DATA DECOMPOSITION TECHNIQUES FOR PARALLEL TREE-BASED K-MEANS CLUSTERING

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Cenk Şen

July, 2002

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Attila Gürsoy (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Özgür Ulusoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute of Engineering and Science

ABSTRACT

DATA DECOMPOSITION TECHNIQUES FOR
PARALLEL TREE-BASED K-MEANS
CLUSTERING

Cenk Şen
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Attila Gürsoy
July, 2002

The main computation in the k-means clustering is distance calculations between cluster centroids and patterns. As the number of the patterns and the number of centroids increases, time needed to complete computations increased. This computational load requires high performance computers and/or algorithmic improvements. The parallel tree-based k-means algorithm on distributed memory machines combines the algorithmic improvements and high computation capacity of the parallel computers to deal with huge datasets. Its performance is affected by the data decomposition technique used. In this thesis, we presented novel data decomposition technique to improve the performance of the parallel *tree-based* k-means algorithm on distributed memory machines. Proposed tree-based decomposition techniques try to decrease the total number of the distance calculations by assigning processors compact subspaces. The compact subspace improves the performance of the pruning function of the tree-based k-means algorithm. We have implemented the algorithm and have conducted experiments on a PC cluster. Our experimental results demonstrated that the tree-based decomposition technique outperforms the random decomposition and stripwise decomposition techniques.

Keywords : Clustering, parallel algorithm, load balancing, data decomposition.

ÖZET

AĞAÇ TABANLI PARALEL K-ORTALI
GRUPLAMA İÇİN VERİ DAĞITIM
TEKNİKLERİ

Cenk Şen
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yard. Doç. Dr. Attila Gürsoy
Temmuz, 2002

K-ortalı gruplamada asıl olan hesaplama yükü veri vektörleri ile grupların ortaları arasındaki uzaklık hesaplamalarıdır. Veri vektörlerinin ve grup ortalarının sayıları arttırıldıkça, hesaplamaları tamamlamak için gerekli olan zaman artar. Bu hesaplama yükü yüksek performanslı bilgisayarlar ve/veya algoritmik gelişmeler gerektirir. Büyük veri kümelerini işlemek için dağınık hafızalı makinalardaki paralel ağaç tabanlı k-ortalı algoritması algoritmik iyileştirmeler ile paralel bilgisayarların yüksek hesaplama kapasitesini birleştirmiştir. Algoritmanın performansı veri dağıtım tekniğinden etkilenmektedir. Bu tezde, dağınık hafızalı makinalardaki paralel ağaç tabanlı k-ortalı algortimasının performansını arttıracak yeni bir veri dağıtım tekniği sunduk. Önerilen ağaç tabanlı dağıtım teknikleri işlemcilere sıkışık altalanlar vererek toplam uzaklık hesaplamalarının sayısını düşürmeyi amaçlamaktadır. Sıkışık altalanlar ağaç tabanlı k-ortalı algoritmasının budama fonksiyonunun performansını arttırmaktadır. Algoritmanın gerçekleştirilmesi ve performans deneyleri gruplandırılmış kişisel bilgisayarlar üzerinde yapılmıştır. Deney sonuçlarımız ağaç tabanlı dağıtım tekniğinin karışık dağıtım ve şeritvari dağıtım tekniklerinden daha iyi performansı olduğunu göstermiştir.

Anahtar sözcükler : Gruplama, paralel algoritma, yük dengesi, veri dağıtımı.

Acknowledgement

I would like to express my special thanks and gratitude to Asst. Prof. Dr. Attila Gürsoy, from whom I have learned a lot, due to his supervision, suggestions, and support during the past year. I would like especially thank to him for his understanding and patience in the critical moments.

I am also indebted to Assoc. Prof. Dr. Özgür Ulusoy and Assist. Prof. Dr. Uğur Güdükbay for showing keen interest to the subject matter and accepting to read and review this thesis.

I would like to thank to my parents and my wife for their morale support and for many things.

I would like to thank to Turkish Armed Forces for giving this great opportunity.

I am grateful to all the honorable faculty members of the department, who actually played an important role in my life to reaching the place where I am here today.

I would like to individually thank all my colleagues and dear friends for their help and support.

CONTENTS

List of Figures

List of Tables

List of Symbols and Abbreviations

PC	: personal computer
n	: number of the input patterns
k	: number of the input cluster
p	: number of the processors in the system
P_i	: i^{th} pattern
f_{ix}	: x^{th} feature of the i^{th} pattern
d	: dimension of the data
Cj	: j^{th} cluster
c_j	: centroid of the j^{th} cluster
0	: asymptotic big OH
Minmax	: minimum of the maximum distances
N	: cell of the pattern tree
SCF	: simple cost function
LCF	: level cost function
CCF	: centroid cost function
LCFC	: centroid and level cost function
BORG	: 32 node distributed memory PC Cluster of Bilkent University

CHAPTER 1

INTRODUCTION

1.1 Overview

Recent times have seen an explosive growth in the availability of various kinds of data. It has resulted in an unprecedented opportunity to develop automated data driven techniques of extracting useful knowledge. Data mining, an important step in this process of knowledge discovery, consists of methods that discover interesting, non-trivial, and useful patterns hidden in the data [2]. The field of data mining builds upon the ideas from diverse fields such as machine learning, pattern recognition, statistics, database systems, and data visualization. But, techniques developed in these traditional disciplines are often unsuitable due to some unique characteristics of today's data-sets, such as their enormous sizes, high dimensionality and heterogeneity [1].

The clustering problem has been addressed in many contexts and by researchers in many disciplines such as data mining [7], statistical data analysis [8], compression [4], vector quantization; this reflects its broad appeal and usefulness as one of the steps in exploratory data analysis. However, clustering is a difficult problem combinatorially, and differences in assumptions and contexts in different communities have made the transfer of useful generic concepts and methodologies slow to occur [3].

Cluster analysis is the organization of a collection of patterns (usually represented as a point in a multidimensional space) into clusters based on similarity. Intuitively, patterns within the same cluster are more similar to each other than they are to a pattern belonging to a different cluster [3]. The clustering process tries to increase similarity between patterns of a particular cluster, and tries to decrease similarity between patterns of different clusters. Clustering has been formulated in various ways in the machine learning [9], pattern recognition [10], optimization [11], and statistics literature [8,12]. Clustering can be simply formalized as follows: Given the desired number of clusters k and a dataset of n points, and a distance based measurement function (e.g., the weighted total/average distance between pairs of points in clusters), we are asked to find a partition of the dataset that minimizes the value of the measurement function [4]. So, the typical pattern clustering activity includes the following steps [6], which are depicted in Figure1.1. The first step is the pattern representation, which refers to the number of classes, the number of available patterns, and the number, type, and scale of the features available to the clustering algorithm. Some of this information may not be controllable by the practitioner. The second step is the definition of a pattern proximity measure appropriate to the data domain. It is the process of identifying the most effective subset of the original features to use in clustering. Pattern proximity is usually measured by a distance function, such as *Euclidean distance*, defined on pairs of patterns. The third step is clustering or grouping. This step can be performed in a number of ways so that there are many different clustering algorithms.

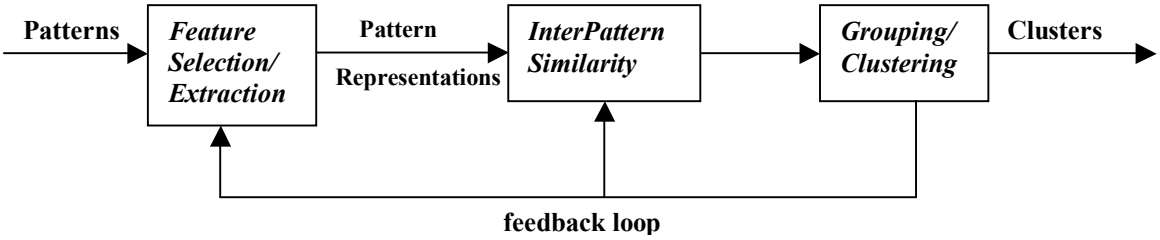


Figure 1.1. Stages in clustering.

Since clustering is applicable to many different areas, variety of clustering techniques can be obtained. Categorization of these techniques according to their clustering structure shows that there are two major clustering algorithms, partitioning clustering and hierarchical clustering. In hierarchical clustering, each group of size greater than one is in turn composed of smaller groups. So, a hierarchical algorithm yields a dendrogram (tree) representing the nested grouping of patterns and similarity levels at which groupings change. In partitioning clustering every pattern is exactly in one group according to similarity measure. Partitional methods have advantages in applications involving large data sets for which the construction of a dendrogram is computationally expensive [3,13].

K-means is a partitional clustering method and because of its easy implementation it is one of the most commonly used clustering algorithm. The k-means method has been shown to be effective in producing good clustering results for many practical applications, for this reason there are variety of different implementations [15]. Inputs of the k-means algorithm are the patterns, the predefined number of clusters. Algorithm employs *Euclidean distance* based similarity metric function between center of predefined number of cluster and patterns. At each step, every pattern is assigned to the nearest cluster. After all patterns assigned, cluster centroids are updated to represent new clusters. These calculation operations are repeated until either no pattern need to be moved or predefined number of iteration of calculation process.

Most of the early cluster analysis algorithms come from the area of statistics and have been originally designed for relatively small data sets. In the recent years, clustering algorithms have been extended to efficiently work for knowledge discovery in large databases and some of them are able to deal with high-dimensional feature items. When they are used to classify large data sets, clustering algorithms become computationally demanding and require high performance machines to get results in reasonable time. Experiences of clustering algorithms taking from one week to about 20 days of computation time on sequential machines are not rare [16]. Thus, scalable parallel computers can provide the appropriate setting where to execute clustering algorithms for extracting knowledge from large-scale data repositories.

Over the last years parallel computing has received considerable attention of researchers in clustering. The main reason for the growing interest is the difficulties to increase performance of sequential computers due to technical and physical limitations. Also, the availability of cheap mass fabricated microprocessors and communication switches makes it more economical to connect hundreds of these components than to build highly specialized sequential computers. Such a collection of processors working in parallel can achieve unlimited performance and is suitable to solve problems of all areas of science.

The huge size of the available data-sets and their high-dimensionality make large-scale data mining applications computationally very intensive, because of its high-performance, parallel computing is becoming an essential component of the solution of the large scale data mining applications. Another opportunity that makes parallel computing popular in data mining is the quality of the data mining results. The quality of data mining results often depends directly on the amount of computing resources available, as the capability of computing resource is increased, the quality of the results are also increase. In fact, data mining applications will be the dominant consumers of supercomputing in the near future [18]. Although, designing parallel data mining algorithms is challenging, there is a necessity to develop effective parallel algorithms for various data mining techniques.

1.2 PROBLEM DEFINITION

In the k-means clustering, we have *n* patterns as an input each of which represents feature vector. Our main aim is to cluster *n* patterns to predefined number of clusters, denoted by *k*, with chosen similarity metric such as *Euclidean distance*.

The main computation in the k-means clustering is distance calculations between cluster centroids and patterns. As the number of the patterns and the number of centroids are increased, time needed to complete computations will be increased. It is clear that, execution time, per iteration of k-means, is sensitive to both the number of patterns and the number of centroids. Since today's datasets are very huge, this computational load requires high performance computers and/or algorithmic improvements.

One of the new approaches to increase efficiency of the k-means algorithm is Alsabti's tree-based k-means clustering with pruning some of the distance calculations [15]. In this approach, a pattern tree is built, each node of this tree contains a subset of patterns, as we traverse the pattern tree some candidate cluster centroids set is determined according to pruning algorithm. When only one candidate cluster is left, patterns, belonging to that tree node, are assigned to the cluster centroid without doing any distance calculation. This technique significantly decreases the execution time of the k-means clustering, by reducing the distance calculations.

Algorithmic improvements make k-means more efficient, however sequential k-means is still not satisfactory for really huge datasets. Parallelization may be one of the best choices for performance improvements of the k-means. The main workload of the k-means is the distance calculations between pattern and cluster centroids. One of the standpoints is that, calculations, made for each pattern, do not differ from pattern to pattern, k distance calculations are performed for each pattern. Besides, another standpoint of parallelization of k-means is that, the distance calculations are independent from each other. With these standpoints efficient parallelization of k-means can be accomplished by distributing equal number of patterns and a local copy of cluster centroids to each processor. Since computation load directly proportional to number of input pattern, each processor makes $(n/p * k)$ calculations, where n is the total number of patterns, p is processor number, and k is number of cluster, per iteration. At the end of the distance computations, root processor makes update operation of the cluster centroids, and the new centroids are sent to the other processors for the next iteration. These steps are repeated until predefined condition is satisfied, such as predefined number of iteration, no pattern needs to be moved to new cluster. In the implementation of the direct parallel k-means, none of the parallelization issues, reference to parallel k-means such as load balancing, data locality becomes a problem.

