CSCE 735 Fall 2023 Parallel Computing

Submitted By: Ashutosh Chauhan
UIN: 232009024

Major Project: Parallelizing Strassen’s Matrix-Multiplication Algorithm

1. (75 points) In this project, you have to develop a parallel implementation of Strassen’s recursive algorithm for computing matrix-multiplications. You should choose one of the following strategies.
a. Develop a shared-memory code using OpenMP, or
b. Develop a CUDA-based code for the GPU.
Your code should compute the product of two matrices of size n x n, where n=2^k , and k is an input to your program. Your code should also accept k’ as an input to allow the user to vary the terminal matrix size.

2. (25 points) Develop a report that describes the parallel performance of your code. You will have to conduct experiments to determine the execution time for different values of k, and use that data to plot speedup and efficiency graphs. You should also experiment with different values of k’ to explore its impact on the execution time.
Discuss any design choices you made to improve the parallel performance of the code and how it relates to actual observations. Include any insights you obtained while working on this project.
Lastly, include a brief description of how to compile and execute the code on platform you have chosen.

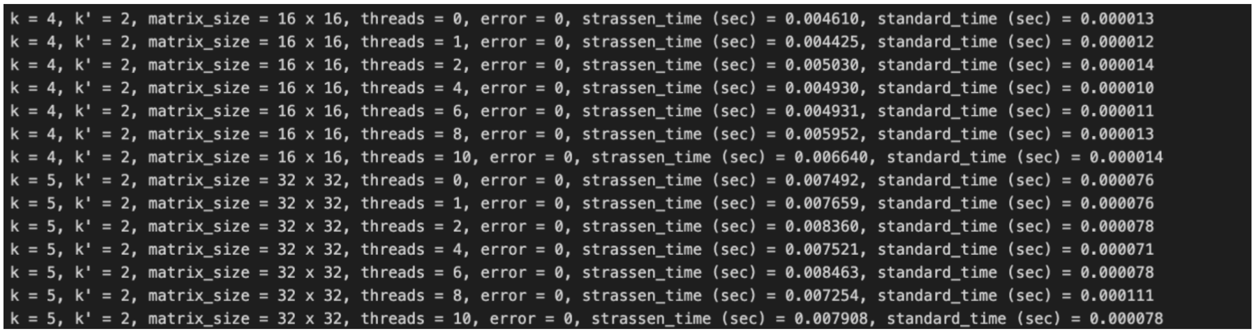
In this project I am using OpenMP, a shared memory technique, to create a parallelized version of Strassen's recursive algorithm. This approach has the advantage of using explicit synchronization and parallel region directives, with an incorporated barrier at the end of each parallel segment. The code is developed using C++ and uses vectors to store the 2-D matrices of size $n \times n$. The values of k and k' are taken as input from the user to vary the size of the matrices as wells as the terminal matrix size. The computation time for both standard algorithm and parallelized Strassen’s algorithm are calculated, and the result of both matrices are compared to determine any error present in the matrix computed using parallelized Strassen’s algorithm.

Experiment:

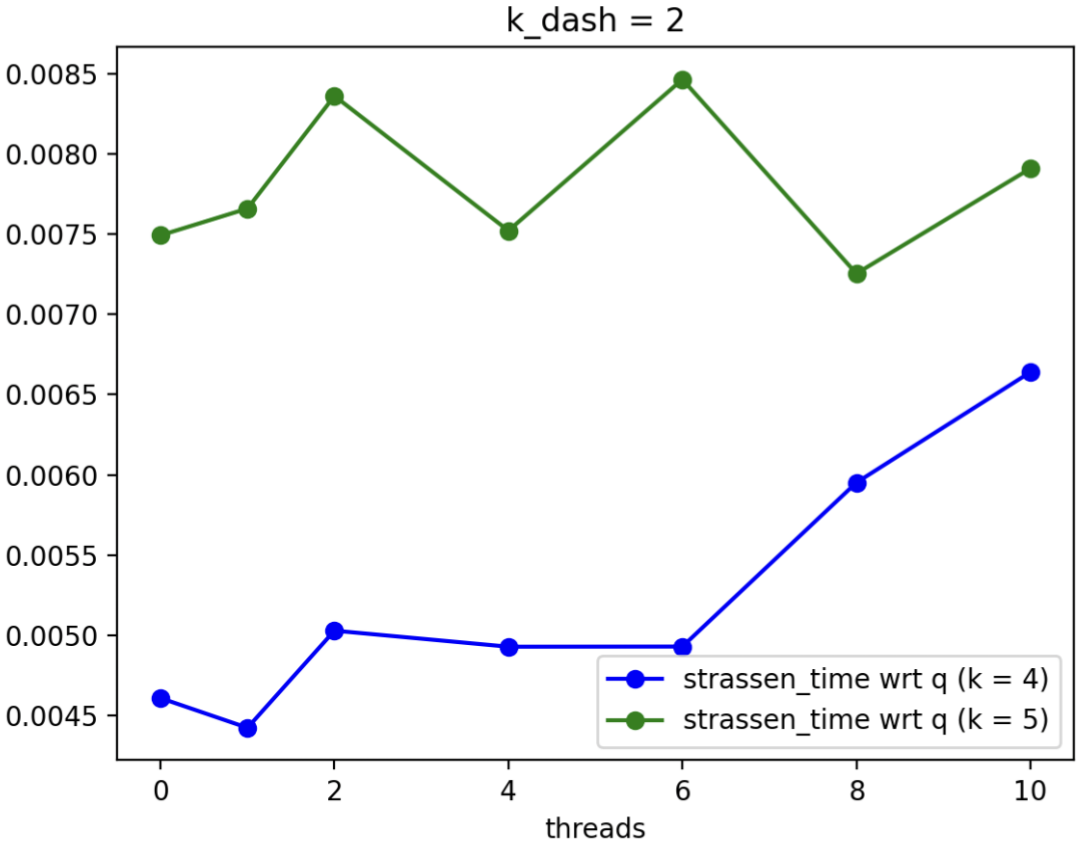
Below are the major approaches used to analyze the parallel performance using k, k‘ and number of threads:
Case 1: Fixed k’ and small matrix size [k as 4 and 5] with different thread count.
Case 2: Fixed k’ and large matrix size by varying k from 8 to 11 with different thread count.
Case 3: Fixed k and changing the value of terminal matrix k’ from 2 to 6 with different thread count.

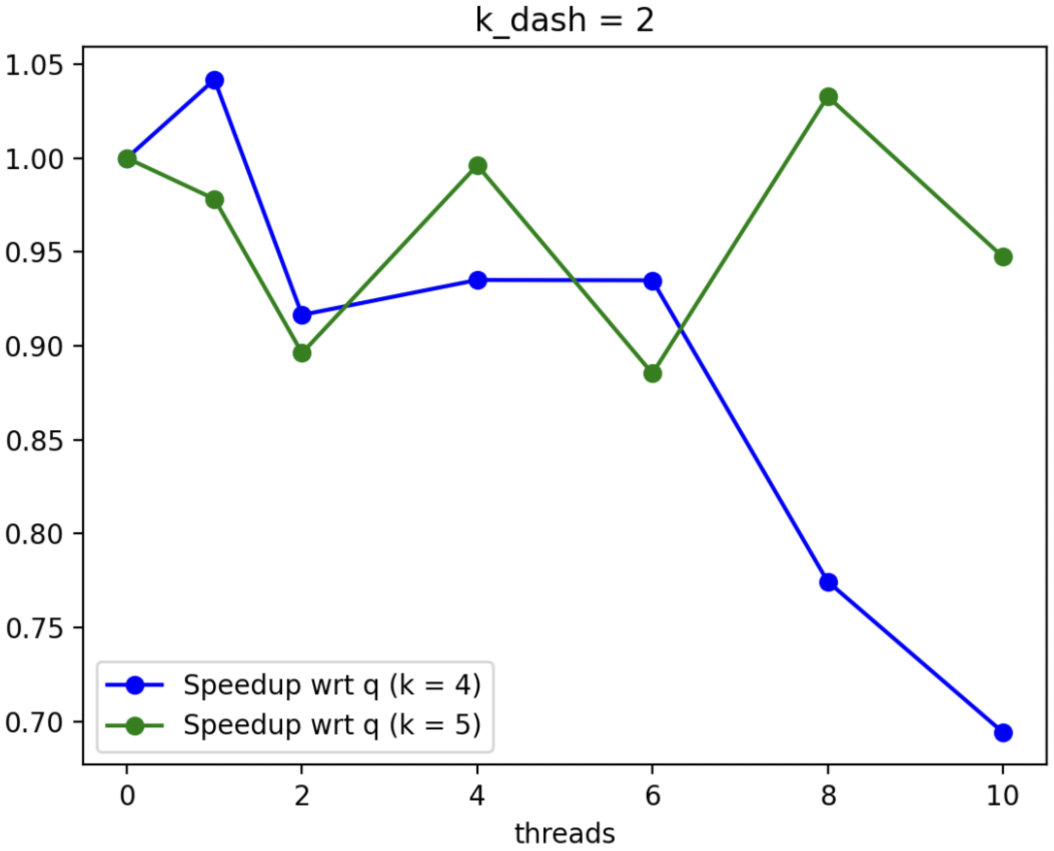
Case 1: Fixed k’ and small matrix size [k as 4 and 5] with different thread count.

