PARALLEL AND DISTRIBUTED COMPUTING REPORT
PROGRAMMING ASSIGNMENT 2

System/ Test Platform:

VM ware: Ubuntu 16.04
RAM: 4 GB
Memory: 50 GB
Processor: Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 2
OS- type: 64-bit
VM Ware is being run on Windows OS.

Problem 1:

Parallel matrix multiplication

Number of threads = P

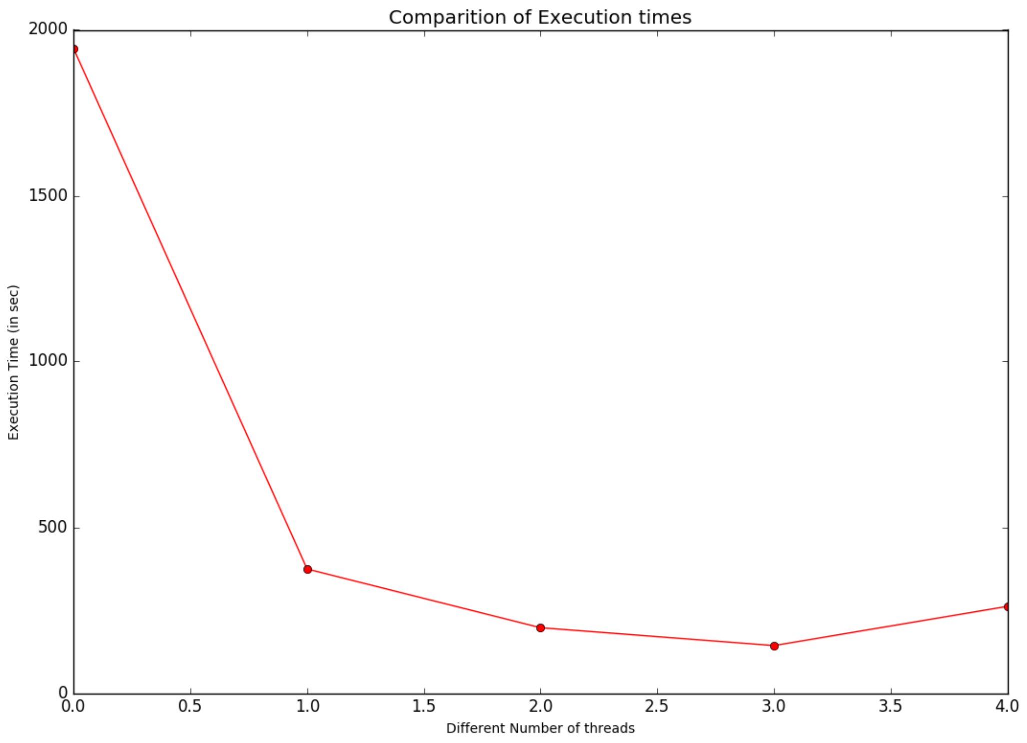
P =1; Execution Time = 1943.48 secs

P = 4; Execution Time = 374.357 sec

P = 16; Execution Time = 197.325 sec

P = 64; Execution Time = 143.43 sec

P = 256; Execution Time = 216.711 sec



In the above plot, 0 – 1 Thread, 1 – 4 Thread, 2 – 16 Thread, 3 – 64 Thread, 4 – 256 Thread
[0, 1, 2, 3, 4] are the indices on the x-axis.

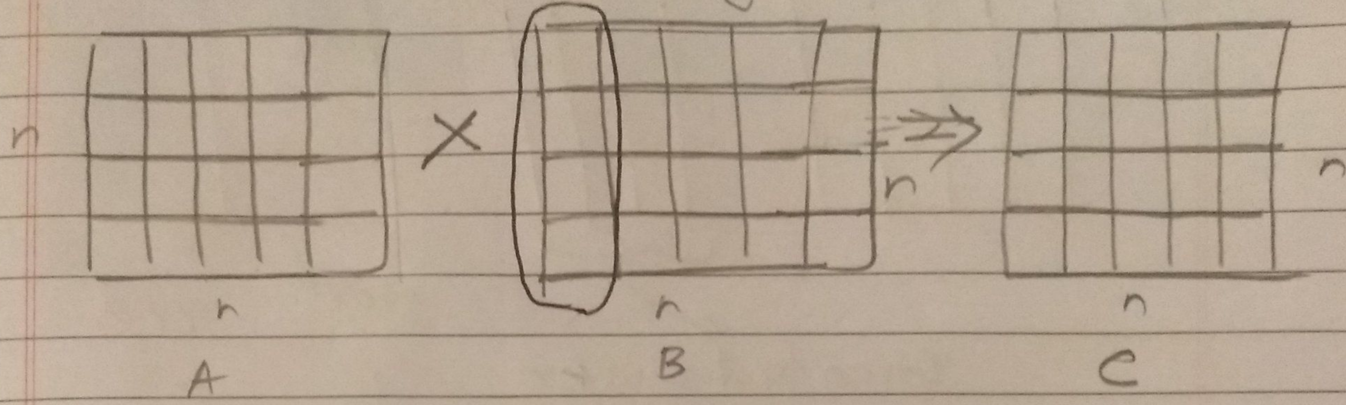
Based on the plot, we can see that as number of threads increases, the execution time decreases drastically.
We can see that from thread 1 to thread 4, the execution time reduces drastically. From thread 4 to thread 16 there is reduction in execution time but it is not as great as from thread 1 to thread 4. Similarly, from thread 16 to thread 64, there is reduction in time but not as much.

