**PDC PROJECT REPORT**

**KMEANS CLUSTERING**

**Group Members:**

Rana Wahaj -> K21-3281

Muhammad Hassan -> K21-4577

Ibad Ur Rehman -> K21-4652

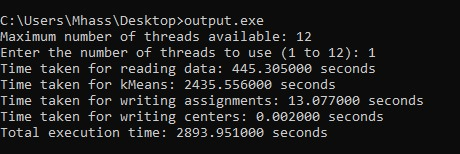
**Introduction:**

In this project, we have parallelized the K-means clustering algorithm using two useful parallel computing techniques: OpenMP and MPI. K-means clustering is an algorithm that partitions data points into clusters based on their similarity. This project's purpose is to show the effectiveness of parallel processing in improving algorithms such as K-means clustering. OpenMP uses shared memory parallelism, employing multiple threads to execute at the same time. MPI uses distributed memory parallelism, communicating and coordinating across multiple processes. We evaluate the scalability of the K-means algorithm using the two parallelization techniques, considering the efficiency of each method. Additionally, we will apply these techniques to large datasets to assess their performance under real-world data.  
  
We will consider the OpenMP technique and its performance first.

**For 100 million rows (5.56 GB Data): -**

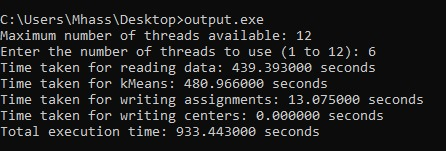
**OpenMP:**

**1 thread:**



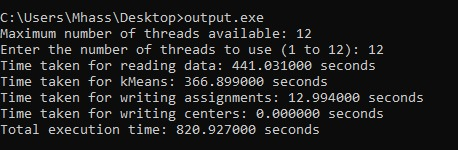
Time: 2435.56 s (about 40 and a half minutes)

**6 threads:**



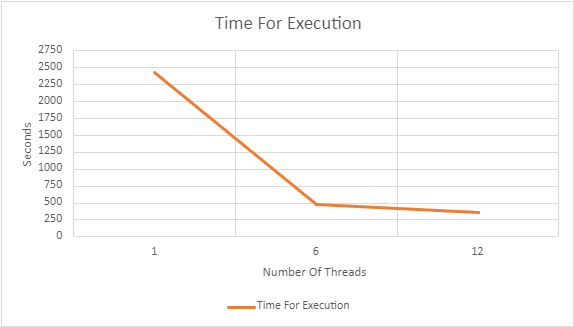
Time: 480.97 s (about 8 minutes)

**12 threads:**



Time: 366.90 s (about 6 minutes)

Here’s The Graphical Representation:

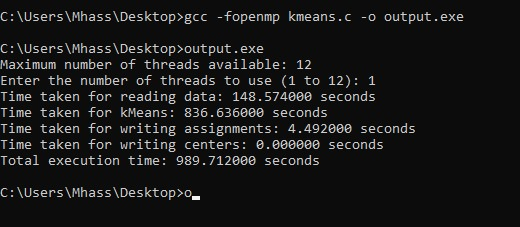


The results obtained from running the K-means algorithm on a dataset of 100 million rows or 5.56 GB using OpenMP show a significant improvement in performance as the number of threads increases. With a single thread, the algorithm took about 40 and a half minutes to complete, by using 6 threads, the time was reduced to about 8 minutes. With 12 threads, we got a reduced time of around 6 minutes. This improvement in processing time is due to concurrent execution of tasks due to multithreading. The algorithm uses computing resources more efficiently by dividing the work among multiple threads. It can be seen that while increasing the number of threads by more than 6 continues to improve the processing time, the rate of improvement decreases. This is due to the increase in parallel overhead and resource contention between running threads. Finding a balance between the number of threads and the parallel overheads is important for optimal performance.

**For 50 million rows (1.73 GB Data): -**

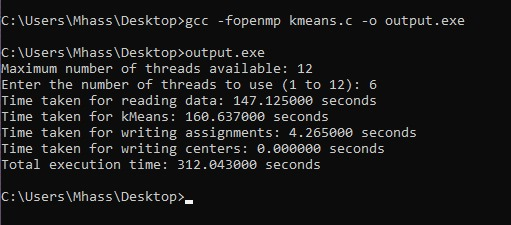
**OpenMP:**

**1 thread:**



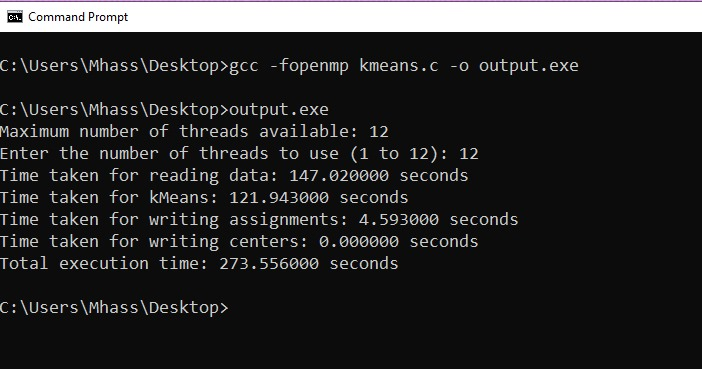
Time: 836.64 s (about 14 minutes)

