Tesis de máster perteneciente al *Máster de Informática y Matemáticas Aplicadas en Ciencias e Ingeniería*, titulada :

Estudio de Paralelización del modelo hidrodinámico COHERENS para sistemas multicore mediante OpenMP

Autor : Francisco López Castejón
Dirigido por : Domingo Giménez Cánovas

2, Septiembre, 2009

Resumen

El objetivo de este trabajo es obtener para el código paralelizado del modelo hidrodinámico marino COHERENS [1], un sistema mediante el cual el usuario pueda conocer con qué número de procesadores obtendrá una mayor eficiencia en el tiempo de ejecución según el tamaño del problema a resolver. El código original se presenta en forma secuencial, por lo que se realizará un estudio teórico del mismo que permita identificar las partes del código más susceptibles de ser paralelizadas y los parámetros que afectan al tiempo de ejecución. El incremento de capacidad del hardware de procesamiento de los sistemas multicore, nos ha llevado a utilizar estos sistemas y OpenMP [2] para su programación por ser el estándar *de facto* para programación en memoria compartida.

Índice

1. Introducción

Actualmente la protección del medio costero-marino y la comprensión de los procesos físicos, químicos y biológicos que se dan en él, es un problema de gran interés tanto para el sector científico como para la empresa privada. A pesar de haberse avanzado mucho en el tema, aún existen gran cantidad de factores que dificultan la total comprensión del mismo y cómo podría comportarse en el caso de darse alguna variación en los parámetros que lo regulan. Definiremos un modelo de circulación costera (*MCC*) como una representación numérica, espacial y temporal de los principales procesos físico-químicos que afectan al comportamiento de las masas de agua costeras [3]. La capacidad de permitirnos obtener una representación aproximada de la realidad hace de los *MCC* una herramienta fundamental a la hora de estudiar el medio marino y su comportamiento.

El aumento en la última década de la capacidad de cálculo de las computadoras monoprocesador ha permitido afrontar nuevos trabajos en el campo de los *MCC*, como por ejemplo abordar estudios de fenómenos a microescala que requieren una gran resolución espacial u obtener predicciones del clima marítimo en un breve espacio de tiempo. Sin embargo, ha sido la capacidad de algunos ordenadores de trabajar en paralelo con varios procesadores simultáneamente, junto a la creación de nuevos *MCC* o la adaptación de los códigos de los ya existentes con el fin de aprovechar toda la potencia de cálculo del cómputo paralelo, lo que ha permitido conseguir alcanzar con éxito las nuevas metas planteadas.

1.1. Objetivos

Los objetivos de este trabajo son: la paralelización mediante el uso de OpenMP del *MCC* denominado COHERENS, y el estudio de la relación entre el tamaño del problema a resolver y el número de procesadores con el que se obtiene la eficiencia máxima en la ejecución de la simulación a realizar. La consecución de estos objetivos, permitirá, usando la mínima cantidad de recursos (procesadores), obtener la mayor reducción posible del tiempo de computación empleado, así como permitir, a un usuario novel del código paralelizado, establecer, de una manera muy acertada, el número de procesadores a usar en cada parte del código.

La elección de este tipo de paralelismo con memoria compartida es debido a la gran difusión que los sistemas multicore tienen hoy en día, lo que permite acceder a un mayor número de plataformas sobre las que realizar las diferentes pruebas de rendimiento una vez paralelizado el software. Para la realización de este estudio se usará el modelo COHERENS. Esta elección viene determinada por :

- Ha sido muy usado para realizar estudios sobre la hidrodinámica costera, tanto por universidades como por empresas privadas [4, 5, 6].
- Se distribuye libremente y permite realizar cambios en su código.

- Posee un código muy estructurado, lo que posibilita una mejor comprensión de su esquema de trabajo.
- Está desarrollado en fortran, lo que permite el uso de librerías y compiladores ampliamente extendidos para la programación paralela [7].
- Existe una documentación del mismo que facilita la tarea de conocer la función de cada una de las subrutinas que lo componen.
- Tener experiencia en su uso al haberse utilizado para la realización de estudios hidrodinámicos [8] y de dispersión de contaminantes [9] en la empresa Taxon S.L dedicada a la consultoría ambiental.

1.2. Trabajos relacionados

La diferenciación entre los distintos modelos existentes en las actualidad no responde a un único criterio, ya que la aparición de un nuevo *MCC* responde a situaciones particulares del mismo, tales como aquellos desarrollados con una filosofía similar pero por diferentes organismos [10], los destinados a resolver problemas concretos tales como la circulación antártica [11], o el uso de diferentes estrategias de resolución numérica de las ecuaciones de la hidrodinámica [12]. Independientemente del fin del modelo desarrollado, la mayoría de ellos han tendido en los últimos años a afrontar la paralelización de sus códigos con el fin de reducir sus tiempos de cálculo [13, 14, 15].

1.3. Herramientas

Las herramientas utilizadas para la consecución de los objetivos marcados al inicio de este trabajo podemos dividirlas en dos categorías:

- Hardware: Se utilizarán distintos entornos multicore para analizar el paralelismo independientemente de la plataforma. En particular se trabajará con:
  - Ordenador portátil de doble núcleo. Este solo ha sido usado en la fase de desarrollo.
  - Cluster *Rosebud* de la Universidad Politécnica de Valencia con 6 nodos, dos tetras dual core correspondientes a los nodos rosebud05 y rosebud06, dos biprocesadores dual core y dos monoprocesadores cuadcore. De los nodos disponible se ha utilizado el nodo rosebud06.
  - Cluster *Hipatia* de la Universidad Politécnica de Cartagena con 14 nodos de 8 cores Xeon E5462 Quad Core, 2 nodos de 16 cores Xon X/350 Quad Core y 2 nodos de 4 cores Xeon 5160 Dual Core.

- Software
  - Modelo de Circulación Costera COHERENS.
  - En Rosebud se ha usado la versión 11 del compilador ifort perteneciente a intel y en Hipatia el compilador gfortran integrado dentro del compilador gcc en su versión 4.1.0.
  - Liberías OpenMP [2] integradas en los compiladores ifort y gfortran.

1.4. Metodología

La metodología de trabajo será:

- *Estudio teórico del modelo secuencial*
  Como primer paso se realizará un estudio teórico del código secuencial del programa con el objetivo de determinar cuáles son las subrutinas que mayor influencia tienen sobre el tiempo de cálculo total y así centrar en ellas el esfuerzo de paralelización.

- *Pruebas de rendimiento y escalabilidad en diferentes sistemas multicore*
  Se probará el código paralelizado en diferentes sistemas multicore con el fin de estudiar su comportamiento para diferentes tamaños de problema y número de procesadores.

- *Experimentación con diferentes estrategias de paralelización*
  Se probarán diferentes estrategias de parametrización y de autoptimización del código con el fin de estudiar con cual se obtendría una mayor reducción del tiempo de cálculo.

1.5. Organización del trabajo

El objetivo de este proyecto es establecer diferentes estrategias de paralelización y determinar la más conveniente para que el código pueda establecer de manera automática el número de procesadores a usar en cada tipo de bucle para conseguir una reducción de los tiempos de cálculo cercanos al máximo que se podría obtener. De esta manera el usuario del código no se implicaría de ninguna manera en esta decisión.

Para alcanzar los objetivos planteados, se han seguido los pasos que esquemáticamente se enumeran en el anterior apartado. Antes de empezar a comenzar el análisis del código, se va a realizar una exposición de las características del modelo COHERENS, desde el punto de vista matemático, numérico y computacional. De esta manera se pretende dar una visión global del modelo y su esquema de funcionamiento. Una vez conocido éste, se pasará a realizar un estudio teórico de los tiempos de cálculo que tendrían cada una de las partes que componen el modelo. Posteriormente, se mostrará, mediante un ejemplo, cómo se realizaría la parelización del código, para lo que se paralelizará el

bucle que tenga un mayor coste de tiempo en la función del modelo COHERENS. La localización de esta función se habrá podido determinar gracias al estudio teórico llevado a cabo previamente. Una vez alcanzado este punto, ya conoceremos cómo funciona el modelo, cuales son las partes del código que tienen un mayor coste computacional, así como un ejemplo de cómo se paralelizarían cada uno de los bucles existentes en el código de COHERENS. El conjunto de bucles que realiza el modelo se clasificaran en 3 tipos en función del número de flops que ejecutan. Finalmente, se pasará a probar diferentes estrategias de paralelizacón en cada uno de los bucles tipo, para de esta manera poder establecer cual sería la mejor decisión a la hora de establecer el número de threads óptimo a usar en cada bucle.

2. El modelo COHERENS

El modelo denominado COHERENS (COupled Hydrodynamical-Ecological model for REgioNal and Shelf seas) fue desarrollado entre los años 1990 y 1999 por *Management Unit of the North Sea Mathematical Models*, *Napier University*, *Proudman Oceanographic Laboratory* y *British Oceanographic Data Center*, dentro del proyecto europeo *MAST PROFILE, NOMADS AND COHERENS*. Desde la publicación de su código en el año 2000 ha sido ampliamente usado tanto dentro como fuera de la Unión Europea, con multitud de propósitos tales como la investigación [4, 5, 6], la enseñanza o el uso como herramienta de trabajo por parte de la industria privada [8, 9]. COHERENS se desarrolló no sólo con el fin de resolver las ecuaciones de la hidrodinámica en 3 dimensiones, sino que, desde su origen, se pensó para funcionar acoplado a un modelo biológico [16] y otro de dispersión de contaminantes, tanto en forma lagrangiana (se simula la trayectoria seguida por una partícula en el fluido) como euleriana (para cada punto de la malla se estudian las variaciones en la concentración de la sustancia a simular).

El modelo biológico y el de dispersión actúan como módulos acoplados a un eje central encargado de la resolución de las ecuaciones hidrodinámicas, siendo opcional la resolución de estos módulos. Debido a que la mayor carga de cálculo está centrada en la parte hidrodinámica, será en ésta en la que centraremos nuestro trabajo.

