Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXIX, págs. 3155-3167 (artículo completo)
Eduardo Dvorkin, Marcela Goldschmit, Mario Storti (Eds.)
Buenos Aires, Argentina, 15-18 Noviembre 2010

ESTIMACIÓN DE LOS PARÁMETROS DE RENDIMIENTO DE UNA GPU

Cristian Perez[a,b] **y Fabiana Piccoli**[a]

[a] *Universidad Nacional de San Luis, Ejército de los Andes 950, 5700 - San Luis - Argentina*
[b] *gridTICs - Univ. Tecnológica Regional Mendoza Mendoza, Argentina*

e-mail: {mpiccoli}@unsl.edu.ar

Palabras clave:

Unidades de Porcesamiento Gráfico, Parámetros de Performance, Potencia de Cálculo, Ancho de Banda de Memoria

Resumen.

El poder computacional asociado a las tecnologías dedicadas a fines específicos, su constante avance y el bajo costo, han constituido una alternativa válida a las supercomputadoras paralelas. El ejemplo más popular de las tecnologías dedicadas es la Unidad de Procesamiento Gráfico (GPU). Una tarjeta de video puede proporcionar hasta 50 veces más poder de cómputo que la computadora huésped en algunas aplicaciones.

En este trabajo se plantea realizar un análisis del rendimiento de la arquitectura GPU siguiendo el modelo de programación de CUDA (por su denominación en inglés: Compute Unified Device Architecture) a fin de evaluar los alcances de su potencialidad y establecer con mayor exactitud sus limitaciones. Esto permite además conocer, por un lado la estructura de la implementación de cada fabricante particular (particularmente, en este trabajo nos centramos en NVIDIA) y, por el otro posibilitar y/o facilitar la predicción de desempeño de aplicaciones desarrolladas en la arquitectura. El análisis se lleva a cabo sobre los parámetros: Ancho de Banda de Memoria Global de GPU, Ancho de Banda de Memoria Compartida (Shared) y Capacidad o Potencia de Cálculo. Su estimación se realiza mediante aplicaciones que incluyen operaciones aritméticas y/o búsqueda de patrones. Finalmente se presentan los resultados obtenidos para diferentes arquitecturas y generaciones de GPUs, realizando un estudio comparativo de los resultados alcanzados con los existentes en la bibliografía.

1. INTRODUCCIÓN

Ante la aparición de las GPU (Unidad de Procesamiento Gráfico o en inglés: Graphic Processing Unit) y su rápida adopción como nueva plataforma de hardware, no sólo para aplicaciones íntimamente relacionadas a su finalidad sino para cómputo de propósito general, se abre un enorme universo de posibilidades, principalmente para la resolución de aplicaciones altamente paralelizables y con gran requerimiento de recursos computacionales.

Por muchos años la GPU fue utilizada exclusivamente para acelerar el cálculo de ciertas aplicaciones relacionadas directamente con el procesamiento de imágenes, tal como videojuegos o 3D interactivas. Su buen desempeño en este ámbito, junto a su constante y rápida evolución (comparada con los microprocesadores de propósito general), un número de instrucciones menor, y sin aritmética de doble precisión Lieberman et al. (2008); Luebke (2007), ha permitido desarrollar un modelo de supercómputo casero en donde, con menos recursos económicos que los requeridos para comprar una PC, es posible resolver cierto tipo de problemas aplicando un modelo de paralelismo masivo sobre una arquitectura de procesadores con varios núcleos, memoria compartida y soporte multithreads Lloyd et al. (2008).

Existen alternativas para procesamiento en GPU, la más ampliamente utilizada es la tarjeta Nvidia NVIDIA (2006), para la cual se ha desarrollado un kit de programación en C, con un modelo de comunicación de datos y de control de threads proporcionado por un driver, el cual provee una interfaz GPU-CPU Joselli et al. (2008). Este ambiente de desarrollo llamado CUDA (el cual significa en inglés: Compute Unified Device Architecture) ha sido diseñado para simplificar el trabajo de sincronización de threads y la comunicación con la GPU Chen y Hang (2008); Luebke (2008), proponiendo un modelo de programación Buck (2007).

Este trabajo está organizado de la siguiente manera, en la siguiente sección se explican las características de la GPU y CUDA: arquitectura y modelo de programación. En la sección 3 se establece la importancia de los métricas de rendimiento. En la siguiente sección se detallan los parámetros a medir en este trabajo y cómo hacerlo. Finalmente se muestran los resultados experimentales obtenidos, las conclusiones y trabajos futuros.

2. GPU: CARACTERÍSTICAS

Un sistema de cómputo con GPU consta de dos componentes básicos, la CPU tradicional y una o más placas GPU, las cuales constituyen lo que se conoce como Streaming Processor Array. La conexión entre ambas componentes, CPU-GPU, se da a través de un bus PCI Express.

