%d %*c", lname, lname, vname)) return (info);
 ndim  = mxGetNumberOfDimensions(arr); dims = mxGetDimensions(arr);
         if (info=pvm_packf("%d", ndim))                  return (info);
         if (info=pvm_packf("%*d", ndim, dims))           return (info);

 switch (class) {
   case mxCELL_CLASS:     return(pack_cell  (arr));
   case mxSTRUCT_CLASS:   return(pack_struct(arr));
   ...
   default:        mexWarnMsgTxt("Empaquetamiento no implementado: ");
                   mexWarnMsgTxt(mxGetClassName(arr)); return(PvmNotImpl);
} }
...
/*-------------------------------------------------------------------------*/
void mexFunction(int nlhs,mxArray*plhs[], int nrhs,const mxArray*prhs[])
/*-------------------------------------------------------------------------*/
{ int  i, bufid;
  struct pvmminfo mi;

  plhs[0] = mxCreateDoubleMatrix(1,nrhs,mxREAL);          /* argout */
  if (!(bufid=pvm_getsbuf()))  mexErrMsgTxt("no current send buffer");
  if (pvm_getminfo(bufid,&mi)) mexErrMsgTxt("cannot pvm_getminfo!");
  if (mi.enc==0x20000000)                                 /* InPlace */
    for (i=0; i<nrhs; i++) mxGetPr(plhs[0])[i] = flat_class(prhs[i]);
  else                                                    /* XDR/Raw */
    for (i=0; i<nrhs; i++) mxGetPr(plhs[0])[i] = pack_class(prhs[i]);
}
```

En el fichero-MEX fuente anterior, se obtiene el *buffer* de envío mediante pvm_getsbuf. Se comprueba entonces su modo de empaquetamiento usando pvm_getminfo. Si al codificación no es PvmDataInPlace, se empaquetan la clase, nombre y dimensiones de la variable, y se realiza un descenso jerárquico a lo largo de los componentes del tipo de datos de la variable, hasta empaquetar los elementos básicos de que consta.

Este descenso se realiza mediante diversas rutinas de apoyo que han sido redactadas para cada tipo de datos MATLAB (pack_cell, pack_struct, ...) las cuales pueden volver a llamar recursivamente a pack_class si, por ejemplo, un elemento de una *cell* fuese una *struct* o viceversa.

Todos los tipos de datos MATLAB pueden ser empaquetados bajo los modos PvmDataDefault y PvmDataRaw. La primera opción traduce los datos a representación XDR (eXternal Data Representation); la segunda opción prescinde de dicha traducción, por lo cual sólo debe usarse cuando los computadores fuente y destino del mensaje usan la misma representación de datos.

Por complitud se programaron también las interfaces para las llamadas de empaquetamiento pvm_[u]pkint, pvm_[u]pkdouble y pvm_[u]pkstr, al ser *int*, *double* y *string* tipos básicos MATLAB. También

por complitud se implementaron pvm_psend y pvm_precv, limitadas al tipo *array double*.

Para la opción PvmDataInPlace se debe tener en cuenta que los tipos MATLAB son opacos: el programador de aplicaciones (incluyendo por supuesto al programador de la *Toolbox* PVMTB) no dispone de información sobre su disposición en memoria, tan sólo dispone de rutinas API para consultar y modificar los valores. Aún así, los arrays no complejos (sólo parte real) de tipos de datos no estructurados (lo cual excluye *cell*, *struct*, *object*, *sparse* y *opaque*) son almacenados consecutivamente en memoria, y para ellos se ha implementado la posibilidad de empaquetarlos con la opción PvmDataInPlace. Esta opción ahorra la copia a memoria del *daemon* durante el empaquetamiento: los datos sólo se leen en el momento de transmitirlos con pvm_send; consecuentemente, el usuario no debe modificar los datos hasta que hayan sido recibidos en el computador destino del mensaje.

Las funciones de funciones (FUN) presentan bajo C un argumento *puntero a función*. Bajo MATLAB, la única forma de referenciar código es textualmente, como *string* a ejecutar mediante **eval** desde fichero-M, o mediante **mexCallMATLAB** desde fichero-MEX. Se requiere pues interponer una función C, puntero a la cual se pasará a la llamada PVM, que evalúe el texto MATLAB indicado por el usuario.

```
/* pvm_recvf [ ('MatlabCMD') ] ---
 * Redefine función de comparación usada para aceptar mensajes
 * MatlabCMD debe ser string evaluable por Matlab (mexCallMATLAB)
 *   aceptando 3 enteros (double) y devolviendo entero (double)
 */
...
struct recvfs{
        char *MatlabCMD;                    /* callback MATLAB deseado */
        int (*oldf)(int bufid,int tid,int tag);     /* antigua recvf */
        int (*newf)(int bufid,int tid,int tag);     /* nueva   recvf */
};
int wrapper(int bufid, int tid, int tag);                /* forward */
struct recvfs recvfun = {NULL,NULL, wrapper};            /* static? */

/*---------------------------------------------------------------------*/
int wrapper(int bufid, int tid, int tag){ /* wrapper llamado desde PVM*/
/*---------------------------------------------------------------------*//*tipo I/O args obligatorio */
  int ret, err;
  mxArray     *in[3], *out;
      *mxGetPr(in[0]= mxCreateDoubleMatrix(1,1,mxREAL)) = bufid;
      *mxGetPr(in[1]= mxCreateDoubleMatrix(1,1,mxREAL)) = tid;
      *mxGetPr(in[2]= mxCreateDoubleMatrix(1,1,mxREAL)) = tag;
        err=mexCallMATLAB(1,&out,3,in,recvfun.MatlabCMD);
  if (err)
    mexWarnMsgTxt("recvf_wrapper: error durante mexCallMATLAB");
                          ret=0;
  if (mxIsNumeric(out)) ret=mxGetScalar(out);
  else
    mexWarnMsgTxt("recvf_wrapper: handler debe retornar int");
                 for(err=0;err<3;err++)
  mxDestroyArray(in[err]);
  mxDestroyArray(out);
  return(ret);
```

```
}

/*-----------------------------------------------------------------------*/
void mexFunction(int nlhs, mxArray*plhs[], int nrhs, const mxArray*prhs[])
/*-----------------------------------------------------------------------*/
{
        if ( (!nrhs) || (mxIsEmpty(prhs[0])) )
/*======================== OLD ========================*/
  {
        recvfun.oldf = pvm_recvf(0);
        recvfun.oldf =         NULL;
 mxFree(recvfun.MatlabCMD);
        recvfun.MatlabCMD =    NULL;
  }

/*======================== ERR ========================*/
else if (!mxIsChar(prhs[0]))
                mexErrMsgTxt("pvm_recvf: se requiere arg string");
else
/*======================== NEW ========================*/
  {
        recvfun.oldf = pvm_recvf(wrapper);
 mxFree(recvfun.MatlabCMD);
        recvfun.MatlabCMD = mxArrayToString(prhs[0]);
  }

}
```

Como se observa en el código mostrado, se define una rutina C, “wrapper”, que hará de recubrimiento al comando del usuario. El usuario indica como argumento un *string* (*MatlabCMD*) evaluable mediante la llamada API **mexCallMATLAB**. Dicho comando de retrollamada (*callback*) debe aceptar tres argumentos numéricos (bufid, tid, tag) y debe devolver un código que PVM interpreta como error (<0), descarte (0), aceptación del mensaje (1) o rango para decidir cuál se acepta (>1).

Se anota entonces (sección NEW en el listado) la antigua función de recepción y se registra con PVM el *wrapper* C que servirá a partir de ahora como función de recepción. Al recibir cada mensaje, PVM llama al *wrapper*, que a su vez llama al *callback* MATLAB previamente anotado. No se devuelve el antiguo *callback* porque no tiene utilidad bajo MATLAB.

Las funciones pvm_addmhf y pvm_delmhf son más complicadas porque requieren el uso de múltiples *wrappers* (uno por cada condición de manejo deseada), y mantener una lista con los identificadores de los manejadores (*handlers*) que se vayan creando con addmhf, para poder liberarlos después usando delmhf. Ambas llamadas PVM se implementan en el mismo fichero-MEX para mantener la lista de manejadores en memoria sin perder su valor entre llamadas.

C.7 Extensiones

Las extensiones se han nombrado con el prefijo pvme_ (PVM_Extension) y están programadas como ficheros-M.

El comando pvme_start_pvmd simplifica el número de argumentos de llamada, pudiendo llegar a

usar un fichero de configuración por defecto, indicado mediante pvme_default_config.

El comando pvme_is comprueba si está ejecutándose el *daemon* pvmd.

Los comandos pvme_kill, pvme_pstat y pvme_sendsig admiten un *array* como argumento, en vez de un único entero tid.

El comando pvme_gids devuelve todos los tids en un grupo, no sólo el de una instancia indicada.

La orden [**info**,names]=pvm_unpack[('vnam'[,'vnam']...)] desempaqueta tantas variables como nombres se le indiquen. Si no se indica ninguno, desempaqueta hasta encontrar la condición de error *end-of-buffer* indicando que no se recibieron más datos. Si no se desea obtener dicho error, se debe conocer de antemano el número exacto de variables que fueron empaquetadas. Alternativamente, la extensión pvme_pack empaqueta las variables tras reunirlas bajo un *cell-array* MATLAB. Al desempaquetar se detecta dicha condición, se ignora la organización *cell-array* y se desempaqueta el número exacto de variables.

Un proceso MATLAB puede recibir mensajes enviados por otros procesos MATLAB usando el comando PVMTB pvm_recv. Pero también el propio *daemon* PVM puede enviar mensajes al proceso MATLAB; mensajes de notificación causados por un evento destacado mediante un comando pvm_notify previo, por ejemplo. Dichos mensajes consisten en uno o más int. pvm_pack almacena información adicional con cada variable MATLAB indicando su nombre, dimensiones y tamaño, información ausente en los mensajes provenientes del *daemon*, naturalmente. Se programó pues una extensión PVMTB pvme_upkntfy, interfaz con la llamada PVM pvm_upkint para poder desempaquetar como variable MATLAB los mensajes de notificación mandados por el propio PVM en respuesta a pvm_notify. Dado que los mensajes consisten en un entero *tid* (notificaciones PvmTaskExit y PvmHostDelete) o en un entero de cuenta y un array de *tids* (notificación PvmHostAdd), la magnitud del primer entero puede usarse como discriminante del tipo de variable MATLAB que debe crearse para almacenar el mensaje. Naturalmente, los int enviados por PVM en los mensajes de notificación se coercen a *double*.

C.8 Makefile

Para automatizar el proceso de compilación de todos estos ficheros-MEX fuente se redactó un Makefile para dirigir la acción de la utilidad make.

Según es usual, las variables describiendo los nombres de los comandos y modificadores a usar se segregaron en un fichero auxiliar Makefile.env, dejando en el Makefile únicamente la descripción del proyecto, es decir, los ficheros a compilar y enlazar y las dependencias existentes entre ellos.

```
# Makefile para src
include Makefile.env

# Fuentes programados por el sistema header.h
PVMSRCS = pvm_halt.c          pvm_exit.c          pvm_kill.c   pvm_sendsig.c \
          ...
          pvm_newcontext.c pvm_setcontext.c pvm_freecontext.c

# Fuentes compartiendo header_col.h
COLSRCS = pvm_reduce.c        pvm_scatter.c       pvm_gather.c
```

```
# Fuentes compartiendo header_msk.h
MSKSRCS = pvm_gettmask.c     pvm_settmask.c   TEV_MASK_INIT.c \
          TEV_MASK_CHECK.c TEV_MASK_SET.c    TEV_MASK_UNSET.c

# Fuentes compartiendo header_reg.h
REGSRCS = pvm_reg_hoster.c pvm_reg_tasker.c pvm_reg_rm.c

# Fuentes compartiendo header_smp.h
SMPSRCS = pvm_start_pvmd.c pvm_spawn.c      pvm_config.c pvm_tickle.c \
          ...
          pvm_pkdouble.c   pvm_upkdouble.c  pvm_hostsync.c

# Fuentes a mano, sin sistema
FUNSRCS = pvm_mhf.c        pvm_recvf.c
PCKSRCS = pvm_pack.c       pvm_unpack.c
NRMSRCS = putenv.c         select.c         unsetenv.c
```

Se enumeran los programas fuente. Los nombres de los objeto y ejecutables se obtienen mediante sustitución de patrones:

```
SRCS    = $(PVMSRCS) $(COLSRCS) $(MSKSRCS) $(REGSRCS) \
          $(SMPSRCS) $(FUNSRCS) $(PCKSRCS) $(NRMSRCS)

   OBJS =        $(patsubst %.c, %.o,    $(SRCS))
PVMOBJS =        $(patsubst %.c, %.o, $(PVMSRCS))
COLOBJS =        $(patsubst %.c, %.o, $(COLSRCS))
MSKOBJS =        $(patsubst %.c, %.o, $(MSKSRCS))
REGOBJS =        $(patsubst %.c, %.o, $(REGSRCS))
SMPOBJS =        $(patsubst %.c, %.o, $(SMPSRCS))

MEXS      = $(patsubst %.c, %.mexlx, $(SRCS))
INSTDIR   = ../pvm/MEX
INSTMEXS = $(addprefix $(INSTDIR)/, $(MEXS))
```

Se añaden los *objetivos* (*targets*) típicos install y clean, y se le indica a la utilidad make el sufijo de los ficheros-MEX bajo UNIX y cómo se obtienen a partir de los correspondientes objeto. Los ficheros-MEX son copiados al subdirectorio de instalación, tras lo cual se eliminan del subdirectorio de compilación.

```
# Target
.PHONY: install clean

install:      $(INSTMEXS)

clean:
        -cd     $(INSTDIR); $(RM) $(MEXS)
        -rmdir $(INSTDIR)

# Build Rules
.SUFFIXES: .mexlx
.INTERMEDIATE: $(OBJS) $(MEXS)

$(INSTDIR):
        mkdir -p $(INSTDIR)
```

```
$(INSTMEXS): $(INSTDIR)/%: % $(INSTDIR)
	$(CP) $< $(INSTDIR)

$(MEXS): %.mexlx: %.o
	$(CC) $(LDFLAGS) $(LDLIBS) $< -o $@
```

La clasificación previamente realizada sobre las llamadas PVM permite controlar con precisión las dependencias .c←.h: Estas se expresan concisamente mediante:

```
$(OBJS): %.o: %.c

$(PVMOBJS): header.h

$(COLOBJS): header_col.h

$(MSKOBJS): header_msk.h

$(REGOBJS): header_reg.h

$(SMPOBJS): header_smp.h
```

Los comandos y modificadores usados en el proceso de compilación son:

```
# Makefile.env
CP = cp
MV = mv
CC = i486-linuxlibc5-gcc

# Flags para compilar ficheros-MEX bajo Matlab5
# (ver $MATLAB/bin/mexopts.sh ../mex, probar mex -v foo.c)
MEXCPPFLAGS = -I$(MATLAB)/extern/include -DMATLAB_MEX_FILE
DEBUGFLAGS  = -g
OPTIMFLAGS  = -O -DNDEBUG
MEXLDFLAGS  = -shared
MEX2NDOBJ = mexversion.o
vpath       mexversion.c  $(MATLAB)/extern/src

# Forma de compilar bajo PVM3.4
PVMCPPFLAGS = -I$(PVM_ROOT)/include
PVMLDFLAGS  = -L$(PVM_ROOT)/lib/$(PVM_ARCH)
PVMLIBS     = -lpvm3 -lgpvm3

# Flags a usar depuración/optimización
CPPFLAGS = $(MEXCPPFLAGS) $(PVMCPPFLAGS)
# CFLAGS   = $(OPTIMFLAGS)
CFLAGS   = $(DEBUGFLAGS)
LDFLAGS  = $(MEXLDFLAGS) $(PVMLDFLAGS)
LDLIBS   = $(PVMLIBS)
```

Se debe notar que bajo el entorno de desarrollo escogido, Linux RedHat 6.2 con MATLAB 5.2, el ejecutable de MATLAB seguía distribuyéndose enlazado contra las antiguas bibliotecas C libc5. Esto obligó a instalar un entorno de compilación de compatibilidad con libc5, como se observa en la definición de $(CC).

Es costumbre añadir a los ficheros-MEX información relativa a la versión MATLAB usada. El código fuente para *versionado* viene dado por MATLAB. Para indicarle a la utilidad make dónde

encontrar dicho código se ha usado la directiva **vpath**.

Para simplificar el Makefile, se ha aprovechado que la dependencia implícita .o←.c usa las variables CPPFLAGS y CFLAGS. Las variables LDFLAGS y LDLIBS se referencian explícitamente en la dependencia .mexlx←.o del Makefile.

Apéndice D

Código para el estudio del *overhead*

D.1 Introducción

En los Capítulos 3 y 4 se ha hecho referencia a programas C y MATLAB desarrollados para evaluar la pérdida de prestaciones asociada al uso de las *Toolboxes* paralelas PVMTB y MPITB, frente al paso de mensajes usando directamente la biblioteca correspondiente.

Los programas presentados consisten en un test *ping-pong* de paso quasi-exponencial. Como ya se comentó en los capítulos correspondientes, este test permite barrer un rango significativo de tamaños de mensaje, manteniendo el tiempo de test a una extensión razonable, sin sacrificar el detalle para los tamaños de mensaje pequeños.

D.2 Listados

A continuación se ofrecen los listados de los programas referenciados. Este directorio permite localizarlos fácilmente y relacionarlos con el apartado en el que se hace referencia a ellos:

	Apartado	Listado	página
PVM en lenguaje C	3.4.1 p.192		
pvmCode/pvmGenFiles: *script* de automatización		D.1	370
pvmCode/BW.c: proceso fuente		D.2	371
pvmCode/BWSlav.c: proceso eco		D.3	374
PVMTB bajo MATLAB	3.4.2 p.204		
pvmCode/BW.m: proceso fuente		D.4	376
pvmCode/BWSlav.m: proceso eco		D.5	379
MPI en lenguaje C	4.4.1 p.238		
mpiCode/mpiGenFiles: *script* de automatización		D.6	381
mpiCode/BW.c: proceso fuente		D.7	382
mpiCode/BWSlav.c: proceso eco		D.8	385
MPITB bajo MATLAB	4.4.2 p.245		
mpiCode/BW.m: proceso fuente		D.9	386
mpiCode/BWSlav.m: proceso eco		D.10	388

07/31/01 13:44:01 pvmCode/pvmGenFiles **1**

```
#!/bin/bash

############################################################################
if [ $# -eq 0 ]; then
echo
echo Usage: ./pvmGenFiles SUB [ COD [ ROU [ ROW [ N [ FN HN0 HN1]]]]]
echo SUB••••••••: subdirectorio en donde guardar fichero mediciones
echo COD•••••[2]: empaquetamiento PvmDataDefault\(0\)/ Raw\(1\)/ InPlace\(2\)
echo ROU•••••[1]: encaminamiento PvmDontRoute\(0\) / PvmRouteDirect\(1\)
echo ROW••••[34]: barrido hasta esta fila del array doubles
echo •N••••••[1]: factor repeticiones para 32MB -\> más pequeño, más veces
echo FN•••••[BW]: nombre base fichero mediciones BW-hn0-hn1-\<cod\>-\<rou\>.dat
echo HN0•••[ox0]: nombre 1er ordenador \(fuente\)
echo HN1•••[ox1]: nombre 2°  ordenador \(eco\)
echo
exit
fi
############################################################################
SUB=${1:-.}     # el valor por defecto no se usa, se ejecutaría el Usage antes
COD=${2:-2};    case $COD in 0) ENCOD=def;; 1) ENCOD=raw;; 2) ENCOD=pla;; esac
ROU=${3:-1};    case $ROU in 0) ROUTE=no;;  1) ROUTE=di;;  esac
ROW=${4:-34};
  N=${5:-1}
 FN=${6:-BW}
HN0=${7:-ox0}
HN1=${8:-ox1}   FN=${FN}-${HN0}-${HN1}-${ENCOD}-${ROUTE}.dat

# MasterName, SlaveName, FileName
MN=BW
SN=BWSlav

############################################################################
# Crear hostfile conteniendo PATH. Matar posible PVM anterior
############################################################################

pvm  <<-@
        halt
        @
cat >   hostfile <<-@
        * ep='pwd':\$PATH
        @

############################################################################
# Arrancar esclavo (remoto), maestro y servidor de grupos (locales)
# ejecutar maestro desde shell (desde consola PVM no se queda bloqueado)
# maestro BW hace pvm_halt, no necesitamos parar PVM nosotros
############################################################################
# Usage: ./BW encode droute row ntimes fname
#       BWSlav encode droute row ntimes
############################################################################
    pvm -n$HN0 hostfile <<-@
        add $HN1
        spawn -$HN0 pvmgs
        spawn -$HN1 $SN $COD $ROU $ROW $N
        @
echo             ./$MN $COD $ROU $ROW $N $SUB/$FN
vmstat
                 ./$MN $COD $ROU $ROW $N $SUB/$FN
vmstat
rm hostfile
```

Listado D.1: pvmCode/pvmGenFiles: *Script* de automatización para el test *ping-pong* en PVM.

08/01/01 11:52:53 pvmCode/BW.c **1**

```
/****************************************************************
% BW: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       [l u d] = BW ( encode, droute, row, ntimes, fname )
%
%  encode       codificación usada en empaquetamiento
%       0       PvmDataDefault, traducción a XDR
%       1       PvmDataRaw,     no se traduce a XDR
%       2       PvmDataInPlace, no se hace buffer en send siquiera
%  droute 0/1   solicita enrutamiento directo con pvm_setopt
%  row  1..34   fila de la columna double hasta donde se desea llegar
%               Los uint8 llegan hasta row+4  [34+4=38 -> 32MB]
%               Debe ser de la forma row=3n+1 [34=3*11+1]
%                si se desea que coincidan tamaños double/uint8 última medición
%  ntimes 1..?  número de veces a repetir transmisión 32MB
%               s bytes -> 2^fix((30-fix(log2(s)))/3.33-1)*ntimes veces
%               ej: ntimes=1, 0-1B 256 veces, 2-15B 128 veces, 16-64B 64 veces,
%               128-1KB 32 veces, 2-15KB 16 veces, 16-64KB 8 veces,
%               128KB-1MB 4 veces, 2-15MB 2 veces, 16-64MB 1 vez.
%  fname        nombre del fichero donde salvar las mediciones:
%
%  l            Latencias (vector columna)
%  u            Datos para uint8   (3 col)
%  d            Datos para doubles (3 col)
%
%       Los datos son arrays de 3 Columnas: [Size TXtm Mtm]
%               Size: tamaño del array
%               TXtm: tiempo de transmisión
%               Mtm:  tiempo para crear array en memoria
%       Filas: Tamaños de array crecientes (ver BWrows.m)
%               fila    dimensiones     tamaño uint8    tamaño double
%                1         1x   1       2^ 0 =  1 B     2^ 3 =   8 B
%                2         1x   2       2^ 1 =  2 B     2^ 4 =  16 B
%                3         1x   3               3 B             24 B
%                4         2x   2       2^ 2    4 B     2^ 5 =  32 B
%                5         2x   4       2^ 3    8 B     2^ 6 =  64 B
%                6         2x   6              12 B             96 B
%                7         4x   4       2^ 4   16 B     2^ 7 = 128 B
%              ...
%               34      2048x2048       2^22 =  4MB     2^25 =  32MB
%               35      2048x4096       2^23 =  8MB     2^26
%               36      2048x6144              12MB
%               37      4096x4096       2^24 = 16MB     2^27
%               38      4096x8192       2^25 = 32MB     2^28
 ****************************************************************/

#include <stdio.h>      /* printf, fflush */
#include <stdlib.h>     /* atoi, getenv, calloc */
#include <pvm3.h>       /* pvm_* */
#include <math.h>       /* floor */

#include <sys/stat.h>   /* para creat */
#include <sys/types.h>
#include <fcntl.h>
#include <sys/time.h>   /* para gettimeofday() */
#include <unistd.h>     /* para write, close */

#define USTR 0
#define UEND 37
#define ROWS 38
#define DSTR 0
#define DEND 33
#define Size 0
#define TXtm 1
#define Mtm  2
#define COLS 3

#define TAG      7
#define SWAPLIM 30

/*************/
double Wtime(){
/*************/
  struct timeval now; gettimeofday(&now, NULL);
  return((double)now.tv_sec + (double)now.tv_usec/1E6);
}

/**************/
int reps(int s){                          /* 2^fix((30-fix(log2(s)))/3.33-1) */
/**************/
int log2s,pwr;
        for(log2s=0; s>>=1; log2s++);
        pwr=floor((30-log2s)/3.33-1);
        return(1<<pwr);
}

/************/
/* PROGRAMA */
/************/
int main(int argc, char*argv[]){
        int encode,droute,row,ntimes,NTIMES;    /* argumentos del programa */
        char*fname;
        int  fd;                                /* file descriptor */
        double var;                             /* Variable para write */

        int Dstr,Dend,Ustr,Uend;                /* índices medición */
        double       *arrayDouble;              /* arrays de datos */
        unsigned char*arrayUint8;
        double  l[ROWS];                        /* Latencias */
        double  u[ROWS][COLS];                  /* arrays de mediciones */
        double  d[ROWS][COLS];

        int ExpAncho,Ancho,Alto,i,indx=-1;      /* índices de bucles */
        double T,Tot;                           /* Variables para tiempo */

        int nhost,me,other,inum,onum,nelem;     /* tids, gids, etc */
        char lat=' ';                           /* pvm_pkbyte para latencia */

/****************************************************************/

        Tot = Wtime();                          /* como tic-toc matlab */
  /**************/
  /* Argumentos */
  /**************/
        if (argc<6) {
    printf("\n"
        "Usage: ./BW encode droute row ntimes fname\n"
        " encode: 0..2 empaquetamiento PvmDataDefault/Raw/InPlace\n"
        " droute: 0/1  encaminamiento  PvmDontRoute/RouteDirect\n"
        "    row: 1..34 fila de la columna double hasta donde se desea llegar\n"
        " ntimes: 1..?  número de veces a repetir transmisión 32MB\n"
        "  fname:       nombre del fichero donde salvar las mediciones\n"
        "\n");
    exit(1);
  }
        encode = atoi(argv[1]);                 /* empaquetamiento */
```

Listado D.2: pvmCode/BW.c: Proceso fuente en PVM. Se mostró un fragmento en el Listado 3.21.

08/01/01
11:52:53

pvmCode/BW.c

2

```
       droute = atoi(argv[2]);                  /* encaminamiento */
          row = atoi(argv[3]);                  /* hasta qué fila barrer */
       ntimes = atoi(argv[4]);                  /* factor repeticiones */
        fname =      argv[5] ;                  /* nombre fichero mediciones */

  if (droute) pvm_setopt(PvmRoute, PvmRouteDirect);
  else        pvm_setopt(PvmRoute, PvmDontRoute);
  if(--row%3) printf("no coincidirán tamaños finales double/uint8\n");
              Dstr=     Ustr=     0;
              Dend=row; Uend=Dend+4;

  pvm_config(&nhost, NULL, NULL);
  if (nhost<2){
    printf("hacen falta 2 ordenadores en PVM\n");
    exit(3);
  }

  /***********************/
  /* Obtener tid del otro */
  /***********************/
  me=pvm_mytid();
  inum=pvm_joingroup("BW");             /* instancia 0 ó 1 */
  pvm_freezegroup("BW",2);
  onum =1-inum;
  other=pvm_gettid("BW",onum);       /* el otro, el del pong */

/* mandar algo, ejercitar posible ruta directa de paso (y posible swap) */
  printf("\nTransmitiendo 32MB (posible swapping)..."); fflush(stdout);
  nelem=2048*2048;
  arrayDouble=calloc(nelem,sizeof(double));
  pvm_initsend(encode); pvm_pkdouble(arrayDouble,nelem,1);
  pvm_send(other,TAG);
  pvm_freebuf(pvm_getsbuf());
  pvm_recv(other,TAG); pvm_upkdouble(arrayDouble,nelem,1);
  pvm_freebuf(pvm_getrbuf());
  free(arrayDouble);
  printf("\n...hecho\n");

/*****************
% BUCLE PING-PONG %
 *****************/
  for (ExpAncho=0; ExpAncho<13; ExpAncho++){
    Ancho=1<<ExpAncho;
    printf("\n(%4d\t",Ancho);

    for (Alto=Ancho; Alto<4*Ancho; Alto+=Ancho){
      printf("x%4d) ",Alto);
      indx++; nelem=Ancho*Alto;

        /**************************************************************/

      if(((indx>=Dstr)&&(indx<=Dend))||         /* algo que hacer */
         ((indx>=Ustr)&&(indx<=Uend))){

        NTIMES=ntimes*reps(0);
        printf("["); fflush(stdout);
        pvm_barrier("BW",2);                    /* hacer parte latencia */

        T=Wtime();
        for (i=0; i<NTIMES; i++){
          pvm_initsend(encode); pvm_pkbyte(&lat,0,1); pvm_send(other,TAG);
          pvm_recv(other,TAG);  pvm_upkbyte(&lat,0,1);
        }
        T=Wtime()-T;            l[indx]=T/2/NTIMES;
      }

        /**************************************************************/

      if ((indx>=Dstr)&&(indx<=Dend)){          /* hacer parte double */

        NTIMES=ntimes*reps(nelem*sizeof(double)); arrayDouble=NULL;
        T=Wtime();
        for (i=0; i<NTIMES; i++){               /* medir reserva memoria */
          free(arrayDouble);
          arrayDouble=calloc(nelem,sizeof(double));
        }
        T=Wtime()-T;    d[indx][Size]=nelem*sizeof(double);
                        d[indx][ Mtm]=T/NTIMES;

        printf("d"); fflush(stdout);            /* ping-pong parte double */
        pvm_barrier("BW",2);

        T=Wtime();
        for (i=0; i<NTIMES; i++){
          pvm_initsend(encode); pvm_pkdouble(arrayDouble,nelem,1);
          pvm_send(other,TAG);
            if (indx>=SWAPLIM)  pvm_freebuf  (pvm_getsbuf()      );
          pvm_recv(other,TAG);  pvm_upkdouble(arrayDouble,nelem,1);
            if (indx>=SWAPLIM)  pvm_freebuf  (pvm_getrbuf()      );
        }
        T=Wtime()-T;    d[indx][TXtm]=T/2/NTIMES;

        free(arrayDouble);      pvm_freebuf  (pvm_getsbuf());
                                pvm_freebuf  (pvm_getrbuf());
                printf(",");

                /******************************************************/

      } else    printf("-,");
                fflush(stdout);

        /**************************************************************/

      if ((indx>=Ustr)&&(indx<=Uend)){          /* hacer parte uint8 */

        NTIMES=ntimes*reps(nelem*sizeof(unsigned char)); arrayUint8=NULL;
        T=Wtime();
        for (i=0; i<NTIMES; i++){
          free(arrayUint8);
          arrayUint8=calloc(nelem,sizeof(unsigned char));
        }
        T=Wtime()-T;    u[indx][Size]=nelem*sizeof(unsigned char);
                        u[indx][ Mtm]=T/NTIMES;

        printf("u"); fflush(stdout);            /* ping-pong parte char */
        pvm_barrier("BW",2);

        T=Wtime();
        for (i=0; i<NTIMES; i++){
          pvm_initsend(encode); pvm_pkbyte(arrayUint8,nelem,1);
          pvm_send(other,TAG);
            if (indx>=SWAPLIM) pvm_freebuf(pvm_getsbuf()     );
          pvm_recv(other,TAG); pvm_upkbyte(arrayUint8,nelem,1);
            if (indx>=SWAPLIM) pvm_freebuf(pvm_getrbuf()     );
        }
        T=Wtime()-T;    u[indx][TXtm]=T/2/NTIMES;
```

Listado D.2: Continuación.

08/01/01 11:52:53 — pvmCode/BW.c — 3

```
       free(arrayUint8);       pvm_freebuf(pvm_getsbuf());
                               pvm_freebuf(pvm_getsbuf());
               printf("]\t" );

               /********************************************************/

     } else    printf("-]\t");
               fflush(stdout);

/*****************************************************************/

    }
  }
  printf("\n");

  pvm_lvgroup("BW");
  pvm_exit();

  Tot=Wtime()-Tot;
  printf("Elapsed time: %8.3f s\n",Tot);

/****************************************************************
% Salvar Datos
 ****************************************************************/

  if ((fd=creat(fname,S_IRWXU))<0){
        printf("no puedo crear fichero %s\n",fname);
        exit(15);
  }
  var=Ustr+1;   write(fd,&var,sizeof(var));     /* índices para MATLAB */
  var=Uend+1;   write(fd,&var,sizeof(var));
  var=Dstr+1;   write(fd,&var,sizeof(var));
  var=Dend+1;   write(fd,&var,sizeof(var));
  var=ROWS;     write(fd,&var,sizeof(var));     /* Tamaño datos */
  var=COLS;     write(fd,&var,sizeof(var));
                write(fd,    l,sizeof(l ));     /* datos */
                write(fd,    u,sizeof(u ));
                write(fd,    d,sizeof(d ));
  if (close(fd)){
        printf("no puedo cerrar fichero %s\n",fname);
        exit(16);
  }

  pvm_halt();
  exit(0);

}
```

Listado D.2: Continuación.

08/01/01 11:53:14 — pvmCode/BWSlav.c — 1

```
/****************************************************************
% BWSlav: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       BWSlav ( encode, droute, row, ntimes )
%
 ****************************************************************/

#include <stdlib.h>     /* atoi  */
#include <pvm3.h>       /* pvm_* */
#include <math.h>       /* floor */

#define TAG      7
#define SWAPLIM 30

/**************/
int reps(int s){                                /* 2^fix((30-log2(s))/3.33-1) */
/**************/
int log2s,pwr;
        for(log2s=0; s>>=1; log2s++);
        pwr=floor((30-log2s)/3.33-1);
        return(1<<pwr);
}

/************/
/* PROGRAMA */
/************/
int main(int argc,char*argv[]){
        int encode,droute,row,ntimes,NTIMES;    /* argumentos */
        int Dstr,Dend,Ustr,Uend;                /* índices medición */
        double       *arrayDouble;              /* arrays de datos */
        unsigned char*arrayUint8;

        int ExpAncho,Ancho,Alto,i,indx=-1;      /* índices de bucles */
        int me,other,inum,onum,nelem;           /* tids, gids, etc */
        char lat=' ';                           /* pvm_pkbyte para latencia */

  /**************/
  /* Argumentos */
  /**************/
  if (argc<5) exit(1);
        encode = atoi(argv[1]);                 /* empaquetamiento */
        droute = atoi(argv[2]);                 /* encaminamiento */
           row = atoi(argv[3]);                 /* hasta qué fila barrer */
        ntimes = atoi(argv[4]);                 /* factor repeticiones */

        Dstr=       Ustr=     0;
        Dend=--row; Uend=Dend+4;

        if (droute) pvm_setopt(PvmRoute, PvmRouteDirect);
        else        pvm_setopt(PvmRoute, PvmDontRoute);

  /************************/
  /* Obtener tid del otro */
  /************************/
  me  =pvm_mytid();
  inum=pvm_joingroup("BW");                     /* instancia 0 ó 1 */
  pvm_freezegroup("BW",2);
  onum =1-inum;
  other=pvm_gettid("BW",onum);                  /* el otro, el del ping */

/* recibir algo, ejercitar posible ruta directa de paso (y posible swap) */
  nelem=2048*2048;
  arrayDouble=calloc(nelem,sizeof(double));
pvm_recv(other,TAG); pvm_upkdouble(arrayDouble,nelem,1);
pvm_freebuf(pvm_getrbuf());
pvm_initsend(encode); pvm_pkdouble(arrayDouble,nelem,1);
pvm_send(other,TAG);
pvm_freebuf(pvm_getsbuf());
free(arrayDouble);

/********/
/* Pong */
/********/
for (ExpAncho=0; ExpAncho<13; ExpAncho++){
  Ancho=1<<ExpAncho;
  for (Alto=Ancho; Alto<4*Ancho; Alto+=Ancho){
    indx++; nelem=Ancho*Alto;

      /**************************************************************/

    if(((indx>=Dstr) && (indx<=Dend))||       /* algo que hacer */
       ((indx>=Ustr) && (indx<=Uend))){

      NTIMES=ntimes*reps(0);
      pvm_barrier("BW",2);                    /* repetir parte latencia */
      for (i=0; i<NTIMES; i++){
        pvm_recv(other,TAG); pvm_upkbyte(&lat,0,1);
        pvm_initsend(encode); pvm_pkbyte(&lat,0,1);
        pvm_send(other,TAG);
      }
    }

      /**************************************************************/

    if ((indx>=Dstr) && (indx<=Dend)){        /* hacer parte double */

      NTIMES=ntimes*reps(nelem*sizeof(double));
      arrayDouble=calloc(nelem,sizeof(double));
      pvm_barrier("BW",2);
      for (i=0; i<NTIMES; i++){
        pvm_recv(other,TAG); pvm_upkdouble(arrayDouble,nelem,1);
          if (indx>=SWAPLIM)  pvm_freebuf (pvm_getrbuf()      );
        pvm_initsend(encode); pvm_pkdouble(arrayDouble,nelem,1);
        pvm_send(other,TAG);
          if (indx>=SWAPLIM)  pvm_freebuf (pvm_getsbuf()      );
      }
      free(arrayDouble);      pvm_freebuf (pvm_getrbuf());
                              pvm_freebuf (pvm_getsbuf());
    }

      /**************************************************************/

    if ((indx>=Ustr) && (indx<=Uend)){        /* hacer parte uint8 */

      NTIMES=ntimes*reps(nelem*sizeof(unsigned char));
      arrayUint8 =calloc(nelem,sizeof(unsigned char));
      pvm_barrier("BW",2);
      for (i=0; i<NTIMES; i++){
        pvm_recv(other,TAG); pvm_upkbyte(arrayUint8,nelem,1);
          if (indx>=SWAPLIM)  pvm_freebuf (pvm_getrbuf()      );
        pvm_initsend(encode); pvm_pkbyte(arrayUint8,nelem,1);
        pvm_send(other,TAG);
          if (indx>=SWAPLIM)  pvm_freebuf (pvm_getsbuf()      );
      }
      free(arrayUint8);       pvm_freebuf (pvm_getrbuf());
                              pvm_freebuf (pvm_getsbuf());
```

Listado D.3: pvmCode/BWSlav.c: Proceso eco en PVM.

08/01/01 11:53:14 — pvmCode/BWSlav.c — 2

