Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXI, págs. 3141-3151 (artículo completo)
Alberto Cardona, Paul H. Kohan, Ricardo D. Quinteros, Mario A. Storti (Eds.)
Salta, Argentina, 13-16 Noviembre 2012

# RESOLUCIÓN DE SISTEMAS HEPTA-DIAGONALES EN GPU

**Pablo Igounet y Pablo Ezzatti**

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

**Palabras Clave:** sistemas hepta-diagonales, SIP, GPU.

**Resumen.** Este artículo presenta la implementación del método Strongly Implicit Procedure (SIP) para la resolución de sistemas lineales hepta-diagonales en Graphics Processing Unit (GPU). En particular, se realizaron tres implementaciones distintas del método SIP que buscan aprovechar diferentes características de las GPUs (accesos eficientes a memoria, uso de memoria compartida, transferencias de datos optimizadas, etc.). Los resultados obtenidos trabajando en simple y doble precisión muestran una importante mejora en el tiempo de ejecución alcanzado, con aceleraciones de hasta 22×, comparado con la versión paralela del método en CPU. Además, se incluyó el solver desarrollado en un modelo numérico de mediano porte, el `caffa3d.MB`, obteniendo una reducción de hasta 19 % en el tiempo total de ejecución del modelo.

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

## 1. INTRODUCCIÓN

La resolución de sistemas lineales está presente en diferentes aplicaciones de computación científica. En particular, los sistemas lineales hepta-diagonales aparecen en diversos modelos numéricos, por ejemplo, en la discretización tradicional para resolver ecuaciones diferenciales en 3-D sobre grillas regulares resueltas con métodos implícitos. Un método ampliamente difundido para la resolución de este tipo de sistemas lineales en el campo de la dinámica de fluidos computacional (CFD) es el Strongly Implicit Procedure (SIP) (Stone, 1968). El SIP es un método iterativo para la resolución de sistemas lineales de banda ($Ax = b$, donde $A$ es una matriz penta- o hepta-diagonal), perteneciente a la familia de los métodos de factorización incompleta. El SIP toma partido de la forma subyacente de las ecuaciones diferenciales para resolver el sistema lineal, en el caso de este trabajo sistemas hepta-diagonal. En el primer paso el SIP calcula la factorización $LU$ incompleta ($\hat{L}\hat{U} = incLU(A)$), donde $\hat{L}$ y $\hat{U}$ son buenas aproximaciones de las matrices $L$ y $U$ pero sin incurrir en *fill-in*. Posteriormente, se itera refinando la solución inicial calculada hasta que el residuo generado por ésta sea lo suficientemente pequeño.

En general, los modelos numéricos presentes en el campo de CFD tienen como una de las etapas más costosas la resolución de los sistemas lineales. Esta situación motiva el uso de técnicas de computación de alto desempeño (HPC) para acelerar dicho cómputo. Adicionalmente, en los problemas de CFD usualmente se requiere la resolución de diversos sistemas lineales para la simulación de escenarios realistas. Por esta razón, aún pequeñas mejoras en los tiempos de ejecución en la resolución de un sistema lineal, puede mejorar en forma significativa el tiempo total de ejecución de un modelo numérico.

Una de las restricciones para el uso de las técnicas tradicionales de HPC son los importantes costos asociados a su uso. Sin embargo, en los últimos años ha crecido el uso de hardware de bajo costo, como los procesadores multicore y procesadores gráficos (GPUs), habilitando así el uso de técnicas de HPC en computadoras de escritorio y a un precio razonable.

