Métodos Iterativos para Sistemas Lineales Grandes[1]

G. Figueroa M.[2] y L.E. Carrera R.[3]
Escuela de Matemática
Instituto Tecnológico de Costa Rica

30 de abril de 2012

[1]Investigación financiada por la VIE, código: # 5402-1701-0101
[2]email: gfigueroa@itcr.ac.cr
[3]email: lecarrera@itcr.ac.cr

Resumen

El poder de cálculo incorporado en las unidades de procesamiento gráfico (GPU) de las nuevas tarjetas de video *Nvidia* ha permitido aplicar el procesamiento en paralelo a la solución de problemas que requieren gran cantidad de cálculos y que no están ligados al ambiente gráfico, para el cual fueron originalmente creadas. En este proyecto se aplica esta tecnología a la solución de sistemas de ecuaciones lineales de gran tamaño.

Abstract

The computing power built into the graphics processing units (GPU) from Nvidia new video cards has permitted parallel processing to solve problems that require computational power and are not linked to the graphical environment for that were originally created. This project apply this technology to the solution of large linear systems.

Palabras claves: sistemas de ecuaciones lineales, matrices ralas, métodos numéricos, programación paralela, CUDA.

keywords: lineal system equations, sparse matrices, numerical methods, parallel programming, CUDA.

Índice general

1. Sobre el proyecto **4**

1.1. Introducción 4

1.2. Problema a investigar 4

1.3. Objetivos 4

1.4. Marco teórico 5

1.5. Metodología 5

1.6. Resumen de resultados 5

2. Preliminares **6**

2.1. Normas vectoriales 6

2.2. Normas matriciales 7

2.3. Matrices especiales 8

3. Métodos iterativos **10**

3.1. Métodos de Jacobi y Gauss-Seidel 11

3.2. Método del gradiente conjugado 12

3.2.1. Interpretación geométrica 13

4. Implementación en CUDA **16**

4.1. Introducción 16

4.2. Matrices ralas (sparse matrices) 17

4.2.1. Transpuesta de una matriz rala 17

4.2.2. Producto matriz vector 18

4.3. Gauss-Seidel en CUDA 19

4.4. Gradiente conjugado en CUDA y OpenMP 20

4.5. Gradiente conjugado normalizado 23

4.6. Resultados 25

4.7. Conclusiones 26

Índice de figuras

3.1. Función cuadrática. 14

4.1. Estructura del modelo de programación. 16

Índice de cuadros

4.1. Formato MSR. 17
4.2. Matrices de prueba. 25
4.3. Gradiente conjugado en OpenMP con diferente número de procesadores. 25
4.4. Comparación Gauss-Seidel y gradiente conjugado. 26
4.5. Comparación gradiente conjugado normalizado y Gauss-Seidel. 26

Capítulo 1

Sobre el proyecto

1.1. Introducción

La ciencia enfrenta, en sus diversas disciplinas, un cambio mayúsculo, se maneja una gran cantidad de información que debe ser almacenada, depurada, analizada y compartida. Hace mil años, la ciencia era empírica: describía los fenómenos naturales. En los últimos siglos pasó a ser teórica, al usar modelos y generalizaciones que permiten simular fenómenos complejos. Hoy, el cuarto paradigma [12] implica la exploración de datos y en este nuevo paradigma el computador juega un papel fundamental en la simulación, visualización y análisis. La razón de esto es el creciente desarrollo que se ha suscitado en las tecnologías computacionales, cada vez con mayor capacidad y velocidad de cálculo, lo cual ha permitido a los investigadores crear modelos matemáticos de gran exactitud y magnitud que le permiten interpretar, simular y optimizar fenómenos complejos.

El objetivo del proyecto es investigar la posibilidad de usar procesamiento masivo en paralelo para la solución de sistemas de ecuaciones lineales de gran tamaño.

1.2. Problema a investigar

Dada una matriz A de tamaño $n \times n$ y un vector $\bar{b} \in \mathbb{R}^n$ se quiere determinar el vector $\bar{x} \in \mathbb{R}^n$ tal que $A\,\bar{x} = \bar{b}$.

Aquí, A es una matriz esparcida de gran tamaño con millones de entradas no cero.

1.3. Objetivos

1. *Objetivo general*

   Diseñar e implementar un método en paralelo para la solución de sistemas de ecuaciones lineales.

2. *Objetivos específicos*

   a) Investigar sobre la existencias de métodos iterativos para la solución de sistemas lineales.

   b) Analizar cual de los métodos iterativos analizados es suceptible de ser paralelizado.

   c) Diseñar un algoritmo paralelo para dicho método iterativo.

   d) Implementar usando programación paralela dicho algoritmo.

   e) Probar y depurar el algoritmo desarrollado sobre algunos sistemas lineales.

1.4. Marco teórico

Muchas procesos en ingeniería se modelan por medio de ecuaciones diferenciales en derivadas parciales ([7], [18], [6]) algunas de las cuales solo pueden ser resueltas por medio de técnicas numéricas como diferencias finitas ([16], [14], [13]) o elementos finitos ([11], [2], [17]). Para esto se discretiza el dominio mediante un mallado que al ser evaluado produce un sistema de ecuaciones lineales caracterizado por ser de gran tamaño y ralo.

Existe una gran variedad de métodos iterativos para resolver este tipo de sistemas lineales ([9], [8]), desde el tradicional algoritmo de eliminación gaussiana, pasando por los métodos de Jacobi, Gauss-Seidel y los métodos de sobrerelajación sucesiva (SOR) ([3]) hasta algunos más especializados como los llamados métodos de proyección ([1], [15]) que incluyen métodos como el de residuo mínimo generalizado (GMRES) y el del gradiente conjugado. Algunos de estos métodos pueden ser aplicados con exito a la solución de sistemas lineales de gran tamaño. En el proyecto se explora la posibilidad de usar procesamiento masivo en paralelo para la solución de este tipo de sistemas lineales, en particular, se aplica el poder de cálculo de los procesadores gráficos (GPU). Estas unidades de proceso gráfico están presentes en las tarjetas gráficas ([22]) para PC y en las consolas de videojuegos y tienen una arquitecturas muy sofisticadas que explota la idea del paralelismo de una forma innovadora y poco convencional ([19], [20]). Con múltiples procesadores especializados y un extraordinario ancho de banda son capaces de mantener velocidades de cálculo por encima de las CPUs más avanzadas.

Esta tecnología actualmente esta siendo aplicada principalmente a la representaciín de escenas tridimensionales en tiempo real haciendo que los entornos gráficos sean cada vez más reales y complejos, pero sus capacidades de cálculo en paralelo hacen que sean muy adecuada para ser aplicados a otros campos como: optimización combinatoria, análisis de datos, visualizacion de la información, etc ([21], [19]).

1.5. Metodología

Se realizó una investigación bibliográfica sobre los diferentes métodos iterativos existentes para la solución de sistemas de ecuaciones lineales de gran tamaño. De estos se eligen los que se adaptan mejor al paradigma de programación en paralelo y se procede a diseñar los correspondientes algoritmos, los cuales al final fueron implementados en OpenMP y/o CUDA.

Para probar los algoritmos diseñados se realizó una búsqueda en internet de bibliotecas de matrices esparcidas de grandes dimensiones, al final se usaron dos colecciones de este tipo de matrices:

- matrices esparcidas de la Universidad de Florida ([23]).
- matrices esparcidas `Matrix Market` ([24]).

A lo largo del proyecto los investigadores realizaron reuniones semanales para discutir sobre el avance de la investigación, definir nuevas estrategias y líneas de investigación.

1.6. Resumen de resultados

Se diseñaron tres algoritmos en paralelo para los métodos: Gauss-Seidel, Gradiente conjugado y Gradiente conjugado normalizado, capaces de resolver sistemas esparcidos de ecuaciones lineales con millones de entradas no cero. Para la implementación de estos algoritmos se usaron los paradigmas OpenMP y CUDA. Estos algoritmos fueron puestos a prueba con sistemas de ecuaciones lineales obtenidos de bibliotecas especializadas ([23], [24]).

Capítulo 2

Preliminares

2.1. Normas vectoriales

Este capítulo resume algunos de los resultados necesarios para introducir los métodos iterativos.

Las normas nos permiten cuantificar el error en un proceso iterativo y de aquí determinar la convergencia de una sucesión de aproximaciones.

