UNIVERSIDAD NACIONAL DE COLOMBIA

Programación paralela sobre arquitecturas heterogéneas

Reinel Tabares Soto

Universidad Nacional de Colombia
Facultad de Ingeniería y Arquitectura
Departamento de Ingeniería Eléctrica, Electrónica y Computación
Manizales, Colombia
2016

Parallel programming over heterogeneous architectures

Reinel Tabares Soto

Universidad Nacional de Colombia
Facultad de Ingeniería y Arquitectura
Departamento de Ingeniería Eléctrica, Electrónica y Computación
Manizales, Colombia
2016

Programación paralela sobre arquitecturas heterogéneas

Reinel Tabares Soto

Tesis presentada como requisito para optar al título de:
Magister en Ingeniería - Automatización Industrial

Director:
(Ph.D.) Gustavo Adolfo Osorio Londoño

Línea de Investigación:
Computación
Grupo de Investigación:
Percepción y Control Inteligente - PCI

Universidad Nacional de Colombia
Facultad de Ingeniería y Arquitectura
Departamento de Ingeniería Eléctrica, Electrónica y Computación
Manizales, Colombia
2016

Dedicatoria

Hay una fuerza más grande que el vapor, la electricidad y la energía atómica: la voluntad.

Albert Einstein.

A mis padres y futura esposa.

Agradecimientos

Agradezco a:
Mis padres por su apoyo incondicional, confianza, por infundirme valores y amor por el conocimiento. Espero ser para ellos motivo de orgullo y felicidad.
Mi tutor (Ph.D.) Gustavo Adolfo Osorio Londoño profesor titular de la Universidad Nacional de Colombia sede Manizales por su impecable trabajo al momento de servir de guía en mi proceso de formación como investigador, porque gracias a sus consejos y recomendaciones se pudieron lograr a cabalidad los objetivos propuestos.
Todos mis amigos y colegas porque sin su ayuda y paciencia habría sido imposible obtener los resultados esperados de este trabajo de investigación.
Agradezco a la Universidad Nacional de Colombia sede Manizales y a Colciencias por su apoyo académico y económico por medio del programa de Jóvenes Investigadores e Innovadores 2014, acuerdo de cooperación especial Nº 0200 de 2014. Resolución 0444 DFIA de 2015. En el marco del proyecto "Programación Paralela sobre Arquitecturas Heterogéneas".

Resumen

A medida que avanzan las investigaciones y la tecnología se generan cada vez volúmenes más grandes de información y esto ocasiona un aumento drástico en los tiempos necesario para procesarla, es por eso que la comunidad científica está en la búsqueda de alternativas para suplir las necesidades de obtener resultados en el menor tiempo posible y optimizando los recurso, una de las posibles alternativas consiste en usar las nuevas arquitecturas heterogéneas que incluyen una interacción CPU-GPU para procesamiento paralelo, es aquí en donde reside la importancia de esta investigación la cual pretende utilizar programación paralela por medio de unidades de procesamiento gráfico (GPU) para la aceleración de algoritmos de cómputo científico, cada uno de los problemas tratados se implementaran de forma paralela en GPU y de forma secuencial en CPU con el fin de contrastar tiempos de simulación y así mostrar bajo qué condiciones y qué arquitectura obtendrán las mejores aceleraciones. A lo largo de todo el documento se explicará detalladamente las estrategias de computación paralela sobre arquitecturas heterogéneas utilizando en mayor proporción dispositivos de procesamiento gráfico (GPU), tomando como base el lenguaje C y las librerías CUDA y OpenCL para la gestión de la misma, con el fin de que el problema se adapte de la mejor forma al dispositivo utilizado. Dentro de los resultado obtenidos se observan aceleraciones de hasta $213X$ para el sistema de Lorenz y convertidor Buck, en el caso de la ecuación de Laplace en 2 dimensiones se puedo lograr aceleraciones de hasta $9{,}7X$, en el caso de la ecuación de Laplace en 3 dimensiones se puedo lograr aceleraciones de hasta $8{,}6X$ y en el caso del análisis estadístico de SNP se obtuvo aceleración de $26{,}9X$ de forma paralela en CPU y de $8{,}2X$ de forma paralela en GPU.

Palabras clave: Unidad de Procesamiento Gráfico de Propósito General GPGPU, Unidad de Procesamiento Gráfico (GPU), Unidad Central de Procesamiento (CPU), Arquitectura de Dispositivo Unificada de Cómputo (CUDA), OpenCL, Polimorfismo de un Solo Nucleótido (SNP), Bifurcación, Dominio de Atracción, Kernel, Programación Paralela, Sistemas No Lineales, Ecuaciones Diferenciales Parciales(EDP), Indexación de Memoria, Multi-thread, Multi-core y Asociación de genoma completo(GWAS).

Abstract

Parallel programming over heterogeneous architectures.

As research progresses and technology are generating larger amounts of information, and this causes a dramatic increase in the need for processing times, which is why the scientific community is in search of alternatives to meet the needs of obtaining results in the shortest possible time and

optimizing resource, one possible alternative is to use the new heterogeneous architectures including a CPU-GPU interaction for parallel processing, it is where the importance of this research lies which intends to use parallel programming through graphics processing units (GPU) for accelerated algorithms for scientific computing, each of the problems addressed are implemented in parallel on GPU and sequentially in CPU in order to contrast simulation times and thus show low that architecture conditions and get the best acceleration. Throughout the document it will be explained in detail strategies parallel computing on heterogeneous architectures using a greater proportion devices graphics processing (GPU), based on the language C and CUDA and OpenCL libraries for managing it, with the so that the problem fits in the best way the device used. Within the result obtained accelerations up to $213X$ for the Lorenz system and Buck converter are observed in the case of the Laplace equation in 2 dimensions can achieve accelerations up to $9{,}7X$, in the case of the equation Laplace in 3 dimensions can achieve accelerations up to $8{,}6X$ and in the case of statistical analysis of SNP acceleration of $26{,}9X$ parallel CPU and $8{,}2X$ parallel GPU was obtained.

Keywords: General Purpose Graphics Processing Unit (GPGPU), Graphics Processing Unit (GPU), Central Processing Unit (CPU), Compute Unified Device Architecture (CUDA), OpenCL, Polymorphism Single Nucleotide (SNP), Bifurcation, Domain attraction , Kernel, Parallel Programming, Nonlinear Systems, Partial Differential Equations (PDE), Indexing of Memory, Multi-thread, Multi-core and Genome Wide Association (GWAS).)

Contenido

1 Introducción

La computación de propósito general en unidades de procesamiento grafico está jugando un papel importante en las aplicaciones de computación científica debido a la gran cantidad de datos que deben ser procesados con estrictas limitaciones de tiempo [53]. Sin embargo, los equipos necesarios para llevar a cabo estas tareas en forma eficiente y rápida son muy costosos, por lo cual la comunidad científica está en la búsqueda de alternativas más económicas. La computación heterogénea se ha convertido en una posibilidad ya que permite combinar CPU clásica con dispositivos de cómputo especializados que están encargados de la mayoría de las tareas de cálculo [58].

Desafortunadamente, el uso de un dispositivo acelerador tal como una unidad de procesamiento gráfico (GPU) no garantiza obtener aceleración. Para lograr el objetivo, el problema tiene que ser adaptado a la estructura del dispositivo [10]. Además, los cuellos de botella generados por la transmisión de grandes cantidades de datos tienen que ser reducidos; por otro lado, las GPU están optimizadas para realizar tareas de procesamiento de imágenes, y en general computación gráfica, mediante el uso de indexación bi-dimensional y tri-dimensional [20]. La estrategia propuesta en esta tesis es adaptar los problemas tratados a este sistema de indexación [29]. En [62] y [41] se muestran un conjunto de experimentos realizados utilizando la indexación bi-dimensional en diferentes arquitecturas con el objetivo de determinar las características óptimas del dispositivo para resolver un problema específico o para ajustar el problema al dispositivo de aceleración disponible.

Al momento de plantear una estrategia paralela de un algoritmo se debe primero entender a profundidad el problema a implementar, luego se debe hacer un análisis de la posibilidad de representar el problema en una estructura paralela, es decir, no hay dependencia de los datos a procesar o de alguna manera se puede controlar tal dependencia. Por último es necesario elegir qué arquitectura se adapta de mejor manera al problema. Lo anterior no asegura que el tiempo de ejecución disminuya, lo que obliga a que se deban hacer ajustes al modelo, a la arquitectura o al algoritmo.

1.1. Arquitecturas

En la actualidad hay dos grandes arquitecturas que permiten implementar un problema de forma paralela (CPU y GPU). En CPU se encuentran diferentes formas de ejecutar algoritmos en forma paralela como por ejemplo multi-hilos [17], multi-núcleos [43], *cluster* [14, 77] y *grid* [28, 69]. Cuando se trata de multi-hilos lo que se hace es tomar un solo núcleo que comparte memoria y se

crean tantos hilos como procesos tenga el problema con el fin de que cada hilo ejecute un proceso de una manera concurrente. En el caso de multi-núcleos, se tiene una sola CPU que posee una determinada cantidad de núcleos cada uno resolviendo un proceso. El cluster consiste en una serie de CPU independientes que se conectan por medio de una red LAN y cada CPU se encargará de un proceso o un conjunto de ellos. Por último una *grid* es semejante a un *cluster* pero conectada por una red WAN.

Al usar GPU las únicas posibilidades son una sola GPU o múltiples GPU (*cluster* de GPU) [70], es importante notar que una CPU puede incluir algunos cientos de núcleos, mientras que en GPU puede tener miles de núcleos, lo anterior no significa que la GPU tenga mejor desempeño que la CPU ya que un solo núcleo de CPU es muy superior a uno de GPU, pero la GPU está optimizada para gestionar la memoria de forma paralela. También se pueden hacer combinaciones entre las arquitecturas anteriormente mencionadas para obtener un máximo beneficio de ambas y esto se conoce como arquitecturas heterogéneas [88]. En esta tesis se utilizará en gran medida arquitectura GPU para resolver los problemas propuestos pero en el capítulo cinco se agregará arquitectura CPU usando múltiples hilos sobre múltiples núcleos.

En la Tabla **1-1** se muestran las arquitecturas de CPU y en la Tabla **1-2** se muestran las arquitecturas de GPU utilizadas en la tesis para resolver los diferentes problemas planteados, algunas características adicionales se mostrarán en el resto del documento ya que no todas las arquitecturas fueron utilizadas en todas las implementaciones.

Tabla 1-1: Arquitecturas de CPU

Especificación	Intel Core I 7-4820K	Intel Xeon E7-4860 v2	Intel Xeon E7- 4820	Intel Xeon X5650	Intel Xeon E5-2670
Cantidad de núcleos	4	12	8	6	8
Frecuencia del reloj de memoria (MHz)	3700	2600	2000	2666	2666
Ancho de Bus (bit)	64	64	64	64	64
Ancho de banda (GB/seg)	59,7	85	26	32	51,2
Cantidad de transistores (millones)	1860	5600	2300	1300	2270
Tamaño del procesador (nm)	22	22	32	32	32
Operaciones en punto flotante (GFLOPS)	32,5	100,2	39,3	33,7	38,7
Precio de lanzamiento (USD)	332	3838	1446	996	1556
Consumo (Watts)	130	130	105	95	115
Fecha de lanzamiento	2013	2014	2011	2010	2012
Arquitectura	Ivy Bridge	Ivy Bridge	Westmere	Westmere	Sandy Bridge
Problema implementado	Lorenz, Buck, Laplace y GWAS	GWAS	GWAS	GWAS	GWAS

1.2. Software y lenguajes

El sistema operativo utilizado es Linux Centos 7 con una versión de Kernel $3{,}18{,}4$. El lenguaje de programación es C con un compilador GCC versión $4{,}4{,}7$ + Python $2{,}7{,}10$. En el lenguaje C se hizo toda la implementación de la soluciones de los diferentes problemas, los resultados fueron graficados en la plataforma de MATLAB. Se utilizó el IPhyton notebook $3{,}1{,}0$ como interpretador WEB para facilitar la programación, y las librerías de CUDA 7 y OpenCL 2 para programación

Tabla 1-2: Arquitecturas de GPU

Especificación	Nvidia GTX 770	Nvidia Quadro 4000	Nvidia GTX 860M	AMD Radeon HD 6770	Nvidia Tesla k20m
Memoria global (MBytes)	2048	2048	2048	1024	5120
Unidades de cómputo	8	8	6	10	13
Cantidad de núcleos	1536	256	640	800	2496
Máxima cantidad de bloques	65535	65535	65535	65535	65535
Máxima cantidad de hilos por bloque	1024	1024	1024	512	1024
Frecuencia del reloj de memoria (MHz)	7012	2808	5012	4800	5200
Ancho de Bus (bit)	256	256	128	128	320
Ancho de banda (GB/seg)	224	89,6	80,2	76,8	208
Cantidad de transistores (millones)	3540	3100	1870	1040	7080
Interfaz del Bus (PCI-E)	3,0x16	2,0x16	3,0x16	2,0x16	2,0x16
Tamaño del procesador (nm)	28	40	28	40	28
Operaciones en punto flotante (GFLOPS)	3213,0	486,4	1306,0	1360,0	3524,0
Precio de lanzamiento (USD)	399	1199	115	129	3199
Consumo (Watts)	230	142	75	108	225
Fecha de lanzamiento	2013	2010	2014	2011	2013
Arquitectura	Kepler	Fermi	Maxwell	Juniper	Kepler
Problema implementado	Lorenz, Buck y GWAS	Lorenz, Buck	Laplace	Laplace	GWAS

paralela sobre GPU.

1.2.1. CUDA

CUDA es un conjuno de herramientas de computación y programación paralela desarrollada por Nvidia, que tiene como objetivo aprovechar la arquitectura de la GPU para propósitos generales como es el caso de la computación científica. Esta plataforma sólo funciona sobre tarjetas Nvidia y se basan en lenguaje C. En esencia, CUDA consiste en una serie de librerías que se integran con un lenguaje de programación que tienen como objetivo gestionar los recursos de la GPU y asignar los procesos de una manera paralela a cada uno de los núcleos de la tarjeta de video. Es importante aclarar que una GPU posee miles de procesadores que no son comparables uno a uno con los de una CPU pero, si se hace una correcta fragmentación del problema, se podría llegar a obtener una eficiencia alta en la GPU. En las librerías de CUDA se debe definir un *Kernel* (función núcleo) que se ejecutará de manera paralela sobre cada uno de los procesadores por medio de identificadores (id) y el resto del código se encargará de gestionar la forma de usar dicho *Kernel*, ver más en [57].

1.2.2. OpenCL

Consiste en una interfaz de programación de aplicaciones y un conjunto de librerías que juntas permiten crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en unidades centrales de procesamiento (CPU), como en unidades de procesamiento gráfico (GPU). Estas librerías fueron creadas como un estándar de computación paralela que permiten el dominio de un amplio número de dispositivos aceleradores, permitiendo además la escritura de código portable [93]. Aunque la portabilidad no es garantía de un buen desempeño en las diferentes plataformas, permite contar con una implementación base de los algoritmos, para luego centrarse en ajustar adecuadamente las propiedades de cada dispositivo, ver más en [65].

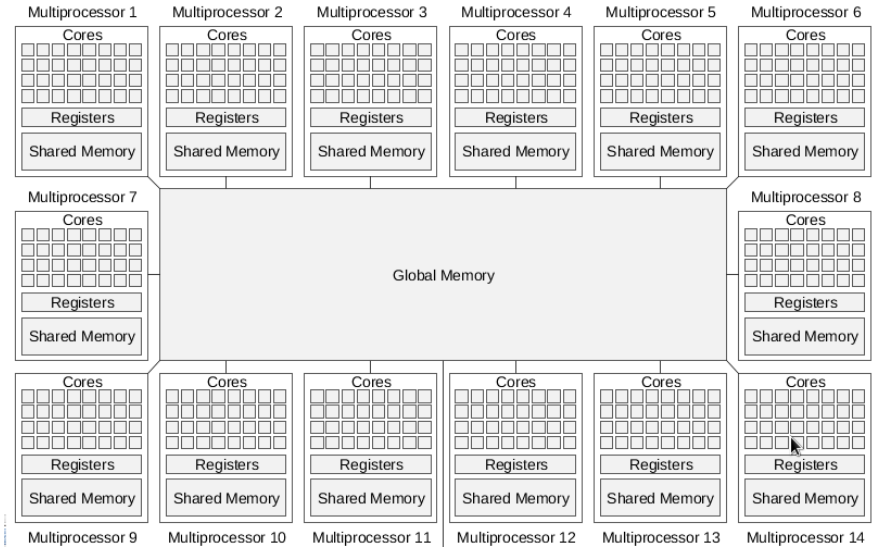


Figura 1-1: Arquitectura de la GPU.

1.3. Indexación paralela en GPU

Al hacer una implementación en paralelo, un paso importante es la indexación del dispositivo, que consiste en segmentarlo para que trabaje de manera paralela en los procesos deseados. Normalmente esta indexación se realiza por medio de identificadores (id) cada uno de los cuales se encargará de un proceso por medio de un hilo. Cuando se indexa un dispositivo como la GPU, todos los núcleos deben tener acceso a una misma memoria conocida como memoria global la cual es de acceso lento. Además todos los núcleos que pertenecen a un mismo segmento (*multiprocessor*) pueden acceder a una memoria adicional conocida como memoria compartida que es de acceso rápido. Por este motivo, al momento de indexar el problema en la GPU, es importante que los datos que requieran información de otros para su proceso se deben indexar de tal forma que la información se encuentre en la memoria compartida, es decir, núcleos de un mismo segmento, para así optimizar los tiempos de acceso a memoria. La desventaja de la memoria compartida es que posee un tamaño muy limitado en comparación con la memoria global, pero si se hace una correcta indexación y un correcto uso de la memoria se pueden lograr excelentes resultados. En la figura **1-1** se muestra cómo es la arquitectura de una GPU.

En esta tesis para el control y flujo de datos se usará la taxonomía de Flynn [26] que consiste en ejecutar la misma instrucción sobre múltiples datos $(SIMD)$.

1.4. Cómo se mide la aceleración de un algoritmo

La aceleración de un algoritmo dice que tan rápida es la implementación en paralelo con respecto a la implementación secuencial. También es conocida como *Speed-Up* y es una magnitud adimen-

sional. Las ecuaciones (1-1), (1-2) y (1-3) muestran cómo se calcula la aceleración, en donde U son las unidades de cómputo, N el número de elementos de proceso por unidad, $R(N)$ el rendimiento máximo teórico, $S(N)$ *Speed-Up* o aceleración, T tiempo secuencial, $TP(N)$ tiempo paralelo y $E(N)$ es la eficiencia.

$$R(N) = U \times N \tag{1-1}$$

$$S(N) = \frac{T}{TP(N)} \tag{1-2}$$

$$E(N) = \frac{S(N)}{N} = \frac{T}{N \times TP(N)} \tag{1-3}$$

En la práctica no es posible lograr aceleraciones teóricas. Por tal razón para entender la aceleración real de un algortimo, se utiliza la ley de Amdahl [3], la cual explica cómo se debe calcular el tiempo en paralelo de un algoritmo y con base a esto predice el comportamiento real de la aceleración del mismo. Amdahl dice: « Idealmente, la aceleración a partir de la paralelización es lineal, doblar el número de elementos de procesamiento debe reducir a la mitad el tiempo de ejecución y doblarlo por segunda vez debe nuevamente reducir el tiempo a la mitad. Sin embargo, muy pocos algoritmos paralelos logran una aceleración óptima. La mayoría tienen una aceleración casi lineal para un pequeño número de elementos de procesamiento, y pasa a ser constante para un gran número de elementos de procesamiento». En la ecuación (1-4) se presenta la ley de Amdhal, en donde f es la parte secuencial del código, reemplazando la ecuación (1-4) en la ecuación (1-2) se obtiene la ecuación (1-5) y haciendo un límite sobre la ecuación (1-5) tendiendo las unidades de proceso a infinito se obtiene (1-6). La ecuación (1-6) significa que la aceleración real de un algoritmo tiende a un valor constante, esto se debe a que a medida que agregamos unidades de cómputo el tiempo de procesamiento disminuye pero los tiempos de transferencia y comunicación entre las unidades aumenta.

$$TP(N) = f \times T + (1 - f) \times \frac{T}{N} \tag{1-4}$$

$$S(N) = \frac{N}{f(N-1)+1} \tag{1-5}$$

$$\lim_{N \to \infty} S(N) = \lim_{N \to \infty} \frac{N}{f(N-1)+1} = \frac{1}{f} \tag{1-6}$$

En la figura **1-2** se observa la aceleración de un algoritmo a medida que se aumenta el número de procesadores y la porción paralela del mismo. Se puede deducir que la aceleración tiene un comportamiento aproximadamente lineal para cantidades bajas de procesadores pero después de un valor crítico la aceleración se comporta de una manera constante. Esto significa que si se desea optimizar el tiempo de simulación de una aplicación es importante aumentar la porción paralela de código y el número de procesadores o hilos a ejecutar teniendo en cuenta que hay un punto de saturación.

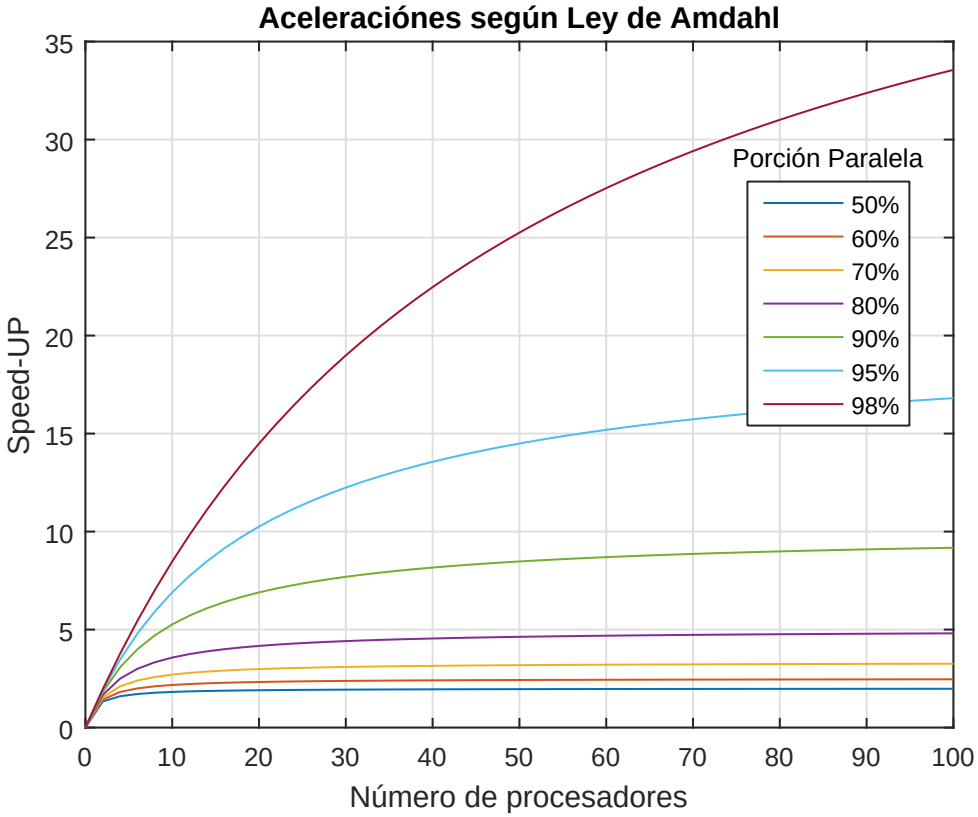


Figura 1-2: Ley de Amdhal.

2 Análisis no lineal del sistema de Lorenz y Convertidor Buck utilizando GPU

El análisis no lineal consiste en el uso de algoritmos para la aproximación de las soluciones de un modelo analítico para encontrar conjuntos invariantes y su estabilidad en un espacio de parámetros. Normalmente estos algoritmos pueden ser formulados como problemas de continuación numérica, de los cuales un caso particular es el cálculo de la estabilidad estructural de soluciones, conocido también como diagrama de bifurcaciones. Una bifurcación de un sistema dinámico es un cambio cualitativo en su dinámica producida por la variación de los parámetros [47, 37]. Otro tipo de análisis que se hace a menudo asociado a los diagramas de bifurcaciones es el cálculo de dominios de atracción para escenarios de coexistencia de soluciones. Un dominio de atracción es una partición del espacio de estados en el que todas las condiciones iniciales pertenecientes a un subconjunto tienden al mismo conjunto límite [15].

La tarea de realizar numéricamente un diagrama de bifurcaciones o un dominio de atracción puede ser compleja computacionalmente dada la cantidad de cálculos que se requieren. Generalmente es necesario realizar cientos, miles o millones de simulaciones variando un conjunto de parámetros o las condiciones iniciales del sistema. Tradicionalmente el análisis numérico no lineal se ha realizado usando procesadores que operan de manera secuencial, esto significa, ejecutar una sola simulación a la vez. Dada la naturaleza numérica de este tipo de problemas, es posible realizar computación paralela, es decir, ejecutar simultáneamente miles de simulaciones. [2, 51, 81].

En este capítulo se presenta el análisis de bifurcaciones y de dominios de atracción de dos tipos de sistemas. El primero de naturaleza suave y no lineal: las ecuaciones de Lorenz. El segundo sistema de naturaleza conmutada, el convertidor Buck controlado por onda seno. Todos los experimentos numéricos se realizaron de manera paralela en dos GPU y de manera secuencial en CPU. Los experimentos paralelos se realizaron en las tarjetas de vídeo GTX 770 y Quadro 4000. Los experimentos secuenciales se implementaron en un procesador Intel $I7\ 4820K$ de $3{,}7$ GHz con una memoria RAM de 16 GB DDR3 de 1333 MHz. Para la implementación de los algoritmos en las GPU se utilizó el lenguaje C usando librerías de CUDA y para la implementación en CPU se utilizaron rutinas en lenguaje C. Los experimentos numéricos realizados de manera paralela lograron aceleraciones (*Speed*-Up) de hasta 213X para la tarjeta Quadro y 156X para la GTX.

