# K Means Clustering using OpenMP

| Student Name (First Last) | Major Contributions |
|---|---|
| Sivaprasad Reddy – K966S589 | |
| Harshavardhan – U848H239 | |
| Harsha Vardhan Reddy – R577P973 | |
| Optional Comments: | |

## ABSTRACT

Popular unsupervised machine learning methodology K-Means clustering (partitioning technique) has been applied in a variety of scientific research and business applications, including image processing, information retrieval, social sciences, and weather forecasting. K means algorithm varies by selecting the initial centroids, distance measures and mean calculations. The performance of the algorithm gradually deteriorates as the dataset size increases (increases time complexity O(NxKxR) ). We suggest an algorithm that produces correct results while maintaining accuracy and time complexity. Time complexity is a metric used in algorithm analysis to determine whether an algorithm is efficient or not. Therefore, the O(nlogn) time complexity is maintained by the suggested approach. The experimental results show that the suggested technique performs well even as the size of the data set increases. The proposed approach was developed using OpenMP in a parallel setting (C).

## Keywords

Data mining, Data set Clustering, Partitioning Time Complexity, K-Means and Distance Measures, Parallel processing, OpenMP.

## 1. INTRODUCTION

The process of extracting knowledge from large amounts of records that are kept in databases or other information repositories is known as data mining. Every day, large amounts of information are collected in numerous business and scientific fields. The predictive analytic methods use the data from the repositories as an input to produce intriguing outcomes that aid in decision-making.

Clustering is a crucial data mining technique that aids analysts in comprehending how the attributes in the data are grouped. Data mining, pattern identification, pattern classification, data compression, machine learning, image analysis, and bioinformatics are just a few fields where clustering is applied.

Clustering is a major data analysis method which is widely used in lots of applications in many technical areas. It is a method of placing the objects of same type into one group and the objects in one group will be different from other group [8, 9]. Application of high quality clustering method results in high quality clusters. The quality of a clustering produced depends on the similarity measure used in this method and also by its ability to discover some or all of the hidden patterns.

### K Means Clustering

One of the most straightforward and well-liked unsupervised machine learning algorithms is K-means clustering. The dataset is divided into K pre-defined, separate, non-overlapping subgroups (clusters) using an iterative process, and each data point is assigned to a single cluster. Finding hidden patterns in the provided data is the main goal. In other words, k-means identifies observations that share crucial traits and groups them into clusters. Among the many practical uses of clustering are customer segmentation, document classification, house price estimation, and fraud detection.

It is based on Euclidean distance and is used to compute the distance among data points and locate the centroids. Suppose, let us consider the two sets of data points X and Y. Let X = (x1, x2, x3... xm) and Y = (y1, y2, y3... ym). The Euclidean distance can be calculated as follows:

*Proposed Sequential Algorithm & Method*

$$d(X, Y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_m - y_m)^2}$$

Input: D = {d1, d2, d3… dn}

K= Initial Clusters

Output: K Clusters

## Procedure (Algorithm):

1. Randomly choose k number of data points from input set D as preliminary centroids.

2. Calculate the Euclidean distance between each data point di and preliminary centroids. Group the data point di to the cluster whose distance is minimum.

3. Repeat the above step until no data points is left.

4. Then, Compute the new centroid for each group of k cluster formed by accumulating data points using Euclidean distance measure.

5. Continue the process till the convergence is met.

6. Finally we get K clusters.

This approach must iteratively calculate the distance between each data point and the centroids. The time complexity will then change to O (N x K x R). Where K is the number of

clusters or centroids, N is the total number of data points, and R is the total number of iterations. Sequential programming is used for the procedure and time complexity mentioned above. The definition of "parallelism" is given here. The approach has good parallelism potential because the calculation of the suitable centroid for each data point is independent of the calculations for the rest.

## Parallel Processing/Parallelism

Parallel processing is used to solve a computational problem. A given task is decomposed into discrete parts and executed parallel with the help of child processes. These child processes executes simultaneously on different CPUs. There are multiple APIs or programming interfaces for parallelism like OpenMP, MPI-2 and CUDA. In this problem, we are using OpenMP for parallelism.

### OpenMP

OpenMP is a standard Application Programming Interface of C, C++ and FORTRAN for a shared memory parallel programming. It provides a model for parallel programming that is portable across shared memory architectures. Compilers from numerous vendors support the OpenMP API. To parallelize the code in OpenMP, the compiler directives are used.

