Universitat Autònoma de Barcelona

Departament d'Arquitectura de Computadors i Sistemes Operatius

Màster en Computació d'altes prestacions

FACTORES DE RENDIMIENTO EN APLICACIONES HÍBRIDAS

Memoria del trabajo de investigación del "Máster en Computació d'altes prestacions", realizada por Abel Castellanos Carrazana, bajo la dirección de Dr. Tomás Margalef. Presentada en la Escuela de Ingeniería (Departamento de Arquitectura de Computadores y Sistemas Operativos)

Bellaterra, Julio 2011

Iniciación a la investigación. Trabajo de fin de máster

Máster en Computación de Altas Prestaciones.

Factores de rendimiento en aplicaciones híbridas

Realizada por Abel Castellanos Carrazana en la Escuela de Ingeniería, en el Departamento de Arquitectura de Computadores y Sistemas Operativos

Dirigida por: Dr. Tomás Margalef

Firmado

Director Estudiante

A mis esposa y mi hijo, a mi familia en general

por el apoyo constante aun en la distancia.

A Tomás, Eduardo, Ania, Joan y Josep

Por su ayuda constante, críticas e ideas

A mis profesores del Master

por todo el conocimiento que me han transmitido

A Carlos, Cesar, Sandra, Hugo, Javier y Claudio

por convertirse en mis amigos,

A los miembros de CAOS y en especial a Ronal

por toda la ayuda que he recibido de ustedes...

¡Muchísimas Gracias!

Resumen

En el entorno actual, diversas ramas de las ciencias, tienen la necesidad de auxiliarse de la computación de altas prestaciones para la obtención de resultados a relativamente corto plazo. Ello es debido fundamentalmente, al alto volumen de información que necesita ser procesada y también al costo computacional que demandan dichos cálculos. El beneficio al realizar este procesamiento de manera distribuida y paralela, logra acortar los tiempos de espera en la obtención de los resultados y de esta forma posibilita una toma decisiones con mayor anticipación.

Para soportar ello, existen fundamentalmente dos modelos de programación ampliamente extendidos: el modelo de paso de mensajes a través de librerías basadas en el estándar *MPI*, y el de memoria compartida con la utilización de *OpenMP*. Las aplicaciones híbridas son aquellas que combinan ambos modelos con el fin de aprovechar en cada caso, las potencialidades específicas del paralelismo en cada uno. Lamentablemente, la práctica ha demostrado que la utilización de esta combinación de modelos, no garantiza necesariamente una mejoría en el comportamiento de las aplicaciones. Por lo tanto, un análisis de los factores que influyen en el rendimiento de las mismas, nos beneficiaría a la hora de implementarlas pero también, sería un primer paso con el fin de llegar a predecir su comportamiento. Adicionalmente, supondría una vía para determinar que parámetros de la aplicación modificar con el fin de mejorar su rendimiento.

En el trabajo actual nos proponemos definir una metodología para la identificación de factores de rendimiento en aplicaciones híbridas y en congruencia, la identificación de algunos factores que influyen en el rendimiento de las mismas.

Palabras clave: modelos de programación, aplicaciones híbridas, factores de rendimiento, computación de altas prestaciones, MPI, OpenMP.

Resum

En l'entorn actual, diverses branques de les ciències, tenen la necessitat de recolzar-se en la computació d'altes prestacions per a l'obtenció de resultats en un relatiu curt temps. Això és degut bàsicament, a l'alt volum d'informació que necessita ser processada i també al cost computacional que demanen aquests càlculs. El benefici al realitzar aquests processaments de forma distribuïda i paral·lela, és que s'aconsegueix escurçar els temps d'espera en l'obtenció de resultats i d'aquest forma possibilita una presa de decisions amb major anticipació.

Per aconseguir això, existeixen fundamentalment dos models de programació àmpliament estesos: el model de pas de missatges mitjançant llibreries basades en l'estàndar MPI, i el model de memòria compartida amb la utilització de OpenMP. Les aplicacions híbrides són aquelles que combinen d'ambdós models amb la finalitat d'aprofitar en cada cas, les potencialitats específiques de paral·lelisme. Lamentablement, la pràctica ha demostrat que la utilització d'aquesta combinació de models, no garantitza necessàriament un millor comportament de les aplicacions. Per tant, un anàlisi dels factors que influeixen en el rendiment, pot beneficiar a l'hora d'implementarles, però també, pot ser un primer pas per aconseguir predir el comportament. Adicionalment, pot suposar una via per a determinar els paràmetres de l'aplicació a modificar amb la finalitat de millor el rendiment.

En el treball actual es proposa definir una metodologia per a la identificació de factors de rendiment en aplicacions híbrides, i en congruència, la identificació de factors que influeixen en el rendiment.

Paraules clau: models de programació, aplicacions híbrides, factors de rendiment, computació d'altes prestacions, MPI, OpenMP.

Abstract

In the current environment, various branches of science are in need of auxiliary high-performance computing to obtain relatively short-term results. This is due mainly to the high volume of information that needs to be processed and the computational cost demanded by these calculations. The benefit to perform this processing using distributed and parallel programming mechanism achieves shorter waiting times in obtaining the results and thus allows making decisions sooner.

To support this, there are basically two widely spread programming models: the model of message passing, through based on the standard libraries *MPI*, and shared memory model with the use of *OpenMP*. Hybrid applications are those that combine both models in order to take in each case, the specific potential of parallelism of each one. Unfortunately, experience has shown that using this combination of models, does not necessarily guarantee an improvement in the behavior of applications. Therefore, an analysis of the factors that influence the performance of hybrid applications will help us to improve his performance base on modifying the original code. Besides, it will be the first step in the long way to predict their behavior. Additionally, it would be a way to determine which parameters of the application have to be modified to improve the performance.

In the current work, we propose a methodology to identify performance factors in hybrid applications and in consequence, the identification of factors that influence the performance of them.

Keywords: programming models, hybrid applications, performance factors, high-performance computing, MPI, OpenMP.

Indice

Índice de figuras

Índice de tablas

Índice de ecuaciones

Capítulo 1

Introducción

1.1 Descripción general

Dentro de la comunidad científica que investiga en torno al tema de *HPC*, una parte importante de los esfuerzos están dirigidos al modelado y evaluación del rendimiento de aplicaciones para las diferentes arquitecturas existentes en la actualidad. Con el advenimiento, en los últimos años de los procesadores *many-cores,* se ha acuñado el término aplicaciones híbridas, a aquellas aplicaciones que utilizan un modelo de programación de paso de mensajes, usando librerías como open *MPI* o *MPICH*; y el modelo de memoria compartida, con librerías como *OpenMP* o *pthread*.

Un estudio del comportamiento y utilización de arquitecturas *many-cores* en la lista de los supercomputadores más potentes [**1**] nos muestra la popularidad que han alcanzado las arquitecturas que incluyen varios *cores* en el núcleo de cómputo (Figura 1.1). Más del 80% de los supercomputadores registrados en la lista cumplen con esta características en su arquitectura y la cantidad de ellos con un solo procesador es prácticamente despreciable.

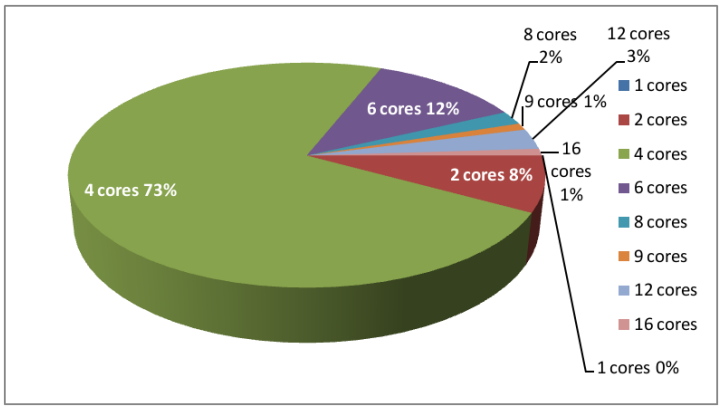


Figura 1.1 Computadores en la lista Top500 según la cantidad de *cores* en su núcleo

Se conoce por *Massively Parallel Processor* a aquellos supercomputadores en los que, en realidad, se tiene un solo computador. El mismo posee cientos o miles de *CPUs* fuertemente interconectadas a través de una red *on-chip* y la memoria suele ser compartida entre todas las CPUs. En contraposición, un *cluster* es una agrupación de computadores conectados por una red de interconexión (ethernet por ejemplo) donde cada uno tiene su propio sistema de memoria y en cada uno, se ejecuta el sistema operativo.

Enfocados en esta clasificación, observamos que los *Massively Parallel Processor* solo representan aproximadamente un 17% de los supercomputadores registrados (Tabla 1.1), mientras que los *cluster* son más del 82%. En general, hay una tendencia marcada al uso de *cluster* de computadores donde sus núcleos poseen multiples *cores* de cómputo.

Esta popularidad no es casual, se debe fundamentalmente a los beneficios que ofrecen los *cluster* como arquitectura a la hora de realizar mantenimiento, mejoras, la menor dificultad en el proceso de recuperación de fallos de hardware, y los costos inherentes a dichas mejoras.

Architecture	Count	Share %	Rmax Sum (GF)	Rpeak Sum (GF)	Processor Sum
Constellations	2	0.40 %	94970	112947	17648
MPP	84	16.80 %	17936809	23875912	2851827
Cluster	414	82.80 %	25641314	40666452	3602852
Totals	**500**	**100%**	**43673092.54**	**64655310.70**	**6472327**

Tabla 1.1 Arquitecturas de computadores más utilizadas

Por tanto, es lógico imaginar que las aplicaciones híbridas, se ajustan perfectamente a las arquitecturas en donde tenemos un *cluster* de *many-cores* ya que ofrecen los dos niveles de paralelismo: el primero entre los diferentes nodos de nuestro *cluster*, y el segundo para la utilización dentro del nodo, de todos o un subconjunto de sus *cores*.

Una evaluación de rendimiento no es más, que el proceso de análisis del comportamiento de una aplicación variando el entorno en la cual es ejecutada y modificando las cargas de trabajo a la que es sometida la misma. A partir de este proceso, se debe llegar a conclusiones en relación a cuáles son los factores fundamentales que determinan este comportamiento y en qué medida influyen en el resultado final de la aplicación.

Los modelos de rendimiento son modelos matemáticos que permiten explicar el comportamiento, predecir circunstancias futuras o abstraer un sistema. Como beneficios fundamentales, aportan la posibilidad de predicción del comportamiento de un sistema en

cuanto al tiempo de ejecución de la aplicación. Lógicamente, la variedad de arquitecturas existentes hace sumamente compleja la tarea de obtener un modelo genérico por lo que, los avances en esta línea de investigación, tratan de circunscribir el problema para un entorno específico como por ejemplo un paradigma de programación paralela sobre un tipo de arquitectura determinada.

Los beneficios que se obtiene de una predicción adecuada del rendimiento de una aplicación ante una carga de trabajo y una cantidad de recursos de cómputo, permitiría realizar una redistribución de estos recursos con el objetivo de obtener mejoras en los tiempos de ejecución y utilización de los mismos de manera más eficiente. Adicionalmente, permitiría estimar una cota de recursos necesarios para que la relación entre el tiempo de ejecución y la cantidad de recursos utilizados alcance una eficiencia cercana al máximo posible. Lamentablemente,

El desafío de obtener un buen rendimiento para las aplicaciones paralelas, constituye una campo de de estudio de los expertos en esta materia. Podemos agregar a esto, el reto que conlleva implementar una aplicación paralela que aproveche lo más eficientemente posible, las características del hardware en el que se pretende explotar la misma.

Con este fin, son utilizadas las herramientas de monitorización análisis y visualización de rendimiento, que posibilitan la toma de decisiones a partir de la información recogida en el período de ejecución. Las misma, ayudan a identificar las secciones con un comportamiento crítico y facilitan la toma de decisiones a la hora de realizar mejoras en nuestras aplicaciones.

1.2 Objetivos

Existen dos modelos fundamentales de programación para el desarrollo de aplicaciones paralelas en el entorno de la Computación de Altas Prestaciones. El primero, es el estándar MPI, que permite realizar aplicaciones de paso de mensajes. El segundo, provee instrumentos para desarrollar aplicaciones en entornos de memoria compartida usando *OpenMP*. Las aplicaciones híbridas combinan estos dos modelos con el fin de aprovechar las bondades de cada uno: una paralelización más gruesa al nivel de paso de mensajes, y una más fina en las secciones paralelas *OpenMP*. Cuán beneficioso o no es la utilización de esta combinación de modelos y que ventajas nos aporta en relación con la programación clásica usando solo MPI, son temas de estudio actuales por la comunidad científica. A su vez, la obtención de modelos de rendimiento que nos permitan predecir el comportamiento de estas aplicaciones y las maneras en que podemos influir en ellas, de manera estática o dinámica con el fin de mejorar

su rendimiento, son líneas abiertas de investigación en las que, las aportaciones, son aun incipientes.

Motivados por esta situación, nuestros objetivos inmediatos son:

- Definir una metodología de trabajo que nos permita encontrar un modelo de rendimiento para aplicaciones híbridas.
- Realizar un estudio de las herramientas de monitorización y análisis utilizadas por la comunidad científica y seleccionar de ellas, las más adecuadas a las necesidades de nuestra investigación.
- Identificar y caracterizar factores que influyen en el rendimiento de las aplicaciones híbridas comparadas con sus equivalentes MPI.

Como objetivos a largo plazo, nos hemos planteado:

- Definir un modelo analítico para predecir el rendimiento de las aplicaciones híbridas.
- Definir modelos para la mejora del rendimiento de manera dinámica

Para nuestro estudio, Nos hemos auxiliado de tres aplicaciones: el *benchmark* de NAS en su versión 3.3.1 que provee versiones híbridas de algoritmos matemáticos ya desarrollados con anterioridad, un *bechmark* que implementa el algoritmo de Jacobi en tres versiones (Híbrido, MPI, Serie) y el *benchmark* SMG2000 elaborada para permitir su ejecución usando solo MPI o combinando MPI y *OpenMP*. En el primer caso, fue necesario modificar la aplicación para obtener una versión pura MPI y así poder realizar las comparaciones de su comportamiento. En capítulos sucesivos se explicará detalladamente las características de cada una de estas aplicaciones.

1.3 Organización del trabajo

El contenido de este trabajo de investigación se encuentra estructurado en los capítulos siguientes:

- **Capítulo 2: Clasificación de las aplicaciones híbridas**

  En este capítulo se describe los dos tipos de clasificaciones de las aplicaciones paralelas recogidas en la literatura. Para ello, se identifican las particularidades de cada clasificación ubicando en cada caso, el lugar que ocupan dentro de la clasificación general. A su vez, se explica detalladamente las diferencias que existen en la forma en la que las aplicaciones

híbridas son ejecutadas en un supercomputador en contraposición con aplicaciones puras de paso de mensajes.

- **Capítulo 3: Análisis y predicción de rendimiento**

El capítulo comienza con una explicación de las diferentes estrategias seguidas para la tarea de predecir el rendimiento de las aplicaciones paralelas. Seguidamente, se describen los principales problemas que debemos considerar a la hora conseguir un modelo de rendimiento para las aplicaciones híbridas. Por último, se expone el estado actual de las investigaciones sobre temas relacionados con la predicción y análisis de rendimiento para estás aplicaciones.

- **Capítulo 4: Metodología y estudio de las herramientas**

Como su nombre lo indica, en este capítulo se propone una metodología para llegar a obtención de modelos de rendimiento para las aplicaciones híbridas. A partir de las necesidades propuestas en cada fase, se realiza un estudio detallado de las herramientas de análisis utilizadas por la comunidad con el fin de detectar las más convenientes para nuestras necesidades. También se evalúan diferentes *benchmarks* para la caracterización de las comunicaciones y se identifica en qué contexto, los resultados de los mismos pueden ser válidos para nuestro trabajo.

- **Capítulo 5: Análisis de factores de rendimiento**

En este capítulo está dividido en dos partes fundamentales. En la primera se realizan algunos análisis a partir de un modelo general de predicción de rendimiento para una fase de las aplicaciones híbridas con lo que, se llega a algunas conclusiones que nos ayudarán en nuestros trabajos futuros. En una segunda parte, se describen las aplicaciones utilizadas para la identificación los factores de rendimiento teniendo en cuenta su funcionamiento, el patrón de comunicación, la calidad de paralelismo *OpenMP* entro otros, y se describe en entorno de experimentación así como las configuraciones de ejecución de los diferentes experimentos. A partir de todo ello, se identifican tres factores con potencial influencia en el rendimiento de las aplicaciones híbridas.

- **Capítulo 6: Conclusiones y trabajos futuros**

Resumen el trabajo de investigación realizado, extrayendo las conclusiones finales derivadas de los estudios realizados. Además se proponen las líneas futuras en las cuales se pretende seguir trabajando.

Capítulo 2

Clasificación de las aplicaciones híbridas

2.1 Introducción

Dentro de las aplicaciones paralelas, existen varias clasificaciones teniendo en cuenta aspectos de las mismas que la caractericen. En particular, las aplicaciones híbridas se identifican del resto, porque combinan los dos modelos de programación paralela: el modelo de paso de mensaje, para implementar el paralelismo al nivel de los nodos del supercomputador utilizado, y el modelo de memoria compartida para un nivel de paralelismo más fino a nivel de regiones de computo paralelo dentro de la aplicación que utilicen todos o un subconjunto de los *cores* de dicho nodo.

Una plataforma híbrida es aquella en la cual la arquitectura de supercomputador está conformada por un *cluster* de computadoras conectadas a través de una red de interconexión entre los nodos y donde cada nodo, a su vez, posee varios *cores* de cómputo que comparten, dependiendo de las particularidades de la arquitectura, acceso a los diferentes niveles de memoria.

Basados en la taxonomía de los modelos de programación para plataformas híbridas [**2**] hemos realizado una adición para que la misma defina, de manera más detallada, las diferentes variantes de aplicaciones que podemos encontrarnos y sus particularidades.

En la figura 2.1, se observa las diferentes clasificaciones de modelos de programación para plataformas híbridas y como están relacionadas conceptualmente dentro de los modelos tradicionales de programación paralela. Por un lado, tenemos las aplicaciones MPI puras, donde el paralelismo esta expresado solo entre nodos de nuestro computador y se realizan pasos de mensajes entre los procesos para intercambiar información o realizar tareas de sincronización en una fase determinada de la ejecución. Por otro lado, tenemos las aplicaciones *OpenMP* puras, en donde el nivel de paralelismo se realiza dentro de cada nodo de cómputo. En este caso, no se realiza paso de mensaje entre los *threads* involucrados ya que los mismos comparten una memoria común. Las

creación, destrucción y sincronización de estos *threads* es realizada por la librería en el inicio, fin de las secciones paralelas o son especificadas explícitamente por el programador.

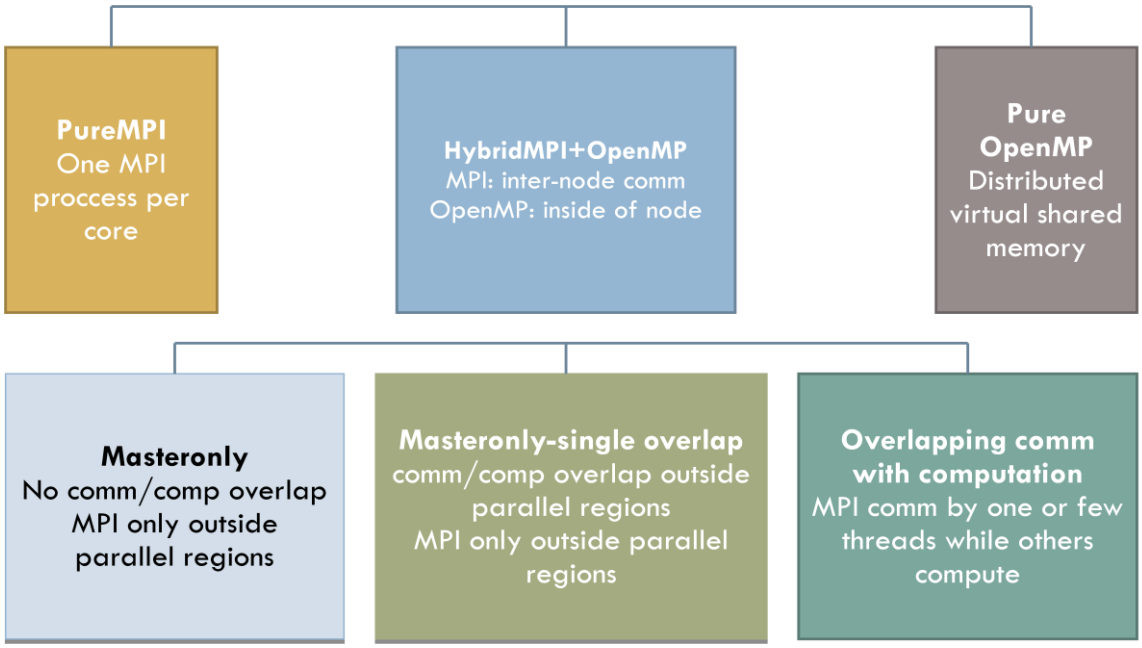


Figura 2.1 Taxonomía de los modelos de programación para aplicaciones híbridas

El modelo híbrido combina ambas alternativas de forma bastante natural de manera que, tendríamos un paralelismo más grueso implementado a nivel de paso de mensajes entre los nodos de nuestro *cluster*, y uno más fino dentro de cada nodo, definido por las regiones *OpenMP*.