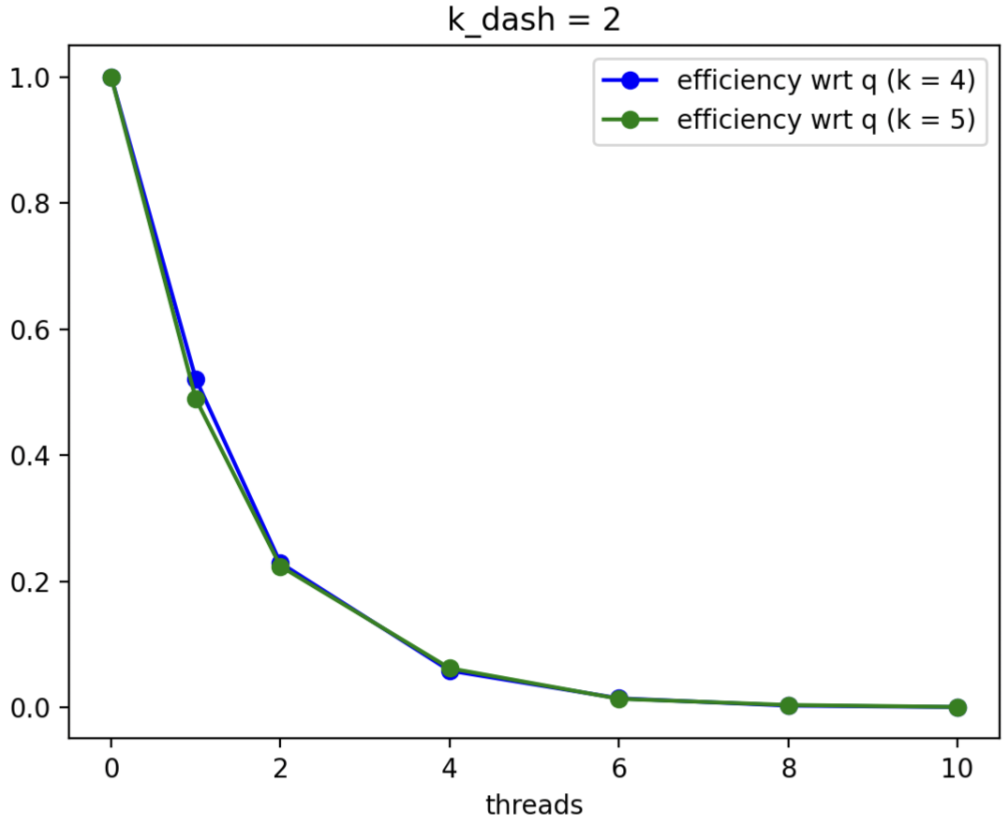
The below plots illustrate the Execution Time, Speedup, and Efficiency for k’ = 2 and k = 4,5.



```
k = 4, k' = 2, matrix_size = 16 x 16, threads = 0, error = 0, strassen_time (sec) = 0.004610, standard_time (sec) = 0.000013
k = 4, k' = 2, matrix_size = 16 x 16, threads = 1, error = 0, strassen_time (sec) = 0.004425, standard_time (sec) = 0.000012
k = 4, k' = 2, matrix_size = 16 x 16, threads = 2, error = 0, strassen_time (sec) = 0.005030, standard_time (sec) = 0.000014
k = 4, k' = 2, matrix_size = 16 x 16, threads = 4, error = 0, strassen_time (sec) = 0.004930, standard_time (sec) = 0.000010
k = 4, k' = 2, matrix_size = 16 x 16, threads = 6, error = 0, strassen_time (sec) = 0.004931, standard_time (sec) = 0.000011
k = 4, k' = 2, matrix_size = 16 x 16, threads = 8, error = 0, strassen_time (sec) = 0.005952, standard_time (sec) = 0.000013
k = 4, k' = 2, matrix_size = 16 x 16, threads = 10, error = 0, strassen_time (sec) = 0.006640, standard_time (sec) = 0.000014
k = 5, k' = 2, matrix_size = 32 x 32, threads = 0, error = 0, strassen_time (sec) = 0.007492, standard_time (sec) = 0.000076
k = 5, k' = 2, matrix_size = 32 x 32, threads = 1, error = 0, strassen_time (sec) = 0.007659, standard_time (sec) = 0.000076
k = 5, k' = 2, matrix_size = 32 x 32, threads = 2, error = 0, strassen_time (sec) = 0.008360, standard_time (sec) = 0.000078
k = 5, k' = 2, matrix_size = 32 x 32, threads = 4, error = 0, strassen_time (sec) = 0.007521, standard_time (sec) = 0.000071
k = 5, k' = 2, matrix_size = 32 x 32, threads = 6, error = 0, strassen_time (sec) = 0.008463, standard_time (sec) = 0.000078
k = 5, k' = 2, matrix_size = 32 x 32, threads = 8, error = 0, strassen_time (sec) = 0.007254, standard_time (sec) = 0.000111
k = 5, k' = 2, matrix_size = 32 x 32, threads = 10, error = 0, strassen_time (sec) = 0.007908, standard_time (sec) = 0.000078
```



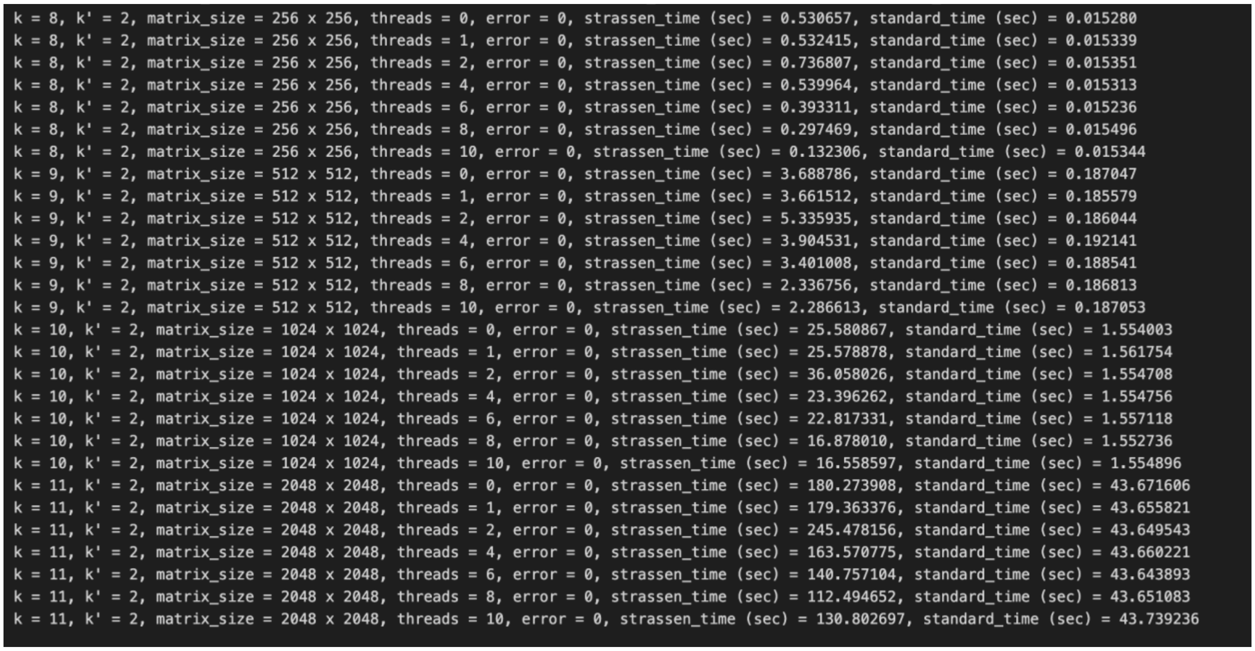

k_dash = 2
1.05
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
2
4
6
8
10
threads
Speedup wrt q (k = 4)
Speedup wrt q (k = 5)


k_dash = 2
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
threads
efficiency wrt q (k = 4)
efficiency wrt q (k = 5)

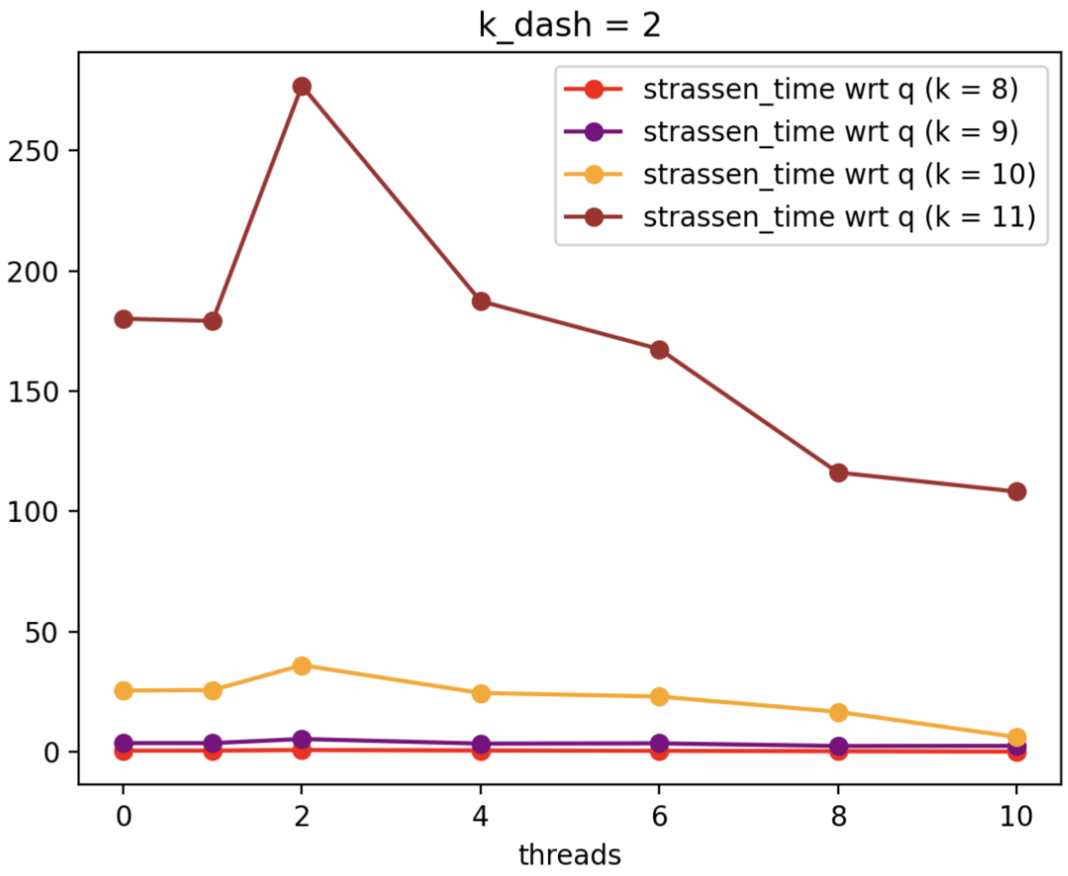
We observed that increasing the number of threads did not significantly enhance parallelism or speedup for lower matrix sizes (i.e., k = 4) since Strassen's Algorithm is recursive and relies on function calls to compute M1, M2, and so on. It can be observed from above also that there is a gradual reduction in speedup and efficiency.
For increasing the performance we will implement the parallel code on larger matrix sizes i.e for values of k equal to 8, 9, 10, and 11. We initially set k' to a smaller value of 2, later increasing it to 6. We also varied the thread counts.

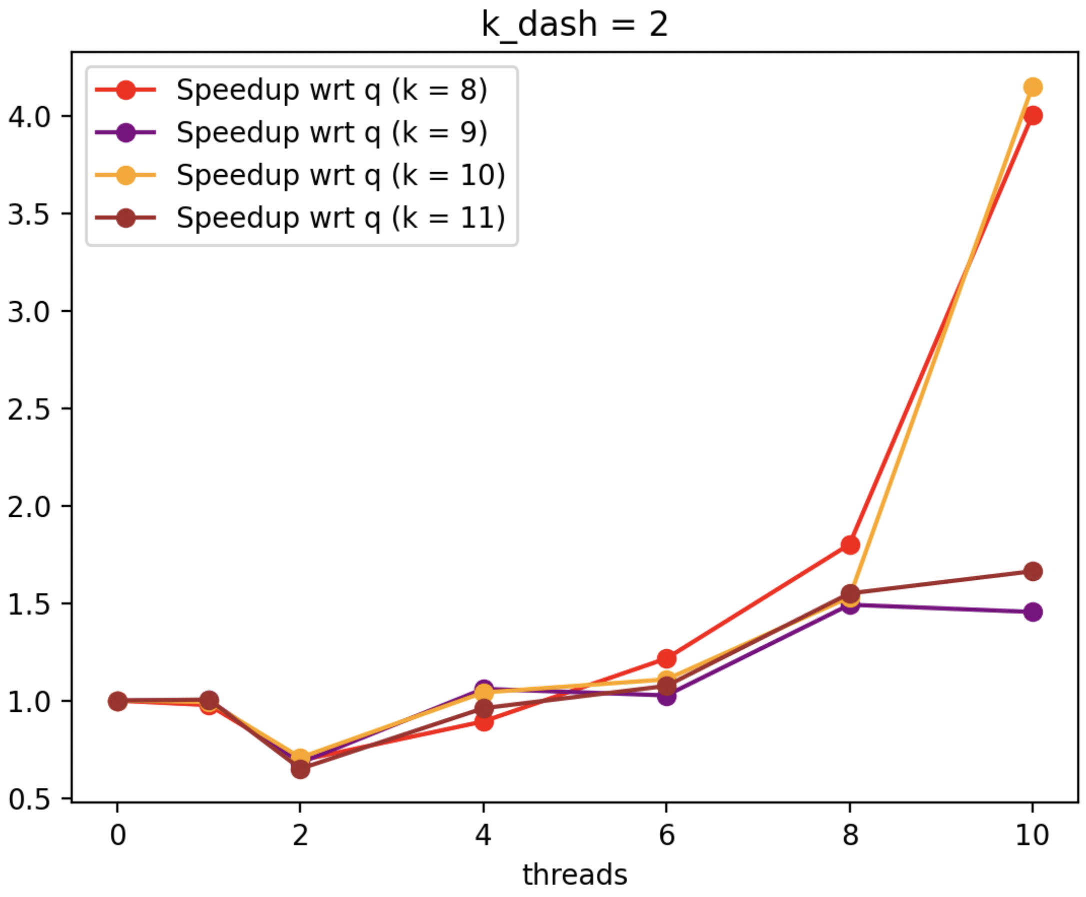
Case 2: Fixed k' and large matrix size by varying k from 8 to 11 with different thread count.