El código del programa se escribió en Fortran 77 en forma secuencial, con una serie de variables que permiten según los valores que tomen, seleccionar diferentes técnicas de resolución o activar nuevos módulos. Estos módulos funcionan acoplados al encargado de resolver la hidrodinámica, el cual actúa como eje central del programa.

2.1. Modelo matemático

Las ecuaciones que rigen el movimiento de un fluido (en nuestro caso el medio marino) son las denominadas Ecuaciones de Navier-Stokes, en honor a sus desarrolladores Claude-Louis Navier y

George Gabriel Stokes, a finales del siglo XIX. Estas ecuaciones nos permiten hallar los valores de la velocidad del fluido (u, v, w) en cada una de las direcciones del espacio (x, y, z) (figura 1).

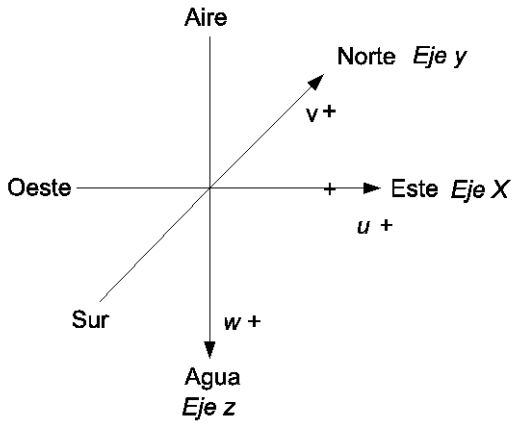


Figura 1: Sistemas de coordenadas usado en las ecuaciones.

Las ecuaciones de Navier-Stokes se obtienen a partir de unos principios básicos, como son la conservación de la masa (ecuación 1) y del momento (la ecuación 3 representa la conservación del momento para el eje X, la 4 para el Y y la 5 el Z), teniéndose en cuenta la ecuación hidrostática (ecuación 2).

La *ecuación de continuidad* es una ecuación de conservación de la masa, que expresa que la suma de las masas entrantes y salientes ha de ser igual a cero:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \tag{1}$$

La *ecuación hidrostática* nos dice que la presión existente a una determinada profundidad viene determinada por la densidad del agua y la gravedad terrestre:

$$\frac{\partial P}{\partial z} = \rho \cdot g \tag{2}$$

A partir de la *ecuación del momento* o ecuación del equilibrio entre las variaciones de cantidad de movimiento, que expresa el conjunto de fuerzas causantes de variaciones en el movimiento de la masa de agua, y aplicando las dos ecuaciones expuestas anteriormente, se obtienen las denominadas ecuaciones de Navier-Stokes, en la forma :

$$\overbrace{\frac{\partial u}{\partial t}}^{\text{Variación de la velocidad con el tiempo}} + \underbrace{u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}}_{\text{Advección horizontal}} + \overbrace{w\frac{\partial u}{\partial z}}^{\text{Advección vertical}} - \underbrace{fV}_{\text{Desviación geostrófica}} =$$

$$\overbrace{F_b x}^{\text{Fuerzas externas, ej: gravedad}} - \underbrace{\frac{1}{\rho}\frac{\partial P}{\partial x}}_{\text{Variación de la presión X}} + \overbrace{\frac{\mu}{\rho}\nabla^2 u}^{\text{Viscosidad}} \tag{3}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + fV = F_b y - \frac{1}{\rho}\frac{\partial P}{\partial y} + \frac{\mu}{\rho}\nabla^2 v \tag{4}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = F_b z - \frac{1}{\rho}\frac{\partial P}{\partial z} + \frac{\mu}{\rho}\nabla^2 w \tag{5}$$

En las ecuaciones 3, 4 y 5 encontramos las variables:

- $u \rightarrow$ Componente del eje X de la velocidad.
- $v \rightarrow$ Componente del eje Y de la velocidad.
- $w \rightarrow$ Componente del eje Z de la velocidad.
- $F_b \rightarrow$ Parametrización de la turbulencia.
- $t \rightarrow$ Tiempo.
- $P \rightarrow$ Presión.
- $\rho \rightarrow$ Densidad.
- $\mu \rightarrow$ Viscosidad.
- $g \rightarrow$ Gravedad.
- $f \rightarrow$ Parámetro de Coriolis.
- $\nabla^2 f \rightarrow$ Operador Laplaciano $\left(\frac{\partial^2 f}{\partial x} + \frac{\partial^2 f}{\partial y} + \frac{\partial^2 f}{\partial z}\right)$

Las ecuaciones implementadas en COHERENS son una derivación de las de Navier-Stokes, ya que se han asumido una serie de aproximaciones, siendo las más importantes la de *Boussinesq* y la de la incompresibilidad del fluido. A la hora de estudiar la dinámica de fluidos, las diferencias de densidades entre dos masas de agua pueden despreciarse, excepto en aquellos casos en que la densidad venga multiplicada por la gravedad terrestre, ya que ese gradiente de densidad sí puede provocar un movimiento del fluido [3]. Esta aproximación recibe el nombre de *Boussinesq*, en honor a su descubridor. Cuando decimos que un fluido es incompresible se está expresando que si sometemos a este a una presión no se producirá una variación en la densidad del mismo, por lo que las variaciones de densidad en las masas de agua marinas serán debidas a otros factores diferentes a la presión, como por ejemplo las variaciones de temperatura y salinidad del agua.

2.2. Modelo numérico

El modo por el cual el COHERENS resuelve las ecuaciones de Navier-Stokes es el denominado splitting [17, 18], basado en el desarrollado por Blumberg & Mellor en 1987 [19]. Este método separa el cálculo del movimiento de las masas de agua en 2D (modo externo), en el que se resuelve el transporte medio en la columna de agua, del cálculo de los valores de la corriente en cada una de las capas en que haya sido discretizado el eje vertical, 3D (modo interno). Este desacoplamiento supone un gran ahorro computacional, ya que la frecuencia de cálculo de cada uno de los modos es diferente, siendo el externo el que con más frecuencia se ejecuta. Sin embargo, la relación existente entre la frecuencia de cálculo del modo interno y externo, debe de ser controlada por el usuario debido a los cálculos realizados en el modo interno no sólo sirven para resolver la ecuación 3D, sino que una vez finalizado, se corrigen los valores 2D con los hallados en el 3D. Si el desacoplamiento existente entre ambos modos es muy grande, podrían darse errores de cálculo que llevarían a provocar inestabilidades del modelo, produciéndose un error en el mismo. Algunos autores [20] han establecido la relación entre la frecuencia de cálculo 3D/2D en un valor de 10.

El objetivo de la resolución de las ecuaciones expuestas anteriormente es el de hallar los valores de las variables u, v, w y ξ (elevación del mar) en cada uno de los puntos (x, y, z) en los que se descompone el área de trabajo (figura 1), para cada uno de los instantes de tiempo t del periodo temporal establecido.

La imposibilidad de resolver explícitamente el sistema de ecuaciones parciales que conforman las ecuaciones 1 y 2 hace que sea necesario recurrir a la vía numérica para su resolución. En el caso de COHERENS el método utilizado es el de diferencias finitas. Mediante este, se calcula el valor aproximado de las derivadas parciales a partir de la diferencia entre el valor actual en un punto y el que tendrá en el siguiente paso de tiempo, si es una derivada parcial respecto al tiempo, o en el punto más cercano al que estamos calculando, si es una derivada parcial respecto al espacio. Por tanto, el método de diferencias finitas implica una discretización tanto del espacio como del tiempo. El conjunto de puntos para los que se calculan las variaciones espaciales recibe el nombre de malla,

y cada uno de los puntos se llama nodo. En la figura 2 podemos ver un ejemplo de una malla usada para la aplicación del modelo en el Mar Menor, pudiendo apreciarse los nodos que la conforman.

A la hora de realizar la discretización espacial, si trabajamos con una distancia entre dos nodos consecutivos pequeña, obtendremos una mayor resolución en los resultados calculados por el modelo, pero también aumentará considerablemente el número de nodos que conforman la malla. Hay que tener en cuenta que esta mayor resolución supone un aumento en el tiempo de cálculo al ser mayor el número de puntos a resolver, por lo que habrá que buscar un equilibrio entre resolución del modelo y el tiempo de cálculo que podemos invertir en la obtención de resultados.

Figura 2: Ejemplo de malla de trabajo para el cálculo mediante diferencias finitas.

A la hora de discretizar un determinado dominio espacial horizontal existen diferentes tipos de malla según el punto en el que se hallan los valores de u, v, w y ξ. El modelo COHERENS usa un tipo de mallas denominadas Arakawa-C debido a sus descubridores, *Messinger* y *Arakawa* [21]. Este tipo de malla resuelve los valores de u, v y w en la frontera entre nodos consecutivos mientras que el valor de ξ lo resuelve en el centro del nodo. En las figura 3 podemos ver un esquema de este tipo de mallas, así como la nomenclatura usada para su descripción. A la distancia entre dos nodos consecutivos en el eje X, Y, Z se le denomina Δx, Δy y Δz respectivamente. Para poder identificar cada uno de los nodos, estos vienen determinados por sus coordenadas (i,j,k), las cuales son relativas al punto (0,0,0) de nuestra malla. Las principales ventaja del uso de este tipo de malla son "que no induce soluciones espureas y minimiza el esfuerzo computacional" [22].

