Parallel K-Means Clustering Algorithm

Vera Krajchevska 196020

June 2024

1 Introduction

Clustering is an essential method in unsupervised learning, which partitions a dataset into unique clusters. In each cluster, the data points share a closer resemblance to one another than to points outside their group. It is valuable for discovering patterns and structures in data and among the many clustering techniques out there, K-means stands out as my favorite. It has applications in areas like image segmentation, pattern and color recognition, document clustering, customer segmentation, cyber profiling, recommendation engines and many other.

K-means is one of the most commonly used algorithms for clustering and is well-known for being efficient, straightforward and easy to implement. The algorithm starts by placing random centroids throughout the dataset. Each data point is subsequently allocated to the nearest centroid, determined by the squared Euclidean distance calculation. This assignment process iterates repeatedly until the centroids stabilize within their respective clusters, marking the convergence of the algorithm. Its computational complexity is O(n*K*t), where n is the number of points, K is the number of clusters, and t is the number of iterations.

Despite how easy it is to understand, the serial implementation of the K-Means algorithm has several drawbacks, especially when dealing with large datasets or requiring high performance. In a modern multi-core system the serial implementation is limited on a single CPU core, which restricts its scalability and makes processing large datasets inefficient and time-consuming, leading to underutilization of available computational resources and imbalances in resource usage. The computational time in a serial K-Means algorithm increases linearly with the number of data points and the number of clusters because each iteration requires computing the distance between each data point and all cluster centroids, which can be computationally intensive. That's why this version is unsuitable for real-time processing, high-dimensional data and big data applications. Also memory access is confined to a single core, which can become a bottleneck when handling large datasets, leading to slower memory access times and increased latency.

The aforementioned disadvantages are the reason I will be looking into parallel computing as one of the solutions for this problem. K-means is highly paral-

lelizable, allowing the task of assigning points to clusters to be easily distributed across multiple CPUs, with each distance calculation occurring independently. The parallel implementation of the K-Means algorithm is often preferred, as it can leverage multiple cores to improve performance and efficiency. By splitting the task across multiple CPUs, it drastically cuts down the computation time, has improved scalability, enhanced resource utilization and increased throughput. These benefits make parallel K-Means an attractive choice for large-scale data processing, high-dimensional data analysis, and real-time applications, ultimately leading to more efficient and effective data clustering solutions.

In this report, first of all I am starting with literature review of the applications of serial and parallel K-means algorithm. Then, I explain every step of the the original version in detail and introduce a metric for measuring the quality of the clusters which is achieved through minimizing the overall variance within each cluster. The solution architecture explains how we can achieve the task parallelization of the K-Means algorithm by incorporating the master-worker model of parallel computing architecture. In the last section, Results, I carried out experiments to assess how the parallel approach compares with the original K-means algorithm, specifically examining execution times and efficiency across different CPU setups.

2 Related work

The K-means clustering algorithm is widely recognized for its role in clustering analysis across scientific research and commercial applications. One major application is image segmentation [1], where clustering plays a pivotal role in dividing images into meaningful regions. Researchers have proposed parallel K-means variants, such as those leveraging the Message Passing Interface (MPI), to enhance efficiency in handling large images. Similarly, other studies have improved color detection by combining K-means with additional methods like mean shift [2], achieving high recognition accuracy and proving valuable for applications such as color-based image search. Another key application area is document clustering [3], where comparisons between K-means and other clustering techniques streamline document summarization, helping users quickly identify essential information.

The versatility of K-means clustering is further evident in fields like urban planning and logistics, where variations of the algorithm are used to address complex spatial and routing challenges. In urban settings, K-means has been adapted to detect congestion points by enhancing cluster initialization [4], overcoming the limitations of traditional methods in city planning. In logistics, integrating K-means with genetic algorithms has optimized truck-drone delivery networks [5], minimizing delivery time and energy consumption compared to standalone truck or drone deliveries.

Business applications of K-means, such as customer segmentation and behavior profiling, demonstrate the algorithm's utility in uncovering user patterns. For instance, K-means has been applied to segment telecom customers based

on spending behaviors [6], aiding targeted marketing and customer satisfaction improvement. It has also been used in cyber profiling to cluster internet user behaviors [7], revealing behavioral patterns like frequent visits to search engines and social media. These applications highlight K-means' adaptability in extracting actionable insights from diverse datasets.

Given the computational demands of K-means clustering on large datasets, many researchers have integrated parallel strategies to enhance performance. For example, dynamic load-balanced models leveraging the Master-Worker paradigm for data distribution have been developed [8], and MPI-based implementations have achieved higher efficiency in large-scale clustering tasks [9]. Other studies have focused on refining cluster initialization, reducing iterations, and computational loads [10]. Comparative analyses of parallel approaches, such as batch-based parallelization, have identified effective strategies for clustering large datasets, including text data for news aggregation [11].

Recently, GPUs have emerged as a powerful tool for accelerating K-means and similar algorithms in resource-intensive applications. GPU-based parallelization effectively addresses space limitations and data transfer challenges, achieving significant speedups in clustering execution times [12]. In database applications, GPU acceleration has been used to optimize aggregation functions in NoSQL databases, improving performance for large-scale data processing [13]. In similar fashion, GPUs have demonstrated exceptional efficiency in tasks like near-duplicate document detection, where parallelizing algorithms such as MinHash on GPUs achieve far greater efficiency than CPU-based alternatives [14]. These examples showcase the value of GPU parallelization in scenarios involving large datasets and demanding computational tasks.