Definición 2.1 *Una norma vectorial en* $\mathbb{R}^n$ *es una función* $\|\cdot\| : \mathbb{R}^n \longrightarrow \mathbb{R}$ *que satisface las siguientes propiedades:*

1. $\|\bar{x}\| \geq 0$, $\forall \bar{x} \in \mathbb{R}^n$.
2. $\|\bar{x}\| = 0$ *si y solo si* $\bar{x} = \bar{0}$.
3. $\|\alpha\bar{x}\| = |\alpha| \, \|\bar{x}\|$, $\forall \alpha \in \mathbb{R}$ *y* $\forall \bar{x} \in \mathbb{R}^n$.
4. $\|\bar{x} + \bar{y}\| \leq \|\bar{x}\| + \|\bar{y}\|$, $\forall \bar{x}, \bar{y} \in \mathbb{R}^n$.

En general se utilizan las normas $\|\cdot\|_2$ y $\|\cdot\|_\infty$.

Definición 2.2 *Para* $\bar{x} = (x_1, x_2, \ldots, x_n)^t \in \mathbb{R}^n$ *definimos la norma* $\|\cdot\|_2$ *y* $\|\cdot\|_\infty$ *por*

$$\|\bar{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \qquad\qquad \|\bar{x}\|_\infty = \max_{1 \leq i \leq n} |x_i|$$

Observación: La norma $\|\cdot\|_2$ se conoce como norma euclideana, debido a que representa la noción usual de distancia respecto al origen.

A partir de una norma vectorial se construye el concepto de distancia.

Definición 2.3 *Sean* $\bar{x} = (x_1, x_2, \ldots, x_n)^t \in \mathbb{R}^n$ *y* $\bar{y} = (y_1, y_2, \ldots, x_n)^t \in \mathbb{R}^n$. *Las distancias* l_2 *y* l_∞ *entre* $\bar{x}$ *e* $\bar{y}$ *se definen por:*

$$l_2 = \|\bar{x} - \bar{y}\|_2 \qquad\qquad l_\infty = \|\bar{x} - \bar{y}\|_\infty$$

Ejemplo 2.1

Si $\bar{x} = (1, -2, 3) \in \mathbb{R}^3$ e $\bar{y} = (0{,}993, -1{,}781, 3{,}012) \in \mathbb{R}^3$ entonces

$$l_2 = \|\bar{x} - \bar{y}\|_2 = 0{,}21944 \qquad\qquad l_\infty = \|\bar{x} - \bar{y}\|_\infty = 0{,}219$$

La equivalencia de normas es un concepto importante en la convergencia de sucesiones de aproximaciones.

Teorema 2.1 *Las normas* $\|\cdot\|_2$ *y* $\|\cdot\|_\infty$ *son equivalentes en el siguiente sentido:*

$$\|\bar{x}\|_\infty \leq \|\bar{x}\|_2 \leq \sqrt{n}\,\|\bar{x}\|_\infty, \forall \bar{x} \in \mathbb{R}^n$$

Observación: Este resultado implica que la convergencia de una sucesión en $\mathbb{R}^n$ es independiente de la norma, es decir, si la sucesión $\left\{\bar{x}^{(i)}\right\}_{i=1}^{\infty}$ converge a $\bar{x}$ con la norma $\|\cdot\|_2$ también converge a $\bar{x}$ con la norma $\|\cdot\|_\infty$ y viceversa. En general todas las normas en $\mathbb{R}^n$ son equivalentes en este sentido.

2.2. Normas matriciales

De forma semejante a como se hizo para $\mathbb{R}^n$ se puede definir el concepto de norma para matriz cuadrada.

Definición 2.4 *Una norma matricial sobre el conjunto de las matrices* $n \times n$ *(*$M_n(\mathbb{R}^n)$*) es una función* $\|\cdot\| : M_n(\mathbb{R}) \longrightarrow \mathbb{R}$ *que satisface las siguientes propiedades.*

1. $\|A\| \geq 0$, $\forall A \in M_n(\mathbb{R})$.
2. $\|A\| = 0$ *si y solo si* $a_{i,j} = 0$, $1 \leq i, j \leq n$.
3. $\|\alpha A\| = |\alpha|\,\|A\|$, $\forall \alpha \in \mathbb{R}$ *y* $\forall A \in M_n(\mathbb{R})$.
4. $\|A + B\| \leq \|A\| + \|B\|$, $\forall A, B \in M_n(\mathbb{R})$.
5. $\|AB\| \leq \|A\|\,\|B\|$, $\forall A, B \in M_n(\mathbb{R})$

Observación: la distancia entre dos matrices A y B de $M_n(\mathbb{R})$ respecto a la norma $\|\cdot\|$ es $\|A - B\|$.

Definición 2.5 *Para* $A \in M_n(\mathbb{R})$ *se define:*

- *la norma* $\|\cdot\|_\infty$

$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^{n} |a_{ij}|$$

- *la norma euclídea o de Frobenius*

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2} = \sqrt{tr(A \times A)}$$

- *la norma* $\|\cdot\|_2$

$$\|A\|_2 = \sqrt{\rho(A^t \times A)}$$

donde $\rho(A)$ *es el radio espectral de* A.

2.3. Matrices especiales

En algunos problemas aplicados surgen con frecuencia matrices con características muy particulares que garantizan la convergencia de los métodos iterativos.

Definición 2.6 *Una matriz $A \in M_n(\mathbb{R})$, es estrictamente dominante en sentido diagional si*

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \qquad\qquad i = 1, 2, \ldots, n$$

Observación: Este tipo de matrices aparecen al determinar los coeficientes de los trazadores cúbicos y al aplicar algunas técnicas numéricas a la solución de ecuaciones diferenciales en derivadas parciales.

Ejemplo 2.2

La matriz A definida por

$$A = \begin{pmatrix} 1 & 1/2 & 1/3 \\ 1/4 & 1 & 1/5 \\ 1/6 & 1/7 & 1 \end{pmatrix}$$

es estrictamente dominante en sentido diagonal.

Teorema 2.2 *Sea $A \in M_n(\mathbb{R})$ estrictamente dominante en sentido diagonal, entonces A es no singular.*

Observación: este resultado garantiza que podemos aplicar el método de eliminación gaussiana sin intercambio de filas ni columnas para obtener la solución única del sistema lineal $A\bar{x} = \bar{b}$ y además los cálculos son estables respecto al crecimiento de los errores de redondeo [3].

Definición 2.7 *Una matriz $A \in M_n(\mathbb{R})$ es definida positiva si $\bar{x}^t A\bar{x} > 0$ para todo vector columna $\bar{x} \in \mathbb{R}^n$ con $\bar{x} \neq 0$.*

Observación: Note que

$$\begin{aligned} \bar{x}^t A\bar{x} &= (x_1, x_2, \ldots, x_n) \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\ &= (x_1, x_2, \ldots, x_n) \begin{pmatrix} \sum_{j=1}^n a_{1j}x_j \\ \sum_{j=1}^n a_{2j}x_j \\ \vdots \\ \sum_{j=1}^n a_{nj}x_j \end{pmatrix} \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j \end{aligned}$$

Ejemplo 2.3

La matriz A definida por

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 2 & 1 \\ 0 & -1 & 3 \end{pmatrix}$$

es definida positiva, pues si $\bar{x} = (x_1, x_2, x_3)^t \neq 0$

$$\bar{x}^t A \bar{x} = x_1^2 + 2x_2^2 + 3x_3^2 > 0$$

Observación: si además, A es simétrica decimos que es una matriz simétrica definida positiva, en este caso, sus valores propios son reales positivos.

Teorema 2.3 *Sea $A \in M_n(\mathbb{R})$ definida positiva entonces*

- *A es no singular.*
- *$a_{ij} > 0$, para $i = 1, 2, \ldots, n$.*
- *$(a_{ij})^2 < a_{ii} a_{jj}$, para cada $i \neq j$.*

El ejemplo anterior evidencia la dificultad de usar la definición para determinar si una matriz es definida positiva. Los siguientes resultados tratan de facilitar este cálculo.

Definición 2.8 *Dada una matriz $A \in M_n(\mathbb{R})$, la k-ésima primera submatriz principal de A está dada por*

$$A_k = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{k1} & a_{k2} & \ldots & a_{kn} \end{pmatrix}$$

Teorema 2.4 *Una matriz $A \in M_n(\mathbb{R})$ es definida positiva si y solo si $|A_k| > 0$ para $k = 1, 2, \ldots, n$*

Ejemplo 2.4

La matriz A definida por

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 2 & 1 \\ 0 & -1 & 3 \end{pmatrix}$$

es definida positiva, pues

$$|A_1| = 1 > 0 \qquad |A_2| = \begin{vmatrix} 1 & -1 \\ 1 & 2 \end{vmatrix} = 3 \qquad A_3 = \begin{vmatrix} 1 & -1 & 0 \\ 1 & 2 & 1 \\ 0 & -1 & 3 \end{vmatrix} = 10$$

Capítulo 3

Métodos iterativos

Los métodos iterativos rara vez se usan para resolver sistemas lineales de dimensión pequeña ya que el tiempo requerido para conseguir una exactitud satisfactoria rebasa el requerido por los métodos directos, como eliminación gaussiana o factorización LU. Sin embargo, en sistemas lineales de gran tamaño con un alto porcentaje de ceros son eficientes. Este tipo de sistemas lineales aparecen en análisis de circuitos y al resolver problemas de valores en la frontera para ecuaciones en derivadas parciales.

