## Tiled Matrix Multiplication

## Questions

1. How many floating operations are being performed in your matrix multiply kernel? Explain.

Outside the for loop we have one initialization and one assignment, and inside the for loop loop we have  $(2 * TILE\_WIDTH + 2) * numAColumns / TILE\_WIDTH flops, so we end up performing <math>2 * numARows * numBColumns * numAColumns FLOPs.$ 

2. How many global memory reads are being performed by your kernel? Explain.

The two if-else statements at the beginning of the for loop read the global memory twice, so one thread performs 2 \* ceil(numAColumns / TILE\_WIDTH) global memory reads. Thus, the kernel performs 2 \* numARows \* numAColumns \* numBColumns global memory reads.

3. How many global memory writes are being performed by your kernel? Explain.

We have one write at the end of the kernel function, so the kernel performs numARows\*numBColumns global memory writes.

4. Describe what further optimizations can be implemented to your kernel to achieve a performance speedup.

We can optimize the block size to be a multiple of the warp size (32) and use a larger TILE\_WIDTH to reduce the number of global memory reads and writes.

5. Compare the implementation difficulty of this kernel compared to the previous MP. What difficulties did you have with this implementation?

I had trouble keeping track of which matrix width to use for setting up the edge cases for the matrix multiplication. In addition, the number of threads per block had to match the tile width!

6. Suppose you have matrices with dimensions bigger than the max thread dimensions. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication in this case.

Using grid-stride loops (link),

Algorithm 1: Grid-stride loop for tiled matrix multiplication

## **OUTPUT**

Figure 1: TiledMatrixMultiplication\_Solution output