A Project Report

On

K-Means Clustering Using OpenMp

BY

BITTU JAISWAL

SE23MAID022

Under the supervision of

Dr. Praveen Kumar Alapati

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

DEGREE OF MASTER OF TECHNOLOGY

AI5307: DISSERTATION PROJECT

ECOLE CENTRALE SCHOOL OF ENGINEERING

MAHINDRA UNIVERSITY

HYDERABAD

(Month 2024)

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, **Dr. Praveen Kumar Alapati**, for their invaluable guidance, encouragement, and support throughout the course of this project. Their expertise and insight has been instrumental in shaping this work.

I would like to express my profound gratitude to **Prof. Dr. Arun Kumar Pujari, HOD, CSE Department, Mahindra University** for their unwavering support and motivation during my academic journey. This achievement would not have been possible without their constant encouragement and belief in me.

Lastly, I am also deeply grateful to the faculty members of **Ecole Centrale School of Engineering, Mahindra University**, for providing a conducive learning environment and access to the necessary resources. Special thanks to my peers and colleagues for their continuous encouragement and fruitful discussions.

Bittu Jaiswal

Master of Technology, [Your Department]

Mahindra University

ECOLE CENTRALE SCHOOL OF ENGINEERING

MAHINDRA UNIVERSITY

HYDERABAD

Certificate

This is to certify that the project report entitled "**K-Means Clustering Using OpenMp**" submitted by Mr/Ms. **Mr. Bittu Jaiswal** (Roll No. **SE23MAID022**) in partial fulfillment of the requirements of the course **AI5307** under my supervision.

Dr. PRAVEEN KUMAR ALAPATI

ECOLE CENTRALE SCHOOL OF ENGINEERING, Hyderabad
Date: 26/12/2024

Abstract

Cluster analysis is a crucial technique in scientific research and business applications for organizing data into meaningful groups. The K-Means clustering algorithm is widely recognized for its efficiency in partitioning datasets but faces limitations when applied to large-scale datasets due to its iterative nature and high execution time. To address these challenges, a parallel implementation of the K-Means algorithm is presented using OpenMP, which significantly reduces execution time while maintaining clustering accuracy.

This work focuses on optimizing K-Means by leveraging parallel computation on multicore systems to handle large volumes of data points efficiently. Experiments are conducted on randomly generated 2D datasets containing up to 1,000,000 points, with varying numbers of clusters (K) and threads. The proposed approach demonstrates superior performance by reducing computational overhead and ensuring faster convergence without compromising the precision of the clustering results

Contents

1 Introduction 6
1.1 Overview 6
1.2 Motivation 6
1.3 Real World Application 6
2 Problem Definition 7
2.1 Problem Description 7
3 Proposed Solution 8
3.1 Expected Outcome 8
3.2 Challenges 9
4 Implementation 9
4.1 Implementation Overview with OpenMP Features 9
4.2 Parallel Regions 9
4.3 Worksharing 9
4.4 Runtime Functions/Variables 9
4.5 Sequential vs. Parallel Programs 10
4.6 Flowchart 10
5 Results 11
6 Conclusion 12
7 References 12

1 Introduction

1.1 Overview

Clustering is a foundational data mining technique widely used to analyze and group data. It identifies patterns and structures within datasets, dividing them into clusters where elements within the same group are more similar to each other than to those in other groups. Among clustering algorithms, K-Means is one of the most prominent due to its simplicity and efficiency. It works by iteratively assigning data points to clusters based on their proximity to centroids, calculated using the Euclidean distance.

The implementation of K-Means on large-scale datasets can be computationally expensive, particularly as the number of data points, clusters, or dimensions increases. To address this, parallel processing techniques such as OpenMP are utilized. OpenMP allows the parallelization of independent operations, like distance computation and cluster assignments, across multiple CPU cores. This results in significant improvements in execution speed and scalability, making it feasible to handle real-world, large-scale datasets efficiently.

