Name: Alan Manuel Loreto Cornidez
Professor: Dr. Ali Akoglu
Class: ECE 560 | High Performance Computing

Matrix Multiplication Performance Analysis

Objective: Practice running configuration sweeping experiments, data collection and profiler based performance analysis.
Use the following 4 test cases generated by the "dataset_generator.cpp"

Case 1: A is 2048x2048, B is 2048x2048

Case 2: A is 2051x2051, B is 2051x2051

Case 3: A is 4096x2048, B is 2048x4096

Case4: A is 4096x2047, B is 2047x4095

For each **version** (basic and tiled matrix multiplication)

For each **test case** (4 cases)

For each **thread block** configuration of 4x4, 8x8, 16x16 and 32x32

- Collect computation time (time spent on kernels excluding time spent on memory allocations, data transfer from host to device and device to host) data.
- Fill the timing table below for each test case and thread block configuration.

Notes: I used my personal GPU for the data that was gathered in this analysis. Here are the statistics for that GPU:

- Device Name: NVIDIA GeForce GTX 1070 Ti
- Computational Capabilities: 6.1
- Maximum global memory size: 8484618240
- Maximum constant memory size: 65536
- Maximum shared memory size per block: 49152
- Maximum block dimensions: 1024 x 1024 x 64
- Maximum grid dimensions: 2147483647 x 65535 x 65535

Figure 1: Table with execution time data for the basic implementation.

Execution Time Data (Basic Matrix Multiply)				
	Thread Block Configuration			
Test Case	4x4	8x8	16x16	32x32
Case 1	174733142.4	98605488.3	54647719	42561675.5
Case 2	163871897.3	115224000.8	70247029.8	51800055.8
Case 3	606180807.3	364963587.9	222337779.3	166903859.3
Case 4	602248576.7	389753635.8	258895105.2	197120770.9

Figure 2: Table with execution time data for the tiled implementation.

Execution Time Data (Tiled Matrix Multiply)				
	Thread Block Configuration			
Test Case	4x4	8x8	16x16	32x32
Case 1	110045562.2	29715972.8	18735894.1	17075206.4
Case 2	125027861	29934003.8	19862011.6	18194286.1
Case 3	422999044.1	122230819.1	74564703.4	67435190.4
Case 4	438072333.7	123786539.6	73923262.6	67207089.8