```
      }

      /***************************************************************/
    }   /* for Alto */
  }     /* for ExpAncho */

  pvm_lvgroup("BW");
  pvm_exit();
  exit(0);
}
```

Listado D.3: Continuación.

08/01/01 12:07:32 pvmCode/BW.m **1**

```
function BW(SUB,encode,droute,row,ntimes,FN,HN0,HN1)
% BW: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       BW ( SUB [,encode [,droute [,row [,ntimes [,FN,HN0,HN1]]]]] )
%
%  SUB          subdirectorio en donde guardar fichero mediciones
%  encode       codificación usada en empaquetamiento
%            0  PvmDataDefault, traducción a XDR
%            1  PvmDataRaw,     no se traduce a XDR
%           [2] PvmDataInPlace, no se hace buffer en send siquiera
%  droute 0/[1] solicita enrutamiento directo con pvm_setopt
%  row  1..[34] fila de la columna double hasta donde se desea llegar
%               Los uint8 llegan hasta row+4  [34+4=38 -> 32MB]
%               Debe ser de la forma row=3n+1 [34=3*11+1]
%                si se desea que coincidan tamaños double/uint8 última medición
%  ntimes   [1] número de veces a repetir transmisión 32MB
%               s bytes -> 2^fix((30-fix(log2(s)))/3.33-1)*ntimes veces
%               ej: para ntimes=1,
%                 0-1 B 256 veces, 2-15B 128 veces, 16-64B 64 veces,
%               128 -1KB 32 veces, 2-15KB 16 veces, 16-64KB 8 veces,
%               128KB-1MB 4 veces, 2-15MB  2 veces, 16-64MB 1 vez.
%  FN      [BW] nombre base fichero mediciones FN-hn0-hn1-<cod>-<rou>.mat
%  HN0    [ox0] nombre del 1er computador (fuente)
%  HN1    [ox1] nombre del 2°  computador (eco)
%
%               En el fichero se salvan:
%       Ustr,Uend, Dstr,Dend    ej: 1,34,1,38
%       ROWS,COLS               ej:        38,4
%       l       Latencias       (vector columna)
%       u       Datos para uint8        (3 col)
%       d       Datos para doubles      (4 col)
%
%       u/d son arrays de 3/4 Columnas: [Size TXtm Mtm Ctm]
%               Size: tamaño del array
%               TXtm:   tiempo de transmisión
%               Mtm:    tiempo para crear array en memoria
%               Ctm:    tiempo para pasar double->uint8 (typecast)
%       Filas: Tamaños de array crecientes (help BWrows)
%

tic
  SWAPLIM=30; PACK=0; DEBUG=0; global TAG ROU DIR DONT  % heredadas de startup
  reps=inline('2.^fix((30-fix(log2(max(s,1)))))./3.33-1)');
  varDbl=      0 ; s=whos('varDbl'); sizeofDbl=s.bytes; % para cuentas tamaños
  varU8 =uint8(0); s=whos('varU8' ); sizeofU8 =s.bytes;
  clear s varDbl varU8

%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
  if nargin<1, help(mfilename), return, end
  if nargin<2, encode= 2 ; end
  if nargin<3, droute= 1 ; end
  if nargin<4, row   =34 ; end
  if nargin<5, ntimes= 1 ; end
  if nargin<6, FN   ='BW'; end
  if nargin<7, HN0 ='ox0'; end
  if nargin<8, HN1 ='ox1'; end
  flag=0;
  flag=flag | ~exist(SUB,'dir');
  flag=flag | fix(encode)~=encode | encode<0 | encode>2;ENC=int2str(encode);
  flag=flag | fix(droute)~=droute | droute<0 | droute>1;DRT=int2str(droute);
  flag=flag | fix(row)   ~=row    | row   <1 | row > 34;ROW=int2str(row);
  flag=flag | fix(ntimes)~=ntimes | ntimes<1;           NTM=int2str(ntimes);
  flag=flag | ~ischar(FN) | ~ischar(HN0) | ~ischar(HN1);
  if flag, help(mfilename), return, end

  if mod(row,3)~=1, warning('no coincidirán tamaños finales double/uint8'), end
  Size=1; TXtm=2; Mtm = 3; Ctm =4;                      % columnas
  indx=0; Ustr=1; Uend=38; Dstr=1; Dend=34;             % filas
                  Uend=row+4;      Dend=row;
                  ROWS=Uend;       COLS=Ctm;
  l=zeros(Uend,1);
  u=zeros(Uend,Mtm); d=zeros(Dend,Ctm);

%%%%%%%%%%%%%%%%%%%%%%%
% Arrancar daemon PVM %
%%%%%%%%%%%%%%%%%%%%%%%
  HN={['-n' HN0], ['* ep=$PATH wd=' pwd], HN1};
  pvm_setopt(3,0);                                      % AutoErr Ignore
  pvme_start_pvmd(HN{:});                               % arrancar daemon
  pvm_setopt(3,1);                                      % AutoErr Warn
  if droute, pvm_setopt(ROU,DIR);                       % ruta directa?
      else, pvm_setopt(ROU,DONT);
   end

%%%%%%%%%%%
% COMANDO %
%%%%%%%%%%%
  pvm_spawn('pvmgs',{},1,'.',1);        % groupserver siempre en el servidor
  inum=pvm_joingroup('BW');             % grupos requeridos para barreras
  if inum~=0, error('no puedo crear grupo'), end

  cmd=['BWSlav(' ENC ',' DRT ',' ROW ',' NTM ');'];
  putenv(['cmd=' cmd]); pvm_export('cmd');

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arranque MATLAB esclavo % El que responde al ping
%%%%%%%%%%%%%%%%%%%%%%%%%%%
  if DEBUG
    DISP=getenv('DISPLAY');
    [numt tids]=pvm_spawn('xterm',...
                    {'-display' DISP '-e' 'matlab'},33,'.',1);
  else
    [numt tids]=pvm_spawn('matlab',{'-nosplash'},       33,'.',1);
  end
  if numt~=1, error('cannot start MATLAB'), end

  pvm_freezegroup('BW',2);              % ahora que hijo puede congelar

  unix('vmstat');
  fprintf('\nTransmitiendo 32MB (posible swapping)...');
  arrayNULL(2048,2048)=0;               % ejercitar posible ruta directa/swap
if PACK,pvm_initsend(encode);     pvm_pack( arrayNULL ); pvm_send(tids,TAG);
        pvm_freebuf(pvm_getsbuf);
        pvm_recv(tids,TAG);     pvm_unpack('arrayNULL');
        pvm_freebuf(pvm_getrbuf);
  else, pvm_initsend(encode); pvm_pkdouble( arrayNULL ); pvm_send(tids,TAG);
        pvm_freebuf(pvm_getsbuf);
        pvm_recv(tids,TAG);  pvm_upkdouble( arrayNULL );
        pvm_freebuf(pvm_getrbuf);
  end
  clear arrayNULL, arrayNULL=[];
  fprintf('\n...hecho\n');
  unix('vmstat');
```

Listado D.4: pvmCode/BW.m: Proceso fuente en PVMTB. Se mostró un fragmento en el Listado 3.23.

08/01/01 12:07:32 pvmCode/BW.m 2

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BUCLE PING-PONG %
%%%%%%%%%%%%%%%%%%%

  for ExpAncho=0:12
    Ancho=2^ExpAncho;
    fprintf('\n(%4d\t',Ancho);

    for Alto=Ancho:Ancho:3*Ancho
      fprintf('x%4d) ',Alto);
      indx=indx+1;

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if(indx>=Dstr & indx<=Dend) |...                  % algo que hacer
        (indx>=Ustr & indx<=Uend)

        NTIMES=ntimes*reps(0);
        fprintf('[');                                   % hacer parte latencia
if PACK
        pvm_barrier('BW',2); T=clock; for i=1:NTIMES
          pvm_initsend(encode);     pvm_pack( arrayNULL ); pvm_send(tids,TAG);
          pvm_recv(tids,TAG);     pvm_unpack('arrayNULL');
        end, T=etime(clock,T);
else
        pvm_barrier('BW',2); T=clock; for i=1:NTIMES
          pvm_initsend(encode); pvm_pkdouble( arrayNULL ); pvm_send(tids,TAG);
          pvm_recv(tids,TAG);  pvm_upkdouble( arrayNULL );
        end, T=etime(clock,T);
end
        l(indx)=T/2/NTIMES;
      end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if indx>=Dstr & indx<=Dend                        % hacer parte double

        NTIMES = ntimes*reps(Alto*Ancho*sizeofDbl);     % medir reserva memoria
        T=clock; for i=1:NTIMES
          clear arrayDouble
          arrayDouble(Alto,Ancho)=0;                    % liberar al final
        end, T=etime(clock,T); s=whos('arrayDouble');
        d(indx,Size)=s.bytes;
        d(indx, Mtm)=T/NTIMES;

        T=clock; for i=1:NTIMES                         % medir cambio de tipo
          arrayUint8=uint8(arrayDouble);
          clear arrayUint8
        end, T=etime(clock,T);
        d(indx, Ctm)=T/NTIMES;

        fprintf('d');                                   % ping-pong parte double
if PACK
        pvm_barrier('BW',2); T=clock; for i=1:NTIMES
          pvm_initsend(encode);     pvm_pack( arrayDouble ); pvm_send(tids,TAG);
                if indx>SWAPLIM,  pvm_freebuf(pvm_getsbuf ); end
          pvm_recv(tids,TAG);     pvm_unpack('arrayDouble');
                if indx>SWAPLIM,  pvm_freebuf(pvm_getrbuf ); end
        end, T=etime(clock,T);
else
        pvm_barrier('BW',2); T=clock; for i=1:NTIMES
          pvm_initsend(encode); pvm_pkdouble( arrayDouble ); pvm_send(tids,TAG);
                if indx>SWAPLIM,  pvm_freebuf(pvm_getsbuf ); end
          pvm_recv(tids,TAG);  pvm_upkdouble( arrayDouble );
                if indx>SWAPLIM,  pvm_freebuf(pvm_getrbuf ); end
        end, T=etime(clock,T);
end
        d(indx,TXtm)=T/2/NTIMES;

        clear arrayDouble,      pvm_freebuf(pvm_getsbuf);
                                pvm_freebuf(pvm_getrbuf);
                fprintf(',');

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      else,     fprintf('-,');
      end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if indx>=Ustr & indx<=Uend                        % hacer parte uint8

        NTIMES = ntimes*reps(Alto*Ancho*sizeofU8);      % medir reserva memoria
        T=clock; for i=1:NTIMES
          clear arrayUint8
          arrayUint8(Alto,Ancho)=uint8(0);              % liberar al final
        end, T=etime(clock,T); s=whos('arrayUint8');
        u(indx,Size)=s.bytes;
        u(indx, Mtm)=T/NTIMES;

        fprintf('u');                                   % ping-pong parte uint8
if PACK
        pvm_barrier('BW',2); T=clock; for i=1:NTIMES
          pvm_initsend(encode);   pvm_pack( arrayUint8 ); pvm_send(tids,TAG);
                if indx>SWAPLIM,pvm_freebuf(pvm_getsbuf); end
          pvm_recv(tids,TAG);   pvm_unpack('arrayUint8');
                if indx>SWAPLIM,pvm_freebuf(pvm_getrbuf); end
        end, T=etime(clock,T);
else
        pvm_barrier('BW',2); T=clock; for i=1:NTIMES
          pvm_initsend(encode); pvm_pkbyte( arrayUint8 ); pvm_send(tids,TAG);
                if indx>SWAPLIM,pvm_freebuf(pvm_getsbuf); end
          pvm_recv(tids,TAG);  pvm_upkbyte( arrayUint8 );
                if indx>SWAPLIM,pvm_freebuf(pvm_getrbuf); end
        end, T=etime(clock,T);
end
        u(indx,TXtm)=T/2/NTIMES;

        clear arrayUint8,       pvm_freebuf(pvm_getsbuf);
                                pvm_freebuf(pvm_getrbuf);
                fprintf(']\t');

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      else,     fprintf('-]\t');
      end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    end
  end
  fprintf('\n');

  pvm_lvgroup('BW');
```

Listado D.4: Continuación.

08/01/01 12:07:32 pvmCode/BW.m **3**

```
% pvm_exit;
            pvm_setopt(3,0);                    % AutoErr Ignore para halt
  pvm_halt; pvm_setopt(3,1);                    % AutoErr Warn otra vez

toc

  unix('vmstat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Salvar datos %
%%%%%%%%%%%%%%%%
  fname=[SUB '/' FN '-' HN0 '-' HN1];
  switch encode
    case 0, fname=[fname '-def'];
    case 1, fname=[fname '-raw'];
    case 2, fname=[fname '-pla'];
  end
  switch droute
    case 0, fname=[fname '-no'];
    case 1, fname=[fname '-di'];
  end
  save(fname, 'Ustr', 'Uend', 'Dstr', 'Dend', 'ROWS', 'COLS')
  save(fname,'l','u','d','-append')
```

Listado D.4: Continuación.

07/31/01
13:43:04

pvmCode/BWSlav.m

1

```
function BWSlav(encode,droute,row,ntimes)
% BWSlav: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       BWSlav ( encode, droute, row, ntimes )
%
%  ver el help para BW
%

  SWAPLIM=30; PACK=0; global TAG ROU DIR DONT
  reps=inline('2.^fix((30-fix(log2(max(s,1))))./3.33-1)');
  varDbl=      0 ; s=whos('varDbl'); sizeofDbl=s.bytes; % para cuentas tamaños
  varU8 =uint8(0); s=whos('varU8' ); sizeofU8 =s.bytes;
  clear s varDbl varU8

%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
  if nargin<4, return, end
  indx=0; Ustr=1; Uend=38; Dstr=1; Dend=34;             % filas
                  Uend=row+4;      Dend=row;

  if droute, pvm_setopt(ROU,DIR);                       % ruta directa?
       else, pvm_setopt(ROU,DONT);
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Comunicación con padre %
%%%%%%%%%%%%%%%%%%%%%%%%%%
  parent=pvm_parent;                                    % Para recibir
  pvm_joingroup  ('BW'  );                              % Para barreras
  pvm_freezegroup('BW',2);

  arrayNULL(2048,2048)=0;              % ejercitar posible ruta directa/swap
if PACK,pvm_recv(parent,TAG);    pvm_unpack('arrayNULL');
        pvm_freebuf(pvm_getrbuf);
        pvm_initsend(encode);      pvm_pack( arrayNULL ); pvm_send(parent,TAG);
        pvm_freebuf(pvm_getsbuf);
  else, pvm_recv(parent,TAG); pvm_upkdouble( arrayNULL );
        pvm_freebuf(pvm_getrbuf);
        pvm_initsend(encode);  pvm_pkdouble( arrayNULL ); pvm_send(parent,TAG);
        pvm_freebuf(pvm_getsbuf);
  end
  clear arrayNULL, arrayNULL=[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BUCLE PING-PONG %
%%%%%%%%%%%%%%%%%%%
  for ExpAncho=0:12
    Ancho=2^ExpAncho;
    for Alto=Ancho:Ancho:3*Ancho
      indx=indx+1;

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if(indx>=Dstr & indx<=Dend)|...                   % algo que hacer
        (indx>=Ustr & indx<=Uend)

        NTIMES=ntimes*reps(0);                          % hacer parte latencia
if PACK
        pvm_barrier('BW',2); for i=1:NTIMES
          pvm_recv(parent,TAG);   pvm_unpack('arrayNULL');
          pvm_initsend(encode);     pvm_pack( arrayNULL ); pvm_send(parent,TAG);
        end
else
        pvm_barrier('BW',2); for i=1:NTIMES
          pvm_recv(parent,TAG);pvm_upkdouble( arrayNULL );
          pvm_initsend(encode); pvm_pkdouble( arrayNULL ); pvm_send(parent,TAG);
        end
end
      end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if indx>=Dstr & indx<=Dend                        % hacer parte double

        NTIMES = ntimes*reps(Alto*Ancho*sizeofDbl);
        arrayDouble(Alto,Ancho)=0;                      % preallocation
if PACK
        pvm_barrier('BW',2); for i=1:NTIMES
          pvm_recv(parent,TAG);   pvm_unpack('arrayDouble');
                if indx>SWAPLIM,  pvm_freebuf(pvm_getrbuf ); end
          pvm_initsend(encode);     pvm_pack( arrayDouble);pvm_send(parent,TAG);
                if indx>SWAPLIM,  pvm_freebuf(pvm_getsbuf ); end
        end
else
        pvm_barrier('BW',2); for i=1:NTIMES
          pvm_recv(parent,TAG);pvm_upkdouble( arrayDouble);
                if indx>SWAPLIM, pvm_freebuf(pvm_getrbuf ); end
          pvm_initsend(encode); pvm_pkdouble( arrayDouble);pvm_send(parent,TAG);
                if indx>SWAPLIM, pvm_freebuf(pvm_getsbuf ); end
        end
end
        clear arrayDouble,      pvm_freebuf(pvm_getrbuf);
                                pvm_freebuf(pvm_getsbuf);
      end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if indx>=Ustr & indx<=Uend                        % hacer parte uint8

        NTIMES = ntimes*reps(Alto*Ancho*sizeofU8);
        arrayUint8(Alto,Ancho)=uint8(0);                % preallocation
if PACK
        pvm_barrier('BW',2); for i=1:NTIMES
          pvm_recv(parent,TAG); pvm_unpack('arrayUint8');
                if indx>SWAPLIM,pvm_freebuf(pvm_getrbuf); end
          pvm_initsend(encode);   pvm_pack( arrayUint8 ); pvm_send(parent,TAG);
                if indx>SWAPLIM,pvm_freebuf(pvm_getsbuf); end
        end
else
        pvm_barrier('BW',2); for i=1:NTIMES
          pvm_recv(parent,TAG);pvm_upkbyte( arrayUint8 );
                if indx>SWAPLIM,pvm_freebuf(pvm_getrbuf); end
          pvm_initsend(encode); pvm_pkbyte( arrayUint8 ); pvm_send(parent,TAG);
                if indx>SWAPLIM,pvm_freebuf(pvm_getsbuf); end
        end
end
        clear arrayUint8,       pvm_freebuf(pvm_getrbuf);
                                pvm_freebuf(pvm_getsbuf);
      end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    end
  end
```

Listado D.5: pvmCode/BWSlav.m: Proceso eco en PVMTB.

07/31/01 13:43:04 pvmCode/BWSlav.m **2**

```
pvm_lvgroup('BW');
pvm_exit;
```

Listado D.5: Continuación.

07/25/01 20:44:45 mpiCode/mpiGenFiles **1**

```
#!/bin/bash

############################################################################
if [ $# -eq 0 ]; then
echo
echo Usage: ./mpiGenFiles SUB [ HMG [ C2C [ ROW [ N [ FN HN0 HN1]]]]]
echo SUB••••••••: subdirectorio en donde guardar fichero mediciones
echo HMG•••••[1]: opción mpirun cluster homogéneo: no \(0\) / sí \(1\)
echo C2C•••••[1]: opción mpirun cliente a cliente: no \(0\) / sí \(1\)
echo ROW••••[34]: barrido hasta esta fila del array doubles
echo •N••••••[1]: factor repeticiones para 32MB -\> más pequeño, más veces
echo FN•••••[BW]: nombre base fichero mediciones BW-hn0-hn1-\<hmg\>-\<c2c\>.dat
echo HN0•••[ox0]: nombre 1er ordenador \(fuente\)
echo HN1•••[ox1]: nombre 2°  ordenador \(eco\)
echo
exit
fi
############################################################################
SUB=${1:-.}      # el valor por defecto no se usa, se ejecutaría el Usage antes
HMG=${2:-1};     case $HMG in 0) HOMOG=_  ;; 1) HOMOG=O  ;; esac
C2C=${3:-1};     case $C2C in 0) CLINT=___;; 1) CLINT=c2c;; esac
ROW=${4:-34};
  N=${5:-1}
 FN=${6:-BW}
HN0=${7:-ox0}
HN1=${8:-ox1}    FN=${FN}-${HN0}-${HN1}-${HOMOG}-${CLINT}.dat

# MasterName, SlaveName, FileName
MN=BW
SN=BWSlav

#########################

OPTS="-nger -toff -nsigs"
if [ $HMG -eq 1 ]; then OPTS="   -O $OPTS"; fi
if [ $C2C -eq 1 ]; then OPTS=" -c2c $OPTS";
                   else OPTS="-lamd $OPTS"; fi

############################################################################
# mpirun activa por defecto los siguientes RunTimeFlags
# TROLLIUSRTF == 0x139292 == RTF_KENYA | RTF_WAIT | RTF_MPIC2C | RTF_TRSWITCH |
#                 RTF_MPIRUN | RTF_IO | RTF_PFDIO | RTF_TTYOUT | RTF_MPISIGS
# -nw desactiva RTF_WAIT
# -lamd desactiva RTF_MPIC2C y activa RTF_MPIGER --> muy malo prestaciones GER
# -toff desactiva RTF_TRSWITCH pero activa RTF_TRACE (-ton activa los dos)
# -f creo que desactiva RTF_[PFD]IO y/o RTF_TTYOUT ???
# -nsigs desactiva RTF_MPISIGS
# RTF_MPIGER se desactiva en mpiGenFiles para comparar prestaciones con MPITB
#
############################################################################
# Crear hostfile y schema. Matar posible LAM anterior
############################################################################

cat >   bhost.def <<-@
        $HN0
        $HN1
        @
lamboot bhost.def
cat >   schema <<-@
        n0 $MN $HMG $C2C $ROW $N $SUB/$FN
        n1 $SN $HMG $C2C $ROW $N
        @
############################################################################
# Usage: ./BW hmg c2c row ntimes fname
#       BWSlav hmg c2c row ntimes
############################################################################
# Arrancar esclavo (remoto) y maestro (local), bloquearse (por defecto)
# y matar LAM???
############################################################################
vmstat
mpirun -w $OPTS schema
vmstat
rm bhost.def schema
```

Listado D.6: mpiCode/mpiGenFiles: *Script* de automatización para el test *ping-pong* en MPI.

07/25/01 21:13:25 — mpiCode/BW.c — 1

```
/****************************************************************
% BW: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       [l u d] = BW ( hmg, c2c, row, ntimes, fname )
%
%  hmg   0/1    flag para cluster homogéneo (mpirun _____/ -O )
%  c2c   0/1    flag para client-to-client  (mpirun -lamd/-c2c)
%  row 1..34    fila de la columna double hasta donde se desea llegar
%               Los uint8 llegan hasta row+4  [34+4=38 -> 32MB]
%               Debe ser de la forma row=3n+1 [34=3*11+1]
%                si se desea que coincidan tamaños double/uint8 última medición
%  ntimes 1..?  número de veces a repetir transmisión 32MB
%               s bytes -> 2^fix((30-fix(log2(s)))/3.33-1)*ntimes veces
%               ej: ntimes=1, 0-1B 256 veces, 2-15B 128 veces, 16-64B 64 veces,
%               128-1KB 32 veces, 2-15KB 16 veces, 16-64KB 8 veces,
%               128KB-1MB 4 veces, 2-15MB 2 veces, 16-64MB 1 vez.
%  fname        nombre del fichero donde salvar las mediciones:
%
%  l            Latencias (vector columna)
%  u            Datos para uint8   (3 col)
%  d            Datos para doubles (3 col)
%
%       Los datos son arrays de 3 Columnas: [Size TXtm Mtm]
%               Size: tamaño del array
%               TXtm: tiempo de transmisión
%               Mtm:  tiempo para crear array en memoria
%       Filas: Tamaños de array crecientes (ver BWrows.m)
%               fila    dimensiones     tamaño uint8    tamaño double
%                1         1x   8       2^ 0 =  1 B     2^ 3 =   8 B
%                2         1x  16       2^ 1 =  2 B     2^ 4 =  16 B
%                3         1x  24               3 B             24 B
%                4         2x   8       2^ 2    4 B     2^ 5 =  32 B
%                5         2x  16       2^ 3    8 B     2^ 6 =  64 B
%                6         2x  24              12 B             96 B
%                7         4x   8       2^ 4   16 B     2^ 7 = 128 B
%              ...
%               34      2048x2048       2^22 =  4MB     2^25 =  32MB
%               35      2048x4096       2^23 =  8MB     2^26
%               36      2048x6144              12MB
%               37      4096x4096       2^24 = 16MB     2^27
%               38      4096x8192       2^25 = 32MB     2^28
 ***************************************************************/

#include <stdio.h>      /* printf, fflush */
#include <stdlib.h>     /* atoi, getenv, calloc */
#include <mpi.h>        /* MPI_* */
#include <math.h>       /* floor */

#include <sys/stat.h>   /* para mkdir, creat */
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>     /* para write, close */

#define USTR 0
#define UEND 37
#define ROWS 38
#define DSTR 0
#define DEND 33
#define Size 0
#define TXtm 1
#define Mtm  2
#define COLS 3
```

```
#define TAG  7
#define RTFHMG 0x2000
#define RTFC2C 0x0080


/**************/
int reps(int s){                        /* 2^fix((30-fix(log2(s)))/3.33-1) */
/**************/
int log2s,pwr;
        for(log2s=0; s>>=1; log2s++);
        pwr=floor((30-log2s)/3.33-1);
        return(1<<pwr);
}

/************/
/* PROGRAMA */
/************/
int main(int argc, char*argv[]){
        int hmg,c2c,row,ntimes,NTIMES;          /* argumentos del programa */
        char*fname;
        int  fd,TRTF,bool1,bool2;               /* file descriptor y flags */
        double var;                             /* Variable para write */

        int Dstr,Dend,Ustr,Uend;                /* índices medición */
        double       *arrayDouble;              /* arrays de datos */
        unsigned char*arrayUint8;
        double  l[ROWS];                        /* Latencias */
        double  u[ROWS][COLS];                  /* arrays de mediciones */
        double  d[ROWS][COLS];

        int ExpAncho,Ancho,Alto,i,indx=-1;      /* índices de bucles */
        double T,Tot;                           /* Variables para tiempo */

        int rank,size,nelem;                    /* rangos etc */
        MPI_Status ST;                          /* MPI_Recv */
        char lat=' ';

/****************************************************************/

        Tot = MPI_Wtime();                      /* como tic-toc matlab */
  /**************/
  /* Argumentos */
  /**************/
        if (argc<6) {
    printf("\n"
        "Usage: ./BW hmg c2c row ntimes fname\n"
        "    hmg: 0/1   cluster (no) homogéneo (_/-O)\n"
        "    c2c: 0/1   ruta daemon/cliente a cliente (-lamd/-c2c)\n"
        "    row: 1..34 fila de la columna double hasta donde se desea llegar\n"
        " ntimes: 1..?  número de veces a repetir transmisión 32MB\n"
        "  fname:       nombre del fichero donde salvar las mediciones\n"
        "\n");
    exit(1);
  }
           hmg = atoi(argv[1]);                 /* homogeneidad */
           c2c = atoi(argv[2]);                 /* encaminamiento */
           row = atoi(argv[3]);                 /* hasta qué fila barrer */
        ntimes = atoi(argv[4]);                 /* factor repeticiones */
         fname =      argv[5] ;                 /* nombre fichero mediciones */

  TRTF=atoi(getenv("TROLLIUSRTF"));
  bool1=(hmg!=0); bool2=((TRTF & RTFHMG)!=0);
  if (bool1^bool2){
```

Listado D.7: mpiCode/BW.c: Proceso fuente en MPI. Se mostró un fragmento en el Listado 4.7.

07/25/01 21:13:25 mpiCode/BW.c 2

```
    printf(" no coincide mpirun (-O) con argumento BW hmg\n");
    exit(2);
  }
  bool1=(c2c!=0); bool2=((TRTF & RTFC2C)!=0);
  if (bool1^bool2){
    printf(" no coincide mpirun (-c2c/-lamd) con argumento BW c2c\n");
    exit(2);
  }
  if(--row%3) printf("no coincidirán tamaños finales double/uint8\n");
              Dstr=     Ustr=     0;
              Dend=row; Uend=Dend+4;

  /******************************/
  /* por similitud a código PVM */ /* obtener tid del otro */
  /******************************/

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  if (size<2){
    printf("hacen falta 2 ordenadores en LAM\n");
    exit(3);
  }
  if (rank){
    printf("yo debería ser rango 0, y el otro 1\n");
    exit(4);
  }

/* mandar algo, ver posible swap */
  printf("\nTransmitiendo 32MB (posible swapping)..."); fflush(stdout);
  nelem=2048*2048;
  arrayDouble=calloc(nelem,sizeof(double));
  MPI_Send(arrayDouble,nelem,MPI_DOUBLE, 1,TAG,MPI_COMM_WORLD);
  MPI_Recv(arrayDouble,nelem,MPI_DOUBLE, 1,TAG,MPI_COMM_WORLD, &ST);
  free(arrayDouble);
  printf("\n...hecho\n");

/*****************
% BUCLE PING-PONG %
 *****************/
  for (ExpAncho=0; ExpAncho<13; ExpAncho++){
    Ancho=1<<ExpAncho;
    printf("\n(%4d\t",Ancho);

    for (Alto=Ancho; Alto<4*Ancho; Alto+=Ancho){
      printf("x%4d) ",Alto);
      indx++; nelem=Alto*Ancho;

        /**************************************************************/

      if(((indx>=Dstr)&&(indx<=Dend))||         /* algo que hacer */
          ((indx>=Ustr)&&(indx<=Uend))){

        NTIMES=ntimes*reps(0);
        printf("["); fflush(stdout);
        MPI_Barrier(MPI_COMM_WORLD);            /* hacer parte latencia */

        T=MPI_Wtime();
        for (i=0; i<NTIMES; i++){
          MPI_Send(&lat,0,MPI_CHAR, 1,TAG,MPI_COMM_WORLD     );
          MPI_Recv(&lat,0,MPI_CHAR, 1,TAG,MPI_COMM_WORLD, &ST);
        }
        T=MPI_Wtime()-T;                l[indx]=T/2/NTIMES;
      }

        /**************************************************************/

      if ((indx>=Dstr)&&(indx<=Dend)){          /* hacer parte double */

        NTIMES=ntimes*reps(nelem*sizeof(double)); arrayDouble=NULL;
        T=MPI_Wtime();
        for (i=0; i<NTIMES; i++){               /* medir reserva memoria */
          free(arrayDouble);
          arrayDouble=calloc(nelem,sizeof(double));
        }
        T=MPI_Wtime()-T;        d[indx][Size]=nelem*sizeof(double);
                                d[indx][ Mtm]=T/NTIMES;

        printf("d"); fflush(stdout);            /* ping-pong parte double */
        MPI_Barrier(MPI_COMM_WORLD);

        T=MPI_Wtime();
        for (i=0; i<NTIMES; i++){
          MPI_Send(arrayDouble,nelem,MPI_DOUBLE, 1,TAG,MPI_COMM_WORLD     );
          MPI_Recv(arrayDouble,nelem,MPI_DOUBLE, 1,TAG,MPI_COMM_WORLD, &ST);
        }
        T=MPI_Wtime()-T;        d[indx][TXtm]=T/2/NTIMES;

        free(arrayDouble);

                printf(",");

                /******************************************************/

      } else    printf("-,");
                fflush(stdout);

        /**************************************************************/

      if ((indx>=Ustr)&&(indx<=Uend)){          /* hacer parte uint8 */

        NTIMES=ntimes*reps(nelem*sizeof(unsigned char)); arrayUint8=NULL;
        T=MPI_Wtime();
        for (i=0; i<NTIMES; i++){
          free(arrayUint8);
          arrayUint8=calloc(nelem,sizeof(unsigned char));
        }
        T=MPI_Wtime()-T;        u[indx][Size]=nelem*sizeof(unsigned char);
                                u[indx][ Mtm]=T/NTIMES;

        printf("u"); fflush(stdout);            /* ping-pong parte char */
        MPI_Barrier(MPI_COMM_WORLD);

        T=MPI_Wtime();
        for (i=0; i<NTIMES; i++){
          MPI_Send(arrayUint8,nelem,MPI_BYTE, 1,TAG,MPI_COMM_WORLD     );
          MPI_Recv(arrayUint8,nelem,MPI_BYTE, 1,TAG,MPI_COMM_WORLD, &ST);
        }
        T=MPI_Wtime()-T;        u[indx][TXtm]=T/2/NTIMES;

        free(arrayUint8);

                printf("]\t");

                /******************************************************/
```

Listado D.7: Continuación.

07/25/01 21:13:25 mpiCode/BW.c 3

```
      } else    printf("-]\t");
                fflush(stdout);

/*****************************************************************/

    }
  }
  printf("\n");

  MPI_Finalize();

  Tot=MPI_Wtime()-Tot;
  printf("Elapsed time: %8.3f s\n",Tot);

/****************************************************************
% Salvar Datos
 ****************************************************************/

  if ((fd=creat(fname,S_IRWXU))<0){
        printf("no puedo crear fichero %s\n",fname);
        exit(15);
  }
  var=Ustr+1;   write(fd,&var,sizeof(var));     /* índices para MATLAB */
  var=Uend+1;   write(fd,&var,sizeof(var));
  var=Dstr+1;   write(fd,&var,sizeof(var));
  var=Dend+1;   write(fd,&var,sizeof(var));
  var=ROWS;     write(fd,&var,sizeof(var));     /* Tamaño datos */
  var=COLS;     write(fd,&var,sizeof(var));
                write(fd,    l,sizeof(l ));     /* datos */
                write(fd,    u,sizeof(u ));
                write(fd,    d,sizeof(d ));
  if (close(fd)){
        printf("no puedo cerrar fichero %s\n",fname);
        exit(16);
  }

  exit(0);

}
```

Listado D.7: Continuación.

07/25/01 20:37:54 — mpiCode/BWSlav.c — 1

```
/****************************************************************
% BWSlav: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       BWSlav ( hmg, c2c, row, ntimes )
%
 ****************************************************************/

#include <stdlib.h>     /* atoi, getenv, calloc */
#include <mpi.h>        /* MPI_* */
#include <math.h>       /* floor */

#define TAG 7
#define RTFHMG 0x2000
#define RTFC2C 0x0080

/**************/
int reps(int s){                                /* 2^fix((30-log2(s))/3.33-1) */
/**************/
int log2s,pwr;
        for(log2s=0; s>>=1; log2s++);
        pwr=floor((30-log2s)/3.33-1);
        return(1<<pwr);
}

/************/
/* PROGRAMA */
/************/
int main(int argc,char*argv[]){
        int hmg,c2c,row,ntimes,NTIMES;          /* argumentos */
        int TRTF,bool1,bool2;                   /* RunTimeFlags */
        int Dstr,Dend,Ustr,Uend;                /* índices medición */
        double       *arrayDouble;              /* arrays de datos */
        unsigned char*arrayUint8;

        int ExpAncho,Ancho,Alto,i,indx=-1;      /* índices de bucles */
        int rank,size,nelem;                    /* rangos etc */
        MPI_Status ST;                          /* MPI_Recv */
        char lat=' ';

  /**************/
  /* Argumentos */
  /**************/
  if (argc<5) exit(1);
           hmg = atoi(argv[1]);                 /* no se usa */
           c2c = atoi(argv[2]);                 /* no se usa */
           row = atoi(argv[3]);                 /* hasta qué fila barrer */
        ntimes = atoi(argv[4]);                 /* factor repeticiones */

        Dstr=       Ustr=     0;
        Dend=--row; Uend=Dend+4;

  TRTF=atoi(getenv("TROLLIUSRTF"));
  bool1=(hmg!=0); bool2=((TRTF & RTFHMG)!=0); if (bool1^bool2) exit(2);
  bool1=(c2c!=0); bool2=((TRTF & RTFC2C)!=0); if (bool1^bool2) exit(2);

  /******************************/
  /* por similitud a código PVM */ /* Obtener tid del otro */
  /******************************/
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  if ((rank!=1) || (size!=2)) exit(3);          /* debe ser rango 1 */
/* recibir algo, por ver posible swap */
  nelem=2048*2048;
  arrayDouble=calloc(nelem,sizeof(double));
  MPI_Recv(arrayDouble,nelem,MPI_DOUBLE, 0,TAG,MPI_COMM_WORLD, &ST);
  MPI_Send(arrayDouble,nelem,MPI_DOUBLE, 0,TAG,MPI_COMM_WORLD);
  free(arrayDouble);

  /********/
  /* Pong */
  /********/
  for (ExpAncho=0; ExpAncho<13; ExpAncho++){
    Ancho=1<<ExpAncho;
    for (Alto=Ancho; Alto<4*Ancho; Alto+=Ancho){
      indx++; nelem=Ancho*Alto;

        /************************************************************/

      if(((indx>=Dstr) && (indx<=Dend))||       /* algo que hacer */
         ((indx>=Ustr) && (indx<=Uend))){

        NTIMES=ntimes*reps(0);
        MPI_Barrier(MPI_COMM_WORLD);            /* repetir parte latencia */
        for (i=0; i<NTIMES; i++){
          MPI_Recv(&lat,0,MPI_CHAR, 0,TAG,MPI_COMM_WORLD, &ST);
          MPI_Send(&lat,0,MPI_CHAR, 0,TAG,MPI_COMM_WORLD     );
        }
      }

        /************************************************************/

      if ((indx>=Dstr) && (indx<=Dend)){        /* hacer parte double */

        NTIMES=ntimes*reps(nelem*sizeof(double));
        arrayDouble=calloc(nelem,sizeof(double));
        MPI_Barrier(MPI_COMM_WORLD);
        for (i=0; i<NTIMES; i++){
          MPI_Recv(arrayDouble,nelem,MPI_DOUBLE, 0,TAG,MPI_COMM_WORLD, &ST);
          MPI_Send(arrayDouble,nelem,MPI_DOUBLE, 0,TAG,MPI_COMM_WORLD     );
        }
        free(arrayDouble);
      }

        /************************************************************/

      if ((indx>=Ustr) && (indx<=Uend)){        /* hacer parte uint8 */

        NTIMES=ntimes*reps(nelem*sizeof(unsigned char));
        arrayUint8 =calloc(nelem,sizeof(unsigned char));
        MPI_Barrier(MPI_COMM_WORLD);
        for (i=0; i<NTIMES; i++){
          MPI_Recv(arrayUint8,nelem,MPI_BYTE, 0,TAG,MPI_COMM_WORLD, &ST);
          MPI_Send(arrayUint8,nelem,MPI_BYTE, 0,TAG,MPI_COMM_WORLD     );
        }
        free(arrayUint8);
      }

        /************************************************************/
    }   /* for Alto */
  }     /* for ExpAncho */

  MPI_Finalize();
  exit(0);
}
```

Listado D.8: mpiCode/BWSlav.c: Proceso eco en MPI.

