Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

Estrategias de paralelización de un código de simulación hidráulica de flujos transitorios 2D en volúmenes finitos

Asier Heradio Lacasta Soto

Director: Javier Burguete Tolosa[1]
Ponente: Victor Viñals Yúfera[2]

(1) Departamento Suelo y Agua
Estación Experimental Aula Dei/CSIC

(2) Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior
Universidad de Zaragoza

Curso 2010/2011
Abril 2011

A Papá, Mamá, María(s), José Miguel,
Eva, Celia, Pepe, Tita, Diego y Tati

Agradecimientos

Quiero agradecer, en primer lugar, a Pilar García, Directora del Grupo de Hidráulica Computacional, la confianza y atención mostradas, así como la posibilidad de realizar este trabajo. En segundo lugar, al resto de componentes de este Grupo por su inestimable colaboración, muy especialmente a Mario Morales, Javier Murillo y Daniel Caviedes, por haber podido sacar el segundo de atención que mis dudas requerían. Agradezco también a Juan Antonio García, por su buen trato y por su interés.

Mis agradecimientos a Javier Burguete y Victor Viñals, por haber aceptado ser el director y ponente respectivos, así como por su ayuda en la revisión y corrección del texto que se presenta. También a Darío Suárez, por haberme podido responder cuantas dudas computacionales iban surgiendo.

Además, me gustaría agradecer de manera especial a Arturo Giner y Guillermo Losilla la colaboración que han mantenido conmigo en todo momento preocupándose de ofrecerme todo lo que estaba en su mano para poder realizar los tests que en este trabajo se presentan.

Resumen

En este proyecto se plantea la optimización de SFS2D, un código científico de simulación hidráulica, secuencial y escrito en Fortran, con una gran carga computacional. SFS2D sirve para modelar situaciones muy diversas que van desde la simulación de las consecuencias de una rotura de presa al estudio de una crecida repentina del caudal de un río. SFS2D ha sido desarrollado por el Grupo de Hidráulica Computacional de la Universidad de Zaragoza y se basa en un método de resolución de flujos de superficial libre que en la actualidad sirve como soporte para desarrollar nuevos modelos numéricos acoplados. Sin embargo, tras varios años de evolución, el rendimiento de SFS2D es escaso y las simulaciones de interés se prolongan demasiado en el tiempo.

Esto es un problema a la hora de obtener resultados, siendo necesaria algún tipo de optimización que haga disminuir estos tiempos lo máximo posible. Para esto y como veremos a lo largo del trabajo, se han estudiado distintas opciones de optimización, desde las proporcionadas por el propio compilador hasta el desarrollo de versiones adaptadas a diversas plataformas multiprocesador. Para ello, se han considerado los dos modelos principales de ejecución paralela en cálculos científicos: memoria compartida y paso de mensajes. La versión paralela de memoria compartida se ha codificado utilizando primitivas OpenMP y es apropiada para su ejecución en máquinas *multicore*, que integran varios procesadores de alto rendimiento en un chip. La versión paralela basada en memoria distribuída se ha programado usando primitivas MPI y es apropiada para su ejecución de un número potencialmente grande de nodos de cálculo independientes pero conectados mediante una red de alto rendimiento (máquinas de memoria distribuida). En la evaluación experimental se observa que el escalado de la versión basada en paso de mensajes es muy bueno también en máquinas de memoria compartida por lo que se considera la aportación principal de este proyecto.

Para caracterizar el rendimiento de nuestras soluciones, usamos como carga de trabajo tres simulaciones diferentes que cubren la casuística general de las simulaciones que se hacen a través del ámbito abarcado por SFS2D. La evaluación del rendimiento se ha realizado además en máquina real, utilizando tres clúster [1] suficientemente distintos como para dar validez a nuestras conclusiones. El primero de ellos es un clúster conformado por equipos de características no destinado a este tipo de ejecuciones. El segundo equipo, denominado Terminus, está especializado en computación y consigue una gran densidad de cálculo mediante una organización en blades y una jerarquía de interconexión optimizada. Por último se utiliza también el nodo de la Red Española de Supercomputación en Zaragoza, Caesaraugusta. Caesaraugusta es un supercomputador destinado únicamente a cálculo científico de memoria distribuida con 512 procesadores interconectados mediante una red de baja latencia (Myrinet).

[1] Conjunto de nodos de cálculo

Índice general

Índice de figuras

Capítulo 1
Introducción

El trabajo que se presenta consiste en la optimización y paralelización de un código científico destinado a hacer cálculos hidráulicos que requieren de una fuerte carga computacional. Éstos calculos pueden ir desde la simulación de una rotura de presa a las consecuencias de un crecimiento repentino del caudal de un río.

El código que se pretende paralelizar se basa en la resolución numérica del modelo de aguas poco profundas (shallow water) utilizado para simular flujos de agua bajo una serie de hipótesis (ver Anexo B). Los flujos de superficie libre de interés en Ingeniería Hidráulica suelen formularse bajo las hipótesis del modelo de aguas poco profundas que propone longitudes verticales mucho menores que las correspondientes horizontales en el problema. El sistema de ecuaciones resultante permite una descripción de la evolución temporal del campo fluido en función del calado y de las componentes de velocidad en los ejes X e Y.

Se trata de resolver un sistema hiperbólico de ecuaciones en derivadas parciales que, en general, no tiene una solución exacta y que por lo tanto, requiere de métodos numéricos para aproximarla. La cuestión de qué métodos son los más adecuados por ofrecer el mejor compromiso entre estabilidad, precisión y tiempo de cálculo permanece todavía abierta [13].

Las situaciones de interés dentro de la aplicación de este modelo al estudio de inundabilidad y otras cuestiones geofísicas normalmente requieren de dominios de cálculo amplios y escalas temporales largas. Por lo tanto, la aplicación práctica de esta clase de modelos es particularmente sensible al compromiso entre calidad de los resultados y eficiencia computacional. Con el fin de lograr la resolución espacial de la topografía que gobierna el movimiento del agua hacen falta mallas de cálculo finas, lo que implica un incremento de la necesidad de almacenamiento de datos, aumentando linealmente el número de operaciones y el tiempo de cálculo en función de la disminución del paso de tiempo. Cuando se desea trabajar con tiempos de cálculo razonables, el uso de códigos paralelizados nos proporciona una solución parcial a estos problemas planteados.

Para construir una implementación paralela eficiente, hay que tener en cuenta principalmente dos factores: equilibrio de carga e independencia de cálculos. Consiguiendo equilibrio de carga se evitan situaciones en las que unos procesadores trabajan y otros no. Evitando la comunicación innecesaria eliminamos tiempos de espera. En memoria compartida esto se consigue reduciendo el acceso a variables compartidas y la necesidad de sincronización, mientras que en memoria distribuida se consigue reduciendo el número y/o el tamaño de los mensajes.

En este proyecto, ambas soluciones se implementan con estándares abiertos altamente esta-

bles: OpenMP para paralelizar el código en máquinas de memoria compartida y MPI para la implementación en esquemas de memoria distribuida. El hecho de abordar estas dos opciones, presenta una ventaja adicional al usuario que utilize el código, dado que le permitirá bien trabajar en su propia máquina, si ésta dispone de una arquitectura que soporte una implementación Multicore, o bien trabajar en infraestructuras ya existentes dedicadas al procesamiento en paralelo.

1.1 Contexto del trabajo

El simulador en cuestión es utilizado en la actualidad como plataforma de desarrollo de nuevos modelos acoplados al modelo básico 2-D y reúne los requisitos de estabilidad suficiente como para pasar a la fase de transferencia a la empresa Inclam [1] dedicada a la consultoría hidráulica y medioambiental.

Este trabajo se ha realizado bajo el proyecto CENIT-TECNOAGUA CEN-20091028.

1.1.1 El Grupo de Hidráulica Computacional

El Grupo de Hidráulica Computacional (GHC) [2] de la Universidad de Zaragoza es un grupo de investigación compuesto en su mayoría por profesores del Área de Mecánica de Fluidos de la Universidad de Zaragoza con participación de investigadores de la Estación Experimental Aula Dei del Consejo Superior de Investigaciones Científicas. Además el grupo forma parte de otro mayor (Grupo de Mecánica de Fluidos Computacional) que tiene el reconocimiento de Grupo de Excelencia por la DGA.

Las áreas de interés del grupo tienen como factor común la discretización de las ecuaciones en diferencias parciales, utilizadas entre otras, para la resolución de ecuaciones de aguas poco profundas en 1-D y 2-D, esquemas de alta resolución 1-D y 2-D o modelos de simulación de transporte de convección-difusión, solutos, sedimentos, modelos acoplados 1-D+2-D o flujo subterráneo.

1.1.2 El Grupo de Arquitecturas de la Universidad de Zaragoza

El grupo de Arquitectura de Computadores de la Universidad de Zaragoza (gaZ) es un grupo de investigación cuyo núcleo son profesores del Departamento de Informática e Ingeniería de Sistemas (DIIS) de la Universidad de Zaragoza integrados en el Instituto de Investigación en Ingeniería de Aragón (I3A).

El factor común de sus investigaciones es la búsqueda de mecanismos sencillos software y hardware relacionados con la jerarquía de memoria y presentes en procesadores y multiprocesadores. Su objetivo es conseguir mayor velocidad, menor consumo o asegurar tiempo máximo de respuesta. Sus proyectos de investigación se financian principalmente por organismos públicos. Participan en SARTECO, la Sociedad de Arquitectura y Tecnología de Computadores.

[1] http://www.inclam.es
[2] http://ghc.unizar.es

Han sido reconocidos por la Comunidad Autónoma de Aragón como Grupo de investigación (emergente en 2003 y consolidado desde 2004). Asimismo forman parte desde su creación en Septiembre de 2004 de la red de excelencia europea HiPEAC (High-Performance and Embedded Architecture and Compilation).

1.2 Estructura de la memoria

La memoria se divide en dos secciones que son la memoria principal y los anexos. En la memoria principal se expone el desarrollo general del proyecto basado en la paralelización del código, mientras que en los anexos se hace referencia a otras tareas desarrolladas como la planificación del proyecto o el análisis del método en profundidad.

En los anexos A y B y H se describen las variables a las que hacemos referencia a lo largo de la misma añadiendo una descripción más detallada del método. Además en estos anexos encontraremos un razonamiento mediante el análisis de los compiladores de la necesidad de utilizar la paralelización para encontrar un mejor rendimiento.

Se incluyen otras tareas de gestión del proyecto o la instalación y gestión de una plataforma de versiones, así como una lista de herramientas utilizadas a lo largo del proyecto en los anexos C, I y G.

Por último en los anexos D y F aparece una colección completa de resultados para cuantificar en detalle los resultados de este trabajo y una lista de características descriptivas de los equipos en los que se ha probado la paralelización. Además en el anexo E se realiza una descripción empírica de la ecuación general planteada para el tiempo de ejecución en el caso de la versión paralela de SFS2D y se adjunta un ejemplo de la descomposición en el anexo J.

Capítulo 2
Conceptos principales

Para poder comprender el alcance del proyecto, es necesario introducir los conceptos en los que se sustenta el mismo. Por ello, se detallarán los fundamentos de paralelización y las herramientas en las que se apoya el trabajo.

La computación científica en todos sus ámbitos, suele requerir de una serie de aplicaciones cuyas necesidades de cálculo suelen ser elevadas [7]. En particular, la aplicación en la que se basa este proyecto, realiza sucesivos pasos temporales y una elevada cantidad de cálculos en cada uno de ellos. Por ello es importante encontrar la forma de realizar estos cálculos de la manera más eficiente posible con la finalidad de alzancar una mayor productividad en términos computacionales. En este caso, la productividad la buscaremos a través de técnicas de paralelización.

La paralelización se puede llevar a cabo en distintos niveles, desde la paralelización de instrucciones a muy bajo nivel con recursos SIMD (Single instruction Multiple Data) de procesamiendo vectorial, hasta la paralelización de tareas u operaciones con máquinas MIMD (Multiple Instruction Multiple Data) según la clasificación de Flynn[3], siendo estas últimas las que cubren las arquitecturas basadas en p procesadores destinados a atacar un problema divisible en k partes. Nosotros nos basaremos en arquitecturas MIMD, a pesar de que el procesador IBM®PowerPC 970FX disponga de un repertorio de instrucciones vectoriales conocido como VLX.

Para valorar las soluciones paralelas utilizaremos dos figuras de mérito. Por un lado utilizaremos el Speed-up S_p [9], que mide la ganancia en terminos temporales de una aplicación paralelizada ejecutada en p procesadores en un tiempo t_p frente a la ejecución de la misma aplicación ejecutada en un procesador en un tiempo t_s. Así, este factor se puede formular como,

$$S_p = \frac{t_s}{t_p} \tag{2.1}$$

Este valor está acotado teóricamente en $(0, p]$ y por lo tanto, siempre podremos comparar nuestro Speed-Up frente al Speed-Up teórico,

$$S_{theorical} = p \tag{2.2}$$

Por otro lado tenemos otro factor, que es el factor de Eficiencia o Performance que mide la relación existente entre el Speed-Up y el número de procesadores utilizados,

$$e = \frac{S_p}{p} \tag{2.3}$$

El valor de la eficiencia está acotado en $[0, 1]$. Idealmente, la eficiencia es 1, aunque realmente esta eficiencia es inalcanzable dado que existe un tiempo de comunicación t_c que implica que el

tiempo de ejecución será como mucho 2.4

$$t_{exec} = \frac{t_s}{p} + t_c \tag{2.4}$$

Esta formulación es una cota máxima que tampoco se suele alcanzar habitualmente, dado que supone una división equilibrada de trabajo en p procesadores y esto no siempre es así. Si en la ejecución para p procesadores escogemos el máximo tiempo de ejecución, $t_M = MAX(t_1, t_2...t_n)$, se cumplirá que:

$$t_M \geq \frac{t_s}{p} \tag{2.5}$$

Con lo que el tiempo de ejecución total será

$$t_{exec} = t_M + t_c \geq \frac{t_s}{p} + t_c \tag{2.6}$$

Además de todo lo contemplado aquí, para calcular de manera rigurosa el tiempo de ejecución habría que tener en cuenta los fallos en los distintos niveles de cache, que en general disminuirán según vayamos distribuyendo el espacio de datos entre las caches de los diferentes procesadores.

Todo lo anteriormente planteado sugiere que la eficiencia e será menor que uno aunque existen modelos que explican cotas del Speed-Up superiores a p y por lo tanto eficiencias superiores a uno [8], además, visibles en este trabajo. Sobre la ganancia teórica que puede tener una aplicación, existen diferentes postulaciones. Una de ellas es la ley de Amdahl[1], que propone el cálculo del Speed-Up como

$$S_p = \frac{p}{p - \alpha(p-1)} \tag{2.7}$$

siendo p el número de procesadores que interviene en el cálculo y α la fracción de código que se ejecuta en paralelo. El factor α disminuye con la ejecución de código secuencial o con la entrada/salida no paralela.

Otra propuesta es la de Gustafson [6], la cual es más optimista y enuncia el Speed-Up teórico, con las mismas variables que antes salvo α que representa la parte no paralelizable, como

$$S_p = p - \alpha(p-1) \tag{2.8}$$

Lo cierto es que la ganancia real está entre la curva que aparece de la formulación 2.7 y 2.8. El balanceo de carga que ya ha aparecido es uno de los factores determinantes en el objetivo de encontrar una paralelización efectiva. Este punto es complejo en nuestro caso dado que el número de elementos sobre los que hay que hacer los cálculos es variable en el tiempo y además, a priori, no se puede establecer ningún tipo de división perfectamente equilibrada. En nuestro caso podemos hacer la suposición de que el volumen de control coincide con el dominio de cálculo y por lo tanto que se va a hacer cálculos en toda la malla aunque en realidad esto sólo sucede cuando todas las celdas tienen calado durante todo el tiempo. Establecer una partición óptima para toda la simulación es una tarea que no se ha llevado a cabo en este trabajo pues la dificultad que entraña merece un proyecto separado.

El último concepto cuya veracidad ha sido comprobada, es la comúnmente conocida como regla 90-90 hecha popular por Jon Bentley's en 1985 que dice *"The first 90 % of the code accounts for the first 90 % of the development time. The remaining 10 of the code accounts for the other 90 % of the development time"*. Esta frase tiene la curiosidad de que los porcentajes no suman 100 y hay quien dice que es un error tipográfico, pero existe otra corriente que piensa que ya

entonces se presumía que la naturaleza de los proyectos de desarrollo de software es necesariamente no cumplir los plazos predichos. Por cualquiera de las dos corrientes se ha visto afectado este proyecto.

Por otro lado el modelo numérico del simulador es un desarrollo lo suficientemente complejo como para que su explicación detallada aparezca en el anexo B. La evolución que ha ido sufriendo año atrás (e incluso durante el desarrollo de este proyecto) ha sido una de las razones de que el análisis del mismo haya resultado muy compleja. Como veremos en capítulos posteriores, la estrategia de paralelización del código no es ni mucho menos trivial y su implementación tampoco.

Es oportuno describir como funciona el algoritmo a nivel computacional. En el apéndice H se puede ver el Diagrama de Flujo de la aplicación a lo largo de la simulación. En este proyecto, sólo se analizará y paralelizará el método *blocks1* correspondiente con el método de orden 1 sin sedimentos aunque *blocks1sedimentos* será relativamente sencillo a partir del primero.

De ahora en adelante, nos referiremos a *blocks1* como al núcleo de cálculo, es decir, la función que desarrolla todo el método numérico sobre un dominio de cálculo que estará representado computacionalmente por una malla (Ver Anexo B). Así mismo, es conveniente explicar que el método es iterativo, lo que implica que la resolución lleva implícita un paso de tiempo asociado. En este caso, este paso de tiempo viene impuesto por la condición de CFL [12]. Este paso de tiempo lo denominaremos Δt y por lo tanto cada iteración se hace con un paso de tiempo asociado para la resolución de la variable de interés W.

La variable W se compone de $W_n = (h_n, u_n, v_n)$ donde h_n es el calado de la celda n, u_n, v_n son las componente de la velocidad del flujo en la celda n y $s_{i,1}, ..., s_{i,n}$ son los términos fuente de la ecuación para la celda n.

Es importante tener en cuenta que a lo largo de la ejecución, puede haber celdas que pasen de estar *secas* a estar *mojadas*, es decir, que tengan calado $h > 0$. Esto supondrá que hay que ampliar el dominio de cálculo, y esto lo indica una función externa a la de *blocks1*, pero que de manera implícita está paralelizada ya que cada dominio de cálculo tendrá que ampliar únicamente las celdas de su dominio.

Capítulo 3
Paralelización de la aplicación

La paralelización de la aplicación es el objetivo principal del proyecto desarrollado con la finalidad de disminuir el tiempo de cálculo. A partir de este proyecto ha nacido una relación con el BIFI que nos ha permitido utilizar sus infraestructuras para realizar nuestros cálculos. Para las pruebas que a continuación presentaré, se han utilizado tres *cluster* de cálculo. El primero es el *cluster* Trombón que pertence al grupo GHC; el segundo pertenece al *cluster* Terminus[1] como parte del programa de Hosted Computing del BIFI de la Universidad de Zaragoza. El tercero es Caesaraugusta [2], nodo de la Red Española de Supercomputación que está instalado en el edificio de Ciencias de la Universidad de Zaragoza y es gestionado por el BIFI.

En este capítulo se trata además, la adaptación de las tareas de preproceso y postproceso propias de la aplicación de cálculo.

3.1 Paralelización en máquinas de memoria compartida

La paralelización en máquinas de memoria compartida se ha implementado bajo el estándar OpenMP [2] como ejercicio de introducción al código SFS2D.

Para la paralelización por este método, la estrategia que se sigue es la de, a partir de las conclusiones obtenidas del análisis de compilación y detalladas en el anexo A, estudiar las secciones críticas en tiempo para poder rebajarlo.

La parte identificada correspondiente al cálculo de W es *blocks1* y se basa en bucles que operan sobre las celdas y las paredes del volumen de control y son estos bucles los que hay que analizar aplicando técnicas similares a las utilizadas en el análisis de dependencias a bajo nivel [14].

Podemos establecer dos tareas basadas en bucle dentro de esta función:

1. Cálculo de ΔW y Δt
2. Actualización de W

En estos dos bucles existen sendas dependencias que es necesario analizar.

[1] http://bifi.es/infrastructures/supercomputing/terminus/index.php
[2] http://bifi.es/infrastructures/caesaraugusta/index.php

3.1.1 Cálculo del ΔW y Δt

El cálculo de todas las variables se hace recorriendo las paredes de la malla de calculo, lo que implica que será un bucle que barrerá todo el dominio de paredes incluídas en el cálculo en el paso de tiempo actual.

En el caso del cálculo de ΔW las dependencias hacen que la cabecera del bucle OpenMP quede de la siguiente forma:

```
!$OMP PARALLEL DEFAULT(PRIVATE) SHARED(NPAREDINCLULLE,
!$OMP+ NUM_NPAREDINCLULLE,PARED,W,TOLERA,VOLC,NORMALP,LADOP,PRIMITIVAS,
!$OMP+ PRIMITIVASRAIZ,RAIZHG,FONDO,RN,B_FRICC,VECCELL,PAREDNV,
!$OMP+ CELLDP,PERDIDAPAREDPUENTE,GANANCIA_ENER_ORBAL,CHI,LADO,
!$OMP+ DW2,VECCELL2 ,NUM_NLISTA,NEQ,
!$OMP+ ACTIVATE_RIENMANN,TIPOF,NVERT,MODEL_FRICTION,ACTIVEGAMMA,
!$OMP+ NSOLUTOS,ACTIVEENTROPY,
!$OMP+ DT,NLISTA,G,RAIZG
!$OMP+ )
!$OMP DO
```

El paso de tiempo Δt es una función de la velocidad y del calado en una celda dada, en particular, para todo el dominio de cálculo, será el menor de ellos, por lo que deberemos añadir además esta linea detrás del *!$OMP DO*:

```
!$OMP+ REDUCTION(MIN:DT)
```

Con esta directiva lo que hacemos es indicar a OpenMP que de todos los menores Δt escoga el menor.

El bucle lo hemos paralelizado diciendo que, por defecto, las variables serán privadas, salvo las que indicamos de manera explícita, mientras que este bucle podríamos hacerlo de una forma equivalente

```
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(K,N,N1,N2,H1,H2
!$OMP+ MOJADOIJ,AREA,NX,NY,NORM,U1,U2,V1,V2,AUX1,AUX2,AUX3,HBAR
!$OMP+ UBAR,VBAR,FISOLBAR,CBAR,UNORMAL,ABSUNORMAL,FROUDE,LANDA,
!$OMP+ DIFQX,DIFQY,AUX4,E,BETA,LANDAL,LANDAR,BETAAUX,LANDAAUX,C1,
!$OMP+ C3,Z2,Z1,PS1,PS2,DZ,ABSDZ,PS,PSMAX,FRICTION,FRICTION2,FRICTION3,
!$OMP+ MAXH,MODUL1,MODUL2,MINU,B_FRICMEDIA,ACTIVA_PARED,NCN,NV,AUX10
!$OMP+ MINRATIO,HQI1,HQI2,HSTAR,HLS,HLR,DT3,DT2,DT4,DTSMALL,PARED_SOLIDA
!$OMP+ ,HRS,DW2AUX,L,J,COND1,COND2,COND3,QL,QR,FR,FL,DFJ1,DFJ2)
!$OMP DO
!$OMP+ REDUCTION(MIN:DT)
```

3.1.2 Actualización de la variable W

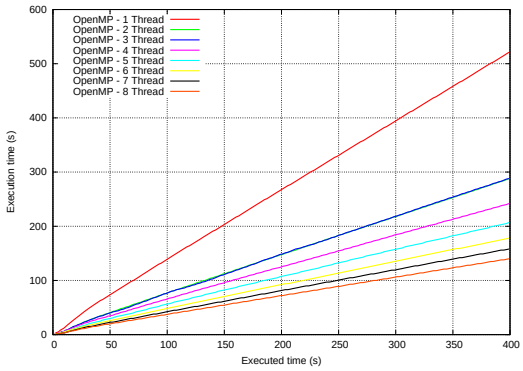
La actualización de la variable W en el paso de tiempo n es un bucle que recorre en i todo el dominio de cálculo como:

$$W_i^n = W_i^{n-1} + \Delta W_i^{n-1} * \Delta t \tag{3.1}$$

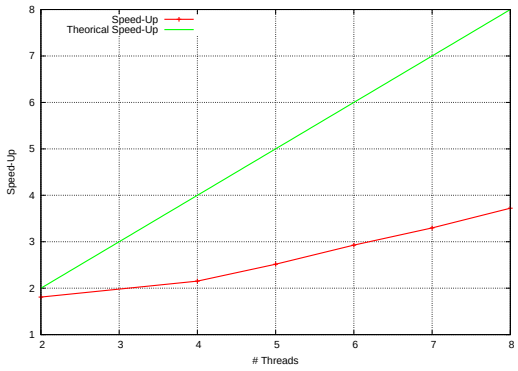
obteniendo de (3.1) el resultado de los nuevos valores de ΔW

3.1.3 Resultados de la ejecución en máquinas de memoria compartida

Los resultados que aquí aparecen son para la arquitectura F.3 con el caso de la figura 3.5 con un calado constante. El binario procede del compilador gFortran 4.4 sin ninguna optimización.