Figure 3: Line Graph Basic Matrix Multiplication

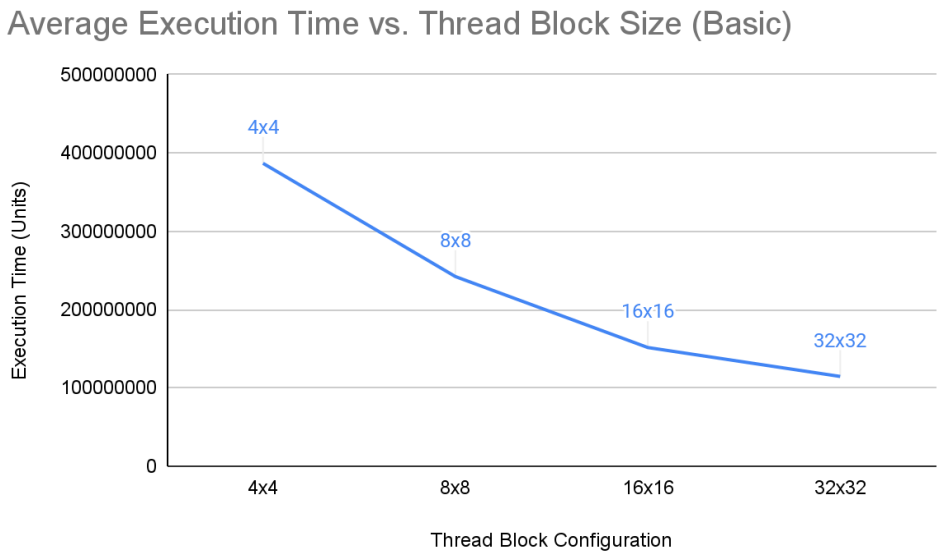


Figure 4: Line Graph Tiled Matrix Multiplication

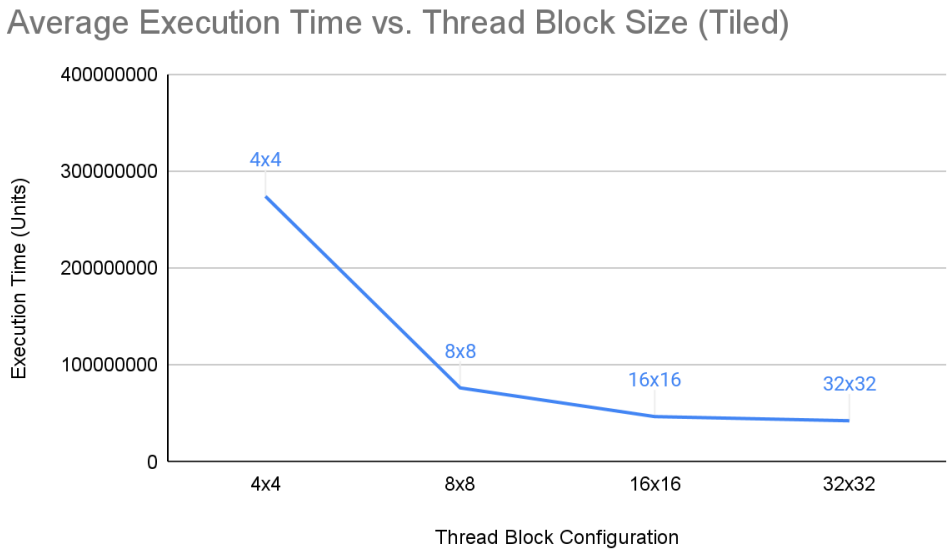
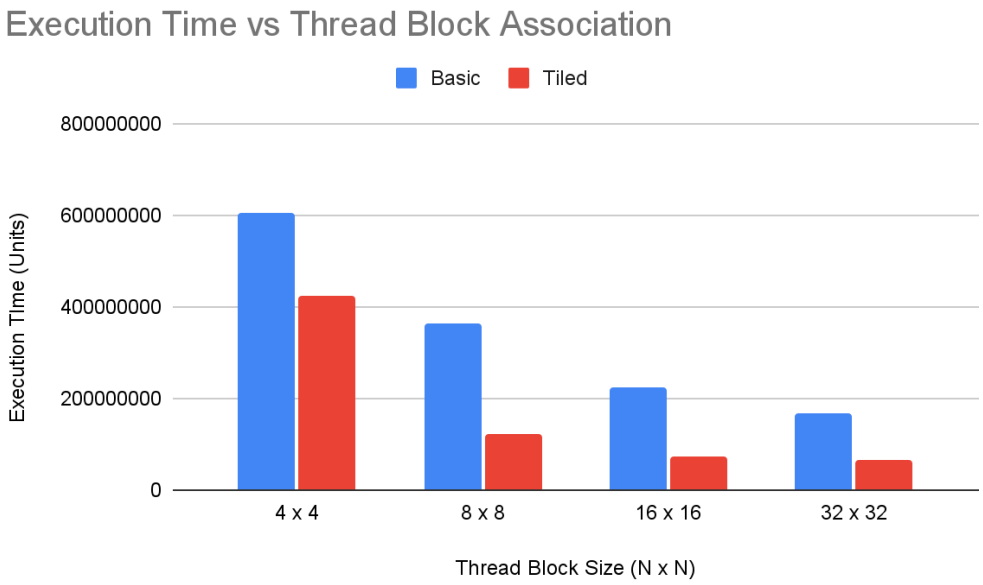


Figure 5: Bar Graph with Tile vs Basic Implementation with respect to block size.



