**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.

**Answer:** For matrix addition usage of shared memory will not reduce the global memory bandwidth consumption as each element of the input arrays are accessed by just one thread.

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

**Answer: D. 512000**

1,000 therads and 512 threads per block results in 512,000 threads. As a private copy of the local variable is generated for each thread there will be a total of 512,000 copiees of the variable created through the lifetime of the kernel.

**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

**Answer: B. 1000**

The scope of a shared variable is within a thread block. A private version of shared variable is created of each thread block during kernel execution. The given CUDA kernel in the previous question had a total of 1,000 thread blocks. Hence we would get a total of 1000 copies of shared variable.

**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

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

**Answer:**

Global memory access = 28 Bytes per thread

36 floating-point operations per thread

compute-to-global-memory-memory-access ratio is 9:7

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

Kernel loads 3.57(100/28) giga operands per second. This achieves 4.59 (3.57 x 9 / 7) GFLOPS.

As this is less than the peak FLOPS of 200 GFLOPS given in device configuration A the kernel is memory-bound.

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

Kernel loads 8.93(250/28) giga operands per second. This achieves 11.48(8.93 x 9 / 7) GFLOPS.

As this is less than the peak FLOPS of 300 GFLOPS given in device configuration B the kernel is memory-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?

**Answer:**

It excutes incorrectly for all values of BLOCK\_SIZE. The kernel correctly transposes the matrix within each block, but fails to transpose the entire input matrix.