En la sección 2.1 se presenta una descripción del sistema de Lorenz. En la sección 2.2 se hace una descripción del convertidor Buck controlado por onda seno. En la sección 2.3 se muestra la metodología para la implementación paralela de los algoritmos de análisis no lineal. En la sección 2.4 se muestran los resultados del análisis no lineal para ambos sistemas y se comparan los tiempos de ejecución entre las diferentes arquitecturas utilizadas y para diferentes aproximaciones numéricas. Finalmente, en la sección 2.5, se especifican algunas conclusiones y observaciones.

2.1. Sistema de Lorenz

Fue introducido por E.N Lorenz en el año de 1963. Es un sistema de ecuaciones diferenciales que ofrece una descripción del impredecible comportamiento del clima [55]. Lorenz experimentó con una celda bidimensional de fluido, la cual era calentada desde abajo y enfriada desde arriba. El movimiento del fluido bajo estas condiciones podría ser descrito por un sistema de ecuaciones diferenciales de muchísimas variables. Luego de una simplificación, Lorenz fue capaz de reducir el problema a un sistema de tres dimensiones con tres parámetros como se describe a continuación:

$$\begin{aligned} \dot{x} &= -\sigma x + \sigma y \\ \dot{y} &= rx - y - xz \\ \dot{z} &= -bz + xy. \end{aligned} \tag{2-1}$$

Las ecuaciones del sistema de Lorenz constituyen un sistema dinámico determinístico tridimensional no lineal derivado de las ecuaciones simplificadas de rollos de convección que se producen en la atmósfera terrestre. Lorenz supone que la velocidad del fluido puede ser descrita por un único rollo y que la temperatura en la capa puede ser descrita por una solución en el estado estacionario y dos modos dependientes del tiempo [86].

$x(t)$, $y(t)$ y $z(t)$ denotan la velocidad rotacional de la celda de convección, la diferencia de temperatura entre las corrientes ascendentes y descendentes, y la distorsión de la temperatura vertical del perfil de conducción (inestabilidad de aire) respectivamente.

El parámetro σ (número de Prandtl), es una constante adimensional proporcional al cociente entre la difusividad de momento (viscosidad) y la difusividad térmica, b es un factor asociado a la geometría del problema de dos placas planas paralelas infinitas y r es el número de Rayleigh, que da una medida de la diferencia de calor entre las placas. La figura **2-1** ilustra una solución particular para $\sigma = 10$, $r = 28$ y $b = 8/3$, conocida también como el atractor caótico de Lorenz.

El sistema de Lorenz ha sido ampliamente analizado desde la década del 60 hasta la actualidad [35, 38, 79, 76]. El desarrollo tecnológico de la computación ha permitido realizar análisis cada vez más elaborados, como por ejemplo estudios de su estabilildad estructural [35, 76] (Análisis de

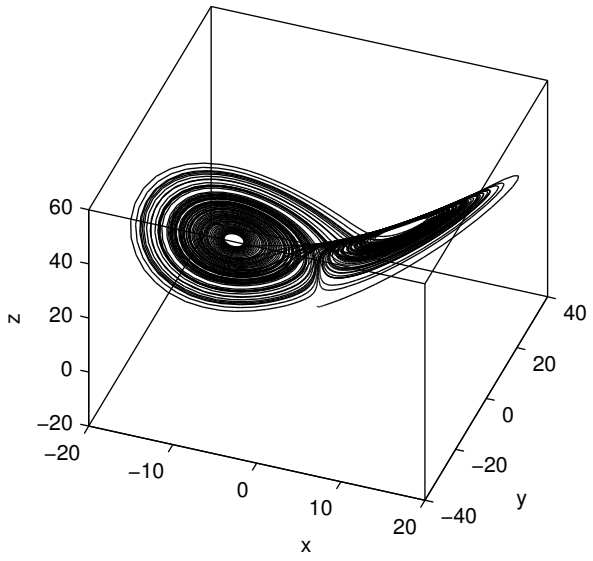


Figura 2-1: Atractor caótico de Lorenz.

bifurcaciones). Tradicionalmente los experimentos numéricos relativos al análisis no lineal se han realizado empleando CPU que operan de manera secuencial, este tipo de procesamiento requiere de cientos o miles de simulaciones con diferentes valores de parámetros o de condiciones iniciales, lo cual puede suponer largos tiempos de ejecución. La aparición de la computación paralela ha permitido acelerar este tipo de experimentos numéricos, dado que permite ejecutar simultánemente miles de cómputos [44]. En [1] se presentan algunas simulaciones del sistema de Lorenz utilizando GPU, las rutinas numéricas fueron programadas utilizando C++ y librerías de OpenCL. En [60] se presenta la aplicación *Fireflies*, que permite realizar simulaciones interactivas de sistemas dinámicos empleando GPU.

2.2. Convertidor Buck controlado por onda seno

El convertidor Buck es un dispositivo electrónico que toma una señal de voltaje DC no controlada y entrega una señal de voltaje DC regulada en un nivel inferior al de entrada. Una alternativa para reducir un voltaje continuo (DC) es usar un circuito divisor de voltaje, pero los divisores disipan energía en forma de calor. Por otra parte, un convertidor Buck puede tener una alta eficiencia (superior al 95 % con circuitos integrados) y autoregulación [66, 80].

La figura **2-2** ilustra el esquema de un convertidor Buck controlado por onda seno. El circuito posee dos dispositivos que conmutan; un diodo D_1 y un interruptor T_1. También posee un inductor

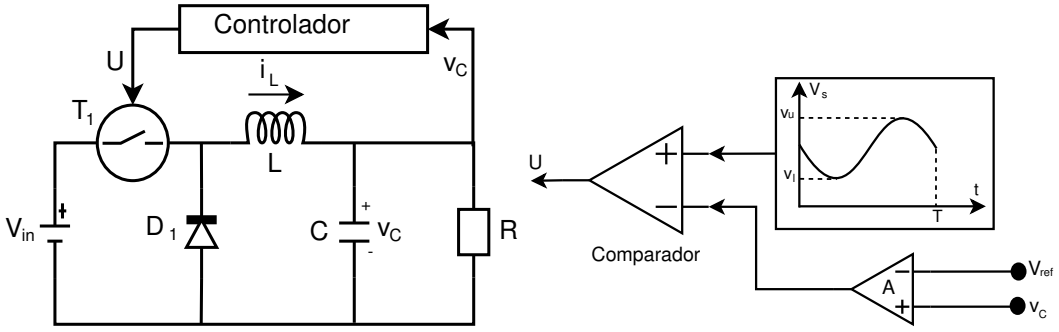
L y opcionalmente un condensador C a la salida. La señal de control U se define como:

$$U = \begin{cases} 1 & si \quad A\,(v_C - V_{ref}) \leq V_s \\ 0 & si \quad A\,(v_C - V_{ref}) > V_s \end{cases} \tag{2-2}$$

Donde:

$$V_s = \frac{(V_u - V_l)}{2} \sin\left(\frac{2\pi}{T}t\right) + \frac{V_u}{2}$$

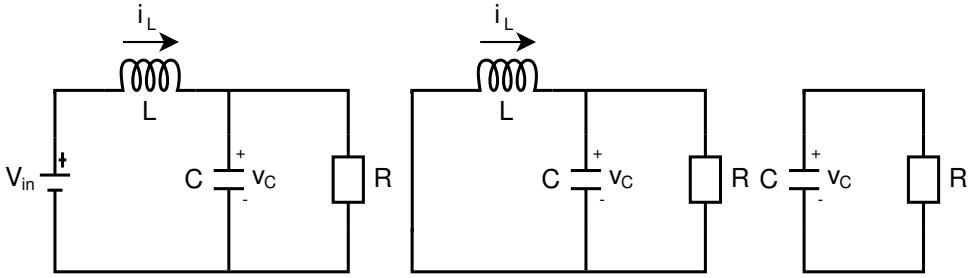
En [7] se presenta un análisis exhaustivo del convertidor Buck controlado mediante esta técnica.



(a) Convertidor Buck controlado (b) Controlador

Figura 2-2: Convertidor Buck controlado por onda seno.

El convertidor Buck puede presentar tres topologías de acuerdo al estado de los interruptores, tal como se muestra en la figura **2-3**.



(a) Topología 1 (b) Topología 2 (c) Topología 3

Figura 2-3: Topologías del convertidor Buck.

Cada topología es modelada mediante un sistema de ecuaciones diferenciales como se muestra a continuación:

Topología 1:

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} -\frac{1}{RC} & \frac{1}{C} \\ -\frac{1}{L} & 0 \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{V_{in}}{L} \end{bmatrix} \tag{2-3}$$

Topología 2:

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} -\frac{1}{RC} & \frac{1}{C} \\ -\frac{1}{L} & 0 \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} \tag{2-4}$$

Topología 3:

$$\begin{bmatrix} \dot{v}_C \\ \dot{i}_L \end{bmatrix} = \begin{bmatrix} -\frac{1}{RC} & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} v_C \\ i_L \end{bmatrix} \tag{2-5}$$

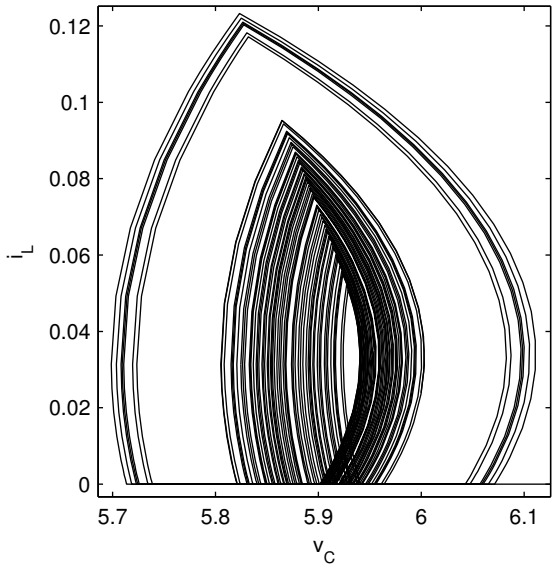


Figura 2-4: Atractor caótico del convertidor Buck.

Dada la naturaleza conmutada del convertidor Buck se puede presentar gran variedad de dinámicas, como por ejemplo las caóticas [27, 71]. La figura **2-4** ilustra un atractor caótico del convertidor Buck. Debido a que este sistema es periódicamente forzado, se puede apreciar un solapamiento de las trayectorias. Para realizar esta simulación se utilizaron los siguientes valores de parámetros $V_{in} = 15V$, $R = 75\Omega$, $L = 0{,}02H$, $C = 470\mu F$, $V_{ref} = 5V$, $A = 8{,}4$, $T = 400\mu S$, $V_l = 308V$ y $V_u = 8{,}2V$.

Mediante el análisis no lineal del convertidor Buck es posible determinar y clasificar numéricamente los diferentes tipos de respuesta que pueden aparecer. Sin embargo, esta tarea no es simple,

ya que además de las miles de simulaciones que se requieren, cada simulación en sí misma puede presentar bloqueos dada la naturaleza no suave del convertidor Buck [89]. En la literatura se reportan algunos trabajos relativos al análisis no lineal del convertidor Buck [27, 71, 6, 7], en todos ellos se emplean algoritmos ejecutados en CPU. En [96] se presentan simulaciones del convertidor Buck utilizando GPU, sin embargo no se realiza ningún tipo de análisis no lineal.

2.3. Metodología para el análisis no lineal en GPU

La estrategia usada fue implementar algoritmos de fuerza bruta; se realizaron miles de simulaciones variando las condiciones iniciales y los parámetros.

Los diagramas de bifurcaciones fueron paralelizados de manera diferente para cada sistema. En el sistema de Lorenz se generó una nube de condiciones iniciales, y de manera iterativa se varió el valor del parámetro a analizar. En cada iteración se enviaran de manera paralela las condiciones iniciales a un conjunto de hilos de la GPU. En este caso se usó un híbrido entre computación paralela y secuencial. En el convertidor Buck se generó una nube de condiciones iniciales y una nube de parámetros, tanto las condiciones iniciales como los valores de parámetros fueron enviados de manera paralela a un conjunto de hilos de la GPU, cada hilo provisto de algún algoritmo de integración.

En los dominios de atracción se generaron nubes de puntos (condiciones iniciales). Cada condición inicial fue enviada a un hilo de ejecución diferente, el cual disponía de un Kernel con un algoritmo de integración. En este estudio se utilizaron los métodos de Runge-Kutta-Fehlberg y Euler [24]. Los procedimientos para hallar los dominios de atracción fueron los mismos en el sistema de Lorenz y en el convertidor Buck. A continuación se muestra el código fuente de un Kernel para hallar dominios de atracción:

```
1  __global__ void rk45Cuda(double *y,int *etiqueta){
2  double q1[]={sqrtf(w),sqrtf(w),r-1};
3  int index=blockIdx.x*blockDim.x+threadIdx.x;
4  rk45(20,y,0.001,index);
5  double norma ,tol=1;
6  norma=sqrt(pow(y[index]-q1[0],2)+pow(y[index+N]...
7  -q1[1],2)+pow(y[index+2*N]-q1[2],2));
8  if(norma<tol){
9      etiqueta[index]=1;
10 }
11 else{
12     etiqueta[index]=2;
13 }
14 }
```

Los tiempos se midieron utilizando las librerías cudaEventRecord y cudaEventElapsedTime. La primera se encarga de detectar los eventos de inicio y finalización de porciones de código, y la segunda de medir el tiempo entre los dos eventos mencionados anteriormente. La estructura de los

algoritmos involucra: asignación de memoria y declaración de variables en CPU y GPU, declaración de condiciones y parámetros iniciales, copia de información de la CPU a la GPU, ejecución de Kernel en la GPU, copia de información de la GPU a la CPU y liberación de memoria en CPU y GPU. La figura **2-5** ilustra la metodología de paralelización empleada.

Los tiempos de cada una de las estructuras fueron medidos independientemente, y el tiempo total de cada simulación fue la suma de cada uno de los tiempos. Para obtener medidas más precisas, cada experimento se repitió 10 veces y se tomó el promedio.

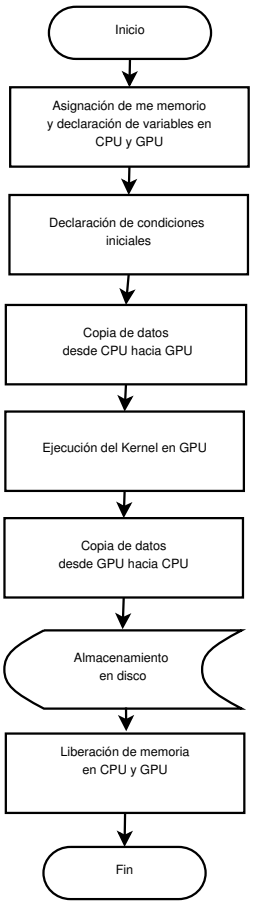


Figura 2-5: Diagrama de flujo de los algoritmos en GPU

Las GPU utilizadas para resolver el problema están compuestas de una cantidad definida de bloques y a su vez cada bloque posee una determinada cantidad de hilos (ver Tabla **2-1**). El objetivo de la paralelización es definir inicialmente una cantidad de bloques e hilos acorde a la cantidad de condiciones iniciales generadas por los límites de la malla y la división de la misma, posteriormente se define un Kernel que posee el método numérico que se paralelizó y a dicho Kernel se asigna la cantidad de bloques e hilos suficientes para satisfacer la cantidad de condiciones iniciales.

Cada Kernel se paralelizó con una indexación de memoria en una dimensión, es decir, todas las condiciones iniciales que estaban en dos o tres dimensiones según el problema se adaptaron a un solo arreglo y se creó un hilo para cada condición inicial.

Tabla 2-1: Especificaciones de las GPU

Specification	GTX 770	Quadro 4000
CUDA Driver Version / Runtime Version	7.5 / 7.0	7.5 / 7.0
CUDA Capability Major/Minor version number	3.0	2.0
Total amount of global memory	2048 MBytes	2045 MBytes
Multiprocessors	8	8
CUDA Cores for multiprocessor	192	32
Total CUDA cores	1536	256
GPU Max Clock rate	1163 MHz	950 MHz
Memory Clock rate	3505 Mhz	1404 Mhz
Memory Bus Width	256-bit	256-bit
L2 Cache Size	524288 bytes	524288 bytes
Total amount of constant memory	65536 bytes	65536 bytes
Total amount of shared memory per block	49152 bytes	49152 bytes
Total number of registers available per block	65536	32768
Maximum number of threads per multiprocessor	2048	1536
Maximum number of threads per block	1024	1024
Max dimension size of a thread block (x,y,z)	(1024, 1024, 64)	(1024, 1024, 64)
Max dimension size of a grid size (x,y,z)	(2147483647, 65535, 65535)	(65535, 65535, 65535)

2.4. Resultados

Todas las simulaciones que se llevaron a cabo se repitieron 10 veces para dar confiabilidad a los resultados y el tiempo que se registró de cada simulación fue el promedio de las 10 repeticiones. Además, una misma simulación se hizo con las dos tarjetas y se registraron sus resultados. Es importante aclarar que los tiempos de simulación no contemplan la escritura del archivo en disco. Las mallas utilizadas en el sistema de Lorenz para todas las simulaciones poseen los siguientes límites X(-1,1), Y(-1,1) Z(-1,1) y con una división de 0.1, 0.05 o 0.01. Para el convertidor Buck se usaron los siguientes límites X(4.9,6.1), Y(0,1.3) y con una división de malla de 0.1, 0.05 o 0.01.

Las simulaciones del dominio de atracción de Lorenz y el convertidor Buck, ambas en dos dimensiones, también fueron implementadas en lenguaje C de una manera secuencial para hacer las comparaciones con respecto a su implementación en paralelo.

2.4.1. Sistema de Lorenz

Análisis de bifurcaciones

Se fijaron los valores de los parámetros σ en 10 y b en $8/3$, y se varió el parámetro r entre 0 y 110. Por cada valor de parámetro se simularon diferentes cantidades de condiciones iniciales durante $50seg$. La figura **2-6** ilustra un digrama de bifurcaciones donde se varió el parámetro r en pasos de 10 y se tomaron 68921 condiciones iniciales por valor de parámetro. Se puede observar como,

a medida que el parámetro incrementa los equilibrios, se transforman en órbitas caóticas. Cuando r vale 28 se presenta el atractor caótico de la figura **2-7**. En la figura **2-8** se ilustra una bifurcación donde r cambia en pasos de 1, se tomaron 68921 condiciones iniciales por valor de parámetro.

La Tabla **2-2** compara los tiempos requeridos por la CPU y las dos GPU para calcular los diagramas de bifurcaciones usando dos aproximaciones numéricas distintas (Runge-Kutta y Euler) como también las respectivas aceleraciones de GPU vs CPU para cada aproximación numérica.

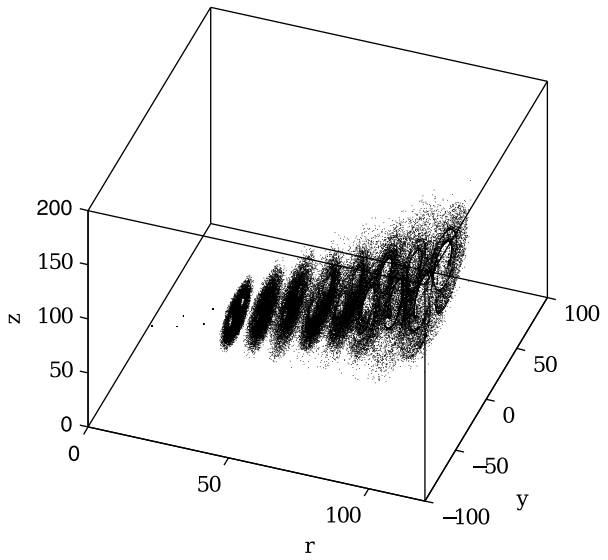


Figura 2-6: Diagrama de bifuraciones del sistema de Lorenz.

Tabla 2-2: Comparación de tiempos y aceleraciones para CPU y GPU de los diagramas de bifurcaciones del sistema de Lorenz.

Sistema Analizado	Aprox numérica	Cantidad condi iniciales	Tiempo CPU [seg]	Tiempo GPU Quadro 4000 [seg]	Speed-Up Quadro 4000	Tiempo GPU GTX 770 [seg]	Speed-Up GTX 770
Lorenz 3D	RK 45	89326611	33588,9057	1631,5268	**20,6**	862,3762	**38,9**
		7581310	2848,9216	154,5276	**18,4**	76,2641	**37,4**
		1018710	413,5106	22,7112	**18,2**	12,0689	**34,3**
	Euler	89326611	14499,4004	154,6888	**93,7**	207,6324	**69,8**
		7581310	1276,5272	13,3797	**95,4**	17,9002	**71,3**
		1018710	410,1122	1,9257	**213,0**	2,6319	**155,8**

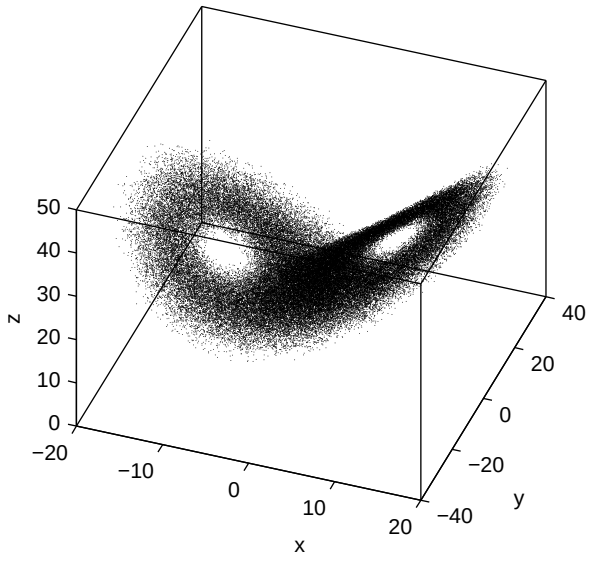


Figura 2-7: Atractor de Lorenz.

Dominios de atracción

Para los parámetros $\sigma = 10$, $r = 13$ y $b = 8/3$ se presentan tres equilibrios. Uno de ellos ubicado en el origen y de naturaleza inestable. Los otros dos estables ubicados en:

$$\begin{aligned} x_1^* &= \begin{bmatrix} \sqrt{b\,(r-1)} & \sqrt{b\,(r-1)} & r-1 \end{bmatrix} \\ x_2^* &= \begin{bmatrix} -\sqrt{b\,(r-1)} & -\sqrt{b\,(r-1)} & r-1 \end{bmatrix} \end{aligned} \tag{2-6}$$

Para los valores de parámetros utilizados los equilibrios son:

$$\begin{aligned} x_1^* &= \begin{bmatrix} 5{,}6569 & 5{,}6569 & 12 \end{bmatrix} \\ x_2^* &= \begin{bmatrix} -5{,}6569 & -5{,}6569 & 12 \end{bmatrix} \end{aligned} \tag{2-7}$$

Los dominios de atracción se hicieron tomando 68921 condiciones iniciales para un valor determinado de parámetros, se tomó cada condición inicial y se simuló durante $20seg$, tiempo para el cual el sistema ya se encuentra en estado estacionario. La figura **2-9** ilustra un dominio de atracción sobre el plano xy. La región negra corresponde a las condiciones iniciales que tienden al equilibrio x_1^*. La región blanca corresponde a las condiciones iniciales que tienden al equilibrio x_2^*. La figura **2-10** ilustra el mismo dominio de atracción sobre el plano xyz. La región azul corresponde a las condiciones iniciales que tienden al equilibrio x_1^*. La región roja corresponde a las condiciones iniciales que tienden al equilibrio x_2^*. Por último, la figura **2-11** muestra la frontera entre los dominios de atracción de los dos equilibrios estables que aparecen en este sistema.

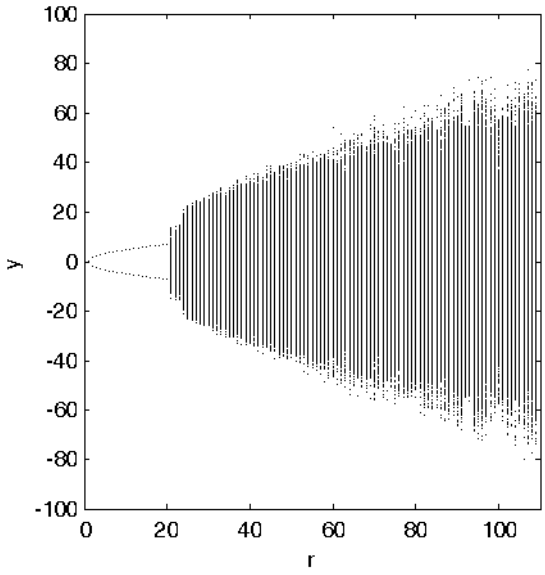


Figura 2-8: Diagrama de bifuraciones del sistema de Lorenz.

Todos los cálculos fueron realizados tanto en CPU como en GPU para dos aproximaciones numéricas distintas (Runge-Kutta y Euler). La Tabla **2-3** muestra los diferentes tiempos obtenidos por la CPU y las dos GPU como también las respectivas aceleraciones de GPU vs CPU para cada aproximación numérica.

