CSCE 735 Major Project

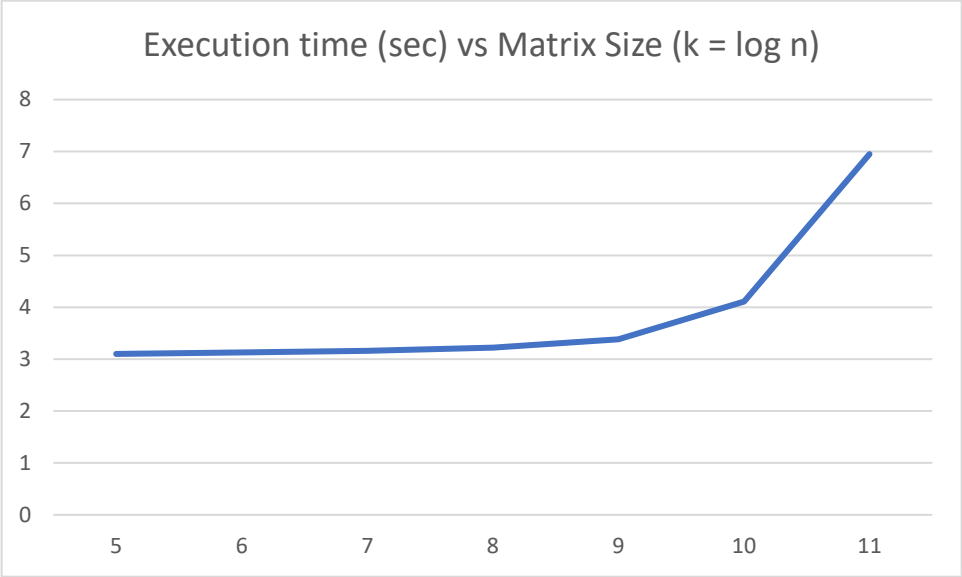
NAME: Fardeen Hasib Mozumder

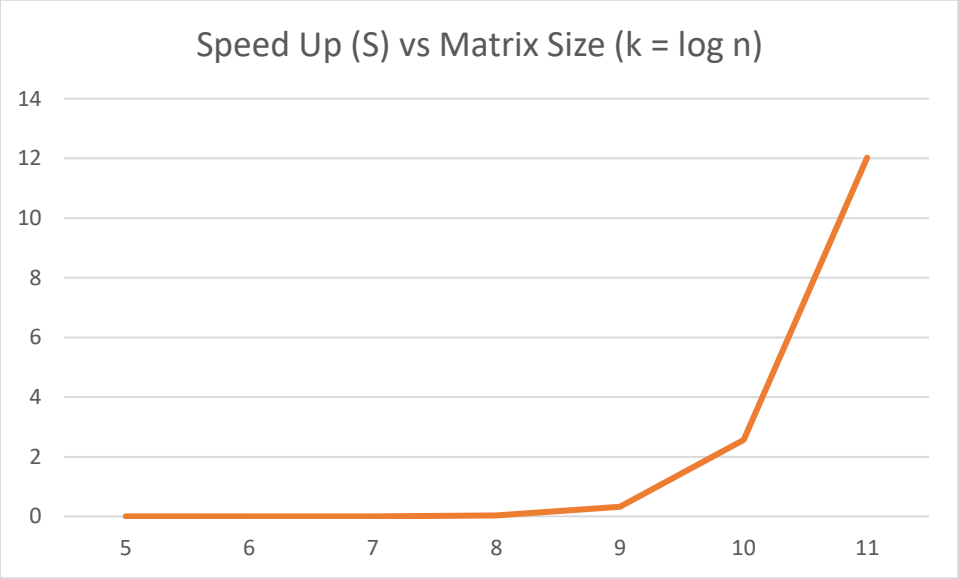
1.

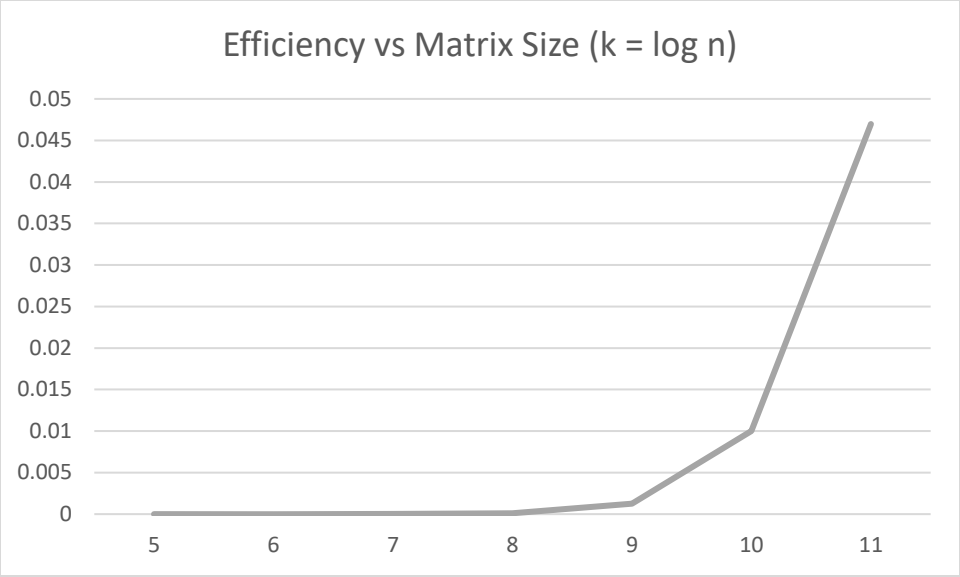
For the project, I implemented Strassen matrix multiplication on CUDA. My implementation can be found in Strassen_cuda.cu file.

