Algorithm Design

vecMax

The provided algorithm computes a partial max for a block of data. This algorithm is not efficient as each thread grabs data from a different region of memory. We optimize this algorithm by coalescing data; that is, we make each thread grab a memory location adjacent to the previous thread's memory access. This algorithm has a complexity of O(n) due to a for loop with an exit condition relative to the input size.

matMult

This matrix multiplication algorithm breaks the matrix into blocks. For each block, a product is computed for each row and column and stored in an accumulator. Once the product is computed, it is stored into global memory. We improve this algorithm using tiling. With tiling, we break the matrix up into 4 "tiles". Then, each thread computes a product for each tile, allowing us to coalesce memory accesses.

Experimental Design

Experimental Conditions

I used the Bonito fish machine for this experiment. This machine uses an Nvidia 1060 GTX 6GB.

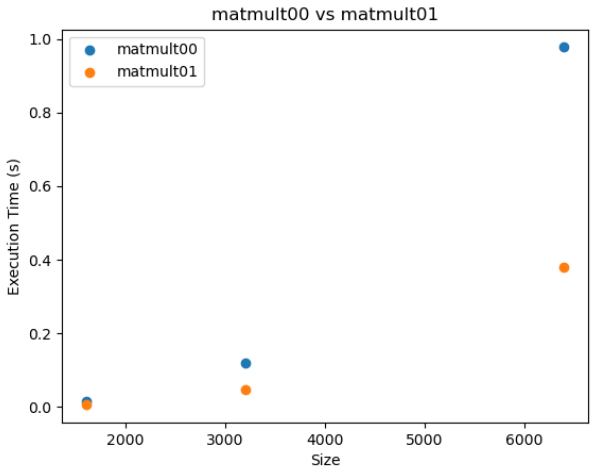
Experimental Results

vecMax

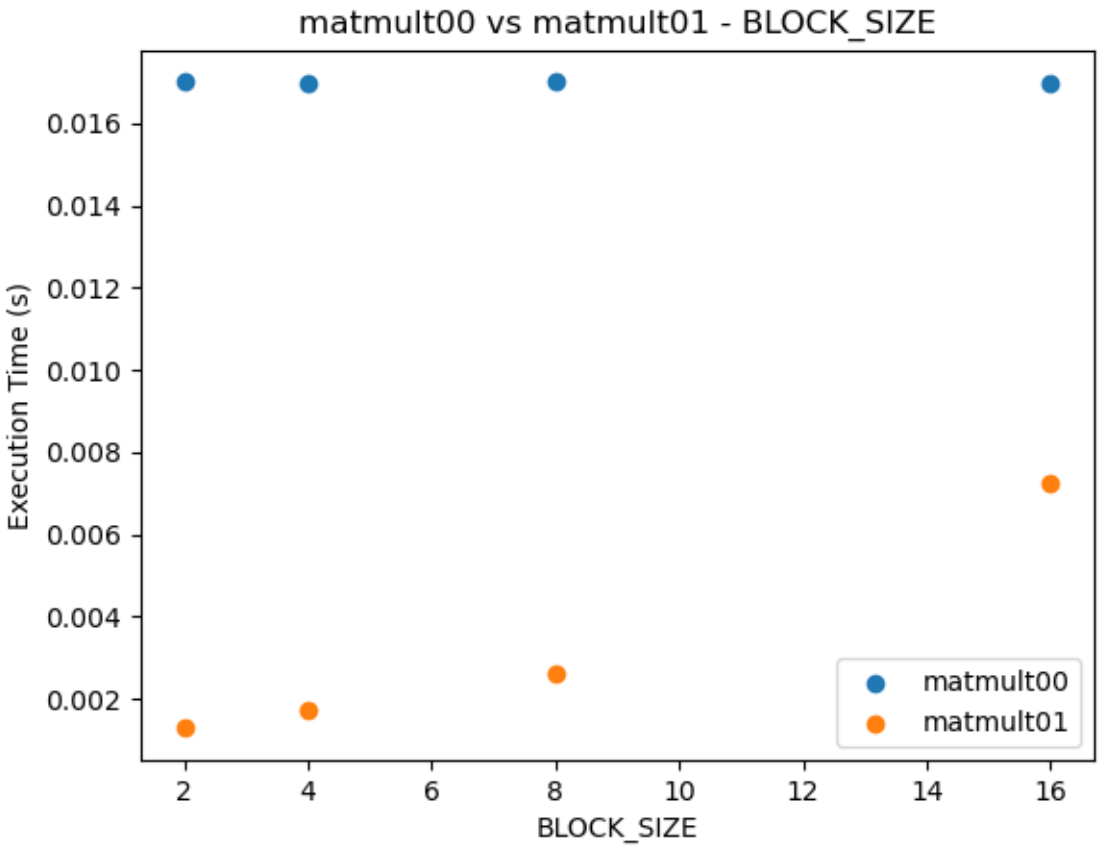
With our improvements in memory access, our CUDA implementation of vecMax has a speed up of approximately 856.80 percent! The mean execution time for the uncoalesced program was 0.3644 seconds, and the mean execution time for the coalesced version is 0.04253 seconds. I did not include a graph as there is only these three data points, collected from 16 total trials.

matMult

For matMult, we can see a drastic improvement in execution time with memory coalescing. Below is a graph where we see that as the matrix size increases, the speedup between the uncoalesced version and coalesced becomes greatest (more than 2x).



Next, we investigate how block_size (the number of threads per block) affects the execution time. For this solution, I was unable to change the block_size of our uncoalesced version without breaking the program, so below we see the results for matmult00 with a constant block size along with matmult01 with a varying block_size. We see that as our block_size decreases (while maintaining a constant matrix size) we see an improvement in execution time.



Conclusion

After all experiments, we can conclude that improving locality for CUDA programs through memory coalescing results in drastic speed improvements. As a rule of thumb, we can maximize the benefits of memory coalescing by using larger problem sizes and smaller block_size's. In the future, I would like to test the change in block_size with different sizes of matrices to come to a safer conclusion regarding BLOCK_SIZE and execution time.