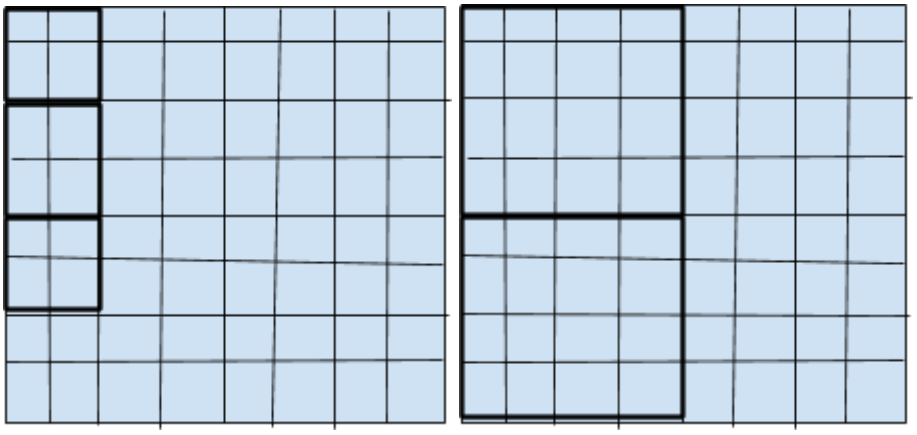
Ejercicios capítulo 4

Alumno: Dany Mauro Diaz Espino

1. Consider matrix addition. Can one use shared memory to reduce the global memory bandwidth consumption? Hint: Analyze the elements accessed by each thread and see if there is any commonality between threads.

No, ya que en este caso, ningún subproceso comparte datos de entrada, cada subproceso toma un elemento de una matriz y lo suma, pero estos elementos no se repiten, por lo que el uso compartido de lectura no se puede utilizar mediante la memoria compartida.

2. Draw the equivalent of Fig. 4.14 for an 8× 8 matrix multiplication with 2× 2 tiling and 4× 4 tiling. Verify that the reduction in global memory bandwidth is indeed proportional to the dimensions of the tiles.



Se reduce el número en un factor de N si los tiles son de N x N, ya que al cargar estos tiles en memoria compartida, estos datos pueden ser utilizados por todos los threads del mismo bloque, sin necesidad de recurrir a la memoria global.

3. What type of incorrect execution behavior can happen if one or both __syncthreads() are omitted in the kernel of Fig. 4.16?

Si el primer __syncthreads() se omite, entonces los threads seguirán avanzando antes de que todos los tiles de M y N se guarden en Mds y Nds. Si el segundo __syncthreads() se omtie, entonces un thread puede avanzar a la siguiente iteración y jalar nuevos M y N elementos a la memoria compartida antes de que los demás terminen de usar los datos que se encontraban ahí.

4. Assuming that capacity is not an issue for registers or shared memory, give one important reason why it would be valuable to use shared memory instead of registers to hold values fetched from global memory? Explain your answer.

La memoria compartida tiene mayor latencia que los registros, pero la ventaja es que las variables que residen en la memoria compartida son accesibles por todos los threads en un

bloque, mientras que los datos de registro son privados en cada thread. La memoria compartida está diseñada para admitir el intercambio eficiente y de gran ancho de banda de datos entre subprocesos en un bloque.

5. For our tiled matrix–matrix multiplication kernel, if we use a 32x32 tile, what is the reduction of memory bandwidth usage for input matrices M and N?
A. 1/8 of the original usage
B. 1/16 of the original usage
C. 1/32 of the original usage
D. 1/64 of the original usage

La respuesta es C. El número de accesos a memoria se reduce según el tamaño del tile, siempre que se cumpla: el tamaño de la matriz es múltiplo del tamaño de los thread blocks y que las matrices son cuadradas.

6. Assume that a CUDA kernel is launched with 1,000 thread blocks, with each having 512 threads. If a variable is declared as a local variable in the kernel, how many versions of the variable will be created through the lifetime of the execution of the kernel?
A. 1
B. 1000
C. 512
D. 512000

La respuesta es D. Las variables escalares declaradas en el kernel se guardan automáticamente en registros, cuyo scope es cada thread individual. En este caso, como hay 1000 thread blocks de 512 threads cada uno, la variable tendrá 512000 versiones diferentes.

7. In the previous question, if a variable is declared as a shared memory variable, how many versions of the variable will be created throughout the lifetime of the execution of the kernel?
A. 1
B. 1000
C. 512
D. 51200

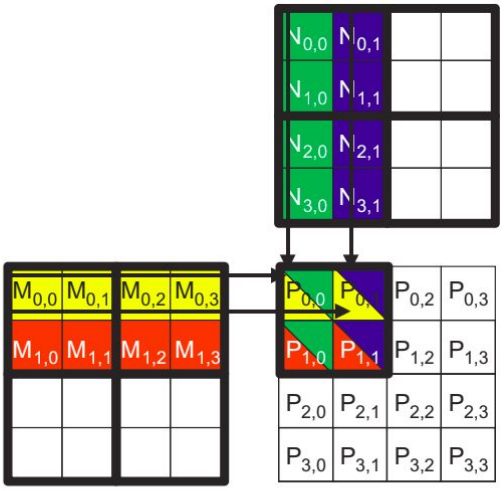
Respuesta B. Si una variable se declara como compartida, entonces su scope es todo el bloque, lo que significa que todos los threads del mismo bloque van a usar la misma variable. En este caso como hay 1000 bloques, se crean 1000 versiones de la variable.