Una clasificación detallada de los modelos híbridos, pasa por considerar las ubicación de las comunicaciones dentro de la lógica de ejecución de nuestro programa y si estas se solapan con las fases de cómputo de la aplicación. De esta forma, tendríamos las aplicaciones híbridas modelo *Masteronly*, las *Masteronly-single overlap* y las aplicaciones con solapamiento cómputo comunicación en los *threads* utilizados. Teniendo como premisa estos criterios, a continuación, pasaremos a explicar estas clasificaciones y las diferencias que distinguen a cada una en particular.

2.2 Aplicaciones híbridas Masteronly

La figura 2.2 nos muestra la estructura fundamental de una aplicación híbrida que cumple con la clasificación Masteronly. Como características fundamentales, la aplicación no realiza ningún tipo de comunicación MPI dentro de las secciones paralelas definidas por los pragma *OpenMP*. Por otro lado, como las comunicaciones son bloqueantes, se garantiza que no hay solapamiento entre el cómputo y las comunicaciones.

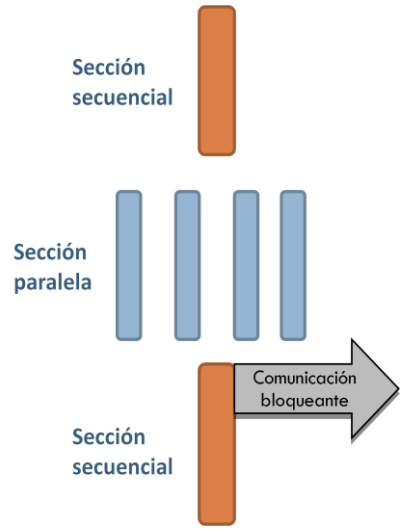


Figura 2.2 Estructura de una aplicación híbrida Masteronly

2.3 Modelo Masteronly-single overlap

En este caso, la diferencia fundamental entre las aplicaciones *Masteronly Single-overlap* y las explicadas anteriormente radica en la posibilidad de existencia de un solapamiento computo-comunicación debido a la utilización de comunicaciones no bloqueantes fuera del paralelismo al nivel de *threads*. Como indica la figura 2.3, la comunicación se sigue manteniendo fuera de la sección paralela pero, al ser una comunicación no bloqueante, se produce un solapamiento entre las comunicaciones y la próxima región de cómputo paralelo.

En este caso el solapamiento a nivel de nodo entre el cómputo y la comunicación puede ser por la existencia de una sección paralela inmediatamente después de la comunicación no bloqueante o por una sección secuencial.

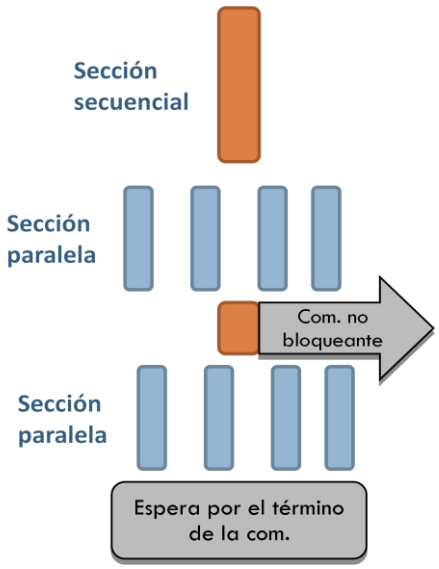


Figura 2.3 estructura de una aplicación híbrida Masteronly Single-overlap

2.4 Aplicaciones con solapamiento múltiple dentro de las regiones paralelas

La última de las categorías describe a las aplicaciones donde la comunicación se realiza en un subconjunto de los *therads* de las regiones paralelas a nivel de OpenMP. En este caso, el solapamiento computo comunicación es total. Esto quiere decir que, las comunicaciones al realizarse dentro de la sección paralela, esta comunicación se solapará con el resto de los *threads* que realicen labores de cómputo.

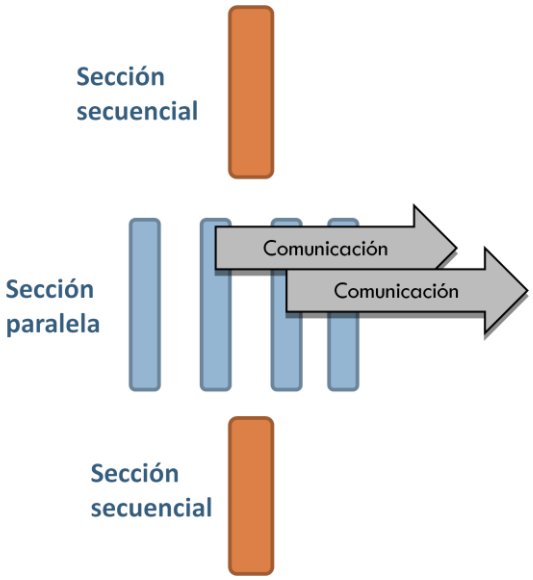


Figura 2.4 Estructura de una aplicación híbrida con solapamiento múltiple computo-comunicación

Lamentablemente, es poco común encontrar aplicaciones cuyo comportamiento sigan esta clasificación. No es casual, se debe fundamentalmente, a que la librería de comunicación MPI no está desarrollada para permitir la comunicación *thread-thread* por lo que, el emisor y el receptor de los mensajes no solo conocen el proceso respectivo que envía o recibe el mensaje, pero no en que *thread* en particular se realizó dicho envió o recepción. Este detalle impone una restricción muy fuerte a las aplicaciones ya que su lógica no puede verse afectada por esta limitante y, en la práctica, no suelen encontrarse aplicaciones con estas características. Lo usual, es que las aplicaciones con comunicación dentro de la región paralela reserven un *thread* solo para las labores de comunicación. Actualmente, se trabaja en una versión de la librería MPI (futura versión 3) que involucre entre los parámetros de la comunicación, el *thread* fuente y destino [**3**]. Con ello, se podría proporcionar una librería de paso de mensajes que permita, de manera coherente, desarrollar aplicaciones con múltiple solapamiento cómputo comunicación. En la práctica ello supondría, la posibilidad de desarrollar aplicación que se beneficien mucho más de esta posibilidad.

2.5 Clasificación basada en el paralelismo de la comunicación

A partir del estándar *MPI-2* [**4**], la librearía de programación está desarrollada para permitir el uso de comunicaciones en entornos donde se utilicen múltiples hilos de ejecución. A partir de esta adición, las aplicaciones pueden ser clasificadas dependiendo de la forma en que se gestionan el paralelismo de las comunicaciones. En la tabla 2.1 se muestran las cuatro variantes ofrecidas por la librearía para dicha gestión.

En este caso, la clasificación no depende de la forma en que esta implementada la aplicación, sino de la manera en que está siendo utilizada la librería de paso de mensajes. Sin embargo, las aplicaciones pueden verse afectadas en su funcionamiento, ante un cambio en el modo de paralelismo de la comunicación si no están programadas para soportar dicho cambio.

Nivel de paralelismo	Descripción
MPI_THREAD_SINGLE	Solo existirá un *thread* de ejecución en la aplicación por lo que las comunicaciones serán realizadas por este único hilo.
MPI_THREAD_FUNNELED	Pueden existir varios *threads* dentro de la aplicación que realicen llamadas a funciones de comunicación MPI solo que el *thread* principal será el único que en realidad realice la llamada a las funciones de comunicación, el resto será obviado.
MPI_THREAD_SERIALIZED	Pueden existir varios *threads* dentro de la aplicación que realicen llamadas a funciones de comunicación MPI pero las llamadas a funciones de comunicación en el librería serán ejecutadas una detrás de la otra de manera secuencial en el orden en que fueron invocadas
MPI_THREAD_MULTIPLE	Las llamadas a funciones de comunicación realizadas por cada hilo, se ejecutarán paralelamente por la librería y es total responsabilidad del programador, tener en cuenta el carácter no determinista del orden en el que pueden ser invocadas esas funciones.

Tabla 2.1 Tipos de paralelismo de comunicaciones definidos por MPI-2.

2.6 Configuración de ejecución de las aplicaciones híbridas

Adentrándonos en las diferentes configuraciones en que las aplicaciones híbridas pueden ser ejecutadas en un supercomputador, y comparándolas con las aplicaciones puras de paso de mensajes, podemos encontrar ciertas diferencias.

La figura 2.5 muestra como se ejecutaría una aplicación pura MPI en un *cluster* donde cada nodo contenga cuatros *cores* de cómputo. Se puede observar en este caso, que se utiliza solo un *core* del nodo en donde es ejecutado cada proceso MPI. En este caso las comunicaciones entre procesos se realizarán a través de la red de interconexión existente entre los diferentes nodos del supercomputador.

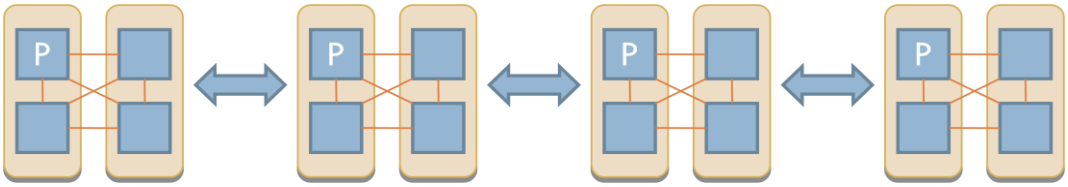


Figura 2.5 Ejecución de una aplicación MPI pura (1 proceso por nodo)

El segundo ejemplo nos muestra el comportamiento de la aplicación MPI pura si se utilizase todos los *cores* de nodo de cómputo para la ejecución de cada uno de los procesos MPI. Aquí las comunicaciones entre los diferentes procesos se realizarían a través de dos medios: el primero utilizado la memoria compartida entre procesos de un mismo nodo y, la segunda variante, a través de la red de interconexión entre los nodos si los procesos involucrados en la comunicación no se ejecutarán dentro del mismo nodo.

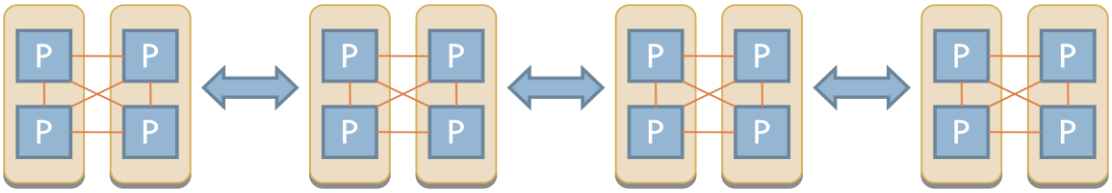


Figura 2.6 Ejecución de una aplicación MPI pura (4 procesos por nodo)

Una típica ejecución de una aplicación híbrida donde se utilizaran todos los recursos de cómputo de nodo es la mostrada en la figura 2.7. En este caso, cada proceso MPI se ejecutaría en cada nodo utilizado y las regiones de cómputo paralelo definidas por los pragmas OpenMP, crearían cuatro hilos que se ejecutarían en cada *core* del nodo de cómputo. De esta forma, tendríamos dos niveles de paralelismo muy bien definidos, uno entre los procesos ejecutados en cada nodo y el segundo, entre los *threads* ejecutados en cada *core.* La comunicación solo se manifestaría entre los diferentes procesos *MPI* y como ellos ocupan un nodo integro, las mismas se realizarían a través de la red de interconexión entre los nodos.

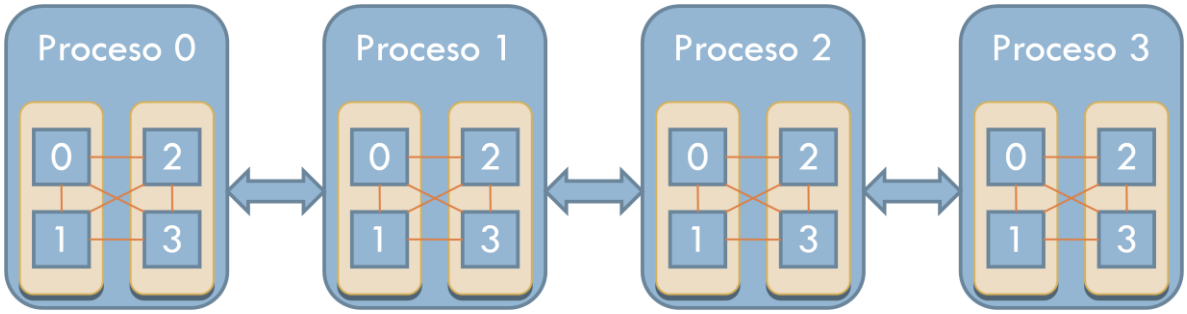


Figura 2.7 Ejecución de una aplicación híbrida (1 proceso por nodo, 1 thread por core)

La manera en que es ejecutada la aplicación para una arquitectura de hardware específica, se conoce como *mapping*. Dependiendo de las características del hardware utilizado, podemos tener diferentes maneras en que puede ser ejecutada una aplicación. Por ejemplo, para la configuración de nodo presentada en la figura anterior en donde la memoria cache de segundo nivel es compartida por un par de *cores* dentro del nodo, podríamos tener diferentes variantes ejecuciones de nuestra aplicación. Un primer ejemplo puede ser una ejecución en la cual se utilicen dos procesos MPI por nodo y dos *cores* de dicho nodo para los *threads* creados en las secciones *OpenMP*. En este caso, podríamos utilizar *cores* que compartan la misma memoria cache de segundo nivel. En caso contrario, si los procesos MPI son creados utilizando *cores* que compartan la misma memoria cache de nivel dos, los *threads* adicionales creados en las regiones de cómputo paralelo utilizaría entre una cache de segundo nivel común pero, cada uno no compartiría dicha cache con el *thread* principal del proceso MPI. Estas variantes de ejecución en la práctica, pueden devolvernos cambios significativos en el rendimiento de las aplicaciones, mas aun si la lógica de las secciones paralelas esta optimizada para hacer un uso lo más eficiente posible de las características del hardware en donde se planea su ejecución.

Capítulo 3

Análisis y predicción de rendimiento

3.1 Introducción

En general, muchos de los esfuerzos fundamentales por parte de la comunidad científica están orientados al análisis del comportamiento de las aplicaciones para un entorno en particular seleccionado. Ello se debe fundamentalmente, a la complejidad que conlleva el llegar a conseguir un modelo general de predicción de rendimiento para las aplicaciones híbridas. Este proceso de análisis suele ser incremental, o sea a partir de análisis sucesivos se va llegando a identificar claramente, para cada aplicación, cuales son los factores determinantes en su rendimiento. En concordancia, se realizan modificaciones con el fin de mejorar dicho comportamiento. Adicionalmente, hay propuesta de modelos evaluativos del comportamiento de secciones del programa como el cómputo o las comunicaciones.

3.2 Estrategias de predicción de rendimiento

Existen varias estrategias para la predicción del rendimiento de las aplicaciones paralelas. Cada una en particular, posee beneficios e inconvenientes que deben ser tomados en cuenta a la hora de decantarnos por una vía u otra. De manera general, la literatura [**5**] considera estos tres grandes grupos:

- **Modelo analítico:** es el mecanismo mediante el cual, a partir de un modelo analítico, llegamos a predecir el comportamiento de una aplicación en cuanto a su rendimiento. Como ventajas fundamentales podemos mencionar que el costo computacional de dicha predicción es bajo debido a que el análisis en fundamentalmente estático. Sin embargo, el hecho de no utilizar información sobre el comportamiento del flujo del programa produce que su resultado sea inexacto y que el margen de error de la predicción sea significativo. A la vez, esta estrategia no utiliza información en tiempo de ejecución de la aplicación, por lo que un cambio significativo en el comportamiento

de la aplicación en un momento determinado de su ejecución no sería considerado. Otra desventaja característica de esta estrategia es lo costoso que resulta llegar a un modelo analítico que tenga en cuenta los suficientes factores como para que su resultado se considerado valioso.

- **Simulación:** La estrategia de simulación**,** a diferencia de del resto, permite la obtención de manera automática de indicadores de rendimiento de la aplicación bajo diferentes condiciones. Posee la bondad adicional, de permitir obtener los resultados para diferentes arquitecturas sin la necesidad de realizar las pruebas directamente sobre ellas e incluso, permite obtener evaluar los resultados en arquitecturas teóricas que aun no han sido desarrolladas. De esta forma, podríamos llegar diseñar una arquitectura en la cual nuestra aplicación obtendría los mejores resultados de rendimiento, sin la necesidad de producir la misma. Su única desventaja, no por ello deja de ser altamente significativa, es la necesidad de empelar un tiempo, usualmente excesivo, para la obtención de resultados.
- **Medición/instrumentación dinámica:** Esta estrategia brinda la posibilidad de que nuestra predicción se adapte al comportamiento dinámico de la aplicación, de esta manera, las predicciones se realizarían para cada fase de la aplicación y el objetivo en este caso sería la modificación de los parámetros del sistema con el fin de mejorar el resultado final de ejecución. Las principales desventajas que agrega radican en que el *overhead* agregado de instrumentación puede ser significativamente alto y perturbar el comportamiento global. La información generada de los eventos monitoreados puede ser enorme y el costo asociado al análisis de la misma en concordancia, podría ser muy alto por lo que, ante un comportamiento así, las conclusiones de la predicción para cada fase de la aplicación llegarían retardadas por lo que no serían de mucha utilidad y en el peor de los casos, la aplicación estaría en un estado de espera por dicha predicción lo que repercutiría en el rendimiento global. Esta estrategia suele ser beneficiosa cuando el costo de la medición dinámica y el análisis de los resultados es mucho menor que la ganancia obtenida al ajustar la aplicación para mejorar su comportamiento.

3.3 Factores que afectan la predicción del rendimiento

Se han identificado los principales factores con mayor influencia la exactitud de un proceso de predicción del comportamiento de las aplicaciones paralelas. Los mismos, son resumidos a continuación:

- **Diversidad de arquitecturas de hardware:** dependiendo del fabricante e incluso de la versión de cada producto, podemos encontrar procesadores con arquitecturas muy diferentes. Las características de rendimiento de cada procesador son variables dependiendo del tipo de operación realizada, la cantidad de *cores* en cada núcleo varía, la estructura y latencias a los diferentes niveles de memoria también son diferentes, la red de interconexión *on-chip* entre los núcleo puede variar, etc.
- **Variedad de modelos de programación:** existen diferentes plataformas para la programación paralela. Adicionalmente, muchos fabricantes realizan versiones nuevas de las librerías y los compiladores con el fin de utilizar, de manera más eficiente, las bondades de su arquitectura. Por este motivo, hay una complejidad adicional a la hora de predecir el comportamiento de las aplicaciones ya que es necesario conocer las particularidades de cada implementación de las librerías utilizadas.
- **Solapamiento de cómputo y comunicación entre procesos MPI:** entre los procesos MPI podemos encontrar, debido al natural desbalance entre ellos, este solapamiento. Al tratar de realizar una estimación del rendimiento, este solapamiento debe ser estimado lo que conlleva a predecir este nivel de desbalance.
- **Solapamiento de cómputo y comunicación dentro del proceso MPI:** dependiendo del modelo de paralelismo utilizado en las comunicaciones y del modelo que siga la aplicación para implementar el cómputo, comentado en el capítulo anterior, podemos encontrar solapamiento entre las comunicaciones y las regiones de cómputo dentro de cada proceso. Este solapamiento influye directamente en el rendimiento de la aplicación ya que usualmente, al aumentar el mismo, la aplicación ofrece un mejor rendimiento. Lamentablemente, la estimación de este solapamiento no es una tarea trivial y requiere grandes esfuerzos para llegar a resultados valiosos.
- **Patrón de acceso a los datos:** las aplicaciones en su labor de cómputo de la solución pueden acceder a los datos de manera diferente. Realizar accesos de manera lineal, o a partir de saltos fijos o realizar accesos a datos dentro un una zona reducida de la estructura. A la vez, pueden repetir dichos accesos en cada iteración o no. Todo ello está demostrado que influye en el rendimiento de la aplicación ya que dependiendo del aprovechamiento que se realice de la localidad espacial y temporal, lo fallos de cache pueden disminuir y estos a su vez, mejorar el rendimiento.
- **El *mapping* de la aplicación:** como fue detallado previamente en el capítulo 2, pueden existir diferentes configuraciones para la ejecución de una aplicación paralela incluso utilizando la misma cantidad de recursos de cómputo. Los resultados para cada una de

las configuraciones pueden variar significativamente a partir de los beneficios que obtengan una configuración frente a otra.

- **Topología de red de interconexión:** existen diferentes topologías para la interconexión entre los diferentes nodos del supercomputador. Por ejemplo las mallas, los toros, los toros 3D, los *fat-tree*, etc. Cada una tiene sus particularidades e influyen en el rendimiento global de la aplicación teniendo en cuenta, adicionalmente la ubicación dentro de la red, de los nodos utilizados en la ejecución de la aplicación.
- **Latencias de comunicación:** depende de las especificaciones técnicas de los fabricantes de los equipos de comunicación, los algoritmos de encaminamiento utilizados por los *routers* y las características del canal físico de comunicación utilizado.
- **Patrón de comunicación de la aplicación:** dependiendo del comportamiento de las comunicaciones en cuanto a cuando ocurren las mismas, quienes son los procesos emisores de mensajes y cuáles son los receptores, el rendimiento de las aplicaciones puede variar. Pueden existir procesos críticos a los que se vuelca mucha comunicación y en consecuencia, retardar el funcionamiento global.
- **No existe un modelo de rendimiento para aplicaciones OpenMP:** hasta el momento, no existe una aportación concreta de un modelo que permita predecir el comportamiento de una aplicación paralela de paso de mensajes o al menos, un modelo para estimar el comportamiento de una región paralela *OpenMP*. En consecuencia, se dificulta encontrar un modelo para las aplicaciones híbridas ya que las mismas están conformadas por regiones de cómputo *OpenMP*.