Parallelization of the Alsabti's tree based k-means method, for shared memory architectures, has been described in [2]. Parallel k-means clustering is more challenging because of the algorithm structure. Irregular tree decomposition of the space, which is directly related to the pattern distribution on the space, and changing computations due to the pruning algorithm, and also changing calculations during traversal of pattern tree, make parallelization of tree based k-means more challenging. Two different kinds of computations are done during traversal of the tree, leaf computations and non-leaf computations. In the leaf computations,

distances between the patterns and cluster centroids are measured and assigned to the nearest cluster. The amount of the leaf computation is directly related to the success of the pruning algorithm. Since the main computation load is proportional to the number of cluster centroids and the number of the patterns, more pruning in the candidate set of centroids and in the patterns, will ensure less leaf computations. So one of the problems is, to increase the candidate set of centroids pruning and to increase the number of pruned patterns, to decrease the leaf computations. In the non-leaf computations, candidate set is compared with the space covered by the node, and unrelated candidate clusters are pruned after these computations. Naturally, some of the candidate set of centroids might have been pruned in the upper level of the pattern tree; so, the number of non-leaf calculations can vary throughout internal nodes of the pattern tree.

Since data decomposition is very important issue for parallel tree-based k-means algorithm, we proposed a data decomposition technique, which decomposes patterns to processors in a manner that local patterns of the each processors is scattered in a smaller area. Thus, performance of the parallel tree-based k-means algorithm increases.

1.3. Outline of the Thesis

This work describes pattern decomposition techniques for parallel tree-based k-means algorithm on distributed system parallel computers with an experimental work on *PC* clusters. The data decomposition is very important issue for the parallel systems, because it affects the performance of the parallel algorithm. In this work we proposed *tree-based* pattern decomposition techniques, which improves the performance of the parallel tree-based k-means algorithm.

In the second chapter of the thesis, sequential *standard* k-means and sequential *tree-based* k-means algorithms are explained in order to understand differences between them. The parallelized version of the algorithms and their parallelizing issues are discussed. The importance of the pattern decomposition over the parallel tree-based k-means algorithm is presented.

In third chapter, proposed decomposition techniques and implementation details of them are described in details. Their standpoints are explained in details. The advantages and disadvantages of each of them are presented. In order to achieve better results, various cost functions, which are used to estimate computational load or leaf computations, are explained.

In the fourth chapter, the experimental results that are obtained by implementation of the algorithm on a *PC* cluster are reported. Each decomposition technique is examined with four metrics, all of these metrics are also overviewed. A comparison of the pattern decomposition technique is presented and the results of the experiments are analyzed. The results of the experiments are explained and supported with appropriate graphics.

In the last chapter, we concluded our work.

CHAPTER 2

BACKGROUND

2.1 K-Means Clustering

The k-means clustering algorithm is the simplest and the most commonly used algorithm. It employs *squared error* similarity criteria, which is widely used criterion function in partitional clustering. It starts with predefined number of initial set of clusters and at each iteration, patterns are reassigned to the nearest cluster based on the distance based similarity measure, this process is repeated until a converge criterion is met such as no reassignment of any pattern to a new cluster or predefined error value. Let us examine k-means algorithm in detail.

Let say we have n input patterns and patterns are denoted by $P_1, P_2, \ldots, P_n$. The pattern P_i (i^{th} pattern) consists of a tuple of describing features where features are denoted by $f_{i1}, f_{i2}, \ldots, f_{id}$. A dimension represents each feature, where d is the number of dimensions of the value space. The second input of the algorithm is the predefined number of clusters, denoted by k. The number of the clusters cannot be changed during the execution of the algorithm. Let $C_1, C_2, \ldots, C_k$ be the clusters, and each cluster is represented by its centroid. Let $c_1, c_2, \ldots, c_k$ be the centroids of the clusters. The sketch of the algorithm is illustrated in Figure 2.1. Algorithm works as follows: First, the initial cluster centroids are formed randomly. The distances between pattern P_i and all clusters are calculated and pattern P_i is assigned to the

closest cluster C_d. This process is repeated for all patterns and all patterns are assigned to a unique cluster. At the end of the iteration all centroids $(c_1, c_2, \ldots, c_k)$ are updated. In the next iteration, distance calculations between patterns and clusters are repeated with the updated centroids. The algorithm will iterate until predefined number of iteration is reached or no pattern is moved to new cluster. At the end of the algorithm, quality of the clustering is measured by the error function:

$$E = \sum_{d=1}^{k} \sum_{P_i \in C_d} |P_i - C_d||^2 \qquad (2.1)$$

```
Algorithm K-Means(P_n, C_k)
     initialize input patterns (P_1, P_2, …, P_n)
     initialize cluster centroids (c_1, c_2, … c_k)
     while (predefined termination condition
is satisfied){
      for i = 1 to n
         for j = 1 to k
          if  | P_i - c_nearest | > | P_i – c_j |
              c_nearest =  c_j
           assign P_i to c_nearest
     endfor
      for j = 1 to k
        calculate and update centroid c_j
           compute Error :
           E = Σ_{d=1}^{k} Σ_{P_i ∈ C_d} |P_i − C_d||^2
     }
```

Figure 2.1: K-means clustering algorithm

The time complexity of the direct *k-means* algorithm can de divided into three parts [15]. The time required for the assigning patterns to the closest cluster (first *for* loop in Figure 2.1) is *O(nkd)*. The time required for updating the cluster centroids (second *for* loop in

Figure 2.1) is *O(nd)*. And the time required for calculating the error function is *O(nd)*. Cleary, the computational cost of the algorithm is directly related to the number of the input patterns and the number of the input clusters. If these two input parameters are increased so much, direct implementation of the *k-means* can be computationally very expensive. This is especially true for today's data mining applications with large number of pattern vectors.

There are some approaches described in the literature, which can be used to reduce the computational cost of the k-means clustering algorithm. One of the approaches uses the information from the previous iteration to reduce the number of the distance calculations. P-Cluster is a k-means based clustering algorithm, which exploits the fact that the change of the assignment of patterns to clusters are relatively few after the first few iterations [27]. It uses simple check that whether the closest cluster of a pattern has been changed or not. If the assignment has not been changed no further distance calculation is required for this pattern.

```
Function TreeTraverse(node,CandidateSet)
       NewSet =  Pruning(node,
CandidateSet)
       if | Newset | = 1 then /*ıf the set
has one element*/
     assign all patterns P_node
           to candidate centroid c_c
             return
        if node type = leaf then
     /*perform k-means with candidate set
          and patterns in the node */
     for each pattern P_i
           find the closest c_c
           assign pattern P_i to c_c
     return
        for each child node do
     TreeTraverse(child, Newset)
```

Figure 2.2: The tree traverse function of the tree-based k-means algorithm.

Alsabti et al. proposed another approach to reduce number of the distance calculations by pruning the cluster centroids and the patterns [15]. It uses a tree structure and organizes patterns with this structure, and then by using some geometrical constraints, it prunes some of the candidates. Even, when the candidate set of a pattern is consisting of only one cluster, it assigns pattern without any distance calculation, details of the algorithm will be given in the next section.

2.2 Tree-Based K-Means

Alsabti et al. proposed a tree-based k-means algorithm supported with pruning activity [15]. Their algorithm uses *k-d tree* structure to organize the input patterns according to their coordinates. The representation of the patterns provides rough grouping of patterns. Thus, patterns in a cell of the pattern tree are surely become closer to each other. The root of the pattern tree represents the all of the input patterns and covers all of the working space where is covered by the all input patterns. The children node of the pattern tree represents patterns in subspaces.

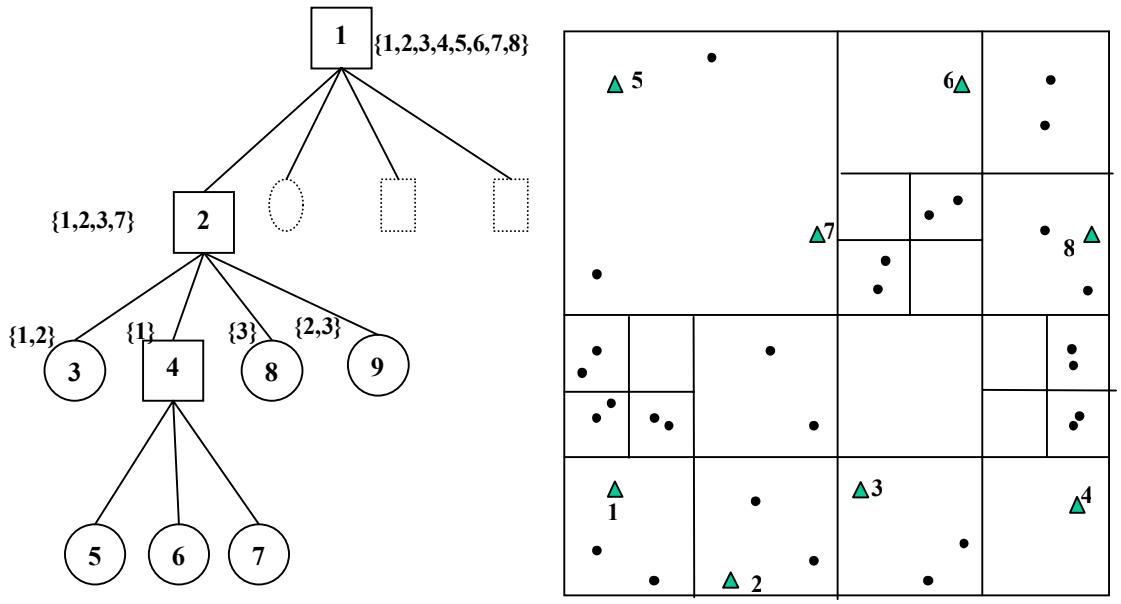


Figure2.3: Example of the working of the pruning function while traversing the pattern tree. Square nodes represent inner nodes, circle nodes represent leaf nodes. The candidate clusters are written within the bracket.

In each iteration, the tree is traversed with *depth first* manner (Figure 2.2) as follows: The traverse of the tree is started at the root node, which has *k* candidate cluster centroids. At the each node of the pattern tree, the pruning function is applied to the candidate cluster set of the node. The pruning function will be explained in the next paragraph in detail. If the candidate cluster set has one cluster centroid, traversal of the tree is not pursued for the children of the node and all the patterns belonging to this node are assigned to the candidate cluster. Otherwise, traversal of the tree is pursued until the leaf node the pattern tree is reached. When a leaf node is reached, the pairwise distance calculations are performed between candidate cluster and patterns of the leaf node like in direct k-means, if the candidate set of clusters contains more than one candidate cluster. But the number of the calculations is possibly less than the calculations performed by the direct k-means, since pruning function eliminates some of the cluster centroids.

The example for the tree-based k-means algorithm is depicted in Figure 2.3. The quadtree of the input space is constructed. The square nodes represent inner nodes of the pattern tree and the circle nodes represent leaf nodes of the pattern tree. The number of the candidate cluster centroids is written near the nodes within a bracket. Initially, there are eight cluster centroids and all of them are candidate cluster for the root node, whose number is one. We started to traverse the tree from the root node and we apply the pruning function. The pruning function prunes different number of the candidate for the nodes so each children of the root node has different number of candidate set. For example, second node (first child of the root) has four candidate cluster centroids, while the fourth node (its second child). Then, the traversal of the tree continues with the second node, the pruning function is applied to the second node, and new candidate cluster set is constructed. Since new candidate cluster set has tree candidates, the standard k-means algorithm is applied between the candidate clusters and the patterns, which belong to second node. Unless, second node is leaf node, we would continue with applying pruning function to the first child of the second node. Then, the traversal of the pattern tree continues with the fourth node, which is second child of the second node. After the pruning function is applied to the seventh node, the candidate set of the seventh function has one candidate, thus all the patterns of the node are assigned to that candidate cluster and traversal of the tree does not pursue the children nodes of the seventh node. In other words, fifth, sixth and seventh nodes are assigned the candidate cluster number one without performing any distance calculation. Traversal of the tree continues with the eighth node. (third child of the second node). The candidate set of the eighth node has one

element after the pruning function is applied. Although, eighth node is a leaf node, all the patterns belong to node will be assigned to the candidate cluster without performing any distance calculation. Traversal goes on top-down manner and traverses all of the nodes of the tree.

```
Function Pruning (node,CandidateSet)
       Newset = CandidateSet
       c_j ∈ (c_1,c_2,.....c_c)          /* centroids
of the Newset */
       P_max , P_min∈ (P_1,P_2,.....P_node) /* patterns of
the node */
       for  each cluster centroid c_j
     max_j = maximum distance between c_j  and P_max
     min_j = minimum distance between c_j  and P_min
       MinMax_dist = MIN(max_k)  (1 ≤ k ≤ c)
       for each c_j do
     if min_j > Minmax_dist then
          Newset – {c_j}
       return (Newset)
```

Figure 2.4 : The pruning function of the tree-based k-means algorithm.

