CSCE 735 Parallel Computing – Final Project

832001192
Chun-Sheng Wu

1. Develop parallel Strassen's recursive algorithm with OpenMP or CUDA

```
[jinsonwu@grace5 FP]$ module load intel
[jinsonwu@grace5 FP]$ icpc -qopenmp -o omp_strassen.exe omp_strassen.cpp
[jinsonwu@grace5 FP]$ ./omp_strassen.exe 8 1
Matrix Size (nxn) = 256, Threads = 2, error = 0, time_strassen (sec) =  0.04065, time_naive =  0.01458
[jinsonwu@grace5 FP]$ ./omp_strassen.exe 8 2
Matrix Size (nxn) = 256, Threads = 4, error = 0, time_strassen (sec) =  0.03351, time_naive =  0.01736
[jinsonwu@grace5 FP]$ ./omp_strassen.exe 10 1
Matrix Size (nxn) = 1024, Threads = 2, error = 0, time_strassen (sec) =  1.00203, time_naive =  3.24115
[jinsonwu@grace5 FP]$ ./omp_strassen.exe 10 2
Matrix Size (nxn) = 1024, Threads = 4, error = 0, time_strassen (sec) =  0.90792, time_naive =  3.06703
[jinsonwu@grace5 FP]$ ./omp_strassen.exe 11 3
Matrix Size (nxn) = 2048, Threads = 8, error = 0, time_strassen (sec) =  7.05781, time_naive = 34.85869
```

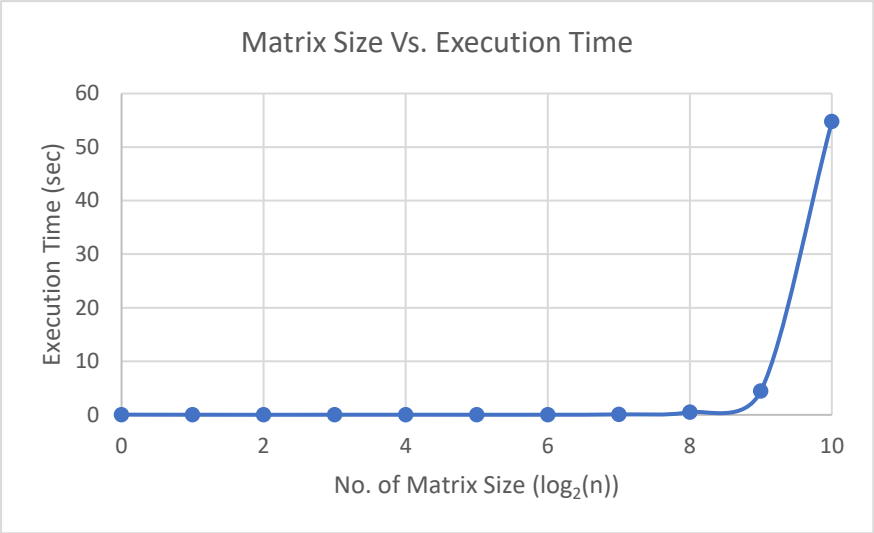
Execution Status of omp_strassen.cpp

2. Brief introduction about design, compile, and execution. Do comparison with different matrix size and explore the impact it brings to the calculation

Since the raised matrix size is proposed to put exponential computing loading on machine, I decided to utilize recursive methodology to simplify the resource modulation with parallel execution. Machine could dynamically perform calculations assigned by threads. Additionally, I employed part of naïve matrix multiplication as the basement. Through the experiment below, we could observe that matrix containing 32x32 elements had the best efficiency to perform exhaustive matrix multiplication through Strassen algorithm (basement = 2x2 matrix like the example mentioned in the explanation pdf). Threads are fixed at 32 due to the previous experiment results and hardware setting of Grace in a single board. Besides, to better and quicker observe the impact parallel execution brought on Strassen algorithm, I set the maximum value in each input matrix (A&B) at 1024. This could be changed by modifying the #define MAX_ELEMENT_VALUE on the top of the code.