Parallel K-means clustering has also been applied to specialized fields like agricultural and environmental data analysis. In particular, researchers have applied parallel K-means to agricultural datasets [15], addressing the limited research in this domain. Along the same lines, massively parallel approaches leveraging tens of thousands of processors on supercomputers have been employed to analyze spatial and temporal data for ecological research [16], demonstrating the feasibility of handling complex, large-scale environmental datasets. These studies underscore the versatility and efficiency of parallel K-means clustering in tackling a wide range of challenges, from urban planning and business analytics to environmental research. Its ability to handle large, complex datasets makes it an invaluable tool across diverse fields.

3 Serial K-Means Algorithm

As the primary objective of this paper is to compare the performance of two implementations of the K-means algorithm—serial and parallel—this section will provide a detailed explanation of how the original algorithm operates, before I continue with the proposed parallel solution.

The classical K-means clustering algorithm is a straightforward yet effective method for partitioning a dataset into a predefined number of disjoint clusters,

denoted as K. It proceeds through iterative steps aimed at progressively refining cluster assignments until convergence or a maximum number of iterations is reached.

The algorithm is shown on Figure 1 and can be explained by the following steps:

1. Initialization: The algorithm begins by randomly selecting K initial centroids from the dataset.

2. Assignment: Each data point in the dataset is assigned to the nearest centroid based on the squared Euclidean distance. This forms initial clusters where each cluster consists of data points closest to its respective centroid.

3. Centroid Recalculation: After the assignment of data points to clusters, the centroids are updated by calculating the mean of all data points assigned to each cluster. This step aims to reposition the centroids to better represent the center of their respective clusters.

4. Iteration: The assignment and update stages are repeated iteratively until one of the convergence criteria is met. Convergence can be defined as either no change in cluster assignments between iterations or reaching a specified maximum number of iterations.

Once the algorithm converges, it produces the definitive clustering outcome. Each data point is linked to the cluster with the nearest centroid. These clusters uncover the patterns and groupings within the dataset, effectively capturing underlying structure and relationships among the data points.

K-means clustering aims to group data points into clusters by minimizing the overall variance within each cluster, which is also referred to as the total sum of squares between clusters. This method computes the total variance within clusters by summing up the squared distances between every data point and its centroid. This approach effectively partitions the data into clusters where each point is closer to its cluster's centroid than to others, ensuring clusters are distinct and well-separated based on their variance.

The within-cluster variation $W(C_k)$ for a particular cluster C_k is calculated using this formula:

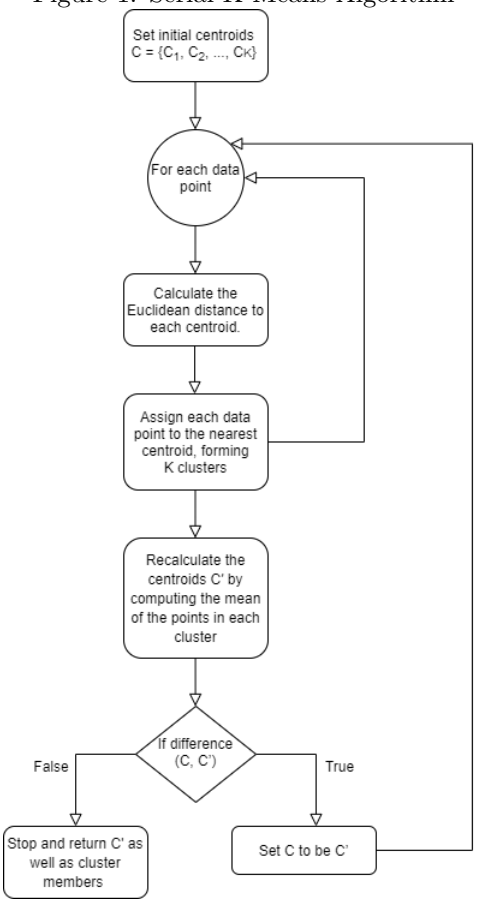
$$W(C_k) = \sum_{x_i \in C_k} \left\| x_i - \mu_k \right\|^2$$

where x_i represents a data point that is a member of the cluster C_k and μ_k represents the centroid of the cluster Ck.

The total within-cluster variation, also known as the total within-cluster sum of squares (WCSS), is the sum of the within-cluster variations for all K clusters:

$$WCSS = \sum_{k=1}^{K} W(C_k)$$

Figure 1: Serial K-Means Algorithm



The total within-cluster sum of squares (WCSS) serves as a metric to assess how well-defined and compact the clusters in the solution are. In K-means clustering, the goal is to minimize this value because a lower WCSS indicates that the clusters are tightly packed around their centroids.

4 Solution architecture for Parallel K-Means Algorithm

K-means clustering is a fundamental technique in machine learning, prized for its simplicity and efficiency in organizing data into distinct clusters. However, as datasets grow larger, the algorithm encounters significant computational challenges. The main challenge stems from its iterative nature, where each data point's assignment and subsequent centroid recalculation demand substantial

computing resources. This scalability issue becomes a bottleneck, limiting the algorithm's responsiveness.

To address this, parallelization becomes essential. Through the utilization of parallel processing, we can allocate the computational tasks across numerous processors or nodes. This approach effectively mitigates scalability issues, enabling the algorithm to handle massive datasets more efficiently. Figure 2 illustrates the pseudo-code for parallelizing K-means clustering, taking input as data points and the number of clusters (K), and producing K-centroids along with cluster memberships.

