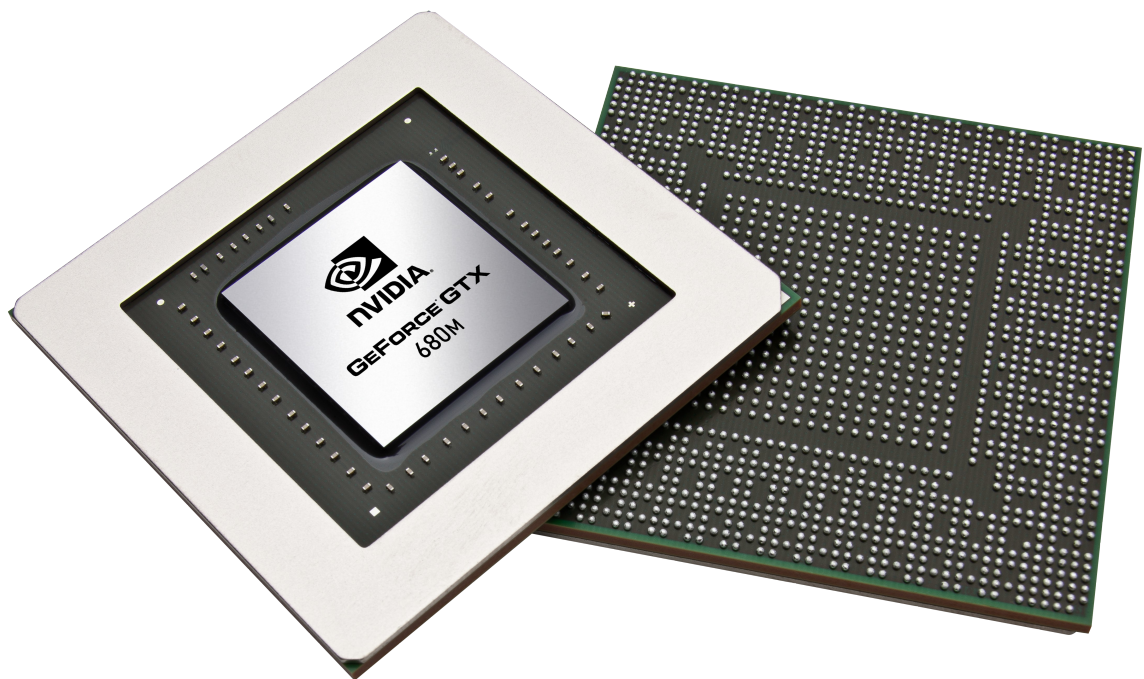


Práctica 8

Programación de GPUs. Librerías



Héctor Garbisu Arocha

Curso 2015/16

Métodos Numéricos para la Computación

Grado en Ingeniería Informática

Escuela de Ingeniería Informática

Universidad de Las Palmas de Gran Canaria

Índice

1. cuBLAS	pág. 3
2. cuSOLVER	pág. 4

1. cuBLAS

Obviando la configuración de Visual Studio y los problemas que tiene, la práctica es sencilla y con carácter analítico una vez más.

Este apartado lo hice en el portátil, que tiene una CPU bastante mejor (proporcionalmente) que la GPU. Los ordenadores del laboratorio son muy parecidos así que espero no tener que repetir pruebas.

Este primer ejercicio consiste en utilizar la librería cuBLAS, que es la forma de usar BLAS en GPU, para multiplicar dos matrices. Para comparar la eficiencia de cuBLAS también haremos la multiplicación utilizando BLAS en la CPU.

El código es el que está explicado en el documento del Tema 2-4.

Este es el resultado cuando se mantiene la línea `cudaDeviceSynchronize()`:

```
C:\WINDOWS\system32\cmd.exe

<CPU> Time (Min,Aver,Max): 440.210 msec.  484.614 msec.  854.734 msec.
-51.8729 6396 -33.061245 90798 78067

<GPU> Time (Min,Aver,Max): 739.122 msec.  739.122 msec.  739.122 msec.
-51.87 29.6396 -33.0612 45.9079 8.78067
```

y ahora sin `cudaDeviceSynchronize()`:

```
C:\WINDOWS\system32\cmd.exe

<CPU> Time (Min,Aver,Max): 439.022 msec.  450.097 msec.  479.729 msec.
67.7388 -35.9849 -18.580313 0774 -49.9708

<GPU> Time (Min,Aver,Max): 81.715 usec.   81.715 usec.   81.715 usec.
67.7388 -35.9849 -18.5803 13.0774 -49.9708
```