Tabla 2-3: Comparación de tiempos y aceleraciones para CPU y GPU de los dominios de atracción del sistema de Lorenz en 2D y 3D.

Sistema Analizado	Aprox numérica	Cantidad condi iniciales	Tiempo CPU [seg]	Tiempo GPU Quadro 4000 [seg]	Speed-Up Quadro 4000	Tiempo GPU GTX 770 [seg]	Speed-Up GTX 770
Lorenz 2D	RK 45	40401	4,9634	0,0565	**87,8**	0,0704	**70,5**
		1681	0,2146	0,0040	**54,1**	0,0056	**38,6**
		484	0,0618	0,0029	**21,3**	0,0035	**17,5**
	Euler	40401	11,6149	0,0656	**177,0**	0,1042	**111,5**
		1681	0,4782	0,0039	**124,0**	0,0057	**84,6**
		484	0,1461	0,0024	**61,0**	0,0042	**35,0**
Lorenz 3D	RK 45	8120601	977,9167	9,5693	**102,2**	11,9512	**81,8**
		68921	8,3023	0,0992	**83,7**	0,1235	**67,2**
		9261	1,2852	0,0154	**83,3**	0,0205	**62,6**
	Euler	8120601	2382,4525	12,8744	**185,1**	18,9107	**126,0**
		68921	19,3553	0,1120	**172,9**	0,1707	**113,4**
		9261	3,1401	0,0167	**187,9**	0,0254	**123,6**

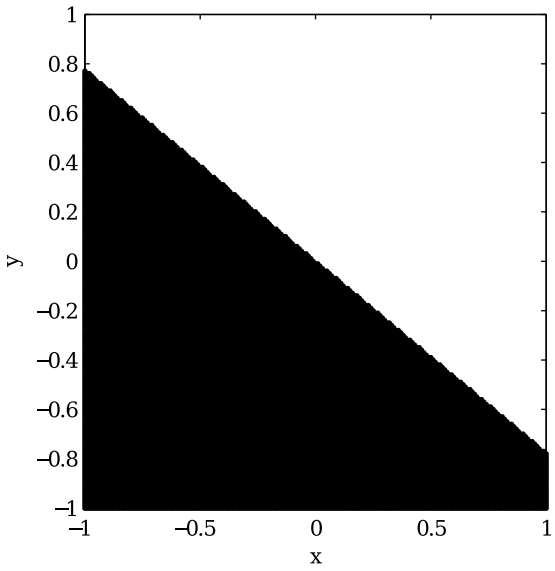


Figura 2-9: Dominios de atracción del sistema de Lorenz. $z = 0$

2.4.2. Convertidor Buck controlado por onda seno

Análisis de Bifurcaciones

Los siguientes valores de parámetros fueron usados: $V_{in} = 15V$, $R = 75\Omega$, $L = 0{,}02H$, $C = 470\mu F$, $V_{ref} = 5V$, $A = 8{,}4$, $T = 400\mu S$, $V_l = 308V$ y $V_u = 8{,}2V$. El parámetro R se varió entre 100Ω y 200Ω. En la figura **2-12** se observa un diagrama de bifurcaciones del convertidor Buck controlado por onda seno. Para realizar este diagrama se tomó un conjunto de condiciones iniciales y parámetros y se simuló de manera paralela durante 200 períodos de la onda que fuerza al sistema, es decir $80ms$.

La Tabla **2-4** muestra los diferentes tiempos obtenidos por la CPU y las dos GPU en el cálculo de los diagramas de bifurcaciones con diferentes conjuntos de condiciones iniciales y parámetros para dos aproximaciones numéricas diferentes (Runge-Kutta y Euler), como también la aceleración de GPU vs CPU.

Tabla 2-4: Comparación de tiempos y aceleraciones para CPU y GPU de los diagramas de bifurcaciones del convertidor Buck.

Sistema Analizado	Aprox numérica	Cantidad condi iniciales	Tiempo CPU [seg]	Tiempo GPU Quadro 4000 [seg]	Speed-Up Quadro 4000	Tiempo GPU GTX 770 [seg]	Speed-Up GTX 770
Conver Buck	RK 45	1585100	8808,6391	1578,0584	**5,6**	496,0478	**17,8**
		67500	382,5439	148,4576	**2,6**	22,2641	**17,2**
		18200	102,6502	49,9459	**2,1**	6,7658	**15,2**
	Euler	1585100	18085,7481	1378,0264	**13,1**	1297,6260	**13,9**
		67500	766,9481	121,4366	**6,3**	111,3326	**6,9**
		18200	211,3079	69,5478	**3,0**	68,7467	**3,1**

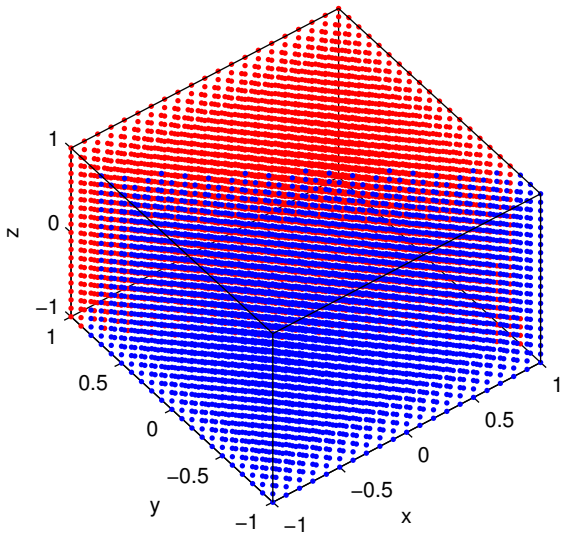


Figura 2-10: Dominios de atracción del sistema de Lorenz.

Dominios de atracción

En la figura **2-12** se observa que hay coexistencia de soluciones. Para $R = 140\Omega$, coexiste una solución uno-periódica con otra solución que bien puede ser cuasi-periódica o caótica. La figura **2-13** ilustra un dominio de atracción donde las regiones en negro pertenecen a las condiciones iniciales que tienden a la órbita uno-periódica. Para realizar esta simulación se tomó un conjunto de condiciones iniciales con unos valores de parámetros fijos y se simuló de manera paralela durante 200 períodos de la onda que fuerza al sistema, es decir $80ms$. La Tabla **2-5** muestra los diferentes tiempos obtenidos por la CPU y las dos GPU en el cálculo de los dominios de atracción con diferentes conjuntos de condiciones iniciales para dos aproximaciones numéricas diferentes (Runge-Kutta y Euler), como también la aceleración de GPU vs CPU.

Tabla 2-5: Comparación de tiempos y aceleraciones para CPU y GPU de los dominios de atracción del convertidor Buck.

Sistema Analizado	Aprox numérica	Cantidad condi iniciales	Tiempo CPU [seg]	Tiempo GPU Quadro 4000 [seg]	Speed-Up Quadro 4000	Tiempo GPU GTX 770 [seg]	Speed-Up GTX 770
Conver Buck	RK 45	15851	53,3021	14,6651	**3,6**	14,2550	**3,7**
		675	2,1848	1,7354	**1,3**	1,2523	**1,7**
		182	0,6035	1,0923	**0,6**	0,7074	**0,9**
	Euler	15851	104,4789	9,4035	**11,1**	8,9034	**11,7**
		675	4,1597	1,4583	**2,9**	1,3587	**3,1**
		182	0,9992	0,5962	**1,7**	0,5822	**1,7**

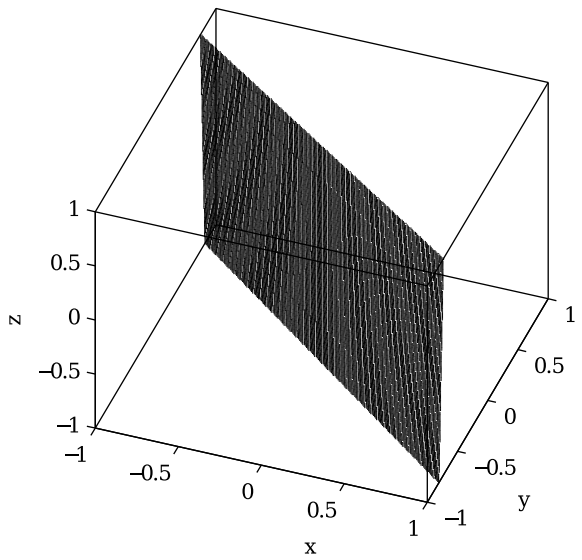


Figura 2-11: Frontera entre los dominios de atracción del sistema de Lorenz.

2.5. Conclusiones

En este capítulo se presentó el análisis de bifurcaciones y los dominios de atracción del sistema de Lorenz y del convertidor Buck controlado por onda seno. Se utilizaron algoritmos de fuerza bruta ejecutados de manera paralela en dos GPU, una GTX 770 y una Quadro 4000, con dos aproximaciones numéricas distintas (Runge-Kutta y Euler). Todos los experimentos se replicaron en una CPU de manera secuencial. En general, las mejores velocidades de procesamiento las presentó la tarjeta Quadro 4000. Es de aclarar que la tarjeta GTX 770 posee mejores prestaciones en hardware que la Quadro 4000, Sin embargo, esta última opera números flotantes de doble precisión con mayor eficiencia que la GTX. Todos los programas fueron desarrollados usando variables flotantes de doble precisión.

En cada una de las simulaciones se generaron nubes de condiciones iniciales y parámetros cada una enviada a un hilo, cada hilo se encargó de resolver el sistema no lineal (Convertidor Buck – Sistema de Lorenz) por medio de un método de integración, por lo que la cantidad de condiciones iniciales debe ser controlada para que la GPU soporte el proceso. En este capítulo se trabajó máximo con $89{,}326{,}611$ condiciones iniciales y parámetros simulados al tiempo, para futuras investigaciones si se desea hacer simulaciones con nubes superiores a las mencionadas se recomienda utilizar multi-GPU o una GPU con mayores prestaciones para así poder procesar una mayor cantidad de condiciones iniciales y/o parámetros simultáneamente y mejorar los tiempos de simulación.

Las tarjetas GTX 770 y Quadro 4000 se escogieron dados sus costos relativamente bajos en el mercado. De esta manera se pudieron obtener aceleraciones en cómputos científicos con dispositivos

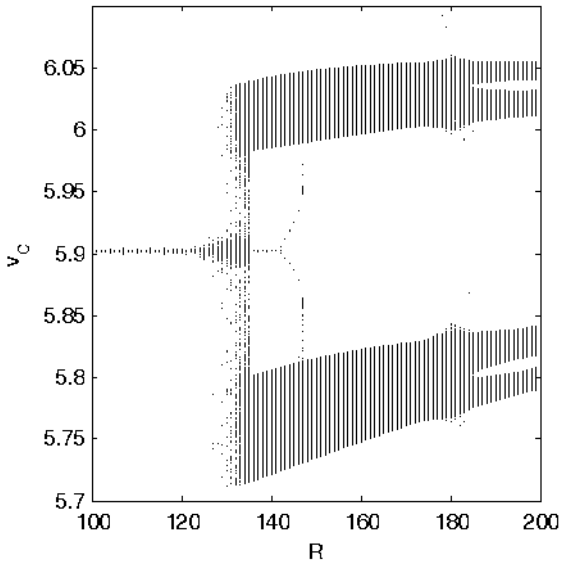


Figura 2-12: Diagrama de bifurcaciones del convertidor Buck.

relativamente económicos y de fácil acceso.

Las tarjetas presentaron tiempos de ejecución más cortos y aceleraciones más grandes en los experimentos numéricos relacionados con el sistema de Lorenz. Cabe resaltar que el convertidor Buck es un sistema conmutado y por lo tanto los algoritmos de integración numérica contaban con rutinas de detección de eventos, lo cual hace a las simulaciones más costosas computacionalmente.

Se observa que para las cantidades menores de datos, las aceleraciones son pequeñas o inexistentes; debido a que los tiempos de simulación contemplan tiempos de inicialización de variables, liberación de memoria, copia de información de CPU a GPU, copia de información de GPU a CPU y ejecución del Kernel. Este último tiempo es el que verdaderamente representa la aceleración del problema, debido a que ésta es la operación que se hace paralela; enviando a cada hilo de la GPU un proceso determinado. Cuando la cantidad de datos es muy baja, los tiempos de ejecución del Kernel son comparables con los demás tiempos mencionados anteriormente, mientras que para grandes cantidades de datos la mayor parte del tiempo de simulación se concentra en la ejecución del Kernel.

Para el sistema de Lorenz se observa que las mayores aceleraciones se obtuvieron con la tarjeta Quadro 4000 y con la aproximación numérica de Euler tanto para las bifurcaciones como para los dominios de atracción, mientras que para el convertidor Buck las mejores aceleraciones para las bifurcaciones se obtuvieron con la Tarjeta GTX 770 y con la aproximación numérica Runge-Kutta y para los dominios de atracción se obtuvieron con la misma tarjeta pero con el método de Euler.

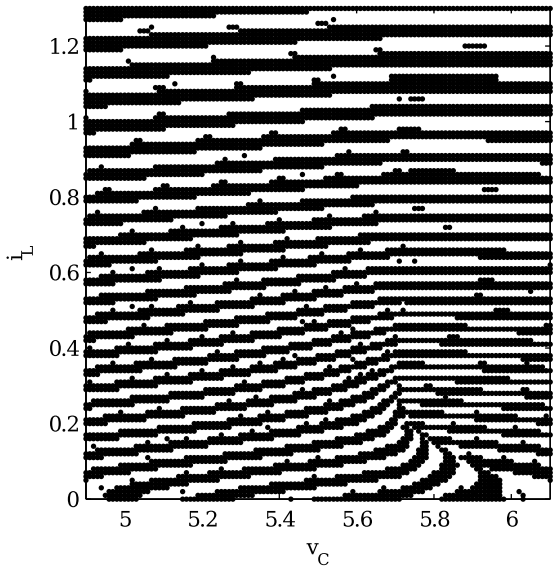


Figura 2-13: Dominios de atracción del convertidor Buck.

Se observan aceleraciones significativas usando indexación unidimensional a pesar de que los problemas tratados son de dos o tres dimensiones. Se esperaría, para futuras investigaciones, obtener mejores resultados utilizando un tipo de indexación en bi o tri dimensional que actualmente es soportada por las GPU.

Las tarjetas presentan mejores desempeños en el sistema de Lorenz que en el convertidor Buck. Esto se debe a que el convertidor posee un Kernel más costoso computacionalmente ya que el método numérico que se paralelizó involucra rutinas de integración y de detección de eventos.

Es importante aclarar que las dos tarjetas que se usaron para la simulaciones son de gama media. Además las dos grandes empresas ensambladoras de GPU (Nvidia y AMD) están en constante actualización de sus arquitecturas, sacando cada vez al mercado tarjetas que superan considerablemente a las anteriores, por consiguiente es importante aprender a manejar este tipo de paradigma de programación y las herramientas que ellos facilitan para tratar con problemas que de una manera secuencial serían casi imposibles de resolver.

Es necesario tener un amplio conocimiento del problema a tratar y de la arquitectura de la tarjeta que se va a usar tratando de adaptar el problema al de una imagen, debido a que si el problema no es susceptible de una estrategia paralela o posee alta dependencia y además no se hace una correcta indexación de la memoria se podrían generar tiempos de simulación más altos que los que se obtendrían de una manera secuencial u obtener resultados erróneos por una sobre escritura en

memoria.

3 Indexación uni-dimensional y bi-dimensional empleando GPU para acelerar la aproximación numérica de soluciones de la ecuación de Laplace

La computación de alto rendimiento está jugando un papel determinante en el campo de la computación científica, debido a que ésta presenta retos cada vez más grandes en cuanto a la cantidad de datos requeridos por las soluciones, y por ende al tiempo de procesamiento de los algoritmos [4]. Sin embargo, los elevados costos de equipos de altas prestaciones necesarios para resolver estos problemas de forma eficiente han generado un gran interés en la comunidad científica por la búsqueda de alternativas más económicas. Entre las posibilidades que se están investigando actualmente se encuentra el uso de la computación heterogénea. Al utilizar computación heterogénea se pretende emplear unidades computacionales clásicas tipo CPU y combinarlas con otro tipo de dispositivos que permitan descargar sobre ellos la mayor parte de la carga computacional [58]. Para lograr el máximo desempeño de una unidad aceleradora, la estructura del problema se debe adaptar a la estructura del dispositivo [10]. En este caso, el método seleccionado para la solución de la ecuación de Laplace es muy similar a la forma en que se realiza el procesamiento de imágenes, por este motivo se consideró utilizar como dispositivo acelerador una unidad de procesamiento gráfico (GPU) [41, 12], además existen dispositivos con una arquitectura especialmente diseñada para el procesamiento de grandes volúmenes de información con propósito científico [5].

Lamentablemente, el utilizar un dispositivo acelerador como la GPU no garantiza inmediatamente la aceleración del proceso. Para lograrlo se debe realizar la adaptación del problema a la estructura del dispositivo y minimizar los cuellos de botella generados por las numerosas transferencia de información requeridas. Por lo tanto, ya que las GPU están optimizadas para realizar el procesamiento de imágenes mediante el uso de indexación bidimensional [20], se plantea como estrategia adaptar el problema a este sistema de indexación en dos dimensiones [29]. Uno de los grandes retos que tiene la implementación de la indexación bidimensional es la dependencia de la arquitectura de la GPU [42], se pueden lograr distintas aceleraciones (*Speed*-Up) basadas en indexación dependiendo del fabricante, por lo cual se analizaron cada una de las arquitecturas disponibles y se realizaron diferentes pruebas utilizando procesos con indexación y así poder de determinar las características óptimas de los dispositivos según el problema a solucionar, o ajustar las caracterís-

ticas del problema a los dispositivos aceleradores con los que se cuenta.

Aún teniendo definido el tipo de dispositivo acelerador y la estrategia que se emplea para buscar una aceleración se presenta un nuevo interrogante, ¿qué tipo de arquitectura interna es la mejor para utilizar? para poder tener una respuesta a esta pregunta se mostrarán algunas diferencias de rendimiento presentes en varias arquitecturas de GPU.

En la sección 3.1 se presenta el planteamiento del problema de la ecuación de Laplace. En la sección 3.2 se explica la descripción del experimento. En la sección 3.3 se muestran los resultados obtenidos de la implementación de la ecuación de Laplace en GPU. Finalmente, en la sección 3.4, se presentan algunas conclusiones y observaciones.

3.1. Planteamiento del problema

Los problemas modelados con ecuaciones diferenciales parciales frecuentemente no poseen una solución analítica conocida, por consiguiente se han desarrollado métodos numéricos que permiten aproximar una solución. Sin embargo, estos métodos necesitan una gran cantidad de datos y de cálculos para obtener una aproximación satisfactoria, lo que hace necesario el uso de herramientas computacionales que permitan realizar estos cálculos científicos de forma eficiente.

La ecuación de Laplace (3-1) modela el comportamiento de una variable en estado estacionario para una región determinada [90]. Problemas relacionados con campos vectoriales tales como la propagación de la temperatura y la distribución de campos magnéticos o gravitatorios son casos típicos en los que se usa la ecuación de Laplace para modelar los fenomenos físicos asociados a cada uno de los casos [45]. En este capítulo se pretende encontrar la temperatura en estado estacionario de una placa metálica aislada excepto en sus bordes donde posee una temperatura preestablecida y constante (condiciones de frontera), además no hay pérdidas de calor. En la figura **3-2** se puede apreciar las condiciones térmicas que cumple el problema en sus estados iniciales.

En particular el Método de Diferencias Finitas discretiza las coordenadas espaciales del problema en una malla de puntos individuales, que a su vez tienen una dependencia de sus vecinos en la malla como se muestra en la figura **3-1** [8, 21]. Este método brinda una buena aproximación de la solución de la ecuación de Laplace y, como se puede observar, existe una gran similitud con la arquitectura presente en una GPU [23].

$$\Delta T = \frac{(\partial^2 T)}{\partial x^2} + \frac{(\partial^2 T)}{\partial y^2} = 0 \tag{3-1}$$

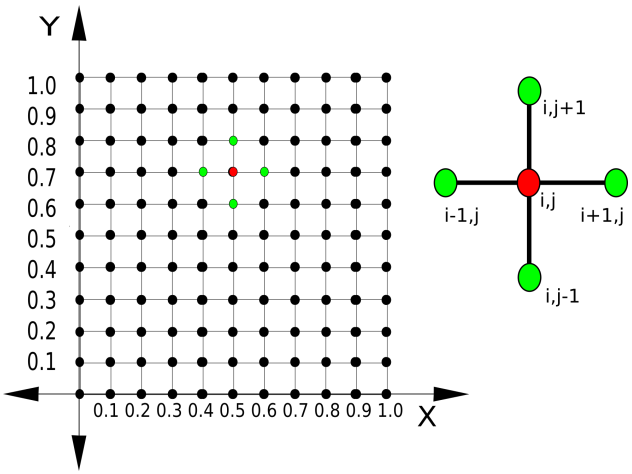


Figura 3-1: Discretización del espacio del problema y dependencia de los datos cercanos.

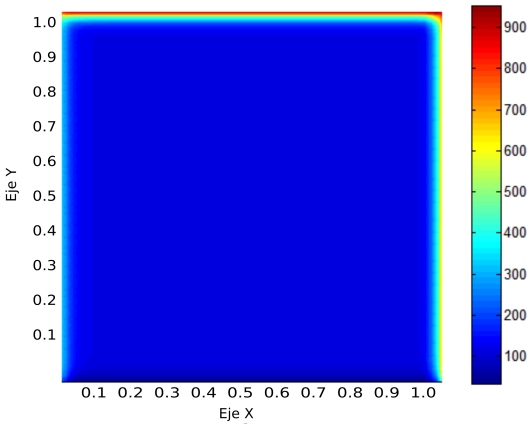


Figura 3-2: Condiciones inciales del problema planteado para la ecuación de Laplace.

Después de definir el problema a solucionar y el método numérico a utilizar, también se debe considerar el algoritmo que se implementará para obtener la solución del sistema de ecuaciones. Dado que es un problema independiente del tiempo es posible utilizar el método de Gauss Seidel, conocido como método de Liebmann [67, 83] cuando se aplica en ecuaciones diferenciales parciales. Este método consiste en calcular cada punto de la malla como un promedio de sus vecinos espaciales como se muestra en la ecuación (3-2), asumiendo que la malla es cuadrada, e iterando cada punto hasta que no se produzca un cambio considerable en la temperatura, otro criterio de parada.

$$T_1(i-1,j) = a;$$
$$T_1(i+1,j) = b;$$
$$T_1(i,j-1) = c;$$
$$T_1(i,j+1) = d;$$

$$T_2(i,j) = \frac{(a+b+c+d)}{4} \tag{3-2}$$

Para implementar en los diferentes dispositivos el algoritmo descrito se utilizan las librerías de OpenCL. Estas fueron creadas como un estándar de computación paralela que permiten el dominio de un amplio número de dispositivos aceleradores, permitiendo además la escritura de código portable [93]. Aunque la portabilidad no es garantía de un buen desempeño en las diferentes plataformas, permite contar con una implementación base de los algoritmos, para luego centrarse en ajustar adecuadamente las propiedades de cada dispositivo.

Para lograr rendimientos óptimos es necesario conocer la arquitectura interna de los dispositivos aceleradores y su relación con las estructuras de los problemas a solucionar. También es importante anotar, que al igual que con otras librerías, se requiere un consumo de recursos adicional para dividir el problema en etapas que puedan ser ejecutadas de forma concurrente.

Teniendo en cuenta la geometría del problema, se utiliza el método de indexación bidimensional disponible en la GPU que permite asignar índices a los datos como si se tratara de los elementos de una imagen, y almacenarlos en la GPU de la misma forma en que están dispuestos. Así es posible conseguir una interacción más veloz en el momento de realizar cálculos entre datos cercanos. Las ventajas del proceso de indexación radican en las características de almacenamiento en memoria, mientras que el proceso normal de almacenamiento se realiza en forma unidimensional, la estructura interna de la GPU puede distribuir la información de forma bi-dimensional [20]. Al permitir una distribución bi-dimensional, es posible, mediante índices de posición horizontal y vertical, lograr que las operaciones que involucran vecinos espaciales se realicen de una manera más rápida.

3.2. Descripción del experimento

Para comprobar el rendimiento que ofrecen los diferentes tipos de arquitecturas, se propone ejecutar los algoritmos para las distintas unidades de procesamiento gráfico GPU, aprovechando los

beneficios que brinda OpenCL en cuanto a portabilidad. El problema específico que se va a tratar es la aplicación de la ecuación de Laplace en 2 dimensiones para resolver el problema de propagación de la temperatura en una placa cuadrada con diferentes tamaños de discretización. El objetivo es medir los tiempos de simulación y concluir para qué arquitectura y para qué tamaño de malla se obtienen los mejores resultados. Con el fin de medir únicamente el tiempo que toma el proceso en ejecutarse, no se escribirá la información en disco. Además, para una de las GPU se tomarán medidas de los tiempos de ejecución de cada una de las secciones del proceso para los distintos tamaños de malla.

Las arquitecturas que se desean comparar son VLIW 3D, Kepler y Maxwell. La arquitectura VLIW 3D es fabricada por AMD. Kepler y Maxwell son fabricadas por Nvidia. La comparación entre estas tres arquitecturas se realizará mediante la simulación del mismo problema codificado en paralelo en OpenCL. Adicionalmente se contrastarán estos resultados con los obtenidos al ejecutar el mismo código de manera secuencial en una CPU. Las tarjetas que se utilizarán para las pruebas son 3: Nvidia GTX 770 de arquitectura Kepler, AMD Radeon HD 6770 con arquitectura VLIW 5D y Nvidia GTX 860M con arquitectura Maxwell. Cronológicamente las arquitecturas de la más nueva a la más antigua son Maxwell, Kepler y VLIW 5D. Estas tienen las siguientes capacidades en memoria: Nvidia GTX 770 2GB, la AMD Radeon HD 6770 1GB y la Nvidia GTX 860M 2GB; todas ellas son GDDR5 lo cual indica que la AMD es la de menor capacidad de almacenamiento de información.