COHERENS usa una discretización vertical de tipo sigma, la cual, a diferencia de la cartesiana en la que el dominio vertical se divide en niveles a distintas profundidades (z), realiza una transformación de la forma $\sigma = (z - \xi)/H$ (figura 4), donde H es la profundidad respecto al nivel medio del mar y

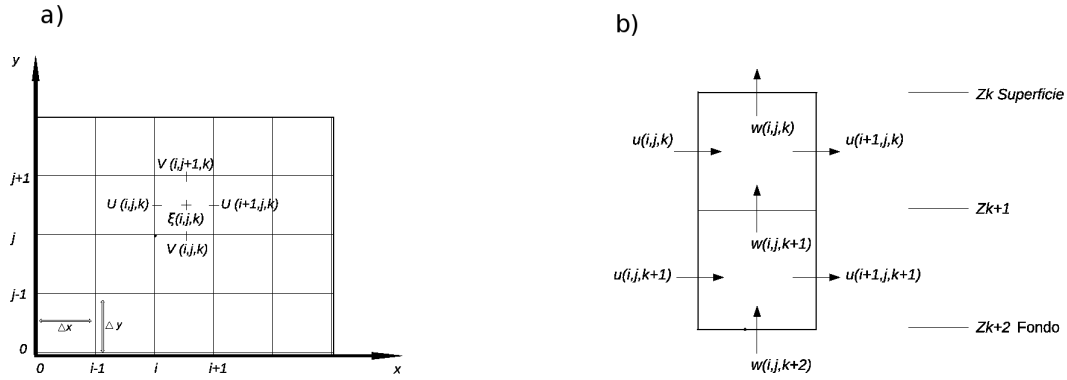


Figura 3: Discretización del área de trabajo horizontal (figura a) y vertical (figura b) de tipo Arakawa-C usada por COHERENS.

ξ la elevación del nivel del mar. El que COHERENS use este tipo de discretización vertical supone dos ventajas: (1) los niveles σ se adaptan a la superficie libre y a la batimetría y (2) se consigue igual número de capas en zonas someras y en las profundas. Esta técnica se empezó a aplicar a modelos meteorológicos [23] pasando a ser usada de una manera muy extendida por los *MCC* en la actualidad [19, 24].

2.3. Modelo computacional

COHERENS posee una serie de variables de control [25], las cuales permiten al usuario, a partir del valor que se les asigne, bien activar o desactivar los diferentes módulos que posee (biológico, de transporte de sedimentos, dispersión de vertidos...) o seleccionar distintos métodos de resolución para el cálculo de diferentes procesos, por ejemplo, a la hora de calcular los coeficientes de la turbulencia, el usuario puede elegir entre resolverlos mediante un esquema de turbulencia cerrado o por formulación algebraica.

La elección de uno u otro método de resolución, así como la activación o desactivación de los módulos que posee, influye sobre el tiempo de cálculo empleado por COHERENS. Para el presente estudio se han activado sólo aquellos módulos que son necesarios para la resolución de las ecuaciones de la hidrodinamica y de advección-difusión. En la tabla 1 se muestran los valores asignados a las variables de control de COHERENS para el estudio realizado en los siguientes apartados. Estos valores corresponden a los más comunmente usados en las simulaciones hidrodinámicas realizadas con COHERENS, ya que ofrecen la mejor relación entre la calidad de los resultados y los tiempos de

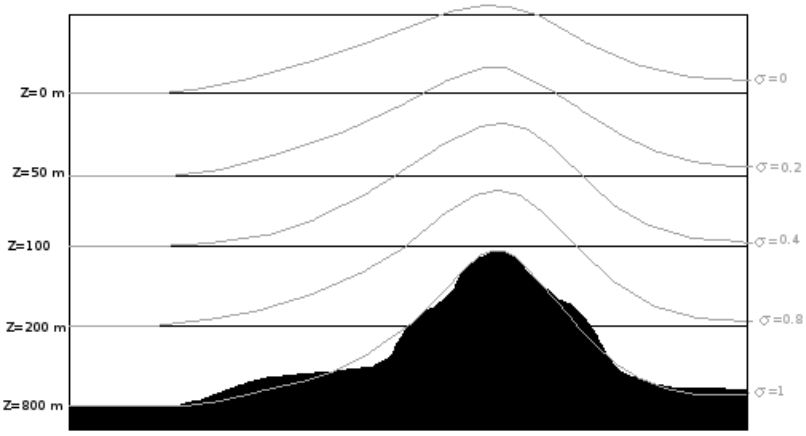


Figura 4: Discretización vertical de forma cartesiana, eje Y izquierdo, y tipo sigma, eje Y derecho.

Tabla 1: Valores asignados a las variables control para este estudio.

Variable	Valor		Variable	Valor
IOPTK	1		ITFORM	5
IBSTR	1		IADVS	3
JDIFS	2		ITYPOBU	2
JTYPOBU	2		IADVC	3
IODIF	2			

cálculo.

El modelo computacional de COHERENS se divide en 3 grandes bloques con diferentes funciones y frecuencias de ejecución. El primero de ellos calcula o lee las condiciones iniciales del modelo, ejecutándose una sola vez. El segundo bloque conforma el grueso del programa, ya que en él se encuentra el bucle temporal encargado del cálculo de las distintas variables en los intervalos de tiempo seleccionados por el usuario. En este bloque existen partes encargadas del cálculo de las variables en 3D y otras en 2D, cada uno con diferentes frecuencias de cálculo, tal y cómo se vio en la sección 2.2. La frecuencia de cálculo 2D (DELT) puede ser calculada previamente por el usuario mediante la condición de Courant-Friedrichs-Lewy (condición CFL [26]), debiendo de ser DELT <CFL (ecuación 6) para asegurar que la simulación produce resultados correctos. Mediante esta condición se pretende evitar los errores de los cálculos numéricos aproximados que aumentan rápidamente en cada paso

de tiempo. Si el tamaño de la rejilla es inferior a la distancia recorrida en el intervalo de paso de tiempo por la onda más rápida (onda de gravedad) que permita la ecuación, los errores aumentarán y deteriorarán la solución física.

$$CFL = \frac{0{,}5 \cdot \text{Anchura del nodo más pequeño de la malla de trabajo}}{\sqrt{\text{Gravedad} \cdot \text{Profundidad Máxima del dominio a modelizar}}} \tag{6}$$

Por tanto, suponiendo que tengamos un escenario con una profundidad máxima de 1000 metros y que el nodo más pequeño tiene un tamaño de 100 metros, el valor máximo del paso de tiempo que asegure la estabilidad del sistema será de 0.5058 segundos. En el caso de que estuviésemos calculando las corrientes para un periodo de 2 días (172800 segundos), podríamos conocer que el modelo hará un total de 342120 bucles 2D, calculando los valores de la corriente en 2D cada 0.5058 seg. La frecuencia de cálculo 3D se calcula a partir de la multiplicación de la frecuncia 2D por una constante (IC3D) establecida por el usuario. La elección del valor de IC3D depende en gran medida de la experiencia del usuario, debiendo de ajustarla a un valor que evite la aparición de errores en los cálculos numéricos pero reduzca el tiempo de calculo 3D respecto al 2D. El valor de IC3D usado por defecto es de 10 [20]. Por tanto, en nuestro ejemplo, el paso de tiempo 3D será de 5.058 segundos, realizando un total 34212 bucles para el cálculo de las corrientes en 3D.

En la figura 5 se puede ver un resumen del modelo computacional de COHERENS con sus funciones principales y los procesos que se resuelven en cada una de ellas.

3. Estudio teórico

Este estudio ha consistido en el cálculo del tiempo teórico de ejecución del programa. No se pretende hallar el valor exacto de este, sino derivar un modelo de ese tiempo. Para ello se ha realizado un recuento del número de operaciones en coma flotante (flops) que se llevan a cabo en cada una de las funciones que componen el programa, ya que son estas operaciones las que suponen un mayor coste en tiempo de cálculo. En la figura 6 se puede ver el coste, en unidades de flops, de cada una de las funciones principales, calculándose en función de las variables x= número de nodos en el eje X, y= número de nodos en el eje Y y z= número de niveles sigma.

Cada una de estas funciones principales está compuesta por varias subfunciones, cada una con un coste en tiempo de cálculo propio. Para ilustrar la metodología aplicada para obtener el número de flops representados en la figura 6, se va a utilizar la función CRRNT2, encargada de la resolución 2D de la ecuación del momento, ya que es esta la que tiene un mayor coste computacional ($350xy + 86xy + 86xy$). En la figura 7 podemos ver cada una de las subrutinas usadas dentro de la función CRRNT2, encargadas de resolver :

- BOUNDC → Condiciones de contorno para el modo 2D.

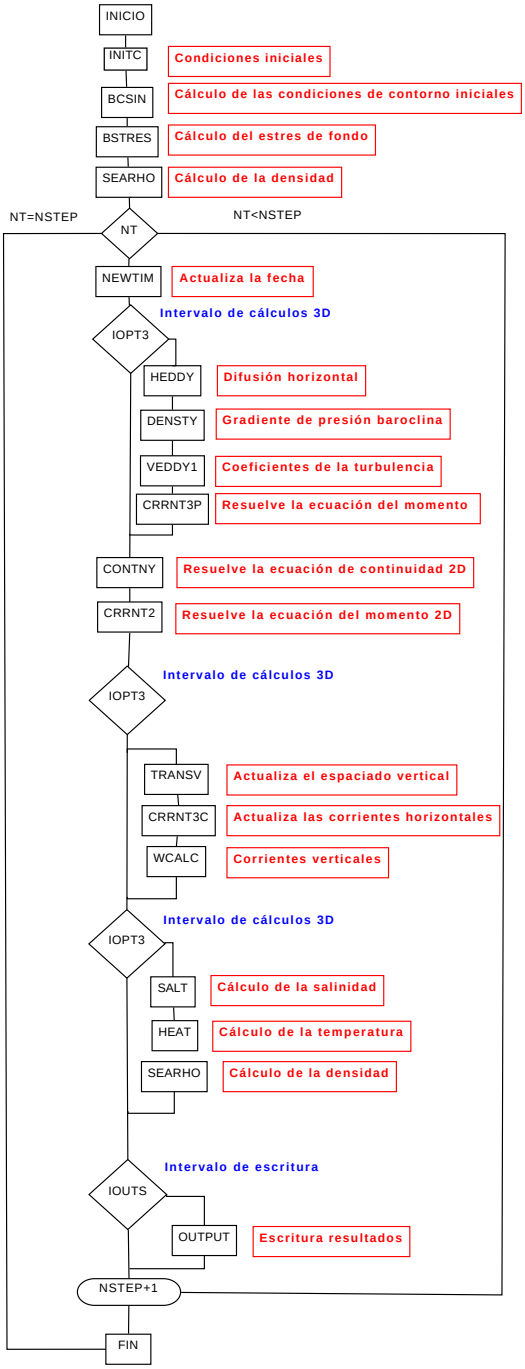


Figura 5: Esquema del modelo computacional de COHERNS con las variables de control establecidas en la tabla 1.

