Parallel Computing
Parallel K-Means Algorithm

1st Rodrigo Rodrigues
Informatics Department
University of Minho
Braga, Portugal
pg50726@alunos.uminho.pt

2nd Daniel Azevedo
Informatics Department
University of Minho
Braga, Portugal
pg50311@alunos.uminho.pt

***Abstract*—Clustering is one of the most popular methods for exploratory data analysis, which is prevalent in many areas such as image segmentation, bioinformatics, pattern recognition and statistics etc. The most famous clustering algorithm is K-means because of its easy implementation, simplicity and efficiency. In addition, MPI (Message Passing Interface) as a message passing programming model features high performance, scalability and portability. Motivated by this, a parallel K-means clustering algorithm with MPI is proposed in this article. O algorithm allows you to apply the clustering algorithm effectively in the parallel environment.**

***Index Terms*—K-means, Algorithm, Parallel, MPI, Master, Worker**

I. Introduction

This phase of practical work aims to assess the development of programs that explore parallelism efficiently, eventually using alternate parallel programming environments and always having as main objective the reduction of the execution time of the program.

This phase of practical work aims to assess the development of programs that explore parallelism efficiently, eventually using alternative parallel programming environments and always having as main objective the reduction of the execution time of the program in relation to the sequential version.

In this phase of the practical work, several options were given to improve/test the implementation of the parallel version of the k-means algorithm developed in previous works. For the development of this last phase, we decided to explore another computing platform/programming environment that exploits parallelism, namely MPI.

K-means can be implemented with MPI (Message Passing Interface) to parallelize the computation and improve the performance of the algorithm.

Furthermore, we will carry out a detailed study of the scalability of the implementation taking into account the chosen input sizes suited to each of the levels of the machine's memory hierarchy.

Finally, we will explain the results obtained as well as a theoretical analysis.

II. A Distributed MPI Implementation

Given that the k-means clustering problem is a computationally intensive NP-Hard problem, a distributed approach is appropriate, since the computational load is divided between several work processes, so the program can be parallelized and accelerated.

To implement K-means with MPI, the following steps can be followed:

1. Initialize the centroids of the clusters randomly or using some other method.

2. Distribute the data among the MPI processes, with the values of the first points.

3. Have each process calculate the distance between each point in its portion of the data and the centroids of the clusters.

4. Have each process assign each point to the cluster with the nearest centroid.

5. Have each process calculate the new centroids of the clusters based on the points that have been assigned to them.

6. Have the processes share the updated centroids with each other.

7. Repeat steps 3-6 until the centroids converge or a maximum number of iterations has been reached.

Essentially, the master process evenly distributes the dataset among the N number of workers.

After all the workers present their results to the master process, it will check if the clusters have already converged, if so it will present the results, otherwise the entire process described above will be repeated, for several iterations.

III. Implementation Scalability

One of the main advantages of k-means is that it is relatively fast and scalable, particularly when compared to other clustering algorithms. It is able to handle large datasets because it only requires simple distance calculations and does not involve any expensive optimization procedures.

The following table will present the execution times varying the number of points and clusters, using 8 processes.

Number of clusters	Number of points	Time(s)
4	2500000	0.353370
4	5000000	0.714897
4	7500000	1.051791
4	10000000	1.404700
8	10000000	1.779580
12	10000000	2.231941
24	10000000	3.681057
32	10000000	4.473551

Looking at the table we can see that there are some factors that can affect the scalability of k-means:

The number of data points: As the number of data points increases, the time required to compute the distances between each data point and the cluster centers will also increase.

The number of clusters: As the number of clusters increases, the time required to compute the cluster centers will also increase.

Regarding the number of clusters, the solution has some scalability since when we double the number of clusters, the execution time does not increase proportionally. This can no longer be said in relation to the increase in the number of points, since the increase is proportional.

IV. Load Balancing

In the k-means clustering algorithm, load balancing refers to the process of distributing the workload of the algorithm evenly across all of the available processors in a parallel computing environment. This is important because it ensures that the algorithm runs efficiently and effectively by avoiding situations where some processors are idle while others are overloaded.

There are several strategies that can be used to achieve load balancing in MPI-based k-means implementations.