**6 threads:**



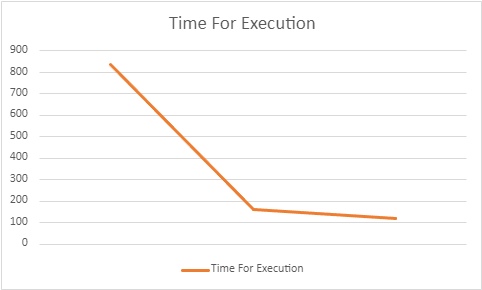
Time: 160.64 s (about 2 and a half minutes)

**12 threads:**



Time: 121.94 s (about 2 minutes)

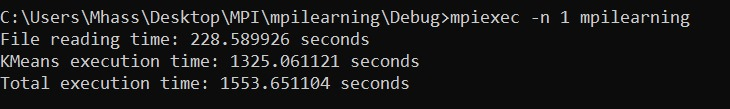
Here’s the graphical representation



The results indicate that the workload benefits from parallelization with OpenMP, as evidenced by the significant reduction in execution time when increasing the number of threads. However, there is a diminishing return on performance improvement as the number of threads increases, which is a common observation due to factors such as increased contention for resources and additional overhead associated with managing more threads. The optimal number of threads often depends on the specific characteristics of the workload and the underlying hardware architecture.

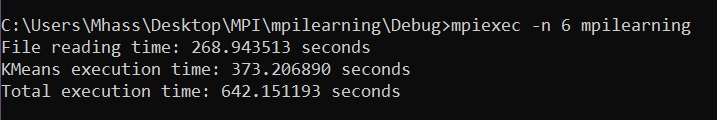
**MPI:**

**1 Processors:**



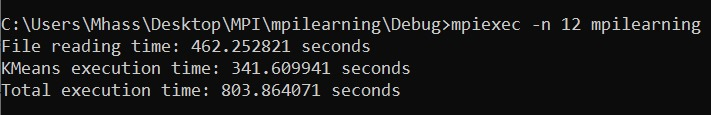
Time: 1325.06 s (about 22 minutes)

**6 Processors:**



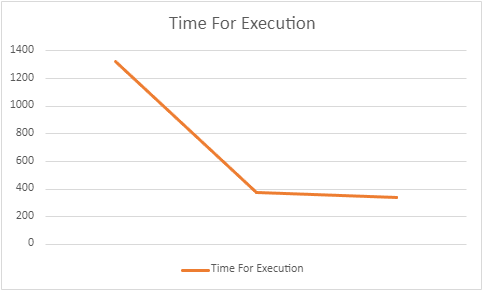
Time: 373.21 s (about 6 minutes)

**12 Processors:**



Time: 341.61 s (about 5 and a half minutes)

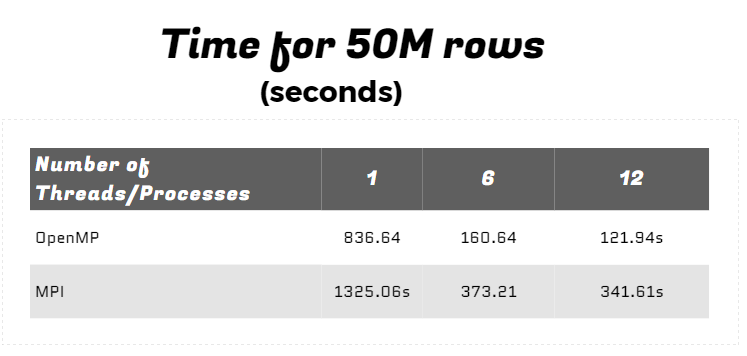
Here’s the graphical representation



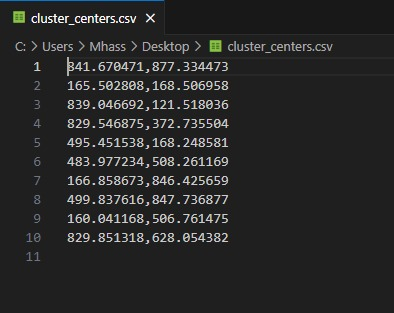
For a single processor, the program takes 1325.06 seconds (approximately 22 minutes) to complete. However, as the number of processors increases to 6, the execution time decreases significantly to 373.21 seconds (about 6 minutes). Further, with 12 processors, the time reduces even more to 341.61 seconds (around 5 and a half minutes).

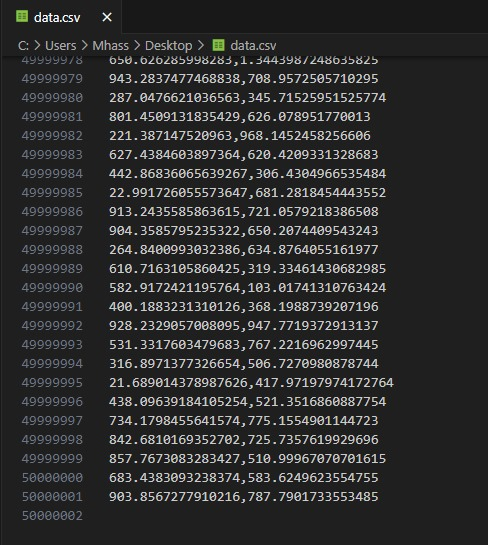
This reduction in execution time indicates the effectiveness of parallelization in distributing the computational load across multiple processors, resulting in faster program execution. The observed trend is a common characteristic of parallel computing, where increased parallelism leads to improved overall performance and reduced time-to-solution for computational tasks.

**Comparison between MPI and OpenMP:**



**Output:**





**CPU Utilization and Temperatures:**