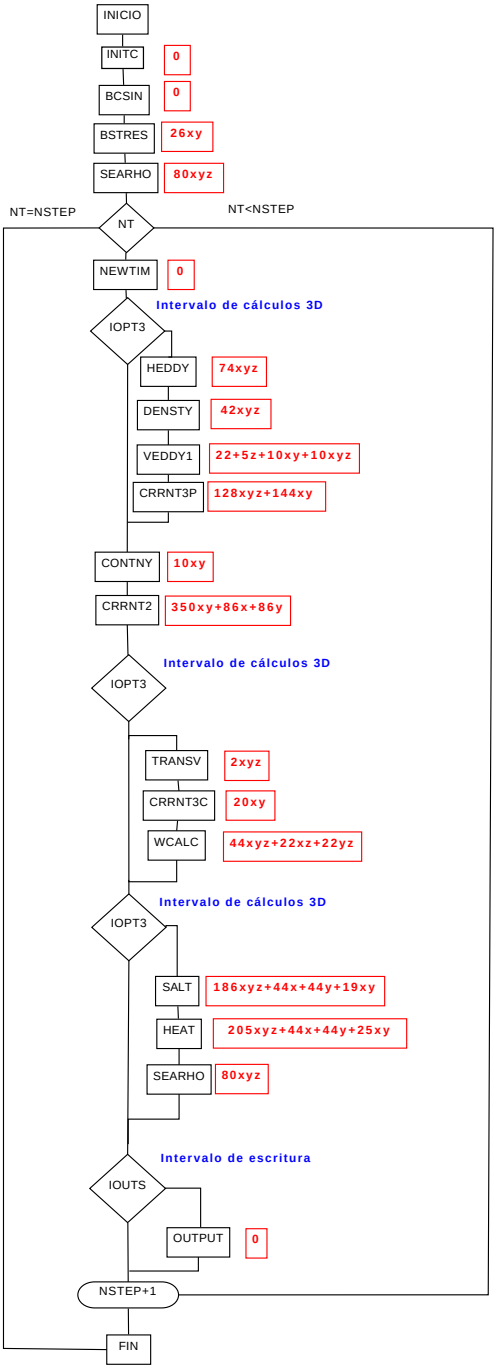


Figura 6: Esquema del modelo teórico del coste computacional de COHERNS con las variables de control establecidas en la tabla 1

- HAD2DU → Cálculo 2D de la advección y difusión horizontal para el momento de la componente U de la corriente.
- HAD2DV → Cálculo 2D de la advección y difusión horizontal para el momento de la componente V de la corriente.
- UDCALC → Resuelve la ecuación del momento para la componente U (velocidad en el eje x) de la corriente.
- VDCALC → Resuelve la ecuación del momento para la componente V (velocidad en el eje y) de la corriente.

De igual manera, dentro de la función CRRNT2, se utilizará aquella subrutina que tenga un mayor coste (HAD2DU) (figura 7), y siguiendo esta línea de trabajo, dentro de la subrutina HAD2DU, se ha seleccionado el bucle con un mayor número de flops, cuyo código podemos ver en la figura 8, el cual servirá de ejemplo para explicar la metodología aplicada.

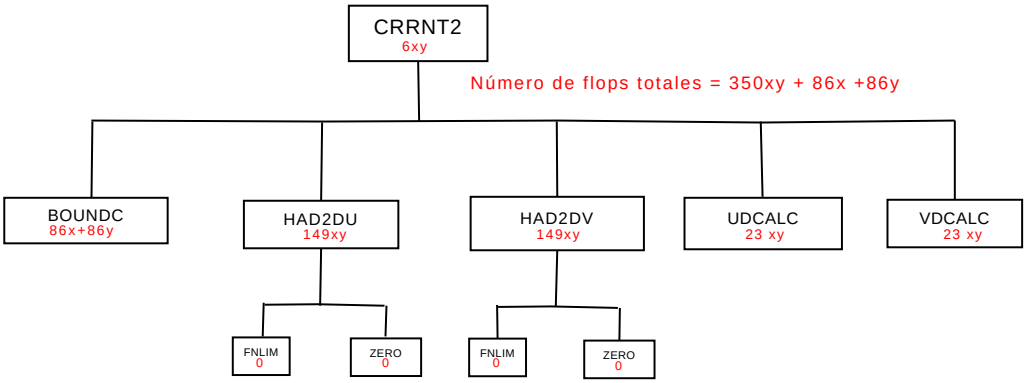


Figura 7: Modelo teórico de la función CRRNT2. Dentro de cada uno de los cuadrados podemos ver el nombre de la función y en color rojo el número de flops que resuelve. Siendo el número de flops totales el sumatorio de estos valores

Al final de cada línea del código mostrado en la figura 8 podemos ver el número de flops que le corresponden. Contabilizando el número total de operaciones realizadas, tendremos un coste de 24 flops. Destacar que no se contabilizan todas las apariciones de flops, sino solamente aquellas que se realizan. Por ejemplo, entre las líneas 7-16 del código 8 aparecen 15 flops, sin embargo, según el valor de la variable I, sólo se ejecutarían 5 flops. De igual manera, según el valor que tengla la variable NPIY, la cual nos indica si el lateral correspondiente al ejeY del nodo para el que se van a realizar los cálculos es tierra (NPIY=0) o agua (NPIY>1), el número de flops que se ejecutan dentro del bucle podría ser 0. En el caso analizado consideramos NPIY=1 para todos los nodos.

```
1          do 521 i=1,nc
2          do 521 j=2,nr
3           if (npiy(j,i).eq.1) then
4            ydifv = (ydiflv(j,i)-ydiflv(j-1,i))/(gy2v(j)*cosphiv(j))          3f
5            xdifv = 0.5*(xdiflv(j,i+1) + xdiflv(j-1,i+1) -
6         1                 xdiflv(j,i)   - xdiflv(j-1,i))/gx2v(j,i)          5f
7            if (i.eq.1) then
8             ydifu = (ydiflu(j,i+1) - ydiflu(j,i))
9         1             /(0.5*gx2v(j,i+1)+1.5*gx2v(j,i))                      5f
10           elseif (i.eq.nc) then
11            ydifu = (ydiflu(j,i) - ydiflu(j,i-1))
12        1             /(0.5*gx2v(j,i-1)+1.5*gx2v(j,i))                      5f
13           else
14            ydifu = (ydiflu(j,i+1) - ydiflu(j,i-1))
15        1             /(0.5*(gx2v(j,i-1)+gx2v(j,i+1))+gx2v(j,i))            5f
16           endif
17           vdh2d(j,i) = ydifv + xdifv + ydifu                                2f
18           vdh2d(j,i) = vdh2d(j,i) + sphcurv(j)*
19        1       (0.5*(xdiflu(j-1,i)+xdiflu(j,i))
20        2        -2.0*sphcurv(j)*dheddyvv(j,i)*vd2(j,i)/h2atv(j,i))           9f
21           endif
22   521    continue
```

Figura 8: Bucle con mayor número de flops dentro de la subrutina HAD2DU, la cual pertence a la función CRRNT2.

4. Paralelización del código

Recordar que el objetivo principal del trabajo no era la paralelización del código del modelo COHERENS, sino realizar un estudio del comportamiento de este, una vez paralelizado, frente a diferentes tamaños de problema, por lo cual sólo se ha realizado la paralelización en algunos de los bucles del código y no en todo el COHERENS, ya que, la paralelización de todo el código tendría un alto valor funcional pero no nos ayudaría a avanzar en los objetivos de este trabajo. Para ilustrar cómo se llevaría a cabo el proceso de paralelización, se ha seleccionado la porción de código mostrada en la figura 8.

El primer paso para la paralelización del código ha sido establecer cuales de las variables que en él aparecen son privadas de cada uno de los threads y cuales han de ser compartidas para permitir el acceso de todos los threads a ellas. Si observamos el código mostrado en la figura 8, se puede observar que las matrices *ydiflv*, *xdiflu*, *xdiflv* y *ydiflu* requieren el acceso a la posición *i* anterior y posterior del bucle realizado, por lo que debemos asegurarnos que estas no sean privadas. Así mismo, una vez calculadas las diferentes variables, todos los threads han de poder escribir en la matriz de resultados *vdh2d*, debiendo ser esta variable compartida. OpenMP solo considera como privadas aquellas variables que se declaren como tales. En el caso que estamos analizando, las variables *ydifv, xdifv* e *ydifu* deberán ser privadas, ya que cada uno de los threads las utiliza de manera independiente. En el caso de no declarlas como tales, cada thread intentaría acceder a una variable que esta siendo usada por otro thread, debiendo esperar a que este termine para su uso, perdiendo de esta manera la mejora de tiempos debida a la paralelización del código, así mismo el valor de una variable que no fuera privada obtenido por un thread podrían ser sustituido por el calculado por otro thread antes de que el primer thread hubiera hecho uso de los valores calculados por él, llevando esto a errores en los resultados obtenidos. En la figura 9, podemos ver un esquema con las variables declaradas privadas (cuadros verdes) y públicas (cuadros grises) en el código paralelizado, el cual podemos ver en la figura 10.