The improvements, which are achieved by the tree-based k-means are seriously rely on obtaining good pruning methods for obtaining less number of candidate cluster centroids for the next level. The pruning method used in Alsabti et.al. algorithm work as follows: For each candidate cluster centroids c_d, the minimum and maximum distances are calculated to any pattern in the cell or subspace. Then determine the minimum distance of the maximum distances (*MinMax*) and eliminate any cluster centroids whose minimum distance is greater than the *MinMax*.

To make *Minmax* clearer, the example of the pruning algorithm is given in Figure 2.5. Let say a node has a subspace in which there are three patterns and has a candidate set with four cluster centroids. In figure, the candidate clusters are represented by squares and circles represent the patterns. The pruning algorithm performs distance calculation between first candidate cluster and all patterns, then determine two distances; one of them is minimum distance between candidate cluster and any pattern in the subspace. The other is maximum distance between candidate cluster and any pattern in the subspace. For example minimum distance for cluster two is distance between cluster two and pattern one. The pruning algorithm finds minimum of the four maximum distances, calls it *MinMax*. In the example, centroid three has the minimum of the maximum distances. In the next step, algorithm compares the all of the minimums of the candidate clusters, if a cluster has a greater distance than the *MinMax,* algorithm prunes it. For example, centroid one and centroid two are pruned, because their minimum distances are greater than the *MinMax.* But, centroid four is not pruned since its minimum distance to any pattern in the subspace is less than the *MinMax*.

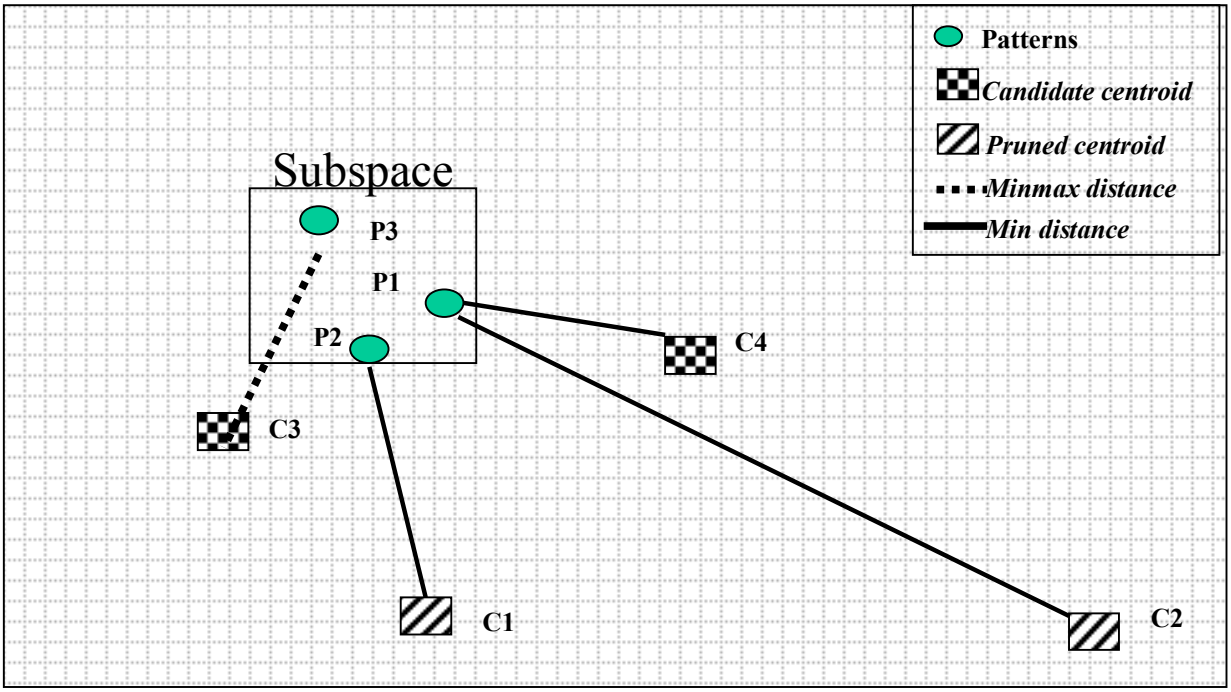


Figure 2.5: The example for the pruning of the candidate cluster centroids.

The pruning function requires distance calculations for the determination of the minimum and the maximum distances of the candidate clusters centroids to any pattern of the cell. Alsabti et al. have shown that maximum distance will be one of the corners of the cell.

Let the $mostdist_j$ be the furthest corner of the cell N_s for the cluster c_j [15]. The coordinates of the $mostdist_j$ $(mostdist_{1j}, mostdist_{2j}, \ldots, mostdist_{dj})$ can be computed as follows:

$$\text{if } |N_d^l - c_{jd}| > |N_d^u - c_{jd}|$$

$$mostdist_{jd} = N_d^l$$

else

$$mostdist_{jd} = N_d^u$$

where N_d^l and N_d^u are the lower and upper coordinates of the cell along the dimension d.

When the $mostdist_{jd}$ is determined we can compute the maximum distance of a candidate cluster as follows:

$$dist = \sqrt{\sum \left(c_{jd} - mostdist_{jd}\right)^2} \qquad (2.2)$$

The value of the minimum distance of the candidate cluster is computed similarly. By using this approach, we do not have to calculate distances between candidate cluster centroids and patterns of the cell. Thus, this approach decreases the number of the distance calculations.

Alsabti et. al. mentioned that this pruning strategy guarantees that no candidate is pruned if it can potentially be closer than any other candidate prototype to a given subspace and the cost of pruning at a node is independent of the number of the patterns in the subspace and can be done efficiently. The results have shown that tree-based k-means is significantly faster than direct k-means. The reader is referred to [15] for the details.

In our implementation, we replaced *k-d tree* with tree-based *quadtree*. The quadtree is a hierarchical data structure based on the principle of recursive decomposition of a space. The quadtree divides space into four equal-sized subspaces and each of which is represented as cells with same level. Each node of the tree has four children and contains the patterns falling in the region of that node. The details of the construction of the quadtree will be given in the next chapter.

2.3 Parallel Direct K-Means

The direct k-means has a problem associated that when the number of the input patterns and the number of the clusters are increased algorithm's computational cost is increasing drastically. There are some approaches to decrease the number of the distance calculations in k-means, which are mentioned before. These approaches [15,27] are directly related with the algorithm of the k-means, they use some geometric constraints to decrease the number of the distance calculations. Another approach can be parallel implementation of the k-means algorithm.

```
Algorithm Parallel k-means (P_n, C_k)
   initialize cluster centroids
   Broadcast cluster centroids
   initialize input patterns /* just for the
first iteration */
   assign the patterns to processors /* just
for the first iteration */
   while (termination condition is satisfied){
      for each local  pattern P_loc_i
          assign pattern to nearest cluster C_d
      for each local cluster copy c_copy_d
         calculate and update centroid
      Reduction on cluster centroids c_k
      compute Error
 }
```

Figure 2.6: Parallel standard k-means clustering algorithm.

In direct k-means algorithm, the main computation work is the distance calculations between patterns and the cluster centroids. It is clear that, for each pattern P_i, k distance calculations are performed. Since the predefined number of the cluster k, is fixed during the iterations, the number of the distance calculations will not be changed from pattern to pattern. In addition he distance calculation between cluster and pattern does not change the location of the cluster for iteration, so that these distance calculation does not depend on each other, completely independent.

In the light of the above explanations, simple and the effective parallelization scheme would be as follows: Total number of the patterns is equally divided to processors in the system and a local copy of the cluster centroids are broadcasted, then each processor performs direct k-means algorithm with its local copy of the cluster and local patterns. Parallel k-means algorithm is sketched in Figure 2.6.

Parallelizing the direct k-means, for distributed memory machines is straightforward due to explained reason. In the parallel direct k-means, each processor is assigned to *n/p* number of patterns, where *n* is the number of patterns and *p* is the number of the processors in the system. The one of the processor, called *root*, broadcasts the cluster centroids to all processors. At each iteration, processors perform single k-means iteration with their local copy of the clusters and their local patterns and update their local copy of the centroids. When the processor completes their computations, the *root* processor performs reduction for the updated local copy of the cluster centroids of the processors, and updates the global centroids, which are the new cluster centroids set for the next iteration. This process continues until predefined termination condition is satisfied.

```
Tree-Based Parallel k-means(P_n, C_k)
   initialize cluster centroids
   Broadcast cluster centroids
   initialize input patterns /*just for the first
iteration*/
   assign the patterns to processors /* just for
the first   iteration*/
   build local tree /* just for the first
iteration*/
  while (termination condition is satisfied){
        traverse tree apply pruning function and
assign
        cluster to closest centroid
        for each local cluster copy c_copy_d
        calculate and update centroid
       Reduction on cluster centroids c_k
       compute Error
 }
```

Figure 2.7: Parallel tree-based k-means clustering algorithm.

At this parallelization scheme, each processor has the same number of the patterns *(n/p)* with a local copy of the cluster centroid. Thus each processor will perform the same number of the distance calculation, which results with balanced computational load, so that one of the important problems of the parallelization is overcome by this parallelization scheme. This parallelization scheme will show almost linearly increasing speedup.

2.4. Parallel Tree-Based K-Means Clustering

The parallelization of the tree-based k-means can be a way to achieve better and faster clustering. But, it is not very straightforward because of the varying computational load balance. There is no so much difference between the parallel tree-based k-means and direct k-means. The most of the steps of the algorithms are same, just pruning function is added to the parallel direct k-means. The algorithm will be explained in details in the next paragraphs.

In parallel tree-based k-means, number of the patterns, assigned to the processors is determined by the pattern decomposition functions, which will be explained in the next section. Then, one of the processors; called *root*, broadcasts the local copy of the cluster centroids to processors. The processors build their local pattern tree with their local patterns. While each processor is traversing its local pattern tree, it is also applying pruning function. This process continues until predefined termination condition is satisfied. When all of the processors finished their job, the root processor performs reduction of the updated local copy of the entire processors and updates global cluster centroids, which are the new centroid set of the next iteration. Figure 2.7 gives the parallel tree-based k-means algorithm.

When we compare the parallelization of the direct k-means and the tree-based k-means, we can easily obtain that parallelization of the tree-based k-means is more challenging because of the varying computational load. The tree-based algorithm performance is directly related with the performance of the pruning, since this pruning activity is changing according to some constraints such as size of the subspace, pattern density of the subspace etc., computational work load depends on the pruning activity is also varying from subspace to subspace.