Consideramos únicamente métodos iterativos cuya sucesión de aproximaciones son definidas por una relación recursiva de la forma $\bar{x}^{(k)} = F(\bar{x}^{k-1})$, es decir, la k-ésima aproximación es función únicamente de la $(k-1)$-ésima y no de las anteriores.

Definición 3.1 *Sea $A \in M_n(\mathbb{R})$ no singular. El par de matrices (M, N) con M no singular que satisfacen $A = M - N$ se llama una descomposición regular de A. El método iterativo basado en la descomposición regular (M, N) de A se define por:*

$$\begin{cases} \bar{x}^{(0)} & \in & \mathbb{R}^n \\ M\bar{x}^{(k)} & = & N\bar{x}^{(k-1)} + \bar{b} \qquad \forall k \geq 1 \end{cases} \tag{3.1}$$

Observación: el método iterativo (3.1) resuelve el sistema lineal $A\bar{x} = \bar{b}$ resolviendo una sucesión de de sistemas lineales de la forma $M \widetilde{x} = \widetilde{b}$, por lo que M debe ser una matriz cuya inversa sea más fácil de calcular que la inversa de A.

Si la sucesión $\bar{x}^{(k)}$ converge a un límite $\bar{x}$ cuando $k \to +\infty$, de la relación (3.1) tenemos que

$$\lim_{k\to+\infty} M\bar{x}^{(k)} = \lim_{k\to+\infty} N\bar{x}^{(k-1)} + \bar{b} \Rightarrow M\bar{x} = N\bar{x} + b \Rightarrow (M-N)\bar{x} = A\bar{x} = \bar{b}$$

En consecuencia, si la sucesión de soluciones aproximadas $\left\{\bar{x}^{(n)}\right\}_{n=1}^{\infty}$ converge a un límite este es necesariamente la solución del sistema lineal.

Definición 3.2 *Un método iterativo se dice convergente si para cualquier escogencia de la aproximación inicial $\bar{x}^{(0)} \in \mathbb{R}^n$, la sucesión de soluciones aproximaciones $\bar{x}^{(k)}$ converge a la solución exacta $\bar{x}$.*

Definición 3.3 *Llamamos al vector $\bar{r}_k = \bar{b} - A\bar{x}$ (respectivamente $\bar{e}_k = \bar{x}^{(k)} - \bar{x}$) residuo (respectivamente error) de la k-ésima iteración.*

Obviamente un método iterativo converge si y solo si $\bar{e}_k$ converge a 0, lo cual es equivalente a que $\bar{r}_k = A\bar{e}_k$ converja a $\bar{0}$. En general, no conocemos $\bar{e}_k$ porque $\bar{x}$ no se conoce. Sin embargo, es fácil calcular $\bar{r}_k$ por lo que en la práctica la convergencia se detecta en el residuo.

La relación (3.1) se puede escribir equivalentemente como

$$\bar{x}^{(k)} = M^{-1}N\bar{x}^{(k-1)} + M^{-1}\bar{b}$$

A la matriz $M^{-1}N$ se le llama matriz de iteración del método iterativo. El siguiente teorema muestra que la convergencia del método iterativo está ligada al radio espectral de $M^{-1}N$.

Teorema 3.1 *El método iterativo definido por (3.1) converge si y solo si el radio espectral $\rho(\cdot)$ de $M^{-1}N$ satisface*

$$\rho\left(M^{-1}N\right) < 1$$

Los métodos iterativos pueden requerir un gran número de iteraciones para converger. Por lo tanto, uno podría pensar que la acumulación de errores de redondeo durante el proceso iterativo puede destruir la convergencia o peor aún, hacer que converja a soluciones equivocadas. Afortunadamente, este no es el caso, como se muestra en el siguiente resulto.

Teorema 3.2 *Considere una descomposición $A = M - N$ con A y M no singulares. Sea $\bar{b} \in \mathbb{R}^n$ y sea $\bar{x} \in \mathbb{R}^n$ la solución del sistema lineal $A\bar{x} = \bar{b}$. Suponiendo que en la iteración k se produce un error $\bar{\epsilon}_k \in \mathbb{R}^n$, lo que significa que $\bar{x}^{(k)}$ no esta dado exactamente por la relación (3.1) sino por*

$$\bar{x}^k = M^{-1}N\bar{x}^{(k-1)} + M^{-1}\bar{b} + \bar{\epsilon}_k.$$

Suponiendo que $\rho\left(M^{-1}N\right) < 1$ y que existe una norma vectorial y una constante positiva ϵ tal que para todo $k \geq 0$ se tiene $\|\epsilon_k\| \leq \epsilon$. Entonces existe una constante K, la cual depende de $M^{-1}N$ pero no de $\bar{\epsilon}$ tal que

$$\limsup_{k \to +\infty} \left\| \bar{x}^{(k)} - \bar{x} \right\| \leq K\bar{\epsilon}.$$

3.1. Métodos de Jacobi y Gauss-Seidel

Estos métodos son ampliamente utilizados debido a su simplicidad.

Definición 3.4 *El método de Jacobi es el método iterativo definido por la descomposición $M = D$ y $N = D - A$, donde la matriz diagonal D esta dada por $D = diag(a_{11}, a_{22}, \ldots, a_{nn})$. La matriz de iteración para este método es $J = M^{-1}N = I - D^{-1}A$.*

Observación: si la matriz diagonal D es no singular el método de Jacobi es bien definido.

Para una matriz $A \in M_n(\mathbb{R})$ considere la descomposición $A = D - E - F$ donde D es la matriz diagonal, $-E$ es la parte inferior de A y $-F$ la parte superior de A. Es decir,

$$\begin{cases} d_{i,j} &= a_{i,j}\delta_{i,j} \\ e_{i,j} &= -a_{i,j} \text{ si } i > j, \text{ y } 0 \text{ en los otros casos} \\ f_{i,j} &= -a_{i,j} \text{ si } i < j, \text{ y } 0 \text{ en los otros casos} \end{cases}$$

Definición 3.5 *El método de Gauss-Seidel es el método iterativo definido por la descomposición $M = D - E$ y $N = F$. La matriz de iteración del método esta dada por $G = M^{-1}N = (D - E)^{-1} F$.*

Observación: el método de Gauss-Seidel esta bien definido si la matiz $D - E$ es no singular o equivalentemente si D es no singular. La matriz $D - E$ es fácilmente invertible pues es triangular.

En el método de Jacobi se calcula $\bar{x}^{(k+1)}$ en términos de todas las entradas de $\bar{x}^{(k)}$

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(-a_{i,1}x_1^{(k)} - \cdots - a_{i,i-1}x_{i-1}^{(k)} - a_{i,i+1}x_{i+1}^{(k)} - \cdots - a_{i,n}x_n^{(k)} + b_i \right)$$

En el método de Gauss-Seidel se usa la información ya calculada en las $i-1$ primeras entradas de $\bar{x}^{(k+1)}$, osea

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(-a_{i,1}x_1^{(k+1)} - \cdots - a_{i,i-1}x_{i-1}^{(k+1)} - a_{i,i+1}x_{i+1}^{(k)} - \cdots - a_{i,n}x_n^{(k)} + b_i \right)$$

Observación: desde un punto de vista práctico, se requieren dos vectores de tamaño n para almacenar $\bar{x}^{(k)}$ y $\bar{x}^{(k+1)}$ en el método de Jacobi, mientras que en el método de Gauss-Seidel las entradas de $\bar{x}^{(k+1)}$ sustituyen progresivamente las entradas de $\bar{x}^{(k)}$.