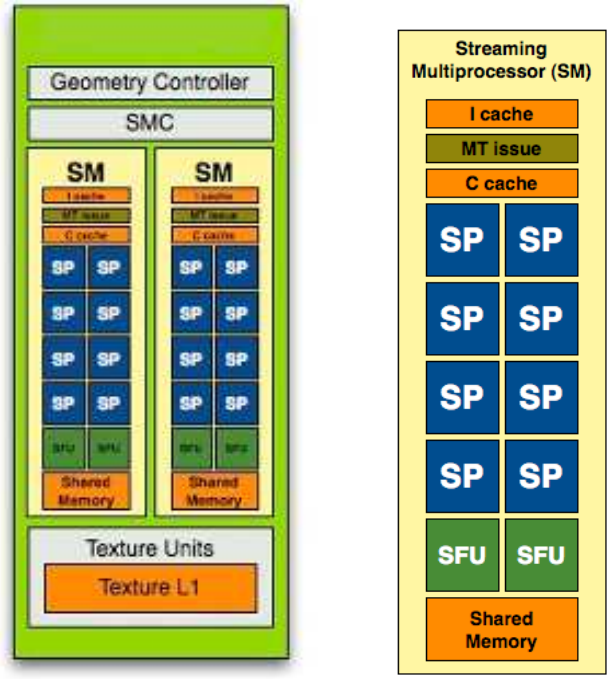
Una GPU puede ser considerada como un co-procesador de muchos núcleos soportando numerosos threads de gránulo fino NVIDIA (2006, 2008b); Ryoo et al. (2008). Se distingue de otras arquitecturas paralelas por la flexibilidad mostrada en la asignación de recursos locales (memoria o registro) a los threads. En forma general, una GPU consta de varios multiprocesadores de streams, cada uno con múltiples unidades de procesamiento, registros y memoria on-chip. Cada multiprocesador de streams puede ejecutar un número variable de threads, entre los cuales el programador decide la división de los recursos. Esta propiedad permite realizar ajustes para obtener mejoras en la performance.

En esta sección se detallan las características de la GPU, tanto las correspondientes al hardware como al modelo de programación propuesto. Si bien la mayoría de las GPU existentes en el mercado tienen características similares, en este trabajo nos enfocamos a la arquitectura y el modelo de programación propuesto por NVIDIA Luebke (2007).

2.1. Arquitectura

En la figura 1(a) se muestra la arquitecura típica de una GPU. Como se puede apreciar, la GPU está formada por un conjunto de unidades denominadas Texture/Processor Cluster (TPC), formados por un grupo de multiprocesadores, SM (Streaming Multiprocessor), unidades de textura, un controlador de geometría y un controlador de lógica, SMC, estas tres últimas heredadas de la tradicional utilización como aceleradora 3D.

Cada una de las placas GPU tiene su memoria RAM, también denominada memoria RAM global. El bus de conexión entre la GPU y su memoria RAM tiene, generalmente, un tamaño mucho mayor que el de la CPU, pudiendo oscilar entre 64 a 384 bits. Esta propiedad permite transferir gran cantidad de información por unidad de tiempo entre la GPU y su memoria.



(a) de GPU (b) de SM

Figura 1: Arquitectura

Cada SM está formando por un conjunto, generalmente 8, procesadores escalares, SP (Scalar Processor o Streaming processor), dos SFUs (Special Function Units), una pequeña cache de instrucciones, una unidad MT (MF issues), responsables de enviar instrucciones a todos los SP y SFUs en el grupo, una cache de sólo lectura de datos y una memoria shared o compartida, generalmente de 16KB. La figura 1(b) muestra la arquitectura de un SM.

Los SP están, a su vez, formados por dos ALUs y una FPU (Unidad de Punto Flotante), no tiene memoria cache de ningún tipo y tienen a cargo la resolución de las operaciones de enteros y flotantes, respectivamente, en la GPU. Además, las SFU tienen 4 unidades de multiplicación de flotantes, las cuales son usadas principalmente para operaciones como seno, coseno e interpolación.

2.2. Modelo de Programación

Como el sistema de computación con GPU consiste de un host, CPU tradicional, y uno o más dispositivos de cómputo, los cuales son los co-procesadores masivamente paralelos. Cada co-procesador aplica el modelo Simple Proceso-Múltiples Datos (SPMD), todas las unidades de cómputo (thread) ejecutan el mismo código, no necesariamente sincronizados, sobre distintos datos, se puede expresar al modelo como STMD (Simple thread-Múltiples Datos). Los threads comparten el mismo espacio de memoria global y el espacio de memoria compartida del multi-procesador al que pertenecen Buck (2007); Chen y Hang (2008); NVIDIA (2008a); Joselli et al. (2008); Luebke (2008); Ryoo et al. (2008).