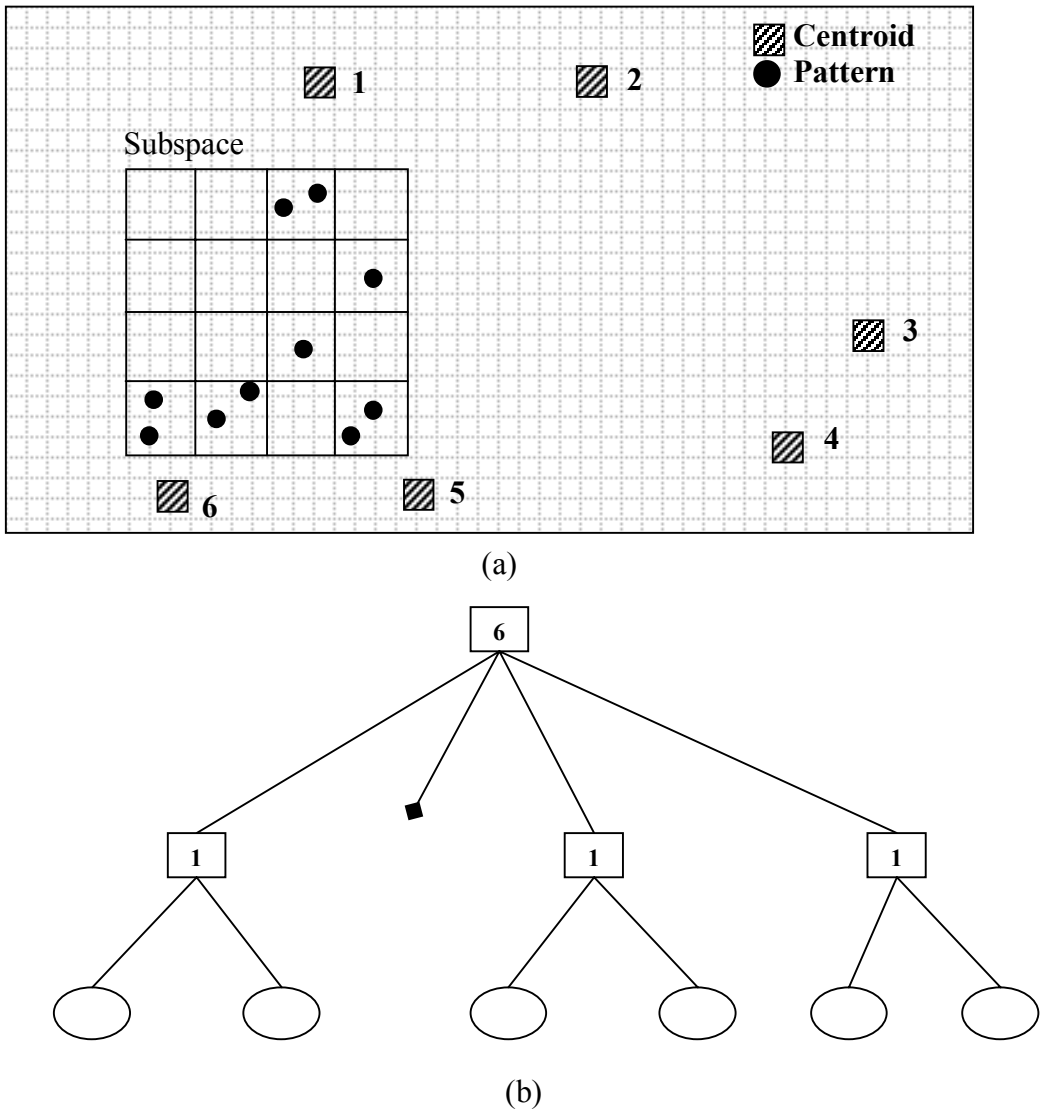


Figure 2.8: (a) The example of the effect of the compact subspace to the tree-based k-means algorithm. Subspace is assigned to a processors. There will be more pruning at the upper level of the pattern tree.(b) The quatree of the subspace. The number of the candidate clusters is written inside the node

The computations during traversal of the pattern tree can be divided into two groups [2]: the internal-node computations and the leaf-node computations. In the internal-node computations, computations for the pruning of the candidate clusters are done. Since some of the candidate cluster centroids might have been pruned upper levels of the pattern tree, the distance calculations can vary across the internal nodes. Similarly, as the number of the candidate set of the clusters and the number of the patterns are different in each cell, this results with difference in the number of the distance calculation and varying computational load. The leaf node computations are the distance calculations between patterns and candidate cluster centroids.

The tree-based k-means performance is directly related with the performance of the pruning algorithm like parallel version of the algorithm. When the number of the pruned candidate clusters and the patterns are increased performance of the algorithm also increases. Therefore, in order to increase the performance of the algorithm, pruning performance of the algorithm must be increased. For example, consider two processors with same number of patterns one with smaller subspace as shown in Figure 2.8 and another with larger subspace as shown in Figure 2.9. Figure 2.8-(b) represents the quadtree of the subspace, which is shown in the Figure 2.8-(a), and the number of candidate cluster centroids is written in the node

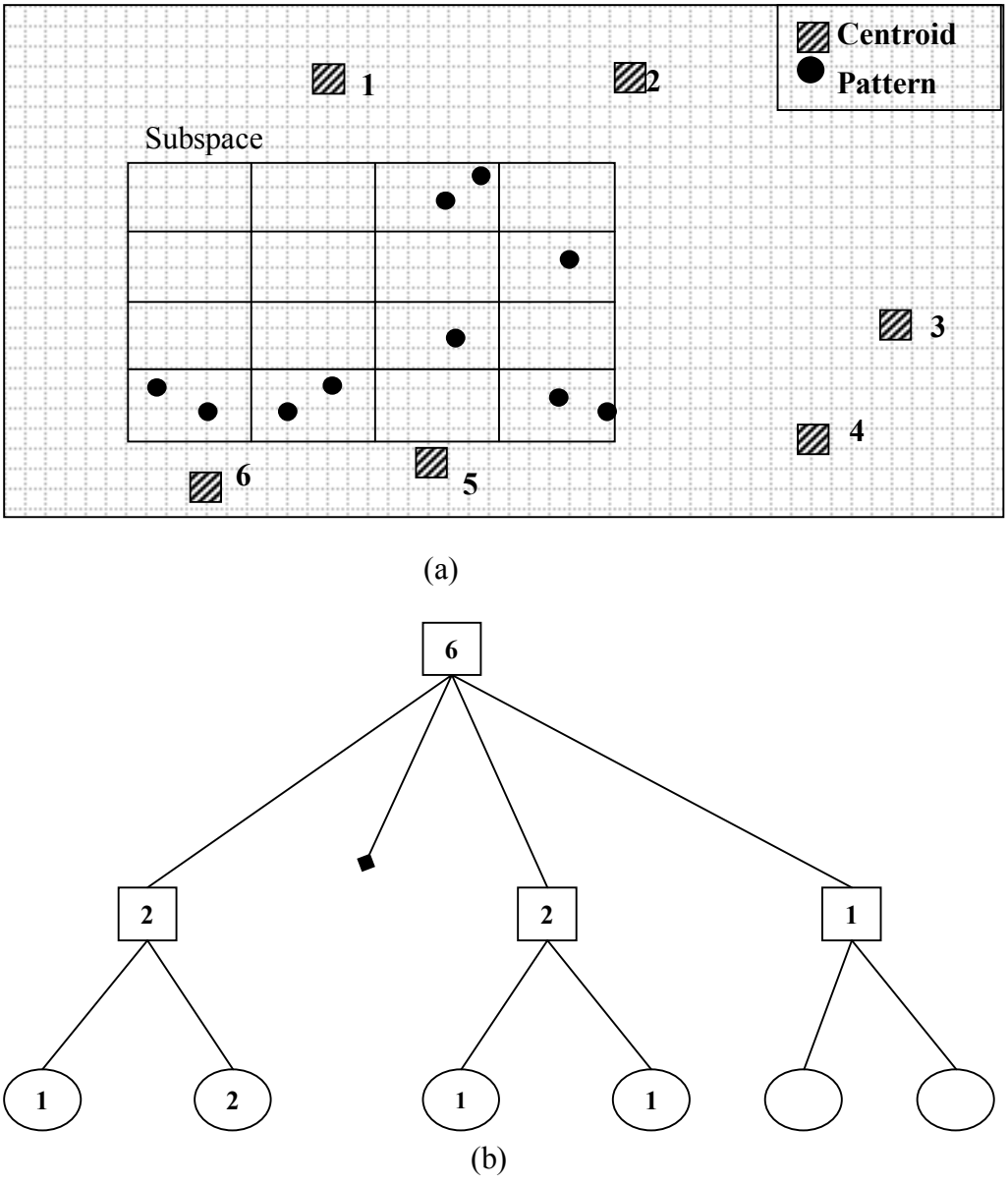


Figure 2.9: (a) The example of the effect of the larger subspace to the tree-based k-means algorithm. The larger subspace is assigned to a processors. In this case there will be less pruning at the upper level of the pattern tree. (b) The corresponding quatree of the subspace. The number of the candidate clusters is written inside the node.

The first processor starts traverse tree from the root node. The traversal of the tree continues with the first child of the root, when the pruning function is applied, five of the candidate clusters centroids (1,2,3,4,5) are pruned, because none of their minimum distances, between them and any pattern in the node are smaller than the *Minmax* distance. Thus, the first child of the root node has the candidate set with a one element, and all of the patterns of the first child node will be assigned to sixth cluster centroid. The first processor continues to traverse tree with the third child. The pruning function is applied to candidate set of cluster, and the five of the six candidate cluster centroids (2,3,4,5,6) are pruned. Since, the third child's candidate cluster set has one element, all of the patterns are assigned to cluster one.

Figure 2.9-(b) represents the quadtree of the larger subspace, which is shown in the Figure 2.9-(a). The second processor is assigned the larger subspaces shown in the Figure 2.9.a. It starts the traverse the tree, with the root node. Then it continues the traverse with first child of the root node. The pruning function is applied to the node and four of the six candidate cluster centroids(1,2,3,4) are pruned. The traversal goes on with the first child of the first child, again the pruning function is applied and all the patterns are assigned to sixth cluster.

Then the next node is second child of the first child. After the pruning function is applied, two cluster centroids are left in the candidate set of the second child. Since the second child is the leaf node, the processor performs distance calculations between candidate clusters and the patterns of the node. The traversal of the tree goes on until the all nodes of the tree are visited. As we explained in the example the first processor who has smaller subspace performed less number of the distance calculations, most of the candidate cluster centroids are pruned at the first level of the tree. However, the second processor, which has same number of patterns but larger area, performed higher number of distance calculations, because most of the candidate clusters of it are not pruned at level one.

As a result, the more pruning of the candidate cluster centroids and patterns at the upper level of the pattern tree will result in the less distance calculations so that a processor with a smaller subspace will perform less number of distance calculation. In the case of larger subspace, the pruning performance is decreased, this will cause more distance calculations. In the light of the above explanations, if we assign compact subspaces to

processors, the number of the distance calculations is decreased, and the performance of the parallel tree-based k-means is increased.

Pattern decomposition is important factor of increasing performance of the parallel tree-based k-means algorithm. To achieve better performance, pattern decomposition techniques must be handled very carefully, so we have examined and proposed new pattern decomposition technique. The decomposition techniques of the parallel tree-based k-means algorithms are explained in the next chapter.

CHAPTER 3

PATTERN DECOMPOSITION

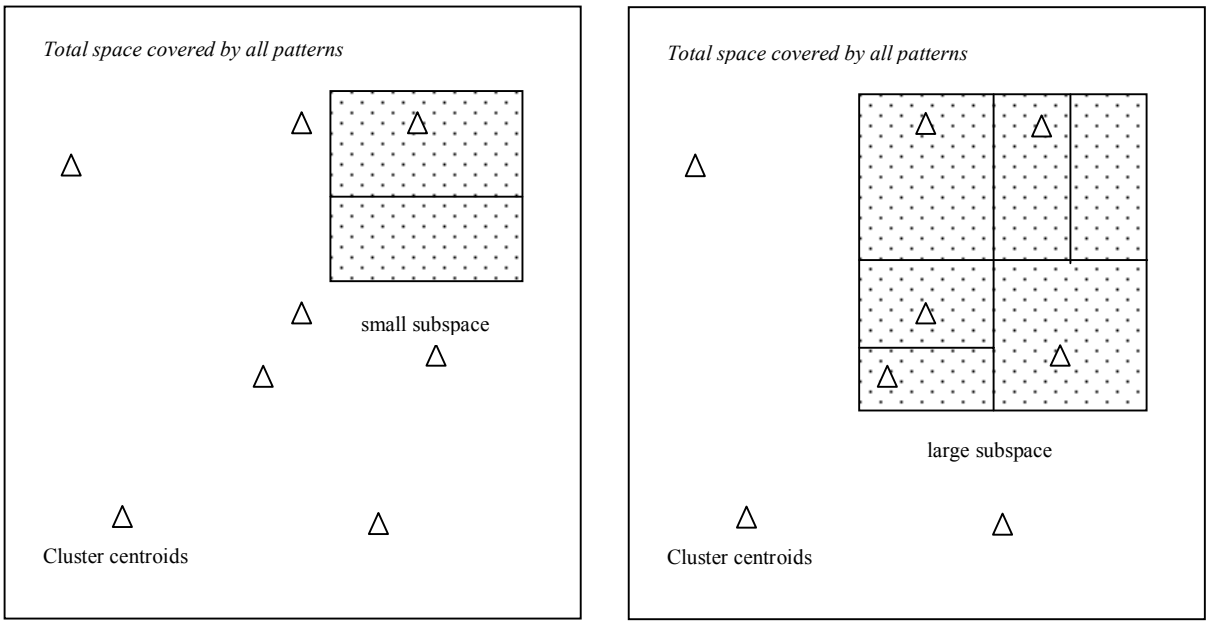
Parallelizing tree-based k-means is not straightforward like standard k-means because of the parallelization scheme problems. The structure of the tree-based k-means is sensitive to load balancing. The pruning technique employed causes this sensitivity. Each processor's pruning performance is varying according to its local patterns distribution. If patterns are distributed in a small space, there will be more pruning of candidate set of clusters at the upper levels of the tree, so leaf computations will be decreased. In the case of sparse distribution, there will be less pruning of candidate set of clusters at the upper levels of the tree, and then there will be more leaf computations than the small space distributed case. As a result, processor whose local patterns are distributed in a small space will have less computation load than the processor whose local patterns are sparse distributed (Figure 3.1).

