Práctica 5.- Resolución de problemas con CUDA

Arquitectura y Computación de Altas Prestaciones. Universidad de Granada. Curso 2023/24

Objetivos: En esta práctica se pretende que los estudiantes apliquen y desarrollen sus conocimientos de CUDA para la resolución de problemas.

Bases: Esta práctica se realizará individualmente y los programas desarrollados estarán pensados para funcionar en un entorno Linux con tarjeta gráfica compatible con CUDA. Habrá que entregar los archivos de código fuente desarrollados y una memoria (en formato PDF) que explique el trabajo realizado. La entrega se realizará mediante la actividad que se creará en SWAD para tal efecto. Las entregas tardías supondrán una penalización en la nota (0.5 puntos por día hasta alcanzar el límite de 5).

Entorno: La mayor parte del trabajo se puede realizar en equipos personales. Sin embargo, según la actividad, habrá que acceder al cluster GenMagic. Cada estudiante tendrá un usuario y contraseña en dicho cluster. Serán asignados por el profesor de forma individual e intransferible. Tampoco se podrán cambiar las claves recibidas. El acceso al cluster puede requerir estar conectado por VPN a la red de la universidad.

Ejercicio 1 (1 punto)

Desarrolla un programa en CUDA C que multiplique, elemento a elemento, dos vectores dados y guarde el resultado en un tercero. Por tanto, el kernel requerido se centrará únicamente en esa operación. Los vectores serán de tipo float, y su longitud, junto con el tamaño de bloque, serán un parámetro del programa. El tamaño del grid se ajustará automáticamente con la expresión vista en el seminario de CUDA como un equivalente a la función "*ceil*". El contenido de los vectores se inicializará aleatoriamente en el rango [0, 60). Si su longitud es menor que 7, tanto los vectores de entrada como el resultante se mostrarán por consola. En cualquier caso se incluirá una comprobación con la misma operación secuencial realizada en CPU.

Ejemplo de llamada para vectores de longitud 1024 y bloques de 512 hilos:

$ ./ejercicio1 1024 512
Operación realizada correctamente!

Una vez esté listo el programa anterior, haz una copia y prepara una versión modificada del mismo en el que el tamaño de grid sea fijo (en 1024, por ejemplo), pero que siga siendo compatible con cualquier tamaño de vector y de bloque dado. Por tanto, el ejemplo de llamada anterior se mantendría igual.

Ejercicio 2 (2 puntos)

¿Recuerdas el cálculo del coeficiente de Tanimoto? (Práctica 2, ejercicio 3) Al final, se desarrollaba en torno a la intersección de los dos conjuntos dados. Se pide entonces que hagas un programa en CUDA C que calcule dicho coeficiente. El kernel a desarrollar se centrará en el cálculo de la intersección de los dos conjuntos dados. Como en el ejercicio previo, el tamaño de bloque será un parámetro y el tamaño de grid se calculará en base a éste. Utiliza funciones atómicas en caso de que tengas que agrupar/combinar resultados. Una vez calculada la intersección en GPU, puedes terminar el cálculo del coeficiente en la CPU. El contexto del programa será el mismo que el de la práctica mencionada. Por tanto, el programa desarrollado entonces puede serte útil para comprobar resultados.

Ejemplo de llamada para vectores de longitud 10000 y bloques de 512 hilos:

$ ./ejercicio2 100000 100000 512
El resultado en GPU es: 0.333333

Ejercicio 3 (3 puntos)

Haz un programa CUDA C que calcule el producto de dos matrices de tamaño arbitrario y valores tipo *double*. Ambas matrices contendrán valores reales (decimales) y aleatorios en el rango [0, 100). El tamaño de bloque será otro parámetro, y el de grid se auto-calculará en base a éste. Incluye también una comprobación secuencial en CPU. Además, para matrices pequeñas, hasta 7x7, se mostrarán por consola tanto las dos que se multiplican como el resultado en CPU y GPU. Finalmente, debes tener también una medición, por un lado del tiempo empleado en CPU, y por otro el que se tarda en la GPU (incluyendo las transferencias).

Informalmente (sin promediado), calcula la aceleración del cálculo en GPU respecto al CPU para algún tamaño de entrada relevante.

Ejemplo de llamada para dos matrices 1000 x1000 y bloques de 512 hilos:

$ ./ejercicio3 1000 1000 1000 512
La operación en CPU (secuencial) ha tardado: 10 segundos
La operación en GPU ha tardado: 1 segundo
Cálculo correcto

(Nota: Valores de tiempo inventados en el ejemplo)

Ejercicio 4 (4 puntos)

Finalmente, haz una nueva versión de tu programa CUDA C del cálculo del coeficiente de Tanimoto, pero sin usar funciones de la familia “atomic”. Es decir, si necesitas hacer alguna reducción, impleméntala siguiendo la estrategia vista en el seminario de CUDA C (sección “Ejemplo de reducción” (*)). La reducción debe completarse enteramente en la GPU.

El esquema de llamada al programa puedes mantenerlo igual al del ejercicio 2.

Una vez listo, compara el rendimiento (aceleración) de esta nueva versión con la del ejercicio 2 y con tus versiones secuencial y paralela, (si llegaste a disponer de una).

(*) Nota: El siguiente enlace puede serte de ayuda en esta ocasión: https://stackoverflow.com/questions/24942073/dynamic-shared-memory-in-cuda