GPGPU - Laboratorio 4

Ejercicio 1

Este ejercicio trata sobre el uso de la memoria compartida de la GPU para evitar el acceso no coalesced a memoria global. Se trabajará sobre el programa que transpone una matriz desarrollado en el práctico anterior. Como se vio en el práctico anterior, si se divide la matriz en *tiles* bi-dimensionales de tamaño igual al tamaño de bloque, y este tiene 32 hilos en la dirección x (blockDim.x=32), la lectura del tile o su escritura en la matriz transpuesta originará 32 transacciones de memoria de 32 bytes, y de cada transacción solo se utilizan 4 bytes (acceso no coalesced). En el práctico anterior se eligió un tamaño de bloque para minimizar este desperdicio de ancho de banda. En este práctico se seguirá una estrategia distinta, que consiste en usar una copia del tile a trasponer en memoria compartida de forma de realizar los accesos que serían ineficientes sobre el memoria compartida en lugar de sobre la memoria global.

- **a)** Reserve un espacio en memoria compartida de tamaño igual al del tile (transpuesto) y realice realice la operación utilizando el tile para evitar los accesos no-coalesced a la memoria global. Compare el desempeño del kernel con las versiones desarrolladas en el práctico anterior.
- **b)** Analice el patrón de lectura y escritura en la memoria compartida y determine cuando ocurren conflictos de bancos. Solucione los conflictos de bancos agregando una columna *dummy* al final del tile de memoria compartida. Compruebe el efecto de esta modificación comparando los tiempos de ejecución con la parte anterior y explique por qué esto soluciona los conflictos de bancos.

Ejercicio 3

El objetivo del ejercicio es utilizar la memoria compartida para la *privatización*. Además se utilizará el patrón de reducción paralela. Antes de empezar, defina una matriz de enteros de 3840 $\times$ 2160 (4K 16:9) y adapte el kernel del histograma realizado en el práctico 2 para que trabaje sobre una matriz de enteros entre 0 y 255.

- **a)** Desarrolle una variante donde cada bloque mantenga un histograma local en memoria compartida y, al final de la recorrida de la imagen, los datos del histograma local se impacten en el histograma global utilizando *atomicAdd*.
- **b)** Desarrolle otra variante donde, una vez realizada la recorrida de la imagen, cada bloque impacte el histograma local en una fila de una "matriz de histogramas". Luego, otro kernel debe realizar la suma por columnas de la matriz para obtener el histograma global.
  - Evite el acceso "no coalesced" eligiendo un tamaño de bloque adecuado.
  - Use el patrón de reducción visto en teórico para sumar cada segmento de una columna.
  - Guarde la suma parcial de cada bloque en la propia matriz de histogramas.
  - Invoque nuevamente el kernel para sumar las sumas parciales del paso anterior hasta que se haya sumado toda la columna.

– Evite acceder a los datos en memoria global de forma “no coalesced” eligiendo un tamaño de bloque adecuado.
– Agregue algunas filas de 0s a la matriz para que el número de filas sea múltiplo del tamaño de bloque elegido.
– Compare el desempeño de esta solución con la de la parte a) y la adaptada del práctico 2.