Las tarjetas Nvidia GTX 770 (GPU-Kepler) y GTX 860M (GPU-Maxwell) trabajan sobre la tecnología PCI-E 3.0x16 y la arquitectura de sus procesadores es de 28 nanómetros, mientras que la AMD Radeon HD 6770 (GPU-AMD) lo hace sobre PCI-E 2.0x16 y la arquitectura de sus procesadores es de 40 nanómetros. Lo anterior tiene directa relación con la velocidad a la que se pueden realizar las transferencias entre la Board y las GPU, dejando a la tarjeta AMD y los procesos al interior de la misma como la menos veloz. El ancho de banda y el bus de memoria de 224 GB/-seg, 256 bit para la GPU-Kepler, 76.8 GB/seg, 128 bit para la AMD y 80 GB/seg, 128 bit para la GPU-Maxwell. Además la velocidad efectiva de la memoria y la cantidad de operaciones en punto flotante son 7012MHz, 3,213 GFLOPS para la GPU-Kepler, 4800MHz, 1,36 GFLOPS para la GPU-AMD y 5012MHz, 1,306 GFLOPS para la GPU-Maxwell. Por último la tarjeta GPU-Kepler posee 1536 unidades de proceso mientras que la GPU-AMD posee 800 y la GPU-Maxwell 640, la información restante puede ser encontrada en la Tabla **3-6**. El experimento se realizará en un computador cuyo sistema operativo es Windows 7 usando la plataforma de desarrollo Visual Studio 2012, procesador Intel I-7 4820K de 3.7GHZ y 16 GB de memoria RAM DDR3 con Bus de 1600 MHZ para las GPU externas, y con un procesador Intel I-7 4710HQ y 12 GB de RAM DDR3 para la GPU-Maxwell que es tipo laptop. Cada simulación se repitirá 10 veces y al final se promediarán los tiempos con el fin de darle confiabilidad y precisión a los resultados.

El experimento se realizó en las diferentes tarjetas, asumiendo placas cuadradas de tamaños:$127 *$

127, $255*255$, $511*511$, $1023*1023$, $2047*2047$ y $4095*4095$. De cada placa se calcularon las aceleraciones tomando como base los tiempos obtenidos por el programa secuencial en la CPU. En las Tablas **3-1** y **3-2** se puede observar el comportamiento del tiempo de respuesta con respecto a los diferentes tamaños de malla. Para todas las ejecuciones, el criterio de parada de las iteraciones es igual a un error de tolerancia de 6 cifras significativas, observándose, de manera general, una cantidad de iteraciones muy cercana al tamaño de una dimensión en cada una de las mallas. Los resultados se pueden ver en las Tablas **3-1** y **3-2** donde todos los tiempos están dados en segundos. Las Figuras **3-4** y **3-5** muestran de forma más clara las aceleraciones obtenidas. Esta información puede ser utilizada para determinar cuáles son los estados óptimos de cada una de las arquitecturas para obtener los tiempos de procesamiento más bajos.

Posteriormente se tomaron las medidas de tiempos discriminadas en cada una de las secciones del proceso. Para la tarjeta Nvidia de arquitectura Kepler, de la misma forma en que se procedió en el caso anterior. Los resultados se muestran en la Tabla **3-4**, en donde se consideran las secciones con mayor porcentaje de impacto en el tiempo total. En la figura **3-7** se muestran los resultados de forma porcentual para observar las partes de mayor consumo computacional según el tamaño de la malla. También se realizó un experimento sin extraer los datos de iteraciones intermedias para evitar el cuello de botella que implica la transferencia de información entre GPU y CPU. Para realizar este cambio y conservar la precisión de los datos, se hizo necesario incorporar semáforos al código, que evitaran sobre-escritura de la memoria de la GPU cuando ésta aún estuviera procesando información. Este cambio se puede ver en la Tabla **3-3** y en la figura **3-6**. Para este experimento no se involucró la GPU-Kepler debido a que ésta no soportaba el uso de este tipo de semáforos.

Finalmente, se realizaron pruebas utilizando una indexación unidimensional en la GPU-Maxwell, con el fin de comprobar si el método bidimiensional genera beneficios en los tiempos de ejecución del proceso. Los resultados se muestran en la Tabla **3-5**, y se pueden contrastar en la figura **3-8**, que muestra los tiempos de ejecución totales de la misma GPU utilizando indexación en una y dos dimensiones.

A continuación se presentan los Kernel usados en la experimentación utilizando librerías de OpenCL
.

3.2.1. Kernel de ejecución para la ecuación de Laplace sin barreras

```
1 __Kernel void Kernel(
2   const int Ndim,
3   __global float* A, __global float* C )
4 {
5     int i,j;
6     float tmp=0;
7     i = get_global_id(0);
8     j = get_global_id(1);
9     tmp+=A[((i-1)*Ndim)+j];
```

```
10      tmp+=A[((i+1)*Ndim)+j];
11      tmp+=A[(i*Ndim)+(j-1)];
12      tmp+=A[(i*Ndim)+(j+1)];
13      C[i*Ndim+j]=tmp/4;
14      tmp=0;
15     };
```

3.2.2. Kernel de ejecución para la ecuación de Laplace en 1 dimensión

```
1  __Kernel void Kernel(
2    const int Ndim,
3    __global float* A, __global float* C )
4  {
5      int i,j;
6      float tmp=0;
7      i = get_global_id(0);
8    tmp+=A[(i-1)];
9    tmp+=A[(i+1)];
10     tmp+=A[(i-Ndim)];
11   tmp+=A[(i+Ndim)];
12     C[i]=tmp/4;
13     }
14    };
```

3.2.3. Kernel de ejecución para la ecuación de Laplace con barreras

```
1  __Kernel void Kernel(
2    const int Ndim,
3    __global float* A, __global float* C )
4  {
5      int i,j;
6      float tmp=0;
7      i = get_global_id(0);
8      j = get_global_id(1);
9      if( i!=Ndim-1 && j!=Ndim-1
10     && i!=0 && j!=0)
11     {
12     tmp+=A[((i-1)*Ndim)+j];
13     tmp+=A[((i+1)*Ndim)+j];
14     tmp+=A[(i*Ndim)+(j-1)];
15     tmp+=A[(i*Ndim)+(j+1)];
16     barrier(CLK_LOCAL_MEM_FENCE
17     | CLK_GLOBAL_MEM_FENCE);
18     C[i*Ndim+j]=tmp/4;
19     tmp=0;
20     barrier(CLK_LOCAL_MEM_FENCE
21     | CLK_GLOBAL_MEM_FENCE);
22     A[i*Ndim+j]=C[i*Ndim+j]
23     }
24    };
```

En la figura **3-3** se observa el diagrama de flujo del algoritmo utilizado para resolver la ecuación de Laplace.

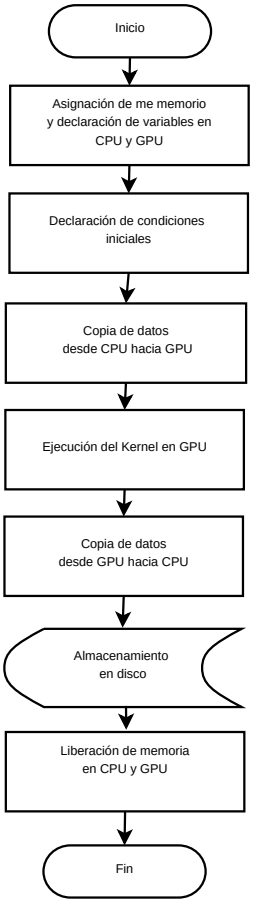


Figura 3-3: Diagrama de flujo de los algoritmos en GPU

3.3. Resultados

Las Figuras **3-4** y **3-5** muestran que las aceleraciones obtenidas dependen del tamaño de malla en forma no lineal. Además en las Tablas **3-1** y **3-2** se puede observar que, para tamaños de malla pequeños no se obtienen aceleraciones y a medida que aumenta la cantidad de datos del proceso, las aceleraciones dadas por las diferentes GPU aumentan hasta llegar a un punto critico a partir del cual empiezan a disminuir.

Los datos obtenidos en la GPU-Kepler muestran los mejores resultados de velocidad en mallas de mayor tamaño, no sólo debido al hecho de ser la GPU más potente, sino también a las características del equipo en que fueron ejecutados los códigos. Sin embargo, al comparar las aceleraciones en mallas de menor tamaño no existen ventajas significativas con respecto a las otras dos GPU, lo cual indica que no necesariamente la GPU más potente brindará los mejores resultados. De hecho, es posible decir que la eficiencia de la GPU depende en gran medida de la cantidad de datos que esté procesando con respecto a la cantidad de elementos de proceso que posee.

En la Tabla **3-4** se puede observar que la inicialización del dispositivo utiliza una cantidad de tiempo muy similar en todos los casos pero que representa un porcentaje importante si la cantidad de datos a procesar no es representativa. También se puede determinar que un posible cuello de botella en el proceso se presenta en la transmisión de información entre la CPU y la GPU si se hace necesario el envío de todos los datos intermedios.

La Tabla **3-5** muestra que el proceso de indexación bidimensional genera una aceleración del proceso al compararlo con la versión de indexación unidimensional. Adicionalmente, se sigue observando que existe un punto óptimo de operación que depende de la cantidad de elementos de proceso de la GPU.

La Tabla **3-3** muestra cómo las aceleraciones presentes en las GPU aumentan de forma significativa al evitar sacar información innecesaria, a pesar de aumentar la complejidad de los códigos ejecutados, lo cual no supondría un problema muy serio sino fuera por los problemas que se pueden encontrar en arquitecturas que no soporten de forma adecuada la implementación de semáforos que aseguren la confiabilidad de los datos.

La figura **3-9** muestra la solución del problema de calor planteado. Como se esperaba hay una distribución equitativa de las condiciones de temperatura en la placa, lo que comprueba que el método númerico está generando una buena aproximación, así como la veracidad de los datos procesados por las GPU.

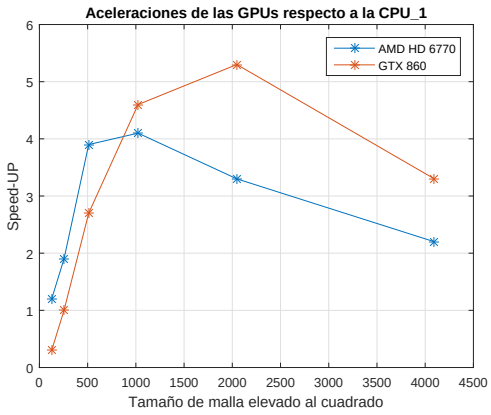


Figura 3-4: Aceleraciones de las diferentes GPU respecto a la CPU_ 1 sin utilizar barreras.

Tabla 3-1: Tiempo de ejecución y aceleración de la ecuación de Laplace para los distintos dispositivos sobre la CPU-1 sin utilizar barreras.

Tamaño de la malla	Tiempo CPU-1[seg]	Tiempo GPU AMD HD 6770[seg]	Speed-Up AMD HD 6770	Tiempo GPU GTX 860[seg]	Speed-Up GTX 860
127x127	0,0540	0,0450	**1,2**	0,2160	**0,3**
255x255	0,2900	0,1520	**1,9**	0,3020	**1,0**
511x511	2,2020	0,5700	**3,9**	0,8150	**2,7**
1023x1023	18,9610	4,5820	**4,1**	4,1220	**4,6**
2047x2047	159,8180	48,1170	**3,3**	30,3580	**5,3**
4095x4095	1196,9960	533,2160	**2,2**	364,2620	**3,3**

Tabla 3-2: Tiempo de ejecución y aceleración de la ecuación de Laplace para los distintos dispositivos sobre la CPU-2 sin utilizar barreras.

Tamaño de la malla	Tiempo CPU-2[seg]	Tiempo GPU GTX 770[seg]	Speed-Up GTX 770
127x127	0,0150	0,2150	**0,1**
255x255	0,1270	0,4560	**0,3**
511x511	1,3310	0,7660	**1,7**
1023x1023	11,8910	3,5980	**3,3**
2047x2047	84,5500	27,5470	**3,1**
4095x4095	659,9760	336,5220	**2,0**

Tabla 3-3: Tiempo de ejecución y aceleración de la ecuación de Laplace para los distintos dispositivos sobre la CPU-1 utilizando barreras.

Tamaño de la malla	Tiempo CPU-1[seg]	Tiempo GPU AMD HD 6770[seg]	Speed-Up AMD HD 6770	Tiempo GPU GTX 860[seg]	Speed-Up GTX 860
127x127	0,0540	0,1030	**0,5**	0,1750	**0,3**
255x255	0,2900	0,1330	**2,2**	0,2050	**1,4**
511x511	2,2020	0,4670	**4,7**	0,4270	**5,2**
1023x1023	18,9610	4,4290	**4,3**	2,2570	**8,4**
2047x2047	159,8180	55,5510	**2,9**	16,5440	**9,7**
4095x4095	1196,9960	503,3930	**2,4**	194,6320	**6,2**

Tabla 3-4: Tiempos significativos en milisegundos que toma cada parte del proceso (Break Down) en la ejecución de diferentes tamaños de malla para la tarjeta Geforce GTX 770 sin utilizar barreras.

Partes del proceso	127*127	255*255	511*511	1023*1023	2047*2047	4095*4095
1. Inicialización del dispositivo	176.95	168.59	200.83	193.81	195.84	258.296
2. Creación datos iniciales en CPU	0.06	0.14	0.28	1.11	4.61	17.75
3. Envío de la información a la GPU	0.18	0.34	0.69	2.14	9.78	37.62
4. Ejecución del Kernel	6.78	65.49	12.24	27.11	90.9	195.41
5. Envío información a CPU	19.43	132.44	363.18	2128.4	18238.3	266522.1
6. Reescritura información en GPU	11.92	88.25	186.15	1230.88	8947.26	69212.35
7. Liberación memoria GPU y CPU	0.17	0.18	0.32	0.70	2.43	8.67

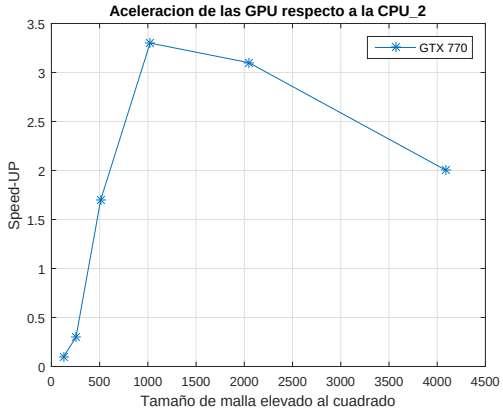


Figura 3-5: Aceleración de la GPU respecto a la CPU_ 2 sin utilizar barreras.

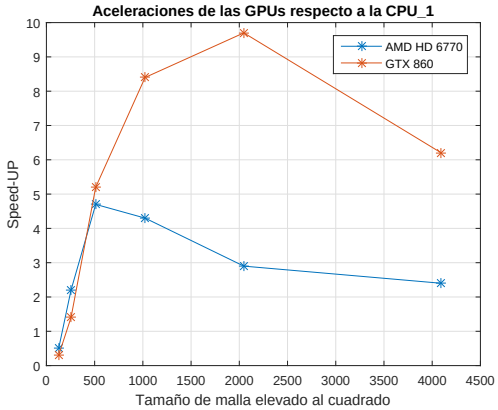


Figura 3-6: Aceleración de las diferentes GPU respecto a la CPU_ 1 utilizando barreras.

3.4. Conclusiones

Luego de realizar las diferentes pruebas se puede concluir que las GPU normalmente no tienen una aceleración considerable para mallas pequeñas ni para mallas de mayor tamaño, sino que cada una de las GPU tiene un punto de operación de mayor eficiencia en diferentes tamaños de mallas que depende de sus características. Esto demuestra que es imprescindible conocer las especificaciones del dispositivo acelerador que se va a utilizar para obtener su máximo desempeño.

Se puede determinar que la estrategia de indexación bidimensional utilizada en la GPU brinda aceleraciones de hasta $5{,}3$ veces con respecto al proceso secuencial (sin barreras), y aceleraciones superiores a las $9{,}7$ veces en el caso de obviar datos innecesarios (con barreras); ya que la dependencia de los datos vecinos en este proceso es similar a la forma en que se realiza el procesamiento de imágenes. Además, se debe tener en cuenta que esta dependencia de los vecinos también genera un mayor tiempo de ejecución por los múltiples accesos a memoria requeridos para leer los datos.

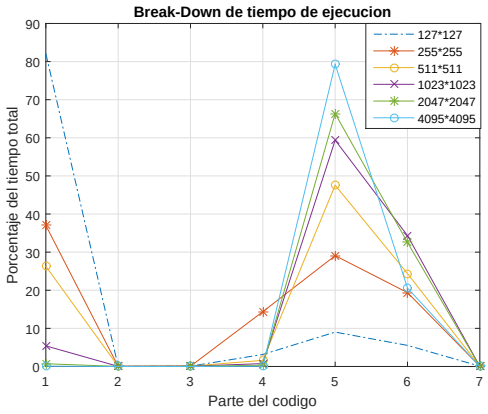


Figura 3-7: Porcentajes de Break-Down de tiempos para cada una de las partes del código para diferentes tamaños de malla.

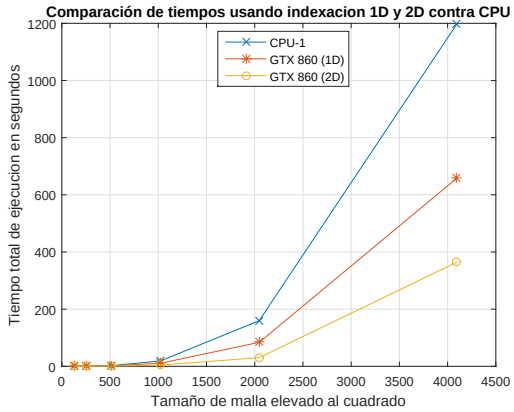


Figura 3-8: Comparación de tiempos totales usando indexacion 1D y 2D contra el proceso secuencial en CPU.

Estos resultados presentan un panorama alentador con respecto a la utilización de indexación tridimensional para problemas volumétricos teniendo en cuenta las condiciones propuestas acerca del máximo de datos a ejecutar de acuerdo a la arquitectura de la GPU.

Finalmente, se puede determinar que la GPU más potente brinda los resultados más rápidos en las mallas más grandes, como era de esperarse; pero también es importante notar que las GPU más pequeñas y económicas presentan mejores rendimientos y aceleraciones para problemas con menor cantidad de datos, lo que indica una fuerte dependencia entre la estructura de la GPU y las características del problema. También se puede observar cómo la implementación de librerías adicionales(barreras) propias de los lenguajes permiten evitar datos innecesarios en la solución del problema generando una reducción en los tiempos de simulación.

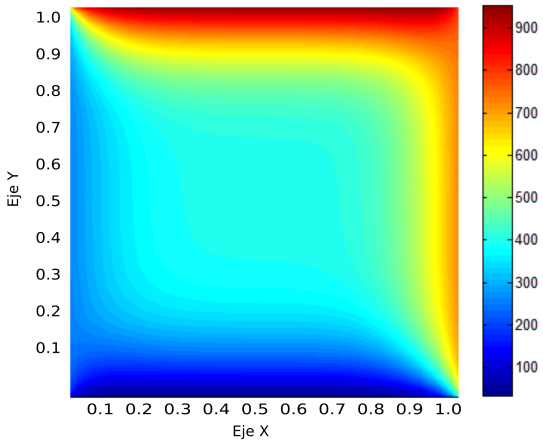


Figura 3-9: Resultado final de la temperatura en la placa plana, empleando la ecuación de Laplace.

Tabla 3-5: Comparación del proceso de indexación con tiempo de ejecución en segundos y la aceleración de la ecuación de Laplace para la GPU GTX 860.

Tamaño	GPU index 1D [seg]	GPU index 2D [seg]	Speed-Up
127x127	0.219	0.216	**1.01**
255x255	0.632	0.302	**2.09**
511x511	1.663	0.815	**2.04**
1023x1023	10.703	4.122	**2.59**
2047x2047	83.972	30.358	**2.76**
4095x4095	657.980	364.262	**1.80**

Tabla 3-6: Especificaciones técnicas de las GPU usadas en el experimento

Especificación	Kepler GTX 770	Maxwell GTX 860	VLIW 5D AMD HD 6770
Memoria global (MBytes)	2048	2048	1024
Unidades de cómputo	8	5	10
Cantidad de núcleos	1536	640	800
Frecuencia máxima del reloj de la GPU (MHz)	1163	1020	850
Frecuencia del reloj de memoria (MHz)	3505	2505	1200
Ancho de Bus (bit)	256	128	128
Tamaño de Cache L2 (bytes)	524288	2097152	262144
Número de registros totales por bloque	65536	65536	65536
Interfaz del Bus (PCI-E)	3.0x16	3.0x16	2.0x16
Tamaño del procesador (nm)	28	28	40
Ancho de banda (GB/seg)	224	80	76.8
Operaciones en punto flotante (GFLOPS)	3.213	1.306	1.36

4 Indexación tri-dimensional empleando GPU para acelerar la aproximación numérica de soluciones de la ecuación de Laplace

En el presente capítulo, el objetivo es analizar el efecto del uso de indexación tri-dimensional para resolver la ecuación de Laplace , con el fin de conseguir la solución estacionaria para el problema de la transferencia de calor en un volumen. También, se presenta el estudio del efecto de la utilización de la precisión doble y simple en la representación de los datos.

Cuando se declara una variable de tipo float se conoce como una variable de precisión simple que utiliza 32 bits por dato. Mientras que al declarar una variable tipo double se le conoce como una variable de precisión doble y se asignan 64 bits. Usar cada tipo de variable depende generalmente de la precisión requerida por la aplicación, pero al utilizar un tipo de variable sea double o float se debe tener en cuenta que el consumo de memoria en el dispositivo es diferente, lo cual hace que su elección sea un punto importante al momento de implementar una solución algorítmica.

Pero los argumentos anteriores no son los únicos a ser considerados, ya que también se debe estudiar el tipo de dispositivo en el que se va ejecutar el código. Si bien al momento de declarar variables de tipo float o double no hay ningún problema en la ejecución de un algoritmo en cualquier hardware, puede que al momento de comparar tiempos y rendimientos se obtengan mejores resultados en uno que en otro, esto se explica porque la arquitectura del hardware puede estar optimizada para un tipo de variable específico [87]. En este capítulo se desea comparar diferentes tipos de arquitecturas de GPU, con el fin de encontrar cómo afecta directamente el uso de ambas variables en el tiempo de ejecución de los códigos.

Mientras una de las tarjetas utilizadas fue una GPU Nvidia GTX 770 la cual está diseñada para trabajar con precisión simple debido a que es una tarjeta construida para video juegos. La otra tarjeta es una GPU Tesla K20M que está diseñada para propósito general y posee una arquitectura optimizada para trabajar tanto en precisión simple como en doble, lo anterior no significa que la GPU Nvidia GTX 770 no tenga la capacidad de procesar variable de doble precisión pero necesitará de dos ciclos de reloj para poder hacerlo mientras que la otra GPU solo necesitará uno, lo cual

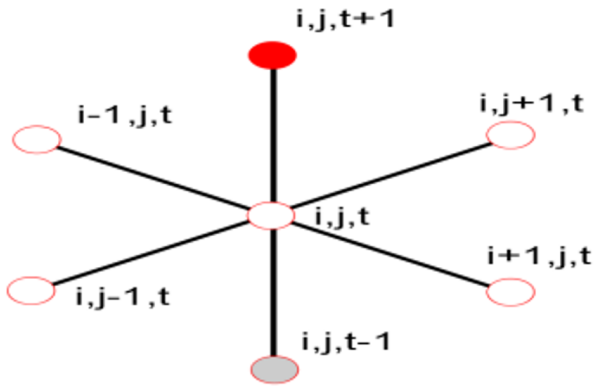


Figura 4-1: Discretización del espacio del problema y dependencia de los datos cercanos en 3 dimensiones.

afectará directamente los tiempos de simulación en ambas tarjetas.

En la sección 4.1 se presenta el diseño del experimento. En la sección 4.2 se muestran los resultados. En la sección 4.3 se presenta la discusión del los resultados. Finalmente en la sección 4.4 se presentan algunas conclusiones y observaciones.

4.0.1. Ecuación de Laplace en tres dimensiones.

La ecuación de Laplace tri-dimensional se utiliza para determinar el valor estacionario de una variable en problemas volumétricos [90]. Ejemplos de problemas en los que la ecuación de Laplace se puede utilizar son los campos magnéticos o gravitatorios [45]. En este caso de estudio , la ecuación de Laplace se utiliza para modelar los fenómenos de transferencia de calor en un bloque metálico. Los problemas que se describen mediante ecuaciones diferenciales parciales tienen raramente una solución analítica, por esta razón, se han desarrollado métodos numéricos que permiten aproximar una solución [83, 67] . Sin embargo , estos requieren una gran cantidad de datos y cálculos para llegar a una solución satisfactoria. Debido a esto , se necesitan herramientas computacionales eficientes . Específicamente, el método de diferencias finitas utiliza una malla discreta de puntos en el que cada punto depende de sus vecinos como se muestra en la figura **4-1** [8, 21]. Este método da una buena aproximación a la solución de la ecuación de Laplace y, como se puede observar, hay una gran similitud con la arquitectura de la GPU [23]. El problema consiste en el cálculo de la temperatura de estado estacionario del bloque suponiendo una temperatura inicial en el bloque igual a 0 con condiciones de contorno que son fijas durante toda la simulación. El algoritmo itera hasta que cada uno de los puntos en el bloque no presenten ningún cambio significativo, obteniendo así el estado de equilibrio y por tal motivo la temperatura final.

