Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXI, págs. 3129-3139 (artículo completo)
Alberto Cardona, Paul H. Kohan, Ricardo D. Quinteros, Mario A. Storti (Eds.)
Salta, Argentina, 13-16 Noviembre 2012

ACELERACIÓN DE UN MODELO DE VOLÚMENES FINITOS EN ARQUITECTURAS MULTI-, MANY-CORE

Ernesto Dufrechou[a], **Pablo Igounet**[a], **Pablo Santoro**[b], **Mónica Fossati**[b] **y Pablo Ezzatti**[a]

[a]*Instituto de Computación, Universidad de la República, 11.300–Montevideo, Uruguay,*
{edufrechou,pigounet,pezzatti}@fing.edu.uy

[b] *Instituto de Mecánica de los Fluidos e Ingeniería Ambiental, Universidad de la República, 11.300–Montevideo, Uruguay,* {psantoro,mfossati}@fing.edu.uy

Palabras Clave: Modelo hidrodinámico, volúmenes finitos, multi-core, GPU.

Resumen. En Uruguay, desde hace algunos años se está trabajando en el modelado numérico del flujo del Río de la Plata y el Frente Marítimo aplicando el modelo tridimensional MOHID. El MOHID utiliza una discretización en volúmenes finitos que resuelve las ecuaciones tridimensionales de Navier-Stokes con la aproximación hidrostática del campo de presiones permitiendo utilizar la técnica de modelos encajados. A través de esta metodología, es posible anidar grillas de resolución espacial creciente, forzando los modelos locales con resultados de aplicaciones de mayor escala. En particular, esto permite forzar el modelo del Río de la Plata con condiciones de borde calculadas con un modelo global de circulación del Atlántico Sur. Uno de los principales problemas de este tipo de enfoques son los altos tiempos de ejecución que implican la resolución de los modelos. Esta situación motiva el estudio e inclusión de técnicas de computación de alto desempeño (HPC) para la aceleración de los mismos.

En el trabajo se evalúan diferentes paradigmas de HPC, abordando tanto el uso de técnicas tradicionales basadas en paralelismo de memoria compartida para explotar procesadores multi-core, así como el uso de procesadores gráficos (GPUs) para acelerar los tiempos de cómputo de los modelos. En particular, y dado que los principales tiempos de ejecución del modelo son debidos a la resolución de sistemas lineales tridiagonales subyacentes de la aplicación del método ADI, los desarrollos se centran en esta etapa.

La evaluación experimental muestra que ambas técnicas aportan importantes mejoras en los tiempos de ejecución para la resolución de los sistemas lineales. Estos resultados alentadores, permiten conjeturar sobre las posibilidades de mejora del desempeño computacional de este tipo de modelos utilizando hardware de bajo costo y que ha mostrado evolucionar en sus capacidades de forma vertiginosa.

1. INTRODUCCIÓN

En Uruguay, en el Instituto de Mecánica de los Fluidos e Ingeniería Ambiental (IMFIA), desde hace algunos años se está trabajando en el modelado numérico del flujo del Río de la Plata y el Frente Marítimo aplicando diferentes modelos numéricos. En particular en los últimos años se ha utilizado con éxito el modelo tridimensional MOHID (Fossati y Piedra-Cueva, 2012). El MOHID (Trancoso et al., 2009) utiliza una discretización en volúmenes finitos que resuelve las ecuaciones tridimensionales de Navier-Stokes con la aproximación hidrostática del campo de presiones permitiendo utilizar la técnica de modelos encajados. A través de esta metodología, es posible anidar grillas de resolución espacial creciente, forzando los modelos locales con resultados de aplicaciones de mayor escala. En particular, esto permite forzar el modelo del Río de la Plata con condiciones de borde calculadas con un modelo global de circulación del Atlántico Sur. Uno de los principales problemas de este tipo de enfoques son los altos tiempos de ejecución que implica la resolución de los modelos. Esta situación motiva el estudio e inclusión de técnicas de computación de alto desempeño (HPC) para la aceleración de los mismos.