(a) Tiempos de ejecución (y) frente a tiempo ejecutado (x)



(b) SpeedUp vs. Theorical SpeedUp

Figura 3.1: Tiempos de ejecución (a) y Speed-Up (b) de OpenMP para el caso de prueba

Los datos recogidos en la figura 3.1 muestran una diferencia notable respecto a rendimiento teórico. El speedUp está calculado de la forma:

$$S_p = \frac{t_s}{t_p} \tag{3.2}$$

Para poder entender estos resultados, es necesario analizar qué está sucediendo a bajo nivel. Para esto vamos a utilizar la herramienta Valgrind [3]. Con esta herramienta, veremos los fallos de cache en los distintos niveles. Cabe destacar que sólo aparecen los niveles L1 y L2 de la cache de datos mientras que la arquitectura dispone también de nivel L3. Por algún motivo desconocido, estos datos no aparecen en la ejecución de Valgrind.

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	12.99	12.99	Ir	
L1 Instr. Fetch Miss	49.03	49.03	I1mr	
L2 Instr. Fetch Miss	7.59	7.59	I2mr	
Data Read Access	9.77	9.77	Dr	
L1 Data Read Miss	0.17	0.17	D1mr	
L2 Data Read Miss	0.41	0.41	D2mr	
Data Write Access	23.53	23.53	Dw	
L1 Data Write Miss	0.25	0.25	D1mw	
L2 Data Write Miss	0.55	0.55	D2mw	
L1 Miss Sum	17.99	17.99	L1m	= I1mr + D1mr + D1mw
L2 Miss Sum	0.49	0.49	L2m	= I2mr + D2mr + D2mw
Cycle Estimation	11.43	11.43	CEst	= Ir + 10 L1m + 100 L2m

(a) Análisis de la función principal

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	46.01	46.01	Ir	
L1 Instr. Fetch Miss	0.07	0.07	I1mr	
L2 Instr. Fetch Miss	23.61	23.61	I2mr	
Data Read Access	54.73	54.73	Dr	
L1 Data Read Miss	49.62	49.62	D1mr	
L2 Data Read Miss	45.33	45.33	D2mr	
Data Write Access	35.53	35.53	Dw	
L1 Data Write Miss	29.77	29.77	D1mw	
L2 Data Write Miss	31.03	31.03	D2mw	
L1 Miss Sum	27.37	27.37	L1m	= I1mr + D1mr + D1mw
L2 Miss Sum	40.26	40.26	L2m	= I2mr + D2mr + D2mw
Cycle Estimation	44.12	44.12	CEst	= Ir + 10 L1m + 100 L2m

(b) Análisis del bucle principal de la función

Figura 3.2: Fallos de cache en el programa principal (a) y fallos de cache en *blocks1* (b) para la aplicación con 1 thread

En la figura 3.1 vemos que el Speed-Up es más o menos lineal pero la pendiente de la curva es notablemente inferior a la de la teórica. Analizando estos datos con los de los resultados del análisis de cache 3.2, 3.3 y 3.4, podemos ver que la ganancia que teóricamente deberíamos tener al dividir en el número de threads, se ve eclipsado por el número de fallos en caché que incurre el

[3] http://valgrind.org/

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	68.52	68.52	Ir	
L1 Instr. Fetch Miss	49.03	49.03	I1mr	
L2 Instr. Fetch Miss	8.68	8.68	I2mr	
Data Read Access	47.02	47.02	Dr	
L1 Data Read Miss	0.46	0.46	D1mr	
L2 Data Read Miss	0.54	0.54	D2mr	
Data Write Access	23.55	23.55	Dw	
L1 Data Write Miss	0.44	0.44	D1mw	
L2 Data Write Miss	0.76	0.76	D2mw	
L1 Miss Sum	17.88	17.88	L1m	= I1mr + D1mr + D1mw
L2 Miss Sum	0.65	0.65	L2m	= I2mr + D2mr + D2mw
Cycle Estimation	62.45	62.45	CEst	= Ir + 10 L1m + 100 L2m

(a) Análisis de la función principal

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	16.65	16.65	Ir	
L1 Instr. Fetch Miss	0.08	0.08	I1mr	
L2 Instr. Fetch Miss	25.19	25.19	I2mr	
Data Read Access	32.13	32.13	Dr	
L1 Data Read Miss	49.14	49.14	D1mr	
L2 Data Read Miss	48.72	48.72	D2mr	
Data Write Access	35.52	35.52	Dw	
L1 Data Write Miss	29.50	29.50	D1mw	
L2 Data Write Miss	30.03	30.03	D2mw	
L1 Miss Sum	27.35	27.35	L1m	= I1mr + D1mr + D1mw
L2 Miss Sum	41.67	41.67	L2m	= I2mr + D2mr + D2mw
Cycle Estimation	18.70	18.70	CEst	= Ir + 10 L1m + 100 L2m

(b) Análisis del bucle principal de la función

Figura 3.3: Fallos de cache en el programa principal (a) y fallos de cache en *blocks1* (b) para la aplicación con 8 threads

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	77.81	77.81	Ir	
L1 Instr. Fetch Miss	49.03	49.03	I1mr	
L2 Instr. Fetch Miss	8.68	8.68	I2mr	
Data Read Access	58.32	58.32	Dr	
L1 Data Read Miss	0.46	0.46	D1mr	
L2 Data Read Miss	0.55	0.55	D2mr	
Data Write Access	23.55	23.55	Dw	
L1 Data Write Miss	0.43	0.43	D1mw	
L2 Data Write Miss	0.76	0.76	D2mw	
L1 Miss Sum	17.89	17.89	L1m	= I1mr + D1mr + D1mw
L2 Miss Sum	0.66	0.66	L2m	= I2mr + D2mr + D2mw
Cycle Estimation	72.81	72.81	CEst	= Ir + 10 L1m + 100 L2m

(a) Análisis de la función principal

Event Type	Incl.	Self	Short	Formula
Instruction Fetch	11.73	11.73	Ir	
L1 Instr. Fetch Miss	0.08	0.08	I1mr	
L2 Instr. Fetch Miss	25.31	25.31	I2mr	
Data Read Access	25.28	25.28	Dr	
L1 Data Read Miss	49.10	49.10	D1mr	
L2 Data Read Miss	48.16	48.16	D2mr	
Data Write Access	35.52	35.52	Dw	
L1 Data Write Miss	29.47	29.47	D1mw	
L2 Data Write Miss	29.54	29.54	D2mw	
L1 Miss Sum	27.31	27.31	L1m	= I1mr + D1mr + D1mw
L2 Miss Sum	41.11	41.11	L2m	= I2mr + D2mr + D2mw
Cycle Estimation	13.51	13.51	CEst	= Ir + 10 L1m + 100 L2m

(b) Análisis del bucle principal de la función

Figura 3.4: Fallos de cache en el programa principal (a) y fallos de cache en *blocks1* (b) para la aplicación con 4 threads

programa principal. Tengamos en cuenta que se disminuyen notablemente estos fallos en *blocks1* pero no en las funciones restantes en las que de hecho, pasa lo contrario.

3.2 Paralelización en máquinas de memoria distribuida

A la vista de los resultados de la paralelización en máquinas de memoria compartida, parece necesario abordar una estrategia de paralelización más específica para el código. Esta paralelización se llevará a cabo bajo el estándar MPI [5].

Las estrategias que podríamos llevar a cabo son:

1. Paralelizar tareas
2. Paralelizar operaciones

Las diferencias entre ambas radican en la forma de implementar el código y por supuesto en el rendimiento que teóricamente nos puede ofrecer una opción u otra. En el caso de paralelizar tareas y aplicado a nuestro caso, esta implementación se basaría en que p procesadores hicieran p tareas distintas e independientes con la finalidad de aumentar el rendimiento aumentando la productividad. Ejemplos de esta forma de paralelizar se encuentran en simulaciones de Monte Carlo en la cual cada uno prueba con diferentes valores y a lo largo del tiempo van comunicándose las aproximaciones. Paralelizar operaciones descompone una tarea en operaciones dependientes de manera distribuída. En nuestro caso esto será repartir el dominio de cálculo en p partes, y asignarle cada parte a cada nodo de cálculo. La estrategia que aquí se propone es la de paralelizar tareas, es decir, dejar a cada nodo de cálculo que ejecute sus operaciones sobre un subdominio. Esto es que cada nodo calcule un subdominio de los mostrados en la figura 3.5. Está técnica se

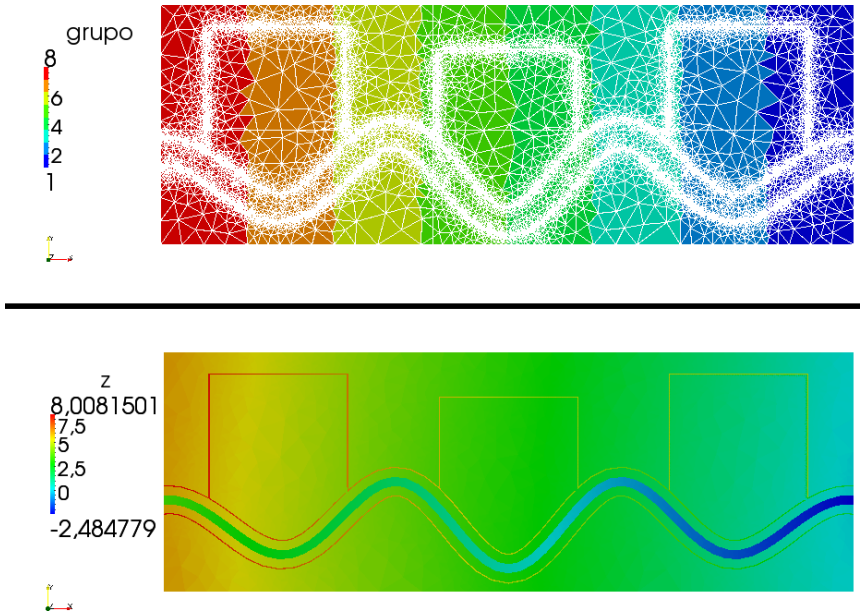


Figura 3.5: particion de la malla en 8 subdominios (*grupo*) y altura topográfica z respecto al 0 (Nivel de salida)

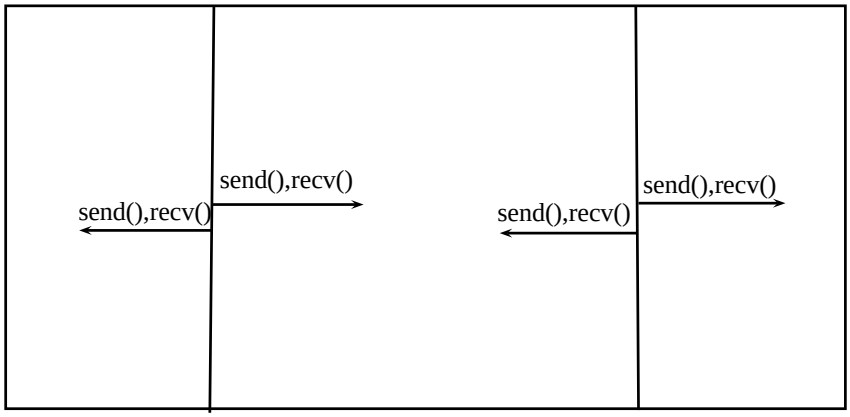


Figura 3.6: esquema de comunicación 1-D

basa pues en crear varios subdominios de cálculo y conectarlos de manera consecutiva, es decir, implementar un esquema de comunicación *1-Dimensional* la cual sólo se hace con, como mucho, dos nodos como vemos en la figura 3.6. Para crear estos subdominios y como veremos en la siguiente sección, hemos de hacer cumplir alguna condición para que la paralelización sea efectiva.

En este modelo cada celda depende de sus vecinas y en particular, en el límite entre dos subdominio de cálculo, las celdas limítrofes dependerán también de las celdas vecinas. Por lo tanto necesitamos, para cada paso de tiempo, transferir los valores de las celdas vecinas de los subdominios D_{s-1} y D_{s+1} que necesita el subdominio D_s como vemos en la figura 3.6.

La parte compleja de la paralelización es elegir la estrategia de comunicación de estas *variables comunes* para cada subdominio de entre las dos disponibles:

1. Elegir un nodo director que vaya recopilando las variables necesarias en cada paso de tiempo y redistribuyéndolas.

2. Ser cada nodo el encargado de transmitir las variables que sabemos va a necesitar cada vecino.

Ambas presentan ventajas e inconvenientes en distintos aspectos, pero caben destacar la sencillez de programación de la opcion 1, frente al buen rendimiento de la 2. La solución implementada será 2 dado que, como veremos, implica una comunicación independiente, estable y equilibrada.

Para paralelizar SFS2D, la segunda implementación representa la forma en que cada subdominio físico se comunica con sus vecinos, que en términos computacionales es, que cada nodo se comunica con sus dos vecinos. Para que esta comunicación se pueda llevar a cabo, el esquema de sincronización es implícito por el funcionamiento en el estándar MPI mediante el cual las llamadas a MPI_{recv} y MPI_{send} son bloqueantes, pero aún así, se ha de establecer algún orden. El elegido sigue el esquema del algoritmo 1.

Algorithm 1 Syncronize w

Require: El número de particiones $p \geq 2$

if $MPI_{rank} = 1$ **then**
  Recibir de 2 y enviar a 2
else
  if $MPI_{rank} = p$ **then**
    Enviar a $n-1$ y recibir de $n-1$
  else
    if $MPI_{rank} MOD(2) = 0$ **then**
      Enviar a $p-1$, recibir de $p-1$, recibir de $p+1$ y enviar a $p+1$
    else
      Recibir de $p+1$, enviar a $p+1$, enviar a $p-1$ y recibir de $p-1$
    end if
  end if
end if
return true

Descrito el esquema de comunicación, las variables que se comunicarán en primera instancia serán las conservadas del tiempo $n-1$

$$W^{n-1} = [h^{n-1}, u^{n-1}, v^{n-1}] \tag{3.3}$$

necesarias para el cálculo de las celdas vecinas en el tiempo n.

Por otro lado y en cada iteración debemos sincronizar el paso de tiempo. En el modelo se establece un paso de tiempo común Δt para todas las celdas, como el mínimo de los Δt *permitido* para cada celda. En particular, nuestro paso de tiempo estará definido como el de la figura 3.4.

$$dt_{min} = MIN(\Delta t), i \in (1, ncell) \equiv MIN(\Delta t_{i,s}), i \in (1, ncell_s), s \in (1, N_{dom}) \tag{3.4}$$

Este proceso lo haremos siguiendo el siguiente esquema del algoritmo 2.

El paso de sincronizar la variable W y el Δt es toda la comunicación que se requiere en cada paso de tiempo. Para esta implementacion la variable compartida es sólamente W.

Algorithm 2 Syncronize Δt

Require: El número de particiones $p \geq 2$
if $MPI_{rank} = 1$ **then**
  $DT(0) = DT_0$
  for l=1..p-1 **do**
    Recibo $DT(mpi_{rank})$ del nodo l
  end for
  Hallo el mínimo de ellos ($dt = MIN(DT(0..p-1))$
  for i=1..p-1 **do**
    Mando dt al nodo l
  end for
else
  Calculo el mínimo dt
  Mando dt_{rank}
  Recibo dt_{min}
end if
return true

Con esta forma de implementar el algoritmo en paralelo, podemos replantear la fórmula del tiempo de cálculo propuesta en el capítulo 2 como sigue en 3.5, donde $ncell_s$ es el numero de celdas del dominio s y f la frontera del dominio.

$$t_{exec} = t_0 + (t_{exec1} * ncell_s) + ((t_{comm1} * N_{f,s}) * 2) \tag{3.5}$$

Está fórmula es difícil de ajustar a valores reales dado que t_{exec1} que es el tiempo de ejecución de una celda y t_{comm1} que es el tiempo de comunicación de una celda puede llegar a ser muy variable. El primero en función de los fallos de caché que se produzcan para acceder a una celda determinada por ejemplo, y el segundo el *jitter* propio de la red de comunicación. El t_0 es el tiempo de inicalizacion, que como veremos en el siguiente capitulo, podemos despreciar e igualmente veremos que aunque es dificil ajustar esta función, se pueden sacar valores que nos hagan una idea del tiempo de ejecución.

3.3 Preproceso y Postproceso de las mallas de cálculo

El preproceso de las mallas es el factor más influyente del rendimiento que va a tener la aplicación. El modelo de simulacion sólo calcula en las celdas mojadas y su dominio de cálculo puede crecer en el tiempo en algunos casos.

En este proyecto, los casos que se han testado parten de la hipótesis de que todas las celdas están mojadas, lo que implica que se hacen cálculos en todas las celdas, con lo que particionar la malla será equivalente a dividir el número de cálculos. Esto no siempre será así ya que, en general, no conocemos dónde estará más mojada la malla y por lo tanto no se puede hacer un pre-balanceo que asegure una perfecta distribución de la carga.

Si suponemos que esto no es así, podemos demostrar que el equilibrado de la carga estará desequilibrada. Supongamos una malla particionada en S subdominios con un número de celdas por subdominio igual en todas ellas y que en un tiempo t_0 el 30 % de las celdas de cada dominio

están mojadas. Si suponemos que la carga estará equilibrada en t_0 implica que

$$\frac{t_{x1}}{t_{x0}} = \frac{(t_{exec1} * ncell_s * 0{,}3) + ((t_{comm1} * N_{f,s}) * 2)}{(t_{exec1} * ncell_s * 0{,}3) + ((t_{comm1} * N_{f,s}) * 2)} = 1 \tag{3.6}$$

Supongamos ahora que en el tiempo t_1 un 10 % de las particiones tiene un incremento al 40 % de las celdas mojadas. Asumiendo que se producen incrementos en determinadas zonas y con el caso propuesto, se obtiene

$$\frac{t_{x1}}{t_{x0}} = \frac{(t_{exec1} * ncell_s * 0{,}4) + ((t_{comm1} * N_{f,s}) * 2)}{(t_{exec1} * ncell_s * 0{,}3) + ((t_{comm1} * N_{f,s}) * 2)} > 1 \tag{3.7}$$

El problema de esto es que sólo el 10 % de las celdas tardan $(t_{exec1} * ncell_s * 0{,}4) + ((t_{comm1} * N_{f,s}) * 2)$, el 90 % de ellas tardan $(t_{exec1} * ncell_s * 0{,}3) + ((t_{comm1} * N_{f,s}) * 2)$, con lo que hemos conseguido desequilibrar el cálculo.

Por tanto y para nuestro caso, vamos a suponer una malla isotrópica (la densidad de celdas es uniforme en toda la geometría) y además vamos a suponer que la mayor parte del tiempo se va a hacer cálculos en la mayor parte de las celdas.

Particionar la malla para nosotros, es establecer un orden en las celdas. En el anexo J Este orden no está implícito en su creación dado que, en principio, en las mallas no estructuradas no existe una manera de establecerlo. Lo que haremos nosotros de manera artificial es ordenarlas para que se cumpla que

$$C_{x,1} < C_{y,2} < ... < C_{z,n}, \forall x, y, z \in [1, ncell_s], s, n \in [1, N_s] \tag{3.8}$$

siendo N_s el número de celdas del dominio S y $C_{i,j}$ el identificador de la celda i del dominio j. Estableciendo este orden, se cumplirá que cada nodo deberá hacer los cálculos en las celdas C_ϕ pertenecientes a un dominio acotado en

$$C_\phi \in (ncell_{s-1}, ncell_s] \tag{3.9}$$

Para poder crear esta distribución, se sigue el esquema del algoritmo 3. Con el algoritmo 3 se persigue particionar la malla de manera equitativa dando muy buenos resultados con mallas isotrópicas y aceptables con mallas anisotrópicas como se ve en la figura 3.7.

A simple vista no se aprecia demasiado el equilibrado de los subdominios, viendose los resultados del reparto en la figura 3.8 para las figuras anteriores.

Existen aplicaciones como ParMetis[4] que particionan otro tipo de mallas aplicando técnicas multinivel. Este tipo de particiones son necesarias para mallas 3-D o para casos en el que exista mallado dinámico, en los cuales y en tiempo-real se ha de ir repartiendo el trabajo. Para nuestros casos esta técnica es suficiente, aunque existen alternativas planteadas en función de gradientes de pendientes o probabilidad de inundación, e incluso técnicas de presimulación para intuir la partición óptima para toda la simulación.

[4] http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

Algorithm 3 Paralellizer

Require: El número de particiones $n \geq 2$

Calcula baricentro de todas las celdas $bc[i]$

Halla $X_m = MIN(bc[i]_x, X_M = MAX(bc[i]_x, Y_m = MIN(bc[i]_y, Y_M = MAX(bc[i]_y)$

El área por celda será $A_c = \frac{|X_M| - |X_m| * |Y_M| - |Y_m|}{ncell}$

Establecemos la anchura de recorrido como $H = \frac{bc[random()]_y}{3}, W = \frac{bc[random()]_x}{3}$

if $|X_M| - |X_m| > |Y_M| - |Y_m|$ **then**

  Ordenamos en el eje X de la malla estableciendo la anchura en celdas de cada particion

  for i=1,n **do**

    while $SumCeldas < \frac{ncell}{n}$ **do**

      for j=1,ncell **do**

        if $bc[i]_x < W_{aux}$ **then**

          Añadimos esta celda al subdominio i y actualizamos SumCeldas

        end if

      end for

      $W_{aux} += W$

    end while

    $NumCeldas = 0$

  end for

else

  Ordenamos en el eje y de la malla estableciendo la altura en celdas de cada particion

  $H_{aux} = Y_M$

  for i=1,n **do**

    while $SumCeldas < \frac{ncell}{n}$ **do**

      for j=1,ncell **do**

        if $bc[j]_y < H_{aux}$ **then**

          Añadimos esta celda a ll subdominio i y actualizamos SumCeldas

        end if

      end for

      $H_{aux} -= H$

    end while

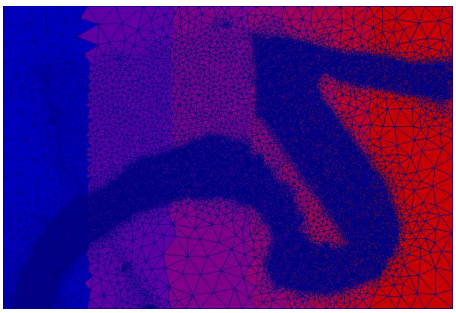
    $NumCeldas = 0$

  end for

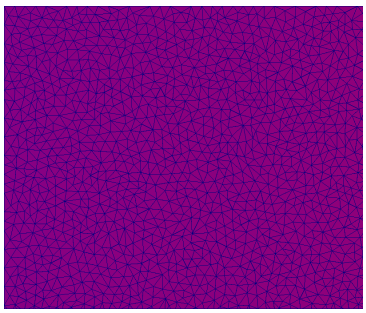
end if

Reescribimos la malla con el nuevo orden

return true



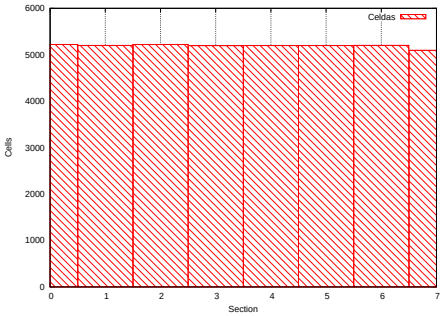
(a) $Malla_1$ con 8-particion



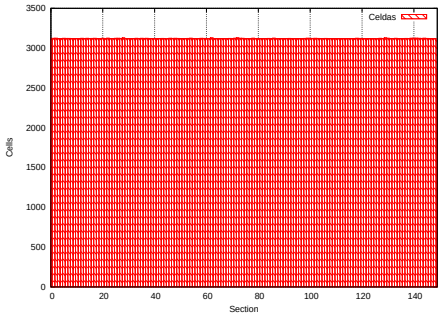
(b) Detalle de $Malla_2$ de 150-particion

(c) $Malla_2$ con 150-particion

Figura 3.7: Particiones con mallas Isotrópicas y Anisotrópicas



(a) Distribución de $Malla_1$ con 8-particion



(b) Distribución de $Malla_2$ con 150-partición

Figura 3.8: Distribución de celdas en subdominios

Capítulo 4
Resultados de la paralelización

Los resultados que aquí se presentan son de la implementación MPI del código SFS2D en diferentes *cluster*. La razón de presentar únicamente estos resultados es por la disponibilidad de infraestructuras suficientes para hacer pruebas de rendimiento en máquinas de tipo Memoria Distribuída, mientras que para la versión OpenMP no disponíamos de máquinas homólogas en características de tipo Memoria Compartida.

Los resultados que aquí se presentan están, por tanto, probados en diferentes máquinas con diferentes configuraciones.

En este capítulo sólo se recogen los resultados para el caso C.1 como caso representativo para sacar las conclusiones pertinentes.

4.1 Casos de prueba

Se han diseñado 3 casos de prueba que contemplan una variedad suficiente como para comprobar el rendimiento de la paralelización. Los casos se describen mejor en el anexo D y aquí sólo nos centramos en el caso C.1 descrito en los anexos.

Los resultados para este caso son representativos porque se produce un llenado rápido de todo el dominio, con lo que se equilibra rápidamente la carga de computacional (20 s.) y a partir de ese momento todos los equipos tienen completamente repartidas las celdas a calcular.