4.0.2. Indexación tri-dimensional.

La estrategia de indexación bi-dimensional en una GPU es muy útil para obtener mejores aceleraciones (*Speed*-Up) en la resolución de problemas que implican dependencias bi-dimensionales de los datos de la vecindad [62] como se estudió en el capítulo anterior. La geometría del problema planteada en este capítulo se ajusta a la capacidad de indexación tri-dimensional de una GPU. El objetivo es colocar a prueba si es posible lograr la misma aceleración o mejor usando una estrategia en tres dimensiones. La indexación consiste en asignar índices a cada uno de los valores que son enviados a la GPU con el fin de que el dispositivo los coloque en el Kernel cerca de los vecinos que serán necesarios para el cálculo [42, 93], de esta manera es posible conseguir una interacción más rápida entre los datos vecinos.

La desventaja de la estrategia convencional para resolver un problema volumétrico de este tipo es el uso de tres bucles anidados que afectan al tiempo de cálculo. El uso de indexación en GPU realiza las operaciones en paralelo y los bucles anidados no son necesarios. Por otra parte, si los datos necesarios en un barrido están cerca, el tiempo de espera para los accesos a memoria se reduce ya que los Kernels pueden acceder a la información que reside en sus vecinos más rápidamente que cualquier otra fuente de datos. Estas ventajas permiten a la GPU acelerar el proceso.

Dado el carácter concurrente del procesamiento de datos, es importante asegurarse de que cada Kernel está accediendo a los datos actualizados. Esto se logra mediante el uso de barreras, los cuales impiden que el Kernel utilice los datos de la siguiente iteración antes de que los demás Kernels hayan finalizado el procesamiento en la actual iteración. Mediante el uso de esta estrategia, no sólo la integridad de los datos está garantizada, además algunas transferencias de datos intermedias entre la CPU y la GPU serán eliminadas, con esto se logrará disminuir los tiempos de ejecución.

4.0.3. Precisión doble Vs precisión simple.

El tipo de variable utilizada para cualquier aplicación define el espacio de memoria necesario para su almacenamiento y la manera en la que el hardware tiene que procesar la misma, por lo tanto es necesario elegirla cuidadosamente. Entre las posibilidades, hay dos de especial importancia en esta aplicación donde se necesita la representación de punto flotante, es decir, de doble precisión y de precisión simple. Estas representaciones de punto flotante utilizan 64 y 32 bits respectivamente. La forma en la cual el hardware procesa estos tipos de variables depende de su construcción [73, 68]. Algunos dispositivos implementan físicamente unidades lógicas y aritméticas de operaciones en 32 bits y otros implementan operaciones en 64 bits. A pesar de que ambas arquitecturas son capaces de realizar operaciones de datos de 64 bits, el tiempo de ejecución se verá afectado, debido a que los dispositivos que no tienen implementado por hardware la arquitectura de 64 bits toman este tipo de variables haciendo una emulación por software y generando así mayores tiempos de simulación. En la sección de resultados se presenta un estudio numérico para el efecto en la aceleración de algoritmos utilizando precisión simple y doble para la representación de punto

flotante en diferentes arquitecturas de GPU [73, 68].

4.1. Diseño del experimento.

Para comparar el rendimiento de diferentes arquitecturas en la aplicación de estrategias de aceleración, se decidió resolver el problema de transferencia de calor en un cubo metálico. El problema consiste en el cálculo de la temperatura de estado estacionario de un bloque con las siguientes condiciones: La temperatura inicial es de 0 grados centígrados y cada cara del cubo tiene una temperatura fija estable. El tiempo que cada dispositivo necesita para obtener la respuesta de estado estacionario del sistema está directamente relacionada con el número de puntos que se procesan, por lo que este valor se incrementa de forma proporcional hasta el valor máximo de tamaño de malla que soporta cada dispositivo.

Se utilizaron dos GPU: Nvidia GTX 860M y Nvidia Tesla K20M en la Tabla **4-1** se encuentran las especificaciones de cada GPU. La GPU GTX está diseñado para vídeo juegos por lo que funciona con datos de precisión simple. La GPU Tesla por otro lado, es una GPU de propósito general y por lo tanto tiene una arquitectura optimizada para trabajar con datos de precisión simple o doble [73, 68]. Los algoritmos se pueden ejecutar en ambas GPU, pero en el caso de la GPU GTX cuando el dato es de doble precisión tiene que usar dos ciclos de reloj para procesar un valor aumentando los tiempos de simulación en este dispositivo.

Tabla 4-1: Especificaciones técnicas de las GPU usadas en el experimento.

Especificación	Nvidia GTX 860M	Nvidia Tesla k20m
Memoria global (MBytes)	2048	5120
Unidades de cómputo	6	13
Cantidad de núcleos	640	2496
Máxima cantidad de bloques	65535	65535
Máxima cantidad de hilos por bloque	1024	1024
Frecuencia del reloj de memoria (MHz)	5012	5200
Ancho de Bus (bit)	128	320
Ancho de banda (GB/seg)	80,2	208
Cantidad de transistores (millones)	1870	7080
Interfaz del Bus (PCI-E)	3,0x16	2,0x16
Tamaño del procesador (nm)	28	28
Operaciones en punto flotante (GFLOPS)	1306,0	3524,0
Precio de lanzamiento (USD)	115	3199
Consumo (Watts)	75	225
Fecha de lanzamiento	2014	2013
Arquitectura	Maxwell	Kepler

Cuatro experimentos se llevaron a cabo para determinar los dispositivos con mejores aceleraciones y las condiciones necesarias para alcanzarlos.

El primer experimento compara el rendimiento de los dos GPU utilizando indexación en tres dimensiones en comparación con el proceso secuencial en CPU. Los tamaños de malla se definen como: $127 * 127 * 31$, $127 * 127 * 63$, $255 * 255 * 31$, $255 * 255 * 63$, $511 * 511 * 31$, $511 * 511 * 63$, se continúa hasta el máximo que alcanza cada dispositivo.

El segundo experimento es un Break-Down de los tiempos de ejecución de los diferentes dispositivos. Este Break-Down ayuda a determinar las etapas que generan los principales cuellos de botella.

El tercer experimento compara el rendimiento de los dos GPU utilizando indexación en tres dimensiones en comparación con el caso de no indexación; el objetivo es verificar si la estrategia tridimensional genera una mejora en el tiempo de procesamiento y definir los tamaños de malla con mejores resultados.

Por último, se analizó el rendimiento en GPU al utilizar precisión doble y simple en la representación de datos en punto flotante. Es importante analizar la diferencia en el tiempo de ejecución entre una GPU optimizada para datos de doble precisión (Tesla) versus una GPU que tiene que emular este proceso (GTX).

4.2. Resultados.

Para este capítulo se tomará la convención de la Tabla 4-2 para mostrar los tamaños de las mallas en las distintas figuras.

La Tabla **4-3** muestra el tiempo de ejecución para diferentes tamaños de malla utilizando indexa-

Tabla 4-2: Convención de tamaño de malla para las distintas figuras presentadas en este capítulo.

Tamaño de malla	Convención Tamaño de malla para las figuras
127*127*31	1
127*127*63	2
255*255*31	3
255*255*63	4
511*511*31	5
511*511*63	6
1023*1023*31	7
1023*1023*63	8
2047*2047*31	9
2047*2047*63	10
4095*4095*31	11

ción tri-dimensional para resolver la ecuación de Laplace en la GPU y estos tiempos se comparán con los tiempos de ejecución de una CPU en serie. Estos tiempos se muestran gráficamente en mili segundos en la figura **4-2**.

Los resultados gráficos se muestran para un tamaño máximo de malla de $1023 * 1023 * 31$ porque los valores más grandes no se pueden comparar en todas las arquitecturas, esta situación es causada por el hecho de que la GPU Tesla está diseñado para computación científica, mientras que la GPU NVIDIA está diseñada para juegos de video, por lo que no puede ejecutar tamaños tan grandes de malla.

Tabla 4-3: Tiempo de ejecución en segundos para varias mallas utilizando indexación tri-dimensional para la ecuación de Laplace.

Tamaño de malla	CPU 1 [seg]	GPU Tesla K20 [seg]	CPU 2 [seg]	GPU GTX860M [seg]
127*127*31	15,6631	2,425	12,4641	2,363
127*127*63	32,274	4,5481	25,8546	4,731
255*255*31	63,2626	8,4308	50,0833	8,465
255*255*63	138,4098	17,2079	104,1868	17,643
511*511*31	260,2138	33,1526	202,9859	38,764
511*511*63	540,1435	69,3387	419,4467	80,504
1023*1023*31	1107,3576	129,7876	861,2354	146,381
1023*1023*63	2356,7176	275,5235	1847,1254	No soportado
2047*2047*31	4715,5565	601,0376	3556,9833	No soportado
2047*2047*63	9269,591	1269,4867	7286,4585	No soportado
4095*4095*31	17826,576	4308,8665	13922,897	No soportado

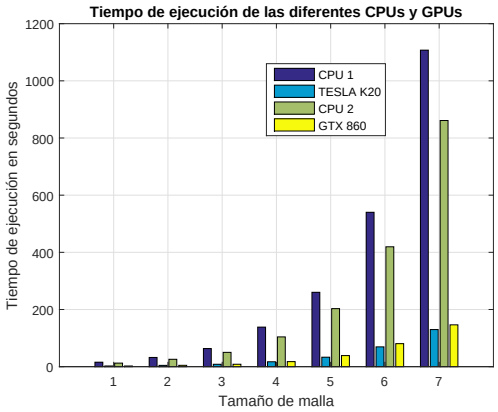


Figura 4-2: Tiempo de ejecución en segundos para varias mallas utilizando indexación tri-dimensional para la ecuación de Laplace.

La Tabla **4-4** muestra las aceleraciones de las dos GPU tomando el tiempo de la CPU como base para la comparación. El tiempo para la GPU Tesla se comparó con la CPU del centro de cómputo de bioinformática (BIOS-Manizales) y el de la GPU GTX se comparó con una CPU convencional. La aceleración se calcula dividiendo el tiempo de ejecución de la GPU por el tiempo de ejecución de la CPU base. La Tabla **4-4** se construye utilizando los datos que se muestran en la Tabla **4-3**. La figura **4-3** muestra los datos contenidos en la Tabla **4-4**.

La Tabla **4-5** muestra los Break-Down de tiempo de ejecución en milisegundos como también los porcentajes de tiempo total usando la GPU Tesla con la indexación tri-dimensional para un tamaño de malla de $127 * 127 * 63$ después de 2000 iteraciones. La Tabla **4-6** muestra el tiempo de ejecución para diferentes tamaños de malla utilizando indexación tri-dimensional en comparación con el caso de no-indexación. También muestra la aceleración entre ambas formas de indexación.

En la figura **4-4** se observaran gráficamente los porcentajes de Break-Down obtenidos en la Tabla **4-5** para una malla de $127 * 127 * 63$ sobre la GPU Tesla para la ecuación de Laplace usando indexación 3D.

Tabla 4-4: Aceleraciones de varias mallas utilizando la indexación tri-dimensional para la ecuación de Laplace.

Tamaño de malla	Aceleración GPU Tesla K20	Aceleración GPU GTX 860M
127*127*31	6,45	5,27
127*127*63	7,09	5,46
255*255*31	7,5	5,91
255*255*63	8,04	5,9
511*511*31	7,84	5,23
511*511*63	7,78	5,21
1023*1023*31	8,53	5,88
1023*1023*63	8,55	No soportado
2047*2047*31	7,84	No soportado
2047*2047*63	7,3	No soportado
4095*4095*31	4,13	No soportado

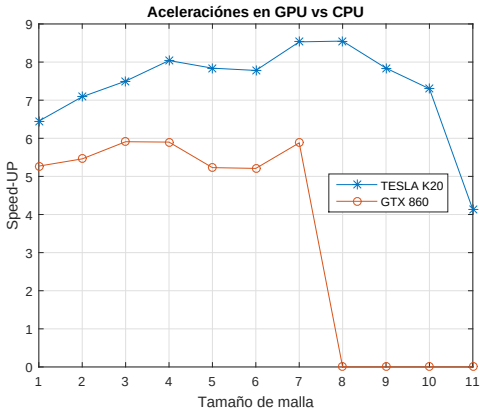


Figura 4-3: Aceleraciones de GPU para diferentes tamaños de malla utilizando indexación tri-dimensional para la ecuación de Laplace.

En la figura **4-5** se observan gráficamente las aceleraciones de indexación tri-dimensional vs no-indexación obtenidas en la Tabla **4-6** para la ecuación de Laplace usando la GPU Tesla para diferentes mallas.

El tiempo de ejecución en segundos para diferentes tamaños de malla utilizando la indexación tri-dimensional de simple y doble precisión en la CPU y la GPU, se pueden observar en la Tabla **4-7**. Las figuras **4-6** y **4-7** muestran el tiempo de ejecución en segundos para diferentes tamaños de malla mediante la indexación tri-dimensional para la ecuación de Laplace implementada en simple y doble precision respectivamente.

En la figura **4-8** y Tabla **4-8** se muestran las aceleraciones obtenidas para diferentes tamaños de malla utilizando simple y doble precisión y la indexación tri-dimensional. La figura **4-8** y la Tabla **4-8** se construyeron a partir de los datos que se muestran en la Tabla **4-7**.

Tabla 4-5: Break-Down en milisegundos para la malla $127 * 127 * 63$ usando indexación tri-dimensional en la GPU Tesla.

Parte del código	127*127*63
Inicialización del dispositivo	317,213
Asiganción de memoria en CPU	0,037
Generación de malla en CPU	3,076
Asiganción de memoria en GPU	4,711
Envio de información a GPU	0,006
Ejecución del Kernel	2009,283
Copia de resultados finales a CPU	2093,322
Liberación de memoria en GPU	98,754
Liberación de memoria en CPU	21,736
TOTAL	4548,143
%Inicialización del dispositivo	6,974
%Asiganción de memoria en CPU	0,001
%Generación de malla en CPU	0,067
%Asiganción de memoria en GPU	0,103
%Envio de información a GPU	0,001
%Ejecución del Kernel	44,178
%Copia de resultados finales a CPU	46,025
%Liberación de memoria en GPU	2,171
%Liberación de memoria en CPU	0,477
%TOTAL	100

Tabla 4-6: Tiempos de ejecución en segundos de diferentes mallas usando no-indexación e indexación tri-dimensional en la GPU Tesla para la ecuación de Laplace.

Tamaño de malla	No-indexación [seg]	Indexación tri-dimensional [seg]	Speed–Up GPU Tesla K20
127*127*31	9,3847	2,4251	**3,86**
127*127*63	15,9298	4,5481	**3,5**
255*255*31	29,2503	8,4308	**3,46**
255*255*63	62,7303	17,2079	**3,64**
511*511*31	154,7457	33,1526	**4,66**
511*511*63	316,0296	69,3387	**4,55**
1023*1023*31	443,5532	129,7876	**3,41**
1023*1023*63	890,4023	275,5234	**3,23**
2047*2047*31	1791,1289	601,0376	**2,98**
2047*2047*63	No soportado	1269,4868	**No soportado**
4095*4095*31	No soportado	4308,8665	**No soportado**

Tabla 4-7: Tiempos de ejecución en segundos de diferentes mallas usando indexación tri-dimensional para la ecuación de Laplace en simple y doble precisión

Tamaño de malla	Precisión simple CPU BIOS	Precisión simple Tesla K20	Precisión simple CPU convencional	Precisión simple GTX 860M
127*127*31	15,6631	2,4251	12,4641	2,3630
127*127*63	32,2740	4,5481	25,8546	4,7310
255*255*31	63,2625	8,4308	50,0833	8,4650
255*255*63	138,4098	17,2079	104,1868	17,6430
511*511*31	260,2138	33,1526	202,9860	38,7640
511*511*63	540,1435	69,3387	419,4467	80,5040
1023*1023*31	1107,3576	129,7876	861,2354	146,3810
1023*1023*63	2356,7175	275,5235	1847,1254	No soportado
2047*2047*31	4715,5565	601,0376	3556,9833	No soportado
2047*2047*63	9269,5910	1269,4868	7286,4585	No soportado
4095*4095*31	17826,5760	4308,8665	13922,8970	No soportado

Tamaño de malla	Precisión doble CPU BIOS	Precisión doble Tesla K20	Precisión doble CPU convencional	Precisión doble GTX 860 M
127*127*31	18,0245	2,3578	14,1582	2,1590
127*127*63	37,1477	4,5313	29,6366	4,4540
255*255*31	75,2808	8,5564	57,3544	9,4000
255*255*63	157,2047	17,4514	119,3931	19,0850
511*511*31	316,1201	33,5062	244,3453	43,4460
511*511*63	656,5373	70,0846	509,7639	90,9310
1023*1023*31	1284,8265	135,6386	993,7081	No soportado
1023*1023*63	2684,9768	284,3229	2053,1366	No soportado
2047*2047*31	5172,9985	No soportado	3944,3348	No soportado
2047*2047*63	10713,2820	No soportado	8134,6510	No soportado
4095*4095*31	21257,5760	No soportado	15837,9950	No soportado

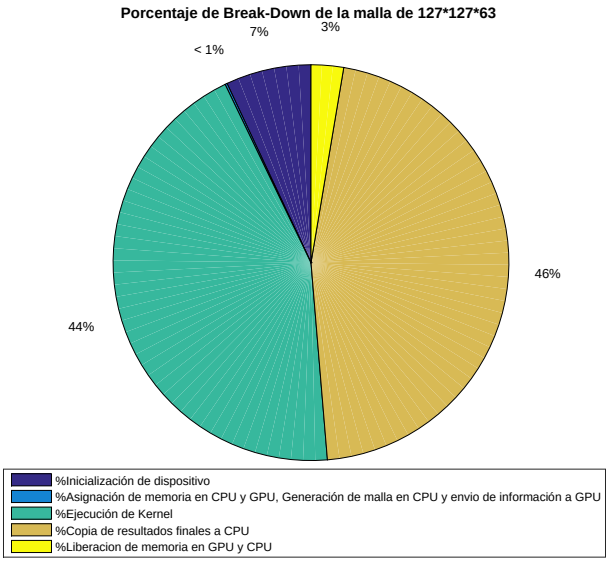


Figura 4-4: Porcentaje de Break-Down para la malla $127 * 127 * 63$ usando indexación tri-dimensional en la GPU Tesla.

Tabla 4-8: Aceleraciones de diferentes mallas usando indexación tri-dimensional en simple vs doble precisión para la ecuación de Laplace.

Speed-Up precisión simple Tesla K20	Speed-Up precisión simple GTX 860M	Speed-Up precisión doble Tesla K20	Speed-Up precisión doble GTX 860M
6,46	5,27	7,64	6,56
7,10	5,46	8,20	6,65
7,50	5,92	8,80	6,10
8,04	5,91	9,01	6,26
7,85	5,24	9,43	5,62
7,79	5,21	9,37	5,61
8,53	5,88	9,47	No soportado
8,55	No soportado	9,44	No soportado
7,85	No soportado	No soportado	No soportado
7,30	No soportado	No soportado	No soportado
4,14	No soportado	No soportado	No soportado

4.3. Discusión

En los datos se observa una dependencia no lineal de la aceleración en relación con el tamaño de las mallas. También muestra que los mejores valores de aceleración se obtienen para ciertos tamaños de malla en cada una de las arquitecturas probadas, este valor máximo depende de la propia arquitectura y las capacidades de memoria. Se obtuvieron resultados similares para la indexación tri-dimensional, así como mejores tiempos de ejecución en comparación con la no-indexación para el problema de Laplace en volúmenes.

Para todos los casos investigados, una parte significativa del tiempo total de cálculo se invierte en la inicialización de la GPU, es decir la escritura y lectura de datos, y podría representar un cuello de botella. Es posible formular la hipótesis de que un incremento en el número de iteraciones en el

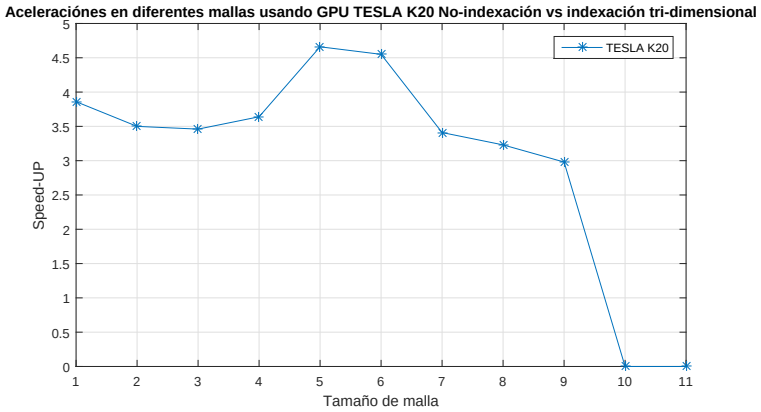


Figura 4-5: Aceleraciones de diferentes mallas usando no-indexación vs indexación tri-dimensional en la GPU Tesla K20 para la ecuación de Laplace.

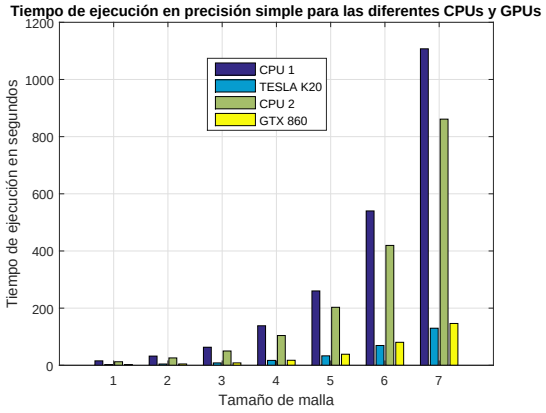


Figura 4-6: Tiempos de ejecución de diferentes mallas usando indexación tri-dimensional en simple precisión para la ecuación de Laplace.

tiempo puede resultar en la reducción de este tiempo. El uso de barreras es necesario para garantizar la calidad de la salida, aumentando el tiempo de ejecución para el Kernel, pero disminuyendo el tiempo de transferencia de datos hacia y desde la GPU, otro posible cuello de botella, pero su uso compensa en gran medida el tiempo total de cómputo.

Al realizar las pruebas del proceso en doble precisión se muestra cómo, evidentemente, el algoritmo toma un grado de complejidad mayor, que se ve representado en el incremento del tiempo en especial cuando se ejecuta de forma secuencial. En el momento de comparar esta implementación secuencial con la implementación en GPU se encontró que si el dispositivo no está diseñado físicamente para este tipo de precisiones se genera un aumento en el tiempo de procesamiento como el mostrado en la GPU GTX. Pero al compararlo con un dispositivo que sí posee esta configuración, GPU Tesla, se encuentra que no se ve un cambio notorio en el tiempo más allá del relacionado con el aumento de la cantidad de datos, lo que genera que este tipo de dispositivos con configu-

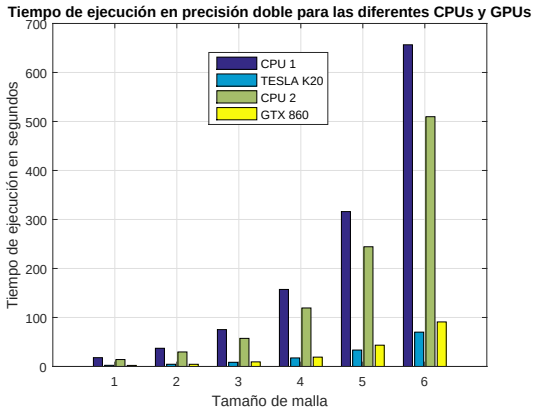


Figura 4-7: Tiempos de ejecución de diferentes mallas usando indexación tri-dimensional en doble precisión para la ecuación de Laplace.

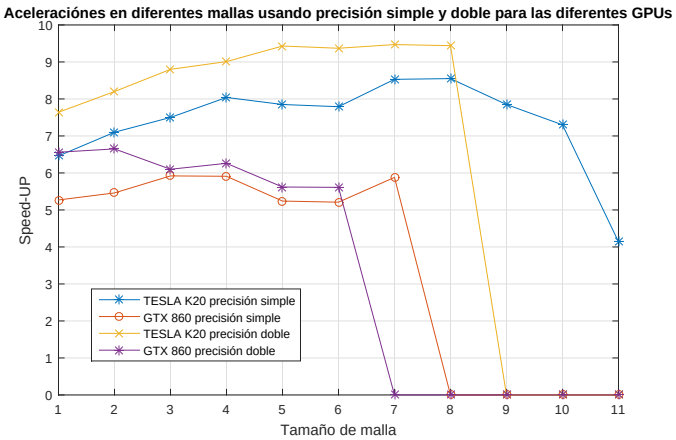


Figura 4-8: Aceleraciones de diferentes mallas usando indexación tri-dimensional en simple vs doble precisión para la ecuación de Laplace.

ración para precisión doble obtengan mejores rendimientos en la aceleración con respecto a sus contrincantes.

