COMPARACION DEL TIEMPO DE EJECUCIÓN DE UN ALGORITMO SOBRE UNA UNIDAD DE PROCESAMIENTO GRÁFICO (GPU) Y SOBRE UNA UNIDAD DE PROCESAMIENTO CENTRAL (CPU)

Comparing the execution time of an Algorithm on a Graphical Processor Unit GPU and on a Central Processing Unit (CPU)

YENSY HELENA GOMEZ
Ingeniera de Sistemas y Computación
Estudiante IX Semestre
Universidad Tecnológica de Pereira
yensy@sirius.utp.edu.co

JOHN HAIBER OSORIO RIOS
Ingeniero de Sistemas y Computación
Docente Universidad Tecnológica de Pereira
john@sirius.utp.edu.co

ANDRES VELASQUEZ
Ingeniero de Sistemas y Computación
Docente Universidad Tecnológica de Pereira
andres@sirius.utp.edu.co

RESUMEN
El presente artículo, muestra el resultado de un proceso de investigación que se inicia como respuesta al creciente interés mundial por la investigación en el área de la computación grafica, más específicamente en el área de computación de alto desempeño y utilización de GPU (*Graphical Processor Unit*) para la aceleración de algoritmos específicos.

El sección 2 da una breve explicación del entorno utilizado y de las modificaciones necesarias para la ejecución de un algoritmo sobre una GPU. El sección 3 explica el código necesario para paralelizar una suma de vectores en *OpenCL*, y el sección 4 evidencia los resultados obtenidos durante la realización de la investigación.

PALABRAS CLAVES: GPU, Computación de Alto Desempeño, OpenCL.

ABSTRACT
This article is the result of a research process in graphical computing, about high performance computing and GPU (Graphical Processor Unit) uses for acceleration of mathematical algorithms.

Chapter 2 is about the environment used and the modifications realized for the algorithm execution on a GPU. Chapter 3 explains the algorithm and the OpenCL code, and the chapter 4 shows the results during the research.

KEYWORDS: *GPU, High Performance Computing, OpenCL*

1. INTRODUCCIÓN

En la actualidad está naciendo un *framework* orientado a la solución de algoritmos utilizando el nivel de paralelismo presente en los procesadores de propósito general y en la adaptación de elementos de procesamiento heterogéneos (DSP, FPGA, GPU) sobre las plataformas de computación convencionales (1). Esto permite que tareas que sean demasiado rigurosas y complejas computacionalmente puedan ser ejecutadas de manera eficiente y rápida sobre algún elemento de procesamiento específico. Tal como OpenCL el cual permite de una manera sencilla tener una plataforma de computo heterogéneo.

Las tarjetas de video poseen una capacidad de cómputo fascinante (330 GFlops y un ancho de banda de memoria de 80 GB/s) (2) debido a que estas han sido diseñadas específicamente para realizar cálculos matemáticos complejos para la renderización en vivo de juegos de video sobre computador, esta capacidad de computo en la actualidad se puede aprovechar gracias a las tecnologías emergentes como OpenCL, CUDA y ATI Stream.

Estas tecnologías permiten utilizar la capacidad de computo de las GPU, convirtiéndolas en elementos de propósito general de ahí que esta nueva tendencia se conozca actualmente como GPGPU (*General Purpose Graphical Processor Unit*) las cuales permiten a través de una interface de software la utilización de las tarjetas graficas para tareas de aceleración.

2. ENTORNO DE PROGRAMACION Y HARDWARE

El algoritmo utilizado para verificar el desempeño de la GPU fue el de suma de vectores de gran tamaño (100 000 a 1000 000 de datos). En cuanto al *software* utilizado se uso como plataforma de desarrollo Visual Studio .NET el cual provee una librería para realizar la conexión directa con el lenguaje de programación OpenCL que permite realizar cálculos computacionales sobre elementos de procesamiento heterogéneos, previamente a la correcta

Fecha de Recepción: 9 de Septiembre de 2010
Fecha de Aceptación: 28 de Abril de 2011

ejecución del *software* se hace necesaria la instalación de los controladores de la tarjeta grafica que se utilizará y del entorno de conexión ya sea CUDA o ATI *Stream*, los cuales soportan código desarrollado en OpenCL.

El *hardware* utilizado fue un computador portátil Dell Inspiron con procesador Intel Core i5 de 2.27 GHz, 4 GB de RAM, y una tarjeta aceleradora ATI Mobility Radeon 4330 con 512 MB de memoria dedicada (3) (4).

Cabe anotar que la plataforma utilizada no se encuentra en el estado de arte de plataformas de cómputo heterogéneo, se ha acelerado el algoritmo utilizando un equipo portátil común, lo que hace las cosas diferentes es el aprovechamiento de la GPU presente en la aceleradora ATI, en la cual se realiza el procesamiento de los datos.

