IMPROVEMENTS IN K-MEANS ALGORITHM TO EXECUTE ON LARGE AMOUNTS OF DATA

Erhan SÜLÜN

OCTOBER, 2004

Improvements in K-means Algorithm to Execute on Large Amounts of Data

By

Erhan SÜLÜN

A Dissertation Submitted to the

Graduate School in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Department: Computer Engineering

Major: Computer Software

Izmir Institute of Technology

Izmir, Turkey

October 2004

We approve the thesis of **Erhan SÜLÜN**

Date of Signature

27.10.2004

.....
Prof. Dr. Halis PÜSKÜLCÜ
Supervisor
Department of Computer Engineering

27.10.2004

.....
Prof. Dr. Sıtkı AYTAÇ
Department of Computer Engineering

27.10.2004

.....
Prof. Dr. Fikret İKİZ
Ege University
Department of Computer Engineering

27.10.2004

.....
Prof. Dr. Sıtkı AYTAÇ
Head of Department

ACKNOWLEDGEMENTS

I would like to give my deepest thanks and respects to my advisor Prof. Dr. Halis PÜSKÜLCÜ for his encouragement, suggestions, valuable comments, support and supervision in all steps of this study.

I also owe very special thanks to Asst. Prof. Dr. Haluk TOPÇUOĞLU for providing a parallel programming laboratory for the executions of parallel program.

Finally, I owe my very special thanks to my family, Mrs. Perihan SÜLÜN and Mr. Necati SÜLÜN, for their encouragements and patience.

ABSTRACT

By the help of large storage capacities of current computer systems, datasets of companies has expanded dramatically in recent years. Rapid growth of current companies' databases has raised the need of faster data mining algorithms as time is very critical for those companies.

Large amounts of datasets have historical data about the transactions of companies which hold valuable hidden patterns which can provide competitive advantage to them. As time is also very important for these companies, they need to mine these huge databases and make accurate decisions in short durations in order to gain marketing advantage. Therefore, classical data mining algorithms need to be revised such that they discover hidden patterns and relationships in databases in shorter durations.

In this project, K-means data mining algorithm has been proposed to be improved in performance in order to cluster large datasets in shorter time. Algorithm is decided to be improved by using parallelization. Parallelization of the algorithm has been considered to be a suitable solution as the popular way of increasing computation power is to connect computers and execute algorithms simultaneously on network of computers. This popularity also increases the availability of parallel computation clusters day by day.

Parallel version of the K-means algorithm has been designed and implemented by using C language. For the parallelisation, MPI (Message Passing Interface) library has been used. Serial algorithm has also been implemented by using C language for the purpose of comparison. And then, algorithms have been run for several times under same conditions and results have been discussed. Summarized results of these executions by using tables and graphics has showed that parallelization of the K-means algorithm has provied a performance gain almost proportional by the count of computers used for parallel execution.

ÖZ

Günümüzün büyük saklama kapasiteli bilgisayar sistemlerinin desteğiyle şirketlerin veritabanı boyutları yakın tarihte ciddi bir şekilde artmıştır. Zaman şirketler açısından büyük önem taşıdığı için, günümüz şirketlerinin hızla büyümüş olan veritabanları daha hızlı veri madenciliği algoritmaları ihtiyacını da birlikte getirmiştir.

Şirketlerin tarihsel hareketlerini tutan büyük boyutlardaki veritabanları şirketlere rekabet avantajı sağlayacak olan değerli gizli bilgiler içermektedir. Ayrıca zaman da şirketler açısından çok önemli olduğu için bu şirketler yüklü miktarlardaki veritabanlarını kısa sürede veri madenciliği ile inceleyip kısa sürede kendilerine rekabet avantajı sağlayacak doğru kararları almaları gerekmektedir. Bu nedenle, klasik veri madenciliği algoritmalarının gözden geçirilerek iyileştirilmesi ve daha kısa sürede veritabanlarındaki gizli bilgileri ortaya çıkaracak hale getirilmeleri gerekmektedir.

Bu projede K-means veri madenciliği algoritmasının büyük veri tabanlarını kısa sürede gruplandıracak şekilde geliştirileceği öne sürülmüştür. Algoritmanın, paralelleştirme yöntemi ile geliştirilmesine karar verilmiştir. Günümüzde, işleme gücünün artırılmasının en popüler yolunun bilgisayarların birbirine bağlanması ve algoritmaların bilgisayar ağları üzerinde eş zamanlı olarak çalıştırılması olduğu için paralelleştirme yöntemi bu geliştirme çalışması için uygun görülmüştür. Ayrıca bu popülarite, paralel bilgisayar laboratuarlarının bulunulabilirliğini de günden güne artırmaktadır.

K-means algoritmasının paralel versiyonu C Programlama Dili kullanılarak geliştirildi. Paralelleştirme işlemi için ise MPI (Message Passing Interface) kütüphanesi kullanıldı. Zaman açısından bir karşılaştırma yapılabilmesi için klasik (seri versiyon) algoritma da C Programlama Dili kullanılarak geliştirildi. Daha sonra, algoritmalar aynı şartlar altında birden fazla kez çalıştırılarak sonuçları tartışıldı. Tablolar ve grafikler kullanılarak özet haline getirilen çalıştırma sonuçları göstermiştir ki K-means algorithmasının paralelleştirilmesi sonucunda hemen hemen paralel çalıştırmada kullanılan bilgisayar sayısı kadar performans kazanımı elde edilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Parallel version of the K-means algorithm has been designed and implemented in this project for the purpose of improvement of K-means algorithm in execution time. Serial (Classical K-means) version of the algorithm has also been implemented for the purpose of comparison with parallel the version in time. Both implementations have been tested on the same environment and results have been discussed. As K-means is a clustering algorithm which is a type of data mining algorithm, data mining and clustering have also been examined in the project. KDD (Knowledge Discovery in Databases) has also been discussed, because data mining is a step of it. After addressing where K-means stands, details of serial and proposed parallel K-means algorithms has been presented. Before examining parallel K-means algorithm, parallelisation concept of algorithms has been introduced in order to prepare a background for the details of parallel K-means algorithm. After all, both versions (serial and parallel) of the algorithm have been executed on the same platform and results have been discussed.

1.1 Motivation and Objectives

Historical data is very important for companies in order to make better business decisions for their future. Historical data includes valuable hidden information which is waiting to be discovered. This information can help companies in order to acquire new customers, protect their existing customers, detect fraudulent use of their services, determine security holes of their systems and lots of other things.

In order for a company to gain competitive advantage in the market, it must save all of its historical data, mine this data to gather patterns-relationships and then interpret these patterns-relationships to get the information and knowledge. For this purpose, large companies establish data warehouse departments. Data warehouse departments of

companies collect and store transactional data from the activities of the company during the day. This data is then evaluated by some statistical works (for example OLAP – On Line Analytical Processing for past and data mining for present) in order to help in their decision making.

In recent years, databases of companies have grown dramatically in size. Together with the size of data, amount of strategic and valuable hidden information has also become great. By the grow of historical data, classical data mining algorithms have become inadequate in performance. Classical algorithms seriously lack in performance while trying to mine huge amounts of data which causes companies to loose time and also the marketing advantage.

Need of fast mining of large databases has brought need of revising existing data mining algorithms. In this project, parallelization of K-means algorithm has been proposed as an improvement in classical K-means algorithm in order to cluster large databases in short times.

Objective of the parallelization of K-means algorithm is to make the algorithm produce exactly same results with the serial algorithm in shorter duration. Serial K-means algorithm lacks in performance especially in large databases.

And also, parallel version should not require special hardware to execute on. The parallel algorithm proposed and developed in this project can execute on a network of standard PC's on which classical K-means execute also.

1.2 Used Materials and Environments

C Programming Language has been used in order to develop both serial and the parallel K-means algorithm. For the parallel version of the algorithm a tool called MPI (Message Passing Interface) has been used. This collection of libraries enables the C programmer in order to coordinate computers in the network for the purpose of performing a task in parallel manner.

Two datasets have been used for the executions of the algorithms. First one of the datasets is Color Histogram Dataset (smaller dataset). This dataset has 68,040 records with 32 dimensions. Dataset holds colour histogram information of 68,040 images in other

words. Second dataset is the US Census Dataset (larger dateset). This datset has 1,000,000 records with 68 dimensions. Both of these datasets have been gathered from the KDD (Knowledge Discovery in Databases) archive of University of California, Irvine (http://kdd.ics.uci.edu/). Lots of datasets are published in the web site of this university and these datasets are available to be used in KDD researches. These data are standard for KDD researchers for the purpose of benchmarking.

Execution of serial and parallel K-means programs has been performed on the parallel programming laboratory of Computer Engineering Department of Marmara University. This is a well constructed network of 11 PC's which have linux operating system and MPI library installed. Details of this laboratory have been presented in chapter 5.5.

1.3 Structure of the Study

In Chapter 2 of the project, data mining and KDD (Knowledge Discovery in Databases) have been introduced. Relations of data mining with data warehousing, OLAP (On Line Analytical Processing) and AI (Artificial Intelligence) have also been discussed in this chapter. And also, types of data mining algorithms have been presented shortly.

K-means algorithm, which is the main focus of this project, has been examined in detail in chapter 3. Steps and implementation details of the algorithm have also been provided. Besides, deficiencies of the serial K-means algorithm have been discussed at the end of this section.

Parallelization concept of algorithms in general has been examined in chapter 4. Before the parallelization of K-means algorithm, aim of parallelization, architectures of parallel algorithms, models of parallel algorithms according to the memory access and important things for a parallel algorithm in order to perform its goal have been discussed in this chapter.

In chapter 5, parallelization of K-means algorithm has examined in detail. Steps and implementation details of the parallel version have presented in detail and expectations from the parallel algorithm have been discussed.

Finally in chapter 6, serial and parallel versions of the algorithm are executed several times on two different datasets in order to make a comparison. Results of both algorithms are gathered and discussed in detail in this section.

CHAPTER 2

KNOWLEDGE DISCOVERY IN DATABASES AND DATA MINING

KDD refers to the overall process of discovering useful knowledge from databases. KDD consist of several steps. Data mining refers to a particular one of those steps of overall KDD process. Data mining is the application of specific algorithms for extracting patterns, which then will be interpreted and evaluated to produce knowledge, from data [Fayyad, Piatetsky-Shapiro, and Smyth 1996]. Main aspect of this Master's Thesis Project is the data mining itself, not the whole KDD process. Therefore, data mining will be examined in more detail then the overall KDD process.

In addition to data mining step, KDD process also has data selection, preprocessing, transformation and interpretation steps as shown in **Figure 2.1** [Fayyad, Piatetsky-Shapiro, and Smyth 1996]. Composition of these steps constitutes the KDD process. In order to understand what data mining is and address where it stands, overall KDD process will be presented shortly. In **Figure 2.1**, overall representation of KDD process, which also includes data mining step, can be seen.

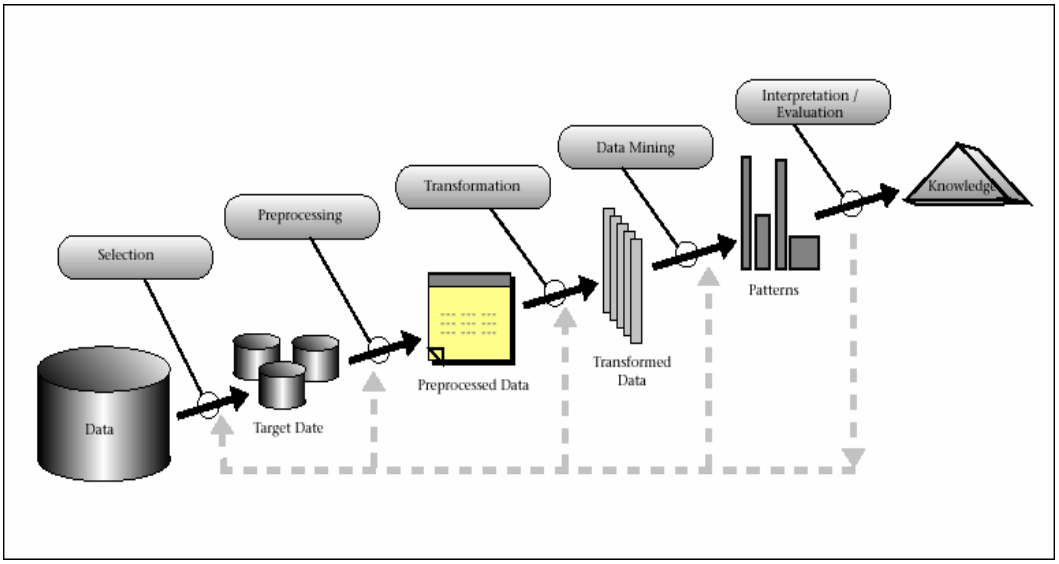

Figure 2.1 Overall Representation of KDD Process

KDD process consists of Data Selection, Preprocessing, Transformation, Data Mining and Interpretation / Evaluation steps. First step of KDD process is to create a target dataset on which KDD will be performed. This is to select an application dataset or a subset of it. Second step of KDD is the cleaning and preprocessing. Basically, removal of noise in data is performed and strategies for handling missing data fields are developed in this step. As a third step, transformation of preprocessed data is performed. This is to find useful features of data which defines the data according to the needs of data mining algorithm. Next step is the data mining itself, which uses transformed data and produces patterns and relationships. Final one is the interpretation and evaluation step which is to comment on mined patterns in order to develop knowledge. In this step, return back to each of other steps may be performed for further iterations.

Simply put, as a result of KDD process, knowledge is produced which helps people make better business decisions [Ganti, Gehrke and Ramakrishnan 1999]. Data mining is a step in KDD which is used to discover hidden patterns and relationships in data. Data mining is a process that uses a variety of data analysis tools to discover patterns and relationships in data that may be used to make valid predictions. As shown in the **Figure 2.1** [Fayyad, Piatetsky-Shapiro, and Smyth 1996], data mining produces patterns, not the knowledge itself. Knowledge can then be produced by interpreting and evaluating these patterns, which are produced by data mining algorithms.

2.1 Data Mining

As mentioned above, data mining is the task of discovering hidden patterns and relationships in databases which are prior to knowledge production. In companies' large databases, there are lots of hidden patterns of strategic importance [Ganti, Gehrke and Ramakrishnan 1999]. Data mining is the only method of digging these databases and finding these valuable patterns. Without data mining, it is impossible to examine such large databases and produce valuable information.

Data mining is very critical for companies in order to produce strategic information by using their historical data. By using data mining, companies can control their costs and

increase revenue [Palace 1996]. Currently, data mining is being used in wide variety of business areas for lots of purposes. Most organizations use data mining in order to manage their customer life cycle such as acquiring new customers, increasing revenue of existing ones and retaining good customers [Edelstein]. When a company defines the characteristics of its customers by using historical data, it can predict the future behaviors of existing and candidate ones, so that it can develop required strategies. And also, a lot of organizations, such as telecommunication, credit card and insurance companies, use data mining in order to detect and reduce fraudulent use of their services [Hengl 2003]. Besides, financial companies use data mining to determine market and industry characteristics and retailers use it in order to decide which product to stock in their stores.

Data mining is strongly related and supported with some other data processing and statistical works. Most important ones of these efforts are mentioned shortly in the following parts in relation with data mining.

2.1.1 Data Warehousing

Success of data mining is strongly related with data warehousing functions of companies. Data mining uses pre-processed data which are supplied by data warehouses. Companies' data warehousing departments develop functions which collect valuable data from business activities continuously. Collected and cleansed data are then stored in data warehouses in order to be used in statistical works. Data mining is the one of the most important ones of those statistical works which uses the data stored in data warehouses.

2.1.2 OLAP (On Line Analytical Processing)

One of the most confusing points in data processing is the difference between data mining and OLAP (On Line Analytical Processing). Although they seem to be similar, they are very different techniques of data processing and they are complements of each other [Two Crows Corporation 1999].

OLAP is a part in evolutionary of decision support tools. OLAP goes further than traditionally querying and reporting of data to search what is in a database. OLAP is used to verify whether certain things are true or not. User of the OLAP tool makes a hypothesis and tries to verify his or her hypothesis by using a series of queries on data [Two Crows Corporation 1999]. If it is not verified by the OLAP tool, user modifies his or her guess and retries to verify it by the tool. In other words, the user makes guesses to find patterns and relationships in data and tries to verify or disprove those guessed patterns and relationships by using OLAP tool. This effort continues until desired patterns and relationships in data are discovered.

OLAP analysis is a deductive process which means that the pattern is found initially and then it is verified by the data. It is clear that when dealing with huge amounts of data that have large number of variables (attributes), it is very hard for users to make valuable hypothesis. This is where importance of data mining arises.

Unlike OLAP, data mining is an inductive process. In data mining, data itself is used to uncover hidden patterns and relationships. Users do not need to make guesses for patterns, because hidden patterns are discovered by data mining algorithms by using data without any interaction of users. Successful data mining algorithms can uncover more valuable patterns in much less time than using OLAP tools especially when the amount of data and number of variables are great.

When it comes to the complementary part of data mining and OLAP, discovered patterns of data mining algorithms can be analyzed by using OLAP tools [Two Crows Corporation 1999]. OLAP is a very useful data processing technique to verify patterns on data before using those patterns to produce knowledge. Companies can test discovered patterns by using OLAP tools in order to find what deficiencies and implications may arise when using those patterns. While OLAP may be used in order to verify the validity of produced patterns, it may also be used to discover possible harms or unexpected results of using even valid patterns.

2.1.3 AI (Artificial Intelligence) and Statistics

Data mining, in fact, is the application of AI and statistical techniques in order to perform a common task. Therefore, advances in both AI and statistics are very beneficial for data mining. Both disciplines make progress on pattern recognition and classification which are also the work are of data mining [Two Crows Corporation 1999]. Data mining does not replace these communities but combines them and goes further for discovering patterns and relationships.

2.2 Types of Data Mining Algorithms

Data mining algorithms are the collection of techniques in order to perform data mining task. Currently, there are a lot of data mining algorithms for a wide range of data mining tasks. Mainly, these algorithms can be categorized into there groups according to the types of patterns which those algorithms try to discover [Apte 1997]. These three types of algorithm are presented in the following parts.