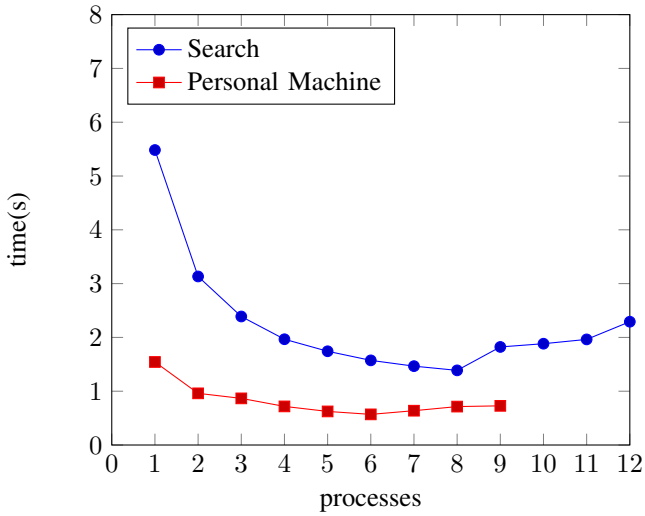
The approach that was used to balance the load is divide and conquer, where the data is divided into smaller blocks and each block is assigned to a separate process. This was done using an MPI function like MPI_SCATTER. The k-means algorithm is applied to each block of data in parallel and the results are combined using an MPI function like MPI_Reduce.

Overall, the key to achieving good load balancing in an MPI-based k-means implementation is to carefully design the parallelization strategy and choose the appropriate MPI functions to distribute the workload and communicate data between processes.

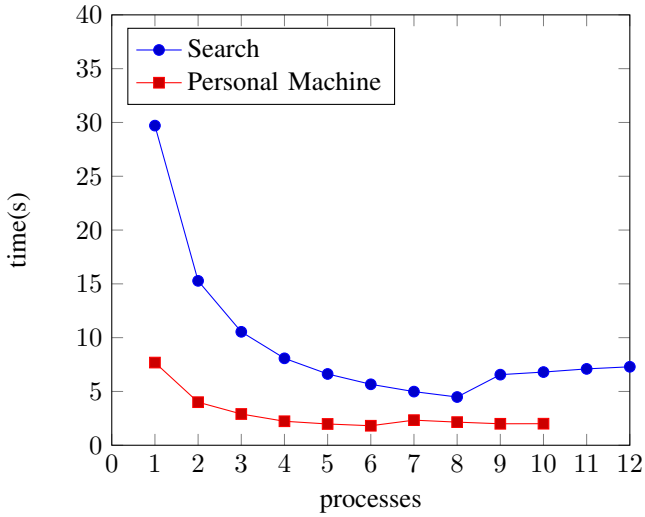
V. Results

In order to test the parallel k-means algorithm, we resorted to two machines, one being the Search and the other a personal machine, having a processor with 6 core and 16 GB of RAM.

The following graph shows the execution time as a function of the number of processes, using 10000000 points and 4 clusters.



The following graph shows the execution time as a function of the number of processes, using 10000000 points and 32 clusters.



As we can see from the previous graphs, in the case of search the ideal number of processes is 8 for both types of input, increasing the number of processes will increase execution time. This increase is due to the fact that there is greater complexity in the communication between the different processes. The same happens when we test on the personal machine, however the ideal number of processes is 6.

A. Limitations

Creating too many worker nodes in hopes of further speeding up parallelization can actually impede its progress. There must be a balance between the workload to be distributed to each worker, and the actual number of worker nodes. If there are more workers than necessary, much of the computational efforts are spent distributing the work (message passing), and not spent computing the actual workload. Thus one of the defining characteristics of how well the distributed program will run is its message complexity and how many messages are being sent back and forth throughout the network.

VI. Conclusion and future work

K-means is an algorithm that is often used in parallel computing because it is relatively simple to implement and can be applied to large datasets efficiently. Parallelization of k-means can be achieved through a variety of methods, such as dividing the data into smaller chunks and processing them concurrently on different processors.

One of the main benefits of parallelizing k-means is the ability to reduce the runtime of the algorithm, especially for large datasets. This can be especially useful in cases where the algorithm needs to be run multiple times or where real-time processing is required.

However, it is important to note that parallelizing k-means can introduce additional complexity and overhead and may not always result in significant performance improvements. As such, it is important to carefully consider the tradeoffs between the potential benefits and costs of parallelizing k-means before implementing it in a parallel computing environment.

In the future we could implement a similar model on other interface, for example CUDA.