Práctica de GPU

Tecnología de la Programación Paralela (TPP)

José E. Román

Esta práctica se compone de dos partes. La primera está basada en CUDA para programar tarjetas gráficas de NVIDIA, mientras que la segunda parte cubre OpenCL.

Programación en CUDA

El objetivo inicial es familiarizarse con el entorno. Para ello, compilaremos y ejecutaremos un programa que proporciona información sobre la(s) GPU(s) del sistema. Posteriormente, pasaremos a implementar diferentes programas que usan *kernels* para realizar cálculos en la GPU, con complejidad creciente.

E1 Información de la GPU

En el directorio `dquery`, compila y ejecuta el programa. La compilación se podría hacer como se muestra a continuación, aunque tanto en este ejercicio como en los otros se proporciona un `makefile` para facilitar la compilación.

```
$ nvcc -o dquery dquery.cu
```

Para ejecutar el programa, se procede como en cualquier otro programa de Linux. La salida del programa proporciona información acerca de las GPUs instaladas y sus características. Este programa es una versión simplificada de `deviceQuery.cpp`, disponible en el SDK de CUDA, concretamente en el directorio `/usr/local/cuda/samples/1_Utilities/deviceQuery`. Modifica el programa para que muestre también otra información, como la anchura del bus de memoria o la velocidad de reloj de la memoria.

Nota: En este caso, el programa no incluye ningún *kernel* sino que únicamente se utilizan funciones del *runtime* de CUDA, por lo que no sería necesario que tuviera la extensión `.cu` y podría compilarse con el compilador de C++ (enlazando las librerías adecuadas, en este caso `-lcudart`).

E2 Suma de vectores

El objetivo es hacer un programa sencillo `vecadd` que sume dos vectores en la GPU. Se proporciona el esqueleto del programa, y hay que completar lo siguiente:

1. Código del *kernel* para sumar los vectores.
2. Reservar la memoria en la GPU.
3. Copiar datos de la CPU a la GPU.
4. Lanzar el *kernel*.
5. Copiar resultado de la GPU a la CPU.
6. Liberar la memoria reservada en la GPU.

E3 Multiplicación de matrices

En este ejercicio vamos a trabajar el producto de matrices.

1. Modifica el programa anterior para que realice el producto de dos matrices. Puede ser una implementación básica, sin utilizar memoria compartida.
2. Implementa una versión de simple precisión para comparar las prestaciones con respecto de la versión de doble precisión.
3. En `/usr/local/cuda/samples` hay dos versiones del producto de matrices, una similar a la estudiada en clase (usando memoria compartida) y otra haciendo una llamada a la librería cuBLAS. Compara las prestaciones de las tres versiones para diferentes tamaños de matriz.

E4 **Reducción**

Se pretende implementar un *kernel* para el producto escalar de dos vectores, $r = x^T y = \sum_{i=1}^{N} x_i y_i$, para lo cual es necesario implementar una reducción.

1. Realiza una implementación trivial y comprueba que el resultado puede ser incorrecto.
2. Implementa dos versiones utilizando operaciones atómicas, una de ellas utilizando la memoria compartida. Compara las prestaciones de las diferentes versiones.

Programación en OpenCL

La inicialización en programas OpenCL es significativamente más compleja que en CUDA. Se proporciona el esqueleto de un programa que puede ser utilizado para varios *kernels*.

E5 **Multiplicación de matrices**

El programa `matmult` de OpenCL está completo salvo por el *kernel*. Crea el fichero `matmult.cl` con una versión sencilla del *kernel* de producto de matrices. A continuación modifica el *kernel* para que implemente una versión a bloques con memoria compartida. Compara ambas versiones entre sí y con las correspondientes de CUDA.