In the parallelized version of K-means, computation is decentralized across multiple processes, each handling a subset of the data. A master process initializes centroids and distributes them to worker processes. Each worker process handles a subset of the data, independently calculating distances between data points and their assigned centroids in parallel. The Euclidean distance formula, used to measure the distance between points and centroids in an n-dimensional space, guides this computation. Its formula is as follows:

$$d(x, c) = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2}$$

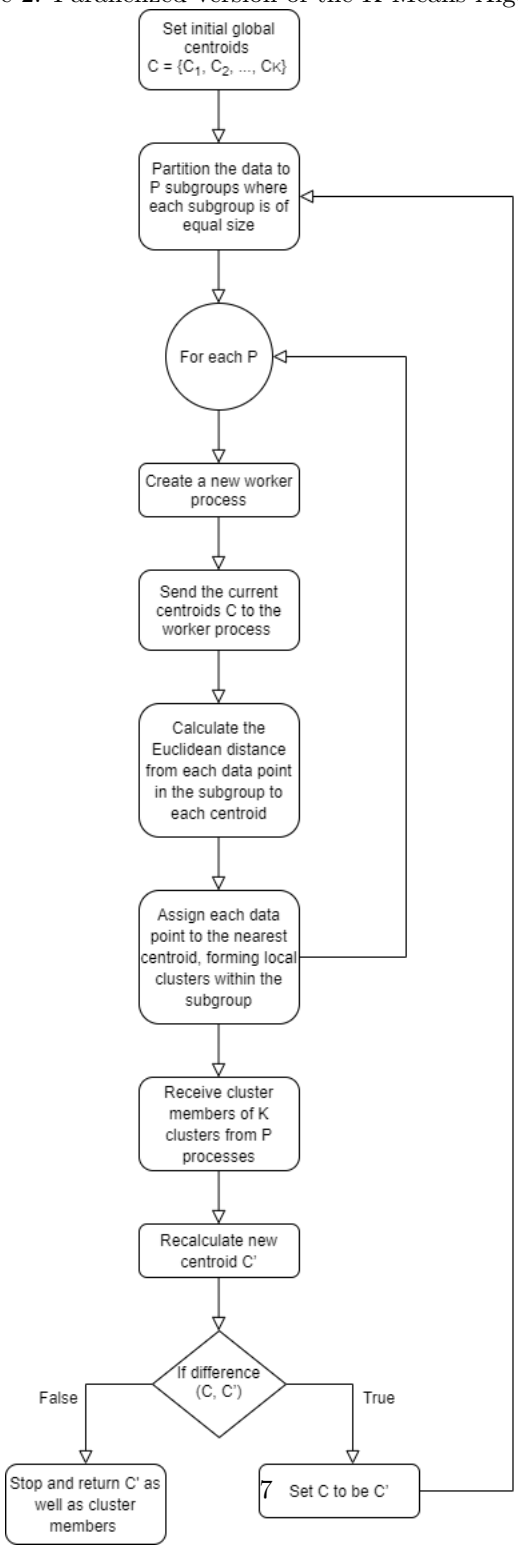
where x represents the coordinates of the data point, c represents the coordinates of the centroid and n is the number of dimensions (features) in the data points.

After distributing tasks in parallel, the master process combines results from all workers to update centroids. Each centroid's update involves averaging the assigned data points, ensuring ongoing improvement until convergence. Let's denote the set of data points assigned to centroid c_i as X_i. The centroid c_i is then updated as follows:

$$c_i = \frac{1}{|X_i|} \sum_{x \in X_i} x$$

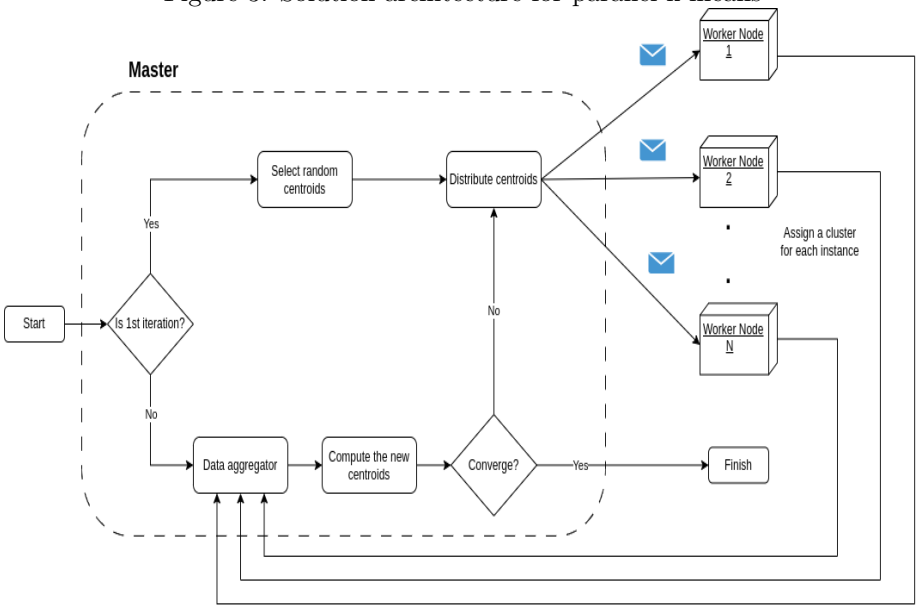
where $|X_i|$ is the number of data points in X_i and $\sum_{x \in X_i} x$ represents the sum of all the data points' coordinates in X_i.

Figure 2: Parallelized version of the K-Means Algorithm



This iterative process of parallel assignment and global update optimizes efficiency while managing communication between master and worker processes. In Figure 3, the solution architecture adopts the master-worker paradigm for parallel processing in K-means clustering. This approach enhances algorithm performance, making K-means well-suited for modern data-intensive applications handling large datasets.

Figure 3: Solution architecture for parallel k-means



5 Results

I intend to assess the effectiveness of both the serial and parallel K-means algorithms through a series of simulations conducted on artificially generated datasets. The experiments are carried out on a computer that has the specifications showcased in Table 1.