07/26/01 09:25:40 — mpiCode/BW.m — 1

```
function BW(SUB,hmg,c2c,row,ntimes,FN,HN0,HN1)
% BW: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       BW ( SUB [,hmg [,c2c [,row [,ntimes [,FN,HN0,HN1]]]]] )
%
%  hmg   0/1    flag para cluster homogéneo (mpirun _____/ -O )
%  c2c   0/1    flag para client-to-client  (mpirun -lamd/-c2c)
%  row          [34], fila de la columna double hasta donde se desea llegar
%               Los uint8 llegan hasta row+4  [34+4=38 -> 32MB]
%               Debe ser de la forma row=3n+1 [34=3*11+1]
%                si se desea que coincidan tamaños double/uint8 última medición
%  ntimes   [1] número de veces a repetir transmisión 32MB
%               s bytes -> 2^fix((30-fix(log2(s)))/3.33-1)*ntimes veces
%               ej: para ntimes=1,
%                 0-1 B 256 veces, 2-15B 128 veces, 16-64B 64 veces,
%               128 -1KB 32 veces, 2-15KB 16 veces, 16-64KB 8 veces,
%               128KB-1MB 4 veces, 2-15MB  2 veces, 16-64MB 1 vez.
%  FN      [BW] nombre base fichero mediciones FN-HN0-HN1-<cod>-<rou>.mat
%  HN0    [ox0] nombre del 1er computador (fuente)
%  HN1    [ox1] nombre del 2°  computador (eco)
%
%               En el fichero se salvan:
%       Ustr,Uend, Dstr,Dend    ej: 1,34,1,38
%       ROWS,COLS               ej:        38,4
%       l       Latencias       (vector columna)
%       u       Datos para uint8        (3 col)
%       d       Datos para doubles      (4 col)
%
%       u/d son arrays de 3/4 Columnas: [Size TXtm Mtm Ctm]
%               Size: tamaño del array
%               TXtm:   tiempo de transmisión
%               Mtm:    tiempo para crear array en memoria
%               Ctm:    tiempo para pasar double->uint8 (typecast)
%       Filas: Tamaños de array crecientes (help BWrows)
%

tic
  DEBUG=0; global TAG                                   % heredada de startup
  reps=inline('2.^fix((30-fix(log2(max(s,1))))./3.33-1)');
  varDbl=      0 ; s=whos('varDbl'); sizeofDbl=s.bytes; % para cuentas tamaños
  varU8 =uint8(0); s=whos('varU8' ); sizeofU8 =s.bytes;
  clear s varDbl varU8

%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
  if nargin<1, help(mfilename), return, end
  if nargin<2, hmg   = 1 ; end
  if nargin<3, c2c   = 1 ; end
  if nargin<4, row   =34 ; end
  if nargin<5, ntimes= 1 ; end
  if nargin<6, FN   ='BW'; end
  if nargin<7, HN0 ='ox0'; end
  if nargin<8, HN1 ='ox1'; end
  flag=0;
  flag=flag | ~exist(SUB,'dir');
  flag=flag | fix(hmg)   ~=hmg    | hmg   <0 | hmg >  2;HMG=int2str(hmg);
  flag=flag | fix(c2c)   ~=c2c    | c2c   <0 | c2c >  1;C2C=int2str(c2c);
  flag=flag | fix(row)   ~=row    | row   <1 | row > 34;ROW=int2str(row);
  flag=flag | fix(ntimes)~=ntimes | ntimes<1;           NTM=int2str(ntimes);
  flag=flag | ~ischar(FN) | ~ischar(HN0) | ~ischar(HN1);
  if flag, help(mfilename), return, end

  if mod(row,3)~=1, warning('no coincidirán tamaños finales double/uint8'), end
  Size=1; TXtm=2; Mtm = 3; Ctm =4;                      % columnas
  indx=0; Ustr=1; Uend=38; Dstr=1; Dend=34;             % filas
                  Uend=row+4;      Dend=row;
                  ROWS=Uend;       COLS=Ctm;
  l=zeros(Uend,1);
  u=zeros(Uend,Mtm); d=zeros(Dend,Ctm);

%%%%%%%%%%%%%%%%%%%%%%%
% Arrancar daemon LAM %
%%%%%%%%%%%%%%%%%%%%%%%
  fid=fopen('bhost','w'); fprintf(fid,'%s\n',HN1,HN0); fclose(fid);
  unix('lamboot bhost');

  TRTF = hmg*RTF_HOMOG + c2c*RTF_MPIC2C;                % ajustar hmg/c2c
  putenv(['TROLLIUSRTF=' int2str(TRTF)])
  MPI_Init;

%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arranque MATLAB esclavo % El que responde al ping
%%%%%%%%%%%%%%%%%%%%%%%%%%%
  if DEBUG
    DISP=getenv('DISPLAY');
    [info children errs]=MPI_Comm_spawn('xterm',...
                {'-display' DISP '-e' 'matlab'},1,'NULL',0,'SELF');
  else
    [info children errs]=MPI_Comm_spawn('matlab',...
                                  {'-nosplash'},1,'NULL',0,'SELF');
  end
  if info | any(errs),  error('no puedo arrancar MATLAB esclavo'), end
  [info numt] = MPI_Comm_remote_size(children);
  if info | numt~=1,    error('no puedo arrancar MATLAB esclavo'), end
  [info NEWORLD]=MPI_Intercomm_merge(children,0);
  if info,              error('no puedo arrancar MATLAB esclavo'), end

  if DEBUG
    MPI_Errhandler_set(NEWORLD,MPI_ERRORS_RETURN);
  end

%%%%%%%%%%%
% COMANDO %
%%%%%%%%%%%
  cmd=['BWSlav(' HMG ',' C2C ',' ROW ',' NTM ');'];
  info=MPI_Bcast(cmd,0,NEWORLD);

  unix('vmstat');
  fprintf('\nTransmitiendo 32MB (posible swapping)...');
  arrayNULL(2048,2048)=0;                % enviar algo, por mirar swap
  MPI_Send(arrayNULL,1,TAG,NEWORLD);
  MPI_Recv(arrayNULL,1,TAG,NEWORLD);
  clear arrayNULL, arrayNULL=[];
  fprintf('\n...hecho\n');
  unix('vmstat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BUCLE PING-PONG %
%%%%%%%%%%%%%%%%%%%

  for ExpAncho=0:12
    Ancho=2^ExpAncho;
    fprintf('\n(%4d\t',Ancho);

    for Alto=Ancho:Ancho:3*Ancho
```

Listado D.9: mpiCode/BW.m: Proceso fuente en MPITB. Se mostró un fragmento en el Listado 4.9.

07/26/01 09:25:40 mpiCode/BW.m **2**

```
fprintf('x%4d) ',Alto);
indx=indx+1;

          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(indx>=Dstr & indx<=Dend) |...                  % algo que hacer
  (indx>=Ustr & indx<=Uend)

  NTIMES=ntimes*reps(0);
  fprintf('[');                                   % hacer parte latencia
  MPI_Barrier(NEWORLD); T=clock; for i=1:NTIMES
    MPI_Send(arrayNULL,1,TAG,NEWORLD);
    MPI_Recv(arrayNULL,1,TAG,NEWORLD);
  end, T=etime(clock,T);
  l(indx)=T/2/NTIMES;
end

          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if indx>=Dstr & indx<=Dend                        % hacer parte double

  NTIMES = ntimes*reps(Alto*Ancho*sizeofDbl);     % medir reserva memoria
  T=clock; for i=1:NTIMES
    clear arrayDouble
    arrayDouble=ones(Alto,Ancho);                 % liberar al final
  end, T=etime(clock,T); s=whos('arrayDouble');
  d(indx,Size)=s.bytes ;
  d(indx, Mtm)=T/NTIMES;

  T=clock; for i=1:NTIMES                         % medir cambio de tipo
    arrayUint8=uint8(arrayDouble);
    clear arrayUint8
  end, T=etime(clock,T);
  d(indx, Ctm)=T/NTIMES;

  fprintf('d');                                   % ping-pong parte double
  MPI_Barrier(NEWORLD); T=clock; for i=1:NTIMES
    MPI_Send(arrayDouble,1,TAG,NEWORLD);
    MPI_Recv(arrayDouble,1,TAG,NEWORLD);
  end, T=etime(clock,T);
  d(indx,TXtm)=T/2/NTIMES;

  clear arrayDouble

          fprintf(',');

          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

else,     fprintf('-,'   );
end

          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if indx>=Ustr & indx<=Uend                        % hacer parte uint8

  NTIMES = ntimes*reps(Alto*Ancho*sizeofU8);      % medir reserva memoria
  T=clock; for i=1:NTIMES
    clear arrayUint8
    arrayUint8(Alto,Ancho)=uint8(0);              % liberar al final
  end, T=etime(clock,T); s=whos('arrayUint8');
  u(indx,Size)=s.bytes;
  u(indx, Mtm)=T/NTIMES;
```

```
      fprintf('u');
      MPI_Barrier(NEWORLD); T=clock; for i=1:NTIMES
        MPI_Send(arrayUint8,1,TAG,NEWORLD);
        MPI_Recv(arrayUint8,1,TAG,NEWORLD);
      end, T=etime(clock,T);
      u(indx,TXtm)=T/2/NTIMES;

      clear arrayUint8

              fprintf(']\t');

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    else,     fprintf('-]\t');
    end

              %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  end
end
fprintf('\n');

MPI_Finalize; clear mex
unix('wipe bhost'); delete('bhost');

toc

unix('vmstat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Salvar datos %
%%%%%%%%%%%%%%%%
  fname=[SUB '/' FN '-' HN0 '-' HN1];
  switch hmg
    case 0, fname=[fname '-_'];
    case 1, fname=[fname '-O'];
  end
  switch c2c
    case 0, fname=[fname '-___'];
    case 1, fname=[fname '-c2c'];
  end
  save(fname, 'Ustr', 'Uend', 'Dstr', 'Dend', 'ROWS', 'COLS')
  save(fname,'l','u','d','-append')
```

Listado D.9: Continuación.

07/25/01
20:30:14

mpiCode/BWSlav.m

1

```
function BWSlav(hmg,c2c,row,ntimes)
% BWSlav: Medición de latencia (ping-pong) y ancho de banda para JPDC
%
%       BWSlav ( hmg, c2c, row, times )
%
%  ver el help para BW
%

  global TAG NEWORLD
  reps=inline('2.^fix((30-fix(log2(max(s,1))))./3.33-1)');
  varDbl=      0 ; s=whos('varDbl'); sizeofDbl=s.bytes; % para cuentas tamaños
  varU8 =uint8(0); s=whos('varU8' ); sizeofU8 =s.bytes;
  clear s varDbl varU8

%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
  if nargin<4, return, end
  indx=0; Ustr=1; Uend=38; Dstr=1; Dend=34;             % filas
                  Uend=row+4;      Dend=row;

  TRTF=str2num(getenv('TROLLIUSRTF'));
  if xor(hmg,bitand(TRTF,RTF_HOMOG))
        error('argumento hmg no coincide con TROLLIUSRTF')
  end
  if xor(c2c,bitand(TRTF,RTF_MPIC2C))
        error('argumento hmg no coincide con TROLLIUSRTF')
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Comunicación con padre %
%%%%%%%%%%%%%%%%%%%%%%%%%%
  arrayNULL(2048,2048)=0;               % enviar algo, por ver posible swap
  MPI_Recv(arrayNULL,0,TAG,NEWORLD);
  MPI_Send(arrayNULL,0,TAG,NEWORLD);
  clear arrayNULL, arrayNULL=[];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BUCLE PING-PONG %
%%%%%%%%%%%%%%%%%%%
  for ExpAncho=0:12
    Ancho=2^ExpAncho;
    for Alto=Ancho:Ancho:3*Ancho
      indx=indx+1;

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if(indx>=Dstr & indx<=Dend)|...                   % algo que hacer
        (indx>=Ustr & indx<=Uend)

        NTIMES=ntimes*reps(0);                          % hacer parte latencia
        MPI_Barrier(NEWORLD); for i=1:NTIMES
          MPI_Recv(arrayNULL,0,TAG,NEWORLD);
          MPI_Send(arrayNULL,0,TAG,NEWORLD);
        end
      end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if indx>=Dstr & indx<=Dend                        % hacer parte double

        NTIMES = ntimes*reps(Alto*Ancho*sizeofDbl);
        arrayDouble(Alto,Ancho)=0;                      % preallocation
        MPI_Barrier(NEWORLD); for i=1:NTIMES
          MPI_Recv(arrayDouble,0,TAG,NEWORLD);
          MPI_Send(arrayDouble,0,TAG,NEWORLD);
        end
        clear arrayDouble
      end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      if indx>=Ustr & indx<=Uend                        % hacer parte uint8

        NTIMES = ntimes*reps(Alto*Ancho*sizeofU8);
        arrayUint8(Alto,Ancho)=uint8(0);                % preallocation
        MPI_Barrier(NEWORLD); for i=1:NTIMES
          MPI_Recv(arrayUint8,0,TAG,NEWORLD);
          MPI_Send(arrayUint8,0,TAG,NEWORLD);
        end
        clear arrayUint8
      end

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    end
  end

  MPI_Finalize;
```

Listado D.10: mpiCode/BWSlav.m: Proceso eco en MPITB.

Apéndice E

Código MATLAB para la transformada *wavelet*

E.1 Introducción

En el Capítulo 5 se ha hecho referencia a programas MATLAB desarrollados para ejercitar las *Toolboxes* paralelas PVMTB y MPITB en una aplicación típica de Visión por Computador como es la transformada *wavelet*.

El algoritmo original es atribuible a Mallat [46]. Los programas presentados contienen instrumentación adicional para recolectar los datos necesarios y realizar las gráficas mostradas en la exposición del capítulo. Esta instrumentación, cómoda de realizar en un lenguaje de rápido prototipado como MATLAB, dilapidaría una cantidad prohibitiva de tiempo de desarrollo en un lenguaje como C o FORTRAN.

E.2 Listados

A continuación se ofrecen los listados de los programas referenciados en el Capítulo 5. El directorio que se muestra a continuación permite localizarlos fácilmente y relacionarlos con el apartado en que se hace referencia a ellos.

	Apartado	Listado	página
Secuencial B/W	5.2.1 p.269		
Sec, programa principal		E.1	391
dwt2, subrutina análisis 2D		E.2	392
dwt, subrutina análisis 1D		E.3	393
idwt2, subrutina reconstrucción 2D		E.4	394
idwt, subrutina reconstrucción 1D		E.5	395
Instrumentación	Figura 5.5(d) p.276		
record, anotación de tiempos y flops		E.6	396
label, etiquetado de anotaciones		E.7	396
recgraph, escritura de anotaciones		E.8	397
stats, estadísticas sobre anotaciones		E.9	398
graph, generación de gráfica 5.5(d)		E.10	399
math, normalización de gráfica 5.5(c)		E.11	400
Secuencial RGB	5.2.2 p.275		
SecRGB, programa principal		E.12	401
PVMTB B/W	5.3.1 p.282		
startup_Slv, *script* arranque esclavos		E.13	403
Par, programa maestro		E.14	404
dwt2ParHost, distribución análisis 2D		E.15	407
dwt2Par, esclavo análisis 2D		E.16	408
dwtflt, subrutina análisis 1D		E.17	409
idwt2ParHost, distribución reconstrucción 2D		E.18	410
idwt2Par, esclavo reconstrucción 2D		E.19	411
idwtflt, subrutina reconstrucción 1D		E.20	412
MPITB B/W	5.3.2 p.287		
startup_Slv, *script* arranque esclavos		E.21	413
Par, programa maestro		E.22	414
dwt2ParHost, distribución análisis 2D		E.23	417
dwt2Par, esclavo análisis 2D		E.24	418
idwt2ParHost, distribución reconstrucción 2D		E.25	419
idwt2Par, esclavo reconstrucción 2D		E.26	420
PVMTB RGB	5.4.1 p.293		
ParRGB, programa maestro		E.27	421
dwt2D3L, programa esclavo		E.28	424
MPITB RGB	5.4.2 p.296		
ParRGB, programa maestro		E.29	425
dwt2D3L, programa esclavo		E.30	428

07/10/01 17:11:32 — pvmtbCode/Sec.m — **1**

```
function [T,R,S]=Sec(ntimes,withgraph,withlena)
% Sec: Ejemplo secuencial B/N análisis wavelet multiresolución
%
%       [T R S] = Sec ( ntimes, withgraph, withlena )
%
%  ntimes       [1],   número de veces a repetir el trabajo (medir tiempo)
%                0 ,    si se desea consultar datos de ejecución anterior
%  withgraph    0/[1], indica si se desean ver las gráficas de tiempo
%                      el tiempo de gráficas no se incluye, naturalmente
%  withlena     [0]/1, indica si se desean ver las imágenes
%                      el tiempo de dibujo se incluye en la gráfica
%
%  T    "Times", registro con campos:
%  totl tiempo total gastado (no incluye tiempo para gráfica tiempos)
%  spwn 0
%
% R     "Round", tiempo de cada vuelta (de Load a Load)
% S     "Stats", cell-array con columnas {'label' mean stdev ntimes}

%       tomado de Mallat "A Theory for Multiresolution Signal Decomposition:
%                               The Wavelet Representation"
%       IEEE Trans. on Patt.Analysis and Mach.Intell. Vol.11 N.7 July 1989

% Necesita:
% lena_funet.tif        Imagen de Lena en B/N, obtenida de ftp.funet.fi
% h.mat         Filtro Wavelet Aproximación (unidimensional)
% g.mat         Filtro Wavelet Detalle      (unidimensional)
%
% dwt2          Descomposición Wavelet Bi-dimensional
%   dwt         Descomposición Wavelet Unidimensional
% idwt2         Re-composición Wavelet Bi-dimensional
%  idwt         Re-composición Wavelet Unidimensional
%
% math          Operaciones de normalización para visualizar transformada
% stats         Estadísticas sobre las anotaciones
% record        Anotar tiempos
% label         Anotar etiquetas
% recgraph      Salvar datos gráfica tiempos
% graph         Generación de gráficas a partir de datos

%%%%%%%%%%%
% ArgChk %
%%%%%%%%%%%
if nargin<3, withlena =0; else, withlena =min(max(round(withlena ),0),1); end
if nargin<2, withgraph=1; else, withgraph=min(max(round(withgraph),0),1); end
if nargin<1, NTIMES   =1; else, NTIMES   =    max(round(ntimes   ),0   ); end

global DIR, DIR=[ getenv('PVMTB_ROOT') '/Users99/' mfilename ];
if ~NTIMES,[T R S]=shwgraph(withgraph); return, end

tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos/flops % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix labels % fp
tm=zeros(1+(7+8*withlena)*NTIMES,6); msg={}; ix=0; labels={};   record('')
% fp=zeros(,1);
%%% 1:       ''
%%% 7*:      'Load' 'D1' 'D2' 'D3' 'R3' 'R2' 'R1'
%%% 8*lena*: 'math' 'cast' 3*('fig' 'shw')
%%% cada bucle va de 'Load' a 'Load'

%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
for indiceNTIMES=1:NTIMES

%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_funet','tif'); load h, load g
gd=[  g(2:length(g));0];                                        % Detail decomp/recomp
gr=[0;g(1:length(g)-1)];                                        record('Load')

img=double(lena);                                               record('db')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Descomposición 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[a1 dh1 dv1 dd1]=dwt2(img,h,gd,'spec'); record('D1'),            label
[a2 dh2 dv2 dd2]=dwt2( a1,h,gd,'spec'); record('D2'),if NTIMES<3,label,end
[a3 dh3 dv3 dd3]=dwt2( a2,h,gd,'spec'); record('D3'),if NTIMES<2,label,end

%%%%%%%%%%%%%%%%%
% Recomposición %
%%%%%%%%%%%%%%%%%
r2=idwt2(a3,dh3,dv3,dd3 ,h,gr, 'spec'); record('R3'),if NTIMES<2,label,end
r1=idwt2(r2,dh2,dv2,dd2 ,h,gr, 'spec'); record('R2'),if NTIMES<3,label,end
r0=idwt2(r1,dh1,dv1,dd1 ,h,gr, 'spec'); record('R1'),           label

%%%%%%%%%%%%
% Gráficas %
%%%%%%%%%%%%
if withlena
 transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);record('math')
  reconstr=uint8(r0);                                           record('cast')

  figure,                                                       record('fig')
  imshow(lena), title('Original'),                              record('shw')
  figure,                                                       record('fig')
  imshow(transform), title('Transformada'),                     record('shw')
  figure,                                                       record('fig')
  imshow(reconstr), title('Reconstrucción'),                    record('shw')
end

%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
end

        toc                                     % La gráfica nunca cuenta
        recgraph;                               % Salvar datos gráfica
[T R S]=shwgraph(withgraph);                    % Ver gráfica si no se impide

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [T,R,S]=shwgraph(withgraph)
global DIR

         HN0=strtok(hostname,'.');
[T R S MSG]=stats(DIR,[mfilename '_' HN0]); disp(MSG), S

if withgraph
  graph(DIR,mfilename,HN0);
end

clear global
```

Listado E.1: Sec.m: Programa secuencial MATLAB para la transformada *wavelet* 2D BW (imagen digitalizada en niveles de gris). Se mostró un fragmento en el Listado 5.1.

05/15/00 16:16:43 pvmtbCode/dwt2.m **1**

```
function [ahav, ahdv, dhav, dhdv]=dwt2(img, h,g, varargin)
% DWT2          2-D Discrete Wavelet Transform. Devuelve approx/det horiz/vert
%
%       [ahav, ahdv, dhav, dhdv] = dwt2 ( img, h,g [,opt] )
%
%  img (matriz) Señal 2-D a transformar
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  g   (vector) Filtro 1-D detalle      (paso alta) para descomposición. lh=lg
%  opt  (str)   acción para evitar efecto bordes convolución
%               'zero' -> añadir ceros a los bordes (por defecto)
%               'spec' -> añadir reflejos especulares de la señal
%
%  ahav(matriz) Señal 2-D aproximación  (coeficientes). Area 1/4 de img
%  ahdv(matriz) Señal 2-D aprox. horiz, detalle vert. (bordes horiz)
%  dhav(matriz) Señal 2-D detalle horiz, aprox. vert. (bordes vert)
%  dhdv(matriz) Señal 2-D detalle horiz, detalle vert.(bordes diagonal)
%

%%%%%%%%%%%%
% Longitudes
%%%%%%%%%%%%
lih =size(img,2); liv =size(img,1);     % Length image horz/vert
lih2=ceil(lih/2); liv2=ceil(liv/2);     % Al decimar puede salir 1 de mas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtrar por filas, cada fila se filtra a lo largo de las columnas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --------       ----
% |      |  --\  |  |
% |      |  --/  |  |
% --------       ----
%   img      h    ah    filtrando con paso baja h se obtiene ah approx. horiz.
%            g    dh    filtrando con paso alta g se obtiene dh detail  horiz.
%                ----
%           --\  |  |
%           --/  |  |
%                ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ah=zeros(liv,lih2);                     % Pre-allocation
dh=zeros(liv,lih2);
for row=1:liv                           %%%%%%%%%%%%%%%%%%%%
   r      = img(row,:);                 % Obtener img->ah,dh
  [ra rd] = dwt(r',h,g, varargin{:});   %%%%%%%%%%%%%%%%%%%%
   ah(row,:) = ra';                     % Pasar vectores columna siempre
   dh(row,:) = rd';
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtrar por columnas, cada columna se filtra a lo largo de las filas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----   h   ----       -> ah->h->se obtiene ahav approx. horiz. approx. vert.
% |  |  --\  |  | ----
% |  |  --/  ---- |  |  -> ah->g->se obtiene ahdv approx. horiz. detail  vert.
% ----   g        ----
%  ah        ahav ahdv
%%%%%%%%%%%%%%%%%%%%%%
% ----   h   ----       -> dh->h->se obtiene dhav detail  horiz. approx. vert.
% |  |  --\  |  | ----
% |  |  --/  ---- |  |  -> dh->g->se obtiene dhdv detail  horiz. detail  vert.
% ----   g        ----
%  dh        dhav dhdv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ahav=zeros(liv2,lih2);                  % Pre-allocation
ahdv=zeros(liv2,lih2);
dhav=zeros(liv2,lih2);                  % Pre-allocation
dhdv=zeros(liv2,lih2);
for col=1:lih2
   c      = ah(:,col);                  %%%%%%%%%%%%%%%%%%%%%%%%
  [ca cd] = dwt(c ,h,g, varargin{:});   % Obtener ah->ahav,ahdv
   ahav(:,col)= ca;                     %%%%%%%%%%%%%%%%%%%%%%%%
   ahdv(:,col)= cd;

   c      = dh(:,col);                  %%%%%%%%%%%%%%%%%%%%%%%%
  [ca cd] = dwt(c ,h,g, varargin{:});   % Obtener dh->dhav,dhdv
   dhav(:,col)= ca;                     %%%%%%%%%%%%%%%%%%%%%%%%
   dhdv(:,col)= cd;
end
```

Listado E.2: dwt2.m: Rutina para análisis *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.2.

03/07/00 11:20:44 pvmtbCode/dwt.m **1**

```
function [a, d]=dwt(x,h,g, varargin)
% DWT                   Discrete Wavelet Transform. Devuelve aprox./detalle
%
%        [a d] = dwt ( x, h,g [,opt] )
%
%  x (vector)   Señal 1-D a transformar
%  h (vector)   Filtro aproximación (paso baja)
%  g (vector)   Filtro detalle      (paso alta) para descomposición. lh = lg
%  opt  (str)   acción para evitar efecto bordes convolución
%               'zero' -> añadir ceros a los bordes (por defecto)
%               'spec' -> añadir reflejos especulares de la señal
%
%  a (vector)   Señal 1-D aproximación  (coeficientes). Longitud mitad que x
%  d (vector)   Señal 1-D detalle       (coeficientes). (la = ld = ceil(lx/2))
%

%%%%%%
% ArgChk
%%%%%%%%%
lx=length(x); lh=length(h); lg=length(g);
if lh~=lg, error('debe coincidir longitud filtros'), end
lh2=floor(lh/2);                % Si lf impar, lf2 es longitud cola
lg2=floor(lg/2);                % (quitando la muestra central)
mode='zero';                    % Opción bordes
if ~isempty(varargin)
  if isstr(varargin{1})
    switch varargin{1}
      case {'spec','zero'},   mode=varargin{1};
      otherwise,              warning('modo no reconocido')
    end
  else,         warning('argumentos adicionales no string')
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2 reflejos de tamaño lf2, Convolución, Centrado y Downsample
% ly == lx + 2lf2                              (filtro f = h/g)
% lz == ly + lf - 1 == lx + 2lf2 + lf - 1      (señal  z = a/d)
% dejarlo en lx, y luego en lx/2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aproximacion
%%%%%%%%%%%%%%
if strcmp(mode,'spec'),  y=[x(lh2+1:-1:2); x ; x(lx-1:-1:lx-lh2)];
else,                    y=[zeros(lh2, 1); x ; zeros(lh2,1)]; end;
a=conv(y,h);            % Filtrado
a=a(lh:lx+lh-1);        % Corte correspondiente
a=a(1:2:lx);            % Downsample. Puede salir ceil(lx/2) una de más
%%%%%%%%%
% Detalle
%%%%%%%%%
if strcmp(mode,'spec'),  y=[x(lg2+1:-1:2); x ; x(lx-1:-1:lx-lg2)];
else,                    y=[zeros(lg2, 1); x ; zeros(lg2,1)]; end;
d=conv(y,g);            % Filtrado
d=d(lg:lx+lg-1);        % Corte correspondiente
d=d(1:2:lx);            % Downsample. Puede salir ceil(lx/2) una de más
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación reflejo: Si lf impar,
%  cuando centro filtro sobre x(1), cola filtro x(2)...x(lf2+1)
% Añadir lf2 muestras reflejo (2..lf2+1), sin repetir x(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación correspondencia entre señales:
% Señal original:            x(1)...x(     lx)
% Efecto bordes:   y(1)... y(lf2+1)...y( lf2+lx)...y(2lf2+lx)
% Conv: z(1)...z(lf2+1)...z(2lf2+1)...z(2lf2+lx)...z(3lf2+lx)...z(4lf2+lx)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf impar:           ...z(lf)   ...z(lx+lf-1)...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf par:             ...z(lf+1) ...z(lx+lf)  ...
%  debería cortarse entre   (lf+1  ...  lx+lf)  ???
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listado E.3: dwt.m: Rutina para análisis *wavelet* 1D B/W. Se mostró un fragmento en el Listado 5.3.

03/07/00
10:05:35

pvmtbCode/idwt2.m

1

```
function img=idwt2(ahav, ahdv, dhav, dhdv, h,g, varargin)
% IDWT2         2-D Inverse Discrete Wavelet Transform. Devuelve imagen
%
%       img = idwt2 ( ahav, ahdv, dhav, dhdv, h,g [,opt] )
%
%  ahav(matriz) Señal 2-D aproximación  (coeficientes).
%  ahdv(matriz) Señal 2-D aprox. horiz, detalle vert. (bordes horiz)
%  dhav(matriz) Señal 2-D detalle horiz, aprox. vert. (bordes vert)
%  dhdv(matriz) Señal 2-D detalle horiz, detalle vert.(bordes diagonal)
%
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  g   (vector) Filtro 1-D detalle      (paso alta) para recomposición. lh = lg
%  opt  (str)   acción para evitar efecto bordes convolución
%               'zero' -> añadir ceros a los bordes (por defecto)
%               'spec' -> añadir reflejos especulares de la señal
%
%  img (matriz) Señal 2-D recompuesta. Area cuádruple
%

%%%%%%%%%%%%
% Longitudes
%%%%%%%%%%%%
l2=size(ahav);
if any(l2~=size(ahdv))|...
   any(l2~=size(dhav))|...
   any(l2~=size(dhdv)), error('imagenes de distintos tamaños'), end
lih2=l2(2);  liv2=l2(1);
lih =lih2*2; liv =liv2*2;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Des-filtrar por columnas, cada columna se filtra a lo largo de las filas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----   h   ----       -> ah->h->se obtuvo ahav approx. horiz. approx. vert.
% |  |  /--  |  | ----
% |  |  \--  ---- |  |  -> ah->g->se obtuvo ahdv approx. horiz. detail  vert.
% ----   g        ----
%  ah        ahav ahdv
%%%%%%%%%%%%%%%%%%%%%%
% ----   h   ----       -> dh->h->se obtuvo dhav detail  horiz. approx. vert.
% |  |  /--  |  | ----
% |  |  \--  ---- |  |  -> dh->g->se obtuvo dhdv detail  horiz. detail  vert.
% ----   g        ----
%  dh        dhav dhdv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ah=zeros(liv,lih2);                     % Pre-allocation
dh=zeros(liv,lih2);                     % Pre-allocation
for col=1:lih2
   ca=ahav(:,col); cd=ahdv(:,col);      %%%%%%%%%%%%%%%%%%%%%%%
   c =idwt(ca ,cd ,h,g, varargin{:});   % Obtener ahav,ahdv->ah
   ah(:,col)= c ;                       %%%%%%%%%%%%%%%%%%%%%%%

   ca=dhav(:,col); cd=dhdv(:,col);      %%%%%%%%%%%%%%%%%%%%%%%
   c =idwt(ca ,cd ,h,g, varargin{:});   % Obtener dhav,dhdv->dh
   dh(:,col)= c ;                       %%%%%%%%%%%%%%%%%%%%%%%
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Des-filtrar por filas, cada fila se filtra a lo largo de las columnas
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% --------       ----
% |      |  /--  |  |
% |      |  \--  |  |
% --------       ----
%   img      h    ah    filtrando con paso baja h se obtuvo ah approx. horiz.
%            g    dh    filtrando con paso alta g se obtuvo dh detail  horiz.
%                ----
%           /--  |  |
%           \--  |  |
%                ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
img=zeros(liv,lih);                         % Pre-allocation
for row=1:liv
   ra=ah(row,:); rd=dh(row,:);              %%%%%%%%%%%%%%%%%%%%
   r =idwt(ra',rd',h,g, varargin{:});       % Obtener ah,dh->img
   img(row,:)= r';                          %%%%%%%%%%%%%%%%%%%%
end                                         % Pasar siempre vectores columna

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% No hace falta deshacerlo simétricamente, filtrado separable conmuta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%
% Filtrar por filas
%%%%%%%%%%%%%%%%%%%
% av=zeros(liv2,lih);                       % Pre-allocation
% dv=zeros(liv2,lih);                       % Pre-allocation
% for row=1:liv2
%  ra=ahav(row,:); rd=dhav(row,:);          %%%%%%%%%%%%%%%%%%%%%%%
%  r =idwt(ra',rd',h,g, varargin{:});       % Obtener ahav,dhav->av
%  av(row,:)= r';                           %%%%%%%%%%%%%%%%%%%%%%%
%
%  ra=ahdv(row,:); rd=dhdv(row,:);          %%%%%%%%%%%%%%%%%%%%%%%
%  r =idwt(ra',rd',h,g, varargin{:});       % Obtener ahdv,dhdv->dv
%  dv(row,:)= r';                           %%%%%%%%%%%%%%%%%%%%%%%
% end
%%%%%%%%%%%%%%%%%%%%%%
% Filtrar por columnas
%%%%%%%%%%%%%%%%%%%%%%
% img=zeros(liv,lih);                       % Pre-allocation
% for col=1:lih
%  ca=av(:,col); cd=dv(:,col);              %%%%%%%%%%%%%%%%%%%%
%  c =idwt(ca, cd, h,g, varargin{:});       % Obtener av,dv->img
%  img(:,col)= c;                           %%%%%%%%%%%%%%%%%%%%
% end
```

Listado E.4: idwt2.m: Rutina para reconstrucción *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.4.

03/07/00 11:20:32 — pvmtbCode/idwt.m — 1

```
function x=idwt(a,d,h,g, varargin)
% IDWT                  Inverse Discrete Wavelet Transform. Devuelve señal
%
%       x = idwt ( a,d, h,g [,opt] )
%
%  a (vector)   Señal 1-D aproximación  (coeficientes)
%  d (vector)   Señal 1-D detalle       (coeficientes). Igual longitud que a
%  h (vector)   Filtro aproximación (paso baja)
%  g (vector)   Filtro detalle      (paso alta) para reconstrucción. lh = lg
%  opt  (str)   acción para evitar efecto bordes convolución
%               'zero' -> añadir ceros a los bordes (por defecto)
%               'spec' -> añadir reflejos especulares de la señal
%
%  x (vector)   Señal 1-D recompuesta. Longitud doble que a/d (lx = 2la = 2ld)
%

%%%%%%%%
% ArgChk
%%%%%%%%
la=length(a); ld=length(d);
lh=length(h); lg=length(g);
if la~=ld, error('debe coincidir longitud aprox-detalle'), end
if lh~=lg, error('debe coincidir longitud filtros'), end
lh2=floor(lh/2);                % Si lf impar, lf2 es longitud cola
lg2=floor(lg/2);                % (quitando muestra central)
mode='zero';                    % Opción bordes
if ~isempty(varargin)
  if isstr(varargin{1})
    switch varargin{1}
      case {'spec','zero'},   mode=varargin{1};
      otherwise,              warning('modo no reconocido')
    end
  else,         warning('argumentos adicionales no string')
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Upsample, 2 reflejos de tamaño lh2, Convolución y Centrado
% ly == 2lx + 2lf2                              (filtro f = h/g)
% lz == ly + lf - 1 == 2lx + 2lf2 + lf - 1      (señal  x = a/d)
% dejarlo en 2lx                                (señal  z = reconstr a/d)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Aproximacion
%%%%%%%%%%%%%%
y=zeros(2*la,1);                % Pre-alloc
y(1:2:2*la)=a;                  % Upsample. a(1),0,a(2),0...a(la),0
if strcmp(mode,'spec'),  y=[y(lh2+1:-1:2); y ; y(2*la-1:-1:2*la-lh2)];
else,                    y=[zeros(lh2, 1); y ; zeros(lh2,1)];     end;
za=conv(y,h);                   % Filtrado
za=za(lh:2*la+lh-1);            % Corte correspondiente
%%%%%%%%%
% Detalle
%%%%%%%%%
y=zeros(2*ld,1);                % Pre-alloc
y(1:2:2*ld)=d;                  % Upsample. d(1),0,d(2),0...d(ld),0
if strcmp(mode,'spec'),  y=[y(lg2+1:-1:2); y ; y(2*ld-1:-1:2*ld-lg2)];
else,                    y=[zeros(lg2, 1); y ; zeros(lg2,1)];     end;
zd=conv(y,g);                   % Filtrado
zd=zd(lg:2*ld+lg-1);            % Corte correspondiente
%%%%%%%%%%%
% Resultado                     % Recuperación normalización R^2
%%%%%%%%%%%
x=2*(za+zd);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Explicación reflejo: Si lf impar,
%  cuando centro filtro sobre y(1), cola filtro y(2)...y(lf2+1)
% Añadir lf2 muestras reflejo (2..lf2+1), sin repetir y(1)==x(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación correspondencia entre señales:
% Señal original:              x(1)...x(lx)
% Upsample:               y/x(1),0,...x(lx),y(2lx)==0
% Efecto bordes:   y(1)...y(lf2+1) ...      y(2lx+ lf2)...y(2lx+2lf2)
% Conv: z(1)...z(lf2+1)...z(2lf2+1)...      z(2lx+2lf2)...z(2lx+3lf2)...+4lf2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf impar:          ...z(lf)    ...      z(2lx+lf-1)...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf par:            ...z(lf+1)  ...      z(2lx+lf)  ...
%  debería cortarse entre  (lf+1   ...        2lx+lf)  ???
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listado E.5: idwt.m: Rutina para reconstrucción *wavelet* 1D B/W. Se mostró un fragmento en el Listado 5.5.

```
function record(M)
% RECORD                    Anotar tiempos y flops para Users99
%
%       record(M)
%
%       Requiere las variables globales "tm msg ix", opcional "fp"
%       tm anota tiempos (clock), fp anota flops (si no está vacía)
%       msg anota nombres (string M) para los ticks de abscisa
%       ix lleva la cuenta de anotaciones realizadas (inicialmente 0)

global tm msg ix fp
ix=ix+1;
tm(ix,:)=clock;
msg{end+1}= M;              % msg={} inicialmente, no pre-alloc
if iscell(fp)               % si no existía, global la pone a []
    fp(ix)=flops;
end
```

Listado E.6: Rutina record.m. Anota la hora a la que se ejecuta, añadiendo una etiqueta descriptiva proporcionada por el usuario. También puede contabilizar los flops.

```
function label
% LABEL                     Función para etiquetar registro tiempos Users99
%
%       Anota el mensaje actual y nº ix como columna del cell-array labels
%       Requiere variables globales "msg ix labels" (inicialmente labels={})

global msg ix labels
labels(:,end+1)={msg{ix}; ix};            % labels={} inicialmente, no pre-alloc
```

Listado E.7: Rutina label.m. Copia anotaciones record que se desea que aparezcan en la gráfica realizada por graph.