Algoritmo GAUSS-SEIDEL($\mathbf{a_{ij}}, \mathbf{b_i}, \mathbf{x^0}, \mathbf{tol}, \mathbf{n}, \mathbf{N}$)

$k \leftarrow 1$
mientras $(k \leq N)$
    para $i = 1$ **hasta** n:
        $x_i = \frac{-\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j^{(0)} + b_i}{a_{ii}}$
    si $\left\| x - x^{(0)} \right\| < tol$:
        devolver x
    sino:
        $k \leftarrow k + 1$
        para $i = 1$ **hasta** n:
            $x_i^{(0)} = x_i$
devolver 0

Teorema 3.3 *Si $A \in M_n(\mathbb{R})$ es una matriz estrictamente dominante en sentido diagonal, entonces para cualquier escogencia de la aproximación inicial $\bar{x}^{(0)}$, tanto el método de Jacobi como el de Gauss-Seidel convergen a la solución única del sistema lineal $A\bar{x} = \bar{b}$.*

3.2. Método del gradiente conjugado

El método del gradiente conjugado es uno de los métodos iterativos más utilizados para resolver sistemas de ecuaciones lineales de gran tamaño. Se aplica a sistemas lineales de la forma $A\bar{x} = \bar{b}$, donde $\bar{x}$ es un vector desconocido, $\bar{b}$ es un vector conocido y A es una matriz cuadrada simétrica y definida positiva.[1] Este tipo de sistemas lineales surgen al aplicar técnicas numéricas de gran importancia como los métodos de diferencias finitas o elementos finitos para resolver ecuaciones diferenciales en derivadas parciales.

Este método ha recibido mucha atención y ha sido ampliamente utilizado en años recientes, aunque los pioneros del método fueron Hestenes y Stiefel en 1952 [10]. El interés actual inicia a partir de que se plantea como un método iterativo, que es la forma en que se le usa con mayor frecuencia en la actualidad.

La idea básica en que descansa el método del gradiente conjugado consiste en construir una base de vectores ortogonales y utilizarla para realizar la búsqueda de la solución en forma más eficiente. Tal forma de proceder generalmente no sería aconsejable porque la construcción de una base ortogonal utilizando

[1] Existen variantes del método para resolver sistemas lineales cuya matriz no sea simétrica ni definida positiva.

el procediemiento de Gramm-Schmidt requiere, al seleccionar cada nuevo elemento de la base, asegurar su ortogonalidad con respecto a cada uno de los vectores construidos previamente. La gran ventaja del método del gradiente conjugado radica en que cuando se utiliza este procedimiento, basta con asegurar la ortogonalidad de un nuevo miembro con respecto al último que se ha construido, para que automáticamente esta condición se cumpla con respecto a todos los anteriores.

Definición 3.6 *El método iterativo conocido como el método del gradiente se define por las descomposición regular*

$$M = \frac{I_n}{\alpha} \qquad\qquad N = \left(\frac{I_n}{\alpha} - A\right)$$

donde α es un parámetro real diferente de cero.

Observación: esto quiere decir que el método del gradiente se define por la sucesión

$$\begin{cases} \bar{x}^{(0)} & \in & \mathbb{R}^n \\ \bar{x}^{(k)} & = & \bar{x}^{(k-1)} + \alpha\left(b - A\bar{x}^{(k)}\right) \qquad \forall k \geq 1 \end{cases}$$

Teorema 3.4 *Sea $A \in M_n(\mathbb{R})$ una matriz con valores propios $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.*

1. *Si $\lambda_1 \leq 0 \leq \lambda_n$, el método del gradiente no converge para cualquier valor de α.*
2. *Si $0 < \lambda_1 \leq \ldots \leq \lambda_n$, el método del gradiente converge si y solo si $0 < \alpha < 1/\lambda_n$. En este caso, el valor óptimo para el parámetro α, el cual minimiza el radio espectral $\rho\left(M^{-1}N\right)$ es*

$$\alpha = \frac{2}{\lambda_1 + \lambda_n} \qquad\qquad \min_{\alpha} \rho\left(M^{-1}N\right) = \frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1} = \frac{cond_2(A) - 1}{cond_2(A) + 1}$$

donde

$$cond_2(A) = \|A\|_2 \left\|A^{-1}\right\|_2$$

Observación: si la matrix A es diagonalizable con valores propios $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$, entonces se tiene un resultado análogo a (*2.*) reemplazando α por $-\alpha$.

Algoritmo GRADIENTE CONJUGADO

```
1  x̄ ← 0̄; r̄ ← b̄; p̄ ← r̄; ρ_new ← ⟨r̄, r̄⟩
2  repetir
3      ρ_old ← ρ_new
4      z̄ ← A × p̄
5      α ← ⟨p̄, z̄⟩/ρ_old
6      x̄ ← x̄ + p̄/α
7      r̄ ← r̄ − z̄/α
8      ρ_new ← ⟨r̄, r̄⟩
9      p̄ ← r̄ + p̄ · ρ_new/ρ_old
10 hasta que ρ_new sea suficientemente pequeño.
```

3.2.1. Interpretación geométrica

En esta sección se justifica el por qué del nombre del método.

Definición 3.7 *Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función. El vector gradiente de f en el punto $\bar{x}$ esta dado por*

$$\nabla f(\bar{x}) = \left(\frac{\partial f}{\partial x_1}(\bar{x}), \ldots, \frac{\partial f}{\partial x_n}(\bar{x}) \right)^t \tag{3.2}$$

Considere el problema de minimizar la función cuadrática $f : \mathbb{R}^2 \to \mathbb{R}$ dada por

$$f(\bar{x}) = \frac{1}{2}\langle A\bar{x}, \bar{x}\rangle - \langle \bar{v}, \bar{x}\rangle = \frac{1}{2}\sum_{i,j=1}^{n} a_{ij}x_i y_j - \sum_{i=1}^{n} b_i x_i$$

donde $A \in M_n(\mathbb{R})$ es una matriz simétrica, $\bar{b} \in \mathbb{R}^n$. La función f tiene un mínimo en $\bar{x}_0$ si $f(\bar{x}) \geq f(\bar{x}_0)$ $\forall \bar{x} \in \mathbb{R}^n$. En la figura 3.1 se muestra la gráfica de la función cuadrática para

$$A = \begin{pmatrix} 4 & 1 \\ 1 & 4 \end{pmatrix}$$

cuyos valores propios son $\lambda_1 = 3$ y $\lambda_2 = 5$ y por lo tanto es simétrica definida positiva.

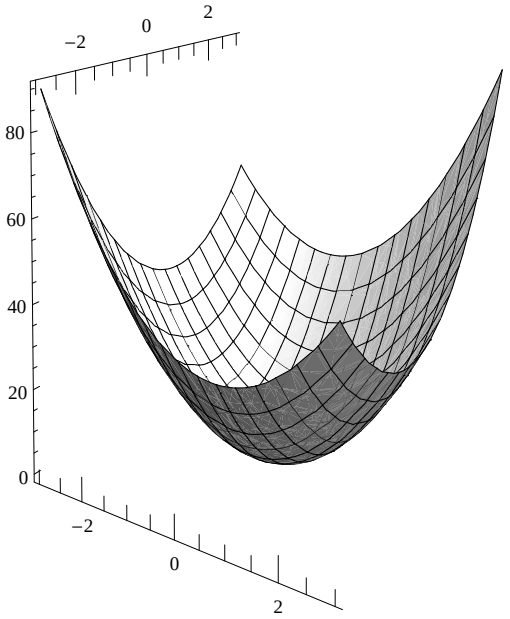

Figura 3.1: Función cuadrática.

Teorema 3.5 *El gradiente de la función f definida por (3.2) es $\nabla f(\bar{x}) = A\bar{x} - \bar{b}$. Aún más, si A es definida positiva, entonces f tiene un único mínimo en $\bar{x}_0$ que es la solución del sistema lineal $A\bar{x} = \bar{b}$.*

Teorema 3.6 *Sea $A \in M_n(\mathbb{R})$ una matriz simétrica y definida positiva y sea f la función definida por (3.2).*

1. *f alcanza un mínimo en $\bar{x} \in \mathbb{R}^n$ si y solo si $\nabla f(\bar{x}) = 0$.*
2. *Sea $\bar{x} \in \mathbb{R}^n$ tal que $\nabla f(\bar{x}) \neq 0$, entonces $\forall \alpha \in]0, 2/\rho(A)[$ tenemos que $f(\bar{x} - \alpha\nabla f(\bar{x})) < f(\bar{x})$*

Observación: de este teorema se infiere un método iterativo para mínimizar f. Construimos una sucesión $\{\bar{x}_n\}_{n=1}^{\infty}$ tal que la sucesión $\{f(\bar{x}_n)\}_{n=1}^{\infty}$ es decreciente

$$\bar{x}_{n+1} = \bar{x}_n - \alpha\nabla f(\bar{x}_n) = \bar{x}_n + \alpha\left(\bar{b} - A\bar{x}_n\right)$$

Y esta sucesión es exactamente el método del gradiente. Esto demuestra que resolver una sistema lineal $A\bar{x} = \bar{b}$ cuya matriz de coeficientes A es simétrica definida positiva es equivalente a minimizar una función cuadrática.