2.2.1 Association Rules Algorithms

Association Rules algorithms (Link Analysis in other words) deal with finding the statistical relations (associations) between two given types of objects that exist in the dataset. In other words, these algorithms find how often events occur together. For example, a statement like “A customer who buys tea from a supermarket will likely buy sugar” is an association. Associations of items in a business unit must be considered carefully in order to develop good strategies.

2.2.2 Classification Algorithms

Classification is assigning the objects in the dataset into a predefined set of classes. Classification is a type of supervised learning, because the set of classes are introduced to the system before executing classification algorithm. Classification of objects in a dataset is very useful both to understand the characteristics of existing objects and to predict the behaviors of new objects. Classification of e-mails, incoming to an e-mail server, into predefined e-mail classes can be an example for this type of data mining. In this way, behaviors of an e-mail server, such as giving priority to e-mails or blocking some ones, can be determined for incoming e-mails.

2.2.3 Clustering Algorithms

Clustering is the grouping of similar objects and a cluster of a set is a partition of its elements that is chosen to minimize some measure of dissimilarity [Kantabutra 1999]. Unlike classification which is a supervised learning technique, clustering is a type of unsupervised learning [Crocker and Keller]. In clustering, objects in the dataset are grouped into clusters, such that groups are very different from each other and the objects in the same group are very similar to each other. In this case, clusters are not predefined which means that result clusters are not known before the execution of clustering algorithm. These clusters are extracted from the dataset by grouping the objects in it. For some algorithms, number of desired clusters is supplied to the algorithm, whereas some others determine the number of groups themselves for the best clustering result. Clustering of a dataset gives information on both the overall dataset and characteristics of objects in it.

As an example for clustering, categorization of documents (books, assays, magazines and etc.) in a document collection can be considered. After grouping these documents, an overall view for the major topics of the collection will be gathered including the number and characteristics of groups of documents in the collection. And also, because documents in each group are similar to each other and represent the features of their group,

an overview of each document will be gathered and access to required document will be much easier.

K-means algorithm, which is the subject of this project, is a type of clustering algorithm. Classical K-means algorithm will be examined and compared with the newly designed parallelized K-means algorithm in the scope of project. Since parallelization is considered, classical K-means algorithm will be referred as serial K-means algorithm and the other one will be referred as parallel K-means algorithm in the following parts. At first, serial K-means will be examined and then the parallel one will be focused.

CHAPTER 3

SERIAL K-MEANS ALGORITHM

K-means is a data mining algorithm which performs clustering. As mentioned previously, clustering is dividing a dataset into a number of groups such that similar items fall into same groups [Kantabutra 1999]. Clustering uses unsupervised learning technique which means that result clusters are not known before the execution of clustering algorithm unlike the case in classification. Some clustering algorithms takes the number of desired clusters as input while some others decide the number of result clusters themselves.

K-means algorithm uses an iterative procedure in order to cluster database [Ali, Ghani and Saeed 2001]. It takes the number of desired clusters and the initial means as inputs and produces final means as output. Mentioned initial and final means are the means of clusters. If the algorithm is required to produce K clusters then there will be K initial means and K final means. In completion, K-means algorithm produces K final means which answers why the name of algorithm is K-means.

After termination of K-means clustering, each object in dataset becomes a member of one cluster. This cluster is determined by searching throughout the means in order to find the cluster with nearest mean to the object. Shortest distanced mean is considered to be the mean of cluster to which examined object belongs.

K-means algorithm tries to group the items in dataset into desired number of clusters. To perform this task it makes some iteration until it converges. After each iteration, calculated means are updated such that they become closer to final means. And finally, the algorithm converges and stops performing iterations.

Expected convergence of K-means algorithm is illustrated in the image below. Algorithm converges in three iterations in the illustrated example. Blue points represent the initial means which may be gathered randomly. Purple points stands for the intermediate means. Finally, red points represent the final means which are also the results of K-means clustering. As presented in the illustration, means move to the cluster centroids by each

iteration of K-means algorithm. When they reach to the cluster centroids, the algorithm converges.

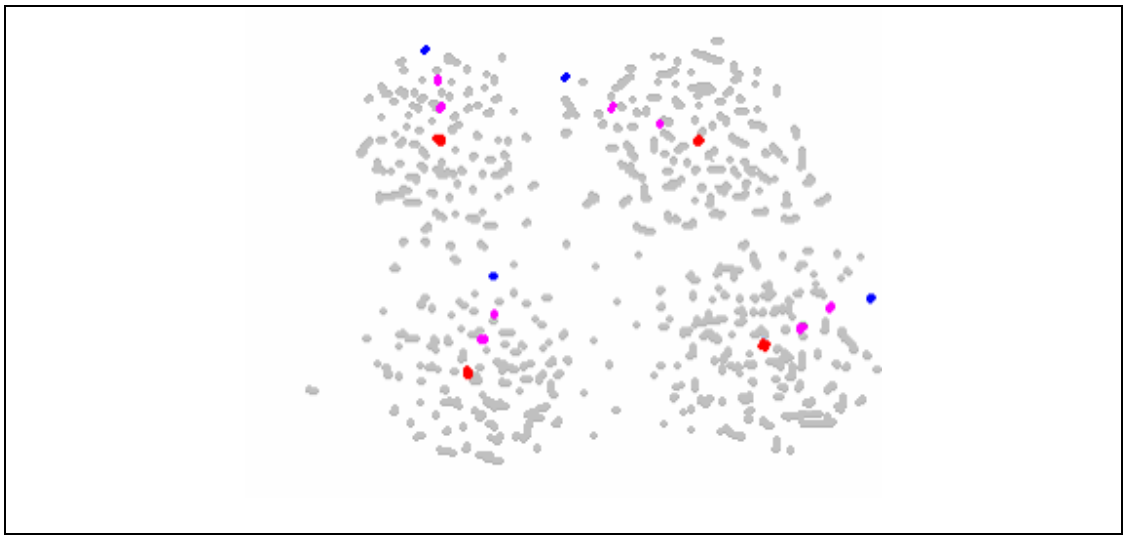

Figure 3.1 Illustration for the Convergence of K-means Algorithm

In following parts, subjects on K-means, steps of it and convergence condition of the algorithm will be discussed.

3.1 Measurement of Distance Between Objects and Means

Different techniques can be used in K-means clustering in order to measure the distance between objects and means. Most popular two distant metrics are Manhattan Distance and Euclidean Distance.

Manhattan distance is the simplest one of those metrics. This metric is the absolute value of difference between object and the mean. Euclidean distance is the square root of addition of squared differences between corresponding dimensions of object and the mean. Since Euclidean distance is the most common distance metric, especially when dealing with multi-dimensional data, Euclidean distance is used for K-means clustering in this project as a distance metric.

3.2 Selection of Initial Means

Selecting of initial means is up to the developer of clustering system [Bradley and Fayyad 1998]. This selection is independent of K-means clustering, because these means are inputs of K-means algorithm. Some developers prefer to select initial means randomly from dataset while some others prefer to produce initial points randomly.

It is known that selection of initial means affects the execution time and success of K-means algorithm. Some strategies are developed to gather better results considering the initial means. The simplest of these strategies is to execute K-means algorithm with different sets of initial means and then select the best results. But this strategy is hardly feasible especially for serial K-means when dataset is large.

Another strategy to gather better clustering results is to refine initial points [Bradley and Fayyad 1998]. If it is possible to begin K-means algorithm with initial means which are closer to final means, it is strongly possible that number of iterations that the algorithm needs to converge will decrease which also lessens the required time for conversion and increase the accuracy of final means.

There are different ways of evaluating clustering results, which will be discussed later, in order to select best results. Developers of clustering systems need to decide on which criteria to use in order to select the best clustering results.

3.3 Steps of K-means Algorithm

As stated earlier, K-means algorithm takes initial means as input. Then it iterates and updates the means in each iteration. Each of updates to means in iterations makes those means closer to final means. This is why K-means algorithm converges after a number of iterations.

Initial means and produced subsequent means are used to assign objects into clusters. Initially, objects are assigned into clusters that have the nearest mean to them by using initial means which are supplied to the algorithm as input. This is the first iteration of algorithm. When all objects are assigned into clusters, cluster means are recalculated by

using the objects in the clusters. These means are supposed to be closer to final means when compared with initial ones. Next, all objects are reassigned to clusters by using new means. This is the conclusion of second iteration. Probably, some objects will move to different clusters when using new means considering their clusters with the previous means. These iterations of K-means algorithm continues until no object moves to another cluster between to iterations. This is the convergence of the algorithm. Since means get closer to final means, which has the minimum distortion compared with other means, by each iteration, it is certain that K-means algorithm will converge after a number of iteration.

Steps of K-means algorithm are seen below in **Figure 3.2**,

- Calculate initial means
- Assign objects into clusters by using initial means
- Do while objects move to another clusters
    - Recalculate means of clusters by using objects belonging to them
    - Assign objects into clusters by using calculated means
- End of while (Convergence of the algorithm)

Figure 3.2 Steps of Classical K-means Algorithm

It is certain that steps of K-means algorithm are very simple and straightforward, but the task performed is great as a result of the iterative structure of algorithm. Simplicity, execution in linear time and the ability of successful clustering makes the K-means algorithm one of the most common clustering algorithms all over the world. A schematic version of steps of Serial K-means algorithm is also presented in **Figure 3.3**.

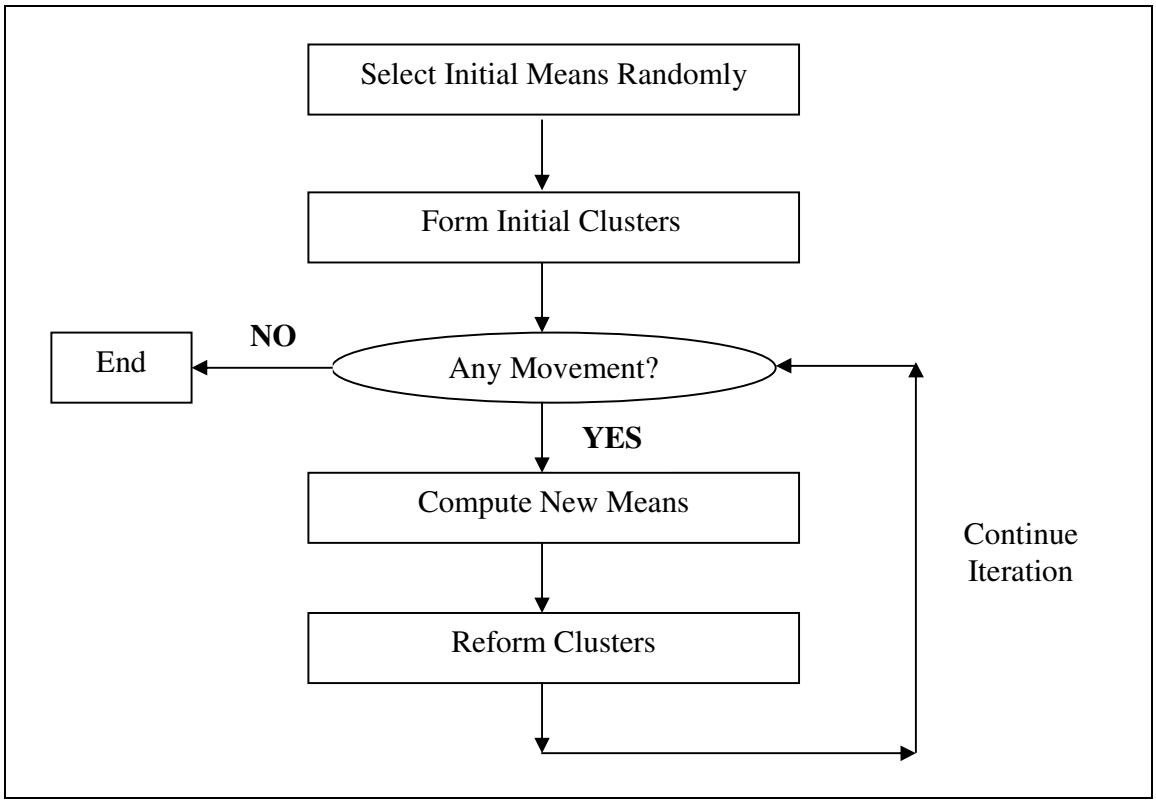

Figure 3.3 Steps of K-means Algorithm in Schematic Representation

3.4 An Example for K-means Convergence

In this section, an example representative small dataset will be clustered in order to simulate the iterations and convergence of K-means algorithm. This dataset consists of eight integer valued instances with only one dimension. Desired number of clusters is two.

States of the process with the dataset in each stage of K-means clustering are presented in the figures below. The computer, executing the K-means algorithm, is referred as process in these figures, because in the following parts of the project, parallelized version of the algorithm will be introduced and in that case there will more than one processes.

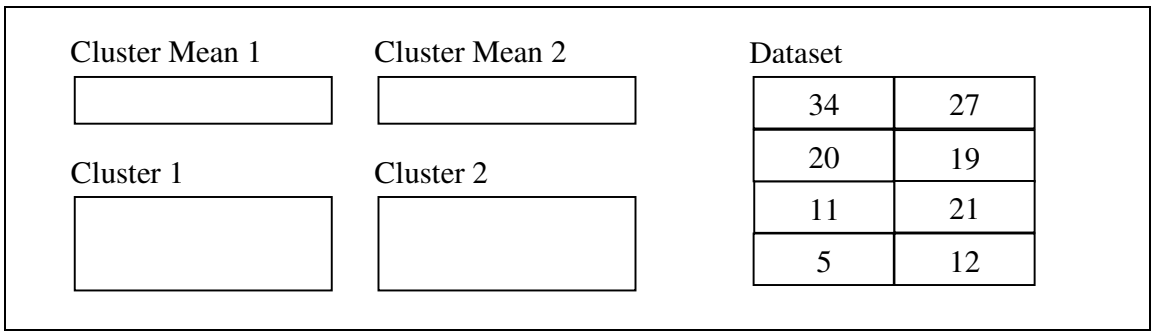

Figure 3.4 Initial State of the Process in Serial K-means

In the initial state, as shown in **Figure 3.4**, cluster means do not exist and clusters of the process is empty. Data in dataset is shown in the right hand side of dataset.

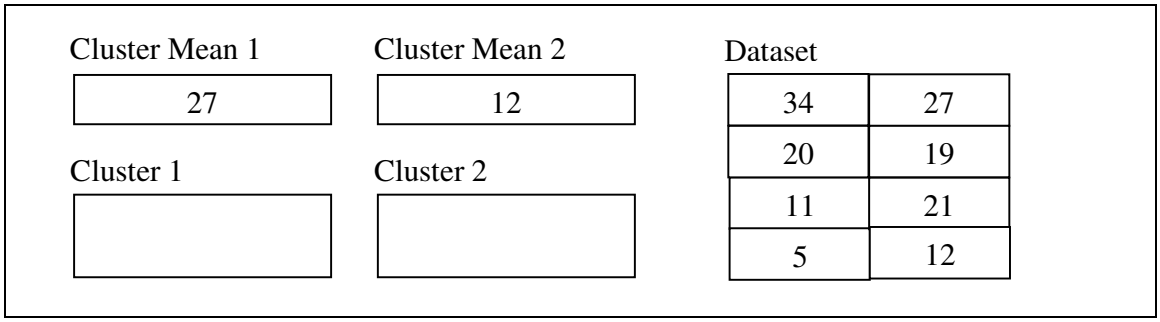

Figure 3.5 Second State of the Process in Serial K-means

In the second state, initial means are selected from the dataset randomly. This selection is up to the designer of K-means clustering. The designer may choose to select initial means randomly from the dataset or produce those means randomly in the range of data. And also, some other one may prefer to produce better initial means which are closer to final means by using some statistical techniques.

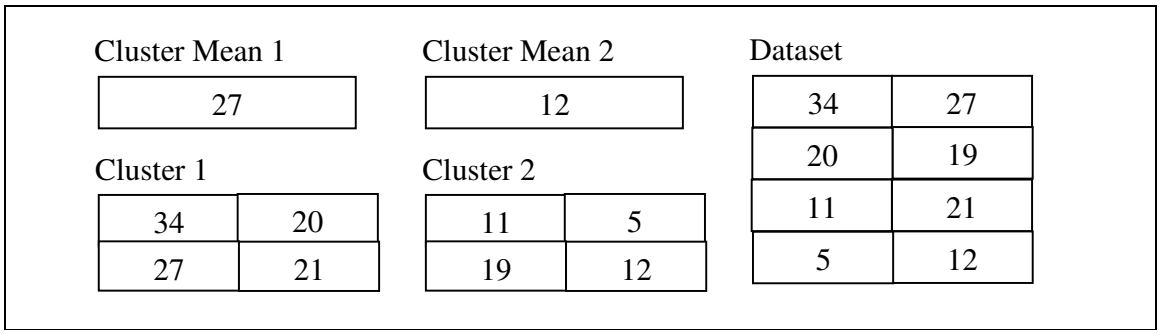

Figure 3.6 Third State of the Process in Serial K-means

In third state, data in dataset are assigned to clusters with the nearest means.

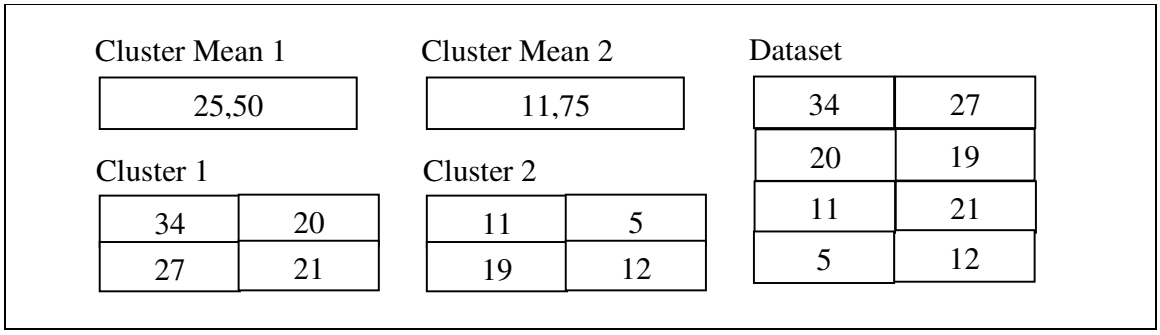

Figure 3.7 Forth State of the Process in Serial K-means

In forth state, means are recalculated using the values of objects in the clusters.