Estos factores por si solos, ejercen influencia en el comportamiento global de la aplicación, pero es necesario destacar que cada uno no actúan individualmente, más bien todos lo hacen a unísono por lo que la tarea es aun más compleja al existir estas influencias múltiples que puede o no anularse entre sí o afectarse en algún grado entre todas ellas.

3.4 Trabajos relacionados

A continuación, pasaremos a comentar las principales aportaciones realizadas en la comunidad de HPC, en torno al análisis, predicción de rendimiento y propuestas de modelos de rendimiento para las aplicaciones híbridas. Los esfuerzos en esta línea, se dividen en tres grupos fundamentales: los estudios de rendimiento para diversas aplicaciones científicas, algunos modelos matemáticos para aplicaciones híbridas que permiten caracterizar el comportamiento de las comunicaciones y el cómputo.

3.4.1 **Análisis de rendimiento**

La publicaciones más comunes encontradas dentro de la literatura sobre el tema de las aplicaciones híbridas, enfocan la investigación al análisis del comportamiento de las mismas contrastándolas con las aplicaciones puras de paso de mensajes y las de memoria compartida. En algunos casos, se realizan análisis más detallados del comportamiento de las comunicaciones y de la eficiencia de las regiones de cómputo paralelo.

A continuación, se describirán algunas de las últimas aportaciones en estos temas:

- En los artículos [**6**] [**7**] [**8**] se realiza un análisis del comportamiento de *benchmark* de NAS en su versión de paso de mensajes. Para realizar la comparación con la versión híbrida de este *bechmark* en ambos artículos se realizaron modificaciones en el código de la aplicación para permitir el paralelismo en las zonas de cómputo intensivo. En estos estudios, se realizan mediciones del comportamiento de las comunicaciones y el cómputo. A la vez, se mide el comportamiento de los contadores de hardware con la finalidad de detectar influencias que podrían tener los fallos de acceso a los diferentes niveles de memoria en el comportamiento global de las comunicaciones y el cómputo.
- El estudio realizado por Geraud Krawezik y Frank Capello [**9**], se enfoca en una comparación entre la versión de paso de mensajes del *benchmark* de NAS con relación a tres implementaciones diferentes de dicha aplicación usando únicamente el modelo de paralelismo de paso de mensajes. Para realizar los análisis del comportamiento contrastado entre estas aplicaciones, el artículo se enfoca fundamentalmente en la medición del rendimiento para cada aplicación así como la evaluación de la escalabilidad de cada una y la medición del comportamiento de los contadores de hardware. Una de las versiones desarrolladas para realizar el estudio comparativo, logra implementar una versión SPMD [**10**] usando únicamente *OpenMP*. Como conclusiones de su estudio, encontramos que para la mayoría de los casos las versiones que usan memoria compartida no ofrecen una mejoría con relación a la versión MPI. Solo en la versión SPMD usando *OpenMP* encuentra un buen rendimiento contrastado con el resto del las versiones sobre todo teniendo en cuenta que el estudio de realiza en varias arquitecturas de hardware. En este caso, el rendimiento obtenido por esta versión es equivalente al de la versión MPI aunque se obtiene una ganancia en las regiones de comunicación.
- Un resultado interesante es el obtenido por Kengo Nakajima de la Universidad de Tokyo [**11**]. En su estudio comparativo entre aplicaciones MPI e Híbridas se obtienen beneficios en rendimiento utilizando dos estrategias fundamentales: modificando la

afinidad de los cores utilizados por la regiones *OpenMP* y realizando modificaciones en el código de manera que el acceso a los vectores en las fases de cómputo se comporte de una manera continua. Gracias a ello, se logra obtener un beneficio significativo al disminuir los fallos de acceso a memoria, llegándose a mejorar el rendimiento global en un 60%.

- La universidad de Edinburgh, a su vez, publicó en el año 2000 [**12**] un artículo donde se realiza un análisis del rendimiento de las aplicaciones híbridas contrastadas con las *OpenMP* para la solución de problemas de modelado de elementos discretos. En este caso, se identifican como factores fundamentales el buen uso de la utilización de la memoria cache y el nivel de eficiencia del paralelismo en cada versión.

3.4.2 Eficiencia de las comunicaciones y el cómputo

Dentro de los aportes en temas relacionados con nuestra investigación, podemos encontrar dos propuestas de modelos que permiten evaluar la eficiencia, para cada caso del comportamiento de las comunicaciones (Ecuación 3.1) y el cómputo (Ecuación 3.2) en las aplicaciones paralelas [**13**]. A continuación, presentamos la expresión para evaluar la eficiencia de las comunicaciones:

$$E_{mp}(n_t p) = \frac{\sum t_{hyb}^{comm}(m_t p)}{\sum t_{mpi}^{comm}(n_t p)}$$

Ecuación 3.1 Eficiencia de las comunicaciones

Donde $t_{comm\ hyb}$ es el tiempo estimado de comunicación de la aplicación híbrida, $t_{comm\ mpi}$ es el tiempo invertido en las comunicaciones si la aplicación es ejecutada sin el paralelismo de OpenMP (MPI pura). La variable p sería el número de procesos MPI, m_t es la cantidad de de threads de las regiones paralelas en las que se realizan procesos de comunicación, n_t es el número de *threads* total por cada proceso. De esta manera, $n_t p$ sería la cantidad total de cores utilizados por la aplicación. Esta expresión nos permite estimar la eficiencia del comportamiento de las comunicaciones en la aplicación híbrida, contrastándola con su equivalente versión MPI.

La Ecuación 3.2 a su vez, nos permite evaluar la eficiencia del cómputo paralelo. En este caso $t_{comp\ serial}$ es el tiempo de cómputo serie

$$e_{mt} = \frac{\sum t_{serial}^{comp}}{n_t\left(\frac{t_{serial}^{comp}}{n_t} + \sum t_{unpar}^{comp} + \sum O\right)}$$

Ecuación 3.2 Eficiencia del cómputo paralelo

A su vez, se ha propuesto una expresión para evaluar el nivel de desbalance del cómputo entre los diferentes procesos [**14**]. La Ecuación 3.3 nos muestra dicha expresión:

$$LB = \frac{\sum_{num\_CPUs}^{x=1}\left(cpu\_time_x\right)}{Max_{num\_CPUs}^{x=1}\left(cpu\_time_x\right) * num\_CPUs} * 100$$

Ecuación 3.3 Nivel de desbalance de cómputo entre procesos

3.4.3 **Modelos para las aplicaciones de paso de mensajes**

Nuestro departamento en particular, en el trascurso de los últimos años, ha llegado a definir modelos de rendimiento para aplicaciones de paso de mensajes. Forman parte de trabajos de investigación anteriores [**15**] [**16**] el desarrollo de modelos de rendimiento para aplicaciones tipo *Master-Worker*, *Pipeline* y la combinación de ambas (*Master-Worker de pipeline*) llegando también a definir estrategias de sintonización dinámica basadas en la modificación, durante el período de ejecución de la aplicación, de parámetros como el número de workers a utilizar y el grado de particionamiento de los datos a ser computado en cada fase. En todos los casos, estos modelos tienen en consideración el tipo de comunicación utilizada por la aplicación, ya que, en el caso de la utilización de comunicaciones asincrónicas o sincrónicas influye en la utilización de uno u otro modelo particular.

Se trabaja paralelamente, en el desarrollo de un modelo para aplicaciones SPMD motivado fundamentalmente porque en la actualidad, es el modelo más utilizado en el desarrollo de aplicaciones paralelas para *HPC* y también en el desarrollo de modelos para aplicaciones de memoria compartida. De los resultados de estas dos potenciales aportaciones futuras, se beneficiará también los modelos de rendimiento para aplicaciones híbridas.

Capítulo 4

Metodología y estudio de las herramientas

4.1 Introducción

Con el objetivo de definir claramente que pasos seguiríamos para llegar a alcanzar los objetivos propuestos en nuestra investigación, nos dimos a la tarea de realizar una revisión en la bibliografía científica, de los aportes en cuestiones metodológicas sobre el tema y buenas prácticas sugeridas. Basados en las aportaciones realizadas en [**17**] y [**18**] hemos planteado nuestra metodología.

4.2 Definición de la metodología

A continuación, la figura 4.1 muestra el flujo de trabajo definido por nuestra metodología. El mismo, consta de siete pasos, en alguno de los cuales pueden existir retrocesos dependiendo de la necesidad de refinamiento de nuestro modelo en base a la inclusión de nuevos parámetros.

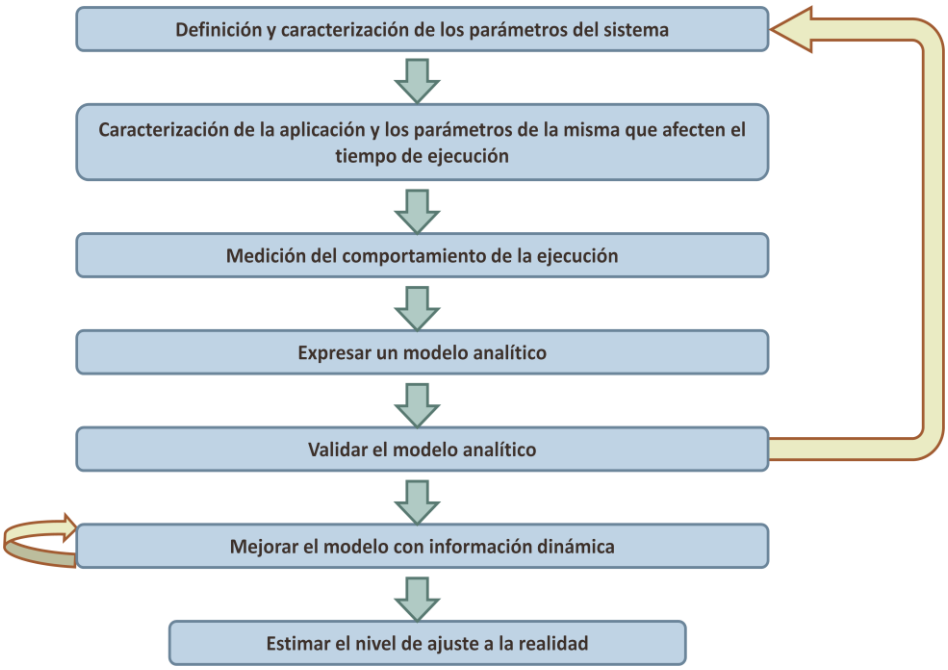


Figura 4.1 Metodología propuesta

Hemos definido, como primera etapa de nuestro proceso, la *Definición y caracterización de los parámetros del sistema*. En la misma se debe realizar un conjunto de test de análisis y caracterización con el fin de obtener información valiosa que describa los aspectos siguientes:

- **Velocidad del procesador**: Dependiendo de nuestro alcance y/o necesidades, podemos pasar de utilizar la especificación del fabricante a desglosar esta característica teniendo en cuenta medidas como el rendimiento en operaciones de punto flotante, las operaciones con enteros, etc.

- **En número de enlaces de comunicación por nodos de cómputo**: describe la cantidad enlaces físicos de comunicación para un nodo dentro de una red de interconexión. Para una topología de red determinada, este parámetro suele ser fijo.

- **El ancho de banda y las latencias de las comunicaciones, entre otras**: este parámetro nos describiría para un tamaño de paquete determinado, cuál sería el costo el tiempo del envío a través de la red de interconexión entre los nodos y qué cantidad de información puede ser enviada por segundo. También se puede describir, cual es el *overhead* añadido de la librería de paso de mensajes para un tamaño de paquete determinado.

- **Los tiempos de acceso a los diferentes niveles de la memoria**: con este parámetro se pretende caracterizar, cuanto penaliza los fallos de cache en el tiempo de ejecución del un programa. Para ello, se pueden utilizar algunos *benchmarks* que permitan la caracterización del comportamiento de las comunicaciones o la utilización de una aplicación sintética que evalué dicho parámetro. Una estrategia más simple y a la vez menos precisa, puede ser la utilización de las especificaciones técnicas del fabricante.

- **Overheads intrínsecos de las librerías así como los *overheads* en la creación de los *threads* etc**: en este caso, como existen diferentes implementaciones del estándar *OpenMP* y a la vez, los compiladores en muchos casos, suelen estar optimizados para una arquitectura en concreto, se sugiere utilizar herramientas de análisis que nos permitan medir dichos tiempos. En cualquier caso, la idea es medir, dependiendo de la cantidad de *threads* utilizados en las regiones de cómputo paralelo, cual es el costo en tiempo que agrega la librería a la hora de crear, destruir y sincronizar (definida explícita o implícitamente en el código) los hilos utilizados.

Los aspectos más importantes a considerar, para la segunda fase de nuestra metodología en donde se realiza una caracterización de la aplicación y de sus parámetros, son los siguientes:

- **Tamaño de los datos de entrada a la aplicación:** se debe medir que tamaño ocupan los datos que son utilizados por la aplicación para la obtención de los resultados. La magnitud de dicha medición (bytes, Kb, Mb) dependerá de las características propias de cada aplicación analizada.

- **La estructura de dichos datos:** sugiere realizar una descripción más detalla de estos datos con el fin de describir cuantas estructuras son y si se utilizan matrices, arreglos etc. Teniendo en cuenta, adicionalmente la dimensión de cada uno.

- **La manera en que la aplicación accede a los mismos:** con este parámetro se pretende describir de manera más detallada y, a partir de un análisis de la lógica de funcionamiento de la aplicación, de qué forma son accedidas las estructuras de los datos, teniendo en cuenta, el aprovechamiento que se realiza de la localidad espacial y temporal de los mismos.

- **El número de nodos utilizados y si hay algunos dedicados exclusivamente a operaciones de comunicación:** puede ser significativo el hecho de utilizar un nodo o un subconjunto de nodos. A su vez, la cantidad de nodos utilizados nos describe la capacidad de cómputo utilizada para una ejecución de la aplicación.

- **El número de tareas MPI, el número de *threads* por cada proceso MPI:** describe el *mapping* de la aplicación. Como se ha explicado anteriormente, para este parámetro se debe también tener en cuenta qué niveles de memoria cache son compartidos por cada *thread* en las secciones de cómputo paralelo dentro del nodo.

- **Buscar la relación (si es posible) entre el tamaño de los datos de entrada y el tiempo de ejecución de la aplicación:** se trata con ello de encontrar la relación existente entre ambos parámetros, lo cual permite enfocarnos es describir comportamientos atípicos para cierto volumen de datos iniciales.

- **El modelo de paralelismo utilizado:** como se ha descrito en capítulos anteriores, con esté parámetro se pretende caracterizar la aplicación híbrida a partir de la taxonomía antes descrita en el capítulo 2.

Para la tercera etapa, la *Medición del comportamiento de la ejecución*, se sugiere utilizar herramientas de análisis que nos permitan realizar mediciones de diversos parámetros durante en tiempo de ejecución. En el capítulo 4.4 se realiza un estudio detallado sobre las posibles herramientas y estrategias a utilizar para esta tarea. Dichos parámetros son:

- **Cantidad de operaciones de cómputo realizadas, desglosadas por el tipo de operación:** a partir del resultado de las ejecuciones, se sugiere medir la cantidad de operaciones con enteros, reales etc. así como el tipo de operación realizada en cada caso.

- **Medidas de los contadores de hardware (Fallos de cache, etc.):** utilizando librerías para la obtención de los contadores de *hardware*, por ejemplo PAPI [**19**] integrada ya dentro de los *kernel* de las últimas versiones de los sistemas operativos Linux; se sugiere medir la cantidad de fallos de acceso a los datos a los diferentes niveles de memoria, así como la cantidad de instrucciones ejecutadas por sección y *thread* de la aplicación.

- **Medir la cantidad de ocasiones que es ejecutada una sección de la aplicación y la duración total de cada una:** para ello se sugiere identificar claramente cada una de dichas secciones y medir, con la utilización de una herramienta de análisis, esté parámetro. Con ello, se pretende identificar que secciones tienen más influencia en el rendimiento final de la aplicación y la posible relación existente entre la cantidad de veces que es llamada una sección con relación su tiempo total de ejecución.

- **Relacionar esta última medición con el tamaño de los datos de entrada:** permite identificar cuáles de estás secciones dependen del tamaño de los datos de entrada. Por lo general, en las aplicaciones científicas, suele encontrarse una dependencia directa entre estos parámetros. Existen casos en los que no hay dependencia ya que el tiempo de cómputo no depende del volumen de los datos sino de la naturaleza de los mismos.

- **Medir la cantidad de ocasiones que son ejecutadas las regiones OpenMP:** utilizando herramientas de análisis se puede contar la cantidad de veces que una región paralela fue ejecutada. A su vez, se puede medir por *thread,* cuantas veces fue ejecutado cada uno y así tener una idea del nivel de desbalance de carga entre estos.

- **Medir los tiempos de *schedulling* y *sincronización* entre *threads*:** aunque en fases anteriores de la metodología se realiza una medición similar. Se sugiere volver a realizarla para la aplicación real con el fin de detectar algún cambio significativo con relación a la medición inicial. Este cambio puede estar provocado por un desbalance de la carga de cómputo entre los *threads* utilizados en la región *OpenMP*.

- **El costo en tiempo de barreras explicitas definidas con pragmas OpenMP:** similar al caso anterior solo que esta vez, se mide en los casos donde la barreras están definidas explícitamente dentro del código a través de pragmas *OpenMP*.

A partir de toda la información recogida, la cuarta fase propone realizar propuestas de un modelo analítico de predicción de rendimiento. Dicho modelo tendrá que ser validado en la fase siguiente donde se contrasta la predicción del mismo con el comportamiento real. Llegado a este punto, el proceso puede retornar a las etapas iniciales con el fin de refinar nuestro modelo teniendo en cuenta más o menos parámetros. En general, se eliminan o adicionan parámetros teniendo en cuenta el nivel de ajuste a la realidad del modelo y a la vez, midiendo el grado de influencia de cada parámetro con respecto al resultado final de la predicción.

La sexta fase, en donde se mejora el modelo con información dinámica, pretende definir estrategias de mejora del rendimiento de forma dinámica a partir de obtener, en tiempo de ejecución de la aplicación, diversos parámetros y sintonizar los mismos acorde con el resultado del comportamiento que vamos obteniendo en cada fase de ejecución. Para ello, es necesario previamente, definir las estrategias que seguiremos para dicha sintonización y comprobar que las mismas, en efecto, provoquen mejoría en el rendimiento global.

Como última fase de nuestra metodología, proponemos la estimación del error de nuestra política de sintonización. Comparando para ello, nuestra predicción de mejora del rendimiento al aplicar una sintonización dinámica, con la mejora real obtenida.

4.3 Caracterización de las comunicaciones

En particular, para realizar la caracterización del comportamiento de las comunicaciones existen dos técnicas principales. La primera se basa en la utilización de un *benchmark* programado con este objetivo [**20**] [**21**] [**22**] [**23**] para esta tarea y la segunda, pasa primero por detectar el patrón de comunicación de la aplicación para una fase de la misma y después elaborar una aplicación sintética que reproduzca dicho patrón y mida el comportamiento de las comunicaciones [**24**]. En ambos casos, la finalidad es obtener un historial de tiempos de envío de mensaje por tamaño de los mismos y promediar los resultados para finalmente elaborar una función que, a partir de un tamaño de mensaje determinado, nos devuelva un tiempo estimado de envío de dicho mensaje.

Es importante mencionar, que en ambos casos se asume que la aplicación esta balanceada. De esta forma, los tiempos de envíos no se verán afectados por el desfasaje que podrá existir en una aplicación real entre los procesos de comunicación MPI debido a la diferencia de carga de

trabajo que posean. En todo caso, dicha diferencia en tiempo de comunicación, medida en una aplicación real, nos serviría para estimar dicho desbalance de carga.

4.3.1 Aplicaciones para la caracterización de las comunicaciones

Producto de la revisión realizada para identificar los diferentes *benchmarks* con funcionalidades para la caracterización de las comunicaciones, se han encontrado los siguientes:

- PerfTest [**25**]
- MPBench [**26**]
- SKaMPI [**27**]
- Pallas MPI Benchmarks [**28**]
- Sphinx [**29**]

En todos los casos, el algoritmo utilizado para la caracterización de las comunicaciones se basa en realizar sucesivamente varias test de comunicación entre pares de nodos y obtener el tiempo resultante de dicha comunicación. Este proceso se repite incrementando de manera gradual el tamaño de los mensajes enviados para así obtener el comportamiento global. En caso de existir una diferencia sustancial entre los resultados de dos tamaños de mensajes diferentes, estos *benchmarks* vuelven a realizar una prueba adicional del algoritmo para un nuevo tamaño de mensaje calculado como la media entre los dos tamaños originales. Si es necesario, se vuelve a repetir esta lógica hasta que las diferencias en los resultados no sean significativas.

Estas aplicaciones, permiten realizar las diferentes pruebas especificando el tipo de comunicación que se pretende caracterizar. Gracias a ello, podemos caracterizar separadamente las comunicaciones bloqueantes, no bloqueantes y las comunicaciones colectivas. En la práctica, se recomienda realizar de esta forma los análisis ya que cada tipo, esta implementado de manera diferente en la librería de paso de mensajes y es de esperar que los comportamientos difieran notablemente entre ellos.

En particular, el *bechmark* MPBench ofrece como resultado adicional, una función de distribución probabilísticas por tamaño de paquetes, que nos proporciona una medida mucho más real del nivel de variabilidad que pueden tener los tiempos de comunicación en nuestra red de interconexión. En la práctica, si se confirma que la variabilidad es baja, no es necesaria la utilización de esta aplicación.