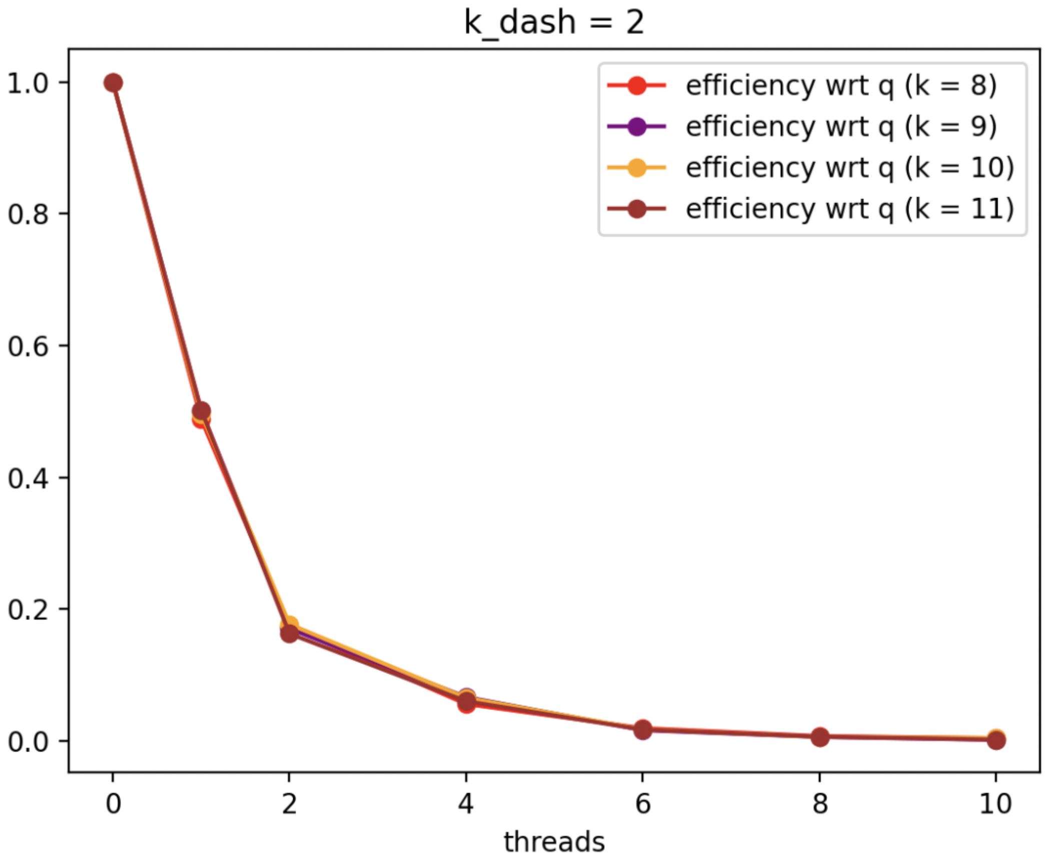
The below plots illustrate the Execution Time, Speedup, and Efficiency for k' = 2 and k = 8, 9, 10, 11 .



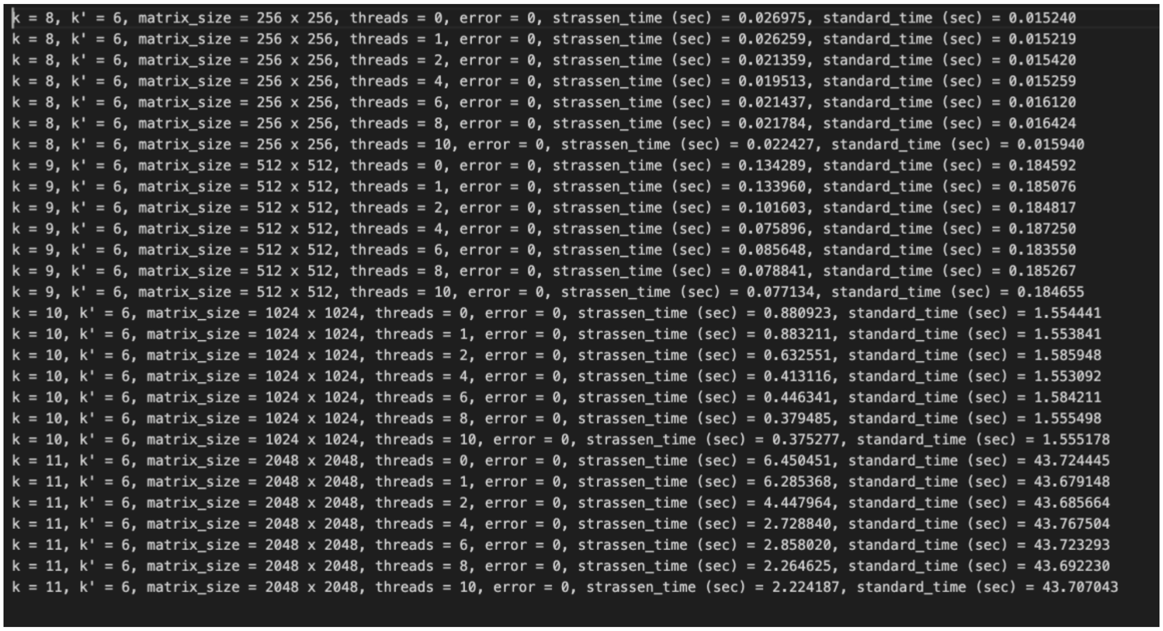
```
k = 8, k' = 2, matrix_size = 256 x 256, threads = 0, error = 0, strassen_time (sec) = 0.530657, standard_time (sec) = 0.015280
k = 8, k' = 2, matrix_size = 256 x 256, threads = 1, error = 0, strassen_time (sec) = 0.532415, standard_time (sec) = 0.015339
k = 8, k' = 2, matrix_size = 256 x 256, threads = 2, error = 0, strassen_time (sec) = 0.736807, standard_time (sec) = 0.015351
k = 8, k' = 2, matrix_size = 256 x 256, threads = 4, error = 0, strassen_time (sec) = 0.539964, standard_time (sec) = 0.015313
k = 8, k' = 2, matrix_size = 256 x 256, threads = 6, error = 0, strassen_time (sec) = 0.393311, standard_time (sec) = 0.015236
k = 8, k' = 2, matrix_size = 256 x 256, threads = 8, error = 0, strassen_time (sec) = 0.297469, standard_time (sec) = 0.015496
k = 8, k' = 2, matrix_size = 256 x 256, threads = 10, error = 0, strassen_time (sec) = 0.132306, standard_time (sec) = 0.015344
k = 9, k' = 2, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 3.688786, standard_time (sec) = 0.187047
k = 9, k' = 2, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 3.661512, standard_time (sec) = 0.185579
k = 9, k' = 2, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 5.335935, standard_time (sec) = 0.186044
k = 9, k' = 2, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 3.904531, standard_time (sec) = 0.192141
k = 9, k' = 2, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 3.401008, standard_time (sec) = 0.188541
k = 9, k' = 2, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 2.336756, standard_time (sec) = 0.186813
k = 9, k' = 2, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 2.286613, standard_time (sec) = 0.187053
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 25.580867, standard_time (sec) = 1.554003
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 25.578878, standard_time (sec) = 1.561754
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 36.058026, standard_time (sec) = 1.554708
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 23.396262, standard_time (sec) = 1.554756
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 22.817331, standard_time (sec) = 1.557118
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 16.878010, standard_time (sec) = 1.552736
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 16.558597, standard_time (sec) = 1.554896
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 180.273908, standard_time (sec) = 43.671606
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 179.363376, standard_time (sec) = 43.655821
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 245.478156, standard_time (sec) = 43.649543
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 163.570775, standard_time (sec) = 43.660221
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 140.757104, standard_time (sec) = 43.643893
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 112.494652, standard_time (sec) = 43.651083
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 130.802697, standard_time (sec) = 43.739236
```


k_dash = 2
strassen_time wrt q (k = 8)
strassen_time wrt q (k = 9)
strassen_time wrt q (k = 10)
strassen_time wrt q (k = 11)
0
50
100
150
200
250
0
2
4
6
8
10
threads

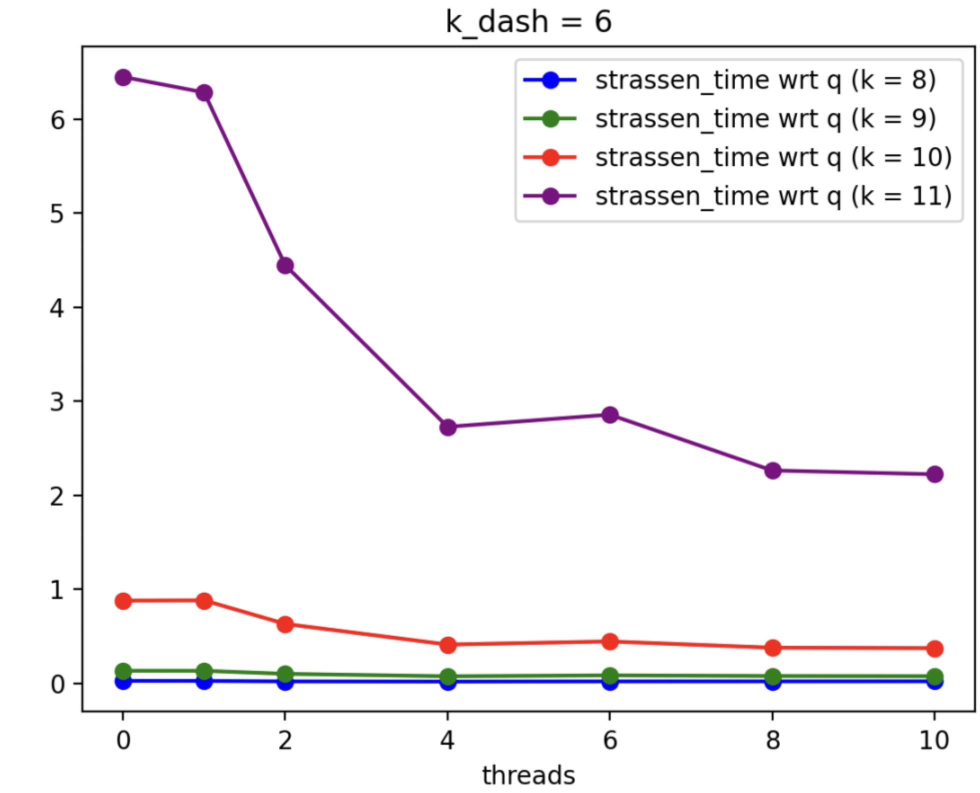

k_dash = 2
Speedup wrt q (k = 8)
Speedup wrt q (k = 9)
Speedup wrt q (k = 10)
Speedup wrt q (k = 11)
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
2
4
6
8
10
threads