5. Pruebas de rendimiento y escalabilidad en diferentes sistemas multicore

se han seleccionado 3 bucles del código de COHERENS, siendo el criterio diferenciador entre ellos el número de flops que se resuelven, 3 flops (figura 11), 8 flops (figura 12), y 24 flops (figura 13). La elección de estos tamaño de bucle se ha realizado para recoger el rango existente en el modelo de flops a resolver en un solo bucle. Los bucles de 3 y 24 flops pertenecen a la función CRRNT2, mientras que el de 8 a la HAD2DU. Mediante el uso de diferentes tipos de bucle, se pretende comprobar, si para un mismo tamaño de problema el número de procesadores con el que se consigue la máxima eficiencia

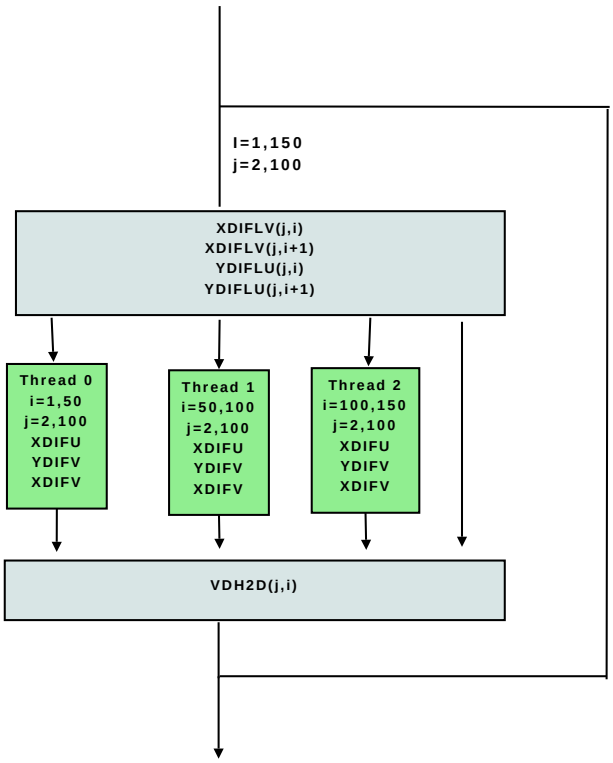


Figura 9: Diagrama de acceso a las variables privadas (XDIFU, YDIFV, XDIFV) y compartidas (XDIFLV, YDIFLU, VDH2D)

```
1       time_begin = omp_get_wtime ( )
2 c$omp parallel
3 c$omp& private (i,j,ydifv,xdifv,ydifu)
4 c$omp do
5        do i=1,nc
6        do j=2,nr
7         if (npiy(j,i).eq.1) then
8          ydifv = (ydiflv(j,i)-ydiflv(j-1,i))/(gy2v(j)*cosphiv(j))
9
10         xdifv = 0.5*(xdiflv(j,i+1) + xdiflv(j-1,i+1) -
11     1               xdiflv(j,i)   - xdiflv(j-1,i))/gx2v(j,i)
12         if (i.eq.1) then
13          ydifu = (ydiflu(j,i+1) - ydiflu(j,i))
14     1            /(0.5*gx2v(j,i+1)+1.5*gx2v(j,i))
15         elseif (i.eq.nc) then
16          ydifu = (ydiflu(j,i) - ydiflu(j,i-1))
17     1            /(0.5*gx2v(j,i-1)+1.5*gx2v(j,i))
18         else
19          ydifu = (ydiflu(j,i+1) - ydiflu(j,i-1))
20     1            /(0.5*(gx2v(j,i-1)+gx2v(j,i+1))+gx2v(j,i))
21         endif
22         vdh2d(j,i) = ydifv + xdifv + ydifu
23         vdh2d(j,i) = vdh2d(j,i) + sphcurv(j)*
24     1       (0.5*(xdiflu(j-1,i)+xdiflu(j,i))
25     2        -2.0*sphcurv(j)*dheddyvv(j,i)*vd2(j,i)/h2atv(j,i))
26        endif
27        end do
28       end do
29 c$omp end do
30 c$omp end parallel
31       time_stop = omp_get_wtime ( )
32       time_elapsed = time_stop - time_begin
```

Figura 10: Código paralelizado mediante OPENMP del bucle usado como ejemplo, cuyo código secuencial podemor ver en la figura 8.

se mantiene constante o por el contrario existe una relación entre el número de flops a resolver en ese bucle y la mejor eficiencia.

```
1  c$omp parallel
2  c$omp& private (i,j)
3  c$omp do
4        do  i=1,nc+1
5        do  j=1,nr
6           ud2f(j,i) = ud2f(j,i) + ud2(j,i)*delt/del3
7        end do
8        end do
9  c$omp end do
10 c$omp end parallel
```

Figura 11: Bucle con 3 flops perteneciente a la función CRRNT2 de COHERENS usado para las pruebas de rendimiento y escalabilidad.

Cada uno de estos tipos de bucles se ejecutó para diferentes configuraciones de tamaños de problemas, siendo en todos los casos igual el número de elementos en el eje X que en el Y, por lo que al referirnos al tamaño del problema lo haremos dándo el número de elementos que componen el eje X. Los tamaños usados han sido, desde 100 elementos hasta 1000 con un incremento de 100 y desde 1000 hasta 5000 con un incremento de 5000, siendo el total de tamaños de problemas usados de 18.

En la figura 14 se muestra el speed up obtenido para algunos de los tamaños de problema sobre los que se hicieron las pruebas, mostrando en el eje X el número de cores utilizados hasta un máximo de 8 y en el eje Y el speed up conseguido para cada uno de los tamaños de problemas. Los resultados expuestos en la figura 14 se presentan clasificados en función de la máquina en que se ejecutó, el tipo de bucle a resolver y el tamaño del problema.

```
1  c$omp parallel
2  c$omp& private (i,j)
3  c$omp do
4          do i=2,nc
5          do  j=1,nr
6           if (npix(j,i).eq.1) then
7            ud2(j,i) = dheddyvu(j,i)*((vd2atc(j,i)/h2atc(j,i)
8       1                 -vd2atc(j,i-1)/h2atc(j,i-1))/gx2u(j,i)
9       2                 +sphcur(j)*ud2(j,i)/h2atu(j,i))
10
11           endif
12           end do
13          end do
14 c$omp end do
15 c$omp end parallel
```

Figura 12: Bucle con 8 flops perteneciente a la función HAD2DU de COHERENS usado para las pruebas de rendimiento y escalabilidad.

Si observamos la figura 14 podemos observar, tal y como era de esperar, que conforme aumenta el tamaño del problema y el número de flops a resolver por el bucle, los speed up conseguidos son mejores, aunque en ningún caso superan el valor de 4. Sólo en el caso de la prueba realizada en *Rosebud* para un bucle de 3flops y un tamaño de problema de 5000, se ha obtenido un speed up cercano a 8, no siendo esta situación representativa del resto. Si comparamos los resultados obtenidos por *Hipatia* y *Rosebud*, vemos como el incremento del speed up conforme aumenta el tamaño del problema es mayor en *Rosebud*, siendo en esta máquina en la que se han obtenido mejores resultados.

Al representar (figura 15 y 16) para un determinado tamaño de problema, los speed up conseguidos para cada uno de los bucles, vemos como solamente cuando trabajamos con un tamaño de problema mayor a 600 y bucles que resuelven mas de 8 flops, los speed up empiezan a ser igual o mayores que 1, obteniéndose en los casos con valores por debajo de este límite mayores tiempo de cálculo con el código paralelo que con el secuencial. Si comparamos el comportamiento de *Hipatia* y *Rosebud*, vemos como en este último se han obtenido menor cantidad de speed up menores a 1.