Entre las distintas herramientas disponibles para programar una GPU, CUDA ha llegado a ser uno de los más populares, define un modelo de programación, el cual facilita el desarrollo de aplicaciones sobre la GPU. Se lo puede definir como un ambiente de programación, el cual permite a los desarrolladores de software crear componentes del programa aislados, las cuales resuelven un problema sobre un dispositivo dedicado, GPU, aplicando paralelismo de datos masivo.

Un programa CUDA es un programa en C/C++ extendido con palabras claves, las cuales especifican funciones paralelas de datos y estructuras de datos a ser ejecutadas en el dispositivo. A estas unidades de cómputo se las denomina *kernels*. Se considera a un programa CUDA formado por múltiples fases, las cuales son ejecutadas sobre la CPU o la GPU. Cuando la fase exhibe poco o nada de paralelismo se resuelve en el host (CPU). Si en cambio la fase muestra paralelismo de datos es implementada como kernel y se ejecuta sobre la GPU. La comunicación entre CPU-GPU, y viceversa, se da a través de transferencias de memoria del host a/desde la memoria global del dispositivo.

Existen algunas restricciones para las funciones *kernels*: no pueden ser recursivas, no admiten declaración de variables estáticas ni un número no-variable de argumento. Un kernel describe el trabajo de un simple thread a ser ejecutado por cientos de ellos. Los threads son organizados en una jerarquía de tres niveles: Grid (Nivel superior, está formado por bloques de threads, comparten el espacio de memoria global), Bloques (Nivel medio, integrado por los threads establecidos por el desarrollador del software) y Threads (Nivel inferior, pueden sincronizar sus tareas y compartir datos dentro del mismo bloque mediante el uso de memoria compartida). La determinación del número de Grids, la cantidad Bloques por Grid y de Threads por Bloque son parámetros que afectan directamente la performance de la aplicación y están sujetos a ajuste.

3. PARÁMETROS DE RENDIMIENTO GENERALES

Dos o más sistemas de computación pueden ser comparados a través de sus características técnicas, las cuales pueden medirse mediante una o más métricas. Entre las más comunes se encuentran: *Instrucciones por segundo, FLOPS, MIPS, Performance por watt, Bandwith de memoria*.

La estimación de cada uno de los parámetros puede ser llevada a cabo a través de herramientas de software. El software para el análisis de la performance se utiliza para determinar la potencialidad de una computadora, una red, un programa o un dispositivo. Las pruebas pueden ser cuantitativas: tiempo de respuesta, MIPS, entre otros Volkov (2008); o cualitativas: confiabilidad, escalabilidad, interoperabilidad, etc. La evaluación de la performance se realiza, la mayoría de las veces, respecto a pruebas de estrés. En la siguiente sección, se detallan los indicadores de desempeño considerados.

3.1. Parámetros de Rendimiento Analizados

En este trabajo se realiza una propuesta para evaluar los siguientes parámetros de la GPU:

- Capacidad de cómputo: Este parámetro determina la capacidad inherente de un procesador o conjunto de procesadores (más precisamente de sus ALUs) para realizar ya sea operaciones lógicas como aritméticas, independiente de su capacidad posterior de escribir su resultado en memoria. Particularmente, cuando la capacidad de cómputo se refiere a operaciones aritméticas con enteros se le denomina MIPS y cuando se refiere a operaciones aritméticas de punto flotante se llama FLOPS. En ciertas arquitecturas la capacidad de realizar operaciones aritméticas: con flotantes o con enteros, u operaciones lógicas difiere substancialmente. En el caso de las GPU, y a diferencia de los CPUs, la capacidad para procesar enteros (MIPS) como flotantes (FLOPS) es similar NVIDIA (2008a).

- Ancho de banda de Memoria: Permite determinar la cantidad de escrituras, lecturas o lecto-escrituras que una memoria particular es capaz de realizar en un tiempo determinado. Estrictamente hablando el ancho de banda hace referencia a la máxima capacidad del canal utilizado para la transferencia de datos del bus que comunica el procesador con la memoria. De acuerdo a la naturaleza de la memoria, la cantidad de lecturas y escrituras por unidad de tiempo pueden diferir substancialmente.

En ambos casos, los valores obtenidos en la práctica son, en la mayoría de las veces, menor al establecido por el medio físico. Esto se debe a las limitaciones y retardos impuestos por la tecnología del hardware usado.

4. DETERMINACIÓN DE LOS PARÁMETROS DE RENDIMIENTO

Para medir cada uno de los parámetros se desarrolló un benchmark, BEN_MyCP, con varias aplicaciones, las cuales permiten evaluarlos.

En las próximas secciones se explica las características de cada una de las aplicaciones y cómo realizar la estimación.

4.1. Capacidad de cálculo

Como se dijo antes, este parámetro determina la capacidad inherente de un procesador o conjunto de procesadores (más precisamente de sus ALUs) para realizar ya sea operaciones lógicas y/o aritméticas, independiente de su capacidad posterior de escribir su resultado en alguna de sus memorias de almacenamiento anexas al sistema. Obtener la capacidad de cómputo permite realizar comparaciones con otras arquitecturas.