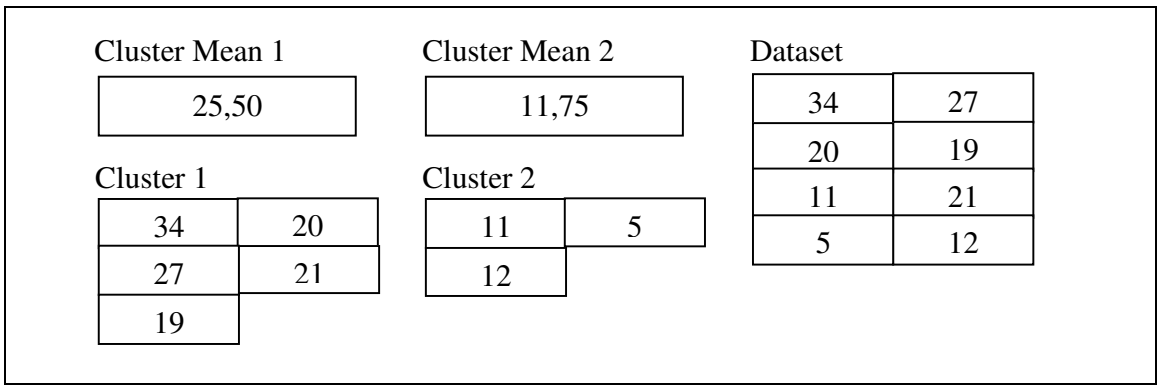

Figure 3.8 Fifth State of the Process in Serial K-means

In fifth state, objects are reassigned into clusters using new means which are equal to 25,50 and 11,75.

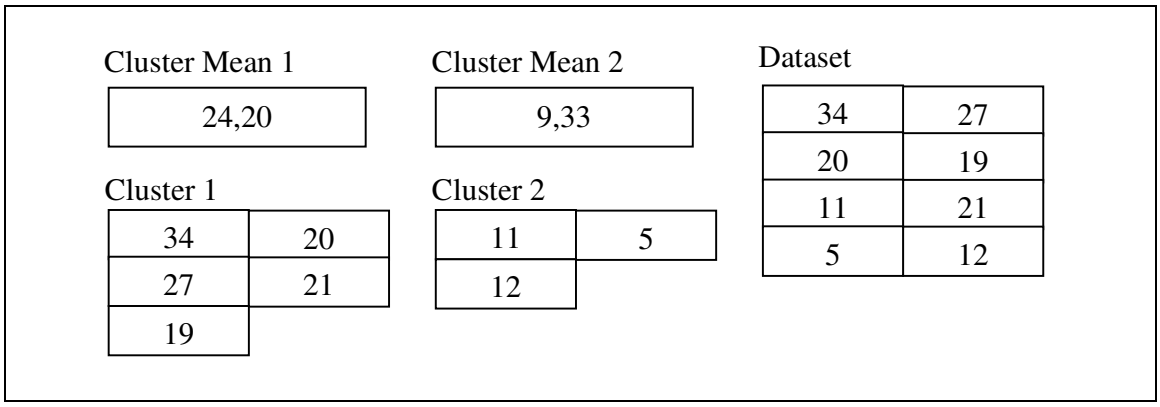

Figure 3.9 Sixth State of the Process in Serial K-means

In sixth state, cluster means are recalculated by using the values of objects in the clusters.

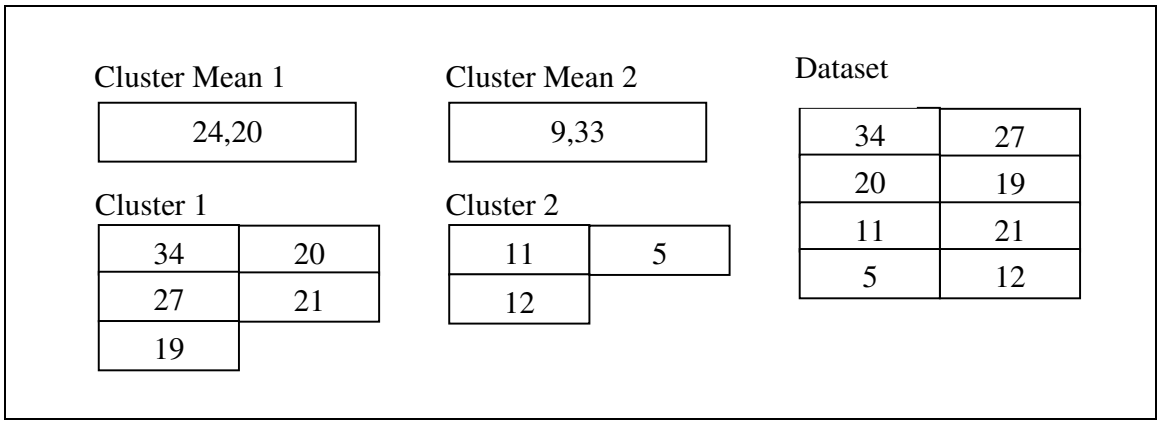

Figure 3.10 Seventh State of the Process in Serial K-means

As it can be seen, seventh and sixth states of the process is the same because no object has moved to another cluster after reassignment of objects by using means 24,20 and 9,33. This condition makes the K-means algorithm to converge and terminate execution. In summary, 24,20 and 9,33 are the result (final) means of K-means clustering algorithm.

3.5 Implementation of K-means Clustering Algorithm

Some decisions have been made before starting to the development of K-means clustering program. One of these decisions is the selection of initial points. Initial means of

K-means have been decided to be generated randomly in the range of data in dataset. Since the purpose of development of K-means program in this project is to compare serial K-means with parallelized K-means in execution times, generating initial means randomly is suitable. It is important here that initial points in both versions must be produced in the same way in order to have the chance of comparing them.

Another decision for the program is the choice of metric for the distances between objects and cluster means. Euclidean Distance has been decided to be used for distance metric, because it is a commonly preferred metric especially for multi dimensional data.

C Programming Language is used for the development of K-means program. C Language has a lot of advantages such as executability on most platforms and fast execution in time. Therefore it is a good choice for performance comparison. And also, the language supports parallel programming by the help of a tool which is called MPI[1] (Message Passing Interface). MPI will also be examined in detail in the following sections.

Data in datasets are read from physical data files. These files are read into memory by using file reading functions of C Language. Data file is read into memory sequentially, because all the data need to be read into memory.

Management of data in memory is performed by using stacks which is a type of link list. Each cluster in memory is represented by a stack and movement of objects in memory from one cluster (stack) to another one is performed by calling “pop” and “push” functions of previous and next stacks. Stacks are preferred to be used instead of arrays, because stacks are dynamic (increases and decreases dynamically by usage of “pop” and “push” functions) unlike arrays which need to be resized when adding or removing an object which decreases the readability of program and brings an overhead for the execution of program.

Code fragments for stack management are presented below. Memory management strategy for a program is very important, because all the business logic in a program relies on the memory management functions which are the background for those logic. A deficiency or an error in memory management functions of a program will certainly cause the program not to work correct.

---

[1] MPI (Message Passing Interface) is a collection of libraries that allow to produce parallel applications by using C Language on network of computers (Ethernet Networks)

At first, a struct type is created in order to define objects (instances) in dataset. A record type, called INSTANCE, is defined as presented blow in **Figure 3.11**. This definition holds both a double typed data and a link to the next instance because of the requirement of linked lists.

```
typedef struct node
{
        double *data;
        struct node *link;
} INSTANCE;
```

Figure 3.11 Type Definition of Instances

"push" and "pop" functions are also presented below in **Figure 3.12** and **Figure 3.13**. These functions are used in order to add and remove items of type INSTANCE into and from stacks. After development of these functions, it is easy to manage stacks (add and remove items).

```
int push(INSTANCE **ptr, double *item)
{
        INSTANCE *p;

        if(empty(*ptr))
        {
                p = malloc(sizeof(INSTANCE));
                if(p != NULL)
                {
                        p->data = item;
                        p->link = NULL;
                        *ptr = p;
                }
                else
                {
                        return 0;
                }
        }
        else
        {
                p = malloc(sizeof(INSTANCE));
                if(p != NULL)
                {
                        p->data = item;
                        p->link = *ptr;
                        *ptr = p;
                }
                else
                {
                        return 0;
                }
        }

        return 1;
}  //int push
```

Figure 3.12 Implementation of “push” Function

As it seen in the code fragment, “push” function at first allocates memory space for the INSTANCE and then connects it to the stack. Code fragment for the “pop” function is presented below.

```
int pop(INSTANCE **ptr, double **item)
{
        INSTANCE *p;

        p = *ptr;
        if(empty(p))
        {
                *item = NULL;
                return 0;
        }
        else
        {
                *item = p->data;
                *ptr = p->link;
                free(p);
                return 1;
        }
}  //int pop
```

Figure 3.13 Implementation of "pop" Function

"pop" function detaches an INSTANCE from the stack with the "last in first out" manner, frees the memory space for that INSTANCE and returns the double typed valued in the INSTANCE by the reference parameter to the caller of function. If the stack is empty, function returns zero as a return value.

Last in first out characteristic of stack does not make any sense for K-means algorithm, because in K-means algorithm all objects (INSTANCE's) are removed from old cluster and added to the new cluster. Sequence of addition and removal of objects is not important for K-means algorithm.

In developed K-means clustering program, all data in dataset is read sequentially into a stack which is then distributed into K clusters at first iteration by the use of initial means. In subsequent iterations, all objects in all stacks (clusters) are popped from the old clusters and pushed to the new clusters by using the intermediate means. A code fragment that performs this task is shown below.

```
get_initial_means(means_array, clus_count, means_dim);
iter_count = 0;
chg_exist = 1;

while (chg_exist == 1)
{
  chg_exist = 0;

  //swap clus_arr_new and clus_arr_old
  clus_arr_tmp = clus_arr_new;
  clus_arr_new = clus_arr_old;
  clus_arr_old = clus_arr_tmp;

  for (k = 0; k < clus_count; k++)
  {
    while (pop(&clus_arr_old[k],&data) == 1)
    {
      new_index = find_new_index(means_array, clus_count, means_dim, data);
      push(&clus_arr_new[new_index], data);
      if (new_index != k)
      {
        chg_exist = 1;
       }
    }
  }

  calculate_means(clus_arr_new, means_array, clus_count, means_dim);
  iter_count++;
}  //while (chg_exist == 1)
```

Figure 3.14 Part of the Program that Performs K-means Clustering

The above code fragment in **Figure 3.14** actually performs the K-means clustering. It uses some functions which are highlighted by bold characters. Above code, in general, produces initial means, pops all items in clusters and pushes them into their new cluster which is the one that has the shortest distanced mean to the item. After reassignment of items, algorithm recalculates means by using items in clusters and continues to iterate. Algorithm terminates (converges) when no movement of items between clusters occurs.

As it can be seen in previous code fragment, K-means clustering part of the program uses some functions which are "get_initial_means", "find_new_index" and "calculate_means". In this part, these functions will be presented and examined shortly.

```
void get_initial_means1(double *p_means_arr, int p_clus_count, int p_dim_count)
{
   int k,i;
   double divident;

   //Select means randomly
   for (k = 0; k < p_clus_count; k++)
   {
      for (i = 0; i < p_dim_count; i++)
      {
            divident = 0;
            while (divident == 0)
            {
                  divident = rand() % 1000;
            }

            p_means_arr[p_dim_count * k + i] = 1 / divident;
      }
   }

} //get_initial_means1
```

Figure 3.15 Generation of Initial Means for the First Dataset

"get_initial_means1", presented in **Figure 3.15**, is the function that produces random initial cluster centroids for the first dataset. Points, in the range of first dataset, are generated by the presented function. It first generates a random number between 0 and 999. Then, it divides 1 by the generated number and produces a result between 0 and 1 which is mostly closer to 0. This method is used in order to generate each dimension of each initial point.

```
Void get_initial_means2(double *p_means_arr, int p_clus_count, int
p_dim_count)
{
   int k,i;

   //Select means randomly
   for (k = 0; k < p_clus_count; k++)
   {
      for (i = 0; i < p_dim_count; i++)
      {
            p_means_arr[p_dim_count * k + i] = rand() % 10;
```

```
        }
    }
} //get_initial_means
```

Figure 3.16 Generation of Initial Means for the Second Dataset

"get_initial_means2", presented in **Figure 3.16**, is the function that produces random initial cluster centroids for the second dataset. It produces points in the range of second dataset. Simply, an integer number between 0 and 9 is generated for each dimension of each initial point.

```
int find_new_index(double *p_means, int p_clus, int p_col, double *p_data)
{
        double min_diff;
        double diff;
        int k;
        int result;

        for (k = 0; k < p_clus; k++)
        {
                diff = calc_euclid_diff(p_means, p_col, p_data, k);
                if (k == 0)
                {
                        min_diff = diff;
                        result = k;
                }
                else
                {
                        if (diff < min_diff)
                        {
                                min_diff = diff;
                                result = k;
                        }
                }
        }

        return result;
}
```

Figure 3.17 Function that Finds the New Cluster of an Object

Above function in **Figure 3.17** is used in order to find the new cluster of an object in dataset. It calculates the Euclidean Difference of the object to each cluster and returns the

index of the cluster with the minimum difference. Differences are calculated by the "calc_euclid_diff" function which is highlighted in the code fragment. This function is presented below.

```
double calc_euclid_diff(double *p_means, int p_col, double *p_data,int p_cur_row)
{
        int k;
        double result;

        result = 0;
        for (k = 0; k < p_col; k++)
        {
           result += pow(p_data[k + 1] - p_means[p_col * p_cur_row + k], 2);
        }
        result = sqrt(result);

        return result;
}
```

Figure 3.18 Function that Calculates Means in Each Iteration

Above function in **Figure 3.18** calculates the Euclidean Difference between an object and a cluster centroid. It sums the squares of differences of each dimension and finally performs a square root operation to calculate the result difference.

```
void calculate_means(INSTANCE **p, double *p_means, int p_clus, int p_dim)
{
            int k, i;
            int count;
            INSTANCE *ptr;

            for (k = 0; k < p_clus; k++)
            {
                    ptr = p[k];
                    count = 0;

                    //First, calculate totals of current cluster
                    while(ptr != NULL)
                    {
                            for (i = 0; i < p_dim; i++)
                            {
```

```
                                p_means[p_dim * k + i] += ptr->data[i + 1];
                        }
                        count++;
                        ptr = ptr->link;
                }

                //Second, Calculate means of current cluster
                if (count > 0)
                {
                        for (i = 0; i < p_dim; i++)
                        {
                                p_means[p_dim * k + i] /= count;
                        }
                }
        }
} //void calculate_means
```

Figure 3.19 Function to Calculate the Means in Each Iteration of K-means Algorithm

Function in **Figure 3.19** calculates the means of all clusters in dataset. It performs add operation on corresponding dimensions of objects in a cluster and finally divides the totals by the count of objects in that cluster in order to produce cluster means. It iterates for each cluster in order to calculate means of all clusters.

3.6 Deficiencies of Serial K-means Algorithm

In today's world, company's databases have grown explosively which makes it very time consuming or sometimes impossible to run traditional data mining algorithms on their data bases. Therefore, serial K-means algorithm will either lack in performance or crash when trying to cluster huge amounts of databases which arises the need of improvement of K-means algorithm in order for the K-means algorithm to cluster large amounts of data in reasonable times.

An ideal data mining algorithm should scale well. In other words, the algorithm should produce true results in reasonable time even the database grows to very large amounts. Therefore, some modifications should be done to traditional data mining algorithms in order to make them scale well in case of huge amounts of data.

The most popular and well working technique of increasing computation power is parallel processing. Therefore, in this project, parallel processing is used in order to improve K-means algorithm for clustering large databases in shorter times. Parallelization techniques and developed parallelized version of K-means algorithm are presented in the following sections in detail.

CHAPTER 4

PARALLELIZATION OF ALGORITHMS

Theory of parallelisation is mainly breaking large tasks into smaller ones, assigning these smaller tasks to multiple working units and finally managing these multiple working units [MHPCC 1999]. When efficient coordinating (well parallelism) of working units is present, tasks can be completed in much shorter durations. Parallelisation of tasks is very common in real life such as building construction.

Parallelisation is also applicable for algorithms. In the scope of this master thesis project, serial version of K-means algorithm is implemented and the parallel version of the algorithm is designed and implemented in C Language. Development of the parallel version of the algorithm is to improve the algorithm in order to scale well for large volumes of data and produce correct results in reasonable time.

As parallelisation is a strategy for performing large and time consuming tasks faster, when parallel computing is considered, it is obvious that process power will be increased and the process time will be decreased with the provision that design and implementation of parallelization is done properly [MHPCC 1999]. And also, properties of algorithms, which are to be parallelized, are very important for parallelization task. Some algorithms are very suitable for parallelization but some of them are not.

In the best case, by parallelization of an algorithm, a performance gain by the proportion of number of parallel computers can be observed. Let's say, if a highly suitable algorithm for parallelization is properly designed and implemented in parallel, it won't be surprising that the parallel version will complete execution "N" times faster than the serial version when "N" computers are used.

Properties of parallel programming environment are also very important for the success of parallelization. Parallel programming tool (the tool which distributes processes into physical processors and manages the synchronization of them) and connection speed of processors (computers in our case) with each other are very important ones of these

properties. When poor connections are available between computers which are to operate synchronously to perform a common task, it may be very difficult to have a performance gain, because physical communication time is much longer than the computation time of computers. But even in these cases, performance gain may be observed for the large volumes of data if the process communication of the parallel algorithm is not so much frequent.

Small volumes of process communication for a parallel algorithm can be present only when the algorithm is parallelized properly. In order for a parallel algorithm to be a well designed one, processing load must be divided equally between processors and the communication of processors must be lessen as much as possible. Because, when processing load is not divided equally between processors, processors with light weight jobs will be idle and the total processing time will be determined by the processors which lasts longer as they have heavy weight jobs. And also, if connections of computers are poor, a considerable loss of time for communication will arise which affects the success of parallel algorithm.