4.4. Conclusiones

Después de realizar los experimentos numéricos propuestos, se evidenció que existe un rendimiento máximo que depende de la estructura del problema, el tamaño de las mallas y la estructura computacional, demostrando que es esencial conocer las especificaciones de la GPU, para obtener un rendimiento máximo. La indexación tri-dimensional utilizando GPU en el caso de la ecuación de Laplace para los problemas volumétricos, alcanza aceleraciones de hasta 9,7 veces en comparación con el proceso secuencial en la CPU, y aceleraciones de hasta 4,7 veces en comparación

a no-indexación en la misma GPU. En cuanto a la utilización de la memoria, la indexación tridimensional permite el uso de toda la capacidad de la GPU.

También se puede observar cómo la GPU que debería presentar los mejores desempeños, debido a sus altas prestaciones, realmente es la más veloz y de mayor capacidad, pero también se evidencia que dispositivos de menor rendimiento generan aceleraciones de magnitud similar siempre que las condiciones del problema se ajusten a sus características, como en el caso de mallas de menor tamaño.

Sólo la GPU Tesla K20m fue capaz de mantener el rendimiento de aceleración cuando los experimentos se llevaron a cabo usando doble precisión como representación de punto flotante. Todas las otras estructuras de cálculo redujeron significativamente el rendimiento. En todos los casos, el uso de datos de 64 bits redujo el tamaño máximo de malla que puede ser procesada.

Por último, el tiempo de cálculo y el uso de memoria se determinaron experimentalmente para problemas volumétricos utilizando la ecuación de Laplace con GPU. La relación entre el rendimiento y el tamaño óptimo de las mallas se encontró para simple y doble precisión para la representación de datos en punto flotante.

5 Estrategias paralelas en CPU y GPU aplicadas al análisis estadístico de polimorfismo de un solo nucleótido

Los estudios de asociación de genoma completo (GWAS) se han convertido en una técnica estándar utilizada por los científicos para investigar la correlación estadística entre los rasgos de ADN de varios individuos. En particular, los científicos usan GWAS en la investigación de la enfermedad de Alzheimer [9], enfermedad de la arteria coronaria [18], diferentes tipos de cáncer, como el de próstata [84], carcinoma de tiroides [84], carcinoma de pulmón [49], entre otros [94]. Esta técnica también se utiliza en los estudios de asociación de los marcadores de obesidad relacionados con la depresión [82], cambios genéticos debido a los medicamentos [39] y estudios de epidemiología genética en general.

En enfermedades determinadas por GWAS por lo general hay dos grupos principales, a saber, los casos y controles [50]. Los casos son individuos en el estudio relacionados con la enfermedad y los controles son individuos sanos. El estudio tiene por objeto encontrar diferencias dentro de los polimorfismos de un solo nucleótido (SNP) para el conjunto de la población (casos y controles), tambien discriminado por sexo. Como resultado, se determinan cambios genéticos raros estadísticamente significativos y luego se investiga si estos cambios están relacionados con una enfermedad rara [40, 72]. Una enfermedad rara o compleja es una enfermedad que no puede ser estudiada a través de la herencia, por tal razón dicha enfermedad puede estar relacionada con el estilo de vida, ambiente de trabajo u otros factores.

Dentro del GWAS, dos procesos relevantes son la clasificación y el análisis de SNPs para varias personas. Los estudios de casos y los controles incluyen operaciones estadísticas como: conteo de genotipo, frecuencias alélicas y genotípicas, relaciones odds ratio entre los casos y controles, equilibrio Hardy-Weinberg en la población estudiada, la determinación del valor de significación de las relaciones a través del p-valor y múltiple generación de modelos de herencia como dominante, co-dominante, recesivo, sobre-dominante, log-aditivo entre genotipos para determinar la posible asociación del gen con la enfermedad y el análisis de SNP múltiples: estimación de las frecuencias de haplotipos, el análisis de asociación de haplotipos con la respuesta, incluyendo el análisis de las interacciones [85, 91], esta última no se incluye en este capítulo.

Los archivos Pedigree generados contienen millones de SNPs a analizar que implican alta complejidad computacional [46]. La primera consecuencia es la necesidad de tiempo de procesamiento, si se realiza en máquinas secuenciales. Algunos investigadores prefieren estudiar diferentes enfoques algorítmicos (la minería de datos, clustering, PCA) [54, 52, 99] en busca de mejores implementaciones computacionales para reducir el tiempo de procesamiento. Las técnicas de procesamiento en paralelo pueden ayudar a mejorar los tiempos de respuesta de los algoritmos secuenciales, por esta razón, se implementan versiones sobre arquitecturas de GPU [100, 63, 56] y CPU[59].

Existen varios enfoques para reducir el tiempo en los procesos intensivos de datos, algunos de ellos son: Interface de Paso de Mensajes (MPI) [34], procesamiento Grid [25], Procesamiento Cloud, procesamiento multi-núcleo en GPU[48] y CPU [22, 75, 32]. Las últimas técnicas de procesamiento secuencial se centran en la aplicación óptima de los modelos matemáticos [74] y la adaptación del algoritmo para arquitecturas de computación de núcleos múltiples. El uso de librerías MPI han mejorado el rendimiento en varios procesadores [19], sin embargo, el movimiento de datos a través de la red implica tiempo adicional que se convierte en otro obstáculo, por no hablar de los costos de tener acceso a este tipo de arquitecturas. Por estas razones, en este capítulo sólo se trabajó una estrategia paralela de núcleos múltiples en máquinas con GPU o CPU.

Es posible lograr la aceleración de estos algoritmos para arquitecturas multi-núcleo y aplicarlas bajo técnicas de computación en paralelo, mediante la distribución de datos y funciones para el análisis. Las estructuras utilizada para representar los datos en cada arquitectura juegan un papel importante en el rendimiento, especialmente cuando se ejecuta en GPU. [36], [32]

En este capítulo se muestra el potencial de las arquitecturas multi-núcleo para acelerar las aplicaciones que implican grandes volúmenes de datos. Los tiempos de respuesta obtenidos al utilizar algoritmos de procesamiento paralelo SNP han demostrado que las funciones genotípicas mejoran en un $96{,}28\,\%$ con respecto a su implementación secuencial ejecutada en un solo núcleo de la CPU. La distribución de los métodos y datos sobre diferentes núcleos puede alcanzar velocidades de hasta $26{,}88$X en la CPU y de hasta $8{,}5$X en GPU. Estos resultados son prometedores ya que permite a los científicos obtener procesos más rápidos para futuros análisis.

En la sección 5.1 se presenta un análisis estadístico del estudio del polimorfismo genético en caso-control . En la sección 5.2 se muestra la implementación de los algoritmos para el análisis estadístico de SNP. En la sección 5.3 se muestran los resultados de los tiempos de ejecución en los diferentes algoritmos y las diferentes arquitecturas tratadas en este capítulo. En la sección 5.4 se presenta la discusión del capítulo. Finalmente en la sección 5.5 se presentan las conclusiones.

5.1. Análisis estadístico del estudio del polimorfismo genético en caso-control

El análisis estadístico de los polimorfismos genéticos describen el comportamiento de los genes en una población. Estos estudios permiten asociar enfermedades basadas en la genética con cambios representados por individuos con la enfermedad (casos) e individuos sanos (controles) [95]. Las definiciones en esta sección son compatibles con [33]. Las funciones matemáticas están representadas en las Tablas **5-1**, **5-3**, **5-4**. Las funciones para la caracterización fueron obtenidas por los genotipos básicos de conteo: las frecuencias alélicas y frecuencias genotípicas. Para la estadística básica: el equilibrio de Hardy-Weinberg, intervalos de confianza, valores de significación y los modelos de herencia.

5.1.1. Frecuencias genotípicas

Es el número de genotipos de una población. Se calcula utilizando frecuencias que cuantifican el proceso de la evolución. Los niveles genéticos de la población se pueden caracterizar por las frecuencias de alelos (véase la Tabla **5-3**) y frecuencias genotípicas (véase la Tabla **5-2**). La distribución de los genotipos se define como el total de los genotipos de la población (véase la Tabla **5-1**). Las frecuencias genotípicas se definen como la suma de cada genotipo dividido por la población total. Por ejemplo, si $CC = 5$, $GG = 2$, y $CG = 7$ un genotipo individual, entonces la frecuencia del genotipo CC es $fcc = 5/14$.

Tabla 5-1: Distribución de genotipos en estudios de Caso-Control

Genotipo	CC	CG	GG	Total
Casos	r_0	r_1	r_2	r
Controles	s_0	s_1	s_2	s
Total	n_0	n_1	n_2	n

Tabla 5-2: Frecuencias genotípicas en estudios de Caso-Control

Genotipo	CC	CG	GG	Total
Casos	$f_{r_0} = r_0/r$	$f_{r_1} = r_1/r$	$f_{r_2} = r_2/r$	f_r
Controles	$fs_0 = s_0/s$	$fs_1 = s_1/s$	$fs_2 = s_2/s$	fs
Total	$fn_0 = n_0/n$	$fn_1 = n_1/n$	$fn_2 = n_2/n$	fn

5.1.2. Frecuencias alélicas

Sea CG un par de alelos, la frecuencia del alelo C se define como la probabilidad de que el genotipo lleva el alelo C, las frecuencias del alelo es una variable discreta que toma valores entre 0 y 1. La fórmula matemática para calcular las frecuencias de los alelos de los casos, controles y la población total se define en la Tabla **5-3**.

Tabla 5-3: Frecuencias alélicas en estudios de Caso-Control

Alelos	C	G	Total
Casos	$2f_{r_0} + f_{r_1}$	$f_{r_1} + 2f_{r_2}$	$2f_r$
Controles	$2fs_0 + fs_1$	$fs_1 + 2s_2$	$2fs$
Total	$2fn_0 + fn_1$	$fn_1 + 2fn_2$	$2fn$

5.1.3. Equilibrio Hardy-Weinberg (HWE)

La función de HWE [13] establece la relación entre el genotipo y las frecuencias alélicas en una población sin selección, mutación o migración, encontrando la distribución de probabilidad para cada alelo [97]. La probabilidad de CC es p^2, la de CG es pq y la de GG es q^2. Para probar el HWE se debe encontrar el $p-valor$, dicho valor se define como valor de significación de la prueba HWE que a nivel matemático se reduce a un valor de probabilidad obtenido por una distribución de probabilidad normal, el χ^2 debe calcularse también, haciendo una proyección de individuos esperados vs observados, la población esperada se calcula multiplicando el total de individuos por cada probabilidad definida en la Tabla **5-4**. La ecuación (5-1) muestra el cálculo de χ^2, en [16] se muestra una interpretación del χ^2.

Tabla 5-4: Distribución de probabilidades para dos alelos

	C(p)	G(q)
C(p)	p^2	pq
G(q)	pq	q^2

$$\chi^2 = \sum_{i=1}^{3} (E - O)^2 / E. \tag{5-1}$$

5.1.4. Análisis de riesgo de polimorfismo

Este análisis se utiliza para medir la magnitud de la diferencia detectada entre los genotipos identificados en los casos y los controles [95] bajo diferentes modelos logísticos de regresión de la

herencia [78, 56]. Los modelos de herencia utilizados con mayor frecuencia se definen en la Tabla **5-5**. Para cada modelo, el algoritmo calcula los odds ratios (OR) con la ecuación (5-2), la ecuación de error probabilístico en (5-3), el intervalo de confianza en la ecuación (5-4) y el $P-Valor$ en la ecuación (5-5), este prueba trata de encontrar cuál es el mejor modelo de la enfermedad relacionada con polimorfismos basados en el $P-Valor$ [100, 98]. Todos los cálculos de $P-Valor$ se realizan con precisión del $95\,\%$ y $5\,\%$ de error, aunque este valor es configurable en el algoritmo.

$$OR = TT_{controles} * TC_{casos}/(TC_{controles} * TT_{casos}) \tag{5-2}$$

$$Error = 1/TT_{controles} + 1/TC_{controles} + 1/TT_{casos} + 1/TC_{casos} \tag{5-3}$$

$$IC_{inf} = Ln(OR) + / - Z * \sqrt{Error} \tag{5-4}$$

$$P - Value\ con\ 95\,\%\ precisin\ and\ 5\,\%\ deerror \tag{5-5}$$

Tabla 5-5: Estadísticas de funciones básicas

Estadísticas de funciones básicas	
Frecuencias Genotípicas (frecuenciasGenotipicas)	
Frecuencias Alélicas (frecuenciasAlelicas)	
Hardy-Weinberg Equilibrium (equilibrioHW)	
Modelos de herencia múltiple por SNP	*Codominante*
	Dominante
	Recesivo
	Superior dominante
	Log-Aditivo

5.1.5. Entradas y salidas de los algoritmos

El archivo PED es un estándar que contiene toda la información de cierta cantidad de individuos para una cantidad determinada de SNP, es importante aclarar que cada SNP posee dos alelos, uno proveniente del padre y otro de la madre. Este archivo solo contiene la información en donde se considera que hay cambios en los SNP relacionados con una enfermedad determinada, además

Individuos	**Sexo**	**Clasificación**	SNP 1		SNP 2	
Individuo 1	1	0	A	T	C	G
Individuo 2	2	1	T	A	G	C
Individuo 3	1	1	A	A	G	G

Figura 5-1: Estructura original del archivo PED. $Sexo : (hombre = 1, mujer = 2)$, $Clasificacin : Sano(control) = 0, Enfermo(caso) = 1$

también contiene el sexo y la clasificación del individuo (Sano o Enfermo). En la figura **5-1** se observa un ejemplo del archivo Pedigree (PED).

La salida del archivo PED consiste en el análisis estadístico del mismo, como tal todos los algoritmos implementados generarán la siguiente información de salida por cada SNP: Conteo de genotipos, conteo de alelos, frecuencias alélicas, equilibrio de Hardy-Weinberg y modelos de herencia. en las Tablas **5-6**, **5-7**, **5-8**, **5-9**, **5-10** se observan las salidas del archivo PED para un SNP determinado.

Tabla 5-6: Conteo de genotipos

Gen	HCo	HCa	MCo	MCa	Total	Clasificación
GG	9	0	6	3	18	2
GA	39	10	36	8	93	3
AA	25	5	24	6	60	1
00	0	0	0	0	0	0
Total	73	15	66	17	171	0

Tabla 5-7: Conteo de alelos

Alelos	Controles	Casos	Total
G	105	24	129
A	173	40	213
Total	278	64	342

Tabla 5-8: Frecuencias alélicas

Alelos	Controles	Casos	Total
G	0.377698	0.375000	0.377193
A	0.622302	0.625000	0.622807

Tabla 5-9: Equilibrio Hardy-Weinberg

HWE	Totales
Etiqueta	P-Valor
Controles	0.317311
Casos	0.317311
Totales	0.045500

Tabla 5-10: Modelos de herencia

Modelo	Gen	Controles	Casos	OR 95 %	CInf	CSup	P-Valor
CO	AA	49	3	1.000000	0.000000	0.000000	0.000000
	GA	75	18	3.920000	1.096250	0.045500	0.000000
	GG	15	11	11.977778	2.949178	48.646483	0.002700
DO	AA	49	3	1.000000	0.000000	0.000000	0.000000
	GA+GG	90	29	5.262963	1.525179	18.161005	0.045500
RE	AA+GA	124	21	1.000000	0.000000	0.000000	0.000000
	GG	15	11	4.330159	1.751506	10.705230	0.002700
SO	AA+GG	64	14	1.000000	0.000000	0.000000	0.000000
	GA	75	18	1.097143	0.505966	2.379058	1.000000
AD	2*AA+GA	173	24	1.000000	0.000000	0.000000	0.000000
	GG	15	11	5.286111	2.176256	12.839930	0.002700

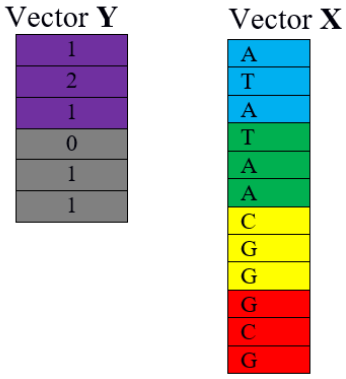


Figura 5-2: Estructura archivo PED vectorizado

5.2. Implementación de los algoritmos para el análisis estadístico de SNP

5.2.1. Implementación algorítmica del análisis de SNP en secuencial en CPU

Para hacer el análisis estadístico del archivo PED, se utilizó la siguiente estrategia. Se separó la información en dos vectores, uno que contenía el sexo y la clasificación y el otro que contenía todos los SNP. Para la estructura del algoritmo el vector de sexo y clasificación se denotará con la variable Y y el vector de SNP se denotará con la variable X.

La información de sexo y clasificación (Y) se adaptó en un vector de tamaño $individuos * 2$.
La información de SNP (X) se adaptó en un vector de tamaño $individuos * SNP * 2$.

El cálculo del tamaño de los vectores anteriores incluyen un 2 debido a que cada SNP contiene dos alelos y cuando se adapta a un solo vector se tienen dos columnas por cada SNP. En la figura **5-1** se observa el archivo PED original y en la figura **5-2** se observa el archivo vectorizado.

Para hacer el análisis de frecuencias genotípicas, alélicas, equilibrio de Hardy-Weinberg y el modelo de una manera secuencial en CPU se utilizaron dos estructuras cíclicas FOR, el primer ciclo se encargó de hacer todo el recorrido de los SNP uno por uno, es decir, este ciclo va desde 0 hasta $individuos * SNP * 2$, con incremento de $2 * individuos$. Interno al primer ciclo se crea otro que se encargará de hacer el análisis estadístico para cada SNP, este ciclo comienza con el primer individuo para cada uno de los SNP definido por el ciclo externo y finaliza en el último individuo para ese SNP. A continuación, se observa la función principal que hace el análisis del archivo PED de una manera secuencial.

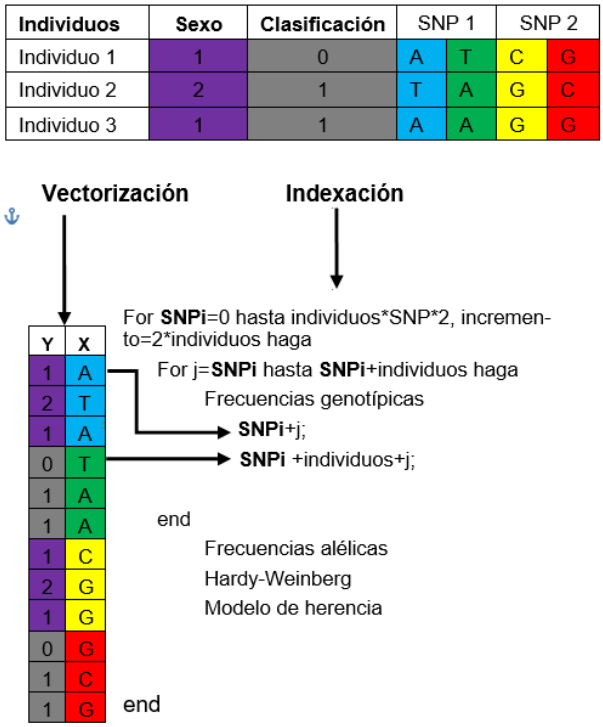


Figura 5-3: Estrategia utilizada para hacer el análisis secuencialmente en CPU.

```
#define individuos=171, SNP=1525239;
void analisisSNPSecuencial(char *y, char *x, int *z){
int j=0,SNPi=0;
  for(SNPi=0; SNPi < individuos*SNP*2 ; SNPi=SNPi+2*individuos){
      for(j= SNPi; j<( SNPi+individuos);j++){
        frecuenciasGenotipicas(y,x,z,j,SNPi);
     }
  frecuenciasAlelicas(z);
  equilibrioHW(z);
  modelo(z);
  }
}
```

Los resultados del análisis se almacenan en el vector Z, el cual es un parámento de entrada del algoritmo anterior, estos resultados se pueden observar en las Tablas **5-6**, **5-7**, **5-8**, **5-9**, **5-10**.

Nótese que la función $frecuenciasGenotipicas$ es la única que está interna a los dos ciclos y recibe como parámetros los vectores de entrada X y Y, el vector de salida Z y el contador j debido a que esta función se encarga de hacer los conteos iniciales de cada par de alelos para cada SNP discriminándolos por sexo y clasificación. Las demás funciones se encuentran sólo en el interior del primer ciclo y sólo necesitan del vector de resultados Z para hacer los respectivos cálculos y terminar así el análisis. En la figura **5-3** se ilustra la estrategia para hacer el análisis de la implementación secuencial en CPU.

5.2.2. Implementación algorítmica del análisis de SNP en paralelo en CPU

La metodología utilizada para la estrategia paralela del procesamiento de datos se basa en la taxonomía de las arquitecturas informáticas desarrolladas por Michael J. Flynn en 1972 [26]. Para el análisis básico estadístico de SNP se utilizó la taxonomía de simple instrucción sobre múltiples datos (SIMD) tanto para CPU como para GPU [11, 13].

La estrategia SIMD se basa en el almacenamiento de SNP en un arreglo dinámico en la memoria definida como $\overrightarrow{SNPVector} = \overrightarrow{\lambda}$, donde los datos de cada individuo se dividen en los diferentes núcleos de la máquina (MC). Por ejemplo, el archivo más grande que se utilizó en las pruebas contiene 171 individuos y 1525239 SNP por cada individuo, de modo que si el recuento se está ejecutando en una máquina con 8 procesadores, se distribuyen 190655 SNP por núcleo para ser categorizado. La cantidad de SNP categorizado por el núcleo está dada por la ecuación (5-6), donde $\overrightarrow{TotalSNPporNucleo} = \overrightarrow{\zeta}$ es el número total de SNP del arreglo que se ejecutarán en cada núcleo.

$$\overrightarrow{\zeta} = \overrightarrow{\lambda}/MC \tag{5-6}$$

Para hacer la implementación en paralelo sobre arquitectura CPU, el archivo PED sufrió el mismo tratamiento que en la implementación secuencial ver Figuras **5-1** y **5-2**. Los vectores tienen las mismas características que en el algoritmo secuencial.

La estrategia en paralelo en CPU se hizo por medio de hilos creando tantos hilos como núcleos tenga la máquina y fragmentandolo en los vector X y Y en porciones iguales según la cantidad de núcleos, esto quiere decir que un hilo de CPU no se encarga de gestionar un solo SNP, sino, un conjunto de ellos.

Para esta implementación es necesario utilizar dos estructuras cíclicas FOR, pero a diferencia de la implementación secuencial el ciclo externo se usará para gestionar los hilos y que parte del vector debe procesar, entonces se debe definir una parte inicial y una parte final de ejecución para cada hilo, pero una vez hecha esta gestión los hilos se ejecutarán de una manera concurrente en cada núcleo de CPU.

Es importante aclarar que el control de la ejecución en paralelo se hace por medio de los ID de los hilos, los cuales se enumeran desde 0 con incremento de 1 hasta n núcleos, ese identificador se calcula con una instrucción predefinida de la clase PTHREADS.

El ciclo interno de esta implementación se encargará de hacer el análisis estadístico para cada SNP, este ciclo comienza con el primer individuo para cada uno de los SNP definido por el ID del hilo

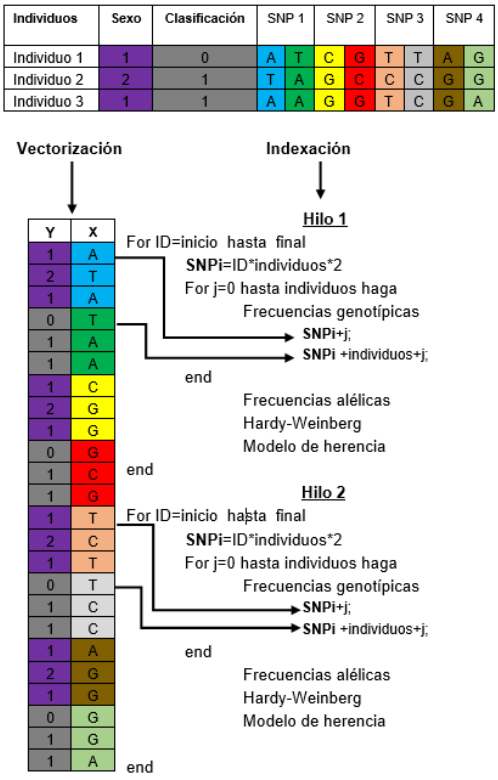


Figura 5-4: Estartegia utilizada para hacer el analisis en paralelo en CPU.

y finaliza en el último individuo para ese SNP. A continuación, se muestra la función principal que hace el análisis del archivo PED de forma paralela en CPU.

```
#define individuos=171, SNP=1525239;
void *SNPHilosCPU(void *threadarg)
{
    struct thread_data *my_data;
    my_data = (struct thread_data *) threadarg;
    int taskid = my_data->thread_id;
    int inicioT= my_data->inicioP;
    int finalT = my_data->finalP;
    int ID;
  for(ID=inicioT; ID<finalT; ID++){
    analisisSNPParaleloCPU (y,x,z,ID);
   }
    pthread_exit(NULL);
}
void analisisSNPParaleloCPU(char *y, char *x, int *z, int ID){
int SNPi=ID*Individuos*2;
int j=0;
      for(j=0;i<individuos; j++){
       frecuenciasGenotipicas(y,x,z,j,SNPi);
      }
frecuenciasAlelicas(z);
equilibrioHW(z);
modelo(z);
}
```

En la figura **5-4** se ilustra la estrategia paralela en CPU suponiendo que la máquina tiene 2 núcleos.

5.2.3. Implementación algorítmica del análisis de SNP en paralelo en GPU

Para hacer la implementación en paralelo sobre arquitectura GPU, el archivo PED sufrió el mismo tratamiento que la implementación secuencial ver Figuras **5-1** y **5-2**. Los vectores tienen las mismas características que el algoritmo en secuencial.