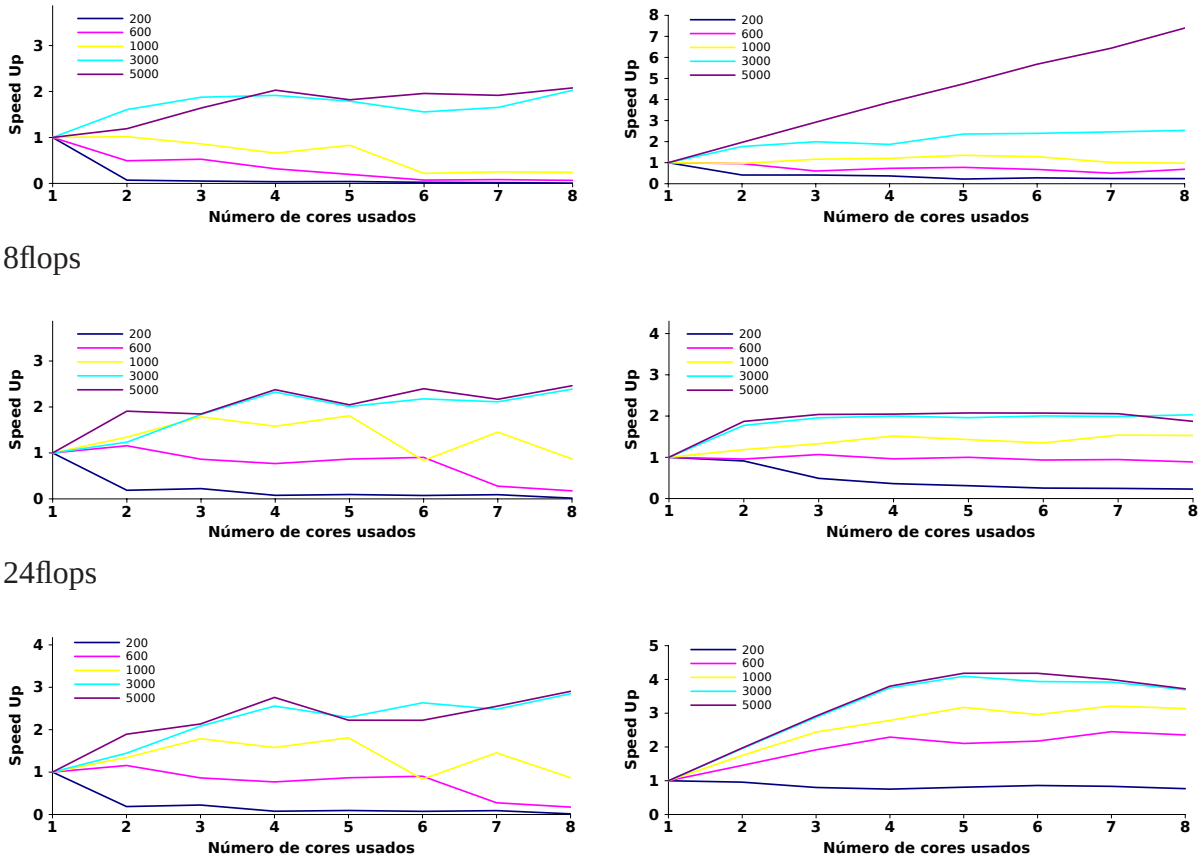
Los resultados obtenidos indican la importancia que tiene la realización de un estudio de escalabilidad y rendimiento del código paralelizado, ya que, tal y como se ha comprobado, no en todos los casos se obtienen mejores tiempos de ejecución usando el código paralelizado respecto al secuencial. Así mismo, el uso del máximo número de procesadores disponibles, tampoco nos asegura que la disminución del tiempo de cálculo sea mayor. También se pone de manifiesto el hecho de que según el tipo de bucle a paralelizar, el número de procesadores óptimo puede variar. Todos estos factores indican la necesidad de adoptar alguna estrategia de paralelización que nos permita usar un número

```
 1  c$omp parallel
 2  c$omp& private (i,j,ydifv,xdifv,ydifu)
 3  c$omp do
 4        do i=1,nc
 5        do j=2,nr
 6         if (npiy(j,i).eq.1) then
 7          ydifv = (ydiflv(j,i)-ydiflv(j-1,i))/(gy2v(j)*cosphiv(j))
 8
 9          xdifv = 0.5*(xdiflv(j,i+1) + xdiflv(j-1,i+1) -
10       1               xdiflv(j,i)   - xdiflv(j-1,i))/gx2v(j,i)
11          if (i.eq.1) then
12           ydifu = (ydiflu(j,i+1) - ydiflu(j,i))
13       1           /(0.5*gx2v(j,i+1)+1.5*gx2v(j,i))
14          elseif (i.eq.nc) then
15           ydifu = (ydiflu(j,i) - ydiflu(j,i-1))
16       1           /(0.5*gx2v(j,i-1)+1.5*gx2v(j,i))
17          else
18           ydifu = (ydiflu(j,i+1) - ydiflu(j,i-1))
19       1           /(0.5*(gx2v(j,i-1)+gx2v(j,i+1))+gx2v(j,i))
20          endif
21          vdh2d(j,i) = ydifv + xdifv + ydifu
22          vdh2d(j,i) = vdh2d(j,i) + sphcurv(j)*
23       1       (0.5*(xdiflu(j-1,i)+xdiflu(j,i))
24       2        -2.0*sphcurv(j)*dheddyvv(j,i)*vd2(j,i)/h2atv(j,i))
25
26         endif
27         end do
28        end do
29  c$omp end do
30  c$omp end parallel
```

Figura 13: Bucle con 24 flops perteneciente a la función CRRNT2 de COHERENS usado para las pruebas de rendimiento y escalabilidad.

Hipatia *Rosebud*

3flops



8flops

24flops

Figura 14: Speed-up obtenidos para los diferentes tipos de bucle y de tamaño de problema en *Hipatia* y *Rosebud*

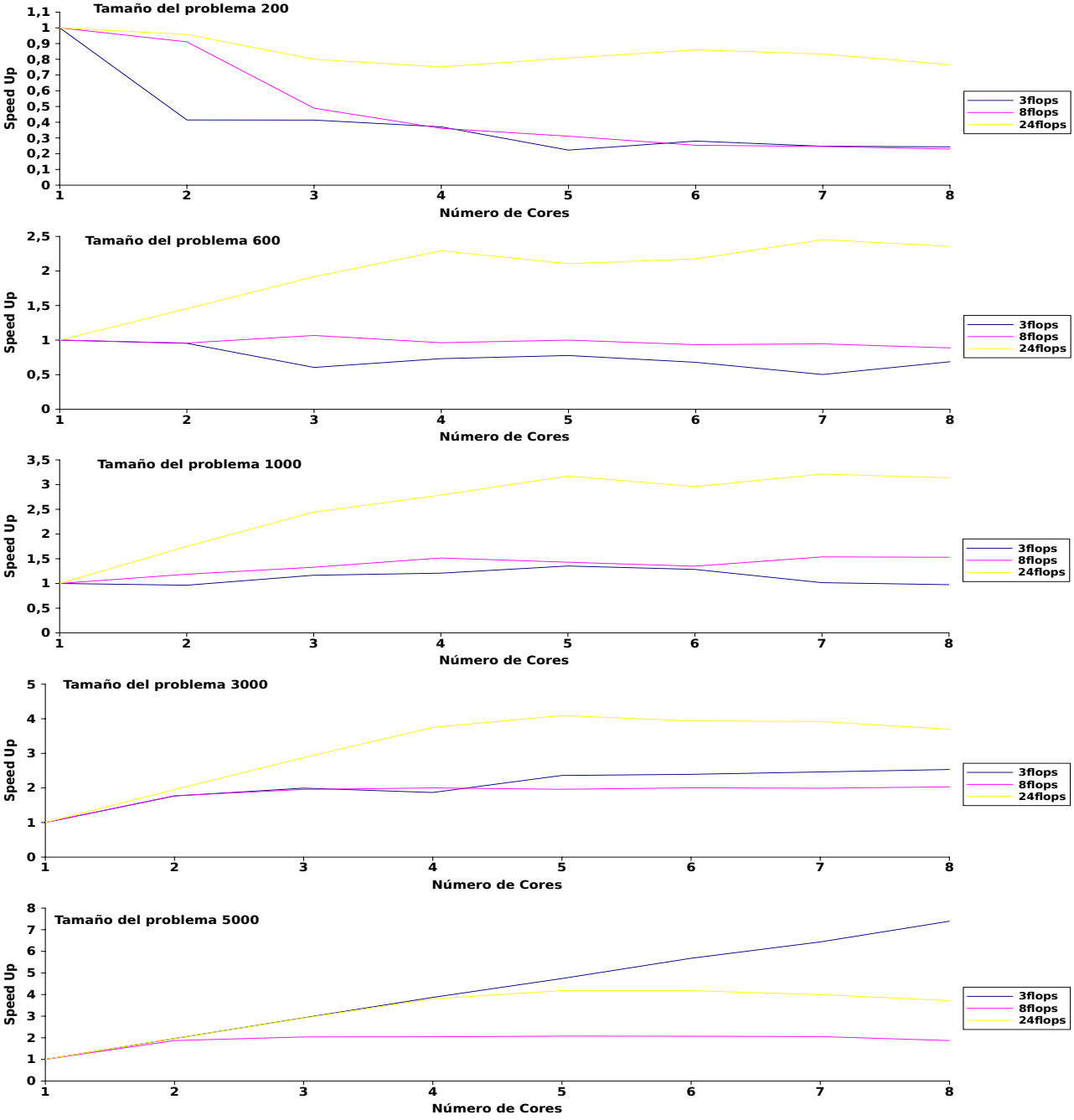


Figura 15: Comparativa del speed up obtenidos para los diferentes tipos de bucle y tamaños de problema en *Rosebud*. En el eje X podemos ver el número de cores usado del total de 8 que tiene el nodo en el que se ejecutaron las pruebas.