Parallel programming is mentioned above as a technique which allows algorithms to perform N times faster in best case when N computers are used. Besides, performance gain and the importance of parallelization may be greater when the data to be used by algorithm is as large that it can not fit into the memory of a single machine. In this case, single machine, which executes serial algorithm, will either crash or continue execution by using swap space (physical disk space) which will cause the execution last very long time. However, in parallel program side, data will be divided by the number of parallel computers. In other words, each computer will hold a small part of the whole dataset. In this way, it is much more probable that the data will fit in to real memory.

Generally, it is easier to find a network of computers than to find a computer with a huge size of memory. This issue of data and memory sizes is another important benefit of parallel computing. But, this situation should not make the sense that a parallel program may complete the task even faster than N times when N is the number of computers. In order to make a healthy benchmark, serial program should also be able to hold all of the data in its memory. In this case, parallel version will be N times faster in the best case.

4.1 Parallel Programming Architectures

Architecture taxonomy according to the data in relation with program instructions will be examined in this part. There are four types of parallel programming architecture as listed below [Beddo 2002].

- *SISD (Single Instruction, Single Data):* Single instruction deals with a single dataset in this case. When memory access is slow, this architecture causes bottlenecks.
- *SIMD (Single Instruction, Multiple Data):* In this case, single instruction deals with multiple data which means that target data of the processors are distinct which solves the bottleneck on accessing the memory.
- *MISD (Multiple Instruction, Single Data):* Application area of multiple instructions dealing with single data is not present currently.
- *MIMD (Multiple Instruction, Multiple Data):* In this case, each process executes different instructions and deals with different sets of data. This types of parallel programs are more complex than the others to design and implement.

4.2 Parallel Programming Models

In this part, parallel programming models according to the memory environments will be discussed. There are three types of programming models according to the memory access which are listed below [Beddo 2002].

- *Shared Memory Model:* In this model, all processes share a single memory space. Processes can be coordinated by changing data items in the shared memory. As memory area is shared in this model, there is a risk of producing error prone programs such as deadlocks.
- *Virtual Shared Memory Model:* This model is very similar to shared memory model, except that memory is not physically shared but the processes access the memory as it is shared physically by some interfaces. Same problems of shared memory model exist in this model also.
- *Distributed Memory Model:* In this case, there is no shared memory are. Each process has its own memory area distinct from the others. This increases the memory access speed. As there is no shared communicating area for processes, they communicate each other by message passing. Deficiency of this model is the time

overhead of message passing. In order to achieve parallelization successfully, message sizes should be minimized.

4.3 Parallel Programming with MPI

Before designing the parallel algorithm, a parallel programming technique should be selected in order to design the parallel algorithm according to that technique. In this project, a tool called LAM-MPI (Local Area Multicomputer – Massage Passing Interface) which uses MPI (Message Passing Interface) standard will be used as the parallel programming tool and technique[2]. LAM-MPI is selected as the parallel programming tool because of its widely acceptance in the world and availability of platforms to execute LAM-MPI programs. In order to execute LAM-MPI programs, an Ethernet network with Linux operating systems is enough. And also the LAM-MPI utility must be installed to the computers.

LAM-MPI utility is a collection of runtime components and libraries which helps writing parallel programs in C Programming Language without dealing with networking tasks and scheduling of processors. Written C program by the programmer must only deal with parallelization of the business logic to be performed. Programmer deals with processes (not the processors) which of each perform the same task on different data and uses massage passing commands to communicate the processes. Scheduling of processes to computers in the network and communicating the computers in the network with each other is the job of LAM-MPI utility.

LAM-MPI uses master slave SIMD (Single Instruction Multiple Data) parallel programming architecture distributed memory model. A process is accepted as master (root) process and the others are accepted as the slave processes. Master process may perform some initial and final tasks, such as getting inputs and giving outputs, and synchronize slave processes. Same instruction (program) is executed on all processes which of each uses different dataset.

---

[2] MPI library has been downloaded from the “http://www.lam-mpi.org/” website.

CHAPTER 5

PARALLEL K-MEANS ALGORITHM

As mentioned in previous chapters, parallel K-means algorithm has been developed and implemented in this project by the aim of performance increase when compared with serial K-means. Parallelization of K-means algorithm has been proposed to be a solution for the need of a faster K-means algorithm in order to cluster large amounts of databases in reasonable durations. And also, by using parallel K-means, it has been aimed to gather exactly same clustering results with serial algorithm, since purpose of parallelization in this project is to perform exactly same clustering in shorter duration.

K-means algorithm has been re-designed to run in parallel manner by using the message passing technique of parallelization. LAM-MPI (Local Area Multicomputer – Message Passing Interface) utility has been used in order to implement the parallelized version of K-means algorithm. This utility provides development of parallel programs which are to be executed on network of computers (ethernet networks). LAM-MPI utility has been preferred to be used for the development of message passing based parallel algorithm because of widely acceptance of the utility.

In order to be able to make a valid comparison of execution times between the serial and parallel algorithms, both algorithms have been implemented in the same way except the manner of execution (serial or parallel). As in serial algorithm, Euclidean Distance has been used for distance measurement. And also, selection of initial means has been performed by random number generation in the range of dataset as it had been performed in the serial algorithm.

In following sections, steps and implementation of parallel K-means algorithm will be presented after introducing the algorithm as an overview. Then, expectations from the parallelized version will be discussed. Finally, properties of the Ethernet network, used for the execution of parallel program, will be presented and discussed.

5.1 Overvie of the Parallel K-means Algorithm

When considering the serial K-means algorithm, it is observed that a single process calculates the means of clusters and tries to assign all the objects in dataset into clusters in the iteration of algorithm. Especially, reassignment of all objects into clusters is a very time consuming task for a single machine. Because, the processor has to calculate the Euclidean distances of an object; which includes subtraction, addition, square and square-root operations; to the cluster centroids in order to find the nearest cluster. When number of objects in the dataset becomes huge, time needed to assign objects into clusters goes beyond the reasonable durations for a single processor.

Since objects in the dataset are independent of each other when assigning them into clusters, this part has been thought to be parallelized such that each process deal with an equally divided part of the dataset. Let's say, if there are N processes, each process deals with a subset which is the $1/N^{th}$ of real dataset. In this part of the parallel algorithm, processes do not need to communicate, because each process has cluster means (these means are general for all clusters and will be broadcasted to all clusters by root process) and its own dataset (subset). This part has been parallelized such that, each process performs reassignment of its own subset independently without considering the other parts of the dataset.

As mentioned previously, algorithm also calculates cluster means in all iterations by using the all object in dataset. This task is also very time consuming for a single machine when the dataset is large and therefore considered to be parallelized. In this part of the algorithm, a communication of processes is needed. For the calculation of means of clusters by using all objects in dataset, one computer needs to have information about the overall dataset. This part has been parallelized such that, each process calculates totals and counts of objects in its subset and sends these values (total and count) to root process. Then, root process calculates the cluster centroids by dividing the global total to the global count gathered from all processes. After calculation of cluster means, root process broadcasts these means to all processes, because all processes need global cluster means for the reassignment procedure.

As it can be observed, only two messaging have been used for each iteration of K-means algorithm. One is for the calculation of cluster means (each process sends sub-totals and sub-counts to root process) and the other one is the broadcasting of calculated means to all processes.

5.2 Steps of Parallel K-means Algorithm

Steps of serial K-means algorithm needs some revision in order to run in parallel manner. In designed parallel version, all the dataset does not remain in one computer's memory; instead, each computer reads and holds an equally divided (by the number of computers used in parallel execution) part of the dataset. This is the point why computers need to communicate for performing the clustering operation. Since each computer has its own memory space and there is no shared memory area, they need to communicate by using message passing.

A root computer has been used in parallel algorithm. Root computer is used for the synchronisation of all computers. It broadcasts data to all computers and gathers data from all computers in order to perform K-means clustering.

Root computer is not idle when other computers perform clustering task. It also performs the same clustering operations as slave computers. It performs synchronisation tasks in addition to those clustering operations.

In LAM-MPI programming, algorithms are parallelized by using processes. These processes are distributed into computers in the Ethernet network by LAM-MPI utility. If there are enough computers in the network, each process is distributed into distinct computers, but if there are not enough computers in the network, groups of processes are distributed into computers. Therefore, in LAM-MPI programming, it is better to talk about processes rather than computers. Developer of the program only knows about the processes, distribution of processes into computers in the network is the duty of LAM-MPI.

Steps of parallel K-means algorithm are seen below, in **Figure 5.1**

- Root process calculates initial means
- Root process broadcasts initial means to all processes
- Each process, including the root process, assigns its internal objects into its internal clusters by using initial means
- Do while any object in any process moves to another cluster of the process
    - Each process sends internal sum and count of objects of its clusters to root process
    - Root process calculates new means of clusters by using the partial sum and count values gathered by all processes
    - Root process broadcasts calculated means to all processes
    - Each process, including the root process, reassigns its internal objects into its internal clusters by using calculated new means
- End of while (Convergence of the algorithm)

Figure 5.1 Steps of Parallel K-means Algorithm

As shown above, root process produces initial means at the beginning of the program and broadcasts those initial means to other processes. Then, each process, including the root process, assigns objects in its memory into clusters by using the initial means as cluster centroids. Then, the iterations of K-means algorithm for convergence start. Each process sends partial sums and item counts of its internal clusters to root process. After this operation, root process gathers all sub totals and sub item counts which are enough for the root process in order to calculate new cluster means. After calculation of new cluster means, root process broadcasts these means to other computers. Next, each computer re-assigns its internal objects into its internal clusters. These iterations continue until no change occurs between internal clusters of any computers.

A schematic version of steps of Parallel K-means algorithm is also presented below in **Figure 5.2**.

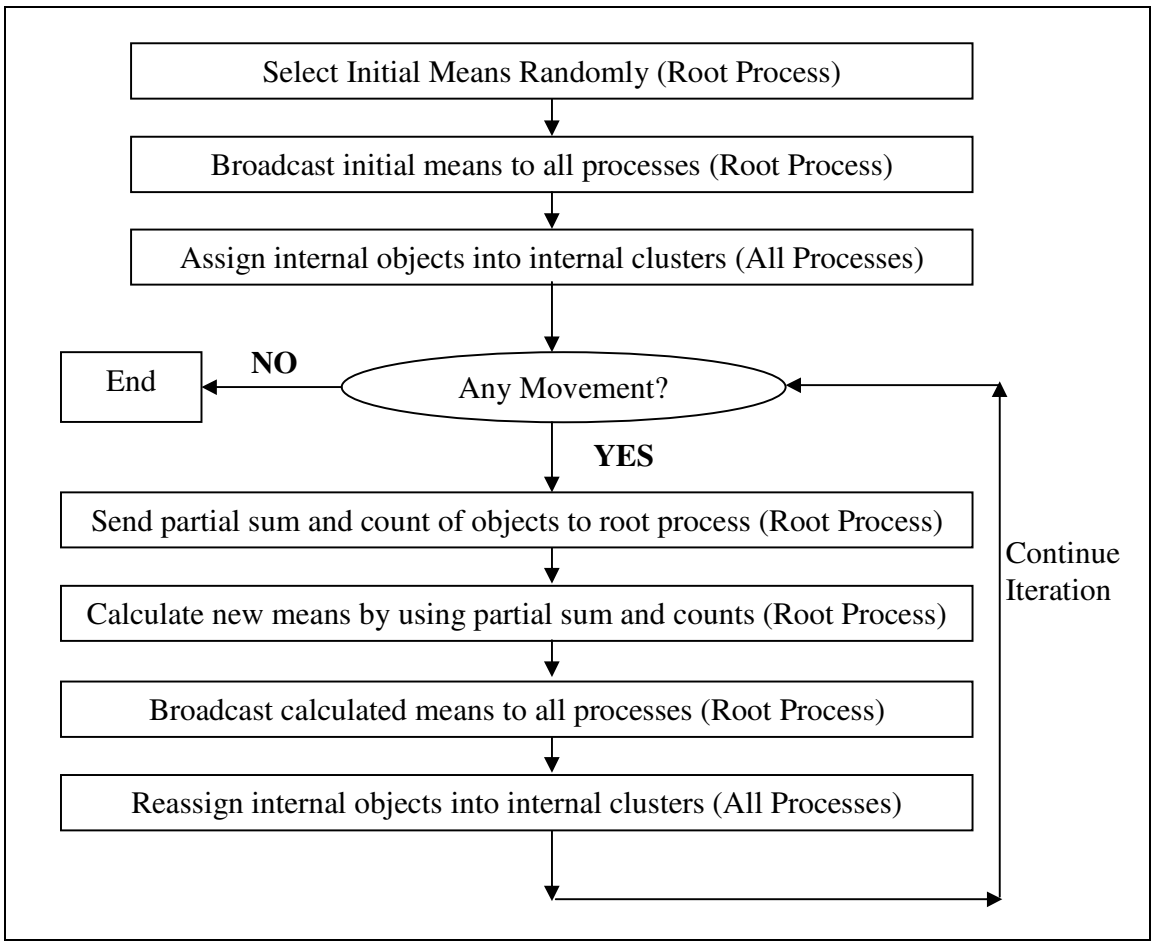

Figure 5.2 Steps of Parallel K-means Algorithm in Schematic Representation

5.3 Example Convergence for Parallel K-means Algorithm

Convergence of parallel K-means algorithm for the representative small dataset used in section 3.4 will be presented in this section. Two processes will be used for illustration purposes. As in serial convergence, desired number of clusters will be 2.

In the previous case (serial algorithm), there was only one process and single memory of it. However, in this case (parallel algorithm), there will be two processes and they will have their own internal memories. Therefore, states of both processes will be displayed in each stage. These stages for parallel K-means convergence are presented

below. Process 1 is determined to be used as root process. Therefore, mean calculation (by using partial sums and counts) and broadcasting of them will be performed by process 1.

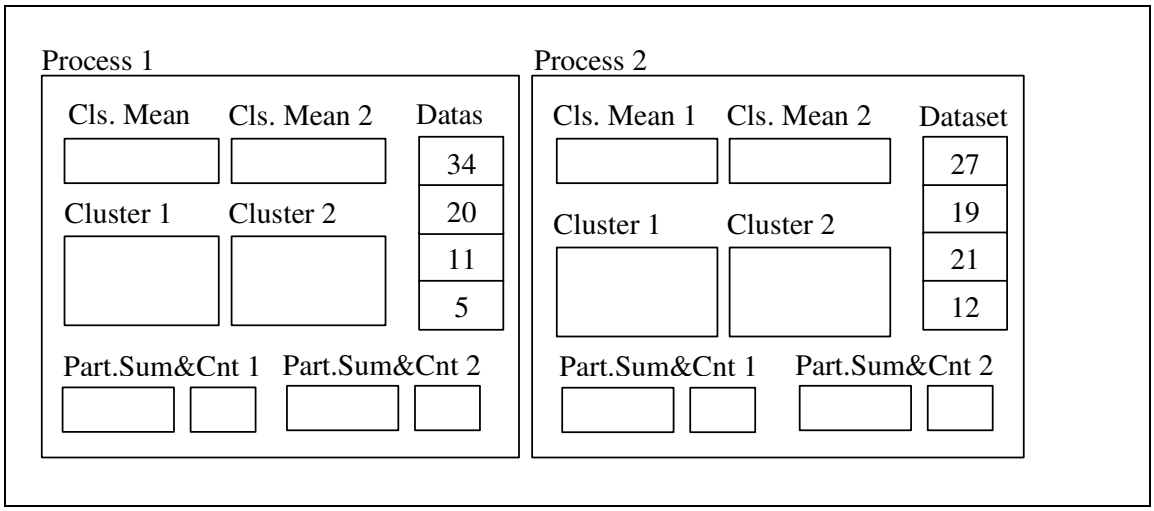

Figure 5.3 Initial State of the Processes in Parallel K-means

In the initial state, in **Figure 5.3**, cluster means do not exist and clusters of the processes are empty. Internal data of processes are shown in the upper right hand side of each process. As mentioned previously, whole dataset is split into parts for each process, and each process has its own internal dataset. In parallel case, each process has also values of partial sums and counts of its internal clusters. These values will be sent to root process (process 1) in each iteration for the purpose of mean calculation.

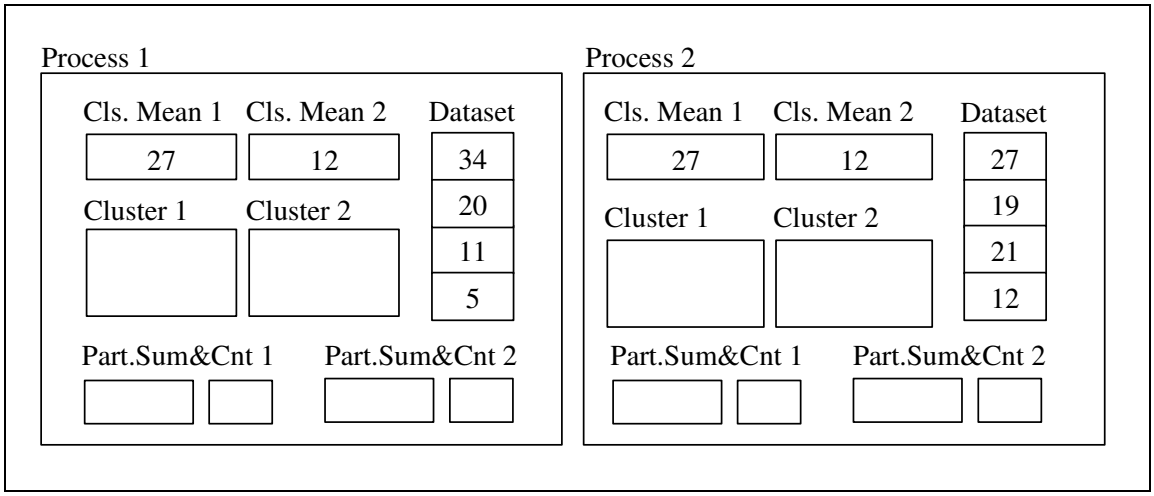

Figure 5.4 Second State of the Processes in Parallel K-means

In second state, root process has generated the initial means and broadcasted them to each process. Therefore, in this state both processes have global initial means in their internal memories.

