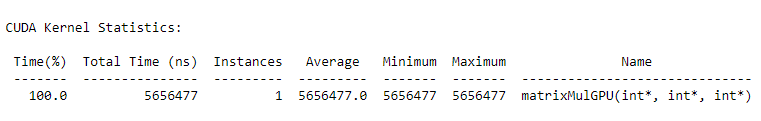
**MEMORIA DE EJERCICICIOS OPCIONALES DEL CURSO DE CUDA**

**PRODUCTO DE MATRICES**

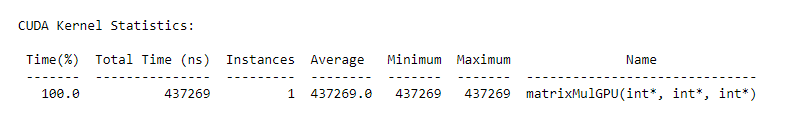
Para calcular el producto de dos matrices cuadradas de tamaño N, se necesita realizar un programa que mediante dos bucles de N iteraciones, se calcule cada elemento de la matriz resultado.

Para analizar dicha mejora, vamos a utilizar el profiler de CUDA para ver el tiempo de ejecución que tendríamos si lo ejecutásemos por defecto y lo compararemos con el tiempo de ejecución del mismo código, pero aprovechando el paralelismo que nos ofrece la GPU.

En primer lugar, si ejecutamos el código sin paralelizar obtenemos el siguiente resultado:

Por otro lado, para paralelizar el código, vamos a aprovechar cada hilo para calcular una celda de la matriz resultante, por lo que hay que calcular a qué celda corresponde cada identificador del hilo. El número de fila y de columna se calcula de la siguiente manera:

El número de bloques que utilizamos sería el número total de hilos que necesitamos, es decir, N x N, divido entre el número de hilos que vamos a utilizar por bloque, es decir, 256 hilos en este caso. Así, obtendríamos el siguiente resultado:

Podemos ver que pasamos de 5,656 ms a 0,437 ms, es decir, se obtiene un factor de mejora de 12.94, lo que viene a ser un 1294 % menos.

**OPTIMIZACIÓN Y ACELEACIÓN DE SAXPY**

En este ejercicio, se nos da un código que contiene algunos errores y que está sin optimizar, aunque sí que está paralelizado, por lo que se propone arreglar dichos fallos y optimizar el código de tal forma que se ejecute el kernel en menos de 200 us.

En primer lugar, hay que solucionar dos fallos que tiene el código, los cuales son:

* Error al calcular el número de hilo: En el código propuesto se calcula el índice multiplicando todas las componentes x de blockIdx, blockDim y threadIdx mientras que lo correcto es multiplicar las componentes x de blockIdx y blockDim y sumar la componente x de threadIdx.
* Falta cudaDeviceSynchronize(), debido a que si la cpu no espera a la finalización de la ejecución del kernel entonces muestra los resultados incorrectos.
* Los datos de las matrices son declarados como float en vez de int, por lo que hay que cambiarlo también.

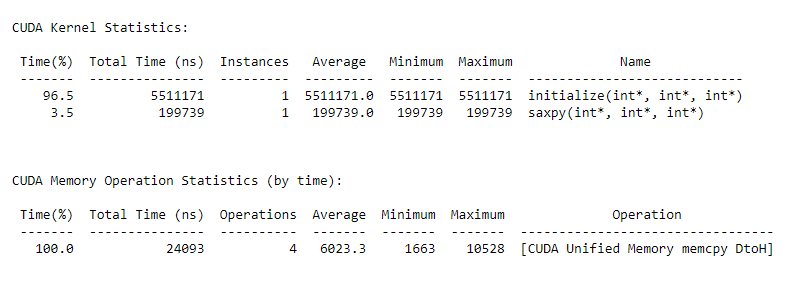
Imagen que contiene Texto

Descripción generada automáticamenteUna vez solucionado esto, los resultados que obtendríamos con el profiler son los siguientes:

Imagen que contiene Escala de tiempo

Descripción generada automáticamenteAhora vamos a añadir una primera optimización, la cual sería ejecutar la función saxpy en la GPU aplicando paralelismo. Para ello, calculamos el índice del array resultado que queremos obtener y hacemos el cálculo que había anteriormente. De esta forma obtendríamos el siguiente resultado:

Hemos conseguido reducir el tiempo de ejecución en 4 ms, pero el tiempo de copiar los datos a kernel y traerlos sigue siendo el mismo, por lo que vamos a añadir una nueva optimización para ello.

La siguiente optimización consiste en inicializar los datos en la GPU, ejecutando como kernel y paralelizando la función de inicialización. Cada hilo se encarga de inicializar la celda *i* de cada array. Entonces, ejecutamos y obtenemos el siguiente resultado:

Aquí vemos una clara mejora tanto en tiempo de ejecución como en tiempo de copiado de datos. Gracias a inicializar los datos en GPU, los datos no tienen que copiarse a la GPU, por lo que solo consumimos tiempo en traerlos de vuelta a la CPU.

Por otro lado, tenemos la reducción en tiempo de ejecución del kernel, ya que se consigue ejecutar por debajo del objetivo, es decir, se ejecuta en 199 us.

**ACELERACIÓN DEL CÁLCULO DE LA CONDUCTIVIDAD TÉRMICA DE LA PLATA**

Para realizar el cálculo de la conductividad térmica de la plata se proporciona un código con la función step\_kernel\_mod, que se encarga de realizar dicho cálculo. Está compuesta por dos bucles donde uno de ellos está anidado, debido a que se debe recorrer una matriz de Nj x Ni donde los elementos de los bordes no se utilizan.

Para acelerar dicha función, vamos a aplicar el mismo método que para el cálculo de la multiplicación de matrices, es decir, el cálculo que se realiza para cada (j,i) lo va a realizar un solo hilo. En este caso, el cálculo de j e i se realiza de una forma un poco diferente:

La variable *idx* es el id del hilo, que se calcula como en la multiplicación de matrices.

Captura de pantalla de computadora

Descripción generada automáticamente con confianza mediaSi ejecutamos el programa como se nos da con el profiler, obtenemos los siguientes datos:

Podemos ver que ha tardado en ejecutar el kernel casi 430 ms.

Imagen que contiene Diagrama

Descripción generada automáticamenteA continuación, vamos a paralelizar el código según lo comentado anteriormente, por lo que obtenemos los siguientes resultados:

En este caso, hemos obtenido un tiempo de ejecución del kernel de 1,84 ms, es decir, una mejor del 233 %.