Hardware Component	**Details**
CPU Model Name	AMD Ryzen 7 7840HS w/ Radeon 780M Graphics
CPU Architecture	x86_64
CPU Core(s) per Socket	8
Thread(s) per Core	2
CPU(s)	16
RAM	16GB

Table 1: Hardware Configuration of the Computer Used

Data Points	Serial K-Means	Parallel K-Means		
		2 CPUs	4 CPUs	6 CPUs
1000	0.064	0.356	0.252	0.299
10000	0.766	0.813	0.484	0.476
100000	7.694	4.280	2.094	1.757
150000	15.114	6.263	3.972	2.641
200000	19.190	7.949	4.596	4.005
250000	20.820	11.460	6.037	4.274
300000	23.864	13.348	6.446	6.222
350000	25.450	14.121	8.095	5.575
400000	37.146	16.715	8.553	6.131
500000	43.950	18.372	11.465	9.294

Table 2: Runtime in seconds of serial and parallel K-means clustering algorithms

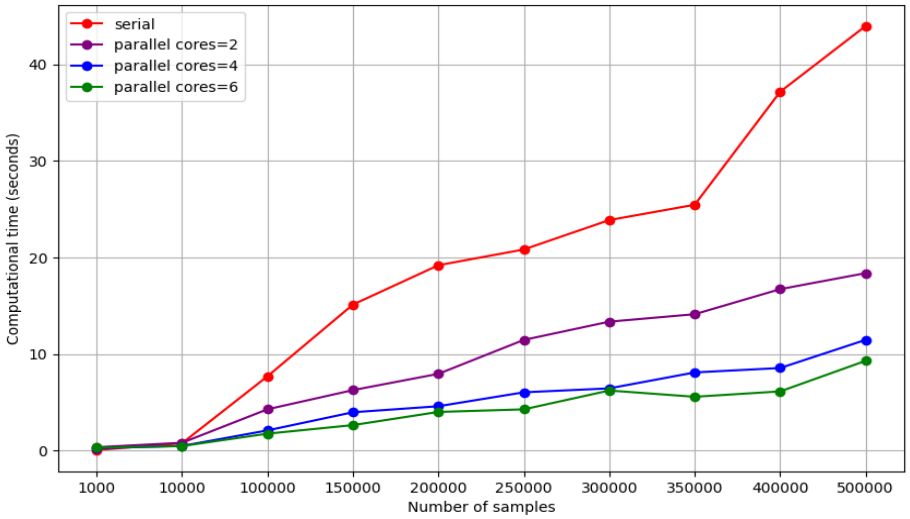
For evaluating the effectiveness of parallelization for the K-Means algorithm on large datasets I use several performance and efficiency metrics such as Execution Time, Speedup and Load Balancing.

5.1 Execution Time

Determining the optimal number of clusters (K) for the best clustering of the dataset was not explored, as the experiments were conducted on synthetic data. Therefore the performance of both the serial and parallel versions of the algorithm will be showcased for only three clusters (K).

Throughout these simulations, I maintain a consistent dataset with two dimensions and I vary the number of data points. In the case of my parallel K-means clustering algorithm, I additionally experiment with different numbers of processors, precisely two, four and six cores. To determine the runtime for each simulation, I perform the experiment seventy times and then calculate the average runtime by dividing the total time by seventy. The reason for this number is that by using a smaller one I may reduce the overall runtime and increase the variability. Therefore, a greater number of executions can provide more stable results. My opinion is that anywhere between fifty and one hundred executions will be satisfactory for these datasets.The results are shown in Table 2 and then plotted on Figure 4 below.

Figure 4: Simulation execution times of parallel and serial K-means clustering algorithm

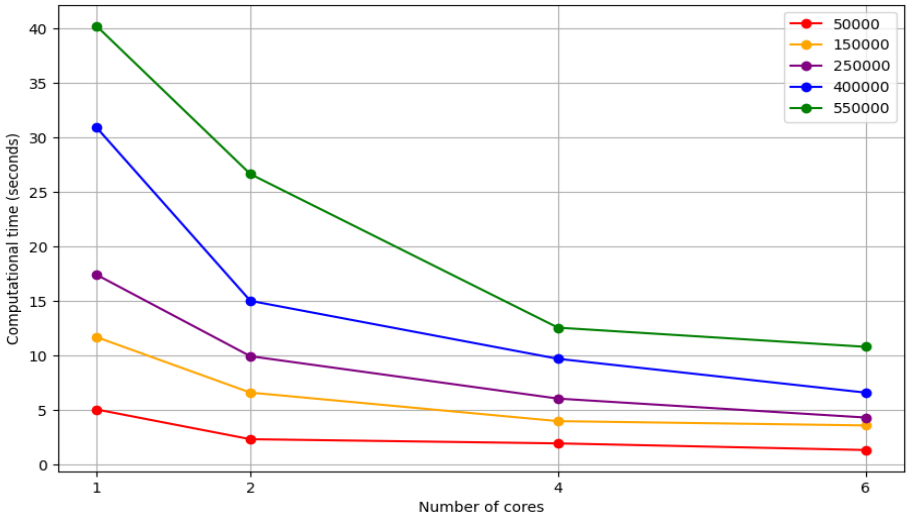


The findings illustrated in Figure 4 indicate that, with datasets containing fewer than 1,000 observations, the serial K-means clustering algorithm exhibits superior performance compared to its parallel counterpart. This disparity is likely attributed to the communication overhead between processors. Nevertheless, as the number of observations surpasses 10,000, the parallel K-means clustering algorithm demonstrates enhanced efficiency. With each subsequent augmentation in dataset size, the disparity in runtimes between the serial and parallel K-means clustering algorithms becomes increasingly notable. The serial K-means which has nearly double the execution times of parallel K-means with two cores, has significant changes for datasets with more than 350,000 observations, so the parallel K-means becomes a necessity for larger datasets to be processed faster. Focusing on the parallel K-means with two cores, it's visible that it has better performance than the serial counterpart, but it's expectedly slower than the parallel K-means with four or six cores. However, on larger datasets the difference between two cores and four or six is expected to be greater, since the two cores will be overloaded in comparison with the load each of the cores will have, if they are four or six, therefore exhibiting better performance. My results propose that for datasets comprising of fewer than 300,000 observations, the runtimes exhibit considerable similarity between four and six cores. The difference between them after that number of observations is still small, but the parallel version with six cores performs slightly better. On my architecture for clustering problems consisting of less than 500,000 observations, considering the time it takes for the system to process the datasets, it's more efficient to stop at four cores as the difference between four and six cores in insignificant and still performs well.