En el presente trabajo se evalúan diferentes paradigmas de HPC, abordando tanto el uso de técnicas tradicionales basadas en paralelismo de memoria compartida como el uso de procesadores gráficos (GPUs) para acelerar los tiempos de cómputo de los modelos. En primera instancia, se evalúa el uso de OpenMP, explotando así las capacidades de los procesadores multi-core. Posteriormente, se estudia el uso de procesadores gráficos (GPUs) para realizar el cómputo del modelo. En particular, y dado que los principales tiempos de ejecución del modelo son debidos a la resolución de sistemas lineales tridiagonales subyacentes de la aplicación del método ADI (Alternative Direction Implicit) para resolver las ecuaciones diferenciales, los desarrollos se centran en esta etapa. Por último, se evaluaron estrategias híbridas de cómputo, multi-core CPU + GPU, que permiten el uso de ambos dispositvos en forma colaborativa para la resolución de las ecuaciones del modelo.

Los resultados obtenidos sobre una plataforma de hardware dotada de una CPU conectada mediante bus PCI a una tarjeta NVIDIA mostraron importantes aceleraciones a nivel de la resolución de los sistemas, lo que impacta en el cómputo del modelo en su totalidad con ciertas mejoras. Estos resultados alentadores, permiten conjeturar sobre las posibilidades de mejora del desempeño computacional de este tipo de modelos utilizando hardware de bajo costo y que ha mostrado evolucionar en sus capacidades de forma vertiginosa.

El resto del documento se estructura de la siguiente manera. En la Sección 2 se introducen los conceptos básicos del modelo numérico MOHID y su aplicación para la simulación del Río de la Plata. Posteriormente, en la Sección 3, se describen las distintas implementaciones desarrolladas. Los experimentos llevados a cabo para evaluar y validar las propuestas se presentan en la Sección 4. Por último, se resumen las conclusiones principales del trabajo y se detallan las líneas de trabajo futuro en la Sección 5.

2. EL MODELO MOHID

El MOHID es una herramienta numérica para el modelado del flujo baroclínico basado en la resolución de las ecuaciones de Navier-Stokes. El modelo resuelve las ecuaciones de movimiento que representan el comportamiento de cuerpos de agua a superficie libre como océanos, estuarios y reservorios. Este tipo de sistemas están basados en las ecuaciones tridimensionales para el flujo incompresible, asumiendo válida las aproximaciones de Boussinesq y Reynolds, así como el equilibrio hidrostático (Neves et al., 1999).

La herramienta utiliza el paradigma de los volúmenes finitos para la discretización del do-

minio de cálculo. Para la resolución de las ecuaciones en el tiempo, el modelo utiliza una aproximación semi-implícita basada en las técnicas ADI (Alternative Direction Implicit). Las discretizaciones tipo ADI en el tiempo emplean una estrategia combinada, resolviendo alternadamente las ecuaciones en las direcciones horizontales verticales. En primer lugar se resuelve una dirección en forma explícita y la otra en forma implícita, y en segundo lugar se repite el procedimiento cambiando las direcciones. Esto permite en muchas ocasiones sortear problemas de estabilidad numérica presentes en los métodos completamente explícitos. Esta aproximación implica la resolución de sistemas lineales tridiagonales, que pueden ser eficientemente resueltos, por ejemplo con el método de Thomas (Golub y Van Loan, 1996).

El método para la resolución de sistemas tridiagonales TDMA (del inglés, TriDiagonal Matrix Algorithm), conocido comúnmente como algoritmo de Thomas, es una simplificación de la factorización LU que permite explotar las particularidades de las matrices tridiagonales. Para esta clase de sistemas, el algoritmo de Thomas obtiene la solución con un $O(n)$ de operaciones, donde n es la dimensión de la matriz. Notar que la eliminación gaussiana (o factorización LU) implica $O(n^3)$ operaciones para matrices completas.