En los casos donde las aplicaciones poseen un patrón de comunicación semejante a estos *benchmarks* y los mismos son ejecutados utilizando las misma cantidad de nodos involucrados en la comunicación de la aplicación, resulta factible utilizarlos como herramientas para la caracterización de las comunicaciones pero, en caso contrario, cuando tenemos aplicaciones en donde el patrón de comunicación involucra a una cantidad mayor de dos nodos, los resultados no son fiables ya que no corresponden a la cantidad de comunicación que se genera en la red ni al volumen de las misma.

4.3.2 Caracterización utilizando la herramienta PerfTest

La figura 4.2 muestra el comportamiento de los tiempos de comunicación para tamaños de mensajes que oscilan entre 1024 bytes y 1800 bytes para 30 procesos involucrados en la comunicación. Estos resultados fueron obtenidos ejecutando un proceso MPI por cada nodo reservado en el cluster. Las líneas verticales representan el rango entre mínimo y máximo tiempo de comunicación registrado. De esta forma, podemos tener una idea clara del nivel de variación de las comunicaciones a partir de las múltiples pruebas realizadas.

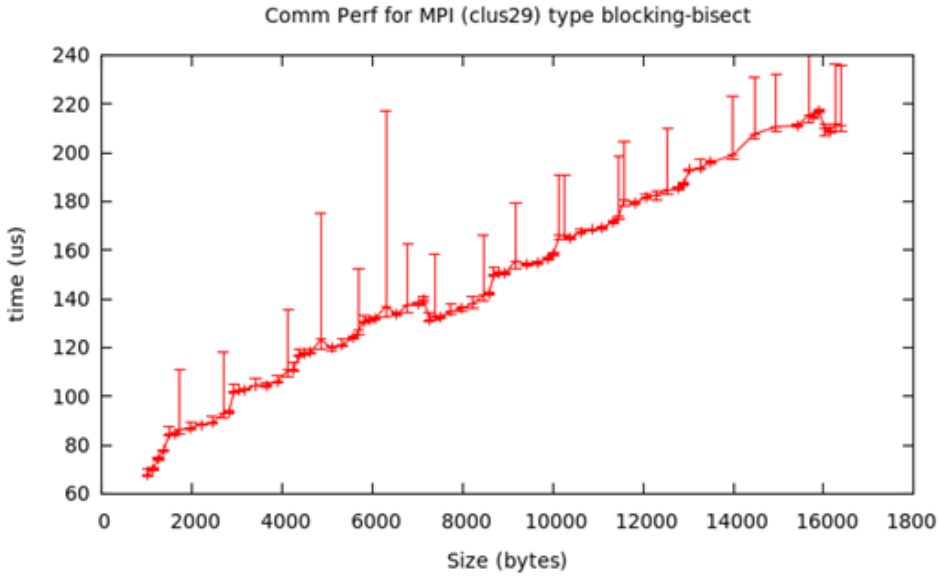


Figura 4.2 Tiempo de comunicación para funciones bloqueantes

La figura 4.3 a su vez, nos muestra el resultado obtenido por la herramienta cuando se ejecuta la caracterización de las comunicaciones colectivas. En este caso, el *benchmark* muestra las gráficas del comportamiento a medida que se incrementa la cantidad de procesos MPI involucrados de la comunicación colectiva y a la vez, se varía el tamaño del mensaje comenzado por uno de 1024 bytes hasta llegar a 17408 bytes. En todo caso y, dependiendo de nuestras necesidades, el rango de tamaño de paquetes es configurable como un parámetro de este *bechmark*.

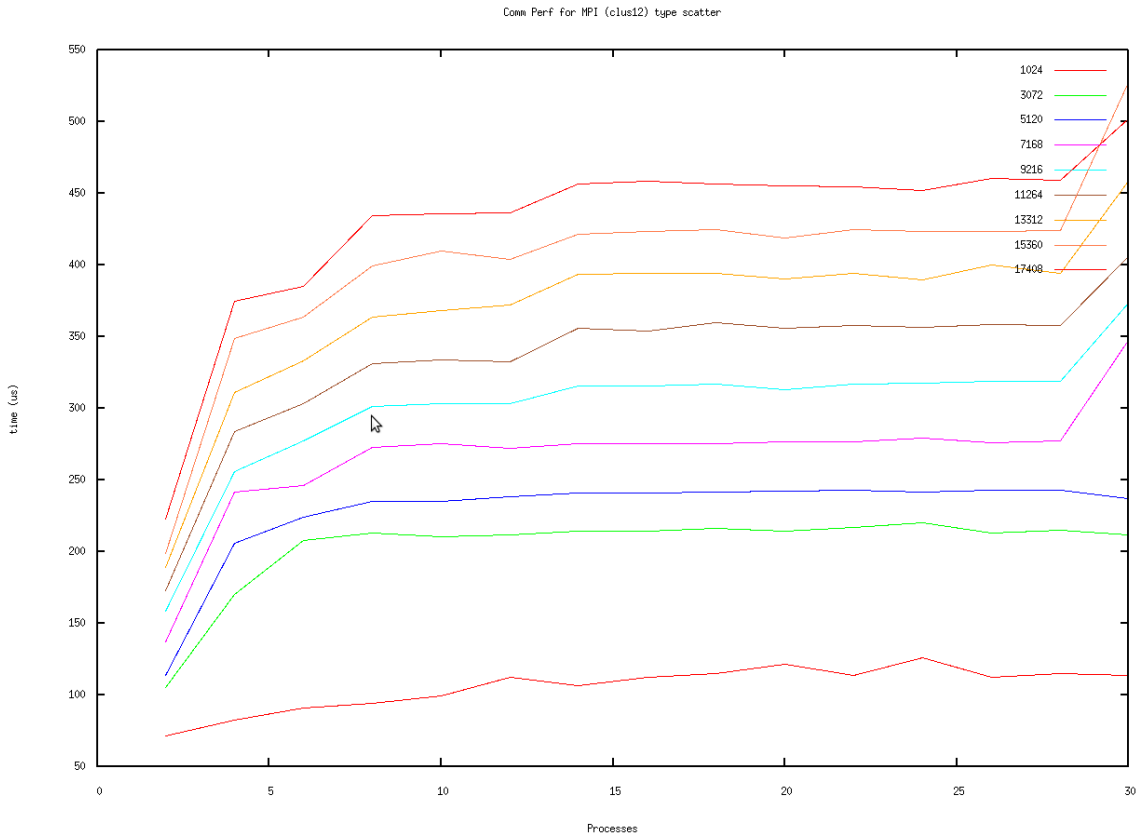


Figura 4.3 Tiempo de comunicación: comunicaciones colectivas

4.4 Herramientas de análisis de rendimiento

El objetivo fundamental de una herramienta de análisis de rendimiento es permitir, a partir de la obtención de resultados del comportamiento de la ejecución de las aplicaciones, obtener información valiosa que nos permita llegar a conclusiones sobre qué factores influyen en el comportamiento de estas aplicaciones. Para ello, muchas nos proveen de mecanismos para la visualización de la ejecución de la aplicación paralela una vez que ésta ha terminado de ejecutarse. A este tipo de análisis se le denomina análisis de rendimiento post-mortem.

Como valor agregado y, a partir de los ficheros de trazas obtenidos en la ejecución gracias a una instrumentación del programa original, estas herramientas también posibilitan la presentación de información específica sobre el comportamiento de la aplicación mediante diferentes formas de visualización: gráficas, tablas, vistas etc. Para ello, primeramente se requiere el uso de herramientas que realicen la monitorización para obtener datos de rendimiento de la ejecución del programa paralelo. Lo usual es que estas herramientas se concentren en realizar mediciones como el número de ocurrencias de un determinado evento en la aplicación, los tiempos de invertidos por secciones de la aplicación, el comportamiento

de parámetros del sistemas y de la aplicación en el momento de la ocurrencia de algún evento, entre otras.

Con dicha información, las herramientas de visualización generan gráficos referentes al comportamiento que sigue la aplicación, utilizando para ello diagramas de Gantt, diagramas circulares, de barra, gráficos tridimensional de tiempos de ejecución por secciones, etc. Esta información más elaborada a nivel visual, debe ayudar al usuario para identificar secciones críticas de la aplicación así como comportamientos no esperados en las diferentes fases de la ejecución. A partir de todo ello y, gracias a un proceso cíclico e incremental, se pueden modificar la aplicación con el fin de obtener mejoras en su comportamiento.

Si solo se apostara por llegar a conclusiones a partir del comportamiento de la aplicación descrito por un diagrama de Gantt, el proceso de identificación de los factores que influyen en el rendimiento de las aplicaciones sería bien complejo y demorado. Cuando la cantidad de recursos involucrados en el cómputo aumenta, el volumen de información a analizar lo hace en igual proporción por lo que, en muchos casos, se convierte en una labor titánica especialmente para usuarios no expertos. La complejidad de esta tarea se debe principalmente a la interpretación y tamaño del fichero de traza.

Es por ello que la tendencia actual en la mayoría de las herramientas, es ofrecer funcionalidades adicionales que permitan realizar un análisis mucho más ágil y veraz, aprovechando técnicas de análisis y correlación de datos, u obteniendo a partir de la información resultante de las trazas, medidas que describan con más detalle las particularidades del funcionamiento de la aplicación en su ejecución.

4.4.1 Vampir

Vampir [**30**] es una herramienta de análisis de rendimiento que permite la visualización gráfica y análisis de los cambios de estado de un programa, mensajes punto a punto, operaciones colectivas y contadores de rendimiento hardware junto con resúmenes estadísticos.

Fue concebida con el fin de que su utilización sea sencilla y fácil para los usuarios, por lo que no es necesario un alto grado de experiencia obtener a partir de ella, resultados del comportamiento de una aplicación para un determinado nivel de detalle. Comenzó a desarrollarse en el Centro de Matemática Aplicada del Centro de Investigación de Jülich y el Centro de Computación de Altas Prestaciones de la Universidad Técnica de Dresden. Vampir está disponible como producto comercial desde 1996. Actualmente el desarrollo de Vampir continúa por parte del Centro de Servicios de Información y Computación de Altas

Prestaciones (ZIH) de la Universidad Técnica de Dresden. Esta validada desde hace muchos años por la comunidad científica ya que la misma ha sido utilizada ampliamente en el entorno de la computación de altas prestaciones. No tiene una distribución open-source y las licencias, incluso las académicas, no son gratuitas.

Vampir a su vez, es capaz de entender los ficheros de trazas generados por otras herramientas de monitorización como TAU, KOJAK o VampirTrace. A partir de la versión 5.0, Vampir soporta el formato Open Trace (OTF), desarrollado por ZIH. Este formato de traza está especialmente diseñado para programas masivamente paralelos y por lo general, las últimas versiones de todas las herramientas actuales vienen con este soporte incluido. A su vez, se puede obtener con esta herramienta gráficos temporales que muestran el comportamiento de las secciones de cómputo y las comunicaciones a lo largo del de tiempo, sobre los cuales el usuario puede desplazarse y hacer zoom, con el objetivo de detectar la causa real de los problemas de rendimiento. Además permite verificar la correcta paralelización y el nivel de balanceo de carga. Vampir posee funcionalidades para la generación de gráficos estadísticos que proporcionan resultados cuantitativos del comportamiento de diversas secciones de la aplicación.

4.4.2 VampirTrace

Esta herramienta [**31**] permite el análisis de aplicaciones desarrolladas con *MPI* y *OpenMP*. En esencia, la herramienta modifica la aplicación para agregarle cierta lógica que detecta y almacena eventos de interés generados en el período de la ejecución de la aplicación. Posee tres formas para realizar esta operación: a nivel de código fuente, en el proceso de compilación o en el proceso de enlace.

La librería que soporta toda la funcionalidad de esta herramienta, se encarga de la recolección de información de todos los procesos involucrados. Esta información, está compuesta por los eventos de comunicación MPI, eventos de las secciones paralelas *OpenMP*, así como información sobre temporización o localización. Incluye adicionalmente la posibilidad de obtener información de los contadores hardware mediante la librería PAPI. Como otras herramientas, permite generar ficheros resultantes de trazas en formato OTF.

Su implementación está basada en el conjunto de herramientas KOJAK y es desarrollado en ZIH, en cooperación con ZAM, Centro de Investigación de Jülich, Alemania y el Laboratorio de Computación Innovadora de la Universidad de Tennessee, EEUU.

Aunque puede ser utilizada con otras herramientas gracias a que los ficheros generados son ya un estándar, está pensada para utilizarse en combinación con Vampir. Siendo VampirTrace la

herramienta encargada de la generación de trazas y Vampir la herramienta para una visualización más amigable de los resultados.

4.4.3 Dimemas-Paraver

Paraver [**32**] y Dimemas [**33**] son dos herramientas de análisis de rendimiento automático desarrolladas en el Centro Europeo de Paralelismo de Barcelona (CEPBA) en 1996 y 1992 respectivamente.

Paraver, en particular permite el análisis y visualización del rendimiento de las aplicaciones que empleen MPI, OpenMP, MPI+OpenMP, Java. Para ello, esta herramienta genera trazas que de los eventos ocurridos durante el periodo de ejecución. Adicionalmente, incluye resúmenes de contadores hardware y la actividad del sistema operativo. En general, Paraver fue desarrollada con el objetivo de responder a la necesidad de tener una percepción cualitativa global del comportamiento de las aplicaciones utilizando para ello mecanismos de presentación visual de los resultados con el objetivo de llegar a la detección de los problemas de rendimiento inherentes a la aplicación estudiada. A su vez, permite identificar comportamientos críticos lo que permite enfocarnos en problemas puntuales con una mayor significación en el rendimiento global de la aplicación.

Algunas de las principales características de Paraver son:

- Análisis cuantitativo detallado del rendimiento del programa.
- Análisis comparativo concurrente de varias trazas.
- Análisis rápido para trazas de gran tamaño.
- Permite trazas con mezcla de paso de mensajes y memoria compartida.
- Permite la personalización de la información a visualizar.
- Generación de métricas derivadas.

Paraver presenta 3 tipos de visualizaciones:

- Vista gráfica: representa el comportamiento de la aplicación en el tiempo de ejecución de la misma
- Vista textual: proporciona el máximo detalle sobre la información mostrada.
- Vista de análisis: proporciona datos cuantitativos.

Dimemas es una herramienta de análisis de rendimiento para programas basados en paso de mensajes usando el estándar MPI. El usuario, a través de esta herramienta puede sintonizar aplicaciones paralelas en su estación de trabajo y una vez hecho esto, la herramienta proporciona un mecanismo de predicción de su rendimiento en la máquina paralela objeto de

la ejecución. Para ello el simulador, reproduce el comportamiento de la aplicación paralela a partir de las mediciones de rendimiento obtenidas. De esta forma, se pueden obtener predicciones de comportamiento para diferentes arquitecturas, variando la carga de los datos de cómputo, modificando parámetros del sistema o modificando el mapeo de la aplicación en los diferentes nodos de cómputo. El simulador genera ficheros de traza interpretables por herramientas de análisis como Paraver y Vampir aunque, puede ser importado por otras herramientas con la utilización de herramientas de transformación de hacia otros formatos.

Esta herramienta es útil en dos fases de la vida de una aplicación: durante su desarrollo, para realizar un análisis de los efectos de diferentes parámetros en el rendimiento sin requerir el uso de la arquitectura sobre la que se desea ejecutar; y después la fase de producción, para seleccionar la mejor arquitectura para ejecutar la aplicación.

Las entradas de Dimemas son: un fichero de traza y un fichero de configuración. El fichero de traza contiene los datos de una ejecución real en una máquina que captura información sobre la CPU y patrones de comunicación. La segunda entrada es un fichero de configuración que contienen un conjunto de parámetros que modelan la arquitectura deseada.

Como salida de la aplicación Dimemas podemos obtener la predicción del tiempo empleado en la ejecución de la aplicación sobre la plataforma especificada o una visualización del fichero de trazas.

4.4.4 OMPP

La herramienta ompP [**34**] [**35**] permite obtener información particularizada para cada *thread* de ejecución y para cada sección de la aplicación analizada. Adicionalmente, nos devuelve en grafo de llamadas a funciones de la aplicación. Por defecto, la información obtenida por la utilización de esta herramienta nos provee de datos sobre el tiempo de ejecución, la cantidad de ocasiones que fue llamada una sección o un hilo. También permite la integración con la librería PAPI lo que nos posibilita monitorizar el comportamiento de los contadores hardware del sistema, como fallos/aciertos de cache, operaciones en punto flotante, etc.

Como resultado de su utilización, la herramienta genera un conjunto de tablas detallas por sección y funciones *OpenMP* donde se describe en detalle el comportamiento de cada una en base a las métricas anteriormente descritas, es incluido en dichos resultados el nivel de desbalance de cada sección comparada con sus similares.

4.4.5 Scalasca

Scalasca [**36**] [**37**] es un conjunto de herramientas de análisis de rendimiento automático que ha sido especialmente diseñado para el uso en sistema de gran escala, pero también está diseñando para su uso en plataformas HPC de pequeña y media escala. Ha sido desarrollado en el Centro de Supercomputación de Jülich, Alemania en el año 2008.

Esta herramienta realiza un análisis de rendimiento que integra fases de *tracing* y *profiling* en tiempo de ejecución. Como la obtención de los ficheros de trazas de ejecución de la aplicación se logra agregando una lógica adicional a la misma, cuando es ejecutada la aplicación, en cada proceso se van generando dichos ficheros de resultados. Esto permite que el mecanismo escale sin grandes dificultades a medida que se involucran una mayor cantidad de nodos en la ejecución.

La actual versión de Scalasca permite el análisis de rendimiento de aplicaciones basadas en MPI, *OpenMP*, y aplicaciones híbridas y puede ser utilizada para en la gran mayoría de las aplicaciones HPC ya que soporta lenguajes como C, C++, y Fortran.

La figura 4.4 muestra la interface Cube QT en la cual se muestran los resultados obtenidos por la monitorización de una aplicación utilizando esta herramienta.

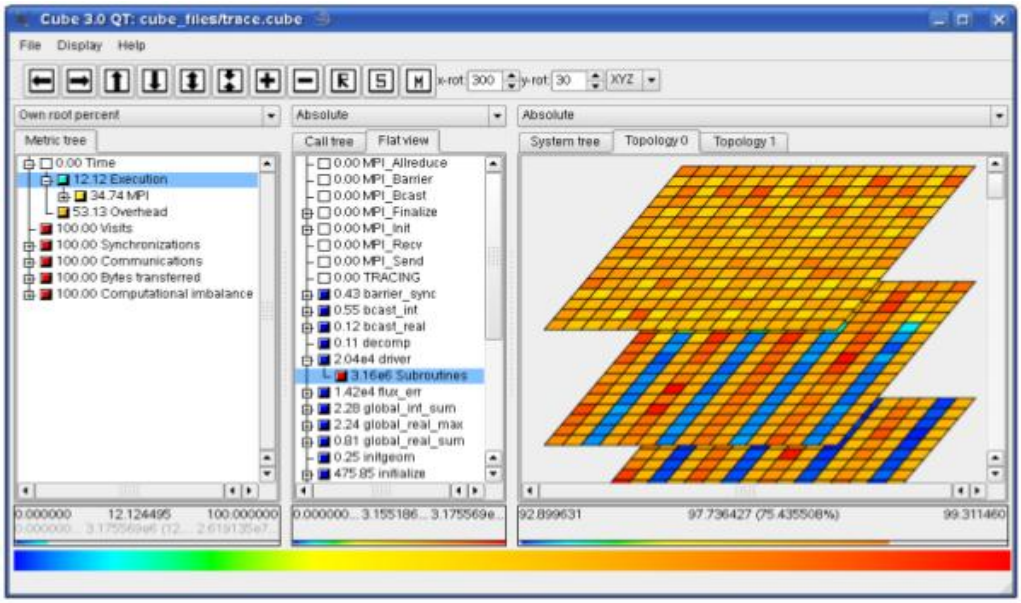


Figura 4.4 Interface Cube QT de Scalasca

En el proceso de recopilación de los datos de rendimiento, una primera etapa obligatoria pasa por instrumentar la aplicación objetivo. Este paso, en esencia, agrega una lógica a la aplicación para permitirle la generación de los ficheros de trazas. Se puede configurar la instrumentación

para obtener solamente ficheros de trazas de ejecución y/o incluir resultados resumidos de métricas de rendimiento.

Los informes obtenidos una vez concluida la ejecución de la aplicación. Deben ser generados a partir de una nueva ejecución de la aplicación pero en modo de análisis. Estos informes contienen métricas de rendimiento para cada llamada a función y recurso del sistema (proceso/hilo), y permiten ser explorados con la utilización de una interfaz gráfica para ello. En caso de necesitar utilizar estas trazas con otra aplicación de análisis, la plataforma permite la conversión de las mismas a los diferentes formatos ya extendidos dentro de la comunidad científica.

Adicionalmente, nos provee de algunos resultados muy útiles a la hora de evaluar el comportamiento de las comunicaciones MPI y del *overhead* agregado por las librerías de *OpenMP*, así como el grado de desbalance entre cada hilo utilizado así como es costo en tiempo en la creación, destrucción y sincronización de los mismos.

