Universidad de la República
Facultad de Ingeniería, INCO
Montevideo, Uruguay.

Práctico 4:

GPGPU

Grupo 10

Integrantes:

Federico Gil - 5.198.750-6 - federico.gil@fing.edu.uy

Diego Amorena - 5.011.502-7 - diegoamorenag@gmail.com

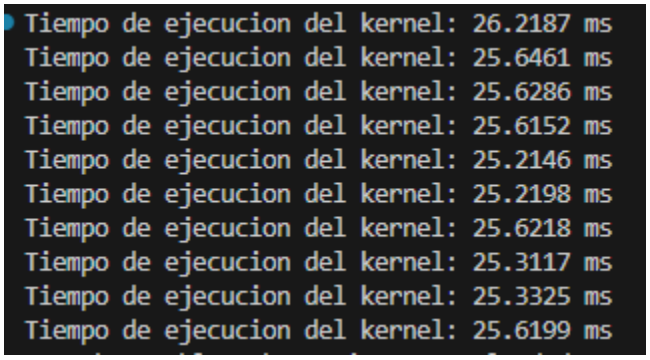
Ejercicio 1

Parte a

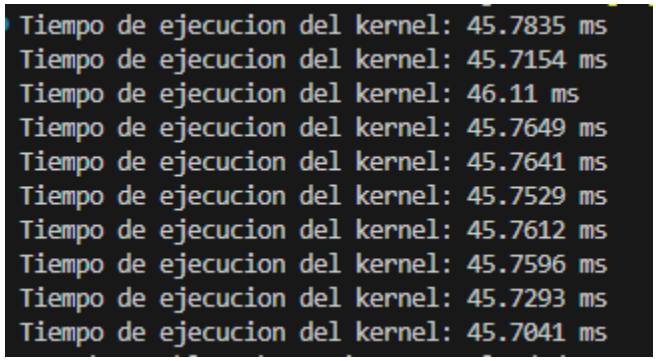
En la ejecución del ejercicio 1 del práctico 3 se obtuvo un tiempo medio de 45.7845 ms y desvío 0.1169 ms para matrices de tamaño 2^14 para tiles de 32x32 en memoria global.

Con la variación de código sugerida, de utilización de la memoria shared, se logró reducir el tiempo promedio a 25.5428 ms y desvío 0.2985 para el mismo tamaño de matriz y de tile.
Es importante mencionar, que se sigue teniendo accesos ineficientes (no-coalesed), sin embargo, al utilizar la memoria compartida estos accesos resultan más rápidos.

En la primera imagen se puede apreciar el resultado para la nueva implementación



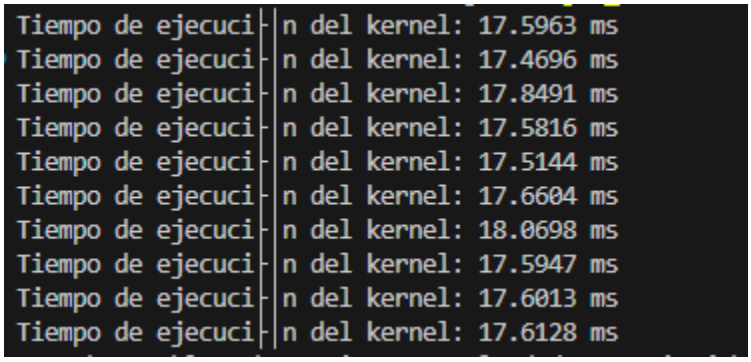
Y en la siguiente los resultados de la ejecución de la implementación anterior:



Parte b

Al agregar una columna adicional (dummy), los accesos a la memoria compartida se desplazan, de modo que los hilos de un warp ya no acceden a la misma columna del banco de memoria. Esto distribuye los accesos de los hilos de manera más uniforme entre los bancos de memoria, eliminando los conflictos.

Al agregar la columna dummy se obtienen tiempos medios de 17.655 ms y desvío de 0.1767 ms. Una mejora significativa respecto a la implementación de la parte anterior y notablemente mejor que la implementación del práctico anterior.



Ejercicio 3

Parte a

Con la implementación donde cada bloque mantiene su histograma local y al finalizar hace la suma atómica a un histograma global se obtuvieron los siguientes resultados. El tiempo medio de ejecución fue de 1.8189 ms y un desvío de 0.0072 ms en diez corridas del programa. Para una imagen de 4K 16:9. Con valores generados al azar entre 0 y 255 para cada posición del arreglo. Para la implementación secuencial se obtuvieron tiempos de 14.889 ms de media y 0.0549 ms de desvío. Mientras que para una matriz de 15360x8640, la media fue 22.0434 ms y desvío de 0.0474 ms y la implementación secuencial, 248.355 ms de media y 4.908 ms de desvío.

Parte b

Para implementar esta parte se tuvo en cuenta el patrón de reducción visto en el teórico, en particular, el primer enfoque de este. Para las ejecuciones con la matriz solicitada de 3840x2160 se obtuvo un tiempo medio de 1.7577 ms y desvío de 0.1153 ms para la implementación paralela, mientras que 14.32 ms de media y 0.2838 ms de desvío para implementación secuencial. Mientras que para matrices más grandes, como la comentada en la parte anterior, el tiempo medio fue de 22.7302 ms y desvió 0.4092 ms en la implementación paralela y 223.7827 ms de media y 1.174 ms de desvío para la implementación secuencial. La mejora no es significativa para estos tamaños de matrices.

Para esta parte se plantean dos alternativas, una que sigue la sugerencia de reducción llamando iterativamente al kernel de reducción y otra que no. La que no lo sigue fue la comentada. La que si lo sigue genera los siguientes resultados, peores que la que no, tiempo

medio 3.20 ms y desvío de 0.55 ms. para la matriz 4k y media de 22.2465 y desvío de 0.2316 ms. Esto puede deberse al tamaño de bloque elegido. Se utilizó 16 para esta parte pues divide bien a las dimensiones. Pero se podría explorar con otros valores de tamaño de bloque.

El archivo Ej3b.cu es la implementacion que aplica el patrón reduce directamente y el archivo Ej3bII.cu el que lo aplica de manera iterativa.