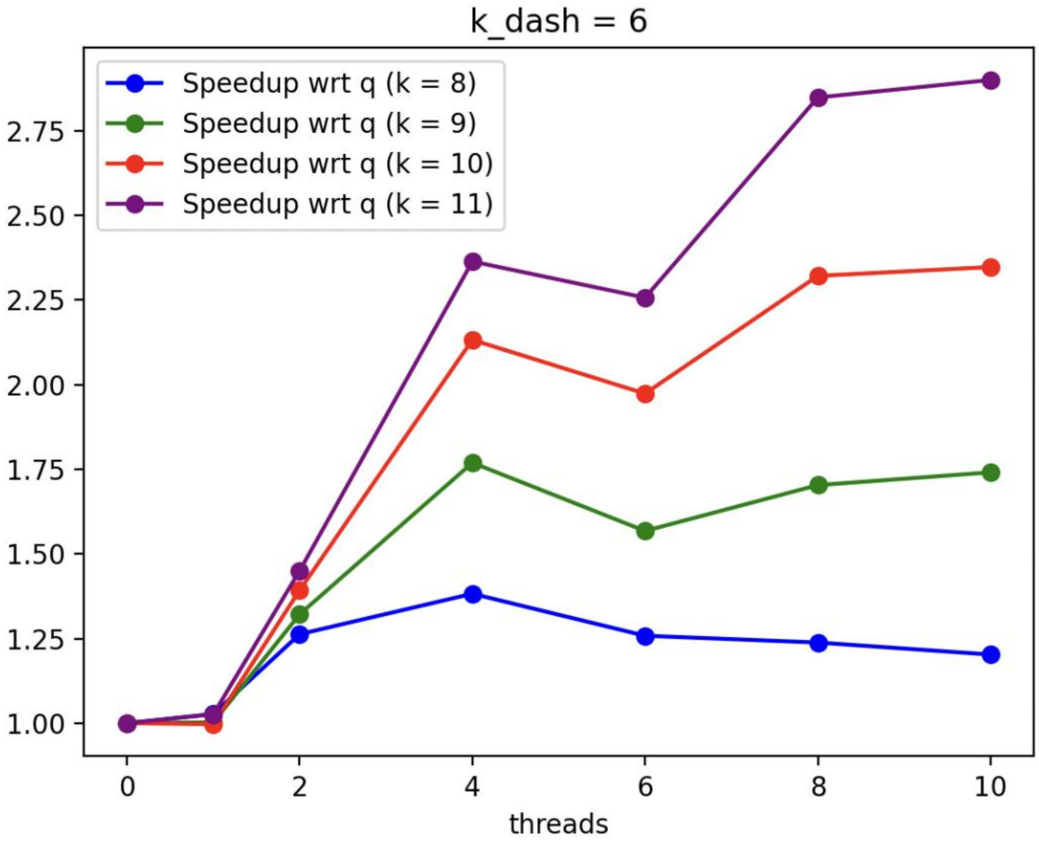


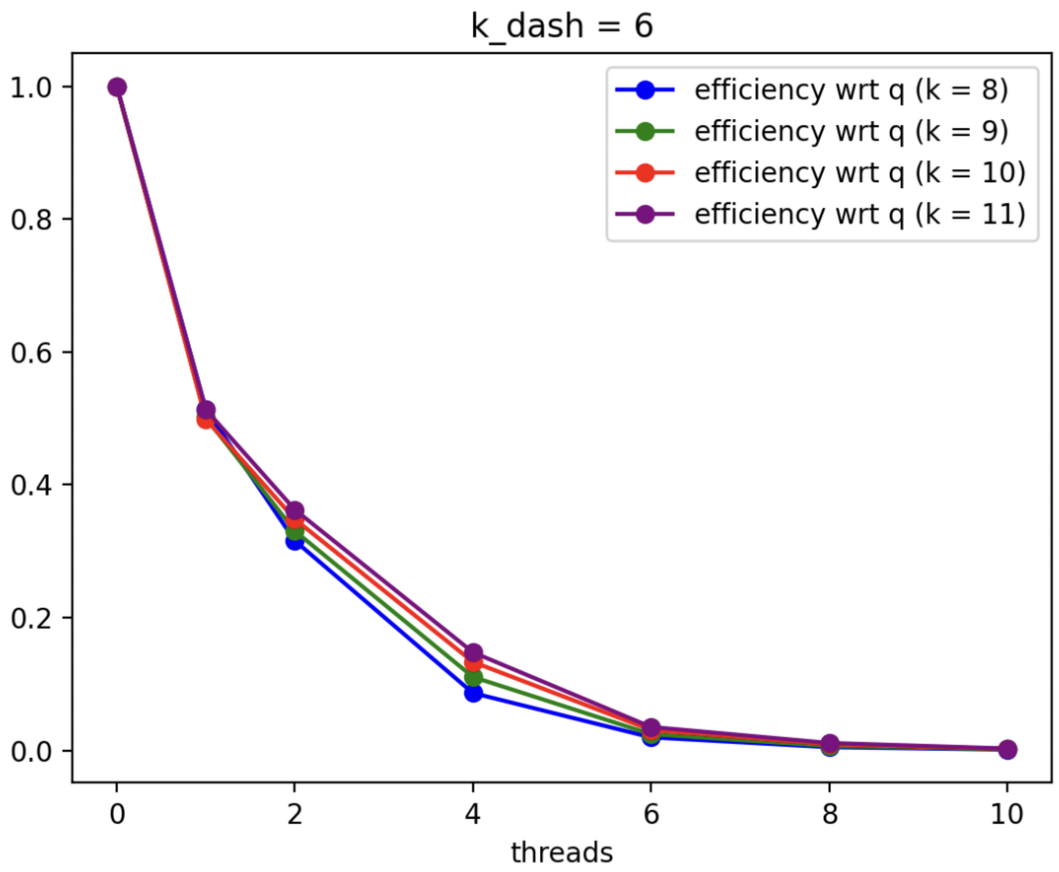
The below plots illustrate the Execution Time, Speedup, and Efficiency for k' = 6 and k = 8, 9, 10, 11 .



```
k = 8, k' = 6, matrix_size = 256 x 256, threads = 0, error = 0, strassen_time (sec) = 0.026975, standard_time (sec) = 0.015240
k = 8, k' = 6, matrix_size = 256 x 256, threads = 1, error = 0, strassen_time (sec) = 0.026259, standard_time (sec) = 0.015219
k = 8, k' = 6, matrix_size = 256 x 256, threads = 2, error = 0, strassen_time (sec) = 0.021359, standard_time (sec) = 0.015420
k = 8, k' = 6, matrix_size = 256 x 256, threads = 4, error = 0, strassen_time (sec) = 0.019513, standard_time (sec) = 0.015259
k = 8, k' = 6, matrix_size = 256 x 256, threads = 6, error = 0, strassen_time (sec) = 0.021437, standard_time (sec) = 0.016120
k = 8, k' = 6, matrix_size = 256 x 256, threads = 8, error = 0, strassen_time (sec) = 0.021784, standard_time (sec) = 0.016424
k = 8, k' = 6, matrix_size = 256 x 256, threads = 10, error = 0, strassen_time (sec) = 0.022427, standard_time (sec) = 0.015940
k = 9, k' = 6, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 0.134289, standard_time (sec) = 0.184592
k = 9, k' = 6, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 0.133960, standard_time (sec) = 0.185076
k = 9, k' = 6, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 0.101603, standard_time (sec) = 0.184817
k = 9, k' = 6, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 0.075896, standard_time (sec) = 0.187250
k = 9, k' = 6, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 0.085648, standard_time (sec) = 0.183550
k = 9, k' = 6, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 0.078841, standard_time (sec) = 0.185267
k = 9, k' = 6, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 0.077134, standard_time (sec) = 0.184655
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 0.880923, standard_time (sec) = 1.554441
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 0.883211, standard_time (sec) = 1.553841
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 0.632551, standard_time (sec) = 1.585948
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 0.413116, standard_time (sec) = 1.553092
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 0.446341, standard_time (sec) = 1.584211
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 0.379485, standard_time (sec) = 1.555498
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 0.375277, standard_time (sec) = 1.555178
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 6.450451, standard_time (sec) = 43.724445
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 6.285368, standard_time (sec) = 43.679148
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 4.447964, standard_time (sec) = 43.685664
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 2.728840, standard_time (sec) = 43.767504
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 2.858020, standard_time (sec) = 43.723293
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 2.264625, standard_time (sec) = 43.692230
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 2.224187, standard_time (sec) = 43.707043
```


k_dash = 6
strassen_time wrt q (k = 8)
strassen_time wrt q (k = 9)
strassen_time wrt q (k = 10)
strassen_time wrt q (k = 11)
0
1
2
3
4
5
6
0
2
4
6
8
10
threads


k_dash = 6
Speedup wrt q (k = 8)
Speedup wrt q (k = 9)
Speedup wrt q (k = 10)
Speedup wrt q (k = 11)
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
0
2
4
6
8
10
threads



Observation for Case 1 and 2 (Changing k, fixed k'):

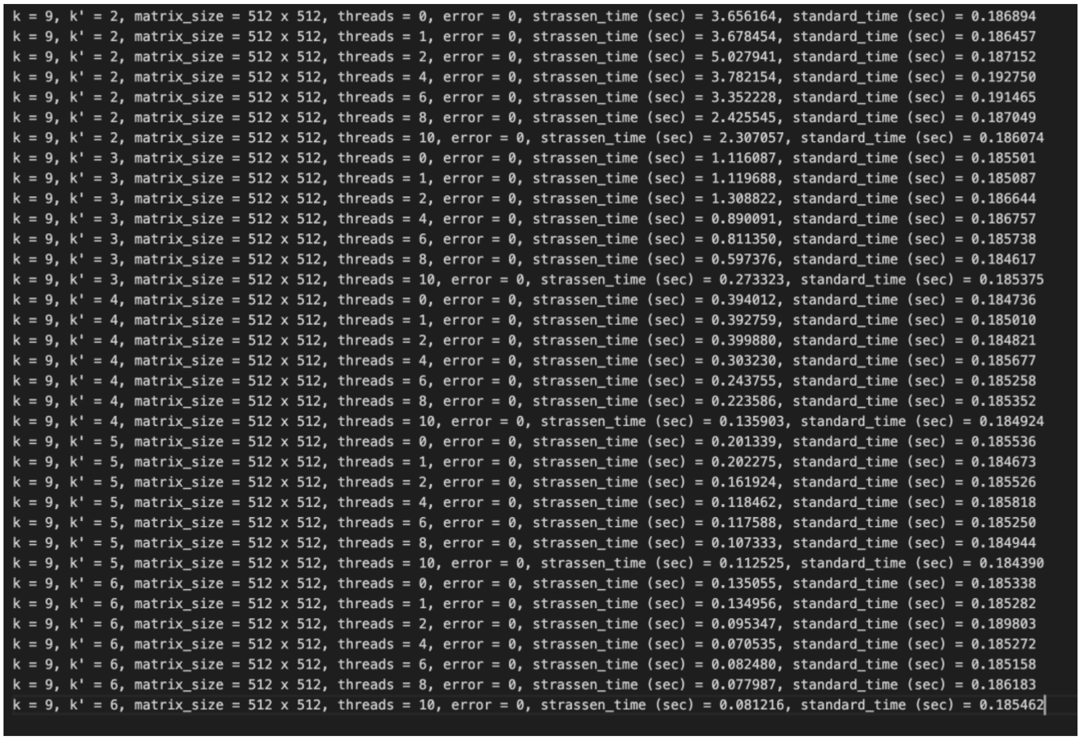
The minimum value of k was initially set to 4, which corresponds to a 16 x 16 matrix. For smaller matrices, parallelized Strassen's multiplication is not very efficient, as seen by the small speedup and low efficiency that we saw. The execution time increased when we increased the value of k to 8, 9, 10, and 11, most likely because of the increased number of recursive calls and total computing time.

This observation is confirmed by the execution time figures across different matrix sizes. The greater the matrix size, the more efficient Strassen's parallel algorithm is, leading to increased speedup. Increasing the number of threads by a factor of two effectively parallelized the code for k values of 8, 9, 10, and 11, resulting in a progressive reduction in execution time and an increase in performance.
But it's important to note that as the number of threads rose, the efficiency plot showed a deteriorating trend.

