Submission Report

Title of the Work

**Parallel Implementation of K-Means Clustering for Big Data**

Author(s):

Name 1:Vandrangi Sai Vivek

Roll No: 2023BCS0161

Name 2:T. Arun Lingeswar

Roll No:2023BCS0173

Course Instructor**:**Dr. John Paul Martin

Department of Computer Science, IIIT KOTTAYAM

Date of Submission : 10/10/2025

Abstract:

Our Topic is Basically Parallelizing K-Means clustering.

K means clustering is a widely used unsupervised classification model in which a group of datapoints are mapped to nearest common point (also called centroid)

These centroid mapping keeps on going till the points are mapped correctly. there is a certain calculative kind of thing for it however our main focus is not on that what happens here is for small data it is ok but when we go for big data like a million or more points the time complexity gets huge we can define that as O(T.N.K.D)

Where T is the number of iterations (consider how many times we checked like we keep on checking multiple times to map all points correctly), N is the dataset size, K is the number of clusters (group of points) D is dimensionality. This takes a large time .so to reduce this we parallelize it. So what we do is we distribute the workload across multiple threads like each thread handles mapping of point to its centroid and accumulates partial sum. which are later taken in next iteration to update centroids. What we have observed later in our results is for small n values it is same but when for large datasets the time decreases drastically. This Parallelizing will help in image segmentation, anomaly detection and market analysis.

1.Introduction:

K-Means is unsupervised clustering algorithm but becomes slow on large datasets.

Series Computation is expensive due to repeated calculation of distance to centroid

Parallelizing using OpenMP makes multi-core CPUs to accelerate clustering

Applications:

image segmentation: dividing images into parts

bioinformatics: study and analyze structure data

network traffic analysis : monitoring and interpreting data flow

finance : analyzing the flow of market.

2.Literature Survey:

In early implementation treated K-Means as a sequential algorithm and applied it successfully on small medium datasets. However, researchers quickly noted that the distance computation takes very huge time (it is maximum in the runtime of total algorithm).

MPI based implementations partition datasets across nodes in a cluster, while MapReduce approaches have been deployed for very large-scale datasets in cloud environments. GPU-based solutions further exploit massive parallelism by distributing distance calculations across thousands of cores.

Despite these advances, OpenMP has emerged as a practical and lightweight method for parallelizing K-Means on shared-memory, multi-core systems, OpenMp allows developers to introduce parallelism at loop level with small changes to serial code while also providing mechanisms (critical sections, reductions) to handle synchronization. Studies and Observations Suggest that OpenMp based K-Means achieves near linear speed for large datasets and maintains the same clustering accuracy as serial execution.

3. Problem Statement and Objectives

3.1 Problem Definition

Efficiently implement and analyze parallel K-Means clustering using OpenMP.

3.2 Objectives

Implement a serial baseline K-Means in C++.

Identify parallelizable segments (point to centroid mapping, centroid updating)

Develop OpenMP implementation with synchronization.

Comparing both runtime of serial and parallel.

4. Methodology and System Architecture

Serial:-

Algorithm:

1.Initialize k centroids randomly

2.Compute distances from each point to all centroids

3.Recalculate each centroid as mean of assigned points

4.stop when centroids stabilize

What happens in background:

Single thread execution processes one point at a time, then one cluster at a time.

Pseudo Code:

initialize centroids[0..k-1] repeat :

{

for each point p:

assign p to nearest centroid

for each cluster j:

centroid[j] = mean(cluster\_j)

}

until convergence

Parallel:-

Strategy:-

Identify Prallel blocks:

Distance calculations are independent per point

Centroid recomputation can be distributed

Use #pragma omp parallel for around point loops

Pseudo code:

// OpenMP Parallel K-Means

do

{

// Parallel Assignment

#pragma omp parallel for

for each point p:

compute distances to centroids

atomically update cluster sums

// Combine results for each cluster j:

centroid[j] = total\_sum[j] / count[j]

} while (centroids change)