La técnica en la que nos hemos basado es la de simular 400 segundos de evolución temporal e ir sacando el tiempo de ejecución cada 200 pasos de tiempo. Estos 200 pasos de tiempo para nuestro problema, significan aproximandamente cada 1.8 s. simulados, dado que el paso de tiempo medio es dt $9 * 10^{-3} s.$, con lo que $T_{muestreo} = 9 * 10^{-3} * 200 s.$, con lo que en 400 s. de simulación habremos obtenido los tiempos de ejecución de $\frac{400 s.}{1{,}8 s.} = 222{,}2$ conjuntos de pasos de tiempo distintos. Obviamente y para simplificar, asumimos que el tiempo de carga inicial es despreciable dado que el algoritmo de inicialización de las matrices es muy pequeño respecto al tiempo de ejecuón y como vemos 4.1, la ecuación sólo incluye el tiempo de cálculo y de comunicación.

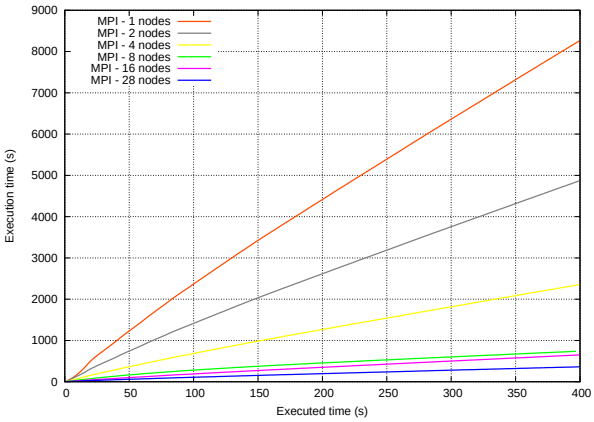
$$t_{exec} = \cancelto{0}{t_0} + (t_{exec1} * ncell_s) + ((t_{comm1} * N_{f,s}) * 2) \tag{4.1}$$

Además, para sacar el Speed-up y el Rendimiento, se utilizará el tiempo de ejecución de 400 s. simulados.

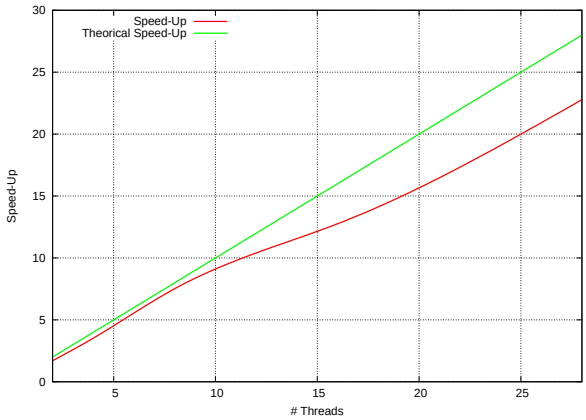
4.2 Rendimiento en Trombón

Los resultados que hemos obtenido en este *cluster* son importantes, dado que esta es la plataforma de la que dispone el grupo para hacer sus simulaciones.

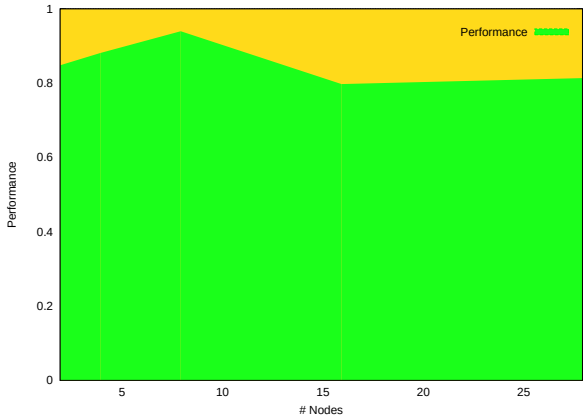
Los rendimientos que se obtienen son bastante aceptables aunque hemos de tener en cuenta que esta infraestructura no se creó pensando en lanzar aplicaciones paralelas. Esto es algo que deberá ser revisado en el futuro para obtener un mejor rendimiento de la instalación.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 400 s. de simulación



(c) Performance en 400 s. de simulación

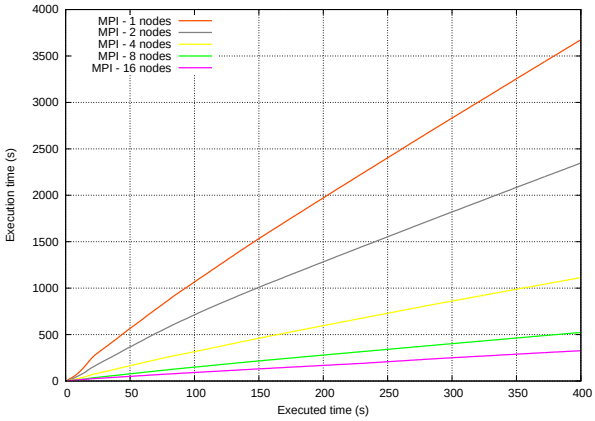
Figura 4.1: Tiempos de ejecución (a), speed-ud (b) y rendimiento (c) del caso C.1 en *Trombón*

Aún así se ve en 4.1 que escala de manera lineal hasta 16. Esta diferencia entre 8 y 16 se debe a que los resultados con 8 nodos son fruto de la ejecución en 2 nodos físicos, cada uno con 4 cores en el procesador, es decir, el factor de comunicación intra-nodo solo influye una vez en tanto que sólo existe una frontera entre un nodo y otro, mientras que a partir de 8, por ejemplo con 16, ya existe una máquina que tiene que comunicarse con otras dos.

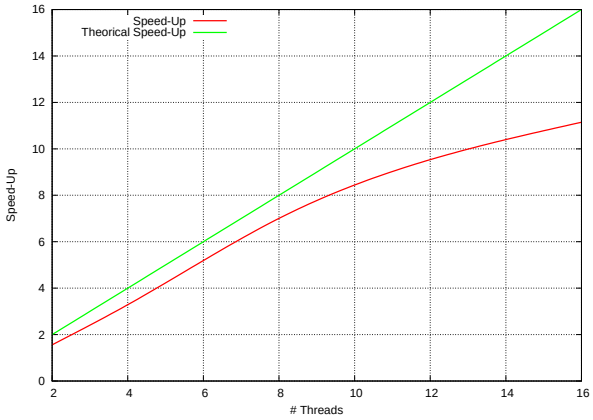
Si tuvieramos más nodos, sería esperable que el rendimiento bajase en este caso particular según aumentásemos los nodos como consecuencia de lo que veremos en la ejecución de Caesaraugusta, que es que la relacion de celdas calculadas frente a celdas comunicadas se va haciendo cada vez sustancialmente mas pequeña.

4.3 Rendimiento en el *cluster* Terminus

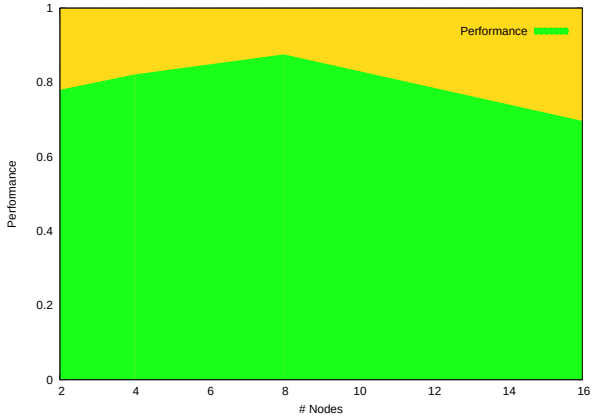
En este *cluster*, las pruebas se han hecho bajo una compilación altamente optimizada para la arquitectura a través del *cluster*. Además es importante notar que la ejecución en menos de 4 nodos se hace on-chip, para 8 se hace on-board y para 16 intra-nodo.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 400 s. de simulación



(c) Performance en 400 s. de simulación

Figura 4.2: Tiempos de ejecución (a), speed-ud (b) y rendimiento (c) del caso C.1 en *Terminus*

Si bien es cierto que la optimización a la hora de la compilación es muy importante, lo destacable de estos resultados es el rendimiento 4.2(c) que saca la paralelización. El hecho de que aumente según aumentamos los nodos (hasta 8) es porque se priman los aciertos en cache, mientras que por el lado contrario, aumentando fuera del nodo, se penaliza la comunicación intra-nodo.

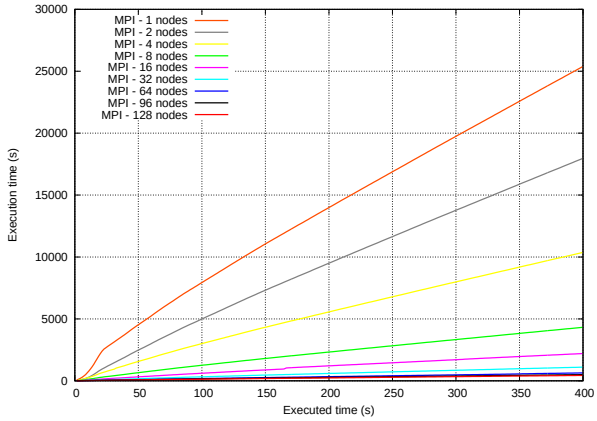
De estos resultados también hay que resaltar que el tiempo de ejecución, ya sólo la ejecución del código sin optimizar, es algo inferior a 2 veces más rápida que en *Trombon* y algo más de 6

veces más que en *Caesaraugusta*.

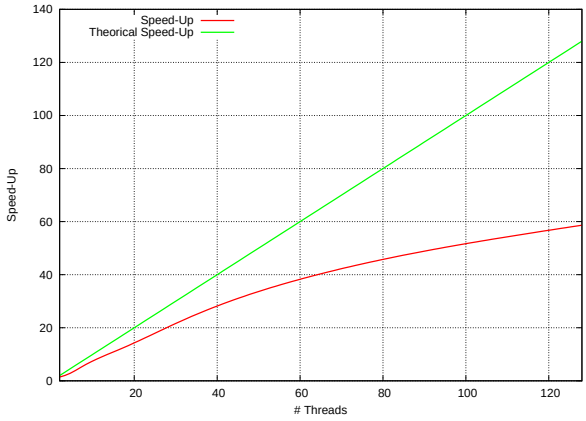
Igual que sucede en la máquina *Trombon*, a partir de 8 nodos el rendimiento se reduce ligeramente debido a la misma razón. Aquí en cambio la configuración es ligeramente distinta lo que implica conclusiones distintas: Terminus tiene 2 nodos físicos, cada uno de ellos con una placa dual y cada una de ellas con dos procesadores; esta configuración implica que a partir de cuatro nodos no sale de la placa pero sí sale del procesador. Es a partir de 8 cuando sale del nodo y se comunica con el otro nodo. fijémonos que la tendencia del speed-up es muy similar en ambos, aunque el performance sea algo superior en esta máquina.

4.4 Rendimiento en el *cluster* RES-Caesaraugusta

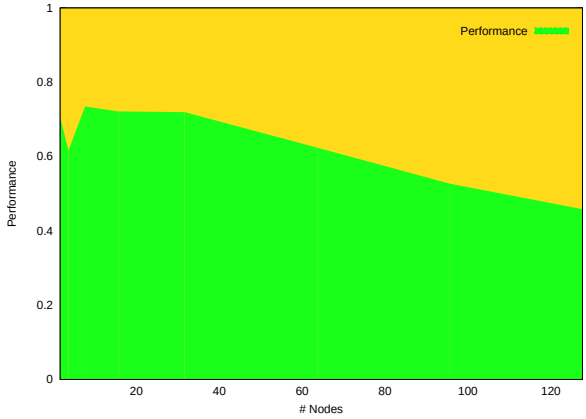
Caesaraugusta tiene una configuración mejor preparada para esta paralelización que el resto de computadoras donde se han hecho pruebas y los resultados que salen son muy interesantes en este sentido. En este caso hemos podido hacer pruebas con hasta 128 nodos de cálculo aunque este número no sea el que mejor rendimiento ofrece como vemos en la figura 4.3.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 400 s. de simulación



(c) Performance en 400 s. de simulación

Figura 4.3: Tiempos de ejecución (a), speed-ud (b) y rendimiento (c) del caso C.1 en *Caesaraugusta*

De estos resultados se desprende la conclusión de que a partir de 40 particiones, en este caso, la ganancia empieza a ser menor.

Una de las conclusiones más importantes que podemos sacar, no sólo de esta ejecución si no de todo el trabajo y que será desarrollada en más profundidad, es la resultante de que a la vista de los resultados, el tiempo de ejecución en 128 nodos de Caesaraugusta compilado con gFortran, es muy similar al de 16 nodos en una máquina de más reciente construcción como es Terminus.

El hecho de utilizar esta máquina es la futura adaptación a simulaciones de gran escala que nos permite esta máquina. De hecho, su caracterización y como se ve en la siguiente sección, será una parte fundamental a analizar antes de ejecutar cualquier simulación con el fin de determinar la duración de la misma.

Capítulo 5
Conclusiones y Trabajo Futuro

5.1 Conclusión del trabajo

La ejecución de aplicaciones paralelas en sistemas que lo soportan es una muy buena forma de disminuir el tiempo de ejecución de éstas, pero es importante conocer las características propias del sistema en el que se va a ejecutar.

Igualmente, la paralelización de una aplicación es una tarea compleja pero que en casos en los que este tipo de estrategia puede funcionar y donde los tiempos de ejecución son largos, da unos resultados muy positivos.

Además el hecho de que la evolución actual de las arquitecturas pase de la evolución que seguía según la Ley de Moore [10] a arquitecturas multi-procesador [11] requiere de una adaptación en la forma de implementar nuestras aplicaciones con el fin de explotar al máximo las capacidades que éstas poseen. No cabe duda de que este paradigma de programación es uno de los que más futuro tiene en adelante.

Es importante además tener en cuenta de que en España, disponemos de una infraestructura muy buena para el cálculo de aplicaciones científicas, que pasa por la RES[1] y a nivel Europeo contamos con otras como la EGEE[2] que favorecen el uso de instalaciones dirigidas a cálculos masivos que requieren de la computación paralela.

5.2 Trabajo Futuro

La paralelización de la aplicación es sólo el principio de la optimización de la herramienta. El hecho de explotar la infraestructura de manera paralela no es lo único que podemos hacer en esta aplicación para sacar más rendimiento. Una de las mejoras que se proponen para la implementación paralelizada y para la secuencial es reorganizar las estructuras de datos para adaptalas a la jerarquía de caches, aumentando su localidad y mejorando la escalabilidad en memoria compartida.

Todos los clúster que se han usado en este trabajo disponen de procesadores que cuentan con unidades de cálculo vectorial, el cual nos daría algo más de rendimiento en secciones muy concretas del código y con una reorganización del mismo, podríamos obtener mucho más utili-

[1] http://www.bsc.es
[2] http://www.eu-egee.org/

zando estas unidades. En esta misma linea, las GPGPUs [3] están muy preparadas para este tipo de cálculos y queda como trabajo futuro el adaptar la aplicación a este tipo de arquitectura.

Otra tarea deseable para completar el trabajo descrito, es el estudio en profundidad de métodos de partición y la consecuente adaptación del código a particiones de tipo 2-D, esto supondría rendimientos superiores a altos números de nodos de cálculo y también encontrar nuevas formulaciones del tiempo de cálculo. Otra alternativa que hemos planteado ha sido la partición según el gradiente de pendientes, que por lógica hidráulica, determinaría la dirección preferente del cauce. En [15] explican formas de discernir estas direcciones preferenciales.

Ademas, es interesante acoplar al algoritmo de partición el modelo matemático desarrollado para calcular el tiempo de ejecución por dos razones: dejar al usuario la libertad de elegir el tiempo que le interesa que dure la simulación, proporcionándole siempre una cota mínima del mismo, y por el hecho de que en infraestructuras diseñadas para este tipo de ejecuciones, es necesario indicar el tiempo aproximado de la duración de la ejecución, con la finalidad de que el scheduler otorgue una prioridad u otra.

5.3 Conclusión personal

Para mí este trabajo, además de un enorme esfuerzo, ha supuesto un reto tanto académico como personal. El primer contacto con la aplicación me hizo pensar en la dificultad que entrañaría la paralelización de la misma, pero igualmente el convencimiento de que se obtendrían unos muy buenos resultados.

Efectivamente y como desde un momento se pensó, la paralelización de una aplicación de estas características es compleja, pero realmente es la única forma de poder conseguir unos tiempos de ejecución razonables. Como hemos visto, estos tiempos de ejecución se han reducido muchísimo respecto a los tiempos que había hasta el momento. Esto ha sido muy bien acogido dentro del Grupo de Investigación de tal manera que en un futuro próximo se pueda replantear el solicitar de nuevo horas de cálculo en infraestructuras de la RES o la compra de algún equipo más para las simulaciones.

Además de todo esto he encontrado muy necesario la comprensión física del problema y la perspectiva de la implementación del método para poder avanzar en el trabajo; ha sido fundamental quitarme las gafas de informático y ponerme desde el punto de vista del modelo para ver más allá de asignaciones o bucles. Desde luego, todo lo aprendido a lo largo de la carrera me ha sido altamente útil, desde el Cálculo o el Álgebra, hasta la Ingeniería del Software o los Fundamentos de Arquitecturas Paralelas.

Estoy muy orgulloso de los resultados del trabajo y considero que el esfuerzo volcado no sólo en este trabajo, sino durante toda la carrera, ha tenido su recompensa.

[3] General Purpose Graphic Processing Unit

Bibliografía

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

[2] B. Chapman, G. Jost, and R. v. d. Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[3] M. J. Flynn and K. W. Rudd. Parallel architectures. *ACM Comput. Surv.*, 28:67–70, March 1996.

[4] P. García-Navarro, P. Brufau, J. Murillo, and J. Burguete. *Volúmenes finitos para ecuaciones hiperbólicas*. 2009.

[5] W. Gropp, R. Thakur, and E. Lusk. *Using MPI-2: Advanced Features of the Message Passing Interface*. MIT Press, Cambridge, MA, USA, 2nd edition, 1999.

[6] J. L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31:532–533, May 1988.

[7] M. T. Health. *Scientific Computing*. McGraw Hill, 1997.

[8] D. Helmbold and C. McDowell. Modelling speedup (n) greater than n. *Parallel and Distributed Systems, IEEE Transactions on*, 1(2):250 –256, Apr. 1990.

[9] A. H. Karp and H. P. Flatt. Measuring parallel processor performance. *Commun. ACM*, 33:539–543, May 1990.

[10] G. E. Moore. Cramming more components onto integrated circuits. *Electronics, Volume 38, Number 8*, April 1965.

[11] G. E. Moore. No exponential is forever. International Solid State Circuits Conference, February 2003.

[12] J. Murillo and P. Garcia-Navarro. Weak solutions for partial differential equations with source terms: Application to the shallow water equations. *JOURNAL OF COMPUTATIONAL PHYSICS Volume: 229 Issue: 11 Pages: 4327-4368*, 2010.

[13] J. Murillo, P. García-Navarro, J. Burguete, and P. Brufau. The influence of source terms on stability, accuracy and conservation in two-dimensional shallow flow simulation using triangular finite volumes. *International Journal for Numerical Methods in Fluids*, 54(5):543–590, 2007.

[14] W. J. P. Silvia M Mueller. On the correctness of hardware scheduling mechanisms for out-of-order execution. *Journal of Circuits, systems and Computers, Volume 8, No. 2*, 1998.

[15] G. M. Stefano Orlandini and M. Franchini. Path-methods for the determination of nondispersive drainage directions in grid-based digital elevation models. *Water Resources Research, VOL. 39, NO. 6, 1144*, June 2003.

Apéndice A
Análisis de compilación del código

La compilación de la aplicación es una parte fundamental en el estudio del rendimiento inicial que podemos obtener sin hacer ninguna optimización en el código, por lo que es interesante prestarle atención a este paso del análisis inicial de la aplicación.

Actualmente, el GHC desarrolla su aplicación en sistemas Windows y Linux con diferentes compiladores para cada Sistema Operativo. En linux, el compilador utilizado hasta el momento es gFortran[1] en su versión 4.4. Éste compilador forma parte de la iniciativa GNU y aparece en los repositorios oficiales de los sistemas operativos utilizados en el grupo (Debian ó Ubuntu). En Windows, el compilador que se utiliza es Lahey Fortran 90 [2] en su versión v.4.5 para 32 bits. Esta versión están en desuso pero el precio de la herramienta y su amigable entorno de programación hace que esta herramienta todavía tenga un peso importante dentro de los usuario de Windows del grupo.

Los equipos que se utilizan a nivel usuario, son todos Intel Core 2 Duo en diferentes variantes, por lo que sabemos que la familia es la Intel®Core 2 que funcionan en 64 bits.

Se considera por tanto, que es interesante analizar el rendimiento del compilador de Intel®[3] con diferentes optimizaciones.

A.1 Análisis de la compilación

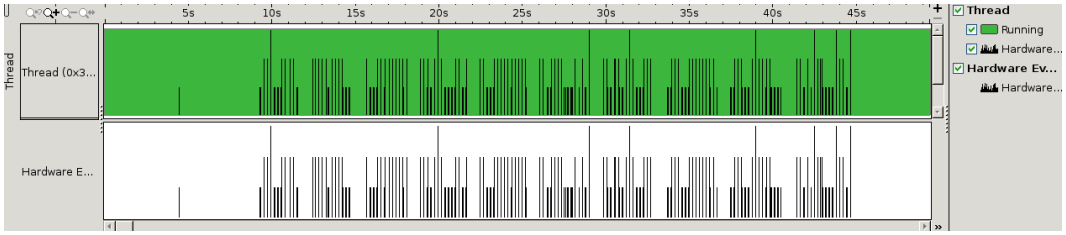
En una primera aproximación al estudio del programa es interesante conocer dónde está perdiendo tiempo el código y ver qué se puede hacer con ello. Para esto vamos a utilizar una herramienta comercial, Intel®vTune Amplifer XE for Linux. Es importante aclarar que por ser para una labor académica no remunerada, me voy a acoger a la licencia non-commercial [4]. Además de la citada herramienta, utilizaremos el compilador de Intel®para este procedimiento.

El análisis se hará con el caso C.2 simulando 100 s. para evitar que la sobrecarga inicial de preprocesamiento de la malla falsee los resultados. Esta premisa la mantendremos a lo largo del trabajo ya que esa sobrecarga inicial se puede reducir optimizando esa sección y por que entre otras cosas, esa tarea no es paralelizable y por lo tanto, no va a poder reducirse con las estrategias

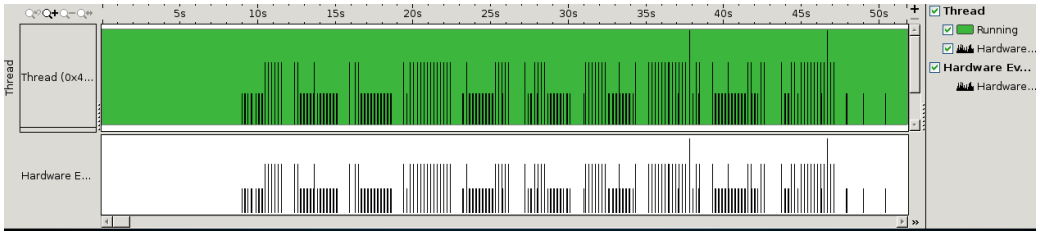
[1] http://gcc.gnu.org/fortran/
[2] http://www.lahey.com/
[3] http://software.intel.com/en-us/articles/intel-composer-xe/
[4] http://software.intel.com/en-us/articles/non-commercial-software-development/

aquí sugeridas. Como podemos observar en el análisis que nos ha proporcionado la herramienta, la función donde más tiempo pierde es en *blocks*1, que es la función dedicada a ejecutar el núcleo del cálculo.

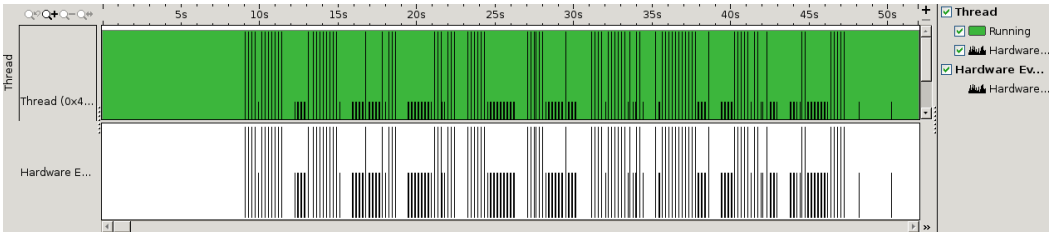
Si analizamos un poco más en profundidad, podemos comprobar los fallos en cache L2 (que son los que nos proporciona la herramienta) y los fallos en cache de instrucciones, y como podemos ver, donde más fallos se producen es en los periodos de captura de datos de blocks1 para el caso de la cache de datos, y durante todo el periodo en la cache de instrucciones. Estos resultados sin ninguna optimización, son muy similares a los datos con la optimización -O3.