In the light of the above explanations, we have implemented three different pattern decomposition techniques of parallel tree based k-means. These decomposition techniques are, *random* pattern decomposition, *stripwise* pattern decomposition and *tree-based* pattern decomposition. All of the decomposition technique will be explained in detail.

3.1 Random Pattern Decomposition

This decomposition technique is similar to technique used in parallel direct k-means. Each processor is assigned *n/p* local patterns; each of these patterns has been chosen randomly. It is mostly probable that, local patterns of each processor are distributed in a sparse manner. This sparse distribution affects the performance of the pruning algorithm, in other words, performance of the tree-based k-means clustering algorithm.

Patterns are distributed among processors in the system as follows; The root processor randomly chooses *n/p* patterns for each processor without looking their location or without paying attention whether they are distributed in a small space or not. The root processor only controls that pattern must be sent only one processor in the system. Then, root processor sends *p* pattern packed, each packed contains *n/p* patterns, to processors in the system.



(a) (b)

Figure 3.1: The effect of the size of the assigned subspaces: (a) The smaller subspaces. There will be more pruning of candidate set of clusters centroids. (b) The larger subspaces. there will be less pruning of candidate set of cluster centroids, pruning will shift to leaves. Although, both processor have the same number of patterns (b) will do more calculations.

It is probable that every processor is assigned larger subspace, and many of the cluster centroids will be related with this subspace, so performance of the pruning algorithm will decrease. The pruning might shift towards to the leaves; this shifting might result in more distance calculation.

With this technique, we are expecting to have a balanced computational workload. Because, all of the processors have subspaces with almost equal sizes and have exactly same number of patterns. Also, *random* decomposition technique is easy to implement and it does not need so much preprocessing work, which has done over the input pattern. However, each processor will have larger subspace (see Figure 3.1) so the pruning algorithm performs less pruning of candidate clusters at the upper level, and the number of the leaf computations will be increase, but it never reaches the number of the distance calculations performed by parallel direct k-means. The pruning advantage of the tree-based k-means is not used very much by *random* pattern decomposition technique.

3.2 Stripwise Pattern Decomposition

If a processor is assigned to a compact subspace, we expect that, processor will perform better pruning activity, and the number of the leaf computations will decrease and then the performance of the algorithm will increase. So, we need a scheme that distributes more compact space to each processor.

Stripwise pattern decomposition technique distributes *n/p* patterns to each processor, but each of *n/p* pattern set is belong to a more compact space when we compare with the *random* pattern decomposition. This set of patterns that are concentrated in small space, ensures better pruning. Thus, the main goal of the tree-based parallel k-means algorithm is achieved.

In this technique, total space, covered by all patterns, is divided into the strips. All input patterns are sorted according to the one of the member of its feature vector space. The root processor divides sorted patterns to *p* strips each of which contains *n/p* patterns. These strips are distributed to each processor. Each processor performs algorithm on these stripped set of patterns, which are concentrated on a small space.

We know that input patterns are distributed randomly. Their concentration is not uniform on the space, which is covered by all patterns, so that, pattern concentration of the stripes also is not uniform. This nonuniformity causes load imbalance. For example consider two processors, one with a pattern strip whose patterns are concentrated in a small space, and another one with a pattern strip, which has same number of patterns but patterns are less concentrated than the other. The first processor will prune many of the candidate cluster centroids at the upper level of the local tree. Because of the pruning algorithm specialty, some of the local patterns are assigned to a cluster without doing any distance calculation. Second processor will not prune many of the candidate cluster centroids, so number of the leaf computations will increase and the number of the tree-pruned patterns, which are assigned to a cluster without a leaf computation, is also increased. In other words, first processor will do less distance calculations than the second processor because first processor's pattern strip is more compact than the other one.

In this decomposition technique we are expect to have less number of leaf computations, because stripwise decomposition will ensure better pruning. Since we will have less number of leaf computations and more tree pruned patterns, performance of the algorithm will be better than the algorithm whose assigned patterns are determined by the random pattern decomposition technique. But, the different concentration of the patterns on a strip will cause varying computations loads for each patterns.

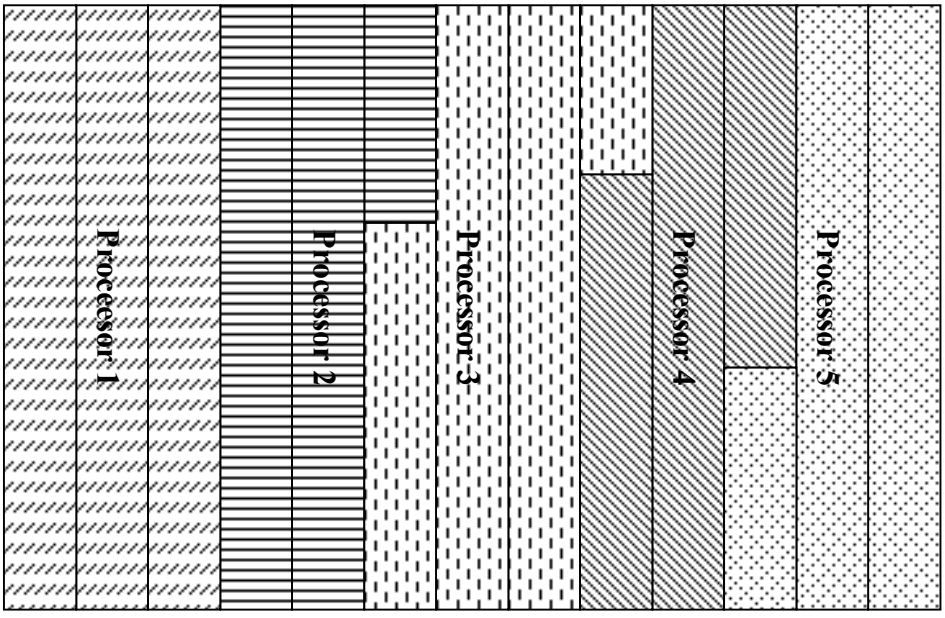


Figure 3.2 : Strip decomposition of patterns . All processor have the same number of patterns but size of the subspaces are different from each other.

3.3 Tree-Based Decomposition of the Patterns

The *random* decomposition technique and *stripwise* decomposition technique have decomposed space, which is covered by all patterns, to subspaces by ignoring concentration of the patterns on these subspaces. They have just taken into account assigned number of patterns for each processor. They have tried to distribute equal number of patterns to each processor.

Size of the subspaces, assigned each processor, is determined by two factors. One of them is number of patterns assigned to each processor, and the other is concentration of patterns. Since each processor has equal number of patterns *(n/p)*, the dominant factor in determination of the size of the compact subspace is concentration of the patterns on that subspace. If the patterns are scattered in a small space, it means that the pattern concentration is high, it is clear that subspace will be compact. But the same number of patterns are scattered in a larger space then the subspace will invade larger area. The processor, whose subspace is highly concentrated, will perform more pruning, and less distance computations. But the other processor, whose subspace is not highly concentrated, will perform less pruning, and more distance calculations. In other words, the processor, whose subspace is compact area, will do less distance calculations than the processor whose subspace is larger. This situation is lead to load imbalance. This situation will be explained in detail in the next section.

To overcome load imbalance hierarchical representations of the physical space can be used. Tree structure is used for hierarchical representation of physical space, in most of the classical problems, such as n-body methods [23]. If we construct a pattern tree and then distribute this hierarchical tree structure, we are expecting to have improvements in the performance of the tree-based parallel k-means clustering algorithm.

In proposed decomposition technique, the input patterns are organized in a tree structure. We have already used the tree in tree-based k-means algorithm. The pattern tree is built by recursively subdividing space cells until predefined termination conditions, such as number of patterns per cell, are satisfied. In two-dimensional patterns, patterns are organized as quadtree structure [24], in which a subdivision divides cell area into four equal rectangular and each of cell has four children. In three-dimensional patterns, patterns are organized as

octree structure, whose cells have eight children and each subdivision operation divides cell into eight cubes.

Let us now describe the tree construction in some details. First of all, the positions of the patterns are used to determine the dimensions of the root cell of the tree. Then, tree is built by adding all patterns into the initially empty root. In our implementation, there are two termination conditions, predefined number of pattern per cell and predefined depth of the pattern tree. The root cell is controlled whether it is achieved to termination condition or not. If the root's assigned number of patterns exceeds the predefined number of patterns per cell, then subdivision operation is started. The root cell is divided into four children (for the two-dimensional patterns, in case of three dimensional input children number will be eight.). Root cell's patterns inserted into its children cell according to their location in the physical space. The subdivision operation is recursively applied to all children cell, until predefined termination condition is satisfied. In our implementation, depth of the tree is controlled at first, if this termination condition is satisfied then tree construction is terminated without controlling other termination condition. Otherwise, depth of the tree will be very high, which causes inefficient use of memory space and time. The result is a tree whose internal nodes are space cells and whose leaves are patterns. Because of the randomly scattered patterns, some of the cells might be empty, which are deleted. The tree is adaptive in that it extends to more levels in regions that have high pattern concentration. Figure 3.3 shows a small two-dimensional example domain and the corresponding quadtree.

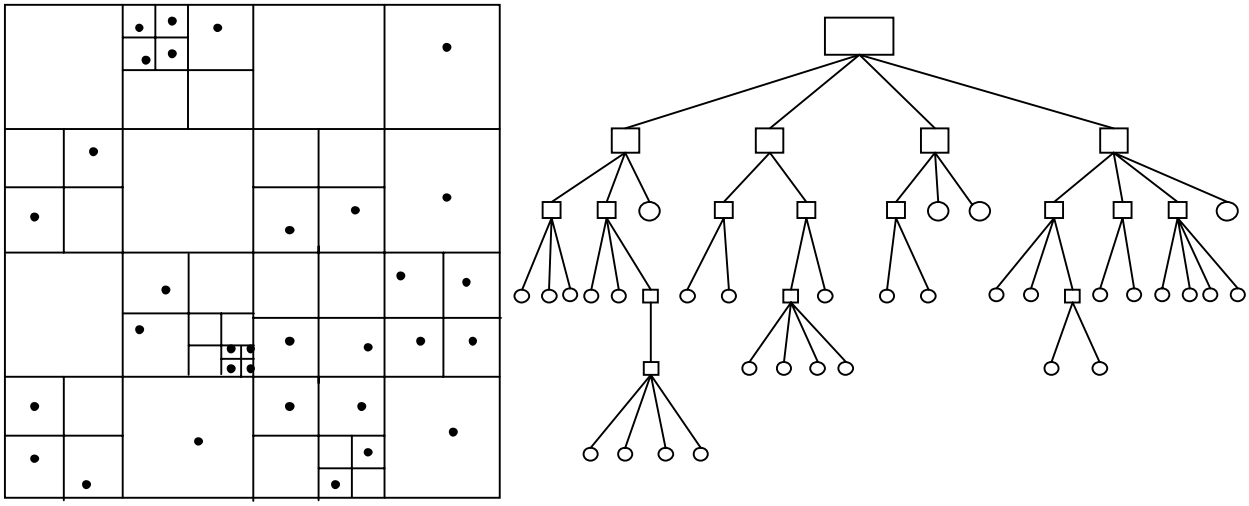


Figure 3.3: A two-dimensional pattern distribution and the corresponding quadtree.