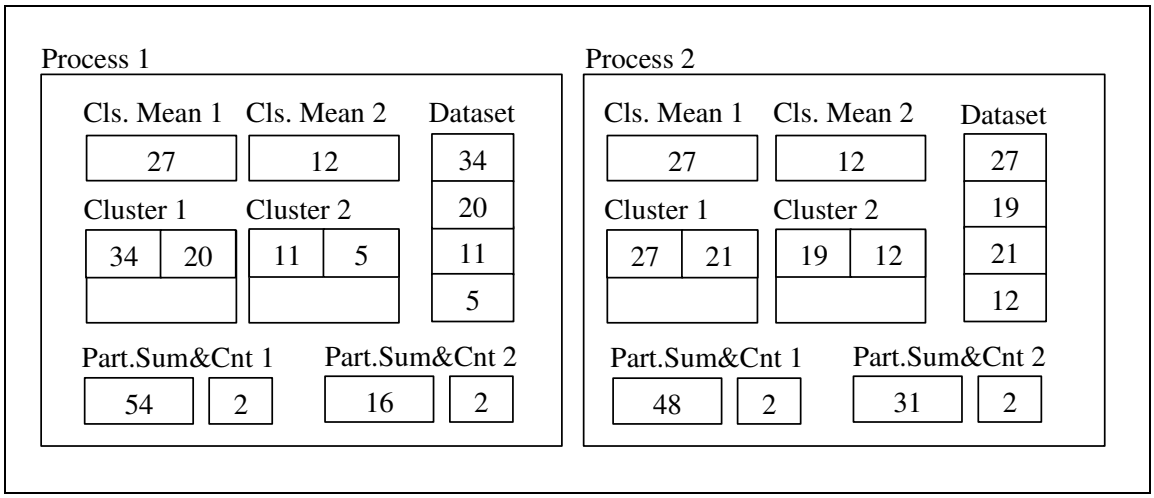

Figure 5.5 Third State of the Processes in Parallel K-means

In third state, both processes have assigned their internal objects into internal clusters and calculated partial sums and counts. These partial sums and counts will be sent to root process (process 1) in next stage in order to help root process to calculate cluster means.

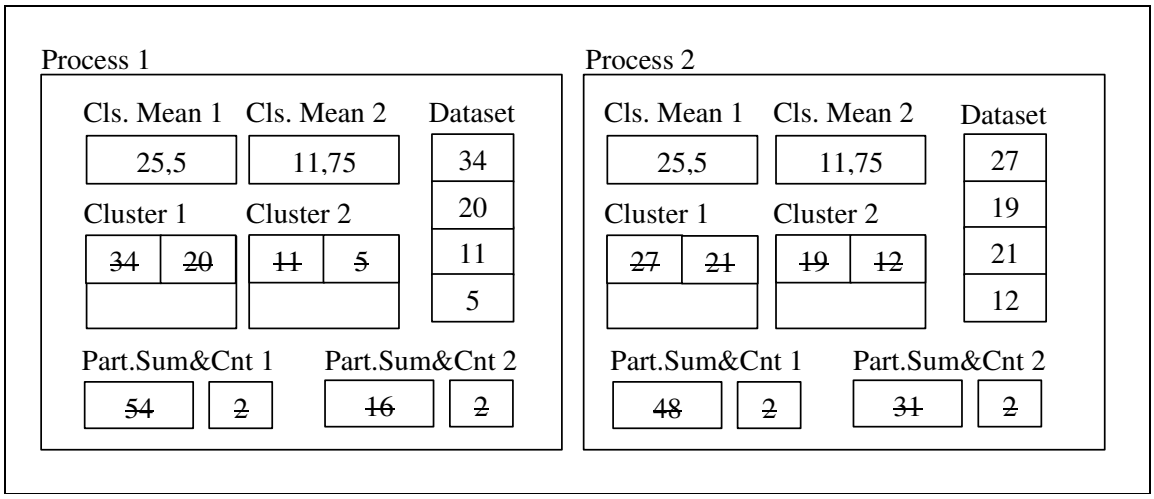

Figure 5.6 Forth State of the Processes in Parallel K-means

In forth state, both processes have sent their partial sums and counts to root process and root process has calculated new means by using these values. When we trace the calculation of root process, root process has gathered the values 54 and 48 for partial sums of cluster1, 2 and 2 for partial counts of cluster1, 16 and 31 for partial sums of cluster2, 2 and 2 for partial counts of cluster2. For the mean of first cluster it calculates the global sum and global count which are 102 (54 + 48) and 4 (2 + 2). And then, divides global sum to global count which makes 25,5 (102 / 4). Mean of the second cluster is also calculated in the same way which makes 11,75 ((16 + 31) / (2 + 2)).

After calculation of new means, root process has broadcasted these means to both processes (including itself). Therefore, in this state both processes have the new means. In this step, after broadcasting of new means, values in internal clusters, partial sums and partial counts are invalid because they had been calculated for previous cluster means. Therefore, they are shown with a line on them. New assignments and calculation for partial values will be performed by using new means in the next stage.

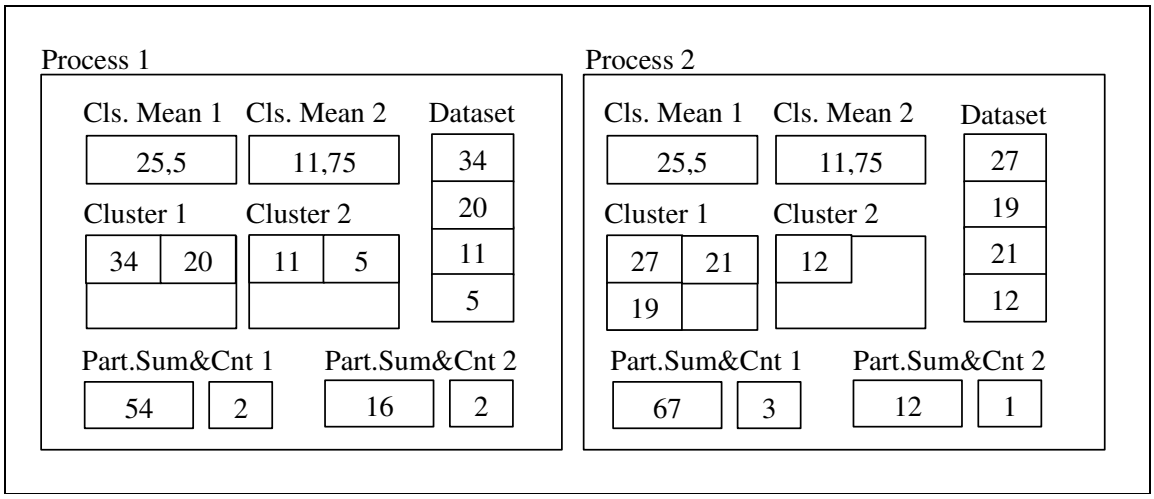

Figure 5.7 Fifth State of the Processes in Parallel K-means

In fifth state, both processes have reassigned objects in their internal dataset into internal clusters. And then, they have calculated the partial sums and partial counts of their internal clusters.

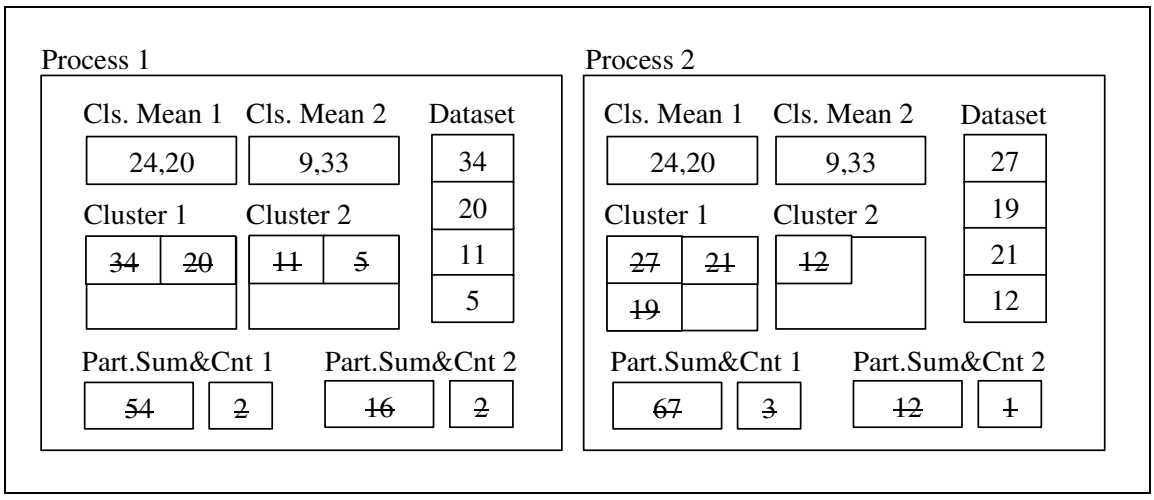

Figure 5.8 Sixth State of the Processes in Parallel K-means

In sixth state, root process has gathered partial values from both processes and calculated new means by using these values (Mean1 = ((54 + 67) / (2 + 3)), Mean2 = ((16 + 12) / (2 + 1))). After calculation of new means, it has broadcasted these values to both processes. Therefore, in this state both processes have new means. In the following stage,

both will reassign objects into internal clusters by using these new means and calculate the partial values.

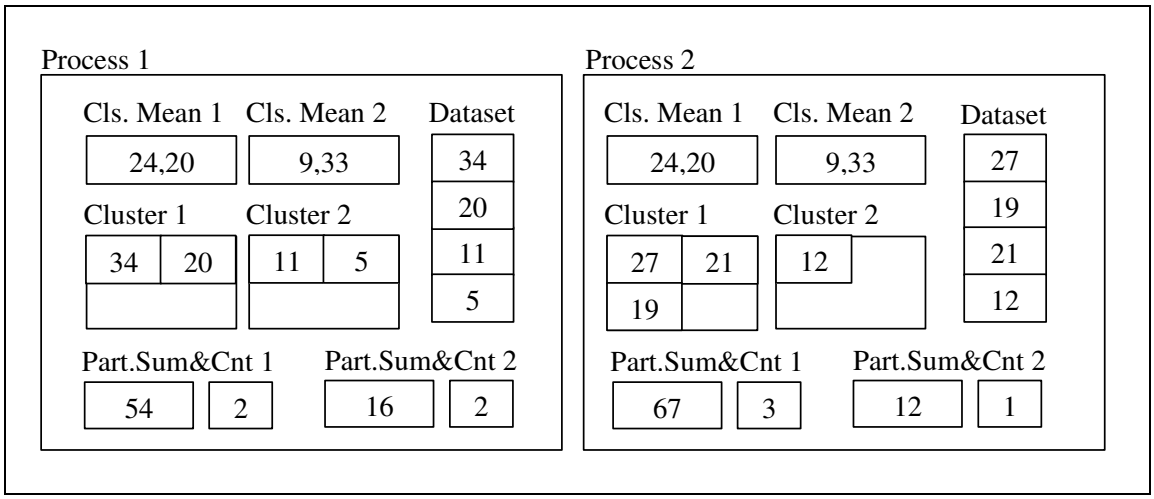

Figure 5.9 Seventh State of the Processes in Parallel K-means

In seventh state, both processes have performed reassignment and calculated partial values which have resulted to a stage which is identical to sixth state. This means that, no object has moved to another cluster when using new means (24,20 and 9,33) which causes the convergence of K-means algorithm. As a result, values of 24,20 and 9,33 are produced as the final means of parallel K-means algorithm which are identical to the results of serial K-means algorithm.

5.4 Implementation of Parallel K-means Algorithm

As discussed previously, parallel K-means algorithm has been implemented in the same way with the serial one except for the manner of execution (serial or parallel). Initial means are generated randomly and Euclidean Distance is used for distance metric as in serial algorithm. Also, C programming language (as in serial one) and LAM-MPI utility has been used for the development of the parallel program.

Data in datasets are read from physical data files by each process. Each process reads an equally divided part of the dataset sequentially into its internal memory. These files are read into memory by using file reading functions of C Language.

Stacks are used for the management of internal memories of processes. Usage of stacks and details of "push" and "pop" functions won't be examined again, because they have been presented in chapter 3. Stacks are used in the same way as the implementation of serial algorithm.

All parts except the ones for the purpose of parallelization are the same in both implementations. Therefore, code fragments related to parallelization and MPI will be presented in this section.

Firstly, MPI specific variables and initializations are seen below in **Figure 5.10**.

```
//MPI variables
int rank, size, dest, tag;

MPI_Init(&argc, &argv);

MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

tag = 2000;
```

Figure 5.10 Definition and Initialization of MPI Variables

"rank", "size", "dest" and "tag" are MPI specific variables. "rank" holds the index of current process in the communication world. "size" gives the number of all processes in that world. "tag" is an integer value which can be used for different communication groups. Since one communication world has been used in this project, each process uses 2000 as the "tag". "dest" is used in order to send a message to the destination.

```
MPI_Barrier(MPI_COMM_WORLD);
```

Figure 5.11 Usage of MPI_Barrier Command

In the code fragment above in **Figure 5.11**, barrier command of the MPI is used in order to block all processes in that command. Each process stops and waits when it reaches

to this point until all processes reach to the point. This command is used after processes read dataset from file. When all processes finish reading data file, time has been taken from system for the measurement of execution time, similar to the serial implementation which also takes system time after file read.

Code fragment in **Figure 5.12** shows the clustering loop (iteration) of the parallel algorithm. Most parts of the code are removed from the fragment except the ones specific to MPI programming in order to focus on MPI parallel programming. Other parts of the program are very similar to serial algorithm which has been examined in detail in chapter 3.

```
//Root process generates initial means
if (rank == 0)
{
      //Generate initial means here
}

//Broadcast initial means
MPI_Bcast(means_array, clus_count * means_dim + 1, MPI_DOUBLE, 0,
           MPI_COMM_WORLD);

while (chg_total > 0)  //Continue iteration
{
   //swap clus_arr_new and clus_arr_old
   clus_arr_tmp = clus_arr_new;
   clus_arr_new = clus_arr_old;
   clus_arr_old = clus_arr_tmp;

   for (k = 0; k < clus_count; k++)
   {
       //Assignment of objects into internal clusters
   }

   //All processes will send total, count and chg info to root
     process with SUM operation
   MPI_Reduce(send_data, recv_data, data_cnt, MPI_DOUBLE,
              MPI_SUM, 0, MPI_COMM_WORLD);

   //Root process will calculate means
   if (rank == 0)
   {
       //Calculation of means
   }

   //Broadcast calculated means
   MPI_Bcast(means_array, clus_count * means_dim + 1,
        MPI_DOUBLE, 0, MPI_COMM_WORLD);
}  //while (chg_total > 0)  //Continue iteration
```

Figure 5.12 MPI Communications in K-means Clustering

As shown above, before entering the clustering loop, root process generates the initial means and broadcasts them to all processes by using “MPI_Bcast” function. This function is executed by all processes including root process. After execution of this function, data stored in root process (initial means in this case) is copied into memories of all processes.

After entering the clustering loop, each process performs assignment of its internal objects into internal clusters and calculates partial sums and counts. And then, all of them, including root process, stores their internal partial sums and counts to the root process by using the SUM option of “MPI_Reduce” function. This function is also executed by all processes. After execution of this function, root process gathers the sum of data in memories of all processes. This data is then used to calculate new means. After calculation of new means, root process broadcasts these means again. This iteration continues until no change of objects occurs between clusters.

5.5 Network Environment for the Test of Parallel Program

Developed parallel program has been tested on the parallel programming laboratory of Computer Engineering Department of Marmara University. This laboratory is a well constructed network of computers with MPI library installed on them. Properties of the laboratory are listed below.

- Number of computers: 11
- Ethernet Speed: 100 Mbit
- CPU frequency of computers: 2.4 Giga Herz
- Memory size of computers: 2 Giga Byte for root computer and 1 Giga Byte for the others
- Linux version installed on computers: RedHat 9.0
- MPI version installed on computers: Lam-MPI 7.0.3-1

5.6 Expectations from the Parallel Algorithm

Aim of the parallelization of K-means algorithm was to improve the algorithm such that it will produce the same results with serial algorithm. As no change has been performed on the K-means algorithm during parallelization, it is expected that the algorithm will produce exactly same clustering results with serial one. This can also be seen by the sample convergences of both the serial algorithm (Section 3.4) and the parallel algorithm (Section 5.3).

In the design of a parallel algorithm, overhead of root computer for synchronisation and the amount of data interchanged between computers should be minimized as possible for the success of parallelization. If root overhead of synchronisation and frequency of data interchange are high and if large amount of data are passed between computers then it won't be surprising that parallelization of the algorithm.

In this project, K-means algorithm has been designed carefully to run in parallel manner such that frequency of messaging and the amount of messages transmits between processes have been lessened as much as possible.

As it can be seen in the steps and implementation of parallel K-means algorithm (Section 5.2 & Section 5.4), parallelized version of the algorithm performs messaging only two times in one iteration (one for mean calculation and one for broadcasting of those means). And also, amount of data transmitted between processes are very low in amount. Only summary data is moved between processes which is about to the size of double multiplied by the count of clusters and dimension of dataset. Let's say, if there are 5 clusters with a 50 dimensional data, size of double multiplied by 250 will be messaged between root process and the other processes for one messaging (two messaging occur in each iteration) which is a very small amount for data transmission in an Ethernet network.

When considering above information, it is expected that parallel algorithm will produce same results with the serial algorithm in a much shorter time. A performance gain is expected almost by the count of computers used.

Also, in the case of very large datasets which do not fit into the memory of a single computer, performance gain would be enormous as the whole dataset is split equally into memories of all computers in the parallelized version.

CHAPTER 6

RESULTS OF SERIAL AND PARALLEL K-MEANS ALGORITHMS

Both serial and parallel versions of the K-means algorithm have been tested on the same platform by using the same datasets. Then, results of them have been collected in order to make a comparison between two versions. Expected result of these executions is that the parallel algorithm will produce exactly the same results with the serial algorithm but in shorter time than the serial one.

6.1 Datasets Used for K-means Clustering

Two different datasets have been used for these executions. First one of those datasets, which is relatively smaller than the second one, is about color histogram information of a pictures collection. Second dataset holds the information about US Census of year 1990 and this dataset is larger than the first one.

