A Comprehensive and Comparative Analysis of Parallel Frameworks and their Applications to Big Data Clustering

Nawfal
Dept. of Mechanical Engineering,
Indian Institute of Technology,
Madras

Srivastava
Dept. of Mechanical Engineering,
Indian Institute of Technology,
Madras

***Abstract*— The K-Means clustering using Lloyd's algorithm is an iterative approach to partition the given dataset into K different clusters. The serial algorithm involves iterative steps where we compute the distance of each datapoint from the centroids and assign the datapoint to the nearest centroid. This approach is essentially known as the expectation-maximization step.**

Clustering involves extensive computations to calculate distances at each iteration, which increases as the number of data points increases. This provides scope for parallelism. However, we must ensure that in a parallel process, each thread has access to the updated centroid value, and no racing condition exists on any centroid values.

In this paper, we will compare two different approaches. The first approach is an OpenMP flat synchronous method where all processes are run in parallel, and we use synchronization to ensure safe updates of clusters. The second approach we adopt is a GPU- based parallelization approach using OpenACC, wherein we will try to make use of GPU architecture to parallelize chunks of the algorithm to observe decreased computation time. We will analyze metrics such as speed up, efficiency, time taken with varying data points, and number of processes to compare the two approaches and understand the relative performance improvement we can get.

I. Introduction

Tourism Clustering is an unsupervised machine-learning technique used for grouping data in such a way that similar samples are clustered into the same group due to some underlying pattern without any explicit markers or labeling. Clustering finds applications in various fields, such as pattern recognition, image segmentation, anomaly detection, and data compression.

Various clustering algorithms exist, with each having its metric in defining similarity, partitioning of data, and quality of clustering. Some of the popular clustering algorithms that are widely used include K-Means, Hierarchical clustering, Density-Based Spatial Clustering of Applications with Noise (DBSCAN), and Gaussian Mixture Models.

The K-Means clustering algorithm is one such algorithm that makes use of an iterative method in clustering N data points into K clusters by computing the cluster mean for every cluster and minimizing the L2 distance of each point from the cluster centers/means during each iteration. The time complexity for the algorithm is O(NTKd), where N is the total number of data points, T is the number of iterations required for the algorithm to converge, K is the number of clusters, and d is the dimension of each data point. Since the values of K and d are generally way smaller than N, they can be ignored. It is also observed that $T \propto N$, hence the effective time complexity reduces to O(N 2). Due to this high time complexity, the computation of the clusters would be time-consuming for large datasets.

However, due to the simple nature of the steps and cal- culations in the algorithm, it provides us with the scope of running the algorithm in a parallel or distributed environment. This paper aims to explore and compare the scope and extent of parallelization of the algorithm, mainly using two parallelization models, namely a Shared Memory Model using OpenMP and a GPU Programming Model using OpenACC.

II. The Lloyd's Algorithm

The Lloyd's algorithm is an iterative method that forms the basis of K-Means clustering. Given a dataset $\boldsymbol{X} \in \mathrm{R}^{d \times N}$, where $\boldsymbol{x}_i$ is a d-dimensional vector, the Lloyd's algorithm can be broken down into three steps as follows.

1) Initialisation: Assuming the dataset is to be clustered into K clusters, we first initialize the cluster centers by randomly selecting K points from the dataset. These ini- tial cluster centers

are denoted by $\boldsymbol{\mu}_1^0, \boldsymbol{\mu}_2^0, \boldsymbol{\mu}_3^0, \ldots, \boldsymbol{\mu}_K^0$, where the superscript indicates the iteration.

2) Reassignment: For the t^{th} iteration, the distance of every $\boldsymbol{x}_i$ from $\boldsymbol{\mu}^t$, is computed for all values of k. If z^t denotes $_k$ $_i$ the cluster indicator of $\boldsymbol{x}_i$ for the t^{th} iteration, the data point $\boldsymbol{x}_i$ is assigned to the cluster k for the $(t+1)^{th}$ iteration according to:

$$z_i^{t+1} = \arg\min_k ||\boldsymbol{x}_i - \boldsymbol{\mu}_k^t||_2^2$$

Where $||\cdot||_2$ is the L2 norm.