El sistema tridiagonal puede ser expresado como:

$$a_i x_{i-1} + b_i x_{i-1} + c_i x_{i+1} = y_i \qquad i = 1, ..., n \quad a_1 = c_n = 0 \tag{1}$$

o en forma matricial:

$$\begin{pmatrix} b_1 & c_1 & 0 & ... & & & & 0 \\ a_2 & b_2 & c_2 & 0 & ... & & & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & ... & & 0 \\ & & . & . & . & & & \\ & & & . & . & . & & \\ & & & & . & . & . & \\ 0 & 0 & 0 & ... & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & ... & 0 & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_{n-1} \\ y_n \end{pmatrix} \tag{2}$$

El algoritmo de Thomas, así como la facotrización LU, implica dos etapas: factorización y resolución. La factorización se puede expresar como:

$$c_i{}' = \begin{cases} \frac{c_i}{b_i} & \text{si } i = 1 \\ \frac{c_i}{b_i - a_i c'_{i-1}} & \text{si } i = 1, 2, ..., n \end{cases} \tag{3}$$

mientras que la resolución:

$$y_i{}' = \begin{cases} \frac{y_i}{b_i} & \text{si } i = 1 \\ \frac{y_i - a_i y'_{i-1}}{b_i - a_i c'_{i-1}} & \text{si } i = 1, 2, ..., n \end{cases} \tag{4}$$

siendo $c_i{}'$ y $y_i{}'$ los coeficientes modificados en el paso i.

Luego, la solución se obtiene mediante la sustitución hacia atras:

$$x_i = \begin{cases} y_i{}' & \text{si } i = n \\ y_i{}' + c_i{}' x_{i+1} & \text{si } i = 1, 2, ..., n-1 \end{cases} \tag{5}$$

En cuanto a las características del MOHID desde el punto de vista de software, el modelo está diseñado en forma jerárquica y empleando el paradigma de orientación a objetos. Está

implementado en el lenguaje Fortran 95 e implica más de 400 mil líneas de código. Además, se pueden identificar claramente distintas capas, principalmente el MOHID está organizado en tres:

- Mohid_Base_1: Esta biblioteca es la base del modelo y reúne las funciones de entrada/salida. También incluye diversas herramientas numéricas de bajo nivel (como solvers lineales) y varios módulos que procesan datos propios de la ejecución del modelo.
- Mohid_Base_2: Esta biblioteca emplea las funcionalidades brindadas por Mohid_Base_1 para resolver diferentes problemas de mayor porte. Alguna de las principales funciones son el manejo de grillas, calcular la iteración entre el cuerpo de agua y los cuerpos externos, y calcular los procesos bioquímicos.
- MohidWater: Incluye las funciones principales para modelar y calcular simulaciones tridimensionales de cuerpos de agua.

2.1. Implementación de Thomas en el MOHID

El algoritmo de Thomas en el MOHID está implementado en el módulo ModuleFunctions, que es parte de la biblioteca Mohid_Base_1. La implementación del algoritmo implica diversas rutinas que varían levemente dependiendo de si el dominio es 2D o 3D. Las rutinas son:

- `subroutine THOMAS_2D`
- `subroutine THOMAS_3D`
- `subroutine THOMAS_3D_i0_j1`
- `subroutine THOMAS_3D_i1_j0`
- `subroutine THOMAS_Z`

El tipo de estrategias empleadas para resolver las ecuaciones diferenciales implica procesar en forma implícita distintos conjuntos de nodos de la grilla en cada sub-paso. Particularmente, la dimensión de los sistemas tridiagonales resueltos por el modelo es igual a la cantidad de nodos de la grilla en la dirección implícita. En otras palabras, si la grilla tiene IJ_{max} nodos en la dirección x y JI_{max} en la dirección y, el modelo resolverá JI_{max} sistemas, con la estructura similar a la Ecuación 2, donde n es igual a IJ_{max}, en el primer sub-paso, y IJ_{max} sistemas de JI_{max} ecuaciones en el segundo sub-paso. Teniendo en cuenta esto, el MOHID no utiliza el almacenamiento clásico (un vector por diagonal), sino que emplea una matriz completa y potencialmente rectangular para almacenar cada diagonal siguiendo la estructura de la grilla de discretización del dominio.