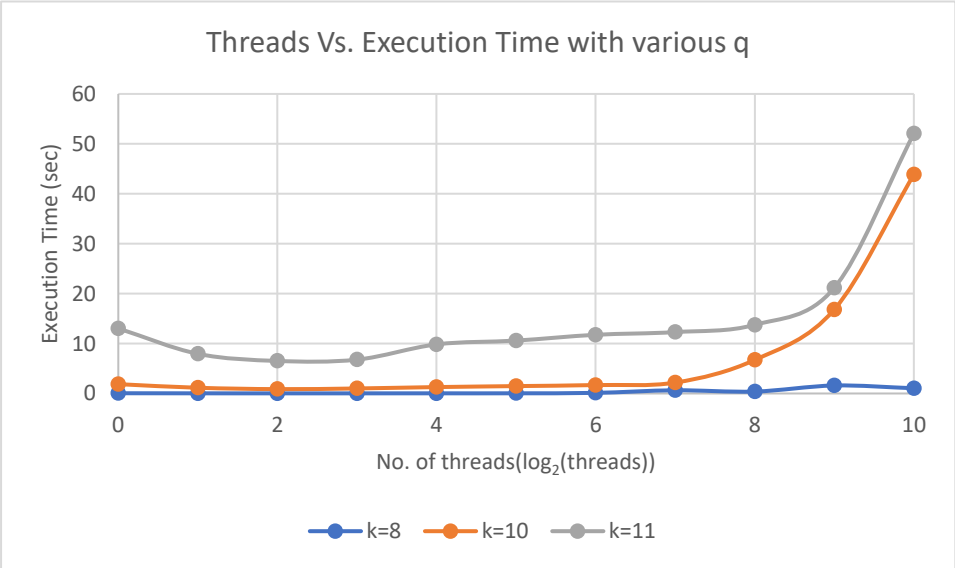
Basement Matrix Size	Matrix Size	Threads	Execution Time (sec)
2	1	32	0.03441
2	2	32	0.01598
2	4	32	0.01066
2	8	32	0.01663
2	16	32	0.01938
2	32	32	0.01301
2	64	32	0.02049
2	128	32	0.08038
2	256	32	0.49424
2	512	32	4.44515
2	1024	32	54.74138