After the pattern tree is built, next step will be partitioning the tree. This technique is very similar to *costzones* partitioning technique, which was discussed in the parallel n-body implementations.[24]. We already have a hierarchical representation of spatial distribution of input patterns, so we will use this advantage by using *costzones* partitioning techniques. Therefore, we will partition tree rather than partition the space and each processor is assigned to a smaller subspace. Partitioning procedure can be explained as follows: The root processor calculates the total cost of the domain with a chosen cost function by traversing the pattern tree. Each processor equally shares total cost in the system. After average cost for each processor is determined, which is equal to division of the total cost with the number of the processors in the system, the root processor distributes leaf cells of the tree. Patterns distribution policy is determined by the cost functions. Patterns may be distributed by either pattern-by-pattern or cell-by-cell. The leaf cell distribution of the pattern tree is lead to have each processor's assigned patterns scattered to a compact space, which is our aim. The *costzones* partitioning scheme is illustrated in Figure 3.4.

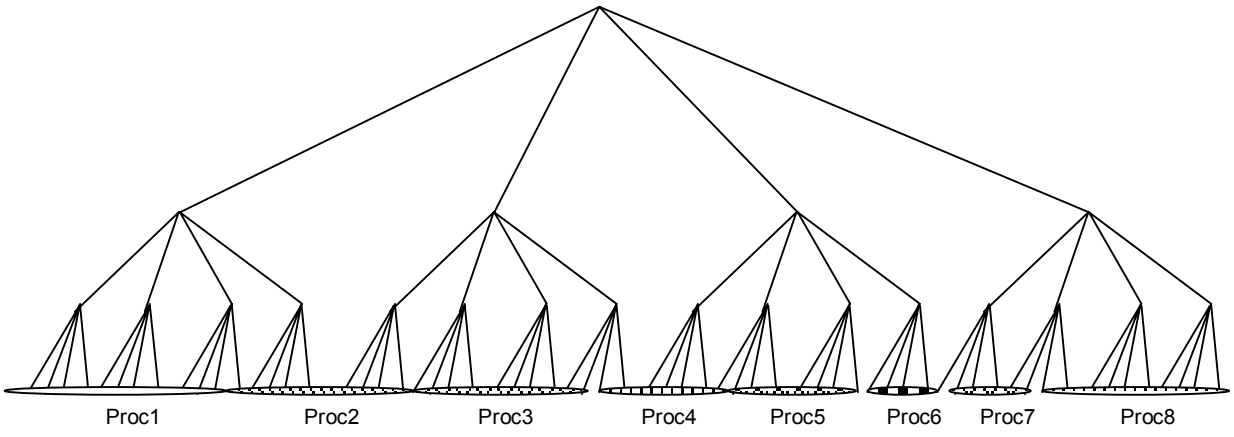


Figure 3.4 : Pattern tree partitioning scheme ([24]).

The main workload is directly proportional to the two factors, which are number of patterns and the number of cluster centroids. The predefined number of the cluster centroids is not varying throughout the clustering operation, so that, each input pattern has equal workload. The total workload is shared among processors by assigning different or equal number of patterns to each processor. Thus, cost functions, which are used to calculate total cost of the clustering operation, decides the number of patterns to be assigned for every

processor in the system. The cost functions will be explained in the following subsections in details.

3.3.1 Cost Functions

3.3.1.1 Simple Cost Function (SCF)

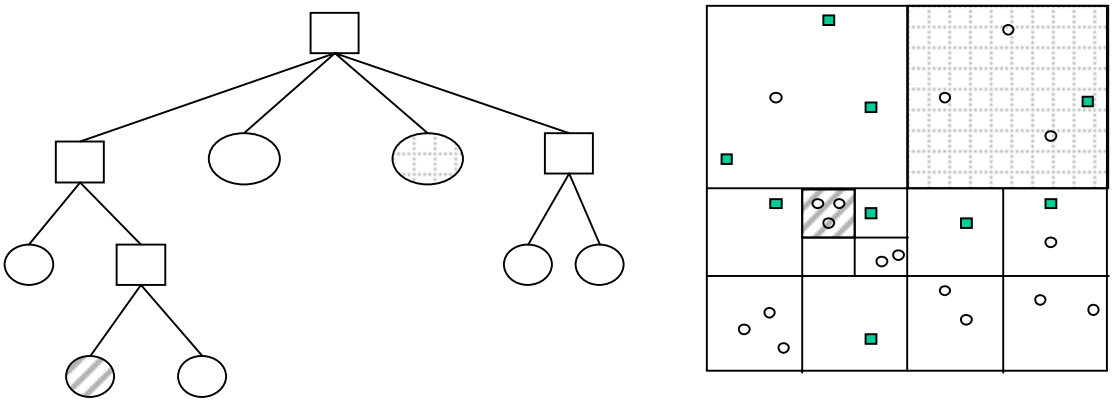
The simple cost function is straightforward. It assumes that, each pattern is associated with an equal workload in the each iteration of the algorithm. The input number of patterns is equal to total cost. The global average cost is calculated by dividing the total cost to number of the processor in the system. In other words, simple cost function assigns equal number of patterns to each processor. It can be formalized as follows:

$$TotalCost = TotalNumberofInputPatterns \quad (3.1)$$

$$AverageCost = TotalCost / numberofprocessor \quad (3.2)$$

3.3.1.2 Level Cost Function (LCF)

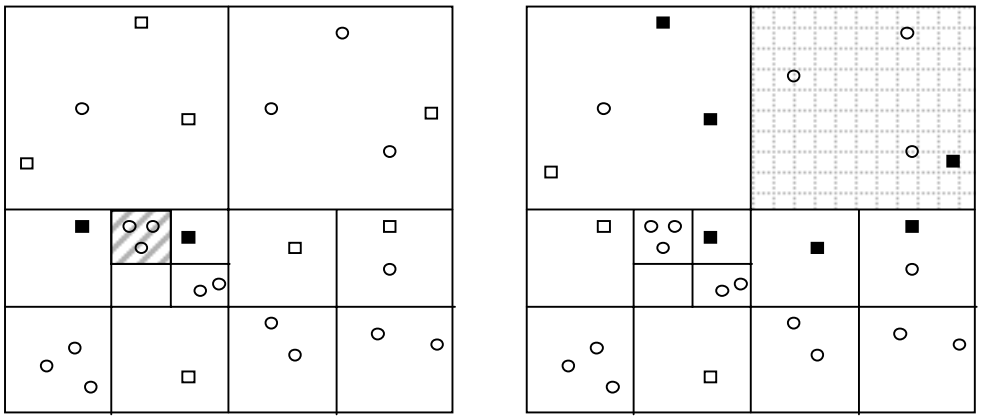
Simple cost function depends on the assumption that, each pattern is associated with equal amount of work at the every iteration of the algorithm. However, tree-based parallel k-means algorithm performs cluster centroids pruning, and each pattern might have different amount of workload, even no workload, at the every iteration of the algorithm. So, there is a necessity for a cost function, in which the pruning specialty of the tree-based parallel k-means algorithm is not ignored.



(a) (b)

Figure 3.5 : The hierarchical representation (a) of the physical distribution of the patterns (b). Patterns are represented by circles and centroids are represented by squares. As the depth of the tree increases, physical representatin of the nodes are getting smaller.

The pattern tree, which is the hierarchical representation of the physical distribution of the patterns, is an adaptive in a way that, the height of the tree is related with the concentration of the patterns. For example, in the Figure 3.5-(a) we have two subspaces, one of them more concentrated than the other subspace, which has eight patters and is located in bottom left of the working space. The less concentrated subspace has three patterns and is located in the upper right of the part of the working space. The more concentrated subspace is represented in the deeper level of the pattern tree, with smaller cells as shown in the Figure 3.5-(b) The less concentrated subspace is represented in the upper level of the pattern tree, with bigger cells as shown in the Figure 3.5-(b). The pattern tree extends more levels in the subspaces where pattern concentration is high. As the depth of the tree is increasing, tree cells represent smaller area. Therefore, the height of the pattern tree can be considered as a parameter of the determination of the cost of a cell.



(a) (b)

Figure 3.6 : The pruning rate of the two different cells, each of which has different height. The pruned clusters are represented by empty squares, alive clusters are represented with filled squares.

The height of a cell can affect the cost of it as follows: In the Figure 3.6, physical subspaces of the two different cells are depicted. The signed area, in Figure 3.6-(a), corresponds to a leaf cell, which is shown in Figure 3.5-(b), whose level is three. When the cost function is applied to the cell, it prunes the seven of the nine cluster centroids, because patterns of the cell are scattered in a small physical subspace, and most of the cluster centroids are located far from them. At the end of the pruning process, the candidate cluster set of the cell will have two candidate clusters. On the other hand, the height of the second cell is one, which is the third child of the root (Figure 3.5-(b)), and its subspace is shown in

the Figure 3.6-(b). When the pruning function is applied to the cell, only three of the nine cluster centroids are pruned (Figure 3.6-(b)). After the pruning function is applied, the candidate set of the cluster will have six candidate cluster centroids. Although, these two cells have the same number of the patterns, the number of the distance calculations will be different. The first cell will be associated with less number of the distance calculations so that costs of the two cells will be different. The deeper cell will have smaller cost.

As a result, the deeper cells might be associated with less number of distance calculations when it is compared the higher level cells. As the height of a cell increases, the cost of the cell decreases. Therefore, we can include the depth of a cell as a parameter to the cost function. Level cost function can be formalized as follows:

$$TotalCost = \sum^{for-all-leafCell} numofpatterns \times (1 + (cons / levelofcell)) \quad (3.3)$$

$$AverageCost = TotalCost / numofprocessors \quad (3.4)$$

$$CellCost = numofpatterns \times (1 + (cons / levelofcell)) \quad (3.5)$$

The root processor calculates the total cost, which is represented by Equation 3.3, by traversing leaf cells of the pattern tree. After the total cost of the pattern tree is calculated, global average cost is determined by using Equation 3.4.At the end of the cost calculation process, the root processor begins to traverse pattern tree, started at smallest numbered leaf cell, each leaf cell's cost is calculated by using Equation 3.5, and then patterns included in that cell are put into a buffer, then the total cost of this buffer is controlled, if the total cost of the buffered patterns is equal or more than the average cost, then all of patterns included in the buffer, are send to a processor in the system. The buffering is used to minimize communication overhead between processors. This process, until all of the leaf cells of the pattern tree are distributed, is repeated.

Since all of the patterns are associated with different amount of work, each processor is assigned different number of patterns. We are expecting to overcome some of the disadvantages of the simple cost function.

3.3.1.3 Centroid Cost Function (CCF)

The basic computational load of the k-means algorithm is the distance calculation between cluster centroid and pattern. The candidate cluster centroids are the basic source of the pattern's computational load. The tree-based decomposition technique organizes input patterns in a quadtree structure, so if we determine how many of the input cluster centroids are candidate for a cell, thus we can easily determine the cost of a cell. Then the distributions of the cell can be made according to this predicted cost.

It is known that, in the tree-based decomposition technique, the root processor organizes the input patterns in quadtree structure. Then the root processor inserts the initial cluster centroids one by one to the pattern tree like input patterns. All of the input clusters are candidate for the root of the tree, as we traverse the tree, candidate set of the cluster centroid are inserted to inner nodes according to geometric constraints recursively. When the leaf node is reached, number of the candidate set of cell is determined. If the distance between center of cell and the candidate cluster centroid is greater than the size of the cell, the cluster centroid is considered to be the candidate cluster centroid. If the none of the cluster centroid satisfies the this condition, the distance between the center of cell and the closest cluster centroid is measured and all of the distances between center of cell and the candidate cluster are compared, if there is a cluster centroid whose distance is equal to the closest centroid, number of the candidate centroid is increased.

$$TotalCost = \sum^{forall\_leafcell} numofpattern \times (numofcandidate\_cluster) \qquad (3.6)$$

$$LocalCost = TotalCost / numofprocessors \qquad (3.7)$$

$$CellCost = numofpatterns \times (numofcandidate\_centroid) \qquad (3.8)$$

We can summarize Centroid Cost Function as follows: The root processor builds global pattern tree and inserts the candidate cluster centroid to pattern tree. Total cost of the pattern tree is calculated with Equation 3.6 by traversing leaf nodes of the tree. The global average cost is calculated by using Equation 3.7. At last, the root processor traverse global pattern tree, for each leaf cell, cost of cell is calculated by Equation 3.8 and the average cost of a target processor is controlled, if its cost less than the global average cost, leaf cell is assigned to target processor. The success of the centroid cost function is depends on the success of the candidate cluster centroids.