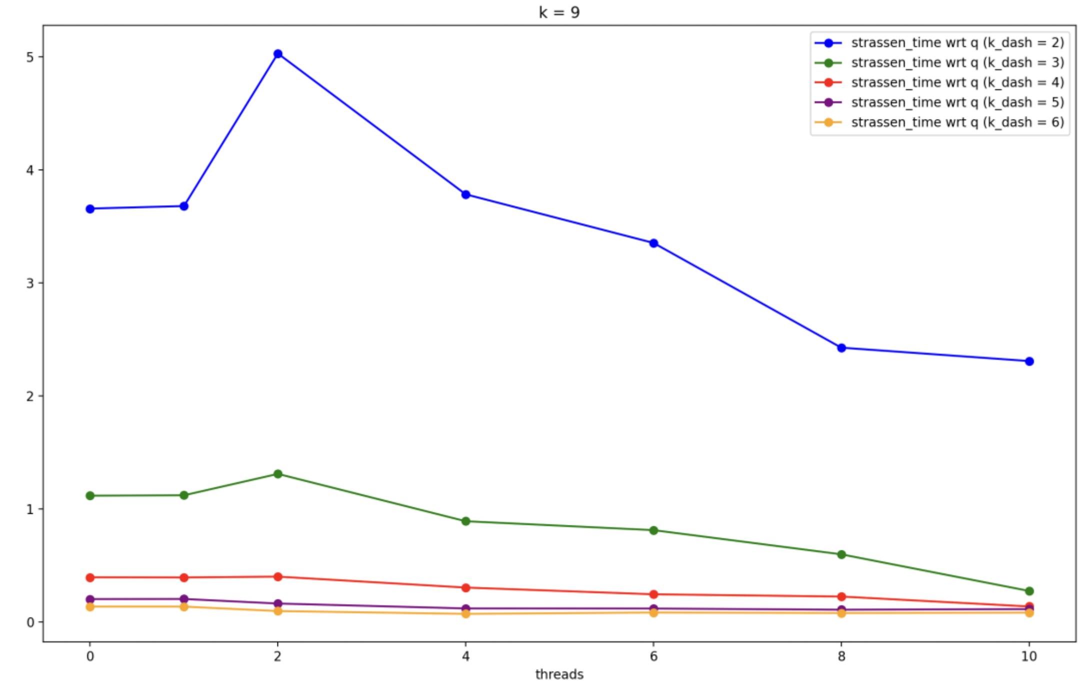
Case 3: Fixed k and changing the value of terminal matrix k’ from 2 to 6 with different thread count.

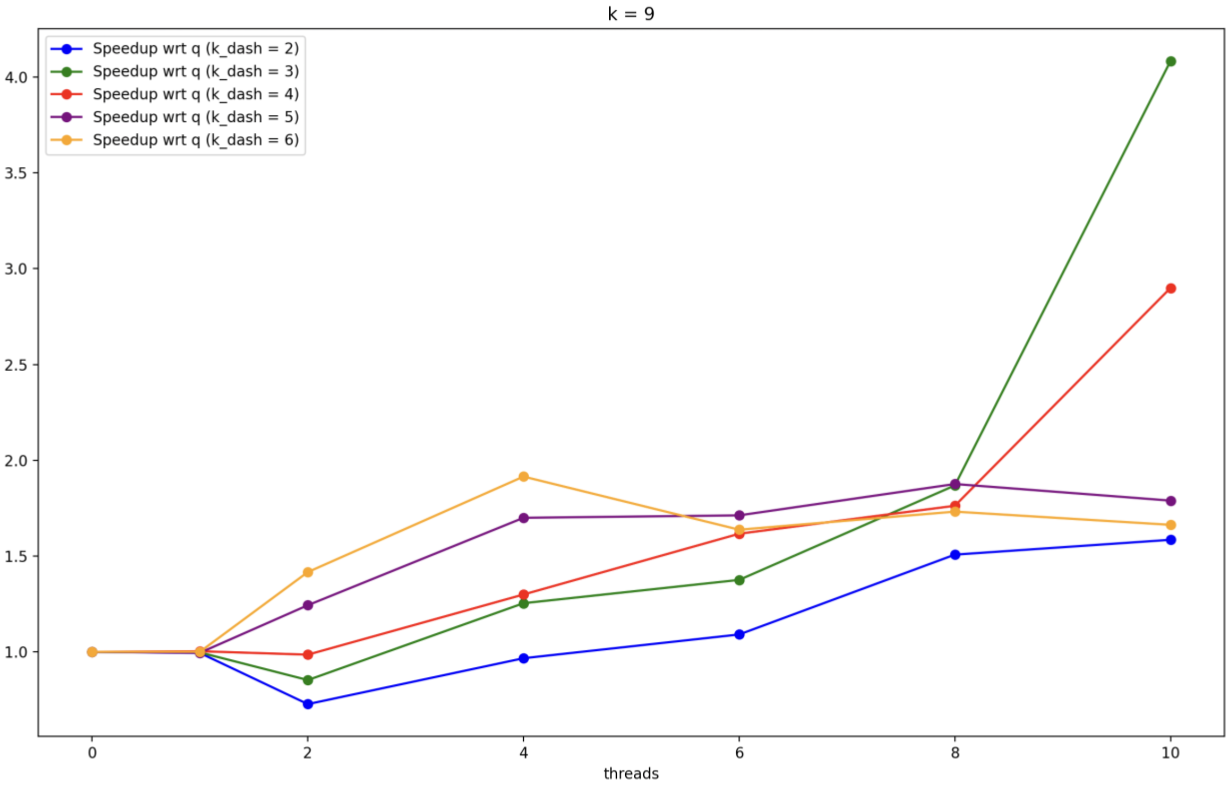
Now we will keep value of k fixed and will change the value of terminal matrix k’ from 2 to 6. In Case 3, we ran the experiment to observe the trend of execution time and parallel performance by varying k’ values for matrix size k = 9, k = 10 and k = 11.

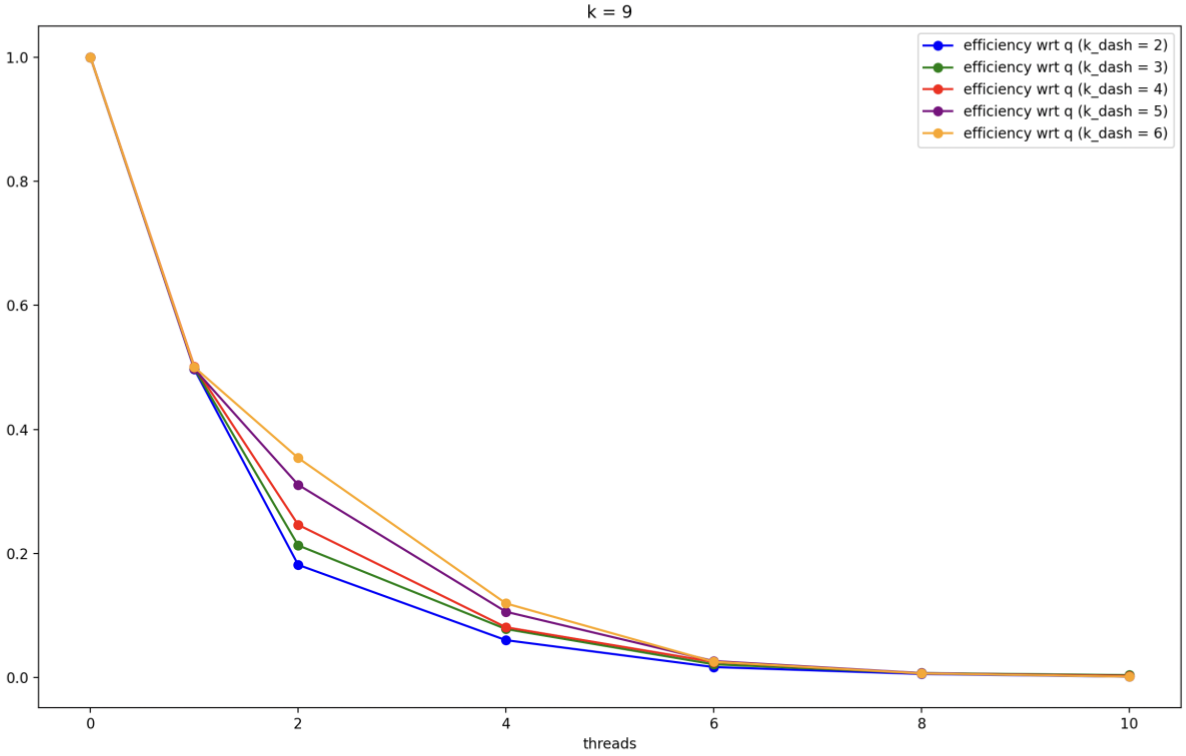
The below plots illustrate the Execution Time, Speedup, and Efficiency for k = 9 and k’ = 2, 3, 4, 5, 6.