5.2 Speedup

Measuring the speedup of parallel algorithms is crucial for understanding their efficiency and scalability. Speedup is defined as the ratio of the execution time of the serial algorithm to the execution time of the parallel algorithm. It provides a quantitative measure of how effectively parallel K-Means algorithm leverages multiple processors to perform a task faster than its serial counterpart. By analyzing the speedup, I aim to demonstrate the performance benefits and potential limitations of parallelizing the K-Means algorithm.

Figure 5: Simulation execution times of different datasets

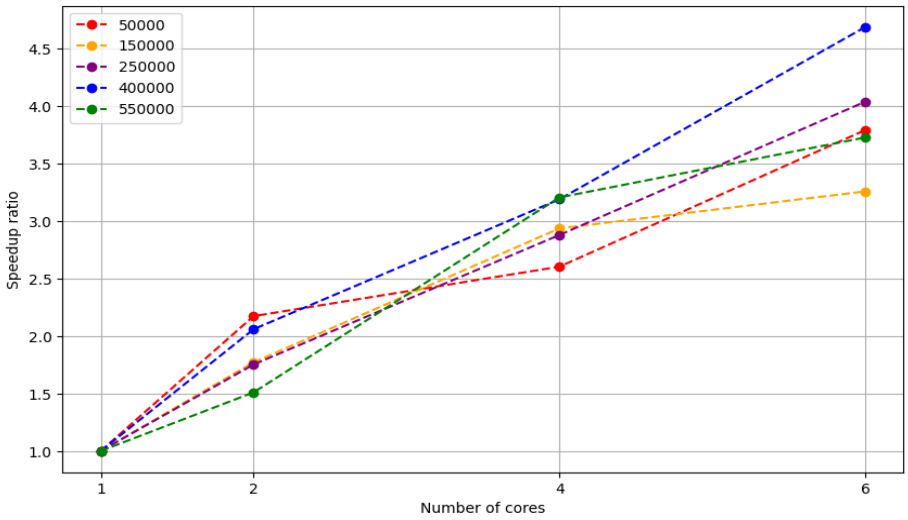


On Figure 5, it is clear that across all dataset sizes I have chosen, the computational time decreases as the number of cores increases. This indicates that parallelization is effective in reducing runtime. The most significant reduction in computational time is observed when moving from one core to two cores, especially for larger datasets like 400,000 and 500,000. This trend continues and is visibly noticeable when moving from two to four cores for all datasets except for the smallest one of 50,000 data points. The improvement slows down when moving from four to six cores. This indicates diminishing returns, which is a common characteristic in parallel computing due to factors such as communication overhead and synchronization costs.

As the size of the dataset increases, the benefits of parallelization become more pronounced. For example, the dataset with 550,000 observations shows a significant drop in computational time from around forty seconds on one core to approximately ten seconds on six cores. For this dataset, the computational time decreases rapidly up to 4 cores but shows less significant improvement beyond that, indicating that further increasing the number of cores yields marginal benefits. Smaller datasets, like the one with 50,000 observations, show less dra-

matic speedup, likely because the overhead of managing parallel tasks becomes more significant relative to the total computation time.

Figure 6: Speedup Ratio Graph for K-Means Clustering



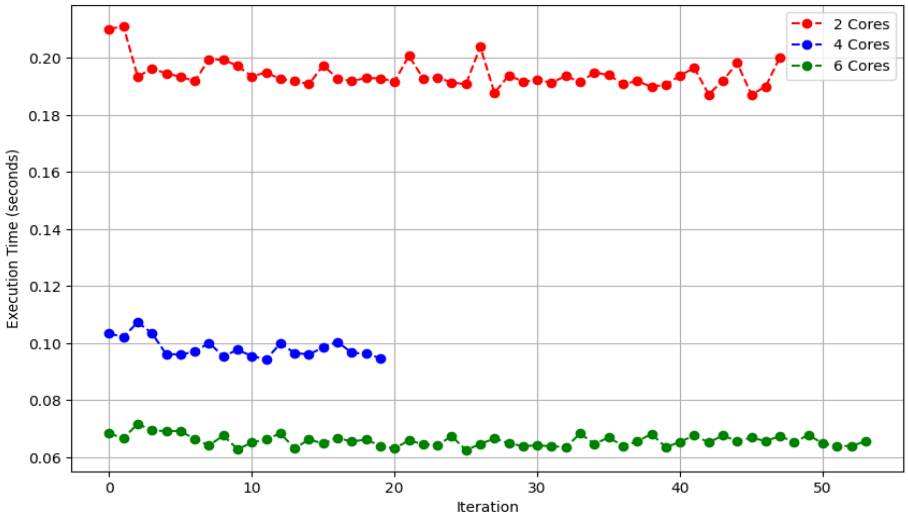
As mentioned earlier, the speedup ratio is calculated as the ratio of the runtime of the serial implementation to the runtime of the parallel implementation with multiple cores and is illustrated on Figure 6. For all datasets, the speedup ratio increases as the number of cores increases. Larger datasets, such as those with 400,000 and 550,000 observations, show higher speedup ratios compared to smaller datasets. This is likely because the overhead of parallelization is more easily amortized over a larger number of data points. For the larger datasets, the speedup ratio increases almost linearly with the number of cores. In some cases, there are instances of superlinear speedup, where the speedup ratio exceeds the number of cores used. The trends indicate that as the dataset size increases, the efficiency of parallelization also improves, resulting in higher speedup ratios.