Both of these datasets have been gathered from the KDD (Knowledge Discovery in Databases) archive of University of California, Irvine (http://kdd.ics.uci.edu/) [Hettich and Bay 1999]. A lot of datasets are published in the web site of this university and these datasets are available to be used in KDD researches. This repository supplies standard datasets, which are very useful for KDD researchers in order to benchmark their algorithms.

6.1.1 Color Histogram Dataset

This dataset contains image features extracted from an image collection. These features are about color histogram of images in the collection. Dataset contains information

for 68,040 images and it has 32 attributes for each image. In other words this is a dataset with 32 dimensions.

Each line in the dataset file represents an image. First column of the line is row-id and the following 32 columns constitute the color histogram attributes vector. Each of 32 attributes in dataset represents the density of a color in the image. It means that, color space is split into 32 parts, and density of each part in the image is presented in dataset. Because these attributes contains the density of colors in the image, histogram intersection can be used to measure the similarity between images. But this is not in the scope of this project. Only, clustering of the dataset will be performed in this project in order to calculate clustering time.

Each of the attributes in dataset holds a value between 0 and 1 having 6 decimals which means that it contains continuous data. Some statistical values (mean, min value, max value, variance and standard deviation) for each dimension of color histogram dataset are presented in **Table 6.1**.

Table 6.1 Statistical Values for Color Histogram Dataset

Attribute Number	Mean	Min Value	Max Value	Variance	Standart Deviation
1	0.063897	0	0.961688	0.010697	0.103424
2	0.086427	0	0.984752	0.014954	0.122287
3	0.066824	0	0.924375	0.011981	0.109459
4	0.062394	0	0.995938	0.015788	0.125651
5	0.105824	0	1.000104	0.024289	0.155849
6	0.087096	0	0.919587	0.01362	0.116705
7	0.044644	0	0.895882	0.006362	0.079764
8	0.028169	0	0.938856	0.003838	0.061952
9	0.040842	0	0.905319	0.004615	0.067934
10	0.032316	0	0.870524	0.005198	0.072096
11	0.012677	0	0.772502	0.001448	0.038057
12	0.016994	0	0.956147	0.000885	0.029741
13	0.048576	0	0.941949	0.007595	0.087149
14	0.021772	0	0.92969	0.002765	0.052582
15	0.007352	0	0.698658	0.000676	0.026001
16	0.005935	0	0.92978	0.000459	0.021416
17	0.053699	0	0.979688	0.010476	0.102354
18	0.064447	0	0.95605	0.013481	0.116109

19	0.038117	0	0.965208	0.007889	0.08882
20	0.028463	0	0.998438	0.00734	0.085674
21	0.01916	0	0.957836	0.003179	0.056383
22	0.008503	0	0.782815	0.001088	0.032992
23	0.004367	0	0.907708	0.000564	0.023747
24	0.006586	0	0.864583	0.000986	0.031397
25	0.00665	0	0.829167	0.00054	0.023234
26	0.002324	0	0.923958	0.000218	0.014756
27	0.001182	0	0.715625	0.000135	0.011599
28	0.000972	0	0.819063	0.000133	0.011527
29	0.011189	0	0.793854	0.000993	0.031504
30	0.007517	0	0.905005	0.000846	0.02909
31	0.006566	0	0.901771	0.000787	0.028054
32	0.008622	0	0.980729	0.001857	0.043089

6.1.2 US Census Dataset of year 1990

Second dataset contains information about US Census of year 1990. This dataset is a discretized version of US Census row dataset for the purposes of data mining algorithms. A lot of less useful attributes in the original dataset had been dropped by the publisher of the dataset. And also, some continuous attributes had been discretized and some discrete variables which have a large number of possible values had been collapsed, so that they had fever possible values. In summary, dataset had been modified to have discrete values in all attributes.

Dataset contains 1,000,000 records and there are 68 categorical attributes in it. This is not the full census dataset of US but a part of it. 1,000,000 records were thought to be enough for the purposes of this project.

Each line in dataset represents a person and each column of the row holds data about the person. First column of the dataset file is row-id and the other 68 columns make the attributes vector of dataset. Labels which define the meaning of attributes are listed in **Table 6.2**. It is not considered to be necessary to give detailed information about meaning of attributes, as the aim of this work is to compare clustering time of different algorithms not to interpret clustering results of the datasets. It is important here to verify that clustering

results are accurate (clustering is really being performed) and each algorithm produces the same accurate results. Then it comes to the comparison of execution times of different algorithms.

Table 6.2 Attribute Names of US Census Dataset

1	2	3	4	5	6	7	8
Age	Ancstry1	Ancstry1	Avail	Citizen	Class	Depart	Disabl1
9	10	11	12	13	14	15	16
Disabl2	English	Feb55	Fertil	Hispanic	Hour89	Hours	Immigr
17	18	19	20	21	22	23	24
Income1	Income2	Income3	Income4	Income5	Income6	Income7	Income8
25	26	27	28	29	30	31	32
Industry	Korean	Lang1	Looking	Marital	May75880	Means	Military
33	34	35	36	37	38	39	40
Mobility	Mobillim	Occup	Othserv	Perscare	POB	Poverty	Pwgt1
41	42	43	44	45	46	47	48
Ragechld	Rearning	Relat1	Relat2	Remplpar	Riders	Rlabor	Rownchld
49	50	51	52	53	54	55	56
Rpincome	RPOB	Rrelchld	Rspouse	Rvetserv	School	Sept80	Sex
57	58	59	60	61	62	63	64
Subfam1	Subfam2	Tmpabsnt	Travtime	Vietnam	Week89	Work89	Worklwk
65	66	67	68				
WWII	Yearsch	Yearwrk	Yrsserv				

Some statistical values (mean, min value, max value, variance and standard deviation) for each dimension of US Census dataset are presented in **Table 6.3** and **Table 6.4**. As dataset has 68 attributes, table has been divided into two parts of 34 rows.

Table 6.3 First Half of Statistical Values for US Census Dataset

Attribute Number	Mean	Min Value	Max Value	Variance	Standart Deviation
1	3.85	0	7	4.2	2.05
2	3.3	0	11	16.32	4.04
3	1.58	1	12	2.88	1.7
4	0.12	0	4	0.45	0.67
5	0.29	0	4	0.94	0.97
6	1.24	0	9	3.09	1.76
7	1.4	0	5	3.1	1.76
8	1.43	0	2	0.71	0.84
9	1.47	0	2	0.72	0.85

10	0.21	0	4	0.42	0.65
11	0.02	0	1	0.02	0.14
12	1.18	0	13	3.47	1.86
13	0.14	0	9	0.82	0.91
14	1.58	0	5	2.99	1.73
15	1.4	0	5	3.07	1.75
16	0.45	0	10	2.87	1.7
17	0.9	0	4	1.2	1.09
18	0.05	0	1	0.05	0.22
19	0.01	0	1	0.01	0.1
20	0.2	0	1	0.16	0.4
21	0.14	0	1	0.12	0.35
22	0.03	0	1	0.03	0.18
23	0.07	0	1	0.06	0.25
24	0.04	0	1	0.04	0.2
25	3.92	0	12	14.05	3.75
26	0.02	0	1	0.02	0.14
27	1.73	0	2	0.35	0.59
28	0.54	0	2	0.75	0.87
29	1.9	0	4	3.51	1.87
30	0.01	0	1	0.01	0.1
31	0.86	0	12	4.07	2.02
32	2.81	0	4	2.84	1.68
33	1.34	0	2	0.38	0.61
34	1.52	0	2	0.69	0.83

Table 6.4 Second Half of Statistical Values for US Census Dataset

Attribute Number	Mean	Min Value	Max Value	Variance	Standart Deviation
35	1.77	0	8	4.11	2.03
36	0	0	1	0	0.04
37	1.52	0	2	0.69	0.83
38	0.28	0	6	0.94	0.97
39	1.83	0	2	0.19	0.43
40	1.12	0	3	0.51	0.71
41	1.79	0	4	3.5	1.87
42	1.49	0	5	2.39	1.55
43	1.75	0	13	6.81	2.61
44	0.08	0	9	0.54	0.73

45	35.29	0	223	4243.96	65.15
46	0.48	0	8	0.52	0.72
47	2.22	0	6	5.74	2.4
48	0.24	0	1	0.18	0.42
49	1.87	0	5	2.16	1.47
50	16.92	10	52	133.76	11.57
51	0.26	0	1	0.19	0.44
52	2.15	0	6	4.88	2.21
53	0.76	0	11	5.47	2.34
54	1.25	0	3	0.35	0.6
55	0.02	0	1	0.02	0.13
56	0.51	0	1	0.25	0.5
57	0.06	0	3	0.15	0.39
58	0.03	0	3	0.03	0.17
59	0.79	0	3	1.71	1.31
60	1.48	0	6	3.86	1.97
61	0.03	0	1	0.03	0.18
62	0.83	0	2	0.74	0.86
63	1	0	2	0.47	0.68
64	1.1	0	2	0.55	0.74
65	0.04	0	1	0.04	0.19
66	8.45	0	17	16.67	4.08
67	1.82	0	7	4.45	2.11
68	0.14	0	2	0.16	0.4

6.2 Execution Strategy for Testing the Algorithms

In order to compare two different versions (Serial and Parallel) of K-means algorithm, both algorithms have been executed on two different sets of data (Color Histogram Dataset and US Census Dataset) which are mentioned previously. Each execution is repeated for 5 times by using different sets of random initial points in order to have more reliable results. Final results will be gathered by computing the means of those 5 runs all of which use different random initial points. It is obvious that, means of those 5 runs will give more valuable results than using the result of a single run.

Also, parallel version of the algorithm have been executed by using different number of computers (1, 2, 5 and 11 computers) in order to understand the relation of performance gain with the number of computers used. Here, it is important to make a parallel run by using 1 computer in order to compare with the serial one and evaluate the reliability of serial and parallel implementations. Trial of the parallel algorithm with only one computer can not produce results in shorter time than the serial algorithm, because there is only one CPU (Central Processing Unit) and physical memory for the use of parallel algorithm. Parallel algorithm should process clustering in the same duration with serial one in best, and sometimes it should last a little bit longer because of the parallelisation overhead when using 1 computer. When the number of computers increases, it should be observed that execution time of the parallel algorithm decreases almost proportional by the number of computers used. These results have been gathered as expected and will be presented in the following parts of the document.

It is important that, corresponding runs of each version must have the same random initial points in order for us to observe that the produced clustering results are the same and compare the execution times. Let's say, second run of the serial algorithm must use the same random initial points with second run of parallel version. In fact, parallel algorithm will also be executed by using different number of computers which also be repeated for 5 times. Executions of the parallel algorithm for different number of computers can be considered as different versions of parallel algorithm. Therefore, for the previous example, second run of the serial algorithm and each of the parallel versions must use the same random initial numbers and produce same results.

Because there are two different datasets, which are to be used for benchmarking, and because clustering of these datasets will be repeated 5 times with different random initial points, 10 different random initial point sets must be produced in total. Five set of random initial points for the first dataset, plus five sets of random initial points for the second dataset makes 10 different sets of random initial points. In order to distinguish between random initial points they have been named by representing the dataset of it and execution number of it. Naming of both datasets and random initial points are listed in tables below.

Table 6.5 Naming of Datasets

Dataset Name	**Narrative**
Dataset1	Color histograms of images (smaller dataset)
Dataset2	1990 US Census data (larger dataset)

Table 6.6 Naming of Random Initial Points

Random Initial Set	**Narrative**
Random11	Random initial points for the 1st dataset's 1st run
Random12	Random initial points for the 1st dataset's 2nd run
Random13	Random initial points for the 1st dataset's 3rd run
Random14	Random initial points for the 1st dataset's 4th run
Random15	Random initial points for the 1st dataset's 5th run
Random21	Random initial points for the 2nd dataset's 1st run
Random22	Random initial points for the 2nd dataset's 2nd run
Random23	Random initial points for the 2nd dataset's 3rd run
Random24	Random initial points for the 2nd dataset's 4th run
Random25	Random initial points for the 2nd dataset's 5th run

Random initial numbers have been produced in different ranges for Dataset1 and Dataset2, because data stored in datasets have different ranges. Random1x (random initial points for the first dataset) sets have been produced between 0 and 1, having 6 decimal points unlike Random2x (random initial points for the second dataset) sets which are produced between 1 and 10 without having decimal points. These ranges have been selected according to the nature of datasets in order to have better clustering.

It will be better to name the algorithms also in order to refer them clearly in the following parts. As mentioned previously parallel algorithm is executed on different numbers of computers and these executions can be considered as the different versions of parallel algorithm. Naming of the mentioned versions is presented in the table below.

Table 6.7 Naming of K-means Algorithm Versions

Algorithm Name	**Narrative**
Serial	Serial version
Parallel1	Parallel version executed by using 1 computer
Parallel2	Parallel version executed by using 2 computer
Parallel5	Parallel version executed by using 5 computer
Parallel11	Parallel version executed by using 11 computer

Finally, execution sequence of algorithms can be presented by using the naming standard listed above tables. Each algorithm (5 algorithms) has been executed by using each dataset (2 datasets) and by using each random initial point set (5 random initial points). Therefore, 50 executions in total have been performed (5 * 2 * 5 = 50). Mentioned executions are listed in the tables below. Table has been split into 5 sub tables in order to increase the understandability. As can be observed in the tables, Random1x sets are used with Dataset1 and Random2x sets are used with Dataset2.

Table 6.8 Execution Sequence and Naming of Serial Algorithm

Execution Name	**Narrative**
Run111	Serial on Dataset1 by using Random11
Run112	Serial on Dataset1 by using Random12
Run113	Serial on Dataset1 by using Random13
Run114	Serial on Dataset1 by using Random14
Run115	Serial on Dataset1 by using Random15
Run121	Serial on Dataset2 by using Random21
Run122	Serial on Dataset2 by using Random22
Run123	Serial on Dataset2 by using Random23
Run124	Serial on Dataset2 by using Random24
Run125	Serial on Dataset2 by using Random25

Table 6.9 Execution Sequence and Naming of Parallel1 Algorithm

Execution Name	Narrative
Run211	Parallel1 on Dataset1 by using Random11
Run212	Parallel1 on Dataset1 by using Random12
Run213	Parallel1 on Dataset1 by using Random13
Run214	Parallel1 on Dataset1 by using Random14
Run215	Parallel1 on Dataset1 by using Random15
Run221	Parallel1 on Dataset2 by using Random21
Run222	Parallel1 on Dataset2 by using Random22
Run223	Parallel1 on Dataset2 by using Random23
Run224	Parallel1 on Dataset2 by using Random24
Run225	Parallel1 on Dataset2 by using Random25

Table 6.10 Execution Sequence and Naming of Parallel2 Algorithm

Execution Name	Narrative
Run311	Parallel2 on Dataset1 by using Random11
Run312	Parallel2 on Dataset1 by using Random12
Run313	Parallel2 on Dataset1 by using Random13
Run314	Parallel2 on Dataset1 by using Random14
Run315	Parallel2 on Dataset1 by using Random15
Run321	Parallel2 on Dataset2 by using Random21
Run322	Parallel2 on Dataset2 by using Random22
Run323	Parallel2 on Dataset2 by using Random23
Run324	Parallel2 on Dataset2 by using Random24
Run325	Parallel2 on Dataset2 by using Random25

Table 6.11 Execution Sequence and Naming of Parallel5 algorithm

Execution Name	Narrative
Run411	Parallel5 on Dataset1 by using Random11
Run412	Parallel5 on Dataset1 by using Random12
Run413	Parallel5 on Dataset1 by using Random13
Run414	Parallel5 on Dataset1 by using Random14
Run415	Parallel5 on Dataset1 by using Random15
Run421	Parallel5 on Dataset2 by using Random21
Run422	Parallel5 on Dataset2 by using Random22
Run423	Parallel5 on Dataset2 by using Random23
Run424	Parallel5 on Dataset2 by using Random24
Run425	Parallel5 on Dataset2 by using Random25

Table 6.12 Execution Sequence and Naming of Parallel11 Algorithm

Execution Name	Narrative
Run511	Parallel11 on Dataset1 by using Random11
Run512	Parallel11 on Dataset1 by using Random12
Run513	Parallel11 on Dataset1 by using Random13
Run514	Parallel11 on Dataset1 by using Random14
Run515	Parallel11 on Dataset1 by using Random15
Run521	Parallel11 on Dataset2 by using Random21
Run522	Parallel11 on Dataset2 by using Random22
Run523	Parallel11 on Dataset2 by using Random23
Run524	Parallel11 on Dataset2 by using Random24
Run525	Parallel11 on Dataset2 by using Random25

It is obvious that, corresponding runs of algorithms must produce exactly same results. When explaining by using the names given above, results of Run111-Run211-Run311-Run411-Run511 must be equal to each other, because each of them uses Dataset1 and Random11. Similarly, results of Run112-Run212-Run312-Run412-Run512 must be same too, because they also use the same dataset and random initial points which are Dataset1 and Random12.

When considering that there are 2 different datasets and 5 different random initial point sets for each of these datasets, there must be 10 distinct clustering results of all executions. In the table below, these clustering results are named and executions, used datasets and random point sets in order to get these clustering results are listed.

Table 6.13 Expected Results of Executions

Result Name	Execution Names to Produce this Result	Used Dataset and Random Point Set
Result1	Run111, Run211, Run311, Run411, Run511	(Dataset1 & Random11)
Result2	Run112, Run212, Run312, Run412, Run512	(Dataset1 & Random12)
Result3	Run113, Run213, Run313, Run413, Run513	(Dataset1 & Random13)
Result4	Run114, Run214, Run314, Run414, Run514	(Dataset1 & Random14)
Result5	Run115, Run215, Run315, Run415, Run515	(Dataset1 & Random15)
Result6	Run121, Run221, Run321, Run421, Run521	(Dataset2 & Random21)
Result7	Run122, Run222, Run322, Run422, Run522	(Dataset2 & Random22)
Result8	Run123, Run223, Run323, Run423, Run523	(Dataset2 & Random23)
Result9	Run124, Run224, Run324, Run424, Run524	(Dataset2 & Random24)
Result10	Run125, Run225, Run325, Run425, Run525	(Dataset2 & Random25)

As shown in **Table 6.13**, there are 10 distinct result sets of 50 executions. In order for the parallelization of K-means algorithm to achieve its goal, executions listed in the same row must produce same results and the execution times must decrease when the number of computers used in the parallel versions increases (when moving right in the same row).