El lenguaje de programación utilizado es C#, y se ha aprovechado una librería creada por la empresa CMSOFT llamada *OpenCLTemplate* (5) que permite realizar las tareas de lectura y escritura a la memoria de la GPU de manera simple.

3. CREACIÓN Y EJECUCIÓN DE CÓDIGO OPENCL

Cuando se trabaja con aceleración sobre GPU utilizando el sistema operativo *Windows* se debe tener en cuenta que:

- *Windows* no permite que la GPU se quede por más de dos segundos realizando tareas sin enviar datos a la pantalla.
- El sistema operativo no permite que se utilice toda la memoria de la tarjeta grafica, típicamente reduce este valor a la mitad.

Para solucionar estos dos inconvenientes se deben realizar los siguientes pasos:

- *Click Start* y ejecute regedit.exe (escriba "regedit" como un comando para abrir el editor de registro de *windows*).
- Vaya a HKEY_LOCAL_MACHINE\SYSTEM\CURRENTCONTROLSET\CONTROL\GraphicsDrivers.
- Cree dos REG_DWORD: TdrDelay y TdrDdiDelay.
- Coloque TdrDelay en 128 y TdrDdiDelay a 256.
- Dirijase a la entrada HKEY_LOCAL_MACHINE\SYSTEM\CURRENTCONTROLSET\CONTROL\SESSION MANAGER\ENVIRONMENT y cree una variable de tipo REG_SZ que se llame GPU_MAX_HEAP_SIZE. Ubicar este valor igual que el tamaño de la memoria de la tarjeta grafica.
- Posteriormente cierre el editor de registro y reinicie el sistema.

Después de realizar los pasos mencionados, el equipo de cómputo queda preparado para ejecutar código en OpenCL.

El *framework* OpenCL (*Open Computer Language*), se basa en funciones *kernel*, las cuales se ejecutan de manera paralela sobre las unidades de cómputo que posea la GPU, a continuación se ve la función kernel que permite la suma de dos vectores:

```
__kernel void
        float SumaVectores(__global    float * v1,
                __global    float * v2)
        {
           int i = get_global_id(0);
           v1[i] = v1[i] + v2[i];
        }
```

El código en OpenCL debe ser creado, compilado y ejecutado desde el entorno de desarrollo de .NET. Básicamente la tarea de ejecución de un *kernel* se basa en los siguientes pasos:

- Creación de la función Kernel. Se debe crear un *string* de caracteres que contenga la función como tal, en este caso se toma el código mencionado con anterioridad y se asigna a una variable de tipo *String*.
- Inicializar entorno OpenCL, en este caso lo que se debe hacer es configurar la librería para que utilice la GPU como unidad de procesamiento.
- Se compila el código fuente contenido en la variable utilizada.
- Se busca dentro del código de la función *kernel*, la función específica a ejecutar, esto debido a que el *kernel* puede contener muchas funciones.
- Crear los vectores en memoria principal y pasarlos a la memoria del dispositivo.
- Se elige la cantidad de hilos o *workers* que realizaron los cálculos.
- Se ejecuta la función *kernel*.
- Se escribe el resultado de vuelta a la memoria principal.

Los pasos anteriores son genéricos para cualquier tipo de función a paralelizar.

El código mostrado permite realizar la suma de dos vectores y ejecutarla sobre los procesadores presentes en la GPU, cada suma es realizada en un elemento de procesamiento, el cual después de quedar libre pasa de nuevo a esperar más datos para procesar. En la Figura 1

se ve un diagrama de la arquitectura típica de una Unidad de Procesamiento Grafica (6).

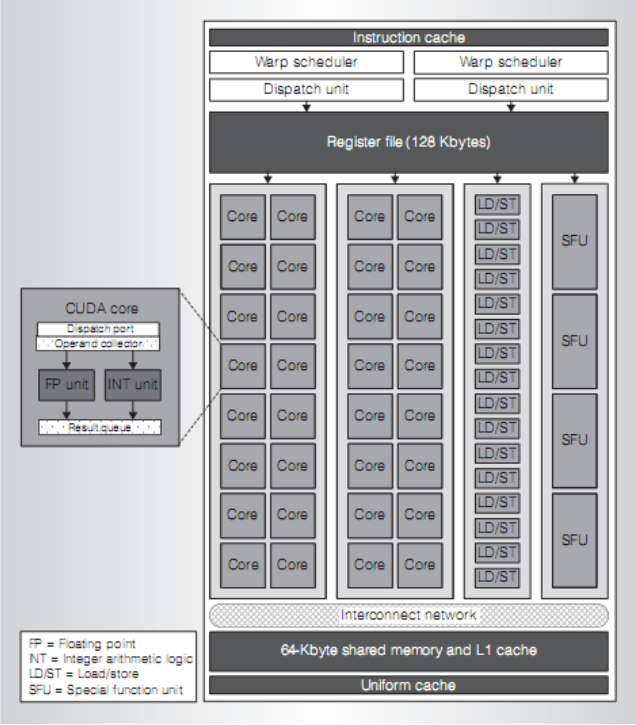

Figura 1. Arquitectura multiprocesador Fermi, con 32 procesadores CUDA.