3) Mean Calculation: Once every $\boldsymbol{x}_i$ has been reassigned in the t^{th} iteration, the respective cluster centers/means $\boldsymbol{\mu}_k^{t+1}$ are to be calculated according to: Σ_N

$$\boldsymbol{\mu}_{kt+1} = \frac{\sum_{i=1}^{N} \mathbf{1}(z_{ti+1} = k)\boldsymbol{x}_i}{\sum_{i=1}^{N} \mathbf{1}(z_{it+1} = k)} \quad (2)$$

Where $\mathbf{1}(z)$ is the proposition function defined as,

$$\mathbf{1}(z) = \begin{cases} 1 \text{ if z is true,} \\ 0 \text{ otherwise.} \end{cases} \quad (3)$$

calculating the error $\mathbf{E}$, which is then transferred to a global error variable.

The results obtained after running the parallelized algorithm for various threads for both the two and three-dimensional

The algorithm iterates over steps 2 and 3 until the algorithm converges. Convergence of Lloyd's algorithm implies that the cluster indicators of every $\boldsymbol{x}_i$ do not change for further iterations. The Lloyd's algorithm produces a hard clustering for each data point and it always converges to a local minima. Thus, the algorithm is sensitive to the initialization step and will produce different clusterings based on the initialization.

III. The Serial Lloyd's Algorithm

For evaluating the serial version of the algorithm as well as the parallelised version in the upcoming sections, we shall make use of three datasets. The datasets are of sizes 100,000, 200,000, and 500,000, and all of them are generated in a similar manner using a mixture of Bi-variate Gaussian Distributions. Additionally, we will also be using three-dimensional datasets consisting of 100,000, 200,000, 400,000, 800,000, and 1,000,000 samples. By using datasets of varying sizes, it would allow us to evaluate the frameworks with respect to the scaling of the dataset.

The main metric concerning this review is the time com- plexity of the algorithm as well as the number of iterations required for convergence for various cases. The results after running the serial algorithm on the datasets for clusters of $k = 4, 8,$ and 11 are shown in table I below:

TABLE I
SIZE OF THE DATASET (N) VS TIME FOR CONVERGENCE FOR VARYING NUMBER OF CLUSTERS

Size (N)	K = 4	K = 8	K = 11
500000 (2D)	1.664616	5.313805	25.744963
1000000 (3D)	2.255409	34.27957	73.925911

IV. Parallel Lloyd's Algorithm

As seen in the previous section, Lloyd's algorithm takes an increasing amount of time when the size of the dataset increases, as well as when the number of clusters increases. However, the main advantage of Lloyd's algorithm is the underlying simplicity of its steps and calculations. A large portion of the steps in the algorithm can be parallelized, which shall be elucidated in the subsequent subsections of the paper.

A. OpenMP

With OpenMP, the aim is to work with a Shared Memory Model on the algorithm. We've implemented a data paral- lelization model with task data parallelization sections for this particular algorithm, the dataset is to be divided among the number of threads specified by the user. Each thread will independently perform the reassignment step as well as calculate the local cluster means. Once these local cluster means have been calculated, these are transferred to a global variable; the global variable is used by the master thread in datasets have been presented in tables II and III. The number of clusters to be produced is fixed to a value of 8 for the 2dimensional dataset and 4 for the 3-dimensional dataset:

TABLE II
TIME FOR CONVERGENCE (IN SECONDS) FOR VARIOUS THREAD COUNTS ON THE 2D DATASET

Size (N)	p = 2	p = 4	p = 8	p = 16
100000	0.680664,	0.381361,	0.273247	0.378534
200000	0.788368	0.414747	0.324386	0.310875
500000	10.988341	5.538359	4.244740	3.648641