Para medir la capacidad de cómputo de la GPU, el trabajo se enfocó particularmente en la capacidad aritmética para resolver operaciones en punto flotante: FLOPS (Pudiendo generalizar a operaciones con enteros, MIPS por las características enunciadas en la sección anterior).

Se desarrolló una aplicación, la cual realiza intensivas operaciones matemáticas de multiplicación y adición sucesivas sobre un registro en cada uno de los SPs de cada SM de la GPU. Para lograr resultados confiables, se evita el acceso a la memoria, tanto a la memoria principal del sistema como a la memoria global y compartida de la GPU, sólo se accede en el momento de guardar los resultados finales. Los resultados intermedios son almacenados provisoriamente en los registros internos de cada SP.

El problema es resuelto por múltiples threads, cada thread pertenece a un bloque del grid y es responsable de odtener el resultado de una operación. El tiempo total que implica el problema puede expresarse como:

$$T\_total = T\_op + T\_acceso\_MG \quad (1)$$

Donde T_op es el tiempo que demoran los threads, $D\_Grid \times D\_Block$ en realizar iterativamente n operaciones ($\frac{n}{2}$ operaciones mad) , esto significa que $T\_op = n \times \frac{D\_Grid \times D\_Block}{P}$ y $T\_acceso\_MG = m \times T\_W\_MG$, siendo P la potencia de cálculo y m el número de escrituras en memoria global de GPU (m coincide con la cantidad de threads, es decir $m = D\_Grid \times D\_Block$), reemplazando en la ecuación 1, se obtiene que:

$$T\_total = n \times \frac{D\_Grid \times D\_Block}{P} + D\_Grid \times D\_Block \times T\_W\_MG$$

Al ser P la potencia de cálculo, $\frac{1}{P}$ es un tiempo extremadamente pequeño y mucho menor que el tiempo de escritura individual en memoria global de GPU, $T\_W\_ind\_MG$, NVIDIA (2008a). Si la operación se realiza iterativamente n veces y n es lo suficientemente grande, el tiempo involucrado en la operación iterativa será mucho mayor que el tiempo requerido para almacenar un dato en la memoria global de GPU. Experimentalmente se determinó que un n de 4 órdenes de magnitud logra obtener con precisión la potencia de cálculo en una amplia gama de GPUs en la actualidad. Ante los constantes desarrollos tecnológicos tanto en la evolución de la velocidad de la memoria global del GPU y en la potencia de GPU, n podrá ser ajustado convenientemente. Además un n grande nos permitiría obtener resultados confiables, eliminando el error propio de la medición del tiempo. Por todo lo expuesto y con n lo suficientemente grande, se puede despreciar el tiempo de escritura en memoria global de la GPU, pudiendo expresar el tiempo total de la siguiente manera:

$$T\_total = n \times \frac{D\_Grid \times D\_Block}{P}$$

La potencia de cálculo (en FLOPS o MIPS) puede finalmente expresarse como:

$$P = \frac{D\_grid \times D\_block \times n}{T\_total}$$

Quedando formulada la potencia de cálculo en función del tiempo total implicado por la aplicación y las características de su resolución en la GPU.

4.2. Tasa de Transferencia de Memoria

Este parámetro, como se indicó antes, determina la cantidad de escrituras, lecturas o lecto-escrituras capaz de realizar una memoria particular en un tiempo determinado. El ancho de banda hace referencia también a la máxima capacidad del canal utilizado para la transferencia de datos del bus que comunica el procesador con la memoria. Generalmente, los valores obtenidos en la práctica son menores al establecido por el medio físico. Esto obedece a las limitaciones y retardos impuestos por la tecnología de hardware usado.

De acuerdo a la naturaleza de la memoria, la diferencia en unidades de tiempo entre la lectura y la escritura puede diferir substancialmente. En la mayor parte de las tecnologías usadas en la electrónica computacional, las lecturas son más rápidas que las escrituras.

En el modelo de CUDA, como se mencionó más arriba, los threads pueden acceder a distintos tipos de memoria durante la ejecución de un kernel. Cada thread accede a: una memoria local privada, a la memoria shared, la cual comparte con los demas threads del bloque, y a la memoria global de la GPU, accesible por todos los thread. En este trabajo medimos la tasa de transferencia de memoria shared y de memoria global, detallado en las siguientes secciones.

4.2.1. Memoria Global

Las mediciones realizadas para la memoria global tuvieron en cuenta las tres operaciones básicas de memoria: Lectura (R) , Escritura (W) y Lecto-Escritura(RW). A continuación se explica cada una.

