

# Assignment 1 Report

Varun Gopal

CO22BTECH11015

The main objective of the program is to develop a multi-threaded solution to find the square of matrix using the matrix multiplication algorithm.

This program is implemented using the creation of multiple threads and assigning each thread some rows of elements to calculate. The main function distributes all the  $n$  rows of the matrix among  $k$  threads for each method in a different way. After the completion of all threads for each method, the parent outputs the result of each method and time taken in their respective output files.

## **Explanation of Code: -**

First 'N' (Number of rows), 'K' (Number of threads) and the corresponding  $N \times N$  matrix (a) are read from the input file{inp.txt}. Now we find the square of the matrix using matrix multiplication. We use a function named dot frequently which computes the dot product of the row and the column passed to it. To do this We first create an array to store the ids of the  $K$  threads. These are used by the main function to identify each thread and wait till all the threads finish their execution. This distribution is done by distributing the rows uniformly among the threads. To Make the algorithm run faster we divide the total number of elements to be computed in the C matrix viz.  $N \times N$  among  $K$  threads. We follow different distribution schemes for different methods:

Distribution for Method 1(Chunk): - Thread 1 gets 1 to  $N/K$  rows, Thread 2 gets  $N/K+1$  to  $2N/K$ ..... and so on. The remaining  $N\%K$  rows are allotted to the last thread. More Theoretically, Thread  $K$  gets  $(K-1) N/K+1$  to  $N$  numbers. Where all the arithmetic is performed in terms of integers.

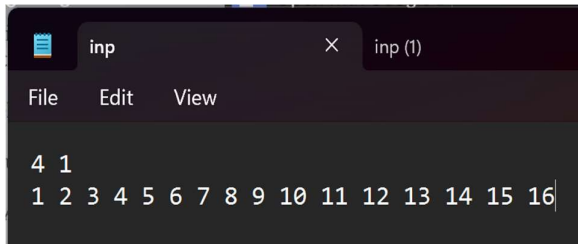
Distribution for Method 2(Mixed): - Thread 1 gets 1,  $k+1$ ,  $2 \times k+1$ ...rows, Thread 2 gets 2,  $k+2$ ,  $2 \times k+2$  .... rows and so on. The remaining  $N\%K$  rows are allocated to the last thread same as in the case of method 1. This random

Distribution for Method 3: - We use a distribution technique which can be described as follows: - First we find the norm of all the row vectors of the matrix and sort them according to their values. Now the time taken to perform computations on larger values is more than the small values. Therefore, the elements in C which are computed from a row of higher norm take more time to execute. Therefore, we assign the rows uniformly among all the  $k$  threads with uniform norms.

The input matrix is squared and printed in their respective output files along with the time taken to perform computations for each method on the top.

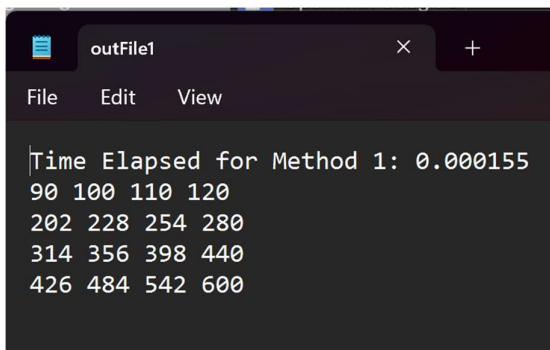
Illustration of the code for N=4 and K=1 is given with matrix being 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 in the row major order.

Input File: inp.txt

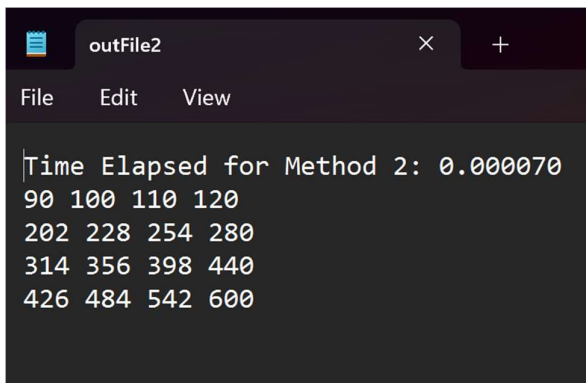


```
inp
File Edit View
4 1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

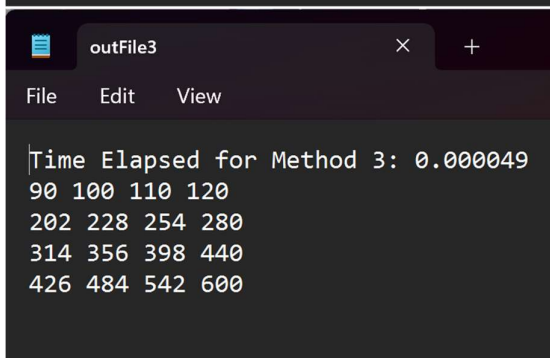
Output Files:- outFile1.txt, outFile2.txt, outFile3.txt