(a) Sin optimización



(b) Optimización -O1



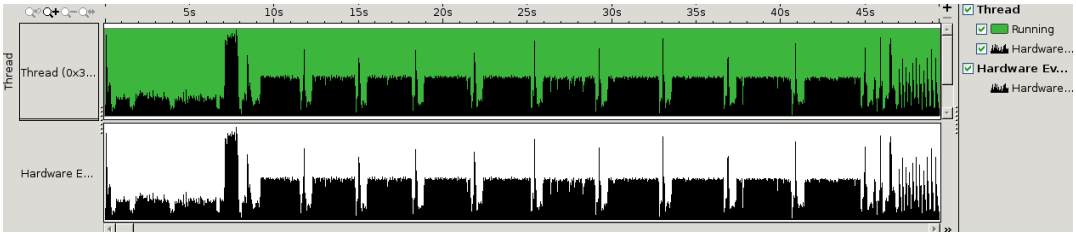
(c) Optimización -O3

Figura A.1: Análisis de cache L2 para diferentes optimizaciones

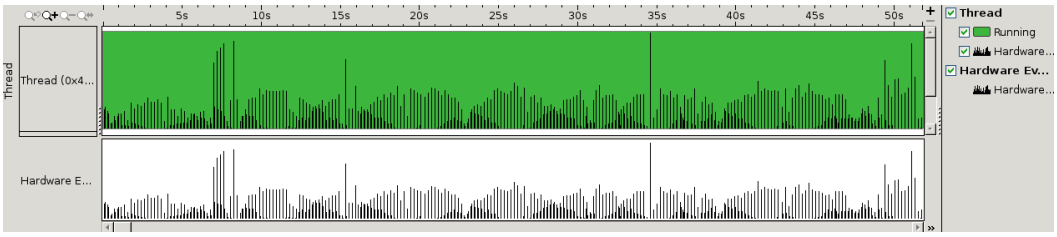
En cambio para la optimización -O1[5] la caché de instrucciones y la cache de datos falla con menos frecuencia. La explicación para esto es que la optimización -01 es en tamaño, y está orientado precisamente a producir menos instrucciones.

Estos resultados no dejan de ser una mera orientación para comprender cómo funciona el programa de una manera superficial dado que un análisis más exhaustivo supondría un trabajo adicional fuera del ámbito de este PFC.

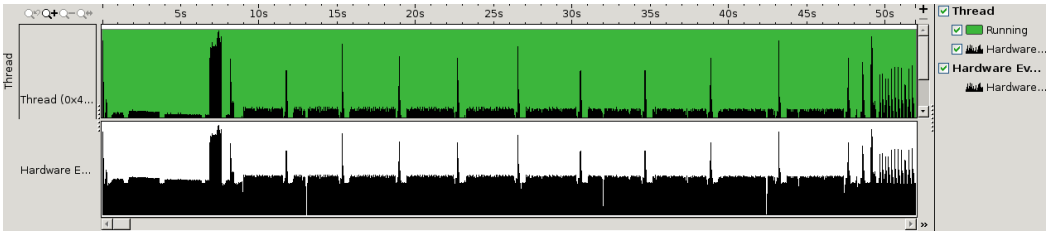
[5] -O1 es la optimización en tamaño del ejecutable



(a) Sin optimización



(b) Optimización -O1



(c) Optimización -O3

Figura A.2: Análisis de cache de instrucciones para diferentes optimizaciones

A.2 Rendimiento de los compiladores

En esta sección voy a exponer los rendimientos que tenían hasta ahora las compilaciones que se hacían en el grupo con el fin de hacernos una idea cuantitativa de lo que cuesta hacer las simulaciones. Sin estos resultados no tiene sentido plantear ningún tipo de optimización en la implementación.

Una primera aproximación es estudiar el rendimiento de los compiladores que se utilizan hasta ahora. Como podemos ver en la figura A.3 la diferencia de tiempos de ejecución es notoria. Esto puede deberse al código generado por el compilador (en términos de eficiencia del código generado) o a que se generen más pasos de iteración. Para ello podemos comparar los dt de ambas simulaciones como vemos en la figura pero realmente las diferencias no son tan notorias como para que puedan producir muchos mas pasos de ejecución. Las diferencias que ahora estamos viendo, se deben a los diferentes redondeos que provocarán cada compilación. Hemos de pensar que la variable w (ver anexo B) alcanza valores infinitesimales. Una diferencia entre uno y en esta variable puede producir una ligera variación en la velocidad u, v y por lo tanto hacer variar el paso de tiempo.

Como vemos, gfortran saca cuantitativamente más rendimiento, así que voy a comparar los rendimientos de gFortran con el compilador de Intel. El caso seleccionado para esta comparación, es el caso de la figura 3.5, simulado desde un estado estacionario con un hidrograma de entrada como el que vemos en la figura A.4.

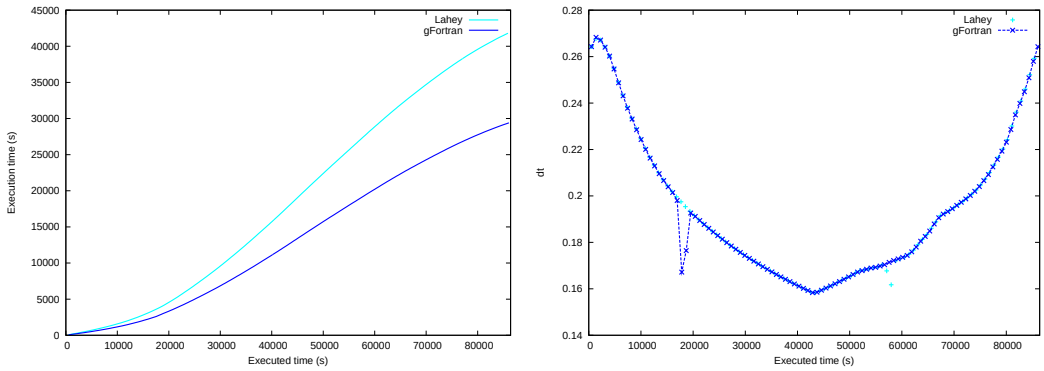
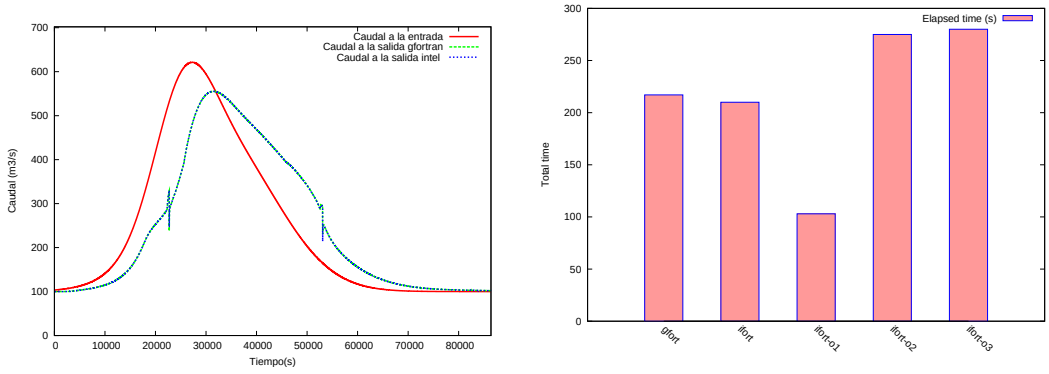


Figura A.3: Evolución temporal en tiempo acumulado y *dt* de la simulación con diferentes compiladores



(a) Hidrograma E/S **(b)** Tiempo de CPU de blocks1

Figura A.4: Tiempo de ejecución y comparación de resultados de gFortran e Intel Fortran Compiler

En esta misma figura podemos ver los tiempos de ejecución de ambos compiladores. En este caso, no hemos comprobado las diferentes optimizaciones de gFortran por que lo que pretendíamos ver era la supuesta mejora que nos iba a asegurar el compilador de Intel frente al de GNU, pero como vemos tampoco es abismal. Tal y como habíamos visto en el análisis de la sección anterior, la optimización -01 maneja mejor la cache de instrucciones, y esto hace disminuir de manera notable el tiempo de ejecución, con lo que para este caso en particular, preferiremos la compilación con este flag. Además, también se han probado las compilaciones con el juego de instrucciones SSE4 y otras, pero no han representado ninguna diferencia destacable.

Apéndice B
Modelo 2D de flujo de lámina libre con promedio en la vertical

Este capítulo es un estracto de [4].

B.1 Ecuaciones generales

Las ecuaciones que describen el flujo tridimensional de superficie libre pueden ser obtenidas a partir de las ecuaciones de Navier-Stokes que expresan los principios físicos de conservación de la masa y conservación del movimiento en las tres direcciones del espacio:

$$\nabla \mathbf{v} = 0 \tag{B.1}$$

$$\begin{aligned} \frac{\partial u}{\partial t} + \nabla(u\mathbf{v}) + \frac{\partial p}{\partial x} - \frac{\partial \tau_{xx}}{\partial x} - \frac{\partial \tau_{xy}}{\partial y} - \frac{\partial \tau_{xz}}{\partial z} = 0 \\ \frac{\partial v}{\partial t} + \nabla(v\mathbf{v}) + \frac{\partial p}{\partial y} - \frac{\partial \tau_{xy}}{\partial x} - \frac{\partial \tau_{yy}}{\partial y} - \frac{\partial \tau_{yz}}{\partial z} = 0 \\ \frac{\partial w}{\partial t} + \nabla(w\mathbf{v}) + \rho g + \frac{\partial p}{\partial z} - \frac{\partial \tau_{xz}}{\partial x} - \frac{\partial \tau_{yz}}{\partial y} - \frac{\partial \tau_{zz}}{\partial z} = 0 \end{aligned} \tag{B.2}$$

Puesto que las variaciones de densidad son despreciables o nulas, no existen vínculos entre la ecuación de conservación de la energía y las de la masa y movimiento y en este caso son suficientes las dos últimas ecuaciones para conocer la evolución del flujo.

Para fijar las soluciones de las ecuaciones diferenciales (B.1) y (B.2) es preciso definir las condiciones de contorno en las fronteras. En este caso existen dos zonas: la entrefase fluido-sólido (fondo) que es fija y la entrefase fluido-fluido (superficie libre) que puede cambiar continuamente. Las condiciones de frontera en ambas superficies son de dos tipos: cinemáticas y dinámicas. Primero vamos a plantear las condiciones cinemáticas en el fondo y en la superficie libre y después haremos lo mismo con las condiciones de frontera de tipo dinámico.

Las condiciones cinemáticas están relacionadas con la velocidad y nos dicen que las partículas de agua en su movimiento no pueden cruzar ninguna frontera. Para el fondo significa que la componente de la velocidad normal a la superficie sólida debe ser cero (fondo sólido, impermeable y fijo).

$$\mathbf{v}\hat{\mathbf{n}}_b = u\frac{\partial z_b}{\partial x} + v\frac{\partial z_b}{\partial y} - w = 0 \tag{B.3}$$

con $\hat{\mathbf{n}}_b = (\partial z_b/\partial x, \partial z_b/\partial y, -1)$ el vector normal a la superficie líquida en contacto con la sólida hacia afuera en $z = z_b\,(x, y)$, donde z_b es la cota del fondo medida desde un nivel de referencia horizontal (Fig. B.1).

En la superficie libre las cosas son un poco más complicadas ya que ésta se puede mover. En este caso lo que debe ser nula es la velocidad normal relativa a esa superficie y representa que el fluido no se puede salir del propio fluido, no puede atravesar tampoco la superficie libre

$$\frac{\partial H}{\partial t} + u\frac{\partial H}{\partial x} + v\frac{\partial H}{\partial y} - w = 0 \tag{B.4}$$

en $z = H\,(x, y, t)$, donde H es el nivel de la superficie libre medido desde el nivel de referencia (Fig. B.1).

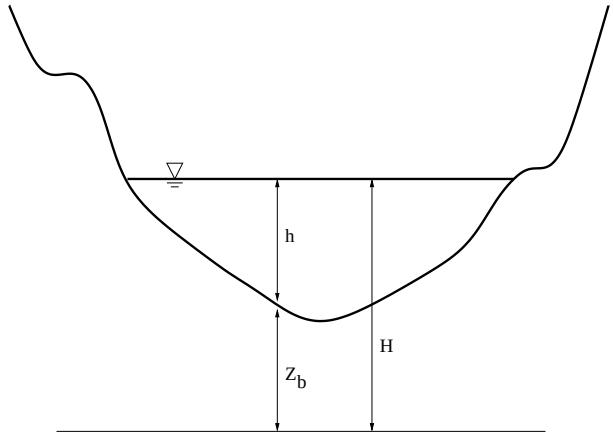


Figura B.1: Perfil del cauce.

Queremos hacer notar aquí que el cauce puede tener una pendiente elevada tanto en dirección x como en dirección y y por ello hay que elegir bien los ejes de referencia sobre los cuales se van a tomar medidas. En este caso, lo que se hace es suponer una referencia horizontal arbitraria para simplificar notablemente la formulación y se considera que el ángulo que define la pendiente es casi despreciable.

Las condiciones de contorno dinámicas nos dan información sobre las fuerzas que actúan en los contornos. Si el flujo es viscoso y el fondo fijo, la fuerza que actúa es la de la viscosidad y por lo tanto las partículas que se encuentran en contacto con el fondo están pegadas a él, por lo cual puede imponerse la condición de no deslizamiento que conduce a:

$$u = v = 0 \tag{B.5}$$

en $z = z_b(x, y)$.

En la superficie libre se supone continuidad de esfuerzos; es decir, los esfuerzos en el fluido justo por debajo de la superficie libre son los mismos que los del aire justo encima. Estos esfuerzos pueden ser de dos tipos: normales y cortantes ó tangentes a la superficie. En el caso de los esfuerzos normales a la superficie donde interviene el término de presión, despreciando los efectos de la tensión superficial,

$$P = P_a \tag{B.6}$$

donde P_a es la presión atmosférica. El nivel absoluto de presiones no es importante y se puede tomar como cero. Las diferencias sólo podrían ser importantes en el caso de que se quisiera estudiar el efecto de las variaciones de presión atmosférica en el movimiento del agua.

Los esfuerzos tangenciales que actúan en la superficie libre son debidos a esfuerzos viscosos y la condición de contorno nos dice que deben ser iguales al esfuerzo tangencial aplicado al otro lado de la frontera de la superficie libre y que puede ser provocado por el viento. De este modo,

el modelo contempla que en la superficie del mar, por ejemplo, puede actuar un esfuerzo cortante debido al viento. Este esfuerzo cortante externo $\boldsymbol{\tau}_s = (\tau_{sx}, \tau_{sy})$ tangente a la superficie del agua es

$$\tau_{sx} = (\mathcal{T} \cdot \mathbf{n}_H)_x = -\tau_{xx}\frac{\partial H}{\partial x} - \tau_{xy}\frac{\partial H}{\partial y} + \tau_{xz} \tag{B.7}$$

en $z = H$ y de forma similar para la dirección y. El vector de esfuerzos producido por el viento se supone conocido y se trata como una fuerza externa. La magnitud y la dirección de la fuerza del viento en la superficie del mar vienen determinadas por el flujo en la atmósfera. Normalmente se supone que el módulo de la velocidad del viento es conocida W y se acepta la fórmula semi-empírica dada por Gill,

$$\tau_s = \rho c_W W^2 \tag{B.8}$$

y la dirección se supone que es la dirección de la velocidad que lleva el viento. El coeficiente c_W no es constante, depende de la velocidad del viento; por ejemplo, es del orden de 0,001 si la velocidad del viento se mide a unos 10 m de altura.

Resolver este sistema de ecuaciones es muy costoso computacionalmente ya que hace falta resolver la ecuación tridimensional de Poisson para obtener la distribución de presiones en cada paso temporal. La condición de contorno de la superficie libre implica una no linealidad extra en el problema y para muchas aplicaciones se prefiere resolver el sistema de ecuaciones de aguas poco profundas que se obtiene siguiendo una serie de aproximaciones.

Después de realizar un estudio de las escalas características del problema llegamos a las ecuaciones de aguas poco profundas tridimensionales que reescribimos a continuación:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{B.9}$$

$$\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} = -g\frac{\partial h}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} \tag{B.10}$$

$$\frac{\partial v}{\partial t} + \frac{\partial uv}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} = -g\frac{\partial h}{\partial y} + \frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} \tag{B.11}$$

$$\frac{\partial p}{\partial z} + \rho g = 0 \tag{B.12}$$

Se hace notar que con las ecuaciones de movimiento (B.10), (B.11) podemos conocer los valores de u y v, w la obtendríamos a partir de la ecuación de conservación de la masa (B.9) y, por último, la variable h tendría que ser determinada por las condiciones de contorno de la superficie libre (B.4), siendo aún así un procedimiento todavía complicado.

Por esto se recurre al promedio en la vertical cuya hipótesis fundamental es que Las ondas que se producen en la superficie varían suavemente, lo cual es equivalente a decir que la distribución de presiones en la vertical es hidrostática o que la aceleración en la vertical es pequeña.

B.2 Ecuaciones promediadas en la vertical

Para pasar a la forma bidimensional de las ecuaciones, dejando la profundidad como variable dependiente, hay que dar un paso más. Con el objetivo de eliminar de las ecuaciones la información

del movimiento en la dirección vertical z se promedian las ecuaciones en esta dirección haciendo uso de las definiciones de los promedios de las variables.

$$\bar{u} = \frac{1}{h}\int_{z_b}^{H} u dz \tag{B.13}$$

$$\bar{v} = \frac{1}{h}\int_{z_b}^{H} v dz \tag{B.14}$$

El proceso de promediar en la vertical las ecuaciones convierte el problema tridimensional en uno bidimensional de grosor variable h donde los contornos ya no están en la superficie libre y el fondo sino en el perímetro.

El promedio en la vertical de las ecuaciones del flujo de superficie libre bajo las hipótesis del modelo de aguas poco profundas conduce a una versión muy común del sistema de ecuaciones en 2D que repetimos aquí:

$$\frac{\partial h}{\partial t} + \frac{\partial (hu)}{\partial x} + \frac{\partial (hv)}{\partial y} = 0$$

$$\frac{\partial (hu)}{\partial t} + \frac{\partial (hu^2)}{\partial x} + \frac{\partial (huv)}{\partial y} = -gh\frac{\partial H}{\partial x} + c_f u\sqrt{u^2+v^2} + h\nu_{\mathbf{T}}\nabla^2 u$$

$$\frac{\partial (hv)}{\partial t} + \frac{\partial (huv)}{\partial x} + \frac{\partial (hv^2)}{\partial y} = -gh\frac{\partial H}{\partial y} + c_f v\sqrt{u^2+v^2} + h\nu_{\mathbf{T}}\nabla^2 v$$

B.2.1 Término de fricción y modelos de turbulencia

El coeficiente c_f que aparece en el término de fricción se expresa habitualmente en términos del coeficiente de rugosidad de Manning n o de Chézy,

$$c_f u\sqrt{u^2+v^2} = \frac{n^2 u\sqrt{u^2+v^2}}{h^{\frac{4}{3}}} \tag{B.15}$$

$$c_f v\sqrt{u^2+v^2} = \frac{n^2 v\sqrt{u^2+v^2}}{h^{\frac{4}{3}}} \tag{B.16}$$

El coeficiente de rugosidad n en la práctica se determina a partir de medidas experimentales o se estima a partir de valores que ya han sido almacenados en tablas. La ecuación de Manning aquí descrita es de naturaleza empírica y por tanto es el resultado de un proceso de ajuste a una curva de datos experimentales. La primera dificultad que surge a la hora de usar este coeficiente de rugosidad es la precisión con la que ha sido estimado. El coeficiente n depende en principio del número de Reynolds del flujo, de la rugosidad de los contornos y de la forma geométrica de la cuenca. La rugosidad de la superficie del contorno representa un valor crítico a la hora de estimar n, con valores pequeños si el material es fino y valores altos en el caso contrario. El valor de n también debe de dar cuenta de la vegetación retardando el flujo y proporcionando valores altos de n, dependiendo también de la altura de agua. El modelo de fricción dado por (B.15) y (B.16) se basa en la teoría de capa límite estacionaria sobre pared rugosa.

Con el objeto de calcular las variables hidrodinámicas, es necesario fijar el valor del coeficiente de Manning, como ya hemos dicho, y el valor de la viscosidad cinemática de remolino. La viscosidad turbulenta $\nu_{\mathbf{T}}$ depende de las características del flujo y puede variar de un punto a otro del dominio. Por tanto, es necesario plantear un modelo de turbulencia que nos permita evaluar el valor de $\nu_{\mathbf{T}}$ en cada punto del dominio.

B.2.2 Versión común de las ecuaciones de aguas poco profundas

El término que proviene de promediar en la vertical el gradiente de presión ha dado lugar a los términos $g\partial H/\partial x$, $g\partial H/\partial y$, que a su vez se pueden descomponer, teniendo en cuenta que $H = h + z_b$ en

$$g\frac{\partial H}{\partial x} = g\frac{\partial h}{\partial x} + g\frac{\partial z_b}{\partial x} \tag{B.17}$$

$$g\frac{\partial H}{\partial y} = g\frac{\partial h}{\partial y} + g\frac{\partial z_b}{\partial y} \tag{B.18}$$

Los términos $\partial h/\partial x$, $\partial h/\partial y$ se agrupan junto a las otras derivadas del mismo tipo (términos convectivos). Las variaciones del fondo se expresan en forma de pendiente

$$S_{0x} = -\frac{\partial z_b}{\partial x} \tag{B.19}$$

$$S_{0y} = -\frac{\partial z_b}{\partial y} \tag{B.20}$$

Los términos de fricción del agua con el fondo del cauce se representan por S_f, pendiente de la línea de energía en cada dirección

$$S_{fx} = \frac{c_f u\sqrt{u^2+v^2}}{gh} \tag{B.21}$$

$$S_{fy} = \frac{c_f v\sqrt{u^2+v^2}}{gh} \tag{B.22}$$

dando lugar al siguiente sistema de ecuaciones que es la forma más conocida de representación del modelo de aguas poco profundas

$$\frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} + \frac{\partial hv}{\partial y} = 0 \tag{B.23}$$

$$\frac{\partial hu}{\partial t} + \frac{\partial hu^2}{\partial x} + gh\frac{\partial h}{\partial x} + \frac{\partial huv}{\partial y} = gh(S_{0x} - S_{fx}) \tag{B.24}$$

$$\frac{\partial hv}{\partial t} + \frac{\partial huv}{\partial x} + \frac{\partial hv^2}{\partial y} + gh\frac{\partial h}{\partial y} = gh(S_{0y} - S_{fy}) \tag{B.25}$$

Este sistema de ecuaciones en su forma conservativa es decir, escritas las ecuaciones de la forma más cercana posible a un sistema de leyes de conservación de masa y cantidad de movimiento, es

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \mathbf{E} = \mathbf{S} \Rightarrow \frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot (\mathbf{F}, \mathbf{G}) = \mathbf{S} \tag{B.26}$$

con

$$\mathbf{U} = \begin{pmatrix} h \\ hu \\ hv \end{pmatrix}, \ \mathbf{F} = \begin{pmatrix} hu \\ hu^2 + g\frac{h^2}{2} \\ huv \end{pmatrix}, \ \mathbf{G} = \begin{pmatrix} hv \\ huv \\ hv^2 + g\frac{h^2}{2} \end{pmatrix},$$
$$\mathbf{S} = \begin{pmatrix} 0 \\ gh\,(S_{0x} - S_{fx}) \\ gh\,(S_{0y} - S_{fy}) \end{pmatrix} \tag{B.27}$$

$\mathbf{U}$ representa el vector de variables conservadas (h profundidad del agua (Fig. B.1), hu y hv caudales unitarios a lo largo de las direcciones coordenadas x, y respectivamente), $\mathbf{F}$ y $\mathbf{G}$ son los flujos de las variables conservadas a través de los lados de un volumen de control, y contienen el flujo convectivo y los gradientes de presión hidrostática. La parte derecha de la igualdad en el sistema de ecuaciones, $\mathbf{S}$, contiene las fuentes y sumideros de la cantidad de movimiento a lo largo de las dos direcciones coordenadas, provenientes de las variaciones del fondo del cauce y de las pérdidas por fricción que deben estar relacionadas con el campo de velocidades.