1.2 Motivation

In today's data-driven world, enormous volumes of data are being collected daily across various domains such as healthcare, e-commerce, social networks, and scientific research. Efficiently analyzing and extracting meaningful insights from these datasets is critical for informed decision-making and advancing technologies. Clustering, as a core method of unsupervised learning, plays a key role in uncovering hidden patterns and relationships in data. However, traditional clustering methods often face limitations when applied to massive datasets, especially in terms of computational speed and scalability. The motivation for this project lies in addressing these challenges by leveraging parallel processing techniques to accelerate clustering algorithms. By integrating OpenMP with the K-Means algorithm, this work demonstrates how parallel computing can overcome bottlenecks, enabling faster convergence and handling of real-world, high-dimensional, and large-scale datasets.

1.3 Real World Application

Clustering has a wide array of applications in various industries, solving critical real-world challenges:

- **Market Segmentation:** Businesses group customers based on buying behavior, preferences, or demographics to enhance personalized marketing and improve customer satisfaction.
- **Healthcare:** Clustering helps in patient grouping based on symptoms, diagnoses, or treatment responses, enabling targeted healthcare delivery and optimizing resources.
- **Fraud Detection:** Financial institutions use clustering to identify anomalous patterns in transaction data, which may indicate fraudulent activities.
- **Bioinformatics:** Clustering plays a key role in gene expression analysis, protein structure prediction, and drug discovery by grouping similar biological data.

- **Image Analysis:** In computer vision, clustering is used for image compression, object detection, and image segmentation, enabling more efficient storage and processing.
- **Traffic Management:** Urban planners use clustering to analyze traffic patterns and identify congestion hotspots, improving route optimization and infrastructure planning.
- **Social Network Analysis:** Clustering helps group users with similar interactions and interests, assisting in community detection and targeted advertising.
- **Recommendation Systems:** E-commerce and streaming platforms cluster users based on preferences to offer personalized recommendations, enhancing user experience.

By addressing computational challenges through parallel processing, the project bridges the gap between clustering algorithms and their real-world applications, ensuring scalability and efficiency in handling massive datasets.

2 Problem Definition

2.1 Problem Description

The task involves optimizing the K-Means clustering algorithm to handle large datasets more efficiently by utilizing parallel processing techniques. The goal is to improve the computational performance of K-Means, which is known to be computationally expensive for large datasets, especially when the number of data points, clusters, or dimensions increases. Specifically, this problem addresses the challenges faced in clustering large-scale datasets and explores how parallelism can enhance the speed and scalability of the K-Means algorithm.

Core Problem

The K-Means clustering algorithm works by assigning data points to clusters based on their proximity to centroids. The iterative process involves:

1. Assigning each data point to the nearest centroid.
2. Updating the centroids by calculating the mean of the points in each cluster.
3. Repeating these steps until the centroids converge or a maximum number of iterations is reached.

However, as the size of the data and number of clusters grow, this process can become slow and inefficient, especially when performed serially. The primary issues are:

- **Computational Bottleneck:** Calculating the distance between each data point and every centroid in each iteration is computationally expensive.
- **Scalability:** As the number of data points (n), clusters (K), and dimensions (features) increases, the algorithm's time complexity increases, making it impractical for real-world datasets.

3 Proposed Solution

The proposed solution focuses on optimizing the K-Means clustering algorithm through parallelization using OpenMP. The solution is designed to handle large datasets, improve computational efficiency, and explore the scalability of the algorithm in real-world applications. By leveraging parallel computing techniques, we aim to reduce execution time, especially for large datasets with millions of points and hundreds of clusters. The solution involves the following steps:

- **Data Generation:** Randomly generate a large dataset of points (1 million points in the current implementation), where each point has two coordinates (x, y).
- **K-Means Algorithm:** The algorithm iteratively performs two major steps:
  1. **Cluster Assignment:** Each data point is assigned to the closest centroid.
  2. **Centroid Update:** The centroids are updated by calculating the mean of the points in each cluster.

