Roy Liu

Feb. 20th, 2020

Problem 1:

1.a Implement your own basic (naive) CUDA implementation of Matrix Multiply on GPU. Run your example with increasing Matrix Sizes (100*100, 1000*1000, 10000*10000), with thread block dimension of (16,16). Plot your execution time (with and without memory transfer). (30pts)

Answer:

I wrote two programs named *MM_cpu .c* and *MM_gpu1.cu* to calculate the run time with three loops sizes, calculating time by using C's *sys/time.h* library.

The result shows cpu's run time increases with loop size's inflation significantly. Contributed by the SIMD method and big amount of cores, GPU's run time trend is relatively flat. Furthermore, the run time of GPU without data transfer calculation drops down further.

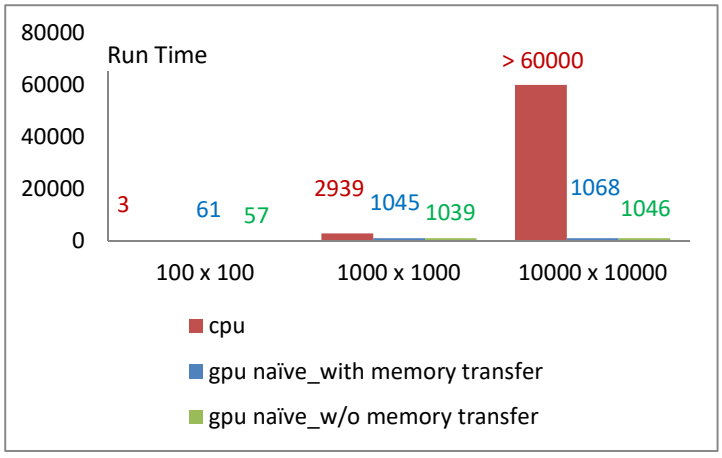

Fig. 1 Matrix multiplication runs with CPU and GPU

Problem 2:

Repeat problem 1, this time try to use memory coalescing to enhance the memory access

efficiency and reduce memory divergence. Plot your execution time (with and without memory transfer). (20pts)

Answer:

In this experiment, I revised the codes in Kernel, reflecting the coalescing algorithm in the program named *MM_gpu1.cu.* , the matrix is transposed. With this optimization, GPU combines (coalesces) the memory access across the thread in SM, the ratio of data using increased, parallel executing threads are going through more contiguous memory access. As the result, the run time drops down, Fig 2. Depicts the result stated above.

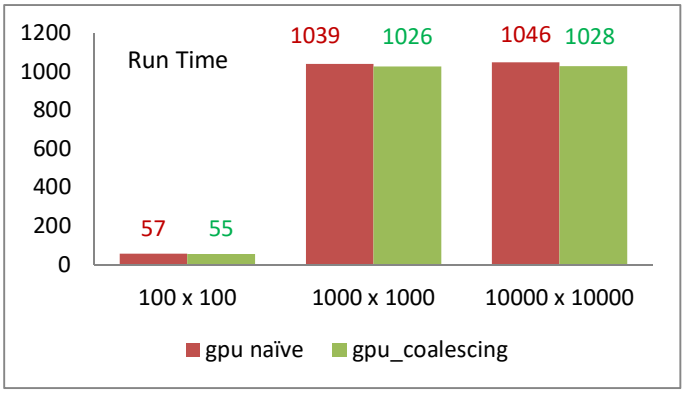

Fig. 2 Matrix multiplication runs with coalescing algorithm

Problem 3:

Repeat problem 1 (without memory coalescing), this time try to use shared memory with tiling. Run your example with increasing Matrix Sizes (100*100, 1000*1000, 10000*10000), with thread block dimension of (16,16). Plot your execution time (with and without memory transfer) and compare the results against your baseline implementation (in problem 1). (40pts)

Answer:

In this experiment, I revised the code (program named *MM_gpu3.cu*) to reflect the optimization of tiling. I declared the shared memory, tiled the matrix by transposing with strides. With this algorithm, during the matrix multiplying, the tiled matrixes decrease the data transportation between local memory and global memory. Meanwhile the data being used in shared memory means the benefit of shared memory's low latency is leveraged. As the result, the run time drops down as expected. Fig3. shows run time improvement.

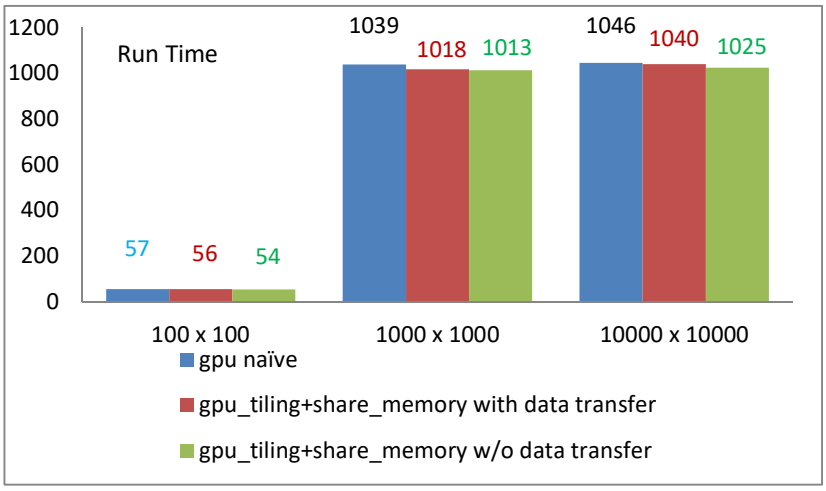

Fig. 3 Fig. 2 Matrix multiplication runs with tiling algorithm

Problem 4:

Repeat problem 3, this with memory coalescing. Plot your execution time (with and without memory transfer) and compare the results against your implementation in problem 3. (10pts)

Answer:

In this experiment, above two algorithms are combined, so run time drops down further under all job sizes' conditions. The code in program is *MM_gpu4.cu*, the performance improvement is presented as below figure.

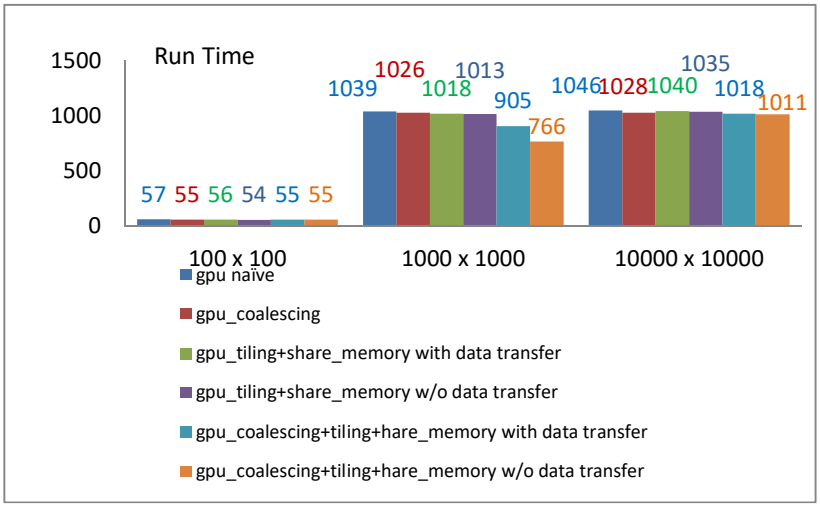

Fig. 4 Fig. 2 Matrix multiplication runs with coalescing and tiling algorithms