It creates as many threads which are processing cores in the system. Thus, for a dual-core system, two threads are created, for a quad-core system, four are created; and so forth. Then all the threads simultaneously execute the parallel region. When each thread exits the parallel region, it is terminated. OpenMP provides several additional directives for running code regions in parallel, including parallelizing loops. In addition to providing directives for parallelization, OpenMP allows developers to choose among several levels of parallelism. E.g., they can set the number of threads manually. It also allows developers to identify whether data are shared between threads or are private to a thread.

## 2. RELATED WORK

Several attempts were made by researchers to improve the effectiveness and efficiency of the K-means algorithm.

Yuan et al. [1] have anticipated a technique to find out the initial centroids but it did not recommend any enhancement for deriving the time complexity of the algorithm.

Fahim AM et al. [2] proposed a better technique for assigning data-points to clusters. The original k-means algorithm suffers from more time complexity because for each iteration, we need to calculate the Euclidean distance between the data points and all preliminary centroids. The approach discussed have used two distance methods out of which one was similar to the clustering algorithm and the other one was application of heuristics to minimize the number of distance computations. But, the weakness that was identified in this approach was that the initial centroids were selected in random as it was done with the basic k-means clustering algorithm. Also, the performance in terms of efficiency of the final clusters was not assured.

Rasmussen et al. [3] recommended the parallel processing using Distributed Array Processor and it have shown a considerable improvement over serial processing for the hierarchical agglomerative cluster investigation of large data sets.

Olson et al. [4] have also recommended parallel algorithms on hierarchical clustering by making use of different distance metrics to find out the interclusters. Even though many measure applied, the expected results were not up to the mark.

Zhao et al. [5] projected a speedy parallel K-means algorithm based on MapReduce. It ca efficiently to process large datasets hardware.

## Proposed parallel Partioning

The performance of cluster analysis on multi-core computers for very big datasets and also to outliers is introduced using a parallel partitioning K-Means algorithm with outlier analysis in OpenMP. Preprocessing is used to get rid of noisy or missing data from the raw huge datasets that will be categorized. Preprocessed data is subjected to the best sorting algorithm, arranged in ascending order, and then divided equally into L partitions. Initialize the OpenMP process for each partition in order to carry out parallel processing on multi-core platforms. Mean values are calculated for each partition to create K local clusters until convergence is achieved. The results obtained at each K local clusters formed at P processes is synchronized to apply a global reduction mechanism to construct a global K clusters. This procedure is continued till the convergence is met. The outlier is also handled by applying the one of the best sorting technique to the preprocessed data, so that the data points in the large datasets are rearranged in an ascending order. The computational time is reduced to a great extent without effecting the time complexity and efficiency.

*Algorithm of the proposed method*

**Input:** *D= {d1, d2, d3, d4,.….dn}*

*K*= Initial Clusters

**Output:** *K* Clusters

### Procedure:

1. Initialize the OpenMP Processes as *P={P0, P1, P2,... Pn}*
2. Take the preprocessed dataset and apply best sorting technique and arrange the datasets in an ascending order.
3. Partition the sorted dataset into *k* parts and assign each part of dataset to each process.
4. For each partition, calculate the mean values and form the *k* local clusters.
5. Continue the process to form final *K* local clusters till the convergence is met.
6. Now, synchronize the results from the individual process
7. Then obtain the Global *K* clusters from all local *K* clusters.
8. Continue the procedure till the convergence is met.

Key features of the algorithm:

● Initialization: The first K data points are chosen as the initial centroids.

● Data-parallelism: Each thread assigned N/num_threads number of points.

● Thread function: Each thread runs a loop (with a max_iter value of 100), in every iteration of which it computes the closest cluster

centroid for every point assigned to it, and then assigns the point to that cluster. After every point is assigned to a cluster, the global (shared) cluster centroids are updated with the values computed for their coordinates from the points that were assigned to that cluster. This is again an instance of data-parallelism.

● Stopping-condition: The L2-norm is computed for every cluster centroid coordinates by comparing against corresponding values in the previous iteration. The norms are then summed and compared against the threshold value, chosen to be 1e-6. All threads break from the iterations loop as soon as the delta value goes below the threshold.

Mutual-exclusion and Critical section: The first thread-synchronization strategy which ensures that updates to the shared cluster centroid array are undertaken by each thread only is undertaken using pragma omp critical construct. This is essential to avoid data race due to different threads attempting to modify the shared array.

Barriers: After every iteration, barriers are used for thread synchronization so that all threads view the same updated value of the centroids array. Two instances of barriers are used. The first one for synchronization of the centroids update, and the second one following the first to synchronize the value of delta, again a shared variable among all threads. The pragma omp barrier construct is used for progress synchronization.