2.

Here we can see the execution time vs matrix size, speed up vs matrix size and efficiency vs matrix size plots, when k_bar was fixed to 5.







Comment: We can see that overall parallel performance (speed up and efficiency) increases as we increase the size of matrix. The reason is twofold. First, the advantage of using Strassen matrix multiplication over naïve matrix multiplication becomes noticeable when we use bigger size matrixes. Second, introducing a lot of threads creates a lot of memory read/write and associated serial operations which ultimately outweighs the performance gain specially for smaller size matrixes.

The effect of varying k_bar (terminal matrix size) is shown in the following,

k_bar=4
Matrix Size = 16, Terminal Matrix Size = 1, error = 0, time_strassen (sec) = 0.58675, time_naive = 0.00003
Matrix Size = 32, Terminal Matrix Size = 2, error = 0, time_strassen (sec) = 0.56508, time_naive = 0.00019
Matrix Size = 64, Terminal Matrix Size = 4, error = 0, time_strassen (sec) = 0.56588, time_naive = 0.00149
Matrix Size = 128, Terminal Matrix Size = 1, error = 0, time_strassen (sec) = 147.28667, time_naive = 0.01204
Matrix Size = 256, Terminal Matrix Size = 16, error = 0, time_strassen (sec) = 0.61030, time_naive = 0.09538
Matrix Size = 512, Terminal Matrix Size = 32, error = 0, time_strassen (sec) = 0.69695, time_naive = 1.07465
Matrix Size = 1024, Terminal Matrix Size = 64, error = 0, time_strassen (sec) = 1.09016, time_naive = 10.53764
Matrix Size = 2048, Terminal Matrix Size = 128, error = 0, time_strassen (sec) = 2.60870, time_naive = 83.63550

k_bar=5
Matrix Size = 32, Terminal Matrix Size = 1, error = 0, time_strassen (sec) = 3.10676, time_naive = 0.00019
Matrix Size = 64, Terminal Matrix Size = 2, error = 0, time_strassen (sec) = 3.13691, time_naive = 0.00150
Matrix Size = 128, Terminal Matrix Size = 4, error = 0, time_strassen (sec) = 3.16693, time_naive = 0.01202
Matrix Size = 256, Terminal Matrix Size = 8, error = 0, time_strassen (sec) = 3.22621, time_naive = 0.09575
Matrix Size = 512, Terminal Matrix Size = 16, error = 0, time_strassen (sec) = 3.38146, time_naive = 1.08911
Matrix Size = 1024, Terminal Matrix Size = 32, error = 0, time_strassen (sec) = 4.10997, time_naive = 10.54462
Matrix Size = 2048, Terminal Matrix Size = 64, error = 0, time_strassen (sec) = 6.89923, time_naive = 83.57262

k_bar=7
Matrix Size = 128, Terminal Matrix Size = 1, error = 0, time_strassen (sec) = 146.01805, time_naive = 0.01202
Matrix Size = 256, Terminal Matrix Size = 2, error = 0, time_strassen (sec) = 146.84802, time_naive = 0.09564
Matrix Size = 512, Terminal Matrix Size = 4, error = 0, time_strassen (sec) = 148.24294, time_naive = 1.11844
Matrix Size = 1024, Terminal Matrix Size = 8, error = 0, time_strassen (sec) = 152.69666, time_naive = 10.57479
Matrix Size = 2048, Terminal Matrix Size = 16, error = 0, time_strassen (sec) = 159.62930, time_naive = 83.60043

k_bar=8
Matrix Size = 256, Terminal Matrix Size = 1, error = 0, time_strassen (sec) = 1020.78421, time_naive = 0.09571
Matrix Size = 512, Terminal Matrix Size = 2, error = 0, time_strassen (sec) = 1020.58189, time_naive = 1.07590
Matrix Size = 1024, Terminal Matrix Size = 4, error = 0, time_strassen (sec) = 1036.24879, time_naive = 10.54013
Matrix Size = 2048, Terminal Matrix Size = 8, error = 0, time_strassen (sec) = 1068.02040, time_naive = 83.54124

Comment: We can see that for bigger values of k_bar, the execution time is higher and selecting smaller k_bar results in faster execution time. This is because selecting bigger k_bar results in performing matrix multiplication on smaller size matrixes one at a time using cuda device, and so hinders utilizing the threads properly. Also, combining the results together after multiplication on small chunks to get the final output requires a lot of memory read/write operations which can significantly reduce parallel performance. Whereas, selecting smaller k_bar values allow us to do matrix multiplication on bigger matrixes one at a time on cuda devices which significantly improves parallel performance while implementing Strassen algorithm.

Instruction to run the code:

Load the cuda intel 2020a version. For some unknown reason while using latest CUDA version, I was getting weird results. Using an older version of cuda fixed my issue.

➔ module load intelcuda/2020a

Then, compile the Strassen_cuda.cu using the following command,

➔ nvcc -o strassen_cuda.exe strassen_cuda.cu

Then run the executable using

➔ ./strassen_cuda.exe k k_bar
Where, k>k_bar>1

For performance improvement, I implemented cached (shared memory) matrix multiplication on cuda device (GPU) instead of the naïve (global memory) version, which significantly improves the runtime of the Strassen algorithm.

I took help from the following tutorial and repos to complete the project:

1. (71) Programming with CUDA: Matrix Multiplication - YouTube
2. CoffeeBeforeArch/cuda_programming: Code from the "CUDA Crash Course" YouTube series by CoffeeBeforeArch (github.com)
3. spectre900/Parallel-Strassen-Algorithm: Parallelizing Strassen's matrix multiplication using OpenMP, MPI and CUDA. (github.com)