Capítulo 4

Implementación en CUDA

4.1. Introducción

Usaremos el procesamiento masivo en paralelo para resolver sistemas lineales, en particular, se aplica el poder de cálculo de los procesadores gráficos GPU's presentes en las tarjetas gráficas NVIDIA ([22]). Este tipo de unidades de procesamiento gráfico tienen una arquitectura muy sofisticada que explota la idea del paralelismo de una forma innovadora y poco convencional. Usando múltiples procesadores especializados y un extraordinario ancho de banda son capaces de mantener velocidades de cálculo por encima de las CPU's.

CUDA (Compute Unified Device Architecture) es un compilador junto con un conjunto de herramientas de desarrollo creadas por NVIDIA que permiten usar una variación del lenguaje de programación C para codificar algoritmos en las unidades de procesamiento gráfico GPU'S.

El modelo de programación CUDA aprovecha el paralelismo que ofrecen los múltiples núcleos de las GPU'S al lanzar un gran número de hilos de forma simultánea. Por ello, si la solución a un problema dado requiere realizar muchas tareas independientes las GPU'S son una muy buena alternativa pues pueden ofrecer un gran rendimiento.

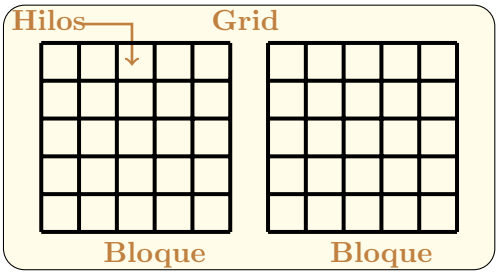

Figura 4.1: Estructura del modelo de programación.

La estructura que se utiliza en este modelo de programación está definido por un grid, dentro del cual hay bloques de hilos (figura 4.1). Esto permite trabajar con matrices bastante "grandes". Aunque el número de bloques esta restringido a 65 535 bloques en una llamada (por dimensión), así que se tendría la restricción del número de hilos por bloque multiplicado por el número de bloques. Esto no parece importante si se tiene en cuenta que en los dispositivos de computabilidad $2.x$, el máximo número de hilos es de 1024, lo que nos da más de 67 millones de filas. Sin embargo, el problema se presenta con la memoria compartida del dispositivo, y que en algunos casos, para la sincronización de hilos se tiene el límite de 32 hilos, lo cual nos da tan solo poco más de 2 millones de filas.

4.2. Matrices ralas (sparse matrices)

Una matriz $A \in M_n(\mathbb{R})$ se considera "rala", si la mayoría de sus entradas son cero. Esta característica permite un trabajo más eficiente tanto en almacenamiento como en tiempo de ejecución, si usamos una representación distinta a la usual. Existen muchos formatos para almacenar este tipo de matrices, cada uno con sus ventajas y desventajas, en particular usaremos el formato MSR (modified sparse row) el cual requiere que la diagonal de la matriz no contenga entradas nulas, situación que se presenta por ejemplo, en las matrices que se obtienen al aplicar alguna técnica numérica como diferencias finitas o elementos finitos a la solución de una ecuación en derivadas paciales. En la tabla 4.1 se muestra las estructura de este formato.

i:	0	1	...	$n-1$	n	$n+1$	$n+2$	...	nnz
AA:	a_{00}	a_{11}	...	$a_{n-1,n-1}$	NaN	$a_{i_0 j_0}$	$a_{i_1 j_1}$	...	$a_{i_k j_k}$
JA:	$n+1$	$JA_0 + nnz'_0$	...	$JA_{n-2} + nnz'_{n-2}$	$nnz+1$	j_0	j_1	...	j_k

Cuadro 4.1: Formato MSR.

La primera fila tan solo muestra los índices para los vectores AA y JA, ambos de tamaño $nnz+1$, donde nnz es el número de elementos que no son cero en la matriz (suponiendo que todos los elementos de la diagonal sean distintos de cero). Se guardan entonces todos los elementos de la diagonal en los primeros n espacios; en el siguiente se deja "un hueco", el cual vamos a llenar con un "NaN" (not a number), y luego, se colocan los elementos de la primera fila que son distintos de cero (a excepción de la diagonal); luego los de la segunda fila y así hasta la última fila.

El segundo vector es de índices. El primer valor es el índice donde comienzan los de la primera fila, que tiene nnz'_0 elementos distintos de cero (valor que **no** incluye la diagonal), y así con el resto de filas. En el n-ésimo índice se tiene el índice siguiente al final de la última fila, es decir, $JA_{n-1} + nnz'_{n-1}$.

Observe que en un algoritmo no hace falta pasar ni el tamaño de la matriz ni el número nnz, pues es información que se encuentra en el vector de índices: $n = JA_0 - 1$; $nnz = JA_n - 1$.

4.2.1. Transpuesta de una matriz rala

Para desarrollar los algoritmos de solución de sistemas lineales fue necesario implementar en CUDA la transpuesta de una matriz y el producto matriz por vector para el formato de matrices ralas MSR.

El algoritmo para determinar la transpuesta de una matriz rala realiza dos pasadas. La primera de ellas para determinar el número de elementos que hay en cada columna (que será el número de elementos que hay en cada fila de la matriz transpuesta):

```
1     for (i = n + 1; i <= nnz; i++) {
2         iCol = JA[i];
3         ++JAT[iCol + 1];
4     };
```

Observe que lo que se está modificando es la fila "siguiente" (`iCol + 1`), ello debido a que estamos "moviendo" el índice donde comienza la siguiente fila. Luego debemos actualizar los índices de las filas:

```
1     for (i = 1; i <= n; i++)
2         JAT[i] += JAT[i - 1];
```

En la segunda pasada se asignan los valores respectivos:

```
1     for (iFila = 0; iFila < n; iFila++) {
2         nFila = JA[iFila];          // indice donde comienza la i-esima fila
```

```
3           pFila = AA + nFila;         // puntero a los valores de la i-esima fila
4           pIndCol = JA + nFila;       // puntero a los indices de la i-esima fila
5           nFila = JA[iFila + 1] - nFila; // numero de elementos de la fila
6           for (i = 0; i < nFila; i++) {   // Se recorre la fila
7               valor = pFila[i];       // valor del elemento
8               iCol = pIndCol[i];      // indice de la columna
9               iValor = JAT[iCol] + nnzi[iCol]; // indice actual en AA^T
10              AAT[iValor] = valor;    // guardando valor
11              JAT[iValor] = iFila;    // guardando indice columna
12              ++nnzi[iCol];
13          };
14      };
```

El valor de `nnzi[iCol]` nos dice cuántos valores se han guardado para la fila respectiva.

No es sencillo utilizar paralelismo para generar la matriz transpuesta, debido a que se generan problemas conocidos en inglés como "race conditions", cuando se intenta realizar dos o más operaciones, al mismo tiempo, pero debido a la naturaleza del dispositivo o sistema, las operaciones deben realizarse en la secuencia apropiada con el fin de que se realicen correctamente. En todo caso, el tiempo que toma no es significativo, y se requiere hacer solamente una vez.

4.2.2. Producto matriz vector

Recuerde que si $\bar{b} = A\bar{x}$, entonces:

$$b_i = \sum_{j=0}^{n-1} a_{ij} x_j.$$

La implementación en C del producto matriz rala, vector, es la siguiente:

```
1  void matvect(float* AA, unsigned* JA, float* vector, float* resp) {
2      unsigned n = JA[0] - 1; // numero de filas
3      float ppunto;
4      unsigned i, iCol;
5      unsigned iActual, iFinal
6      for (i = 0; i < n; i++) {
7          // Se inicializa el producto punto con la diagonal.
8          ppunto = AA[i] * vector[i];
9          // Se obtienen los indices del comienzo y final de la fila
10         iActual = JA[i], iFinal = JA[i+1];
11         while (iActual < iFinal) {
12             iCol = JA[iActual];
13             ppunto += AA[iActual] * vector[iCol];
14             iActual++;
15         };
16         resp[i] = ppunto;
17     };
18 };
```

En la línea 2 se encuentra el número n, y en la línea 3 se define la variable `ppunto` que va a ir guardando el resultado parcial; la variable `i` se utiliza para recorrer las filas, mientras que `icol` se utiliza para obtener el índice de la columna, para obtener el elemento respectivo del vector (observe de la ecuación que el índice del vector coincide con el índice de la columna a_{ij}).