05/14/00 20:21:49 — pvmtbCode/recgraph.m — 1

```
function recgraph
% RECGRAPH              Salvar datos para gráficas de Users99
%
%       recgraph
%
%       Requiere variables globales "tm msg ix DIR", y tal vez "fp labels"
%       Se salvan datos en DIR
%       Si está definida "fp" se usa para gráfica flops
%       Si "labels={'etiq1',...'etiqN'; pos1,... posN}", se usa para
%               gráfica tiempo total, etiquetada con dichas "labels"
%

global tm msg ix DIR fp labels
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pasar tiempos a parciales % Eliminar 1ª anotación
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=ix:-1:2,                                  % 1ª anotación es hora inicial
    tm(i,6)=etime(tm(i,:),tm(i-1,:));           % convertir a tiempos parciales
    if ~isempty(fp)                             % Si no existía, global hace []
        fp(i)=fp(i)-fp(i-1);                    % ...a flops parciales, si hay
    end
end,tm=tm(:,6);                                 % quedarse con segundos
    tm=tm(2:ix); msg=msg(2:ix);                 % 1ª anotación es hora inicial
if ~isempty(fp), fp =fp (2:ix); end, ix=ix-1;   % eliminarla, decrementar total
if ~isempty(labels), for i=1:size(labels,2)     % y corregir pos. etiquetas
        labels{2,i}=labels{2,i}-1;
end,end

%%%%%%%%%%%%%%%
% Mensajillos %
%%%%%%%%%%%%%%%
HOST=strtok(hostname,'.'); st=dbstack;
if length(st)<2, error('No llamar desde prompt'), end
[p FUN]=fileparts(st(2).name);
                                % Esto se calcula otra vez en stats
ttotl=sum(tm);                  % De todas formas, hace falta para el mensaje
if strcmp('Spwn',msg{1})        % find(strcmp('Spwn',msg))
           tspwn= tm(1);        % devuelve [] si no encuentra
           MSG=sprintf('[%s:%s]: %.4f seconds, %.4f w/o Spawn\n',...
                        HOST,FUN,ttotl,ttotl-tspwn);
else,      tspwn=0;
           MSG=sprintf('[%s:%s]: %.4f seconds\n',HOST,FUN,ttotl);
end                                             % Mensajillo para ventana cmd
TITLE=[HOST ': Time log for ' FUN];             % Mensajillo para título

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Salvar datos como para regenerar gráfica %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
cwd=pwd; [p n e]=fileparts(DIR);                % Antes de cambiarse a DIR
st=mkdir(p,[n e]);      cd(DIR)                 % crearlo por si no existe
FNAM=[FUN '_' HOST];    save(FNAM,'tm','msg','ix')              % overwrite
if ~isempty(fp    ),    save(FNAM,'fp',         '-append'), end
if ~isempty(labels),    save(FNAM,'labels',     '-append'), end
                        save(FNAM,'MSG','TITLE','-append')
cd(cwd)                                 % Volver adonde estábamos
```

Listado E.8: Rutina recgraph.m. Salva a disco los datos de instrumentación.

05/15/00
20:09:29

pvmtbCode/stats.m

1

```
function [T,R,S, MSG,tm,labels,msg]=stats(DIR,FNAM,trans)
% STATS                 Estadísticas para datos gráficas Users99
%
%        [T R S MSG tm labels msg] = stats ( DIR, FNAM, trans )
%
% DIR   Directorio en que se encuentra el fichero
% FNAM  Nombre del fichero generado por aplicación Users99, formato .MAT
% trans Tabla de traducción, filas de la forma {'old' 'new'}
%       Antes de hacer estadísticas, se reemplaza 'old'->'new'
%
% T     "Times", registro con campos:
%  totl tiempo medido en total
%  spwn si 1ª anotación es 'Spwn', tiempo en que se anotó
%
% R     "Round" tiempo total de cada vuelta (de Load a Load ó '' a '')
% S     "Stats" cell-array con columnas {'label' mean stdev ntimes}
%
% MSG    Mensaje asociado a la gráfica (T)
% tm     Tiempos medidos
% labels Etiquetas de la gráfica de tiempos acumulados
% msg    Tabla total de mensajes
%

if nargin<3, trans={}; end
%%%%%%%%%%%%%%%%%%%%%%%%
% Cargar datos gráfica %
%%%%%%%%%%%%%%%%%%%%%%%%
cwd=pwd; cd(DIR); load(FNAM,'-MAT'); cd(cwd)    % Ir, cargar y volver

%%%%%%%%%
% Times %
%%%%%%%%%
% disp(MSG);                                    % se salvó en fichero
         T.totl =sum(tm);
if strcmp('Spwn',msg{1}),T.spwn = tm(1);        % find(strcmp('Spwn',msg))
else,                    T.spwn =    0 ;        % devuelve [] si no encuentra
end

%%%%%%%%%
% Round %
%%%%%%%%%
R = [];
Lidx=find(strcmp('Load',msg));          % "Load" index
if isempty(Lidx)                        % En esclavos no hay 'Load', sino ''
  msg={'' msg{:}};                      % Añadir marca inicial ficticia,
   tm=[0;  tm(:)];                      % reducir al caso de "Load"
  for il=1:size(labels,2), labels{2,il}=labels{2,il}+1; end
  Lidx=find(strcmp('',msg));            % Hay "" en esclavos, sobre todo
end                                     % después de añadir éste

NTIMES=length(Lidx);                            % deducido de #Loads ó #''
Lidx(end+1)=length(tm)+1;                       % ahí caería el siguiente Load
R=zeros(NTIMES,1);                              % Prealloc tiempo total/vuelta
for i=1:NTIMES, R(i)=sum(tm(Lidx(i):Lidx(i+1)-1)); end

%%%%%%%%%
% Stats %
%%%%%%%%%
while ~isempty(trans)                           % quedan traducciones
  Tidx=(find(strcmp(trans{1,1},msg)));          % ocurrencias de 1ª etiqueta
  if ~isempty(Tidx)                             % si hay,
    msg(Tidx)=trans(1,2);                       % sustituir por 2ª etiqueta
  end
  trans(1,:)=[];                                % eliminar traducción
end

% S={'' 0 0 0};
  S={'Round' mean(R) std(R,1) length(R)};       % no debe estar vacía (find)
for idx=1:length(msg)                           % repasar todas las etiquetas
  if isempty(find(strcmp(msg(idx),S(:,1))))     % si no está ya promediada
    S{end+1,1}=msg{idx};                        % reservarle su fila
        Lidx=find(strcmp(msg(idx),msg));        % buscar todas sus ocurrencias
    S{end  ,2}=mean(tm(Lidx));
    S{end  ,3}= std(tm(Lidx),1);
    S{end  ,4}= length(Lidx);
  end
end
% S(1,:)=[];
```

Listado E.9: Rutina stats .m. Genera estadísticas a partir de los datos de instrumentación.

05/27/00 20:44:38 — pvmtbCode/graph.m — 1

```
function graph(DIR,FUN,HOST)
% GRAPH                 Función para gráficas de tiempos y flops para Users99
%
%       graph ( DIR, FUN, HOST )
%
%  DIR (str)    Directorio donde está el fichero de datos "FUN_HOST.mat"
%  FUN (str)    Función que creó el fichero
%  HOST(str)    Host    que creó el fichero
%

%%%%%%%%%%%%%%%%%%%
% Recuperar datos %
%%%%%%%%%%%%%%%%%%%
CWD = pwd; FNAM=[FUN '_' HOST];                 % Acaba en '.mat'
cd(DIR),   load(FNAM, '-MAT')

%%%%%%%%%%%%%%%%%%%%%%%
% Reconstruir gráfica %
%%%%%%%%%%%%%%%%%%%%%%%
    if exist('fp',    'var'),   NP=3;           % Número de subplots
elseif exist('labels','var'),   NP=2;
else,                           NP=1; end

figure
subplot(NP,1,1),bar(tm),                        % 1ª figura tiempos parciales
                set(gca,'XTick',     1:ix)
                set(gca,'XTickLabel', msg)
                xlabel('SubTask'), ylabel('Seconds')
                title(TITLE)

if NP>1                                         % 2ª figura opcional: cumulat.
subplot(NP,1,2),barh([tm zeros(ix,1)]','stack')
                xlabel(MSG)                     % etiquetas seleccionadas
                AX=axis; AX(3:4)=[0.2 1.8]; axis(AX)
        if exist('labels','var')
                for i=2:ix, tm(i)=tm(i)+tm(i-1); end    % reacumular
                for i=labels
                  xpos=i{2}; if xpos==1, xpos= tm(xpos) /2;
                  else,       xpos=(tm(xpos-1)+tm(xpos))/2; end
                  text(xpos,1, i{1},... % 'Color',[1 1 1],...
                        'HorizontalAlignment','center')
        end,    end
end

if NP>2                                         % 3ª figura opcional: flops
subplot(NP,1,3),bar(fp),
                set(gca,'XTick',     1:ix)
                set(gca,'XTickLabel', msg)
                xlabel('SubTask'), ylabel('Flops')
end

cm=white; cm(1,:)=[0.5 0.5 0.5]; colormap(cm);

saveas(gcf,[FNAM '.fig'])
cd(CWD)
```

Listado E.10: Rutina graph.m. Genera gráficas como la 5.5(d).

```
function transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1)
% MATH          Función de apoyo para normalizar transformada
%
%     transform = math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1)
%
%     Organiza Aproximación y Detalles en la forma habitual
%     y centra el tramo [media-stdev media+stdev]
%     en el rango representable [0 1]
%     antes de pesar con la media de la aproximación
%

transform=[[[zeros(size(a3)) dh3;...          % Organización habitual
              dv3    dd3] dh2;...              % salvo hueco aproximación
                 dv2      dd2] dh1;...
                     dv1       dd1];
M      = mean(transform(:));
st     = std (transform(:));                   % normalizar transformada 0..1
transform = (transform-M)/(2*st) + 0.5;        % porque si no no se ve
transform=transform*mean(a3(:));               % Pesarla según peso canal RGB
[x,y] = size(a3); transform(1:x,1:y)=a3;       % Pegarle aproximación
 transform=uint8(transform);
```

Listado E.11: Rutina math.m. Ecualiza el nivel de iluminación de la transformada para que puedan distinguirse visualmente los coeficientes.

07/10/01 17:11:47 pvmtbCode/SecRGB.m 1

```
function [T,R,S]=SecRGB(ntimes,withgraph,withlena)
% SecRGB: Ejemplo secuencial color análisis wavelet multiresolución
%
%       [T R S] = SecRGB ( ntimes, withgraph, withlena )
%
%  ntimes       [1],   número de veces a repetir el trabajo (medir tiempo)
%                0 ,    si se desea consultar datos de ejecución anterior
%  withgraph    0/[1], indica si se desean gráficas de tiempo
%                      el tiempo de gráficas no se incluye, naturalmente
%  withlena     [0]/1, indica si se desean ver las imágenes
%                      el tiempo de dibujo se incluye en la gráfica
%
%  T    "Times", registro con campos:
%  totl tiempo total gastado (no incluye tiempo para gráfica tiempos)
%  spwn 0
%
% R     "Round", tiempo de cada vuelta (de Load a Load)
% S     "Stats", cell-array con columnas {'label' mean stdev ntimes}

%       tomado de Mallat "A Theory for Multiresolution Signal Decomposition:
%                             The Wavelet Representation"
%       IEEE Trans. on Patt.Analysis and Mach.Intell. Vol.11 N.7 July 1989

% Necesita:
% lena_chuck.tif        Imagen de Lena en RGB, obtenida de chuck
% h.mat         Filtro Wavelet Aproximación (unidimensional)
% g.mat         Filtro Wavelet Detalle      (unidimensional)
%
% dwt2          Descomposición Wavelet Bi-dimensional
%   dwt         Descomposición Wavelet Unidimensional
% idwt2         Re-composición Wavelet Bi-dimensional
%   idwt        Re-composición Wavelet Unidimensional
%
% math          Operaciones de normalización para visualizar transformada
% stats         Estadísticas sobre las anotaciones
% record        Anotar tiempos
% label         Anotar etiquetas
% recgraph      Salvar datos gráfica tiempos
% graph         Generación de gráficas a partir de datos

%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<3, withlena =0; else, withlena =min(max(round(withlena ),0),1); end
if nargin<2, withgraph=1; else, withgraph=min(max(round(withgraph),0),1); end
if nargin<1, NTIMES   =1; else, NTIMES   =    max(round(ntimes   ),0   ); end

global DIR, DIR=[ getenv('PVMTB_ROOT') '/Users99/' mfilename ];
if ~NTIMES, [T R S]=shwgraph(withgraph); return, end

tic
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos/flops % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix labels % fp
tm=zeros(1+(19+9*withlena)*NTIMES,6); msg={}; ix=0; labels={};  record('')
% fp=zeros(,1);
%%% 1: ''
%%% 19*:     'Load' 'RD1'-'BD3' (9) 'RR3'-'BR1' (9)
%%% 9*lena*: 'math' 'cat' 'cast' 3*('fig' 'shw')
%%% cada bucle va de 'Load' a 'Load'

%%%%%%%%%%%%%%%%

% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
for indiceNTIMES=1:NTIMES

%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_chuck','tif'); load h, load g
gd=[  g(2:length(g));0];                                     % Detail decomp/recomp
gr=[0;g(1:length(g)-1)];                                     record('Load')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Descomposición 3 BitPlanes, 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
RED  =double(lena(:,:,1));                       record('db');
[Ra1 Rdh1 Rdv1 Rdd1]=dwt2(  RED,h,gd,'spec'); record('D1'),              label
[Ra2 Rdh2 Rdv2 Rdd2]=dwt2(  Ra1,h,gd,'spec'); record('D2'),if NTIMES<3,label,end
[Ra3 Rdh3 Rdv3 Rdd3]=dwt2(  Ra2,h,gd,'spec'); record('D3'),if NTIMES<2,label,end

GREEN=double(lena(:,:,2));                       record('db');
[Ga1 Gdh1 Gdv1 Gdd1]=dwt2(GREEN,h,gd,'spec'); record('D1'),              label
[Ga2 Gdh2 Gdv2 Gdd2]=dwt2(  Ga1,h,gd,'spec'); record('D2'),if NTIMES<3,label,end
[Ga3 Gdh3 Gdv3 Gdd3]=dwt2(  Ga2,h,gd,'spec'); record('D3'),if NTIMES<2,label,end

BLUE =double(lena(:,:,3));                       record('db');
[Ba1 Bdh1 Bdv1 Bdd1]=dwt2( BLUE,h,gd,'spec'); record('D1'),              label
[Ba2 Bdh2 Bdv2 Bdd2]=dwt2(  Ba1,h,gd,'spec'); record('D2'),if NTIMES<3,label,end
[Ba3 Bdh3 Bdv3 Bdd3]=dwt2(  Ba2,h,gd,'spec'); record('D3'),if NTIMES<2,label,end

%%%%%%%%%%%%%%%%%
% Recomposición %
%%%%%%%%%%%%%%%%%
Rr2=idwt2(Ra3,Rdh3,Rdv3,Rdd3, h,gr, 'spec'); record('R3'),if NTIMES<2,label,end
Rr1=idwt2(Rr2,Rdh2,Rdv2,Rdd2, h,gr, 'spec'); record('R2'),if NTIMES<3,label,end
Rr0=idwt2(Rr1,Rdh1,Rdv1,Rdd1, h,gr, 'spec'); record('R1'),              label

Gr2=idwt2(Ga3,Gdh3,Gdv3,Gdd3, h,gr, 'spec'); record('R3'),if NTIMES<2,label,end
Gr1=idwt2(Gr2,Gdh2,Gdv2,Gdd2, h,gr, 'spec'); record('R2'),if NTIMES<3,label,end
Gr0=idwt2(Gr1,Gdh1,Gdv1,Gdd1, h,gr, 'spec'); record('R1'),              label

Br2=idwt2(Ba3,Bdh3,Bdv3,Bdd3, h,gr, 'spec'); record('R3'),if NTIMES<2,label,end
Br1=idwt2(Br2,Bdh2,Bdv2,Bdd2, h,gr, 'spec'); record('R2'),if NTIMES<3,label,end
Br0=idwt2(Br1,Bdh1,Bdv1,Bdd1, h,gr, 'spec'); record('R1'),              label

%%%%%%%%%%%%
% Gráficas %
%%%%%%%%%%%%
if withlena
  Rt=math(Ra3,Rdh3,Rdv3,Rdd3,Rdh2,Rdv2,Rdd2,Rdh1,Rdv1,Rdd1);
  Gt=math(Ga3,Gdh3,Gdv3,Gdd3,Gdh2,Gdv2,Gdd2,Gdh1,Gdv1,Gdd1);
  Bt=math(Ba3,Bdh3,Bdv3,Bdd3,Bdh2,Bdv2,Bdd2,Bdh1,Bdv1,Bdd1);record('mth')
  transform=      cat(3,Rt ,Gt ,Bt ) ;                       record('cat')
  reconstr =uint8(cat(3,Rr0,Gr0,Br0));                       record('u8c')

  figure,                                                    record('fig')
  imshow(lena), title('Original'),                           record('shw')
  figure,                                                    record('fig')
  imshow(transform), title('Transformada'),                  record('shw')
  figure,                                                    record('fig')
  imshow(reconstr), title('Reconstrucción'),                 record('shw')
end

%%%%%%%%%%%%%%%%
```

Listado E.12: SecRGB.m: Programa secuencial MATLAB para la transformada *wavelet* 2D RGB (imagen digitalizada en color).

```
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
end

        toc                                     % La gráfica nunca cuenta
        recgraph;                               % Salvar datos gráfica
[T R S]=shwgraph(withgraph);                    % Ver gráfica si no se impide

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [T,R,S]=shwgraph(withgraph)
global DIR

        HN0=strtok(hostname,'.');
[T R S MSG]=stats(DIR,[mfilename '_' HN0]); disp(MSG), S

if withgraph
  graph(DIR,mfilename,HN0);
end

clear global
```

Listado E.12: Continuación.

07/10/01 20:01:56 pvmtbCode/startup_Slv.m **1**

```
function info = startup_Slv
% STARTUP_SLV           Script para arrancar tareas esclavas
%
%       Pensado para llamarlo desde startup.m al arrancar,
%               si es tarea PVM hija (comprobar PVMEPID)
%       Hecho function para evitar clear -> workspace propio
%       Pensado para seguir protocolo NUMCMDS/QUIT con un proceso maestro

if isempty(getenv('PVMEPID'))
                      error('startup_Slv: uso interno'), end

disp('Arrancando instancia Matlab hija ...')

global TAG RAW PLACE
TAG=7; RAW=1; PLACE=2;                          % Para pvm_recv/initsend
hostname                                        % No liarse con tantos MATLAB

global NUMCMDS indiceBuclePVM                   % global hace NUMCMDS=[]
NUMCMDS=0;                                      % Al desempaquetar, [] no basta
pvm_recv(pvm_parent,TAG); pvm_unpack;           % Se espera variable NUMCMDS
disp('Recibido nº comandos')
pvm_initsend(RAW); pvme_pack(NUMCMDS,QUIT);     % y flag QUIT=0/1
pvm_send(pvm_parent,TAG);                       % Responder mensaje acknowledge
disp('Respuesta de reconocimiento')

if NUMCMDS, for indiceBuclePVM=1:NUMCMDS        % Bucle controlado por NUMCMDS
    indiceBuclePVM                              % Para ver progreso
    pvm_recv(pvm_parent,TAG); pvm_unpack; eval(cmd);
  end
else, indiceBuclePVM=0; QFLG=0; while ~QFLG     % Bucle controlado por quit
    indiceBuclePVM=indiceBuclePVM+1             % Para ver progreso
    pvm_recv(pvm_parent,TAG); pvm_unpack;
        QFLG = (strcmp(cmd,'quit') | strcmp(cmd,'exit') );
    if ~QFLG, eval(cmd); end
  end
end

if QUIT, quit, end                              % Si se indicó salir, hacerlo
```

Listado E.13: *script* startup_Slv .m para PVMTB. Pensado para ejecutarse en procesos esclavos, establece el protocolo NUMCMDS / QUIT con el maestro. Se mostró un fragmento en el Listado 5.6.

06/03/00 13:47:27 **pvmtbCode/Par.m** 1

```
function [T,R,S]=Par(ntimes,withgraph,withdetail,withlena,nslaves)
% Par: Ejemplo paralelo B/N análisis wavelet multiresolución
%
%       [T R S] = Par ( ntimes, withgraph, withdetail, withlena, nslaves)
%
%  ntimes       [1],    número de veces a repetir el trabajo (medir tiempo)
%                0 ,     si se desea consultar datos de ejecución anterior
%  withgraph    0/[1], indica si se desean ver las gráficas de tiempo
%                       el tiempo de gráficas no se incluye, naturalmente
%  withdetail   [0]/1, indica si se desea desglosar tiempo host en paralelo
%  withlena     [0]/1, indica si se desean ver las imágenes
%                       el tiempo de dibujo se incluye en la gráfica
%  nslaves      [3]/4, número de procesos MATLAB esclavos a arrancar
%
%  T    "Times", registro con campos:
%  totl tiempo total gastado (no incluye tiempo para gráfica tiempos)
%  spwn tiempo gastado hasta Spawn MATLAB hijos
%
%  R    "Round", tiempo de cada vuelta (de Load a Load)
%  S    "Stats", cell-array con columnas {'label' mean stdev ntimes}

%       tomado de Mallat "A Theory for Multiresolution Signal Decomposition:
%                        The Wavelet Representation"
%       IEEE Trans. on Patt.Analysis and Mach.Intell. Vol.11 N.7 July 1989
%
%       4 ordenadores (1,2,3,4) en 2 grupos de 2 ordenadores (1,2),(3,4)

% Necesita:
% lena_funet.tif        Imagen de Lena en B/N, obtenida de funet
% h.mat         Filtro Wavelet Aproximación (unidimensional)
% g.mat         Filtro Wavelet Detalle      (unidimensional)
%
% dwt2ParHost   Descomposición paralelizada, lo que hace el host
%   dwt2Par     Lo que hace cada uno de los esclavos
%     dwtflt    dwt con 1 solo filtro
% idwt2ParHost  Re-composición paralelizada, lo que hace el host
%   idwt2Par    Lo que hace cada uno de los esclavos
%     idwtflt   idwt con 1 solo filtro
%
% math          Operaciones de normalización para visualizar transformada
% stats         Estadísticas sobre las anotaciones
% record        Anotar tiempos
% label         Anotar etiquetas
% recgraph      Salvar datos gráfica tiempos
% graph         Generación de gráficas a partir de datos
% timegraph     Gráfica de tiempos correlacionados

%       %%%%%%%%%%% DESCOMPOSICION %%%%%%%%%%%%%%% dwt2Par.m %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 ----         ----         ----
%                            --->h->| 1| ah sup\  /| 1|     ->h-> | 1| ahav
%                           /        ----        \/ |  |----       ----
%                 --------  |        ----        /\ ----|  |       ----
% --------        | img1 | -|  />h->| 2| ah inf/  \     | 2| ->g-> | 2| ahdv
% | img  |  --\   --------   \/      ----               ----       ----
% |      |  --/   --------   /\      ----         ----             ----
% --------        | img2 | -|  \>g->| 3| dh sup\  /| 3|     ->h-> | 3| dhav
%                 --------  |        ----        \/ |  |----       ----
%                           \        ----        /\ ----|  |       ----
%                            --->g->| 4| dh inf/  \     | 4| ->g-> | 4| dhdv
%                                    ----               ----       ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Se filtra primero por filas - a lo largo de las columnas, horizontal
%       El primer grupo (1,2) aproximación (ah) - el segundo (3,4) detalle (dh)
%         El primero del grupo filtra mitad superior imagen - primeras filas
%         El segundo del grupo filtra mitad inferior imagen - últimas  filas
%         Se pasan mutuamente sus resultados para tener todo el filtrado (ah/dh)
%               1 ah superior\/ 1 tiene ah      3 dh superior\/ 3 tiene dh
%               2 ah inferior/\ 2 tiene ah      4 dh inferior/\ 4 tiene dh
%       Se filtra entonces por columnas - a lo largo de las filas, vertical
%         El primero del grupo filtra aproximación xhav
%         El segundo del grupo filtra detalle      xhdv
%               1 ah->ahav     3 dh->dhav
%               2 ah->ahdv     4 dh->dhdv
%       Se devuelve resultado en lhs dwt2Par, no mediante PVM
%        hay dependencia secuencial entre sucesivas descomposiciones
%       Pero si host calcula ahav, podría seguir... muy poca ventaja?!?
%        harían falta más ordenadores... infrautilización

%       %%%%%%%%%%% RE-COMPOSICION %%%%%%%%%%%%%%% idwt2Par.m %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----             ---- <-+--    ----                --------
% | 1| ahav ->h -> | 1|   V      | 1| ah sup ->h -> |     1| ah sup
% ----        *2   |  |----      ----           *2   --------
% ----             ----|  |      ----                --------
% | 2| ahdv ->gr->     | 2|      | 2| ah inf ->h -> |     2| ah inf
% ----        *2       ----      ----           *2   --------
% ----             ---- <-+--    ----                --------
% | 3| dhav ->h -> | 3|   V      | 3| dh sup ->gr-> |     3| dh sup
% ----        *2   |  |----      ----           *2   --------
% ----             ----|  |      ----                --------
% | 4| dhdv ->gr->     | 4|      | 4| dh inf ->gr-> |     4| dh inf
% ----        *2       ----      ----           *2   --------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Cada uno deshace el último filtrado por columnas, vertical
%       El primero manda al segundo la mitad inferior de lo deshecho (viceversa)
%       Cada uno suma la mitad que puede
%               1 ahav->aha (mitad inf)\/(suma sup) ah superior
%               2 ahdv->ahd (mitad sup)/\(suma inf) ah inferior
%               3 dhav->dha (mitad inf)\/(suma sup) dh superior
%               4 dhdv->dhd (mitad sup)/\(suma inf) dh inferior
%       Cada uno deshace el primer filtrado horizontal en la mitad que puede
%         y se lo manda al host para sumar (no merece la pena local/re-enviar)
%               1 ah->a superior---/host suma sup\
%               2 ah->a inferior-\/               \host tiene img
%               3 dh->d superior-/\               /
%               4 dh->d inferior---\host suma inf/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Se devuelve resultado en lhs idwt2Par, no mediante PVM
%        hay dependencia secuencial entre sucesivas re-composiciones
%       No se ve claro cómo se podría seguir sin tener siguiente recomposición

global NTIMES                   % hizo falta para pasar a [i]dwt2ParHost dlbl
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<5, NUMSLV    =3; else, NUMSLV    =min(max(round(nslaves   ),3),4);end
if nargin<4, withlena  =0; else, withlena  =min(max(round(withlena  ),0),1);end
if nargin<3, withdetail=0; else, withdetail=min(max(round(withdetail),0),1);end
if nargin<2, withgraph =1; else, withgraph =min(max(round(withgraph ),0),1);end
if nargin<1, NTIMES    =1; else, NTIMES    =    max(round(ntimes    ),0   );end

global DIR, DIR=[ getenv('PVMTB_ROOT') '/Users99/' mfilename ];
if ~NTIMES, [T R S]=shwgraph(withgraph,withdetail,NUMSLV); return, end

tic
```

Listado E.14: Par.m: Programa maestro PVMTB para transformada *wavelet* paralela de una imagen B/W. Se mostró un fragmento en el Listado 5.7.

06/03/00
13:47:27

pvmtbCode/Par.m

2

```
%%%%%%%%%%%%%%%%%%%%%%%% Número de esclavos==3/4
% Arranque daemon PVM % No sería justo contar tiempo, MPI exige lamboot previo
%%%%%%%%%%%%%%%%%%%%%%%%
if ~pvme_is, error('PVM not initialized, run pvm console or xpvm'), end
if pvm_config~=NUMSLV+1, error('wrong number of computers in PVM'), end
if pvm_setopt(1,3)<0,error('cannot setopt RouteDirect'), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% [x]pvm debería haber arrancado número exacto de hosts
%%%%%%% para que los ficheros se creen con nombres reproducibles
%%%%%%% sin que interfiera el algoritmo round-robin memorioso de PVM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix labels
tm=zeros(2+NTIMES*(1+6*(1+2*withdetail)+8*withlena),6);
                        msg={}; ix=0; labels={};        record('');
%%% 2:          '' 'Spwn'
%%% 1*:              'Load'
%%% 6*(1+2*detl)*: 'D1'-'D3' 'R3'-'R1' cada uno con 'Snd' 'dlbl' 'Rcv' (*3)
%%% 8*lena*: 'math' 'cast' 3*('fig' 'shw')
%%% cada bucle va de 'Load' a 'Load'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arranque MATLAB's esclavos % Número de esclavos==3/4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DISP=getenv('DISPLAY');
% [numt tids]=pvm_spawn('xterm',...
%                       {'-display' DISP '-e' 'matlab'},33,'.',NUMSLV);
  [numt tids]=pvm_spawn('matlab',{'-nosplash'},33,'.',NUMSLV);
if numt~=NUMSLV, error('cannot start MATLABs'), end

global RAW PLACE, RAW=1; PLACE=2;
global TAG NUMCMDS
TAG=7; NUMCMDS=6*NTIMES; QUIT=1;        % Enviaremos 3 des-, 3 re-composiciones
for i=1:numt
  pvm_initsend(RAW); pvme_pack(NUMCMDS,QUIT); pvm_send(tids(i),TAG);
end                                     % Ignorar respuesta confirmación
for i=1:numt, pvm_recv(tids(i),TAG); end
if NUMSLV==3, tids=[pvm_mytid tids]; end        % Añadir host si va a trabajar
                                                    record('Spwn'),label
%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
for indiceNTIMES=1:NTIMES

%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_funet','tif'); load h, load g
gd=[  g(2:length(g));0];                            % Detail decomp/recomp
gr=[0;g(1:length(g)-1)];                            record('Load')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Dependencia secuencial
% Descomposición 3 niveles % Para etiquetar detalle host hay que pasar string
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Si no detalle, se etiqueta desde aquí
wd=withdetail; dl='';                        if  wd,    dl='D1';        end
[a1 dh1 dv1 dd1]=dwt2ParHost(lena,h,gd,tids,dl); if ~wd,record('D1'), label,end
                                             if  wd,    dl='D2';        end
[a2 dh2 dv2 dd2]=dwt2ParHost(  a1,h,gd,tids,dl); if ~wd,record('D2'),
                                                     if NTIMES<3, label,end
                                                                        end
                                                  if  wd,    dl='D3';        end
[a3 dh3 dv3 dd3]=dwt2ParHost(  a2,h,gd,tids,dl); if ~wd,record('D3'),
                                                          if NTIMES<2, label,end
                                                                             end
%%%%%%%%%%%%%%%%%
% Recomposición % Dependencia secuencial
%%%%%%%%%%%%%%%%%
                                                 if  wd,     dl='R3';        end
r2=idwt2ParHost(a3,dh3,dv3,dd3, h,gr, tids,dl); if ~wd, record('R3'),
                                                          if NTIMES<2, label,end
                                                                             end
                                                 if  wd,     dl='R2';        end
r1=idwt2ParHost(r2,dh2,dv2,dd2, h,gr, tids,dl); if ~wd, record('R2'),
                                                          if NTIMES<3, label,end
                                                                             end
                                                 if  wd,     dl='R1';        end
r0=idwt2ParHost(r1,dh1,dv1,dd1, h,gr, tids,dl); if ~wd, record('R1'), label,end

%%%%%%%%%%%%
% Gráficas %
%%%%%%%%%%%%
if withlena
 transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);record('math')
  reconstr=uint8(r0);                                    record('cast')

  figure,                                                record('fig')
  imshow(lena), title('Original'),                       record('shw')
  figure,                                                record('fig')
  imshow(transform), title('Transform'),                 record('shw')
  figure,                                                record('fig')
  imshow(reconstr), title('Reconstruction'),             record('shw')
end

%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
end

%%%%%%%%%%%%%%
% Acabar PVM %
%%%%%%%%%%%%%%
pvm_exit;                % pvm_halt mataría xpvm, exit sirve para xpvm

        toc                                     % La gráfica nunca cuenta
        recgraph;                               % Salvar datos gráfica
[T R S]=shwgraph(withgraph,withdetail,NUMSLV);  % Ver gráfica si no se impide

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [T,R,S]=shwgraph(withgraph,withdetail,NUMSLV)
global DIR

         HN0=strtok(hostname,'.');
if withdetail
        [HN1 HN2 HN3 HN4]=pvmHnames;            % (confiar en $PVMBHOST)
[T R S MSG tm1 lbl1]=stats(DIR,['idwt2Par_' HN1]); disp(MSG), S
[T R S MSG tm2 lbl2]=stats(DIR,['idwt2Par_' HN2]); disp(MSG), S
[T R S MSG tm3 lbl3]=stats(DIR,['idwt2Par_' HN3]); disp(MSG), S, if NUMSLV==4
[T R S MSG tm4 lbl4]=stats(DIR,['idwt2Par_' HN4]); disp(MSG), S, end
end
[T R S MSG tm0 lbl0 msg]=stats(DIR,[mfilename '_' HN0]); disp(MSG), S

if withgraph
  graph(DIR,mfilename,HN0);                     % Ver gráfica si no se impide
```

Listado E.14: Continuación.

06/03/00 13:47:27 — pvmtbCode/Par.m — 3

```
 if withdetail
        args={DIR [mfilename '_corr'] MSG 'D1S' msg};
        args={args{:} HN0 tm0 lbl0 HN1 tm1 lbl1 HN2 tm2 lbl2 HN3 tm3 lbl3};
    graph(DIR,'idwt2Par',HN1);
    graph(DIR,'idwt2Par',HN2);
    graph(DIR,'idwt2Par',HN3); if NUMSLV==4
    graph(DIR,'idwt2Par',HN4);
    args = {args{:} HN4 tm4 lbl4}; end
        timegraph(args{:});
 end
end

clear global
```

Listado E.14: Continuación.