En este trabajo se presentan tres implementaciones del método SIP sobre GPU, las versiones están basadas en las diferentes recorridas de las matrices propuestas por Deserno et al. (2002) para paralelizar el método y buscan aprovechar diferentes características de las GPUs (accesos eficientes, memoria compartida, transferencias de datos optimizadas, etc.). Los resultados experimentales obtenidos, sobre una plataforma que incluye una CPU multi-core y una GPU capaz de ejecutar CUDA, alcanzan valores de aceleración de hasta 22$\times$ para la resolución de los sistemas lineales. Además, en el trabajo se evalúa el impacto del desempeño computacional de un modelo numérico, el `caffa3d.MB` (Usera et al., 2008) para la simulación fluidos viscosos y/o turbulentos en 3D al utilizar el solver propuesto, alcanzando reducciones del tiempo de ejecución de hasta un 19 % del tiempo total del modelo.

El documento se estructura de la siguiente forma. La Sección 2 describe someramente el método SIP y se releva el trabajo relacionado. Luego, en la Sección 3 se describen las propuestas desarrolladas. La evaluación experimental, tanto de los solvers en forma independiente como de la inclusión de los solvers al modelo numérico `caffa3d.MB` se puede encontrar en la Sección 4. Por último, en la Sección 5 se resumen las conclusiones arribadas durante el trabajo y las posibles líneas de trabajo futuro.

## 2. EL MÉTODO SIP

El SIP (Stone, 1968) es un método iterativo de resolución de sistemas lineales de banda ($Ax = b$, donde $A$ es una matriz hepta- o penta-diagonal, relacionada con la discretización de ecuaciones diferenciales). Inicialmente, el método SIP computa una factorización LU incom-

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

pleta ($\hat{L}\hat{U} = incLU(A)$) guiado por un escalar $\alpha$ (un parámetro para mejorar la aproximación calculada), donde $\hat{L}$ y $\hat{U}$ son buenas aproximaciones de las matrices $L$ y $U$ pero sin incurrir en fill-in. El valor típicamente utilizado para $\alpha$ es 1.8 (más detalles se pueden encontrar en Ferziger y Perić (2002)).

Luego, se ejecuta un procedimiento iterativo para refinar la solución inicial hasta que el residuo sea lo suficientemente pequeño. El Algoritmo `SIP` resume el método.

**Algoritmo `SIP`:**
(1) Hacer factorización LU incompleta : $\hat{L}\hat{U} \approx A$
(2) Calcular residuo: $r_n = b - Ax_n$
(3) Calcular vector $R_n$ (eliminación hacia adelante):
$$R_n = \hat{L}^{-1} r_n$$
(4) Calcular $\delta x$ (sustitución hacia atrás):
$$\hat{U}\delta x = R_n$$
(5) Volver a (2), hasta que el residuo sea lo suficientemente chico.

En el primer paso se resuelve la factorización incompleta ($\hat{L}\hat{U} \approx A$). En particular, el SIP para matrices hepta-diagonal se basa en resolver el sistema lineal correspondiente a la discretización de las ecuaciones diferenciales de un dominio 3-D de dimensión $N_i \times N_j \times N_k$ como se muestra en la Figura 1. La molécula de puntos se nombra según la nomenclatura geográfica propuesta por Ferziger y Perić (2002), $A_w$ (west), $A_s$ (south), $A_p$ (point considered), $A_n$ (north), $A_e$ (east), $A_t$ (top) y $A_b$ (bottom).

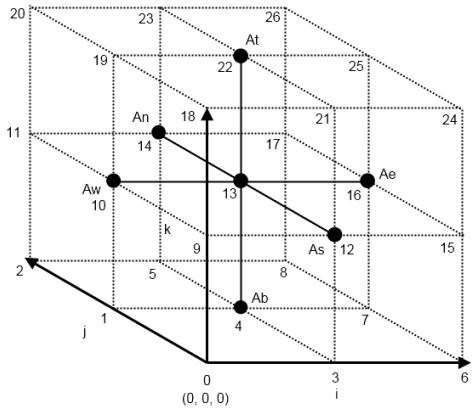


Figura 1: Ejemplo de grilla 3D, mostrando la molécula de cálculo con 7 nodos.

