Parallel implementation of K-means clustering algorithm using OpenMP, MPI and Pthread: Project Report

Bethelehem Dagnachew
(FTP0215/09)

Rufael Fekadu
(FTP0835/09)

August 28, 2021

Abstract

Nowadays, more and more complex algorithms are being developed for different purposes. One area that seem to be every where today is machine learning. Machine learning algorithms require lots and lots of data to perform better. In fact it is said that a machine learning algorithm is as good as the data it has been trained on. But as the data used to train the algorithm is increased, so does the execution time. So programmers and data scientists need to implement faster and more efficient programs in order to get the best performance. In this project we plan to implement K means clustering, a very popular unsupervised learning algorithm, using OpenMP and MPI libraries to get a faster implementation.

Overview of the project

In this project, we implement serial and parallel implementation of K-means clustering, which is a simple unsupervised learning algorithm that is used to group data points by analysing their patterns. The task of k-means is to divide points within a space in K groups based on their characteristics. Here we implement The K-means Algorithm using OpenMP, MPI and pthread libraries. The performance of the serial and parallel implementation is compared using number of clusters and number of threads. Also, the Speed up and Efficiency of k-means Algorithm is analysed for both serial and parallel implementation. The report also contains the observed performance of the k-means clustering using different datasets.

Kmeans Clustering

kmeans Clustering is used when you have unlabeled data, data without defined categories or groups. The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K. The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. [1]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 [2].

kmeans clustering works on the following steps[3]

1. Choose the number of clusters k
2. Select k random points from the data as centroids.
3. Assign all the points to the closest cluster centroid.
4. Recompute the centroids of newly formed clusters.
5. Repeat steps 3 and 4 until the centroids doesn't move or the maximum iteration is reached.

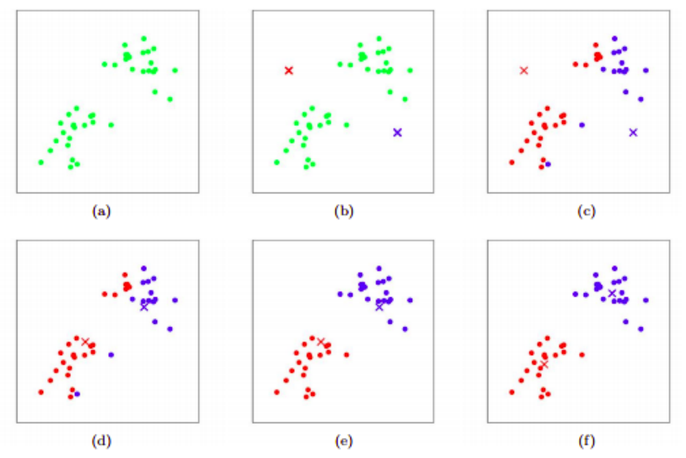


Figure 1: Steps in kmeans clustering.

OpenMp

According to [4], OpenMP is a library for parallel programming in the SMP (symmetric multi-processors, or shared-memory processors) model. When programming with OpenMP, all threads share memory and data. OpenMP supports C, C++ and Fortran. The OpenMP functions are included in a header file called omp.h. More efficient, and lower-level parallel code is possible in OpenMP, however OpenMP hides the low-level details and allows the programmer to describe the parallel code with high-level constructs, which is as simple as it can get.

MPI

According to [5], Message Passing Interface(MPI) is a system that aims to provide a portable and efficient standard for message passing. It is widely used for message passing programs, as it defines useful syntax for routines and libraries in different computer programming languages such as Fortran, C, C++ and Java.

Pthread

[6]POSIX Threads, usually referred to as pthreads, is an execution model that exists independently from a language, as well as a parallel execution model. It allows a program to control multiple different flows of work that overlap in time. Each flow of work is referred to as a thread, and creation and control over these flows is achieved by making calls to the POSIX Threads API. It specifies a set of interfaces (functions, header files) for threaded programming commonly known as POSIX threads, or Pthreads. A single process can contain multiple threads, all of which are executing the same program. These threads share the same global memory (data and heap segments), but each thread has its own stack (automatic variables).

Kmeans Implementation

Serial Kmeans

There are different ways to implement the kmeans algorithm using c, but various issues need to taken under consideration. Issues like memory management and complexity, as there seems to be a trade off between them. That is, the more simple and elegant the implementation is the more memory it consumes. For the sake of simplicity and readability of the code, we chose a convenient implementation of the algorithm.

Structures implemented

The following structures were implemented as shown in table 1.