TABLE III
TIME FOR CONVERGENCE (IN SECONDS) FOR VARIOUS THREAD COUNTS
ON THE 3D DATASET

Size (N)	p = 2	p = 4	p = 8	p = 16
100000	3.33448	1.672675	1.220420	1.230586
200000	7.327056	3.714937	2.728632	2.359286
400000	14.286552	8.2062708	6.013132	4.937502
800000	22.893556	13.358098	10.931103	9.245712
1000000	35.973150	19.965121	16.016981	13.495912

B. *OpenACC*

With OpenACC, the aim is to create a CPU-GPU work- sharing environment for the algorithm. Just like OpenMP, a task parallelisation model is made use of here. The dif- ference between the OpenMP and OpenACC model is that the **parallel** directive isn't called at the beginning before the algorithm begins. Rather, the directive is called at the blocks of code corresponding to the various steps of the algorithm. Due to this, there's a constant forking/de-forking of gangs and workers in each iteration, unlike the OpenMP version, which enables the use of additional directives such as **acc loop**, **parallel loop**, **atomic**, and **reduction** to further parallelize, optimize and speed up the code.

The results obtained after running the parallelized algorithm for both the two and three-dimensional datasets have been presented in tables IV and V. The number of clusters to be produced is fixed to a value of 8 for the 2-dimensional dataset and 4 for the 3-dimensional dataset:

TABLE IV
TIME FOR CONVERGENCE (IN SECONDS) FOR THE 2D DATASET

Size (N)	Time Taken (seconds)
100000	0.7213
200000	0.283524
500000	0.518219

V. RESULTS

For the purpose of this paper, we shall observe and discuss the results under the following metrics:

TABLE V
TIME FOR CONVERGENCE (IN SECONDS) FOR THE 3D DATASET

Size (N)	Time Taken (seconds)
100000	0.087148
200000	0.486771
400000	0.548548
800000	0.743832
1000000	0.802407

- Similarity of clustering
- Speedup as a function of number of threads
- Efficiency as a function of number of threads
- Time taken with increasing scaling of the problem

In the speedup and efficiency metrics, we will discuss the results purely for the OpenMP framework since we can control the number of threads spawned. In the OpenACC framework, we rely on the smart usage of gangs and vectors based on the size of the problem.

A. *Similarity of Clustering*

The results of clustering the 3-dimensional dataset into 4 clusters have been plotted below for both the serial and the parallel program by OpenACC. We can observe that the parallel program achieves clustering similar to the serial program for K = 4.

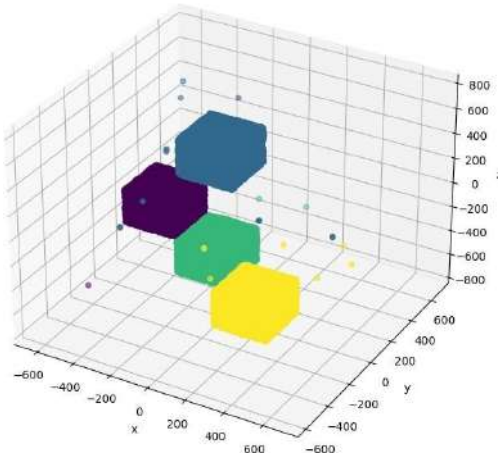


Fig. 1. Results of Serial K-Means on 400k datapoints

The results of clustering the 2-dimensional dataset into 11 clusters have been plotted below for both the serial and the parallel program by OpenACC. The parallel program achieves similar clustering as the serial program.