Usando las matrices $\hat{L}$ y $\hat{U}$, el SIP itera en los pasos (2) a (4) hasta alcanzar un residuo los suficientemente pequeño. Luego, el vector $R_n$ puede ser computado fácilmente:

$$R^l = (r^l - \hat{L}^l_S R^{l-1} - \hat{L}^l_W R^{l-N_j})/\hat{L}^l_P$$

Esta ecuación puede ser resuelta utilizando un orden creciente en $l$. Luego, de que $R_n$ es calculado, $\delta x$ se computa considerando un orden decreciente en el índice $l$. $\delta x$ se calcula según la siguiente formula:

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

$$\delta x^l = R^l - \hat{U}_N^l \delta x^{l+1} - \hat{U}_E^l \delta x^{l+N_j}$$

### 2.1. Trabajo relacionado

En los últimos años varios trabajos han demostrado los beneficios de utilizar GPUs para acelerar el cómputo de problemas de propósito general y en particular operaciones de álgebra lineal. En este sentido, algunos trabajos destacables son: para kernels de álgebra lineal densa (Barrachina et al., 2008), inversión de matrices (Ezzatti et al., 2011), solver de sistemas lineales (Tomov et al., 2010; Li y Saad, 2010; Naumov, 2011) y ecuaciones matriciales (Benner et al., 2009). Sin embargo, no hemos encontrado propuestas que discutan la implementaciones del método SIP para matrices hepta-diagonales en plataformas basadas en many-cores (GPUs).

Sobre arquitecturas tradicionales de HPC sí se han presentado diversos trabajos tendientes a paralelizar el método SIP (Halada y Lucká, 1999; Reeve et al., 2001). Especialmente destaca el trabajo de Deserno et al. (2002), que estudia la paralelización del SIP sobre arquitecturas de memoria compartida reconociendo diferentes órdenes para procesar los nodos de la grilla y así quebrar las dependencias de datos. A continuación se describen los diferentes órdenes.

#### Orden secuencial

Esta variante procesa los coeficientes en forma secuencial, según el orden de numeración de los nodos en la grilla.

#### Orden por líneas

Considerando las dependencias de dato del SIP se puede establecer que:

- El cálculo del residuo no presenta restricciones.
- La eliminación hacia adelante, solo presenta dependencia entre elementos del vector $r$. En particular, para computar el nodo $l$, son necesarios los valores de los nodos $l-1$, $l-N_i$ y $l - N_i \times N_j$ son necesarios.
- La sustitución hacia atrás, solo presenta dependencias entre elementos del vector $r$. En particular, para computar el nodo $l$, los valores de los nodos $l + 1$, $l + N_i$ y $l + N_i \times N_j$ son necesarios.

Teniendo en cuenta las restricciones expresadas anteriormente, las líneas formadas por los nodos que cumplen que $j + k$ es constante pueden ser computados (tanto la eliminación hacia adelante como hacia atrás) en forma paralela. En cada línea, los nodos pueden ser computados en forma ascendente en $i$ para la eliminación hacia adelante y en orden descendente para la sustitución hacia atrás.

#### Orden por planos

Adicionalmente al paralelismo conseguido al realizar los cálculos por líneas, se puede alcanzar un mayor nivel de paralelismo si los nodos se agrupan por planos donde $i + j + k$ es constante.

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

## 3. PROPUESTA

La propuesta consiste en la implementación de diversas versiones del método SIP para el caso hepta-diagonal utilizando ambas arquitecturas, CPU y GPU, siguiendo los ordenes antes descritos. En particular, se implementaron cuatro versiones del método SIP, una versión para CPU y tres para GPU. Todas las versiones paralelas se basan en el orden por planos, ya que es el que permite mayor nivel de paralelismo. Además, experimentos preliminares (no formalizados) mostraron que este tipo de estrategias se comportaban mejor que el orden por líneas en GPU (y obviamente mejor que el orden secuencial).