Tasa de Lectura

Obtener la Tasa de Lectura o Ancho de Banda de Lectura, B_R, implica medir cuántas lecturas se pueden realizar en una unidad de tiempo. Desarrollar un algoritmo para medir la Tasa de Lectura implica en primer instancia lograr mediciones confiables, es decir evitar mediciones que incluyen mecanismos de optimización.

El compilador de CUDA/C posee algunos mecanismos de optimización, uno de ellos consiste en evitar las lecturas de memoria, si ésta no va a ser usada en ninguna operación o procesamiento posterior. El compilador ignora directamente la lectura real de la memoria, por lo que realizar mediciones con estas características implicaría obtener resultados no válidos. Para llevar a cabo una medición correcta del parámetro se debió desarrollar una aplicación que *obliga* a efectivizar dicha lectura. Por ello se diseñó e implemetó un algoritmo de búsqueda y cómputo de patrones en memoria Goyal et al. (2008) Nottingham y Irwin (2009). En su implementación, cada thread se encarga de analizar una posición de memoria, para lo cual es necesario realizar su lectura y, luego, la comparación con el valor de muestra, si el patrón cumple con la característica requerida, se contabiliza. Cada thread es responsable de una posición de memoria, según el tamaño de la memoria a analizar, los threads pueden pertenecer a distintos bloques.

En este caso, el tiempo total del proceso puede expresarse como:

$$T\_total = T\_R\_MG + T\_total\_acum\_Patrones$$

Los accesos a memoria en la aplicación realizada cumplen con la propiedad de accesos *coallesed* o alineados NVIDIA (2008a) para permitir el aprovechamiento máximo del bus de memoria, cada uno de los n thread lee una posición para verificar su coincidencia con el patrón a buscar, $n = D\_grid \times D\_Block$, y determinar la cantidad de patrones que satisfacen la condición. Establecer el total de patrones se realiza según la cantidad de bloques m utilizados en la solución del problema, $m = D\_grid$. El tiempo total de lectura de la memoria global puede expresarse en función del tiempo individual, $T\_R\_MG = n \times T\_R\_ind\_MG$. Reemplazando en la expresión anterior todo lo expuesto, ésta queda:

$$T\_total = D\_grid \times D\_Block \times T\_R\_ind\_MG + D\_grid \times T\_acum\_Patrones$$

Como $T\_acum\_Patrones$ mide una operación matemática por bloque y se cumple que la cantidad de bloques es mucho menor que la cantidad total de threads $m << n$, el último término puede despreciarse, por lo que:

$$T\_total = D\_grid \times D\_Block \times T\_R\_ind\_MG$$

Teniendo en cuenta esta definición y la de ancho de banda, se establece que:

$$T\_total = \frac{D\_grid \times D\_Block \times Udad\_min\_R}{B_R}$$

Al ser la unidad mínima de información un entero, o sea 4 bytes, el ancho de banda de lectura B_R en GBytes por segundo es:

$$B_R = \frac{4 \times D\_grid \times D\_Block}{T\_total \times 1024^3}$$

De esta manera se calcula el ancho de banda de lectura de la memoria global de GPU a partir de la medición del tiempo transcurrido en al ejecución de la aplicación.

Tasa de Escritura

Para la tasa de escritura su utilizó un algoritmo responsable de asignar un valor específico a un área extensa de la memoria global de GPU. La tarea de asignación de un valor a una posición de memoria global está a cargo de un thread, es por ello que existen tantos threads como posiciones de memoria a escribir.

En este caso, con una operación de asignación es suficiente, no es necesario realizar operaciones sobre los datos para obtener mediciones confiables, el tiempo de ejecución puede expresarse como:

$$T\_total = T\_total\_W\_MG$$

El tiempo total de almacenamiento en la memoria global de la GPU, $T\_total\_W\_MG$, puede ser expresador en función de las n escrituras y de su tiempo individual, $T\_total = n \times T\_W\_ind\_MG$. Esto es posible porque al igual que en el caso de la Tasa de Lectura, no existen optimizaciones en el acceso a memoria, los n threads acceden a la memoria para resolver el problema, o sea $n = D\_Grid \times D\_Block$. Considerando la definición de Ancho de Banda y haciendo los reemplazos pertinentes, se obtiene:

$$T\_total = \frac{D\_Grid \times D\_Block \times size(Udad\_min\_W)}{B_W}$$

Dado que al unidad mínima de información es un entero de 4 Bytes, el ancho de banda B_W en GBytes/s se expresa de al siguiente manera:

$$B_W = \frac{4 \times D\_Grid \times D\_Block}{T\_total \times 1024^3}$$

Dejando así expresado el ancho de banda de escritura para la memoria global de la GPU.

Tasa de Lecto-Escritura

Una operación de Lectura/Escritura implica un acceso a la memoria global por cada una de las operaciones descriptas en los dos puntos anteriores. Para medir la velocidad de lectura/escritura en memoria global de GPU se desarrolló un algoritmo encargado de leer una determinada posición de memoria, realizar una operación simple y, finalmente, escribir el resultado en la misma posición. Estas tres operaciones son realizadas por un thread sobre una posición de memoria, lo cual implica que se tendrán tantos threads como posiciones de memoria se deben leer y escribir. Los threads están organizados en bloques de un grid.