B. *Speedup*

The Speedup $\psi(n, p)$ as a function of number of threads (p) for both the 2D and 3D datasets has been plotted as well. We observed an increase in the speedup as the size of the dataset increases across the 2D and 3D datasets, except in

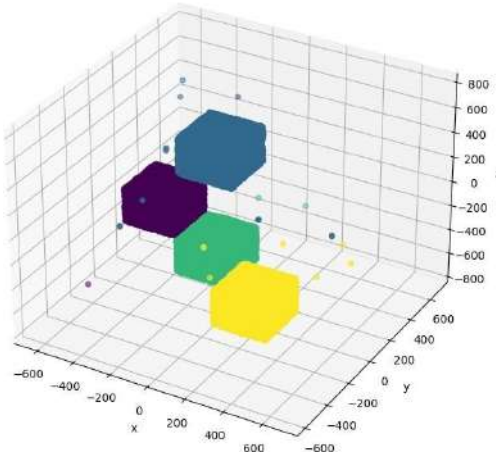


Fig. 2. Results of Parallel K-Means on 400k datapoints

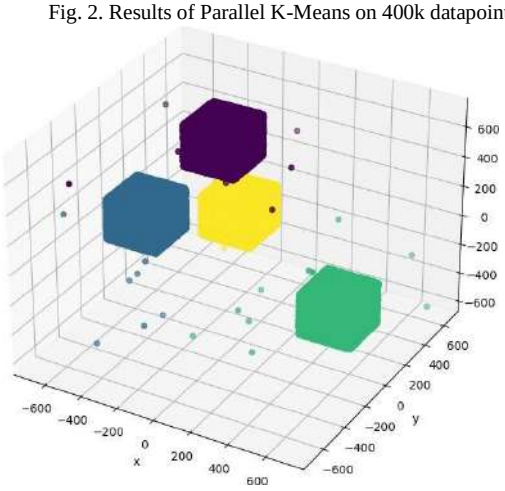


Fig. 3. Results of Serial K-Means on 1 Million datapoints

certain instances of the smallest datasets due to the small size of computation not benefitting from the time taken to spawn an increased number of threads. We also observed that the speedup values are larger for the larger datasets, indicating strongly that for large datasets parallelization can offer a significant boost.

C. C. Efficiency

We plot the Efficiency $\epsilon(n, p)$ as a function of the number of threads (p) for both the 2D and 3D datasets. We can observe that the highest efficiency occurs for the number of threads = 2, which drops as the number of threads increases.

D. D. Scaling

Here we plot the variation of time taken with the scaling/size of the dataset for both the 2D and 3D datasets. We can see

