Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXIV, págs. 1207-1233 (artículo completo)
Sebastián Giusti, Martín Pucheta y Mario Storti (Eds.)
Córdoba, 8-11 Noviembre 2016

RESOLUCIÓN DE POISSON POR MÉTODO DE DESCOMPOSICIÓN DE DOMINIO EN MULTI-GPU

Fernando F. Benitez[a], Mario A. Storti[a] y Jorge D'Elia[a]

[a]*Centro de Investigación de Métodos Computacionales (CIMEC), CONICET, Predio CONICET Santa Fe, Colectora Ruta Nac 168, Km 472, Paraje El Pozo, Santa Fe, Argentina, http://www.cimec.org.a*

Palabras Clave: Graphics Processing Unit, Multi-GPGPU, Ecuación de Poisson, Diferencias Finitas.

Resumen. La solución de la ecuación de Poisson mediante un enfoque multi-GPU, con GPUs ubicados en diferentes nodos del clúster que interactúan mediante el protocolo MPI (Message Passing Interface) es un tema de investigación relativamente reciente. La idea es usar descomposición de dominio, de modo que cada GPU resuelve una parte del dominio computacional y MPI es usado para intercambiar datos de borde. En este trabajo se presenta un solver para la ecuación de Poisson basado en PCG (Gradientes Conjugados Precondicionado) en el contexto del método de volúmenes finitos sobre mallas estructuradas cartesianas de paso constante. El precondicionamiento consiste en resolver con FFT (Fast Fourier Transform) que es un método directo con complejidad algorítmica O(N log N) en cada subdominio, con condiciones ficticias (por ejemplo Dirichlet homogénea) en las interfases. El esquema itera sobre el vector global, si bien el residuo del problema precondicionado es no nulo sólo en la interfase y por lo tanto equivale a iterar sólo sobre la misma. Se presenta las propiedades de convergencia del algoritmo y su performance computacional en GPU. Además se presenta una extensión del algoritmo anterior en el cual los subdominios se solapan en una pequeña banda alrededor de la interfase. De esta forma se busca incrementar la tasa de convergencia, pero por otra parte es necesario transmitir entre las placas toda la información de la banda. Se presenta las propiedades de convergencia del algoritmo y su performance computacional en GPU.

1. INTRODUCCIÓN

En los sistemas paralelos y distribuidos, las GPGPUs (General-Purpose Computing on Graphics Processing Units) se han convertido en un serio competidor en el clásico ambiente de las CPUs. El alto rendimiento que concierne a operaciones de punto flotante ofrece capacidades que las hacen muy atractivas de forma tal que los algoritmos numéricos pueden acelerarse sustancialmente siempre y cuando se mapeen bien a la característica específica del hardware. Además, la tecnología de transferencia de datos se sigue perfeccionando buscando reducir la brecha ya existente entre el ancho de banda y la velocidad de cálculo. El presente trabajo se centra en el problema de Poisson que surge en una amplia gama de aplicaciones y disciplinas como la dinámica de fluidos computacional, física de gravitación, electrostática, etc. Este problema involucra resolver un sistema lineal disperso simétrico y definido positivo (SPD). Un algoritmo iterativo de uso común y ampliamente estudiado para resolver sistemas SPD es el método de gradientes conjugados (CG) y en conjunción con un precondicionador apropiado se tiene el método de gradientes conjugados precondicionado (PCG). Aunque otros métodos altamente sofisticados como los métodos multi-grilla son generalmente más rápidos, los métodos de Krylov siguen siendo una alternativa importante debido a que son fáciles de implementar y permiten manejar condiciones de borde complejas naturalmente. Sin embargo, un mapeo directo y eficiente de estos algoritmos en arquitecturas de computación paralelas con múltiples GPUs es difícil debido a las características específicas de desempeño de las GPUs. En particular, los esquemas de acceso a memoria, latencia y ancho de banda exhiben comportamientos diferentes con respecto a sistemas tradicionales basados en CPU.

El objetivo del presente trabajo ha sido implementar algoritmos para resolver el problema de Poisson y que sean adecuados para arquitecturas multi-GPU. Para ello se utilizaron métodos de Krylov. En particular se estudiaron cuestiones inherentes a precondicionadores y estrategias de comunicación para alcanzar speedups razonables.

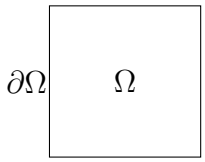
Las secciones 1 a 7 presentan el marco teórico del problema de Poisson y métodos ya conocidos para la discretización del problema, técnicas de diagonalización de sistemas para el desarrollo de resolvedores rápidos de Poisson (que serán utilizados como precondicionadores), técnicas de descomposición de dominio, métodos de Krylov y cuestiones con respecto a la comunicación de datos entre GPUs. La sección 8 presenta los casos de estudio y los resultados obtenidos. Finalmente se presentan las conclusiones del trabajo en la última sección.

2. ECUACIÓN DE POISSON

La ecuación de Poisson es una ecuación diferencial en derivadas parciales (PDE) de segundo orden, su forma general es

$$-\Delta\phi = f \tag{1}$$

Donde Δ es el operador Laplaciano continuo, mientras que ϕ y f son funciones reales sobre un dominio Ω y contorno $\partial\Omega$.



En un espacio euclidiano y utilizando coordenadas cartesianas la ecuación de Poisson tiene

las siguientes formulaciones

$$\begin{aligned} -\left(\frac{\partial^2}{\partial^2 x}\right)\phi = f \quad 1D \\ -\left(\frac{\partial^2}{\partial^2 x}+\frac{\partial^2}{\partial^2 y}\right)\phi = f \quad 2D \\ -\left(\frac{\partial^2}{\partial^2 x}+\frac{\partial^2}{\partial^2 y}+\frac{\partial^2}{\partial^2 z}\right)\phi = f \quad 3D \end{aligned} \tag{2}$$

2.1. Condiciones de bordes

Las condiciones de bordes más usuales son (con ejemplos en 1D) :

1. Condición de borde Dirichlet

$$\phi|_{x=0} = 0 \quad \phi|_{x=1} = 0 \tag{3}$$

2. Condición de borde Neumann

$$\frac{\partial\phi}{\partial x}|_{x=0} = 0 \quad \frac{\partial\phi}{\partial x}|_{x=1} = 0 \tag{4}$$

3. Condición de borde periódica

$$\phi|_{x=0} = \phi|_{x=1} \tag{5}$$

En este trabajo se asume condiciones de borde Dirichlet cero en el dominio del problema y condiciones de borde Dirichlet cero y periódicas en el subdominio para construir el precondicionador.

3. MÉTODO DE DISCRETIZACIÓN DE LA ECUACIÓN DE POISSON

Una vez definida la ecuación de Poisson y sus condiciones de bordes existen varios métodos que permiten transformar la EDP en un sistema de ecuaciones algebraicas para que puedan ser resueltos en una computadora como el método de diferencias finitas (FDM), elementos finitos (FEM) y volúmenes finitos (FVM). Para este trabajo se utilizo FVM que en una malla estructurada de paso constante y para el problema de Poisson puede verse como FDM

3.1. Método de diferencias finitas

Este método utiliza la formulación diferencial de las ecuaciones donde las derivadas parciales de la función evaluadas en los nodos del dominio son reemplazadas por aproximaciones en términos de los valores nodales de la función. La aplicación de este método es sencillo en mallas estructuradas. Además es fácil obtener esquemas de alto orden. Entre sus inconvenientes se tiene que es complicada su aplicación en dominios de geometrías irregulares

Aplicando FDM al operador diferencial laplaciano 1D se halla una aproximación de segundo orden que tiene la siguiente forma

$$(\frac{d^2\phi}{d^2x})_i = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} + O(\Delta x^2) \tag{6}$$

Si se desprecia el termino $O(\Delta x^2)$ en la ecuación (6) se obtiene el operador Laplaciano discreto en 1D aplicado a ϕ.

$$(\Delta\phi)_i = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} \tag{7}$$

Tomando este último término y los valores nodales de la función del miembro derecho de la ecuación (2) en su versión 1D se obtiene la discretización por FDM del problema de Poisson en 1D

$$-\frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = f_i \tag{8}$$

3.2. Método de diferencias finitas en 2 y 3 dimensiones

El método puede extenderse a mas dimensiones (Burden y Faires, 2002). A continuación se presenta el Laplaciano discreto aplicado a ϕ en 2 y 3 dimensiones.

$$(\Delta\phi)_{ij} = \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}, \tag{9}$$

$$(\Delta\phi)_{ijk} = \frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{\Delta x^2} + \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{\Delta y^2} + \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{\Delta z^2} \tag{10}$$

A continuación se definen los operadores discretos en su forma matricial para el caso 1D teniendo en cuenta las condiciones de borde.

La matriz del operador Laplaciano discreto para condiciones de borde Dirichlet es

$$\Delta_h^D = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \tag{11}$$

Donde $h = \frac{1}{N+1}$ siendo N la cantidad de puntos.