```
outFile1
File Edit View
Time Elapsed for Method 1: 0.000155
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```



```
outFile2
File Edit View
Time Elapsed for Method 2: 0.000070
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```



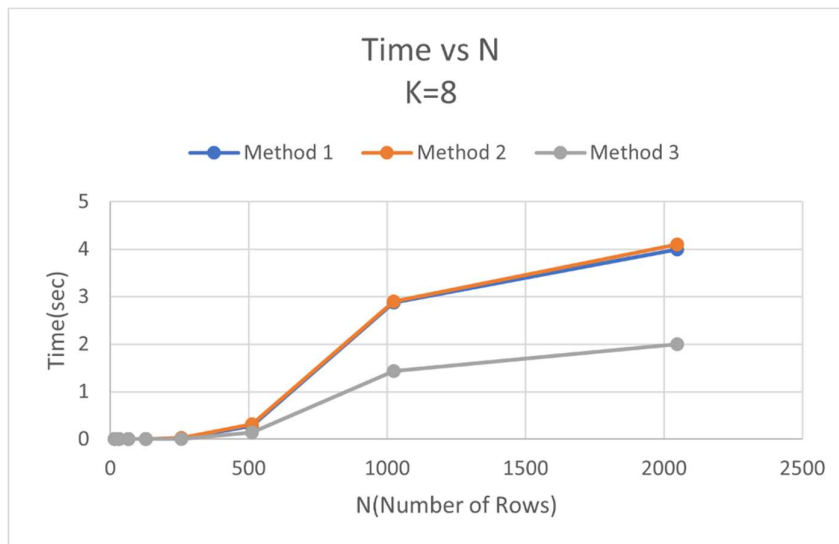
```
outFile3
File Edit View
Time Elapsed for Method 3: 0.000049
90 100 110 120
202 228 254 280
314 356 398 440
426 484 542 600
```

# Analysis of Output:

The running times of the program were measured by varying N and K for each execution using all the 3 methods. It is an improvement over the traditional way of programming where we create a single threaded process and compute all the elements of the matrix C. The traditional way finds all the entries of numbers sequentially, runs on a single core at a time. This led to large running times of the programs. Multithreading improves the utilization of CPU cores for a task to complete. All the K threads run on different cores at the same time. As all the processes will be running parallelly, execution time decreases as all the sets of numbers are checked parallelly.

Below is the analysis of how the running time of the program changes with variations in N(size) and K(Threads) for all the 3 methods: -

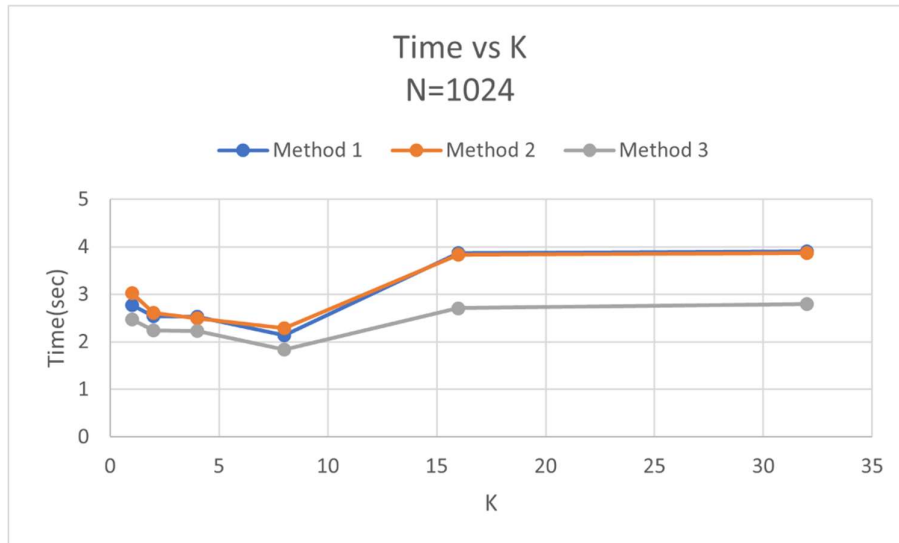
## 1.Time vs Size, N:



From the above graph, for a fixed number of threads (viz. K=8 in this case), the running time of the program increases for all the 3 methods as number of rows of the square matrix input increases. This is obvious due to the fact that the program has to perform more computations as n increases.

Method 1 and Method 2 show similar execution times as N varies. However, Method 3(Normalisation method) can be seen as an improvement over the first 2 methods as the work is getting more uniformly distributed among the threads.

## 2. Time vs Number of Threads, K:



From the above graph, as the number of threads increases for the given value of  $n$ , execution time initially decreases. However, when the number of threads exceeds the number of cores on the CPU, the time increases as management of threads takes excess time and dominates the computational time taken. Therefore, when run on a 10 core CPU, Execution time initially decreases up to  $K=8$  and then increases for  $K=16$  and  $32$ . On a whole, the execution times decrease as the threads run in parallel and divide the computations among themselves. As the number of threads increases, the entries computed by each thread decrease thereby decreasing the running time of the whole task.

Method 1 and 2 take comparable execution times and the optimal method of the 2 depends on the dataset that we chose. Method 3 is better as it normalises the rows before computations.

### Extra Credit Method:

We distribute the rows to each thread giving weightage to their norms. Refer [Norm \(mathematics\) - Wikipedia](#) for Euclidean norm and its effect on matrix multiplication. We ensure that the whole load of norms is getting uniformly distributed across all the threads.

So that the throughput of the overall program is increased. However, this improvement in performance is visible when the values of the matrix are widely varying. But if the order of norms of all the rows are similar much improvement is not observed.  $N^2$  extra computations are to be performed to find the norm of all the rows initially. However, this can be compensated if the values in the matrix follow a distribution with a higher variance.

### Summary of the Report:

The Assignment expects us to find the square of the matrix using multithreading with various different ways of distribution of data. We take advantage of the multicore system by running different threads on different cores parallelly.