05/15/00
17:16:46

pvmtbCode/dwt2ParHost.m

1

```
function [ahav, ahdv, dhav, dhdv]=dwt2ParHost(img, h,g, tids, dlbl)
% DWT2PARHOST           Parallel DWT, Host Sequence
%
%       [ahav ahdv dhav dhdv] = dwt2ParHost ( img, h,g, tids, dlbl )
%
%  img (matriz) Señal 2-D a analizar. Se devuelven las 4 componentes
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  g   (vector) Filtro 1-D detalle      (paso alta) para descomposición. lh=lg
%  tids(vector) 4 tids PVM que colaboran, esta función la ejecuta el host
%  dlbl (str)   Detail label:si no vacía, indica que se desea desglosar tiempos
%
%  ahav(matriz) Señal 2-D aproximación  (coeficientes). Area 1/4 img
%  ahdv(matriz) Señal 2-D aprox. horiz, detalle vert. (bordes horiz)
%  dhav(matriz) Señal 2-D detalle horiz, aprox. vert. (bordes vert)
%  dhdv(matriz) Señal 2-D detalle horiz, detalle vert.(bordes diagonal)
%

rf=~isempty(dlbl);                                      % Record flag
if rf, global tm msg ix DIR labels, else, DIR=''; end
global TAG RAW PLACE NTIMES
%%%%%%%%%%%%
% Longitudes
%%%%%%%%%%%%
lih =size(img,2); liv =size(img,1);     % Length image horz/vert
lih2=ceil(lih/2); liv2=ceil(liv/2);     % Al decimar puede salir 1 de más
if length(tids)~=4, error('requiere 4 tareas PVM esclavas'), end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Enviar mitades imagen % Tareas trabajan en 2 grupos de 2 tids
%%%%%%%%%%%%%%%%%%%%%%%%%
T1=int2str(tids(1));    T2=int2str(tids(2));
T3=int2str(tids(3));    T4=int2str(tids(4));
DL=['''' DIR ''','''' dlbl ''''];

img1=double(img(     1:liv2,:)); siz1=size(img1);       % Mitad sup. a primeros
img2=double(img(liv2+1:liv ,:)); siz2=size(img2);       % Mitad inf. a segundos

if tids(1)~=pvm_mytid                           % Puede que host trabaje también
cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img1'');',...
     'dwt2Par(img1, h,h,' T2 ', 0, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(h,  cmd); pvm_send(tids(1),TAG);
        pvm_initsend(PLACE); pvm_pack(img1);   pvm_send(tids(1),TAG);
end

cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img2'');',...
     'dwt2Par(img2, h,g,' T1 ', 1, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(h,g,cmd); pvm_send(tids(2),TAG);
        pvm_initsend(PLACE); pvm_pack(img2);   pvm_send(tids(2),TAG);

cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img1'');',...
     'dwt2Par(img1, g,h,' T4 ', 0, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(g,h,cmd); pvm_send(tids(3),TAG);
        pvm_initsend(PLACE); pvm_pack(img1);   pvm_send(tids(3),TAG);

cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img2'');',...
     'dwt2Par(img2, g,g,' T3 ', 1, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(  g,cmd); pvm_send(tids(4),TAG);
        pvm_initsend(PLACE); pvm_pack(img2);   pvm_send(tids(4),TAG);

                                                if rf,record([dlbl 'S']),end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Tiempo muerto o trabajamos ? %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                  ----           ----           ----
%                           --->h->| 1| ah sup\  /| 1|     ->h-> | 1| ahav
%                          /       ----        \/ |  |----       ----
%                --------  |       ----        /\ ----|  |       ----
% --------       | img1 | -|  />h->| 2| ah inf/  \    | 2| ->g-> | 2| ahdv
% | img  |  --\  --------   \/     ----               ----       ----
% |      |  --/  --------   /\     ----           ----           ----
% --------       | img2 | -|  \>g->| 3| dh sup\  /| 3|     ->h-> | 3| dhav
%                --------  |       ----        \/ |  |----       ----
%                          \       ----        /\ ----|  |       ----
%                           --->g->| 4| dh inf/  \    | 4| ->g-> | 4| dhdv
%                                  ----               ----       ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if tids(1)==pvm_mytid                       % Puede que host trabaje también
ahav=dwt2Par(img1, h,h,tids(2), 0, rf,dlbl);
else                                        % Si tiempo muerto, recibir ahav

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recibir Transformada, Pre-allocation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pvm_recv(tids(1),TAG); pvm_unpack('ahav'); end
pvm_recv(tids(2),TAG); pvm_unpack('ahdv');
pvm_recv(tids(3),TAG); pvm_unpack('dhav');
pvm_recv(tids(4),TAG); pvm_unpack('dhdv');

                                                if rf,record([dlbl 'R'])
     dl=dlbl(end); if tids(1)==pvm_mytid, dl ='0';end   % Host ya anota D1x...
                              if dl=='1'|(dl=='2'&NTIMES<3)|...
                                         (dl=='3'&NTIMES<2), label, end, end
```

Listado E.15: dwt2ParHost.m: Rutina PVMTB de distribución para análisis *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.8.

05/15/00 17:37:12 — pvmtbCode/dwt2Par.m — 1

```
function res=dwt2Par(img, f1,f2, tid,slv, wd,dlbl)
% DWT2PAR               Parallel DWT, Slave sequence
%
%       res = dwt2Par ( img, f1,f2, tid,slv, wd,dlbl )
%
%  img (matriz) Señal 2-D mitad superior/inferior
%  f1  (vector) Filtro 1-D primera etapa (horizontal)
%  f2  (vector) Filtro 1-D segunda etapa (vertical)
%
%  tid (int)    tid de la tarea PVM que tiene la otra mitad
%  slv (int)    0/1, indica si somos la 2ª mitad
%
%  wd           0/1, indica si se ha de etiquetar (withdetail para host)
%  wd (string)  Directorio salvar datos/gráficas (work dir para esclavo)
%               '' si no se desea escribir en disco nada
%  dlbl (str)   Prefijo de etiquetado
%
%       El resultado (res) se devuelve mediante PVM a la tarea madre
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                           ----         ----         ----
%                       --->h->| 1| ah sup\  /| 1|     ->h-> | 1| ahav
%                      /       ----        \/ |  |----       ----
%             --------  |       ----        /\ ----|  |       ----
% --------    | img1 | -|  />h->| 2| ah inf/  \    | 2| ->g-> | 2| ahdv
% | img  |  --\ --------   \/     ----              ----       ----
% |      |  --/ --------   /\     ----         ----         ----
% --------    | img2 | -|  \>g->| 3| dh sup\  /| 3|     ->h-> | 3| dhav
%             --------  |       ----        \/ |  |----       ----
%                      \       ----        /\ ----|  |       ----
%                       --->g->| 4| dh inf/  \    | 4| ->g-> | 4| dhdv
%                           ----              ----       ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global TAG RAW PLACE                    % puede pillarlo del host o del esclavo
global NUMCMDS indiceBuclePVM           % puede pillarlo del host o del esclavo
%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix DIR labels             % a esclavos no llega global host
if ~ischar(wd), rf=wd;                  % HOST: record flag <=> wd(with detail)
else,  DIR=wd;  rf=~isempty(DIR);       % SLAV: record <=> ''~= wd (work dir)
     if rf
        if isempty(ix)                  % Si 1ª vez del esclavo, reservar anot.
        tm=zeros(4*NUMCMDS,6); msg={}; ix=0; labels={}; end,   record('')
     end
end
%%% 4: 'Row' 'xchg' 'Col' 'Snd'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Longitudes, Constantes, etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lih =size(img,2); liv2=size(img,1);     % Length image horz/vert
lih2=ceil(lih/2); liv =2*liv2;          % Se le pasó una mitad de la imagen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtrado por filas, 1ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tmp=zeros(liv,lih2);                    % Pre-allocation
for row=1:liv2                          % Sólamente una mitad img1/img2
  r  = img(row,:);                      % Obtener img->ah/dh, según f1==h/g
  rf1= dwtflt(r',f1, 'spec');           %    sólo mitad sup/inf de ah/dh
  tmp(slv*liv2+row,:) = rf1';           % primera ó segunda según slv==0/1
end
                                        if rf, record([dlbl 'r']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Envío/Recepción  intermedio
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
img2=tmp(slv*liv2+1:(slv+1)*liv2, : );                  % Dicha mitad
pvm_initsend(PLACE); pvm_pack(img2); pvm_send(tid,TAG); % mandarla al otro

pvm_recv(tid,TAG); pvm_unpack('img2');                  % mitad del compañero
tmp((1-slv)*liv2+1:(2-slv)*liv2, : )=img2;              % pegarla antes/después
                                        if rf, record([dlbl 'x'])
  dl=dlbl(end); nt=NUMCMDS/6;   if dl=='1'|(dl=='2'&nt<3)|...
                                           (dl=='3'&nt<2), label, end, end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtrado por columnas, 2ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
res=zeros(liv2,lih2);                   % Pre-allocation
for col=1:lih2                          % Toda la columna
  c  = tmp(:,col);                      % Obtener tmp->xhav/xhdv según f2==h/g
  cf2= dwtflt(c ,f2, 'spec');           %    toda la columna de tmp==xh (ah/dh)
  res(:,col)= cf2;
end
                                        if rf, record([dlbl 'c']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Envío Transformada al padre %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if pvm_parent>0                         % Por si acaso el host también trabaja
  pvm_initsend(PLACE); pvm_pack(res);   % Observar recgraph destroza tm, msg...
  pvm_send(pvm_parent,TAG);             if rf, record([dlbl 's'])
                        if indiceBuclePVM==NUMCMDS, recgraph, end, end
end
```

Listado E.16: dwt2Par.m: Esclavo PVMTB para análisis *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.9.

03/06/00
11:05:43

pvmtbCode/dwtflt.m

1

```
function z=dwtflt(x,f, varargin)
% DWTFLT                Filtrado y Decimación (downsample) para Transf.Wavelet
%
%       z = dwtflt ( x, f [,opt] )
%
%  x (vector)   señal 1-D a filtrar
%  f (vector)   respuesta impulsiva filtro wavelet (aprox/detalle) para conv
%  opt  (str)   acción para evitar efecto bordes convolución
%               'zero' -> añadir ceros a los bordes (por defecto)
%               'spec' -> añadir reflejos especulares de la señal
%
%  z (vector)   señal resultado (aproximación/detalle), según f=h/g
%

%%%%%%%%
% ArgChk
%%%%%%%% lf2 mitad filtro. Si lf impar, lf2 es longitud a ambos lados centro
lx=length(x); lf=length(f); lf2=floor(lf/2);    % longitudes
mode='zero';                                    % opción bordes
if ~isempty(varargin)
  if isstr(varargin{1})
    switch varargin{1}
      case {'spec','zero'},   mode=varargin{1};
      otherwise,              warning('modo no reconocido')
    end
  else,         warning('argumentos adicionales no string')
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2 reflejos de tamaño lf2, Convolución, Centrado y Downsample
% ly == lx + 2lf2
% lz == ly + lf - 1 == lx + 2lf2 + lf - 1
% dejarlo en lx, y luego en lx/2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if strcmp(mode,'spec'),  y=[x(lf2+1:-1:2); x ; x(lx-1:-1:lx-lf2)];
else,                    y=[zeros(lf2, 1); x ; zeros(lf2,1)]; end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación reflejo: Si lf impar,
%  cuando centro filtro sobre x(1), cola filtro x(2)...x(lf2+1)
% Añadir lf2 muestras reflejo (2..lf2+1), sin repetir x(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
z=conv(y,f);            % Filtrado
z=z(lf:lx+lf-1);        % Corte correspondiente
z=z(1:2:lx);            % Downsample. Puede salir ceil(lx/2) una de más

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación correspondencia entre señales:
% Señal original:              x(1)...x(     lx)
% Efecto bordes:   y(1)... y(lf2+1)...y( lf2+lx)...y(2lf2+lx)
% Conv: z(1)...z(lf2+1)...z(2lf2+1)...z(2lf2+lx)...z(3lf2+lx)...z(4lf2+lx)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf impar:          ...z(lf)   ...z(lx+lf-1)...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf par:            ...z(lf+1) ...z(lx+lf)  ...
%  debería cortarse entre   (lf+1  ...  lx+lf)  ???
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listado E.17: dwtflt .m: Rutina MATLAB para análisis *wavelet* 1D B/W.

05/15/00
17:14:50

pvmtbCode/idwt2ParHost.m

1

```
function r=idwt2ParHost(ahav,ahdv,dhav,dhdv, h,g, tids, dlbl)
% IDWT2PARHOST          Parallel IDWT, Host Sequence
%
%       r = idwt2ParHost ( ahav,ahdv,dhav,dhdv, h,g, tids, dlbl )
%
%  ahav(matriz) Señal 2-D aproximación  (coeficientes)
%  ahdv(matriz) Señal 2-D aprox. horiz, detalle vert. (bordes horiz)
%  dhav(matriz) Señal 2-D detalle horiz, aprox. vert. (bordes vert)
%  dhdv(matriz) Señal 2-D detalle horiz, detalle vert.(bordes diagonal)
%
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  g   (vector) Filtro 1-D detalle      (paso alta) para re-composición. lh=lg
%  tids(vector) 4 tids PVM que colaboran, esta función la ejecuta el host
%  dlbl (str)   Detail label:si no vacía, indica que se desea desglosar tiempos
%
%  img (matriz) Señal 2-D reconstruida. Area = 4 veces ahav
%

rf=~isempty(dlbl);                                        % Record flag
if rf, global tm msg ix DIR labels, else, DIR=''; end
global TAG RAW PLACE NTIMES
%%%%%%%%%%%%
% Longitudes
%%%%%%%%%%%%
l2=size(ahav);
if any(l2~=size(ahdv))|...
   any(l2~=size(dhav))|...
   any(l2~=size(dhdv)), error('imagenes de distintos tamaños'), end
lih2=l2(2);  liv2=l2(1);
lih =lih2*2; liv =liv2*2;
if length(tids)~=4, error('requiere 4 tareas PVM esclavas'), end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Repartir descomposiciones % Tareas trabajan en 2 grupos de 2 tids
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DL=['''' DIR ''',''' dlbl ''''];

if tids(1)~=pvm_mytid                   % Puede que host trabaje también
cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''ahav'');',...
     'idwt2Par(ahav, h,h,' int2str(tids(2)) ', 0, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(h,  cmd); pvm_send(tids(1),TAG);
        pvm_initsend(PLACE); pvm_pack(ahav);   pvm_send(tids(1),TAG);
end
cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''ahdv'');',...
     'idwt2Par(ahdv, h,g,' int2str(tids(1)) ', 1, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(h,g,cmd); pvm_send(tids(2),TAG);
        pvm_initsend(PLACE); pvm_pack(ahdv);   pvm_send(tids(2),TAG);

cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''dhav'');',...
     'idwt2Par(dhav, g,h,' int2str(tids(4)) ', 0, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(g,h,cmd); pvm_send(tids(3),TAG);
        pvm_initsend(PLACE); pvm_pack(dhav);   pvm_send(tids(3),TAG);

cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''dhdv'');',...
     'idwt2Par(dhdv, g,g,' int2str(tids(3)) ', 1, ' DL ');'];
        pvm_initsend(RAW); pvme_pack(  g,cmd); pvm_send(tids(4),TAG);
        pvm_initsend(PLACE); pvm_pack(dhdv);   pvm_send(tids(4),TAG);

                                                    if rf,record([dlbl 'S']),end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Tiempo muerto o trabajamos ? %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----              ---- <-+--    ----               --------
% | 1| ahav ->h -> | 1|   V      | 1| ah sup ->h -> |     1| a sup\
% ----        *2   |  |----      ----          *2   --------       \      r
% ----           ----|  |        ----               --------        \  --------
% | 2| ahdv ->g ->     | 2|      | 2| ah inf ->h -> |     2| a inf\  >-|      |
% ----        *2       ----      ----          *2   --------       \/  |______|
% ----           ---- <-+--      ----               --------       /\  |      |
% | 3| dhav ->h -> | 3|   V      | 3| dh sup ->g -> |     3| d sup/  >-|      |
% ----        *2   |  |----      ----          *2   --------       /  --------
% ----           ----|  |        ----               --------      /
% | 4| dhdv ->g ->     | 4|      | 4| dh inf ->g -> |     4| d inf/
% ----        *2       ----      ----          *2   --------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recibir Reconstrucción, Pre-allocation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r  =zeros(liv ,lih);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if tids(1)==pvm_mytid                  % Puede que host trabaje también
tmp=idwt2Par(ahav, h,h, tids(2), 0, rf,dlbl);
else                                   % Si tiempo muerto, recibir ah sup
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pvm_recv(tids(1),TAG); pvm_unpack('tmp'); end                     % ah    sup
pvm_recv(tids(3),TAG); pvm_unpack('res'); r(     1:liv2,:)=tmp+res; % ah+dh sup

pvm_recv(tids(2),TAG); pvm_unpack('tmp');                          % ah    inf
pvm_recv(tids(4),TAG); pvm_unpack('res'); r(liv2+1:liv ,:)=tmp+res; % ah+dh inf

                                                 if rf,record([dlbl 'R'])
     dl=dlbl(end); if tids(1)==pvm_mytid, dl ='0';end   % Host ya anota D1x...
                              if dl=='1'|(dl=='2'&NTIMES<3)|...
                                         (dl=='3'&NTIMES<2), label, end, end
```

Listado E.18: idwt2ParHost.m: Rutina PVMTB de distribución para reconstrucción *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.10.

05/15/00
16:42:43

pvmtbCode/idwt2Par.m

1

```
function res=idwt2Par(img, f1,f2, tid,flg, wd,dlbl)
% IDWT2PAR               Parallel IDWT, Slave sequence
%
%       res = idwt2Par ( img, f1,f2, tid,flg, wd,dlbl )
%
%  img (matriz) Señal 2-D descomposición wavelet
%  f1  (vector) Filtro 1-D primera etapa (horizontal)
%  f2  (vector) Filtro 1-D segunda etapa (vertical)
%
%  tid (int)    de la tarea PVM con la cual se coopera
%  flg (int)    0/1, indica si somos la 2ª descomposición del grupo
%
%  wd           0/1, indica si se ha de etiquetar (withdetail para host)
%  wd (string)  Directorio salvar datos/gráficas (work dir para esclavo)
%               '' si no se desea escribir en disco nada
%  dlbl (str)   Prefijo de etiquetado
%
%       El resultado (res) se devuelve mediante PVM a la tarea madre
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----            ---- <-+--    ----               --------
% | 1| ahav ->h -> | 1|   V      | 1| ah sup ->h -> |     1| a sup\
% ----        *2   |  |----      ----          *2   --------       \      r
% ----            ----|  |      ----               --------        \  --------
% | 2| ahdv ->g ->     | 2|      | 2| ah inf ->h -> |     2| a inf\  >-|      |
% ----        *2       ----      ----          *2   --------       \/  |______|
% ----            ---- <-+--    ----               --------       /\  |      |
% | 3| dhav ->h -> | 3|   V      | 3| dh sup ->g -> |     3| d sup/  >-|      |
% ----        *2   |  |----      ----          *2   --------        /  --------
% ----            ----|  |      ----               --------       /
% | 4| dhdv ->g ->     | 4|      | 4| dh inf ->g -> |     4| d inf/
% ----        *2       ----      ----          *2   --------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global TAG RAW PLACE                   % puede pillarlo del host o del esclavo
global NUMCMDS indiceBuclePVM          % puede pillarlo del host o del esclavo
%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix DIR labels            % a esclavos no llega global host
if ~ischar(wd), rf=wd;                 % HOST: record flag <=> wd(with detail)
else,  DIR=wd;  rf=~isempty(DIR);      % SLAV: record <=> ''~= wd (work dir)
     if rf
        if isempty(ix)                 % Si 1ª vez del esclavo, reservar anot.
        tm=zeros(4*NUMCMDS,6); msg={}; ix=0; labels={}; end,   record('')
     end
end
%%% 4: 'c' 'X' 'r' 's'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Longitudes, Constantes, etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lih2=size(img,2); liv2=size(img,1);    % Length image horz/vert
lih =2*lih2;      liv =2*liv2;         % Se le pasó un cuarto de imagen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Deshacer filtrado por columnas, 2ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tmp=zeros(liv,lih2);                   % Pre-allocation
for col=1:lih2                         % Hace su mitad completa
  c  = img(:,col);                     % P.ej.: si éste recibió ahav (tids(1))
  cf2= idwtflt(c ,f2, 'spec');         %  el otro (tids(2)) tiene ahdv
  tmp(:,col) = cf2;                    % Pasar siempre columnas
end
                                        if rf, record([dlbl 'c']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Sumar una mitad con mitad del otro
% Envío/Recepción  intermedio % Ej.: tids(1) ahav->aha- mitades superiores ah
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%     tids(2) ahdv->ahd/         inferiores ah
 una=tmp(   flg *liv2+1:(flg+1)*liv2, : );              % mitad    interesante
otra=tmp((1-flg)*liv2+1:(2-flg)*liv2, : );              % mitad no interesante
pvm_initsend(PLACE); pvm_pack(otra); pvm_send(tid,TAG); % mandarla al otro

pvm_recv(tid,TAG); pvm_unpack('otra');                  % Recibir sobre otra
tmp = una+otra;                                         % y sumarla
                                        if rf, record([dlbl 'x'])
  dl=dlbl(end); nt=NUMCMDS/6;   if dl=='1'|(dl=='2'&nt<3)|...
                                           (dl=='3'&nt<2), label, end, end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Deshacer filtrado por filas, 1ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
res=zeros(liv2,lih);                    % Pre-allocation
for row=1:liv2                          % La mitad sup/inf? lo que haya en tmp
  r  = tmp(row,:);                      % Obtener a/d según f2==h/g
  rf1= idwtflt(r',f1, 'spec');          %    toda la fila de tmp==(ah/dh)(s/inf)
  res(row,:)= rf1';                     % Pasar siempre columnas
end
                                        if rf, record([dlbl 'r']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Envío Transformada al padre %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% res=uint8(res);                       % No se podría sumar luego en host
if pvm_parent>0
  pvm_initsend(PLACE); pvm_pack(res);   % Por si acaso el host trabaja
  pvm_send(pvm_parent,TAG);             if rf, record([dlbl 's'])
                        if indiceBuclePVM==NUMCMDS, recgraph, end, end
end
```

Listado E.19: idwt2Par.m: Esclavo PVMTB para reconstrucción *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.11.

03/07/00 11:20:26 pvmtbCode/idwtflt.m **1**

```
function z=idwtflt(x,f, varargin)
% IDWTFLT              Interpolación (upsample) y Filtrado para Transf.Wavelet
%
%       z = idwtflt ( x, f [,opt] )
%
%  x (vector)   señal 1-D a filtrar
%  f (vector)   respuesta impulsiva filtro wavelet (aprox/detalle) para conv
%  opt  (str)   acción para evitar efecto bordes convolución
%               'zero' -> añadir ceros a los bordes (por defecto)
%               'spec' -> añadir reflejos especulares de la señal
%
%  z (vector)   señal resultado (reconstrucción a partir de aprox/detalle)
%

%%%%%%%%
% ArgChk
%%%%%%%% lf2 mitad filtro. Si lf impar, lf2 es longitud a ambos lados centro
lx=length(x); lf=length(f); lf2=floor(lf/2);    % longitudes
mode='zero';                                     % opción bordes
if ~isempty(varargin)
  if isstr(varargin{1})
    switch varargin{1}
      case {'spec','zero'},   mode=varargin{1};
      otherwise,              warning('modo no reconocido')
    end
  else,         warning('argumentos adicionales no string')
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Upsample, 2 reflejos de tamaño lf2, Convolución y Centrado
% ly == 2lx + 2lf2
% lz == ly + lf - 1 == 2lx + 2lf2 + lf - 1
% dejarlo en 2lx
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y=zeros(2*lx,1);                % Pre-alloc
y(1:2:2*lx)=x;                  % Upsample. x(1),0,x(2),0...x(lx),0
if strcmp(mode,'spec'),  y=[y(lf2+1:-1:2); y ; y(2*lx-1:-1:2*lx-lf2)];
else,                    y=[zeros(lf2, 1); y ; zeros(lf2,1)];     end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación reflejo: Si lf impar,
%  cuando centro filtro sobre y(1), cola filtro y(2)...y(lf2+1)
% Añadir lf2 muestras reflejo (2..lf2+1), sin repetir y(1)==x(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
z=conv(y,f);                    % Filtrado y recuperación normalización R^2
z=2*z(lf:2*lx+lf-1);            % Corte correspondiente. Sale 2lx

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Explicación correspondencia entre señales:
% Señal original:              x(1)...x(lx)
% Upsample:               y/x(1),0,...x(lx),y(2lx)==0
% Efecto bordes:   y(1)...y(lf2+1) ...      y(2lx+ lf2)...y(2lx+2lf2)
% Conv: z(1)...z(lf2+1)...z(2lf2+1)...      z(2lx+2lf2)...z(2lx+3lf2)...+4lf2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf impar:          ...z(lf)    ...      z(2lx+lf-1)...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Si lf par:            ...z(lf+1)  ...      z(2lx+lf)  ...
%  debería cortarse entre  (lf+1   ...        2lx+lf)  ???
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listado E.20: idwtflt .m: Rutina MATLAB para reconstrucción *wavelet* 1D B/W.

07/10/01 21:06:36 mpitbCode/startup_Slv.m **1**

```
function info = startup_Slv
% STARTUP_SLV           Script para arrancar tareas esclavas
%
%       Pensado para llamarlo desde startup.m al arrancar,
%               si es tarea MPI hija (comprobar LAMPARENT)
%       Hecho function para evitar clear -> workspace propio
%       Pensado para seguir protocolo NUMCMDS/QUIT con un proceso maestro
%       Observar que MPI_Init es la primera llamada MPI
%               eso desbloquea el MPI_Spawn

if isempty(getenv('LAMPARENT'))
                        error('startup_Slv: uso interno')
elseif strcmp(getenv('LAMPARENT'),'0')
                        warning('Parece arrancado con XMPI?'), return, end

disp('Arrancando instancia Matlab hija ...')

info=MPI_Init;
if info~=MPI_SUCCESS,   error('startup_mm: MPI_Init: no se arrancó LAM?'), end

MPI_Errhandler_set('WORLD','RETURN');           % Manía de abortar MPI
[info parent] =MPI_Comm_get_parent;
global NEWORLD                                  % para pasar a tareas
[info NEWORLD]=MPI_Intercomm_merge(parent,1);   % ponerse detrás en comunicador
MPI_Errhandler_set(NEWORLD,'RETURN');

global TAG
TAG=7;                                          % Para MPI_Send/Recv
hostname                                        % No liarse con tantos MATLAB

global NUMCMDS indiceBucleMPI
NUMCMDS=0; QUIT=0;                              % Usar variables directamente
MPI_Recv(NUMCMDS,0,TAG,NEWORLD);                % se espera NUMCMDS
MPI_Recv(QUIT,   0,TAG,NEWORLD);                % y QUIT
disp('Recibido nº comandos')
MPI_Send(NUMCMDS,0,TAG,NEWORLD);                % Responder mensaje acknowledge
disp('Respuesta de reconocimiento')

for indiceBucleMPI=1:NUMCMDS                    % Bucle de respuesta
  indiceBucleMPI                                % Para ver progreso
  [info stat]=MPI_Probe(-1,-1,NEWORLD);         % averiguar tamaño antes
  [info elem]=MPI_Get_elements(stat,[]);        % datos vienen empaquetados
  buffer=zeros(1,elem);                         % hacer sitio
  MPI_Recv(buffer,0,TAG,NEWORLD);
  MPI_Unpack(buffer,0,NEWORLD);                 % desempaquetar desde principio
  eval(cmd);
end

if QUIT, quit, end                              % Si se indicó salir, hacerlo
```

Listado E.21: *script* startup_Slv .m para MPITB. Pensado para ejecutarse en procesos esclavos, establece el protocolo NUMCMDS / QUIT con el maestro. Se mostró un fragmento en el Listado 5.12.

05/16/00 13:53:24 mpitbCode/Par.m **1**

```
function [T,R,S]=Par(ntimes,withgraph,withdetail,withlena,nslaves)
% Par: Ejemplo paralelo B/N análisis wavelet multiresolución
%
%       [T R S] = Par ( ntimes, withgraph, withdetail, withlena, nslaves)
%
%  ntimes       [1],   número de veces a repetir el trabajo (medir tiempo)
%                0 ,    si se desea consultar datos de ejecución anterior
%  withgraph    0/[1], indica si se desean ver las gráficas de tiempo
%                      el tiempo de gráficas no se incluye, naturalmente
%  withdetail   [0]/1, indica si se desea desglosar tiempo host en paralelo
%  withlena     [0]/1, indica si se desean ver las imágenes
%                      el tiempo de dibujo se incluye en la gráfica
%  nslaves      [3]/4, número de procesos MATLAB esclavos a arrancar
%
%  T    "Times", registro con campos:
%  totl tiempo total gastado (no incluye tiempo para gráfica tiempos)
%  spwn tiempo gastado hasta Spawn MATLAB hijos
%
%  R    "Round", tiempo de cada vuelta (de Load a Load)
%  S    "Stats", cell-array con columnas {'label' mean stdev ntimes}

%       tomado de Mallat "A Theory for Multiresolution Signal Decomposition:
%                        The Wavelet Representation"
%       IEEE Trans. on Patt.Analysis and Mach.Intell. Vol.11 N.7 July 1989
%
%       4 ordenadores (1,2,3,4) en 2 grupos de 2 ordenadores (1,2),(3,4)

% Necesita:
% lena_funet.tif        Imagen de Lena en B/N, obtenida de funet
% h.mat         Filtro Wavelet Aproximación (unidimensional)
% g.mat         Filtro Wavelet Detalle      (unidimensional)
%
% dwt2ParHost   Descomposición paralelizada, lo que hace el host
%   dwt2Par     Lo que hace cada uno de los esclavos
%     dwtflt    dwt con 1 solo filtro
% idwt2ParHost  Re-composición paralelizada, lo que hace el host
%   idwt2Par    Lo que hace cada uno de los esclavos
%     idwtflt   idwt con 1 solo filtro
%
% math          Operaciones de normalización para visualizar transformada
% stats         Estadísticas sobre las anotaciones
% record        Anotar tiempos
% label         Anotar etiquetas
% recgraph      Salvar datos gráfica tiempos
% graph         Generación de gráficas a partir de datos
% timegraph     Gráfica de tiempos correlacionados

%       %%%%%%%%%%% DESCOMPOSICION %%%%%%%%%%%%%%% dwt2Par.m %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 ----          ----          ----
%                             --->h->| 1| ah sup\  /| 1|      ->h-> | 1| ahav
%                            /       ----        \/ |  |----        ----
%               --------    |        ----        /\ ----|  |        ----
% --------      | img1 | -|  />h->| 2| ah inf/  \     | 2| ->g-> | 2| ahdv
% | img  |  --\ --------    \/       ----                ----        ----
% |      |  --/ --------    /\       ----          ----              ----
% --------      | img2 | -|  \>g->| 3| dh sup\  /| 3|      ->h-> | 3| dhav
%               --------    |        ----        \/ |  |----        ----
%                            \       ----        /\ ----|  |        ----
%                             --->g->| 4| dh inf/  \     | 4| ->g-> | 4| dhdv
%                                    ----                ----        ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Se filtra primero por filas - a lo largo de las columnas, horizontal
%       El primer grupo (1,2) aproximación (ah) - el segundo (3,4) detalle (dh)
%         El primero del grupo filtra mitad superior imagen - primeras filas
%         El segundo del grupo filtra mitad inferior imagen - últimas  filas
%         Se pasan mutuamente sus resultados para tener todo el filtrado (ah/dh)
%               1 ah superior\/ 1 tiene ah      3 dh superior\/ 3 tiene dh
%               2 ah inferior/\ 2 tiene ah      4 dh inferior/\ 4 tiene dh
%       Se filtra entonces por columnas - a lo largo de las filas, vertical
%         El primero del grupo filtra aproximación xhav
%         El segundo del grupo filtra detalle      xhdv
%               1 ah->ahav      3 dh->dhav
%               2 ah->ahdv      4 dh->dhdv
%       Se devuelve resultado en lhs dwt2Par, no mediante MPI
%        hay dependencia secuencial entre sucesivas descomposiciones
%       Pero si host calcula ahav, podría seguir... muy poca ventaja?!?
%        harían falta más ordenadores... infrautilización

%       %%%%%%%%%%% RE-COMPOSICION %%%%%%%%%%%%%%% idwt2Par.m %%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----             ---- <-+--     ----              --------
% | 1| ahav ->h -> | 1|   V       | 1| ah sup ->h -> |     1| ah sup
% ----        *2   |  |----       ----          *2   --------
% ----             ----|  |       ----              --------
% | 2| ahdv ->gr->     | 2|       | 2| ah inf ->h -> |     2| ah inf
% ----        *2       ----       ----          *2   --------
% ----             ---- <-+--     ----              --------
% | 3| dhav ->h -> | 3|   V       | 3| dh sup ->gr-> |     3| dh sup
% ----        *2   |  |----       ----          *2   --------
% ----             ----|  |       ----              --------
% | 4| dhdv ->gr->     | 4|       | 4| dh inf ->gr-> |     4| dh inf
% ----        *2       ----       ----          *2   --------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Cada uno deshace el último filtrado por columnas, vertical
%       El primero manda al segundo la mitad inferior de lo deshecho (viceversa)
%       Cada uno suma la mitad que puede
%               1 ahav->aha (mitad inf)\/(suma sup) ah superior
%               2 ahdv->ahd (mitad sup)/\(suma inf) ah inferior
%               3 dhav->dha (mitad inf)\/(suma sup) dh superior
%               4 dhdv->dhd (mitad sup)/\(suma inf) dh inferior
%       Cada uno deshace el primer filtrado horizontal en la mitad que puede
%         y se lo manda al host para sumar (no merece la pena local/re-enviar)
%               1 ah->a superior---/host suma sup\
%               2 ah->a inferior-\/               \host tiene img
%               3 dh->d superior-/\               /
%               4 dh->d inferior---\host suma inf/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       Se devuelve resultado en lhs idwt2Par, no mediante MPI
%        hay dependencia secuencial entre sucesivas re-composiciones
%       No se ve claro cómo se podría seguir sin tener siguiente recomposición

global NTIMES                   % hizo falta para pasar a [i]dwt2ParHost dlbl
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<5, NUMSLV    =3; else, NUMSLV    =min(max(round(nslaves   ),3),4);end
if nargin<4, withlena  =0; else, withlena  =min(max(round(withlena  ),0),1);end
if nargin<3, withdetail=0; else, withdetail=min(max(round(withdetail),0),1);end
if nargin<2, withgraph =1; else, withgraph =min(max(round(withgraph ),0),1);end
if nargin<1, NTIMES    =1; else, NTIMES    =    max(round(ntimes    ),0   );end

global DIR, DIR=[ getenv('MPITB_ROOT') '/Users99/' mfilename ];
if ~NTIMES, [T R S]=shwgraph(withgraph,withdetail,NUMSLV); return, end

tic
```

Listado E.22: Par.m: Programa maestro MPITB para transformada *wavelet* paralela de una imagen B/W.

05/16/00 13:53:24 mpitbCode/Par.m **2**

```
%%%%%%%%%%%%%%%%%%% Número de esclavos==3/4
% Test daemon MPI % No sería justo contar tiempo MPI_Init, obliga a salir
%%%%%%%%%%%%%%%%%%%
[info flag]=MPI_Initialized;
if info | ~flag, error('MPI not initialized, run MPI_Init'), end
[info attr flag]=MPI_Attr_get(MPI_COMM_WORLD,MPI_UNIVERSE_SIZE);
if info | ~flag, error('MPI_UNIVERSE_SIZE keyval undefined?'), end
if attr<NUMSLV+1, error('not enough computers on LAM'), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% lamboot/$LAMBHOST debería haber arrancado suficientes hosts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix labels
tm=zeros(2+NTIMES*(1+6*(1+2*withdetail)+8*withlena),6);
                        msg={}; ix=0; labels={};        record('');
%%% 2:          '' 'Spwn'
%%% 1*:            'Load'
%%% 6*(1+2*detl)*: 'D1'-'D3' 'R3'-'R1' cada uno con 'Snd' 'dlbl' 'Rcv' (*3)
%%% 8*lena*: 'math' 'cast' 3*('fig' 'shw')
%%% cada bucle va de 'Load' a 'Load'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arranque MATLAB's esclavos % Número de esclavos==3/4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DISP=getenv('DISPLAY');
% [info children errs]=MPI_Comm_spawn('xterm',...
%               {'-display' DISP '-e' 'matlab'},NUMSLV,'NULL',0,'SELF');
  [info children errs]=MPI_Comm_spawn('matlab',...
                                 {'-nosplash'},NUMSLV,'NULL',0,'SELF');
if info | any(errs),    error('cannot start MATLABs'), end
[info numt] = MPI_Comm_remote_size(children);
if info | numt<NUMSLV,  error('cannot start MATLABs'), end
global NEWORLD                        % Cansado pasarlo a [i]dwt2ParHost
[info NEWORLD]=MPI_Intercomm_merge(children,0);
if info,                error('cannot start MATLABs'), end
MPI_Errhandler_set(NEWORLD,MPI_ERRORS_RETURN);

global TAG NUMCMDS
TAG=7; NUMCMDS=6*NTIMES; QUIT=1;        % Enviaremos 3 des-, 3 re-composiciones
for i=1:numt
  info = MPI_Send(NUMCMDS,i,TAG,NEWORLD);
  info = MPI_Send(QUIT,   i,TAG,NEWORLD);
end                                   % Ignorar respuesta confirmación
for i=1:numt, [info stat]=MPI_Recv(NUMCMDS,i,TAG,NEWORLD); end
rnks=1:NUMSLV;
if NUMSLV==3, rnks=[0 rnks]; end        % Añadir host si va a trabajar
                                                    record('Spwn'),label
%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
for indiceNTIMES=1:NTIMES

%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_funet','tif'); load h, load g
gd=[  g(2:length(g));0];                                % Detail decomp/recomp
gr=[0;g(1:length(g)-1)];                                record('Load')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Dependencia secuencial
% Descomposición 3 niveles % Para etiquetar detalle host hay que pasar string
%%%%%%%%%%%%%%%%%%%%%%%%%%%% Si no detalle, se etiqueta desde aquí
wd=withdetail; dl='';                            if  wd,    dl='D1';        end
[a1 dh1 dv1 dd1]=dwt2ParHost(lena,h,gd,rnks,dl); if ~wd,record('D1'), label,end
                                                 if  wd,    dl='D2';        end
[a2 dh2 dv2 dd2]=dwt2ParHost(  a1,h,gd,rnks,dl); if ~wd,record('D2'),
                                                         if NTIMES<3, label,end
                                                                            end
                                                 if  wd,    dl='D3';        end
[a3 dh3 dv3 dd3]=dwt2ParHost(  a2,h,gd,rnks,dl); if ~wd,record('D3'),
                                                         if NTIMES<2, label,end
                                                                            end
%%%%%%%%%%%%%%%%%
% Recomposición % Dependencia secuencial
%%%%%%%%%%%%%%%%%
                                                if  wd,     dl='R3';        end
r2=idwt2ParHost(a3,dh3,dv3,dd3, h,gr, rnks,dl); if ~wd, record('R3'),
                                                         if NTIMES<2, label,end
                                                                            end
                                                if  wd,     dl='R2';        end
r1=idwt2ParHost(r2,dh2,dv2,dd2, h,gr, rnks,dl); if ~wd, record('R2')
                                                         if NTIMES<3, label,end
                                                                            end
                                                if  wd,     dl='R1';        end
r0=idwt2ParHost(r1,dh1,dv1,dd1, h,gr, rnks,dl); if ~wd, record('R1'), label,end

%%%%%%%%%%%%
% Gráficas %
%%%%%%%%%%%%
if withlena
 transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);record('math')
  reconstr=uint8(r0);                                   record('cast')

  figure,                                               record('fig')
  imshow(lena), title('Original'),                      record('shw')
  figure,                                               record('fig')
  imshow(transform), title('Transform'),                record('shw')
  figure,                                               record('fig')
  imshow(reconstr), title('Reconstruction'),            record('shw')
end

%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
end

%%%%%%%%%%%%%%
% Acabar MPI %
%%%%%%%%%%%%%%
% info=MPI_Finalize;                            % No permitiría otro MPI_Init

        toc                                     % La gráfica nunca cuenta
        recgraph;                               % Salvar datos gráfica
[T R S]=shwgraph(withgraph,withdetail,NUMSLV);  % Ver gráfica si no se impide

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [T,R,S]=shwgraph(withgraph,withdetail,NUMSLV)
global DIR

         HN0=strtok(hostname,'.');
if withdetail
        [HN1 HN2 HN3 HN4]=parsebhost;           % (confiar round-robin spawn)
[T R S MSG tm1 lbl1]=stats(DIR,['idwt2Par_' HN1]); disp(MSG), S
```

Listado E.22: Continuación.