False-sharing: To minimize false sharing, shared variable updates for the centroids array are minimized. Moreover, local variables are used wherever feasible (for instance in delta value computations, variable temp is used to store calculated delta value which is then updated at the end of thread execution).

Load-balancing: The nature of parallelization probably does not require an explicit load-balancing strategy since it's an SPMD-based parallelization scheme. One obvious source of unbalanced load would be the case of a slightly larger number of points (bounded by num_threads) processed by the last thread.

## 3. EXPERIMENTAL RESULTS

The proposed method is implemented on desktop with Intel core2duo processor with 3GB RAM and 500GB hard disk. The desktop machine is booted with windows XP operating system and the parallel and distributed environment is the Windows version of OpenMP standard is used.

The Proposed method has evaluated on different set of datasets starting from 10K, 50K, and 1 Lakh, 2 Lakh, 5 lakh & 10 Lakh. The accuracy of the proposed method outperforms sequential K Means algorithm is almost 90.8% in all cases.

The CPU Times of the proposed parallel partitioning method is also optimized compared with Sequential K-means method. The below table shows the CPU times (in sec) of sequential K-means with that of proposed method.

GitHubcode: https://github.com/Sivaprasadramb/kmeansopenmp

| Thread/ Datasets | 10K | 20K | 1 Lakh | 2 Lakh | 5 Lakh | 10 Lakh |
|---|---|---|---|---|---|---|
| 1 | 0.016 | 0.175 | 0.383 | 0.142 | 0.672 | 1.204 |
| 2 | 0.010 | 0.121 | 0.267 | 0.125 | 0.459 | 0.745 |
| 4 | 0.005 | 0.072 | 0.168 | 0.075 | 0.276 | 0.408 |
| 8 | 0.003 | 0.043 | 0.103 | 0.042 | 0.176 | 0.270 |
| 16 | 0.004 | 0.045 | 0.110 | 0.044 | 0.162 | 0.281 |

Table: The practical execution time between Threads & Datasets

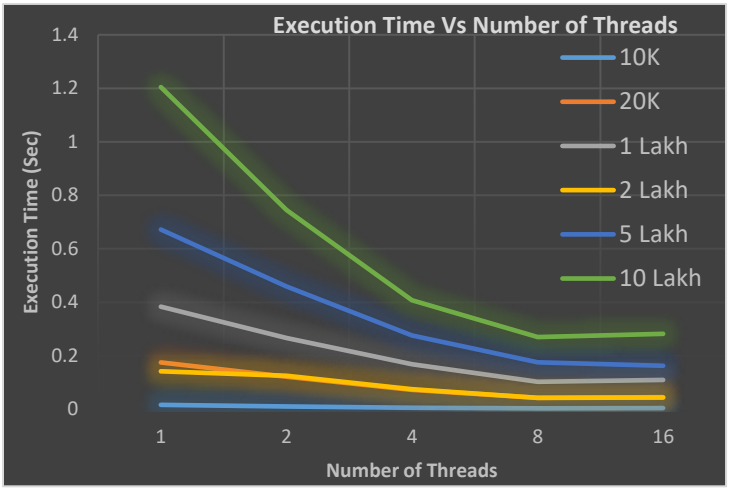


Fig: Graph between Execution Time and Number of threads

As expected from any multithreaded programming method, the computation time of the algorithm decreases as the number of threads increases so much so that for 16 threads, the overall computation time for various datasets is close enough to zero. The difference between computation times of different datasets on a fixed number of threads is different because of the unequal distribution of points for each of the respective threads.

The Speedup of the system is calculated from the sequential execution time and parallel execution time. The below table shows the speed of the system that we proposed.

| Thread/ Datasets | 10K | 20K | 1 Lakh | 2 Lakh | 5 Lakh | 10 Lakh |
|---|---|---|---|---|---|---|
| 1 | 0.972 | 1.017 | 0.956 | 0.99 | 1.776 | 0.552 |
| 2 | 1.572 | 1.463 | 1.373 | 1.12 | 2.596 | 0.893 |
| 4 | 2.776 | 2.456 | 2.173 | 1.877 | 4.316 | 1.63 |
| 8 | 4.306 | 4.128 | 3.559 | 3.277 | 6.771 | 2.463 |
| 16 | 3.388 | 3.949 | 3.332 | 3.188 | 7.332 | 2.361 |