La matriz del operador Laplaciano discreto para condiciones de borde periódicas es

$$\Delta_h^P = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & -1 \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ -1 & & & -1 & 2 \end{bmatrix} \tag{12}$$

Donde $h = \frac{1}{N}$ siendo N la cantidad de puntos.

4. DIAGONALIZACIÓN DE SISTEMAS

4.1. Diagonalización para problema con condiciones periódicas

Se sabe que los autovalores y autovectores del Laplaciano discreto Δ_h^P son

$$\lambda_k = h^{-2}(2 - 2\cos\frac{2\pi k}{N}) \tag{13}$$

$$v^{jk} = e^{\frac{i2\pi jk}{N}} \quad j = 0...N-1 \quad k = 0...N-1 \tag{14}$$

Donde el índice j indica el j-ésimo elemento del autovector y el índice k indica el k-ésimo autovector

Para obtener estos resultados se puede hacer lo siguiente

$$\begin{aligned} h^2\Delta_h^P v^{jk} &= 2v^{jk} - v^{(j-1)k} - v^{(j+1)k} \\ &= v^{jk}(2 - v^{-k} - v^k) \end{aligned} \tag{15}$$

usando la ecuación (14) y la relación de Euler

$$e^{i\phi} = \cos\phi + i\sin\phi \tag{16}$$

el término $-v^{-k} - v^k$ de la ecuación (15) puede escribirse

$$\begin{aligned} -v^{-k} - v^k &= -\left(e^{\frac{-i2\pi k}{N}} + e^{\frac{i2\pi k}{N}}\right) \\ &= -\cos\frac{2\pi k}{N} + i\sin\frac{2\pi k}{N} - \cos\frac{2\pi k}{N} - i\sin\frac{2\pi k}{N} \\ &= -2\cos\frac{2\pi k}{N} \end{aligned} \tag{17}$$

Por lo tanto, si se reemplaza este último resultado en la ecuación (15) se tiene

$$h^2\Delta_h^P v^{jk} = (2 - 2\cos\frac{2\pi k}{N})v^{jk} \tag{18}$$

Si se observa la ecuación (18) se puede ver que en ambos miembros se tiene v^{jk}. Por lo tanto, por definición de autovalores y autovectores, v^{jk} es un autovector de Δ_h^P y el autovalor asociado es $\lambda_k = h^{-2}(2 - 2\cos\frac{2\pi k}{N})$

Si se arreglan todos los autovectores teniendo en cuenta exponente negativo y se multiplica por el coeficiente $\frac{1}{\sqrt{N}}$ para obtener una matriz de transformación se tiene la Transformada de Fourier Discreta (DFT). Si se obtiene el conjugado de cada elemento de esta matriz se tiene la Transformada Inversa de Fourier Discreta (IDFT).

$$F_{jk} = \frac{1}{\sqrt{N}}v^{-jk}, \quad F_{jk}^{-1} = \frac{1}{\sqrt{N}}v^{jk} \tag{19}$$

Estas matrices son simétricas y orto-normales. Usando estas matrices de transformación se puede escribir

$$\Delta_h^P = F^{-1}\Lambda F \tag{20}$$

Donde

$$\Lambda = diag(h^{-2}(2 - 2\cos\frac{2\pi k}{N})) \tag{21}$$

Ahora si se sustituye esta relación en la ecuación de Poisson

$$\begin{aligned} \Delta_h^P \phi &= f \\ F^{-1}\Lambda F\phi &= f \end{aligned} \tag{22}$$

Si se aplica F a ambos lados de la ecuación

$$FF^{-1}\Lambda F\phi = Ff \tag{23}$$

Luego con $\hat{\phi} = F\phi$ y $\hat{f} = Ff$ se tiene

$$\lambda_k \hat{\phi}_k = \hat{f}_k \tag{24}$$

Entonces un camino para resolver la ecuación de Poisson para ϕ_k es obtener $\hat{\phi}_k = \lambda_k^{-1}\hat{f}_k$ y aplicar la Transformada Inversa de Fourier a $\hat{\phi}_k$ para tener finalmente ϕ_k. Sin embargo hay que considerar el caso $k = 0$:

$$\lambda_0 \hat{\phi}_0 = \hat{f}_0 \tag{25}$$

El autovalor λ_0 es cero, y el sistema no tiene solución a menos que $\hat{f}_0$ no sea otro valor que cero. Esta es la condición de *compatibilidad* para el problema de Poisson con condiciones de borde periódicas. Si $\hat{f}_0 = 0$ entonces $\hat{\phi}_0$ puede tomar cualquier valor, pero si se fija a cero, entonces se obtiene la solución promedio de mínimos cuadrados.

La DFT y la IDFT pueden ser calculadas de forma eficiente mediante la Transformada Rápida de Fourier (FFT) y la Transformada Rápida de Fourier inversa (IFFT) respectivamente donde la complejidad algorítmica de ambas es de $O(Nlog(N))$. Por lo tanto el algoritmo del resolvedor rápido de Poisson con condiciones de borde periódicas es

1. Aplicar la FFT a f.
2. Dividir cada elemento de f por los autovalores no ceros, fijar a cero en otro caso. Con ello se obtiene $\hat{\phi}$.
3. Aplicar la IFFT para obtener finalmente ϕ.

4.2. Diagonalización para problema con condiciones Dirichlet

Se sabe que los autovalores y autovectores del Laplaciano discreto Δ_h^D son

$$\lambda_k = h^{-2}(2 - 2\cos\frac{\pi k}{N+1}) \tag{26}$$

$$v^{jk} = \sqrt{\frac{2}{N+1}}\sin(\frac{\pi jk}{N+1}) \quad j = 1...N \quad k = 1...N \tag{27}$$

Como k va de 1 a N no se tiene autovalores iguales a cero, además se puede observar que todos los autovalores son positivos, por lo tanto Δ_h^D es definida positiva.

La descomposición es la siguiente

$$\Delta_h^D = S^{-1}\Lambda S \tag{28}$$

donde la matriz S es la Transformada Seno Discreta (DST),

$$S_{jk} = \sqrt{\frac{2}{N+1}} \sin(\frac{\pi jk}{N+1}) \tag{29}$$

Además esta matriz de transformación cumple con las propiedades $S = S^t$ y $S = S^{-1}$.

4.3. Relación entre la DST y la DFT

La DST de tamaño N puede ser calculada desde la DFT de tamaño $M = 2N + 2$ de la siguiente forma

$$(S_N x)_k = i(F_{2N+2} z)_{k+1}, \quad k = 1, .., N \tag{30}$$

donde x es la entrada y z es la extensión impar, por ejemplo, si se tiene $x = [a \quad b \quad c]$, la extensión impar es $z = [0 \quad a \quad b \quad c \quad 0 \quad -c \quad -b \quad -a]$

Usando esta relación, el algoritmo del resolvedor rápido de Poisson con condiciones bordes Dirichlet es

1. Realizar la extensión impar de la entrada
2. Aplicar la FFT al resultado
3. Dividir elemento a elemento los grupos de componentes $k = 2, .., N + 1$ y $k = 2N + 2, ..., N + 3$ de $\hat{z}_k$ por los autovalores de Δ_h^D.
4. Aplicar la IFFT al resultado.
5. Extraer del resultado las componentes $k = 2, ..., N + 1$.

4.4. Diagonalización en 2 dimensiones

Se considera la ecuación de Poisson en un dominio rectangular con condiciones Dirichlet:

$$-\Delta u = f, \quad x, y \in \Omega \tag{31}$$
$$u = 0, \quad x, y \in \partial\Omega \tag{32}$$

Utilizando el Laplaciano discreto 2D se tiene

$$(\Delta^D_{h_x h_y} u)_{i,j} = (2u_{i,j} - u_{i+1,j} - u_{i-1,j})\frac{1}{h_x^2} + (2u_{i,j} - u_{i,j+1} - u_{i,j-1})\frac{1}{h_y^2} = f_{i,j} \tag{33}$$

Donde $h_x = \frac{L_x}{N_1+1}$ y $h_y = \frac{L_y}{N_2+1}$ donde N_1 y N_2 son la cantidad de nodos internos en dirección x e y respectivamente. Se puede observar en la ecuación (33) que aplicar el Laplaciano discreto 2D es lo mismo que aplicar el Laplaciano discreto 1D por filas y columnas a u y sumar las expresiones.