3.3.1.4 Centroid and Level Cost Function (CLCF)

The previous cost functions take into account just one parameter either level of the pattern tree or the number of candidate cluster centroid. But we thought that, if a function cares both level of the pattern tree and the number of the candidate cluster centroids, predictions about cost of a cell would be more reliable, so we can get better results.

$$TotalCost = \sum^{forall\_leafcell} numofpattern \times (numofcandidate\_clusters) \times (1 + const / levelofcell) \quad (3.10)$$

$$CellCost = numofpatterns \times (numofcandidate\_clusters) \times (1 + const / levelofcell) \quad (3.11)$$

Centroid Level Cost function is combination of the *CCF* and *LCF*, so that we will not explain the details of the implementation. It calculates the total cost of the pattern tree by using Equation 3.10. Then the global average cost is defined by using Equation 3.8. The root processor calculates the cost of each cell by using Equation 3.11 and distributes leaf cell. The success of the function is directly related with the initial centroids as *CCF*.

3.3.2. Child Numbering Issues of the Tree

We have two cell numbering techniques, uniform numbering of cells and the non-uniform numbering of cells. In the uniform numbering of cells, every cell's children are numbered in the same manner (starting with same corner, with same direction). In the non-uniform numbering of cells, different numbering manner is implemented for every child of a cell (starting point and direction is decided according to some parameters. These two techniques are explained in the next section.

3.3.2.1 Uniform Numbering of Cells

Tree decomposition technique partitions the pattern tree rather than space, which yields subspaces that are contiguous as laid out in the plane. The quality of this contiguity depends on how the locations of cells in the tree map to their locations in the physical space. In other words, how well this contiguity directly depends on how well the ordering scheme, in which the children of the cells are numbered. The simplest ordered scheme of the children is the same ordering for every cell of the tree. This numbering scheme is called uniform numbering. The uniform numbering is also most efficient scheme for determining which child of a given cell particle falls into. However, the most important disadvantage is that there is no single

ordering scheme that ensures contiguity in the planarized pattern tree will always correspond to contiguity in the physical space.

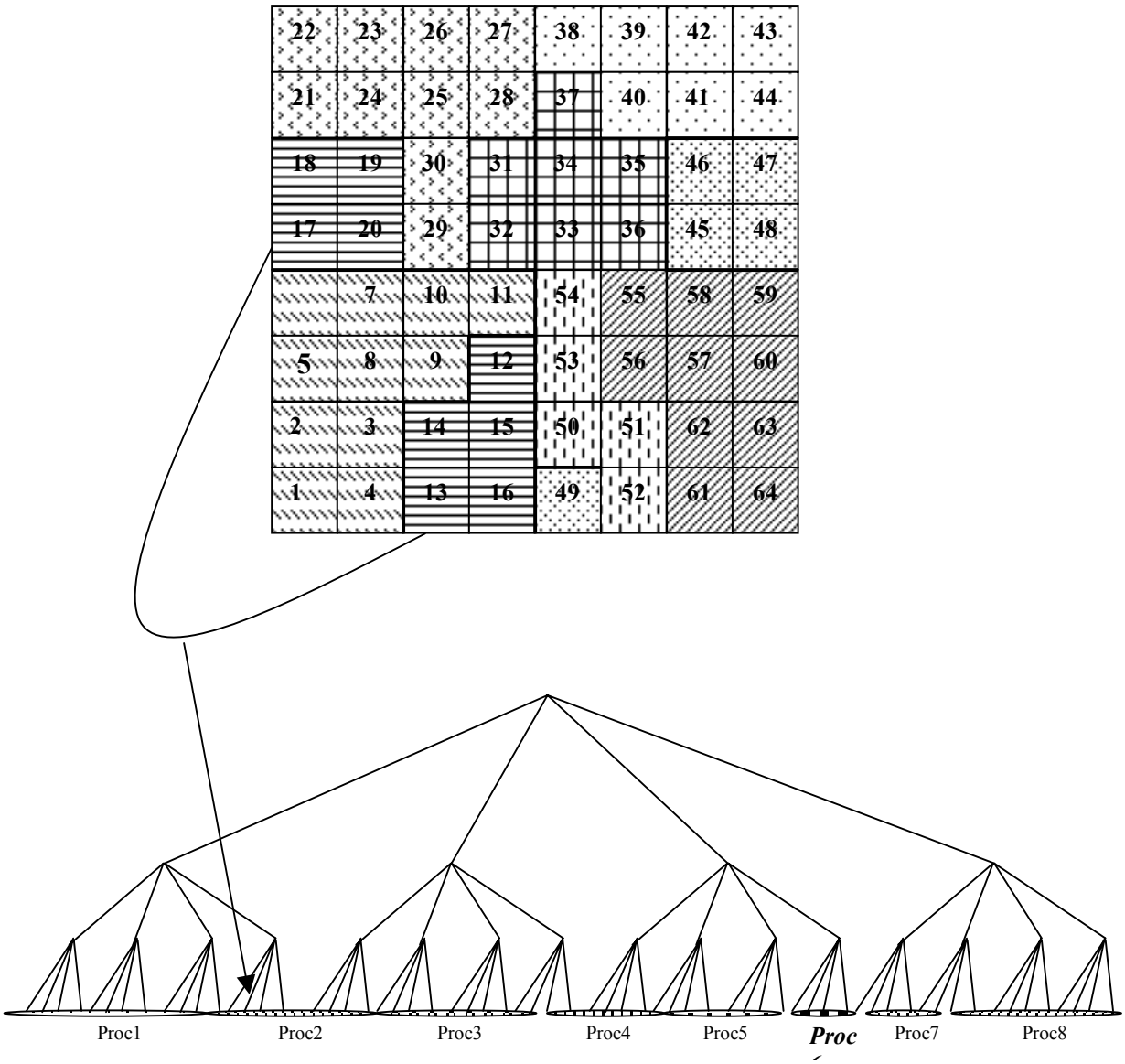


Figure 3.7:Tree decomposition with uniform numbered children. ([23])

Figure 3.7 illustrates that some of the subspaces that are assigned to processors are lack of physical contiguity. The uniform numbering of the children of the pattern tree causes this discontinuity. As you can see in the Figure 3.7, children are numbered clockwise, starting from the bottom left child. Although, some of the leaf cells are represented next to each other in the planarized pattern tree, they may not be near each other, if their ancestors are not same. For example, *processor2* might do more distance calculations than *processor1*, since *processor2* must deal with two subspaces, which are apart from each other. This apartness of

the subspaces will decrease the performance of the pruning algorithm. As a result, uniform numbering of children of the pattern tree, may decrease performance of the tree-based parallel k-means algorithm.

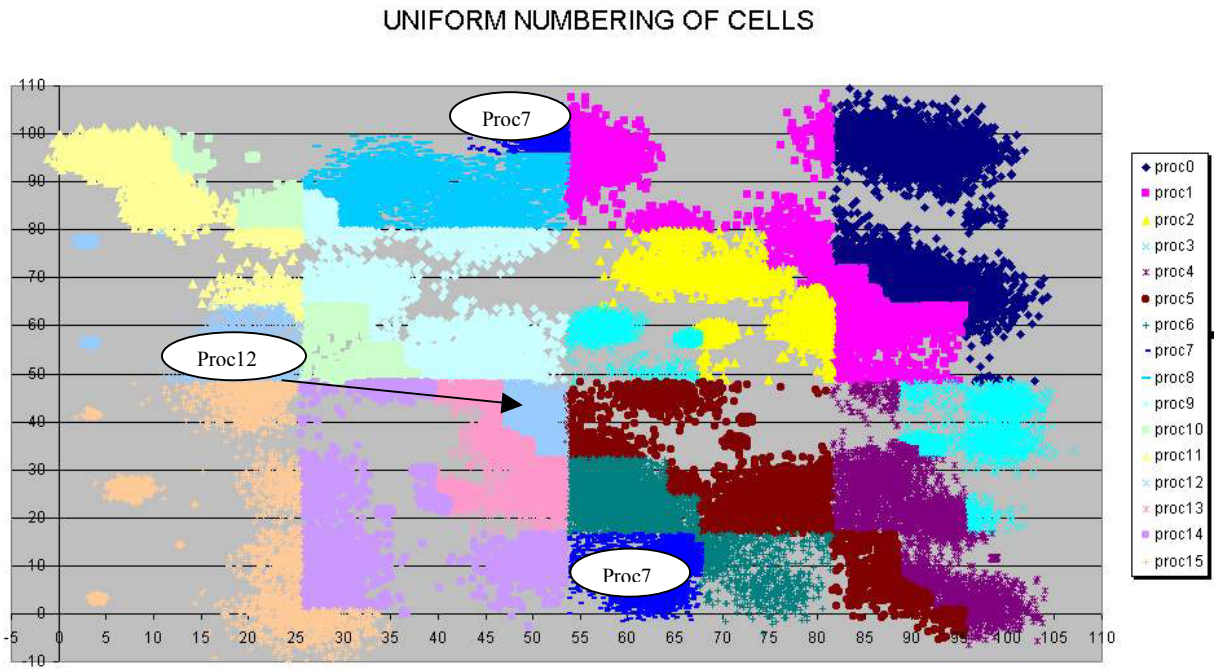


Figure 3.8: 100,000 patterns are distributed among 16 processors by uniform numbering of cells. Some of the subspaces are not contiguous.

In Figure 3.8, 100,000 patterns are distributed among 16 processors by using uniform cell numbering technique. This technique yields subspaces, which are assigned to processors that are not contiguous. For example, *processor7* is assigned two subspaces, these subspaces constructed by distributing leaf cell of the tree. Although, assigned cells to processors is next to each other, physical subspaces, which are represented by assigned leaf cells, are not next to each other. The two apart subspaces of the *processor7* directly decrease performance of the pruning algorithm, and also performance of the tree-based parallel k-means algorithm.

3.3.2.2 Non-Uniform Numbering of Cells

It is explained that uniform numbering of children of the pattern tree causes discontinuity in the subspaces, which decrement performance of the algorithm. The simple solution for this situation is presented in [23]. In this solution ordering scheme of the children is not same for all cell so that we can call this scheme as non-uniform ordering. And also it makes contiguity in the planarized pattern tree correspond to contiguity in space. In this numbering scheme, ordering of a cell *C* is determined by two factors: (i) ordering style of the children of the cell's parent (ii) and which child of its parent *C* is in that ordering scheme.

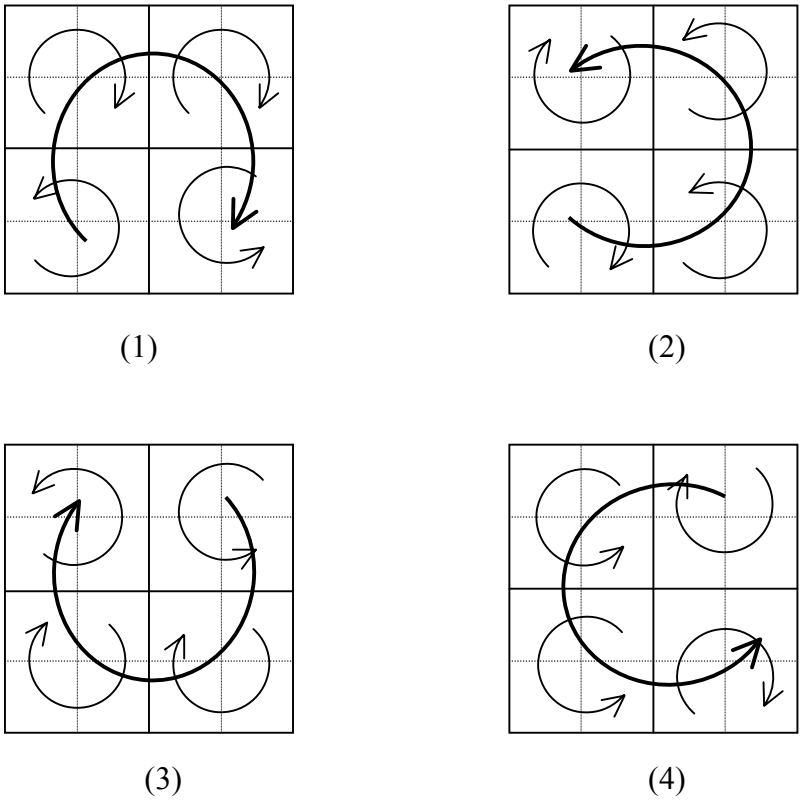


Figure 3.9 : Ordering scheme of cells.