Se crean tantos hilos de GPU como SNP, con el objetivo de que cada hilo se encargue de hacer el análisis de cada SNP de una manera independiente, lo cual permite que, para hacer el estudio de frecuencias genotípicas, alélicas, equilibrio de Hardy -Weinberg y el modelo se haga necesario el uso de una sola estructura cíclica FOR eliminando el ciclo externo de la implementación secuencial, esto significa que no se hace un ciclo para recorrer cada SNP sino que todos se analizan de una manera concurrente por medio de hilos independientes [100, 92, 31].

Es importante aclarar que el control de la ejecución en paralelo se hace por medio de los ID de los hilos los cuales se enumeran desde 0 con incremento de 1 hasta n hilos, ese identificador se calcula con una instrucción predefinida de CUDA que se puede observar en el código.

El ciclo de esta implementación se encargará de hacer el análisis estadístico para cada SNP, este ciclo comienza con el primer individuo para cada uno de los SNP definido por el ID del hilo y finaliza en el último individuo para ese SNP. La operación $ID * individuos * 2$ se encarga de apuntar a la primera columna de cada SNP y la operación $ID * individuos * 2 + individuos$ apunta a la segunda columna de cada SNP.

A continuación, se muestra la función principal que hace el análisis del archivo PED de forma paralela en GPU.

```
1  #define individuos=171, SNP=1525239;
2  __global__ void analisisSNPParaleloGPU(char *y, char *x, int *z){
3  int ID = blockIdx.x * blockDim.x + threadIdx.x;
4  int SNPi=ID*Individuos*2;
5  int j=0;
6       for(j=0;i<individuos; j++){
7         frecuenciasGenotipicas(y,x,z,j,SNPi)
8       }
9  frecuenciasAlelicas(z)
10 equilibrioHW(z)
11 modelo(z)
12 }
```

En la figura **5-5** se muestra la estructura de la implementación paralela en GPU.

5.3. Resultados

La solución del problema fue ejecutada en secuencial en CPU y en paralelo tanto en CPU como en GPU. Para la obtención de cada simulación se repitió el experimento 20 veces para cada ejecución

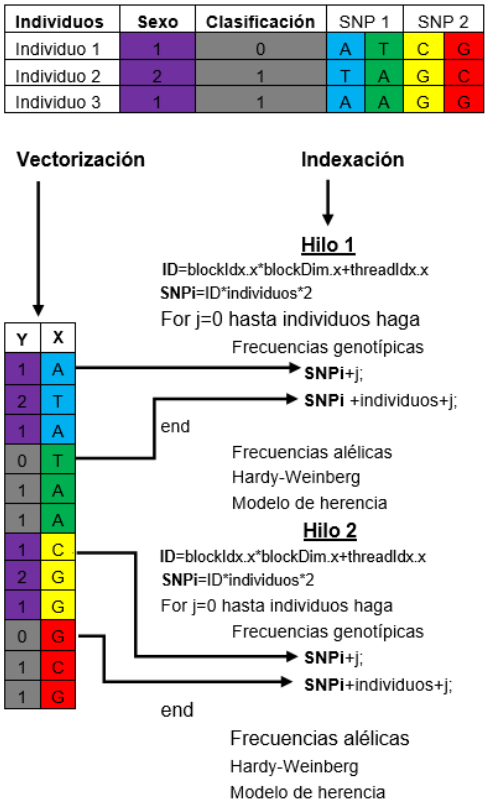


Figura 5-5: Estartegia utilizada para hacer el análisis en paralelo en GPU.

con el fin de dar resultados más confiables, la Tabla **5-11** muestra los resultados obtenidos con un conjunto básico $\{Cs\}_{i=1}^{n=48}$. Los archivos de datos contienen máximo 1525239 SNP por individuo y 171 individuos, cada SNP contiene los detalles de homocigotos con una mayor y menor frecuencia y heterocigotos para casos, controles, hombres y mujeres.

Las especificaciones de la máquina que se utilizó para la ejecución de algoritmos secuenciales y en paralelo en multi-núcleo fue: Arquitectura = $Intel(R)Xeon(R)CPUE7-4860v2@2{,}60GHz$, Núcleos de máquina = 48 y RAM=$128GB$.

Para la implementación en GPU se utilizaron las siguientes herramientas: Linux Centos 7 como sistema operativo con una versión de Kernel de $3{,}18{,}4$; lenguaje de programación C con un compilador GCC de versión $4{,}4{,}720120313$ y Phyton $2{,}7{,}10$. Toda la implementación de la solución del problema se hizo en lenguaje C, mientras que toda la gestión visual de los resultados se hizo por medio del IPhyton Notebook $3{,}1{,}0$ el cual se utilizó como intérprete Web para facilitar la programación. Para la programación paralela en GPU se utilizaron las librerías de CUDA 7.

Las especificaciones de la GPU utilizada para la ejecución de los algoritmos paralelos fue: núcleos = 2496, frecuencia de reloj = $2600MHz$, memoria RAM = $320bits$, ancho de banda = $208GB/seg$,

ancho de banda en flops = $3{,}52TFLOPS$, memoria global de la GPU = $5GB$ y procesador = $TeslaK20M$.

Tabla 5-11: Tiempo de ejecución repetíciones=20 CPU and GPU Versión paralela

Corrida(núcleos)	**1(Secuencial)**	4	8	12	16	20	24
Media(ms)	10885.55	2857.62	1463.78	1031.98	759.99	631.87	588.43
stdev(ms)	155.06	56.92	23.57	143.14	25.37	71.96	102.80
Min(ms)	10659.88	2802.30	1416.97	949.11	716.46	564.34	475.87
Max(ms)	11169.13	3027.23	1514.88	1411.81	797.05	867.18	752.74
Corrida(núcleos)	28	32	36	40	44	48	**GPU(2496)**
Media(ms)	641.47	590.42	535.79	489.18	449.60	425.45	1309.16
stdev(ms)	22.69	20.05	18.44	15.97	18.51	19.81	0.29
Min(ms)	609.16	562.28	496.99	465.00	415.43	396.56	1308.74
Max(ms)	697.98	629.12	569.90	515.50	486.79	468.02	1309.77

5.3.1. Resultados en CPU

El mejor tiempo secuencial después de 20 repeticiones fue de $T_s = 10659{,}88ms$, el mejor tiempo en paralelo con 48 núcleos fue de $T_p = 396{,}55ms$, la mejor aceleración alcanzada fue de $26{,}88X$ generando un rendimiento por núcleo del $96{,}28\,\%$. La aceleración fue calculada con la ecuación 5-7 . En la figura **5-6** se muestran los resultados de aceleración con 20 repeticiones por cada cantidad de núcleos diferentes.

$$A = T_s/T_p \tag{5-7}$$

La figura **5-7** muestra la media y la desviación estándar de 20 ejecuciones del algoritmo de la versión paralela en CPU vs la ejecución secuencial, los números de núcleos varía de acuerdo a la Tabla **5-11**, se selecciona este número de núcleos porque los otros resultados tienen tiempo de datos similares. Los mejores tiempos fueron, ejecución mínima $Tpmin = 396{,}56ms$ y ejecución máxima $Tpmax = 468{,}02ms$, ambas con 48 núcleos.

5.3.2. Resultados en GPU

La Tabla **5-11** muestra el mejor rendimiento de la GPU obtenida con la arquitectura $TeslaK20M$. El archivo de datos utilizado para el experimento tiene 171 individuos y 1525239 SNP por individuo lo que obliga a crear 2979 bloques en el código de CUDA, con 512 hilos por bloque en la GPU para cubrir el total de SNP en el archivo y las aceleraciones obtenidas en GPU se hicieron con los resultados del tiempo de ejecución mínimo en secuencial $T_{s_{min}} = 10659{,}88$ms tomado de la Tabla **5-11**.

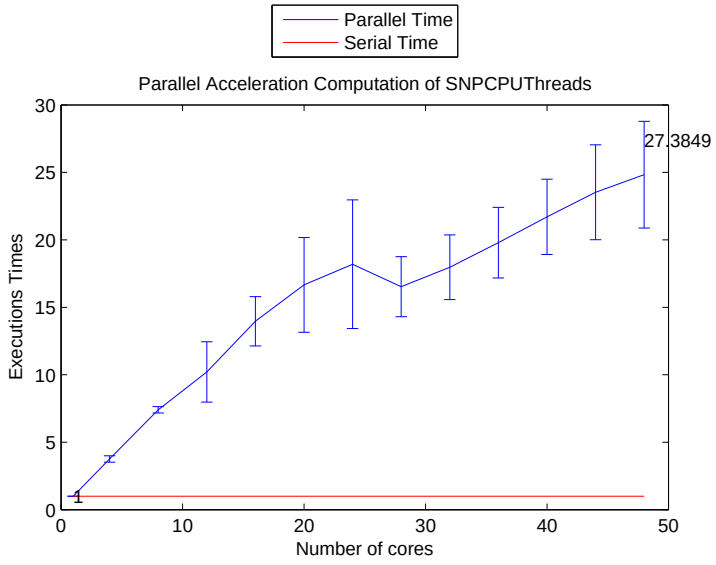


Figura 5-6: Aceleración de los algortimos en CPU con diferentes números de núcleos vs versión secuencial.

La figura **5-8** muestra que la aceleración lograda en GPU es de $8{,}15X$ veces mayor que la versión secuencial, lo que genera un rendimiento del $87,73\,\%$ por núcleo con respecto a la implementación en secuencial.

5.4. Discusión

La Tabla **5-12** muestra la mejor aceleración alcanzada con la arquitectura CPU, la cual fue de $26{,}88X$ con respecto a la secuencial, este es el valor mínimo de ejecución con 48 núcleos. En la versión paralela en GPU se logró una aceleración de $8{,}15X$ con respecto a la versión secuencial. También se muestran los mejores tiempos: $396{,}56ms$ en la versión paralela en CPU, $1308{,}74ms$ en la versión paralela en GPU y $10659{,}88ms$ en la versión secuencial. Se puede decir que la mejor aceleración para este problema se consigue con una arquitectura multi-núcleo en CPU. Pero es de aclarar que si se aumentan las operaciones por SNP es de esperar que los tiempos de GPU mejoren hasta el punto de superar a CPU, debido a que la eficiencia por núcleo en GPU es menor que la de CPU.

La figura **5-9** muestra la media de los tiempos de ejecución secuencial y en paralelo para las distintas arquitecturas tratadas, es importante aclarar que las simulaciones ejecutadas en paralelo en CPU se hicieron aumentando gradualmente el número de núcleos hasta llegar al valor máximo de 48 núcleos, mientras que en GPU todos los experimentos se hicieron con 2496 núcleos. Según la figura **5-9** se observa claramente que para pequeñas cantidades de núcleos en CPU aproximadamente 9 núcleos el tiempo de ejecución en paralelo en CPU es mayor que en GPU dando así una mejor aceleración en GPU pero después de 9 núcleos de CPU los tiempos de CPU son inferiores a

Figura 5-7: Tiempos de simulación con diferentes núcleos en Paralelo en CPU vs versión Secuencial.

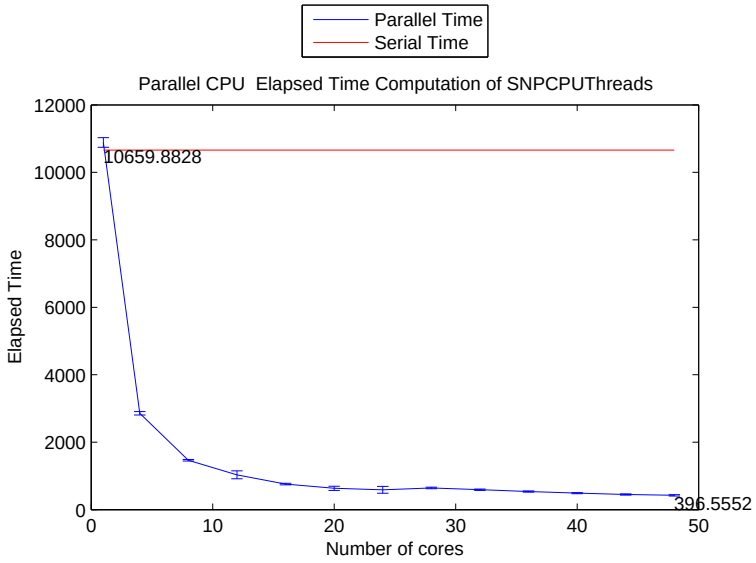


Tabla 5-12: GPU and CPU Better Results

Especificación	Secuencial-CPU	Paralelo-CPU	Paralelo-GPU
Total de núcleos	1	48	2496
Mejor ejecución en mili segundos	10659.88	396.56	1308.74
Eficiencia por núcleo %	1	96.28	87.73
Mejor aceleración	1	26.88	8.15

los de GPU generando así una mejor aceleración en CPU en comparación con GPU.

Todos los tiempos de simulación incluidos en este capítulo sólo miden la ejecución de la función principal, es decir, la que hace el análisis estadístico, en ningún momento se tomaron tiempos de lectura y escritura de archivos.

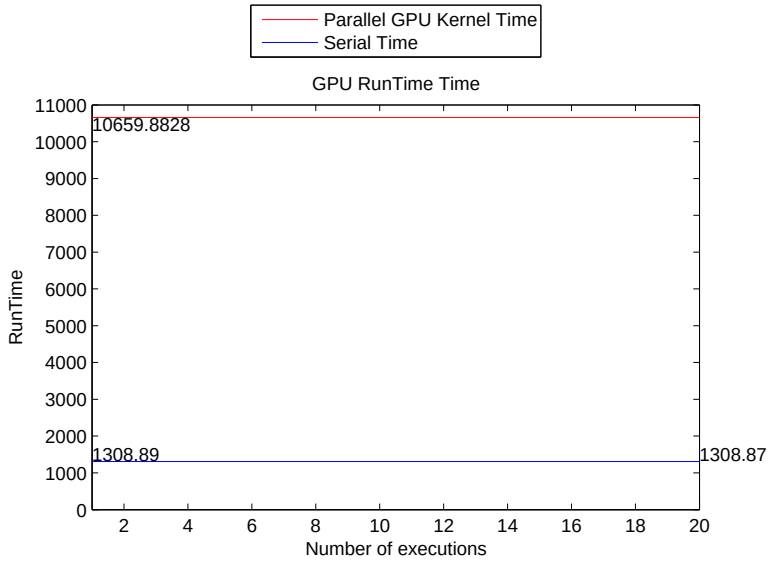
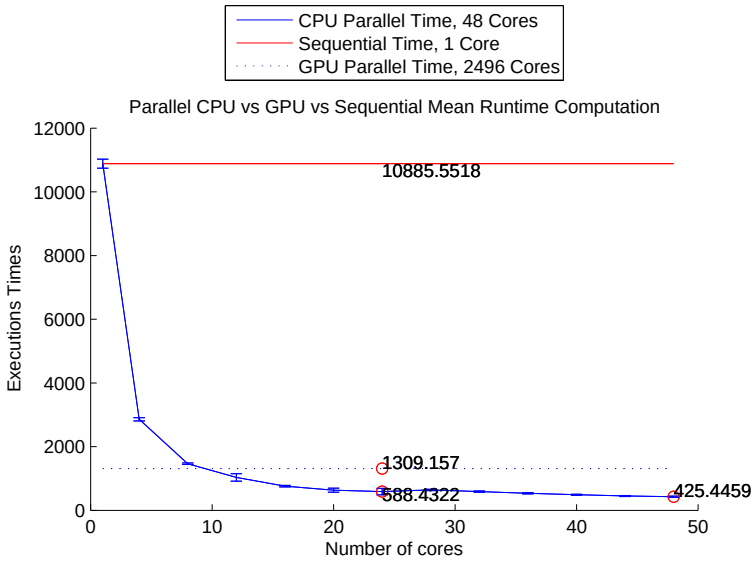


Figura 5-8: Ejecuciones en paralelo en GPU vs Versión secuencial.

Figura 5-9: Tiempos de ejecución en GPU, CPU y versión secuencial.



5.5. Conclusiones

Los tiempos de la GPU son mejores que los de la CPU cuando el uso de la CPU esta cerca de los 9 núcleos o menos, pero cuando los algoritmos paralelos en CPU utilizan más de 9 los tiempos son mejores en CPU que en GPU.

La estrategia usada para paralelizar en GPU posee mayor granularidad que en CPU debido a que la GPU posee una mayor cantidad de núcleos y por tal razón una mayor cantidad de hilos, lo cual permite pensar que si se aumentan la cantidad de SNP o la cantidad de operaciones estadísticas por SNP es probable decir que la aceleración en GPU podría llegar a dar mejor que en CPU.

El problema que se trató en este capítulo es idóneo para paralelizar debido a que el análisis básico estadístico que se hizo es independiente para cada SNP por tal razón según los resultados expuestos se lograron aceleraciones.

6 Discusión y trabajo futuro

6.1. Discusión

Al finalizar la investigación se evidencia como la computación de propósito general utilizando dispositivos de GPU o Multi-núcleo permiten obtener aceleraciones sobre algoritmos de cómputo científico. Se observó que para poder tener aceleraciones se debe tener un amplio conocimiento del problema a resolver, un profundo dominio de las herramientas computacionales de programación paralela sobre arquitecturas heterogéneas, una extensa consulta bibliográfica sobre investigaciones similares y una adaptación del problema al dispositivo empleado. A continuación se mostraran los principales resultados obtenidos en este documento de tesis.

En el segundo capítulo se ilustró el análisis de bifurcaciones y los dominios de atracción de dos sistemas utilizando algoritmos de fuerza bruta ejecutados de manera secuencial en CPU y paralela en GPU. El primer sistema que se analiza es el de Lorenz. Este sistema suave y no lineal ya ha sido estudiado empleando GPU. El segundo sistema analizado es suave a trozos y se trata del convertidor Buck controlado por onda seno. De este sistema no se reporta en la literatura algún tipo de análisis no lineal empleando GPU. Los experimentos numéricos fueron ejecutados en dos tarjetas aceleradoras en GPU de forma paralela y en CPU de manera secuencial lográndose aceleraciones de hasta $213X$. De este capítulo se publicó un artículo en el congreso *2015 IEEE 2nd Colombian Conference on Automatic Control (CCAC)* y se referencia en [30].

En el tercer y cuarto capítulo se presentó el uso de indexación bi-dimensional y tri-dimensional existente en la GPU, para acelerar algoritmos de aproximación de soluciones de sistemas de ecuaciones diferenciales parciales. Estas aproximaciones utilizan ecuaciones recurrentes en donde la dependencia de los datos vecinos juega un papel determinante en la velocidad de cómputo. Para realizar estos cálculos se involucran grandes cantidades de datos y frecuentes accesos a memoria, por lo que es conveniente utilizar estructuras computacionales que permitan realizar operaciones de forma paralela y concurrente para procesar rápidamente la información. Además, la capacidad de indexación de la memoria permite generar mejores tiempos de simulación. Se compararón tres arquitecturas diferentes y se contrastarón los desempeños contra el proceso implementado de forma secuencial sobre CPU. Los resultados mostrarón cómo se pueden lograr aceleraciones hasta de $9{,}7X$ para el caso de la ecuación de Laplace en 2 dimensiones y de $8{,}6X$ para el caso de la ecuación de Laplace en 3 dimensiones. De estos capítulos se generaron dos publicaciones una en el congreso *Computing Colombian Conference (10CCC), 2015 10th* y su referencia es [62] y otra en

la revista *Antioqueña de las Ciencias Computacionales e Ingeniería del Software* y su referencia es [61].

En el quinto capítulo se hizo énfasis en el análisis estadístico sobre polimorfismo de un solo nucleótido. El creciente volumen de información en los archivos de polimorfismos de un solo nucleótido (SNPs) para estudios de asociación de genoma completo (GWAS) ha aumentado el tiempo secuencial de procesamiento. La mayoría de los enfoques para hacer frente a la complejidad computacional de este problema prefieren investigar en la optimización del modelo matemático en lugar de considerar una arquitectura de cálculo paralelo. En este capítulo, se presentó una comparación de dos de las plataformas más populares y disponibles en el mercado, es decir, multi-núcleo (CPU - THREADS) y la unidad de procesamiento gráfico (GPU - CUDA). Los principales resultados mostraron una aceleración de $26{,}88X$ en la CPU con 48 núcleos y aceleraciones de $8{,}15X$ en la GPU con 2496 núcleos. El índice de rendimiento por cada núcleo de CPU es del $96{,}28\,\%$ y en la GPU es del $87{,}73\,\%$, con respecto a la versión secuencial de un solo núcleo. El estudio fue realizado en un archivo de 171 individuos cada uno con 1525239 SNP lo cual hace que el tamaño del archivo en disco sea de aproximadamente $1GB$. De este capítulo se tiene un artículo en proceso para someter a consideración de una revista indexada.

6.2. Trabajo futuro

- Visualización en tiempo real de las bifurcaciones, los dominios de atracción, propagación de calor en una placa de 2 dimensiones y en un volumen de 3 dimensiones, lo cual implica que los resultados deben ser enviados directamente a la salida de vídeo para mostrarlos por pantalla. Esto requiere conocimientos de computación.

- Implementación de las bifurcaciones, los dominios de atracción, análisis estadístico de SNP, propagación de calor en una placa de 2 dimensiones y en un volumen de 3 dimensiones usando librerías de OpenMP y MPI y arquitectura XeonPhi, clúster y Multi-GPU para realizar una comparación de cuál es la librería y la arquitectura más eficiente para resolver este tipo de problemas. Algunos de los problemas tratados en esta tesis no se implementaron sobre OpenCL, CUDA o Multi-núcleo razón por la cual también se proponen como trabajo futuro.

- Análisis no lineal sobre GPU del convertidor Boost Flyback. A la fecha no se reporta este tipo de análisis.

- Investigación sobre la incidencia de la precisión simple y doble sobre los tiempos de ejecución, porcentajes de error, complejidad algorítmica y consumo energético de los problemas tratados en esta tesis.

- Investigar cómo la complejidad de una algoritmo y la aproximación numérica utilizada para resolver un problema afecta los tiempos de simulación y el consumo de energía.
- Creación de algoritmos en paralelo para el análisis estadístico de SNP que incluyan estimación de las frecuencias de haplotipos [64], análisis de asociación de haplotipos con la respuesta y desequilibrio de ligamentos.