  These steps are repeated until convergence or a maximum number of iterations is reached.
- **Parallelization with OpenMP:**
  - The assignment of points to the nearest centroid is parallelized using OpenMP to assign data points to clusters concurrently.
  - The updating of centroids is also parallelized, improving efficiency when calculating the mean position of points in each cluster.
- **Performance Testing:** The implementation runs tests with varying numbers of clusters (K) and CPU threads to evaluate the impact of parallelization on execution time and convergence rate.

3.1 Expected Outcome

- **Improved Efficiency:** The parallelized version of K-Means should perform faster than the traditional serial version, particularly as the number of clusters and data points increases.
- **Scalability:** The solution should be scalable to handle large datasets with millions of points and hundreds of clusters.
- **Practical Application:** By demonstrating the ability to process large datasets quickly, the parallelized K-Means algorithm can be applied to real-world applications in various industries, such as healthcare (patient clustering), e-commerce (market segmentation), and social network analysis.

3.2 Challenges

- **Memory Usage:** Managing large datasets in memory, especially when dealing with millions of points, can become a challenge.
- **Load Balancing:** Properly balancing the workload across multiple CPU cores to avoid bottlenecks and ensure optimal performance.
- **Convergence Behavior:** Ensuring that the parallelized algorithm converges correctly and efficiently while minimizing computational overhead.

4 Implementation

4.1 Implementation Overview with OpenMP Features

In this implementation of the K-Means algorithm, OpenMP is used to parallelize key sections of the algorithm, particularly the cluster assignment and centroid update steps. Below is an overview of how OpenMP is utilized in the code.

4.2 Parallel Regions

OpenMP parallelism is introduced with the `#pragma omp parallel` directive. This marks the beginning of a parallel region, allowing multiple threads to execute different parts of the code concurrently. For example, the cluster assignment (`ass_clustr()`) and centroid update (`updat_cntroid()`) functions are executed in parallel regions to improve efficiency when processing large datasets.

```
#pragma omp parallel
```

4.3 Worksharing

To divide the work among multiple threads, OpenMP's worksharing constructs like `#pragma omp for` are used. In this implementation, the assignment of points to clusters is parallelized using `#pragma omp for`. This ensures that each thread handles a portion of the data when computing the distance between points and centroids.

```
#pragma omp parallel for
for (int i = 0; i < n_ponts; i++) {
    // Code to assign points to the nearest centroid
}
```

4.4 Runtime Functions/Variables

OpenMP provides functions such as `omp_get_num_threads()` to get the number of threads being used and `omp_set_num_threads()` to set the number of threads. These functions help in fine-tuning the parallel execution of the program. For example, in this implementation, the number of threads can be set dynamically to optimize performance based on system resources.

```
int my_thread_id = omp_get_thread_num();
omp_set_num_threads();  // Setting the number of threads
```

4.5 Sequential vs. Parallel Programs

This use of OpenMP effectively demonstrates how parallel computing can accelerate the K-Means algorithm, making it feasible to process large-scale datasets quickly.

