GPGPU - Laboratorio 4

Ejercicio 1

Este ejercicio pretende ilustrar el uso de la memoria compartida de la GPU para evitar el acceso no coalesced a memoria global.

Dada una matriz $A_{ij} \in R^{n \times n}$ su transpuesta es una matriz $A^T \in R^{n \times n} : A^T_{ij} = A_{ji}$, $i, j = 0, ..., n$.

- **a)** Desarrolle una función que dada una matriz en la memoria de la GPU devuelva su matriz transpuesta. La función no debe modificar la matriz de entrada. La primera versión no utiliza la memoria compartida.

Divida conceptualmente la matriz en *tiles* bi-dimensionales (cuadrados) de tamaño igual al tamaño de bloque elegido en la configuración de la grilla. Reserve un espacio en memoria compartida de tamaño igual al del tile y realice realice la operación en tres pasos, como se muestra en la Figura 1:

1. Cada thread carga en la posición correspondiente del *tile* en memoria compartida el elemento de la matriz correspondiente a su posición.
2. Cada thread carga en un registro el valor correspondiente a la posición del tile transpuesto.
3. Cada thread escribe el valor del registro en la posición correspondiente del resultado en memoria global.

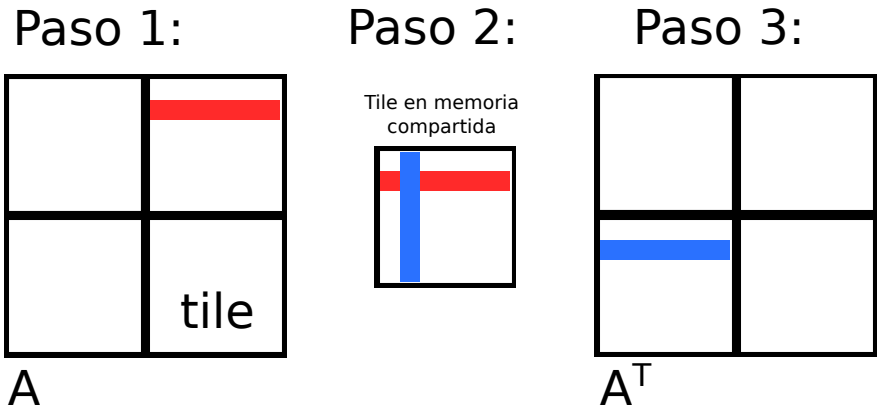

Figura 1: Luego de completado el paso 1, la memoria compartida de cada bloque se encuentra cargada con el *tile* correspondiente. En el paso 3 se escribe el tile correspondiente en la matriz transpuesta leyéndolo de la memoria compartida por columnas y escribiéndolo en memoria global por filas de forma coalesced.

- **b)** Desarrolle una versión de la transposición siguiendo el procedimiento que fue explicado anteriormente. Compare el desempeño del kernel con el de la parte a). Explique detalladamente por qué el procedimiento debería producir una mejora en el tiempo de ejecución.

- **c)** Analice el patrón de lectura y escritura en la memoria compartida y determine cuando ocurren conflictos de bancos.
- **d)** 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. Explique detalladamente por qué esto soluciona los conflictos de bancos.

Ejercicio 2

El objetivo de este ejercicio es reutilizar datos en el kernel de convolución desarrollado en el práctico anterior utilizando la memoria compartida.

La solución consiste en cargar los datos utilizados por cada bloque de hilos a memoria compartida una sola vez, y luego realizar la recorrida de cada hilo sobre memoria compartida, evitando cargas redundantes desde memoria global.

Dadas las características del filtro, aquellos threads que operen sobre el borde de cada bloque deberán acceder a pixels (que están dentro de su ventana) pertenecientes a otro bloque de la imagen. Deberán cargarse en memoria compartida los pixels correspondientes a cada thread y además aquellos pixels que pertenecen a la ventana de los threads del borde, tal como muestra la Figura 2.

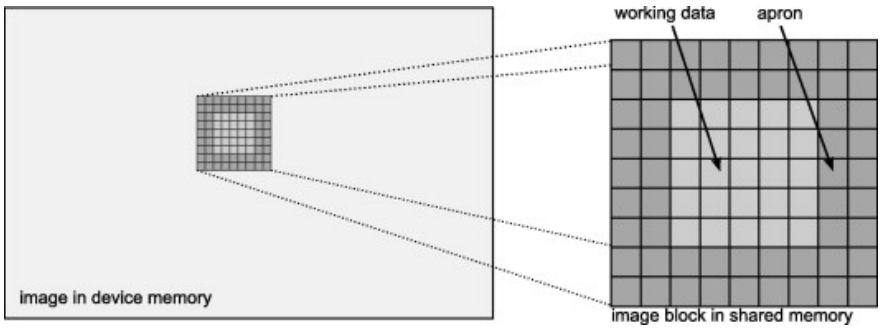

Figura 2: Datos cargados en memoria compartida

La carga a memoria compartida debe llevarse a cabo con el mayor grado de paralelismo posible pero utilizando únicamente la cantidad de threads necesaria para realizar el cómputo, es decir, algunos hilos deberán cargar más de un valor debido a los "bordes".

Compare el tiempo de ejecución del kernel con el del práctico anterior para distintos tamaños de bloque y explique los resultados.

Ejercicio 3

El objetivo del ejercicio es utilizar la memoria compartida para la *privatización*. Además se utilizará el patrón de reducción paralela. Adapte el kernel del práctico 2 para que compute el histograma de la imagen (los valores de cada pixel van del 0 al 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*. Compare el desempeño de esta variante con la del práctico 2.
- **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. De ser posible, evite acceder a los datos en memoria global de forma "no coalesced". Explique detalladamente su solución en el informe.