**Tools:** C++ with OpenMP, GCC compiler, Linux/Windows platform.

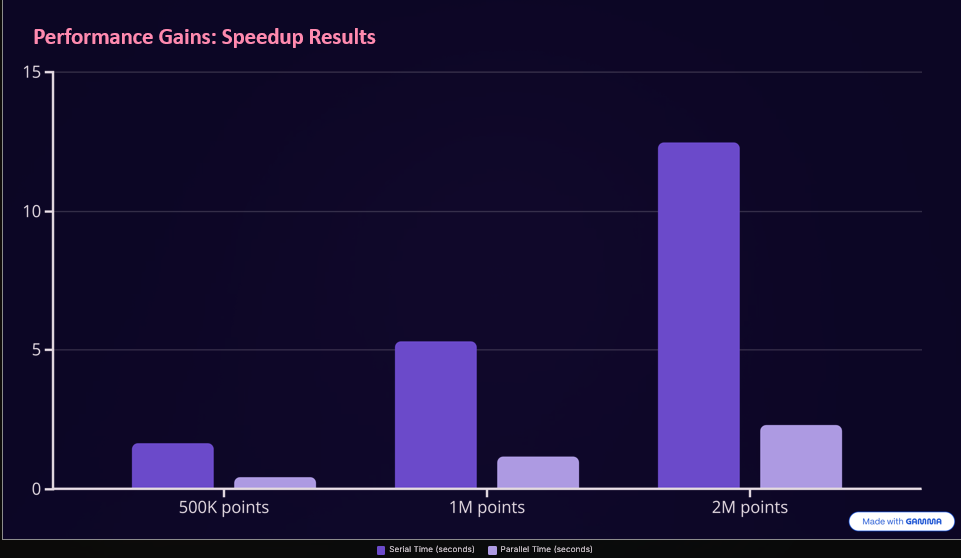
5. Results and Analysis

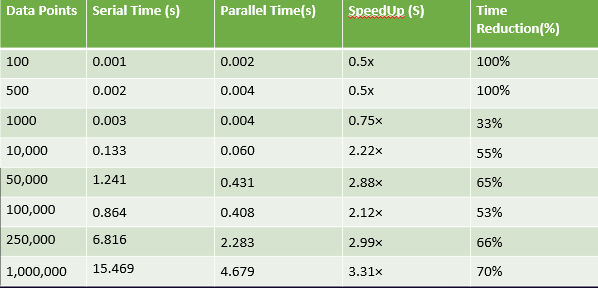
Serial runtime grows linearly with dataset size (O(T·N·K·D)).

Parallel runtime reduces nearly by factor of number of threads (ideal case).

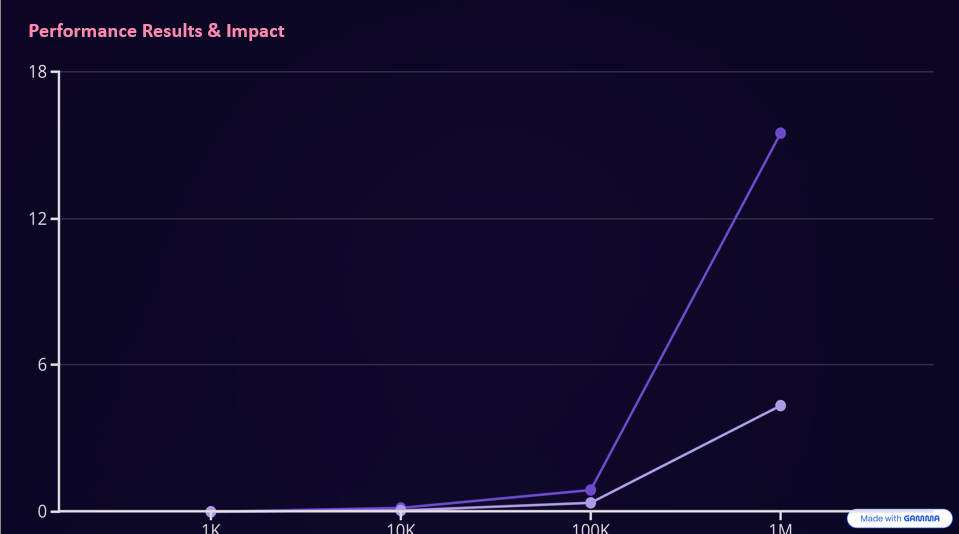
Performance table and graph:

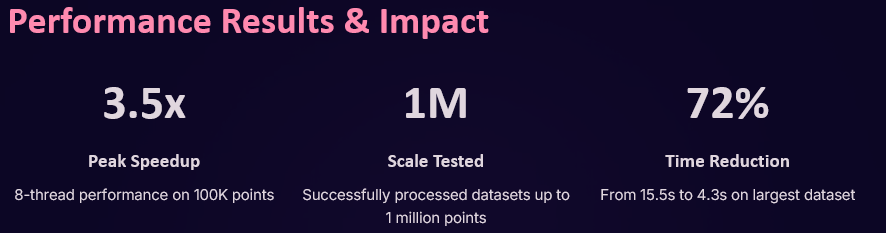
Sir all these are actually kept in our ppt so I am just pasting a screen shot the colours may differ. hope you understand.





