Taller 4: CUDA
Daniel Salas Calderón

1) ¿Qué es CUDA?

Es una arquitectura de NVIDIA para realizar cálculos paralelamente, aprovechando la unidad de procesamiento gráfica para este fin. Se hace una división de los cálculos de computo mediante vertex y pixel shaders. Además puede ser utilizada en operaciones de propósito general y no necesariamente gráfico.

2) ¿Qué es un kernel en CUDA y cómo se define?

Un kernel representa una ejecución independiente, que divide su operación sobre un número definido N de hilos distintos, para lo cual divide la ejecución espacial y temporalmente aprovechando la arquitectura de los GPU's como sistema multiprocesador, además cada bloque multiprocesador, puede ejecutar una cierta cantidad de hilos y todo lo que eso implica, como compartir memoria.

3) ¿Cómo se maneja el trabajo a procesar en CUDA? ¿Cómo se asignan los hilos y bloques?

Es necesario definir kernels en el hilo principal, así como la cantidad de bloques e hilos por bloques con que se desea trabajar:

kernel<<<No bloques, No hilo por bloque>>>(parametros)

De esta manera el kernel divide el trabajo entre cada uno de los bloques y para cada bloque es asignado un conjunto de hilos, los cuales se pueden compartir memoria entre ellos siempre y cuando estén dentro del mismo bloque.

4) Investigue sobre la plataforma Jetson TX2 ¿cómo está compuesta la arquitectura de la plataforma a nivel de hardware?

Jetson TX2 es un superordenador de inteligencia artificial que funciona para la arquitectura NVIDIA pascal, la cual destaca por su bajo consumo energético. Jetson TX2 tiene las siguientes especificaciónes físicas:

- ➔ GPU: 256 núcleos con la arquitectura NVIDIA Pascal™.
- ➔ CPU: NVIDIA Denver 2 doble de 64 bits, Quad ARM A57
- ➔ Video: Codificación y decodificación a 60 fps de 4K × 2K

Taller 4: CUDA
Daniel Salas Calderón

- ➔ Cámara: 12 canales de CSI que permiten hasta 6 cámaras; 2,5 gigabytes/segundo/canal
- ➔ Memoria: LPDDR4 de 8 GB; 58,3 gigabytes/segundo
- ➔ Almacenamiento: eMMC de 32 GB
- ➔ Conectividad: 802.11ac WLAN, Bluetooth
- ➔ Red: Ethernet de 1 GB
- ➔ Sistema operativo compatible: Linux para Tegra
- ➔ Tamaño: 50 × 87 mm

Ejemplo CUDA

1. Analice el código vecadd.cu y el Makefile correspondiente. A partir del análisis del código, extraiga cuáles son los pasos generales para la generación de aplicaciones utilizando CUDA.

- Primeramente es necesario incluir la biblioteca “cuda.h” para poder trabajar con el conjunto de funciones para la arquitectura de NVIDIA.
- Luego es necesario reservar la memoria suficiente para poder trabajar con las matrices.
- Seguidamente se define el número de bloques y su tamaño, así como las dimensiones del grid a utilizar.
- El paso siguiente incluye reservar la memoria que se utilizará en la ejecución por cuda (que sería la misma cantidad que se asignó anteriormente).
- Se copian los datos hacia el GPU
- Se llama al método que ejecutará las operaciones mediante CUDA.
- Se copian los resultados desde el GPU al host del computador.

Taller 4: CUDA
Daniel Salas Calderón

2. Analice el código fuente del kernel vecadd.cu. A partir del análisis del código, determine

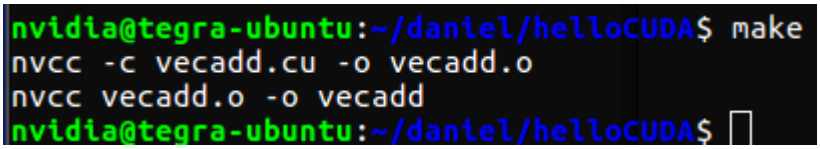
¿Qué operación se realiza con los vectores de entrada?

Realizan una suma vectorial.