$$\begin{pmatrix} 0 & a_2 & \dots & a_{IJmax} & \\ a_{IJmax+1} & \dots & & & \\ & & & & a_n \end{pmatrix} \begin{pmatrix} b_1 & b_2 & \dots & b_{IJmax} & \\ b_{IJmax+1} & \dots & & & \\ & & & & b_n \end{pmatrix} \begin{pmatrix} c_1 & c_2 & \dots & c_{IJmax} & \\ a_{IJmax+1} & \dots & & & \\ & & & c_{n-1} & 0 \end{pmatrix} \tag{6}$$

Como se puede deducir, utilizar este tipo de almacenamiento simplifica enormemente la codificación. En otras palabras, disponer de las diagonales (las incógnitas y términos independientes) almacenadas como matrices bidimensionales permite parametrizar sencillamente la

dirección en la cual se está resolviendo los sistemas en forma implícita por el algoritmo ADI. Las variantes 3D del algoritmo almacenan las matrices de manera similar, es decir, se utilizan tensores 3D con la dimensión de la grilla como estructuras de almacenamiento.

2.2. Trabajos relacionados

En los últimos años se han presentado diversos trabajos centrados en estudiar la resolución de sistemas tridiagonales en GPU, en particular, y dado que el foco de este trabajo son los métodos directos de resolución, a continuación se describen los principales esfuerzos en esta materia.

Sengupta et al. (2007) presentaron una implementación eficiente de la operación *scan* sobre GPU. Una de las operaciones que utilizaron para validar su propuesta fue la resolución de sistemas lineales tridiagonales. En especial, los autores presentaron una de las implementaciones pioneras del método reducción cíclica en GPU.

Posteriormente, los trabajos se pueden dividir en dos grandes grupos, aquellos que buscan resolver eficientemente un único sistema lineal tridiagonal en GPU y por otro lado los esfuerzos tendientes a resolver varios sistemas tridiagonales al mismo tiempo explotando el poder de cómputo de las GPUs modernas.

Dentro del primer grupo se puede mencionar el artículo de Lamas et al. (2009). Los autores implementan la reducción cíclica (RC) y otros métodos variantes del mismo, por ejemplo divide & conquer RC and recursive doubling. En el trabajo se muestran importantes valores de speedup, pero estos son calculados comparando implementaciones en GPU contra su contraparte en CPU para cada método. Aunque estos métodos explotan los beneficios de las GPUs, su desempeño en CPU es superado por otros algoritmos (por ejemplo el algoritmo de Thomas), volviendo injusta la comparación. Siguiendo la misma línea, en Alfaro et al. (2011) se estudian e implementan diferentes algoritmos de la familia RC y el uso de diversas técnicas para mejorar el uso de las GPUs (acceso coalesced, uso de memoria compartida, transferencias asincrónicas, etc.).

Algunos de los principales trabajos sobre la resolución de varios sistemas al mismo tiempo se describen a continuación. En el artículo Zhang et al. (2009) se discuten diferentes estrategias para resolver en forma simultánea hasta 512 sistemas de tamaño pequeño (hasta 512 incógnitas). Entre otros, los autores evalúan los métodos RC, RC paralela, y técnicas híbridas que emplean por ejemplo reducción cíclica en las primeros iteraciones, cuando el número de ecuaciones a resolver es grande, y RC paralela (o RC paralela y recursive doubling) para las últimas. Los mejores resultados se obtienen con la versión híbrida de RC y RC paralela. Posteriormente, en Sakharnykh (2009) se resume el trabajo anterior y se incluye un estudio para alcanzar acceso a memoria coalesced.

Simultáneamente, Goddeke y Strzodka (2011) implementaron una versión eficiente de RC y estudiaron el uso de técnicas de precisión mixta para alcanzar resultados con precisión equivalente a doble pero con menos costo computacional.