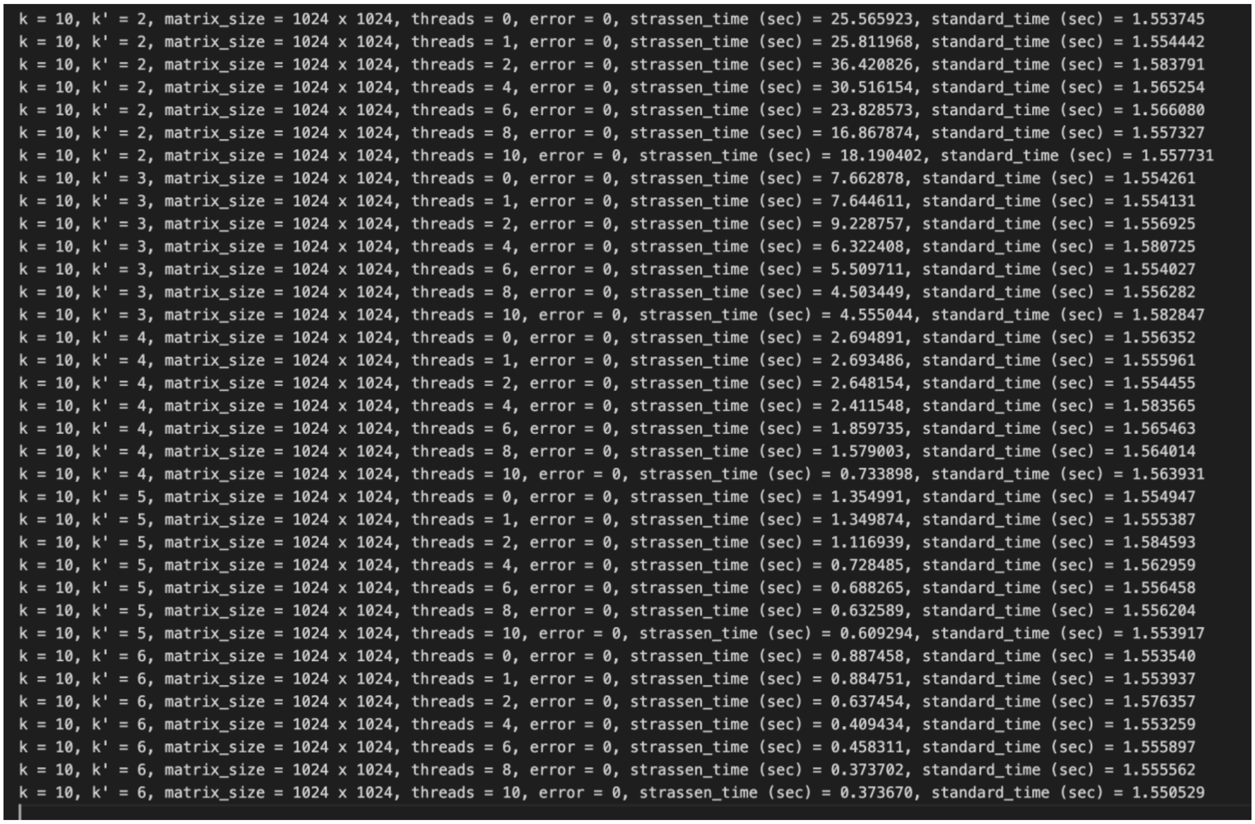
```
k = 9, k' = 2, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 3.656164, standard_time (sec) = 0.186894
k = 9, k' = 2, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 3.678454, standard_time (sec) = 0.186457
k = 9, k' = 2, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 5.027941, standard_time (sec) = 0.187152
k = 9, k' = 2, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 3.782154, standard_time (sec) = 0.192750
k = 9, k' = 2, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 3.352228, standard_time (sec) = 0.191465
k = 9, k' = 2, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 2.425545, standard_time (sec) = 0.187049
k = 9, k' = 2, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 2.307057, standard_time (sec) = 0.186074
k = 9, k' = 3, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 1.116087, standard_time (sec) = 0.185501
k = 9, k' = 3, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 1.119688, standard_time (sec) = 0.185087
k = 9, k' = 3, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 1.308822, standard_time (sec) = 0.186644
k = 9, k' = 3, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 0.890091, standard_time (sec) = 0.186757
k = 9, k' = 3, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 0.811350, standard_time (sec) = 0.185738
k = 9, k' = 3, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 0.597376, standard_time (sec) = 0.184617
k = 9, k' = 3, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 0.273323, standard_time (sec) = 0.185375
k = 9, k' = 4, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 0.394012, standard_time (sec) = 0.184736
k = 9, k' = 4, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 0.392759, standard_time (sec) = 0.185010
k = 9, k' = 4, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 0.399880, standard_time (sec) = 0.184821
k = 9, k' = 4, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 0.303230, standard_time (sec) = 0.185677
k = 9, k' = 4, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 0.243755, standard_time (sec) = 0.185258
k = 9, k' = 4, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 0.223586, standard_time (sec) = 0.185352
k = 9, k' = 4, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 0.135903, standard_time (sec) = 0.184924
k = 9, k' = 5, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 0.201339, standard_time (sec) = 0.185536
k = 9, k' = 5, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 0.202275, standard_time (sec) = 0.184673
k = 9, k' = 5, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 0.161924, standard_time (sec) = 0.185526
k = 9, k' = 5, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 0.118462, standard_time (sec) = 0.185818
k = 9, k' = 5, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 0.117588, standard_time (sec) = 0.185250
k = 9, k' = 5, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 0.107333, standard_time (sec) = 0.184944
k = 9, k' = 5, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 0.112525, standard_time (sec) = 0.184390
k = 9, k' = 6, matrix_size = 512 x 512, threads = 0, error = 0, strassen_time (sec) = 0.135055, standard_time (sec) = 0.185338
k = 9, k' = 6, matrix_size = 512 x 512, threads = 1, error = 0, strassen_time (sec) = 0.134956, standard_time (sec) = 0.185282
k = 9, k' = 6, matrix_size = 512 x 512, threads = 2, error = 0, strassen_time (sec) = 0.095347, standard_time (sec) = 0.189803
k = 9, k' = 6, matrix_size = 512 x 512, threads = 4, error = 0, strassen_time (sec) = 0.070535, standard_time (sec) = 0.185272
k = 9, k' = 6, matrix_size = 512 x 512, threads = 6, error = 0, strassen_time (sec) = 0.082480, standard_time (sec) = 0.185158
k = 9, k' = 6, matrix_size = 512 x 512, threads = 8, error = 0, strassen_time (sec) = 0.077987, standard_time (sec) = 0.186183
k = 9, k' = 6, matrix_size = 512 x 512, threads = 10, error = 0, strassen_time (sec) = 0.081216, standard_time (sec) = 0.185462
```



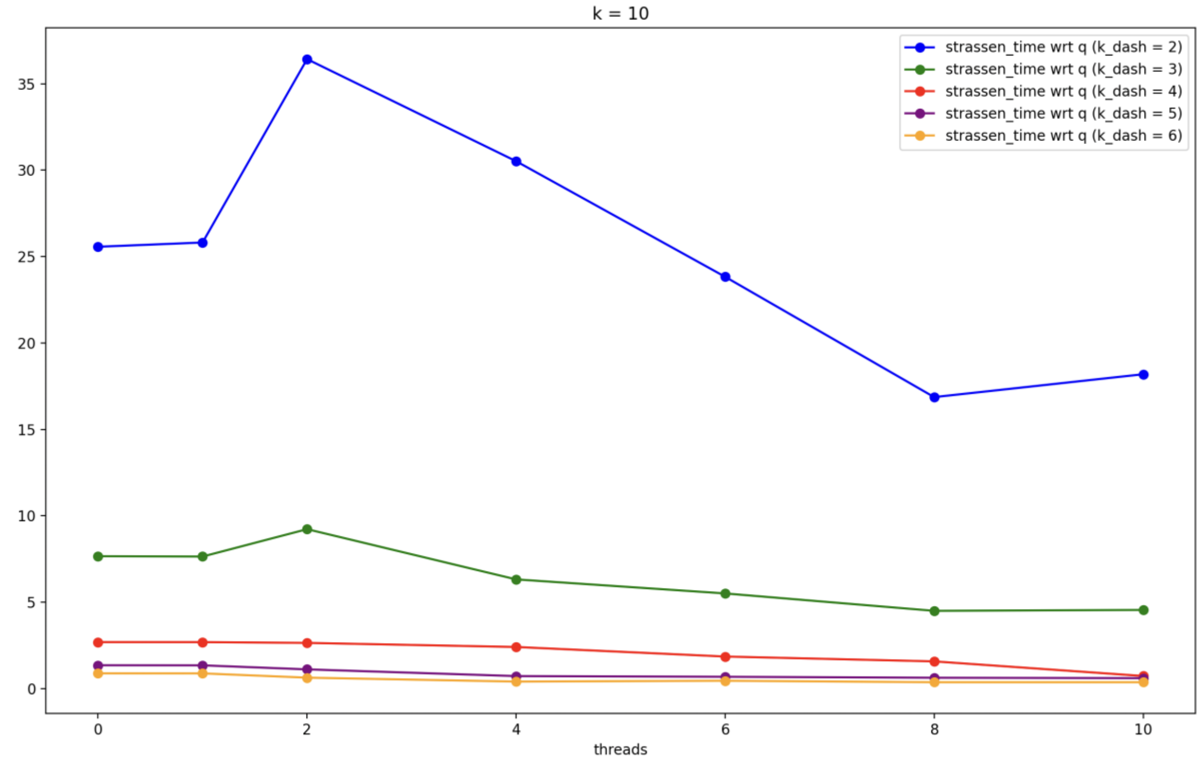

k = 9
Speedup wrt q (k_dash = 2)
Speedup wrt q (k_dash = 3)
Speedup wrt q (k_dash = 4)
Speedup wrt q (k_dash = 5)
Speedup wrt q (k_dash = 6)
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
2
4
6
8
10
threads

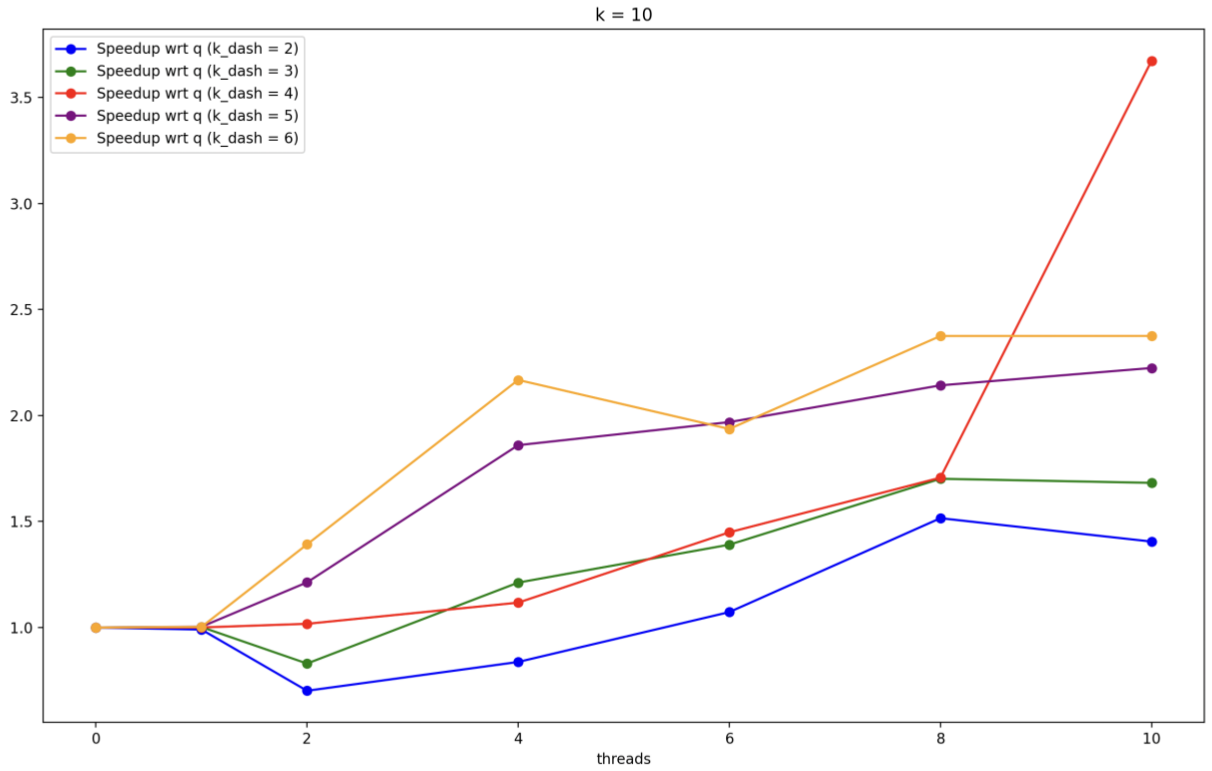

k = 9
efficiency wrt q (k_dash = 2)
efficiency wrt q (k_dash = 3)
efficiency wrt q (k_dash = 4)
efficiency wrt q (k_dash = 5)
efficiency wrt q (k_dash = 6)
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
threads

