Paralelismo - 23GIIN

Actividad 1 - Portafolio

Gagliardo Miguel Angel

31 de Octubre de 2023

Al hablar de CUDA[1], primero debemos **definir** la computación sobre GPUs (Graphics Processing Unit o - en español - Unidades de Procesamiento Gráfico) como el uso de una tarjeta gráfica para realizar cálculos científicos de propósito general.

El modelo de computación sobre GPUs consiste en usar conjuntamente una CPU (Central Processing Unit) y una GPU, de manera que formen un modelo de computación heterogéneo. Siguiendo este modelo, una parte **secuencial** del procesamiento se ejecutaría sobre la CPU y la parte **más costosa o pesada** del cálculo sobre la **GPU**. Desde el punto de vista de un usuario, la app simplemente se ejecutará más rápido, porque se está utilizando la potencia y altas prestaciones de una GPU para aumentar el rendimiento.

Dada esta descripción nos encontramos con un problema inicial, y es que antiguamente para el uso de GPUs para cálculo se necesitaba utilizar lenguajes de programación específicos para gráficos como ser OpenGL o CG para programar la misma, para esto los desarrolladores debían especializarse y esto limitaba claramente el acceso por parte de la ciencia al uso del rendimiento y potencial de las GPUs.

NVIDIA fue consciente de fue consciente del potencial que suponía acercar este enorme rendimiento a la comunidad científica en general y decidió investigar la forma

de modificar la arquitectura de sus GPUs para que fueran completamente programables para aplicaciones científicas, además de añadir soporte para **lenguajes de alto nivel como C y C++**. De esta manera en 2009, NVIDIA introduce **CUDA** (Compute Unified Device Architecture) a sus GPUs.

CUDA es una nueva arquitectura para cálculo paralelo de propósito general, con un nuevo repertorio de instrucciones y un nuevo modelo de programación paralela, con soporte para lenguajes de alto nivel (además de C y C++, se agregan Java, Fortran, Python y varios más[2]) y constituidas por cientos de núcleos que pueden procesar de manera concurrente miles de *threads* de ejecución.

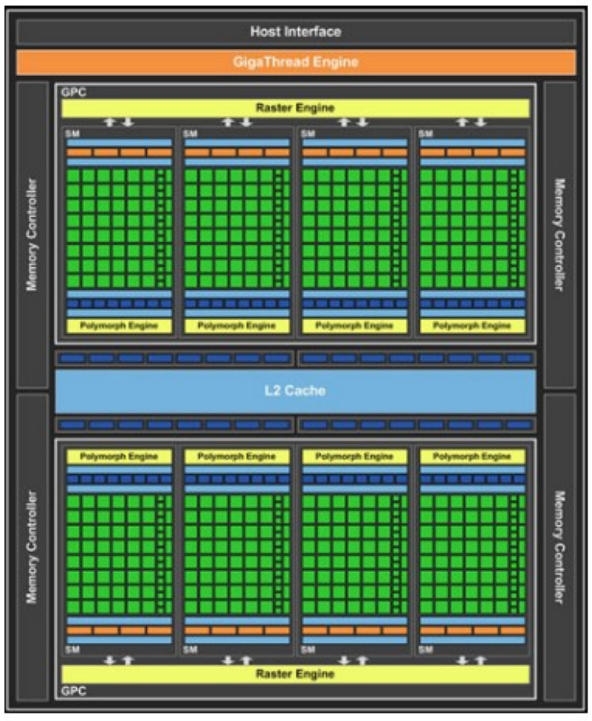
En esta arquitectura, cada núcleo tiene ciertos **recursos compartidos**, incluyendo registros y memoria. La memoria compartida, integrada en el propio chip de la GPU, permite que las tareas que se están ejecutando en los cores compartan datos sin tener que enviarlos a través del bus de memoria del sistema.

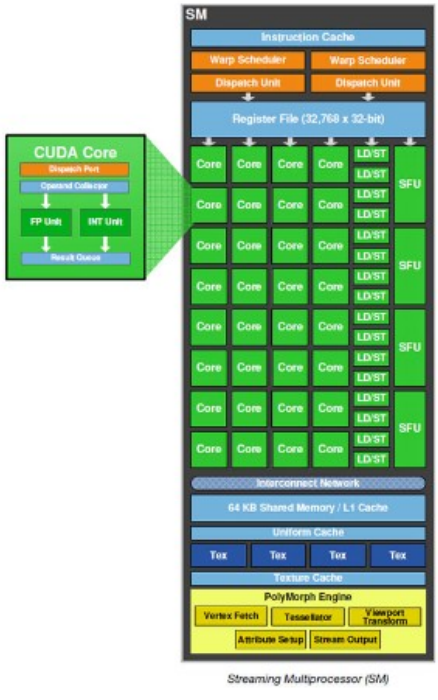
En la arquitectura clásica de una tarjeta gráfica podemos encontrar la presencia de dos tipos de procesadores, los *procesadores de vértices o Vertex Processor* y los *procesadores de fragmentos o Pixel Processor*, dedicados a tareas distintas e

independientes, y con repertorios de instrucciones diferentes. Esto presenta dos problemas importantes:

- Por un lado un desbalance de la carga entre ambos procesadores, al ser cada tarea unívoca.
- Por otro, la diferencia entre sus respectivos repertorios de instrucciones.