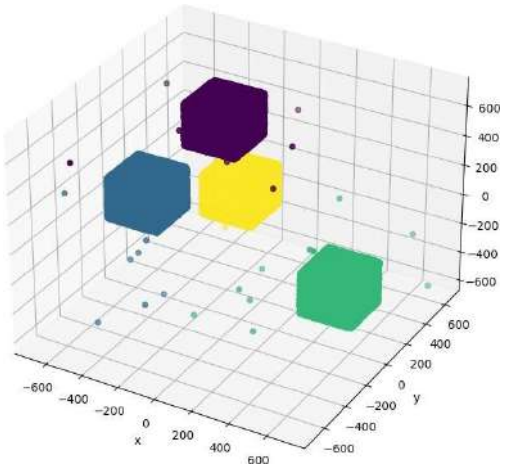


Fig. 4. Results of Parallel K-Means on 1 Million datapoints

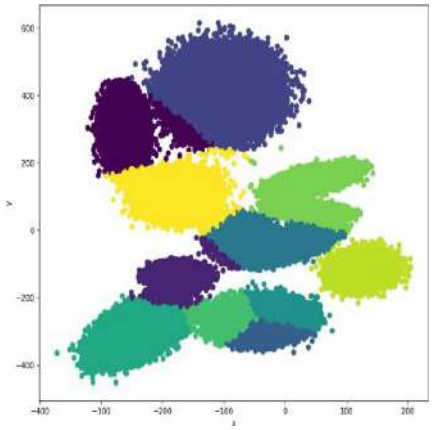


Fig. 5. Results of Serial K-Means on 2D Dataset

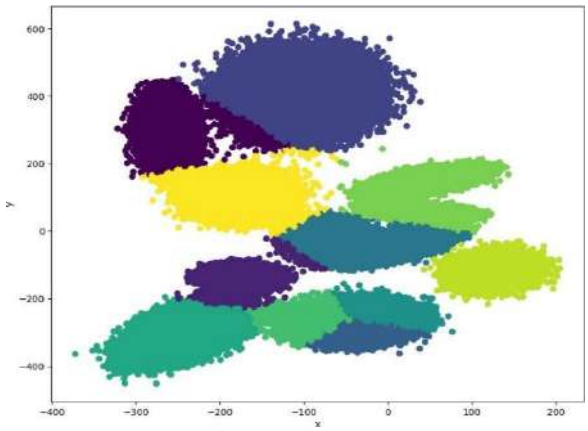


Fig. 6. Results of Parallel K-Means on 2D Dataset

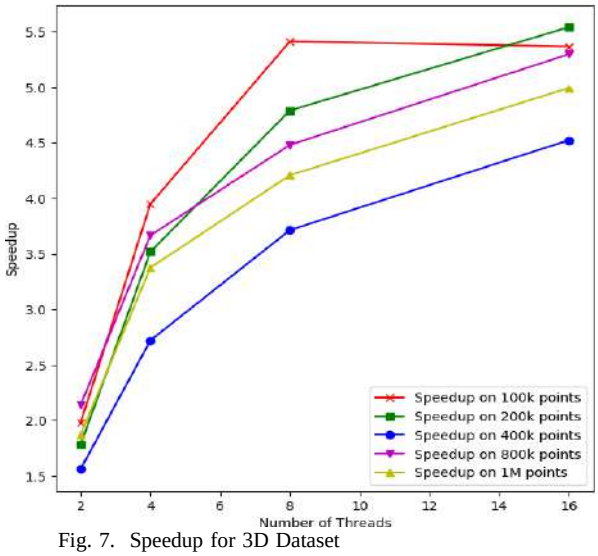


Fig. 7. Speedup for 3D Dataset

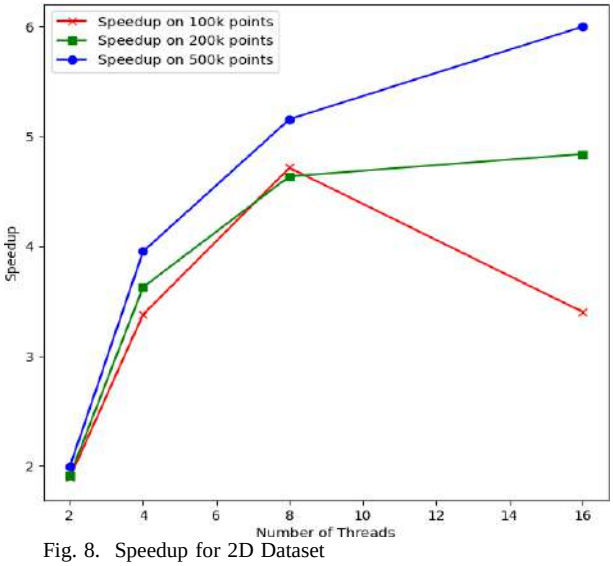


Fig. 8. Speedup for 2D Dataset

that for the same value of K, the time taken to compute the clusters increases as the size of the dataset increases. We also observed that for the same dataset size OpenACC framework performs better than OpenMP in terms of time taken. Another observation is that with OpenACC time taken does not increase much as we scale the dataset. Implications of this are further discussed in the conclusion.

VI. Conclusion

Both parallel versions were able to produce results with no loss in accuracy and an appreciable decrease in computation time. Within the OpenMP framework, we achieve significant speedup as we increase the number of threads, unless thesize of the dataset is too small, in which case the cost of spawning threads outweighs the computation cost. When comparing both the OpenMP and OpenACC frameworks, it is observed that the OpenACC version performs better in terms of saving computation time. This provides scope for decreasing computation time in extremely large datasets with real-world data or complex applications of the clustering paradigm such as image segmentation, anomaly detection, etc.