4.3. Gauss-Seidel en CUDA

Como ya se explicó, la fórmula para cada elemento de $\bar{x}^{(k)}$ está dada por:

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left(b_i - \sum_{j \neq i} a_{i,j} x_j^{(k)} \right)$$

Se puede observar que el método de Jacobi calcula casi por completo el vector del error, por lo que vamos a aprovechar dicha característica para hacer una iteración del método al momento de calcular el error.

Debido al gran tamaño de la matriz A no se chequea el error cada vez que se ejecuta el algoritmo, en su lugar se optó por hacerlo cada cierto número de veces, lo cual se define en la constante `GS_NUM_ITER`.

Se definen las siguientes constantes:

```
#define NUM_HILOS 64
#define NUM_BLOQUES 32
#define SALTO 2048 // NUM_HILOS * NUM_BLOQUES
#define GS_NUM_ITER 100
```

La idea de definir estas constantes es modificarlas para probar cuándo se obtienen, en general, los mejores resultados, lo cual claramente depende del dispositivo (GPU) que se use.

Como se mencionó, el algoritmo de Gauss-Seidel se ejecuta un cierto número de veces (`GS_NUM_ITER`), esto se administra mediante un programa en C cuyo código es el siguiente:

```
    // Inicializar el vector X en ceros.
    CUDA_inicializarX<<<NUM_BLOQUES, NUM_HILOS>>>(n, dev_vX);

    // Ciclo de iteraciones.
    do {
        // Se ejecuta Gauss-Seidel GS_NUM_ITER - 1 veces
        for (i = 1; i < GS_NUM_ITER; i++)
            CUDA_gs<<<NUM_BLOQUES, NUM_HILOS>>>
                (n, dev_AA, dev_JA, dev_vB, dev_vX);

        // Se ejecuta Gauss-Seidel una vez m\'as, y se calcula el error.
        // El error queda dividido, donde cada bloque aporta parte del valor.
        CUDA_gsError<<<NUM_BLOQUES, NUM_HILOS>>>
            (n, dev_AA, dev_JA, dev_vB, dev_vX, dev_errorDividido);

        // Se hace la reducci\'on del error.
        CUDA_sumaGlobalWrapper(dev_errorPartido, dev_errorGlobal);

        // Se copia el error del Device (CUDA) al Host (CPU)
        cudaMemcpy(&error, dev_errorGlobal, sizeof(double),
                    cudaMemcpyDeviceToHost);

        // Se actualiza el n\'umero de iteraciones
        numIter += GS_NUM_ITER;
        error = sqrt(error);
    } while(error > *epsilon && numIter < *maxNumIter);
```

El código en CUDA de Gauss-Seidel es el siguiente:

```
    // Cada hilo se encarga de una fila.
    for (i = hilo + NUM_HILOS * bloque; i < n; i += SALTO) {
        // Se inicializa el valor con b_i.
```

```
4           xi = vB[i];
5
6           // Similar al producto punto
7           iActual = JA[i], iFinal = JA[i + 1];
8           while (iActual < iFinal) {
9               // Se le resta a b_i, A[i,j] * x[j].
10              icol = JA[iActual];
11              xi -= AA[iActual] * vX[iCol];
12              iActual++;
13          };
14
15          // Actualizando el valor global.
16          vX[i] = xi/AA[i];
17      };
```

4.4. Gradiente conjugado en CUDA y OpenMP

El algoritmo para este método se define de la siguiente manera ([5]).

Algoritmo GRADIENTE CONJUGADO

1 $\bar{x} \leftarrow \bar{0}$; $\bar{r} \leftarrow \bar{b}$; $\bar{p} \leftarrow \bar{r}$; $\rho_{\text{new}} \leftarrow \langle \bar{r}, \bar{r} \rangle$
2 **repetir**
3 $\quad \rho_{\text{old}} \leftarrow \rho_{\text{new}}$
4 $\quad \bar{z} \leftarrow A \times \bar{p}$
5 $\quad \alpha \leftarrow \langle \bar{p}, \bar{z} \rangle / \rho_{\text{old}}$
6 $\quad \bar{x} \leftarrow \bar{x} + \bar{p}/\alpha$
7 $\quad \bar{r} \leftarrow \bar{r} - \bar{z}/\alpha$
8 $\quad \rho_{\text{new}} \leftarrow \langle \bar{r}, \bar{r} \rangle$
9 $\quad \bar{p} \leftarrow \bar{r} + \bar{p} \cdot \rho_{\text{new}}/\rho_{\text{old}}$
10 **hasta** que ρ_{new} sea suficientemente pequeño.

Observe que existen valores que se requieren para cálculos posteriores, lo cual hace que para su implementación en paralelo se deba separar el algoritmo en 4 partes, las cuales se describen a continuación.

De este algoritmo se implementaron dos versiones, una en OpenMP y otra en CUDA con el fin de poder comparar resultados posteriormente. Téngase en cuenta que ambos paradigmas son diferentes, OpenMP hace uso de los procesadores (CPU's) mientras que CUDA utiliza los procesadores gráficos (GPU's).

Inicialización

Corresponde a la línea 1. Aparte de la matriz A y los vectores $\bar{b}$ y $\bar{x}$, se requieren vectores $\bar{r}$ y $\bar{p}$, y el valor ρ. Así:

Algoritmo GC0

1 $x_i \leftarrow 0$, para $i = 0, 1, \ldots, n-1$
2 $r_i \leftarrow b_i$ para $i = 0, 1, \ldots, n-1$
3 $p_i \leftarrow b_i$ para $i = 0, 1, \ldots, n-1$
4 $\rho_{\text{new}} \leftarrow \sum b_i^2$

En OpenMP se puede utilizar la función `memcpy` para copiar el vector $\bar{b}$, así que el código consiste nada más en inicializar $\bar{x}$ y calcular el producto punto para ρ.

```
#pragma omp parallel for reduction(+:rhoNew) schedule(static)
    for (i = 0; i < n; i += 1) {
        vX[i] = 0.0;
        double bi = vB[i];
        rhoNew += bi * bi;
    };
```

OpenMP maneja la reducción en caso de que se le indique. El valor de la variable se inicializa automáticamente en cero. El atributo `schedule` lo que sugiere es una distribución del trabajo. En este caso, como el trabajo que hacen todos los hilos es el mismo, por lo que se utiliza la cláusula `static`, la cual distribuye el trabajo de manera fija entre los hilos.

En CUDA, el algoritmo es el siguiente:

```
    for (i = hilo + bloque * NUM_HILOS; i < n; i += SALTO) {
        bi = vB[i];
        vX[i] = 0, vr[i] = bi, vp[i] = bi;
        rho_bloque[hilo] += bi * bi;
    };

    // Se hace la reducci\'on del valor de \rho en el bloque.
    i = NUM_HILOS/2;
    while (i) {
        __syncthreads();
        if (hilo < i)
            rho_bloque[hilo] += rho_bloque[hilo + i];
        i /= 2;
    };

    // Se guarda parte del valor de rho de manera global.
    if (hilo == 0)
        rho_dividido[bloque] = rho_bloque[0];
```

Parte 1

Ahora se van a implementar las líneas 3 y 4 del algoritmo de Gradiente Conjugado. Cada hilo calcula un z_i, y luego un pedacito de v mediante reducción.

Algoritmo GC1

---

1 $z_i \leftarrow \sum_{j=0}^{n-1} A_{i,j} p_j$ para $i = 0, 1, \ldots, n-1$

2 $\alpha \leftarrow \left(\sum_{i=0}^{n-1} p_i z_i \right) / \rho$

En OpenMP, el algoritmo es el siguiente:

```
#pragma omp parallel for reduction(+:alfa) schedule(guided, 100)
        for (i = 0; i < n; i += 1) {
            pi = vp[i];
            zi = pi * AA[i];
            iActual = JA[i], iFinal = JA[i + 1];
            while (iActual < iFinal) {
                iCol = JA[iActual];
                zi += AA[iActual] * vp[iCol];
```

```
            };
            vz[i] = zi;
            alfa += pi * zi;
        };
        alfa /= rhoOld;
```

En este caso, para el atributo `schedule` se utilizó la cláusula `guided`, la cual comienza con un bloque "grande" para cada hilo, y luego va disminuyendo la cantidad de trabajo hasta llegar al valor especificado.