05/16/00
13:53:24

mpitbCode/Par.m

3

```
[T R S MSG tm2 lbl2]=stats(DIR,['idwt2Par_' HN2]); disp(MSG), S
[T R S MSG tm3 lbl3]=stats(DIR,['idwt2Par_' HN3]); disp(MSG), S, if NUMSLV==4
[T R S MSG tm4 lbl4]=stats(DIR,['idwt2Par_' HN4]); disp(MSG), S, end
end
[T R S MSG tm0 lbl0 msg]=stats(DIR,[mfilename '_' HN0]); disp(MSG), S

if withgraph
  graph(DIR,mfilename,HN0);                      % Ver gráfica si no se impide
 if withdetail
        args={DIR [mfilename '_corr'] MSG 'D1S' msg};
        args={args{:} HN0 tm0 lbl0 HN1 tm1 lbl1 HN2 tm2 lbl2 HN3 tm3 lbl3};
    graph(DIR,'idwt2Par',HN1);
    graph(DIR,'idwt2Par',HN2);
    graph(DIR,'idwt2Par',HN3); if NUMSLV==4
    graph(DIR,'idwt2Par',HN4);
    args = {args{:} HN4 tm4 lbl4}; end
        timegraph(args{:});
 end
end

clear global
```

Listado E.22: Continuación.

05/15/00 16:34:54 — mpitbCode/dwt2ParHost.m — 1

```
function [ahav, ahdv, dhav, dhdv]=dwt2ParHost(img, h,g, rnks, dlbl)
% DWT2PARHOST           Parallel DWT, Host Sequence
%
%       [ahav ahdv dhav dhdv] = dwt2ParHost ( img, h,g, rnks, dlbl )
%
%  img (matriz) Señal 2-D a analizar. Se devuelven las 4 componentes
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  g   (vector) Filtro 1-D detalle      (paso alta) para descomposición. lh=lg
%  rnks(vector) 4 ranks MPI que colaboran, esta función la ejecuta el host
%  dlbl (str)   Detail label:si no vacía, indica que se desea desglosar tiempos
%
%  ahav(matriz) Señal 2-D aproximación  (coeficientes). Area 1/4 img
%  ahdv(matriz) Señal 2-D aprox. horiz, detalle vert. (bordes horiz)
%  dhav(matriz) Señal 2-D detalle horiz, aprox. vert. (bordes vert)
%  dhdv(matriz) Señal 2-D detalle horiz, detalle vert.(bordes diagonal)
%

rf=~isempty(dlbl);                                      % Record flag
if rf, global tm msg ix DIR labels, else, DIR=''; end
global TAG NEWORLD NTIMES
%%%%%%%%%%%%
% Longitudes
%%%%%%%%%%%%
lih =size(img,2); liv =size(img,1);     % Length image horz/vert
lih2=ceil(lih/2); liv2=ceil(liv/2);     % Al decimar puede salir 1 de más
if length(rnks)~=4, error('requiere 4 tareas MPI esclavas'), end

%%%%%%%%%%%%%%%%%%%%%%%%%
% Enviar mitades imagen % Tareas trabajan en 2 grupos de 2 ranks
%%%%%%%%%%%%%%%%%%%%%%%%%
T1=int2str(rnks(1));    T2=int2str(rnks(2));
T3=int2str(rnks(3));    T4=int2str(rnks(4));
DL=['''' DIR ''','''' dlbl ''''];

img1=double(img(     1:liv2,:)); siz1=size(img1);       % Mitad sup. a primeros
img2=double(img(liv2+1:liv ,:)); siz2=size(img2);       % Mitad inf. a segundos

if rnks(1)                                     % Puede que host trabaje también
cmd=['siz1=[0 0];       MPI_Recv(siz1, 0,TAG,NEWORLD);',...
     'img1=zeros(siz1); MPI_Recv(img1, 0,TAG,NEWORLD);',...
     'dwt2Par(img1, h,h,' T2 ', 0, ' DL ');'];
[info psiz]=MPE_Pack_size(h,  cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,  cmd,buffer, 0 ,NEWORLD);  % sizeof(mxDOUBLE)==8 B
                 MPI_Send(buffer,rnks(1),TAG,NEWORLD);
                 MPI_Send(siz1  ,rnks(1),TAG,NEWORLD);
                 MPI_Send(img1  ,rnks(1),TAG,NEWORLD);
end
cmd=['siz2=[0 0];       MPI_Recv(siz2, 0,TAG,NEWORLD);',...
     'img2=zeros(siz2); MPI_Recv(img2, 0,TAG,NEWORLD);',...
     'dwt2Par(img2, h,g,' T1 ', 1, ' DL ');'];
[info psiz]=MPE_Pack_size(h,g,cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,g,cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(2),TAG,NEWORLD);
                 MPI_Send(siz2  ,rnks(2),TAG,NEWORLD);
                 MPI_Send(img2  ,rnks(2),TAG,NEWORLD);

cmd=['siz1=[0 0];       MPI_Recv(siz1, 0,TAG,NEWORLD);',...
     'img1=zeros(siz1); MPI_Recv(img1, 0,TAG,NEWORLD);',...
     'dwt2Par(img1, g,h,' T4 ', 0, ' DL ');'];
[info psiz]=MPE_Pack_size(g,h,cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(g,h,cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(3),TAG,NEWORLD);
                 MPI_Send(siz1  ,rnks(3),TAG,NEWORLD);
                 MPI_Send(img1  ,rnks(3),TAG,NEWORLD);
                 MPI_Send(img1  ,rnks(3),TAG,NEWORLD);

cmd=['siz2=[0 0];       MPI_Recv(siz2, 0,TAG,NEWORLD);',...
     'img2=zeros(siz2); MPI_Recv(img2, 0,TAG,NEWORLD);',...
     'dwt2Par(img2, g,g,' T3 ', 1, ' DL ');'];
[info psiz]=MPE_Pack_size(  g,cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(  g,cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(4),TAG,NEWORLD);
                 MPI_Send(siz2  ,rnks(4),TAG,NEWORLD);
                 MPI_Send(img2  ,rnks(4),TAG,NEWORLD);
                                                      if rf,record([dlbl 'S']),end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Tiempo muerto o trabajamos ? %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   ----            ----            ----
%                            --->h->| 1|  ah sup\  /| 1|      ->h-> | 1| ahav
%                           /       ----         \/ |  |----        ----
%                 --------  |       ----         /\ ----|  |        ----
% --------        | img1 | -|  />h->| 2|  ah inf/  \    | 2|  ->g-> | 2| ahdv
% | img  |  --\   --------   \/     ----                ----        ----
% |      |  --/   --------   /\     ----            ----            ----
% --------        | img2 | -|  \>g->| 3|  dh sup\  /| 3|      ->h-> | 3| dhav
%                 --------  |       ----         \/ |  |----        ----
%                           \       ----         /\ ----|  |        ----
%                            --->g->| 4|  dh inf/  \    | 4|  ->g-> | 4| dhdv
%                                   ----                ----        ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~rnks(1)                                % Puede que host trabaje también
ahav=dwt2Par(img1, h,h,rnks(2), 0, rf,dlbl);
else                                       % Si tiempo muerto, recibir ahav

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recibir Transformada, Pre-allocation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ahav=zeros(liv2,lih2); MPI_Recv(ahav,rnks(1),TAG,NEWORLD); end
ahdv=zeros(liv2,lih2); MPI_Recv(ahdv,rnks(2),TAG,NEWORLD);
dhav=zeros(liv2,lih2); MPI_Recv(dhav,rnks(3),TAG,NEWORLD);
dhdv=zeros(liv2,lih2); MPI_Recv(dhdv,rnks(4),TAG,NEWORLD);

                                                      if rf,record([dlbl 'R'])
                dl=dlbl(end); if ~rnks(1),dl ='0';end   % Host ya anota D1x...
                              if dl=='1'|(dl=='2'&NTIMES<3)|...
                                         (dl=='3'&NTIMES<2), label, end, end
```

Listado E.23: dwt2ParHost.m: Rutina MPITB de distribución para análisis *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.13.

05/14/00 21:11:31 mpitbCode/dwt2Par.m 1

```
function res=dwt2Par(img, f1,f2, rnk,slv, wd,dlbl)
% DWT2PAR               Parallel DWT, Slave sequence
%
%       res = dwt2Par ( img, f1,f2, rnk,slv, wd,dlbl )
%
%  img (matriz) Señal 2-D mitad superior/inferior
%  f1  (vector) Filtro 1-D primera etapa (horizontal)
%  f2  (vector) Filtro 1-D segunda etapa (vertical)
%
%  rnk (int)    Rango de la tarea MPI que tiene la otra mitad
%  slv (int)    0/1, indica si somos la 2ª mitad
%
%  wd           0/1, indica si se ha de etiquetar (withdetail para host)
%  wd (string)  Directorio salvar datos/gráficas (work dir para esclavo)
%               '' si no se desea escribir en disco nada
%  dlbl (str)   Prefijo de etiquetado
%
%       El resultado (res) se devuelve mediante MPI a la tarea madre
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                           ----         ----         ----
%                  --->h->| 1| ah sup\  /| 1|     ->h-> | 1| ahav
%                 /         ----        \/ |  |----       ----
%         --------  |         ----        /\ ----|  |       ----
% --------  | img1 | -|  />h->| 2| ah inf/  \     | 2| ->g-> | 2| ahdv
% | img  |  --\  --------   \/     ----              ----       ----
% |      |  --/  --------   /\     ----         ----       ----
% --------  | img2 | -|  \>g->| 3| dh sup\  /| 3|     ->h-> | 3| dhav
%         --------  |         ----        \/ |  |----       ----
%                 \         ----        /\ ----|  |       ----
%                  --->g->| 4| dh inf/  \     | 4| ->g-> | 4| dhdv
%                           ----              ----       ----
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global TAG NEWORLD                      % puede pillarlo del host o del esclavo
global NUMCMDS indiceBucleMPI           % puede pillarlo del host o del esclavo
%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix DIR labels             % a esclavos no llega global host
if ~ischar(wd), rf=wd;                  % HOST: record flag <=> wd(with detail)
else,  DIR=wd;  rf=~isempty(DIR);       % SLAV: record <=> ''~= wd (work dir)
     if rf
        if isempty(ix)                  % Si 1ª vez del esclavo, reservar anot.
        tm=zeros(4*NUMCMDS,6); msg={}; ix=0; labels={}; end,   record('')
     end
end
%%% 4: 'Row' 'xchg' 'Col' 'Snd'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Longitudes, Constantes, etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lih =size(img,2); liv2=size(img,1);     % Length image horz/vert
lih2=ceil(lih/2); liv =2*liv2;          % Se le pasó una mitad de la imagen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtrado por filas, 1ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tmp=zeros(liv,lih2);                    % Pre-allocation
for row=1:liv2                          % Sólamente una mitad img1/img2
  r  = img(row,:);                      % Obtener img->ah/dh, según f1==h/g
  rf1= dwtflt(r',f1, 'spec');           %    sólo mitad sup/inf de ah/dh
  tmp(slv*liv2+row,:) = rf1';           % primera ó segunda según slv==0/1
end
                                        if rf, record([dlbl 'r']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Envío/Recepción  intermedio
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
img2=tmp(slv*liv2+1:(slv+1)*liv2, : );                      % Dicha mitad
% MPI_Send            (img2,        rnk,TAG,NEWORLD);   % mandarla al otro
  MPI_Sendrecv_replace(img2,rnk,TAG,rnk,TAG,NEWORLD);
% MPI_Recv            (img2,        rnk,TAG,NEWORLD);   % mitad del compañero
tmp((1-slv)*liv2+1:(2-slv)*liv2, : )=img2;              % pegarla antes/después
                                        if rf, record([dlbl 'x'])
  dl=dlbl(end); nt=NUMCMDS/6;   if dl=='1'|(dl=='2'&nt<3)|...
                                           (dl=='3'&nt<2), label, end, end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Filtrado por columnas, 2ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
res=zeros(liv2,lih2);                   % Pre-allocation
for col=1:lih2                          % Toda la columna
  c  = tmp(:,col);                      % Obtener tmp->xhav/xhdv según f2==h/g
  cf2= dwtflt(c ,f2, 'spec');           %    toda la columna de tmp==xh (ah/dh)
  res(:,col)= cf2;
end
                                        if rf, record([dlbl 'c']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Envío Transformada al padre %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[info rank]=MPI_Comm_rank(NEWORLD);     % Por si acaso el host también trabaja
if rank                                 % Observar recgraph destroza tm, msg...
  MPI_Send(res,0,TAG,NEWORLD);          if rf, record([dlbl 's'])
                        if indiceBucleMPI==NUMCMDS, recgraph, end, end
end
```

Listado E.24: dwt2Par.m: Esclavo MPITB para análisis *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.14.

05/15/00
16:46:27

mpitbCode/idwt2ParHost.m

1

```
function r=idwt2ParHost(ahav,ahdv,dhav,dhdv, h,g, rnks, dlbl)
% IDWT2PARHOST          Parallel IDWT, Host Sequence
%
%       r = idwt2ParHost ( ahav,ahdv,dhav,dhdv, h,g, rnks, dlbl )
%
%  ahav(matriz) Señal 2-D aproximación  (coeficientes)
%  ahdv(matriz) Señal 2-D aprox. horiz, detalle vert. (bordes horiz)
%  dhav(matriz) Señal 2-D detalle horiz, aprox. vert. (bordes vert)
%  dhdv(matriz) Señal 2-D detalle horiz, detalle vert.(bordes diagonal)
%
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  g   (vector) Filtro 1-D detalle      (paso alta) para re-composición. lh=lg
%  rnks(vector) 4 ranks MPI que colaboran, esta función la ejecuta el host
%  dlbl (str)   Detail label:si no vacía, indica que se desea desglosar tiempos
%
%  img (matriz) Señal 2-D reconstruida. Area = 4 veces ahav
%

rf=~isempty(dlbl);                                      % Record flag
if rf, global tm msg ix DIR labels, else, DIR=''; end
global TAG NEWORLD NTIMES
%%%%%%%%%%%%
% Longitudes
%%%%%%%%%%%%
l2=size(ahav);
if any(l2~=size(ahdv))|...
   any(l2~=size(dhav))|...
   any(l2~=size(dhdv)), error('imagenes de distintos tamaños'), end
lih2=l2(2);  liv2=l2(1);
lih =lih2*2; liv =liv2*2;
if length(rnks)~=4, error('requiere 4 tareas MPI esclavas'), end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Repartir descomposiciones % Tareas trabajan en 2 grupos de 2 ranks
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
DL=[''''' DIR ''',''' dlbl ''''];

if rnks(1)                              % Puede que host trabaje también
siz=size(ahav);
cmd=['siz =[0 0];       MPI_Recv(siz , 0,TAG,NEWORLD);',...
     'ahav=zeros(siz);  MPI_Recv(ahav, 0,TAG,NEWORLD);',...
     'idwt2Par(ahav, h,h,' int2str(rnks(2)) ', 0, ' DL ');'];
[info psiz]=MPE_Pack_size(h,  cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,  cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(1),TAG,NEWORLD);
                 MPI_Send(siz,   rnks(1),TAG,NEWORLD);
                 MPI_Send(ahav,  rnks(1),TAG,NEWORLD);
end
siz=size(ahdv);
cmd=['siz =[0 0];       MPI_Recv(siz , 0,TAG,NEWORLD);',...
     'ahdv=zeros(siz);  MPI_Recv(ahdv, 0,TAG,NEWORLD);',...
     'idwt2Par(ahdv, h,g,' int2str(rnks(1)) ', 1, ' DL ');'];
[info psiz]=MPE_Pack_size(h,g,cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,g,cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(2),TAG,NEWORLD);
                 MPI_Send(siz,   rnks(2),TAG,NEWORLD);
                 MPI_Send(ahdv,  rnks(2),TAG,NEWORLD);
siz=size(dhav);
cmd=['siz =[0 0];       MPI_Recv(siz , 0,TAG,NEWORLD);',...
     'dhav=zeros(siz);  MPI_Recv(dhav, 0,TAG,NEWORLD);',...
     'idwt2Par(dhav, g,h,' int2str(rnks(4)) ', 0, ' DL ');'];
[info psiz]=MPE_Pack_size(g,h,cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(g,h,cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(3),TAG,NEWORLD);
                 MPI_Send(siz,   rnks(3),TAG,NEWORLD);
                 MPI_Send(dhav,  rnks(3),TAG,NEWORLD);
siz=size(dhdv);
cmd=['siz =[0 0];       MPI_Recv(siz , 0,TAG,NEWORLD);',...
     'dhdv=zeros(siz);  MPI_Recv(dhdv, 0,TAG,NEWORLD);',...
     'idwt2Par(dhdv, g,g,' int2str(rnks(3)) ', 1, ' DL ');'];
[info psiz]=MPE_Pack_size(  g,cmd,NEWORLD); buffer=zeros(1,ceil(psiz/8));
                 MPE_Pack(  g,cmd,buffer, 0 ,NEWORLD);
                 MPI_Send(buffer,rnks(4),TAG,NEWORLD);
                 MPI_Send(siz,   rnks(4),TAG,NEWORLD);
                 MPI_Send(dhdv,  rnks(4),TAG,NEWORLD);

                                                  if rf,record([dlbl 'S']),end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Tiempo muerto o trabajamos ? %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----             ---- <-+--    ----                 --------
% | 1| ahav ->h -> | 1|   V      | 1| ah sup ->h -> |     1| a sup\
% ----        *2   |  |----      ----          *2     --------      \      r
% ----             ----|  |      ----                 --------       \  --------
% | 2| ahdv ->g ->     | 2|      | 2| ah inf ->h -> |     2| a inf\  >-|      |
% ----        *2       ----      ----          *2     --------      \/  |______|
% ----             ---- <-+--    ----                 --------      /\  |      |
% | 3| dhav ->h -> | 3|   V      | 3| dh sup ->g -> |     3| d sup/  >-|      |
% ----        *2   |  |----      ----          *2     --------       /  --------
% ----             ----|  |      ----                 --------      /
% | 4| dhdv ->g ->     | 4|      | 4| dh inf ->g -> |     4| d inf/
% ----        *2       ----      ----          *2     --------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recibir Reconstrucción, Pre-allocation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r  =zeros(liv ,lih);
tmp=zeros(liv2,lih);
res=zeros(liv2,lih);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~rnks(1)                             % Puede que host trabaje también
tmp=idwt2Par(ahav, h,h, rnks(2), 0, rf,dlbl);
else                                    % Si tiempo muerto, recibir ah sup

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MPI_Recv(tmp,rnks(1),TAG,NEWORLD); end                       % ah    sup
MPI_Recv(res,rnks(3),TAG,NEWORLD); r(     1:liv2,:)=tmp+res;    % ah+dh sup

MPI_Recv(tmp,rnks(2),TAG,NEWORLD);                           % ah    inf
MPI_Recv(res,rnks(4),TAG,NEWORLD); r(liv2+1:liv ,:)=tmp+res;    % ah+dh inf

                                                  if rf,record([dlbl 'R'])
                dl=dlbl(end); if ~rnks(1),dl ='0';end   % Host ya anota D1x...
                              if dl=='1'|(dl=='2'&NTIMES<3)|...
                                         (dl=='3'&NTIMES<2), label, end, end
```

Listado E.25: idwt2ParHost.m: Rutina MPITB de distribución para reconstrucción *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.15.

05/15/00 16:42:45 mpitbCode/idwt2Par.m **1**

```
function res=idwt2Par(img, f1,f2, rnk,flg, wd,dlbl)
% IDWT2PAR              Parallel IDWT, Slave sequence
%
%       res = idwt2Par ( img, f1,f2, rnk,flg, wd,dlbl )
%
%  img (matriz) Señal 2-D descomposición wavelet
%  f1  (vector) Filtro 1-D primera etapa (horizontal)
%  f2  (vector) Filtro 1-D segunda etapa (vertical)
%
%  rnk (int)    Rango de la tarea MPI con la cual se coopera
%  flg (int)    0/1, indica si somos la 2ª descomposición del grupo
%
%  wd           0/1, indica si se ha de etiquetar (withdetail para host)
%  wd (string)  Directorio salvar datos/gráficas (work dir para esclavo)
%               '' si no se desea escribir en disco nada
%  dlbl (str)   Prefijo de etiquetado
%
%       El resultado (res) se devuelve mediante MPI a la tarea madre
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ----           ---- <-+--    ----             --------
% | 1| ahav ->h -> | 1|   V     | 1| ah sup ->h -> |     1| a sup\
% ----        *2   |  |----    ----         *2    --------      \       r
% ----           ----|  |      ----             --------         \  --------
% | 2| ahdv ->g ->     | 2|     | 2| ah inf ->h -> |     2| a inf\  >-|      |
% ----        *2      ----     ----         *2    --------       \/  |______|
% ----           ---- <-+--    ----             --------         /\  |      |
% | 3| dhav ->h -> | 3|   V     | 3| dh sup ->g -> |     3| d sup/  >-|      |
% ----        *2   |  |----    ----         *2    --------        /  --------
% ----           ----|  |      ----             --------        /
% | 4| dhdv ->g ->     | 4|     | 4| dh inf ->g -> |     4| d inf/
% ----        *2      ----     ----         *2    --------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global TAG NEWORLD                      % puede pillarlo del host o del esclavo
global NUMCMDS indiceBucleMPI           % puede pillarlo del host o del esclavo
%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix DIR labels             % a esclavos no llega global host
if ~ischar(wd), rf=wd;                  % HOST: record flag <=> wd(with detail)
else,  DIR=wd;  rf=~isempty(DIR);       % SLAV: record <=> ''~= wd (work dir)
     if rf
        if isempty(ix)                  % Si 1ª vez del esclavo, reservar anot.
        tm=zeros(4*NUMCMDS,6); msg={}; ix=0; labels={}; end,   record('')
     end
end
%%% 4: 'c' 'X' 'r' 's'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Longitudes, Constantes, etc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
lih2=size(img,2); liv2=size(img,1);     % Length image horz/vert
lih =2*lih2;      liv =2*liv2;          % Se le pasó un cuarto de imagen

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Deshacer filtrado por columnas, 2ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tmp=zeros(liv,lih2);                    % Pre-allocation
for col=1:lih2                          % Hace su mitad completa
  c  = img(:,col);                      % P.ej.: si éste recibió ahav (rnks(1))
  cf2= idwtflt(c ,f2, 'spec');          %  el otro (rnks(2)) tiene ahdv
  tmp(:,col) = cf2;                     % Pasar siempre columnas
end
                                        if rf, record([dlbl 'c']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Sumar una mitad con mitad del otro
% Envío/Recepción  intermedio % Ej.: rnks(1) ahav->aha- mitades superiores ah
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%     rnks(2) ahdv->ahd/         inferiores ah
 una=tmp(   flg *liv2+1:(flg+1)*liv2, : );               % mitad    interesante
otra=tmp((1-flg)*liv2+1:(2-flg)*liv2, : );               % mitad no interesante
% MPI_Send            (otra,        rnk,TAG,NEWORLD);   % mandarla al otro
  MPI_Sendrecv_replace(otra,rnk,TAG,rnk,TAG,NEWORLD);
% MPI_Recv            (otra,        rnk,TAG,NEWORLD);   % Recibir sobre otra
tmp = una+otra;                                          % y sumarla
                                        if rf, record([dlbl 'x'])
  dl=dlbl(end); nt=NUMCMDS/6;   if dl=='1'|(dl=='2'&nt<3)|...
                                           (dl=='3'&nt<2), label, end, end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Deshacer filtrado por filas, 1ª etapa
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
res=zeros(liv2,lih);                    % Pre-allocation
for row=1:liv2                          % La mitad sup/inf? lo que haya en tmp
  r  = tmp(row,:);                      % Obtener a/d según f2==h/g
  rf1= idwtflt(r',f1, 'spec');          %    toda la fila de tmp==(ah/dh)(s/inf)
  res(row,:)= rf1';                     % Pasar siempre columnas
end
                                        if rf, record([dlbl 'r']), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Envío Transformada al padre %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% res=uint8(res);                       % No se podría sumar luego en host
[info rank]=MPI_Comm_rank(NEWORLD);
if rank                                 % Por si acaso el host trabaja
  MPI_Send(res,0,TAG,NEWORLD);          if rf, record([dlbl 's'])
                        if indiceBucleMPI==NUMCMDS, recgraph, end, end
end
```

Listado E.26: idwt2Par.m: Esclavo MPITB para reconstrucción *wavelet* 2D B/W. Se mostró un fragmento en el Listado 5.16.

06/03/00 13:46:57 **pvmtbCode/ParRGB.m** 1

```
function [T,R,S]=ParRGB(ntimes,withgraph,withdetail,...
                        withlena,withmath,nslaves,throttle)
% ParRGB: Ejemplo paralelo color análisis wavelet multiresolución
%
%       [T R S] = ParRGB ( ntimes, withgraph, withdetail,...
%                          withlena, withmath, nslaves, throttle )
%
%  ntimes      [1],    número de veces a repetir el trabajo (medir tiempo)
%               0 ,    si se desea consultar datos de ejecución anterior
%  withgraph   0/[1],  indica si se desean gráficas de tiempo
%                      el tiempo de gráficas no se incluye, naturalmente
%  withdetail  [0]/1,  indica si se desea desglosar tiempo en paralelo
%                      si se activa, esclavos salvan datos a disco
%  withlena    [0]/1,  indica si se desean ver las imágenes
%                      el tiempo de dibujo se incluye en la gráfica
%  withmath    [0]/1,  indica si paralelizar norm. transformada y typecast
%  nslaves     [2]/3,  número de procesos MATLAB esclavos a arrancar
%  throttle    [0]/1,  el host trabaja y espera D1-D3 esclavos a la mitad
%
%  T    "Times", registro con campos:
%  totl tiempo total gastado (no incluye tiempo para gráfica tiempos)
%  spwn tiempo gastado hasta Spawn MATLAB hijos
%
% R     "Round", tiempo de cada vuelta (de Load a Load)
% S     "Stats", cell-array con columnas {'label' mean stdev ntimes}

%       tomado de Mallat "A Theory for Multiresolution Signal Decomposition:
%                         The Wavelet Representation"
%       IEEE Trans. on Patt.Analysis and Mach.Intell. Vol.11 N.7 July 1989
%
%       3 ordenadores, cada uno descompone un bit plane R-G-B
%       Pueden ser 3 esclavos, ó el host y 2 esclavos

% Necesita:
% lena_chuck.tif        Imagen de Lena en RGB, obtenida de chuck
% h.mat         Filtro Wavelet Aproximación (unidimensional)
% g.mat         Filtro Wavelet Detalle      (unidimensional)
%
% dwt2D3L       Descomposición y Recomposición 3 niveles para cada canal
%   dwt2        Descomposición Wavelet Bi-dimensional
%     dwt       Descomposición Wavelet Unidimensional
%   idwt2       Re-composición Wavelet Bi-dimensional
%     idwt      Re-composición Wavelet Unidimensional
%
% math          Operaciones de normalización para visualizar transformada
% stats         Estadísticas sobre las anotaciones
% record        Anotar tiempos
% label         Anotar etiquetas
% recgraph      Salvar datos gráfica tiempos
% graph         Generación de gráficas a partir de datos
% timegraph     Gráfica de tiempos correlacionados

%       %%%%%% DES/RE-/COMPOSICION %%%%%%%%%%%%%%% dwt2D3L.m %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   Bitplane   Transform       Reconstr
%     -------- ------------> --------    -------- PVM to --------
%   --------B|                  | Slv3 | -\ ||_|___|   -\   |       | -\ PVM to
% --------G| |                  | HOST?| -/ |  |   |   -/   |       | -/ parent
% |     R| |--                  --------    -------- parent --------
% |      |--   ------------>   Slv2 ...
% --------     ------------>   Slv1 ...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global NTIMES                   % hizo falta para pasar control label a dwt2D3L
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<7, throttle  =0; else, throttle  =min(max(round(throttle  ),0),1);end
if nargin<6, NUMSLV    =2; else, NUMSLV    =min(max(round(nslaves   ),2),3);end
if nargin<5, withmath  =0; else, withmath  =min(max(round(withmath  ),0),1);end
if nargin<4, withlena  =0; else, withlena  =min(max(round(withlena  ),0),1);end
if nargin<3, withdetail=0; else, withdetail=min(max(round(withdetail),0),1);end
if nargin<2, withgraph =1; else, withgraph =min(max(round(withgraph ),0),1);end
if nargin<1, NTIMES    =1; else, NTIMES    =    max(round(ntimes    ),0   );end
if withmath & ~withlena
  warning('withmath==1 and withlena==0'), end
if NUMSLV==3 & throttle,  throttle=0;
  warning('Unsensible throttle  ==1 while NUMSLV   ==3, setting to 0'), end

global DIR, DIR=[ getenv('PVMTB_ROOT') '/Users99/' mfilename ];
if ~NTIMES, [T R S]=shwgraph(withgraph,withdetail,NUMSLV); return, end

if throttle, global trRED trGREEN tids, end     % para throttle en dwt2D3L

tic
%%%%%%%%%%%%%%%%%%%%%%%%% Número de esclavos==2/3
% Arranque daemon PVM % No sería justo contar tiempo, MPI exige lamboot previo
%%%%%%%%%%%%%%%%%%%%%%%
if ~pvme_is, error('PVM not initialized, run pvm console or xpvm'), end
if pvm_config~=NUMSLV+1, error('wrong number of computers in PVM'), end
if pvm_setopt(1,3)<0,error('cannot setopt RouteDirect'), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% [x]pvm debería haber arrancado número exacto de hosts
%%%%%%% para que los ficheros se creen con nombres reproducibles
%%%%%%% sin que interfiera el algoritmo round-robin memorioso de PVM
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix labels
tm=zeros( 2 + NTIMES*( 5+ 9*withdetail + withlena*(8+(1-withmath)) ), 6);
                        msg={}; ix=0; labels={};        record('')
%%% 2: '' 'Spwn'
%%% 4*: 'Load' 'Snd' ('Wav') 'Recv' ('math') 'Recv' (lena)
%%%(1+9*detl)*:  'Wav'== 'D1'-'D3' 'math/grp' 'Snd' 'R3'-'R1' 'cast/asgn' 'Snd'
%%%(lena*~math)*:'math'
%%% 8*lena*:     'cat' 'ca/st' 3*('fig' 'shw')
%%% cada bucle va de 'Load' a 'Load'
%%% tomamos el caso peor en que NUMSLV==2, si NUMSLV==3 no hay 'Wav'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arranque MATLAB's esclavos % Número de esclavos==2/3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DISP=getenv('DISPLAY');
% [numt tids]=pvm_spawn('xterm',{'-display' DISP '-e' 'matlab'},33,'.',NUMSLV);
  [numt tids]=pvm_spawn('matlab',{'-nosplash'},33,'.',NUMSLV);
if numt~=NUMSLV, error('cannot start MATLABs'), end

global RAW PLACE, RAW=1; PLACE=2;
global TAG
TAG=7; NUMCMDS=1*NTIMES; QUIT=1;        % Enviaremos 1 des/re-composición
for i=1:numt                            % QUIT=0 para no perder gráficas
  pvm_initsend(RAW); pvme_pack(NUMCMDS,QUIT); pvm_send(tids(i),TAG);
end                                     % Ignorar respuesta confirmación
```

Listado E.27: ParRGB.m: Programa maestro PVMTB para transformada *wavelet* paralela de una imagen RGB. Se mostró un fragmento en el Listado 5.17.

06/03/00
13:46:57

pvmtbCode/ParRGB.m

2

```
for i=1:numt, pvm_recv(tids(i),TAG); end
                                                    record('Spwn'), label
                MTH=int2str(withmath);
if withdetail,  SLVDIR=DIR; else, SLVDIR=''; end % esclavos salvan a disco?
                THR=int2str(throttle);
%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
for indiceNTIMES=1:NTIMES

%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_chuck','tif'); load h, load g
gd=[  g(2:length(g));0];                            % Detail decomp/recomp
gr=[0;g(1:length(g)-1)];                            record('Load')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% No dependencia secuencial
% Descomposición 3 BitPlanes, 3 niveles % Llevan sus propios record
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Consultar ficheros datos al final
img=lena(:,:,1);
  cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img''); img=double(img);',...
       'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');'];
  pvm_initsend(RAW); pvme_pack(h,gd,gr,cmd); pvm_send(tids(1),TAG);
  pvm_initsend(PLACE); pvm_pack(img);        pvm_send(tids(1),TAG);

img=lena(:,:,2);
  cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img''); img=double(img);',...
       'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');'];
  pvm_initsend(RAW); pvme_pack(h,gd,gr,cmd); pvm_send(tids(2),TAG);
  pvm_initsend(PLACE); pvm_pack(img);        pvm_send(tids(2),TAG);

img=lena(:,:,3); if NUMSLV==3
  cmd=['pvm_recv(pvm_parent,TAG); pvm_unpack(''img''); img=double(img);',...
       'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');'];
  pvm_initsend(RAW); pvme_pack(h,gd,gr,cmd); pvm_send(tids(3),TAG);
  pvm_initsend(PLACE); pvm_pack(img);        pvm_send(tids(3),TAG);
                                                    record('Snd')
else,                                               record('Snd')
          img=double(img);
[trBLUE, rBLUE]=dwt2D3L(img,h,gd,gr,withmath,withdetail,throttle);
                                         if ~withdetail,record('Wav'),label,end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Esperar Descomposición % Si host trabaja, ya habrá llegado. Si no, esperar
%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~throttle
  pvm_recv(tids(1),TAG);pvm_unpack('trRED'  );
  pvm_recv(tids(2),TAG);pvm_unpack('trGREEN'); if NUMSLV==3
  pvm_recv(tids(3),TAG);pvm_unpack('trBLUE' ); end
                                                    record('RcvD'),label
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Trabajar antes de Recv REcomposición %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ¿conviene o no conviene paralelizar "math()"? En caso de que no convenga,
% éste es un buen lugar para dejar tiempo math host hidden
% aprovechando aquí tiempo que se gastaría después, de paso que esperamos Recv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~withmath
  trRED  =math(trRED  {:});
  trGREEN=math(trGREEN{:});
  trBLUE =math(trBLUE {:});                                 record('mth')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recibir Re-composición %
%%%%%%%%%%%%%%%%%%%%%%%%%%
pvm_recv(tids(1),TAG); pvm_unpack('rRED'  );
pvm_recv(tids(2),TAG); pvm_unpack('rGREEN'); if NUMSLV==3
pvm_recv(tids(3),TAG); pvm_unpack('rBLUE' ); end
                                                            record('RcvR'), label
%%%%%%%%%%%%%%%%%%%%%%%%%%
% ¿conviene o no conviene paralelizar "uint8()"?
%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~withmath
  rRED  =uint8(rRED  );
  rGREEN=uint8(rGREEN);
  rBLUE =uint8(rBLUE );                                     record('u8')
end

%%%%%%%%%%%%
% Gráficas %
%%%%%%%%%%%%
if withlena     % math/uint8 ya lo hizo esclavo(withmath) ó host (~withmath)
  transform=cat(3,trRED,trGREEN,trBLUE);
  reconstr =cat(3, rRED, rGREEN, rBLUE);                    record('cat')

  figure,                                                   record('fig')
  imshow(lena), title('Original'),                          record('shw')
  figure,                                                   record('fig')
  imshow(transform), title('Transform'),                    record('shw')
  figure,                                                   record('fig')
  imshow(reconstr), title('Reconstruction'),                record('shw')
end

%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
end

%%%%%%%%%%%%%%
% Acabar PVM %
%%%%%%%%%%%%%%
pvm_exit;               % pvm_halt mataría xpvm, exit sirve para xpvm

        toc                                     % La gráfica nunca cuenta
        recgraph;                               % Salvar datos gráfica
[T R S]=shwgraph(withgraph,withdetail,NUMSLV);  % Ver gráfica si no se impide

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [T,R,S]=shwgraph(withgraph,withdetail,NUMSLV)
global DIR

         HN0=strtok(hostname,'.');
if withdetail
        [HN1 HN2 HN3]=pvmHnames;                % (confiar en $PVMBHOST)
[T R S MSG tm1 lbl1]=stats(DIR,['dwt2D3L_' HN1]); disp(MSG), S
[T R S MSG tm2 lbl2]=stats(DIR,['dwt2D3L_' HN2]); disp(MSG), S, if NUMSLV==3
[T R S MSG tm3 lbl3]=stats(DIR,['dwt2D3L_' HN3]); disp(MSG), S, end
end
[T R S MSG tm0 lbl0 msg]=stats(DIR,[mfilename '_' HN0]); disp(MSG), S

if withgraph
    graph(DIR,mfilename,HN0);                   % Ver gráfica si no se impide
```

Listado E.27: Continuación.

06/03/00
13:46:57

pvmtbCode/ParRGB.m

3

```
 if withdetail
        args={DIR [mfilename '_corr'] MSG 'Snd' msg};
        args={args{:} HN0 tm0 lbl0 HN1 tm1 lbl1 HN2 tm2 lbl2};
    graph(DIR,'dwt2D3L',HN1);                    % y también de esclavos
    graph(DIR,'dwt2D3L',HN2); if NUMSLV==3      % si se pidió detalle
    graph(DIR,'dwt2D3L',HN3);
        args = {args{:} HN3 tm3 lbl3}; end
        timegraph(args{:});
 end
end

clear global
```

Listado E.27: Continuación.