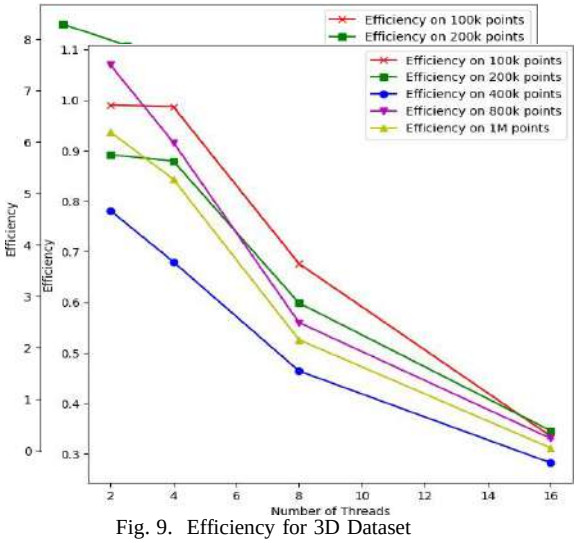


Fig. 9. Efficiency for 3D Dataset

Fig. 10. Efficiency for 2D Dataset

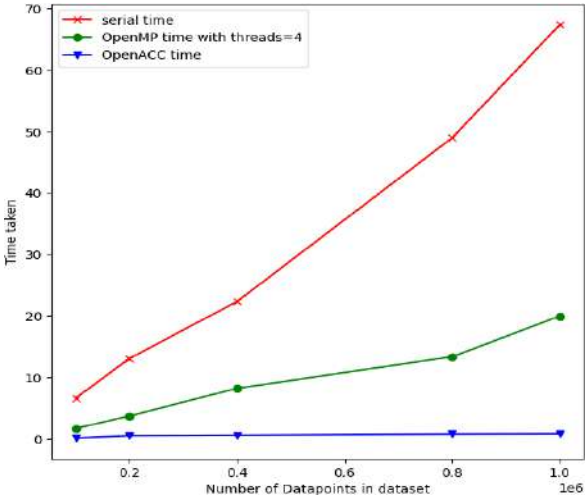


Fig. 11. Time taken vs Scaling for 3D Datasets

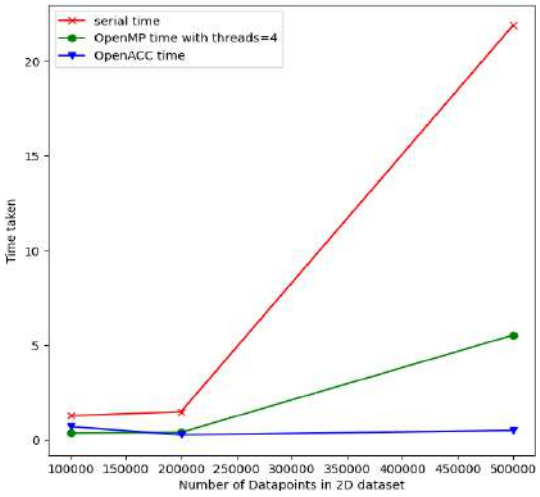


Fig. 12. Time taken vs Scaling for 2D Datasets

REFERENCES

[1] D. S. B. Naik, S. D. Kumar and S. V. Ramakrishna, "Parallel processingof enhanced K-means using OpenMP," IEEE International Conference on Computational Intelligence and Computing Research, 2024.

[2] Kwedlo, Wojciech and Czochanski, Pawel J., A Hybrid MPI/OpenMP Parallelization of *K* -Means Algorithms Accelerated Using the Triangle Inequality. IEEE Access, 2019, pp.42280–42297.

[3] Bhimani, Janki and Leeser, Miriam and Mi, Ningfang, Accelerating K- Means clustering with parallel implementations and GPU computing. 2015 IEEE High Performance Extreme Computing Conference (HPEC), 2015, pp. 1–6.

[4] Baydoun, M., Ghaziri, H. & Al-Husseini, "CPU and GPU parallelizedkernel K-means" J Supercomput, 2018.