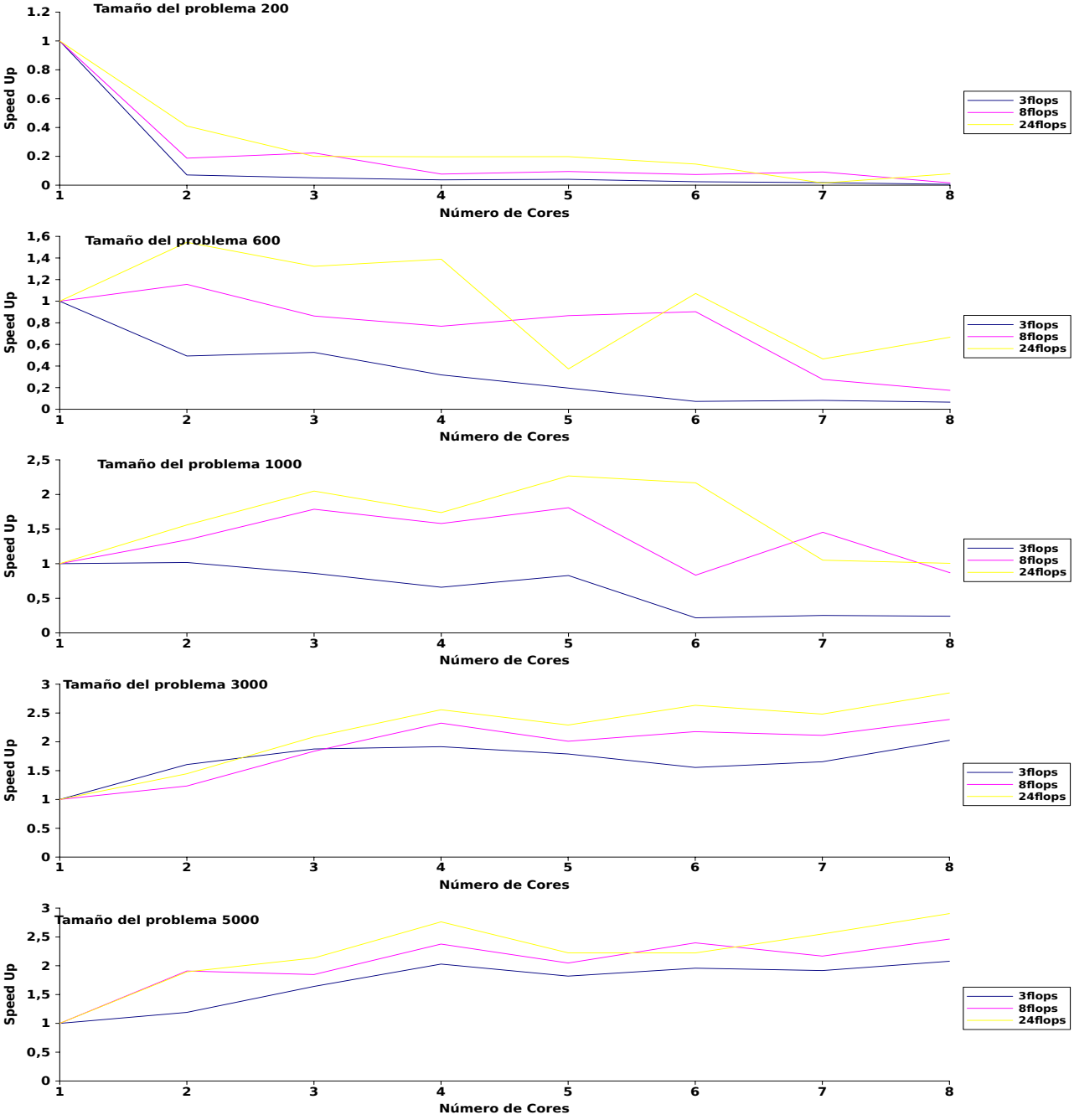


Figura 16: Comparativa del speed up obtenidos para los diferentes tipos de bucle y tamaños de problema en *Hipatia*. En el eje X podemos ver el número de cores usado del total de 8 que tiene el nodo en el que se ejecutaron las pruebas.

de procesadores variables según el tamaño del problema y el tipo de bucle que se paralelize, asi como la conveniencia de la autoptimización, aplicando técnicas utilizadas con anterioridad en otro campos [27] y [28].

6. Experimentación con diferentes estrategias de paralelización

Tal y como vimos en el apartado anterior, existe una necesidad de establecer una serie de estrategias previas a la hora de ejecutar un código paralelizado que nos aseguren que estamos obteniendo unos tiempos de ejecución de nuestro código cercanos al mejor posible en la máquina en la que estemos trabajando. La estrategia planteada en el presente trabajo está basada en la capacidad que tiene OpenMP, mediante la orden omp_set_num_threads, de establecer un número de threads a usar distinto para cada parte del código. De tal forma que todo el código paralelo no se ejecute con el máximo número de procesadores disponible, sino que para cada bucle paralelizado se intenta usar el número de procesadores con el que se obtenga una mayor disminución del tiempo de cálculo. Para facilitar los experimentos realizados se han establecido 3 tipos de bucles a partir del número de flops, coincidentes con los usados en el apartado anterior: bucles de 3 flops, 8 flops y 15 flops. Las pruebas experimentales se han realizado en *Hipatia* y en *Rosebud*, pero sólo se mostrarán los resultados obtenidos en *Rosebud*, al ser la metodología empleada en ambos idénticas y con el fin de facilitar la exposición de los resultados.

A partir de los datos obtenidos en las diversas pruebas, se ha podido determinar el número de cores con el que se obtiene una mayor disminución de tiempos para cada uno de los bucles tipo.

La primera de las estrategias a afrontar sería la realización, por parte de un usuario nuevo, de todas las pruebas que se han ejecutado en el presente estudio, llegando a obtener la tabla 2 para cada tipo de bucle, lo que le permitiría asignar el mejor número de procesadores para cada bucle según el tamaño de su problema.

La estrategia comentada en el párrafo anterior, supone un volumen de trabajo previo a la ejecución muy grande, siendo uno de nuestros objetivos el que el usuario se implique lo mínimo posible en la tarea de asignación del número de procesadores. Para ello aprovecharemos los conocimientos adquiridos en las pruebas anteriormente realizadas, tomando algunos de los tamaños de problema usados, como referencia para calcular el número de procesadores a usar en cada tipo de bucle. Los valores seleccionados los podemos ver en negrita en la tabla 2.

El cálculo del número de procesadores a partir de los valores de referencia se hará de dos formas distintas, para comprobar con cual de ellas se obtiene mejores resultados, que serán:

- Mediante la asignación del número de procesadores a partir del tamaño de referencia más cercano al tamaño del problema a resolver

Tabla 2: Número de threads con el que se ha obtenido menor tiempo para cada tamaño de problema para el caso de un bucle de 8 flops. En negrita se muestran los valores usados como referencia para las diferentes estrategias de paralelización

Rosebud			
Tamaño	Thread	Tamaño	Thread
200	1	2000	8
300	1	2500	8
400	2	**3000**	8
600	3	3500	8
800	5	4000	8
1000	7	4500	8
1500	8	**5000**	5

- Obteniendo el número de procesadores mediante el cálculo por interpolación a partir de los valores de referencia.

La última estrategia a afrontar por el usuario, sería el uso del máximo de procesadores disponibles, por lo que también se ha tenido en cuenta esta opción a la hora de realizar los experimentos.

En la figura 17 podemos ver los resultados obtenidos para cada tipo de bucle según la estrategia de paralelización aplicada, en el eje Y se muestra el cociente entre el tiempo obtenido con una determinada estrategia partido del mejor tiempo obtenido para ese tipo de bucle y tamaño de problema. Podemos observar que la menor reducción de tiempos se consigue con el uso de todos los procesadores disponibles. Al comparar el método de asignación por proximidad y de interpolación, comprobamos que de manera general se obtienen tiempos de cálculos más próximos al menor conseguido mediante el cálculo del número de procesadores por interpolación.

7. Conclusiones y trabajos futuros

A priori, cabría pensar que a la hora de ejecutar un código paralelizado, el uso del máximo número de procesadores disponibles implicaría un mejor tiempo de ejecución del código, pero esto no es así, como queda demostrado en el trabajo desarrollado. El análisis de la escalabilidad del código paralelizado ha puesto de manifiesto la existencia de un número de procesadores límite a partir del cual las reducciones de tiempo obtenidas son muy pequeñas, como podemos ver en la figura 14. También se ha comprobado como, el uso de un código paralelizado, puede suponer, en vez de una disminución en los tiempos de cálculo, un aumento de este (figura 15 y 16), siendo a partir de un

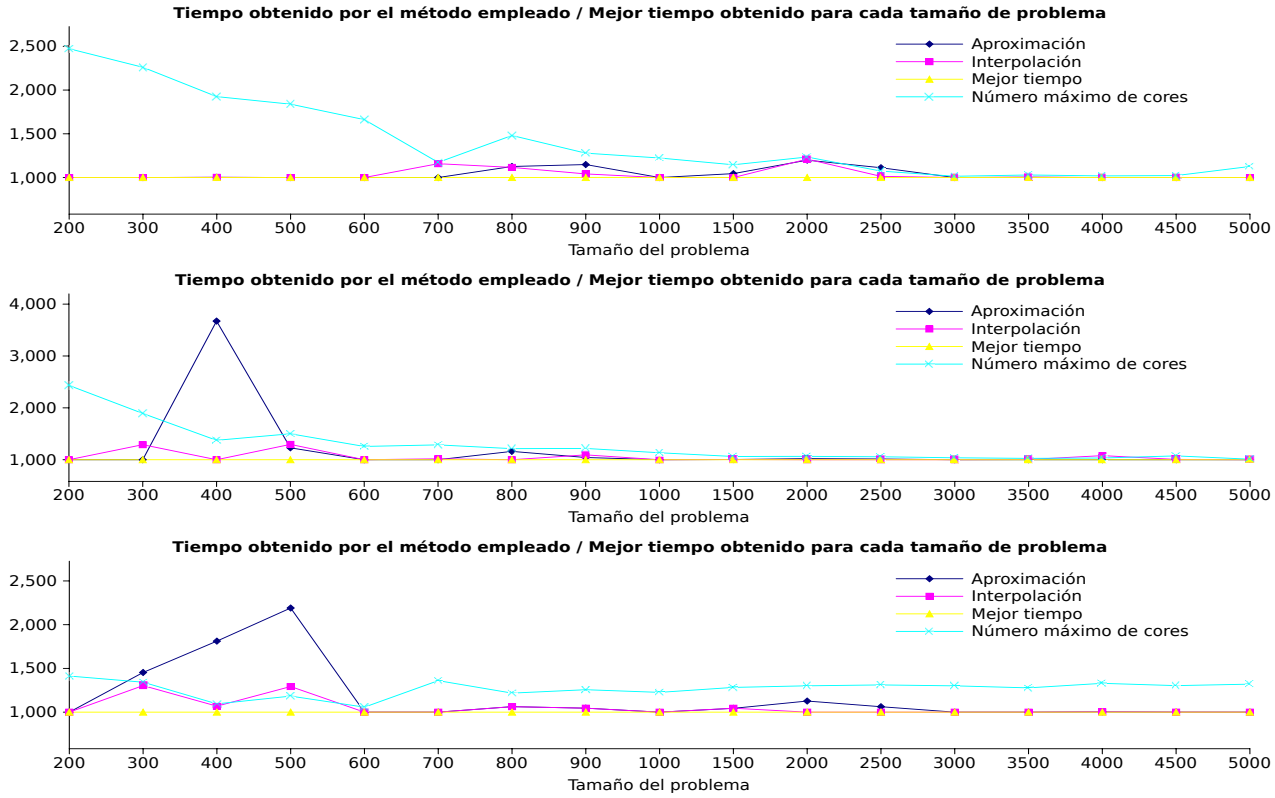


Figura 17: Comparativa de los tiempos obtenidos, para un bucle de 3, 8 y 24 flops, mediante interpolación, aproximación y uso del máximo de procesadores respecto al mejor obtenido para cada tamaño de problema.

tamaño de problema de x=600 e y=600 elementos, cuando la paralelización del código empieza a suponer una mejora, aunque leve, en los tiempos de ejecución.