Es importante recalcar que la función *get_global_id(0)*, retorna el hilo o *worker* que va a realizar el procesamiento del dato, en el proceso de ejecución de la función *kernel* desde C# se definen el número de trabajadores, para este ejemplo en particular el valor es igual al número de datos contenido en los vectores (7).

4. RESULTADOS OBTENIDOS

Las siguientes graficas muestran los tiempos de ejecución en microsegundos obtenidos al realizar 10 ejecuciones del mismo algoritmo.

La **Figura 2** evidencia un hecho que es ampliamente conocido y es que cuando se realizan cambios a la memoria de la GPU el tiempo de ejecución del algoritmo sobre el elemento de procesamiento se ve afectado, en este caso el *speed-up* entregado por la GPU es de 0,94x es decir su utilización en lugar de acelerar el procesamiento lo desacelera. Esto es debido al tiempo que tarda en llevar los datos desde la memoria principal hasta la memoria del dispositivo.

La **Figura 3** muestra el tiempo de ejecución del algoritmo sobre la GPU sin realizar lectura de memoria, en este caso el algoritmo se ejecuta y no se tiene en cuenta el tiempo que toma cambiar de contexto desde la memoria principal a la memoria del elemento de procesamiento, se logran un *speed-up* promedio de 133,18x, lo que deja aun más claro lo planteado en el párrafo anterior.

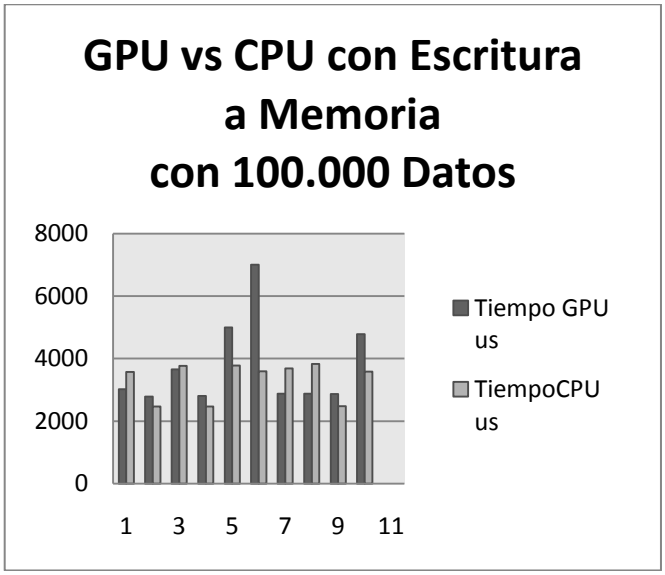

Figura 2. Comparativa tiempo ejecución GPU y CPU.

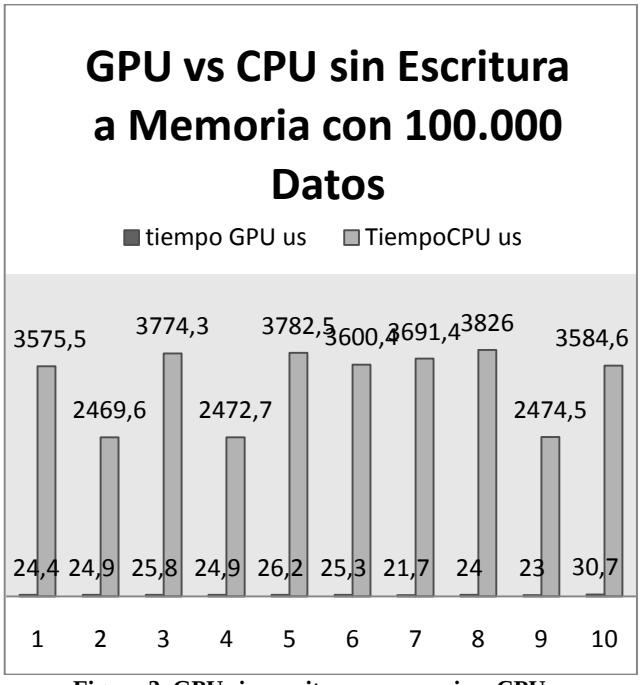

Figura 3. GPU sin escritura a memoria y CPU

La **Figura 4,** deja claro que existen puntos dentro de los algoritmos en los cuales vale la pena pagar el costo computacional del cambio de contexto, es claro que cuando la cantidad de datos a procesar y el peso matemático del algoritmo es suficiente, la aceleración o *speed-up* se nota aun cuando se tienen en cuenta los tiempos de lectura y escritura a la memoria de la GPU, en

este caso puntual cuando la cantidad de datos es de 1000 000 se logra una aceleración media de 1,205X.