Cada rutina recibe las siete diagonales ($A_w$, $A_s$, $A_p$, $A_n$, $A_e$, $A_b$ y $A_t$), el parámetro $\alpha$, el término independiente ($b$), y el máximo número de iteraciones de refinamiento. Utilizando la información recibida resuelve el sistema lineal y retorna el vector solución ($x$), el número de iteraciones realmente realizadas y el residuo final ($R_n$). Las rutinas utilizan la norma 1 para el cálculo del residuo.

Deducir que nodos pertenecen a cada plano y cual nodo tiene que ser procesado por cada hilo es una tarea que implica una lógica intrincada. Teniendo en cuenta esto, y considerando que no es una tarea pesada desde el punto de vista de cómputo esto se realiza en forma centralizada en CPU antes de comenzar la resolución propiamente dicha. La estructura de datos computada también es transferida de CPU a GPU.

Las diferencias entre las rutinas se discuten a continuación.

### 3.1. SIP 7 en CPU por planos ($\texttt{SIP7}_{\texttt{CPU}}\texttt{-HP}$)

Esta variante procesa los nodos de la grilla en CPU, agrupando los nodos por planos y explotando el paralelismo ofrecido por plataformas multi-core. $\texttt{SIP7}_{\texttt{CPU}}\texttt{-HP}$ está implementada en C usando OpenMP.

### 3.2. SIP 7 en GPU por planos ($\texttt{SIP7}_{\texttt{GPU}}\texttt{-HP}$)

Esta variante procesa los nodos en GPU agrupados por planos. Las principales etapas de la variante son:

1. Primero, los parámetros de entrada son copiados desde la memoria de la CPU a la memoria de la GPU.
2. Luego, se ejecuta en GPU el solver.
3. Finalmente, los parámetros de salida son copiados desde la GPU a la CPU.

En la ejecución del solver en GPU, cada hilo procesa un nodo de la grilla (que se corresponde con una ecuación en la grilla). Como se explicó anteriormente, determinar que nodo tiene que procesar cada hilo en cada plano no es una tarea trivial, por lo tanto como primera etapa se almacena en GPU una estructura auxiliar de datos para determinar esto. En cada iteración se tiene como entrada el offset del plano en el vector de ordenación, y cada hilo accede a la estructura para determinar que nodo procesar, en la Figura 2 se presenta en forma gráfica la estructura.

### 3.3. SIP en GPU por planos usando acceso coalesced ($\texttt{SIP7}_{\texttt{GPU}}\texttt{-HPC}$)

Esta variante extiende $\texttt{SIP7}_{\texttt{GPU}}\texttt{-HP}$ buscando obtener acceso coalesced. La eliminación hacia adelante y la sustitución hacia atrás pueden ser reordenadas para obtener un acceso coalesced, sin embargo, para alcanzar dicho objetivo es necesario reordenar la matriz $A$ y este