Al haberse realizado las pruebas en dos máquinas distintas, *Rosebud* e *Hipatia*, se ha podido comprobar como el mismo código, según dónde se ejecute, presenta resultados diferentes aunque con un comportamiento general similar.

Para poder tener un mejor conocimiento de la respuesta del código a la paralelización, se han seleccionado 3 tipos de bucles de los existentes en COHERENS en función del número de flops que ejecutan (3 flops, 8 flops, 15 flops). Esto ha permitido comprobar, que a la hora de hallar el número de procesadores a usar, no solo hay que tener en cuenta el tamaño del problema a resolver, sino también el tipo de bucle que se está ejecutando. Las pruebas realizadas han mostrado que el speed up conseguido es mayor en el caso de tamaños de problemas grandes y con bucles que ejecutan 15 flops.

Por último, se han probado diferentes estrategias que la persona encargada de ejecutar el código paralelizado podría usar para determinar el número óptimo de procesadores para cada tipo de bucle. Para ello se han utilizado unos valores de referencia (tabla 2) extraídos de las diferentes pruebas realizadas. Los resultados obtenidos (figura 17) muestran como la peor opción el uso del máximo número de procesadores disponibles, ya que es con la que se obtienen menores reducciones de tiempo. La estrategia que ha presentado mejores resultados en la mayoría de los casos es la del cálculo del número de procesadores por interpolación, en comparación con el cálculo por el valor más próximo.

Por tanto, mediante esta trabajo se ha demostrado la importancia que tiene la realización de un estudio previo del código paralelizado, descartando la idea de que a mayor número de procesadores mejores tiempo de ejecución. Así mismo se ha comprobado como el uso de algunas de las estrategias planteadas durante la instalación del código, permitirían obtener unos tiempos de cálculo cercanos a los óptimos para la máquina en que se este ejecutando el código.

El poder dotar a COHERNES de la capacidad de determinar por sí mismo el número óptimo de cores a usar en cada parte de su código, es de gran importancia desde el punto de vista económico. Cuando se inició el proyecto de paralelización de COHERENS [29] para poder conseguir mejores tiempos de cálculo en las simulaciones realizadas en la empresa Taxon S.L, se pensó en comprar una máquina con un gran número de cores, siendo este un desembolso muy importante. Trabajos como el realizado, ponen de manifiesto que el estudio del comportamiento del código paralelizado, la adopción de estrategias adecuadas de paralelización y la creación de programas de autoptimazción, pueden suponen un gran ahorro para la empresa. En el caso concreto estudiado en este proyecto, la rentabilidad del gasto realizado en la compra de una máquina nueva expresada en euros/speed up, seria muy baja a partir de un número de cores mayor de 4, siendo, en términos económicos, la compra óptima para las simulaciones realizadas la de una máquina con 4 cores.

Los siguientes pasos a realizar una vez finalizado este trabajo, podrían ir enfocados en diferentes vías:

- Se han estudiado sólo los bucles 2D. Sin embargo, el modelo COHERENS posee gran cantidad

de bucles en 3D, por lo que habría que estudiar el comportamiento de estos para diferentes tamaños de problemas y número de flops.

- Las estrategias de paralelización realizadas se han basado en estudiar cómo afecta el número de procesadores al tiempo de cálculo, sin embargo se podrían afrontar otras estrategias, como por ejemplo la policompilación, mediante la que se pretende averiguar cual sería el compilador con el que se obtendrían mejores tiempos para nuestro problema.

- En el trabajo desarrollado, se han trabajado con mallas cuadradas, en el que el número de elementos en el eje X es igual al del eje Y. Sin embargo, no se puede asegurar que el comportamiento del código para un determinado tamaño de problema sea igual si los datos están distribuidos con una distribución cuadrada o rectangular, en la que el número de elementos en el eje X no coincide con el del Y.

- Solamente se han utilizado, en el presente estudio, dos máquinas diferentes, por lo que sería conveniente repetir los experimentos realizados en diferentes máquinas, para intentar establecer un comportamiento general común a todas.

- Desarrollo de una versión de COHERENS con capacidad de autoptimización, de tal forma que antes de ejecutarse la simulación el código decidiese por si mismo el número de cores óptimo a usar en cada parte del código. Esta línea de investigación ya se ha iniciado, y se esta preparando un trabajo para el *XVIII International Conference on Computatinal Methods in Water Resources* en Junio del 2010.

Referencias

[1] Página web de COHERENS. `http://www.mumm.ac.be/~patrick/mast/`.

[2] Página web de OpenMP. `http://openmp.org`.

[3] Brian Williams. *Hydrobiological Modelling*. Brian Williams, 2006.

[4] A. C. Cardoso, J. F. Lopes, and M. T. Moitac. 3D ecological modelling of the Aveiro coast (Portugal). In *International Congress on Environmetal Modelling and Software (iEMSs).*, page 156, 2008.

[5] Xiu-Bua, Liang-Sheng, and Hong-Sheng Zhang. Numerical simulation of summer circulation in the East China sea and its application in estimating the sources of red tides in the Yangtze river estuary and adjacent sea areas. *Journal of Hydrodinamics*, 19(3):272–281, 2007.

[6] D. Marinova, J. Galbiatía, G. Giordanib, P. Viarolib, A.Ñorrod, S. Beneivellic, and J.M. Zaldivara. An integrated modelling approach for the management of clam farming in coastal lagoons. *Aquaculture*, 269(1-4):306, 2007.

[7] Página web de g77. `http://gcc.gnu.org`.

[8] F. López, A. Perán, J. Gutiérrez, and A. Belmonte. Asistencia técnica para la elaboración de estudios de corrientes marinas en distintas zonas del archipielago canario, y la posterior simulación de la dispersión de contaminantes productos de la acuicultura. Technical report, Taxon Estudios Ambientales, 2004.

[9] F. López, A. Perán, J. Gutiérrez, and A. Belmonte. Simulación numérica de la dispersión de agua de rechazo de la desaladora de Denia. Technical report, Taxon Estudios Ambientales, 2005.

[10] Página web de ROMS. `http://www.myroms.org`.

[11] Página web de FRAM. `http://www.mth.uea.ac.uk/ocean/fram.html`.

[12] Daniel Y. Le Roux and Charles A. Lin. Finite elements for shallow-water equation ocean models. *Monthly Weather Review*, 126:1931–1951, 1998.

[13] M. J. Beare and D. P. Stevens. Optimisation of a parallel ocean general circulation model. *Ann. Geophysicac*, 15:1369–1377, 1997.

[14] Y. Ping Wang, Tony Song, Yi Chao, and Hongchun Zhang. Parallel computation of the regional ocean modelling system. *International journal of high performance computing applications*, 19:375–385, 2005.

[15] G. Sanino, V. Artale, and P. Lanucara. An hybrid OpenMP-MPI parallelization of the Princepton ocean model. *Paralell Computing Advances and Current Issues*, pages 222–229, 2001.

[16] P. Luytens, J. E. Jonse, R. Proctor, A. Tabor, P. Tett, and K. Wild-Allen. Coherens- a coupled hydrodinamical-ecological model for regional and shelf seas: user documentation. Technical report, Managament Unit of the Mathematical Models for the North Sea, MUMM, 1999.

[17] T. J. Simons. Verification of numerical models of lake Ontario. *Physical Oceanography*, 4:507–523, 1974.

[18] R. V. Madala and S. A. Piacsck. A semi-implicit numercial model of baroclinic oceans. *Computational Physical*, 23:167–178, 1977.

[19] A. Blumberg and G. L. Mellor. A description of a three-dimensional coastal ocean circulation model. *Coastal and Estuarine Sciences*, 4:1–16, 1987.

[20] Zygmunt Kowalik and Tadepalli Satyanarayana Murty. *Numerical modeling of ocean dynamics*. World Scientific, 1993.

[21] F. Messinger and A. Arakawa. Numerical methods used in atmospheric models. Technical report, GARP Pubication Series 17, 1976.

[22] A. Balzano. Evaluation of methods for numerical simulation of wetting and drying in shallow water flow models. *Coastal Engineering*, 34:83–107, 1998.

[23] N. A. Philips. A coordinate system having some special advantage for numerical forecasting. *Journal of the Atmospheric Sciences*, 14:184–185, 1957.

[24] Y. P Sheng, H. K. Lee, and K. H. Wang. *On numerical strategies of estuarine and coastal modeling. Estuarine and coastal modeling*. ASCE, 1990.

[25] P. J. Luyten, J. E John, P. Roger, T. Andy, T. Paul, and W. Karen. *COHERENS-User Documentation-Part V Reference Manual*. Managament Unit of the Mathematical Models for the North Sea, MUMM, 1999.

[26] R. Courant, K. Friedrichs, and H. Lewy. Über die partiellen differenzengleichungen der Mathematischen Physik. *Mathematische Annalen*, 100:32–74, 1928.

[27] J. Cuenca, D. Giménez, and J. González. Architecture of an automatic tuned linear algebra library. *Elsevier Science*, pages 187–220, 2004.

[28] L.P. García, J. Cuenca, and D. Giménez. Auto-optimization of linear algebra parallel routines: the cholesky factorization. In *PARCO 2005*, 2005.

[29] Francisco López. Paralelización del modelo hidrodinámico secuencial coherens para sistemas multicore mediante openmp. Meeting on Parallel Routines Optimization and Applications, 2008. Murcia.