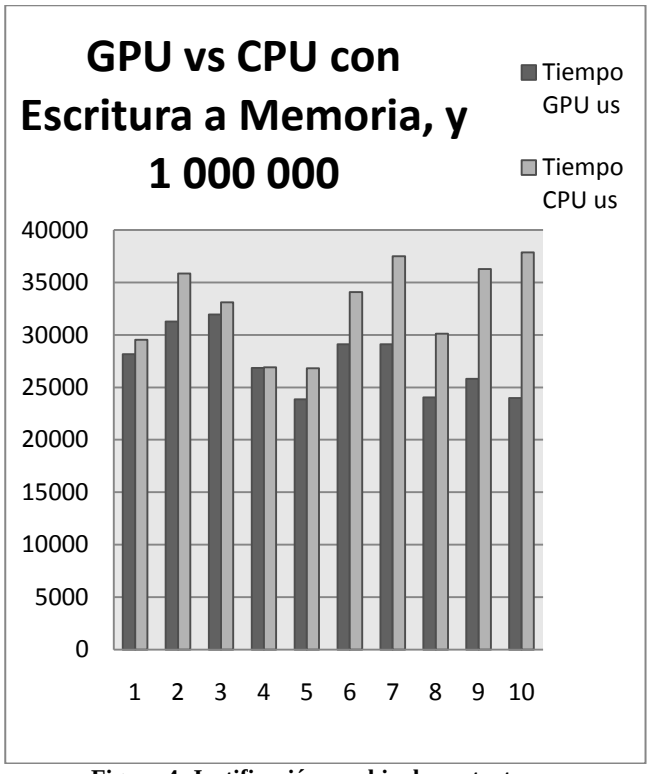

Figura 4. Justificación cambio de contexto.

5. CONCLUSIONES

Se logro la apropiación de la tecnología, que será utilizada en futuras implementaciones.

Se detectó para el algoritmo planteado que el uso de la GPU es útil siempre y cuando el tiempo gastado para realizar el proceso de escritura y lectura a la GPU no exceda al tiempo de ejecución sobre la CPU, el uso del elemento de procesamiento cuando se tiene en cuenta el tiempo de lectura de la GPU tiene sentido cuando se sobrepasan los 500000 datos.

También se obtuvo que el *speed-up* del algoritmo comparado con la CPU cuando no se tiene en cuenta el cambio de contexto sea del orden de 133x, lo que evidencia en gran medida el desempeño de los elementos de este tipo.

Para terminar, la actual implementación es apenas el punto de partida para la evaluación y comparación de aceleración de muchos algoritmos matemáticos que se irán implementando sobre sistemas de cómputo heterogéneos.

6. BIBLIOGRAFIA

[1]. **Nvidia.** Nvidia GPU Computing. *Nvidia GPU Computing.* [En línea] 2010. http://www.nvidia.es/page/gpu_computing.html.

[2]. *GPU Computing.* **D. Owens, John, y otros.** Mayo de 2008, Proceedings of the IEEE.

[3]. **ATI.** ATI Mobility Radeon™ HD 4300 Series Specs . *ATI Mobility Radeon™ HD 4300 Series Specs* . [En línea] 2010. http://www.amd.com/us/products/notebook/graphics/ati-mobility-hd-4000/hd-4300/Pages/hd-4300-specs.aspx.

[4]. Notebook Check. [En línea] http://www.notebookcheck.net/AMD-ATI-Mobility-Radeon-HD-4330.13973.0.html.

[5]. **Cmsoft.** CMSOFT. [En línea] http://www.cmsoft.com.br.

[6]. *The GPU Computing Era.* **Nickolls, John y J. Dally, William.** s.l. : IEEE Computer Society. 0272-1732.

[7]. **Group, Khronos.** Khronos OpenCL API Registry. [En línea] http://www.khronos.org/registry/cl/.

[8]. **CMSOFT.** Configuring OpenCL. [En línea] 2010. http://www.cmsoft.com.br/index.php?option=com_content&view=category&layout=blog&id=58&Itemid=95.