In my system, from thread 64 to thread 256, there is slight increase in the execution time, which is counter-intuitive. But I think that is because the number of cores in my system as less as compared to 256 threads. And hence, the threads are run one after the other.

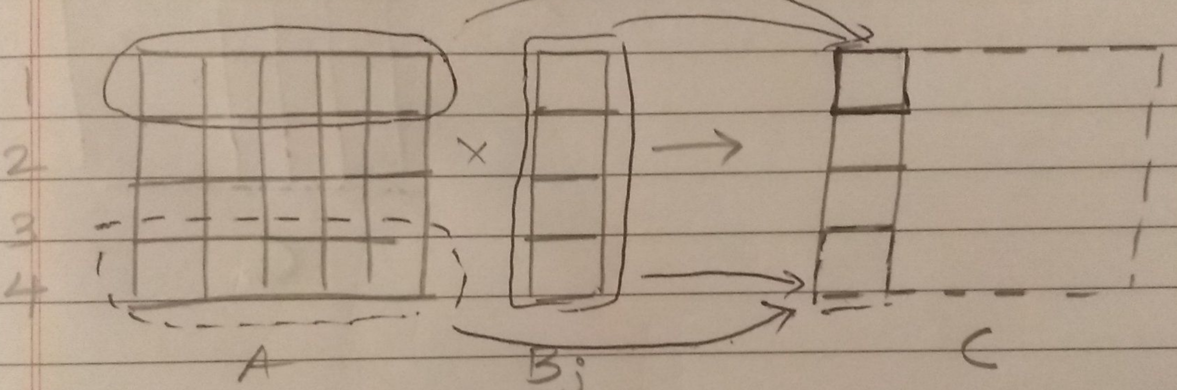
The explanation for how the matrix multiplication is done using threads is given below in the attached pages

The basic idea is that we partition the columns of the B matrix, and give it to each thread. Thus, each thread is responsible for a set of columns of the output. As there is no overlap between the threads, locking of any kind is not necessary

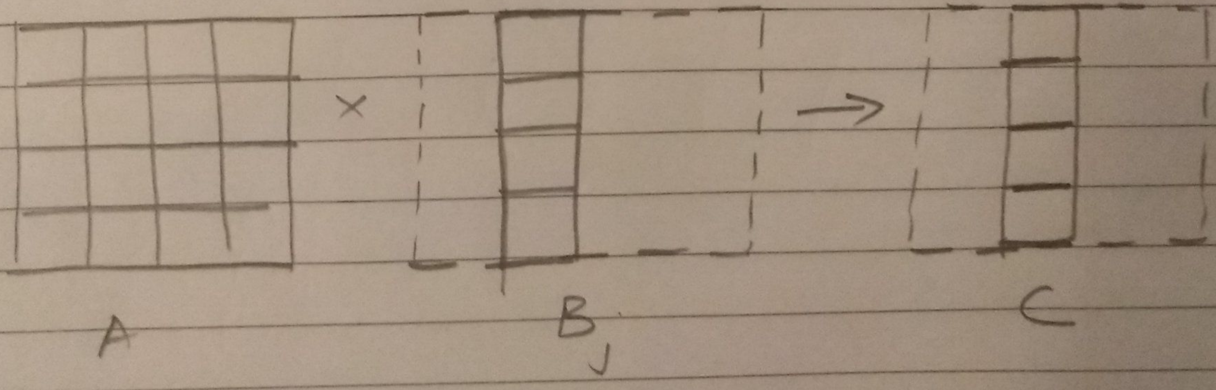
Matrix Multiplication using pthreads :-



Thread 1 :