Si se arma el sistema (33) considerando el operador Laplaciano 1D de $N_i \times N_i$ con $i = 1, 2$.

$$\Delta_h^D = \frac{1}{h^2} \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \tag{34}$$

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1N_2} \\ u_{21} & u_{22} & \cdots & u_{2N_2} \\ \vdots & \vdots & \ddots & \vdots \\ u_{N_1 1} & u_{N_1 2} & \cdots & u_{N_1 N_2} \end{bmatrix} \tag{35}$$

$$F = \begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1N_2} \\ f_{21} & f_{22} & \cdots & f_{2N_2} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N_1 1} & f_{N_1 2} & \cdots & f_{N_1 N_2} \end{bmatrix} \tag{36}$$

Se tiene

$$\begin{gathered} \Delta^D_{N1\times N1} U + U \Delta^D_{N2\times N2} = F \\ S^{-1}_{N1\times N1} \Lambda_{k1} S_{N1\times N1} U + U S^{-1}_{N2\times N2} \Lambda_{k2} S_{N2\times N2} = F \end{gathered} \tag{37}$$

Aplicando $S_{N1\times N1}$ por izquierda y $S^{-1}_{N2\times N2}$ por derecha se tiene

$$\Lambda_{k1} S_{N1\times N1} U S^{-1}_{N2\times N2} + S_{N1\times N1} U S^{-1}_{N2\times N2} \Lambda_{k2} = S_{N1\times N1} F S^{-1}_{N2\times N2} \tag{38}$$

Definiendo $\hat{U} = S_{N1\times N1} U S^{-1}_{N2\times N2}$ y $\hat{F} = S_{N1\times N1} F S^{-1}_{N2\times N2}$ se tiene

$$\Lambda_{k1} \hat{U} + \hat{U} \Lambda_{k2} = \hat{F} \tag{39}$$

Luego este sistema puede ser visto componente a componente como

$$\lambda_i \hat{u}_{ij} + \hat{u}_{ij} \lambda_j = \hat{f}_{ij} \tag{40}$$

Entonces

$$\hat{u}_{ij} = \frac{\hat{f}_{ij}}{\lambda_i + \lambda_j} \tag{41}$$

Finalmente se obtiene el resultado haciendo

$$U = S^{-1}_{N1\times N1} \hat{U} S_{N2\times N2} \tag{42}$$

El algoritmo es el siguiente:

1. Calcular $\hat{F}$ aplicando la IDST a las filas de F y luego la DST a las columnas del resultado.
2. Calcular $\hat{U}$ dividiendo el resultado por la suma de los autovalores λ_i y λ_j
3. Calcular U aplicando la DST a las filas de $\hat{U}$ y luego la IDST a las columnas del resultado

Como la DST es igual a la IDST se puede ver a la operación DST 1D por filas y columnas como la DST 2D. Mediante un procedimiento similar se deriva los algoritmos para el caso 2D con condiciones periódicas.

4.5. Diagonalización en 3 dimensiones

De la misma forma que se obtiene la ecuación (41) se deriva una ecuación para el caso 3D mediante el uso de la DST 3D

$$\hat{u}_{ijk} = -\frac{\hat{f}_{ijk}}{\lambda_i + \lambda_j + \lambda_k} \tag{43}$$

Luego puede hallarse U aplicando la DST 3D al resultado. Mediante un procedimiento similar se deriva los algoritmos para el caso 3D con condiciones de borde periódicas.
Más detalles sobre esta temática puede encontrarse en Buzbee et al. (1970) , Van Loan (1992) y Storti et al. (2013).

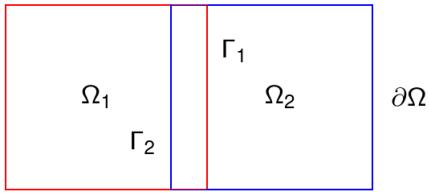
5. DESCOMPOSICIÓN DE DOMINIO

Para el desarrollo de los conceptos se tendrá en cuenta una subdivisión de 2 subdominios por simplicidad, sin embargo los mismos son válidos para un número mayor de subdominios.

5.1. Métodos Schwarz

Se asume que se tiene la siguiente subdivisión del dominio y que hay superposición.

$$\Omega = \cup_{n=1}^{2} \Omega_n$$



Una forma de definir los operadores locales de subdominios es mediante el uso de operadores de restricción que son matrices con entradas cero salvo en posiciones que corresponden a los nodos del subdominio. Para el caso de 2 subdominios son

$$R_1 = \begin{bmatrix} 1 & 0 & \dots & & & & 0 \\ 0 & \ddots & \ddots & & & & \vdots \\ \vdots & \ddots & & & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}, \quad R_2 = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ \vdots & & & & & \ddots & \vdots \\ & & & & \ddots & \ddots & 0 \\ 0 & \dots & & & & 0 & 1 \end{bmatrix} \tag{44}$$

Haciendo uso de las matrices de restricción se puede definir el conjunto incógnitas del primer subdominio $R_1 u$ y el conjunto de incógnitas del segundo subdominio $R_2 u$. También se puede definir las restricciones de A para cada subdominio con estas matrices

$$A_j = R_j A R_j^t \quad j = 1, 2 \tag{45}$$

5.2. Método aditivo de Schwarz como precondicionador para el PCG

Una versión muy utilizada de la familia de métodos de Schwarz es el método aditivo de Schwarz (ASM) (Toselli y Widlund, 2006; Gander, 2008; Dryja, 2016) cuya formulación es la siguiente

$$u^{n+1} = u^n + (R_1^t A_1^{-1} R_1 + R_2^t A_2^{-1} R_2)(f - Au^n) \tag{46}$$

El ASM, además de ser utilizado en métodos iterativos estacionarios también puede ser utilizado como precondicionador en métodos iterativos de Krylov. Específicamente la idea es utilizar a la matriz de iteración del ASM como precondicionador para el PCG.

$$M_{AS}^{-1} = \sum_{i=1}^{N} R_i^t (R_i A R_i^t)^{-1} R_i$$

Donde R_i es la matriz de restricción correspondiente al subdominio Ω_i. La idea es que el paso de precondicionamiento se calcule de forma eficiente. Como el término $(R_i A R_i^t)^{-1}$ es un operador local igual a A_i^{-1} (la inversa del Laplaciano que opera en el subdominio Ω_i) entonces aplicar la inversa sobre el residuo equivale a resolver de forma local un problema de Poisson en el subdominio. Luego esta operación puede ser calculada mediante un resolvedor rápido de Poisson.

6. ALGORITMOS

En este trabajo se presenta los algoritmos CG y PCG en su versiones para sistemas de memoria distribuida pero no se expone un análisis profundo de los métodos. Estos métodos se pueden ver mas detalladamente en Golub y Van Loan (1996). Pueden verse super-indices n en las matrices y vectores indicando a que $rank$ pertenece. También se puede ver las operaciones del CG como son el producto matriz-vector, SAXPY, DOT y las funciones de comunicación que sirven para intercambiar los valores de borde y los valores de superposición.

Algoritmo 1: Gradientes Conjugados para sistemas distribuidos

$\mathbf{r}_0^n \leftarrow \mathbf{b}^n - \mathbf{A}^n\mathbf{x}_0^n$
$\mathbf{p}_0^n \leftarrow \mathbf{r}_0^n$
$reduccion\_distribucion(\mathbf{r}_0^n\mathbf{r}_0^n, rr_{viejo}, suma)$
for $i = 1$ to $itermax$ **do**
  $intercambio\_halo(\mathbf{p}_{i-1}^n)$
  $reduccion\_distribucion(\mathbf{p}_{i-1}^n\mathbf{A}^n\mathbf{p}_{i-1}^n, pAp, suma)$
  $\alpha \leftarrow rr_{viejo}/pAp$
  $\mathbf{x}_i^n \leftarrow \mathbf{x}_{i-1}^n + \alpha\mathbf{p}_{i-1}^n$
  $\mathbf{r}_i^n \leftarrow \mathbf{r}_{i-1}^n - \alpha\mathbf{A}^n\mathbf{p}_{i-1}^n$
  $reduccion\_distribucion(\mathbf{r}_i^n\mathbf{r}_i^n, rr_{nuevo}, suma)$
  if $sqrt(rr_{nuevo}) < tol$ **then**
    return $\mathbf{x}_i^n$
  end if
  $\mathbf{p}_i^n \leftarrow \mathbf{r}_i^n + (rr_{nuevo}/rr_{viejo})\mathbf{p}_{i-1}^n$
  $rr_{viejo} = rr_{nuevo}$
end for

Algoritmo 2: Gradientes Conjugados Precondicionado para sistemas distribuidos con superposición de subdominios