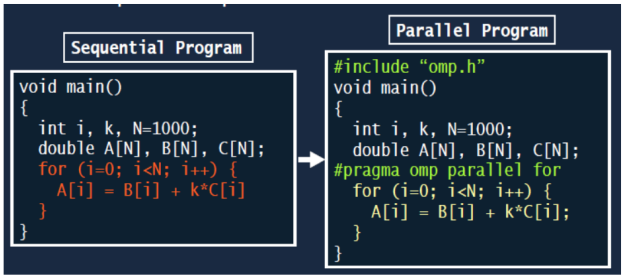
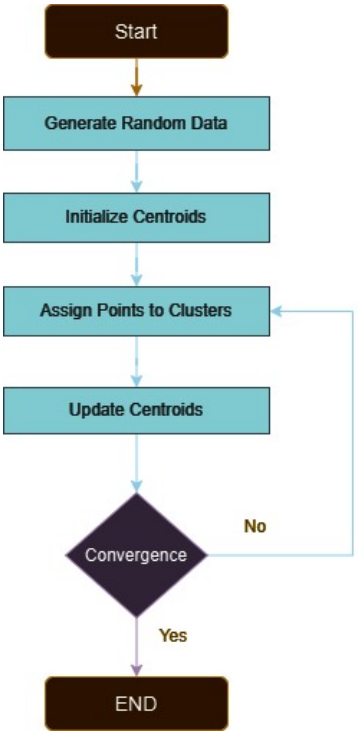
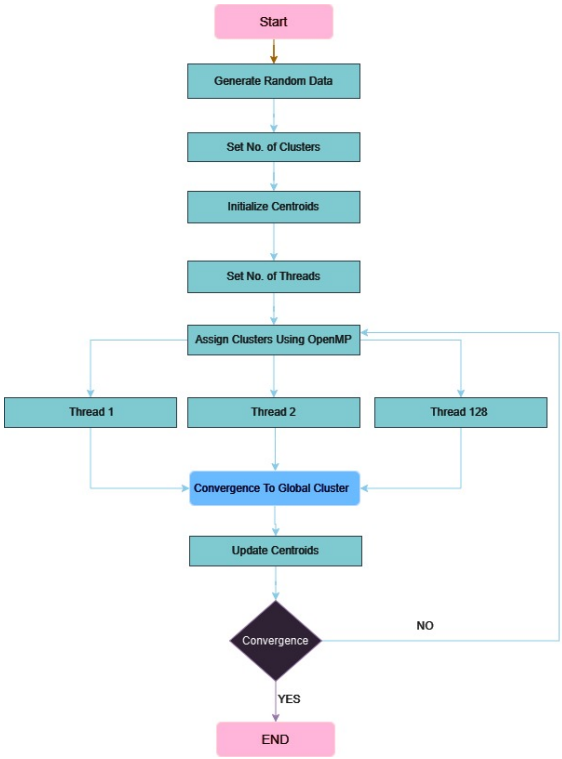


Figure 1: Sequential to Parallel.

4.6 Flowchart



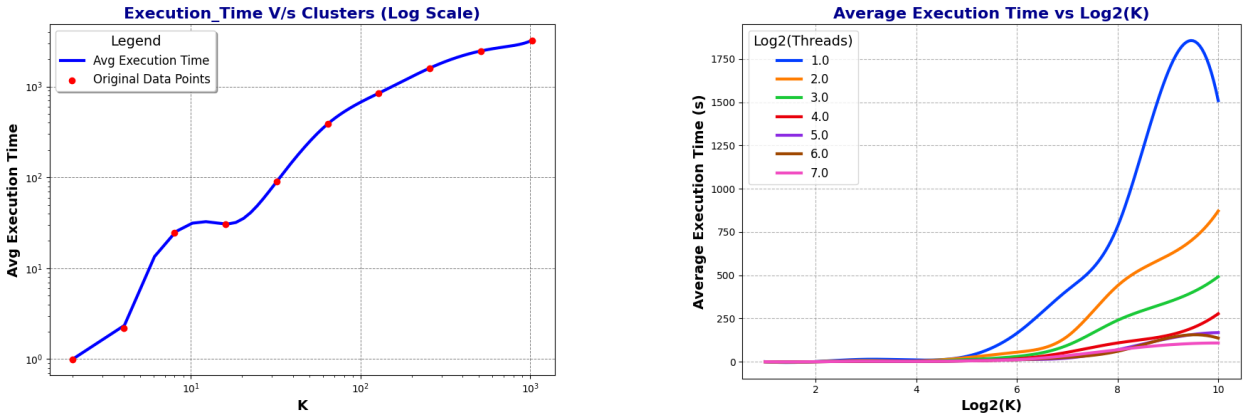
(a) Sequentia



(b) Parallel Using OpenMP

5 Results

The graph presented compares the execution times of sequential and parallel implementations