Discussion

1. Fix input size, discuss execution time trends with respect to change in thread block configuration for basic and tiled solutions individually. What is the optimal thread block configuration for each version? Discuss why a specific block size results with best/worst performance for each version? Does optimal block size depend on the input size?

When an input is of a larger size, optimizations like the tiled method can help improve execution time. Regardless of whether tiling and shared memory were used to optimize execution, larger thread blocks were always faster than using smaller thread blocks.

2. Discuss execution time trends among square matrices only for basic and tiled versions individually.

Basic

With the basic implementation, a larger block size pretty much always led to greater increases in performance for my GPU. It did not appear like my GPU was being stressed to its full capabilities. As tile size grew it was still possible to see mostly steady improvements.

Tiled

The tiled implementation is unique, one of the details to note is that the improvement from a small block size to a larger block size results in a very large performance improvement. After that initial jump, execution time begins to get shorter, but the improvements become small very quickly.

3. Discuss execution time trends among square and non-square matrices for basic and tiled versions individually.

Basic

Comparing execution times for the basic implementation we can see that Case 1 and Case 2 (which are square matrix operations) execute much faster than Case 3 and Case 4 (which are not square matrix operations). While the output of Cases 1 and 2 are about half the size of the outputs of Cases 3 and 4, the time was less than half.

$$\frac{Average_{basic}\,(Case\ 1\ \&\ Case\ 2)}{Average_{basic}\,(Case\ 2\ \&\ Case\ 4)} = 0.274$$

Tiled

The same pattern can be observed when examining the time data for the tiled version execution time. We can see that the comparison between the square and non-square matrices is nearly identical to the basic implementation.

$$\frac{Average_{tiled}\,(Case\ 1\ \&\ Case\ 2)}{Average_{tiled}\,(Case\ 2\ \&\ Case\ 4)} = 0.265$$

4. Discuss performance of basic and tiled solution versions for a given problem size and across different problem sizes. Include any additional plot that will help your claim.

These tests were conducted with a 4x4 block size. Tests were conducted 10 times and the average was taken for the execution time.

Execution for Different Square Matrices			
Matrix Size	Tiled Execution Time	Basic Execution Time	Tiled Time / Basic Time
32 x 32	26562	29962	0.8865343858
512 x 512	1281234	2143257	0.5977975019
1024 x 1024	12413871	20684526	0.600152573
2048 x 2048	104927121	163443634	0.6419774108

As examined in the data, when data is small, the execution time is about 90%. As the data set grows, the advantages of the tiled version begin to shine, that is, until the data set grows larger, there is actually less of an advantage between the two algorithms, albeit, the tiled method is still faster, however, the gaps in execution speed thin.

5. Can the basic version execute faster than the tiled version for the same problem size? You need to use dataset_generator to change the problem size and show execution time performance.

I created test cases for small outputs (thread block size was 4). The tests were run ten times and then the average of the execution time was taken. In cases 2 and 3 you can see that execution time is actually slower for the tiled multiplication than it is for the basic implementation. Performance is slower on the tiled implementation when the problem size is small because of the extra time spent on copying memory to the shared registers.

Tile Vs Basic (Small Matrix Size)		
	Tiled Times	*Basic Times*
Case 1 (2,2,2)	*24991*	*25799*
Case 2 (4,4,4)	***29152***	***24947***
Case 3 (1,1,1)	***29438***	***26973***

The execution time for tiled implementations is slower for small matrices, that is, kernels, where the matrices are smaller. Ultimately, your application determines what the optimal implementation is. In this case, the tiled version of the algorithm is slower than the basic implementation.

6. Can the basic version run faster than the tiled version for the same block size? Your discussions need to be based on your experimental results.

Yes, as seen in the chart above, all of the thread clock sizes were the same. When the array sizes were small and square matrices of size 4 x 4 and 1 x 1, the execution was faster with the basic matrix multiplication than with the tiled implementation.