5.3 Load Balancing

Load balancing in parallel computing refers to the even distribution of work across all available processing units to ensure that no single core is significantly more or less loaded than others. Effective load balancing maximizes the utilization of all cores and minimizes idle times, leading to better overall performance. In the context of parallel algorithms, particularly with large datasets, achieving optimal load balancing is crucial for maintaining efficiency and scalability. To analyze load balancing, I employed synthetic dataset of 50,000 data points and monitored the execution times of each iteration of different numbers of cores.

Figure 7 showcases the execution times of each iteration of the algorithm ran on two, four and six cores. We can clearly see that the execution time for two

Figure 7: Execution times of each iteration of one execution on different cores



cores is consistently around 0.18 to 0.20 seconds. The time is fairly stable with slight variations, indicating a balanced load but with higher execution times compared to other configurations. Whereas, the execution time for four cores is lower than with two cores, generally around 0.10 to 0.11 seconds. There are some fluctuations, suggesting occasional imbalances, but overall, the load is better distributed than with two cores. Finally, the execution time is the lowest when running the algorithm on six cores, around 0.06 to 0.07 seconds. This configuration shows the most stability and least variation in execution time, indicating the best load balancing among the three configurations. The consistent execution time suggests that the workload is evenly distributed across all six cores, leading to efficient parallel processing.

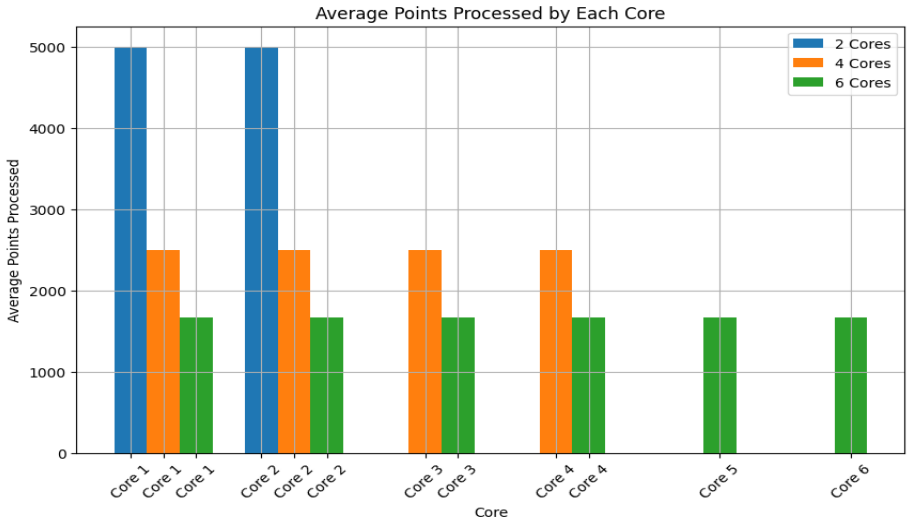
The occasional shorter iterations, particularly noticeable in the four-core configuration, can be attributed to a few factors, such as algorithm convergence and workload distribution. This means the k-means algorithm may converge faster on some iterations, especially if the initial centroids are closer to the final clusters, requiring fewer updates, or minor imbalances in workload distribution among the cores can cause slight variations in execution time. In parallel computing, perfect load balancing is challenging due to factors like data distribution, core synchronization, and computational overhead.

The significant drop in execution time from two cores to four cores, and further from four cores to six cores, highlights the effectiveness of parallelization. As the number of cores increases, the workload is divided more finely, reducing the execution time per iteration. However, the diminishing returns as cores increase indicate a typical parallel performance trend where overhead and communication costs start to counterbalance the benefits of adding more cores.

To further illustrate the effectiveness of load balancing and parallelization,

I plotted the average number of points processed by different numbers of cores, indicating the workload distribution across each core. The bar plot demonstrates how the workload is evenly distributed among the cores, reflecting good load balancing.