8. Consider performing a matrix multiplication of two input matrices with dimensions N × N. How many times is each element in the input matrices requested from global memory in the following situations?
A. There is no tiling.
En este caso cada elemento se extrae N veces. Al no haber tiling, es necesario hacer la multiplicación de filas por columnas, y al ser matrices de N x N, significa que un elemento de

la matriz A va a ser extraído N veces, que es el número de columnas, y un elemento de la matriz B va a ser extraído N veces también, que es el número de filas.

B. Tiles of size T × T are used.

Cuando se usan tiles de T x T la multiplicación se hace en fases, sin embargo el número de veces que un elemento es accesado se reduce a N/T.



Por ejemplo en la imagen se ve una matriz de 4 x 4, y un tile de 2 x 2. En este caso el elemento M0,0 tiene que ser accesado 2 veces, ya que se va a usar en dos bloques: (N0,0 N0,1 N1,0 N1,1) y (N0,2 N0,3 N1,2 N1,3). De igual manera el elemento N0,0 va ser accesado dos veces, para ser usado en los bloques: (M0,0 M0,1 M1,0 M1,1) y (M2,0 M 2,1 M3,0 M3,1).

9. A kernel performs 36 floating-point operations and 7 32-bit word global memory accesses per thread. For each of the following device properties, indicate whether this kernel is compute- or memory-bound.

A. Peak FLOPS= 200 GFLOPS, Peak Memory Bandwidth= 100 GB/s

Compute: (36 operaciones)/(200 * 10^9 operaciones/s) = 0.18 * 10^(-9) s
Memory: (7*4 Bytes)/(100 * 10^9 bytes/s) = 0.28 * 10^(-9) s
Como 0.28 * 10^(-9) s > 0.18 * 10^(-9) s entonces es memory-bound

B. Peak FLOPS= 300 GFLOPS, Peak Memory Bandwidth= 250 GB/s

Compute: (36 operaciones)/(300 * 10^9 operaciones/s) = 0.12 * 10^(-9) s
Memory: (7*4 Bytes)/(250 * 10^9 bytes/s) = 0.112 * 10^(-9) s
Como 0.112 * 10^(-9) s < 0.12 * 10^(-9) s entonces es compute-bound.

10. To manipulate tiles, a new CUDA programmer has written the following device kernel, which will transpose each tile in a matrix. The tiles are of size BLOCK_WIDTH by BLOCK_WIDTH, and each of the dimensions of matrix A is known to be a multiple of BLOCK_WIDTH. The kernel invocation and code are shown below. BLOCK_WIDTH is known at compile time, but could be set anywhere from 1 to 20.

```
dim3 blockDim(BLOCK_WIDTH,BLOCK_WIDTH);
dim3 gridDim(A_width/blockDim.x,A_height/blockDim.y);

BlockTranspose<<<gridDim, blockDim>>>(A, A_width, A_height);

__global__ void
BlockTranspose(float* A_elements, int A_width, int A_height)
{
        __shared__ float blockA[BLOCK_WIDTH][BLOCK_WIDTH];
        int baseIdx = blockIdx.x * BLOCK_SIZE + threadIdx.x;
        baseIdx += (blockIdx.y * BLOCK_SIZE + threadIdx.y) * A_width;


        blockA[threadIdx.y][threadIdx.x] = A_elements[baseIdx];
        A_elements[baseIdx]=blockA[threadIdx.x][threadIdx.y];
}
```

A. Out of the possible range of values for BLOCK_SIZE, for what values of BLOCK_SIZE will this kernel function execute correctly on the device?

Un BLOCK_SIZE menor que o igual a 5 funcionará correctamente. Un bloque configurado de tal manera solo contendrá un warp de threads y ejecutará todo el programa correctamente. Un BLOCK_SIZE de más de 5 creará bloques de subproceso con más de un warp, y no se garantiza que los valores escritos por un warp en la matriz de memoria compartida sean adyacentes a los valores que necesitara otro warp.

B. If the code does not execute correctly for all BLOCK_SIZE values, suggest a fix to the code to make it work for all BLOCK_SIZE values.

Para garantizar el orden de las lecturas y escrituras de la memoria compartida en los subprocesos de este kernel, se debe colocar una llamada a __syncthreads () entre las líneas que leen y escriben la matriz de memoria compartida.