3. PROPUESTA

La propuesta está basada en utilizar el poder de cómputo de las GPUs para acelerar la resolución de los sistemas tridiagonales presentes en el MOHID. En este sentido, y como se mostró en el Apartado 2.2, existen diferentes estrategias para alcanzar una implementación paralelizable del solver de sistemas tridiagonales. En nuestro caso, si bien se evaluaron las estrategias antes descritas, en forma preliminar, resultados (no formalizados) mostraron que utilizar estos métodos no arrojan tan buenos tiempos de ejecución como al utilizar un solver construido úni-

camente sobre el método de Thomas. Notar que este tipo de modelos basados en estrategias ADI para resolver modelos en 3D implican la resolución de gran cantidad de sistemas tridiagonales independientes, y esto se puede realizar en forma paralela. Por ejemplo, se necesita resolver $N_x \times N_y$ sistemas de dimensión N_z al trabajar en un dominio tridimensional de dimensión $N_x \times N_y \times N_z$. Entonces, al simular dominios 3D de dimensión considerable la resolución de los sistemas tridiagonales ofrece un nivel de paralelismo interesante, aún para el uso eficiente de GPUs.

Teniendo en cuenta lo expresado anteriormente, se implementaron diferentes versiones de solvers tridiagonales y se profundizó sobre la versión basada en ejecutar varias instancias del método de Thomas en forma paralela.

3.1. Solver para sistemas tridiagonales en paralelo

En primera instancia se implementó una versión paralela sobre CPU tendiente a explotar el poder de cómputo de los procesadores multicore. La versión es equivalente a la original ($\texttt{SolvTri}_{CPU}$) pero utilizando OpenMP para introducir paralelismo, $\texttt{SolvTri}_{CPUpar}$. En esta versión se paraleliza la resolución de los diferentes sistemas, pudiendo resolver en paralelo tantos sistemas como la multiplicación de las dos dimensiones de la grilla empleada (en las que se está resolviendo explícitamente las ecuaciones). En particular, y dada la naturaleza balanceada del problema, se emplean tantos hilos de ejecución como unidades de proceso se dispone.

Basados en la variante paralela sobre CPU se implementó una variante del solver para sistemas tridiagonales en GPU denominada $\texttt{SolvTri}_{GPU}$. Como primer paso de la variante se envían los datos, las matrices y vectores independientes, desde la memoria de la CPU a GPU. Luego se resuelven en GPU todos los sistemas y finalmente, los resultados son devueltos de memoria de GPU a CPU.

La estrategia más adecuada en estos casos es resolver un sistema por CUDA thread. Cada thread resuelve el sistema utilizando el algoritmo de Thomas y no hay dependencia de datos entre distintos threads. Los threads se organizan en una grilla de bloques bidimensional. Se eligió un tamaño de bloque adecuado teniendo en cuenta cada arquitectura y la dimensión de la grilla de bloques es el resultado de dividir cada una de las direcciones del ADI entre el tamaño de bloque.

3.2. Integración del solver al MOHID

Además de la versión original del MOHID ($\texttt{MOHID}_{CPU}$) y la versión con el solver que incluye paralelismo mediante OpenMP ($\texttt{MOHID}_{CPUpar}$) se implementaron dos versiones que integran el solver de los sistemas tridiagonales en GPU, $\texttt{SolvTri}_{GPU}$. Las mismas se describen a continuación.

La primera versión del modelo MOHID en GPU, $\texttt{MOHID}_{GPU}$, es una solución trivial. Es decir, el modelo ejecuta en forma similar a la versión original (en CPU) y para cada resolución de sistemas tridiagonales se invoca a la rutina $\texttt{SolvTri}_{GPU}$. Como se mencionó anteriormente esto implica, enviar los datos de memoria de CPU a GPU, resolver los sistemas en GPU y retornar los resultados, realizando todas estas tareas en forma secuencial y consecutivas a los restantes pasos del modelo MOHID.

Generalmente, el uso de las GPUs para acelerar etapas de modelos se ve condicionado por los tiempos de transferencia entre CPU y GPU. Teniendo en cuenta esto, y dada que la estrategia de cómputo del MOHID computa secuencialmente los datos necesarios para formar las matrices de los sistemas lineales, se implementó una segunda versión de la integración del solver

que busca solapar parte de los envíos desde la CPU a la GPU con cómputos del MOHID en CPU. En particular se hace uso de las transferencias asincrónicas (sobre memoria no paginable) para enviar dos matrices (que contienen las diagonales) mientras se computa la tercera. De esta manera se podrían ocultar parte de los tiempos de transferencia. Esta versión se denomina $\texttt{MOHID}_{GPUsola}$.