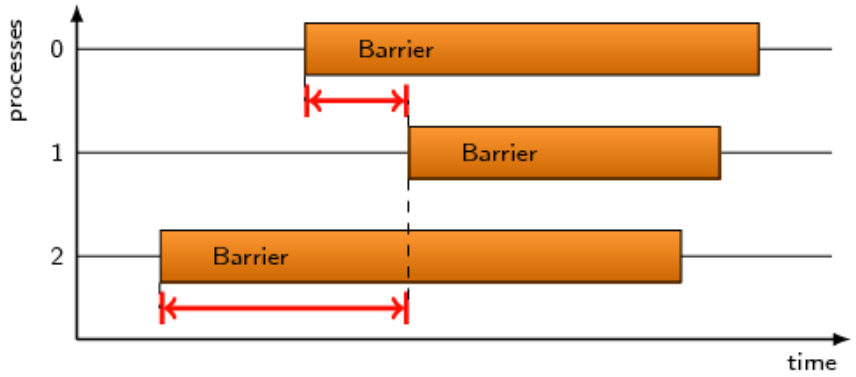


Figura 4.5 Tiempo de espera producto de una barrera

La figura 4.5 muestra un diagrama que nos permite entender una de las métricas obtenidas a partir de la utilización de Scalasca. La misma mide el tiempo malgastado por la llamada a una función *MPIBarrier* en cada proceso. Con la misma, podemos tener una idea de cuánto tiempo se pierde al realizar una sincronización colectiva entre los diferentes procesos involucrados en la misma.

Otra de las métricas útiles es la descrita en la figura 4.6 en donde se muestra como la herramienta mide la cantidad de tiempo acumulado perdido cuando un proceso queda en espera de la recepción de un mensaje debido a que el envío se realiza después de la ejecución de la función de recepción.

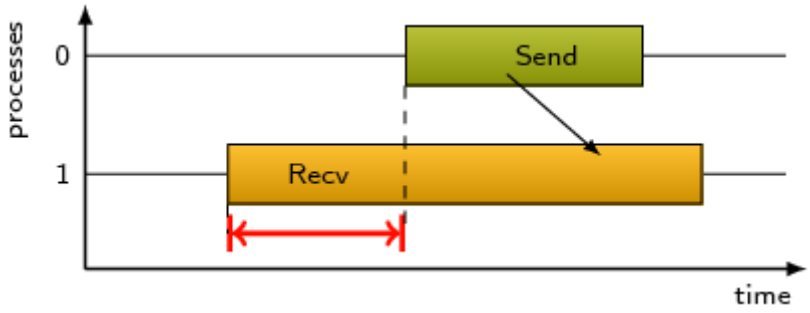


Figura 4.6 Tiempo perdido por un envío retrazado

En el caso de las comunicaciones colectivas, también hay métricas para evaluar la perdida en tiempo producto de las mismas. Es el caso mostrado en la figura 4.7 en donde se calcula este tiempo acumulado para cada proceso involucrado en la comunicación.

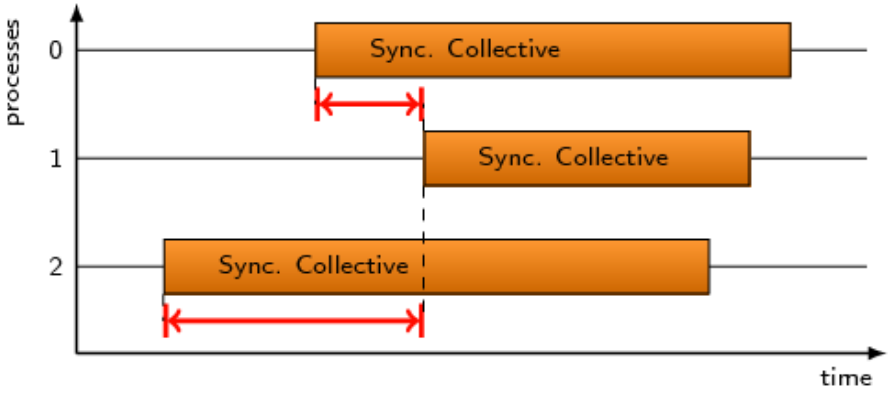


Figura 4.7 Tiempo inicial invertido en la comunicación colectiva

Un último ejemplo de mediciones realizadas por la herramienta que pueden resultar muy útiles es la mostrada en la figura 4.8. En la misma, se miden los tiempos desperdiciados por la librería *OpenMP*, en el momento de la creación y destrucción de los hilos en las regiones de cómputo paralelo.

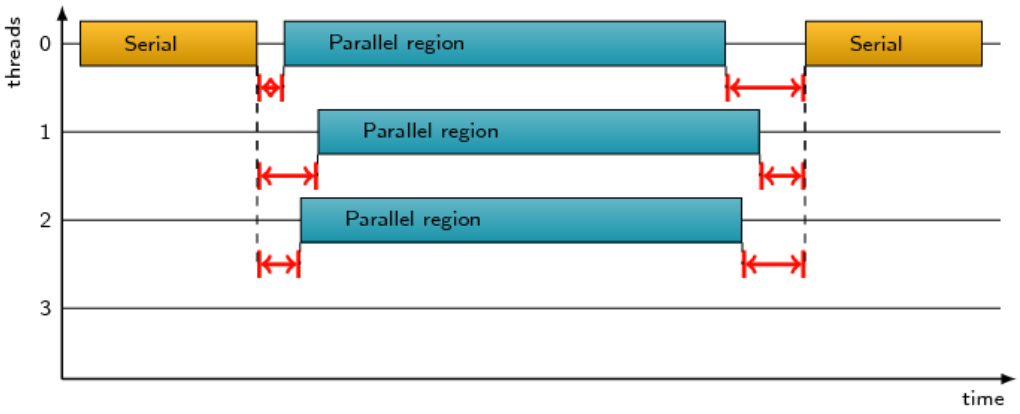


Figura 4.8 Tiempo desperdiciado en la creación y destrucción de los hilos

Los ejemplos anteriores constituyen un subconjunto de las mediciones realizadas por Scalasca que pueden resultar valiosas con el fin de detectar problemas que afecten el rendimiento. Una

descripción completa de todas puede ser revisada a partir de la documentación referenciada en este trabajo.

4.4.6 TAU

TAU (*Tuning and Analysis Utilities*) [**38**] [**39**] [**40**] es una herramienta de análisis de aplicaciones paralelas que permite realizar un análisis de las aplicaciones través de conjunto de herramientas automáticas para instrumentación, la medición en tiempo de ejecución y posterior análisis y visualización de los parámetros observados. Fue desarrollado en 1992 en la Universidad de Oregon, en EEUU, en colaboración con el Centro de Investigación de Jülich y el Laboratorio Nacional de Los Alamos.

Dentro de sus principales bondades se encuentran el gran número de plataformas hardware y software que soporta. Es posible ejecutar esta herramienta en la gran mayoría de las plataformas de HPC y permite utiliza librerías para la instrumentación que posibilitan llegar a la mayoría de los lenguajes que son utilizados en la actualidad, por ejemplo C, C++, Java, Python, Fortran. Otra de sus bondades radica en la posibilidad de obtener información tanto del comportamiento de las comunicaciones usando MPI como de los *threads* definidos por las regiones paralelas en OpenMP. La figura 4.9 muestra una de las herramientas incluidas en el paquete Tau llamada *ParaProf*. La misma permite la visualización de información de los resultados ejecución de la aplicación detallados los mismos por nodo, *therad* y sección de la aplicación.

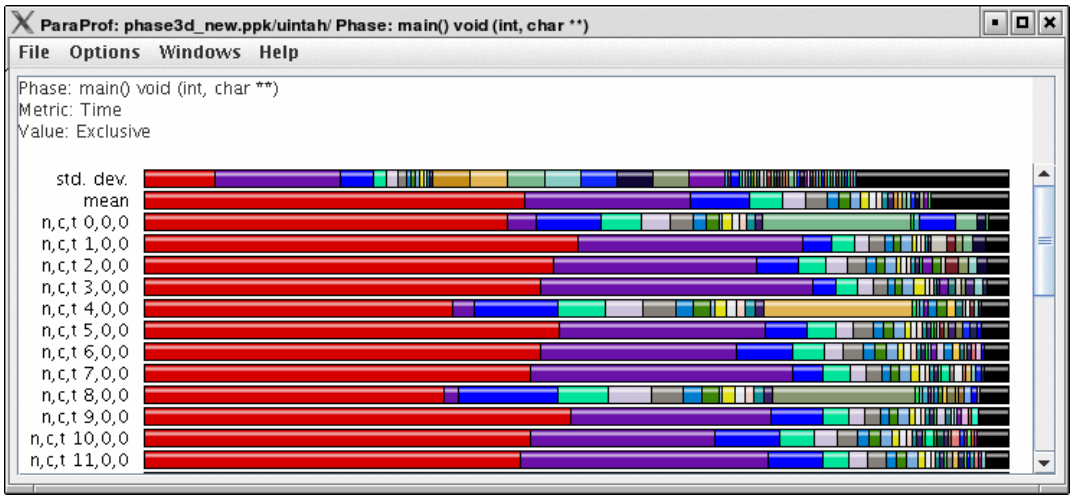
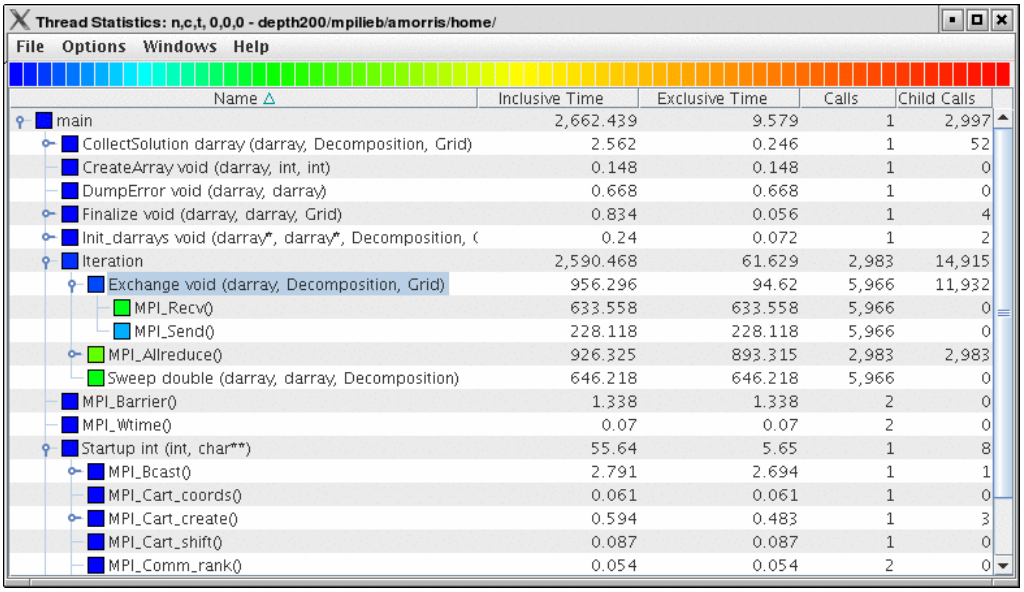


Figura 4.9 ParaProf: Herramienta de visualización de profiling

Esta visualización de los resultados puede ser obtenida de manera tabular filtrando los mismos por un par *nodo-thread* determinado. En este caso, la figura 4.10 muestra un ejemplo de lo planteado. Se pueden excluir de estos resultados algunas secciones de la aplicación dependiendo de las necesidades que tenga el usuario de la herramienta. Por defecto, las cuatro columnas de resultados muestran en tiempo inclusivo, el tiempo exclusivo, la cantidad de ocasiones que fue ejecutada esa sección y la cantidad de ocasiones que fueron ejecutadas otras secciones en el contexto de una sección de la aplicación.

El tiempo inclusivo se refiere al tiempo total invertido por la aplicación para una sección de la misma incluyendo los tiempo de las secciones contenidas por esta. El exclusivo, como su nombre lo indica sustrae al tiempo original de la sección, el invertido en las sub-secciones contenidas dentro de ella.



Name △	Inclusive Time	Exclusive Time	Calls	Child Calls
main	2,662.439	9.579	1	2,997
CollectSolution darray (darray, Decomposition, Grid)	2.562	0.246	1	52
CreateArray void (darray, int, int)	0.148	0.148	1	0
DumpError void (darray, darray)	0.668	0.668	1	0
Finalize void (darray, darray, Grid)	0.834	0.056	1	4
Init_darrays void (darray*, darray*, Decomposition, (	0.24	0.072	1	2
Iteration	2,590.468	61.629	2,983	14,915
Exchange void (darray, Decomposition, Grid)	956.296	94.62	5,966	11,932
MPI_Recv()	633.558	633.558	5,966	0
MPI_Send()	228.118	228.118	5,966	0
MPI_Allreduce()	926.325	893.315	2,983	2,983
Sweep double (darray, darray, Decomposition)	646.218	646.218	5,966	0
MPI_Barrier()	1.338	1.338	2	0
MPI_Wtime()	0.07	0.07	2	0
Startup int (int, char**)	55.64	5.65	1	8
MPI_Bcast()	2.791	2.694	1	1
MPI_Cart_coords()	0.061	0.061	1	0
MPI_Cart_create()	0.594	0.483	1	3
MPI_Cart_shift()	0.087	0.087	1	0
MPI_Comm_rank()	0.054	0.054	2	0

Figura 4.10 Resultados de la ejecución para el nodo 0, core 0, thread 0

Para el proceso de instrumentación, TAU permite la instrumentación en varias formas: la primera es a través de la inserción en el código fuente de la aplicación de llamadas a funciones de su librería. Utilizando esta vía, se definen los puntos exactos donde tenemos intención de obtener la medición de nuestros parámetros. Otras de las variantes con las que podemos instrumentar una aplicación utilizando TAU, se basa en el uso de ficheros de instrumentación donde se definen las cabeceras de las funciones que querríamos instrumentar. Una última variante, nos permite a partir de la librería Dyninst, instrumentar aplicaciones donde solo poseemos el binario de la misma.

La capa de análisis y visualización permite el uso de varios módulos. Estos módulos se dividen en componentes para *profiling* y componentes para *tracing*. La generación de la información de *profiling*, en particular, se realiza una vez que se concluye la fase de recopilación de todos los eventos generados en la fase de *tracing* de la aplicación. En particular, para la visualización del historial de ejecución de la aplicación a partir de la información de *tracing* recopilada, las últimas versiones de TAU incluyen la herramienta JumpShot (figura 4.11).

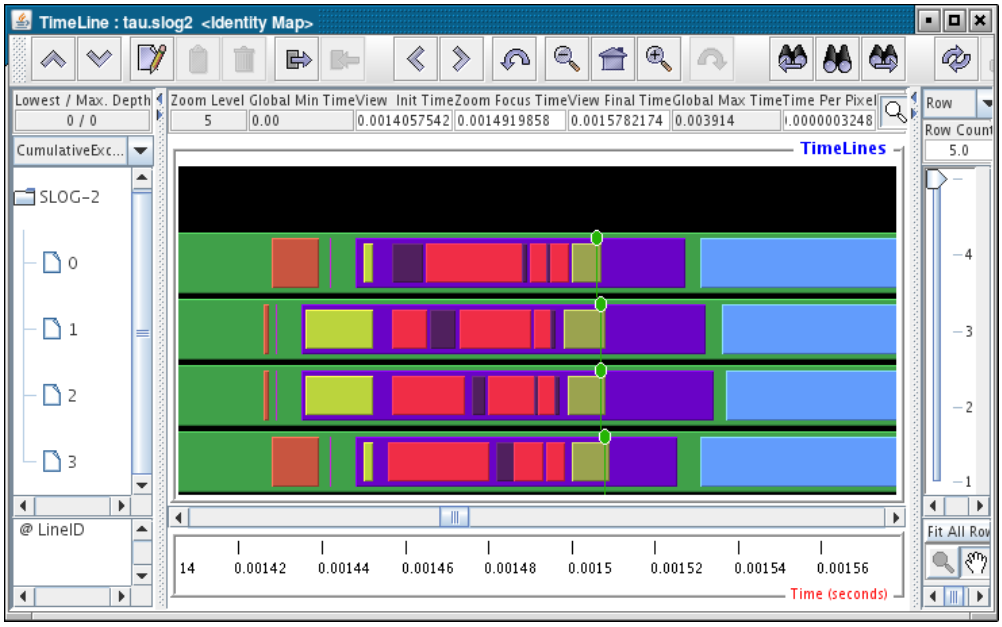


Figura 4.11 Herramienta JumpShop

Otra de las potencialidades de TAU radica en la posibilidad de agregar, a los resultados obtenidos en la fase de *profiling*, el comportamiento de la medición de múltiples contadores de *hardware*. Para permitir esta posibilidad, TAU debe ser instalado configurando previamente la integración con la librería PAPI habilitada para ello. Gracias a esta potencialidad, podemos obtener para cada sección de la aplicación y cada hilo de las secciones paralelas, como se comportan estos contadores por ejemplo: los fallos de cache de L1 o L2, la cantidad de instrucciones ejecutadas, la cantidad de ciclos invertidos, etc.

Dentro de las herramientas más interesantes incluidas en la distribución de TAU, podemos encontrar a *PerfExplorer*. Esta aplicación funciona como un pequeño repositorio de resultados experimentales. Para ello, es necesario ir agregando los resultados obtenidos en los múltiples experimentos y etiquetarlos con un identificador para diferenciarlos del resto. A su vez, la herramienta provee la definición de una jerarquía para agrupar experimentos y un mecanismo para generar métricas adicionales a partir de expresiones matemáticas definidas por el usuario. A partir de este punto, la herramienta nos permite generar gráficas de manera

automática de speedup por eventos (ver figura 4.8), eficiencia, tiempo de ejecución o la generación de una gráfica personalizada donde incluiríamos por parámetros de nuestro interés.

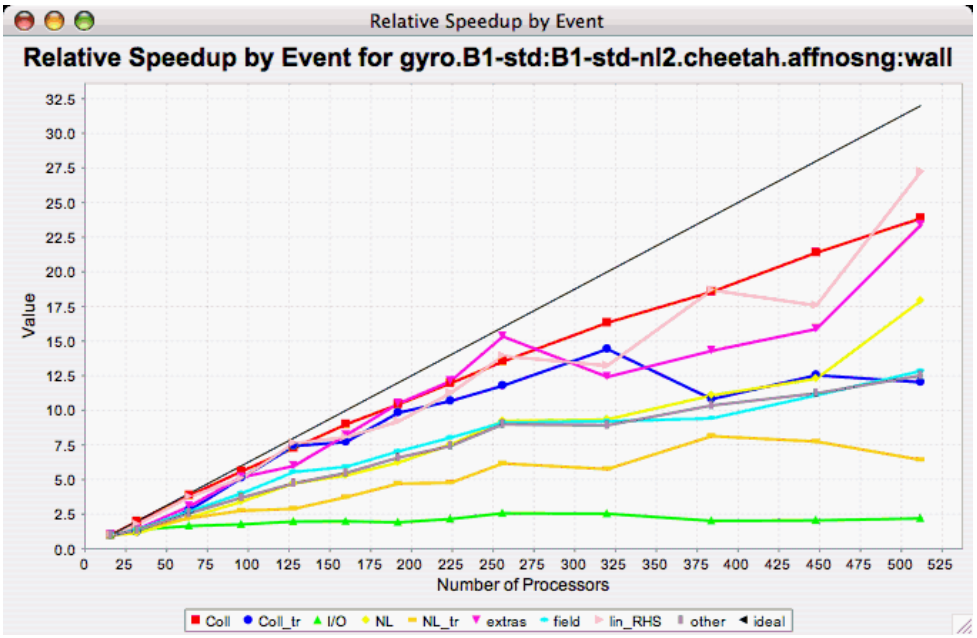


Figura 4.12 PerfExplorer: Gráficas de speedup por eventos utilizando

Por último, a partir de la recopilación de un volumen lo suficientemente significativo de resultado, posee pruebas que permiten realizar un análisis de correlación entre todos los parámetros que hemos incluido en nuestra fase de medición (Figura 4.9). De esta manera, nos facilita la labor de detección de los parámetros que mayor influencia tienen en el rendimiento final y no inclusión de otros cuya significación sea baja.