Figure 8: Load of each core in parallel K-means processing

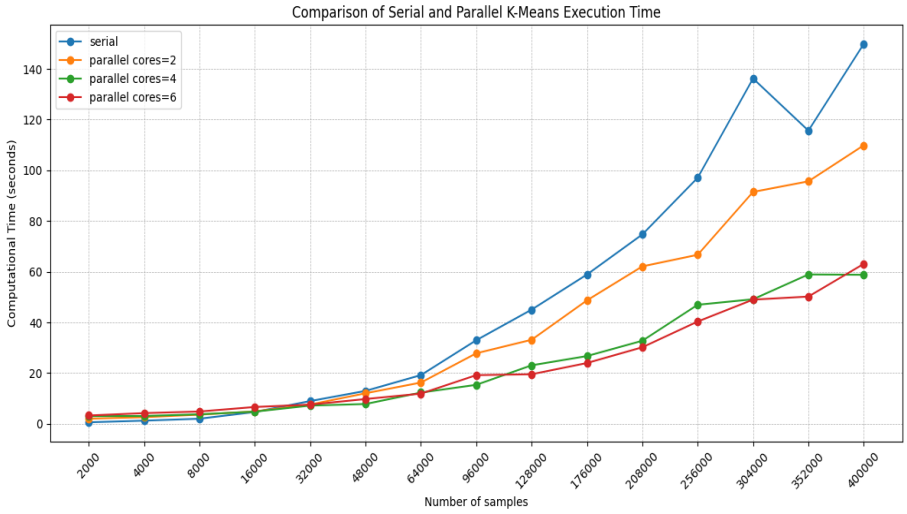


6 Performance of Parallel K-Means Algorithm using vector dataset

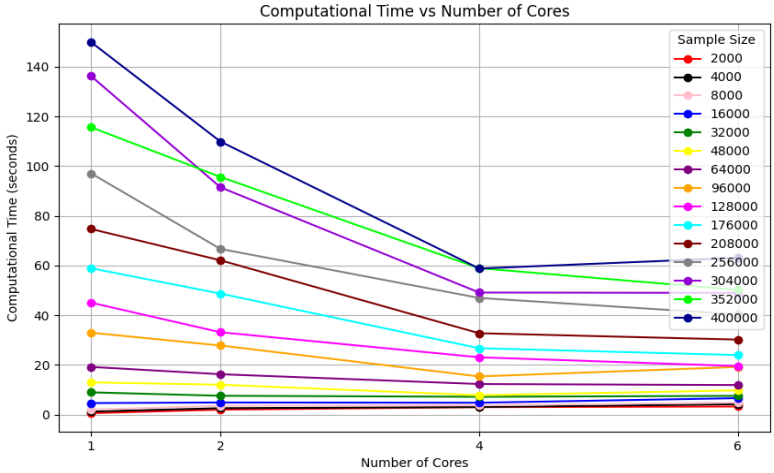
Explanation of the Glove dataset

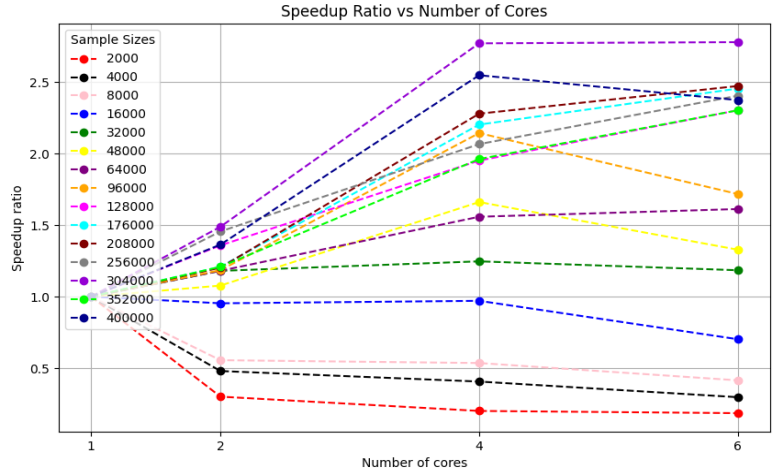
7 Results

7.1 Execution Time

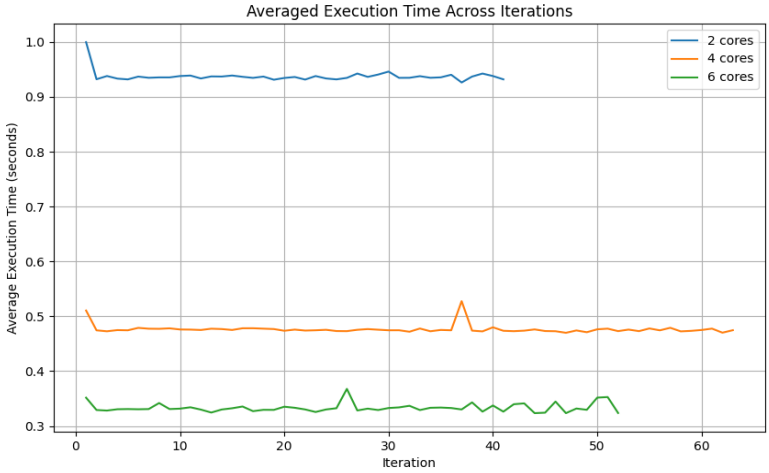


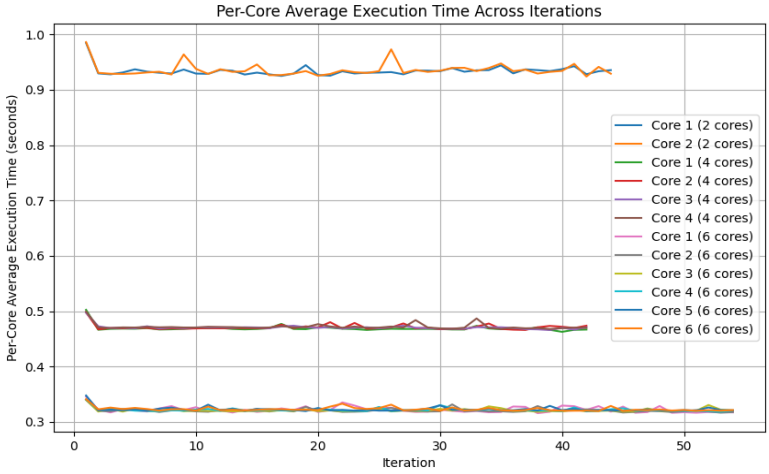
7.2 Speedup

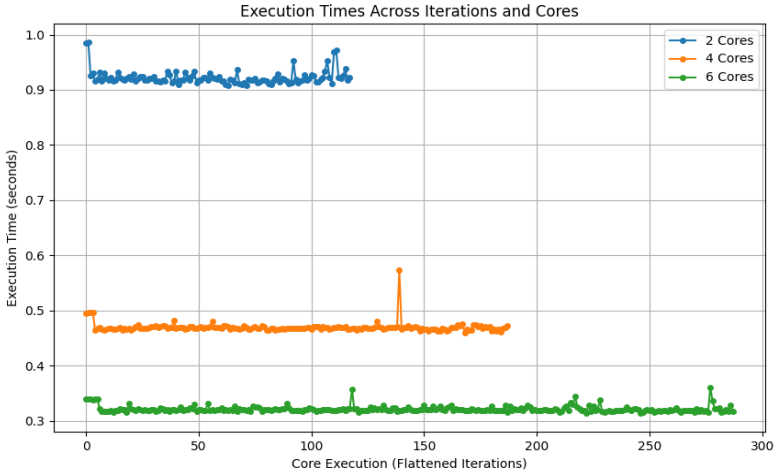

Computational Time vs Number of Cores
Computational Time (seconds)
Number of Cores
Sample Size
2000
4000
8000
16000
32000
48000
64000
96000
128000
176000
208000
256000
304000
352000
400000