Statistics of preliminary experiment deciding basement matrix size

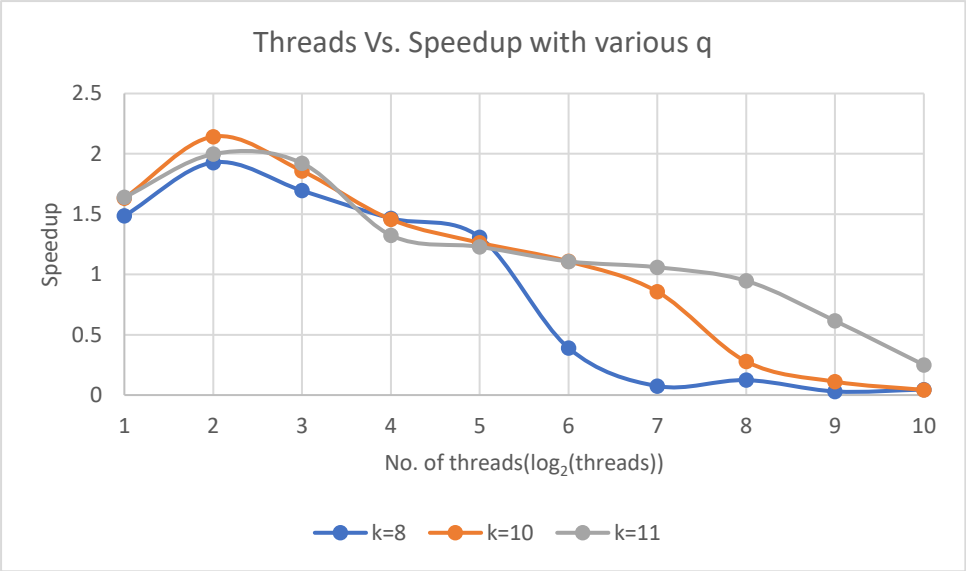


Execution time in various matrix size with basement matrix size = 2x2

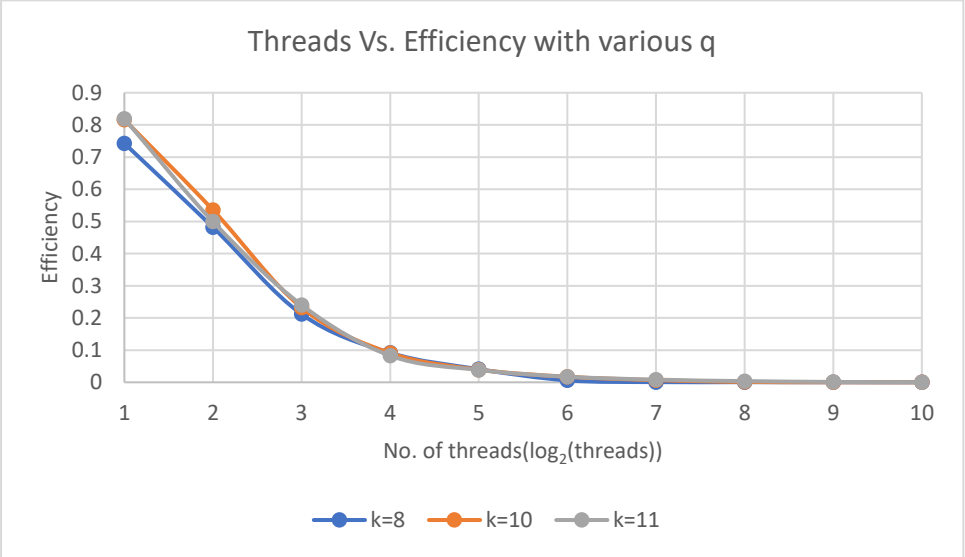
Followed by is the performance comparison in various thread numbers with basement matrix = 32x32. I experimented different matrix size with k = 8, 10, 11 (256, 1024, 2048) respectively. The maximum of matrix size at 2048x2048 was because matrix size beyond this setting encountered out-of-memory when execution. Due to the consequence of basement matrix size experiment, matrix size should be above 64 (k=6) to exactly know the benefit of parallel calculation with Strassen's algorithm. Therefore, k=8 was chosen as the lower boundary in the comparison.



Execution time comparison



Speedup comparison



Efficiency comparison

As expected, higher k required longer time to finish multiplication. 1024x1024 matrix (k=10) obtained slightly better speedup and efficiency than 256x256 & 2048x2048 matrix (k=8 & 11). It achieved the best speedup with threads=4 and efficiency with threads=2. To consider a large-scale multiplication, we can apply divide and conquer method to split a huge matrix into a plethora of 1024x1024 matrices. Subsequently, implement parallel computing on microprocessors to obtain the desired optimization.

Lastly, I modified the input format and made it just like the previous assignments. Command "./omp_strassen.exe k q", where k is matrix size (2^k) and q is number of threads (2^q), after compiling (icc or icpc). Calculations through Strassen's algorithm and naïve will be performed and tested in the following.

```
Matrix Size = 1, Threads = 32, error = 0, time_strassen (sec) =  0.03441, time_naive =  0.00000
Matrix Size = 2, Threads = 32, error = 0, time_strassen (sec) =  0.01598, time_naive =  0.00000
Matrix Size = 4, Threads = 32, error = 0, time_strassen (sec) =  0.01066, time_naive =  0.00000
Matrix Size = 8, Threads = 32, error = 0, time_strassen (sec) =  0.01663, time_naive =  0.00000
Matrix Size = 16, Threads = 32, error = 0, time_strassen (sec) =  0.01938, time_naive =  0.00001
Matrix Size = 32, Threads = 32, error = 0, time_strassen (sec) =  0.01301, time_naive =  0.00005
Matrix Size = 64, Threads = 32, error = 0, time_strassen (sec) =  0.02049, time_naive =  0.00029
Matrix Size = 128, Threads = 32, error = 0, time_strassen (sec) =  0.08038, time_naive =  0.00240
Matrix Size = 256, Threads = 32, error = 0, time_strassen (sec) =  0.49424, time_naive =  0.01963
Matrix Size = 512, Threads = 32, error = 0, time_strassen (sec) =  4.44515, time_naive =  0.45058
Matrix Size = 1024, Threads = 32, error = 0, time_strassen (sec) = 54.74138, time_naive =  3.71021
```