De esta manera, (B.26) representa un sistema hiperbólico de ecuaciones diferenciales en derivadas parciales acopladas y no lineales. Si escribimos el sistema de ecuaciones en formulación no conservativa

$$\frac{\partial \mathbf{U}}{\partial t} + (\mathbf{A}, \mathbf{B}) \cdot \nabla \mathbf{U} = \mathbf{S} \tag{B.28}$$

las matrices Jacobianas de los vectores de flujo son

$$\mathbf{A} = \frac{\partial \mathbf{F}}{\partial \mathbf{U}} = \begin{pmatrix} 0 & 1 & 0 \\ c^2 - u^2 & 2u & 0 \\ -uv & v & u \end{pmatrix}, \quad \mathbf{B} = \frac{\partial \mathbf{G}}{\partial \mathbf{U}} = \begin{pmatrix} 0 & 0 & 1 \\ -uv & v & u \\ c^2 - v^2 & 0 & 2v \end{pmatrix} \tag{B.29}$$

y la matriz Jacobiana del flujo normal a una dirección dada por $\hat{\mathbf{n}}$ se puede escribir como

$$\mathcal{A} = \mathbf{A}\hat{n}_x + \mathbf{B}\hat{n}_y = \begin{pmatrix} 0 & \hat{n}_x & \hat{n}_y \\ -u(\mathbf{u}\cdot\hat{\mathbf{n}}) + c^2\hat{n}_x & \mathbf{u}\cdot\hat{\mathbf{n}} + u\hat{n}_x & u\hat{n}_y \\ -v(\mathbf{u}\cdot\hat{\mathbf{n}}) + c^2\hat{n}_y & v\hat{n}_x & \mathbf{u}\cdot\hat{\mathbf{n}} + u\hat{n}_y \end{pmatrix} \tag{B.30}$$

Los valores propios del Jacobiano $\mathbf{J_n}$ son

$$\begin{aligned} a^1 &= \mathbf{u}\cdot\hat{\mathbf{n}} + c \\ a^2 &= \mathbf{u}\cdot\hat{\mathbf{n}} \\ a^3 &= \mathbf{u}\cdot\hat{\mathbf{n}} - c \end{aligned} \tag{B.31}$$

y sus vectores propios

$$\mathbf{e}^1 = \begin{pmatrix} 1 \\ u + c\hat{n}_x \\ v + c\hat{n}_y \end{pmatrix}, \ \mathbf{e}^2 = \begin{pmatrix} 0 \\ -c\hat{n}_y \\ c\hat{n}_x \end{pmatrix}, \ \mathbf{e}^3 = \begin{pmatrix} 1 \\ u - c\hat{n}_x \\ v - c\hat{n}_y \end{pmatrix} \tag{B.32}$$

B.3 Esquema numérico

El dominio donde se mueve el flujo, se subdivide, en un conjunto de celdas para su resolución numérica. En el modelo presentado hay libertad a la hora de elegir el tipo de celdas: hexágonos, cuadriláteros, triángulos, etc... y además pueden formar parte de una malla estructurada o de una malla no estructurada. La elección de la malla es un factor importante en la simulación numérica.

Respecto a la técnica de resolución de las ecuaciones, se ha usado un método de volúmenes finitos porque combina lo mejor de los métodos de elementos finitos y su flexibilidad geométrica, con lo mejor de los métodos en diferencias finitas, su flexibilidad en la definición del flujo discreto (valores discretos de las variables dependientes y sus flujos asociados). El primer paso es escribir (B.28) en la forma

$$\frac{\partial \mathbf{U}}{\partial t} + \overrightarrow{\nabla}\mathbf{E} = \mathbf{S} \tag{B.33}$$

donde $\mathbf{E} = (\mathbf{F}, \mathbf{G})^T$.

Aplicando el teorema de Gauss sobre la celda de cálculo Ω_i fija en el tiempo, (B.33) se escribe como:

$$\frac{\partial}{\partial t}\int_{\Omega_i}\mathbf{U}d\Omega_i + \oint_{\partial\Omega_i}\mathbf{En}dl = \oint_{\partial\Omega_i}\mathbf{Tn}dl \tag{B.34}$$

donde $\mathbf{Tn}$ es un vector que expresa el término fuente a través de la superficie $\partial\Omega$, l denota la variable de integración de la superficie alrededor del volumen Ω_i y $\mathbf{n}$ es el vector exterior normal unitario.

Una vez formulado el problema en volúmenes finitos se ha de elaborar una estrategia adecuada para calcular el flujo numérico a través de la superficie. La forma hiperbólica del sistema de ecuaciones hace que este problema sea resuelto adecuadamente utilizando un esquema numérico perteneciente a la familia de los métodos de Godunov. Este tipo de método calcula el flujo numérico que actualiza el valor de cada celda de cálculo i promediando el valor de las diferentes soluciones aproximadas que aparecen al definir un problema de Riemann en la superficie entre el volumen y cada uno de los volúmenes vecinos j. Dentro las posibles opciones que permiten generar una solución aproximada, en este trabajo se utiliza la aproximación propuesta por Roe, que a diferencia de otras considera todas las velocidades de propagación de información contenidas en el Jacobiano de la matriz. Cuando aparecen términos fuente formularlos a través de una matriz en la pared, como en (B.35), permite desarrollar soluciones aproximadas más complejas adecuadamente. De esta manera, el flujo normal $\mathbf{En}$ y su jacobiano cobran protagonismo. Se evalúa la matriz jacobiana del flujo normal y se diagonaliza, permitiendo que el esquema numérico se base en vectores y valores propios.

La matriz Jacobiana J_n será la base para llevar a cabo la discretización numérica que se presenta en este trabajo. Para comenzar la técnica de volúmenes finitos, el sistema (B.28) se integra en el volumen o malla de celdas, Ω :

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{U}d\Omega + \int_{\Omega}(\overrightarrow{\nabla}\mathbf{E})d\Omega = \int_{\Omega}\mathbf{S}d\Omega \tag{B.35}$$

se asume que la tercera integral se puede formular de la siguiente manera:

$$\int_{\Omega}\mathbf{S}d\Omega = \oint_{\partial\Omega}(\mathbf{Tn})dl \tag{B.36}$$

donde $\mathbf{T}$ es una matriz adecuada. Esto lleva a la siguiente formulación:

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{U}d\Omega + \oint_{\partial\Omega}\mathbf{En}dl = \oint_{\partial\Omega}\mathbf{Tn}dl \tag{B.37}$$

Cuando el dominio se subdivide en celdas Ω_i en una malla fija en el tiempo, Figura B.2, la ecuación (B.37) también se puede aplicar a cada celda.

$$\frac{\partial}{\partial t}\int_{\Omega}\mathbf{U}_i d\Omega_i + \sum_{k=1}^{NE}\int_{e_k}^{e_{k+1}}\mathbf{E}_j\mathbf{n}_k l_k = \sum_{k=1}^{NE}\int_{e_k}^{e_{k+1}}\mathbf{Tn}_k l_k \tag{B.38}$$

En el primer orden las cantidades vectoriales son uniformes en cada celda y la ecuación (B.37) queda reducida a:

$$\frac{(\mathbf{U}_i^{n+1}-\mathbf{U}_i^n)}{\Delta t}A_i + \sum_{k=1}^{NE}(\delta\mathbf{E}-\mathbf{T})_k\mathbf{n}_k l_k = 0 \tag{B.39}$$

donde $\delta\mathbf{E} = \mathbf{E}_j - \mathbf{E}_i$, con $\mathbf{E}_j$ y $\mathbf{E}_i$ el valor de la función $\mathbf{E}$ en la celda vecina j y en la celda i respectivamente y conectadas a través del lado k, $\mathbf{n}_k$ es el vector normal hacia afuera del borde

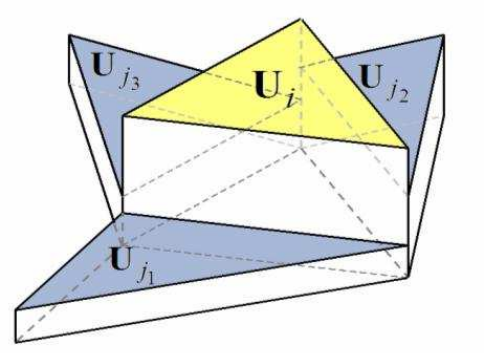


Figura B.2: Representación constante de las varaibles en cada celda.

de la celda k, l_k es la longitud correspondiente a dicho borde, NE es el número de bordes que necesita para definir la celda y $\mathbf{T}_k$ es término fuente evaluado en la pared.

Debido al carácter no lineal del flujo $\mathbf{E}$, el Jacobiano aproximado, $\widetilde{\mathbf{J}}_{\mathbf{n},k}$ permite una linealización local

$$\delta(\mathbf{E}\ n) = \widetilde{\mathbf{J}}_{\mathbf{n},k}\ \delta U \tag{B.40}$$

dando lugar a un sistema con 3 valores propios reales $\widetilde{\lambda}_k^m$ y vectores propios $\widetilde{\mathbf{e}}_k^m$, que se construyen con las siguientes variables promedio

$$\widetilde{u}_k = \frac{u_i\sqrt{h_i} + u_j\sqrt{h_j}}{\sqrt{h_i} + \sqrt{h_j}} \quad \widetilde{v}_k = \frac{v_i\sqrt{h_i} + v_j\sqrt{h_j}}{\sqrt{h_i} + \sqrt{h_j}} \quad \widetilde{c}_k = \sqrt{g\frac{h_i + h_j}{2}} \tag{B.41}$$

dando lugar a

$$\widetilde{\lambda}_k^1 = (\widetilde{\mathbf{u}}\mathbf{n} + \widetilde{c})_k \quad \widetilde{\lambda}_k^2 = (\widetilde{\mathbf{u}}\mathbf{n})_k \quad \widetilde{\lambda}_k^3 = (\widetilde{\mathbf{u}}\mathbf{n} - \widetilde{c})_k \tag{B.42}$$

y

$$\widetilde{\mathbf{e}}_k^1 = \begin{pmatrix} 1 \\ \widetilde{u} + \widetilde{c}n_x \\ \widetilde{v} + \widetilde{c}n_y \end{pmatrix}_k \quad \widetilde{\mathbf{e}}_k^2 = \begin{pmatrix} 1 \\ -\widetilde{c}n_y \\ -\widetilde{c}n_x \end{pmatrix}_k \quad \widetilde{\mathbf{e}}_k^3 = \begin{pmatrix} 1 \\ \widetilde{u} - \widetilde{c}n_x \\ \widetilde{v} - \widetilde{c}n_y \end{pmatrix}_k \tag{B.43}$$

Las matrices $\widetilde{\mathbf{P}}_k$ y $\widetilde{\mathbf{P}}_k^{-1}$, se pueden construir a partir de los vectores propios $\widetilde{\mathbf{e}}_k^m$ de $\widetilde{\mathbf{J}}_{\mathbf{n},k}$ de forma que la diagonalicen

$$\widetilde{\mathbf{J}}_{\mathbf{n},k} = (\widetilde{\mathbf{P}}\mathbf{\Lambda}\widetilde{\mathbf{P}}^{-1})_k \qquad \widetilde{\mathbf{P}}_k = \begin{pmatrix} \widetilde{\mathbf{e}}_k^1 & \widetilde{\mathbf{e}}_k^2 & \widetilde{\mathbf{e}}_k^3 \end{pmatrix} \tag{B.44}$$

donde $\widetilde{\lambda}_k^m$ son los valores propios de la matriz diagonal Λ. También, a partir de la matriz Jacobiana aproximada

$$\widetilde{\mathbf{J}}_{\mathbf{n},k}\widetilde{\mathbf{e}}_k^m = \widetilde{\lambda}_k^m\widetilde{\mathbf{e}}_k^m \qquad m = 1, 2, 3 \tag{B.45}$$

El problema se reduce a un problema unidimensional proyectado sobre la dirección $\mathbf{n}$ en cada borde de celda. Además, la diferencia del vector $\delta\mathbf{U}$ a través de cada lado de la celda se proyecta sobre la base de vectores propios

$$\delta\mathbf{U}_k = \sum_{m=1}^{3} (\alpha\widetilde{\mathbf{e}})_k^m \tag{B.46}$$

Donde las expresiones que dan los coeficientes α_k son:

$$\alpha_k^{1,3} = \frac{\delta h_k}{2} \pm \frac{1}{2\widetilde{c}_k} \left(\delta \mathbf{q}_k - \widetilde{\mathbf{u}}_k \delta h_k\right) \mathbf{n}_k \qquad \alpha_k^2 = \frac{1}{2\widetilde{c}_k} \left(\delta \mathbf{q}_k - \widetilde{\mathbf{u}}_k \delta h_k\right) \mathbf{n}_{T,k} \tag{B.47}$$

con $\mathbf{n}_{T,k} = (-n_y, n_x)$. La contribución de $\delta(\mathbf{En})_k$ en una celda k se puede escribir como:

$$\delta(\mathbf{En})_k = \sum_{m=1}^{3} (\widetilde{\lambda}\alpha\widetilde{\mathbf{e}})_k^m l_k \tag{B.48}$$

Siguiendo la discretización unificada, los términos fuente $(\mathbf{Tn})_k$ se escriben de la siguiente manera:

$$(\mathbf{Tn})_k = \begin{pmatrix} 0 & -g\widetilde{h}(\delta z + d_{\mathbf{n}} S_f) n_x & -g\widetilde{h}(\delta z + d_{\mathbf{n}} S_f) n_y \end{pmatrix}_k^T \tag{B.49}$$

donde siguiendo

$$S_{f,k} = \left(\frac{n^2 \widetilde{\mathbf{u}}\mathbf{n} |\widetilde{\mathbf{u}}| d_{\mathbf{n}}}{\text{máx}(h_i, h_j)^{4/3}} \right)_k \tag{B.50}$$

siendo $d_{\mathbf{n}}$ es la distancia entre los centroides de las celdas que comparten el lado k proyectado sobre la dirección n.

La pendiente del fondo y el término de fricción se pueden descomponer en la base de los vectores propios con el objetivo de asegurar el equilibrio discreto con los términos de flujo (B.48), de tal manera que se asegura en los casos estacionarios con velocidad nula y no nula:

$$(\mathbf{Tn})_k = \widetilde{\mathbf{P}}_k \mathbf{B}_k = \sum_{m=1}^{3} (\beta^m \widetilde{\mathbf{e}}^m)_k \tag{B.51}$$

con $\mathbf{B}_k = \begin{pmatrix} \beta_1 & \beta_2 & \beta_3 \end{pmatrix}_k^T$. Los coeficientes son

$$\beta_k^{1,3} = \mp \frac{\widetilde{c}_k}{2} (\delta z + d_{\mathbf{n}} S_f)_k \quad \beta_k^2 = 0 \tag{B.52}$$

El esquema descentrado explícito de primer orden toma la forma

$$\mathbf{U}_i^{n+1} = \mathbf{U}_i^n + \Delta t \sum_{k=1}^{NE} \mathbf{\Psi}_{i,k}^n \tag{B.53}$$

donde la contribución de cada lado de la celda, $\mathbf{\Psi}_{i,k}$, sólo recoge la información en la dirección entrante, Figura (B.3):

$$\mathbf{\Psi}_{i,k} = \sum_{m=1}^{3} ((\widetilde{\lambda}^- \alpha - \beta^-)\widetilde{\mathbf{e}})_k^m l_k / A_i \tag{B.54}$$

donde $\widetilde{\lambda}^- = \frac{1}{2}(\widetilde{\lambda} - |\widetilde{\lambda}|)$ y $\beta^- = \frac{1}{2}(\beta - |\beta|)$.

Para nuestro algoritmo y como nos hemos referido en todo el trabajo, la variable W es

$$W_i^n = [W_{i,1}^n, W_{i,2}^n, W_{i,3}^n] \tag{B.55}$$

Donde

$$W_{i,1}^n = U_{i,1}^n, W_{i,2}^n = U_{i,2}^n / U_{i,1}^n, W_{i,3}^n = U_{i,3}^n / U_{i,1}^n \tag{B.56}$$

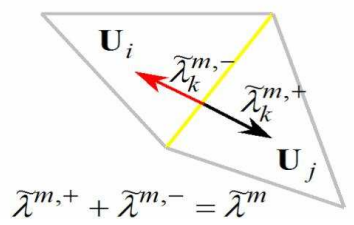


Figura B.3: Selección de la información necesaria en el método descentrado.

Apéndice C
Gestión del proyecto software

La traza temporal del desarrollo del trabajo ha sido algo variable en el tiempo. En un inicio, se planteó el desarrollo del Proyecto desde el día 15 de Septiembre de 2010, con intención de que para abril estuviera terminado. En medio de este tiempo, tuve que realizar una parada del trabajo desde el 20 de diciembre hasta el 12 de febrero con motivo de los último exámenes de los estudios de Ingeniería Informática. A partir del día 12 de febrero, volví a incorporarme al desarrollo del proyecto, con el inconveniente de que esos casi 2 meses de "parón", supusieron una perdida de adherencia en cuanto a la soltura de desarrollo del código se refiere. Con esta distribución temporal pues, me referiré a dos tramos bien diferenciados del PFC: el primero que va desde el 15 de septiembre hasta el 20 de diciembre de 2010 y el que va desde el 12 de febrero hasta el 30 de abril.

Si bien es cierto que para el primer periodo tenía terminado una buena parte del PFC, quedaba todavía lo más complicado, que concernía a la paralelización en máquinas de memoria distribuida. Durante ese primer periodo, se desarrollaron las tareas de análisis de código, análisis de compilación y desarrollo de la estrategia de paralelización en máquinas de memoria compartida además de la incorporación de una herramienta de gestión de versiones y un seminario a los miembros de GHC para explicarle el uso del mismo. Durante el segundo, estuvo dedicado prácticamente al desarrollo del código adaptado a paralelización en máquinas de memoria distribuida y al desarrollo del texto del PFC que ha consistido en fusionar los subdocumentos que iba haciendo según desarrollaba las distintas tareas y revisarlos en su conjunto. Esto queda reflejado en la figura C.1.

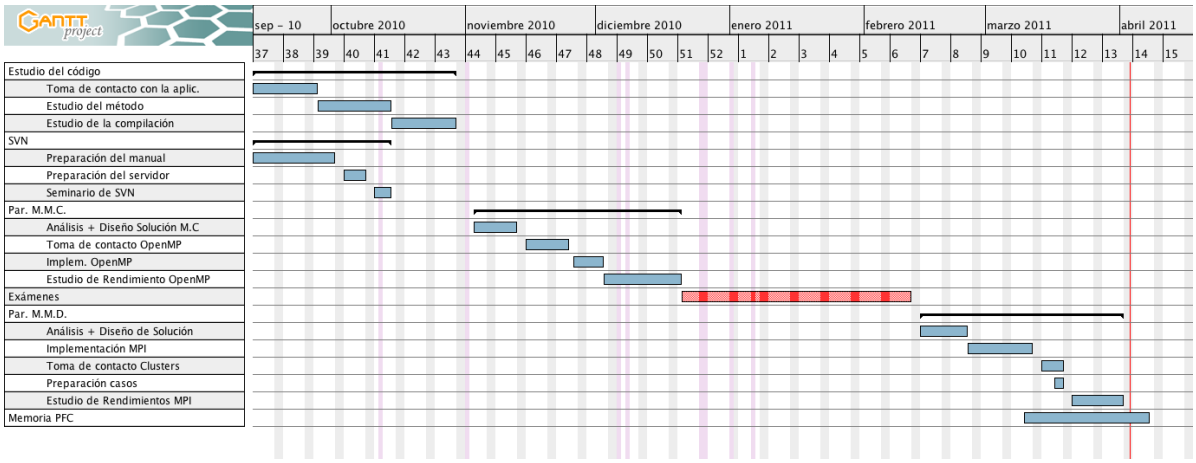


Figura C.1: Diagrama de Gantt del Proyecto

Además de las tareas especificadas en el diagrama, se han ido haciendo otras paralelas como el desarrollo de herramientas de preprocesado de las mallas y postprocesado, o scripts útiles para el estudio de los rendimientos. Así mismo, no están incluídos los aprendizajes que han conllevado el uso de todas las herramientas que se han usado durante el trabajo.

Durante todo el proyecto han aparecido complicaciones a distintas escalas, de las cuales hay dos muy destacables. La primera la complicación que apareció en un momento determiniado por el mal funcionamiento de la aplicación paralelizada con MPI. Llevó 2 días depurar el mal funcionamiento para determinar que era por un fallo en una inicialización que no había aparecido hasta ahora por el compilador que se utilizaba. El segundo gran obstáculo fue la adaptación al uso de la infraestructura en Caesaraugusta. Hasta ahora nunca había trabajado con colas de ejecución ni con el compilador *XLF* ni las optimizaciones que se utilizan. Ni mucho menos me considero ahora un experto en ello, pero sí he cogido solutura con esta forma de ejecutar aplicaciones.

La dedicación al proyecto ha sido de una media diaria de 9 horas reales, que teniendo en cuenta un factor de uso del 90 % hace un total de 822 horas reales ó 739 horas efectivas.

Para la mejor comprensión del método y de la mecánica de fluídos, he asistido (y sigo haciéndolo) como oyente a la asignatura Fundamentos de Fluidos y Procesos Fluidodinámicos de 2^o curso de Ingeniería Industrial con un montante de 4 horas semanales desde el 18 de febrero de 2011.

Apéndice D
Casos de análisis

En este anexo se describe con más profundidad los casos que se han elegido para estudiar el rendimiento de la paralelización así como los motivos de su elección. Aquí se puede encontrar por lo tanto, una colección de resultados referentes al trabajo desarrollado.

D.1 Caso C.1 - Dos depósitos

Este caso llamado así por su geometría, resuelve la evolución de un fluido que inicialmente está en un depósito C_1 conectado por un canal a un segundo depósito C_2. La geometría exacta del caso es como se especifica en la figura D.1.

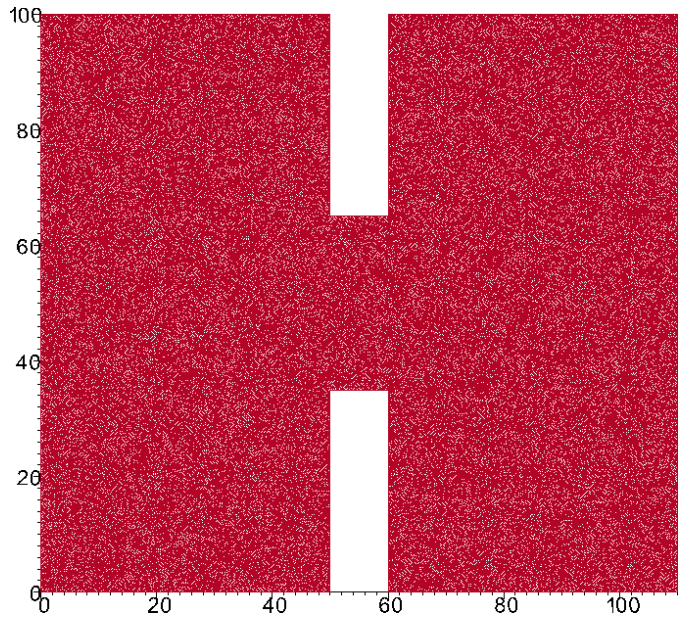
Se ha elegido este caso porque tiene una evolución temporal en el tiempo en la carga computacional, ya que inicialmente el 50 % de las celdas están mojadas, pasando en un tiempo determinado (20 s.) a estar el 100 % de las celdas mojadas con lo que se pretende ver la implicación que tiene este desequilibrado inicial en el tiempo de ejecución con la paralelización.

Cuando se empezó a diseñar este caso se pensó que sería bueno, para modificar las condiciones de comunicación, variar la anchura del canal por el que se comunican las dos cubetas. Lo que no se esperaba era que esto iba a hacer cambiar las condiciones del problema dado que, al disminuir la anchura del canal, aumentan las velocidades a las que pasa el agua (ver figura D.2 y esto hace condicionar el paso de tiempo, lo que modifica totalmente el problema y los resultados pasan a no ser comparables.

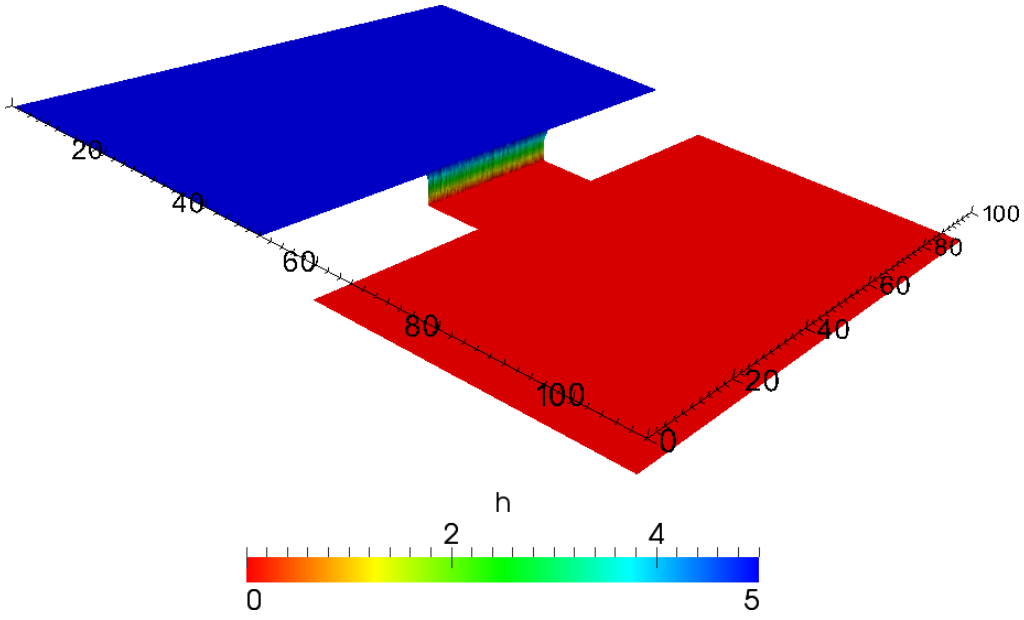
Como se ve en la figura D.2, el caso que mas velocidad lleva es el que tiene una anchura de canal más pequeña, pero el que más tarda es el que tiene una anchura intermedia (de 30 cm.) por lo que es ese el que elegiremos como caso para probar los rendimientos. En la misma figura se ve el tiempo de ejecución del caso que se utilizó para medir lo que se está explicando. Es importante notar que los tiempos que ahí aparecen no son con las mismas condiciones con las que se desarrollará el caso.