6.3 K-means Clustering Results of Datasets

Before benchmarking the serial and parallel versions of the K-means algorithm, initial runs for 10 different result sets have been performed by using classical K-means (serial one) without considering the times that they take to perform clustering. Results will only be discussed by the point of view of clustering concept. Whether the implemented K-means algorithm performs clustering successfully or not will be the point of view for this part. In the following parts, these produced results will not be examined again. They will only be evaluated by being equal to these previously examined results or not.

In order not to list pages of numbers, only summarised data about the clustering results will be listed in the document. These complete results of each of 50 executions can be found in the CD of the project.

As mentioned previously, K-means algorithm takes the number of clusters and initial means (initial points) as inputs and produces final means as output. Algorithm produces K means (this is why it is called K-means) where K is the number of clusters. At the termination of K-means algorithm, all points are grouped into one of K clusters and the total distances of points to the corresponding cluster means are minimized.

Number of produced clusters, which is one of the inputs of K-means algorithm, is up to the user of K-means clustering algorithm. The algorithm makes its best to produce the desired number of clusters. In this project, Dataset1 (smaller dataset) has been grouped into 5 clusters (K=5) and Dataset2 (larger dataset) has been grouped into 7 clusters (K=7) by using serial and parallel K-means algorithms. As the selection of result cluster counts does not make any sense for the scope of this project, count of result clusters are selected

without any consideration. As mentioned previously, clustering times is the first deal for this project.

Now, it is time to examine 10 distinct clustering results which have been produced by the execution of serial algorithm 10 times (for 2 datasets and 5 different initial points for each dataset). Total distortion using initial means and the total distortion using final means will also be listed in this section. Distortion is the sum of distances (Euclidean distances in this case) of all objects in a dataset into their cluster centroids. Final distortion must always be less than the initial distortion, because aim of the K-means clustering algorithm is to lessen this distortion. In other words, aim of the algorithm is to group similar objects into same clusters, which decreases the total distortion. As additional information, item counts of the result clusters will be supplied in order to have an overview of distribution of objects into clusters.

6.3.1 Execution Results of Dataset1

At first, execution results for Dataset1 will be listed as tables. Dataset1 has been clustered into 5 clusters as shown in the tables below by using 5 different random initial point sets. Initial numbers have been selected as numbers between 0 and 1 having 6 decimal points which is suitable to the nature of real dataset.

Cluster means have not been listed in these tables. These values have been presented in the CD of the project. A number of cluster count multiplied by dimension count (5 * 32 for the first dataset, 7 * 68 for the second dataset) means are present for one clustering result which require very much space to be listed in a document.

In following tables, item count column shows the number of items (objects) fall in to that cluster. Percent column shows the percentage of cluster to the whole dataset according to the item counts. And finally, cluster name is a naming method for addressing specific clusters. This column is used in order to match specific clusters of each execution, each of which uses different random initial number sets.

When different initial points are used, clustering results may differ. This is a small difference and results resemble each other. Because of the different initial points, order of

clusters may also differ. Cluster name column is used in order to match these clusters, order of which changes because of the initial points. In other words, a cluster which is produced as the second result cluster of dataset may be produced as the third result cluster when different initial points are used. This matching operation has been performed by observing the item counts of clusters.

Table 6.14 Item Distribution for Result1 Which Uses Random11

Cluster No	Item Count	Percent (%)	Cluster Name
1	12196	17.92	A
2	14143	20.79	B
3	5246	7.71	C
4	14161	20.81	D
5	22294	32.77	E
Total :	68040	100	

Table 6.15 Item Distribution for Result2 Which Uses Random12

Cluster No	Item Count	Percent (%)	Cluster Name
1	13151	19.33	B
2	23582	34.66	E
3	7758	11.40	A
4	6077	8.932	C
5	17472	25.68	D
Total :	68040	100	

Table 6.16 Item Distribution for Result3 Which Uses Random13

Cluster No	Item Count	Percent (%)	Cluster Name
1	7791	11.45	A
2	23590	34.67	E
3	13133	19.30	B
4	17492	25.71	D
5	6034	8.87	C
Total :	68040	100	

Table 6.17 Item Distribution for Result4 Which Uses Random14

Cluster No	Item Count	Percent (%)	Cluster Name
1	23591	34.67	E
2	6034	8.87	C
3	17492	25.71	D
4	13130	19.30	B
5	7793	11.45	A
Total :	68040	100	

Table 6.18 Item Distribution for Result5 Which Uses Random15

Cluster No	Item Count	Percent (%)	Cluster Name
1	13122	19.29	B
2	17475	25.68	D
3	7793	11.45	A
4	6044	8.88	C
5	23606	34.69	E
Total :	68040	100	

6.3.2 Execution Result's of Dataset2

In this section, execution results of Dataset2 will be examined. This dataset will be clustered into 7 clusters by using 5 different initial point sets. For this dataset, random initial numbers are produced between 1 and 10 having no decimal, because the attributes of dataset mostly occur between 1 and 10 as discrete values. This makes the initial points suitable with the nature of sample points which helps K-Means algorithm in order to converge in shorter time into better results.

Table 6.19 Item Distribution for Result6 Which Uses Random21

Cluster No	Item Count	Percent (%)	Cluster Name
1	232748	23.27	A
2	71900	7.19	B
3	60523	6.05	C
4	266843	26.68	D
5	129264	12.93	E
6	189470	18.95	F
7	49252	4.93	G
Total :	1000000	100	

Table 6.20 Item Distribution for Result7 Which Uses Random22

Cluster No	Item Count	Percent (%)	Cluster Name
1	189470	18.95	F
2	71900	7.19	B
3	60523	6.05	C
4	232748	23.27	A
5	266843	26.68	D
6	129264	12.93	E
7	49252	4.93	G
Total :	1000000	100	

Table 6.21 Item Distribution for Result8 Which Uses Random23

Cluster No	Item Count	Percent (%)	Cluster Name
1	249993	25.00	D
2	186606	18.66	F
3	71905	7.19	B
4	239017	23.90	A
5	67306	6.73	C
6	39628	3.96	G
7	145545	14.55	E
Total :	1000000	100	

Table 6.22 Item Distribution for Result9 Which Uses Random24

Cluster No	Item Count	Percent (%)	Cluster Name
1	49252	4.93	G
2	129264	12.93	E
3	60523	6.05	C
4	189470	18.95	F
5	71900	7.19	B
6	232748	23.27	A
7	266843	26.68	D
Total :	1000000	100	

Table 6.23 Item Distribution for Result10 Which Uses Random25

Cluster No	Item Count	Percent (%)	Cluster Name
1	280027	28.00	D
2	160799	16.08	G
3	60523	6.05	C
4	87758	8.78	E
5	149595	14.96	A
6	189470	18.95	F
7	71828	7.18	B
Total :	1000000	100	

When results are examined in detail, it can be observed that 5 different clustering of Dataset1 and again 5 different clustering of Dataset2 produces similar results. However, because initial points differ in these executions, sequence of clusters changes. Let's say 1st cluster of one execution may arise as the 3rd cluster of another execution because of the change of random initial points. In order to clarify these clusters, a naming column to final tables has been added. Matching of clusters between different runs has been performed by examining item counts of clusters.

First run of the Dataset1 (Result1) and last run of the Dataset2 (Result10) are less similar to other results when considering item counts of clusters. Therefore, matching of clusters mentioned above may not be exactly right, especially for Result1 and Result10. Because, slightly different clustering results may occur when initial means differ. However, the other runs have produced very similar (some times same) results to each other. Those two less similar executions may prove that result of K-means algorithm is dependent to the

selected initial means. When considering final distortions of results, those mentioned two executions are also similar to others. A final summary of results are listed in table below.

Table 6.24 Summary of 10 Results of K-means Clustering

Result Name	Initial Distortion	Final Distortion	Decrease in Distortion (%)	Iteration Count
Result1	30740.00	21398.94	30.39	78
Result2	28804.89	21180.73	26.47	30
Result3	30467.40	21180.95	30.48	53
Result4	30418.94	21180.96	30.37	38
Result5	30547.58	21180.85	30.66	34
Result6	64615815	10221581	84.18	51
Result7	64917583	10221581	84.25	49
Result8	64724989	17220502	73.39	63
Result9	66206826	10221581	84.56	50
Result10	64855999	9976791	84.62	32

Since total distortions have been decreased considerably, in can be concluded that items have been grouped better by using final means than using initial means which shows that K-means clustering has been successfully performed.

Another important point in **Table 6.24** is iteration counts. It can be observed that total decreases in distortion are very close to each other but the iteration counts for convergence of the algorithm are very different when using different initial points. This shows how much the time of execution is depended to the initial points.

6.4 Comparison of Results of Serial and Parallel K-means

After gathering 10 different clustering results, previously mentioned 50 runs have been performed by gathering time information also, for the purpose of benchmarking. When results are examined, it has been observed that each parallel version (parallel algorithm is same but the computer counts are different) have produced exactly the same results with the serial algorithm which are also equal to previously gathered 10 results. Therefore, in this section, these results won't be listed and interpreted again but the

execution times will be considered. All of the detailed results can be examined by viewing the CD of the project.

The goal of parallelization was to produce exactly same results in shorter times. First step of this goal have been achieved for this point as same results are gathered in serial and parallel versions. Now it is time to check for execution times. Execution times are listed in the tables below. Because 50 runs are very large for a table, results are split into 10 different tables for 10 different results.

Table 6.25 Execution Summary in Time for Dataset1 and Random11

Execution Name	Iteration Count	Execution Time(Seconds)	Ratio of Performance Gain	Number of Computers
Run111 (Serial)	78	198	1	1
Run211 (Parallel1)	78	199	0.9950	1
Run311 (Parallel2)	78	100	1.9800	2
Run411 (Parallel5)	78	40	4.9500	5
Run511 (Parallel11)	78	24	8.2500	11

Table 6.26 Execution Summary in Time for Dataset1 and Random12

Execution Name	Iteration Count	Execution Time(Seconds)	Ratio of Performance Gain	Number of Computers
Run112 (Serial)	30	76	1	1
Run212 (Parallel1)	30	77	0.9870	1
Run312 (Parallel2)	30	38	2.0000	2
Run412 (Parallel5)	30	15	5.0667	5
Run512 (Parallel11)	30	9	8.4444	11

Table 6.27 Execution Summary in Time for Dataset1 and Random13

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run113 (Serial)	53	135	1	1
Run213 (Parallel1)	53	135	1	1
Run313 (Parallel2)	53	68	1.9853	2
Run413 (Parallel5)	53	28	4.8214	5
Run513 (Parallel11)	53	16	8.4375	11

Table 6.28 Execution Summary in Time for Dataset1 and Random14

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run114 (Serial)	38	96	1	1
Run214 (Parallel1)	38	98	0.9796	1
Run314 (Parallel2)	38	48	2.0000	2
Run414 (Parallel5)	38	19	5.0526	5
Run514 (Parallel11)	38	12	8.0000	11

Table 6.29 Execution Summary in Time for Dataset1 and Random15

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run115 (Serial)	34	87	1	1
Run215 (Parallel1)	34	87	1	1
Run315 (Parallel2)	34	44	1.9773	2
Run415 (Parallel5)	34	18	4.8333	5
Run515 (Parallel11)	34	10	8.7000	11

Table 6.30 Execution Summary in Time for Dataset2 and Random21

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run121 (Serial)	51	5474	1	1
Run221 (Parallel1)	51	5482	0.9985	1
Run321 (Parallel2)	51	2767	1.9783	2
Run421 (Parallel5)	51	1096	4.9945	5
Run521 (Parallel11)	51	662	8.2689	11

Table 6.31 Execution Summary in Time for Dataset2 and Random22

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run122 (Serial)	49	5267	1	1
Run222 (Parallel1)	49	5276	0.9983	1
Run322 (Parallel2)	49	2658	1.9816	2
Run422 (Parallel5)	49	1055	4.9924	5
Run522 (Parallel11)	49	637	8.2684	11

Table 6.32 Execution Summary in Time for Dataset2 and Random23

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run123 (Serial)	63	6771	1	1
Run223 (Parallel1)	63	6779	0.9988	1
Run323 (Parallel2)	63	3417	1.9816	2
Run423 (Parallel5)	63	1355	4.9970	5
Run523 (Parallel11)	63	818	8.2775	11

Table 6.33 Execution Summary in Time for Dataset2 and Random24

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run124 (Serial)	50	5373	1	1
Run224 (Parallel1)	50	5384	0.9980	1
Run324 (Parallel2)	50	2714	1.9797	2
Run424 (Parallel5)	50	1078	4.9842	5
Run524 (Parallel11)	50	650	8.2662	11

Table 6.34 Execution Summary in Time for Dataset2 and Random25

Execution Name	Iteration Count	Execution Time (Seconds)	Ratio of Performance Gain	Number of Computers
Run125 (Serial)	32	3439	1	1
Run225 (Parallel1)	32	3445	0.9983	1
Run325 (Parallel2)	32	1736	1.9810	2
Run425 (Parallel5)	32	690	4.9841	5
Run525 (Parallel11)	32	415	8.2867	11

In order to have overall information about execution times, these times are summarized in the tables below. And also, mean of 5 different runs are listed at the last row of tables. As it can be observed, selection of initial means is also an important criterion for the convergence time of K-means algorithm. For example, first dataset converges in 198 seconds in first run (first set of random initial means) and converges in 76 seconds in second run (second set of random initial means).

Table 6.35 Execution Times in Seconds for the First Dataset

Run	Serial	1 Process	2 Processes	5 Processes	11 Processes
1	198	199	100	40	24
2	76	77	38	15	9
3	135	135	68	28	16
4	96	98	48	19	12
5	87	87	44	18	10
Mean	118.40	119.20	59.60	24.00	14.20

Table 6.36 Execution Times in Seconds for the Second Dataset

Run	Serial	1 Process	2 Processes	5 Processes	11 Processes
1	5474	5482	2767	1096	662
2	5267	5276	2658	1055	637
3	6771	6779	3417	1355	818
4	5373	5384	2714	1078	650
5	3439	3445	1736	690	415
Mean	5264.80	5273.20	2658.40	1054.80	636.40

When calculating the means of above tables, ratio of performance gain becomes 0.9954 for Parallel1, 1.9845 for Parallel2, 4.9676 for Parallel5 and 8.3200 for Parallel11.

This results show that parallel version with one computer of the K-means algorithm completes execution almost at the same time with the serial one. This is very normal, because when one computer is used, parallelization of algorithm makes no sense. There is a very little performance loss for one computer parallel version which is because of the overhead of parallelization.

When 2 and 5 computers are used with parallelized version of algorithm, it completes execution almost 2 and 5 times faster than the serial algorithm. Performance gain is not exactly 2 or 5, because there is a little messaging overhead in parallelized version.

When 11 computers are used in parallel version, the performance gain becomes 8.32 instead of 11. This situation can be explained by several reasons. At first, because root computer is responsible for the synchronization of all computers, when the number of computers used increases, the overhead of root computer increases too. As all computers are dependent to root computer, this situation may cause a performance loss. And also, traffic of messages in the network increases by the raising number of computers. This traffic load may also cause a performance loss in parallel computing. Besides, performance lack of a computer in parallel computing network causes performance loss which decreases the performance gain, because all computers need to wait for the slowest computer before moving to the next iteration of K-means clustering.

Another important result of executions is the effect of initial points to the clustering time. When considering execution times of different initial points for the same algorithm

version, in can be observed that sometimes a performance change with a proportion larger than two is present. Such as, serial algorithm for the first dataset converges in 198 seconds with first initial point set and converges in 87 seconds with fifth initial point set which makes a performance change by the proportion of 2.6.

Now, it is time to present execution performance results in graphics in order to clarify results in tables above. At first, execution times (means of 5 runs with different initial points) of different versions of the algorithm will be presented in graphics one for first dataset and one for second dataset.