Por otro lado, el algoritmo en CUDA consiste en calcular primero z_i, y luego utilizar dicho valor para, mediante una reducción, ir calculando el valor de v. Cada z_i lo calcula un hilo, dado que la matriz es rala, es un cálculo pequeño.

```
    for (i = hilo + bloque * NUM_HILOS; i < n; i += SALTO) {
        pi = vp[i];
        zi = pi * AA[i];
        iActual = JA[i], iSiguiente = JA[i+1];
        while (iActual < iSiguiente) {
            icol = JA[iActual];
            zi += AA[iActual] * vp[iCol];
            iActual++;
        };
        vz[i] = zi;
        alfa_bloque[hilo] += pi * zi;
    };
```

Luego se hace la reducción de `alfa_bloque`.

Parte 2

Ahora se van a calcular las líneas 6 a 8 del algoritmo:

Algoritmo GC2

1 $x_i \leftarrow x_i + p_i/\alpha,\ i = 0, 1, \ldots, n-1$
2 $r_i \leftarrow r_i - z_i/\alpha,\ i = 0, 1, \ldots, n-1$
3 $\rho_{\text{new}} \leftarrow \sum_{i=0}^{n-1} r_i^2$

```
#pragma omp parallel for reduction(+:rhoNew) schedule(static)
        for (i = 0; i < n; i += 1) {
            vX[i] += vp[i] / alfa;
            ri = vr[i] - vz[i] / alfa;
            vr[i] = ri;
            rhoNew += ri * ri;
        };
```

De manera similar, en CUDA cada hilo calcula un x_i y un r_i, y mediante reducción.

```
    for (i = hilo + bloque * NUM_HILOS; i < n; i += SALTO) {
        vx[i] = vx[i] + vp[i] / (*alfa);
        ri = vr[i] - vz[i] / (*alfa);
        vr[i] = ri;
        rho_bloque[hilo] += ri * ri;
    };
```

Parte 3

Este último paso consiste únicamente en el cálculo del vector $\bar{p}$:

Algoritmo GC3

1 $p_i \leftarrow r_i + p_i \cdot \rho_{\text{new}}/\rho_{\text{old}}$, $i = 0, 1, \ldots, n-1$

El código en OpenMP es:

```
1 #pragma omp parallel for schedule(static)
2         for (i = 0; i < n; i += 1)
3             vp[i] = vr[i] + alfa * vp[i];
```

y en CUDA:

```
1     for (i = hilo + bloque * NUM_HILOS; i < n; i += SALTO)
2         vp[i] = vr[i] + vp[i] * rho;
```

4.5. Gradiente conjugado normalizado

Cuando la matriz de coeficientes del sistema lineal $A\bar{x} = \bar{b}$ no es simétrica no se puede aplicar el método del gradiente conjugado, pues el problema de minimizar la función cuadrática f no es equivalente al de resolver el sistema lineal $A\bar{x} = \bar{b}$. Existen muchos métodos que pueden ser aplicados a este caso, en particular se trabajo el método del gradiente conjugado normalizado que básicamente aplica el método del gradiente conjugado al sistema lineal $A^tAx = A^tb$, pues A^tA es una matriz simétrica definida positiva siempre que A sea no singular.

El seudocódigo para este método es el siguiente:

Algoritmo Gradiente Conjugado Normalizado

1 $\bar{x} \leftarrow \bar{0}$; $\bar{r} \leftarrow \bar{b}$; $\bar{p} \leftarrow A^T \times \bar{r}$; $\bar{s} \leftarrow \bar{p}$; $\rho_{\text{new}} \leftarrow \langle \bar{s}, \bar{s} \rangle$
2 **repetir**
3 $\quad \rho_{\text{old}} \leftarrow \rho_{\text{new}}$
4 $\quad \bar{z} \leftarrow A \times \bar{p}$
5 $\quad \alpha \leftarrow \langle \bar{z}, \bar{z} \rangle / \rho_{\text{old}}$
6 $\quad \bar{x} \leftarrow \bar{x} + \bar{p}/\alpha$
7 $\quad \bar{r} \leftarrow \bar{r} - \bar{z}/\alpha$
8 $\quad \bar{s} \leftarrow A^T \times \bar{r}$
9 $\quad \rho_{\text{new}} \leftarrow \langle \bar{s}, \bar{s} \rangle$
10 $\quad \bar{p} \leftarrow \bar{s} + \bar{p} \cdot \rho_{\text{new}}/\rho_{\text{old}}$
11 **hasta** que ρ_{new} sea suficientemente pequeño.

De manera similar a como se hizo para el método del gradiente conjugado este se dividió en partes las cuales fueron implementadas en CUDA.

Inicialización

Corresponde a la línea 1.

Algoritmo GCN0

1 $x_i \leftarrow 0$, para $i = 0, 1, \ldots, n-1$
2 $r_i \leftarrow b_i$ para $i = 0, 1, \ldots, n-1$
3 $p_i \leftarrow \sum_{j=0}^{n-1} A_{i,j}^T b_j$ para $i = 0, 1, \ldots, n-1$
4 $s_i \leftarrow p_i$ para $i = 0, 1, \ldots, n-1$
5 $\rho_{\text{new}} \leftarrow \sum s_i^2$

Parte 1

Corresponde a las líneas 4 y 5.

Algoritmo GCN1

1 $z_i \leftarrow \sum_{j=0}^{n-1} A_{i,j} p_j$ para $i = 0, 1, \ldots, n-1$
2 $\alpha \leftarrow \left(\sum_{i=0}^{n-1} z_i^2\right) / \rho_{\text{old}}$

Parte 2

Corresponde a las líneas 6 y 7.

Algoritmo GCN2

1 $x_i \leftarrow x_i + p_i/\alpha$, $i = 0, 1, \ldots, n-1$
2 $r_i \leftarrow r_i - z_i/\alpha$, $i = 0, 1, \ldots, n-1$

Parte 3

Corresponde a las líneas 8 y 9.

Algoritmo GCN3

1 $s_i = \sum_{j=0}^{n-1} A_{i,j}^T r_j$, $i = 0, 1, \ldots, n-1$
2 $\rho_{\text{new}} \leftarrow \sum_{i=0}^{n-1} s_i^2$

Parte 4

Corresponde a la línea 10.

Algoritmo GCN4

1 $p_i \leftarrow s_i + p_i \cdot \rho_{\text{new}}/\rho_{\text{old}}$, $i = 0, 1, \ldots, n-1$

4.6. Resultados

El cuadro 4.2 resume las características de las matrices de prueba usadas, estas fueron tomadas de la *Colección de Matrices Ralas de la Universidad de Florida* ([23]), en la cual colaboran investigadores de todo el mundo, compartiendo matrices que surgen de problemas reales para la evaluación de algoritmos.

La primera columna especifica el nombre de la matriz, la segunda su tamaño (n), la tercera el número de elementos no cero (nnz), la cuarta si es simétrica o no y la última el área de aplicación de donde proviene la matriz A.

Los problemas que se resolvieron son de la forma $A\bar{x} = \bar{b}$, donde la matriz rala A se tomó de dicha colección, y el vector $\bar{b}$ se construyó mediante $\bar{b} = A\bar{e}$, donde $\bar{e}$ es un vector de unos de tamaño $n \times 1$. Como criterios de parada en cada algoritmo se uso $\|\bar{r}_n\|_2 < 10^{-7}$ y un número máximo de iteraciones igual a n.

#	Nombre	n	nnz	Simétrica	Tipo de problema
1	ASIC-320ks	321 671	1 827 807	No	Simulación de circuitos
2	FEM-3D-thermal2	1 479 000	3 489 300	No	FEM 3D térmico no lineal
3	parabolic-fem	525 825	3 674 625	Si	Dinámica de fluidos
4	apache2	715 176	4 817 870	Si	Estructuras
5	ecology2	999 999	4 995 991	Si	Ecología
6	cage13	445 315	7 479 343	No	Grafo electroforesis DNA
7	G3-circuit	1 585 478	7 660 826	Si	Simulación de circuitos
8	thermal2	1 228 045	8 580 313	Si	FEM térmico no-estructurado
9	atmosmodd	1 270 432	8 814 880	No	Dinámica de fluidos
10	atmosmodl	1 489 752	10 319 760	No	Dinámica de fluidos
11	ohne2	181 343	11 063 545	No	Dispositivo semiconductor
12	Hook-1498	1 498 023	60 917 445	Si	Estructuras – FEM 3D
13	Serena	1 391 349	64 531 701	Si	Simulación reserva de gas
14	cage15	5 154 859	99 199 551	No	Grafo electroforesis DNA

Cuadro 4.2: Matrices de prueba.

Los algoritmos se ejecutaron en una computadora AMAX con 24 Gigas de RAM, una tarjeta Tesla C2050 con 448 núcleos CUDA, con 3 gigas de memoria global; dos CPU's Intel Xeon x5660, 2.67 GHz cada uno de 6 núcleos.

