Name: **12.01.2023**
Student ID:

CENG443 Final Exam - Fall 2022
(90 minutes)

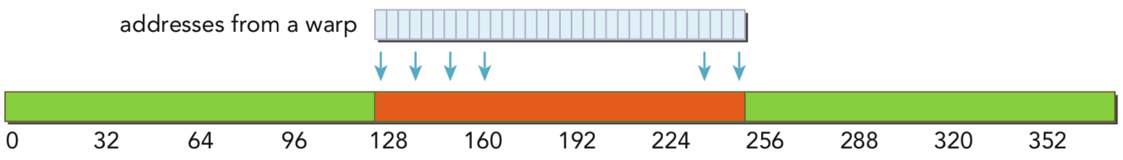
Q.1 (15 Points) Assuming that capacity is not an issue for registers or shared memory in a GPU device, give one important reason why it would be valuable to use shared memory instead of registers to hold values fetched from global memory? Explain your answer with an example.

Q.2 (15 Points) 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 by justifying your answer.

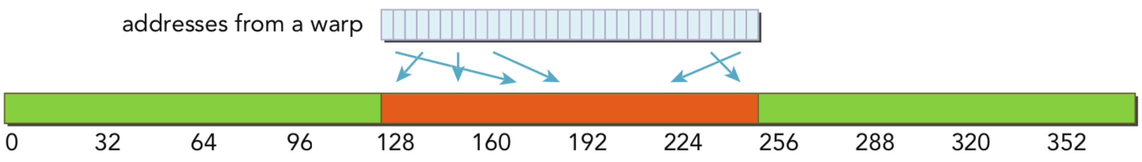
A. Peak FLOPS= 200 GFLOPS, Peak Memory Bandwidth= 100 GB/s
B. Peak FLOPS= 300 GFLOPS, Peak Memory Bandwidth= 250 GB/s

Q.3 (25 Points)
Assume the burst size in a global memory of a GPU device is 128-byte, where each memory transaction loads 128-byte data from the global memory. Calculate the total number of memory transactions and the bus utilization (i.e., # bytes requested/# bytes loaded) for the following cases (Although the illustrations demonstrate only 6 four-byte accesses, you should consider that there are 32 four-byte accesses (by each thread in a warp) covering all threads in a warp).

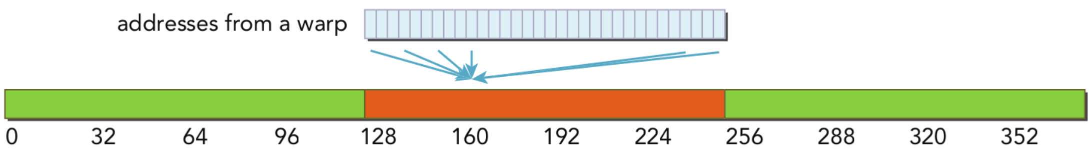
a) The addresses requested by all threads in a warp fall within one burst section of 128 bytes, in consecutive order by thread ID.



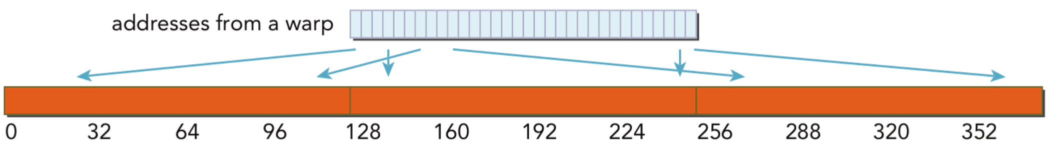
b) The addresses requested by all threads in a warp fall within one burst section of 128 bytes, not consecutive by thread ID, but rather randomized within a 128-byte range.



c) All threads in the warp request the same address.



d) The threads in a warp request 32 four-byte addresses scattered across global memory (across 3 cache lines).



Q.4 (25 Points) What type of incorrect execution behavior can happen if one or both `__syncthreads()` are omitted in the following kernel?

```
__global__ void MatrixMulKernel(float* d_M, float* d_N, float* d_P, int Width) {
      __shared__ float Mds[TILE_WIDTH][TILE_WIDTH];
      __shared__ float Nds[TILE_WIDTH][TILE_WIDTH];
      int bx = blockIdx.x; int by = blockIdx.y;
      int tx = threadIdx.x; int ty = threadIdx.y;
      int Row = by * TILE_WIDTH + ty;
      int Col = bx * TILE_WIDTH + tx;
      float Pvalue = 0;

      for (int ph = 0; ph < Width/TILE_WIDTH; ++ph) {
            Mds[ty][tx] = d_M[Row*Width + ph*TILE_WIDTH + tx];
            Nds[ty][tx] = d_N[(ph*TILE_WIDTH + ty)*Width + Col];
            __syncthreads();
            for (int k = 0; k < TILE_WIDTH; ++k) {
                  Pvalue += Mds[ty][k] * Nds[k][tx];
            }
            __syncthreads();
      }
      d_P[Row*Width + Col] = Pvalue;
}
```

Q.5 (20 Points) Consider performing a 1D convolution on an array of size n with a mask of size m:

a) How many halo cells are there in total?

b) How many multiplications are performed if halo cells are treated as multiplications (by 0)?

c) How many multiplications are performed if halo cells are not treated as multiplications?