\*For very small datasets (100–500 points), parallel runtime is slightly higher than serial due to **thread creation and overhead**, hence negative or <1 speedup





6. Discussion and Observations

Parallelization provides significant speedup for large datasets.

Speedup improved with dataset size, as overhead becomes less dominant.

Scalability is limited by synchronization costs and thread management.

7. Conclusion and Future Scope

Successfully implemented K-means clustering with OpenMp.

Took Linear Time for large datasets.

Future work: we are able implement through MPI (distributed systems) and CUDA(GPUs) for handling terabyte-scale data.

References:

Quinn, M. J. *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill, 2004.

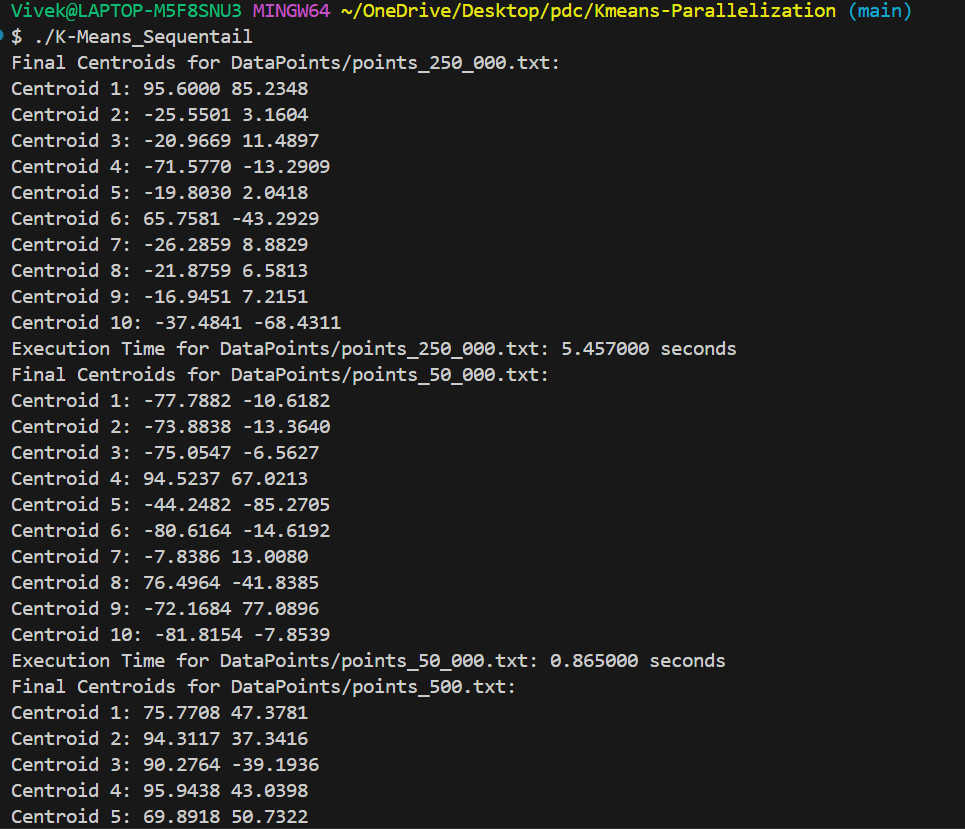
OpenMP Architecture Review Board. *OpenMP API Version 5.0*, 2023.

