CSCE 735 Fall 2022

Parallel Computing

Major Project

Sriram Chakravarthy

(UIN: 629009746)

Dec 7, 2022

1. Code submitted in zip folder.

Platform: CUDA

Compile command:

nvcc -arch=sm_80 -ccbin=icc -std=c++17 -x cu -lcublas -o strassen_matmul.exe strassen_matmul.cpp

Execute:

./strassen_matmul.exe k k' blockdim2d

Where k -> matrix dimension, k' -> terminal matrix dimension, blockdim2d -> dim of 2d block in terms of number of threads

Example: blockdim2d = 32 => #threads = 32 x 32 = 1024

blockdim2d range : [1, 32]

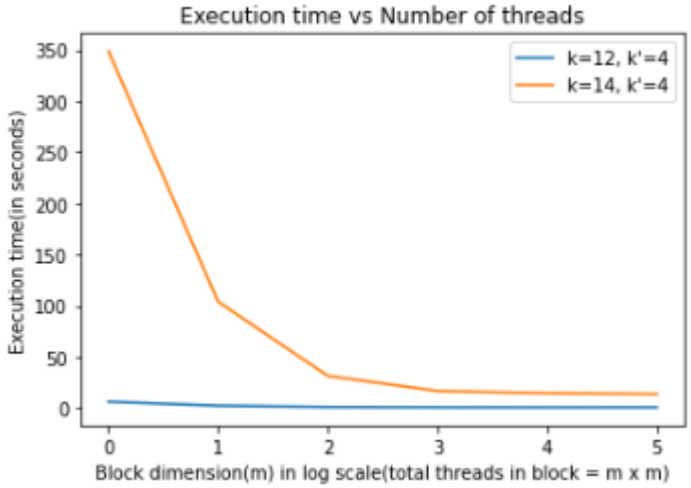
2.

- **Different number of threads(blockdim2d), for k = 12, 14 , keeping k' = 4**

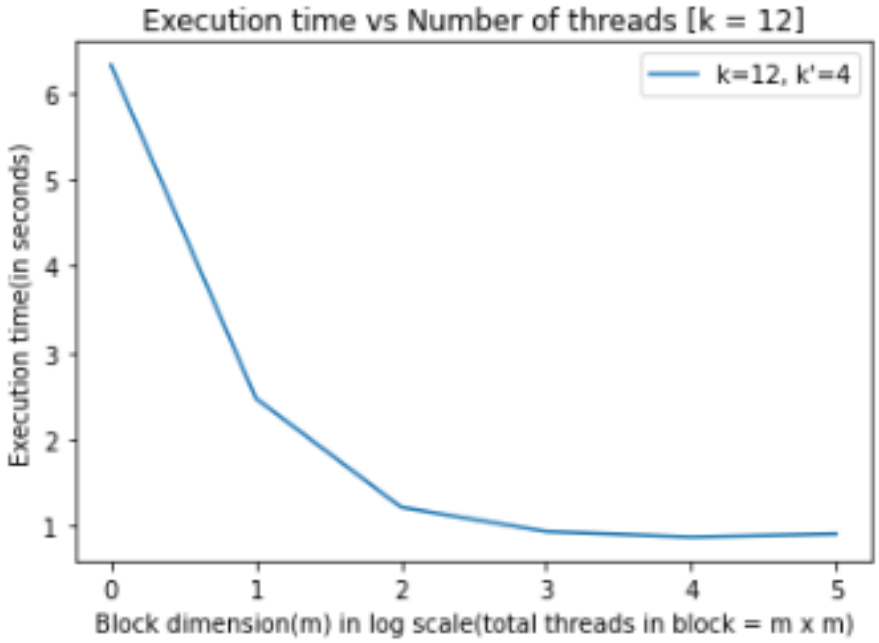
  Results:

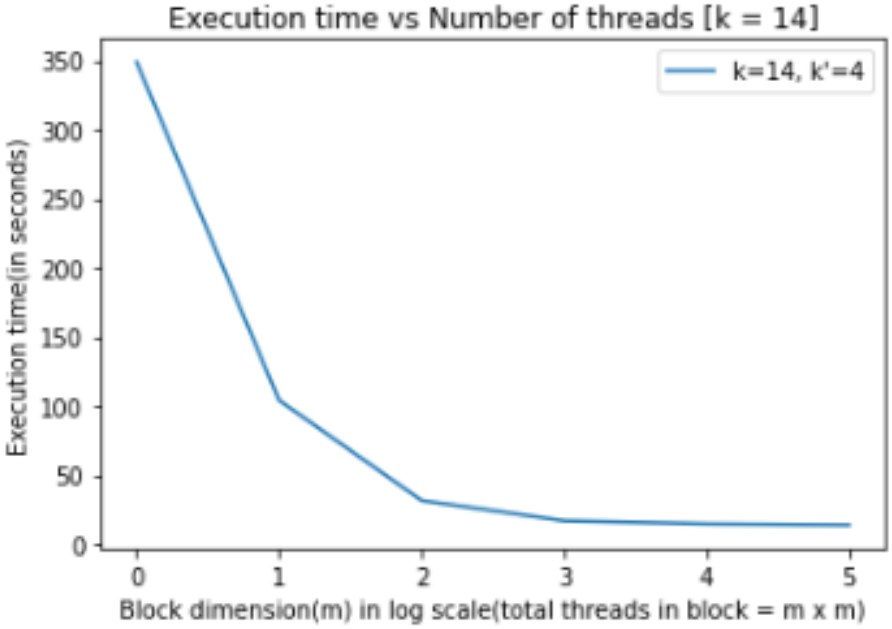
k	Block dimension	Num threads	Execution time (seconds)
12	1	1	6.31
12	2	4	2.47
12	4	16	1.22
12	8	64	0.93
12	16	256	0.87
12	32	1024	0.91
14	1	1	348.777
14	2	4	104.058
14	4	16	31.55
14	8	64	16.86
14	16	256	14.67
14	32	1024	13.87

  a. **Execution times**

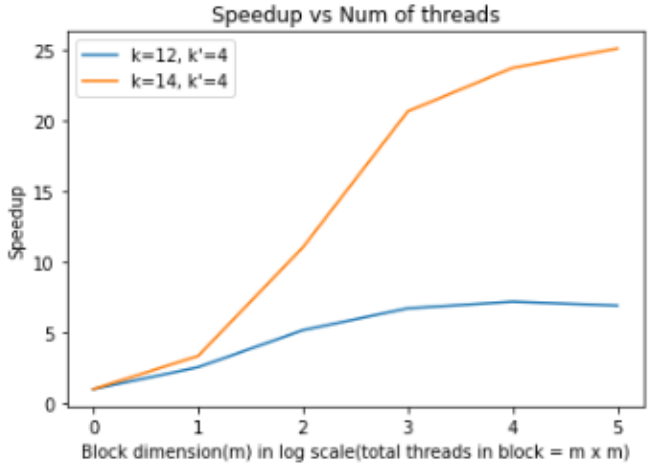


For a given k, the execution time decreases as expected with the number of threads. For k = 14, the execution time is significantly higher than for k =12. Since the variation of execution time is not clear in the same graph, below are the two cases plotted separately.


Execution time vs Number of threads [k = 12]
k=12, k'=4
Execution time(in seconds)
6
5
4
3
2
1
0
1
2
3
4
5
Block dimension(m) in log scale(total threads in block = m x m)


Execution time vs Number of threads [k = 14]
k=14, k'=4
Execution time(in seconds)
350
300
250
200
150
100
50
0
0
1
2
3
4
5
Block dimension(m) in log scale(total threads in block = m x m)

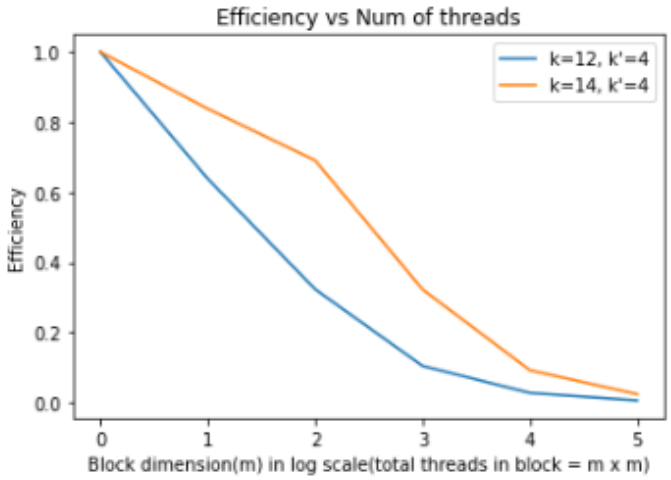
b. Speedup



In both cases of k = 12 and k = 14, I have considered the runtime where thread-block is of dimension (1x1) i.e., one thread as the base case and calculated the speedup and efficiency for other cases accordingly.