Basement matrix size experiment result

```
Matrix Size (nxn) = 1024, Threads = 1, error = 0, time_strassen (sec) =  1.87060, time_naive =  3.31943
Matrix Size (nxn) = 1024, Threads = 2, error = 0, time_strassen (sec) =  1.14543, time_naive =  3.58517
Matrix Size (nxn) = 1024, Threads = 4, error = 0, time_strassen (sec) =  0.87320, time_naive =  3.42664
Matrix Size (nxn) = 1024, Threads = 8, error = 0, time_strassen (sec) =  1.00689, time_naive =  3.33373
Matrix Size (nxn) = 1024, Threads = 16, error = 0, time_strassen (sec) =  1.28484, time_naive =  3.58385
Matrix Size (nxn) = 1024, Threads = 32, error = 0, time_strassen (sec) =  1.48110, time_naive =  3.32969
Matrix Size (nxn) = 1024, Threads = 64, error = 0, time_strassen (sec) =  1.68800, time_naive =  3.58438
Matrix Size (nxn) = 1024, Threads = 128, error = 0, time_strassen (sec) =  2.18229, time_naive =  3.59279
Matrix Size (nxn) = 1024, Threads = 256, error = 0, time_strassen (sec) =  6.72340, time_naive =  3.81985
Matrix Size (nxn) = 1024, Threads = 512, error = 0, time_strassen (sec) = 16.83035, time_naive =  3.69399
Matrix Size (nxn) = 1024, Threads = 1024, error = 0, time_strassen (sec) = 43.84639, time_naive =  3.55159

Matrix Size (nxn) = 256, Threads = 1, error = 0, time_strassen (sec) =  0.04793, time_naive =  0.01741
Matrix Size (nxn) = 256, Threads = 2, error = 0, time_strassen (sec) =  0.03227, time_naive =  0.01734
Matrix Size (nxn) = 256, Threads = 4, error = 0, time_strassen (sec) =  0.02486, time_naive =  0.01813
Matrix Size (nxn) = 256, Threads = 8, error = 0, time_strassen (sec) =  0.02828, time_naive =  0.01962
Matrix Size (nxn) = 256, Threads = 16, error = 0, time_strassen (sec) =  0.03271, time_naive =  0.01948
Matrix Size (nxn) = 256, Threads = 32, error = 0, time_strassen (sec) =  0.03661, time_naive =  0.01993
Matrix Size (nxn) = 256, Threads = 64, error = 0, time_strassen (sec) =  0.12281, time_naive =  0.01745
Matrix Size (nxn) = 256, Threads = 128, error = 0, time_strassen (sec) =  0.64375, time_naive =  0.04742
Matrix Size (nxn) = 256, Threads = 256, error = 0, time_strassen (sec) =  0.38676, time_naive =  0.08760
Matrix Size (nxn) = 256, Threads = 512, error = 0, time_strassen (sec) =  1.59594, time_naive =  0.09833
Matrix Size (nxn) = 256, Threads = 1024, error = 0, time_strassen (sec) =  1.04798, time_naive =  0.19259

Matrix Size (nxn) = 2048, Threads = 1, error = 0, time_strassen (sec) = 13.02552, time_naive = 33.59783
Matrix Size (nxn) = 2048, Threads = 2, error = 0, time_strassen (sec) =  7.94801, time_naive = 34.93131
Matrix Size (nxn) = 2048, Threads = 4, error = 0, time_strassen (sec) =  6.52178, time_naive = 38.99442
Matrix Size (nxn) = 2048, Threads = 8, error = 0, time_strassen (sec) =  6.77623, time_naive = 39.29542
Matrix Size (nxn) = 2048, Threads = 16, error = 0, time_strassen (sec) =  9.83172, time_naive = 39.38340
Matrix Size (nxn) = 2048, Threads = 32, error = 0, time_strassen (sec) = 10.59821, time_naive = 35.14001
Matrix Size (nxn) = 2048, Threads = 64, error = 0, time_strassen (sec) = 11.75660, time_naive = 34.03263
Matrix Size (nxn) = 2048, Threads = 128, error = 0, time_strassen (sec) = 12.29747, time_naive = 33.45201
Matrix Size (nxn) = 2048, Threads = 256, error = 0, time_strassen (sec) = 13.75349, time_naive = 34.59909
Matrix Size (nxn) = 2048, Threads = 512, error = 0, time_strassen (sec) = 21.17387, time_naive = 44.03670
Matrix Size (nxn) = 2048, Threads = 1024, error = 0, time_strassen (sec) = 52.06498, time_naive = 38.64687
```

Comparison with various threads at fixed matrix size