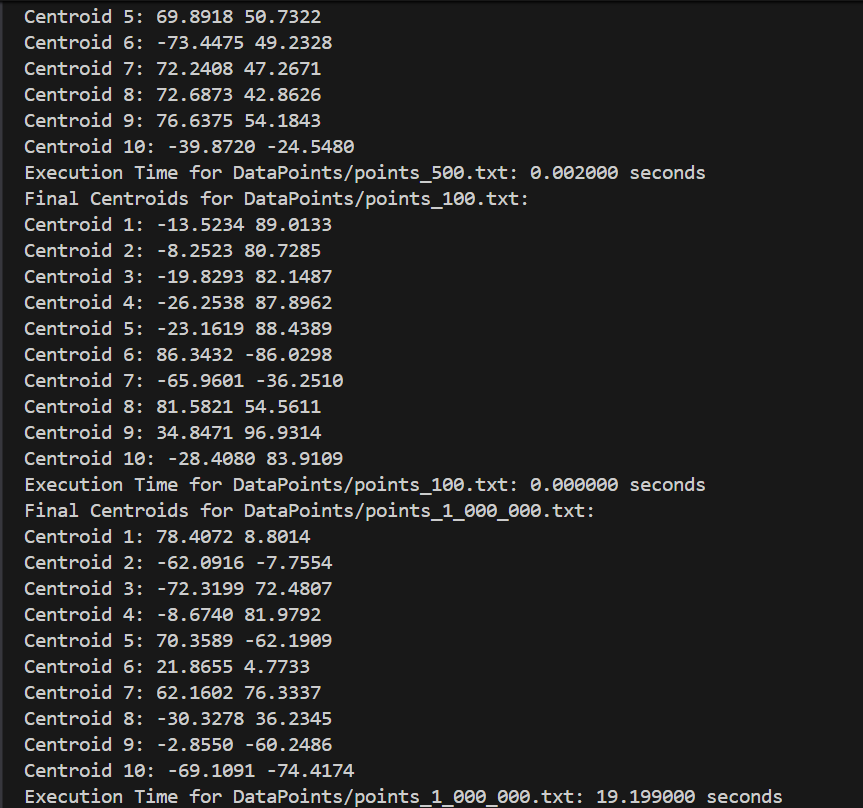
Youtube videos.

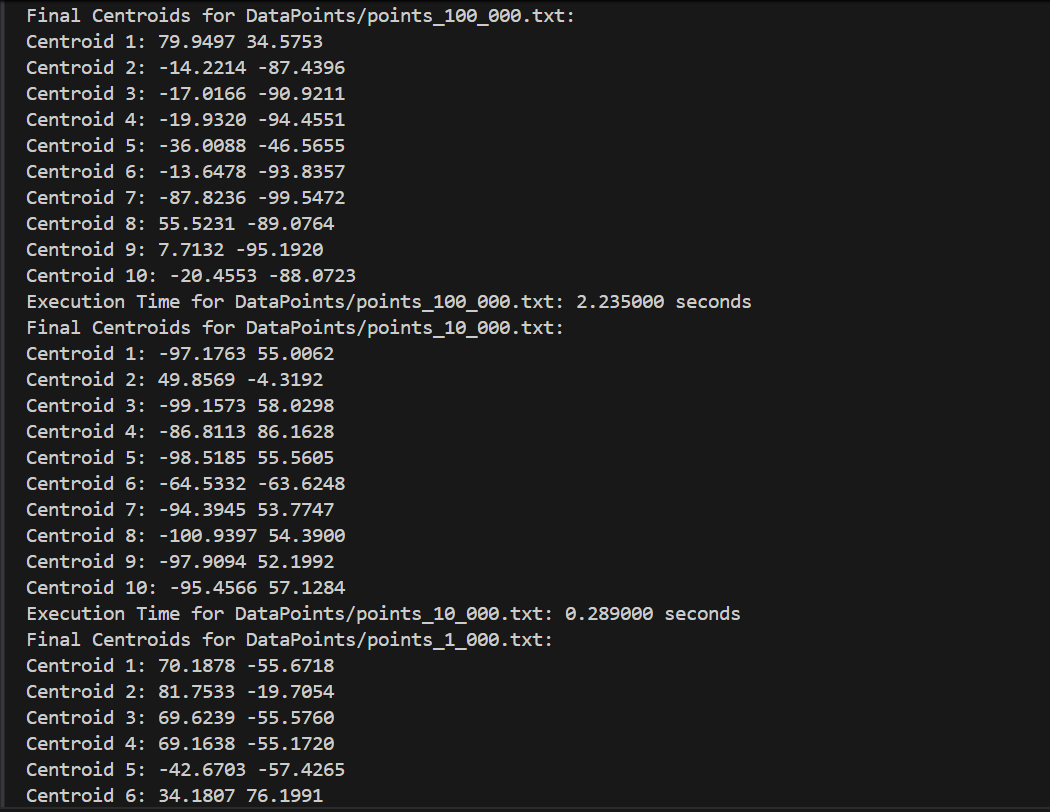
Full codes :