$\mathbf{r}_0^n \leftarrow \mathbf{b}^n - \mathbf{A}^n\mathbf{x}_0^n$
$intercambio\_superposicion(\mathbf{r}_0^n, \mathbf{rsup}^n)$
$\mathbf{qsup}^n \leftarrow (\mathbf{Msup}^n)^{-1}\mathbf{rsup}^n$
$suma\_superposicion(\mathbf{qsup}^n, \mathbf{q}_0^n)$
$\mathbf{p}_0^n \leftarrow \mathbf{q}_0^n$
$reduccion\_distribucion(\mathbf{r}_0^n\mathbf{q}_0^n, rq_{viejo}, suma)$
for $i = 1$ to $itermax$ **do**
  $intercambio\_halo(\mathbf{p}_{i-1}^n)$
  $reduccion\_distribucion(\mathbf{p}_{i-1}^n\mathbf{A}^n\mathbf{p}_{i-1}^n, pAp, suma)$
  $\alpha \leftarrow rq_{viejo}/pAp$
  $\mathbf{x}_i^n \leftarrow \mathbf{x}_{i-1}^n + \alpha\mathbf{p}_{i-1}^n$
  $\mathbf{r}_i^n \leftarrow \mathbf{r}_{i-1}^n - \alpha\mathbf{A}^n\mathbf{p}_{i-1}^n$
  $reduccion\_distribucion(\mathbf{r}_i^n\mathbf{r}_i^n, rr, suma)$
  if $sqrt(rr) < tol$ **then**
    return $\mathbf{x}_i^n$
  end if
  $intercambio\_superposicion(\mathbf{r}_i^n, \mathbf{rsup}^n)$
  $\mathbf{qsup}^n \leftarrow (\mathbf{Msup}^n)^{-1}\mathbf{rsup}^n$
  $suma\_superposicion(\mathbf{qsup}^n, \mathbf{q}_i^n)$
  $reduccion\_distribucion(\mathbf{r}_i^n\mathbf{q}_i^n, rq_{nuevo}, suma)$
  $\mathbf{p}_i^n \leftarrow \mathbf{q}_i^n + (rq_{nuevo}/rq_{viejo})\mathbf{p}_{i-1}^n$
  $rq_{viejo} = rq_{nuevo}$
end for

7. MULTI-GPU

A continuación se detallan razones para las cuales se desearía combinar los enfoques de programación paralela de MPI (Forum, 2009) y Arquitectura Unificada de Dispositivos de Cómputo (CUDA) (Sanders y Kandrot, 2010)

1. Para resolver problemas con un tamaño de datos demasiado grande para caber en la memoria de una sola GPU.
2. Para resolver problemas que requieren tiempo de cálculo excesivamente largo en un solo nodo.
3. Para acelerar una aplicación MPI existente con las GPUs.
4. Para que una aplicación de un solo nodo con multi-GPU pueda escalar a través de múltiples nodos.

Una dificultad que se presenta cuando se utiliza este tipo de arquitectura es que la transmisión de datos entre las GPUs es ineficiente. Dados dos hosts con GPUs la transferencia de datos entre las 2 GPUs sería del modo siguiente

$$GPU_1 \rightarrow Host_1$$
$$Host_1 \rightarrow Host_2$$
$$Host_2 \rightarrow GPU_2$$

Este esquema indica que como primer paso los datos almacenados en un vector en la memoria de la GPU_1 debe transferirse primero a un vector almacenado en el sistema de memoria del $Host_1$. Esta operación se puede realizar con la instrucción *cudaMemcpy* de CUDA o bien con el operador de asignación entre vectores *host* y *device* de una biblioteca de manejo de vectores en GPU como Thrust. Una vez terminada esta operación, se procede a transferir los datos de este último vector a un vector que reside en el sistema de memoria del $Host_2$ mediante una llamada de paso de mensajes MPI. Luego que finaliza esta operación, se procede a transferir los datos de este vector a un vector que reside en la memoria de la GPU_2 usando algunas de las operaciones mencionadas en el primer paso. Llevar a cabo estas tres operaciones puede derivar en baja performance del programa.

Sin embargo, mediante la tecnología GPUDirect2, MPI puede enviar y recibir buffers de GPU directamente, sin tener que organizar primero los buffers en la memoria del host. GPUDirect2 está disponible en varias implementaciones, siendo algunas de ellas OpenMPI y MVAPICH2.

8. CASOS DE ESTUDIO

El problema propuesto es la ecuación

$$-\Delta\phi = 1$$

El caso 2D un dominio rectangular de dimensiones $L \times L$ con $L = 1$.

El caso 3D es una caja de dimensiones $L \times L \times L$ con $L = 1$.

Para cada problema de las Tablas 1 y 2 se asumen condiciones de contorno Dirichlet homogéneas y fuente constante $f = 1$.

Problema	Dimensión del dominio	Nro. de subdominios
1	512x512	2
2	1024x1024	2
3	2048x2048	2

Tabla 1: Casos de estudio

Problema	Dimensión del dominio	Nro. de subdominios
1	64x64x64	2
2	128x128x128	2
3	200x200x200	2

Tabla 2: Casos de estudio

8.1. Hardware y software

Clúster Coyote - CIMEC

- 2 Xeon E5-1660 (6 núcleos).
- 2 EVGA GeForce GTX 580 (512 núcleos), micro-arquitectura Fermi, Compute Capability 2.0.
- CUDA 6.5
- CUFFT 6.5
- Thrust 6.5
- FFTW 3.3.4
- OpenMPI 1.8.4 con soporte CUDA-aware (GPUDirect2)
- Fedora 17

Clúster Mendieta - CCAD de la Universidad Nacional de Córdoba

- 2 aceleradoras NVIDIA Tesla K20X (2496 núcleos), micro-arquitectura Kepler, Compute Capability 3.5.
- CUDA 6.5
- CUFFT 6.5
- Thrust 6.5
- MVAPICH2 2.1a con soporte CUDA-aware (GPUDirect2)
- CentOS 6.8

Las pruebas se realizaron considerando para la versión CPU todos los núcleos disponibles del Xeon E5-1660 (un total de 6 núcleos). El tipo de dato utilizado para el cómputo es doble precisión. Además para las operaciones DOT, SAXPY se utilizó la biblioteca Thrust. La operación Ap se implementó mediante el desarrollo del kernel producto matriz-vector ya que este producto es de estilo *matrix free*. Las operaciones FPS DST y FPS FFT se desarrollaron mediante el uso de las bibliotecas FFTW y CUFFT para las versiones de CPU y GPU respectivamente. La operación Halo se implementó mediante funciones MPI con soporte GPUDirect2 y también utilizando buffers auxiliares.

8.2. Dominio 2D

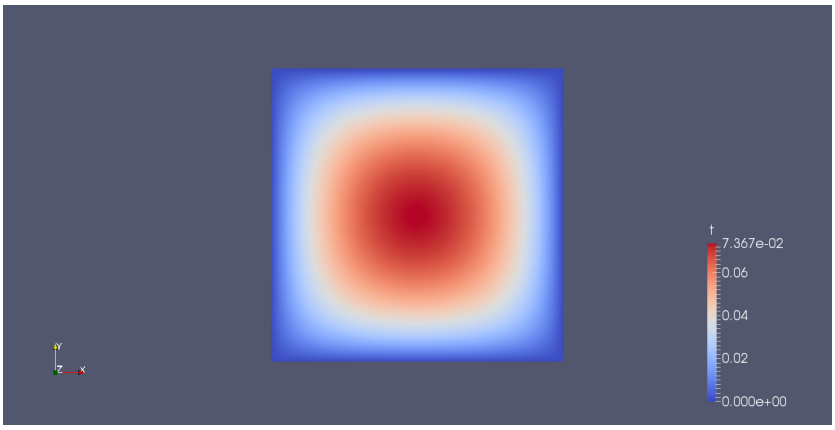


Figura 1: Dimensión 2048x2048

En la Figura 1 puede verse la visualización de la solución para el caso 2D.

8.2.1. Resultados - Operaciones

A continuación se presenta la participación de las operaciones en el tiempo global. Las operaciones DOT, SAXPY, Halo y Ap corresponden al CG, mientras que las operaciones FPS DST y FPS FFT corresponden al paso de precondicionamiento del PCG por DST y FFT sin superposición.

Dimensión	DOT [%]	SAXPY [%]	Halo [%]	Ap [%]
256x512	25.24	29.03	8.16	37.13
512x1024	21.84	25.98	7.42	44.75
1024x2048	16.75	25.74	6.48	51.02

Tabla 3: Participación en el tiempo global - CG - Xeon E5-1660

En las Tablas 3, 4 y 5 se puede observar que la operación Ap es la más costosa para el CG. Por otro lado se observa que a medida que el tamaño de los subdominios crece, la participación de comunicación (Halo) disminuye por el hecho de que la participación de cómputo (suma de participaciones de DOT, SAXPY y Ap) se incrementa. Esto es por que las operaciones de cómputo trabajan con datos en 2D (en todo el subdominio) mientras que la comunicación trabaja con datos 1D (borde del subominio , fila o columna). Los datos en 2D crece en mayor proporción que los datos en 1D a medida que aumenta el tamaño del problema.