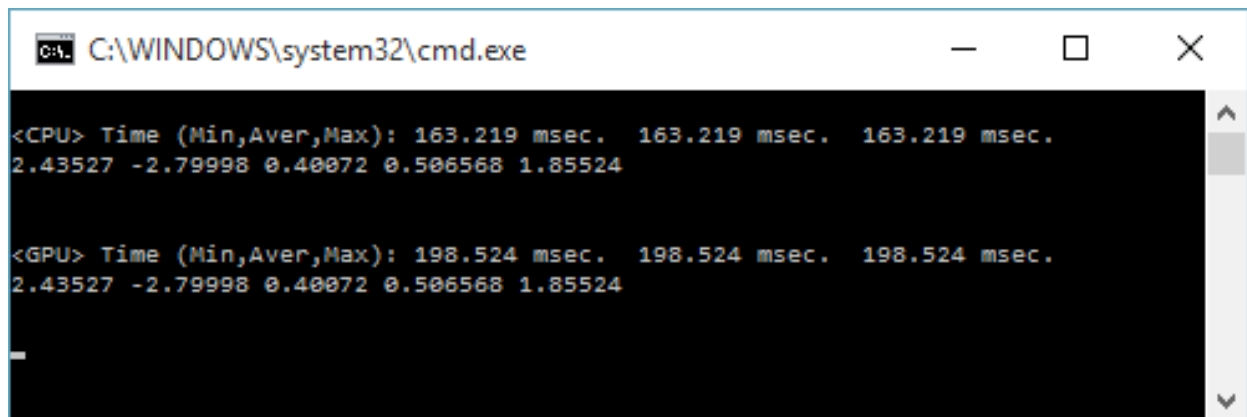
En el primer caso estamos hablando de una medición de tiempo realista sobre la multiplicación en GPU ya que el tiempo se mide cuando cada hilo CUDA ha terminado su parte de la multiplicación. La comparativa con el tiempo de la CPU es similar al de la práctica anterior y es que, en casos tan simples y con GPUs tan rudimentarias, no vale la pena enfrentarse a las complicaciones y tiempos de transferencia de memoria que supone usar la tarjeta gráfica.

La segunda captura es una curiosidad más que otra cosa, ya que el tiempo se ha medido antes de que la multiplicación se complete. En ese caso se han lanzado todos los hilos y el proceso que mide el tiempo ha seguido por su lado, paralelo a éstos. El tiempo que se muestra corresponde a lo que se tarda en 'pedirle' a la GPU que haga la multiplicación.

2. cuSOLVER

En este caso vamos a resolver sistemas de ecuaciones. Para evitar que el proceso de resolución degenera en valores muy pequeños, para los que se tiene poca precisión, se suma 10 a los valores de la diagonal y el resto se mantiene aleatorio entre 0 y 1.

La resolución en CPU se hace con LAPACK, exigiendo hacer copias de las matrices para que los datos puedan ser reutilizados a la hora de calcular los resultados con cuSOLVER.



```
C:\WINDOWS\system32\cmd.exe

<CPU> Time (Min,Aver,Max): 163.219 msec. 163.219 msec. 163.219 msec.
2.43527 -2.79998 0.40072 0.506568 1.85524

<GPU> Time (Min,Aver,Max): 198.524 msec. 198.524 msec. 198.524 msec.
2.43527 -2.79998 0.40072 0.506568 1.85524
```

Los 5 valores de las variables son iguales, así que podemos suponer que de las dos formas se ha obtenido el resultado correcto.

Una vez más, las ventajas de usar GPGPU no están claras gracias al ejemplo. Sin embargo, la tendencia es que la cantidad de núcleos de una GPU es más fácil de incrementar que el rendimiento de una CPU (el coste es menor).

Así que lejos de caer en el desánimo, haber probado que para mi tarjeta gráfica de gama media-baja no se pierde mucho tiempo en comparación con un i7, me da una idea de que en otros sistemas donde la potencia de cómputo esté distribuida más hacia el lado gráfico, hará que el programa compilado aquí sea mucho más eficiente.