CUDA y PyCUDA

Alejandro Hernández Rodríguez, David Guzmán, Mario Garrido

24 de noviembre de 2019

1. CUDA

1.1. ¿Por qué un GPU?

La principal problemática con un CPU hoy en día, y la razón por la que tenemos que pensar en paralelo es que estamos llegando a un límite de velocidad de las arquitecturas actuales. Esto es debido a que para que tenga una mayor velocidad un procesador, hay que utilizar más potencia en este. Después de cierto punto, se vuelve ineficiente o imposible mantener el procesador lo suficientemente frío como para operar correctamente.

Por otro lado, visualicemos cuantos hilos se pueden ejecutar en paralelo en un CPU.

Ejemplo:

- Procesador Intel Skylake (2017).
- Velocidad de 3.5 GHz base.
- Tiene ocho núcleos físicos.
- Puede realizar operaciones de vectores de 512 bits, es decir, 16 operaciones simultáneas con números de 32 bits. (AVX)
- Hyperthreading, básicamente divide el núcleo físico y lo hace ver como dos núcleos lógicos, lo que deja realizar aproximadamente dos hilos al mismo tiempo.
- Esto nos lleva a un máximo teórico de $2 \cdot 16 \cdot 8 = 256$ hilos paralelos.

Por otro lado:

- Nvidia GTX 980 (2014).
- Velocidad de 1.127 GHz base.
- Posee 2048 unidades de cómputo para sombreado.
- Esto nos lleva a 2048 hilos en paralelo sólo de unidades de sombreado.

Como podemos notar, incluso varias generaciones atrás, las GPUs podían realizar una mayor cantidad de operaciones concurrentes que una CPU normal, y no están atadas tan fuertemente por las limitaciones de un CPU (correlación directa entre velocidad y nivel de paralelismo, así como velocidad y calor). Hoy en día, esta diferencia se ha incrementado enormemente.

1.2. Paradigmas de la programación de GPU

Aunque es claro que se pueden realizar muchas más operaciones concurrentes en un GPU que en un CPU, también es lógico que estas operaciones no pueden resultar tan intensivas como las que le delegaríamos a un CPU. Esto es debido a que los GPU buscan tener más procesadores más pequeños y simples. Asimismo, existen otras limitantes que generan los siguientes tres fundamentos para la programación de un GPU:

- Como hay muchas unidades de cómputo, se cambia una facilidad de control por más cómputo.

- Se debe seguir un modelo de programación paralelo.
- Se optimiza para volumen de cómputo, no latencia.

Cabe mencionar que el modelo de computadora que se utiliza con una combinación de CPU y GPU es denominado heterogéneo. En este modelo, existen dos roles, el de anfitrión y el de dispositivo.

Ambos tienen su propia memoria y capacidades, por lo que el sincronizar y delegar sus trabajos correctamente es vital para poder sacarles provecho.

1.3. ¿Qué es CUDA?

CUDA (Compute Unified Device Architecture) es una plataforma de computo paralelo desarrollada por Nvidia para sus GPUs. Esta permite utilizar este hardware para secciones de cómputo altamente paralelizables.

El origen de CUDA se remonta al 2006, cuando se integran conceptos desarrollados previamente para extender C y darle capacidades de paralelismo.

1.4. ¿Por qué CUDA?

Por el dominio de mercado de Nvidia, la plataforma CUDA es la más utilizada en aplicaciones diversas como Deep Learning.

Aunque existen competidores, no tienen la misma cantidad de soporte, ni el mismo desempeño en GPUs de Nvidia.

La implementación de CUDA resulta ventajosa debido a que simplifica la programación en un GPU. Esto lo logra debido a dos factores:

- La facilidad de compartir memoria entre el anfitrión y el dispositivo. Esto debido a que CUDA provee de operaciones de traslado de memoria sencillo entre ambos dispositivos.
- El uso de las funciones kernel. Un kernel es la sección paralela del código, y CUDA tiene la facilidad de que se implementa como cualquier otra función. Sin embargo, debe de realizarse una asignación cuidadosa del dominio de trabajo de cada kernel individual ya que por defecto un GPU posee una arquitectura SIMD.

1.5. Desventajas de CUDA

- Limitado únicamente a equipos Nvidia.
- Existe un cuello de botella generado por el bus de datos entre el GPU y el CPU.
- No es tan popular.

1.6. Ventajas de CUDA

- Un GPU tiene un nivel de paralelización varias magnitudes más alto que un CPU, lo que deriva en una gran capacidad de resolver problemas paralelizables, como por ejemplo, las operaciones tensoriales. En CUDA, la implementación para resolver este tipo de problemas es muy sencilla.
- Tiene soporte para escalabilidad. Es decir, puedes tener varias GPUs trabajando en paralelo sin muchos problemas.

2. PyCUDA

Como ya hemos visto, las GPU Nvidia utilizan CUDA para realizar cálculos en paralelo. Sin embargo, la implementación original de CUDA es en lenguaje C. Si queremos usar CUDA en otros lenguajes, como por ejemplo Python, debemos acercarnos a las implementaciones de CUDA para este mismo. Aquí entra PyCUDA, que es un wrapper para CUDA en Python.

PyCUDA nos da toda la funcionalidad de CUDA implementada en forma de funciones, y nos da una opción de ejecutar el código C de CUDA en el mismo, por lo que resulta muy versátil para la programación con CUDA y Python.

2.1. Ventajas de PyCUDA

- Conveniencia. Abstracciones como pycuda.compiler.SourceModule y pycuda.gpuarray.GPUArray hacen que la programación en CUDA sea aún más conveniente que con el tiempo de ejecución basado en C de Nvidia.
- Comprobación automática de errores. Todos los errores de CUDA se traducen automáticamente en excepciones de Python.
- Velocidad. La capa base de PyCUDA está escrita en C++, por lo que a pesar de que hay un sacrificio en el rendimiento a cambio de la comodidad de Python éste no es dramático.

2.2. Desventajas de PyCUDA

- Sólo un número reducido de librerías de Python se pueden usar con PyCUDA.
- No hay tanta documentación ni comunidad como en otras librerías de Python.

3. Settings en Colab

Google Colab nos permite usar las GPU que tienen disponibles, además, ya viene con CUDA preinstalada. Para usar las GPU de Colab abrir un nueva notebook de Python 3 en Google Colab y ejecutar la siguiente línea

```
!pip install pycuda
```

Ésto instalará PyCUDA, una vez se termine de instalar ir a Entorno de ejecución → Cambiar tipo de entorno de ejecución → Acelerador de Hardware → GPU.

Referencias

1. Para checar las funciones de parte de scikit-cuda ver scikit-cuda.
2. Para ver los avances de Nvidia con CUDA checar Nvidia CUDA Zone
3. Algunas aplicaciones en las que se usa CUDA CUDA Technology
4. Documentación de CUDA CUDA Toolkit Documentation
5. Documentación de PyCUDA PyCUDA Documentation
6. Muy breve introducción a PyCUDA PyCUDA Intro