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

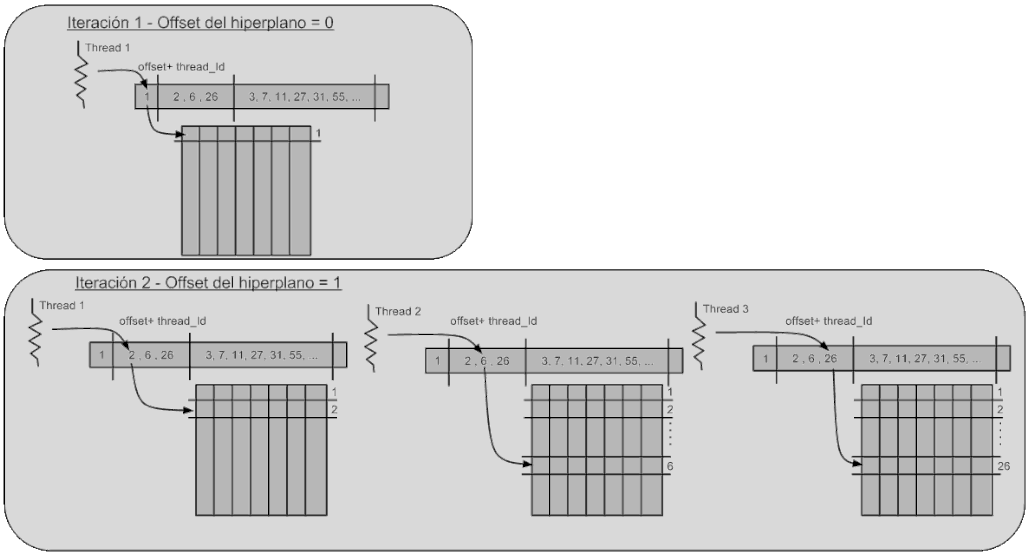


Figura 2: Estructura de datos que expresa el orden de ejecución.

paso incrementa notoriamente el costo computacional por lo tanto únicamente se aplica acceso coalesced para el cómputo del residuo y la norma.

### 3.4. SIP en GPU por planos unificando kernels (SIP7$_{\text{GPU}}$-HPCuK)

Las versiones en GPU del SIP que fueron presentadas anteriormente implican el uso de un kernel por cada plano a procesar. Esta versión se diseño para unificar el cálculo de diversos planos (en forma secuencial por plano) en un mismo kernel, buscando evaluar si el uso de divesos kernels introduce o no un overhead significativo.

## 4. EVALUACIÓN

En esta sección se presenta la evaluación experimental de las diferentes implementaciones del SIP desarrolladas. Además de evaluar los solvers en forma independiente, se describe el proceso para integrar las rutinas propuestas a un modelo numérico de mediano porte, el `caffa3d.MB`, y se estudian los tiempos de ejecución y aceleración de la nueva versión del modelo.

### 4.1. Plataforma experimental

La evaluación experimental se desarrolló en una plataforma compuesta por una GPU conectada a un procesador multi-core. La Tabla 1 presenta los detalles de las plataforma utilizada.

| Plataforma | CPU | Cores | GPU | GPU Cores |
|---|---|---|---|---|
| I | Intel E7400 @ 2.8 GHz 4GB | 2 | GTX 480 | 480 |

Tabla 1: Plataforma utilizada para la experimentación

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

### 4.2. Casos de prueba

Los sistemas lineales utilizados para evaluar las implementaciones son construidos a partir de un modelo sencillo de CFD. La aplicación considerada es la simulación de una tapa impulsada por el flujo dentro de una cavidad cúbica con $Re$=1000. Se aplican condiciones de borde sin desplazamiento a todas las paredes, y velocidad constante $V_o$ a una. Este es un caso tradicional para evaluar la resolución de las ecuaciones de Navier-Stokes, con soluciones numéricas disponibles en la bibliografía (Albensoeder y Kuhlmann, 2005; Iwatsu et al., 1990) para validar los resultados. La Figura 3 presenta en forma gráfica el caso.

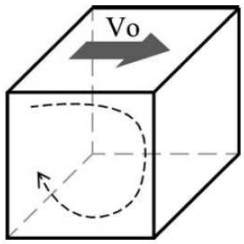


Figura 3: Diagrama del problema de flujo en una cavidad cúbica. $Re = Vo \cdot h/\nu = 1000$.

Se evaluaron las implementaciones utilizando casos que difieren en el tamaños de grillas, uno para una grilla de $64 \times 64 \times 64$ y otro para una grilla de $128 \times 128 \times 128$ denominados $\text{CAVS}_{64}$ and $\text{CAVS}_{128}$ respectivamente.

### 4.3. Evaluación experimental

Se realizaron experimentos utilizando tanto aritmética de simple como de doble precisión. En todos los casos, los tiempos de transferencia de memoria de CPU a GPU son considerados dentro del tiempo de ejecución del algoritmo. Para el caso de la integración en el modelo `caffa3d.MB` se utilizó la versión del SIP en precisión simple ya que es la precisión que el mismo utiliza para su versión original en CPU.