The speedup increases with the number of threads as expected. For k = 14, increase in speedup is more visible compared to k = 12 since the matrices are 16 times larger and there is more scope for parallelization.

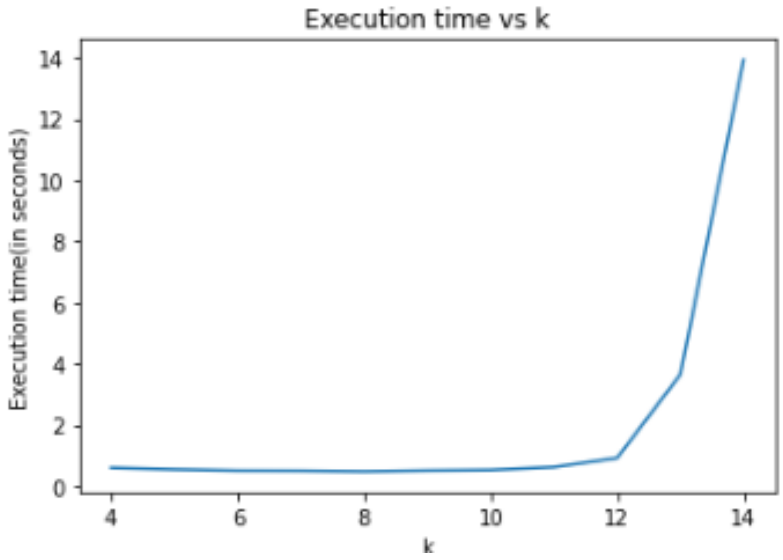
c. Efficiency



Efficiency decreases with increase in number of threads per block in both cases. This shows that the speedup does not scale with the increase in number of threads.

- **Different values of k , keeping k’ = 4 and blockdim2d = 32**
    a. **Execution times**

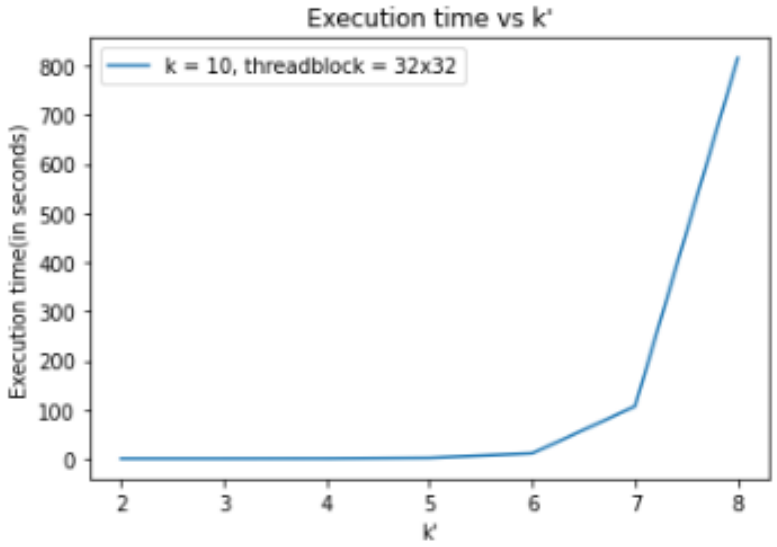
k	k'	Block dimension	Num threads	Execution time (seconds)
4	4	32	1024	0.6
5	4	32	1024	0.546413
6	4	32	1024	0.509474
7	4	32	1024	0.504439
8	4	32	1024	0.482109
9	4	32	1024	0.512853
10	4	32	1024	0.52883
11	4	32	1024	0.622991
12	4	32	1024	0.918945
13	4	32	1024	3.65034
14	4	32	1024	13.9221



The execution time increases as k increases beyond 10, for a given k’ and number of threads per block. For lower values of k, the 1024 threads used won’t significantly help with the parallelization as the matrix sizes are less. At every step beyond k =10, the execution time increases significantly as the matrix dimension increases and there are significantly more computations.

- **Different values of k', keeping k = 10 and blockdim2d = 32**
    a. **Execution times**

k	k'	Block dimension	Num threads	Execution time (seconds)
10	2	32	1024	0.32
10	3	32	1024	0.312
10	4	32	1024	0.319
10	5	32	1024	0.54
10	6	32	1024	1.78
10	7	32	1024	10.573
10	8	32	1024	816.16



The execution time increases as k' increases and approaches k. This shows that the terminal matrix multiplication is quite fast compared to the Strassen recursive computations. This could be due to the additional split, add, merge computations in the Strassen part. As k' increases, the number of levels of Strassen recursive steps increases and results in higher execution times.