#	Struct	Attribute	discription
1	struct Cluster	double x_coord double y_coord double new_x_coord double new_y_coord int size	Holds info about a single cluster
2	struct Point	double x_coord; double y_coord int cluster_id	Holds info about a single point
3	struct Clusters	cluster c; int item int size	Holds info about vector of Clusters
3	struct Points	point p; int item int size	Holds info about vector of points

Table 1: Structures in the implementation.

Functions implemented

The following functions were implemented as shown in table 2

#	Function	discription
1	point init_point(int x_cord,int y_cord)	Initializes and returns a point
2	cluster init_cluster(int x_cord,int y_cord)	Initialize and return a cluster
3	points init_points(const char *dataset_filename)	Initializes and returns a vector of points from a file
4	clusters init_clusters(int num_cluster)	initialize and return a vector of clusters
5	double euclidean_dist(point* point, cluster* clust)	Calculates the eucledian distance between a point and a cluster centroid
6	void compute_distance(points* po, clusters* clust)	Compute distance between each point and each cluster and assign the points to the clusters.
7	void free_point(cluster* clu)	prepares the cluster for the next iteration by resetting its size.
8	bool update_coords(cluster * clus)	check if the centroid of cluster is to be moved and moves it

Table 2: Functions in the implementation.

Parallelization Strategy

The clustering algorithm requires massive computations, with distance between each data point and each centroid being calculated. Since calculation of the appropriate centroid for each data point is independent of the others, the algorithm provides a good scope for parallelism.

For parallelization of the k-means algorithm, our primary approach is to adopt data-parallelism. The N points were equally split among the number of threads (in case of an imperfect data split among threads, the remainder points are allotted to the last thread). The following approaches were made in order to implement the parallel algorithm.

Openmp parallelization Strategy

We parallelize the program on two parts.

- **Initialization:** The first task we tried to parallelize is the initialization of the points and clusters using "#pragma omp sections".
- The main part of the parallelization comes inside the compute_distance function. we used "#pragma omp for" to parallelize the outer for loop that computes the eucledian distance between each point and every cluster.

Pthrad parallelization Strategy

The Pthread use similar implementation strategy with the openMP. Only the syntax is different from the openMP. The function we used in the implementation is also similar with the openMP implementation. However we modiffy the Compute distance function with compute thread function.

MPI parallelization Strategy

In order to perform Kmeans using MPI we adopted a slightly different implementation from the serial one. since message need to be passed between processes, the serial implementation in this project need to be modified. That is, instead of using a more modularized implementation,like the one in the serial implementation, we used a single array to hold the data points because it is convenient for message passing.
The functions we used are listed in table 3

#	Function	discription
1	float* init_points(const char *dataset_filename, int site_per_proc,int nprocs)	Initializes and returns a point
2	float distance2(const float *v1, const float *v2, const int d)	copmute the L2 norm of two vectors
3	int assign_site(const float* site, float* centroids, const int k, const int d)	assign point to cluster
4	void add_site(const float * site, float * sum, const int d)	Adds a point to the cluster

Table 3: Functions in the mpi implementation.

Tests

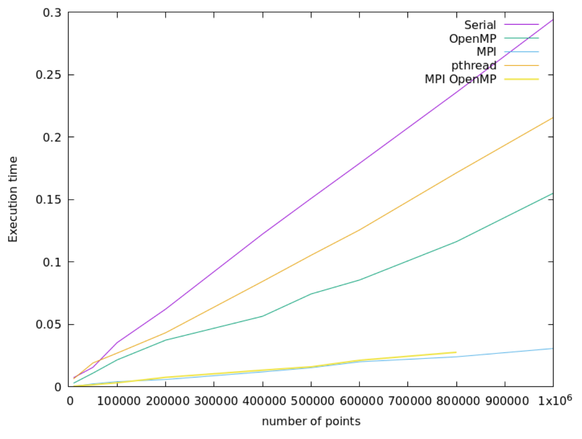
We tested both the serial and parallel code using performance parameters. Both the serial and parallel implementations were tested by using different number of clusters and different number of dataset points. We compare the serial and parallel implementations performance in terms of speed and efficiency parameter. We Also compare the Openmp, Pthread and MPI parallel implementation using performance parameters. Further more we implement the hybrid of the Openmp and the MPI to check the performance of the parallel implementation.

For the purpose of analysing the execution time, we chose to look at the time per iteration of the execution since random initialization of the clusters will end up going different number of iterations.