Las tablas 2 y 3 presentan los tiempos de ejecución (en segundos) para las implementaciones en CPU y GPU del solver SIP para sistemas lineales hepta-diagonales. Adicionalmente, la tabla incluye el valor de aceleración alcanzado sobre la ejecución en CPU.

| Grilla | $\text{SIP7}_{\text{CPU}}$-HP | $\text{SIP7}_{\text{GPU}}$-HP | $\text{SIP7}_{\text{GPU}}$-HPC | $\text{SIP7}_{\text{GPU}}$-HPCuK | Aceleración |
|---|---|---|---|---|---|
| $\text{CAVS}_{64}$ | 0.287 | 0.076 | 0.055 | 0.193 | 5.22 |
| $\text{CAVS}_{128}$ | 7.940 | 0.590 | 0.359 | 0.591 | 22.12 |

Tabla 2: Tiempo de ejecución de las implementaciones de SIP 7 en simple precisión.

De los resultados obtenidos (tanto en precisión simple como doble) se observa que las versiones en GPU alcanzan un mejor desempeño, hasta 22$\times$ en el mejor caso, que las implementaciones en CPU. Además, los resultados muestran escalar con la dimensión de los problemas, permitiendo así resolver problemas de mayor dimensiones en tiempos de ejecución sensiblemente menores. Por otra parte el método $\text{SIP7}_{\text{GPU}}$-HPCuK no presenta buenos resultados

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

| Grilla | SIP7$_{\text{CPU}}$-HP | SIP7$_{\text{GPU}}$-HP | SIP7$_{\text{GPU}}$-HPC | SIP7$_{\text{GPU}}$-HPCuK | Aceleración |
|---|---|---|---|---|---|
| CAVS$_{64}$ | 0.440 | 0.086 | 0.064 | 0.238 | 6.88 |
| CAVS$_{128}$ | 9.150 | 0.650 | 0.418 | 0.682 | 21.89 |

Tabla 3: Tiempo de ejecución de las implementaciones de SIP 7 en doble precisión.

respecto al método sin unificar los kernels, esto se debe posiblemente a que el nivel de paralelismo que se pierde tiene mayor impacto en la performance que el overhead de la ejecución de múltiples kernels.

De forma de entender mejor donde están las zonas críticas del código de las versiones implementadas en GPU se computaron y analizaron los tiempos de ejecución de las distintas etapas lógicas de la versión SIP7$_{\text{GPU}}$-HPC del SIP (la versión que obtuvo los mejores resultados). Las tablas 4 y 5 presentan los tiempos de ejecución (en milisegundos) para la versión SIP7$_{\text{GPU}}$-HPC discriminada por tiempo de transferencia, tiempo para factorización y tiempo de resolución para precisión simple y doble respectivamente (el tiempo de resolución en las tablas es para 10 iteraciones). También, se presenta el tiempo total de ejecución que se completa con el tiempo para procesar los datos de las recorridas. Los resultados obtenidos muestran que el tiempo de transferencia, factorización y cada paso iterativo de resolución son similares para el caso de precisión simple. Y que para el caso de precisión doble los tiempos de transferencia crecen en mayor medida que los tiempos de cálculo tanto en la factorización como en cada iteración de la resolución.

| Grilla | Transf. | Factorización | Resolución | Total |
|---|---|---|---|---|
| CAVS$_{128}$ | 41 | 37 | 264 | 359 |

Tabla 4: Tiempo de ejecución de cada etapa lógica de SIP7$_{\text{GPU}}$-HPC en precisión simple.

| Grilla | Transferencia | Factorización | Resolución | Total |
|---|---|---|---|---|
| CAVS$_{128}$ | 73 | 39 | 289 | 418 |

Tabla 5: Tiempo de ejecución de cada etapa lógica de SIP7$_{\text{GPU}}$-HPC en doble precisión.