Y así es como NVIDIA, en búsqueda de una arquitectura unificada donde no se distingue entre ambos tipos de procesadores llega a crear **CUDA**, donde todos los núcleos de ejecución necesitan el mismo *repertorio de instrucciones* y prácticamente *los mismos recursos*.





Streaming Multiprocessor (SM)

En la imagen, se puede observar la presencia de unas unidades de ejecución denominadas *Streaming Multiprocessors* (SM), siendo 8 unidades en el ejemplo que están interconectadas por una zona de *memoria común*. Cada SM está compuesto a su vez por núcleos de cómputo llamados “núcleos CUDA” o *Streaming Processors* (SP), que son encargados de ejecutar las instrucciones, en el ejemplo vemos que hay *32 núcleos por cada SM*, lo que hace un total de **256 núcleos de procesamiento.** Este diseño de hardware junto con las SDKs permite la programación sencilla de los núcleos de la GPU utilizando un lenguaje de alto nivel para CUDA. El programador sólo necesita escribir un programa secuencial dentro del cual se llama a lo que se conoce como **kernel**, que puede ser una simple función o un programa completo. Este kernel se ejecuta de forma paralela dentro de la GPU como un conjunto de *threads* y que el programador puede organizar dentro de una jerarquía en la que pueden agruparse en bloques (blocks), y que a su vez se pueden distribuir formando una malla (grid).

Entonces, un *thread block* es un conjunto de hilos concurrentes que pueden cooperar entre ellos a través de mecanismos de sincronización y **compartir accesos a un espacio de memoria exclusiva de cada bloq**ue. Y una **malla** es un conjunto de bloques que pueden ser ejecutados independientemente y que por lo tanto pueden ser lanzados en paralelo en los SMs. Cuando se invoca un kernel, el programador especifica el número de *threads per block* y el número de bloques que conforman la

malla. Una vez en la GPU, a cada hilo se le asigna un único número de identificación dentro de su bloque, y cada bloque recibe un identificador dentro de la malla. Esto permite que cada hilo decida sobre qué datos tiene que trabajar, lo que simplifica enormemente el direccionamiento de memoria cuando se trabaja con datos multidimensionales, como ser procesamiento de imágenes o resolución de ecuaciones diferenciales de alta complejidad.

Como dato de color, en el año 2018 **por primera vez en la historia**[3] la mayoria (nada menos que un 56%) de los *flops* (Floating Point Operations per Second) provinieron de GPUs en lugar de CPUs a causa de la introduccion de las GPUs NVIDIA de la empresa **Tesla** que utilizar para sus supercomputadores. Otros hitos de esta tecnología de NVIDIA es formar parte de **dos tercios del TOP500** desde Junio 2021[4,] a la vez que **en conjunto con la arquitectura ARM** provee hasta 2.8x mejor eficiencia energética que la competencia[5].

Finalmente y para terminar en cuanto a mi **experiencia personal**, a través de la VIU he realizado el curso y certificación de Fundamentals of Accelerated Computing with CUDA C/C++[6] que me ha dado una visión general de lo fantástica que es esta tecnología. El curso está orientado a gente sin experiencia en la materia (aunque se requiere cierto conocimiento de C/C++) y a través de sencillos ejercicios se va

visualizando cómo, utilizando la SDK de CUDA para C++, a partir de un programa sencillo y midiendo en cada paso los tiempos de procesamiento cuánto tarda en correr *end to end* un programa a medida que se va haciendo más *offload* de la aplicación en la GPU simulada en el laboratorio que NVIDIA provee, una experiencia muy recomendable.

REFERENCIAS

1. NVIDIA: What is CUDA - https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/
2. NVIDIA: About CUDA - https://developer.nvidia.com/about-cuda
3. Top500: New GPU-Accelerated Supercomputers Change the Balance of Power on the TOP500 **-** https://www.top500.org/news/new-gpu-accelerated-supercomputers-change-the-balance-of-power-on-the-top500/
4. NVIDIA: Accelerating the TOP500 Supercomputers - https://network.nvidia.com/files/doc-2021/TOP500-JUNE-2021_0.pdf
5. NVIDIA: Green Light! TOP500 Speeds Up, Saves Energy with NVIDIA - https://blogs.nvidia.com/blog/2020/06/22/top500-isc-supercomputing/
6. NVIDIA: Fundamentals of Accelerated Computing with CUDA C/C++ - https://courses.nvidia.com/courses/course-v1:DLI+C-AC-01+V1/