Bibliografía

[1] A. ALKHATHLAN, A A. ; KHALIFA, M. *On OpenCL and Chaotic Phenomena*. 2015

[2] AHAMED, 4. A. ; MAGOULES, F.: Iterative methods for sparse linear systems on graphics processing unit. En: *IEEE 14th International onference on High Performance Computing and Communication* (2012)

[3] AMDAHL, Gene M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. En: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. New York, NY, USA : ACM, 1967 (AFIPS '67 (Spring)), p. 483–485

[4] ANTAL, M.J.J.: Biomass Pyrolysis: A Review of the Literature Part 1 - Carbohydrate Pyrolysis. En: *Advances in Solar Energy Vol. 1 American Solar Energy Society*, 1982, p. 61–111

[5] A.R.BRODTKORB.: *Scientic Computing on Heterogeneous Architectures.*, Faculty of Mathematics and Natural Sciences, University of Oslo., Tesis de Grado, 2010

[6] BANERJEE, S. ; VERGHESE, G. C.: *Nonlienar phenomena in power electronics*. Wiley, 2001

[7] BASTIDAS, José Daniel M.: *Analysis and bifurcations of a DC-DC buck converter controlled by sine wave*, Universidad Nacional de Colombia, Tesis de Grado, 2012

[8] BELL, Nathan ; DALTON, Steven ; OLSON, Luke N.: Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods. En: *SIAM Journal on Scientific Computing* 34 (2012), Nr. 4, p. C123–C152

[9] BERTRAM, L. ; TANZI, R. E.: Genome-wide association studies in Alzheimer's disease. En: *Human Molecular Genetics* 18 (2009), Nr. R2, p. R137–R145. – ISSN 0964–6906

[10] BOLZ, Jeff ; FARMER, Ian ; GRINSPUN, Eitan ; SCHRÖODER, Peter: Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. En: *ACM Trans. Graph.* 22 (2003), Juli, Nr. 3, p. 917–924. – ISSN 0730–0301

[11] BOTTOLO, Leonardo ; CHADEAU-HYAM, Marc ; HASTIE, David I. ; ZELLER, Tanja ; LIQUET, Benoit ; NEWCOMBE, Paul ; YENGO, Loic ; WILD, Philipp S. ; SCHILLERT, Arne ;

ZIEGLER, Andreas ; NIELSEN, Sune F. ; BUTTERWORTH, Adam S. ; HO, Weang K. ; CASTAGNE, Rapha??le ; MUNZEL, Thomas ; TREGOUET, David ; FALCHI, Mario ; CAMBIEN, Francois ; NORDESTGAARD, Borge G. ; FUMERON, Frederic ; TYBJAERG-HANSEN, Anne ; FROGUEL, Philippe ; DANESH, John ; PETRETTO, Enrico ; BLANKENBERG, Stefan ; TIRET, Laurence ; RICHARDSON, Sylvia: GUESS-ing Polygenic Associations with Multiple Phenotypes Using a GPU-Based Evolutionary Stochastic Search Algorithm. En: *PLoS Genetics* 9 (2013), Nr. 8. – ISBN 1553–7404 (Electronic)\r1553–7390 (Linking)

[12] B.ROCKER., H.Anzt. T.Hahn. V.: GPU Accelerated Scientific Computing: Evaluation of the NVIDIA Fermi Architecture; Elementary Kernels and Linear Solvers. En: *Engineering Mathematics and Computing Lab.E. Karlsruhe.* (2010)

[13] BROWNING, Brian L. ; BROWNING, Sharon R.: A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. En: *American journal of human genetics* 84 (2009), feb, Nr. 2, p. 210–23. – ISSN 1537–6605

[14] BUTLER, Ralph M. ; LUSK, Ewing L.: Message Passing Interfaces Monitors, messages, and clusters: The p4 parallel programming system. En: *Parallel Computing* 20 (1994), Nr. 4, p. 547 – 564. – ISSN 0167–8191

[15] C. GREBOGI, E. O. ; YORKE, J.A.: Basin Boundary Metamorphoses: Changes in Accessible Boundary Orbits. En: *Physica D* (1987)

[16] CERDA L, JAIME ; VILLARROEL DEL P, LUIS: Interpretación del test de Chi-cuadrado (X2) en investigación pedítrica. En: *Revista chilena de pediatra* 78 (2007), 08, p. 414 – 417. – ISSN 0370–4106

[17] CHANDRA, Robit ; DAGUM, Leonardo ; KOHR, Dave ; MAYDAN, Dror ; MCDONALD, Jeff ; MENON, Ramesh: *Parallel Programming in OpenMP*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2001. – ISBN 1–55860–671–8, 9781558606715

[18] CHEN, Shanshan ; WANG, Xiaojing ; WANG, Junhan ; ZHAO, Yuanyuan ; WANG, Dan ; TAN, Chengcheng ; FA, Jingjing ; ZHANG, Rongfeng ; WANG, Fan ; XU, Chaoping ; HUANG, Yufeng ; LI, Sisi ; YIN, Dan ; XIONG, Xin ; LI, Xiuchun ; CHEN, Qiuyun ; TU, Xin ; YANG, Yanzong ; XIA, Yunlong ; XU, Chengqi ; WANG, Qing K.: Genomic variant in CAV1 increases susceptibility to coronary artery disease and myocardial infarction. En: *Atherosclerosis* 246 (2016), mar, p. 148–56. – ISSN 1879–1484

[19] CHIKKAGOUDAR, Satish ; WANG, Kai ; LI, Mingyao: GENIE: a software package for gene-gene interaction analysis in genetic association studies using multiple GPU or CPU cores. En: *BMC research notes* 4 (2011), Nr. 1, p. 158. – ISBN 1756–0500

[20] COHEN., J.: High Performance Algebraic Multigrid for Commercial Applications. En: *GPU technology conference.* (2013)

[21] Kap. Symbolic Testing of OpenCL Code In: COLLINGBOURNE, Peter ; CADAR, Cristian ; KELLY, Paul H. J.: *Hardware and Software: Verification and Testing: 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 203–218. – ISBN 978–3–642–34188–5

[22] CRAWFORD, Gregory E. ; HOLT, Ingeborg E. ; WHITTLE, James ; WEBB, Bryn D. ; TAI, Denise ; DAVIS, Sean ; MARGULIES, Elliott H. ; CHEN, Yidong ; BERNAT, John A. ; GINSBURG, David ; ZHOU, Daixing ; LUO, Shujun ; VASICEK, Thomas J. ; DALY, Mark J. ; WOLFSBERG, Tyra G. ; COLLINS, Francis S.: Genome-wide mapping of DNase hypersensitive sites using massively parallel signature sequencing (MPSS). (2006), p. 123–131

[23] D.EGLOFF.: High Performance Finite Difference PDE Solvers on GPUs. En: *QuantAlea GmbH-ResearchGate* (2010)

[24] E. HAIRER, S. N. ; WANNER, G.: *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer-Verlag, 1993

[25] ESTRADA, Karol ; ABUSEIRIS, Anis ; GROSVELD, Frank G. ; UITTERLINDEN, André G. ; KNOCH, Tobias a. ; RIVADENEIRA, Fernando: GRIMP: A web- and grid-based tool for high-speed analysis of large-scale genome-wide association using imputed data. En: *Bioinformatics* 25 (2009), Nr. 20, p. 2750–2752. – ISBN 2200820186

[26] FLYNN, Michael J.: Some computer organization and their effectiveness. En: *IEEE Transaction on Computers* C-21 (1972), Nr. 9, p. 948–960

[27] FOSSAS, E. ; OLIVAR, G.: Study of chaos in the buck converter. En: *IEE Transactions on Circuits and systems I* (2002)

[28] FOSTER, I. ; ZHAO, Y. ; RAICU, I. ; LU, S.: Cloud Computing and Grid Computing 360-Degree Compared. En: *2008 Grid Computing Environments Workshop*, 2008. – ISSN 2152–1085, p. 1–10

[29] GAIKWAD, Abhijeet ; TOKE, Ioane M.: GPU Based Sparse Grid Technique for Solving Multidimensional Options Pricing PDEs. En: *Proceedings of the 2Nd Workshop on High Performance Computational Finance*. New York, NY, USA : ACM, 2009 (WHPCF '09). – ISBN 978–1–60558–716–5, p. 6:1–6:9

[30] GALLO, G. ; TABARES, R. ; OSORIO, G.: Nonlinear analysis of the Lorenz system and buck converter using GPU. En: *Automatic Control (CCAC), 2015 IEEE 2nd Colombian Conference on*, 2015, p. 1–6

[31] GONZALEZ-DOMINGUEZ, Jorge ; SCHMIDT, Bertil ; KASSENS, Jan C. ; WIENBRANDT, Lars: Hybrid CPU/GPU acceleration of detection of 2-SNP epistatic interactions in GWAS.

En: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8632 LNCS (2014), p. 680–691. – ISBN 9783319098722

[32] GONZALEZ-DOMINGUEZ, Jorge ; WIENBRANDT, Lars ; KAASSENS, Jan C. ; ELLINGHAUS, David ; SCHIMMLER, Manfred ; SCHMIDT, Bertil: Parallelizing epistasis detection in GWAS on FPGA and GPU-accelerated computing systems. En: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 12 (2015), Nr. 5, p. 982–994. – ISSN 15455963

[33] GRIFFITHS, Anthony J. ; MILLER, Jeffrey H. ; SUZUKI, David T. ; LEWONTIN, Richard C. ; GELBART., William M.: *An Introduction to Genetic Analysis. 7th edition.* New York: W. H. Freeman;, 2000

[34] GROPP, William ; LUSK, Ewing ; DOSS, Nathan ; SKJELLUM, Anthony: A high-performance, portable implementation of the {MPI} message passing interface standard. En: *Parallel Computing* 22 (1996), Nr. 6, p. 789 – 828. – ISSN 0167–8191

[35] GUCKENHEIMER, J. ; WILLIAMS, R. F.: Structural Stability of Lorenz Attractors. (1979)

[36] GUI, Hongsheng ; LI, Miaoxin ; SHAM, Pak C. ; CHERNY, Stacey S.: Comparisons of seven algorithms for pathway analysis using the WTCCC Crohn's Disease dataset. En: *BMC research notes* 4 (2011), jan, Nr. 1, p. 386. – ISSN 1756–0500

[37] GUKENHEIMER, J. ; HOLMES, P.: *Dynamical Systems, and Bifurcations of Vector Fields*. Springer-Verlag, 1983

[38] H. PEITGEN, H. J. ; SAUPE, D.: *Chaos and Fractals: New Frontiers of Science*. Springer-Verlag, 1992

[39] HALLBERG, Niclas. Ibañez Luisa. Bondon-Guitton Emmanuelle. Kreutz Reinhold. Carvajal Alfonso. Lucena M Isabel. Ponce Esther Sancho. Molokhia Mariam. Martin Javier. Axelsson Tomas. Yue Qun-Ying. Magnusson Patrik K E. Wadelius M.: Genetic variants associated with antithyroid drug-induced agranulocytosis: a genome-wide association study in a European population. En: *The Lancet Diabetes and Endocrinology* 4 (2016), Nr. 6, p. 507 – 516. – ISSN 2213–8587

[40] HURRELL, Alice ; REEBA, Oliver ; FUNLAYO, Odejinmi: Recurrent ectopic pregnancy as a unique clinical sub group: a case control study. En: *SpringerPlus* 5 (2016), jan, Nr. 1, p. 265. – ISSN 2193–1801

[41] ITU, L. M. ; SUCIU, C. ; MOLDOVEANU, F. ; POSTELNICU, A. ; SUCIU, C.: GPU optimized computation of stencil based algorithms. En: *Roedunet International Conference (RoEduNet), 2011 10th*, 2011. – ISSN 2068–1038, p. 1–6

[42] J.EATON.: GPU-Accelerated Algebraic Multigrid for Commercial Applications. Manager NVAMG CUDA Library. En: *NVIDIA*. (2013)

[43] JIN, Haoqiang ; JESPERSEN, Dennis ; MEHROTRA, Piyush ; BISWAS, Rupak ; HUANG, Lei ; CHAPMAN, Barbara: High performance computing using {MPI} and OpenMP on multi-core parallel systems. En: *Parallel Computing* 37 (2011), Nr. 9, p. 562 – 575. – Emerging Programming Paradigms for Large-Scale Scientific Computing. – ISSN 0167–8191

[44] KINDRATENKO, V.: *Numerical Computations with GPUs*. Springer, 2014

[45] KNIBBE, H. ; OOSTERLEE, C.W. ; VUIK, C.: GPU Implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method. En: *Journal of Computational and Applied Mathematics* 236 (2011), Nr. 3, p. 281 – 293. – Aspects of Numerical Algorithms, Parallelization and Applications. – ISSN 0377–0427

[46] KOO, C.L.a ; LIEW, M.J.a ; MOHAMAD, M.S.a ; SALLEH, A.H.M.a ; DERIS, S.a ; IBRAHIM, Z.B, SUSILO, B.c ; HENDRAWAN, Y.c ; WARDANI, A.K: Software for detecting gene-gene interactions in genome wide association studies. En: *Biotechnology and Bioprocess Engineering* 20 (2015), Nr. 4, p. 662–676

[47] KUZNETSOV, Yu.A.: *Elements of Applied Bifurcation Theory*. Springer, 2004

[48] LEE, Sungyoung ; KWON, Min S. ; HUH, Ik S. ; PARK, Taesung: CUDA-LR: CUDA-accelerated logistic regression analysis tool for gene-gene interaction for genome-wide association study. En: *2011 IEEE International Conference on Bioinformatics and Biomedicine Workshops, BIBMW 2011* (2011), p. 691–695. ISBN 9781457716133

[49] LENG, Shuguang ; THOMAS, Cynthia L. ; SNIDER, Amanda M. ; PICCHI, Maria A. ; CHEN, Wenshu ; WILLIS, Derall G. ; CARR, Teara G. ; KRZEMINSKI, Jacek ; DESAI, Dhimant ; SHANTU, Amin ; LIN, Yong ; JACOBSON, Marty R. ; BELINSKY, Steven A.: Radon Exposure, IL-6 Promoter Variants, and Lung Squamous Cell Carcinoma in Former Uranium Miners. En: *Environmental health perspectives* 124 (2016), apr, Nr. 4, p. 445–51. – ISSN 1552–9924

[50] LEWALLEN, Susan ; COURTRIGHT, Paul: Epidemiology in practice: Case-control studies. En: *Community Eye Health Journal* 11 (1998), Nr. 28, p. 57–58. – ISBN 0953–6833

[51] LI, 6. R. ; SAAD, Y.: GPU-accelerated preconditioned iterative linear solvers. En: *The Journal of supercomputing* (2013)

[52] LIN, Ming ; WEI, Lee-Jen ; SELLERS, William R. ; LIEBERFARB, Marshall ; WONG, Wing H. ; LI, Cheng: dChipSNP: significance curve and clustering of SNP-array-based loss-of-heterozygosity data. En: *Bioinformatics* 20 (2004), Nr. 8, p. 1233–1240

[53] Kap. Heterogeneous Systems for Energy Efficient Scientific Computing In: LIU, Qiang ; LUK, Wayne: *Reconfigurable Computing: Architectures, Tools and Applications: 8th International Symposium, ARC 2012, Hong Kong, China, March 19-23, 2012. Proceedings*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 64–75. – ISBN 978–3–642–28365–9

[54] LIU, Yang: *Data Mining Methods for Single Nucleotide Polymorphisms Analysis in Computational Biology*, Tesis de Grado, 2011. – AAI3510948

[55] LORENZ, E. N.: Deterministic Nonperiodic Flow. En: *Journal of the Atmospheric Sciences* (1963)

[56] LU, Mian ; ZHAO, Jiuxin ; LUO, Qiong ; WANG, Bingqiang ; FU, Shaohua ; LIN, Zhe: GSNP: A DNA single-nucleotide polymorphism detection system with GPU acceleration. En: *Proceedings of the International Conference on Parallel Processing* (2011), p. 592–601. – ISBN 9780769545103

[57] LUEBKE, D.: CUDA: Scalable parallel programming for high-performance scientific computing. En: *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, 2008. – ISSN 1945–7928, p. 836–838

[58] NE. M.ALVES. D.MEDEL., N.Rodriguez. M.Murazzo. D.: Integración de Computación Heterogénea con Hadoop para Cloud Computing. En: *XV workshop de investigadores en ciencia de la computación.* (2013)

[59] MATUSEVICH, David S. ; CABRERA, Wellington ; ORDONEZ, Carlos: Accelerating a Gibbs sampler for variable selection on genomics data with summarization and variable pre-selection combining an array DBMS and R. En: *Machine Learning* 102 (2015), Nr. 3, p. 483–504. – ISBN 1099401555348

[60] MERRISON-HORT, R. *Fireflies: New software for interactively exploring dynamical systems using GPU computing*. 2015

[61] MONSALVE, M. A. T. ; CASTRILLON, N. L. M. ; SOTO, R. T. ; OSORIO, G.: Indexación tri-dimensional empleando GPU para acelerar la aproximación numérica de soluciones de la ecuación de Laplace. En: *Revista Antioqueña de las Ciencias Computacionales e Ingenieria del Software.*, 2015, p. 37–42

[62] MONSALVE, M. A. T. ; CASTRILLON, N. L. M. ; SOTO, R. T. ; OSORIO, G.: Indexing GPU acceleration for solutions approximation of the Laplace equation. En: *Computing Colombian Conference (10CCC), 2015 10th*, 2015, p. 568–574

[63] MOVCHAN, Aleksander V. ; ZYMBLER, Mikhail L.: *Parallel Algorithm for Local-best-match Time Series Subsequence Similarity Search on the Intel MIC Architecture*. Vol. 66. Elsevier Masson SAS, 2015. – 63–72 p. ISSN 18770509

[64] MUNOZ FERNÁNDEZ, Adriana ; PUIG CANO, Adrianet ; CABRERA ZAMORA, Maritza ; FERNÁNDEZ ÁGUILA, Julio ; MARTÍNEZ ANTUAPMA, Gisela: Marcadores geníticos en pacientes con anemia drepanocítica de la provincia de Cienfuegos: Haplotipos del bloque b y a -Talasemia. En: *Revista Cubana de Hematología, Inmunología y Hemoterapia* 16 (2000), 08, p. 142 – 144. – ISSN 0864–0289

[65] MUNSHI, A.: The OpenCL specification. En: *2009 IEEE Hot Chips 21 Symposium (HCS)*, 2009, p. 1–314

[66] N. MOHAN, T. M. U. ; ROBBINS, W. P.: *Electrónica de potencia: Convertidores, aplicaciones y diseño*. McGraw Hill, 2009

[67] N.ALIAS.: Some parallel numerical method in solving parial differential equations. En: *The 2nd International Conference on Computer Engineering and Technology (ICCET 2010)-ResearchGate* (2010)

[68] NATHAN WHITEHEAD, Alex Fit-Florea: Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs. En: *Mathematical Institute University of Oxford.* (2012)

[69] Y NESTOR DUQUE MENDEZ, Gustavo I.: Arquitecturas y modelos de programación en computación grid. En: *Scientia Et Technica* XIII (2007), Nr. 37. – ISSN 0122–1701

[70] NICKOLLS, John ; BUCK, Ian ; GARLAND, Michael ; SKADRON, Kevin: Scalable Parallel Programming with CUDA. En: *Queue* 6 (2008), März, Nr. 2, p. 40–53. – ISSN 1542–7730

[71] OLIVAR, Gerard: *Chaos in the Buck Converter*, Tesis de Grado, 1997

[72] ONSORY, Khadijeh ; MOUSAVI, Mona ; NOURI, Zahra Haji M. ; BARZI, Nastaran V.: Association between estrogen and progesterone receptors gene polymorphisms with prostate cancer. En: *Koomesh* 17 (2016), Nr. 2, p. 455–463. – ISSN 16087046

[73] Kap. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing In: OSTERMANN, Simon ; IOSUP, Alexandria ; YIGITBASI, Nezih ; PRODAN, Radu ; FAHRINGER, Thomas ; EPEMA, Dick: *Cloud Computing: First International Conference, CloudComp 2009 Munich, Germany, October 19–21, 2009 Revised Selected Papers*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, p. 115–131. – ISBN 978–3–642–12636–9

[74] PEI, Yu-Fang ; LI, Jian ; ZHANG, Lei ; PAPASIAN, Christopher J. ; DENG, Hong-Wen: Analyses and comparison of accuracy of different genotype imputation methods. En: *PloS one* 3 (2008), jan, Nr. 10, p. e3551. – ISSN 1932–6203

[75] PRABHU, Snehit ; PE, Itsik: Ultrafast genome-wide scan for SNP – SNP interactions in common complex disease. (2012), p. 1–11. – ISBN 1549–5469 (Electronic)\r1088–9051 (Linking)

[76] R. BARRIO, A. S. ; SHILNIKOV, L.: Kneadings, Symbolic Dynamics and Painting Lorenz Chaos. En: *International Journal of Bifurcation and Chaos* (2012)

[77] RABENSEIFNER, R. ; HAGER, G. ; JOST, G.: Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes. En: *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, 2009. – ISSN 1066–6192, p. 427–436

[78] RAMOS, Bruna Ribeiro de A. ; MENDES, Niele D. ; TANIKAWA, Aline A. ; AMADOR, Marcos Antônio T. ; DOS SANTOS, Ney Pereira C. ; DOS SANTOS, Sidney Emanuel B. ; CASTELLI, Erick C. ; WITKIN, Steven S. ; DA SILVA, Márcia G.: Ancestry informative markers and selected single nucleotide polymorphisms in immunoregulatory genes on preterm labor and preterm premature rupture of membranes: a case control study. En: *BMC Pregnancy and Childbirth* 16 (2016), Nr. 1, p. 30. – ISSN 1471–2393

[79] RAND, D.: The Topological Classification of Lorenz Attractors. En: *Proc. Cambridge Philos. Soc.* (1978)

[80] RASHID, Muhammad H.: *Power electronics handbook*. Academic Press, 2001

[81] REIMUNDEZ, Carlos J.: *Estudio de rendimiento en GPU*, Universidad Complutense de Madrid, Tesis de Grado, 2010

[82] RODRÍGUEZ-LÓPEZ, Raquel ; GONZÁLEZ-CARPIO, Marta ; SERRANO, M. V. ; TORRES, Guadalupe ; DE CÁCERES, M. Teresa G. ; HERRERA, Trinidad ; ÁNGEL ROMÁN ; RUBIO, Marta ; MÉNDEZ, Pilar ; HERNÁNDEZ-SÁEZ, Rosario ; NÚÑEZ, Manuela ; LUENGO, Luis M.: Asociación de polimorfismos en el gen {FTO} con la obesidad mórbida en la población extremeña. En: *Endocrinología y Nutrición* 57 (2010), Nr. 5, p. 203 – 209. – ISSN 1575–0922

[83] RP.CANALE, S.C.Chapra.: *Métodos Numéricos para Ingenieros.* Mcgraw-Hill., 2006

[84] SAUNDERS, Edward J. ; DADAEV, Tokhir ; LEONGAMORNLERT, Daniel A. ; OLAMA, Ali Amin A. ; BENLLOCH, Sara ; GILES, Graham G. ; WIKLUND, Fredrik ; GRÖNBERG, Henrik ; HAIMAN, Christopher A. ; SCHLEUTKER, Johanna ; NORDESTGAARD, Børge G ; TRAVIS, Ruth C. ; NEAL, David ; PASAYAN, Nora ; KHAW, Kay-Tee ; STANFORD, Janet L. ; BLOT, William J. ; THIBODEAU, Stephen N. ; MAIER, Christiane ; KIBEL, Adam S. ; CYBULSKI, Cezary ; CANNON-ALBRIGHT, Lisa ; BRENNER, Hermann ; PARK, Jong Y. ; KANEVA, Radka ; BATRA, Jyotsna ; TEIXEIRA, Manuel R. ; PANDHA, Hardev ; GOVINDASAMI, Koveela ; MUIR, Ken ; EASTON, Douglas F. ; EELES, Rosalind A. ; KOTE-JARAI, Zsofia: Gene and pathway level analyses of germline DNA-repair gene variants and prostate cancer susceptibility using the iCOGS-genotyping array. En: *British journal of cancer* 114 (2016), mar, Nr. 8, p. 945–952. – ISSN 1532–1827

[85] SOLE, Xavier ; GUINO, Elisabet ; VALLS, Joan ; INIESTA, Raquel ; MORENO, Victor: SNPStats: A web tool for the analysis of association studies. En: *Bioinformatics* 22 (2006), Nr. 15, p. 1928–1929. – ISBN 3900051070

[86] TABOR, M.: *Chaos and Integrability in Nonlinear Dynamics: An Introduction*. Wiley, 1989

[87] THALL, Andrew: Extended-precision Floating-point Numbers for GPU Computation. En: *ACM SIGGRAPH 2006 Research Posters*. New York, NY, USA : ACM, 2006 (SIGGRAPH '06). – ISBN 1–59593–364–6

[88] URIBE-PAREDES, Roberto ; CAZORLA, Diego ; ARIAS, Enrique ; SÁNCHEZ, José L.: Un sistema heterogéneo Multicore/GPU para acelerar la búsqueda por similitud en estructuras métricas. En: *Ingeniare. Revista chilena de ingeniería* 22 (2014), 01, p. 26 – 40. – ISSN 0718–3305

[89] V. ACARY, O. B. ; BROGLIATO, B.: *Nonsmooth Modelind and Simulation for Switched Circuits*. Springer, 2011

[90] VILLEGAS., J.J.Ramírez. C.A. Vanegas. A.: Método de Diferencias Finitas para la Solución de Ecuaciones en Derivadas Parciales. En: *Scielo* (2006)

[91] WANG, Qian ; LU, Qiongshi ; ZHAO, Hongyu: A review of study designs and statistical methods for genomic epidemiology studies using next generation sequencing. En: *Frontiers in genetics* 6 (2015), jan, Nr. MAR, p. 149. – ISSN 1664–8021

[92] WANG, Qiao ; SHI, Fan ; KOWALCZYK, Andrew ; CAMPBELL, Richard M. ; GOUDEY, Benjamin ; RAWLINSON, David ; HARWOOD, Aaron ; FERRA, Herman ; KOWALCZYK, Adam: GWISFI: A universal GPU interface for exhaustive search of pairwise interactions in case-control GWAS in minutes. En: *Proceedings - 2014 IEEE International Conference on Bioinformatics and Biomedicine, IEEE BIBM 2014* (2014), p. 403–409. ISBN 9781479956692

[93] WEINBUB., M. Wagner. K. Rupp. J.: A Comparison of Algebraic Multigrid Preconditioners using Graphics Processing Units and Multi-Core Central Processing Units. En: *Institute for Microelectronics-TU Wien.* (2012)

[94] WELLCOME, The ; CASE, Trust ; CONSORTIUM, Control: Genome-wide association study of 14,000 cases of seven common diseases and 3,000 shared controls. En: *Nature* 447 (2007), jun, Nr. 7145, p. 661–78. – ISSN 1476–4687

[95] WU, Michael C. ; KRAFT, Peter ; EPSTEIN, Michael P. ; TAYLOR, Deanne M. ; CHANOCK, Stephen J. ; HUNTER, David J. ; LIN, Xihong: Powerful SNP-Set Analysis for Case-Control Genome-wide Association Studies. En: *American Journal of Human Genetics* 86 (2010), Nr. 6, p. 929–942. – ISBN 1537–6605 (Electronic)\r0002–9297 (Linking)

[96] XUE-XIN LIU, Hai W. ; YU, Hao: A GPU-Accelerated Envelope-Following Method for Switching Power Converter Simulation. En: *Design, Automation and Test in Europe Conference and Exhibition* (2012)

[97] YOO, Jinho ; LEE, Youngbok ; KIM, Yujung ; RHA, Sun Y. ; KIM, Yangseok: SNPAnalyzer 2.0: a web-based integrated workbench for linkage disequilibrium analysis and association analysis. En: *BMC bioinformatics* 9 (2008), Nr. 1, p. 290. – ISBN 1471–2105 (Electronic)\r1471–2105 (Linking)

[98] YUNG, Ling S. ; YANG, Can ; WAN, Xiang ; YU, Weichuan: GBOOST: A GPU-based tool for detecting gene-gene interactions in genome-wide case control studies. En: *Bioinformatics* 27 (2011), Nr. 9, p. 1309–1310. – ISBN 1367–4811 (Electronic)\n1367–4803 (Linking)

[99] ZHENG, Xiuwen ; LEVINE, David ; SHEN, Jess ; GOGARTEN, Stephanie M. ; LAURIE, Cathy ; WEIR, Bruce S.: A high-performance computing toolset for relatedness and principal component analysis of SNP data. En: *Bioinformatics* 28 (2012), Nr. 24, p. 3326–3328

[100] ZHU, Zhixiang ; TONG, Xiaoran ; ZHU, Zhihong ; LIANG, Meimei ; CUI, Wenyan ; SU, Kunkai ; LI, Ming D. ; ZHU, Jun: Development of GMDR-GPU for Gene-Gene Interaction Analysis and Its Application to WTCCC GWAS Data for Type 2 Diabetes. En: *PLoS ONE* 8 (2013), Nr. 4. – ISBN 1932–6203 (Electronic)\r1932–6203 (Linking)