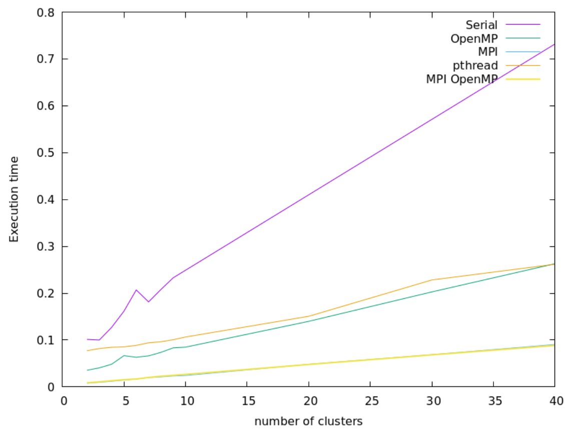
Results

We have implemented and evaluated the OpenMP,pthread and MPI parallel implementaion of kmeans clustering. further we also compare the performance of each implementation. Also we mix OpenMP implementation with MPI implementation so that the drawback of one method could be solved by the other.

As shown in the figure 3, as the number of clusters increase the execution time of all of the implementations increase. further more, the MPI implementation has lowest execution time for different number of input points, and the time difference only grows larger and larger. Also it can be shown from fig 2b that, as the number of threads is increased the execution time decreases up to some point and becomes constant. In figure 3b it can clearly be shown that, as the number of process is increased the execution time of the MPI implementation starts to decrease but after a certain point it grows linearly. This is expected since the overhead related to creating a processes is very costly.

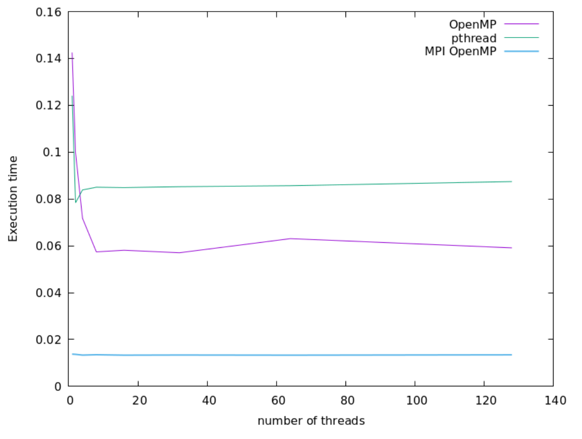


(a)

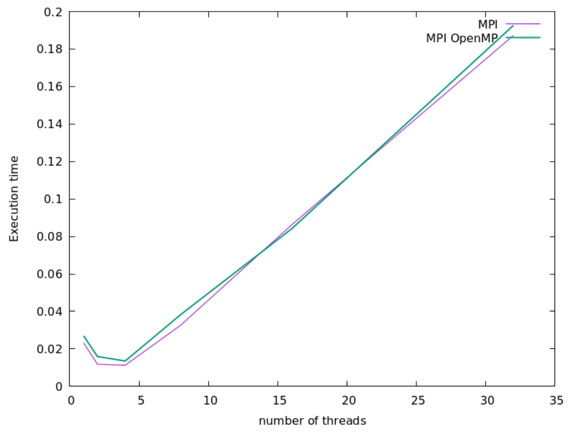


(b)

Figure 2: Execution time of the serial,OpenMP, Pthread, MPI and hybrid of MPI openMP implementation (a) Vs the number of dataset points (b)Vs the number of cluster



(a)



(b)

Figure 3: Execution time of the serial,OpenMP, Pthread, MPI and hybrid of MPI openMP implementation (a) Vs the number of threads (b)Vs the number of process

Challenges

Although we made significant work in the project, there were challenges along the way which made the project difficult. The first one is the lack of resources and documentation. Even though there are plenty of publications in this area we were not able to access them because of access limits. Another and more technical challenge we faced were the issue of choosing a memory efficient serial implementation of the kmeans clustering algorithm.

Conclusion

In this project we implement the k-means clustering using Openmp, MPI and Pthread. We also test the performance the k-means clustering by hybrid of Openmp and MPI implementation. It was observed that serial K-means execution is much slower than parallel execution. And the MPI implementation is much faster than the OpenMp, Pthread, hybrid MPI and OpenMp and serial implementation. And the OpenMp implementation give better performance than the Pthread implementation.

References

[1] Andrea Trevino. Introduction to k-means clustering. 2016.

[2] S. Deva Kumar Ds Bhupal Naik and V Ramakrishna Sajja. Parallel processing of enhanced k-means using OpenMP. 2013.

[3] Techopedia". K-Means Clustering. https://www.techopedia.com/definition/32057/k-means-clustering, 2016.

[4] Blaise Barney. OpenMP. https://hpc.llnl.gov/tuts/openMP/, 2021.

[5] Techopedia". Message Passing Interface (MPI). https://www.techopedia.com/definition/115/message-passing-interface-mpi, 09 2015.

[6] pthreads- linux manual page. https://man7.org/linux/man-pages/man7/pthreads.7.html, 2010.