Table: The practical Speedup between Threads & Datasets

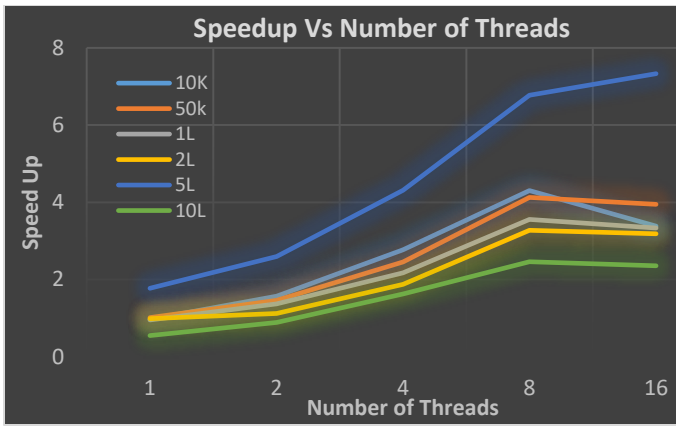


Fig: Graph between speedup and number of threads

Theoretically, if the problem size is too low, the speedup decreases as the number of cores increases. This is because the cost of overheads will be far more than actual cost of computation, thereby giving low speedup. But for larger problem sizes, the overhead cost is negligible and the work could be distributed among many processors, thus the speedup increases as the number of processors increases. The best speedups are obtained for the larger-sized threads (5 Lakhs).

## 4. CONCLUSION

Most of the current clustering algorithms are less effective when dealing with outliers and huge datasets. On multi-core systems, a parallel improved clustering algorithm is developed in order to improve accuracy while requiring the least amount of time. In the suggested approach, Data must first be pre-processed. Prior to being separated into divisions, the preprocessed data must be sorted in ascending order. Calculate the Mean and set the starting point to centroids for each split. Calculating the global Mean is an iterative process that produces K-clusters. Results from experiments using various datasets show that the suggested method consistently achieves greater than 90.8% accuracy. When compared to the Sequential K-means approach, the CPU times of the suggested parallel partitioning method are also optimized. With more threads, the execution (computation) time is faster. As the number of threads is increased, the speedup will also increase.

## Future Scope:

The given algorithm is coarse grain, especially for large data inputs. In such cases, it might be a good idea to implement the given code on a distributed memory system. Also, MPI provides special libraries for parallel data input, which can be a huge advantage in terms of speedup. Also we have stuck to just the clustering of 3 dimensional data points. We can attempt to cluster multi-dimensional data points along with implementing dimensionality reduction. We can try to implement this using GPU's but this would be quite challenging since the cost of overheads of GPU's is paid only if we are working with hundreds of threads at the same time.

## 5. REFERENCES


[1] F. Yuan, Z. H. Meng, H. X. Zhang, and C. R. Dong, "A new algorithm to get the initial centroids," *Proc. of the 3rd International Conference on Machine Learning and Cybernetics*, pp. 26–29, 2004.

[2] Fahim A.M, Salem A. M, Torkey A and Ramadan M. A, "An Efficient enhanced k-means clustering algorithm," *Journal of Zhejiang University*, 10(7):1626–1633, 2006.

[3] E. Rasmussen, P. Willett, "Efficiency of hierarchic agglomerative clustering using the ICL distributed array processor," *Journal of Documentation*, vol. 45, March 1989

[4] C. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313-1325, August 1995.

[5] W. Zhao, H. Ma, Q. He, "Parallel K-Means Clustering Based on MapReduce in Cloud Computing," vol. 5931, pp. 674-679, 2009.

[6] K. A. Abdul Nazeer, SD.Madhu Kumar and M. P. Sebastian, "Enhancing the k-means clustering algorithm by using a O(n logn) heuristic method for finding better initial centroids," in Second International Conference on Emerging Applications of Information Technology, pp.261264,2011.

[7] Zhenhua LV, Yingjie Hu, Haidong Zhong,Jianping Wu,Bi Li and Hui Zhao, "Parallel K-Means clustering of Remote Sensing Images based on Map reduce," *WISM 2010,LNCS6318*, pp.162-170,2010.

[8] Mohammed El Agha, Wesam M. Ashour, "Efficient and Fast Initialization Algorithm for K-means Clustering," *I.J. Intelligent Systems and Applications*, vol.1, pp.21-31,2012.

[9] Zhou Haiyan, Zhao Jianyang, "A new K-means Initial Cluster Class of the Center Selection Algorithm Based on the Great Graph," *In Communications in Information Science and Management Engineering*, vol.2, pp.1720,2012.