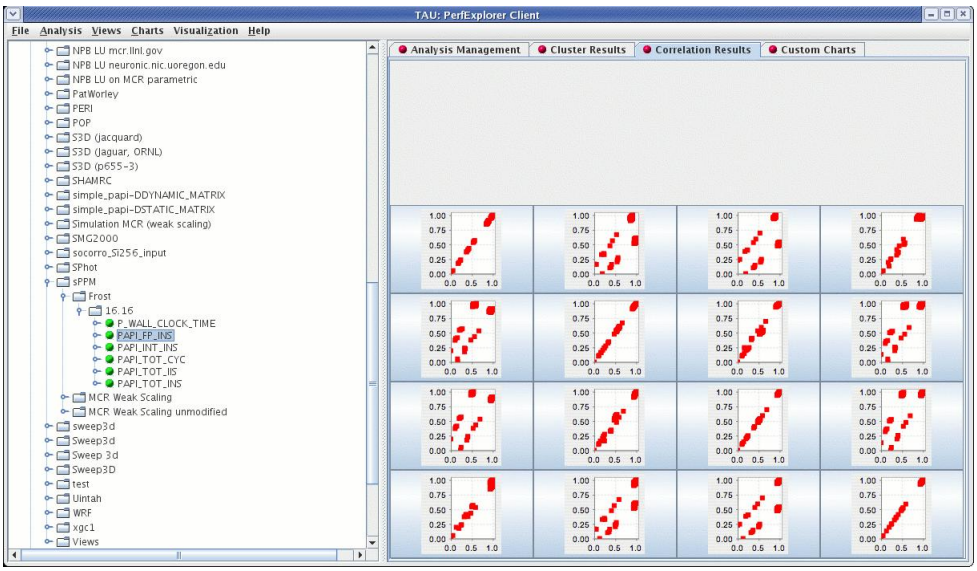


Figura 4.13 PerfExplorer: Análisis de correlación utilizando

Capítulo 5
Análisis de factores de rendimiento

5.1 Modelo de predicción para una fase de la aplicación

A partir de las expresiones generales en [**13**] y [**16**] podemos plantear un modelo general de predicción del tiempo de ejecución para una fase de un proceso en una aplicación híbrida. La misma seria:

$$T_{fase} = T_{\substack{comp \\ serie}} + T_{\substack{comp \\ paralelo}} + T_{\substack{comm \\ collectivas}} + T_{\substack{comm \\ pto-pto}} - \Delta T_{\substack{solap \\ comp-comm}}$$

Ecuación 5.1 Modelo general de predicción de una fase de un proceso

De forma general, y asumiendo que la aplicación está balanceada, el tiempo de ejecución para una fase está compuesto por el tiempo de ejecución de la parte serie de la aplicación híbrida $T_{comp\ serie}$, o sea, la parte que no se encuentra paralelizada con regiones OpenMP, más en tiempo de cómputo de la región paralela definida por $T_{comp\ paralelo}$ adicionándole además el tiempo invertido en las comunicaciones colectivas y punto a punto $T_{comm\ collectivas}$ y $T_{comm\ pto\text{-}pto}$ menos el tiempo en donde las comunicaciones y el cómputo se solaparon $\Delta T_{solap\ comp\text{-}comm}$. En el caso de las aplicaciones híbridas *Masteronly*, por las propias características de estas aplicaciones antes mencionadas, el tiempo de solapamiento cómputo comunicación es cero. Si tenemos en cuenta que, en el capítulo 3 describimos las diferentes estrategias con las que podemos predecir el tiempo invertido por las comunicaciones, solo necesitaríamos un modelo general para predecir el comportamiento de las regiones de cómputo serie y paralelo. El problema se reduce en este caso a encontrar un modelo o mecanismo para la predicción del rendimiento en regiones paralelas ya que una región serie puede ser entendida como un caso particular de región paralela en la que solo tenemos un hilo y no existe un costo agregado en tiempo de ejecución por el *overhead* añadido en la creación, destrucción y sincronización de los hilos utilizados.

Por otra parte, si nuestro mecanismo de predicción de las comunicaciones es lo suficientemente bueno, una vez que la aplicación no esté balanceada, la diferencia entre el resultado de esta expresión y el resultado real medido en una fase de la aplicación no devolvería una diferencia que describiría con bastante aproximación el tiempo desperdiciado por cada proceso provocado por el desbalance entre los procesos MPI involucrados en la ejecución.

Aunque no tengamos aun desarrollado un modelo de predicción para las regiones de cómputo paralelo, una estrategia interesante puede ser instrumentar la aplicación para obtener estos tiempos y a través de un mecanismo de sintonización dinámica modificar los parámetros de la aplicación con el fin de mejorar su rendimiento.

Esta vía adicionalmente, nos permite enfrentar la estimación de rendimiento en el caso de las aplicaciones híbridas *Masteronly-single overlap*. En las mismas, como se ha mencionado anteriormente, podemos encontrar solapamiento entre el cómputo y las comunicaciones. De igual forma, podríamos medir el cómputo solapado y, en los casos donde el volumen de envío del mensaje sea pequeño, la expresión debería darnos un buen criterio para evaluar el desbalance.

Es necesario explicar que, tal y como está implementado el mecanismo de comunicación en la librería MPI, una comunicación no bloqueante donde el tamaño del mensaje a enviar en los suficientemente pequeño no afecta para nada la región de cómputo que se solaparía con esta comunicación. MPI internamente, tiene definido un tamaño de buffer tope para los mensajes, si los mimos no exceden dicho tamaño, el mensaje es enviado directamente al buffer de la tarjeta de red y la ejecución continuaría automáticamente. En el caso en que los mensajes excedan dicho buffer, la librería MPI utilizaría un hilo para ir enviado los fragmentos del mensaje a la tarjeta de red por lo qué, en caso de solaparse esta fase de comunicación con una fase de cómputo paralelo de la aplicación, el hilo creado por la librería y los utilizados en la región de cómputo de la aplicación, estarían compitiendo por el uso de los *cores* y la memoria cache. El efecto que provocaría este comportamiento aun no se ha evaluado pero lo lógico es que ambos se afecten por ello.

A partir de estos análisis, la estrategia a seguir en etapas futuras, pasa por realizar un análisis con el fin de integrar este modelo genérico, a los modelos ya existentes para las aplicaciones de paso de mensajes llegando así a extender el modelo para predecir el tiempo para una fase de la aplicación híbrida o al menos, un modelo que permita sintonizar factores de rendimiento, ya identificados previamente en aplicaciones MPI, en las aplicaciones híbridas.

5.2 Aplicaciones seleccionadas

Con el objetivo de identificar posibles factores de rendimiento en las aplicaciones híbridas se seleccionaron tres aplicaciones como fuente para realizar las experimentaciones. La primera aplicación seleccionada fue el NAS-MZ (*Multizone*). Su elección se debe fundamentalmente a su amplia utilización dentro de la comunidad científica pero también, por ser una implementación híbrida de las versiones originales y porque posee dos tipos de aplicaciones: una donde la carga de cómputo esta balanceada y en el segundo la carga presenta un desbalance a partir de un patrón fijo. Al ser una aplicación con varios años de desarrollo previo en sus versiones MPI y *OpenMP*, los desarrolladores garantizan una buena paralelización en las secciones *OpenMP* por lo que la versión final híbrida, explota de manera bastante eficiente las bondades del paralelismo en los dos niveles.

La segunda aplicación utilizada es un *benchmark* que implementa el método de Jacobi para la solución de problemas. La versión original de esta aplicación posee tres variantes de implementación MPI, Híbrida y secuencial. Fue escogida ya que al analizar el código se detecto que el paralelismo en las regiones *OpenMP* era bastante elemental y, dada la naturaleza del algoritmo, no ofrecía ventajas en su versión híbrida con relación a la versión MPI.

En el tercer caso, la aplicación seleccionada fue el *becnhmark* smg2000. Su elección de debió fundamentalmente, al patrón de comunicación que posee dicha aplicación. El mismo es sigue un comportamiento bastante complejo y sobre todo, la aplicación genera una cantidad elevada de mensajes durante su tiempo de ejecución.

En los tres casos, estas aplicaciones fueron desarrolladas siguiendo el paradigma de aplicaciones SPMD [**10**].

5.2.1 NAS-MZ Bechmark

El NAS-MZ en su versión 3.3.1 [**41**] [**42**] es una continuación de los *bechmarks* desarrollados por el departamento de la NASA para la evaluación el rendimiento antes diferentes arquitecturas. Esta versión en particular, está desarrollada para explotar los dos niveles de paralelismo: el paralelismo entre procesos a través de las comunicaciones MPI y el paralelismo de *threads* con la utilización de regiones *OpenMP*. El paquete, está compuesto por tres aplicaciones LU, BT y SP donde cada una utiliza un mecanismo de solución diferente para la resolución de ecuaciones diferenciales. En los casos de la versión BT y SP el problema es subdivido en zonas tridimensionales y la cantidad de las mismas depende del tamaño del problema. La figura 5.1 muestra como se realiza el particionamiento del problema.

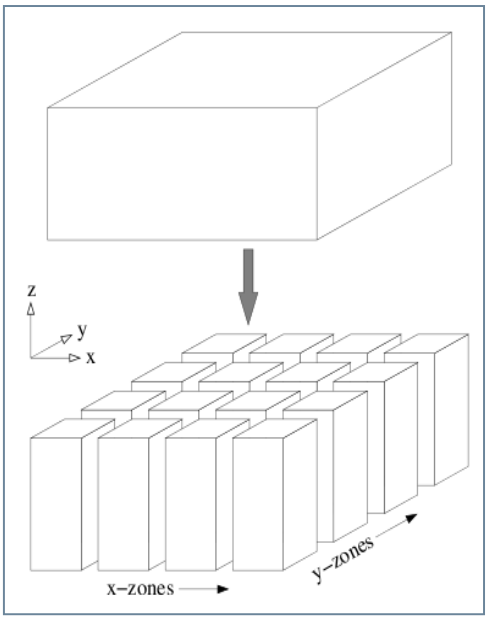


Figura 5.1 Subdivisión del problema en bloques 3D NAS-MZ

Además de las diferencias en cuanto al método de solución empleado por la versión SP y la BT, otra diferencia importante entre ellos radica en cómo se comporta la distribución de la carga de cómputo para cada zona. En el caso de la versión SP, la distribución se realiza de forma más o menos balanceada entre cada zona logrando con ello un balance del cómputo entre los diferentes procesos involucrado. Sin embargo, la versión BT intenta comportarse de una forma más realista, por lo que la carga se distribuye irregularmente entre las diferentes zonas. Con ello los desarrolladores pretenden generar un comportamiento similar al de las aplicaciones reales en donde el balance natural de la carga de cómputo no se garantiza (ver figura 5.2).

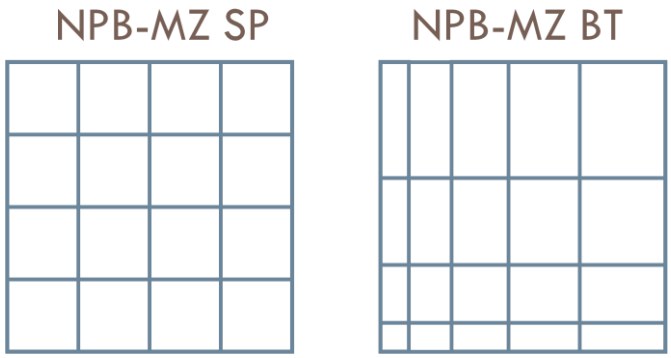


Figura 5.2 Distribución de la carga de cómputo para los benchmarks SP y BT

En cada fase de la aplicación se realizan intercambios entre las fronteras de las zonas. La figura 5.3 muestra el comportamiento de este intercambio para una zona ubicada en la esquina inferior izquierda del particionamiento realizado. Las comunicaciones se realizan entre las zonas aledañas, intercambiándose los valores de los extremos de la zona a sus vecinos en la

forma que se explica en la figura. De esta manera, las zonas en las esquinas se comunicarían con 2 de sus zonas vecinas mientras que las zonas ubicadas en el centro del particionamiento que comunicarían con cuatro zonas vecinas.

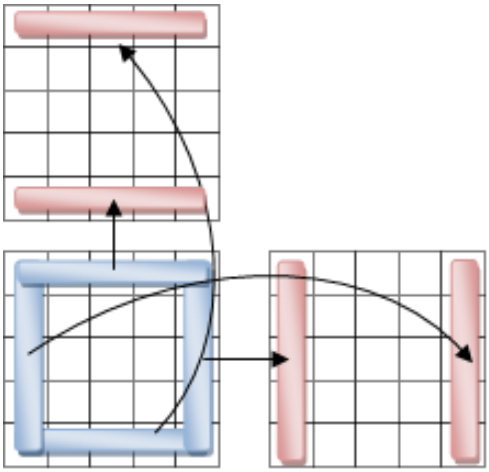


Figura 5.3 Patrón de transferencia de datos entre las zonas (NAS-MZ)

Ambas aplicaciones, debido a la manera en que fueron implementadas, tienen restricciones muy fuertes en cuanto a los parámetros de ejecución de las mismas. Esta particularidad, limita las posibles variantes en que pueden ser ejecutadas a una cantidad más reducida, limitada además por los recursos de cómputo utilizados en nuestro departamento. Por ejemplo la versión BT solo permite ejecutar la aplicación siempre que la cantidad de procesos *MPI* sea el cuadrado de un entero y a su vez, en ambos casos las dimensiones del problema deben ser divisibles por la cantidad de procesos utilizados.

Para la realización de los experimentos, las siguientes tablas muestran las configuraciones de ejecución para el *benchmark* SP. En este caso todos los experimentos se realizaron manteniendo constante la dimensión de los datos utilizados para realizar el cómputo de la solución final.

Para la obtención de la versión MPI pura de este *benchmarks* fue necesario modificar la aplicación para eliminar los pragmas *OpenMP* del su código original. Usualmente no es necesario realizar esta operación ya que un simple cambio en los parámetros de compilación permite no considerar estos pragmas, pero como la aplicación fue desarrollada en Fortran, el compilador de mismo no brinda la posibilidad de compilar desestimando estos pragmas.

NODOS	PROCESOS MPI	CLASE	DIMENSIONES DE LOS DATOS
1	4	C	480 x 320 x 28
4	16		
8	32		
16	64		

Tabla 5.1 Parámetros de ejecución para NAS-MZ SP clase C MPI puro

NODOS	PROCESOS MPI	THREADS	THREADS POR PROCESO	CLASE	DIMENSIONES DE LOS DATOS
1	1	4	4	C	480 x 320 x 28
4	4	16			
8	8	32			
16	16	64			

Tabla 5.2 Parámetros de ejecución para NAS-MZ SP clase C Híbrido

NODOS	PROCESOS MPI	CLASE	DIMENSIONES DELOS DATOS
1	4	C	480 x 320 x 28
4	16		
9	36		
16	64		
25	100		

Tabla 5.3 Parámetros de ejecución para NAS-MZ BT clase C MPI puro

NODOS	PROCESOS MPI	THREADS	THREADS POR PROCESO	CLASE	DIMENSIONES DE LOS DATOS
1	1	4	4	C	480 x 320 x 28
4	4	16			
9	9	36			
16	16	64			
25	100	400			

Tabla 5.4 Parámetros de ejecución para NAS-MZ BT clase C Híbrido

5.2.2 **Jacobi**

El *bechmark* de Jacobi seleccionado, en su versión híbrida utiliza una estructura en forma de matriz para almacenar los datos que utilizará en el proceso de cómputo de la solución. Esta aplicación a su vez, tiene dos parámetros de configuración que hemos dejado fijos: el tamaño de la matriz a procesar, en nuestro caso 20000 por 20000 y la cantidad de iteraciones del algoritmo. El patrón de comunicación utilizado por la aplicación se muestra en la figura 5.4:

Figura 5.4 Patrón de comunicación del bechmark Jacobi para 5 procesos MPI

Como se puede ver, cada proceso MPI se comunica con sus dos vecinos adyacentes salvo los procesos ubicados en las esquina, en este caso el proceso 0 y el proceso mayor que solo se comunican con un solo vecino. En esta comunicación se intercambian las filas de los extremos de la fracción que utiliza cada proceso para su cómputo y en una segunda etapa de la comunicación, se realiza una comunicación colectiva en la cual se distribuye entre todos los nodos el valor del error global de cálculo de la solución.

Para realizar los diferentes experimentos, se generaron cuatro versiones de la aplicación. Unas compiladas sin la inclusión regiones paralelizadas con pragmas *OpenMP* y la otra incluyéndolos. A su vez, se generó dos versiones adicionales de la aplicación en la cual se adicionó un código que genera, para la cantidad de procesos MPI utilizados en la ejecución, una desbalance de la carga de cómputo definido por la cantidad de filas que son entregadas a cada proceso para su trabajo. La figura 5.5 muestra las diferencias en la forma de repartición de las filas entre la versión balanceada del algoritmo y la versión desbalanceada.

Figura 5.5 Desbalance de cómputo generado al modificar la lógica del Jacobi

El la versión original las cantidad de filas que son asignadas a cada proceso se calcula a partir de dividir la cantidad total entre el número de procesos involucrados en el cómputo. En caso de que la cantidad total de filas no sea divisible por la cantidad de procesos, el resto de la división sería asignado al último de los procesos. En la versión desbalanceada, la cantidad de filas asignadas dependerá del número del proceso en cuestión de manera que, al aumentar dicho número, la cantidad de filas asignadas aumentará en igual proporción.

Para el cálculo de los nuevos valores de la matriz en cada fase de la aplicación, este *bechmark* se comporta según se muestra en la figura 5.6:

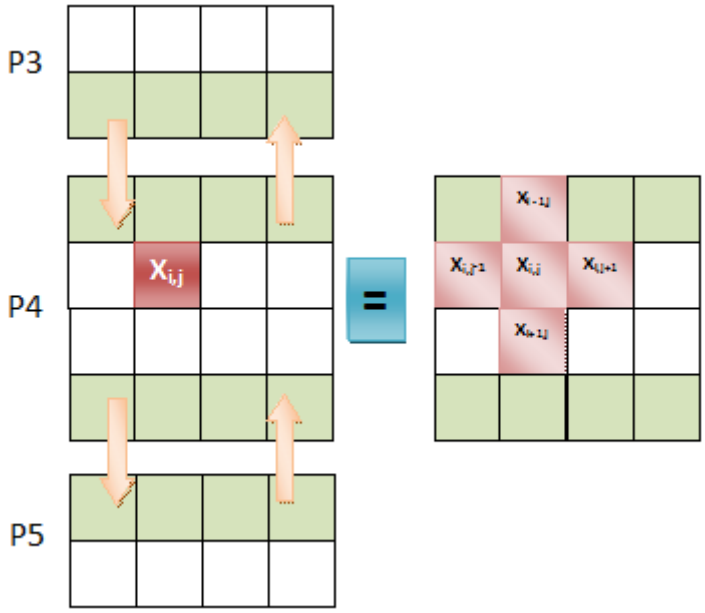


Figura 5.6 Funcionamiento del algoritmo Jacobi

En cada fase, los procesos vecinos intercambian sus filas adyacentes. Una vez realizada esta tarea en el nuevo valor de cada celda de la matriz es calculado a partir de los antiguos valores de sus vecinos y del suyo propio. Es importante hacer notar que, existe una dependencia muy fuerte de datos en el proceso de cálculo de la nueva solución ya que es necesario conservar los antiguos valores de la matriz para el cálculo completo de los nuevos valores. Esto obligó a los desarrolladores de esta aplicación a tener una copia de la matriz donde se almacenan los valores resultantes de las celdas obtenidos en la iteración anterior. Esta estrategia es conocida en la literatura como *replicación de datos* [**43**] y por lo general, no suele ofrecer mejoras notables en el rendimiento de la versión híbrida de la solución con su versión MPI pura. Bajo este patrón de acceso a los datos, en el caso que se utilicen matrices de dimensiones considerables, provoca fallos de acceso a memoria cache de manera reiterada para los accesos a elementos vecinos que no se encuentran en la misma fila del elemento a ser calculado por lo que, la utilización de varios hilos para el cálculo de los nuevos valores de las celdas. No debería generar un beneficio significativo en el rendimiento de la versión híbrida de la aplicación

La tabla 5.5 muestra la los diferentes parámetros de ejecución utilizados para los experimentos. En este caso, los experimentos fueron repetidos utilizando las versiones balanceadas y desbalanceadas de las aplicaciones MPI e híbrida respectivamente.

Se puede observar que en todos los casos, la cantidad de procesos MPI utilizados en la versión de paso de mensajes coincide con el resultado de multiplicar la cantidad de procesos utilizados

en las ejecuciones híbridas por la cantidad de hilos empleados en las secciones paralelas *OpenMP*.

MPI		HIBRIDO				DIMENSIONES DE LOS DATOS	ITER.
NODOS	PROCESOS	NODOS	PROCESOS	THREADS	THREADS x PROC		
3	12	3	3	12	4	20000 X 20000	1000
4	16	4	4	16			
5	20	5	5	20			
6	24	6	6	24			
7	28	7	7	28			
8	32	8	8	32			
9	36	9	9	36			
10	40	10	10	40			
11	44	11	11	44			
12	48	12	12	48			
13	52	13	13	52			
14	56	14	14	56			
15	60	15	15	60			
16	64	16	16	64			
17	68	17	17	68			
18	72	18	18	72			
19	76	19	19	76			
20	80	20	20	80			
21	84	21	21	84			
22	88	22	22	88			
23	92	23	23	92			
24	96	24	24	96			
25	100	25	25	100			
26	104	26	26	104			
27	108	27	27	108			
28	112	28	28	112			

Tabla 5.5 Parámetros de ejecución (Híbrido y MPI) para los experimentos utilizando Jacobi

5.2.3 SMG2000

Esta aplicación [**44**] [**45**] es un implementa un algoritmo de solución de sistema lineales derivados de el método de diferencias finitas. El código resuelve un problema bidimensional o tridimensional dependiendo de los parámetros de ejecución de la aplicación.

En su última versión el código permite ser compilado para generar una versión MPI y también permite la generación de una versión híbrida de la solución.

La aplicación posee un patrón de comunicación bastante complicado y el mismo depende no solo de la dimensión del problema, sino también de la cantidad de procesos involucrados. Es

por ello que resulta bastante engorroso obtener el patrón de comportamiento global solo con mirar el resultado de los eventos de ejecución. Para agilizar esta tarea, nos auxiliamos de la herramienta pas2p [**46**] desarrollada por otro grupo de investigación dentro nuestro departamento. La figura 5.7 muestra el resultado obtenido por la herramienta pas2p donde se describe el comportamiento de las comunicaciones para un problema de tres dimensiones, 8 procesos MPI involucrados en la ejecución, utilizando el solver SMG como método de solución.