### 4.4. Integración del GPU-SIP en el modelo caffa3d.MB

Además de la evaluación en forma independiente del SIP en GPU, el solver SIP7$_{\text{GPU}}$-HPC se incluyo al modelo numérico caffa3d.MB de forma de evaluar el impacto de las mejoras

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

en un proceso completo. El modelo numérico se encuentra implementado en Fortran95 por lo que se implementó un wrapper para invocar las nuevas rutinas en C utilizando CUDA desde el modelo `caffa3d.MB`.

Adicionalmente al uso del wrapper, se introdujeron otras pequeñas modificaciones al solver de GPU, las mismas se describen a continuación:

- La estructura de datos con el orden de ejecución de los vectores es copiado una única vez a la GPU, al comienzo de la ejecución del modelo.
- Dado que el modelo `caffa3d.MB` resuelve tres sistemas diferentes con la misma matriz $A$, la matriz $A$ es copiada de CPU a GPU una única vez cada estos tres sistemas a resolver.
- Por la misma razón, la factorización $LU$ es computada en GPU únicamente una vez por cada tres sistemas.

### 4.4.1. Experimentos

Esta sección presenta los resultados obtenido en la evaluación expe-rimental de la versión del modelo `caffa3d.MB` basada en GPU utilizando precisión simple. Desde el punto de vista numérico, ambas versiones (computando en CPU y en GPU) obtuvieron resultados que no mostraron diferencias significativas.

En la Tabla 6 se presentan los tiempos de ejecución del modelo `caffa3d.MB` para las casos $\text{CAVS}_{64}$ y $\text{CAVS}_{128}$ simulando 10 pasos de tiempo. Los resultados presentados en la tabla muestran que se pueden obtener niveles de aceleración razonables cuando se utiliza la versión basada en GPU del modelo `caffa3d.MB`, alcanzando aceleraciones de hasta un 19 % del tiempo total del modelo. Adicionalmente, las aceleraciones obtenidas escalan con la dimensión de los casos tratados.

| Casos | Original `caffa3d.MB` | Utilizando GPU `caffa3d.MB` | % de aceleración |
|---|---|---|---|
| $\text{CAVS}_{64}$ | 85.290 | 74.284 | 13 % |
| $\text{CAVS}_{128}$ | 750.632 | 612.011 | 19 % |

Tabla 6: Tiempo de ejecución (en segundos) del modelo `caffa3d.MB` basado en GPU.

## 5. CONCLUSIONES Y TRABAJO FUTURO

En el trabajo se presentan diferentes implementaciones del solver SIP para matrices heptadiagonales que hacen el uso de GPU para acelerar el tiempo de ejecución. La evaluación experimental del trabajo se realizó tanto en aritmética de precisión simple como doble.

Los resultados obtenidos muestran que todas las versiones en GPU alcanzan mejores desempeños que la implementación en CPU. El método $\texttt{SIP7}_{\texttt{GPU}}\texttt{-HP}$ muestra aceleraciones de hasta 13.5×, mientras que la versión que utiliza acceso coalesced ($\texttt{SIP7}_{\texttt{GPU}}\texttt{-HPC}$) obtuvo aceleraciones de hasta un 22× en el mejor caso. Las mejoras obtenidas se reducen cuando se utiliza precisión doble, pero las aceleraciones obtenidas siguen siendo importantes, alcanzando en el mejor caso 21.89× contra la implementación en doble precisión para CPU.

También se incluyó la versión más eficiente del solver en GPU en el modelo numérico `caffa3d.MB` y se evaluó su impacto en el desempeño del modelo completo. Para alcanzar

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

dicha integración fue necesario el desarrollo de un módulo interfaz para la invocación de las rutinas desde Fortran. Los experimentos muestran mejoras aceptables para los casos evaluados que alcanzan el 19 % del tiempo de ejecución del algoritmo original.