En particular, la simulación la haremos de 400 s. con un volcado de tiempo cada 200 pasos de tiempo. Además se ha utilizado un CFL=0.9 y un coeficiente de Manning $m = 0{,}03$.

Los resultados de esta simulación son los utilizados en el análisis del trabajo, así que están ya suficientemente explicados. De ellos se deduce, para todos los casos que la evolución que se sufre en la carga sí repercute en los tiempos de ejecución. Como podemos ver en la figura D.3 el



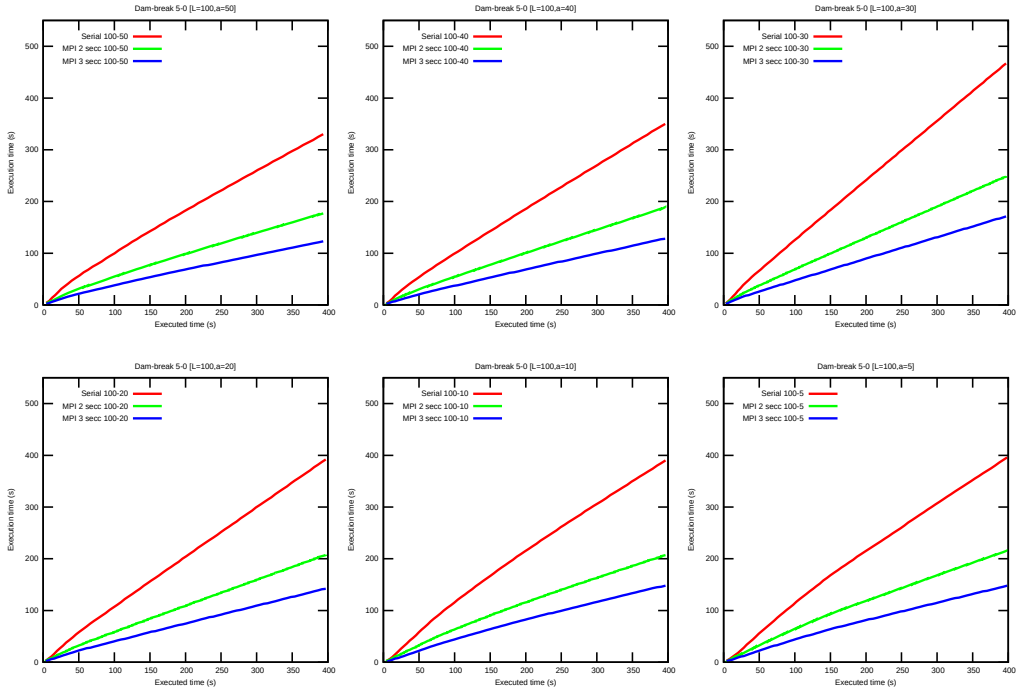
(a) Planta de la geometría



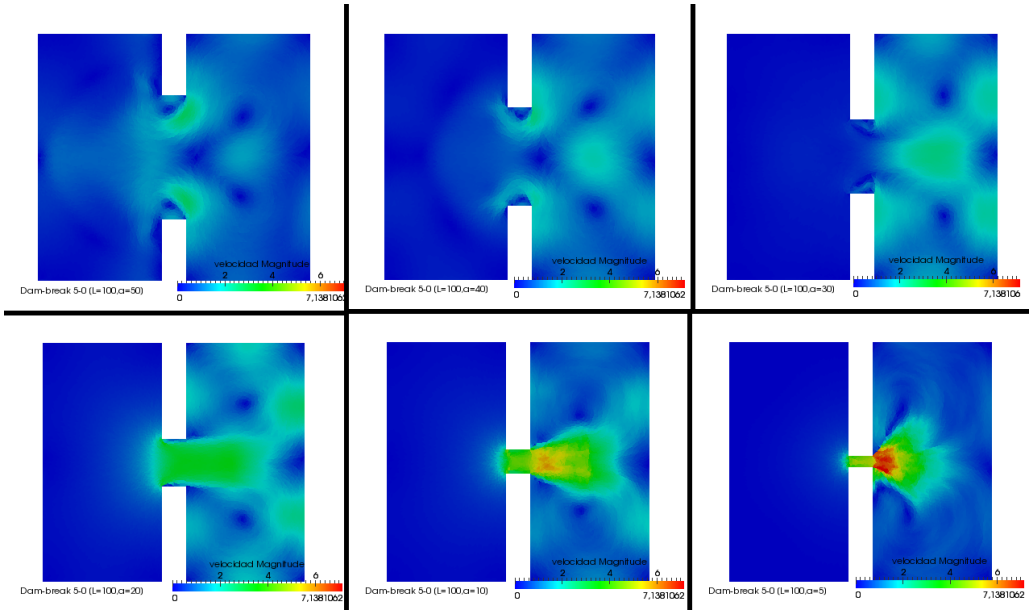
(b) Condiciones iniciales del caso

Figura D.1: Descripción del caso C.1

performance de la paralelizacion va aumentando dado que los tiempos de ejecución de la implementación paralela van siendo menores por el equilibrado que se produce de manera implicita en el problema.



(a) Tiempos de ejecucion de las distintas configuraciones



(b) Representación del módulo de las velocidades

Figura D.2: Resultados de la variación del caso C.1

D.2 Caso C.2 - Alagon 0-2500-0

El siguiente caso a analizar fue uno utilizado por el grupo para estudiar la inundación de unos depósitos en el río Ebro a su paso por Alagón. En este caso, se parte de un estado estacionario

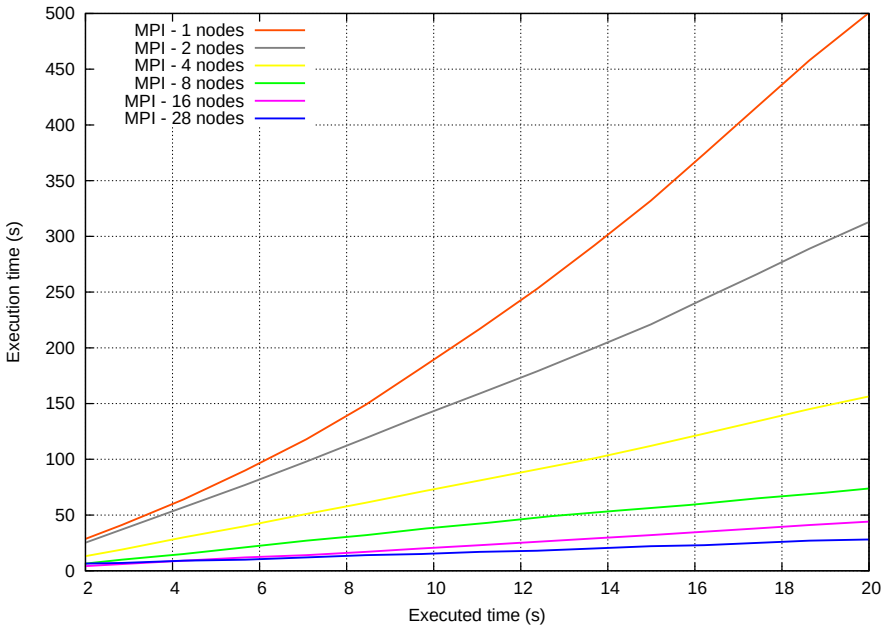


Figura D.3: Evolucion del tiempo de ejecucion en el intervalo (0-20)

con un calado constante en las celdas correspondientes al cauce, y se aumenta el caudal desde 0 hasta $2500 m^3/s$ en un tiempo de 45000 segundos y de manera lineal. Para controlar mejor el tiempo de la simulación sólo la haremos hasta $37000 s$..

Esta malla no permite ser particionada, segun el método propuesto de partción, en más de 24 secciones, con lo que las pruebas sólo se pueden hacer con hasta 24 nodos.

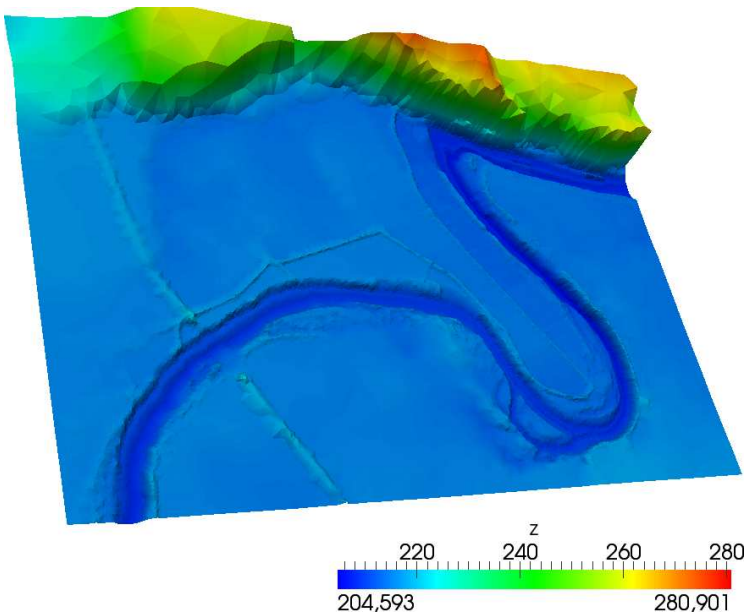
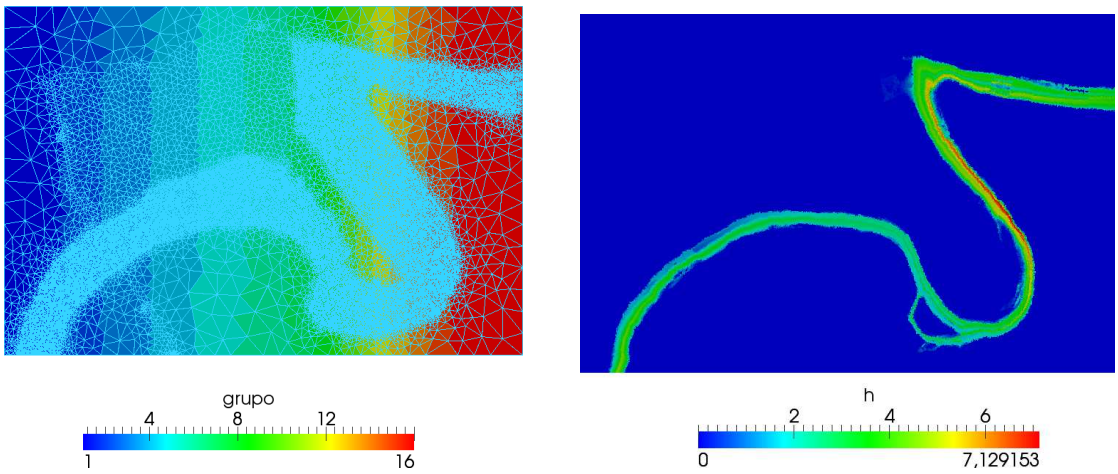


Figura D.4: Representación tridimensional de la geometría del problema

La geometría y el estado inicial es el que se puede ver en la figura D.5. Es importante notar que la particion inicial está perfectamente repartida en el número de celdas en todo el dominio, pero que a lo largo de la evolución temporal, esto ya no estará equilibrado.



(a) 16-partición del dominio **(b)** Condición inicial del problema

Figura D.5: Distribución del problema de Alagón

La evolución temporal desarrolla una expansión del flujo por fuera del cauce inundando sus alrededores. En términos computacionales, esto es representado por nuevas celdas mojadas que pasan a incluirse en el cálculo, pero como vemos, esta repartición no es equilibrada en el tiempo (ver figura D.6).

Viendo que la distribución deja de estar equilibrada en algún momento, lo esperable es que los tiempos de ejecución dejen de ser lineales a causa de este desequilibrio. Efectivamente es así, y no sólo es así, si no que además el Speed-Up y el Rendimiento pasan a estar completamente descompensados de acuerdo a las premisas que se han ido razonando a lo largo del PFC.

En el caso de la ejecución en *trombón* los resultados son los de la figura D.7, y en ellos podemos ver una evolución del speed-up inversamente proporcional al crecimiento de nodos de cálculo. Esto es así en el resto de ejecuciones también y se debe al desequilibrio que ocurre en el tiempo, en particular en el tiempo de ejecución del que se han extraído los datos. Hemos de tener en cuenta que según aumentemos el número de particiones y por lo tanto disminuyamos el número de celdas por nodo de cálculo, el desequilibrio si se produce, se acentuará más.

En el caso de *Terminus* en el que los tiempos de ejecución son menores de por sí, todavía tiene más repercusión este desequilibrado. En la figura se ve que la curva generada para el speed-up pasa a ser incluso decreciente. En este *cluster* esto sucede porque la consecuencia del desequilibrado ya no sólo es que el rendimiento decrezca, si no que puede causar que incluso aumentar el número de nodos aumente el tiempo de ejecución. Esto es así porque la velocidad de cálculo es muy superior a la de comunicación intra-blade, haciendo que este factor sea muy influyente en el caso de la ejecución de 16 nodos.

Otra consecuencia que no se refleja en trombón pero sí aquí es que si el dominio de cálculo va variando en el tiempo, hay una función encargada de hacer esta tarea. Al paralelizar los dominios

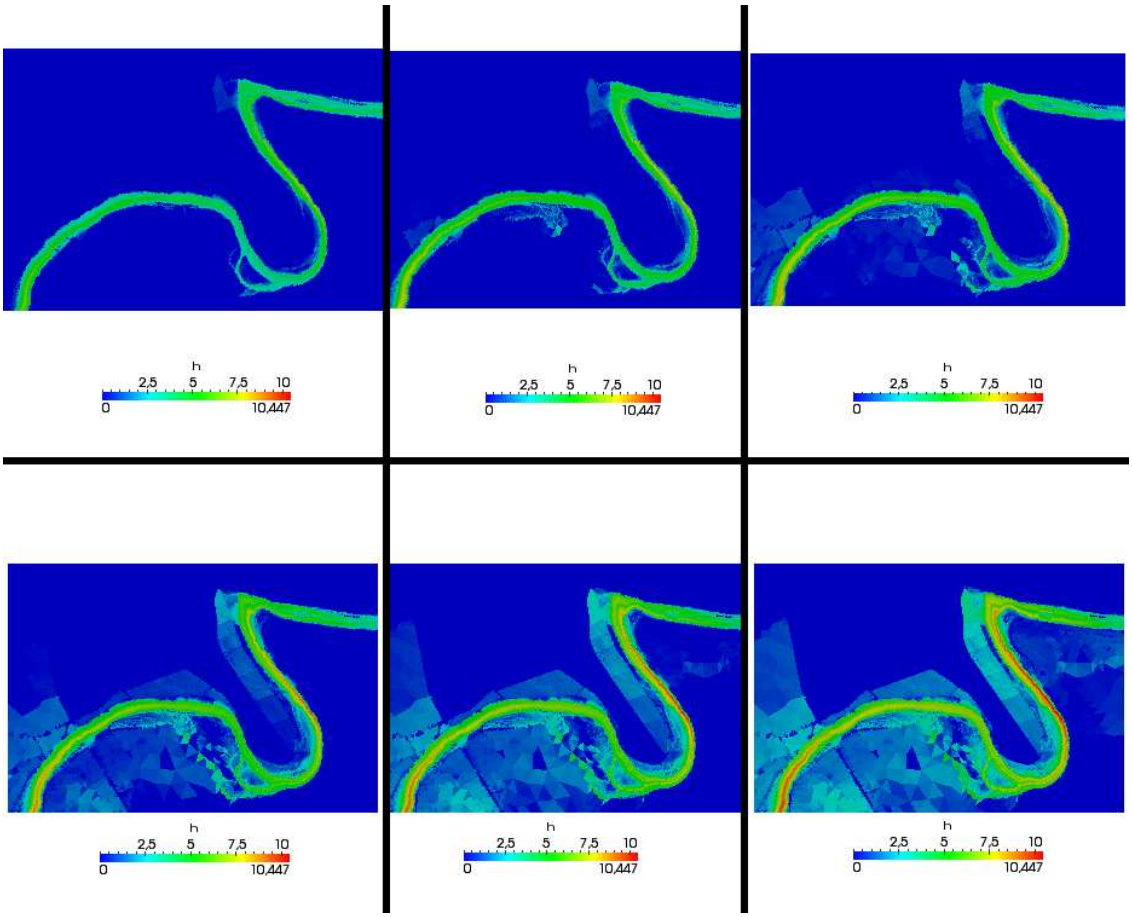
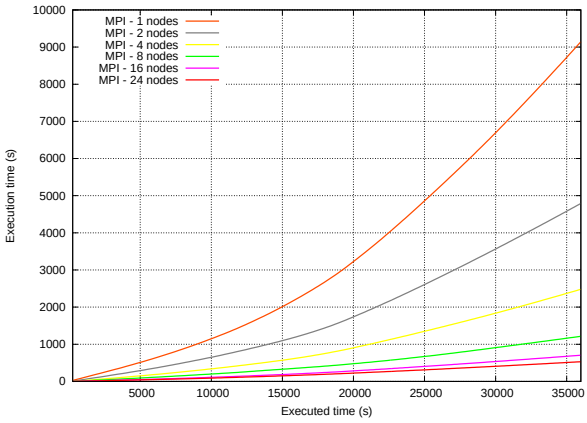


Figura D.6: Evolución temporal en (orden Arriba-Izda-Abajo-Derecha) 10 ks, 15ks, 20 ks, 25 ks, 30 ks, 35 ks

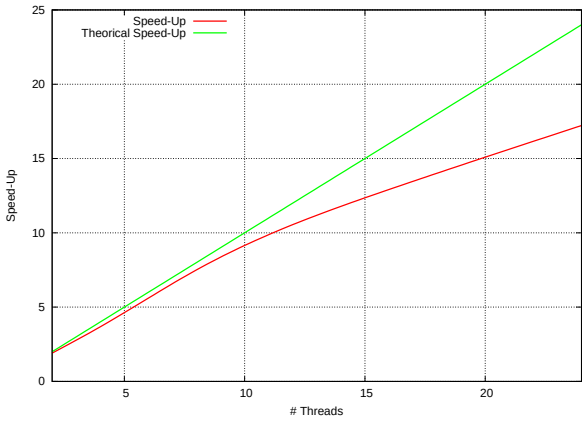
de cálculo, no sólo paralelizamos el cálculo, si no también esta tarea. En particular y como hemos visto en la figura D.6 va variando todo el tiempo. el repartir esta tarea puede hacer que incluso vaya mas rápido dado que esa función hace un uso intensivo de acceso a memoria, y mediante la paralelización estamos ayudando a que decrezcan estos accesos con las consecuencias que esto tiene en la cache. De aquí se explica que el Speed-Up sea superior a p en algunos casos. A pesar de haber planteado que el Speed-Up, teóricamente ha de ser menor o igual a p, existen modelos que explican, precisamente en casos como estos, cómo es posible explicar la superación de esta cota [8]

Por otro lado la ejecución en *Caesaraugusta* nos proporciona los valores que vemos en la figura D.9. Los resultados de los parámetros que estamos midiendo todavía son más irregulares. Hemos de tener en cuenta que en Caesaraugusta la ejecución distribuída se hace siempre fuera del nodo, es decir, igual que en *Trombón* y en *Terminus* aprovechabamos los cores de los procesadores, aquí todo lo que se ejecute de manera paralela, se hace saliendo de manera forzosa del nodo. Aún así podemos ver que esto es beneficioso para la ejecución. Aquí tenemos que tener en cuenta todavía más el factor de la apliación dinámica del dominio. Los procesadores de *Caesaraugusta* tienen una cache de 512 KB, que comparada con la de *Terminus* por ejemplo es 20 veces menor, con lo que el desequilibrio que se va produciendo se va compensando con lo bien que funciona la función de ampliar el dominio de manera distribuída.

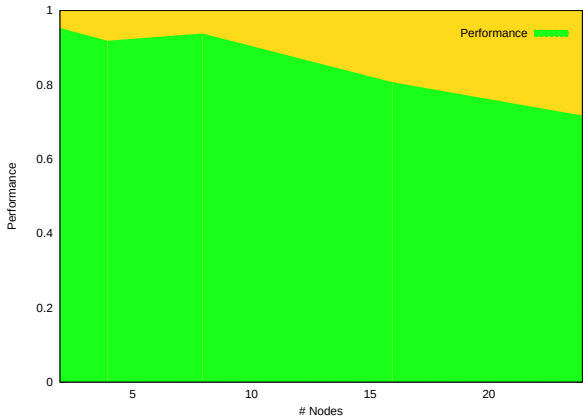
En general, los casos que ejecutaremos tendrán este perfil de desconocimiento del dominio



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 36500 s. de simulación



(c) Performance en 36500 s. de simulación

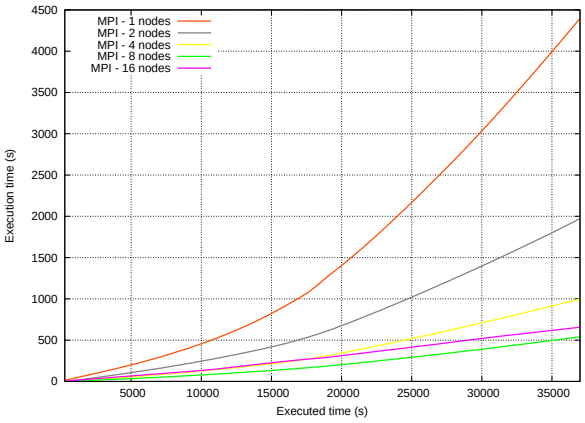
Figura D.7: Resultados de ejecución del caso C.2 en Trombón

promedio de cálculo, por lo que hacer esta prueba es muy importante para analizar como se va a comportar la paralelización. En general, las mallas de cálculo están diseñadas para reforzar las zonas donde a priori se sospecha que los cálculos han de ser más refinados, por lo que el hacer el balanceo inicialmente tampoco influye de manera tan negativa como podríamos haber imaginado.

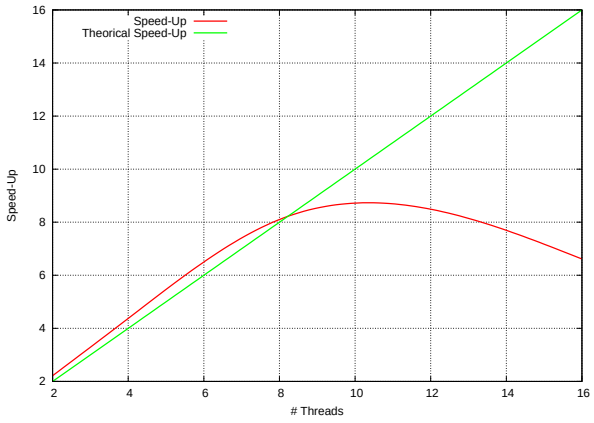
D.3 Caso C.3 - Cross

El tercer caso con el que se han realizado pruebas es el de un canal con pendiente de 1 % que baja a un depósito que contiene otro canal de la misma pendiente. La geometría es la que se ve en el figura D.10 y un ejemplo de particion es el de la figura D.11.

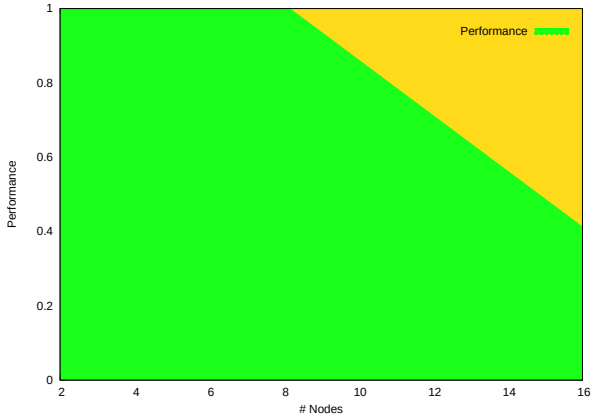
El caso parte de un calado inicial $h = 10{,}0$ y se le inyecta un caudal de $40m^3/s$ durante 20 segundos para de ahí en adelante inyectarle sólo $5{,}5m^3/s$. El suelo tiene un coeficiente de Manning de 0.03 y se le impone unas condiciones de contorno de número de Frude a la salida 0.9. Esto implicará que el fluido pasará de régimen subcrítico a la entrada a régimen supercrítico en el canal, para volver a subcrítico a la salida, provocando un resalto hidráulico en la transición del cubo al segundo canal.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 36500 s. de simulación



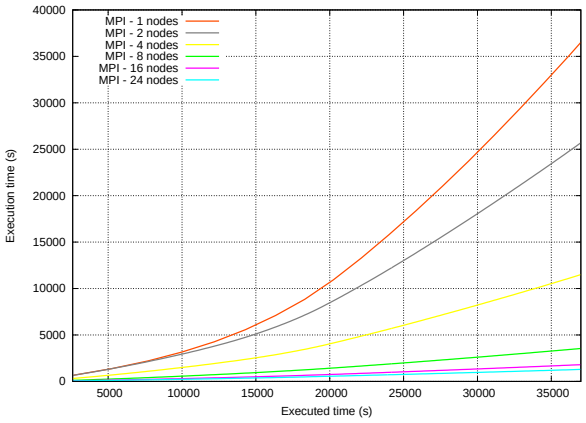
(c) Performance en 36500 s. de simulación

Figura D.8: Resultados de ejecución del caso C.2 en Terminus

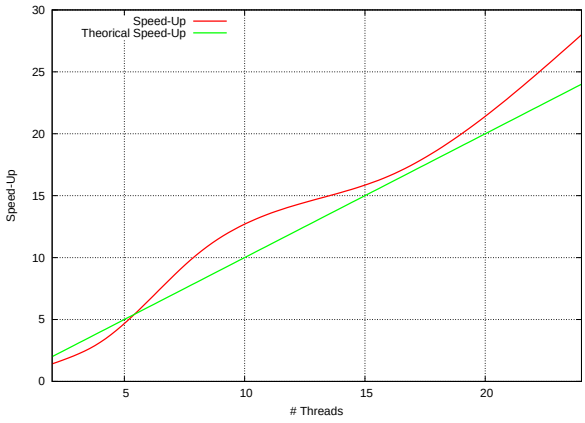
Este es un buen ejemplo del rendimiento que nos ofrece la ejecucion cuando inicialmente sabemos que se van a hacer cálculos en todas las celdas. Este ejemplo puede servir para sacar un rendimiento límite para este problema. Simularemos durante 1000 s.