Speedup Ratio vs Number of Cores
Speedup ratio
Number of cores
Sample Sizes
2000
4000
8000
16000
32000
48000
64000
96000
128000
176000
208000
256000
304000
352000
400000

7.3 Load Balancing






Execution Times Across Iterations and Cores
2 Cores
4 Cores
6 Cores
Execution Time (seconds)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
50
100
150
200
250
300
Core Execution (Flattened Iterations)

8 Conclusion

In conclusion, K-means clustering is a vital machine learning technique valued for its simplicity and efficiency in organizing data into distinct clusters. However, its iterative nature and computational demands pose significant challenges as datasets grow larger, creating scalability issues. The implementation of parallel processing addresses these challenges by distributing computational tasks across multiple processors, significantly enhancing performance and efficiency. My study demonstrates that the parallelized version of K-means achieves substantial reductions in computational time, especially for large datasets, making it a robust solution for big data applications. By employing a master-worker paradigm, this approach effectively balances the workload, optimizes resource utilization, and accelerates convergence. The empirical results confirm that increasing the number of cores leads to improved performance, though with diminishing returns, emphasizing the need for balanced resource allocation. Overall, parallelizing K-means clustering transforms it into a highly scalable and efficient algorithm suitable for modern, data-intensive environments. This enhanced capability enables more responsive and effective data clustering solutions, addressing the demands of contemporary machine learning applications.

References

[1] Ahmet Esad Top, F Şükrü Torun, and Hilal Kaya. Parallel and distributed image segmentation based on colors using k-means clustering algorithm. In *Proceedings of the ICES 2019: 5th International Conference on Engineering Sciences*, 2019.

[2] Xingming Zheng and Ningzhong Liu. Color recognition of clothes based on k-means and mean shift. In *2012 IEEE international conference on intelligent control, automatic detection and high-end equipment*, pages 49–53. IEEE, 2012.

[3] Rakesh Chandra Balabantaray, Chandrali Sarma, and Monica Jha. Document clustering using k-means and k-medoids. *arXiv preprint arXiv:1502.07938*, 2015.

[4] Xiaojuan Ran, Xiangbing Zhou, Mu Lei, Worawit Tepsan, and Wu Deng. A novel k-means clustering algorithm with a noise algorithm for capturing urban hotspots. *Applied Sciences*, 11(23):11202, 2021.

[5] Sergio Mourelo Ferrandez, Timothy Harbison, Troy Webwer, Robert Sturges, and Robert Rich. Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management*, 9(2):374–388, 2016.

[6] Băcilă Mihai-Florin, Rădulescu Adrian, and Mărar Ioan Liviu. Prepaid telecom customers segmentation using the k-mean algorithm. *THE ANNALS OF THE UNIVERSITY OF ORADEA*, page 1112, 2012.

[7] Muhammad Zulfadhilah, Yudi Prayudi, and Imam Riadi. Cyber profiling using log analysis and k-means clustering. *International Journal of Advanced Computer Science and Applications*, 7(7), 2016.

[8] Yufang Zhang, Zhongyang Xiong, Jiali Mao, and Ling Ou. The study of parallel k-means algorithm. In *2006 6th World Congress on Intelligent Control and Automation*, volume 2, pages 5868–5871. IEEE, 2006.

[9] Dr Urmila R Pol. Enhancing k-means clustering algorithm and proposed parallel k-means clustering for large data sets. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(5), 2014.

[10] Jinlan Tian, Lin Zhu, Suqin Zhang, and Lu Liu. Improvement and parallelism of k-means clustering algorithm. *Tsinghua Science and Technology*, 10(3):277–281, 2005.

[11] D Najkov, Vladimir Zdraveski, and Marjan Gusev. Real-time clustering of text data for news aggregation. In *2023 31st Telecommunications Forum (TELFOR)*, pages 1–4. IEEE, 2023.

[12] Salvatore Cuomo, Vincenzo De Angelis, Gennaro Farina, Livia Marcellino, and Gerardo Toraldo. A gpu-accelerated parallel k-means algorithm. *Computers & Electrical Engineering*, 75:262–274, 2019.

[13] Stefan Jovanov, Vladimir Zdraveski, and Marjan Gusev. Gpu in applications with non-relational dbs. In *2020 28th Telecommunications Forum (TELFOR)*, pages 1–4. IEEE, 2020.

[14] Dimitar Peshevski, Vladimir Zdraveski, and Sashko Ristov. Parallel near-duplicate document detection using general-purpose gpu. *Computing and Informatics*, 43(3):583–610, 2024.

[15] V Ramesh, K Ramar, and S Babu. Parallel k-means algorithm on agricultural databases. *International Journal of Computer Science Issues (IJCSI)*, 10(1):710, 2013.

[16] Jitendra Kumar, Richard T Mills, Forrest M Hoffman, and William W Hargrove. Parallel k-means clustering for quantitative ecoregion delineation using large data sets. *Procedia Computer Science*, 4:1602–1611, 2011.