Como se mencionó antes, el compilador de CUDA/C optimiza el código, en este caso al realizar conjuntamente sucesivas lecturas, operaciones aritméticas y escrituras en memoria, las primeras no se efectivizan, sino que se realizaban directamente sobre los registros de los procesadores dando una velocidad ficticia, superior a la real. Para resolverlo se utilizaron bloques de memoria lo suficientemente extensos para evitar el uso de los registros de la GPU, obligando a realizar tanto las lecturas como las escrituras sobre la memoria global.

Para calcular el ancho de banda, B_{RW}, al igual que los casos anteriores, se toma como punto de partida el tiempo total involucrado en al operación:

$$T\_total = T\_total\_R\_MG + T\_total\_Op + T\_total\_W\_MG$$

Esta expresión puede ser reformulada considerando el tiempo de lectura/escritura en función de los tiempos individuales de lectura y de escritura y del tiempo que demora la operación a realizar, $T\_RW\_ind = T\_R\_ind\_MG + T\_Op + T\_W\_ind\_MG$, y las n posiciones de memorias sobre las cuales se realiza una lectura, una escritura y una operación matemáticas,

$$T\_total = n \times (T\_R\_ind\_MG + T\_Op + T\_W\_ind\_MG)$$

Como los tiempos de acceso a memoria, ya sea para lectura o escritura, son muy superiores a los tiempos de cómputo NVIDIA (2008a), éste puede despreciarse. Además se estableció que cada thread es responsable de una posición de memoria, por lo que $n = D\_Grid \times D\_Block$ y el tiempo de realizar la operación de Lecto-Escritura en función del ancho de banda y según el tamaño de las unidades a operar es $T\_RW = \frac{Udad\_min\_R + Udad\_min\_W}{B_{RW}}$. La expresión anterior queda:

$$T\_total = D\_Grid \times D\_Block \times \frac{(Udad\_min\_R + Udad\_min\_W)}{B_{RW}}$$

Al ser la unidad mínima de lectura y la unidad mínima de escritura iguales, números enteros de 4 bytes, el ancho de banda de Lecto-Escritura, B_{RW} en GBytes/s es:

$$B_{RW} = \frac{8 \times D\_Grid \times D\_Block \times Udad\_min\_inf}{T\_total \times 1024^3}$$

Estableciendo, a partir del tiempo total implicado en resolver una aplicación simple, el ancho de banda de Lecto-Escritura para la memoria global de GPU.

4.2.2. Memoria Shared

En el modelo de programación propuesto por CUDA los threads acceden a distintos tipos de memoria durante la ejecución de un kernel, pueden acceder a la memoria local privada, propia de cada thread; a la memoria shared, compartida por todos los threads de un bloque; y la memoria global, accedida por todos los threads en el sistema como se detalló en la sección anterior. La memoria shared es considerada una memoria interna al SM con tamaño limitado.

Para medir la tasa de transferencia de memoria shared se utilizó un algoritmo similar al utilizado para potencia de cálculo, la diferencia radica en que el resultado de la operación matemática iterativa utiliza un registro de dicha memoria. Al ser la memoria shared una memoria compartida por todos los SP del SM, para un correcto cálculo de la métrica, cada thread trabajó sobre una dirección diferente de la memoria shared, de manera tal de que cada SP utilize un registro de un banco de memoria shared diferente. Finalmente, luego de la operación iterativa el resultado de cada thread se guarda en la memoria global de GPU, escribiendo cada uno en la memoria global.

Para deducir el ancho de banda de la memoria shared, B_{MS}, al igual que para todos los casos anteriores expresamos el tiempo total de la aplicación desarrollada como:

$$T\_total = T\_total\_cal + T\_total\_W\_MS + T\_total\_W\_MG$$

Al tiempo total de cáculo, T_total_cal se lo puede expresar en función de la cantidad de operaciones a realizar, $\#op$, y de la potencia de cálculo analizada en la sección 4.1, $T\_total\_cal = \frac{\#op}{P}$. Además siendo n la cantidad de iteraciones realizadas sobre la memoria shared, m el número de operaciones de escritura en memoria global y el tiempo total de escritura en la memoria es $T\_total\_W\_MS = n \times D\_Grid \times D\_Block \times T\_W\_ind\_MS$, se puede expresar:

$$T\_total\_W\_MS - \frac{\#op}{P} = n \times D\_grid \times D\_block \times T\_W\_ind\_MS + m \times T\_W\_ind\_MG$$