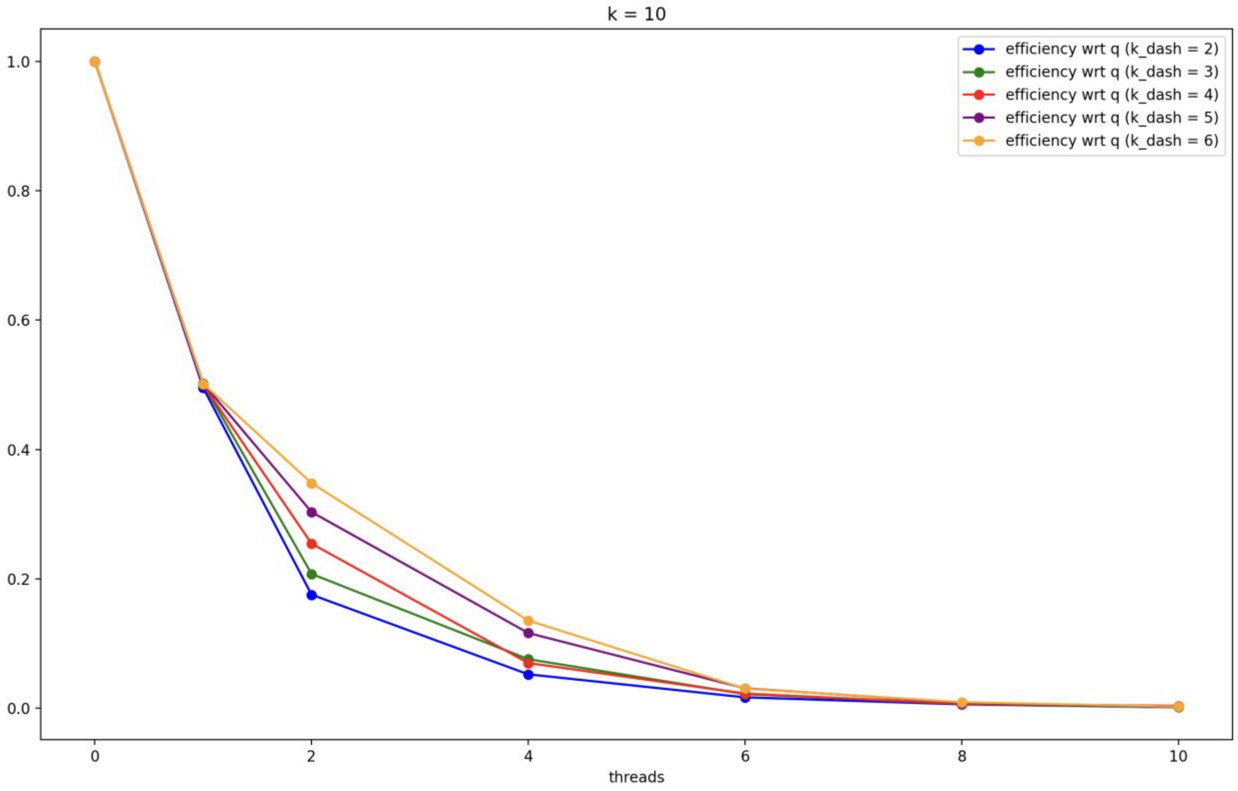
The below plots illustrate the Execution Time, Speedup, and Efficiency for k = 10 and k’ = 2, 3, 4, 5, 6.



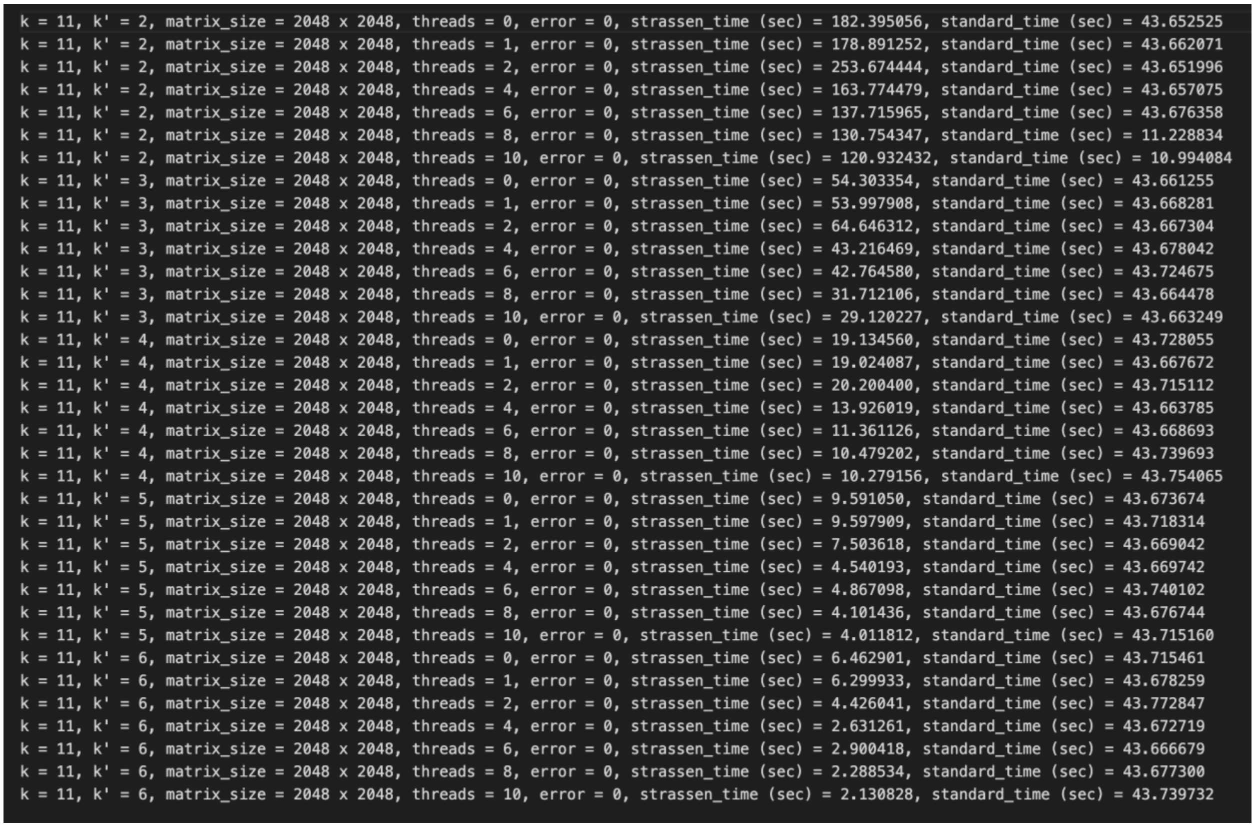
```
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 25.565923, standard_time (sec) = 1.553745
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 25.811968, standard_time (sec) = 1.554442
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 36.420826, standard_time (sec) = 1.583791
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 30.516154, standard_time (sec) = 1.565254
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 23.828573, standard_time (sec) = 1.566080
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 16.867874, standard_time (sec) = 1.557327
k = 10, k' = 2, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 18.190402, standard_time (sec) = 1.557731
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 7.662878, standard_time (sec) = 1.554261
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 7.644611, standard_time (sec) = 1.554131
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 9.228757, standard_time (sec) = 1.556925
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 6.322408, standard_time (sec) = 1.580725
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 5.509711, standard_time (sec) = 1.554027
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 4.503449, standard_time (sec) = 1.556282
k = 10, k' = 3, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 4.555044, standard_time (sec) = 1.582847
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 2.694891, standard_time (sec) = 1.556352
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 2.693486, standard_time (sec) = 1.555961
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 2.648154, standard_time (sec) = 1.554455
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 2.411548, standard_time (sec) = 1.583565
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 1.859735, standard_time (sec) = 1.565463
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 1.579003, standard_time (sec) = 1.564014
k = 10, k' = 4, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 0.733898, standard_time (sec) = 1.563931
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 1.354991, standard_time (sec) = 1.554947
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 1.349874, standard_time (sec) = 1.555387
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 1.116939, standard_time (sec) = 1.584593
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 0.728485, standard_time (sec) = 1.562959
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 0.688265, standard_time (sec) = 1.556458
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 0.632589, standard_time (sec) = 1.556204
k = 10, k' = 5, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 0.609294, standard_time (sec) = 1.553917
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 0, error = 0, strassen_time (sec) = 0.887458, standard_time (sec) = 1.553540
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 1, error = 0, strassen_time (sec) = 0.884751, standard_time (sec) = 1.553937
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 2, error = 0, strassen_time (sec) = 0.637454, standard_time (sec) = 1.576357
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 4, error = 0, strassen_time (sec) = 0.409434, standard_time (sec) = 1.553259
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 6, error = 0, strassen_time (sec) = 0.458311, standard_time (sec) = 1.555897
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 8, error = 0, strassen_time (sec) = 0.373702, standard_time (sec) = 1.555562
k = 10, k' = 6, matrix_size = 1024 x 1024, threads = 10, error = 0, strassen_time (sec) = 0.373670, standard_time (sec) = 1.550529
```



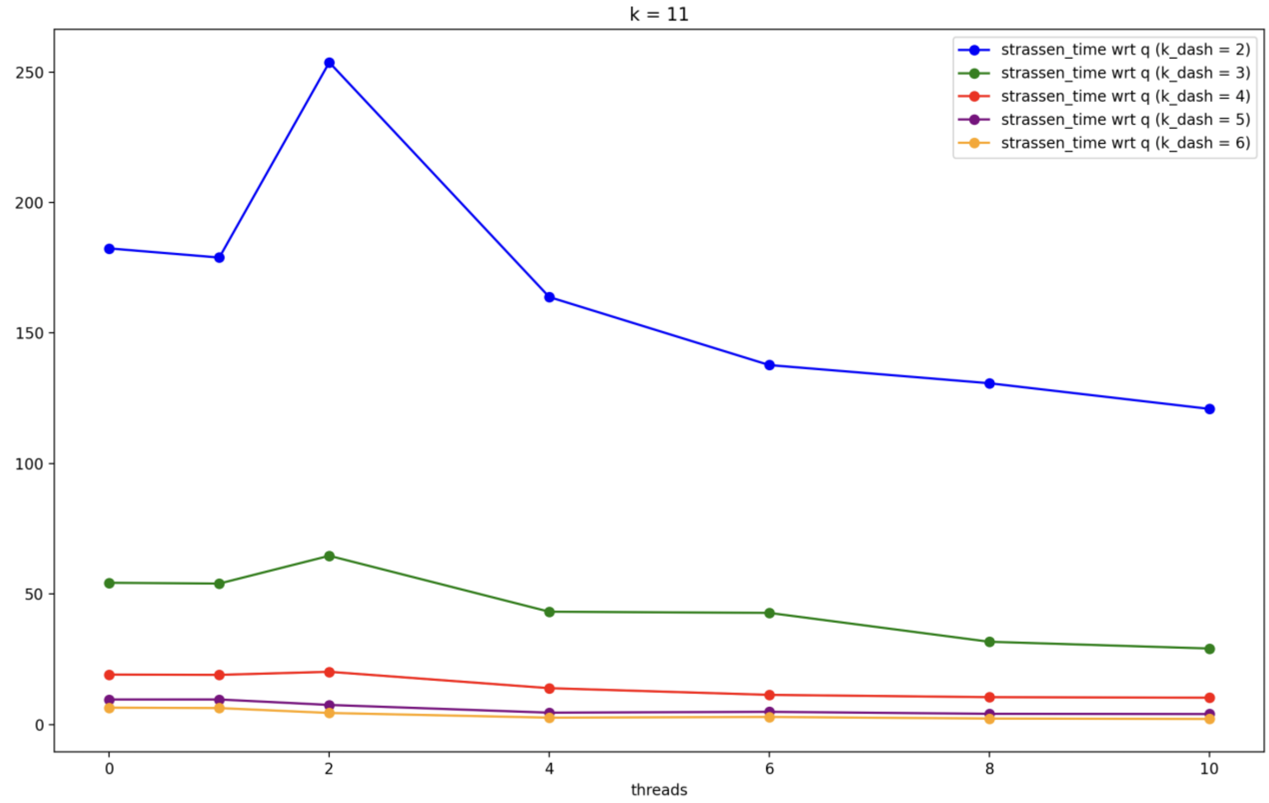

k = 10
Speedup wrt q (k_dash = 2)
Speedup wrt q (k_dash = 3)
Speedup wrt q (k_dash = 4)
Speedup wrt q (k_dash = 5)
Speedup wrt q (k_dash = 6)
3.5
3.0
2.5
2.0
1.5
1.0
0
2
4
6
8
10
threads