4. EXPERIMENTACIÓN

En esta sección se presentan los experimentos realizados para evaluar el desempeño computacional de las variantes implementadas.

4.1. Plataforma de experimentación

La evaluación experimental se desarrolló sobre dos plataformas de hardware, cada una compuesta por una GPU conectada a un procesador multi-core. La Tabla 1 presenta los detalles de las plataformas utilizadas.

Plataforma	CPU	Cores	GPU	GPU Cores
I	Intel Xeon E5530 @ 2.27GHz 48GB	4	C1060	240
II	Intel i7-2600 @ 3.40GHz 16GB	4	C2070	444

Tabla 1: Plataforma utilizada para la experimentación.

4.2. Casos de prueba

Para evaluar y validar las propuestas se creó un caso de estudio sencillo, definido por un canal de 7000 m $\times$ 7000 m y 20 m de profundidad, con tres laterales cerrados y con un forzante en el restante lado de 7000 m^3/s. El nivel en el lado abierto se define como 0 y se incluye salinidad y temperatura en el modelado del cuerpo de agua.

Sobre el caso se crearon tres grillas diferentes para la discretización, de forma de poder evaluar el desempeño computacional en diferentes contextos. Las grillas están definidas por discretizaciones de 70 $\times$ 70 $\times$ 28, 140 $\times$ 140 $\times$ 28 y 140 $\times$ 140 $\times$ 56, definiendo los casos $\texttt{flow\_channel}_{70-28}$, $\texttt{flow\_channel}_{140-28}$ y $\texttt{flow\_channel}_{140-56}$, respectivamente.

4.3. Evaluación experimental

En una primera etapa se evaluaron los diferentes solvers en forma independiente. En este sentido, las tablas 2 y 3 presentan los tiempos de ejecución para la resolución de los sistemas tridiagonales empleando las distintas variantes implementadas, secuencial en CPU ($\texttt{SolvTri}_{CPU}$), paralela en CPU ($\texttt{SolvTri}_{CPUpar}$) y paralela en GPU ($\texttt{SolvTri}_{GPU}$), en ambas plataformas. Además, en las tablas mencionadas se ofrecen las aceleraciones alcanzadas por las versiones paralelas.

De los resultados expresados en las tablas anteriores se puede observar que las versiones paralelas superan ampliamente a la versión secuencial del solver tridiagonal. Además, en ambas plataformas la variante de GPU obtiene aceleraciones inferiores a las conseguidas por la versión

Grilla	$\texttt{SolvTri}_{CPU}$	$\texttt{SolvTri}_{CPUpar}$	Aceleración	$\texttt{SolvTri}_{GPU}$	Aceleración
$\texttt{flow\_channel}_{70-28}$	4.6	1.2	3.8	1.6	2.9
$\texttt{flow\_channel}_{140-28}$	17.2	4.8	3.6	5.2	3.3
$\texttt{flow\_channel}_{140-56}$	39.3	9.9	4.0	10.3	3.9

Tabla 2: Tiempo de ejecución (en milisegundos) de las implementaciones en la Plataforma I.

Grilla	$\texttt{SolvTri}_{CPU}$	$\texttt{SolvTri}_{CPUpar}$	Aceleración	$\texttt{SolvTri}_{GPU}$	Aceleración
$\texttt{flow\_channel}_{70-28}$	2.3	0.7	3.3	1.1	2.1
$\texttt{flow\_channel}_{140-28}$	9,2	2,5	3.7	3.8	2.4
$\texttt{flow\_channel}_{140-56}$	19,8	5.4	3.7	7.6	2.6

Tabla 3: Tiempo de ejecución (en milisegundos) de las implementaciones en la Plataforma II.

$\texttt{SolvTri}_{CPUpar}$, sin embargo, la versión $\texttt{SolvTri}_{GPU}$ presenta mayor nivel de escalabilidad que la $\texttt{SolvTri}_{CPUpar}$.