Dimensión	DOT [%]	SAXPY [%]	Halo [%]	Ap [%]
256x512	24.61	31.01	9.97	34.40
512x1024	23.05	29.87	7.93	39.14
1024x2048	19.81	24.22	7.20	48.76

Tabla 4: Participación en el tiempo global - CG - GeForce GTX 580

Dimensión	DOT [%]	SAXPY [%]	Halo [%]	Ap [%]
256x512	26.30	29.71	7.16	36.82
512x1024	22.14	28.03	6.78	43.04
1024x2048	19.70	24.54	6.01	49.74

Tabla 5: Participación en el tiempo global - CG - Tesla K20X

Dimensión	FPS DST [%]	FPS FFT [%]
256x512	83.47	34.63
512x1024	84.79	34.38
1024x2048	89.72	37.89

Tabla 6: Participación en el tiempo global - PCG - Xeon E5-1660

Dimensión	FPS DST [%]	FPS FFT [%]
256x512	87.67	29.76
512x1024	84.95	40.13
1024x2048	97.49	57.21

Tabla 7: Participación en el tiempo global - PCG - GeForce GTX 580

Dimensión	FPS DST [%]	FPS FFT [%]
256x512	88.01	30.40
512x1024	85.03	39.67
1024x2048	97.78	65.26

Tabla 8: Participación en el tiempo global - PCG - Tesla K20X

Por otro lado, para el PCG, el paso de precondición por DST es la operación mas costosa, como puede verse en las Tablas 6, 7 y 8, además su participación aumenta a medida que el tamaño del problema se incrementa. El paso de precondición por FFT tiene un porcentaje de participación menor con respecto al de precondición por DST ya que como no realiza la extensión impar de la entrada, el volumen de datos con el que trabaja es menor, por lo tanto es mas rápida su ejecución.

8.2.2. Resultados - Gradientes Conjugados

En la Tabla 9 se observa que a medida que el tamaño aumenta los speedups obtenidos con las GPUs se incrementan.

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X[s]
512x512	829	8.45 (1x)	2.05 (4.12x)	1.08 (7.86x)
1024x1024	1672	56.02 (1x)	8.92 (6.30x)	4.42 (12.71x)
2048x2048	3377	228.03 (1x)	18.88 (12.08x)	9.37 (24.34x)

Tabla 9: Tiempos CG

8.2.3. Resultados - Gradientes Conjugados Precondicionado

El paso de precondicionamiento es sin superposición de subdominios.

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X [s]
512x512	54	30.42 (1x)	2.84 (10.70x)	1.16 (26.23x)
1024x1024	80	149.68 (1x)	10.18 (14.69.x)	4.78 (31.28x)
2048x2048	119	573.81 (1x)	32.92 (17.43x)	16.38 (35.03x)

Tabla 10: Tiempos PCG precondicionamiento con DST sin superposición de subdominios

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X[s]
512x512	248	36.31 (1x)	2.21 (16.43x)	1.09 (33.38x)
1024x1024	479	197.50 (1x)	5.40 (36.54x)	2.73 (72.16x)
2048x2048	954	719.21 (1x)	10.79 (66.66x)	5.81 (123.79x)

Tabla 11: Tiempos PCG precondicionamiento con FFT

En las Tablas 10 y 11 se observa que a medida que el tamaño aumenta los speedups obtenidos con las GPUs se incrementan. El speedup del PCG FFT en comparación con el PCG DST son mas mayores y crecen con mayor proporción a medida que se incrementa el tamaño del problema.

Superposición de subdominios

El paso de precondicionamiento con superposición de subdominios se realiza con DST.

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X [s]
512x512 sup 20	8	1.15 (1x)	0.47 (2.43x)	0.44 (2.62x)
1024x1024 sup 20	8	4.81 (1x)	1.31 (3.67x)	0.62 (7.74x)
2048x2048 sup 20	16	12.32 (1x)	4.44 (2.73x)	2.26(5.45x)

Tabla 12: Tiempos PCG+DST con superposición de subdominios

En la Tabla 12 se observa los mejores tiempos (haciendo una comparación con los métodos anteriores) para cada problema, por lo tanto este método ha resultado ser el más rápido. Por otro lado, si bien los speedups alcanzados en las GPUs son superiores, no son en gran medida como los anteriores.

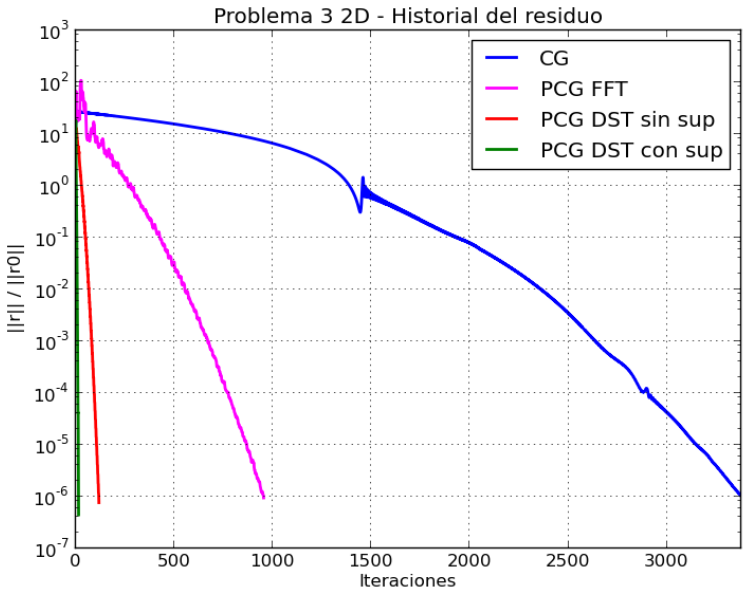


Figura 2: Dimensión 2048x2048

Problema	CG	PCG+DST sin sup	PCG+FFT sin sup
3	4.14x	7.24x	2.57x

Tabla 13: Comparación de speedup de PCG+DST con superposición de subdominios con el resto de los métodos.

8.2.4. Historial del residuo y speedup

En la Figura 2 se observa que el PCG DST con superposición tiene la mayor velocidad de convergencia. En la Tabla 13 se observa una comparación de los speedups del método mas rápido (PCG DST con superposición) con respecto a los otros métodos (CG, PCG DST sin superposición, PCG FFT sin superposición) considerando el hardware con mayores prestaciones.

8.2.5. Comparación para distintas capas

A continuación se presenta el promedio de tiempos obtenidos en el problema 3 con el PCG para la versión GPU con el fin de observar el speedup a medida que se incrementa el número de capas superpuestas y el historial del residuo.

Dimensión	tiempo [s]
2048x2048 sup 0	32.92 (1x)
2048x2048 sup 20	4.44 (7.41x)
2048x2048 sup 50	2.78 (11.84x)
2048x2048 sup 100	1.70 (19.36x)

Tabla 14: Comparación de tiempo y speedup para distintas capas - GeForce GTX 580

En la Tabla 14 y la Figura 3 se observa que a medida que se incrementan las capas de superposición se incrementa el speedup y la velocidad de convergencia. Este último resultado

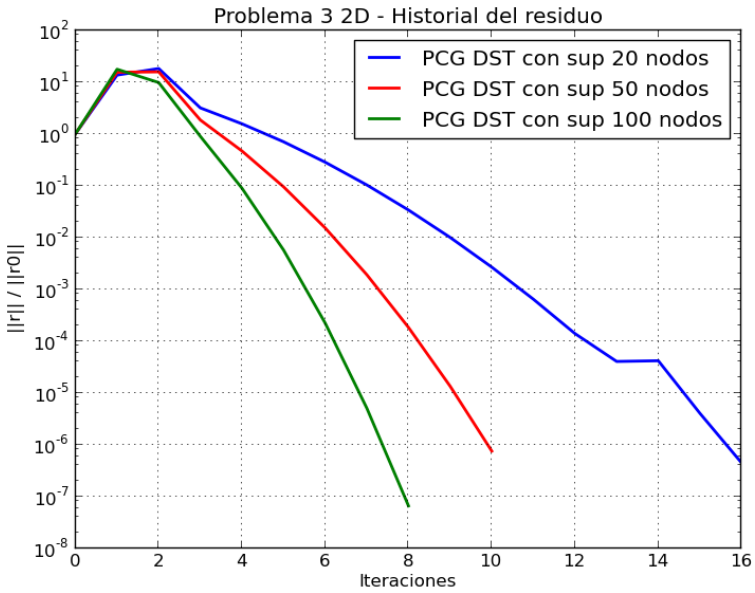


Figura 3: Dimensión 2048x2048

se debe a que el condicionamiento del sistema mejora a medida que aumenta las capas de superposición.

8.3. Dominio 3D

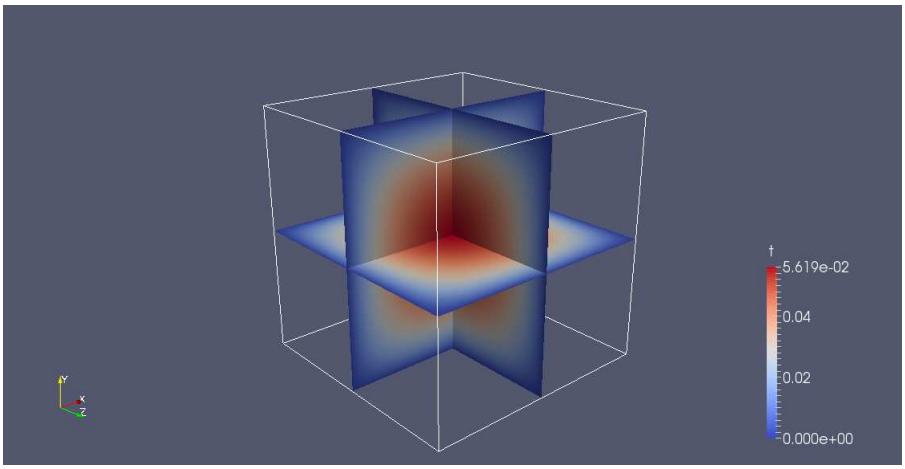


Figura 4: Dimensión 200x200x200 - cortes verticales y horizontales

En la Figura 4 puede verse la visualización de la solución para el caso 3D.