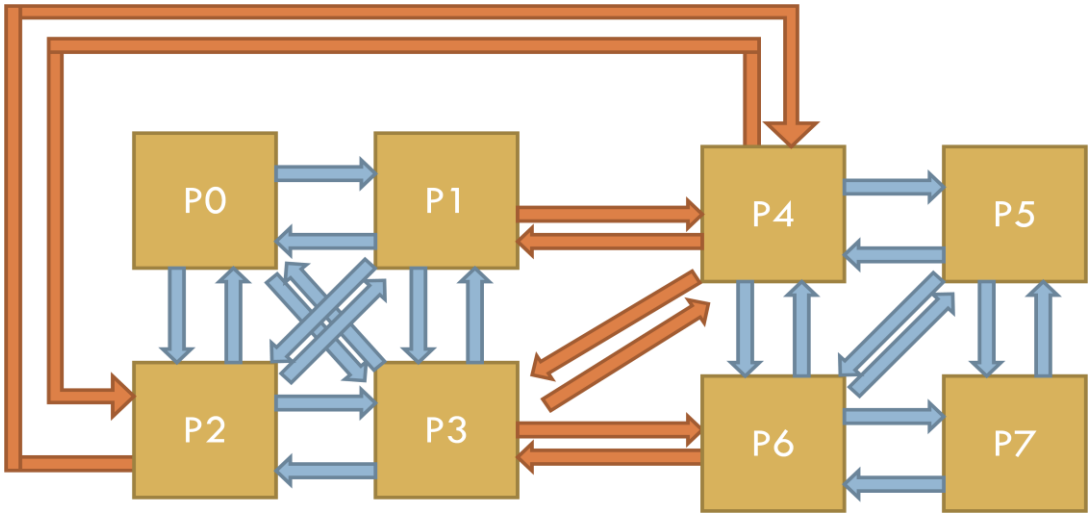


Figura 5.7 Patrón de comunicación del smg2000 MPI puro (8 procesos).

Las líneas rojas, representan las comunicaciones que se realizan a través de la red de interconexión entre los nodos de nuestro clúster mientras que, las azules representan las comunicaciones realizadas por MPI usando la memoria compartida entre los procesos ya que los mismos se encuentran en el mismo nodo. Como se puede ver, las comunicaciones inter-nodo en este caso, son unas 8 para 8 nodos.

Esta misma aplicación, ejecutada de manera híbrida, tendría un patrón de comunicación mucho más sencillo. La figura 5.8 muestra como sería el mismo. Como puede verse, ahora las comunicaciones inter-nodo son solo dos y los *cores* de cada nodo, son utilizados ahora por cada hilo respectivamente en el cómputo paralelo de las regiones OpenMP.

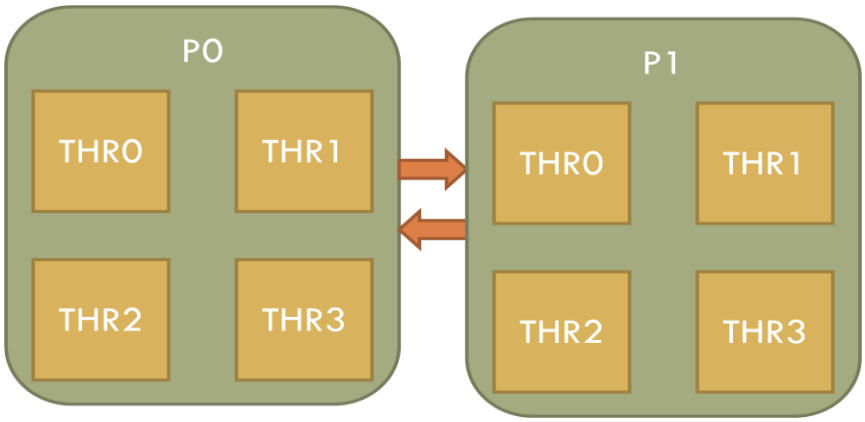


Figura 5.8 Patrón de comunicación del smg2000 Híbrido (2 procesos, 4 hilos por proceso)

En los experimentos que se realizarán se trata de evaluar si se produce un beneficio significativo con esta reducción en la cantidad de comunicación realizada. Para ello, las tablas 5.6 y 5.7 muestran las configuraciones con las que han sido ejecutados los experimentos para comparar el rendimiento de ambas versiones de la aplicación. Es importante hacer notar que, con el fin de mantener el tamaño del problema igual para ambas versiones, se ha aumentado por 4 el volumen de los datos de cómputo en la versión híbrida. Para ello, se aumento el doble en la dimensión PX y PY. El volumen de datos para el cómputo está definido por la expresión (Nx * Px * Ny *Py * Nz *Pz).

NX x NY x NZ	PX x PY x PZ	BX x BY x BZ	TAMAÑO DE LOS DATOS	DIMENSIONES	Solver
2 x 4 x 4	100 x 100 x 100	1 x 1 x 1	32000000	3	0 (SMG)
3 x 4 x 4	100 x 100 x 100	1 x 1 x 1	48000000		
4 x 4 x 4	100 x 100 x 100	1 x 1 x 1	64000000		
5 x 4 x 4	100 x 100 x 100	1 x 1 x 1	80000000		
6 x 4 x 4	100 x 100 x 100	1 x 1 x 1	96000000		

Tabla 5.6 Parametros de ejecución para smg2000 version MPI pura

NX x NY x NZ	PX x PY x PZ	BX x BY x BZ	TAMAÑO DE LOS DATOS	DIMENSIONES	Solver
2 x 2 x 2	200 x 200 x 100	1 x 1 x 1	32000000	3	0 (SMG)
3 x 2 x 2	200 x 200 x 100	1 x 1 x 1	48000000		
4 x 2 x 2	200 x 200 x 100	1 x 1 x 1	64000000		
5 x2 x 2	200 x 200 x 100	1 x 1 x 1	80000000		
6 x 2 x 2	200 x 200 x 100	1 x 1 x 1	96000000		

Tabla 5.7 Parámetros de ejecución para smg2000 versión híbrida

5.3 Resultados experimentales

Para permitir la ejecución de las aplicaciones, en su versión híbrida y MPI, se lanzaron las mismas utilizando los parámetros de ejecución del comando *mpirun*. En la tabla 5.5 se muestra las configuraciones para cada caso. La configuración de los parámetros de ejecución está realizada para que, en ambos casos la cantidad de recursos de cómputo utilizados coincidan. Por ejemplo, para uno aplicación MPI que utilice 8 procesos en su ejecución, cada uno mapeado a cada *core* de los dos nodos utilizados, una de las configuración equivalente en su versión híbrida tendría que utilizar solamente 2 procesos MPI en dos nodos y 4 hilos en las secciones de cómputo paralelo.

MPI	**mpirun –np** *#procesos* **–npernode** *4*	Se ejecutan 4 procesos uno en cada core del nodo. La cantidad total de los procesos utilizados debe ser divisible por 4.
Híbrida	**export** *OMP_NUM_THREAD*=4 **mpirun** –np (#procesos/4) –npernode 1	Se define la variable de entorno utilizada por OpenMP para que utilice 4 hilos en las regiones paralelas. Se ejecutará un proceso MPI por cada nodo del clúster. La cantidad de procesos en este caso debe ser ¼ de los utilizados para la ejecución de la versión MPI de la aplicación

Tabla 5.8 Configuración de la ejecución de las aplicaciones MPI e Híbrida

A su vez, la tabla 5.6 muestra las características fundamentales del entorno de ejecución utilizado en todos los experimentos realizados.

Cluster IBM	
Front-End Node	IBM x3650 Dual-Core Intel(R) Xeon(R) CPU 5150 @2.66GHz 4MB L2 (2x2) 8 GB Fully Buffered DIMM 667 MHz
32 IBM x3550 Nodes	2 x Dual-Core Intel(R) Xeon(R) CPU 5160 @ 3.00GHz 4MB L2 (2x2) 32KB L1 (datos) 32KB L1 (inst.) 12 GB Fully Buffered DIMM 667 MHz Hot-swap SAS Controller 160GB SATA Disk
Red	Integrated dual Gigabit Ethernet. Red en estrella
Versiones del compilador y librería MPI	OpenMPI versión 1.4.3 Compilador gcc versión 4.5.2

Tabla 5.9 Características del entorno de ejecución de los experimentos

5.3.1 Calidad del paralelismo en la sección OpenMP

La figura 5.9 muestra el resultado de la ejecución de la aplicación de Jacobi en su versión MPI pura contrastada con la versión híbrida. Como era esperado, se observa que el rendimiento de la versión híbrida se comporta para todos los casos peor que la versión MPI. Esto se debe, a que la paralelización de las regiones de cómputo no ofrece ganancias en cuento al tiempo de ejecución. De hecho, la diferencia que observamos está dada por la penalización que agrega *OpenMP* en la creación y destrucción de los cuatros hilos utilizados en el cómputo. El empeoramiento del rendimiento por este motivo, llega a ser de hasta un 22% del tiempo de ejecución de la versión MPI.

Una buena paralelización en las regiones *OpenMP* implica no solo programar teniendo en cuenta las especificidades de la arquitectura que estamos utilizando, también obliga a modificar la lógica de nuestra aplicación con el fin de beneficiarnos lo más posible con la localidad espacial y temporal de los datos. Si nuestra versión híbrida no consigue este objetivo, el resultado que obtendremos será por lo general siempre peor que la versión MPI. Lamentablemente, conseguir este mejor comportamiento en las aplicaciones híbridas no es una labor trivial, la misma exige un esfuerzo marcado por parte de los programadores de las aplicaciones paralelas y por lo general, la aplicación pasa por diferentes fases de optimizaciones hasta llegar a la versión final.

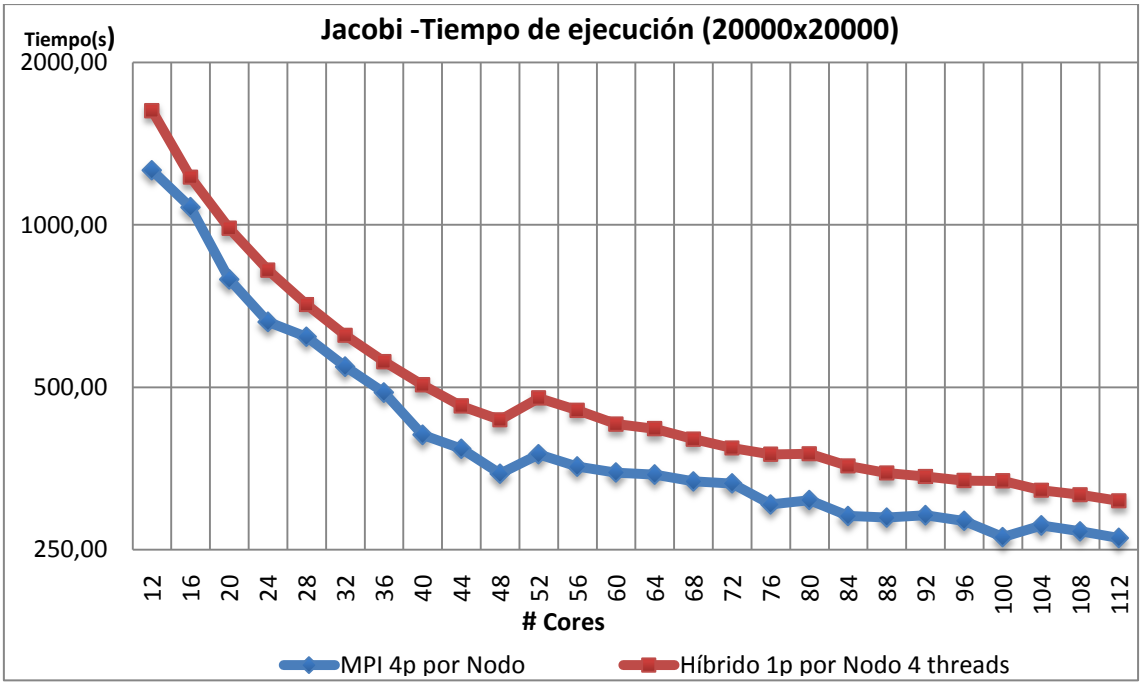


Figura 5.9 Comportamiento de Jacobi balanceado (MPI vs Híbrido)

5.3.2 Propiedad natural de las aplicaciones híbridas para balancear la carga de cómputo

Analicemos ahora, cuan sensibles son las aplicaciones MPI puras en contraposición con las aplicaciones híbridas, cuando existe un marcado desbalance en la carga de cómputo. La figura 5.10 nos muestra el resultado de la ejecución del *bechmark* de NAS SP en su versión híbrida en comparación con la versión MPI.

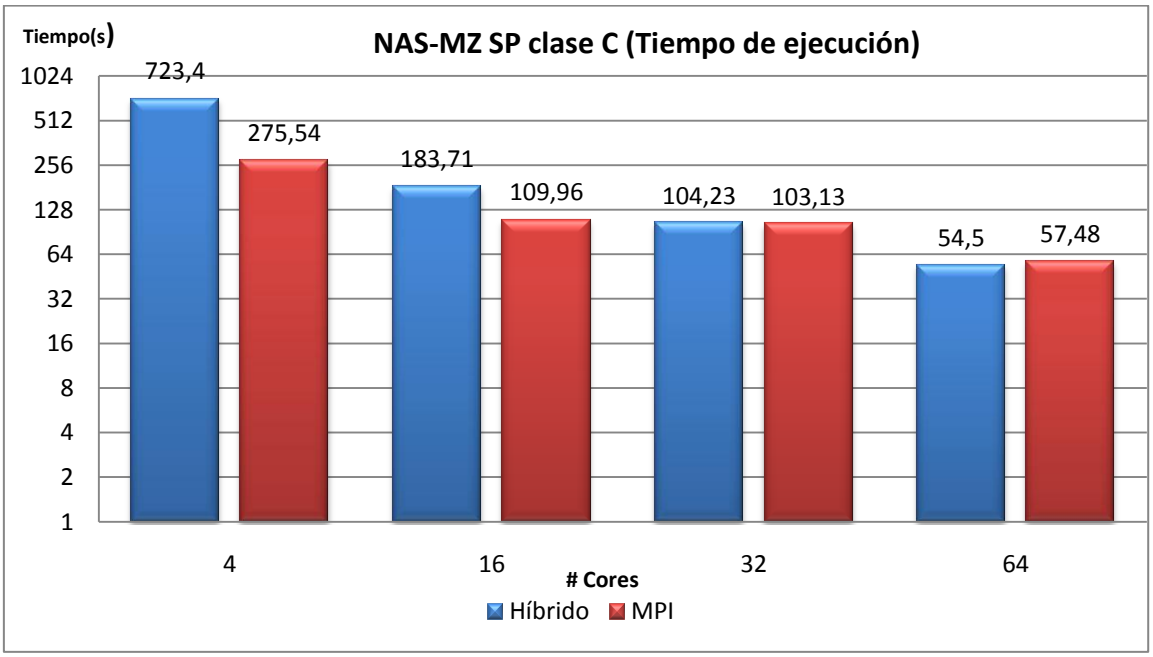


Figura 5.10 NAS-MZ SP Tiempo de ejecución MPI vs Híbrida

En este caso, la buena calidad del paralelismo implementado en las secciones *OpenMP* influye en que el resultado global de la versión híbrida se comporte similar a la versión MPI. Solo para una cantidad pequeña de *cores*, el rendimiento de la aplicación en su versión híbrida se comporta notablemente peor. Al utilizar la versión SP de esta aplicación, garantizamos que se distribuya de manera más o menos semejante la carga de cómputo entre los diferentes procesos MPI involucrados ambas versiones.

Veamos ahora el resultado para la versión BT de la aplicación. Como ya hemos mencionados en secciones previas, esta versión se distingue por generar un desbalance de la carga de cómputo entre los procesos involucrados. La figura 5.11 nos muestra los resultados de la ejecución con la utilización desde cuatro y hasta cien *cores* para ambas versiones de la aplicación.

En este caso se observan algunos cambios en la relación de rendimiento entre ambas versiones de la aplicación. La versión híbrida de la aplicación se comparta mejor a medida que se aumenta la cantidad de *cores* utilizados por la aplicación. Dicha mejora llega a ser de hasta hasta un 13% gracias al mejor balance de cómputo realizado en las regiones *OpenMP*.

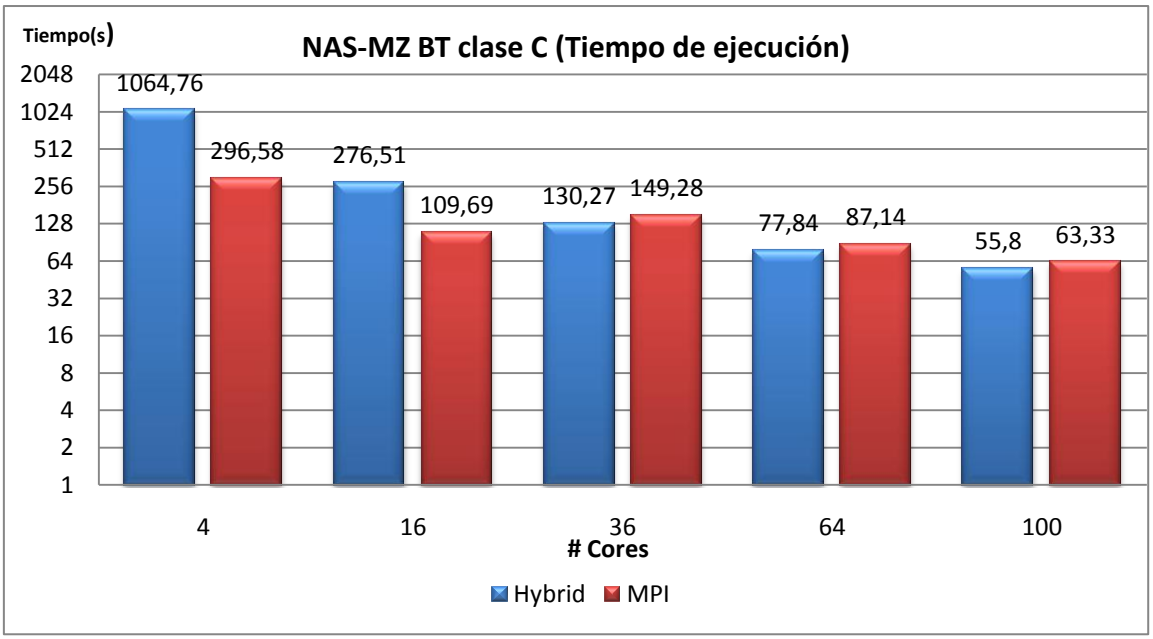


Figura 5.11 NAS MZ BT Tiempo de ejecución MPI vs Híbrida

Un resultado similar podemos encontrar en la gráfica 5.12. En la misma podemos observar como al generar un desbalance marcado en la aplicación, la versión híbrida se comporta mejor en su rendimiento ya que el mejor balance del cómputo generado por las regiones *OpenMP*, tiende a reducir las diferencias entre los tiempos de ejecución de los diferentes procesos *MPI* involucrados.

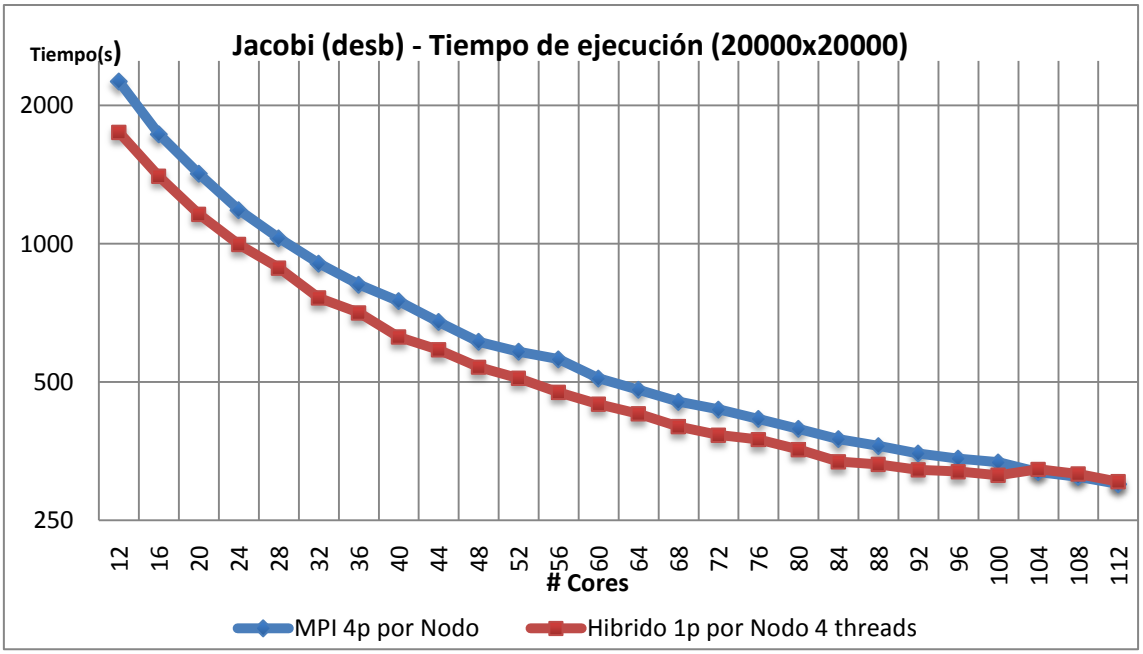


Figura 5.12 Comportamiento de Jacobi (MPI vs Híbrido) con desbalance en la carga de cómputo

Mirando detalladamente la relación cómputo comunicación para cada versión de la aplicación, en el experimento donde son utilizados 68 *cores*, la gráficas 5.13 y 5.14 nos demuestran como en ambos casos el cómputo va en aumento a medida que el número del proceso lo hace también. Este comportamiento es esperado dada la forma en que fue generado el desbalance de carga en estos experimentos. Los tiempos de cómputo y comunicación mostrados en ambas gráficas, describen el tiempo invertido en estas fases para cada uno de los 68 procesos, en el caso de la aplicación MPI pura, y 18 procesos en la aplicación híbrida.