06/01/00 18:50:28 pvmtbCode/dwt2D3L.m 1

```
function [transform,reconstr]=dwt2D3L(img,h,gd,gr, withmath,wd, throttle)
% DWT2D3L              2-D 3-Level DWT and IDWT
%
%       [transform reconstr] = dwt2D3L ( img, h,gd,gr ,withmath,wd, throttle )
%
%  img (matriz) Señal 2-D a analizar. Se devuelve transformada y reconstrucción
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  gd  (vector) Filtro 1-D detalle      (paso alta) para descomposición. lh=lgd
%  gr  (vector) Filtro 1-D detalle      (paso alta) para reconstrucción.   =lgr
%
%  withmath     0/1, indica si se hacen matemáticas (o las hace el host)
%  wd           0/1, indica si se ha de etiquetar (withdetail para host)
%  wd (string)  Directorio salvar datos/gráficas (work dir para esclavo)
%               '' si no se desea escribir en disco nada
%  throttle     0/1,  se obliga que host espere Send Descomposición
%
%  transform    Señal 2-D transformada con organización habitual. Area=img
%               si withmath=0, transform={a3,dh3...dd1}
%  reconstr     Señal 2-D reconstrucción desde nivel 3. Igual a img
%

global TAG RAW PLACE                     % puede pillarlo del host o del esclavo
global NTIMES NUMCMDS indiceBuclePVM    % puede pillarlo del host o del esclavo
%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix DIR labels             % a esclavos no llega global host
if ~ischar(wd), rf=wd;                  % HOST: record flag <=> wd (with detail)
else,  DIR=wd;  rf=~isempty(DIR);       % SLAV: record <=> ''~= wd (work dir)
     if rf
        if isempty(ix)                  % Si 1ª vez del esclavo, reservar anot.
        tm=zeros(10,6); msg={}; ix=0; labels={}; end,   record('')
     end
end
%%% 10: 'D1'-'D3' 'math/Pck' 'SndD' 'R3'-'R1' 'cast' 'SndR'

parent=pvm_parent; if parent<0, parent=0; end
if ~parent & throttle
  global tids trRED trGREEN             % no desempaquetar aquí sino en host
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Descomposición 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[a1 dh1 dv1 dd1]=dwt2(img,h,gd,'spec'); if rf, record('D1'), label, end

[a2 dh2 dv2 dd2]=dwt2( a1,h,gd,'spec'); if rf, record('D2')
                         if (~isempty(NTIMES) & NTIMES <3)|...
                            (~isempty(NUMCMDS)& NUMCMDS<3),  label, end, end

[a3 dh3 dv3 dd3]=dwt2( a2,h,gd,'spec'); if rf, record('D3')
                         if (~isempty(NTIMES) & NTIMES <2)|...
                            (~isempty(NUMCMDS)& NUMCMDS<2),  label, end, end
%%%%%%%%%%%
% "math()" es cuestión de dibujo, no debería estar paralelizado
%   pero estudiar el caso con detalle merece la pena, sobre todo porque
%   esta función tiene 2 partes y la DEScomposición se devuelve primero
%   con lo cual el host puede trabajar con las DEScomposiciones
%   entre los dos Recv, el de las DEScomposiciones y el Recv final (REcomp)
% Al final se envía REcomposición
% Es mejor paralelizar, o es mejor que el host haga math mientras espera Recv?
%%%%%%%%%%%
if withmath
transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);if rf,record('mth'),end
else
transform=    {a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1};
end

if ~parent                                % Host: recibir ya
 if throttle
  pvm_recv(tids(1),TAG); pvm_unpack('trRED'  ); if rf,record('Wr'),label,end
  pvm_recv(tids(2),TAG); pvm_unpack('trGREEN'); if rf,record('Wg'),label,end
 end
else                                      % Esclavos: dejar enviado resultado
  if withmath,  pvm_initsend(PLACE);pvm_pack(transform);
  else,         pvm_initsend(RAW); pvme_pack(transform);if rf,record('Pck'),end
  end
                pvm_send(parent,TAG);           if rf, record('SndD'), end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Re-composición 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r2=idwt2(a3,dh3,dv3,dd3, h,gr, 'spec'); if rf,  record('R3')
                         if (~isempty(NTIMES) & NTIMES <2)|...
                            (~isempty(NUMCMDS)& NUMCMDS<2),  label, end, end

r1=idwt2(r2,dh2,dv2,dd2, h,gr, 'spec'); if rf,  record('R2')
                         if (~isempty(NTIMES) & NTIMES <3)|...
                            (~isempty(NUMCMDS)& NUMCMDS<3),  label, end, end

r0=idwt2(r1,dh1,dv1,dd1, h,gr, 'spec'); if rf,  record('R1'),label, end

%%%%%%%%%%%
% "uint8()" es cuestión de dibujo, no debería estar paralelizado
%%%%%%%%%%%
if withmath,    reconstr=uint8(r0);     if rf, record('u8'), end
else,           reconstr=      r0 ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Esclavos: dejar enviado resultado %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if parent                         % Host no hace recgraph, anota donde aplicación
  pvm_initsend(PLACE); pvm_pack(reconstr);      % recgraph destroza tm, msg...
  pvm_send(parent,TAG);                 if rf, record('SndR')
                        if indiceBuclePVM==NUMCMDS, recgraph;pvm_exit; end, end
end                               % si no se hace pvm_exit, tampoco pasa nada
```

Listado E.28: dwt2D3L.m: Programa esclavo PVMTB para transformada *wavelet* paralela de una imagen RGB a tres niveles. Se mostró un fragmento en el Listado 5.18.

06/01/00 20:14:29 mpitbCode/ParRGB.m **1**

```
function [T,R,S]=ParRGB(ntimes,withgraph,withdetail,...
                                withlena,withmath,nslaves,throttle)
% ParRGB: Ejemplo paralelo color análisis wavelet multiresolución
%
%       [T R S] = ParRGB ( ntimes, withgraph, withdetail,...
%                                   withlena, withmath, nslaves, throttle )
%
%  ntimes       [1],    número de veces a repetir el trabajo (medir tiempo)
%                0 ,    si se desea consultar datos de ejecución anterior
%  withgraph    0/[1],  indica si se desean gráficas de tiempo
%                       el tiempo de gráficas no se incluye, naturalmente
%  withdetail   [0]/1,  indica si se desea desglosar tiempo en paralelo
%                       si se activa, esclavos salvan datos a disco
%  withlena     [0]/1,  indica si se desean ver las imágenes
%                       el tiempo de dibujo se incluye en la gráfica
%  withmath     [0]/1,  indica si paralelizar norm. transformada y typecast
%  nslaves      [2]/3,  número de procesos MATLAB esclavos a arrancar
%  throttle     [0]/1,  el host trabaja y espera D1-D3 esclavos a la mitad

%
%  T    "Times", registro con campos:
%  totl tiempo total gastado (no incluye tiempo para gráfica tiempos)
%  spwn tiempo gastado hasta Spawn MATLAB hijos
%
% R     "Round", tiempo de cada vuelta (de Load a Load)
% S     "Stats", cell-array con columnas {'label' mean stdev ntimes}

%       tomado de Mallat "A Theory for Multiresolution Signal Decomposition:
%                               The Wavelet Representation"
%       IEEE Trans. on Patt.Analysis and Mach.Intell. Vol.11 N.7 July 1989
%
%       3 ordenadores, cada uno descompone un bit plane R-G-B
%       Pueden ser 3 esclavos, ó el host y 2 esclavos

% Necesita:
% lena_chuck.tif        Imagen de Lena en RGB, obtenida de chuck
% h.mat         Filtro Wavelet Aproximación (unidimensional)
% g.mat         Filtro Wavelet Detalle      (unidimensional)
%
% dwt2D3L       Descomposición y Recomposición 3 niveles para cada canal
%   dwt2        Descomposición Wavelet Bi-dimensional
%     dwt       Descomposición Wavelet Unidimensional
%   idwt2       Re-composición Wavelet Bi-dimensional
%     idwt      Re-composición Wavelet Unidimensional
%
% math          Operaciones de normalización para visualizar transformada
% stats         Estadísticas sobre las anotaciones
% record        Anotar tiempos
% label         Anotar etiquetas
% recgraph      Salvar datos gráfica tiempos
% graph         Generación de gráficas a partir de datos
% timegraph     Gráfica de tiempos correlacionados

%       %%%%%% DES/RE-/COMPOSICION %%%%%%%%%%%%%%% dwt2D3L.m %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                               Bitplane   Transform        Reconstr
%     -------- ------------> --------    -------- MPI to --------
%   --------B|               | Slv3 | -\ ||_|___|   -\   |      | -\ MPI to
% --------G| |               | HOST?| -/ |  |   |   -/   |      | -/ parent
% |     R| |--               --------    -------- parent --------
% |      |--   ------------>   Slv2 ...
% --------     ------------>   Slv1 ...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
global NTIMES                   % hizo falta para pasar control label a dwt2D3L
%%%%%%%%%%
% ArgChk %
%%%%%%%%%%
if nargin<7, throttle  =0; else, throttle  =min(max(round(throttle  ),0),1);end
if nargin<6, NUMSLV    =2; else, NUMSLV    =min(max(round(nslaves   ),2),3);end
if nargin<5, withmath  =0; else, withmath  =min(max(round(withmath  ),0),1);end
if nargin<4, withlena  =0; else, withlena  =min(max(round(withlena  ),0),1);end
if nargin<3, withdetail=0; else, withdetail=min(max(round(withdetail),0),1);end
if nargin<2, withgraph =1; else, withgraph =min(max(round(withgraph ),0),1);end
if nargin<1, NTIMES    =1; else, NTIMES    =    max(round(ntimes    ),0   );end
if withmath & ~withlena
  warning('withmath==1 and withlena==0'), end
if NUMSLV==3 & throttle,  throttle=0;
  warning('Unsensible throttle  ==1 while NUMSLV   ==3, setting to 0'), end

global DIR, DIR=[ getenv('MPITB_ROOT') '/Users99/' mfilename ];
if ~NTIMES, [T R S]=shwgraph(withgraph,withdetail,NUMSLV); return, end

if throttle, global Rreq Greq, end      % para throttle en dwt2D3L

tic
%%%%%%%%%%%%%%%%%%% Número de esclavos==2/3
% Test daemon MPI % No sería justo contar tiempo, MPI exige lamboot previo
%%%%%%%%%%%%%%%%%%
[info flag]=MPI_Initialized;
if info | ~flag, error('MPI not initialized, run MPI_Init'), end
[info attr flag]=MPI_Attr_get(MPI_COMM_WORLD,MPI_UNIVERSE_SIZE);
if info | ~flag, error('MPI_UNIVERSE_SIZE keyval undefined?'), end
if attr<NUMSLV+1, error('not enough computers on LAM'), end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% lamboot/$LAMBHOST debería haber arrancado suficientes hosts
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix labels
tm=zeros( 2 + NTIMES*( 5+ 9*withdetail + withlena*(8+(1-withmath)) ), 6);
                        msg={}; ix=0; labels={};        record('')
%%% 2: '' 'Spwn'
%%% 4*: 'Load' 'Snd' ('Wav') 'Recv' ('math') 'Recv' (lena)
%%%(1+9*detl)*:  'Wav'== 'D1'-'D3' 'math/grp' 'Snd' 'R3'-'R1' 'cast/asgn' 'Snd'
%%%(lena*~math)*:'math'
%%% 8*lena*:     'cat' 'ca/st' 3*('fig' 'shw')
%%% cada bucle va de 'Load' a 'Load'
%%% tomamos el caso peor en que NUMSLV==2, si NUMSLV==3 no hay 'Wav'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arranque MATLAB's esclavos % Número de esclavos==2/3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DISP=getenv('DISPLAY');
% [info children errs]=MPI_Comm_spawn('xterm',...
%                {'-display' DISP '-e' 'matlab'},NUMSLV,'NULL',0,'SELF');
  [info children errs]=MPI_Comm_spawn('matlab',...
                                   {'-nosplash'},NUMSLV,'NULL',0,'SELF');
if info | any(errs),    error('cannot start MATLABs'), end
[info numt] = MPI_Comm_remote_size(children);
if info | numt<NUMSLV,  error('cannot start MATLABs'), end
global NEWORLD                      % Cansado pasarlo a [i]dwt2ParHost
[info NEWORLD]=MPI_Intercomm_merge(children,0);
if info,                error('cannot start MATLABs'), end
```

Listado E.29: ParRGB.m: Programa maestro MPITB para transformada *wavelet* paralela de una imagen RGB. Se mostró un fragmento en el Listado 5.19.

06/01/00
20:14:29

mpitbCode/ParRGB.m

2

```
MPI_Errhandler_set(NEWORLD,MPI_ERRORS_RETURN);

global TAG
TAG=7; NUMCMDS=1*NTIMES; QUIT=1;        % Enviaremos 1 des/re-composición
for i=1:numt
  info = MPI_Send(NUMCMDS,i,TAG,NEWORLD);
  info = MPI_Send(QUIT,   i,TAG,NEWORLD);
end                                     % Ignorar respuesta confirmación
for i=1:numt, [info stat]=MPI_Recv(NUMCMDS,i,TAG,NEWORLD); end
                                                        record('Spwn'), label
                MTH=int2str(withmath);
if withdetail,  SLVDIR=DIR; else, SLVDIR=''; end % esclavos salvan a disco?
                THR=int2str(throttle);
%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
for indiceNTIMES=1:NTIMES

%%%%%%%%%%%%%%%%%%%%
% Imagen y Filtros %
%%%%%%%%%%%%%%%%%%%%
lena=imread('lena_chuck','tif'); load h, load g
gd=[  g(2:length(g));0];                                % Detail decomp/recomp
gr=[0;g(1:length(g)-1)];                                record('Load')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% No dependencia secuencial
% Descomposición 3 BitPlanes, 3 niveles % Llevan sus propios record
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Consultar ficheros datos al final
img=lena(:,:,1); siz=size(img);
cmd=['siz=[0 0];        MPI_Recv(siz,  0,TAG,NEWORLD);',...
     'img=zUint8(siz);  MPI_Recv(img,  0,TAG,NEWORLD); img=double(img);',...
     'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');' ];
[info psiz]=MPE_Pack_size(h,gd,gr,cmd,NEWORLD);  pbuf=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,gd,gr,cmd,pbuf,0,NEWORLD);
                 MPI_Send(pbuf,        1,TAG,NEWORLD);
                 MPI_Send(siz,         1,TAG,NEWORLD);
                 MPI_Send(img,         1,TAG,NEWORLD);

img=lena(:,:,2); siz=size(img);
cmd=['siz=[0 0];        MPI_Recv(siz,  0,TAG,NEWORLD);',...
     'img=zUint8(siz);  MPI_Recv(img,  0,TAG,NEWORLD); img=double(img);',...
     'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');' ];
[info psiz]=MPE_Pack_size(h,gd,gr,cmd,NEWORLD);  pbuf=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,gd,gr,cmd,pbuf,0,NEWORLD);
                 MPI_Send(pbuf,        2,TAG,NEWORLD);
                 MPI_Send(siz,         2,TAG,NEWORLD);
                 MPI_Send(img,         2,TAG,NEWORLD);

img=lena(:,:,3); siz=size(img); if NUMSLV==3
cmd=['siz=[0 0];        MPI_Recv(siz,  0,TAG,NEWORLD);',...
     'img=zUint8(siz);  MPI_Recv(img,  0,TAG,NEWORLD); img=double(img);',...
     'dwt2D3L(img,h,gd,gr,' MTH ',''' SLVDIR ''',' THR ');' ];
[info psiz]=MPE_Pack_size(h,gd,gr,cmd,NEWORLD);  pbuf=zeros(1,ceil(psiz/8));
                 MPE_Pack(h,gd,gr,cmd,pbuf,0,NEWORLD);
                 MPI_Send(pbuf,        3,TAG,NEWORLD);
                 MPI_Send(siz,         3,TAG,NEWORLD);
                 MPI_Send(img,         3,TAG,NEWORLD);
end
                                                        record('Snd')
img=double(img);                                        record('dbl')
                                        % para calcular tamaño empaquetado
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dejar preparado Irecv Descomposición %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if withmath
  trRED  =zUint8(siz);  [i Rreq]=MPI_Irecv(trRED  ,1,TAG,NEWORLD);
  trGREEN=zUint8(siz);  [i Greq]=MPI_Irecv(trGREEN,2,TAG,NEWORLD); if NUMSLV==3
  trBLUE =zUint8(siz);  [i Breq]=MPI_Irecv(trBLUE ,3,TAG,NEWORLD); end
else            % Aprovechamos que podemos acotar el tamaño empaquetado. Fiu!
  [info psiz1]=MPI_Pack_size({[0] [1] [2] [3] [4] [5] [6] [7] [8] [9]},NEWORLD);
  [info psiz2]=MPI_Pack_size(img,NEWORLD);      psiz=psiz1+psiz2;
  trRED  =zeros(1,psiz);[i Rreq]=MPI_Irecv(trRED  ,1,TAG,NEWORLD);
  trGREEN=zeros(1,psiz);[i Greq]=MPI_Irecv(trGREEN,2,TAG,NEWORLD); if NUMSLV==3
  trBLUE =zeros(1,psiz);[i Breq]=MPI_Irecv(trBLUE ,3,TAG,NEWORLD); end
end
                                                        record('Buf')
                                                if NTIMES<5,label,end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Trabajar durante DEScomposición % dwt2D3L va chequeando con Testsome/Iprobe
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if NUMSLV==2
[trBLUE, rBLUE]=dwt2D3L(img,h,gd,gr,withmath,withdetail,throttle);
                                    if ~withdetail,record('Wav'),label,end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Esperar Descomposición % Si host trabaja, ya habrá llegado. Si no, esperar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        MPI_Wait(Rreq);
        MPI_Wait(Greq); if NUMSLV==3
        MPI_Wait(Breq); end,   if NUMSLV==3|~withdetail,record('Wait')
                                                if NTIMES<5,label,end,  end
if ~withmath
        MPI_Unpack(trRED  ,0,NEWORLD,'trRED'  );
        MPI_Unpack(trGREEN,0,NEWORLD,'trGREEN'); if NUMSLV==3
        MPI_Unpack(trBLUE ,0,NEWORLD,'trBLUE' ); end,   record('Upk')
                                                if NTIMES<5,label,end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Trabajar antes de Recv REcomposición %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ¿conviene o no conviene paralelizar "math()"? En caso de que no convenga,
% éste es un buen lugar para dejar tiempo math host hidden
% aprovechando aquí tiempo que se gastaría después, de paso que esperamos Recv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~withmath
  trRED  =math(trRED  {:});
  trGREEN=math(trGREEN{:});
  trBLUE =math(trBLUE {:});                             record('mth')
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Recibir Re-composición %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if withmath,    rRED  =zUint8(siz);
                rGREEN=zUint8(siz); if NUMSLV==3
                rBLUE =zUint8(siz);  end
else,           rRED  =zeros (siz);
                rGREEN=zeros (siz); if NUMSLV==3
                rBLUE =zeros (siz); end
end
                                                        record('buf')
    MPI_Recv(rRED,  1,TAG,NEWORLD);
    MPI_Recv(rGREEN,2,TAG,NEWORLD); if NUMSLV==3
```

Listado E.29: Continuación.

06/01/00
20:14:29

mpitbCode/ParRGB.m

3

```
    MPI_Recv(rBLUE, 3,TAG,NEWORLD); end
                                                        record('RcvR'), label
%%%%%%%%%%%%%%%%%%%%%%%%%%
% ¿conviene o no conviene paralelizar "uint8()"?
%%%%%%%%%%%%%%%%%%%%%%%%%%
if ~withmath
  rRED  =uint8(rRED  );
  rGREEN=uint8(rGREEN);
  rBLUE =uint8(rBLUE );                                 record('u8')
end

%%%%%%%%%%%%
% Gráficas %
%%%%%%%%%%%%
if withlena     % math/uint8 ya lo hizo esclavo(withmath) ó host (~withmath)
  transform=cat(3,trRED,trGREEN,trBLUE);
  reconstr =cat(3, rRED, rGREEN, rBLUE);                record('cat')

  figure,                                               record('fig')
  imshow(lena), title('Original'),                      record('shw')
  figure,                                               record('fig')
  imshow(transform), title('Transform'),                record('shw')
  figure,                                               record('fig')
  imshow(reconstr), title('Reconstruction'),            record('shw')
end

%%%%%%%%%%%%%%%%
% BUCLE NTIMES % En teoría usaríamos una imagen distinta cada vez
%%%%%%%%%%%%%%%%
end

%%%%%%%%%%%%%%
% Acabar MPI %
%%%%%%%%%%%%%%
% info=MPI_Finalize;                            % No permitiría otro MPI_Init

        toc                                     % La gráfica nunca cuenta
        recgraph;                               % Salvar datos gráfica
[T R S]=shwgraph(withgraph,withdetail,NUMSLV);  % Ver gráfica si no se impide

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [T,R,S]=shwgraph(withgraph,withdetail,NUMSLV)
global DIR

         HN0=strtok(hostname,'.');
if withdetail
        [HN1 HN2 HN3]=parsebhost;               % (confiar round-robin spawn)
[T R S MSG tm1 lbl1]=stats(DIR,['dwt2D3L_' HN1]); disp(MSG), S
[T R S MSG tm2 lbl2]=stats(DIR,['dwt2D3L_' HN2]); disp(MSG), S, if NUMSLV==3
[T R S MSG tm3 lbl3]=stats(DIR,['dwt2D3L_' HN3]); disp(MSG), S, end
end
[T R S MSG tm0 lbl0 msg]=stats(DIR,[mfilename '_' HN0]); disp(MSG), S

if withgraph
    graph(DIR,mfilename,HN0);                   % Ver gráfica si no se impide
 if withdetail
        args={DIR [mfilename '_corr'] MSG 'Snd' msg};
        args={args{:} HN0 tm0 lbl0 HN1 tm1 lbl1 HN2 tm2 lbl2};
    graph(DIR,'dwt2D3L',HN1);                   % y también de esclavos
    graph(DIR,'dwt2D3L',HN2); if NUMSLV==3      % si se pidió detalle
    graph(DIR,'dwt2D3L',HN3);
        args = {args{:} HN3 tm3 lbl3}; end
        timegraph(args{:});
 end
end

clear global
```

Listado E.29: Continuación.

06/01/00 19:23:56 — mpitbCode/dwt2D3L.m — 1

```
function [transform,reconstr]=dwt2D3L(img,h,gd,gr, withmath,wd,throttle)
% DWT2D3L              2-D 3-Level DWT and IDWT
%
%       [transform reconstr] = dwt2D3L ( img, h,gd,gr ,withmath,wd,throttle )
%
%  img (matriz) Señal 2-D a analizar. Se devuelve transformada y reconstrucción
%  h   (vector) Filtro 1-D aproximación (paso baja)
%  gd  (vector) Filtro 1-D detalle      (paso alta) para descomposición. lh=lgd
%  gr  (vector) Filtro 1-D detalle      (paso alta) para reconstrucción.   =lgr
%
%  withmath     0/1, indica si se hacen matemáticas (o las hace el host)
%  wd           0/1, indica si se ha de etiquetar (withdetail para host)
%  wd (string)  Directorio salvar datos/gráficas (work dir para esclavo)
%               '' si no se desea escribir en disco nada
%  throttle     0/1,  se avisa con msg a host que espere Send Descomposición
%
%  transform    Señal 2-D transformada con organización habitual. Area=img
%               si withmath=0, transform={a3,dh3...dd1}
%  reconstr     Señal 2-D reconstrucción desde nivel 3. Igual a img
%

global TAG NEWORLD                      % puede pillarlo del host o del esclavo
global NTIMES NUMCMDS indiceBucleMPI    % puede pillarlo del host o del esclavo
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Preallocation tiempos % Variables para record/recgraph
%%%%%%%%%%%%%%%%%%%%%%%%%%
global tm msg ix DIR labels             % a esclavos no llega global host
[info rank]=MPI_Comm_rank(NEWORLD);     % Para saber si Test/Send
if ~ischar(wd), rf=wd;                  % HOST: record flag <=> wd (with detail)
else,  DIR=wd;  rf=~isempty(DIR);       % SLAV: record <=> ''~= wd (work dir)
     if rf
        if isempty(ix)                  % Si 1ª vez del esclavo, reservar anot.
        tm=zeros(10,6); msg={}; ix=0; labels={}; end,   record('')
     end
end
%%% 10: 'D1'-'D3' 'math/Pck' 'SndD' 'R3'-'R1' 'cast' 'SndR'

if throttle, global Rreq Greq, end      % peticiones de ParRGB para throttle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Descomposición 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[a1 dh1 dv1 dd1]=dwt2(img,h,gd,'spec'); if rf, record('D1'), label, end

[a2 dh2 dv2 dd2]=dwt2( a1,h,gd,'spec'); if rf, record('D2')
                         if (~isempty(NTIMES) & NTIMES <3)|...
                            (~isempty(NUMCMDS)& NUMCMDS<3),  label, end, end

[a3 dh3 dv3 dd3]=dwt2( a2,h,gd,'spec'); if rf, record('D3')
                         if (~isempty(NTIMES) & NTIMES <2)|...
                            (~isempty(NUMCMDS)& NUMCMDS<2),  label, end, end
%%%%%%%%%%%
% "math()" es cuestión de dibujo, no debería estar paralelizado
%   pero estudiar el caso con detalle merece la pena, sobre todo porque
%   esta función tiene 2 partes y la DEScomposición se devuelve primero
%   con lo cual el host puede trabajar con las DEScomposiciones
%   entre los dos Recv, el Irecv (DEScomposiciones) y el Recv final (REcomp)
% Al final se envía REcomposición
% Es mejor paralelizar, o es mejor que el host haga math mientras espera Recv?
%%%%%%%%%%%
if withmath
transform=math(a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1);if rf,record('mth'),end
else
transform=    {a3,dh3,dv3,dd3,dh2,dv2,dd2,dh1,dv1,dd1};
end

if ~rank
 if throttle                               % Host: recibir ya
  MPI_Wait(Rreq);                          if rf,record('Wr'),label,end
  MPI_Wait(Greq);                          if rf,record('Wg'),label,end
 end
else                                       % Esclavos: dejar enviado resultado
  if withmath,  MPI_Send(transform,0,TAG,NEWORLD);
  else,         [info psiz]=MPI_Pack_size(transform,NEWORLD);
                pbuf=zeros(1,ceil(psiz/8));     if rf, record('Buf'), end
        MPI_Pack(transform,pbuf,0,NEWORLD);     if rf, record('Pck'), end
        MPI_Send      (pbuf,0,TAG,NEWORLD);
  end,                                          if rf, record('SndD'), end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Re-composición 3 niveles %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
r2=idwt2(a3,dh3,dv3,dd3, h,gr, 'spec'); if rf,  record('R3')
                         if (~isempty(NTIMES) & NTIMES <2)|...
                            (~isempty(NUMCMDS)& NUMCMDS<2),  label, end, end

r1=idwt2(r2,dh2,dv2,dd2, h,gr, 'spec'); if rf,  record('R2')
                         if (~isempty(NTIMES) & NTIMES <3)|...
                            (~isempty(NUMCMDS)& NUMCMDS<3),  label, end, end

r0=idwt2(r1,dh1,dv1,dd1, h,gr, 'spec'); if rf,  record('R1'),label, end

%%%%%%%%%%%
% "uint8()" es cuestión de dibujo, no debería estar paralelizado
%%%%%%%%%%%
if withmath,    reconstr=uint8(r0);     if rf, record('u8'), end
else,           reconstr=      r0 ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Esclavos: dejar enviado resultado %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if rank                         % Observar que recgraph destroza tm, msg...
  MPI_Send(reconstr,0,TAG,NEWORLD);     if rf, record('SndR')
                        if indiceBucleMPI==NUMCMDS, recgraph, end, end
end
```

Listado E.30: dwt2D3L.m: Programa esclavo MPITB para transformada *wavelet* paralela de una imagen RGB a tres niveles. Se mostró un fragmento en el Listado 5.20.

Apéndice F

Glosario

Se relacionan a continuación las siglas y vocabulario especializado más comúnmente usado a lo largo de esta memoria.

Amdahl (ley de): Cuando una mejora es aplicable a una parte de un sistema, el efecto global de la mejora está limitado por la parte no susceptible de mejora.

Así, si F es la fracción de una aplicación que no se puede paralelizar, la ganancia en prestaciones que se puede obtener con P procesadores es $\frac{P}{1+F(P-1)}$, cantidad que está limitada por $\frac{1}{F}$, valor al que tiende la primera expresión cuando el número de procesadores P se incrementa ilimitadamente.

Esta ley fue formalizada por primera vez en el artículo de Gene Amdahl de 1967 [3].

API: *Application Program Interface*, Interfaz para Programas de Aplicación. Biblioteca proporcionada usualmente por el fabricante de un sistema software o un dispositivo físico para permitir invocar funciones del mismo desde otros programas desarrollados por el usuario. Provee un nivel de abstracción que permite la portabilidad de código. Programando con una API se evita programar directamente un dispositivo hardware o un sistema software. Así, la API MATLAB permite llamar funciones MATLAB desde un programa C, y las APIs OpenGL y Direct3D ocultan el acelerador gráfico al programador de aplicaciones gráficas.

array: vector o matriz. En los lenguajes de programación se suele disponer de un tipo de datos que permite almacenar varios datos de un mismo tipo base, y acceder a ellos por indexación. El nuevo tipo de datos se denomina *array de <tipo-base>*, por ejemplo, array de enteros.

El significado original denota una organización alineada, real o figurada, como por ejemplo en "*DSP array*".

ATLAS: *Automatically Tuned Linear Algebra Software*, Software de Algebra Lineal Automáticamente Afinado. Interfaz a BLAS y LAPACK que de forma empírica adapta dicho software a la máquina en que se instala. Comprueba las distintas ganancias debidas a la cache de datos, reutilización de la cache de instrucciones, cambio de la estructura de los bucles, reordenación de instrucciones FPU, bloqueo de registros y cache, etc, escogiendo la combinación más eficiente en cada máquina concreta. Disponible en Netlib, **http://www.netlib.org/atlas/**

bastón: Neurona especializada como fotorreceptor. Producen en su axón un nivel contínuo de tensión eléctrica según la iluminación recibida, al contrario que las neuronas de impulsos. Son muy sensibles, llegando casi a registrar fotones aislados a nivel de iluminación muy bajo (escotópico). Excitables según la energía del fotón recibido, lo cual los hace máximamente receptivos al azul o violeta, lo cual explica que una escena poco iluminada se perciba como gris azulada

Beowulf: Un (cluster) Beowulf es un computador paralelo de altas prestaciones, construido a partir de componentes hardware comunes (commodity, ver COTS), con un S.O. libre como Linux o FreeBSD, interconectado mediante una red privada de altas prestaciones. Los computadores del cluster suelen ser PCs o Estaciones de Trabajo. Consultar la FAQ Beowulf en alguno de los links en la bibliografía, p.ej. **http://www.beowulf.org/howto/howto.html**. En [83] se ofrecen otras acepciones del término.

big endian: "de los del lado grueso", ªnchoextremistas". Un huevo tiene dos extremos, el grueso y el fino. En los "Viajes de Gulliver", Johnatan Swift relata las aventuras de un reino en el que los habitantes se dividen en partidarios de cascar un huevo duro por el extremo grueso o por el fino. Los partidarios de esta última opción se denominaban “little endians”.

Similares “guerras religiosas” pueden desatarse sobre la conveniencia de que un computador considere como más significativa la primera o la última palabra de un dato que ocupe varias palabras de almacenamiento. La arquitectura de datos en la cual se considera menos significativa la palabra (parte de un dato multi-palabra) almacenada en una dirección más baja (menor, más próxima a la dirección 0), se denomina “little endian”.

En los “Viajes de Gulliver”, el protagonista requisa la flota de los “little endians” antes de que puedan entrar en batalla.

BLACS: *Basic Linear Algebra Communication Subprograms*, Subprogramas de Comunicación para Algebra Lineal Básica. Paquete de soporte para ScaLAPACK, basado en PVM o MPI. Disponible en Netlib, **http://www.netlib.org/blacs/**

BLAS: *Basic Linear Algebra System*, Sistema Básico de Algebra Lineal. Biblioteca de llamadas o “bloques constructivos” para realizar operaciones vectoriales y matriciales, organizada en 3 niveles. BLAS 1 contiene las operaciones vector-vector, BLAS 2 las matriz-vector, y BLAS 3 las matriz-matriz. Disponible en Netlib, **http://www.netlib.org/blas/**

BMCV2000: *IEEE International WorkShop on Biologically Motivated Computer Vision*, **http://image.korea.ac.kr/BMCV2000/committee.html**

BOOTP: *Internet Bootstrap Protocol*, Protocolo de Arranque Internet (RFC-951, RFC-1532, RFC-1533). Método según el cual un ordenador sin disco, conectado a la red, conociendo únicamente su dirección hardware (usualmente Ethernet) puede obtener una dirección IP, el fichero imagen a arrancar, y servicio de disco.

broadcast: Difusión, transmisión a todos los destinos. *IP Broadcast*: consultar RFC-0919, RFC-0922, por ejemplo (*Broadcasting Internet datagrams in the presence of subnets*).

buffer: Almacenamiento temporal. Zona de memoria destinada a almacenar datos temporalmente. Los drivers* de dispositivo usan buffers y DMA* para no ocupar la CPU durante

las relativamente lentas operaciones de lectura y escritura. Los dispositivos más rápidos (disco, red. . .) disponen incluso de otro buffer “hardware” distinto del proporcionado por el driver/S.O. en memoria principal.

cluster: Racimo. Es un tipo de sistema para procesamiento paralelo o distribuido [64, p.5 §1.2]. Consiste en un conjunto de computadores (PCs, estaciones de trabajo, etc) interconectados con alguna tecnología de red, que trabajan juntos como un único recurso de cálculo integrado. En clusters homogéneos, todos los nodos tienen similar arquitectura y el mismo sistema operativo instalado; en clusters heterogéneos, los nodos podrán diferir en arquitectura y sistema operativo instalado. Entre sus atractivos se puede destacar que son escalables, su construcción es abordable, tienen un bajo coste, usan hardware común, tecnología de red de altas prestaciones como Gigabit Ethernet o Myrinet, y componentes software estándar como UNIX, MPI, PVM.

cono: Neurona especializada como fotorreceptor. Existen tres tipos, Rojo, Verde y Azul, según la longitud de onda a la que son más receptivos. Producen en su axón un nivel contínuo de tensión eléctrica según la iluminación recibida, al contrario que las neuronas de impulsos. Requieren nivel de iluminación alto (fotópico). Tamaño menor que los bastones.

COTS: *Commercial Off-The-Shelf*, equipo disponible comercialmente “directamente de la estantería”. Se aplica a equipo, hardware y software, de distribución generalizada y disponibilidad inmediata, que no hay que encargar o solicitar específicamente. Los PCs, *switches*, tarjetas de red y cableado, así como los Sistemas Operativos típicamente usados con PCs, son ejemplos de equipamiento COTS.

daemon: Demonio, proceso servidor que atiende peticiones de aplicaciones clientes. Típicamente un *daemon* se ejecuta en el *background* (segundo plano) al arrancar el ordenador, permaneciendo conectado a un puerto/*socket* a la espera de peticiones de servicio. Los servicios más conocidos se arrancan dinámicamente en demanda mediante el *superdaemon* inetd.

DMA: *Direct Memory Access*, Acceso Directo a Memoria. Mecanismo por el cual un controlador puede usar el bus temporalmente para leer o escribir en memoria principal, sin intervención de la CPU. Utilizado en conjunción con un controlador de dispositivo, que consume o produce los datos leídos o a escribir, respectivamente. Mecanismo diseñado para funcionar por robo de ciclo en sistemas monoprocesador, aunque por extensión también se denomina DMA en sistemas SMP*.

dominio de colisión: En un sistema de interconexión basado en conmutación de paquetes se suele disponer de un medio de transmisión compartido. Si dos equipos (computadores) intentan acceder al medio simultáneamente se produce un error de transmisión denominado colisión, debiendo reintentar ambos equipos la transmisión. El conjunto de equipos que pueden colisionar entre sí forman un único dominio de colisión.

driver: Gestor. Rutinas de control y estado para manejar un dispositivo conectado a un computador. Proporciona las llamadas requeridas por el sistema operativo concreto para poder abrir, leer, escribir, consultar estado, fijar estado y cerrar el dispositivo. En la mayoría de sistemas operativos, esto permite manejar el dispositivo como si se tratara de un fichero.

DSM: *Distributed Shared Memory*, Memoria Compartida Distribuida. Sistema que permite usar la memoria (distribuida) de distintos computadores como si fuera compartida. Ver VSM y la referencia [38].

DSP: *Digital Signal Processor*, Procesador Digital de Señal. CPU con repertorio máquina especializado en tareas frecuentemente encontradas en aplicaciones de procesamiento de señal: Producto y Acumulación, Convolución, Filtrado, FFT, etc. Suelen incorporar elementos hardware específicos en su arquitectura, como temporizadores, conversores A/D y D/A, *codecs*, puertos serie de alta velocidad, etc.

Están muy extendidos los Kits de Desarrollo, en los cuales se ofrece una mini-tarjeta de evaluación con el DSP y la circuitería de apoyo necesaria para conectar los periféricos más usuales (teclado numérico, leds, micrófono, altavoces, conexión con un PC mediante el puerto serie, etc.) junto con el software de desarrollo (compilador cruzado para PC, monitor, depurador, incluso simulador) y los manuales correspondientes a un precio realmente asequible.

