Curso 2022-2023

Práctica 5: Iniciación en CUDA

18/11/2022
Prácticas de Programación Concurrente y Paralela
Pablo Revuelta Sanz – José Ranilla

Tipo de Práctica: Guiada con nota
Entrega Documentación: En el Campus Virtual al final de la sesión de prácticas
Duración: 1 Sesión de prácticas (entre el 18 y el 23 de noviembre, según el grupo)

Presentación

La asignatura no dispone de horas suficientes para abordar en profundidad todos los aspectos relevantes de las tecnologías usadas para cada paradigma de paralelismo. En esta práctica el alumnado tomará contacto con CUDA, a través de programas que debe completar y ejecutar.

Como punto de partida recordar (informar) que las GPU NVIDIA cumplen con el estándar IEEE IEEE 754-2008 para las operaciones en coma flotante, que puede diferir del soportado por la CPU/compilador. Por tanto, los resultados obtenidos con CPU y GPU pueden ser diferentes; el error/diferencia dependerá del tipo de operaciones y del número de veces que se apliquen. Si hay diferencias deben ser de magnitudes cercanas a la precisión del tipo de dato utilizado (del orden de 10^{-17} o similar).

El entorno de Producción

Para esta práctica se usará, bajo la supervisión del gestor de colas, los nodos de cálculo con GPU. Éstos tienen una GPU NVIDIA GeForce GTX 1660 Ti, cada una de ellas con 6 GB GDDR6, 1536 Cuda cores, 5.5 TFLOPs y Cuda Compute Capability 7.5

En esta práctica hay que ejecutar a través del SGE, salvo que el alumnado disponga de un entorno hardware (GPU CC 7.5 o superior) y software (CUDA) propio y adecuado.

Tareas a realizar

Copiar el fichero *PRAC05.tgz* de */opt/PracticasPCP2022_2023/Practica05/* a un directorio del *$HOME* propio.

Abordar (resolver) los ejercicios respetando el orden establecido en este enunciado. En caso contrario, los objetivos fijados (la adquisición de conocimiento) no se alcanzarán.

Ejercicio 1º

En los apuntes de teoría, accesibles desde el CV, hay ejemplos sencillos de *kernels* CUDA. Uno de ellos, "*vecadd*", está codificado de diferentes formas en el fichero "*Funciones.cu*". Abrir *Funciones.cu* para ver la codificación de los *kernels*. Editar *VecAdd1.cu* para ver la llamada a los *kernels*.

En **modo interactivo**:

1. Compilar la primera versión: "*make VecAdd1"*.
2. Ejecutar "*./VecAdd1 –2000 32 2000 21*". No olvidar el signo menos del primer 2000. ¿Qué observas?
3. Editar *VecAdd1*. Busque el comentario "Paso 1º". Borrar las 4 líneas que siguen al comentario "Paso 1º" y quitar los comentarios a las 4 líneas que siguen a las borradas. Compilar de igual forma que en el punto 1. Ejecutar en las mismas condiciones que en el punto 2. ¿Qué observas?
4. Ahora ejecutar "*./VecAdd1 2000 32 2000 21*". ¿Qué observas?
5. Editar *VecAdd1*. Buscar el comentario "Paso 2º". Quitar el comentario para habilitar las 7 líneas que le siguen. Compilar de igual forma que en el punto 1 y ejecutar en las mismas condiciones que en el punto 4. ¿Qué observas?

Pasar a **modo batch** (trabajos usando SGE). El *wrapper* para lanzar los trabajos que necesitan GPUs es **ColaGPU**.

6. Crear un trabajo y ejecutar en las mismas condiciones que el punto 5. ¿Qué observas?
7. Editar *VecAdd1*. Buscar el comentario "Paso 3º". Borrar las 3 líneas que siguen al comentario "Paso 3º" y quitar los comentarios a las 3 líneas que siguen a las borradas. Ejecutar en las mismas condiciones que en el punto 6. ¿Qué observas?
8. Cambiar el *script* para ejecutar "*VecAdd1 2000 2048 2000 21*". ¿Qué observas?
9. Buscar en *VecAdd1* el resto de comentarios del estilo "Paso x". Quitar los comentarios a la/s línea/s que le siguen y borrar, si existen, las siguientes que hacen lo mismo sin control de errores. Ejecutar. ¿Qué observas?
10. Preparar un trabajo con las siguientes órdenes de ejecución "VecAdd1 10000 4 1000 2121" y "VecAdd1 1000000 1024 100 2121". ¿Qué observa?