(a) Sequentia

(b) Parallel using Different Threads

Figure 3: Sequential vs. Parallel Execution time

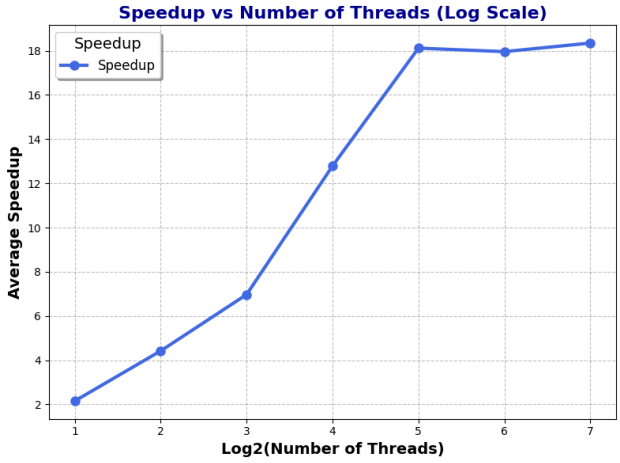


Figure 4: Sequential to Parallel.

using OpenMP for varying numbers of threads. As the number of threads increases in the parallel implementation, a noticeable reduction in execution time can be observed compared to the sequential approach. This demonstrates the effectiveness of parallelization in utilizing multiple threads to distribute the workload and reduce overall computation time. However, as the number of threads continues to grow, the improvement in execution time diminishes, indicating the onset of diminishing returns due to factors such as overhead and resource contention. The graph clearly highlights the trade-offs in parallel computing, where optimal performance is achieved at a certain number of threads before the benefits plateau.

6 Conclusion

The results from Table 1 demonstrate the relationship between thread count and speedup in parallel

Threads	Average Speedup
2	2.15x
4	4.42x
8	6.97x
16	12.79x
32	18.12x
64	17.96x
128	18.35x

Table 1: Speedup for Different Thread Counts

execution. Initially, as the number of threads increases from 2 to 128, the speedup improves significantly, with the highest speedup observed at 128 threads (18.35x). This trend highlights the effective parallelization of the task, where more threads lead to greater utilization of available processing units. However, the speedup starts to level off as the thread count exceeds 32, with only a slight increase between 32 and 128 threads. This indicates diminishing returns due to factors such as overhead from thread synchronization, memory access contention, and hardware limitations. Ultimately, the results suggest that optimal performance is achieved at around 32 threads, beyond which adding more threads yields minimal additional benefit. This emphasizes the importance of balancing thread count with the problem's inherent parallelism and system architecture.

7 References

[1]F. Yuan, Z. H. Meng, and H. X. Zhang, "A new algorithm to get the initial centroid," *International Conference on Machine Cybernetics*, pp. 26–29, 2004.

[2]Fahim A. M. and Salem A. M. T., "An Efficient enhanced K-means clustering algorithm," *Journal of Zhejiang University*, 2006.

[3]E. Rasmussen and P. Willett, "Agglomerative clustering using processor," *Journal of Documentation*, vol. 45, no. 3, pp. 1313-1325, March 1989.

[4]C. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, pp. 431-444, 1995.

[5]W. Zhao, H. Ma, and Q. He, "Parallel K-Means clustering algorithm based on MapReduce in Cloud Computing," *Cloud Computing*, pp. 674-679, 2009.

[6]S. Goil and Harasha Nagesh, "Efficient and Scalable Subspace Clustering for Very Large Data Sets," *Journal of Parallel and Distributed Computing*, vol. 12, no. 4, 1999.

[7]D. S. Bhupal Naik, S. Deva Kumar, and S. V. Ramakrishna, "Parallel Processing of Enhanced K-Means Using OpenMP," *Department of Computer Science and Engineering, Vignan University*, Andhra Pradesh, India.