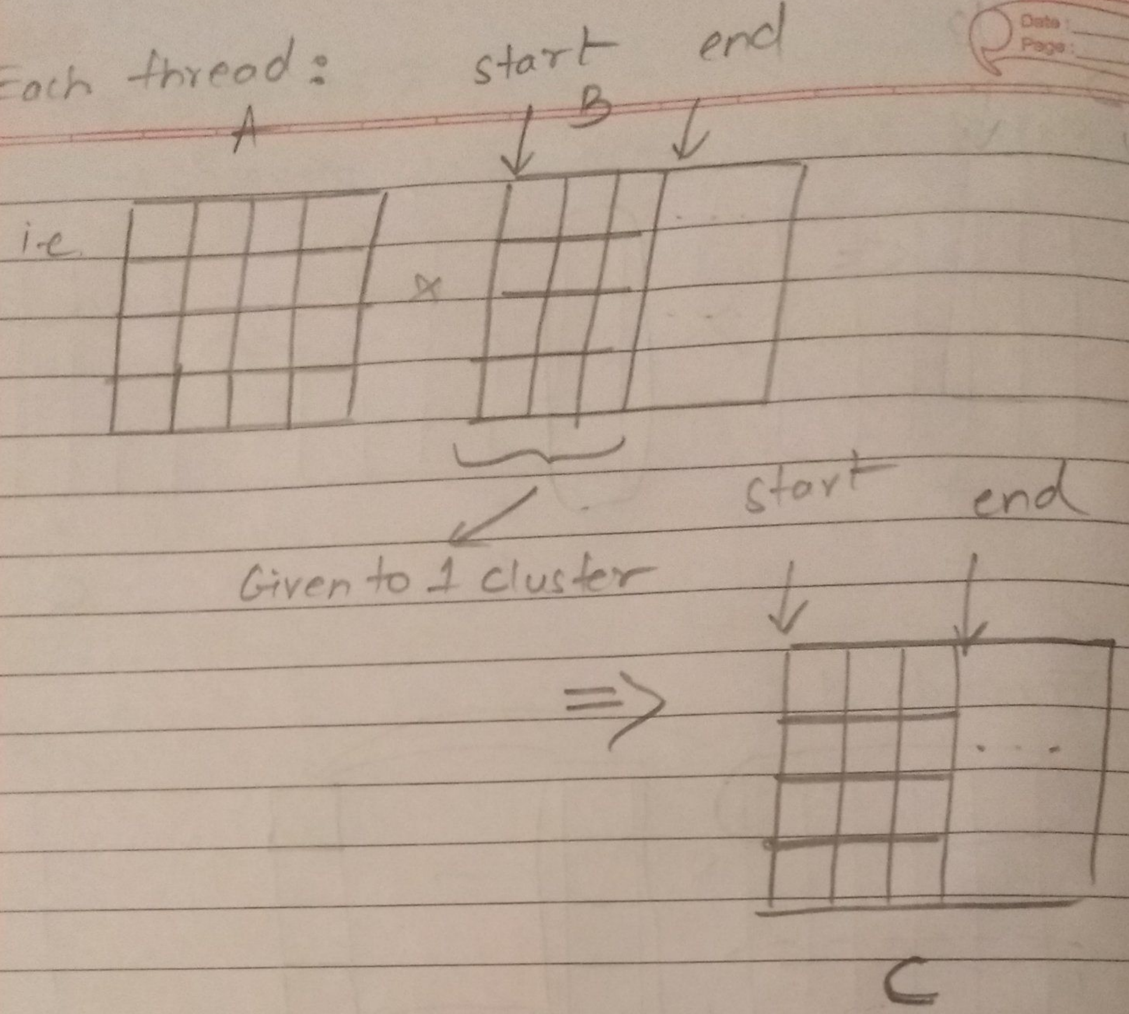


Thread 2 :



Thus, each thread is responsible for particular column of the output matrix C
In our implementation, each thread is given a set of columns instead of 1.
The assignment of columns to each thread
start = thread_id * n/p
end = (thread_id + 1) * n/p − 1

For Each thread:



Problem 2:

Parallel K-means

Execution time has been averaged over 5 runs.