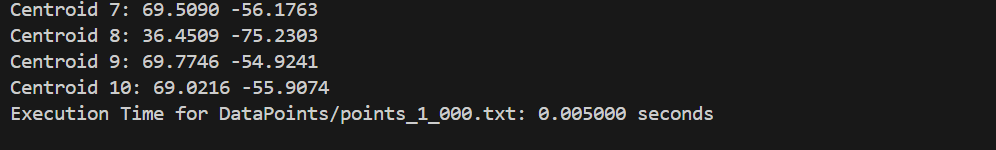
Sequential :

Output:-



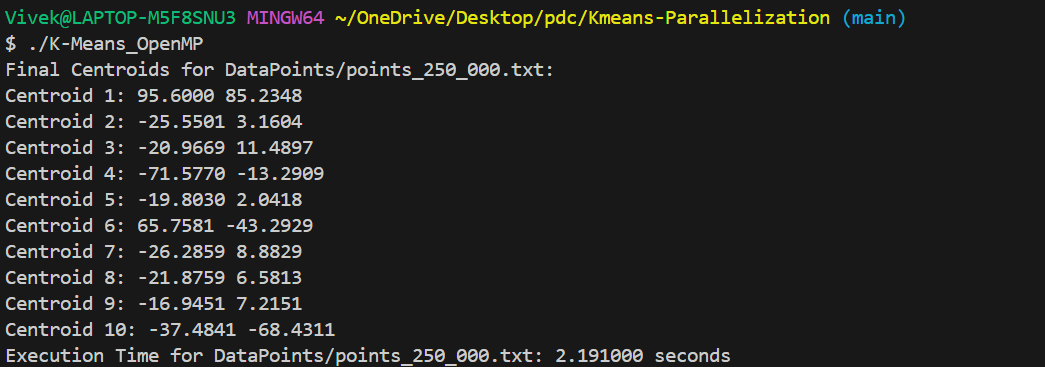


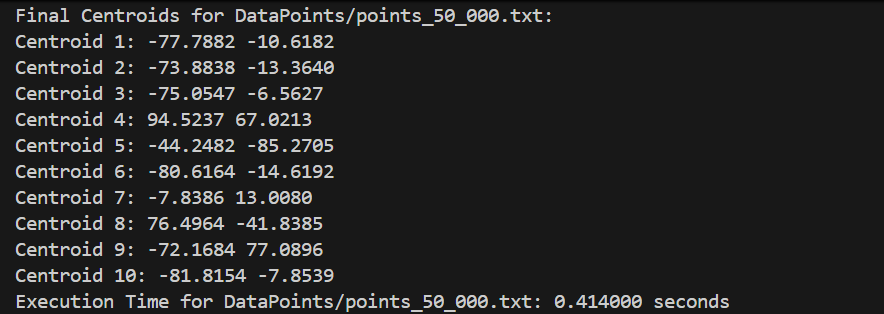


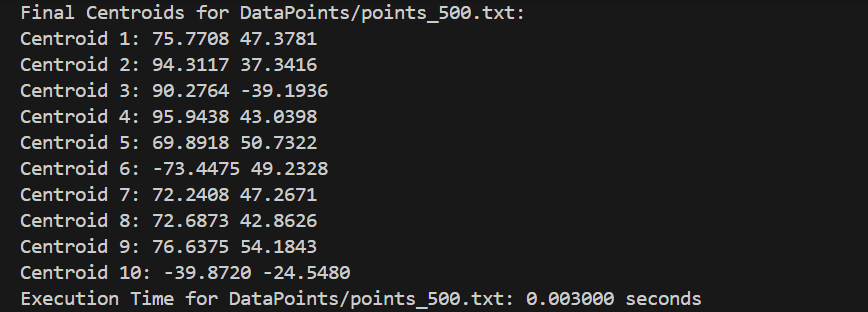


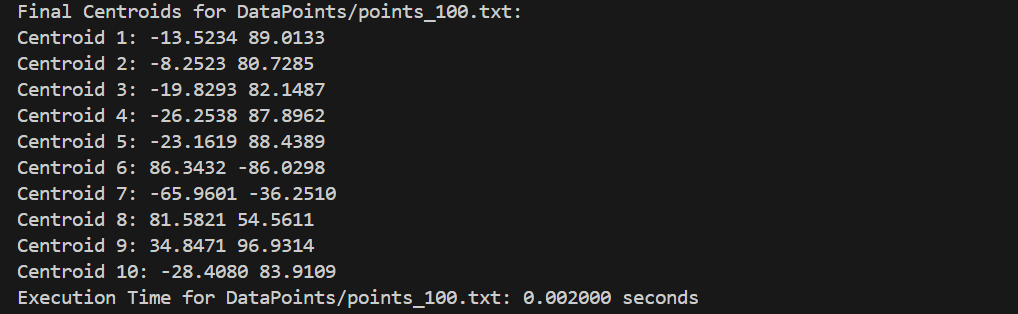