Un detalle interesante en este comportamiento, y que nos permite explicar la mejora el rendimiento de la versión híbrida esta en los valores de media, máximo y mínimo del cómputo y las comunicaciones. En ambos casos, la media del cómputo entre todos los procesos esta cercano a los 143 segundos, siendo un poco mayor (unos 10 segundos en la versión híbrida).

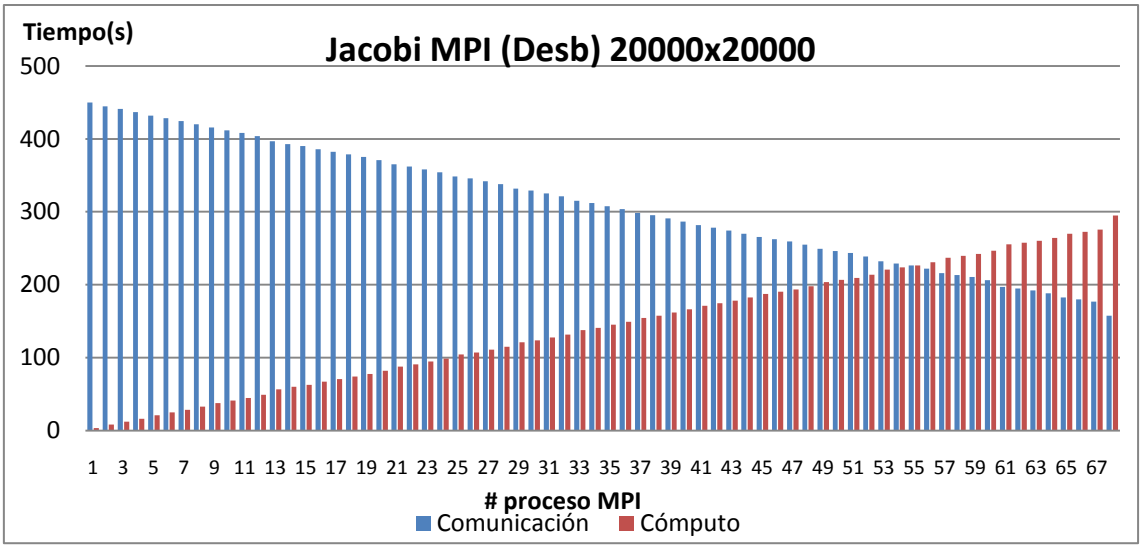


Figura 5.13 Jacobi versión MPI. Tiempo total cómputo vs comunicación por cada proceso (68 cores)

Sin embargo la media de las comunicaciones en el caso de la versión MPI es 309 segundos mientras que la versión híbrida es de 240 segundos. Con solo observar el valor máximo y mínimo de los tiempos invertidos en la comunicación, podemos ver que la versión MPI en máximo ronda los 450 segundos y el mínimo 157 segundos. En el caso de la versión híbrida, el máximo ronda los 375 segundos y el mínimo los 130 segundos. En todos los casos la versión híbrida a logrado no solo disminuir la media sino también el valor mínimo y máximo invertido en las comunicaciones gracias a que la diferencia en el cómputo se acorta mas entre los procesos con relación a la cantidad utilizada de ellos.

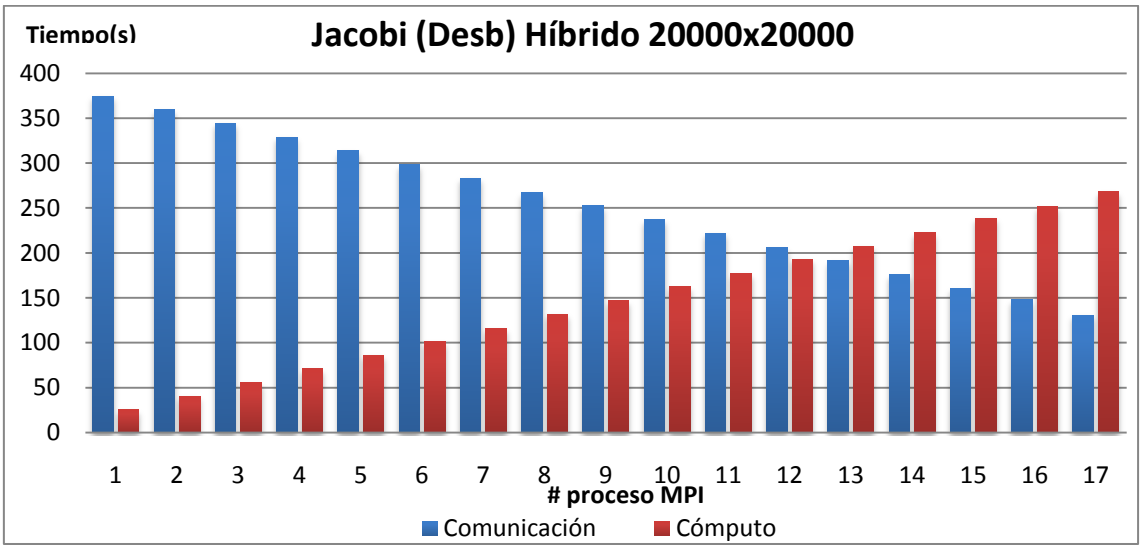


Figura 5.14 Jacobi versión Híbrida. Tiempo total cómputo vs comunicación por cada proceso (68 cores)

5.3.3 Disminución de la cantidad de comunicación global en cada fase

Como un último factor con potencial influencia en el rendimiento de las aplicaciones híbridas contrastadas con sus equivalentes MPI tenemos la posibilidad de que, producto del proceso de conversión a la versión híbrida, la cantidad de comunicaciones inter-nodo disminuyan notablemente. En este experimento (gráfica 5.15), fueron ejecutadas ambas versiones del *bechmark* smg2000 para una carga de trabajo semejante para cada recurso de cómputo.

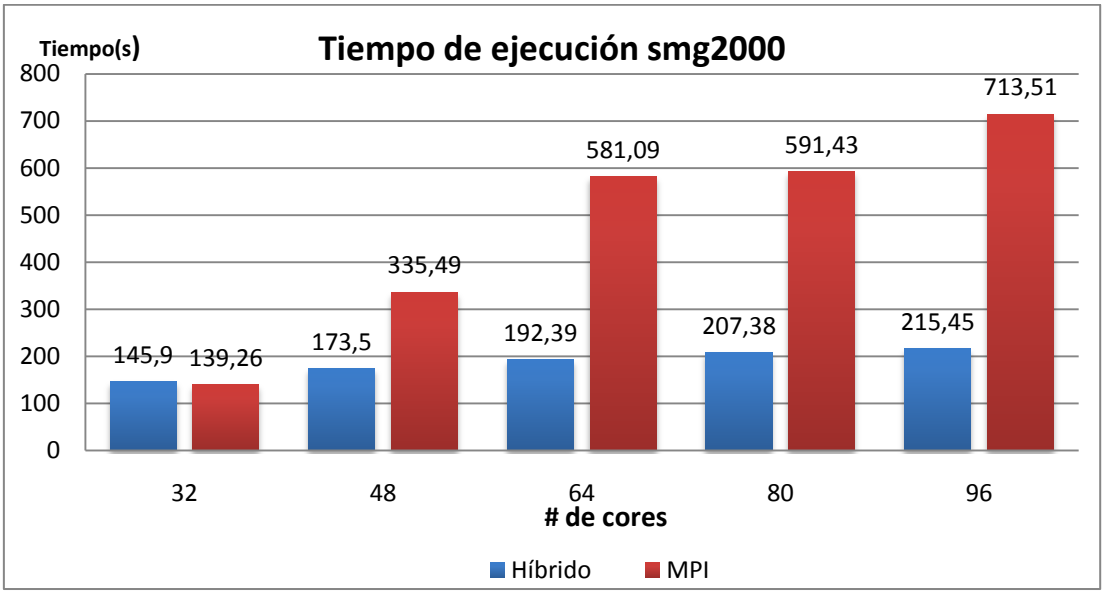


Figura 5.15 Rendimiento del SMG 2000 versión MPI vs Híbrida

En los resultados se puede notar, que la mejoría en el rendimiento es marcada, siendo en el mejor de los casos, de hasta un 70% con relación a la versión MPI pura de la aplicación. Como las diferencias son realmente significativas en los resultados, es comprensible asumir que pueden existir otros factores que influyan en esta mejoría tan marcada en el rendimiento. Por limitantes de tiempo, no fue posible realizar las modificaciones adicionales a la aplicación para medir exactamente, que por ciento de esta mejoría es provocado por la disminución de las comunicaciones. Esta tarea, se ha incluido dentro de los trabajos futuros propuestos al final este trabajo.

Capítulo 6

Conclusiones y trabajos futuros

6.1 Conclusiones

Se ha definido una metodología de trabajo coherente y basada en metodologías propuestas anteriormente, con el fin de detectar, en una primera fase, factores de rendimiento en aplicaciones híbridas y como segunda fase, llegar a un modelo de rendimiento para aplicaciones tipo *Masteronly* y *Masteronly-single overlap*.

También se realizó un estudio de las herramientas de análisis existentes y se seleccionaron a *Scalasca* y *TAU* como las más afines a nuestras necesidades. Para ello se realizó una revisión, por cada aplicación de sus potencialidades y si las mismas cumplían las necesidades planteadas en nuestra metodología.

A partir de la metodología propuesta, se realizó un estudio de las estrategias para la caracterización de las comunicaciones y a la vez, una búsqueda de los *benchmarks* más populares utilizados para esta tarea. Como resultado, se realizó una caracterización de las comunicaciones para la arquitectura del supercomputador utilizado.

Adicionalmente, se presentó una versión inicial, aun sujeta a mejoras, de un modelo de predicción de rendimiento general para una fase de las aplicaciones híbridas. A partir del mismo, se obtienen algunas conclusiones teóricas que serán validadas en trabajos futuros.

Se identificaron tres características propias de las aplicaciones híbridas que influyen en su rendimiento:

- **La eficiencia del paralelismo en las secciones *OpenMP***: Se detecto que una calidad de paralelización pobre en las regiones *OpenMP* no ofrece ventajas en cuento a mejoría del rendimiento de la aplicación híbrida con relación a su equivalente MPI. Esto se debe a que la solución híbrida agrega un *overhead* adicional en la creación y destrucción de los *threads* y adicionalmente, en las secciones secuenciales de la aplicación, solo se utiliza un solo *core*

para el hilo principal mientras que el resto está en desuso. Por lo tanto, si la paralelización no ofrece beneficios significativos para una misma cantidad de recursos de cómputo, el resultado esperado sería la penalización por estos factores en el tiempo de ejecución total.

- **La característica natural que poseen para realizar un mejor balance del cómputo en sistema desbalanceados en cuanto a carga de trabajo**: en este caso de detecto que las aplicaciones híbridas pueden ofrecer un mejor rendimiento que su equivalente MPI puras si el nivel de desbalance en la carga de trabajo es considerable. Puede ser engañoso comparar los rendimientos entre versiones híbridas y MPI si no tenemos en cuenta este factor. Por otro lado, en los casos en donde el costo computacional de balancear la carga de cómputo es elevado, es recomendable escoger una solución híbrida como mecanismo de implementación de la solución, ya que aún en las aplicaciones donde la calidad del paralelismo en las regiones *OpenMP* es bajo, el mejor balance de la carga de la versión híbrida influye en una mejoría del rendimiento global comparada con la versión MPI.

- **La posibilidad de disminuir la cantidad de comunicaciones en cada fase de la aplicación:** hemos encontrado que, al transformar una aplicación MPI a su equivalente híbrida, dependiendo del patrón seguido por las comunicaciones inter-nodo, se puede obtener una mejora del rendimiento. La cantidad de comunicaciones realizadas disminuye, gracias a ello, a la mitad. Para que sea determinante este factor, la cantidad de comunicaciones generadas en una fase de la aplicación, debe ser relativamente alta.

6.2 Trabajos futuros

Como trabajos futuros, queda pendiente el estudio detallado del comportamiento del tiempo de comunicación por cada proceso en los experimentos usando el *benchmark* smg2000.

También se pretende trabajar en la validación del modelo general de predicción de rendimiento para una fase de las aplicaciones híbridas. Para ello, al no tener un modelo de rendimiento para las secciones *OpenMP*, nos proponemos trabajar para integrar el modelo general para una fase propuesto, a los modelos existentes para aplicaciones de paso de mensajes. Una vez conseguido esto, realizaríamos experimentos instrumentando las aplicaciones con el fin de obtener los tiempos de ejecución para las secciones de cómputo y comunicación solapada, y con ello comprobar si los modelos de rendimiento ya existentes para MPI integrados con este modelo preliminar, ajustan de manera certera los parámetros de sintonización ya detectados en las aplicaciones MPI.

En una segunda etapa, pretendemos llegar a definir un modelo analítico completo para predecir el rendimiento de las aplicaciones híbridas enfocados en las *Masteronly* y *Masteronly-single overlap* y como último paso, el refinamiento de dicho modelo para definir estrategias de mejora del rendimiento de forma dinámica a partir de la instrumentación dinámica de la aplicación.

Adicionalmente, se pretende seguir regularmente la evolución de nuevo estándar MPI 3 con el fin de trabajar sobre el desarrollo de modelos de rendimiento para aplicaciones híbridas con solapamiento completo cómputo comunicación. Para ello es necesario que exista una versión suficientemente estable de esta librería de paso de mensajes.

Bibliografía

[1] Top500 Supercomputing Site. [Online]. http://www.top500.com

[2] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes," in *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, 2009, pp. 427-436.

[3] MPI Forum. [Online]. https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/MPI3Hybrid

[4] High Performance Message Passing Library. [Online]. http://www.open-mpi.org/

[5] R. Susukita et al., "Performance prediction of large-scale parallell system and application using macro-level simulation," in *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, 2008, pp. 1-9.

[6] Xingfu Wu and Valerie Taylor, "Performance characteristics of hybrid MPI/OpenMP implementations of NAS parallel benchmarks SP and BT on large-scale multicore supercomputers," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, pp. 56-62, 2011.

[7] Franck Cappello and Daniel Etiemble, "MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks," in *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 2000.

[8] Gabriele Jost and Haoqiang Jin and Dieter An Mey and Ferhat F. Hatay. Comparing the OpenMp, MPI, and Hybrid Programming Paradigms on a SMP Cluster. [Online]. http://nas.nasa.gov/News/Techreports/2003/PDF/nas-03-019.pdf

[9] Géraud Krawezik, "Performance comparison of MPI and three openMP programming styles on shared memory multiprocessors," in *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, 2003, pp. 118-127.

[10] Per Brinch Hansen, "Model programs for computational science: A programming methodology for multicomputers," *Concurrency - Practice and Experience*, vol. 5, no. 5, pp. 407-423, 1993.

[11] K. Nakajima, "Flat MPI vs. Hybrid: Evaluation of Parallel Programming Models for Preconditioned Iterative Solvers on #147;T2K Open Supercomputer #148;," in *Parallel Processing Workshops, 2009. ICPPW '09. International Conference on*, 2009, pp. 73-80.

[12] D.S. Henty, "Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling," in *Supercomputing, ACM/IEEE 2000 Conference*, 2000, p. 10.

[13] L. Adhianto and B. Chapman, "Performance modeling of communication and computation in hybrid MPI and OpenMP applications," in *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 2, 2006, p. 6 pp.

[14] M. Garcia, J. Corbalan, and J. Labarta, "LeWI: A Runtime Balancing Algorithm for Nested Parallelism," in *Parallel Processing, 2009. ICPP '09. International Conference on*, 2009, pp. 526-533.

[15] E. Cesar, A. Moreno, J. Sorribes, and E. Luque, "Modeling Master/Worker applications for automatic performance tuning," *Parallel Computing*, vol. 32, no. 7-8, pp. 568-589, 2006, Algorithmic Skeletons.

[16] Jose Alexander Guevara Quintero, "Definition of a resource management strategy for dynamic performance tuning of complex applications," Departamento de Arquitectura de Computadoras y Sistemas Operativos, Universidad Autónoma de Barcelona, PhD Tesis 2010.

[17] L. M. Liebrock and S. P. Goudy, "Methodology for modelling SPMD hybrid parallel computation," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 8, pp. 903-940, 2008.

[18] Ronal Muresano, Dolores Rexachs, and Emilio Luque, "Methodology for Efficient Execution of SPMD Applications on Multicore Environments," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, 2010, pp. 185-195.

[19] Performance Application Programming Interface. [Online]. http://icl.cs.utk.edu/papi/index.html

[20] Benchweb. [Online]. www.netlib.org/benchweb

[21] J. Bull, James Enright, and Nadia Ameer, "A Microbenchmark Suite for Mixed-Mode OpenMP/MPI," in *Evolving OpenMP in an Age of Extreme Parallelism*, Matthias Müller, Bronis de Supinski, and Barbara Chapman, Eds.: Springer Berlin / Heidelberg, 2009, vol. 5568, pp. 118-131, 10.1007/978-3-642-02303-3_10.

[22] William Gropp and Ewing Lusk, "Reproducible Measurements of MPI Performance Characteristics," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Jack Dongarra, Emilio Luque, and Tomàs Margalef, Eds.: Springer Berlin / Heidelberg, 1999, vol. 1697, pp. 681-681, 10.1007/3-540-48158-3_2.

[23] D. A. Grove and P. D. Coddington, "Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers," *The Journal of Supercomputing*, vol. 34, pp. 201-217, 2005, 10.1007/s11227-005-2340-2.

[24] R. Preissl et al., "Detecting Patterns in MPI Communication Traces," in *Parallel Processing,*

2008. ICPP '08. 37th International Conference on, 2008, pp. 230-237.

[25] Perftest benchmark. [Online]. http://www.mcs.anl.gov/research/projects/mpi/mpptest/

[26] Nor Asilah Wati and Paul Coddington, "Comparison of MPI Benchmark Programs on Shared Memory and Distributed Memory Machines (Point-to-Point Communication)," *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 469-483, 2010.

[27] Werner Augustin, Marc-Oliver Straub, and Thomas Worsch, "Benchmarking One-Sided Communication with SKaMPI 5," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Beniamino Di Martino, Dieter Kranzlmüller, and Jack Dongarra, Eds.: Springer Berlin / Heidelberg, 2005, vol. 3666, pp. 301-308, 10.1007/11557265_40.

[28] Pallas GmbH. Pallas MPI benchmarks documentation. [Online]. http://people.cs.uchicago.edu/~hai/vm1/vcluster/PMB/PMB-MPI1.pdf

[29] The Sphinx Parallel Microbenchmark Suite. [Online]. http://www.llnl.gov/CASC/sphinx/sphinx.html

[30] W. E. Nagel, A. Arnold, M. Weber, H.-Ch. Hoppe, and K. Solchenbach, "VAMPIR: Visualization and Analysis of MPI Resources," *Supercomputer*, vol. 12, pp. 69-80, 1996.

[31] VampirTrace 5.11 User Manual. [Online]. http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/software_werkzeuge_zur_unterstuetzung_von_programmierung_und_optimierung/vampirtrace/dateien/VT-UserManual-5.11.pdf

[32] Vincent Pillet et al., "PARAVER: A Tool to Visualize and Analyze Parallel Code," In WoTUG-18, 1995.

[33] Jesus Labarta, Sergi Girona, and Toni Cortes, "Analyzing scheduling policies using Dimemas," *Parallel Computing*, vol. 23, no. 1-2, pp. 23-34, 1997, Environment and tools for parallel scientific computing.

[34] Karl Fürlinger and Michael Gerndt, "ompP: A profiling tool for OpenMP," in *OpenMP Shared Memory Parallel Programming*, Matthias and Chapman, Barbara and de Supinski, Bronis and Malony, Allen and Voss, Michael Mueller, Ed.: Springer Berlin / Heidelberg, 2008, pp. 15-23.

[35] The OpenMP Profiler ompP: User Guide and Manual. [Online]. http://projekt17.pub.lab.nm.ifi.lmu.de/ompp/usage.pdf

[36] Scalasca Product Site. [Online]. http://www.scalasca.org/

[37] Markus Geimer et al., "The Scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702-719, 2010.

[38] TAU (Tuning and Analysis Utilities). [Online]. http://www.cs.uoregon.edu/research/tau/home.php

[39] Sameer S. Shende and Allen D. Malony, "The Tau Parallel Performance System," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287-311, Summer 2006.

[40] Kevin A. Huck and Allen D. Malony, "PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing," *SC Conference*, vol. 0, p. 41, 2005.

[41] NAS Parallel Benchmark, Multi-Zone Versions. [Online]. http://www.nas.nasa.gov/News/Techreports/2003/PDF/nas-03-010.pdf

[42] Haoqiang Jin and Rob F. Van, "Performance characteristics of the multi-zone NAS parallel benchmarks," *Journal of Parallel and Distributed Computing*, vol. 66, no. 5, pp. 674-685, 2006, IPDPS '04 Special Issue.

[43] Sandip V.Kendre and D.B.Kulkarni, "Optimized Convex Hull With Mixed (MPI and OpenMP) Programming On HPC," *International Journal of Computer Applications*, vol. 1, no. 5, pp. 80-84, 2010, Published By Foundation of Computer Science.

[44] Reza Zamani. Communication Characteristics of Message-Passing Applications, and Impact of RDMA on their Performance. [Online]. http://cupofdenial.xtreemhost.com/reza/files/Reza_Zamani_MScE_thesis.pdf

[45] The SMG2000 Benchmark Code. [Online]. https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/smg/

[46] A. Wong, D. Rexachs, and E. Luque, "Extraction of Parallel Application Signatures for Performance Prediction," in *High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on*, 2010, pp. 223-230.