Ejercicio 2º

Editar el contenido del fichero "*VecAdd2.cu*" centrando la atención en las líneas de código entre los comentarios "*/* BEGIN NEW */*" y "*/* END NEW */*".

1. Preparar un trabajo y ejecutar "*VecAdd2 1000000 1024 100 2121*". ¿Qué observas en los tiempos obtenidos?

Ejercicio 3º

Editar el contenido del fichero "*VecAdd3.cu*" centrando la atención en las líneas de código entre los comentarios "*/* BEGIN NEW */*" y "*/* END NEW */*".

1. Preparar un trabajo y ejecutar "*VecAdd3 1000000 1024 100 2121*". ¿Qué observas en los tiempos obtenidos?
2. Ahora ejecutar el trabajo con la orden "*VecAdd3 1000000 32 100 2121*". ¿Qué observas en los tiempos obtenidos?
3. Ahora ejecutar el trabajo con la orden "*VecAdd3 1000000 16 100 2121*". ¿Qué observas en los tiempos obtenidos?

Ejercicio 4º

Editar el contenido del fichero "*VecAdd4.cu*" centrando la atención en las líneas de código entre los comentarios "*/* BEGIN NEW */*" y "*/* END NEW */*".

1. ¿Qué ha cambiado?
2. Preparar un trabajo y ejecutar "*VecAdd4 1000000 1024 100 2121*". ¿Qué observas en los tiempos obtenidos?

Ejercicio 5º

Editar el contenido del fichero "*VecAdd5.cu*" centrando la atención en las líneas de código entre los comentarios "*/* BEGIN NEW */*" y "*/* END NEW */*".

1. ¿Qué ha cambiado?
2. Preparar un trabajo y ejecutar "*VecAdd5 1000000 1024 100 2121*". ¿Qué observas en los tiempos obtenidos?

Ejercicio 6º

Resolver tantos ejercicios como sea posible de la siguiente lista (y al menos uno).

[1] Codificar un *kernel* tal que dados dos vector x e y, de tamaño n, modifique los elementos de x en el siguiente sentido

$$x[i] = y[i]^2 + x[i].$$

Seguidamente, codificar el *kernel* anterior usando *shared*. El tamaño de la *shared* se determinará en tiempo de ejecución[1].

[2] Codificar un *kernel* que, dados 2 vectores x y v y una matriz A siendo x y v de tamaño n y A de dimensiones nxn, resuelva el problema $v = A * x$.
Seguidamente, codificar el *kernel* anterior usando *shared*. El tamaño de la *shared* se determinará en tiempo de ejecución. Ver transparencias de los apuntes del tema de teoría relacionado.

[3] Codificar un *kernel* que, dadas 3 matrices C, B y A todas ellas de dimensiones nxn, resuelva $C = \beta C + \alpha AB$. Ver transparencias de los apuntes del tema CEX correspondiente.

[4] Codificar un *kernel* que, dada una matriz A, de dimensiones mxp, obtenga otra matriz B, de dimensiones pxm, que sea su transpuesta.
Seguidamente, codificar el *kernel* anterior usando *shared*. El tamaño de la *shared* se determinará en tiempo de ejecución.

En todos los ejercicios propuestos se puede usar cualquiera de las formas de reservar memoria vistas en los prototipos *VecAddx*.

¿Qué y cuándo entregar la práctica?

Al final de la sesión de prácticas el alumnado debe subir al Campus Virtual las respuestas a las preguntas formuladas en este enunciado. Los códigos del apartado "Ejercicio 6º" deben adjuntarse en la entrega.

Los nombres de los *kernels* deben ser añadidos a *Funciones.cu* y seguir el siguiente patrón de nombres: kernel*6_X* y kernel*6_XSh*, donde "X" corresponde a la numeración del ejercicio dentro del apartado (1, 2, 3 y 4) y con "Sh" se indica que es la versión correspondiente que usa *shared memory*.

Al final, entregar *Funciones.cu* en el campus virtual.

1 En esta serie de ejercicios es posible que el uso de *shared* carezca de utilidad. El objetivo es que el alumnado practique su uso.