Computación Paralela

Tema 4: Práctica 1

Mayo, 2023

Los programas mencionados en los ejercicios propuestos disponen de un Makefile que permite compilar y ejecutar el código. Para compilar cada uno de ellos, hay que introducir el comando: `make build FILE=<programa>` (sin extensión .cu) y para ejecutarlos: `make run FILE=<programa>`. Por otro lado, si se desean eliminar los archivos con extensión `.o` generados durante la fase de compilación así como el propio ejecutable , basta con teclear: `make clean FILE=<programa>`

Los alumnos que deseen instalar el Toolkit de CUDA en su ordenador personal (bien sea en el Subsistema de Windows para Linux (WSL) o en Ubuntu) pueden seguir los pasos indicados en esta web.

Ejercicio 1

Compila el programa contenido en el directorio `deviceQuery` y, a continuación, ejecútalo para obtener la siguiente información relativa a la GPU:

- Arquitectura de la GPU:
- Driver/Runtime Version:
- Compute Capability:
- Número de SMs:
- Número de CUDA Cores por SM:
- Número Máximo de Threads por SM:
- Número Máximo de Threads por Bloque:
- Dimensiones Máximas del Grid:
- Dimensiones Máximas del Bloque de Threads:

- Cantidad Total de Memoria Global:
- Cantidad Total de Memoria Shared por Bloque:
- Número Total de Registros Disponibles por Bloque:
- Memoria Shared Total por SM:

Ejercicio 2

Realiza los siguientes cambios sobre el código proporcionado en el directorio `cudaTemplate`:

- Completa las instrucciones que se han omitido y asegúrate de que el programa funciona correctamente.
- Analiza cómo se utiliza la memoria compartida en el código del kernel y verifica si su tamaño es correcto. Si no es así, modifícalo de forma adecuada
- ¿Podría suprimirse en el kernel la primera llamada a `__syncthreads()`? ¿Y la segunda?.
- Modifica las zonas del código que sea necesarias para que se pueda hacer uso de un grid y de un bloque de hilos tridimensional.

Ejercicio 3

Realiza los siguientes cambios sobre el código proporcionado en el directorio `vectorAdd`:

- Completa el código fuente en las zonas indicadas, incluyendo en el kernel `vectorAdd` las instrucciones necesarias para que se realice correctamente la suma de los vectores A y B.
- Analiza el código que calcula el número de bloques del grid. ¿Qué sucede si el número total de elementos del vector no es múltiplo del tamaño del bloque de threads?
- Ejecuta el código variando el número de threads por bloque para diferentes tamaños de problema (`numElements`)
- Extiende el programa para que lleve a cabo en un vector D la resta de los elementos de los vectores A y B.
- Añade en la zona adecuada del código las funciones necesarias para la toma de tiempos.

Ejercicio 4

Realiza los siguientes cambios sobre el código proporcionado en el directorio `cudaOpenMP`:

- Modifica el código para que tanto el tamaño del vector como el incremento utilizado y el número de hilos por bloque se le puedan pasar como parámetros de entrada.
- Completa el código del kernel `addConstant` con las instrucciones que sea necesarias para que realice la función que se indica.
- Modifica el código para que el cómputo no se lleve a cabo únicamente en la GPU, sino que tanto los threads OpenMP como la GPU realicen la parte de cómputo que les corresponda.
- Añade en la zona adecuada del código las funciones necesarias para la toma de tiempos.

Ejercicio 5

Realiza los siguientes cambios sobre el código proporcionado en el directorio `cudaMPI`:

- Adapta el código para que permita utilizar grid y bloques bidimensionales. Además, estos valores se han de proporcionar como parámetros de entrada.
- Completa el código del kernel incluido en el fichero `cudaMPI.cu` con las instrucciones que sea necesarias para que realice la función que se indica.
- Modifica el código para que el cómputo no se lleve a cabo únicamente en la GPU, sino que tanto los procesos MPI como la GPU realicen la parte de cómputo que les corresponda.
- Añade en la zona adecuada del código las funciones necesarias para la toma de tiempos.