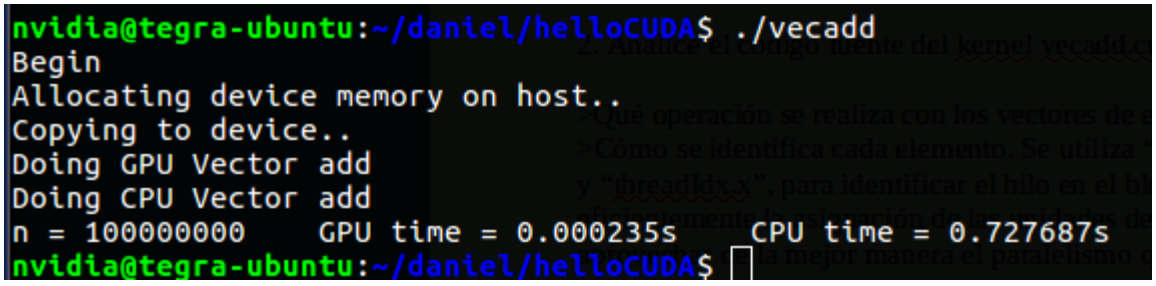
¿Cómo se identifica cada elemento?

Se utiliza “blockIdx.x”, “blockDim.x” para identificar los bloques y “threadIdx.x”, para identificar el hilo en el bloque de ejecución. Su objetivo es administrar eficientemente la asignación de las unidades de procesamiento en conjunto con los hilos para aprovechar de la mejor manera el paralelismo que ofrece la arquitectura.

3. Realice la compilación del código vecadd.cu, utilizando el comando make.



4. Realice la ejecución de la aplicación vecadd.



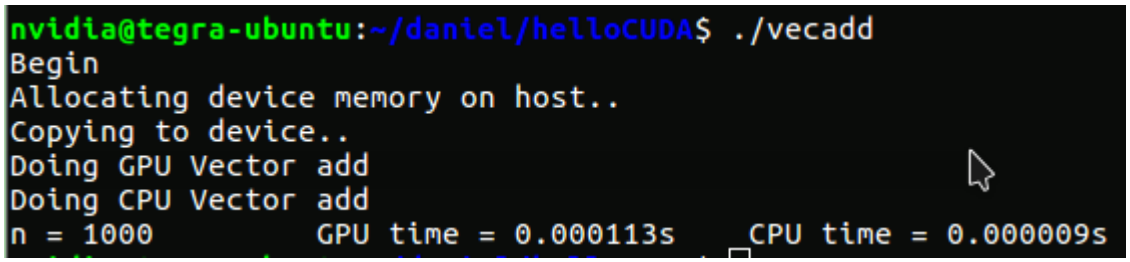
¿Qué hace finalmente la aplicación?

Taller 4: CUDA
Daniel Salas Calderón

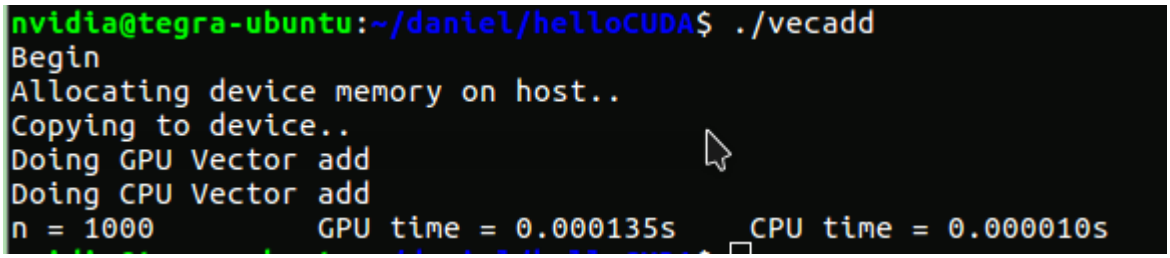
La aplicación realiza una comparación entre los tiempos tomados utilizando la paralelización con CUDA y utilizando el CPU normalmente.

5. Cambie la cantidad de hilos por bloque y el tamaño del vector. Compare el desempeño antes al menos 5 casos diferentes.