8.3.1. Resultados - Operaciones

A continuación se presenta la participación de las operaciones en el tiempo global. Las operaciones DOT, SAXPY, Halo y Ap corresponden al CG, mientras que las operaciones FPS DST y FPS FFT corresponden al paso de precondicionamiento del PCG por DST y FFT respectivamente.

Dimensión	DOT [%]	SAXPY [%]	Halo [%]	Ap [%]
32x64x64	14.89	17.04	9.93	58.13
64x128x128	11.39	16.43	8.03	60.14
100x200x200	11.02	16.31	7.65	65.01

Tabla 15: Participación en el tiempo global - CG - Xeon E5-1660

Dimensión	DOT [%]	SAXPY [%]	Halo [%]	Ap [%]
32x64x64	16.65	18.81	46.22	18.31
64x128x128	22.31	23.82	24.14	29.72
100x200x200	21.28	23.24	19.61	35.86

Tabla 16: Participación en el tiempo global - CG - GeForce GTX 580

Dimensión	DOT [%]	SAXPY [%]	Halo [%]	Ap [%]
32x64x64	23.95	18.01	37.94	20.09
64x128x128	22.03	26.84	23.92	27.20
100x200x200	18.85	28.09	16.78	36.27

Tabla 17: Participación en el tiempo global - CG - Tesla K20X

Dimensión	FPS DST [%]	FPS FFT [%]
32x64x64	79.73	26.06
64x128x128	88.06	28.40
100x200x200	93.20	38.37

Tabla 18: Participación en el tiempo global - PCG - Xeon E5-1660

Dimensión	FPS DST [%]	FPS FFT [%]
32x64x64	88.17	38.69
64x128x128	91.39	40.06
100x200x200	97.28	43.12

Tabla 19: Participación en el tiempo global - PCG - GeForce GTX 580

Dimensión	FPS DST [%]	FPS FFT [%]
32x64x64	81.97	24.68
64x128x128	82.94	29.49
100x200x200	90.05	38.74

Tabla 20: Participación en el tiempo global - PCG - Tesla K20X

En las Tablas 15, 16 y 17 podemos observar un comportamiento similar a 2D (en este caso las operaciones de comunicación trabajan con datos 2D y las de cómputo trabaja con datos 3D). Sin embargo se puede destacar la diferencia de participación de la operación Ap es acentuada entre la version CPU con las versiones GPU. Esto se debe a que la cantidad de puntos del subdominio y la computación aritmética por nodo se ha incrementado considerablemente (en 2D el stencil es de 5 puntos, mientras que en 3D es de 7 puntos). Para los pasos de precondición del PCG (ver Figuras 18, 19 y 20) se observan el mismo comportamiento que en 2D.

8.3.2. Resultados - Gradientes Conjugados

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X [s]
64x64x64	129	11.46 (1x)	3.60 (3.18x)	3.31 (3.47x)
128x128x128	261	63.22 (1x)	4.41 (14.33x)	4.32 (14.63x)
200x200x200	410	373.21 (1x)	9.54 (39.12x)	7.39 (50.50x)

Tabla 21: Tiempos CG

En la Tabla 21 se observa que a medida que el tamaño aumenta los speedups obtenidos con las GPUs se incrementan. Con respecto a 2D la diferencia son más notables. Esto se debe a que como se dijo anteriormente la carga aritmética se ha incrementado.

8.3.3. Resultados - Gradientes Conjugados Precondicionado

El paso de precondicionamiento es sin superposición de subdominios.

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X [s]
64x64x64	18	12.47 (1x)	4.81 (2.59x)	4.60 (2.71x)
128x128x128	28	83.57 (1x)	5.21 (16.32x)	4.98 (16.78x)
200x200x200	37	723.90 (1x)	16.78 (43.14x)	8.45 (85.67x)

Tabla 22: Tiempos PCG precondicionamiento con DST sin superposición de subdominios

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X [s]
64x64x64	57	11.06 (1x)	4.09 (2.70x)	4.07 (2.71x)
128x128x128	105	47.08 (1x)	4.61 (10.21x)	4.52 (10.41x)
200x200x200	156	315.53 (1x)	15.25 (20.69x)	9.21 (34.25x)

Tabla 23: Tiempos PCG precondicionamiento con FFT

En las Tablas 22 y 23 se observa que a medida que el tamaño aumenta los speedups obtenidos con las GPUs se incrementan.

Superposición de subdominios.

El paso de precondicionamiento con superposición de subdominios se realiza con DST.

Dimensión	iter.	Xeon E5-1660 [s]	GeForce GTX 580 [s]	Tesla K20X [s]
64x64x64 sup 10	5	3.66 (1x)	1.92 (2.1x)	1.74 (1.9x)
128x128x128 sup 10	6	6.56 (1x)	2.51 (2.61x)	2.47 (2.65x)
200x200x200 sup 10	8	34.09 (1x)	3.49 (9.77x)	3.02 (11.28x)

Tabla 24: Tiempos PCG+DST con superposición de subdominios

Un comportamiento similar al caso 2D se observa en 3D. En la Tabla 24 se muestra los tiempos para cada problema. Comparando con los algoritmos anteriores este método ha resultado

ser uno de los más rápidos. Por otro lado, si bien los speedups alcanzados en las GPUs son superiores, no lo son en gran medida como en los métodos anteriores.

8.3.4. Historial del residuo y speedup

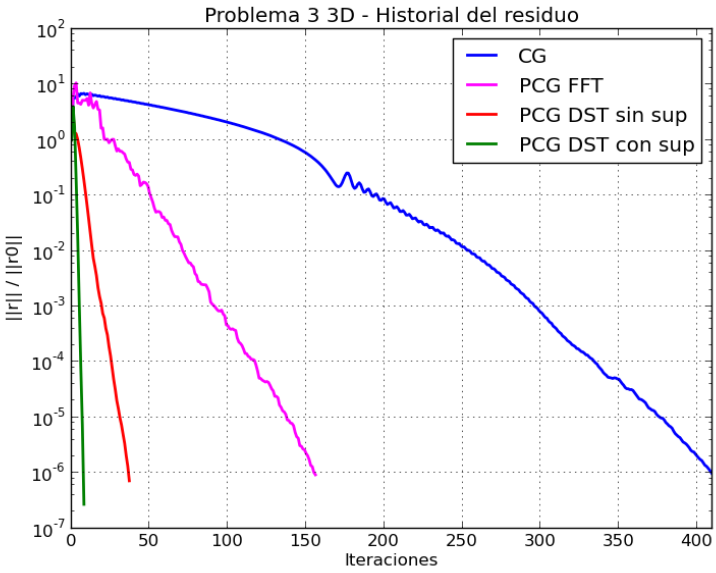


Figura 5: Dimensión 200x200x200

Problema	CG	PCG+DST sin sup	PCG+FFT sin sup
3	2.44x	2.79x	3.04x

Tabla 25: Comparación de speedup de PCG+DST con superposición de subdominios con el resto de los métodos.

En la Figura 5 se observa que el PCG DST con superposición tiene la mayor velocidad de convergencia. En la Tabla 25 se observa una comparación de los speedups de uno de los métodos más rápidos (PCG DST con superposición) con respecto a los otros métodos (CG, PCG DST sin superposición, PCG FFT sin superposición) considerando el hardware de mayores prestaciones. Se puede observar que los métodos CG y PCG DST con superposición tienen similares speedups. Esto ocurre por que el número de condición del sistema no es lo suficientemente grande. El CG toma 410 iteraciones para converger en contraste con las 8 iteraciones del PCG, pero cada iteración no es tan costosa como si lo es en el PCG ya que este último tiene que resolver un sistema (paso de precondición). Como el número de condición depende del tamaño del problema se espera que esta situación cambie a favor del PCG si se tratan problemas de mayor tamaño.

8.3.5. Comparación para distintas capas

Como en el caso 2D se presenta el promedio de tiempos obtenidos en el problema 3 con el PCG, con el fin de observar el speedup a medida que se incrementa el número de capas superpuestas y el historial del residuo.