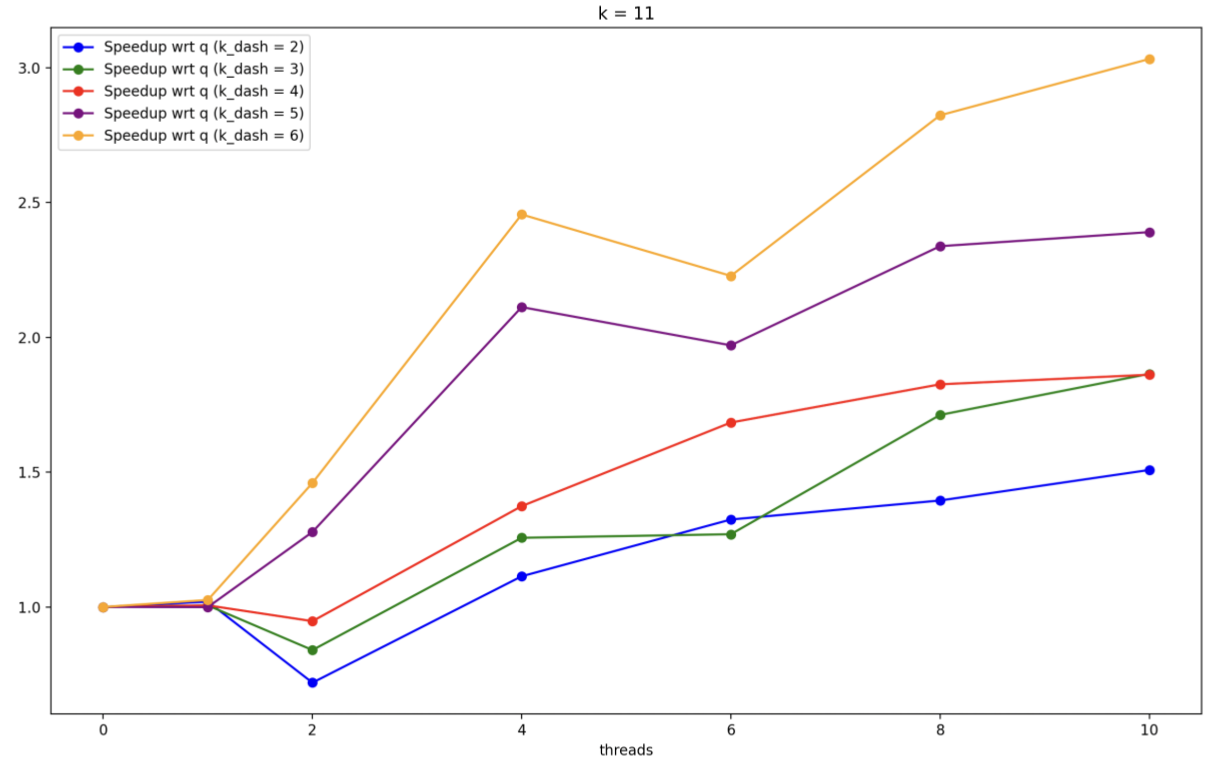

k = 10
efficiency wrt q (k_dash = 2)
efficiency wrt q (k_dash = 3)
efficiency wrt q (k_dash = 4)
efficiency wrt q (k_dash = 5)
efficiency wrt q (k_dash = 6)
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
threads

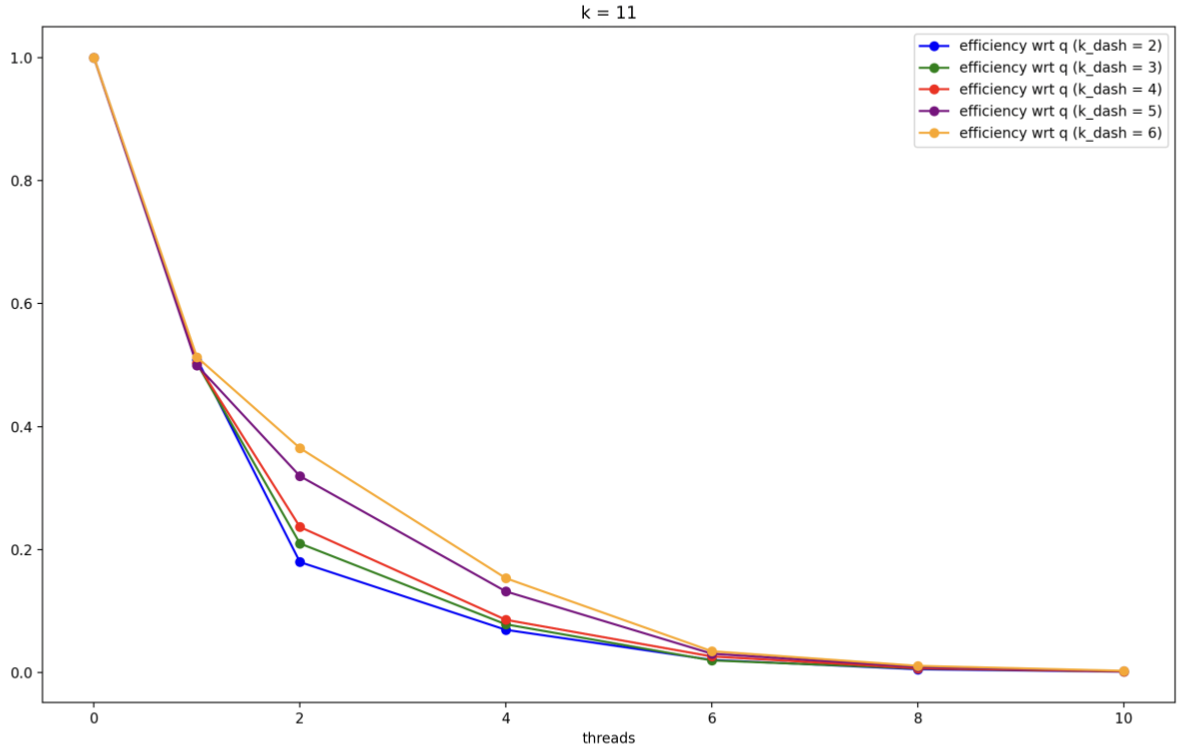
The below plots illustrate the Execution Time, Speedup, and Efficiency for k = 11 and k’ = 2, 3, 4, 5, 6.



```
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 182.395056, standard_time (sec) = 43.652525
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 178.891252, standard_time (sec) = 43.662071
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 253.674444, standard_time (sec) = 43.651996
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 163.774479, standard_time (sec) = 43.657075
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 137.715965, standard_time (sec) = 43.676358
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 130.754347, standard_time (sec) = 11.228834
k = 11, k' = 2, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 120.932432, standard_time (sec) = 10.994084
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 54.303354, standard_time (sec) = 43.661255
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 53.997908, standard_time (sec) = 43.668281
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 64.646312, standard_time (sec) = 43.667304
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 43.216469, standard_time (sec) = 43.678042
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 42.764580, standard_time (sec) = 43.724675
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 31.712106, standard_time (sec) = 43.664478
k = 11, k' = 3, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 29.120227, standard_time (sec) = 43.663249
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 19.134560, standard_time (sec) = 43.728055
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 19.024087, standard_time (sec) = 43.667672
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 20.200400, standard_time (sec) = 43.715112
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 13.926019, standard_time (sec) = 43.663785
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 11.361126, standard_time (sec) = 43.668693
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 10.479202, standard_time (sec) = 43.739693
k = 11, k' = 4, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 10.279156, standard_time (sec) = 43.754065
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 9.591050, standard_time (sec) = 43.673674
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 9.597909, standard_time (sec) = 43.718314
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 7.503618, standard_time (sec) = 43.669042
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 4.540193, standard_time (sec) = 43.669742
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 4.867098, standard_time (sec) = 43.740102
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 4.101436, standard_time (sec) = 43.676744
k = 11, k' = 5, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 4.011812, standard_time (sec) = 43.715160
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 0, error = 0, strassen_time (sec) = 6.462901, standard_time (sec) = 43.715461
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 1, error = 0, strassen_time (sec) = 6.299933, standard_time (sec) = 43.678259
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 2, error = 0, strassen_time (sec) = 4.426041, standard_time (sec) = 43.772847
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 4, error = 0, strassen_time (sec) = 2.631261, standard_time (sec) = 43.672719
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 6, error = 0, strassen_time (sec) = 2.900418, standard_time (sec) = 43.666679
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 8, error = 0, strassen_time (sec) = 2.288534, standard_time (sec) = 43.677300
k = 11, k' = 6, matrix_size = 2048 x 2048, threads = 10, error = 0, strassen_time (sec) = 2.130828, standard_time (sec) = 43.739732
```




k = 11
Speedup wrt q (k_dash = 2)
Speedup wrt q (k_dash = 3)
Speedup wrt q (k_dash = 4)
Speedup wrt q (k_dash = 5)
Speedup wrt q (k_dash = 6)
3.0
2.5
2.0
1.5
1.0
0
2
4
6
8
10
threads


k = 11
efficiency wrt q (k_dash = 2)
efficiency wrt q (k_dash = 3)
efficiency wrt q (k_dash = 4)
efficiency wrt q (k_dash = 5)
efficiency wrt q (k_dash = 6)
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
threads

Observation for Case 3 (Changing k’, fixed k):

When we increased the value of k' while keeping k constant, we saw a decrease in execution time in addition to an increase in speed. The reason for this phenomenon could be that more levels for regular multiplication are produced by greater k' values. As a result, parallelizing this portion of the algorithm is more effective than parallelizing the recursive calls using pragma tasks. Here parallelizing normal multiplication is more efficient than using Strassen's method as using tasks in recursive calls results in overhead. As a result, speedup is improved, and execution time is gradually decreased.

Design choices to improve the parallel performance of the code:

The main function accepts user inputs for k, k', and num_threads. These values undergo validation before commencing the primary operations. The k value determines the size of matrices A and B, while k' determines the terminal matrix size. The num_threads value specifies the number of threads for code execution. The entire code is divided into distinct sub-modules, each preceded by necessary descriptive comments for enhanced comprehension. Code modularization is a pivotal aspect of development, and the entire code is structured with this principle in mind.

To parallelize Strassen's algorithm matrices of equally sized blocks derived from A and B, as well as matrices required for the computation of C are initialized. The necessary for loops are parallelized using the OpenMP directive **‘#pragma omp parallel for’**. The seven matrix multiplications (M1 to M7) are segmented into distinct tasks and parallelized using the following OpenMP directives **‘#pragma omp parallel’**, **‘#pragma omp single’** and **‘#pragma omp task’**. The final product matrix C is computed from the equally sized blocks of C using for loops, which are parallelized using the OpenMP directive ‘**#pragma omp parallel for’**.

Insights developed after working on this project:

1) Optimization of Thread Count:
    a. Thread based parallelization can speed up execution.
    b. Selecting the ideal thread count is essential for minimizing overhead and maximizing speed.
2) Advantage of a Hybrid Approach:
    a. It is advantageous to join Strassen's recursive approach with conventional matrix multiplication.
    b. Optimizing parallel performance requires careful adjustment of parameters k and k'.
3) Matrix Size:
    a. Greater matrix sizes yield the best results from Strassen's multiplication.
    b. Smaller matrices show limited speedup because of excessively long execution times.
4) Task Management in Recursive Sequence:
    a. Recursive calls provide the advantage of parallelization but they may also cause overhead.
    b. For effective parallelization recursion termination levels must be adjusted precisely.

Code Compilation and Execution

The following steps need to be followed to run the code correctly.

1. Login to Grace portal
2. Load the compiler module intel prior to compiling your program

   ```
   module load intel
   ```

3. Compile the program with OpenMP pragmas using below line.

   ```
   icc -qopenmp -o parallel_strassen_matrix_mul.exe parallel_strassen_matrix_mul.cpp
   ```

4. To run code in line directly use following line with 3 arguments – k, k’ and num_threads

   ```
   ./parallel_strassen_matrix_mul.exe <k> <k’> <num_threads>
   ```

   To run everything at once run the grace file shared with this code using following syntax.

   ```
   sbatch grace_job.parallel_strassen
   ```