Dado que una porción importante de los tiempos de ejecución de la variante $\texttt{SolvTri}_{GPU}$ se deben a comunicaciones, el siguiente estudio busca establecer que proporción del tiempo de ejecución de la rutina propuesta es dedicada a cálculos y cual a transferencias. En este sentido las tablas 4 y 5 desagregan ambos tiempos para cada caso de prueba en la Plataforma I y Plataforma II respectivamente.

Grilla	T. comunic.	T. cálculos	T. totales
$\texttt{flow\_channel}_{70-28}$	1.2	0.4	1.6
$\texttt{flow\_channel}_{140-28}$	4.3	0.9	5.2
$\texttt{flow\_channel}_{140-56}$	8.6	1.7	10.3

Tabla 4: Tiempo de ejecución (en milisegundos) de cada etapa de la rutina $\texttt{SolvTri}_{GPU}$ en la Plataforma I.

Con los datos plasmados en las tablas 4 y 5 se pueden observar tres puntos destacados. Primero, la importancia en cuanto a tiempo de ejecución de las transferencias de datos desde la memoria de la CPU a la de GPU y viceversa. Los tiempos implican entre un 60 % y un 85 % del total del tiempo del solver. Segundo, el tiempo de ejecución del solver en GPU (sin tener en cuenta transferencias) es aproximadamente 5 veces menor que la versión paralela en CPU y más de 17 veces superior en eficiencia que la versión secuencial. Por último, en ambas tablas se puede notar que la incidencia de los tiempos de transferencia entre CPU y GPU aumentan con la dimensión del problema. Este aspecto permite vislumbrar una importante limitante al aumentar la dimensión de los problemas.

Grilla	T. comunic.	T. cálculos	T. totales
$\texttt{flow\_channel}_{70-28}$	0.9	0.2	1.1
$\texttt{flow\_channel}_{140-28}$	3.3	0.5	3.8
$\texttt{flow\_channel}_{140-56}$	6.6	1.0	7.6

Tabla 5: Tiempo de ejecución (en milisegundos) de cada etapa de la rutina $\texttt{SolvTri}_{GPU}$ en la Plataforma II.

Como conclusión de esta etapa de experimentación se puede destacar que el tipo de estrategias empleadas son útiles en GPU cuando se dispone de una gran cantidad de sistemas de baja dimensión, en otros contextos (baja cantidad de sistemas o sistemas de gran porte) es necesario aplicar otro tipo de métodos, por ejemplo utilizar algoritmos basados en reducción cíclica. Además, se puede mencionar que las versiones paralelas ofrecen importantes ganancias pero el uso de GPU apenas consigue resultados casi similares al uso de procesadores multicore. Sin embargo, el avance en los dispositivos gráficos y la posibilidad de ocultar los tiempos de transferencia permiten conjeturar sobre resultados positivos.

En una segunda etapa se evaluó el impacto del uso de los diferentes solvers en la ejecución del modelo MOHID por completo. Este estudio se realizó únicamente en la Plataforma I. En este sentido, en la Tabla 6 se presentan los tiempos de ejecución del modelo completo y el solver para las cuatro versiones.

	T. solver	Aceleración	T. total	Aceleración
$\texttt{MOHID}_{CPU}$	180.1	-	1095,5	-
$\texttt{MOHID}_{CPUpar}$	50.6	3.6	966.0	13 %
$\texttt{MOHID}_{GPU}$	80.0	2.3	995.4	10 %
$\texttt{MOHID}_{GPUsola}$	62.5	2.9	939.6	17 %

Tabla 6: Tiempo de ejecución (en segundos) de las diferentes versiones del MOHID en la Plataforma I.

Los resultados obtenidos muestran la importante aceleración de los tiempos de ejecución de los solvers paralelos con respecto al original. Sin embargo, dada la incidencia poco importante del tiempo de los solvers en el tiempo total de ejecución del modelo, la propuesta impacta de forma marginal en el modelo por completo, siendo necesario migrar a GPU otras etapas del modelo para alcanzar mejoras realmente significativas.

Un aspecto a destacar de esta evaluación es la importancia de superponer cálculo en CPU y transferencia de datos (o cálculos en GPU). Esto permitió a la versión $\texttt{MOHID}_{GPUsola}$ superar a la versión paralela en CPU del solver.