Dimensión	tiempo [s]
200x200x200 sup 0	16.78 (1x)
200x200x200 sup 10	3.49 (4.81x)
200x200x200 sup 20	2.91 (5.76x)
200x200x200 sup 25	2.58 (6.50x)

Tabla 26: Comparación de tiempo y speedup para distintas capas - GeForce GTX 580

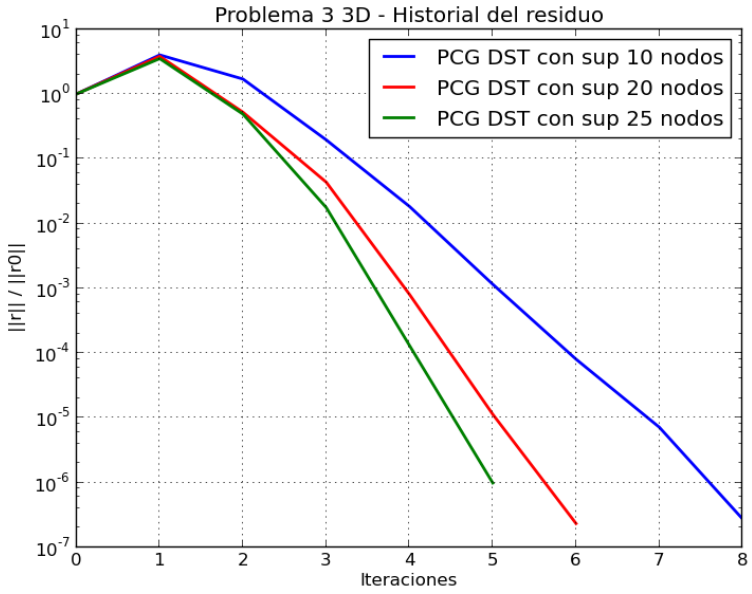


Figura 6: Dimensión 200x200x200

En la Tabla 26 y en la Figura 6 se observa que a medida que se incrementan las capas de superposición se incrementa el speedup y la velocidad de convergencia. Este último resultado se debe a que el condicionamiento del sistema mejora a medida que aumenta las capas de superposición.

8.4. Potencia pico teórica de cada plataforma de ejecución

CPU

Para el cálculo de la potencia pico teórica de un nodo con CPU se utilizó la siguiente fórmula

$$np[GFlops] = (s) \cdot (nn) \cdot (ipc) \cdot (ncpn)$$

Donde np representa la potencia en un solo nodo con CPU en GFlops, s la velocidad de CPU en GHz, nn el número de núcleos de CPU, ipc número de instrucciones por ciclo y $ncpn$ el número de CPUs por nodo. Finalmente para obtener la potencia de la plataforma multiplicamos por la cantidad de nodos, que para este trabajo se utilizaron 2.

Entonces la potencia pico teórica resulta

$$316,8[GFlops] = 3{,}30 \cdot 6 \cdot 8 \cdot 1 \cdot 2$$

GPU

Para el cálculo de la potencia pico teórica de un nodo con GPU se utilizó la siguiente fórmula

$$pg[GFlops] = (nsm) \cdot (npsm) \cdot (cpn) \cdot (nfpc)$$

Donde pg representa la potencia en un solo nodo con GPU en GFlops, nsm el número de multiprocesadores (Stream multiprocessor o SM), $npsm$ el número de núcleos por multiprocesador, cpn número de ciclos por núcleo y $nfpc$ el número de Flops por ciclo. Finalmente para obtener la potencia de la plataforma multiplicamos por la cantidad de nodos, que para este trabajo se utilizaron 2.

NVIDIA Tesla K20X

Las aceleradoras NVIDIA Tesla K20X (K20) poseen 2.496 núcleos. Estos núcleos se organizan en 13 SM que funcionan a una frecuencia 0,706 GHz. Cada SM posee 192 núcleos, todos los cuales están disponibles para cálculos de punto flotante de simple precisión, pero no todos están disponibles para doble precisión. Por lo tanto en cada SM solo 64 núcleos pueden realizar cálculos de doble precisión a una velocidad de 2 flops por ciclo.

$$2349,568[GFlops] = 13 \cdot 64 \cdot 0,706 \cdot 2 \cdot 2$$

EVGA GeForce GTX 580

Las aceleradoras EVGA GeForce GTX 580 (GTX580) poseen 512 núcleos. Estos núcleos se organizan en 16 SM que funcionan a una frecuencia 1.544 GHz. Cada SM posee 32 núcleos, todos los cuales están disponibles para cálculos de punto flotante de doble precisión a una velocidad de 2 flops por cada 8 ciclos de reloj.

$$396[GFlops] = 16 \cdot 32 \cdot 1{,}544 \cdot \frac{2}{8} \cdot 2$$

Teniendo en cuenta la potencia teórica de las plataformas y los resultados obtenidos no se obtiene una buena relación de performance entre las distintas plataformas. Esto puede deberse a que se necesite optimizar en mayor medida el código, lo que implica un mayor esfuerzo a nivel de implementación que hay que analizar. Por otro lado, también se debería hacer un estudio para obtener la potencia pico efectiva de cálculo para contrastar los resultados y tener así un mejor análisis, ya que como se sabe, la potencia pico teórica a nivel práctico es imposible de alcanzar.

8.5. Comunicación de datos

8.5.1. Ancho de banda de GPU

En general se tiene dos tipos, el ancho de banda teórico y el efectivo. El ancho de banda teórico se calcula teniendo en cuenta la frecuencia de reloj del hardware y la longitud de palabra de su interfaz siguiendo la fórmula

$$f \cdot (10^6)[hz] \cdot lb[b] \cdot \frac{1[GB]}{1000^3 \cdot 8[b]} = AB_{teorico}[GBps],$$

Donde f representa la frecuencia de reloj en MHz, lb la cantidad de líneas de bits, b la unidad de bits, B de bytes, GB de Gigabytes y $GBps$ de Gigabytes por segundo.

Los valores necesarios para efectuar este cálculo pueden obtenerse de la hoja de especificaciones de la GPU.

A continuación se detallan el ancho de banda teórico de las GPUs utilizadas en la Tabla 27.

Hardware	frecuencia [MHz]	líneas de interfaz [bits]	$AB_{teorico}$ [GBps]
GeForce GTX 580	2004	384 (GDDR5)	192.4
Tesla K20X	2600	384 (GDDR5)	249.6

Tabla 27: Ancho de banda teórico

Por otro lado, el ancho de banda efectivo se calcula en base a la cantidad de bytes leídos y la cantidad de bytes escritos sobre la memoria global por el kernel y el tiempo de procesamiento.

A continuación se detallan el ancho de banda efectivo de las GPUs en la Tabla 28 que fueron medidos a través de la herramienta otorgada por el SDK de CUDA (esta herramienta permite saber el máximo ancho de banda que se puede aspirar a obtener con los kernels desarrollados en las GPUs utilizadas) con el fin de comparar con el ancho de banda efectivo utilizado por el kernel producto matriz-vector 3D.

Hardware	$AB_{efectivo}$ [GBps]
GeForce GTX 580	163.70
Tesla K20X	195.28

Tabla 28: Ancho de banda efectivo

El ancho de banda efectivo representa una medida real, mientras que el ancho de banda teórico supone condiciones óptimas de ejecución que a fines prácticos es imposible de alcanzar.

8.5.2. Cálculo de rendimiento de ancho de banda de un kernel

Luego para calcular el ancho de banda efectivo utilizado por un kernel se utiliza la siguiente fórmula

$$\frac{[(BL + BE) \cdot 1024^{-3}]}{tiempo\_total},$$

Donde BL es la cantidad de bytes leidos por el kernel y BE es la cantidad de bytes escritos por el kernel, tiempo_total es el tiempo de procesamiento del kernel y el factor 1024^{-3} es un factor de escala a GBps.

Para el kernel producto matriz-vector 3D se necesitan $(Nx+2)\cdot(Ny+2)\cdot(Nz+2)$ lecturas sobre la memoria global. Entonces se tiene $(Nx+2)\cdot(Ny+2)\cdot(Nz+2)\cdot B$ lecturas, donde B hace referencia al alineamiento en bytes requerido por el tipo de dato, este caso 8 para doble precisión. Asimismo la cantidad de escrituras necesaria es la de lecturas sin considerar los nodos en el borde, por lo tanto

$$BL + BE = ((Nx+2)\cdot(Ny+2)\cdot(Nz+2) + (Nx)\cdot(Ny)\cdot(Nz))\cdot 8,$$

Finalmente se presenta la Tabla 29 con los rendimientos obtenidos.

Se observa en general un aumento de rendimiento del ancho de banda a medida que crece el tamaño de los subdominios.

Dimensión del subdominio	GeForce GTX 580 [GBps]	Tesla K20X [GBps]
32x64x64	72.14 (44.07 %)	98.47 (50.43 %)
64x128x128	95.73 (58.48 %)	117.32 (60.08 %)
100x200x200	101.05 (61.73 %)	123.61 (63.30 %)

Tabla 29: Rendimiento de ancho de banda del kernel producto matriz-vector 3D

8.5.3. Buffers auxiliares vs. sin buffers auxiliares

A continuación se presenta los resultados de las mediciones de tiempo promedio de la función intercambio_superposición del PCG con superposición de subdominios mediante el uso de buffers auxiliares alojados en la memoria del host para transmitir datos de GPU a GPU y el uso de la tecnología GPUDirect2.