En el cuadro 4.3 se muestra el desempeño del algoritmo desarrollado en OpenMP para el método del gradiente conjugado usando diferente número de procesadores.

Matriz	NNZ	Iteraciones	1	2	5	8	10
parabolic-fem	3 674 625	951	19.96	10.32	6.42	6.25	6.35
apache2	4 817 870	5 461	128.23	71.22	46.13	43.16	44.80
ecology2	4 995 991	5 870	192.56	95.54	59.47	59.35	60.46
G3-circuit	7 660 826	16 881	802.31	477.51	270.52	296.72	299.08
thermal2	8 580 313	4 729	249.78	129.90	70.30	71.41	72.29
Hook-1498	60 917 445	9 278	–	–	702.95	846.52	–
Serena	64 531 701	91 076	–	–	5318.38	8720.91	–

Cuadro 4.3: Gradiente conjugado en OpenMP con diferente número de procesadores.

En el cuadro 4.4 se muestra una comparación de los métodos Gauss-Seidel y gradiente conjugado para matrices simétricas definidas positivas. Si el número de iteraciones está precedido por un asterisco, significa

que no se logro alcanzar el error de $\|\bar{r}_k\|_2 < 10^{-7}$ en el número de iteraciones preestablecido.

		Gradiente conjugado			Gauss-Seidel	
Nombre	NNZ	Iteraciones	CUDA	OpenMP	Iteraciones	CUDA
parabolic-fem	3 674 625	951	2.52	6.25	278 100	439.68
apache2	4 817 870	5 454	17.59	43.16	* 715 200	1316.54
ecology2	4 995 991	5 870	21.48	60.46	* 1 000 000	1880.70
G3-circuit	7 660 826	16 881	107.46	270.52	1 492 700	5381.24
thermal2	8 580 313	4 729	39.83	70.30	* 1 228 100	6968.18
Hook-1498	60 917 445	9 278	696.95	702.95	–	–
Serena	64 531 701	91 076	7457.93	5318.38	–	–

Cuadro 4.4: Comparación Gauss-Seidel y gradiente conjugado.

En el cuadro 4.5 se muestra una comparación de los tiempos de ejecución para los métodos del gradiente conjugado normalizado y Gauss-Seidel para matrices no simétricas. Si el número de iteraciones está precedido por un asterisco, es porque no se alcanzó el error de $\|\bar{r}_k\|_2 < 10^{-7}$ en el número de iteraciones preestablecido. Si tiene dos asteriscos, significa que el algoritmo no convergió.

Nombre	NNZ	Iteraciones	GC Normalizado	Iteraciones	Gauss-Seidel
FEM-3D-thermal2	3 489 300	8 102	63.20	** 900	3.30
ASIC-320ks	1 827 807	1 972	8.67	* 321 700	536.68
cage13	7 479 343	68	1.05	100	0.69
atmosmodd	8 814 880	18 158	173.06	–	–
atmosmodl	10 319 760	11 706	130.58	–	–
ohne2	11 063 545	181 344	6005.10	–	–
cage15	99 1995 51	70	15.36	100	10.05

Cuadro 4.5: Comparación gradiente conjugado normalizado y Gauss-Seidel.

Estos resultados fueron expuestos en el XVIII Simposio Internacional de Métodos Matemáticos Aplicados a las Ciencias (XVIII SIMMAC), celebrado del 21 al 24 de febrero del 2012 en la Escuela de Matemática de la Universidad de Costa Rica.

4.7. Conclusiones

Las principales conclusiones a las que hemos llegado despues de realizar esta investigación son:

- El paradigma de programación CUDA es una tecnología que resulta muy eficiente al resolver problemas en los cuales se requieren realizar muchas tareas independientes sobre una gran cantidad de datos.
- En general los métodos iterativos son eficientes para resolver sistemas de ecuaciones lineales de gran tamaño. Para los métodos estudiados se diseñaron e implementaron versiones en paralelo usando los paradigmas CUDA y OpenMP con éxito.
- Aunque CUDA es una herramienta muy poderosa no es aplicable de forma eficiente a cualquier tipo de problema, esta especialemente diseñada para ejecutar muchas tareas independientes sobre grandes cantidades de datos.

- En los casos en que el método de Gauss-Seidel convergía más rápido que el método del gradiente conjugado, el método del gradiente conjugado convergía aún así con bastante rapidez.
- La eficiencia de CUDA se ve mermada cuando se hacen accesos a la memoria global, pareciera que esto se presenta sobre todo cuando se realiza una operación de escritura. Esto es un punto muy importante al diseñar algorimtos para esta arquitetura.
- De las pruebas realizadas con OpenMP (cuadro 4.3) el tiempo de ejecuación no mejora significativamente después de que se utilizan más de 5 procesadores.
- El desempeño de OpenMP mejora significativamente conforme el tamaño de la matriz aumenta (cuadro 4.4). Esto se debe a que el tiempo que se pierde en el proceso de dividir el trabajo se recupera por la mayor cantidad del mismo.
- De las pruebas realizadas pareciera que OpenMP es más eficiente que CUDA en matrices "grandes", creemos que la razón de esto es una combinación de lo que se acotaba en el ítem anterior y de que muchos accsesos (escritura) a memoria global reducen el desempeño de CUDA.
- En algunos casos sobre todo para matrices muy grandes por encima de los sesenta millones de entradas creemos que se puede mejorar la convergencia aplicando alguna técnica de precondicionamiento, incluso en aquellos casos en los que no se logro la convergencia, por ejemplo con el método de Gauss-Seidel (cuadro 4.4 y 4.5), una técnica de estas podría ayudar.

Bibliografía

[1] Allaire, G; Kaber, Sidi Mahmoud. Numerical Linear Algebra. Springer-Verlag, USA. 2007.

[2] Buchanan, G.; Finite Element Analysis; McGraw-Hill; New York, 1995.

[3] Burden R., Faires, D.; Análisis Numúrico; International Thomson; México, 1998.

[4] Chen, KE; Matrix Preconditioning Techniques and Applications; Cambridge University Press, New York, 2005.

[5] Demmel, J. ; Applied Numerical Linear Algebra; SIAM; USA, pp. 311, 1997.

[6] Duchateau Paul y Zachmann David; Appied Partial Differential Equations, Publications Inc; New York, 1989.

[7] Farlow, Stanley ; Partial Differential Equations, Dover Publications Inc; USA, 1993.

[8] Greenbaum A.; Iterative Methods for Solving Linear System; SIAM; USA, 1997.

[9] Hackbusch W.; Iterative Solution of Large Sparse System of Equations; Springer Verlag; New York, 1994.

[10] Hestenes, M. y Stiefel, E. *Methods of Conjugate Gradients for Solving Linear Systems*. Journal of Research of the National Bureau of Standards, 49(6):409-436, December 1952.

[11] Hughes, Thomas; The Finite Element Method, Publications Inc; USA, 2000.

[12] Hey, T., Tansley, S. Tolle, K. (eds) *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft Research, Washington, 2009.

[13] Johnson, Claes; Numerical Solution of Partial Differential Equations by the Finite Element Method; Publications Inc; New York, 2009.

[14] LeVeque, Randall; Finite Differential Methods for Ordinary and Partial Differential Equations; SIAM; USA, 2007.

[15] Saad, Y.; Iterative Methodos for Sparse Linear System; SIAM; USA, 2003.

[16] Smith, G.; Numerical Solution of Partial Differential Equations : Finite Differential Methos; Oxford University Press; New York, 1985.

[17] Süli, Endre y Mayers David; An introduction to Numerical Analysis; Cambridge university Press; New York, 2008.

[18] Zachmanoglou, E. y Thoe, Dale; Introduction to Partial Differential Equations with Applications, Dover Publications Inc; USA, 1986.

[19] Sanders, Jason y Kandrot, Edward; CUDA by Example: An Introduction to General-Purpose GPU Programming; Addison-Wesley Professional, USA, 2010

[20] Kirk, David y Wen-mei, W. Hwu; Programming Massively Parallel Processors: A Hands-on Approach; Morgan Kaufmann; USA, 2010.

[21] Wen-mei, W. Hwu; GPU Computing Gems Jade Edition; Morgan Kaufmann; USA, 2011.

[22] `Nvidia: http://www.nvidia.com/object/tesla_computing_solutions.html`, consultada el 25 de marzo 2011.

[23] `Colección de matrices ralas de la Universidad de Florida:`

`http://www.cise.ufl.edu/research/sparse/matrices/`, consultada el 24 de febrero del 2012.

[24] `Colección de matrices Matrix Market: http://math.nist.gov/MatrixMarket/`, consultada el 20 de febrero del 2012.