A partir del trabajo realizado y las conclusiones extraídas, se plantean distintas líneas de trabajos futuros. En primera instancia continuar mejorando los algoritmos implementados, buscando hacer mayor aprovechamiento del acceso coalesced y de la memoria compartida de la GPU. También se plantea estudiar la migración de otras etapas del modelo `caffa3d.MB` a la GPU de forma de reducir el tiempo de ejecución de dichas etapas así como también de eliminar el costo de la transferencia de datos que implica la ejecución en CPU y GPU. Por último, parece interesante evaluar otro tipo de técnicas para acelerar este tipo de aplicaciones como el uso de múltiples GPUs y precisión mixta. En especial esta última, teniendo en cuenta que la capacidad de cómputo en las GPUs ha credido en mayor medida que las velocidades de tranferencia entre CPU-GPU y las nuevas placas Kepler tienen importanes diferencias entre evaluar aritmética de precisión simple y doble.

## REFERENCIAS


Albensoeder S. y Kuhlmann H.C. Accurate three-dimensional lid-driven cavity flow. *J. Comput. Phys.*, 206(2):536–558, 2005. ISSN 0021-9991.

Barrachina S., Castillo M., Igual F.D., Mayo R., y Quintana-Ortí E.S. Evaluation and tuning of the level 3 cublas for graphics processors. En *IPDPS*, páginas 1–8. IEEE, 2008.

Benner P., Ezzatti P., Quintana-Ortí E.S., y Remón A. Using hybrid CPU-GPU platforms to accelerate the computation of the matrix sign function. En H. Lin, M. Alexander, M. Forsell, A. Knüpfer, R. Prodan, L. Sousa, y A. Streit, editores, *7th Int. Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Lecture Notes in Computer Science, Vol. 6043, páginas 132–139. Springer-Verlag, 2009.

Deserno F., Hager G., Brechtefeld F., y Wellein G. Basic optimization strategies for cfd-codes. *Technical report, Regionales Rechenzentrum Erlangen*, 2002.

Ezzatti P., Quintana-Ortí E.S., y Remón A. Using graphics processors to accelerate the computation of the matrix inverse. *The Journal of Supercomputing*, 58(3):429–437, 2011.

Ferziger J. y Perić M. *Computational methods for fluid dynamics*. Numerical methods: Research and development. Springer-Verlag, 2002. ISBN 9783540594345.

Halada L. y Lucká M. A parallel strongly implicit algorithm for solving of diffusion equations. En *Proceedings of the 4th Inter. ACPC Conference Including Special Tracks on Parallel Numerics and Parallel Computing in Image Processing, Video Processing, and Multimedia: Parallel Computation*, ParNum'99, páginas 78–84. Springer-Verlag, London, UK, 1999.

Iwatsu R., Hyun J.M., y Kuwahara K. Analyses of three-dimensional flow calculations in a driven cavity. *Fluid Dynamics Research*, 6(2):91, 1990.

Li R. y Saad Y. Gpu-accelerated preconditioned iterative linear solvers. Technical report, university of minnesota, 2010.

Naumov M. Incomplete-lu and cholesky preconditioned. iterative methods using cusparse and cublas. Nvidia white paper, 2011.

Reeve J.S., Scurr A.D., y Merlin J.H. Parallel versions of stone's strongly implicit algorithm. *Concurrency and Computation: Practice and Experience*, 13(12):1049–1062, 2001.

Stone H. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM Journal of Numerical Analysis*, 1, 1968.

Tomov S., Dongarra J., y Baboulin M. Towards dense linear algebra for hybrid gpu accelerated manycore systems. *Parallel Comput.*, 36(5-6):232–240, 2010. ISSN 0167-8191.


Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar

Usera G., Vernet A., y Ferré J. A parallel block-structured finite volume: Method for flows in complex geometry with sliding interfaces. *Flow, Turbulence and Combustion*, 81:471–495, 2008.

Copyright © 2012 Asociación Argentina de Mecánica Computacional http://www.amcaonline.org.ar