El primer análisis lo vamos a efectuar sobre la simulación en *Trombón*. En este *cluster*, el rendimiento es bastante aceptable según vemos en las gráficas de la figura D.12. En ellas nos encontramos que el Rendimiento decrece de manera notoria a partir de 16 nodos. Hay que tener en cuenta que aquí hemos hecho una simulción con 32 nodos. Estos nodos son virtuales en el sentido de que cada procesador de Trombón dispone de 4 cores y 8 threads, en particular, hemos hecho trabajar a 4 de los procesadores con 5 threads para ver como iba el rendimiento y como vemos decrece rápidamente con ejecuciones de este estilo. Sigue ganando algo, pero el crecimiento es mucho menor. Esto nos recuerda a los resultados que presentamos en secciones anteriores para la implementación OpenMP, en los cuales nos encontrábamos que a partir de 4 threads en esta máquina, el crecimiento del Speed-up era menor que el producido hasta 4 threads.

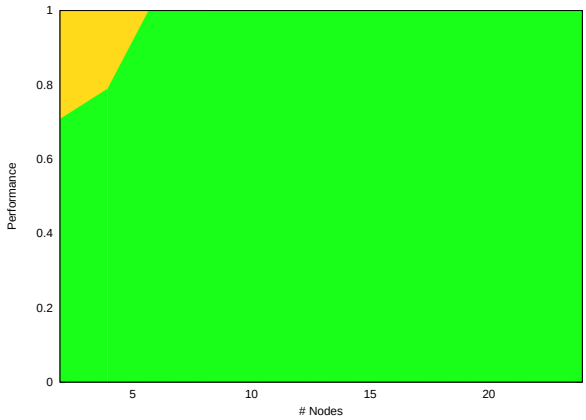
En la ejecución en el *cluster* terminus los resultados son superiores, igual que en el caso anterior, a los de *Trombón*. En este caso el rendimiento que saca la aplicación es superior al 90 %.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 36500 s. de simulación



(c) Performance en 36500 s. de simulación

Figura D.9: Resultados de ejecución del caso C.2 en Caesaraugusta

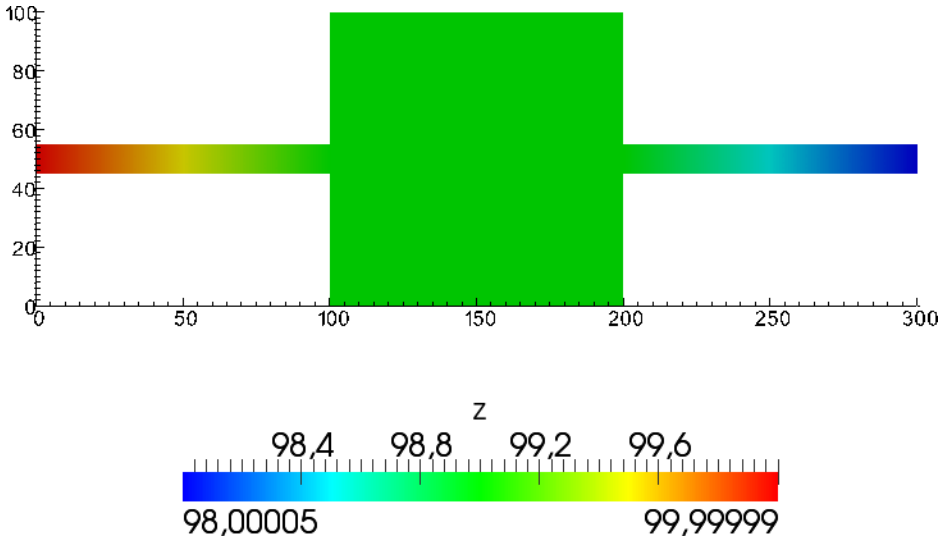


Figura D.10: Geometría del problema C.3

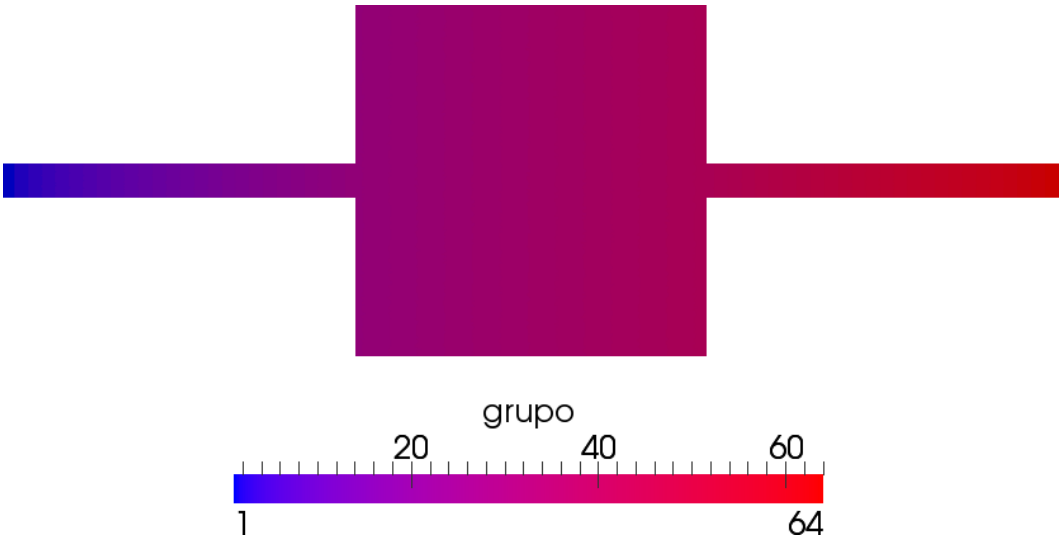
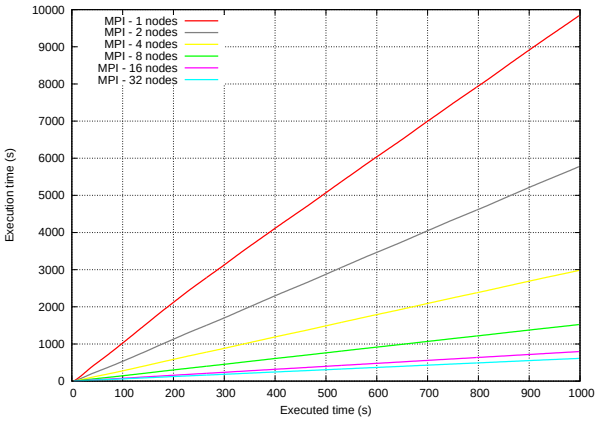
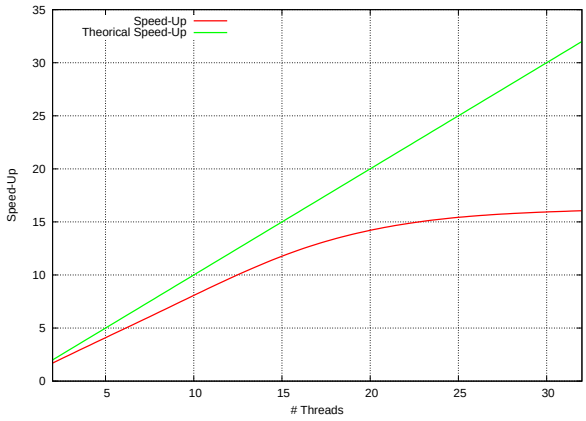


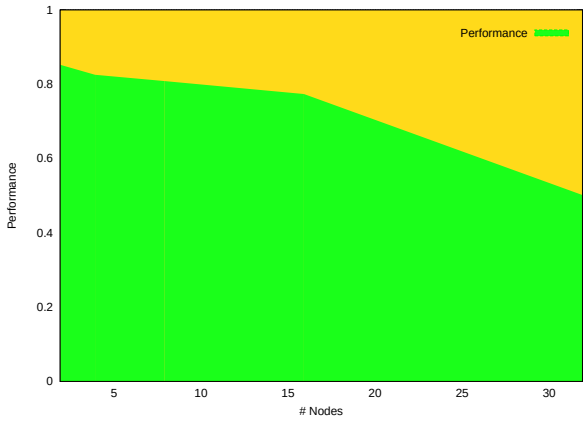
Figura D.11: 64-particion de la malla del problema C.3.



(a) Tiempos de ejecución frente a tiempos ejecutados



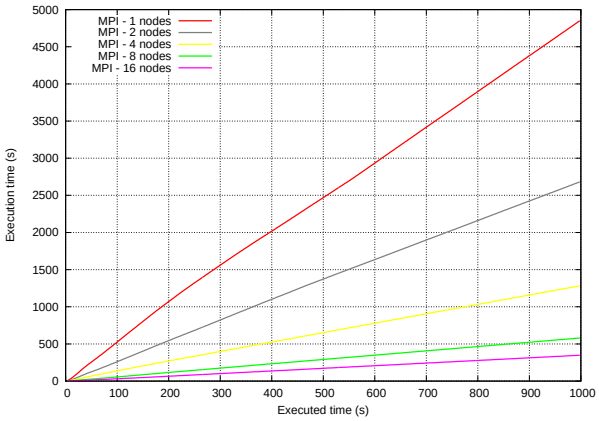
(b) Speed-up en 1000 s. de simulación



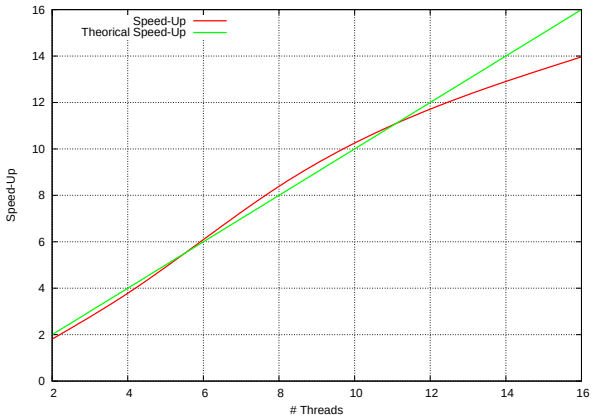
(c) Performance en 1000 s. de simulación

Figura D.12: Resultados de ejecución del caso C.3 en Trombón

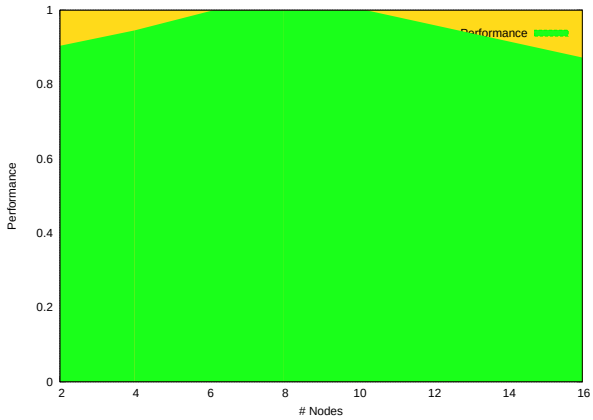
Como vemos en la figura, el Speed-Up es muy cercano al teórico sobrepasándolo, igual que en el caso anterior, en la ejecución en 8 cores.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 1000 s. de simulación



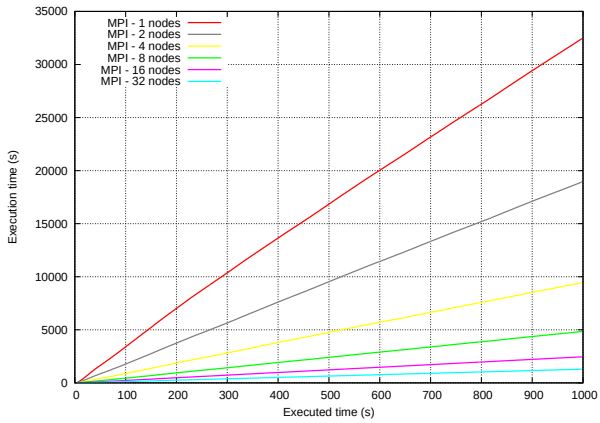
(c) Performance en 1000 s. de simulación

Figura D.13: Resultados de ejecución del caso C.3 en Terminus

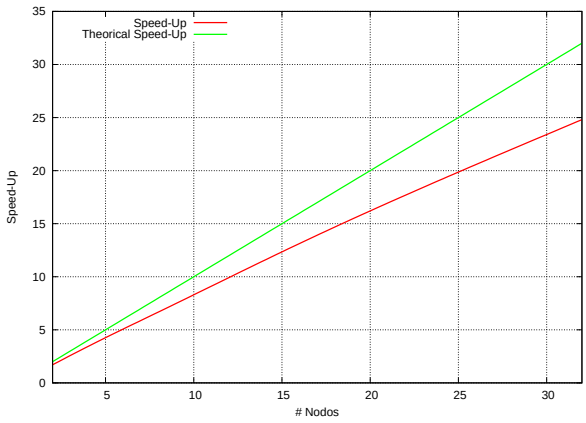
Como conclusión general y vistos los tres casos en *Terminus* con la configuración en la que actualmente está dispuesto, la ejecución con 8 cores saca los mejores rendimientos en general, lo que nos hace pensar que la configuración de chips-on-board es muy adecuada para este tipo de ejecuciones. Si bien está configuración es más cara que conectar los nodos en red, para simulaciones en las que queramos un buen rendimiento sin ser tiempos de ejecución excesivamente largos, esta configuración funciona realmente bien.

El último cluster en el que hemos realizado las pruebas de rendimiento es en *Caesaraugusta*. En la figura D.14 podemos comprobar lo constante que es el rendimiento. En este caso se hace muy poco acceso a memoria ya que éste sólo tiene relevancia en la preparación del caso (factor t_0) que es donde se hacen cálculos de preparación de dominio. A diferencia de otros casos el factor cache no influye de manera drástica y los factores que van a determinar el tiempo de cálculo van a ser el tiempo de cálculo y el tiempo de comunicación. El segundo factor es constante en todos los casos, ya que en el caso en el que la partición sea 2, la frontera que más paredes soporta con mucha diferencia es la central, igual que en el resto de los casos, luego el factor que variará según

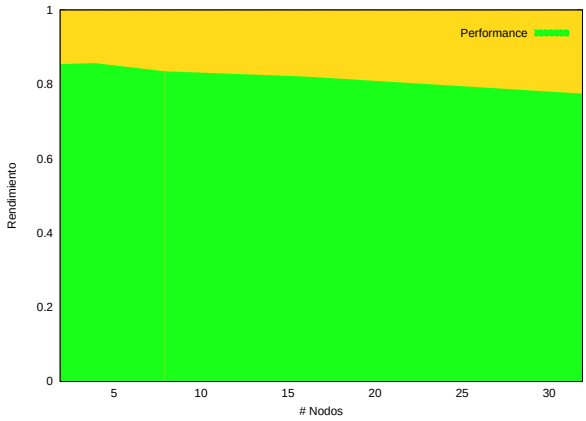
ampliemos el dominio, será unicamente el tiempo de cálculo. el hecho de que la infraestructura sea tan homogenea, nos lleva a explicar ese rendimiento constante.



(a) Tiempos de ejecución frente a tiempos ejecutados



(b) Speed-up en 1000 s. de simulación



(c) Performance en 1000 s. de simulación

Figura D.14: Resultados de ejecución del caso C.3 en Caesaraugusta

Como conclusión de este caso, se ve que las velocidades de procesamiento y de comunicación hacen variar mucho los rendimientos de una máquina a otra. Los resultados obtenidos son muy buenos en todos los casos con los matices anteriormente comentados.

Apéndice E
Regresion de la ecuacion t_x en Caesaraugusta

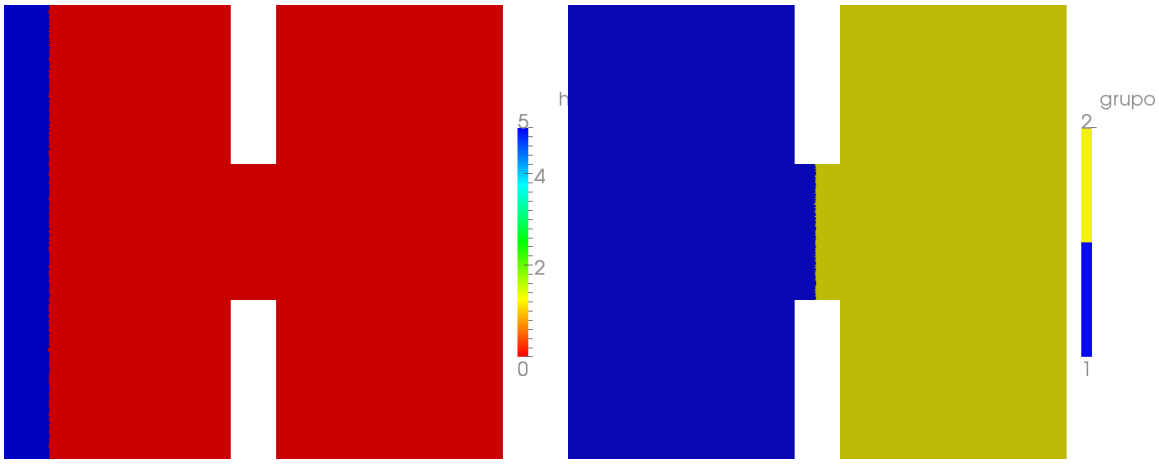
En el capítulo anterior hemos introducido la fórmula del tiempo de ejecución de la forma

$$t_{exec} = t_0 + (t_{exec1} * ncell_s) + ((t_{comm1} * N_{f,s}) * 3) \tag{E.1}$$

y como hemos explicado en éste, podemos aproximarla a

$$t_{exec} = \cancelto{0}{t_0} + (t_{exec1} * ncell_s) + ((t_{comm1} * N_{f,s}) * 3) \tag{E.2}$$

Es un ejercicio interesante ver, lo bien que se ajusta esta función a la realidad, para poder así predecir a priori el tiempo de ejecución aproximado que nos tomará una simulación con nuestro modelo de paralelización.



(a) calado inicial h_9 del caso de estudio **(b)** Distribución de dominios d_o, d_1

Figura E.1: Diseño del experimento de regresión

Para hacer éste análisis, lo que haremos será tomar el caso C.3 modificado con una $h_0 = 5m$ y vamos a dejar que evolucione durante $17s.$, tiempo que requiere llegar al segundo dominio. Esta modificación además implica que inicialmente hay un 12 % de celdas mojadas en el dominio d_0 y ninguna en el dominio d_1. Por último sacaremos tiempos de ejecución en cada paso de

tiempo, haciendo una relación $celdas_{mojadas} - t_x$ y aplicaremos una regresión lineal de la función.

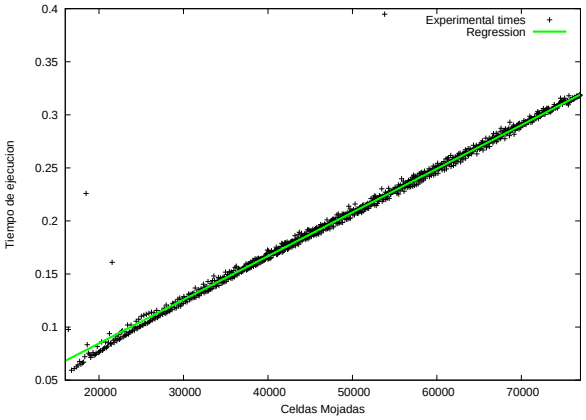


Figura E.2: Diseño del experimento de regresión

Los resultados se ajustan muy bien a esta función como vemos en la figura E.2 y como se observa, existen valores que destacan en tiempos no determinados. Estos valores se corresponden al *jitter* anteriormente nombrado que igualmente, no altera demasiado el resultado. Los resultados del ajuste a una funcion $f(x) = ax + b$ son

$$a = 4{,}11272e-06, \quad b = 0{,}00242021, \quad RMS = 0{,}0080191 \tag{E.3}$$

Ajustamos a esta funcion dado que la funcion E tiene el elemento $((t_{comm1} * N_{f,s}) * 3)$ constante en el tiempo (similar al término b) y en nuestro caso el término $a * x$ es el término $(t_{exec1} * ncell_s)$.

Lo que si es curioso es el desajuste del origen de ordenadas. Esto puede ser por que con pocas celdas, los tiempos de ejecucion son muy pequeños y por lo tanto la medicion de los mismos es ms susceptible de error.

Caracterizando la ecuación a nuestros términos y para el caso de estudio y teniendo en cuenta que $b = 2 * n_{0,1} * t_{comm} * 3$ siendo $n_{0,1}$ número de paredes en la intersección $(0, 1)$ que en nuestro caso es $n_{0,1} = 82$, se deduce que el tiempo de comunicación de un elemento es,

$$t_{comm} = \frac{0{,}00242021}{2 * 3 * 82} = 0,000004919s. \tag{E.4}$$

Con lo que la ecuación queda

$$t_x = 4{,}11272 * 10^{-6} * n_{cell} + (4{,}919 * 10^{-6} * 3 * 2 * n_{0,1}) \tag{E.5}$$

En general, para los casos en los que tenemos más secciones y además estas secciones no están totalmente equilibradas, el tiempo de ejecución de una iteración con esta fórmula se reescribe como,

$$t_x = 4{,}11272 * 10^{-6} * (MAX(n_{cell}) + (4{,}919 * 10^{-6} * 3 * 2 * V * MAX(n_{i,j}))) \tag{E.6}$$

Siendo $MAX(n_{cell})$ el máximo número de celdas implicadas y $MAX(n_{i,j})$ la sección que más paredes comparte con sus vecinas. Nótese que áparece un término V en la ecuación. Este término es el número de vecinos de la sección, que para el caso anterior era $V = 1$, pero que en general,

esto sólo se cumple en la primera sección y en la última, el resto de subdominios tienen 2 vecinos.

Sin poder extenderlo para todos los casos, salvo para aquél en el que todo el tiempo todas las celdas tengan calado $h > 0$, podemos deducir el término general como

$$t_{Xtotal} = I * (4{,}11272e - 06 * (MAX(n_{cell}) + (4{,}919 * 10^{-6} * 3 * 2 * V * MAX(n_{i,j})))) \quad \text{(E.7)}$$

donde I es el número total de iteraciones necesarias para alcanzar el tiempo de simulación.

Apéndice F
Equipos de simulación

F.1 Características de *Caesaraugusta*

En este equipo, las simulaciones se han hecho sin aprovechar todas las características espaciales que la infraestructura nos permitía, obligándole a utilizar un procesador, de los dos disponibles, por nodo. Por tanto, en esta configuración cuando nos refiramos a nodo, nos referimos a nodo físico, y hemos de tener en cuenta que cada nodo está conectado por red.

1. Processor: 512 processors PowerPC 970FX 2.2 GHz
2. Memory: 1TB RAM memory
3. Network: Interconnection networks Myrinet
4. System: Linux
5. Total Nodes: 256 Nodes

F.2 Características de *Terminus*

Dos procesadores por blade. Nuestro *cluster* está formado por dos Blades conectados por Gigabit, y la comunicación intra-blade es en placa a través de una placa base dual. En este equipo se utilizarán primero los cores del procesador como nodos de cálculo (4), pasando a los cores totales por blade (8) escalando por último a los cores totales en nuestro *cluster* (16).