EISPACK: *Eigenvalue, Eigenvector, Singular value decomposition Package*. Paquete para autovalores, autovectores y descomposición en valores singulares. Escrito en FORTRAN. Superado por LAPACK. Disponible en Netlib, **http://www.netlib.org/eispack/**

Embarrassingly parallel: Embarazosamente paralela. Aplicación cuya paralelización es trivial, no requiriendo apenas modificación del código secuencial, sincronización o intercambio de información. Se espera que el *speedup** de una aplicación embarazosamente paralela sea virtualmente lineal, ya que el código ejecutado es virtualmente idéntico al secuencial y no hay apenas comunicación de datos. Es difícil determinar a quién debe atribuirse la acuñación del término. En la FAQ del newsgroup `comp.parallel` se comenta el reto de Alan Karp, quien parece atribuir el término a Cleve Moler,
http://www.mathworks.com/company/cleve_bio.shtml.
Barry Wilkinson [95] menciona como fuente a Geoffrey Fox,
http://www.npac.syr.edu/users/gcf/homepage/.

FFTW: *Fastest Fourier Transform in the West*. Biblioteca de subrutinas C para calcular la Transformada de Fourier (Discreta) en una o más dimensiones, sobre datos reales o complejos de longitud arbitraria. **http://www.fftw.org/**. FFTW ganó el Premio Wilkinson al Software Numérico en 1999, **http://www-fp.mcs.anl.gov/wilkinson/award/3rd-1999.htm**

flag: Bandera, indicador. Valor booleano (0 o 1) que indica que se cumple determinada condición. Así, muchas CPUs disponen de flags de estado indicando propiedades del resultado de la última operación (Zero, Negative, oVerflow. . .). Por extensión, en el contexto de lenguajes de programación, las variables lógicas que agrupan varias condiciones, manejándose por tanto bit a bit, así como las constantes usadas para consultar y alterar los bits individualmente mediante enmascaramiento, se denominan flags también.

FTP: *File Transfer Protocol*, Protocolo de Transferencia de Ficheros. Permite copiar ficheros entre dos ordenadores conectados a Internet. Consultar RFC-0414 por ejemplo (FTP status and further comments).

fóvea: Zona de la retina con mayor resolución espacial, donde la distribución de fotorreceptores de color (conos) es más densa. Carece de fotorreceptores bastones (monocromáticos), que son de mayor tamaño y sensibilidad. Es por tanto prácticamente ciega a bajo nivel de iluminación (escotópico), semifuncional a nivel medio (mesópico), y óptima con buen nivel de iluminación (fotópico). Está ligeramente descentrada para coincidir con el punto de vergencia en visión próxima. Al objeto de dejar sitio para más receptores, las correspondientes neuronas (bipolares, horizontales, amacrinas, ganglios) y sus axones son desalojados, situándose alrededor.

Grand Challenges: Grandes Retos, aplicaciones comúnmente consideradas como de altas prestaciones: predicción del tiempo atmosférico y clima, superconductividad, biología estructural, diseño farmacéutico, dinámica de vehículos, voz y visión por computador... Consultar la FAQ de comp.parallel,
http://wotug.ukc.ac.uk/parallel/internet/usenet/comp.parallel/FAQ/22

GUI: *Graphical User Interface*, Interfaz Gráfico de Usuario. Programa que presenta al usuario una o varias ventanas, posiblemente con menús, listas de selección, cuadros de edición, recuadros de selección, botones de acción, zonas gráficas (canvas), reguladores (slider), etc, permitiéndole proporcionar datos de entrada y recibir salida del programa de forma gráfica, sin teclear.

handshake: Chocar (estrechar) la mano. Juego de palabras, ya que el sentido de la expresión es "indicación de conformidad". Las operaciones que implican a varios procesos o dispositivos pueden requerir sincronización entre ellos. Tomemos por ejemplo la transferencia de datos entre dos dispositivos de los que no se conoce de antemano su velocidad. Se proporciona por tanto un par de líneas de control. Con la primera, el emisor puede indicar la intención de transmitir (activándola) y que los datos han sido emitidos (desactivándola). Con la otra, el receptor podría indicar que está listo para recibir y que los datos han sido recibidos. Cada dispositivo espera la reacción del otro antes de pasar a la siguiente etapa, lo cual garantiza la operación correcta independientemente de las velocidades relativas de los dispositivos. En total hay 4 cambios de estado, llamándose a este método *handshake* completo a 4 bandas.

La situación es similar a la forma en que se estrechan las manos, siendo ésta también una operación que requiere sincronización (si ha de tener éxito).

Si se conocen de antemano las velocidades relativas, es posible simplificar a 2 bandas. Existe otra extensión del método a 6 bandas, apta para broadcast/multicast*. Por analogía, la nomenclatura se aplica también a procesos ejecutándose concurrentemente realizando una operación que requiere sincronización.

hardware: Ferretería, quincalla. Juego de palabras: para distinguir el soporte físico de un computador de su soporte lógico, se recurre a denominar al primero "hardware" y al segundo "software" sustituyendo el adjetivo "hard" (duro) por "soft" (blando). Se refiere a que el software no tiene entidad física, no es tangible. Llevando más lejos aún el juego de palabras, se denomina "firmware" al software asociado íntimamente con un hardware determinado, almacenado por tanto en un soporte físico permanente no magnético. Por ejemplo, los

programas POST y BIOS de un PC, el intérprete de una impresora PostScript, o el propio microcódigo de una CPU microprogramada son firmware.

El vocablo ha sido aceptado en el Diccionario Español (con la acepción descrita, no la original inglesa de ferretería).

HPC: *High Performance Computing*, Computación de Altas Prestaciones. Se aplica este calificativo a las aplicaciones de cómputo que requieren mayores prestaciones que las normalmente disponibles con una estación de trabajo o computador personal. Por extensión, equipos que posibilitan la realización de dichas aplicaciones: estaciones de trabajo científicas, supercomputadores, redes de altas prestaciones, etc. Las aplicaciones típicamente HPC se denominan también “Grand Challenges”*. Al avanzar la tecnología, algunas aplicaciones dejan de ser consideradas como grandes retos.

HTC: *High Throughput Computing*, Computación de alto rendimiento. Sistema de Computación en el que se intenta maximizar el uso de los procesadores. Los sistemas de colas y equilibradores de carga son ejemplos de sistemas HTC. Ver la cita [72].

hub: Centro, eje. En el contexto de redes de computadores, concentrador, dispositivo de comunicaciones al que se conectan varios equipos terminales de datos, por analogía con el eje de una rueda. Un hub funciona como repetidor de paquetes, permitiendo conectar varios equipos a un único punto de conexión a la red (de ahí el término *concentrador*). Ver también *switch*.

IA32: *Intel Architecture 32 bits*, Arquitectura Intel de 32 bits. Familia de CPUs a la que pertenecen el 80386, Pentium, Pentium 4, etc.

ILP: *Instruction Level Parallelism*, Paralelismo a Nivel de Instrucción. Una secuencia de instrucciones a ejecutar puede carecer de dependencias entre ellas, de manera que podrían ejecutarse simultáneamente en distintas unidades funcionales de una CPU. Una inteligente reordenación de las instrucciones máquina por parte del programador o del compilador puede mejorar el tiempo de ejecución de un programa sobre una CPU con múltiples unidades funcionales.

LAM: *Local Area Multicomputer*, Multicomputador de Area Local. Implementación del estándar de paso de mensajes MPI, desarrollada y mantenida por la Universidad de Notredame, IN, USA. **http://www.lam-mpi.org**.

LAPACK: *Linear Algebra Package*, Paquete de Algebra Lineal. Escrito en FORTRAN77, resuelve los problemas de eficiencia de LINPACK y EISPACK en máquinas de memoria compartida, reorganizando los algoritmos para usar pequeños bloques de matrices en los bucles más internos, lo cual maximiza los aciertos de cache. Requiere software que implemente eficientemente estas operaciones en bloque, como por ejemplo ATLAS* (BLAS* 3). Disponible en Netlib, **http://www.netlib.org/lapack/**

LGN: Núcleo geniculado lateral, estructura ubicada en el tálamo a la cual proyectan sus axones los ganglios retinales, y de la cual parten axones hacia el córtex óptico.

LINPACK: *Linear Algebra Package*. Colección de rutinas FORTRAN para analizar y resolver sistemas de ecuaciones lineales y problemas lineales de mínimos cuadrados. Diseñado para computadores de los '70, ha sido superado por LAPACK, más apropiado para ejecución eficiente en supercomputadores vectoriales y de memoria compartida. Disponible en Netlib, **http://www.netlib.org/linpack/**

little endian: "de los del lado fino". Cuando un computador maneja datos de longitud superior a la de una posición de memoria, debe escoger entre dos opciones: almacenar primero la parte más significativa (*big** endian), o la menos significativa (*little* endian). Ambas opciones tienen sus ventajas e inconvenientes, existiendo CPUs comerciales de cada tipo. La familia IA32* es *little* endian.

Master-Slave : Maestro-Esclavo. Paradigma de programación paralela en el que un proceso inicial (master) controla el arranque (spawn) y ejecución de otros procesos subsidiarios (slaves) que cooperan en una aplicación paralela. Típicamente, los procesos esclavos no se comunican entre sí, ya que sólo del maestro depende la decisión de arrancar procesos adicionales, y sólo él conoce cuántos procesos existen en un momento dado.

Dado que el maestro puede transmitir a los esclavos información sobre los restantes procesos, nada impide que éstos puedan comunicarse, volviéndose esta categoría ambigua cuando los esclavos realizan alguna operación que no sea simple comunicación con el maestro: colectivas como barreras y reducciones, o incluso paso de mensajes explícito entre esclavos.

En la práctica, el término Master-Slave se usa en contraposición a SPMD*, resaltando que al menos existen dos programas distintos, el maestro y el esclavo. Sin embargo este uso del término es incorrecto, dado que nada impide redactar un programa Master-Slave en estilo SPMD, utilizando una bifurcación condicional en donde una rama implementa el código maestro y la otra el esclavo.

MATLAB: *MATrix LABoratory*, Laboratorio Matricial. Entorno interpretado de cómputo y simulación de alto nivel. Desarrollado por *The MathWorks*, *Inc.* **http://www.mathworks.com**

M-Files: Ficheros .m. En MATLAB, código fuente escrito en el lenguaje MATLAB. El texto es ejecutado por el intérprete MATLAB.

MEX-Files: Ficheros .mex (MATLAB **EX***ecutables*). Programas C o FORTRAN que realizan llamadas a la API MATLAB. Una vez compilados, pueden ser invocados desde el intérprete MATLAB.

middleware: Otro juego de palabras; ver *hardware* y *software*. Se refiere a software que no es ni del sistema ni de aplicación, ocupando una categoría intermedia (*middle*) entre ambos. El tipo de software que se añade a un computador en red para posibilitar su utilización en un cluster de computadores es típicamente clasificado como middleware: bibliotecas de paso de mensajes, sistemas de colas con equilibrado de carga y migración de procesos, sistemas de memoria compartida virtual (distribuida)...

MIMD: *Multiple Instruction Stream, Multiple Data Stream*. Múltiples Instrucciones, Múltiples Datos. Modelo de ejecución paralela en el que cada procesador actúa independientemente.

MMX: *MultiMedia eXtensions*, ampliaciones multimedia (SIMD, SWAR) al repertorio máquina de la familia de CPUs Intel IA32 a partir del Pentium.

MPI: *Message Passing Interface*, Interfaz de Paso de Mensajes. Estándar que especifica una serie de llamadas de biblioteca útiles para desarrollar aplicaciones paralelas mediante paso de mensajes. El Foro MPI agrupa representación industrial, académica y de diversas organizaciones. **http://www.mpi-forum.org** También disponible en Netlib, junto con la implementación MPICH, **http://www.netlib.org/mpi/**

MPMD: *Multiple Program, Multiple Data*. Versión restringida de MIMD en la que, en oposición a SPMD, los distintos procesadores no ejecutan el mismo programa. Este término no pretende clasificar arquitecturas paralelas, sino el estilo de redacción de programas paralelos. Ver también *Master-Slave*.

multicast: Transmisión a múltiples destinos. *IP Multicast*: consultar RFC-1112 por ejemplo (*host extensions for IP Multicasting*).

NFS: *Network File System*, Sistema de Ficheros en Red. Estándar de facto desarrollado por Sun MicroSystems para compartir discos a través de la red.

NIC: *Network Interface Circuit*, Circuito de Interfaz con la Red. Tarjeta de red.

overhead: Sobrecarga. Pérdida de prestaciones asociada al uso de un software. Así, PVMTB permite llamar a PVM desde MATLAB, pero añade a la llamada PVM el trabajo adicional de ejecutar su propio código, existiendo por tanto un *overhead* asociado a su uso.

PBLAS: *Parallel BLAS*, Sistema de Algebra Lineal Básico Paralelo. Versión BLAS* para máquinas de memoria distribuida. Paquete de apoyo para ScaLAPACK, basado en PVM o MPI.

ping-pong: conocido juego (deporte olímpico) de paleta. Comparando un par de computadores con jugadores de este deporte, y el mensaje que se pasa entre ellos con la pelota utilizada (el tablero debería ser el cable o medio de transmisión, y las raquetas tal vez los circuitos NIC) la operación de enviar y recibir sucesivamente varios mensajes tendría el aspecto de una partida de *ping-pong*.

En el test *ping-pong* se envían y reciben varias veces una serie de mensajes de tamaño variable, cronometrando el tiempo empleado en ello. Para cada tamaño se calcula el tiempo medio de ida y vuelta (*round-trip*). Se espera así eludir la inevitable variabilidad en las mediciones de tiempo, que produciría resultados no muy sensatos si se realizaran una única vez para cada tamaño.

Se espera también que el coste del bucle **for** utilizado para la repetición e incluido necesariamente en el tramo cronometrado sea insignificante en comparación con el coste del paso del mensaje.

También se suele ofrecer como resultado la mitad del tiempo de ida y vuelta, o tiempo de transmisión (*ping*, *transmit*, *one-way time*) que incluye por tanto el tiempo de recepción en destino.

PVM: *Parallel Virtual Machine*, Máquina Paralela Virtual. Es un sistema completo de paso de mensajes, incluyendo una serie de comandos de control y monitorización, y una biblioteca de llamadas. Estándar de facto, ya que durante mucho tiempo fue el único software de amplia distribución disponible para paso de mensajes. Además soporta prácticamente cualquier arquitectura en la que concebiblemente se desee utilizar paso de mensajes, así como la interoperación entre ellas. Desarrollado en el Oak Ridge National Laboratory, USA. Disponible en Netlib, **http://www.netlib.org/pvm3/** Página Web en Oak Ridge, **http://www.epm.ornl.gov/pvm**

recubrimiento: ver *Wrapper*.

RFC: *Request For Comments*, Petición de Comentarios. Mecanismo semi-formal para establecer protocolos, prácticas comunes y comentarios sobre el desarrollo de Internet. Empezaron como una serie de notas, y han jugado un importante papel en el proceso de estandarización de Internet (IP). **http://www.rfc-editor.org/index.html**

ScaLAPACK: *Scalable LAPACK*, Paquete de Algebra Lineal Escalable. Subconjunto de rutinas LAPACK rediseñadas para computadores paralelos MIMD de Memoria Distribuida. Está escrito en estilo SPMD, usando paso de mensajes explícito (PVM o MPI) para comunicación entre procesos. Asume que las matrices se almacenan en memoria en descomposición bidimensional cíclica en bloque. Se basa en una versión de memoria distribuida de BLAS (PBLAS) y una serie de rutinas de comunicación (BLACS). Disponible en Netlib, **http://www.netlib.org/scalapack/**

script: Guión. Vocablo inicialmente usado para referirse a las *macroinstrucciones* del intérprete de comandos de un Sistema Operativo (como los *shells* de Unix). El término se aplica actualmente a cualquier fichero en texto legible conteniendo instrucciones que serán interpretadas y ejecutadas, evitando al usuario teclear la secuencia completa cada vez que desea reproducir su funcionalidad.

Por ejemplo, cuando se teclea en MATLAB un nombre de comando desconocido, el intérprete busca en disco un fichero con dicho nombre, por si fuera un *script* creado por el usuario.

SIMD: *Single Instruction Stream, Multiple Data Stream*. Una Instrucción, Múltiples Datos. Modelo de ejecución paralela en el que cada unidad de procesamiento ejecuta la misma instrucción en el mismo instante, pero sobre sus propios datos. Usualmente asociado a manipulación de arrays o vectores.

SMP: *Symmetric MultiProcessing*, Multiprocesamiento Simétrico. Aplicado a computadores con varias CPUs homólogas, en oposición a “coprocesamiento”, en el cual una CPU delega en un coprocesador la realización de una tarea especializada (coprocesador matemático, gráfico, de E/S, por ejemplo).

socket: Toma, enchufe. Descriptor de fichero para comunicación en red. Típicamente, un proceso servidor mantiene abierto un *socket* ligado (bind()) a una dirección host:puerto, en donde espera (listen ()) y atiende (accept()) peticiones. Un proceso cliente debe crear (socket()) y conectar (connect()) otro *socket* a dicha dirección, tras lo cual ambos procesos pueden comunicarse (read()/ write ()) y cerrar posteriormente la conexión (**close**()).

software: Soporte lógico de un computador. El término proviene de un juego de palabras, ver *hardware**. El software de un computador suele clasificarse según su funcionalidad: software del sistema, de administración, chequeo del equipo, detección de funcionamiento erróneo y ajuste fino (*tuning*), software de aplicación, comandos del usuario, *shells*. . .

speedup: Ganancia en velocidad. El cociente entre el tiempo de ejecución del programa secuencial óptimo y el de una versión paralela es el speedup asociado a dicha versión.

SPMD: *Single Program, Multiple Data*. Versión restringida de MIMD en la que los distintos procesadores ejecutan un mismo programa. En oposición a SIMD, cada procesador puede presentar un camino de control de flujo distinto a través del programa. Este término no pretende clasificar arquitecturas paralelas, sino el estilo de redacción de programas paralelos. Ver también *Master-Slave*.

SWAR: *SIMD Within A Register*, SIMD dentro de un registro, técnica consistente en aplicar una operación a una palabra formada por subunidades. Dentro de ciertas limitaciones, equivale a realizar la misma operación sobre las múltiples subunidades, siendo conceptualmente comparable a una instrucción SIMD.

switch: Conmutador, equipo de interconexión por conmutación de circuitos orientado a la conexión. En oposición a un hub por conmutación de paquetes, orientado a datagramas. Un switch puede establecer un circuito entre cualesquiera de sus puertos. Permite también segregar cualquier subconjunto de puertos en un dominio de colisión separado.

TCL: *Tool Command Languaje*, Lenguaje de Comandos para Herramientas. Sencillo lenguaje de guiones (*scripting*). **http://www.scriptics.com/advocacy/tclHistory.html.** Debido a su sencillez, su facilidad de E/S a cualquier dispositivo (incluyendo *sockets* IP), la disponibilidad de Tk para desarrollar interfaces gráficos, el amplio conjunto de plataformas soportadas, y el hecho de ser gratuito/libre (licencia BSD), multitud de aplicaciones funcionan sobre Tcl/Tk.

TFTP: *Trivial File Transfer Protocol*, Protocolo Trivial de Transferencia de Ficheros. Versión reducida de FTP, especialmente diseñada para facilitar el arranque de estaciones sin disco. Consultar RFC-1350, STD0033 (The TFTP Protocol Revision 2).

TK: *TCL Kit*, Conjunto de Herramientas TCL que permiten la rápida creación de interfaces gráficos (GUIs).

***Toolbox*:** Cajón de Herramientas, conjunto de comandos para MATLAB. Suelen estar agrupados por temática, así hay un *Signal Processing* Toolbox, *Image Processing* Toolbox, etc. Desempeñan, en un entorno interactivo como MATLAB, el mismo papel que las bibliotecas en un lenguaje compilado.

VLIW: *Very Long Instruction Word*. Arquitectura de procesador en la que el campo de código de operación es relativamente más largo que en otros diseños. Esto permite incluir varias operaciones en una sola palabra de instrucción. Los compiladores para este tipo de CPUs optimizan la secuencia de instrucciones de manera que en cada palabra de instrucción se incluyan el máximo número posible de instrucciones independientes que puedan ser ejecutadas en paralelo por las diversas unidades funcionales disponibles.

VSM: *Virtual Shared Memory*, Memoria Compartida Virtual. Sistema que permite usar la memoria de los computadores en un cluster de SMPs como si toda fuera una gran memoria compartida. Esto permite programar todo el cluster bajo el paradigma de memoria compartida, evitando paso de mensajes explícito. Ver DSM y referencia [38].

wrapper: Recubrimiento. Cuando invocar a una rutina directamente implica algún inconveniente, se programa una rutina de interposición, o *wrapper*, para ser invocada en su lugar. El recubrimiento realiza los arreglos necesarios previos y/o posteriores a la llamada.

XDR: *eXternal Data Representation*, Representación de Datos Externa. Tanto computadores *little** como *big endian** pueden conectarse a Internet. La red debe definir un código para ordenar los distintos bytes que forman un dato multibyte. En general, IP es *big endian* a nivel de byte, lo cual significa que las máquinas *little endian* deben invertir el orden de transmisión de los bytes de una variable entera. Si una máquina de longitud de palabra 16bits empaquetara caracteres por parejas para almacenar *strings*, debería ordenarlos de manera que el primer carácter se transmitiera en primer lugar.

Bibliografía

[1] Active Messages (**AM**) Home Page, **http://now.cs.berkeley.edu/AM/active_messages.html**

[2] Amara's *wavelet* page, **http://www.amara.com/current/wavelet.html**

[3] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", J AFIPS Proceedings of the SJCC, vol.30 pp.483–485, 1967.

[4] Andreas Uhl *wavelet* links, **http://www.cosy.sbg.ac.at/ uhl/wav.html**

[5] The **Beowulf** Project Home Page, **http://www.beowulf.org/**

[6] The Beowulf Documentation Project, **http://www.beowulf-underground.org/doc_project/index.html**
The Beowulf HOWTO, **http://www.linuxdoc.org/HOWTO/Beowulf-HOWTO.html**
http://www.beowulf.org/howto/howto.html

[7] E. K. Blum, X. Wang, P. Leung, "Architectures and message-passing algorithms for cluster computing: Design and Performance", *Parallel Computing* Vol.26, cw313–332 (2000)

[8] P. J. Burt, E. H. Adelson, "The Laplacian Pyramid as a Compact Image Code", IEEE Trans. on Communications, vol.COM-31, no.4, pp.532–540, April 1983.

[9] S. Chapin, J. Worringen, *Operating Systems*, in "Cluster Computing White Paper", (M. Baker, Ed.), pp.14–21, IEEE TFCC April 2000, **http://www.dcs.port.ac.uk/~mab/tfcc/WhitePaper**

[10] C. K. Chui: *An Introduction to Wavelets*, "Wavelet Analisys and its Applications", Vol.1, ISBN 0-12-174584-8, Academic Press, London, 1992.

[11] G. Ciaccio, "A Communication System for Efficient Parallel Processing on Clusters of Personal Computers", Ph.D. Thesis, Computer Science Department, University of New Hampshire, Durham, NH 03824 USA.

[12] COMMSIM Home Page, **http://shay.ecn.purdue.edu/~postal/commsim.html**

[13] Cornell Multitask Toolbox for MATLAB **CMTM**, **http://www.tc.cornell.edu/Services/Software/CMTM/**

[14] O. Coulaud, E. Dillon, "Para++: A High Level C++ Interface for Message Passing", *Journal of Parallel and Distributed Computing* 51, 46–62 (1998)

[15] Cray Inc., "Cray Message Passing Toolkit", **http://www.cray.com/products/software/mpt.html**

[16] Cornell Theory Center (**CTC**): "New Parallel Programming Tools for MATLAB",
http://www.tc.cornell.edu/ /news/releases/2000/cmtm.asp

[17] H. Dietz, "Parallel Processing HOWTO", Linux Documentation Project LDP 5 January 1998,
http://aggregate.org/PPLINUX/

[18] DiskLess nodes HOWTO, **http://www.linuxdoc.org/HOWTO/Diskless-HOWTO.html**

[19] **DP Toolbox** Home Page, **http://www-at.e-technik.uni-rostock.de/dp/**

[20] **Ethernet** standard, **http://standards.ieee.org/getieee802/**

[21] Ethernet Quick Reference Guides, **http://www.ots.utexas.edu/ethernet/quickref.html**

[22] B.L. Evans, S.X. Gu, (1997) "TMath 0.2: A Tcl Interface to MATLAB and Mathematica",
http://www.ece.utexas.edu/∼bevans/projects/tmath.html

[23] J. Fernández Baldomero, "Message Passing under MATLAB", Proceedings of the High Performance Computing Symposium – HPC 2001 (Adrian Tentner, Ed.), pp.73-82. 2001 Advanced Simulation Technologies Conference, Seattle, Washington, April 2001. The Society for Modeling and Simulation International ISBN 1-56555-237-7.
http://www.scs.org/confernc/astc01/prelim-program/html/hpc-pp.html

[24] J. Fernández Baldomero, "PVMTB (Parallel Virtual Machine Toolbox)", II Congreso de Usuarios MATLAB'99, (S. Dormido, Ed.), pp.523–532. UNED, Madrid, Spain, 1999.

[25] Fast Messages (**FM**) Home Page, **http://www-csag.ucsd.edu/projects/comm/fm.html**

[26] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, "PVM and MPI: a Comparison of Features", *Calculateurs Paralleles* vol.8, n.2 (1996), **http://www.epm.ornl.gov/pvm/PVMvsMPI.ps**

[27] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, V. Sunderam, "PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing", The MIT Press, Cambridge, Massachusetts, 1994,
http://www.netlib.org/pvm3/book/pvm-book.html

[28] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Mancheck, V. Sunderam, (1994) "PVM 3 Users Guide and Reference Manual", Oak Ridge National Laboratory, **http://www.netlib.org/pvm3/ug.ps**

[29] W. Gropp, E. Lusk, "Reproducible Measurements of MPI Performance Characteristics", Proceedings of the EuroPVM/MPI'99 Conference, Barcelona, Spain, September 1999.
http://www.caos.uab.es/europvm/programme.html
http://www-unix.mcs.anl.gov/∼gropp/papers.html

[30] W. Gropp, E. Lusk, A. Skjelum, (1994) "Using MPI: Portable Parallel Programming with the Message-Passing Interface", The MIT Press, 1994, **http://www.mcs.anl.gov/mpi/usingmpi/index.html**

[31] Hewlett Packard, "HP Message Passing Library (MPI)", **http://www.hp.com/rsn/mpi/mpihome.html**

[32] J. Hollingsworth, K. Liu, P. Pauca, (1996) "PT v.1.00 Manual and Reference Pages", Technical Report, Mathematics and Computer Science Department, Wake Forest University,
http://www.math.wfu.edu/pt/pt.html

[33] B. B. Hubbard: *The World according to Wavelets: the Story of a Mathematical Technique in the Making*, ISBN 1-56881-072-5, A K Peters, Ltd. Natick, MA 01760, USA, 1998.

[34] D. C. Hyde, B. Wilkinson, *Education*, in "Cluster Computing White Paper", (M. Baker, Ed.), pp.55–57, IEEE TFCC April 2000, **http://www.dcs.port.ac.uk/∼mab/tfcc/WhitePaper**

[35] IBM Corp., "MPI-F Programming Environment",
http://www.research.ibm.com/people/f/franke/MPI/MPI.html

[36] IBM MPI Threads, “Threads based MPI for the IBM RS/6000 SP”,
http://www.research.ibm.com/actc/Tools/MPI_Threads.htm

[37] IBM RS/6000 SP Books, “IBM PVMe for AIX”,
http://www.rs6000.ibm.com/resource/aix_resource/sp_books/pvme/

[38] Keleher, Peter J.: Distributed Shared Memory Home Pages,
http://www.cs.umd.edu/∼keleher/dsm.html

[39] J. Kepner, “Interfacing Interpreted and Compiled Languages to support Applications on a Massively Parallel Network of Workstations (MP-NOW)”, *Cluster Computing* Vol.3, 35–44 (2000)

[40] J.C. Lagarias, J.A. Reeds, M.H. Wright, P.E. Wright, (1998) “Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions”, SIAM Journal of Optimization, Vol.9 Number 1, pp.112–147, referenciado por *The MathWorks* en la documentación para la función fminsearch(),
http://www.mathworks.com/access/helpdesk/jhelp/techdoc/ref/fminsearch.shtml

[41] **LAM** Home Page, **http://www.lam-mpi.org/lam**

[42] LAM Home Pages, Request Progression Interface documentation,
http://www.lam-mpi.org/download/files/lam_rpi.ps.gz

[43] The rest of the Lenna Story, **http://www.lenna.org**
http://www.cs.cmu.edu/∼chuck/lennapg/lenna.shtml

[44] Página web de Stéphane Mallat, **http://www.cmap.polytechnique.fr/ mallat/**
http://cs.nyu.edu/cs/faculty/mallat/

[45] S. G. Mallat, “Multiresolution Approximations and Wavelet Orthonormal Bases of $\mathcal{L}^2\mathcal{R}$”, Trans. of the American Mathematical Society, vol.315 no.1, pp.69–87, September 1989.

[46] S. G. Mallat, “A Theory for Multiresolution Signal Decomposition: The Wavelet Representation”, IEEE Trans. on Pattern Analysis and Mach. Intell. vol.11 no.7, pp.674–693, July 1989.

[47] S. G. Mallat: *A Wavelet Tour of Signal Processing*, ISBN 0-12-466606-X, Academic Press, 1999.
http://www.apcatalog.com/cgi-bin/AP?ISBN=012466606X&LOCATION=US&FORM=FORM2

[48] M. W. Marcellin, M. J. Gormish, A. Bilgin, and M. P. Boliek, “An Overview of JPEG-2000” (invited paper), in Proc. of 2000 Data Compression Conference, pp. 523-541, Snowbird, Utah, March 2000.
http://www-spacl.ece.arizona.edu/∼mwm/
http://vail.ece.arizona.edu/Publications.html
http://vail.ece.arizona.edu/bilgin/Publications.html
http://citeseer.nj.nec.com/264144.html

[49] D. Marr: *Vision*, ISBN 0-7167-1567-8, W H Freeman & Company, 1996.

[50] MathSoft *wavelet* links, **http://www.mathsoft.com/wavelets.html**

[51] *The MathWorks*, Inc. (1998) “MATLAB Application Program Interface Guide”,
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/api/apiguide.pdf

[52] *The MathWorks*Downloads, **http://www.mathworks.com/support/ftp/miscv5.shtml**

[53] *The MathWorks*, Inc. (1999) “MATLAB Release 11 New Features”,
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/newfeat.pdf

[54] *The MathWorks*, Inc. (2001) “Motorola DSP Developer’s Kit 1”,
http://www.mathworks.com/products/motorolasp

[55] *The MathWorks*, Inc. (1996) “Signal Processing Toolbox”,
http://www.mathworks.com/access/helpdesk/help/toolbox/signal/signal.shtml

[56] *The MathWorks*, Inc. (2001) “Developer’s Kit for Texas Instruments DSP” (**TIDSP**),
http://www.mathworks.com/products/tidsp

[57] *The MathWorks*, Inc. (1996) M. Mitisi, Y. Mitisi, G. Oppenheim, J. Poggi, “Wavelet Toolbox for use with MATLAB”,
http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet/wavelet.shtml

[58] **MatPar** Home Page, **http://olympic.jpl.nasa.gov/Reports/Highlights96/matpar.html**
http://olympic.jpl.nasa.gov/Reports/Highlights97/matpar97.html

[59] V.S. Menon, A.E. Trefethen, (1997) “MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing”, Supercomputing’97 ACM SIGARCH and IEEE Computer Society,
http://www.supercomp.org/sc97/proceedings/TECH/MENON/INDEX.HTM

[60] Y. Meyer: *Wavelets: Algorithms & Applications*, ISBN 0-89871-309-9, Society for Industrial and Applied Maythematics (SIAM), Philadelphia, USA, 1999. Traducción basada en lecciones originalmente presentadas por el autor en el Instituto Español en Madrid, España, Febrero 1991.

[61] P. C. Millar, “Recursive Quadrature Mirror Filters—Criteria Spedification and Design Methods”, IEEE Trans. on Acoustics, Speech and Signal Processing, vol.ASSP-33, no.2, pp.413–419, April 1985.

[62] C. Moler, “Why there isn’t a parallel MATLAB”, MATLAB news & notes, Cleve’s Corner Spring 1995,
http://www.mathworks.com/company/newsletter/pdf/spr95leve.pdf

[63] C. Moler, “MATLAB incorporates LAPACK”, MATLAB news & notes, Cleve’s Corner Winter 2000,
http://www.mathworks.com/pany/newsletter/clevescorner/winter2000.cleve.shtml

[64] L. Moura e Silva, R. Buyya, *Parallel Programming Models and Paradigms*, in “High Performance Cluster Computing. Volume 2: Programming and Applications”, (R. Buyya, Ed.), pp.4–27, Prentice Hall PTR, NJ, 1999.

[65] **MPI** Home Page, **http://www.mcs.anl.gov/mpi**

[66] MPI Forum (1994) “MPI: A Message-Passing Interface standard”, International Journal of Supercomputer Applications and High Performance Computing, vol.8, numb.3/4,
http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html

[67] MPI Forum (1997) “MPI-2: Extensions to the Message-Passing Interface”, Technical Report,
http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html

[68] **MPITB** Home Page, **http://atc.ugr.es/javier-bin/mpitb_eng**

[69] NetBoot Home Page, **http://www.han.de/∼gero/netboot/index.html**

[70] N. Nevin, (1996) “The Performance of LAM 6.0 and MPICH 1.0.12 on a Workstation Cluster”, Ohio Supercomputer Center Technical Report OSC-TR-1996-4,
http://www.mpi.nd.edu/downloads/lam/lam-papers.tar.gz

[71] Root over **NFS** clients & server HOWTO,
http://www.linuxdoc.org/HOWTO/Diskless-root-NFS-HOWTO.html

[72] **NHSE** (National HPCC Software Exchange) Review on Cluster Management Software, **http://www.nhse.org/NHSEreview/CMS/**

[73] **OpenMP** Home Page, **http://www.openmp.org/**

[74] **ParaMat** Home Page, **http://www.alphadata.co.uk/dsheet/paramat.html**

[75] Parallel Computer Systems (**Parasys**), "HP-PVM – fast commercial PVM clone", **http://www.parasys.co.uk**

[76] S. Pawletta, W. Drewelow, P. Duenow, T. Pawletta, M. Suesse: "A MATLAB Toolbox for Distributed and Parallel Processing", **http://citeseer.nj.nec.com/pawletta95matlab.html** **http://www.mb.hs-wismar.de/Mitarbeiter/Pawletta/Forschung/Publikationen/zz95_matlab95.ps.gz**

[77] G. Pirani, V. Zingarelli, "An Analytical Formula for the Design of Quadrature Mirror Filters", IEEE Trans. on Acoustics, Speech and Signal Processing, vol.ASSP-32, no.3, pp.645–648, June 1984.

[78] Parallel Problems Server Home Page (**PPServer**), **http://www.ai.mit.edu/projects/ppserver/**

[79] PVMTB Home Page, **http://atc.ugr.es/javier-bin/pvmtb_eng**

[80] M. J. Quinn: *Parallel Computing Theory and Practice, 2nd Edition*, McGraw Hill, New York, 1994.

[81] D. Reed, D. Grunwald, "The Performance of Multicomputer Interconnections Networks", IEEE Computer vol.20 pp.63–73, June 1987.

[82] **RT-LAB** Home Page, **http://www.opal-rt.com/p_rt-lab.html**

[83] D. F. Savarese, T. Sterling: *Beowulf*, in "High Performance Cluster Computing. Volume 1: Architectures and Systems", (R. Buyya, Ed.), pp.625–645, Prentice Hall PTR, NJ, 1999.

[84] **SPIE** International Society for Optical Engineering, tutorial web, **http://www.spie.org/web/meetings/programs/pe99/courses/sc13.html**

[85] IEEE TFCC Technical Area on Single System Image (**SSI**), **http://www.ac.upc.es/homes/toni/CCTaskForce/SSI.html**

[86] W. Stallings: *Organización y Arquitectura de Computadores, 5ª Edición*, Prentice Hall, Madrid, 2000.

[87] Sun Microsystems, "Floating Point Programmer's Guide", Part Number: 800-1552-10, September 1986.

[88] B. W. Suter: *Multirate and Wavelet Signal Processing*, "Wavelet Analisys and its Applications", Vol.8, ISBN 0-12-677560-5, Academic Press, London, 1997.

[89] LAM Web pages, **Top 10 reasons** to prefer MPI over PVM, **http://www.mpi.nd.edu/lam/mpi/mpi_top10.php3**

[90] A.E. Trefethen, V.S. Menon, C. Chang, G.J. Czajkowski, C. Myers, L.N. Trefethen, (1996) "MultiMATLAB: MATLAB on Multiple Processors", Technical Report 96-239, Cornell Theory Center, University of Cornell, **http://www.cs.cornell.edu/Info/People/lnt/multimatlab.html**

[91] M. Unser, "Splines: a Perfect Fit for Signal and Image Processing", IEEE Signal Processing Magazine, November 1999, pp.22–38.

[92] **VIA** Home Page, **http://www.viarch.org/default.htm**

[93] B. A. Wandell: *Foundations of Vision*, ISBN 0-87893-853-2, Sinauer Associates, Inc. Sunderland, MA 01375-0407, USA, 1995.

[94] The Wavelet Digest, **http://www.wavelet.org/cm/ms/what/wavelet/index.html**

[95] B. Wilkinson: *Parallel programming: techniques and applications using networked workstations and parallel computers*, Prentice Hall, 1998.

[96] J. W. Woods, S. D. O'Neil, "Subband Coding of Images", IEEE Trans. on Acoustics, Speech and Signal Processing, vol.ASSP-34, no.5, pp.1278–1288, October 1986.

[97] J. You, P. Bhattacharya, "A Wavelet-Based Coarse-to-Fine Image Matching Scheme in a Parallel Virtual Machine Environment", IEEE Trans. on Image Processing, vol.9, no.9, pp.1547–1559, September 2000.