MP3

1.A kernel is launched with 256 thread blocks, each consisting of 512 threads. The kernel contains a shared memory variable, X. How many versions of X are created during execution of the kernel and how many threads are able to access one version of X? Explain your answer.
(A) 1, 256 * 512
(B) 256, 512
(C) 512, 256
(D) 256 * 512, 512
(E) 256 * 512, 256 * 512

Answer: B (1 point). Each block creates one version of X. Only threads in the same block are able to access shared memory.

2. Assuming an 16x16 tile, approximately what fraction of the memory bandwidth (global memory accesses) required by the simple matrix multiplication kernel (MP2) is required by the tiled matrix multiplication kernel (MP3)? Explain your answer.

Answer: 1/16 (2 points). The number of global memory accesses is proportional to the tile size. In each tile, threads collaboratively load all elements required to compute that tile, and hence reduce the global memory accesses by 1/16.
If answer 16 with reasonable explanation, can also give full points

3. Assuming 32x32 tiles, how many floating-point operations are performed by your tiled matrix multiplication kernel considering the halo cells? Answer in terms of numARows, numAColumns, numBRows, numBColumns, numCRows, and/or numCColumns and explain your answer.

Answer:
The fp operation for halo cells only should involve 0s.
2 * ceil(numCRows / 32) * ceil(numCColumns / 32) * ceil(numAColumns / 32) * 32 * 32 * 32
or 2 * ceil(numCRows / 32) * ceil(numCColumns / 32) * ceil(numBRows / 32) * 32 * 32 * 32
2 points, 1 point if missing the 2 (total 2 operations for addition and multiplication)

4 . Assuming 16x16 tiles, how many global memory data reads in TOTAL are performed by your kernel? How many global memory data writes? Answer in terms of numARows, numAColumns, numBRows, numBColumns, numCRows, and/or numCColumns, and explain your answer. Give the answers for reads and writes separately.

Answer:
numAColumns * numARows * ceil(numCColumns or numBColumns/16) + numBColumns * numBRows * ceil(numCRows or numARows/16) global memory reads.

numCRows * numCColumns global memory writes since we only write once for one output element. 2 total points (1 for read 1 for write) ok if times 4 bytes

5. The kernel code fragment below is intended to make some threads do two tasks while other threads do no work. Meanwhile, threads should be synchronized between task 1 and task 2. Choose the code that is able to realize such functionality and explain your answer.

(A)
```
if (myId < totalSize) {
    // do task 1
    __syncthreads();
    // do task 2
    } else {
    __syncthreads();
    }
```

(B)
```
if (myId < totalSize) {
    // do task 1
    __syncthreads();
    // do task 2
    }
```

(C)
```
if (myId < totalSize) {
    // do task 1
    }
  __syncthreads();
  if (myId < totalSize){
    // do task 2
    }
```

(D)
```
if (myId < totalSize) {
    // do task 1
    __syncthreads();
    } else {
    __syncthreads();
    }
  if (myId < totalSize) {
    // do task 2
    __syncthreads();
    } else {
    __syncthreads();
    }
```

Answer:
C (1 point). Only in C all the threads are able to reach the same __syncthreads().

6. Assume that the dimensions of matrices are bigger than the maximum thread dimensions, so we cannot use the simple matrix multiplication kernel. How should we modify the code to perform matrix multiplication algorithm in this case? Select all that apply.
(A) Increase the block dimension and decrease the grid dimension
(B) Write kernel code that calculates more than one output
(C) Use tiled matrix multiplication kernel
(D) Loading the matrix from global memory to shared memory

Answers:

B (2 points. 1 point for two selections including B). We can write kernel code that calculates more than one output. This will require the system to serialize the operation which will slow down the performance speed; however, it can handle the matrices whose dimensions are bigger than the max thread dimensions.

Tiling uses the same amount of threads as the simple multiply kernel.