Parallel Program:-

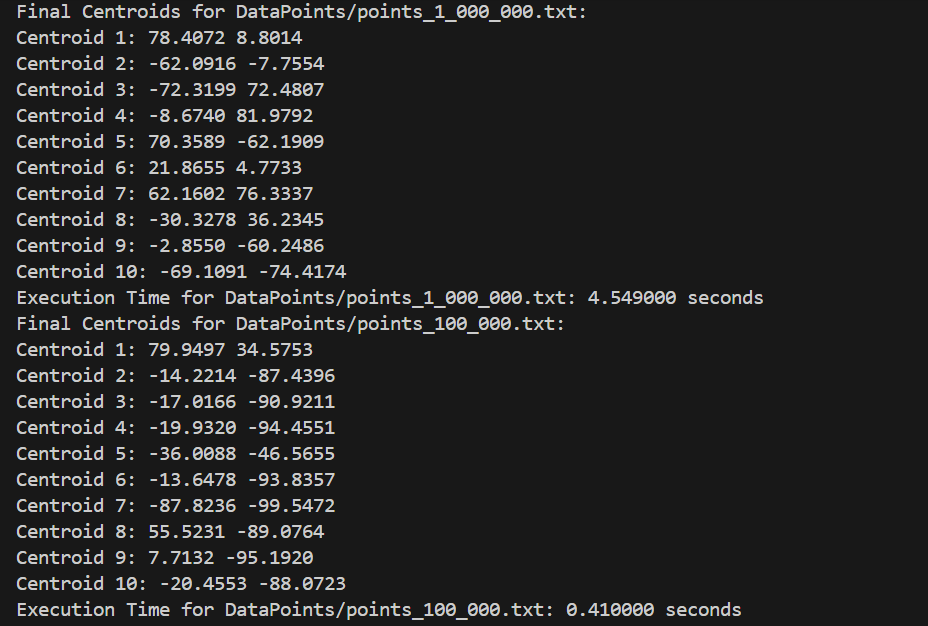
Output:-

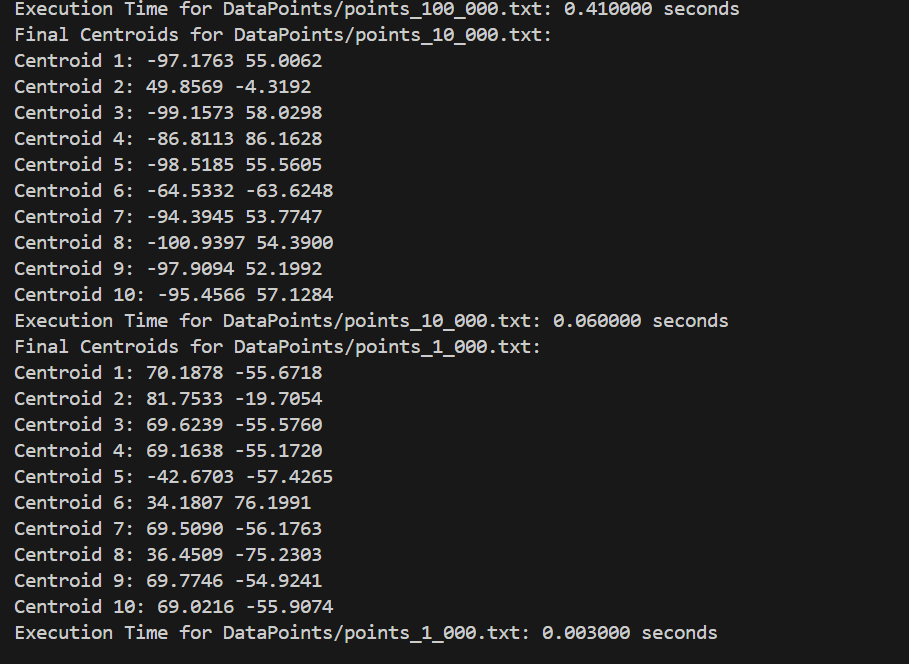












Github Repo: https://github.com/saivivek45/Parallel\_KMeans\_Clustering.git

Thank You………..