Para evitar las optimizaciones del compilador CUDA/C y la eliminación de operaciones de lectura de los threads, luego de la operación matemática interativa el resultado de cada thread se guarda en la memoria global de GPU, por lo cual se realizan $(D\_grid \times D\_block)$ escrituras en memoria global, una por cada thread existente, $m = D\_grid \times D\_block$. Reemplanzando y operando, se obtiene:

$$T\_total\_W\_MS - \frac{\#op}{P} = D\_grid \times D\_block \times (n \times T\_W\_ind\_MS + T\_W\_ind\_MG)$$

Como el ancho de banda de la memoria shared es superior al de la memoria global de GPU NVIDIA (2008a), para determinar el valor de n se estableció que debía cumplir con dos condiciones: $(n \times T\_W\_ind\_MS) >> (T\_W\_ind\_MG)$ y $(n \times T\_W\_ind\_MS)$ debe ser lo suficientemente grande para que el error relativo de la medición de tiempo sea bajo. Se determinó experimentalmente que un n de 3 órdenes de magnitud cumple con las condiciones. Si n cumple con dichas propiedades, el tiempo de escritura en la memoria global puede ser ignorado, lo cual determina que:

$$T\_total\_W\_MS\frac{\#op}{P} = D\_grid \times D\_block \times (n \times T\_W\_ind\_MS)$$

Reemplazando $T\_W\_ind\_MS = \frac{\#Datos\_W\_x\_op}{B_{MS}}$, se obtiene:

$$T\_total\_W\_MS - \frac{\#op}{P} = D\_grid \times D\_block \times (n \times \frac{\#Datos\_W\_x\_op}{B_{MS}})$$

Al ser operaciones elementales de números enteros (o flotantes) de 4 bytes, $\#Datos = 4$, el ancho de banda de memoria shared, B_{MS} en GB/s queda expresado por:

$$B_{MS} = \frac{D\_grid \times D\_block \times n \times 4}{T\_total - (\frac{\#op}{P}) \times 1024^3}$$

Permitiendo, así a partir de esta expresión, establecer el ancho de banda de la memoria shared de la GPU.

5. RESULTADOS EXPERIMENTALES

En esta sección se analizan los valores para las distintas métricas propuestas. El análisis se realizó sobre tres gneraciones de GPU GeForce: la 8500 GT, la 9800GTX+ y la 260GTX, cada una de las cuales con las siguientes características:

Característica	8500 GT	9800 GTX+	260 GTX
Frecuencia de Shader	920 Mhz	1.836 Ghz	1.404 Ghz
Frecuencia de Memoria	400 Mhz	1100 Mhz	1188Mhz
Número de SM	2	16	27
Número de SP	16	128	216
Bus de Memoria	128 bit	256 bit	448 bit

Se evaluó la potencia de cálculo a través del algoritmo diseñado y explicado en la sección 4.1. Para el tamaño de datos operados (enteros o flotantes) de 32 bits y la alta frecuencia del procesador shader, se obtienen los siguientes valores de potencia de cálculo para un SM y para todos los SM de la GPU correspondiente.

	Todos los MPs		Por MP	
GPU	$P_{Teórica}$	$P_{Obtenida}$	$P_{Teórica}$	$P_{Obtenida}$
8500GT	7.42 GFLOPS/GMIPS	27.26 GFLOPS/GMIPS	13.71 GFLOPS/GMIPS	13.61 GFLOPS/GMIPS
9800GTX+	437.73 GFLOPS/GMIPS	436.25 GFLOPS/GMIPS	27.35 GFLOPS/GMIPS	27.21 GFLOPS/GMIPS
260 GTX	563.26 GFLOPS/GMIPS	562.3 GFLOPS/GMIPS	20.86 GFLOPS/GMIPS	20.66 GFLOPS/GMIPS

Se puede observar que en ambos casos los valores obtenidos están por debajo de la máxima capacidad de cálculo de cada GPU. El cálculo de la $P_{Teórica}$ se basó en lo propuesto por Guil y Ujaldón (2008). Además se comprobó, por un lado, si se utiliza un algoritmo sin interferencia de recursos, la potencia de cálculo es directamente proporcional a la cantidad de multiprocesadores en uso. Por el otro se pudo establecer que variando la frecuencia de shader, la potencia de cálculo es linealmente proporcional a dicha frecuencia.

Si se compara cualquiera de las GPU con una CPU ia32/amd64 compatible phenom2 y potencia de cálculo aproximada por core de 10 GFLOPS, la GPU ofrece mejores ventajas principalmente en tareas altamente paralelizables.

En la tabla siguiente se muestran los valores obtenidos de medir cada una de las siguientes tres operaciones en memoria: Lectura, Escritura y Lecto-Escritura.