Dimensión del dominio - nro. de capas	con buffer auxiliar [s]	sin buffer auxiliar [s]
2048x2048 sup 20	$0{,}2282 \times 10^{-3}$	$0{,}2038 \times 10^{-3}$
2048x2048 sup 50	$0{,}4478 \times 10^{-3}$	$0{,}3627 \times 10^{-3}$
2048x2048 sup 100	$0{,}8521 \times 10^{-3}$	$0{,}6114 \times 10^{-3}$

Tabla 30: GeForce GTX 580 - intercambio sup

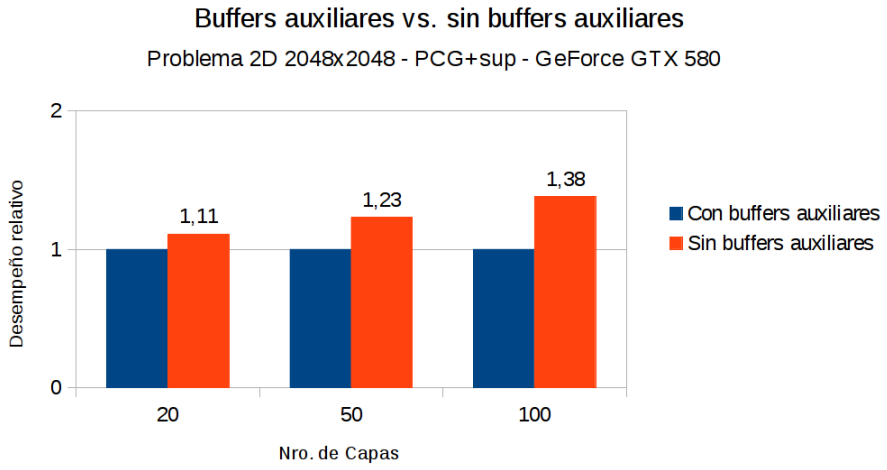


Figura 7: Desempeño relativo de transferencia de datos de GPU a GPU en 2D

En base a estos los resultados obtenidos (ver Tabla 30 y Tabla 31), el uso de la tecnología GPUDirect2 ha resultado ser una mejora, donde el desempeño para problemas en 3D ha sido más pronunciado que en 2D. Se puede decir que para mayor cantidad de datos a transmitir mayor son los beneficios de utilizar esta tecnología.

Dimensión del dominio - nro. de capas	con buffer auxiliar [s]	sin buffer auxiliar [s]
200x200x200 sup 10	$0{,}1241 \times 10^{-2}$	$0{,}0909 \times 10^{-2}$
200x200x200 sup 20	$0{,}3548 \times 10^{-2}$	$0{,}2049 \times 10^{-2}$
200x200x200 sup 25	$0{,}4717 \times 10^{-2}$	$0{,}2516 \times 10^{-2}$

Tabla 31: GeForce GTX 580 - intercambio sup

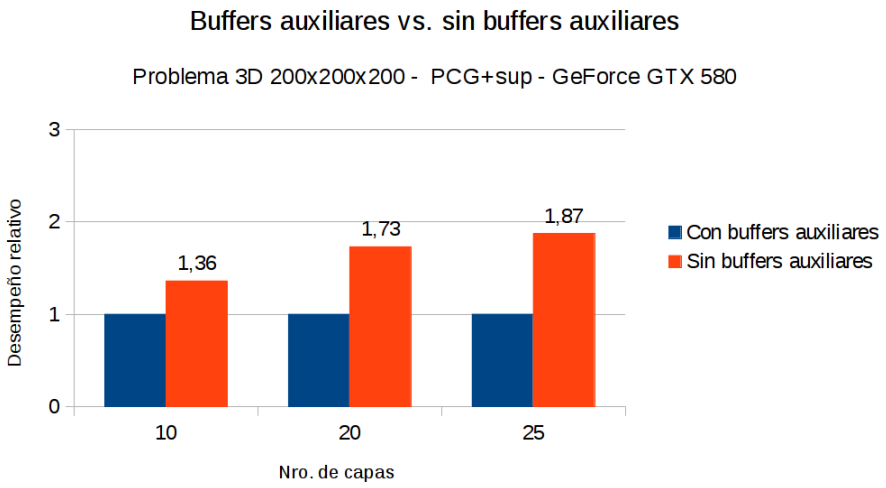


Figura 8: Desempeño relativo de transferencia de datos de GPU a GPU en 3D

9. CONCLUSIONES

Se implementó algoritmos para resolver el problema de Poisson para arquitecturas multi-GPU aprovechando las capacidades de las GPUs en el cálculo de las operaciones más costosas en tiempo, como son el producto matriz por vector y el paso de precondicionamiento y se obtuvieron soluciones acordes a resultados conocidos. Los resultados de las pruebas han demostrado que el método mas rápido ha sido el PCG con superposición de dominios para la versión GPU. Sin embargo, presenta una desventaja aún, que es el consumo de memoria, ya que aunque el precondicionador por DST reduce en gran medida el número de iteraciones requiere más memoria que los otros algoritmos ya que es necesario realizar la extensión impar de la entrada para aplicar el precondicionador. La optimización de memoria y cómputo de esta operación es un punto clave para atacar en trabajos futuros.

Como se ha visto, si se aumenta la región de superposición la cantidad de iteraciones necesarias para alcanzar la solución disminuye. Este resultado esta relacionado con la naturaleza del precondicionador ASM debido a que a medida se incrementa la banda en que se superponen los subdominios el sistema queda mejor precondicionado por lo que PCG halla la solución en menos iteraciones. Por otro lado el precondicionador por FFT no otorgó mejoras con respecto al precondicionador por DST a pesar de que no demanda tanta memoria y es más rápida su aplicación. Esto se debe a que no es tan efectivo como la DST para reducir el número de condición del sistema, por lo tanto la velocidad de convergencia del PCG es mas lenta. Con respecto a la comunicación de datos, las transferencias de halos y regiones de superposición se han logrado acelerar mediante el uso de la tecnología GPUDirect2 con respecto a usar buffers auxiliares, obteniendo mejores rendimientos al incrementar la cantidad de datos a transmitir entre las GPUs.

AGRADECIMIENTOS

- Consejo Nacional de Investigaciones Científicas y Técnicas (proyectos CONICET PIP 112-20111-00978, y 11220150100588CO).
- Agencia Nacional de Promoción Científica y Tecnológica, ANPCyT, (subsidios PICT-E-2014-0191, PICT-2014-0191, PICT-2015-2904);
- Universidad Nacional del Litoral, Argentina (subsidios CAI+D-501-201101-00233-LI);
- Consejo Europeo de Investigación (ERC) Técnicas de Mecánica Computacional en Tiempo Real para Problemas de Multi-Fluidos (REALTIME, Referencia: ERC-2009-AdG).
- Agencia Santafesina de Ciencia, Tecnología e Innovación (subsidios ASACTEI-010-18-2014);

Este trabajo utilizó recursos computacionales del CCAD de la Universidad Nacional de Córdoba (http://ccad.unc.edu.ar/), en particular el Cluster Mendieta, el cual forma parte del SNCAD del MinCyT de la República Argentina.

Los autores hacen un amplio uso de *Software Libre* como GNU/Linux OS, compiladores GCC/G++, Octave, y *Open Source* software como VTK, ParaView, LaTeX entre muchos otros.

REFERENCIAS

Burden R. y Faires J. *Análisis numérico*. International Thomson Editores, 2002. ISBN 9789706861344.

Buzbee B.L., Golub G.H., y Nielson C.W. On direct methods for solving poisson's equations. *SIAM Journal on Numerical Analysis*, 7(4):627–656, 1970. doi:10.1137/0707049.

Dryja M. *Some Domain Decomposition Algorithms for Elliptic Problems (Classic Reprint)*. Forgotten Books, 2016. ISBN 9781332961429.

Forum M.P.I. *MPI: A Message Passing Interface Standard ; Version 2.2*. High-Performance Computing Center, 2009.

Gander M.J. Schwarz methods over the course of time. *ETNA. Electronic Transactions on Numerical Analysis [electronic only]*, 31:228–255, 2008.

Golub G.H. y Van Loan C.F. *Matrix Computations (3rd Ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996. ISBN 0-8018-5414-8.

Sanders J. y Kandrot E. *CUDA by Example: An Introduction to General-Purpose GPU Programming, Portable Documents*. Pearson Education, 2010. ISBN 9780132180139.

Storti M.A., Paz R.R., Dalcin L.D., Costarelli S.D., y Idelsohn S.R. A fft preconditioning technique for the solution of incompressible flow on gpus. *Computers and Fluids*, 74(Complete):44–57, 2013. doi:10.1016/j.compfluid.2012.12.019.

Toselli A. y Widlund O. *Domain Decomposition Methods - Algorithms and Theory*. Springer Series in Computational Mathematics. Springer Berlin Heidelberg, 2006. ISBN 9783540266624.

Van Loan C. *Computational Frameworks for the Fast Fourier Transform*. Frontiers in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 1992. ISBN 9781611970999.