5. CONCLUSIONES Y TRABAJO FUTURO

En el trabajo se realizó un breve estudio de la implementación de solvers de sistemas tridiagonales en GPU. En especial, el foco se centró sobre los métodos para resolver varios sistemas al mismo tiempo.

Se implementó y evaluó una versión del método paralelo sobre CPU y otra sobre GPU. Los resultados alcanzados mostraron importantes mejoras en los tiempos de ejecución al utilizar las versiones paralelas. Al comparar las versiones paralelas, en CPU y GPU, se puede concluir que las versiones en GPU están por debajo de las versiones multi-core. Sin embargo, la versión en GPU muestra mejor escalabilidad.

Como parte del trabajo futuro se proponen las siguientes líneas:

- Utilización de múltiples GPUs: El tipo de estrategia utilizada en este trabajo permite fácilmente extender la propuesta para explotar el uso de múltiples GPUs. Esta opción es únicamente interesante en el caso de abordar problemas de dimensión realmente importante.

- Trabajo híbrido y concurrente CPU+GPU: Si bien en el trabajo se busca solapar cálculos en CPU y transferencias, resulta interesante ahondar en el uso de estrategias que permitan solapar cálculos en ambos dispositivos.

- Utilización de precisión mixta: En la propuesta se utilizó doble precisión. Si bien las GPUs actuales alcanzan un desempeño aceptable en doble precisión es interesante evaluar el uso de precisión mixta ya que transferir los datos en simple precisión mejoraría los tiempos dedicados a comunicaciones, quizá sin comprometer de forma significativa la precisión de los resultados del modelo.

- Migrar otras etapas del MOHID a GPU: El uso de la GPU para acelerar la resolución de los sistemas tridiagonales mostró resultados promisorios, pero comparables con el uso de técnicas basados en multi-core. Utilizar la GPU para acelerar otras etapas del modelo permitiría avanzar en dos aspectos. Por un lado, buscar acelerar otras etapas del modelo utilizando una GPU y así aumentar el porcentaje del modelo cubierto con las técnicas de HPC. Por otro lado, intentar disminuir las transferencias de datos, por ejemplo, si las matrices se construyen en GPU ya no es necesario realizar la transferencia de las mismas.

REFERENCIAS

Alfaro P., Igounet P., y Ezzatti P. A study on the implementation of tridiagonal systems solvers using a GPU. En *Proceedings of the XXX International Conference of the Chileans Computer Science Society (SCCC'2011)*. IEEE, 2011.

Fossati M. y Piedra-Cueva I. A 3d hydrodynamic numerical model of the Río de la Plata and Montevideo‘s coastal zone. *Appl. Math. Modell.*, on line, 2012.

Goddeke D. y Strzodka R. Cyclic reduction tridiagonal solvers on gpus applied to mixed precision multigrid. *IEEE Transactions on Parallel and Distributed Systems*, 22(1):22–32, 2011.

Golub G. y Van Loan C. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

Lamas J., Boo M., Heras D., y Arguello F. *Proyección de Algoritmos de Resolución de Sistemas Tridiagonales en la Tarjeta Gráfica*. XX Jornadas de Paralelismo, La Coruña, España, p.349-354, 2009.

Neves F.R., Martins, Leitao P., y Silva A. 3-d modeling in the sado estuary using a new generic coordiante approach. *J. Geophys. Res.*, 1999.

Sakharnykh N. *Tridiagonal Solvers on the GPU and Applications to Fluid Simulation*. GPU Technology Conference, 2009.

Sengupta S., Harris M., Zhang Y., y Owens J.D. Scan primitives for gpu computing. *22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware, San Diego, California*, 2007.

Trancoso A.R., Braunschweig F., Chambel Leitão P., Obermann M., y Neves R. An advanced modelling tool for simulating complex river systems. *Science of The Total Environment*, 407(8):3004–3016, 2009.

Zhang Y., Owens J.D., y Cohen J. Fast tridiagonal solvers on the gpu. *GPU Technology Conference*, 2009.