GPU	$B_{teórico}$	B_R	B_W	B_{RW}
8500gt	12.8 GB/s	6.5 GB/s	4.54 GB/s	6.07 GB/s
9800gtx+	70.4 GB/s	59.3 GB/s	43.8 GB/s	56.5 GB/s
gtx260	133 GB/s	108.9 GB/s	69 GB/s	104.6 GB/s

Para todas las GPU consideradas, las mediciones obtenidas estuvieron por debajo del máximo ancho de banda calculado a partir del reloj de la memoria, el número de transferencias por ciclo y el tamaño del bus de memoria. Los valores de ancho de banda obtenidos para cada operación fueron, en todos los casos, inferiores al teórico. La diferencia es más notable cuando el número de SM es menor, esto se debe a que no se logra con los SM disponibles saturar la memoria con requerimientos. Además, las características de la administración de memoria global y los SM influye en los máximos valores obtenidos.

Para el caso de la memoria shared, se presentaron varios inconvenientes, uno de ellos es la imposibilidad de contar con valores teóricos confiables de ancho de banda máximo. El otro fue el comportamiento errático de las mediciones, las cuales no reflejaron una conducta determinística respecto de la operación a realizar y los datos involucrados. En estos momentos se están analizando alternativas para su medición, una de ellas es mediante PTX Kerr y Yalamanchili (2010) NVIDIA (2010).

6. CONCLUSIONES Y TRABAJOS FUTUROS

En este trabajo se presentaron diferentes aplicaciones simples, a través de las cuales se pueden estimar cuatro parámetros de performance de la GPU, ellos son la potencia de cálculo y el ancho de banda de la memoria global de lectura, escritura y lecto-escritura. Se realizó un análisis detallado de cómo medirlos, presentando una metodología simple para ser tenida en cuenta en otras aplicaciones. Además se mostraron los resultados obtenidos para cada parámetro sobre diferentes generaciones de GPU. Respecto a la memoria shared, si bien se hizo lo mismo, los resultados obtenidos no fueron satisfactorios, por lo cual es necesario realizar una revisión exhaustiva.

Este trabajo constituye el punto inicial para el análisis de los parámetros de performance de la GPU. Como líneas a seguir en futuros trabajos se encuentran la extensión del análisis a otros pramétros de performance y, principalmente, la comprobación de los resultados obtenidos a través del empleo de otras alternativas de programación de GPU, como es el caso de PTX.

REFERENCIAS

Buck I. Gpu computing with nvidia cuda. *ACM SIGGRAPH 2007 courses ACM*, 2007. New York, NY, USA.

Chen W. y Hang H. H.264/avc motion estimation implementation on compute unified device architecture (cuda). In IEEE, editor, *IEEE International Conference on Multimedia*. 2008.

Goyal N., Ormont J., Smith R., Sankaralingam K., y Estan C. Signature matching in network processing using simd-gpu architectures. In *University of Wisconsin*. 2008.

Guil N. y Ujaldón M. La gpu como arquitectura emergente para supercomputación. In *XIX Jornadas de Paralelismo de Castellon*. 2008.

Joselli M., Zamith M., Clua E., Montenegro A., Conci A., Leal-Toledo R., Valente L., Feijo B., Dórnellas M., y Pozzer C. Automatic dynamic task distribution between cpu and gpu for real-time systems. In *11th IEEE International Conference on Computational Science and Engineering*. 2008.

Kerr A.and Diamos G. y Yalamanchili S. Modeling gpu-cpu workloads and systems. In *3rd Workshop on GP Computation on Graphics Processing Units*. ACM, 2010.

Lieberman M., Sankaranarayanan J., y Samet H. A fast similarity join algorithm using graphics processing units. In *ICDE 2008. IEEE 24th International Conference on Data Engineering 2008*. 2008.

Lloyd D., Boyd C., y Govindaraju N. Fast computation of general fourier transforms on gpus. In *IEEE International Conference on Multimedia and Expo*. 2008.

Luebke D. Cuda: Scalable parallel programming for high-performance scientific computing. In *5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI 2008*. 2008.

Luebke D. H.G. How gpus work computer. *EEE Computer*, 40(2), 2007.

Nottingham A. y Irwin B. Gpu packet classification using opencl: a consideration of viable classification methods. In *Research Conf. of the South African Inst. of Comp. Sc. and Inf. Technologists*. ACM, 2009.

NVIDIA. Nvidia geforce 8800 gpu architecture overview. In *NVIDIA*. 2006.

NVIDIA. Nvidia cuda compute unified device architecture, programming guide version 2.0. In *NVIDIA*. 2008a.

NVIDIA. Nvidia geforce gtx 200 gpu architectural overview. In *NVIDIA*. 2008b.

NVIDIA. Compute ptx: Parallel thread execution isa. version 2.1. 2010.

Ryoo S., Rodrigues C., Baghsorkhi S., Stone S., Kirk D., y Hwu W. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *ACM*. ACM, 2008.

Volkov V. D.J.W. Benchmarking gpus to tune dense linear algebra. In *ACM/IEEE conference on Supercomputing*. IEEE Press, Piscataway, NJ, USA, 2008.