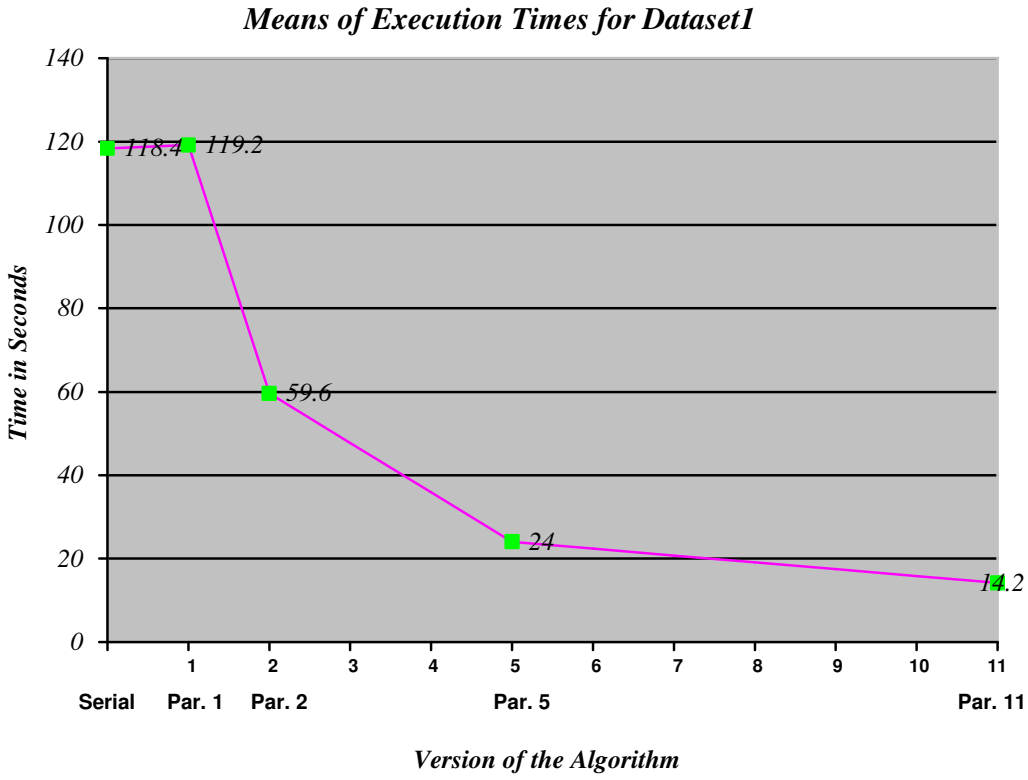

Figure 6.1 Means of Execution Times for Color Histogram Dataset

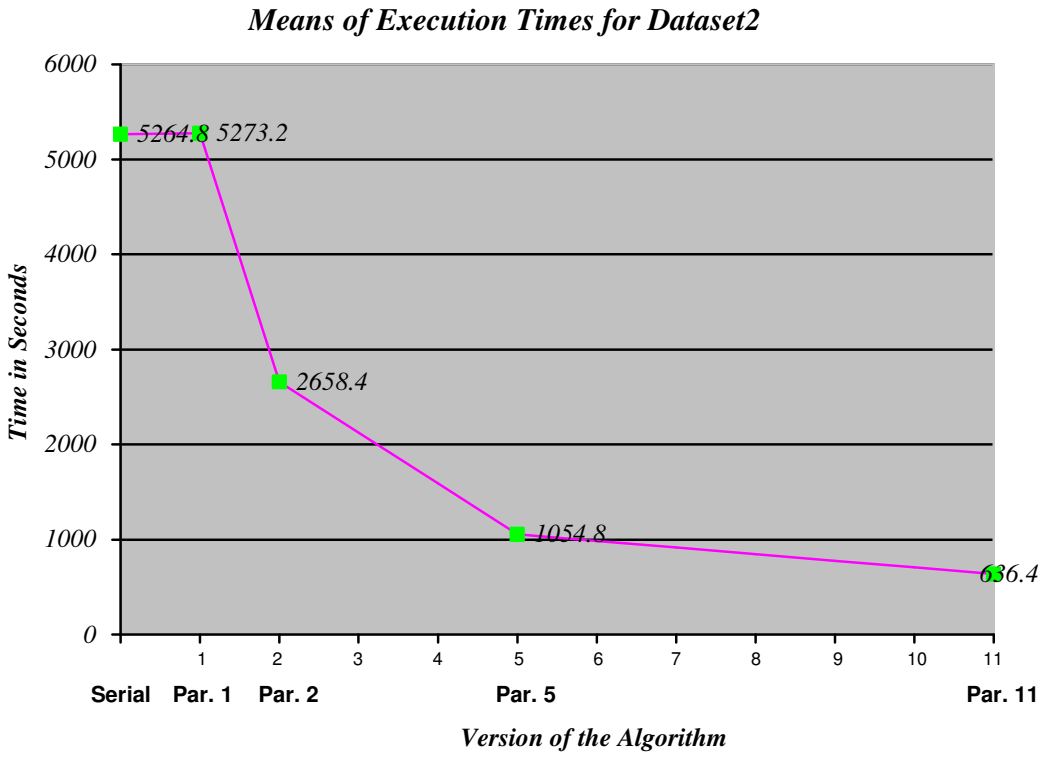

Figure 6.2 Means of Execution Times for US Census Dataset

Performance increase (decrease in required time for execution) corresponding to the increase of computer count with parallel algorithm can be observed easily from the graphics above. When considering the above tables and graphics, it can be concluded that the parallel algorithm provides the same proportion of performance increase both with smaller dataset (68,040 rows with 32 attributes) and with larger dataset (1,000,000 rows with 68 attributes). In other words, parallelisation of K-means, proposed in this project, works on all sizes of datasets.

In above graphics, means of execution times for different runs with random initial points are used. As it has been observed in 50 times executions of K-means algorithm with different variables (dataset, initial points, algorithm version) that initial points are also effective for the execution time of the algorithm, it is also necessary to present graphics by using execution times of different initial points in detail. This graphics (for Dataset1 and for Dataset2) are presented below.

Below graphics are presented to distinguish between different random initial points. Unlike the above graphics which show only the means of different initial points, these ones show all values for different initial points in detail.

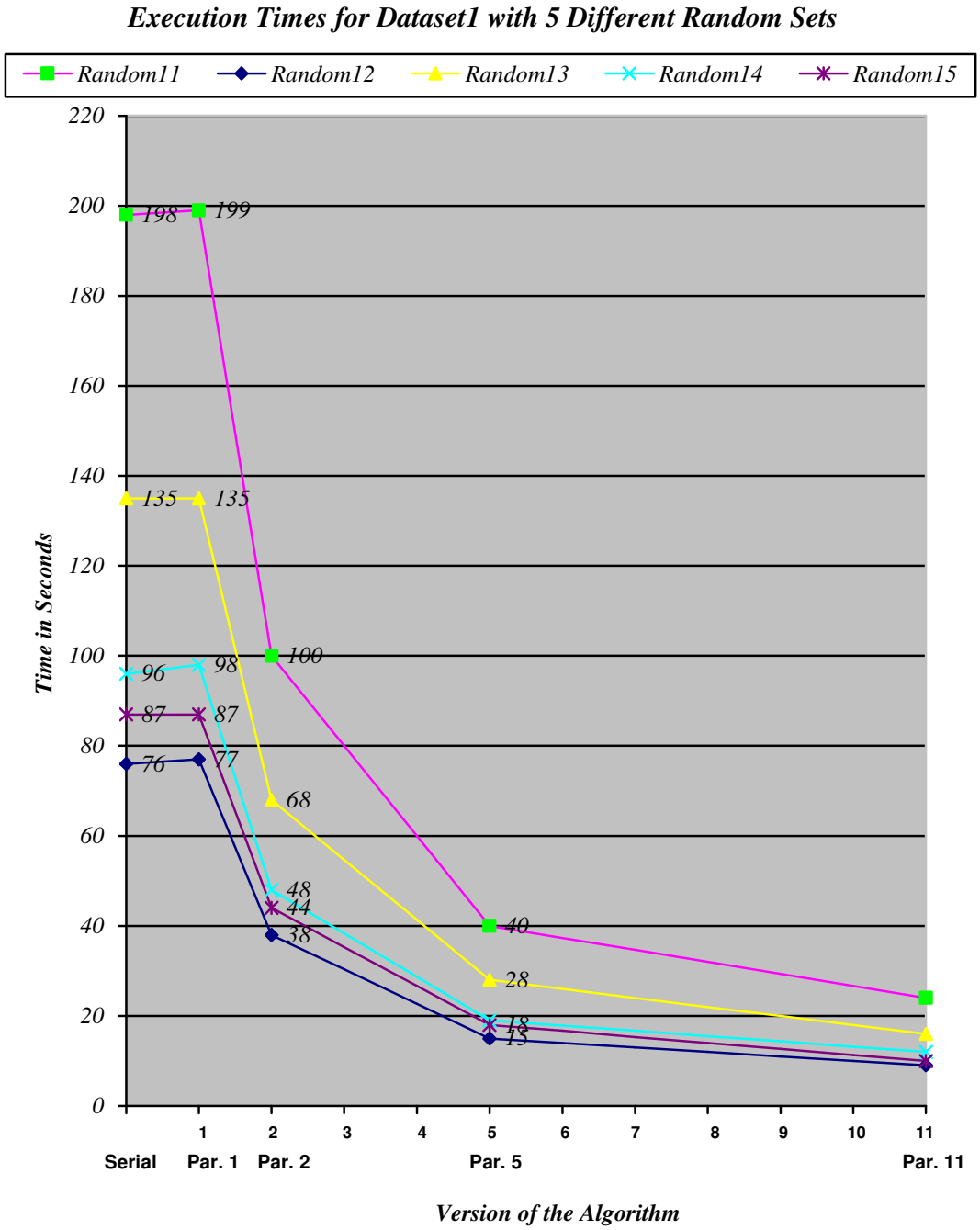

Figure 6.3 Execution Times for Color Histogram Dataset

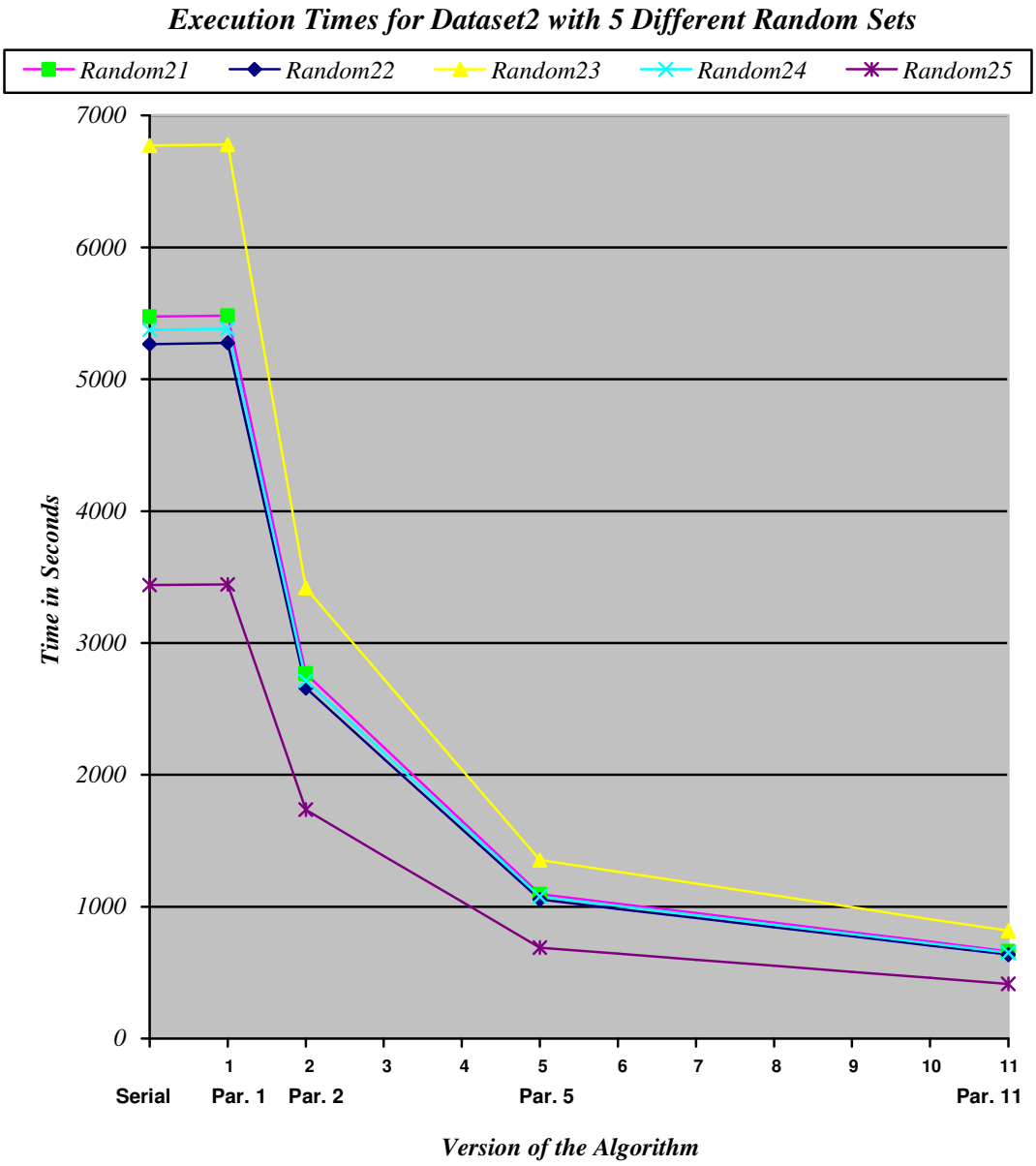

Figure 6.4 Execution Times for US Census Dataset

Effect of both parallelisation and selection of initial points to the performance of K-means clustering algorithm can be seen from detailed graphics. When using 11 computers by the parallelisation, algorithm converges more than 8 times faster. And also, by the change of initial points, performance changes of up to 2.6 times are present in the trial runs of this project. These results brings the idea of connection of both parallelisation of the

algorithm and refining the initial points of the algorithm in order to get even faster executions. This work can be considered as a valuable future work.

When examining 25 runs for each dataset, top and bottom execution times occur as follows. Dataset1 has been clustered in 198 seconds in worst case and in 9 seconds in best case. Dataset2 has been clustered in 6771 seconds in worst case and in 415 seconds in best case. In other words, a performance increase of 22 times for Dataset1 and a performance increase of 16.3 times for Dataset2 is present. These performance gains are very valuable for an algorithm especially for ones which deal with very large amounts of data. In summary, although the most part of performance gain belongs to the parallelism, connection of refining of initial points with parallelism may even take the performance increase of the K-means algorithm further which leads to a great performance increase.

As a result, it can be concluded that developed parallelization of K-means clustering algorithm provides a performance gain almost proportional by the number of computers which shows that parallelization of K-means algorithm has achieved its goal. This performance gain is gathered independent of dataset size, from small to very large datasets, because messaging in developed parallel version is in very small amounts. Furthermore, performance gain will be enormous when dealing with datasets that do not fit into real memory of a single computer. Because dataset is split into real memories of different computers in parallel version, total size of memory area increases dramatically when compared to a single computer.

CHAPTER 7

CONCLUSION AND FUTURE WORKS

Main aspect of this project has been to improve the K-means algorithm so that it can perform clustering on large datasets in reasonable durations. When considering databases of current companies, this improvement becomes very critical, because current companies have huge amounts of data and they need to mine their databases in short durations in order to have marketing advantage.

When examining serial K-means algorithm, it can be observed that the algorithm deals with all objects in dataset serially which very time consuming especially for large databases. When huge datasets are in account, serial K-means algorithm either lacks in performance or crashes because of the larger dataset than the amount of memory of a single machine.

In this project, parallelization of K-means algorithm has been proposed as an improvement for the algorithm. It has been proposed that, parallel version of the algorithm will produce exactly the same results with the serial algorithm in much shorter durations almost by the number of computers used.

Parallel version of the algorithm has been designed and implemented by using C language and LAM-MPI (Local Area Multi Computer - Message Passing Interface) utility which can be used by C programs. Serial algorithm has also been implemented by C language for the purpose of comparison with parallel version.

Design of the parallel algorithm has been performed carefully, such that algorithm requires minimal frequency of messaging between processes with minimal message sizes. Unlike the previous parallelization works of K-means algorithm, which transmit objects in whole dataset between processes in each iteration of the clustering algorithm, only summary of objects in dataset is transmitted between processes. Summary of objects, mentioned above, is the sum and count of objects in a cluster which is a very small data when comparing with the all objects in dataset.

After executions of serial and parallel K-means algorithms, it has been observed that parallel algorithm produces exactly the same results with the serial algorithm in much shorter durations. Performance gain of parallelization is almost by the number computers used for parallel execution. Let's say, algorithm runs almost N times faster when N computers used for parallel execution.

This performance gain is present for all size of datasets unlike the previous works on parallelization of K-means algorithm which run slower than the serial algorithm unless the dataset is large enough. And also previous parallel implementations of K-means algorithm propose a performance gain by K/2 (K is the number of clusters) which is not a considerable gain for large datasets and limited by the number of clusters.

When examining results of different runs with different sets of random initial points, it has been observed that selection of initial points also affects the convergence time of K-means algorithm. In one example, execution time of the algorithm has become the half of the previous run time by the change of random initial points. This shows that, a technique for selecting better initial points than random ones may be developed and used in connection with the parallel algorithm as a future work in order to make K-means algorithm even faster.

REFERENCES

[Ali, Ghani and Saeed 2001] R. Ali, U. Ghani, A. Saeed, **"**Data Clustering and Its Applications", Rudjer Boskovic Institute, 2001

[Apte 1997] C. Apte, "Data Mining – An Industrial Research Perspective", IEEE Computational Science and Engineering, 1997

[Beddo 2002] V. Beddo, "Applications of Parallel Programming in Statistics", University of California, 2002

[Bradley and Fayyad 1998] P. S. Bradley, U. M. Fayyad, "Refining Initial Points for K-means Clustering", Fifteenth International Conference on Machine Learning, pages 91-99, Morgan Kaufmann, San Francisco, CA, 1998

[Bradley, Fayyad and Reina] P. S. Bradley, U. Fayyad, C. Reina, "Scaling Clustering Algorithms to Large Databases", Microsoft Research

[Crocker and Keller] M. Crocker, F. Keller, "Connectionist and Statistical Language Processing", University of Saarlandes

[Edelstein] H. Edelstein, "Building Profitable Customer Relationships With Data Mining", Two Crows Corporation

[Fayyad, Piatetsky-Shapiro and Smyth 1996] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, "From Data Mining to Knowledge Discovery in Databases", American Association for Artificial Intelligence, 1996

[Ganti, Gehrke and Ramakrishnan 1999] V. Ganti, J. Gehrke, R. Ramakrishnan, "Mining Very Large Databases", IEEE Computer, 1999

[Hengl 2003] T. Hengl, "What Is Data Mining? Finding The Pattern", PC AI Magazine, 2003

[Hettich and Bay 1999] S. Hettich, S. D. Bay, "The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science", 1999

[Javid 1999] S. Javid, "Data Mining In The Next Millennium", DM Direct Newsletter, 1999

[Kantabutra 1999] S. Kantabutra, "Parallel K-means Clustering Algorithm on NOWs", Department of Computer Science, Tufts University, 1999

[MHPCC 1999] Maui High Performance Computing Center**, "**Parallel Programming Workshop – Parallel Programming Introduction", http://www.mhpcc.edu/training/workshop/parallel_intro/MAIN.html, 1999

[Moore 2001] A. W. Moore, "K-means and Hierarchical Clustering", School of Computer Science, Carnegie Mellon University, 2001

[MPI] http://www.lam-mpi.org/

[Palace 1996] B. Palace, "Data Mining", Technology Note prepared for Management 274A, Anderson Graduate School of Management at UCLA, 1996

[Skillicorn 1999] D. Skillicorn, "Strategies for Parallel Data Mining", IEEE Concurrency, 1999

[Stoffel and Belkoniene 1999] K. Stoffel, A. Belkoniene, "Parallel k/h-means Clustering for Large Data Sets", Euro-Par, 1999

[Two Crows Corporation 1999] Two Crows Corporation, "Introduction to Data Mining and Knowledge Discovery", 1999