Number of threads = P

Number of iterations = 50

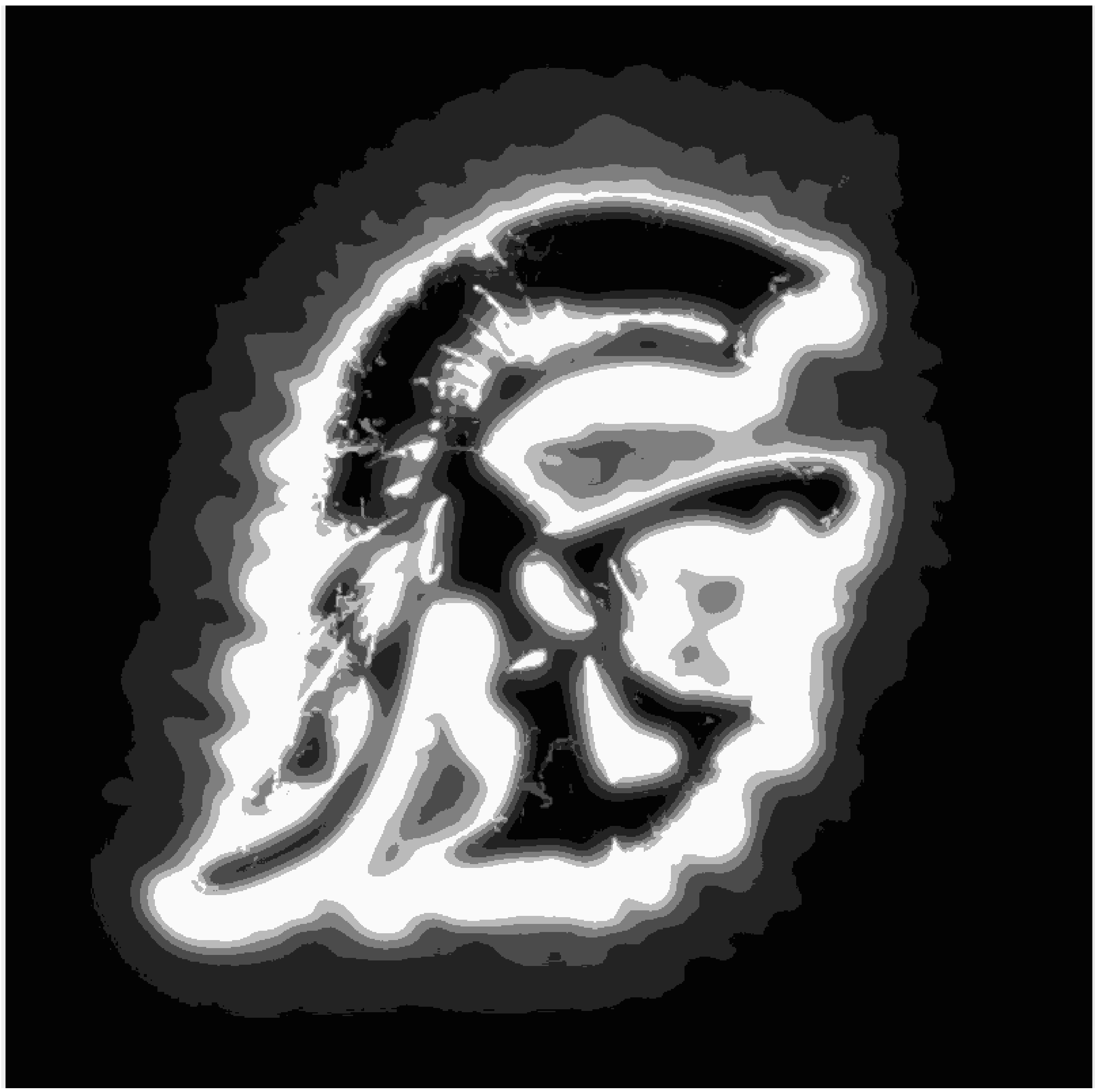
P = 4, Execution Time = 1.229 sec

P = 8, Execution Time = 1.17 sec

Naive Implementation of K-means (without parallelism), Execution Time = 1.597 sec

Input Image:

Output Image:



Reasoning:

We can see that the parallel implementation using pthread model is faster as compared to the naive k-means implementation.

In pthreads, each thread is responsible for a subset of data elements.
Each thread computes to which cluster each of its elements belongs to.
As there are more than one threads, the process of assigning elements to individual clusters is distributed and hence it is completed in less amount of time.
In the end, we only have to combine the results of each thread for each cluster to compute the final mean.

Thus, due to multiple threads, we can get faster results.

We can also observe that as the number of threads increases, the execution time decreases.