1. Processor: 4 778I Xeon Quad core
2. Memory: 4 Mbytes/processor
3. Network: Interconnection networks Gigabit
4. System: Linux
5. Total Nodes: 2 Nodes

Los parámetros de compilación configurados por los técnicos del *cluster* son los siguientes:

```
-D__INTEL_COMPILER=910 -D_MT -D__ELF__ -D__INTEL_COMPILER_BUILD_DATE
    =20070215 -D__unix__ -D__unix -D__linux__ -D__linux -D__gnu_linux__ -
    Dunix -Dlinux -D__x86_64 -D__x86_64__ -mGLOB_pack_sort_init_list -I.
    -I/apps/apps64/openmpi-1.2.8/include -I/apps/apps64/openmpi-1.2.8/lib
     -I/apps/apps64/intelfc91/include -I/apps/apps64/intelfc91/
    substitute_headers -I/usr/lib64/gcc/x86_64-suse-linux/4.1.2/include -
    I/usr/local/include -I/usr/include -I/usr/lib64/gcc/x86_64-suse-linux
    /4.1.2/include -O2 "-reentrancy threaded" -mP1OPT_version=910 -
    mGLOB_source_language=GLOB_SOURCE_LANGUAGE_F90 -mGLOB_tune_for_fort -
    mGLOB_use_fort_dope_vector -mP2OPT_static_promotion -
    mP1OPT_print_version=FALSE -mP3OPT_use_mspp_call_convention -
    mCG_use_gas_got_workaround=T -mP2OPT_align_option_used=TRUE "-
    mGLOB_options_string=-I/apps/apps64/openmpi-1.2.8/include -I/apps/
    apps64/openmpi-1.2.8/lib -pthread -L/apps/apps64/openmpi-1.2.8/lib -
    lmpi_f90 -lmpi_f77 -lmpi -lopen-rte -lopen-pal -ldl -Wl,--export-
    dynamic -lnsl -lutil" -mGLOB_cxx_limited_range=FALSE -
    mGLOB_as_output_backup_file_name=/tmp/ifortm0Hpg8as_.s -
    mGLOB_machine_model=GLOB_MACHINE_MODEL_EFI2 -mP2OPT_subs_out_of_bound
    =FALSE -mIPOPT_ninl_user_level=2 -mIPOPT_args_in_regs=0 -
    mPGOPTI_value_profile_use=T -mIPOPT_activate -mIPOPT_lite -
    mP2OPT_hlo_level=2 -mP2OPT_hlo -mIPOPT_obj_output_file_name=... -
    mP3OPT_asm_target=P3OPT_ASM_TARGET_GAS -mGLOB_obj_output_file=/tmp/
    ifortMZNPvG.o -mGLOB_source_dialect=GLOB_SOURCE_DIALECT_FORTRAN -
    mP1OPT_source_file_name
```

F.3 Características de *trombón*

El máximo número de nodos que podemos utilizar en el *cluster* es 28, dado que cada procesador dispone de 4 cores. Utilizando hyper-threading podríamos llegar a utilizar 56 nodos y no se ha utilizado salvo en el caso en el que hemos simulado con 32 nodos en el caso D.3. Obviamente esta configuración no resultará demasiado ventajosa así que prefiriremos utilizar como mucho hasta 28 cores haciendo cada uno la vez de nodo.

1. Processor: 1 Intel Core i7 CPU 860 @ 2.80GHz
2. Memory: 4 Mbytes/processor
3. Network: Gigabit
4. System: Linux
5. Total Nodes: 7 Nodes

Apéndice G
Herramientas utilizadas en el trabajo

El desarrollo del trabajo ha requerido del uso de una serie de herramientas que se especifican a continuación. Algunas de ellas las había utilizado a lo largo de toda la carrera, y otras he tenido que aprender a utilizarlas. He de decir al respecto que creo que la carrera me ha aportado la capacidad necesaria para poder adaptarme a nuevas herramientas en un plazo de tiempo relativamente corto. Las herramientas y estándares más relevantes que se han utilizado son

- awk
- Bash-scripting
- C
- Debian
- Dropbox
- Eclipse
- Fortran 90
- GanttProject
- gdb
- GCC 4.4
- gnuplot
- HPC-Europa2 VC Live DVD
- ifort
- Intel Vtune
- KCacheGrind
- LaTeX
- mnsubmit
- OpenMPI 1.2.7
- OpenMP 3.0

- paraview
- Perl
- qsub
- sed
- SVN
- tecplot
- TotalView
- triangle
- Valgrind
- vi
- xlf

Apéndice H
DFD de la aplicación

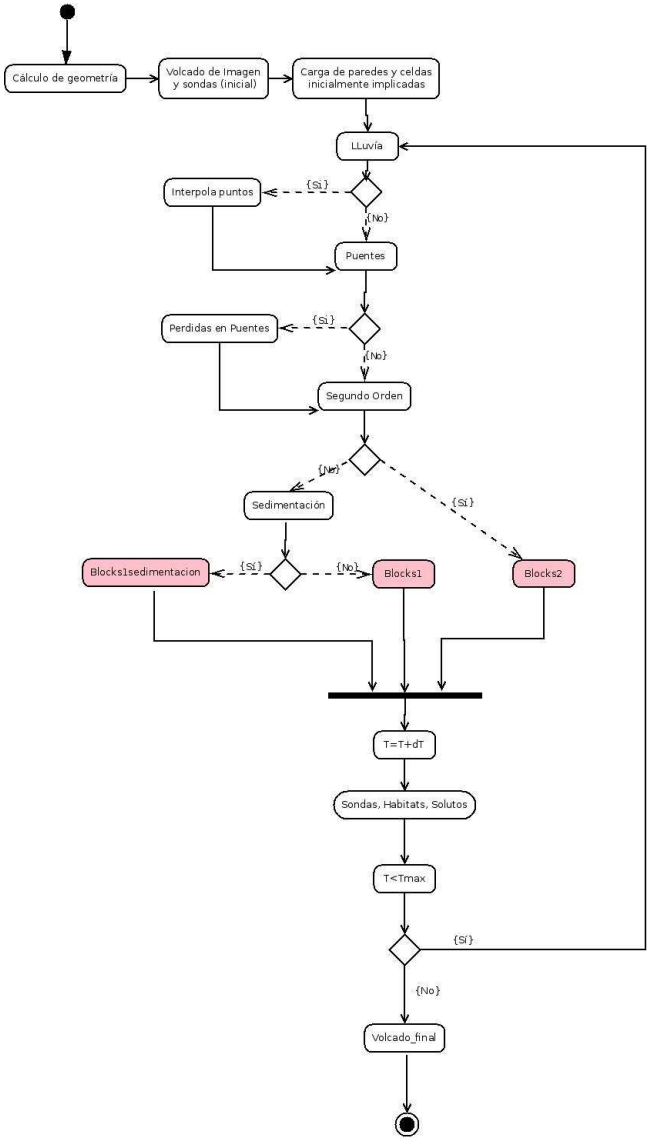


Figura H.1: DFD de SFS2D

Apéndice I
Manual de Subversion

I.1 Introducción

Los proyectos Software involucran, en general, a un gran número de programadores, analistas, diseñadores y directores del mismo, lo que hace necesario controlar los cambios periódicos que se vayan desarrollando en el mismo.

Ante esta necesidad aparece unos elementos fundamentales en el desarrollo de este tipo de proyectos, que son los llamados sistemas de control de versiones, formando parte de la Gestion de la Configuración junto con el control de cambios.

Históricamente, la primera herramienta que apareció fue SCCS, de los laboratorios BELL, en 1972, dando paso posteriormente a RCS y CVS, siendo esta última la que tomó más relevancia en los 80 utilizándose incluso en el desarrollo del kernel del sistema operativo SunOS. Con la aparición de nuevos lenguajes y tecnologías, la herramienta quedó relativamente obsoleta (aunque todavía en uso) por su poca compatibilidad con ficheros destinados a documentación o implementación web, y fue con la intención de mejorar estos puntos con los que en el año 2000 apareció Subversion de CollabNet Inc. llegando en 2009 a ser aceptado en el Apache Incubator, pudiendo recibir financiación de la Apache Software Fundation. Actualmente Subversion está siendo utilizado en proyectos como GCC, GNOME, Python o FreeBSD.

La herramienta Subversion (SVN en adelante) nos permite llevar un control absoluto sobre los diferentes cambios que se puedan producir en un proyecto Software, tanto a nivel de código como de documentación.

I.2 Estructura de un proyecto Software

Los proyectos software sufren multiples cambios a lo largo de su vida, pudiendo ser éstos clasificados según la relevancia de los mismos y por consiguiente determinando si esos cambios son vitales para ser introducidos en la versión actual de desarrollo.

Es frecuente que el equipo de desarrolladores de un proyecto, este bien diseminado según las funcionalidades del mismo (e.d. dedicación paralela y no solapada), pero en muchos de los casos, este tipo de distribuciones no pueden llevarse a cabo de una forma trivial, y es necesario que varios desarrolladores estén trabajando en un mismo módulo. Este último caso es similar al que ocurre en muchas aplicaciones científicas, en las cuales existe un investigador principal que

desarrolla gran parte del código fuente y de forma esporádica aparecen modificaciones de otros, mediante las cuales se proponen funcionalidades nuevas creadas de forma paralela.

Este tipo de desarrollos deben hacerse bajo dos pautas:

- Documentar cualquier cambio producido en el código principal y en los subcódigos en desarrollo
- Mantener un orden bajo el amparo de un árbol de proyecto.

La primera ha de ser establecida por el autor principal del código debiendo ser seguido por todos los desarrolladores mientras que para la segunda será para la que es necesaria utilizar herramientas de control de versiones.

I.2.1 Estructura de proyectos software en ámbitos científicos

Para definir la estructura de este tipo de proyectos, antes debemos indicar qué elementos componen el proyecto y de que manera. Para ello podemos basarnos en MÉTRICA, una metodología promovida por el Ministerio de Administraciones Públicas del Gobierno de España para la elaboración y sistematización de actividades propias del ciclo de vida de los proyectos software, especialmente para el ámbito de las administraciones públicas, pero válidas y recomendables para cualquier otro.

Según la metodología MÉTRICA v.3, los elementos de configuración incluyen:

- Ejecutables
- Código Fuente
- Modelos de Datos
- Pruebas
- ...

Todas ellas almacenando:

- Nombre
- Versión
- Estado
- Localización

Todo esto, debe estar conjuntado en el proyecto de una manera organizada según la estabilidad de los ficheros contenidos, pudiendo establecer esta clasificación:

- Principal (SVN Trunk): Rama de desarrollo principal
- Congelación (SVN Tags): Repositorio con las versiones principales *cerradas*
- Evloución (SVN Branches): Rama de evoluciones del desarrollo principal

I.2.2 Estructura de proyectos software en SVN

A partir de la definición de la estructura genérica mínima que debería tener todo proyecto software, la traducción a SVN, con modelo de carpetas, sigue como en la siguiente figura:

Para que este esquema se mantenga consistente, se recomiendan las siguientes prácticas:

- Actualizar al entorno local las últimas modificaciones que se hayan podido producir en el código. Es importante tener en cuenta que hemos de tener el proyecto ya descargado en local. Este comando es *update*
- Subir al servidor SVN los cambios del entorno local. SVN sólo permitirá estas modificaciones si no existen conflictos con el código ya existente en el repositorio, es decir, no permitira la modificación de un código si otro miembro del equipo ha modificado el mismo elemento de forma paralela desde la última sincronización de código. El comando es *Commit*.

Dado que este último requisito, puede plantear problemas, se propone el siguiente protocolo:

- Antes de comenzar la resolución de una tarea, se deberá asegurar la sincronización del código via *Update*.
- Una vez resuelta la tarea, se deberá haer otro *Update* para traer al entorno local los cambios que hayan podido ser realizados en paralelo, teniendo en cuenta que SVN sabrá integrar los cambios en la mayoría de los casos, siendo en otros necesaria la integración manual (p.e. un cambio en el mismo bucle). Estos casos deberían no existir en tanto que dos personas no deberían estar tocando la misma sección de código.
- Finalmente hacemos *commit* para hacer público el código desarrollado. El alcanze del commit debe limitarse al código relevante a la resolución de la tarea, y no mezclar desarrollos de distintas tareas en un mismo Commit. IMPORTANTE: Para completar el sentido de este tipo de gestiones, cabe destacar que SVN dispone de herramientas para cumplir con los requisitos antes citados mediante Logs. Esto es que cada vez que hacemos *commit*, se debe de incluir un pequeño comentario con los cambios acontecidos.

I.3 Uso de SVN

A través de SVN podemos poner en práctica las pautas anteriormente citadas para la gestión de proyectos. Por esta razón se va a discernir de dos usos de SVN en gestores y desarrolladores. Esto es que el Gestor (persona única) ha de ser el responsable de cerrar versiones y crear nuevas ramificaciones del proyecto, siendo igualmente interesante que todo el mundo conozca el protocolo por si pudiese ser interesante proponer un cambio al proyecto.

I.3.1 Instalación del cliente SVN

La elección del cliente SVN es libre en tanto que existen multiud de ellos, aunque aquí se recomendará *RapidSVN* por la existencia de éste tanto para distribuciones Linux, como para Windows o MacOS.

Si bien es cierto que las explicaciones que aquí se adjuntan están pensadas para el entorno gráfico, es posible para los usuarios de Linux igual para aquellos que están acostumbrados a trabajar con la consola, el manejo de SVN mediante comandos en consola.

I.3.1.1 Instalación en entornos Windows

1. Ir a *www.rapìdsvn.org/download/release/*
2. Elegir la última versión disponible
3. Descargar RapidSVN-x.x.xx.xxxx.exe
4. Seguir instrucciones de instalación

I.3.1.2 Instalación en Linux (Debian)

1. Acceder como Single-User
2. Ejecutar el comando *apt-get install rapidsvn*

I.3.2 SVN para gestores

La sección que sigue está recomendada sea hecha por el administrador del equipo en el que vaya a instalarse el servidor.

Nótese que la explicación está hecha para distribuciones Debian por el caso que nos ocupa.

I.3.2.1 Instalación del servidor SVN

Los pasos que se presentan son los básicos para la configuración de SVN. Es importante conocer el hecho de que toda la instalación se hará en Single-User. Cabe destacar que está incluido en los paquetes de Debian.

1. Acceder a Single User: *su*
2. Ejecutar el siguiente comando: *#apt-get install subversion subversion-tools*
3. Añadimos el grupo para el demonio asociado al servicio. *#groupadd -g 99 svnd*
4. Y el usuario del grupo: *#useradd svnd -d /srv/svn -g svnd -s /bin/false -m -k /dev/null -c Úsuario svnServe'-u 99*
5. Creamos la carpeta donde alojaremos los repositorios, por ejemplo */svn* con *mkdir /svn*
6. Creamos un script, alojado en */etc/init.d/* llamado *svnserve,* en el que se incluya lo siguiente. (Ver código)
7. *Añadimos* al final los enlaces simbólicos con *#update-rc.d svnserve defaults*

```
1 \#!/bin/sh
2 \#
3 \# start/stop svn (Subversion) server.
4 set -e
5 NAME=svnserve
6 DESC=\textquotedbl{}Subversion server\textquotedbl{}
7 DAEMON=/usr/bin/\$NAME
8 PARAMS=\textquotedbl{}-d -T -r /svn\textquotedbl{}
9 DAEMONUSER=svnd
```

```
10 test -x \$DAEMON || exit 0
11 . /lib/lsb/init-functions
12 start\_it\_up()
13 \{
14 log\_daemon\_msg \textquotedbl{}Starting \$DESC\textquotedbl{} \
      textquotedbl{}\$NAME\textquotedbl{}
15 start-stop-daemon --start --quiet --chuid \$DAEMONUSER:\$DAEMONUSER
16 --exec \$DAEMON -- \$PARAMS
17 log\_end\_msg \$?
18 \}
19 shut\_it\_down()
20 \{
21 log\_daemon\_msg \textquotedbl{}Stopping \$DESC\textquotedbl{} \
      textquotedbl{}\$NAME\textquotedbl{}
22 start-stop-daemon --stop --retry 60 --quiet --oknodo --exec \$DAEMON
23 log\_end\_msg \$?
24 \}
25 case \textquotedbl{}\$1\textquotedbl{} in
26 start)
27 start\_it\_up
28 ;;
29 stop)
30 shut\_it\_down
31 ;;
32 restart)
33 shut\_it\_down
34 start\_it\_up
35 ;;
36 {*})
37 echo \textquotedbl{}Usage: /etc/init.d/\$NAME \{start|stop|restart\}\
      textquotedbl{}
38 >\&2
39 exit 1
40 ;;
41 esac
42 exit 0%
```

I.3.2.2 Creación del repositorio

La creación del repositorio se realizará mediante el comando *svnadmin create /svn/nombreProyecto* siendo *nombreProyecto* el nombre del repositorio. Esta operación se realizará sólo en Single-User.

Para realizar la estructura de carpetas propuesta anteriormente, podemos realizar la siguiente operación:

1. Creamos en local la carpeta *nombreProyecto*
2. Dentro de ella creamos *trunk, branches y tags*
3. Utilizamos el siguiente comando:

```
1 svn import /nombreProyecto svn://shrek.cps.unizar.es/nombreProyecto -m '
     Creacion de estructura del proyecto'
```

I.3.2.3 Creación y gestión de Usuarios

Es importante saber que es posible establecer permisos para los usuarios de los repositorios, luego lo que se recomienda es establecer permisos SÓLO de escritura para los usuarios autorizados. Esto es:

1. Ir a *svn/nombreProyecto/conf*
2. Editar el fichero *svnserve.conf* añadiendo las lineas
   auth-access=write
   anon-access=none
   password-db=passwd
3. En el fichero passwd contenido en la misma carpeta, añadir la relación de usuarios contraseñas con el siguiente formato
   usuario1=con1
   usuario2=con2
   usuarioN=conN
4. NOTA IMPORTANTE: en el fichero svnserve.conf es importante que exista una linea (generalmente comentada por defecto, si es así descomentarla) en la que ponga lo siguiente
   anon-access=none
   ya que de no ser así cualquier persona podría ver el contenido de lo que existe en el repositorio.

I.3.3 SVN para desarrolladores

I.3.3.1 Ajustes preliminares

Para poder completar las funcionalidades de VisualSVN vamos a realizar estos ajustes:

1. Para integrar la funcionalidad del editor de textos, vamos a ir al menú Preferences, y ahí incluiremos el editor que utlizamos (por ejemplo, para gedit utilizar */usr/bin/gedit*)
2. Para la funcionalidad Diff y Merge podemos utilizar Meld, includo en los paquetes Debian. Lo instalamos mediante *apt-get install meld*

I.3.3.2 Cierre de versiones

En ciertos momentos del proyecto, puede ser interesante el estabilizar un cierre de una versión y en consecuencia, puede ser interesante regresar a una versión anterior en el caso, por ejemplo, de que se descubra un bug tras una entrega, donde se debería retomar el código desde la versión previa a la entrega, en lugar de continuar en la versión en desarrollo.

En lenguaje SVN, el cierre de una versión se denomina *tag de la versión desarrollada.* SVN maneja copias para este tipo de ramificación en el que sólo guarda una referencia a la rama y versión que se desea copiar, siendo el coste de ésta bajo en tiempo y en espacio, además de constante.

La recomendación es realizar estos cierres con cada hito del proyecto.

Lo interesante de estos tag es tener la certeza de la estabilidad de la versión, lo que implica que NUNCA se debe modificar un tag tras su creación. SVN no pone restricciones sobre esta operación luego es responsabilidad de los desarrolladores el seguir esta buena práctica.

Una vez creado el tag, el desarrollo debe continuar bien en *trunk* o en *branch.*

No debemos olvidar que el tag no es más que una anotación o referencia a un punto del desarrollo en el nivel físico, aunque sí a nivel lógico es el cierre de la versión.

El comando para realizar esto, es *svn copy* y se realizará haciendo una copia arrastrando la carpeta al destino deseado, e indicándole *copy* para la operación.

I.3.3.3 Ramificación del código

El uso de los branches es útil para crear una rama independiente de desarrollo para trabajar en alguna funcionalidad nueva de un proyecto que todavía no se quiere incorporar a la linea principal.

Al crear una nueva ramificación en *branches* el programador cambiará su working-copy a este nuevo branch y trabajará sobre él.

Las operaciones de commit serán sobre su branch, sin afectar al desarrollo principal. Cuando estos cambios estén listos se realizará la operación *merge.* Se ha de tener en cuenta que la ramificación del proyecto supone complicar la estructura del mismo, lo que implica que sólo se deben abrir las necesarias, siendo útil la aprobación de ésta por el director o responsable del proyecto.

I.3.3.4 Fusión de cambios

La operación *Merge* antes citada hace referencia a este punto, en el cual cabe destacar la existencia del comando *Patch* con diferencias en cuanto a lo que fusiona; El primero fusiona cambios en directorios inclusive, mientras que el segundo sólo lo hace de ficheros.

Es importante el uso de estos comando dado que la aplicación manual de la modificación puede ser susceptible del error humano al trascribir los cambios realizados.

I.3.3.5 Modificación de código fuente

La modificación del código fuente reqiere de ser cauto en los pasos a seguir. En primer lugar efectuamos doble click sobre el fichero a modificar en el repositorio local que hemos descargado en el bookmark. Una vez finalizada la modificación volvemos a la ventana de RapidSVN. Veremos que está marcado en rojo con el status *modified* siendo necesario subir los cambios al servidor. Esto es realizar *Update* presionando sobre el botón derecho del ratón para despues ejecutar *Commit* con el botón derecho tambien. Debemos incluir la modificación que hemos realizado para completar la utilidad de la herramienta.

Esta operación es importante realizarla cada vez que acabamos la funcionalidad que estamos desarrollando como máximo, y como mínimo cada día o periodo de elaboración de código.

I.3.3.6 Control de Log's

Seleccionando un fichero, podemos ir, mediante el menú contextual *Query>Log* para ver los cambios que hemos ido realizando revisión a revisión, visualizando además el comentario que habíamos incluido (de ahí la importancia de realizarlos) e incluso visualizar el fichero en ese estado con *view.*

I.3.3.7 Diff

La herramienta *diff* es el apoyo fundamental para el control de cambios. En el menú contextual del Log, podemos presionar ese comando para visualizar las diferencias entre el archivo que tenemos actualmente en nuestro equipo.

Apéndice J
Ejemplo de malla particionada

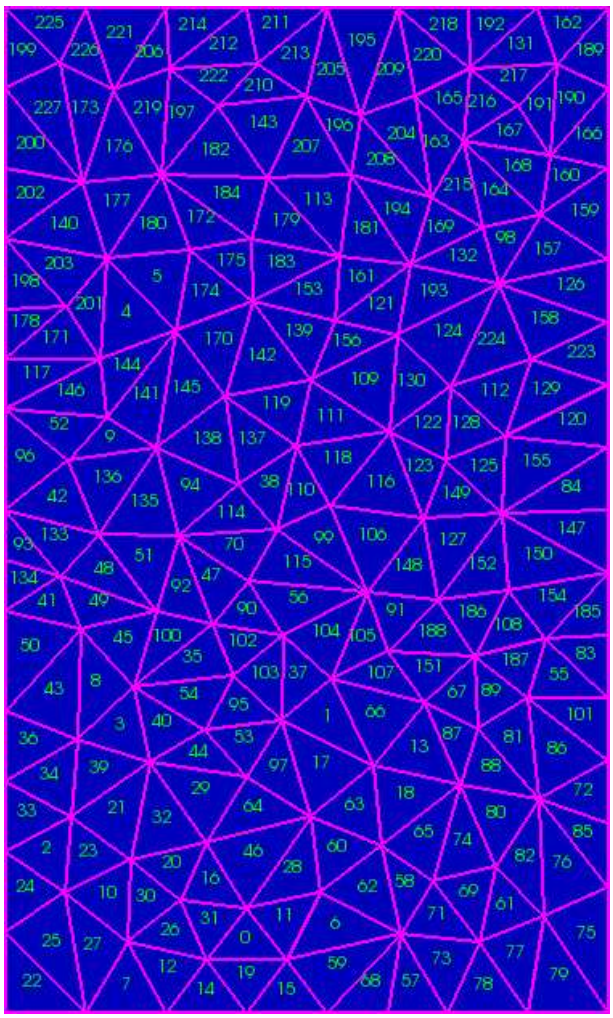


Figura J.1: Malla de cálculo original

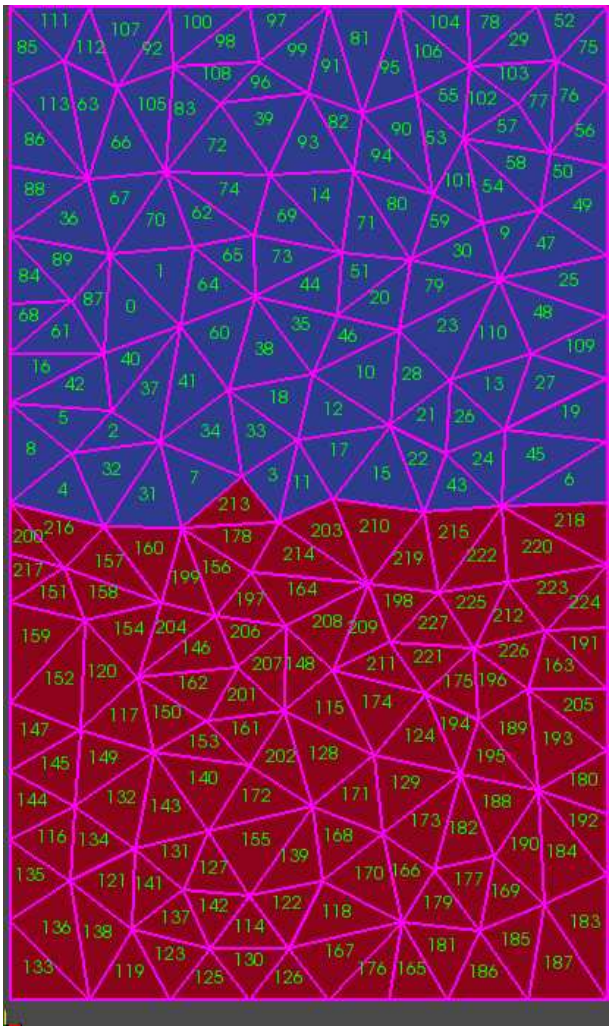


Figura J.2: Malla de cálculo aplicando 2-partición (Nótese la descomposición en la indexación de las celdas)