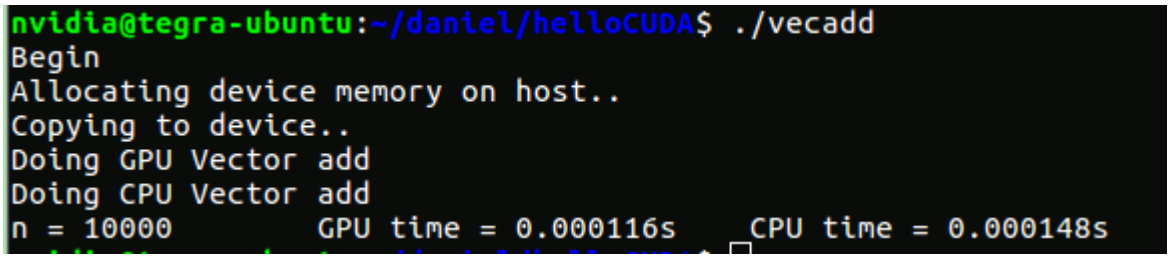
Vector de tamaño 1000 a 400 threads por bloque.



Vector de tamaño 1000 a 800 threads por bloque.

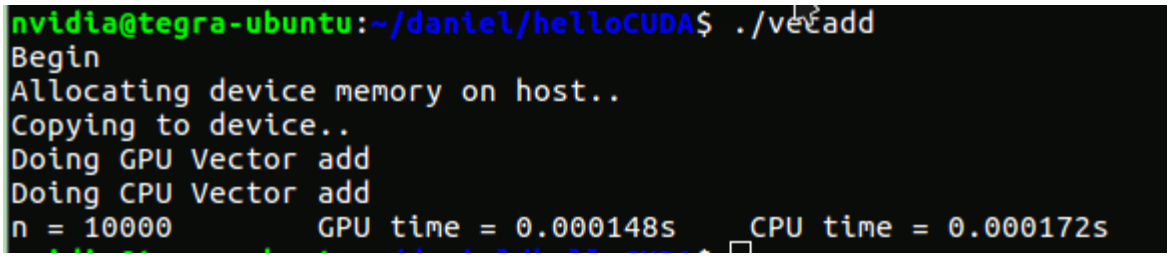


Vector de tamaño 10 000 a 10 threads por bloque.

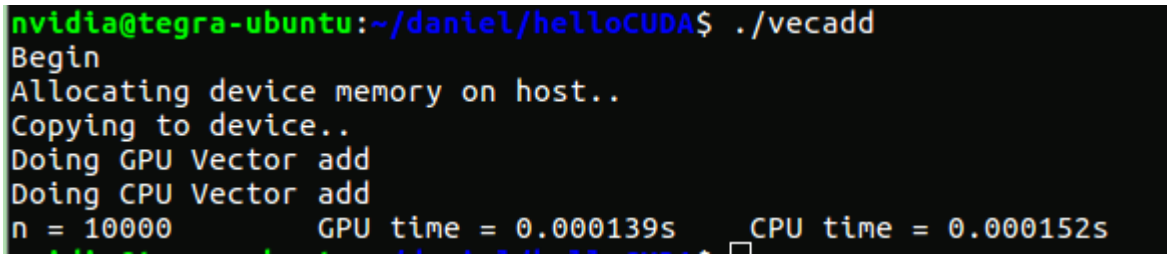


Taller 4: CUDA
Daniel Salas Calderón

Vector de tamaño 10 000 a 100 threads por bloque.



Vector de tamaño 10 000 a 1000 threads por bloque.



Lo primero a observar del comportamiento del programa es que para bajo número de elementos resulta mucho mejor no usar CUDA, pues esto implica mucho overhead a la hora de asignar la memoria que CUDA necesita y también el tiempo gastado en copiar datos de la máquina al GPU y viceversa.

Luego es posible observar, cómo al menos para la aplicación en cuestión al aumentar el número de threads luego de 10 más bien el tiempo de ejecución aumentó, la razón puede darse por que el porcentaje de paralelización del método de sumar los vectores limita el tiempo total. Otro factor que influye es el tamaño del vector, pues luego de cierta cantidad de threads, estos más bien no ayudan a resolver el problema pues puede que más bien estén de “sobra”, entonces el tamaño de los datos con los que se trabaja es realmente importante.

LINK REPO: https://github.com/dranys/TALLER-CUDA