To give details let us give a two-dimensional example. Since, input patterns are two-dimensional, their corresponding pattern tree is a quadtree, which has four children for every cell. There are eight different ordering ways to order siblings of a cell: two possible directions, with clockwise and counterclockwise, and four possible starting points (as cell has four children). However, four of the eight ways are sufficient to our implementation. Figure 3.9 shows four ordering scheme that we use in our implementation and illustrates how the ordering of a child is determined by the ordering for its parent. An arrow represents the ordering of a cell. The big arrow represents cell's parent's ordering direction. The smaller arrows show the cells own ordering direction, according to its number. Cell, which is ordering his children, first controls its parent's ordering scheme then determines its number according to these two parameters, it decides his children ordering scheme. For example, target cell's parent is ordered in clockwise fashion starting with left-bottom (Figure 3.9-(1)), and target cell is fourth child of its parent, then the target cell ordering scheme will be started at the upper-right and will number its children through counterclockwise direction. The other ordering determination possibilities can be determined by using Figure 3.9.

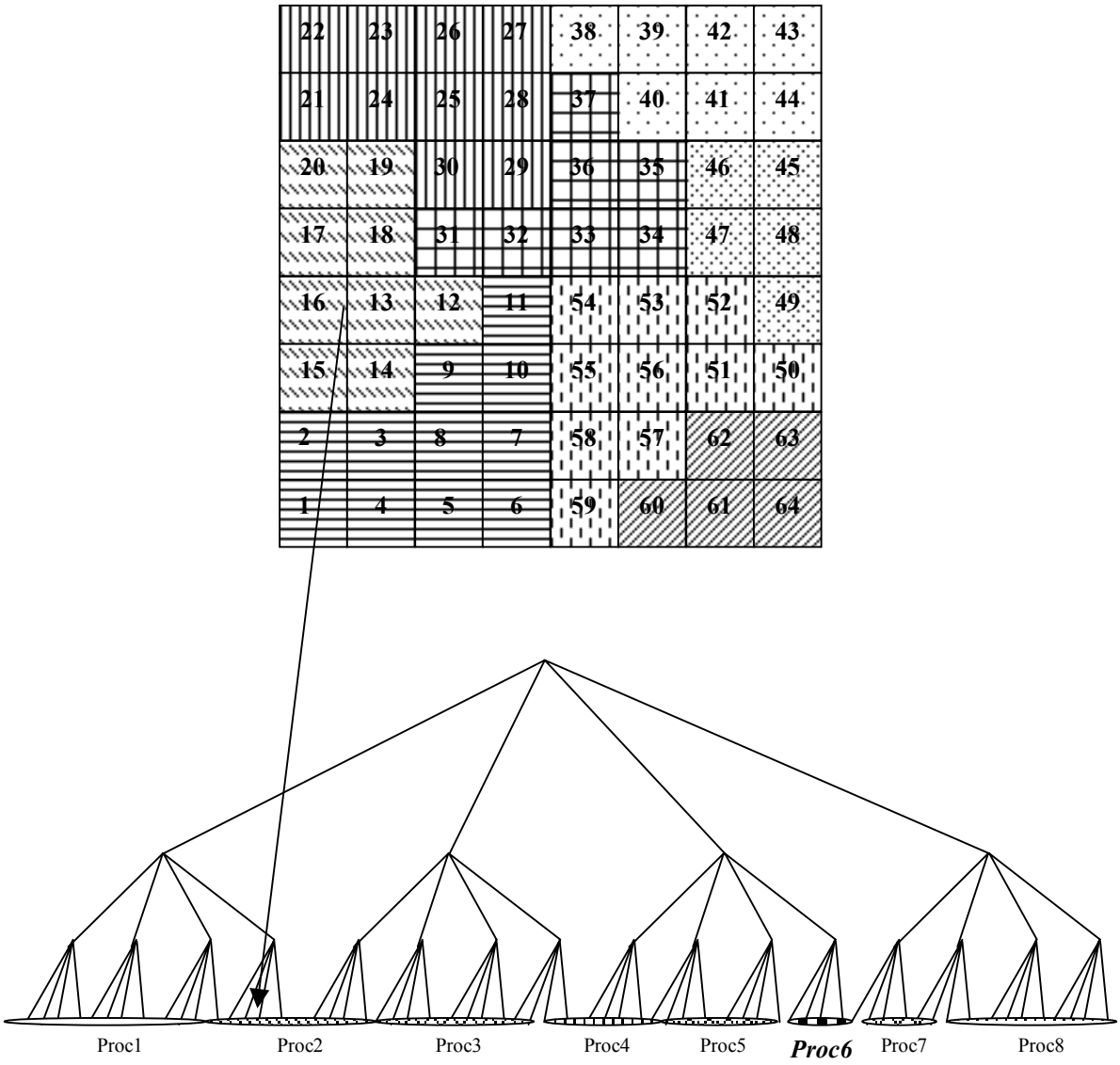


Figure 3.10:Tree decomposition with non-uniform numbered children ([23]).

Figure 3.10 shows the resulting subspaces given the same distribution as in Figure 3.7. All the subspaces assigned to processor are contiguous in this ordering scheme. None of the processors obtain two district subspaces, so we are expecting to increase performance of the tree-based parallel k-means algorithm.

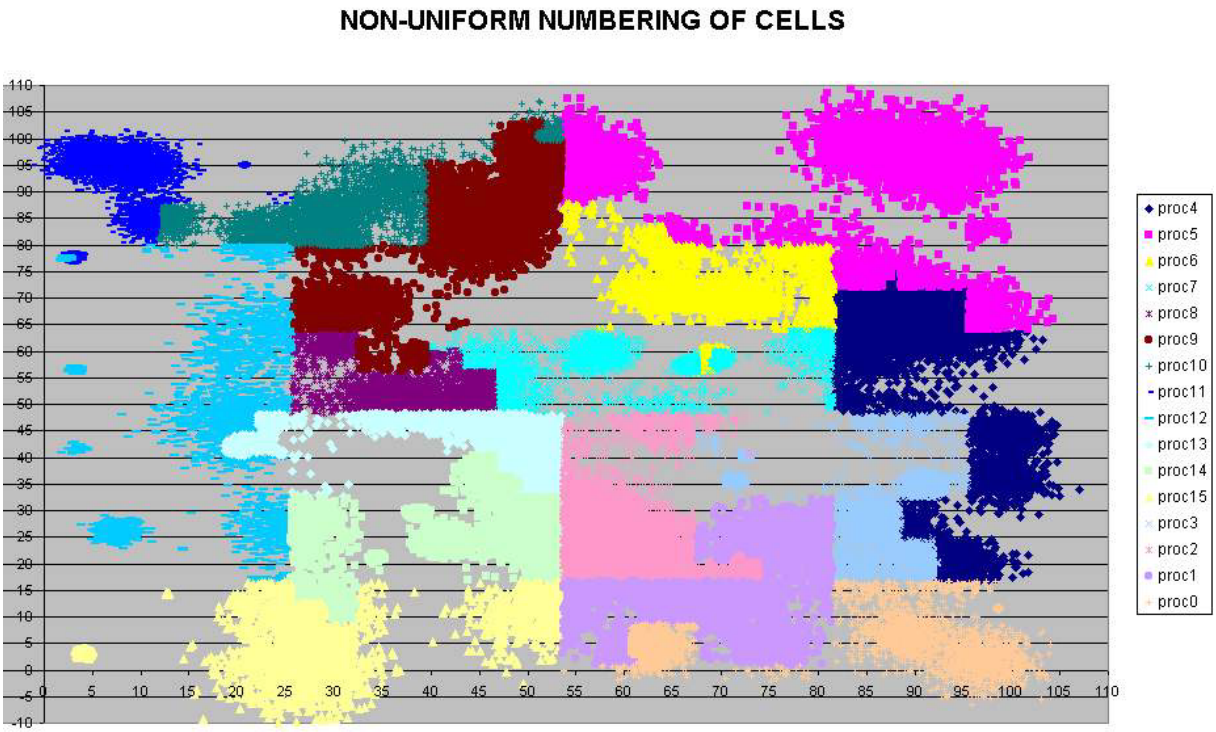


Figure 3.11: 100,000 patterns are distributed among 16 processors by using non-uniform numbering of cells. Yielding subspaces are contiguous.

In the Figure 3.11, 100,000 patterns are distributed among 16 processors. All of the subspaces are contiguous. None of the processors has apart subspaces. We expect that the performance of the tree-based k-means algorithm increases.

CHAPTER 4

EXPERIMENTAL RESULTS

We have conducted several experiments with different datasets, in order to observe possible contributions of the new pattern decomposition technique to the parallel tree-based k-means algorithm. We have examined the speed up of the algorithm and the total number of distance calculations with different pattern decomposition techniques. For the next two sections of the chapter detailed information about the experimental platforms and the datasets are given.

4.1 Experimental Platform

We run all experiments on *BORG* system, which is a 32-node PC-cluster, built with cheap commodity hardware connected with a low latency and high bandwidth interconnection network, and equipped with GNU/Linux operating system and some standard parallel programming tools such as MPI [25].

BORG system is a multi-user system and has three main hardware components, which are *Nodes*, *Interconnection* networks and *Interface Computer*.

1. *Interface Computer.* This computer is a workstation with Pentium III 500Mhz, with 512 MB SdRam and 26GB hard drive. Its operating system is GNU/Linux. It has a gigabit Network Interface Card (NIC), which connects to the uplink of switch and a fast Ethernet to connect to the Net. It provides interactions with users through console and network.
2. *Nodes.* There are 32 nodes Pentium II 400Mhz CPU, 64 MB Sdram, 6GB hard drive and Intel EtherExpress Pro 10/100 NIC.
3. *Interconnection Networks.* The interconnection network is a 3CON Super-Stack II 3900 smart switch, which has 36 100Base-TX ports and a gigabit

uplink. The ports connect to nodes and uplink connects to the interface computer.

The all of the codes are written by using *C++* and *MPI* (Message Passing Interface) and run on *BORG* system. We have conducted our experiments by using 24 Nodes at most.

4.2 Experimental Datasets

We have used several datasets all of which have been generated synthetically by a data-generating program. This program generates *k* (number of clusters) points randomly in a cube of appropriate dimensionality. For the i^{th} point, program generates *i*(2n/(k+1) k)* random points around it using uniform distribution. The result is clusters with non-uniform number of points.

We have nine different datasets with different number of points and generated by using different number of cluster centroids. We have used three different number of input patterns, and three different number of cluster centroids, in order to obtain more realistic results. Properties of the each data set can be seen in Table 4.1

Dataset	Dimensionality	Num. of Clusters	Num. of Patterns	Characteristic
DS11	2	100	100,000	Random
DS21	2	200	100,000	Random
DS41	2	400	100,000	Random
DS15	2	100	500,000	Random
DS25	2	200	500,000	Random
DS45	2	400	500,000	Random
DS110	2	100	1,000,000	Random
DS210	2	200	1,000,000	Random
DS410	2	400	1,000,000	Random

Table 4.1: Properties of the datasets.

We know that k-means algorithm depends on the choice of the starting centroids. In other words, initial choice of the starting cluster centroids determines the performance of the algorithm, effects the execution time and total number of distance calculations performed by algorithm. To eliminate the effect of the initial choice of cluster centroids in the timing measurements and the distance calculations measurements, identical initial cluster centroids are used for varying number of processors.

4.3 Evaluating Pattern Distribution Methods

Sequential algorithms are usually evaluated in terms of their execution time and this execution time directly related to the size of the input data. In the parallel algorithms, execution time depends on size of the input, architecture of the parallel computer and the number of the processors. So, we evaluate pattern decomposition techniques according to some evaluation metrics such as, running time, speed up, efficiency etc.

Random decomposition, *stripwise* decomposition, *tree-based* decomposition techniques are used in the experiments. *Tree-based* pattern decomposition techniques are categorized into three main sub-techniques. These techniques are used with four cost function in the experiments.

In the *UniformNumbering (SCF)* technique, pattern tree's children are numbered uniformly and cost of a cell is determined by simple cost function (SCF). In the *NonUniformNumbering (LCF)* technique, pattern tree's children are numbered non-uniformly and cost of a cell is determined by level cost function. In the *InsertCentroid (CCF)* technique, pattern tree's children are numbered with non-uniform numbering and cost of a cell is determined by centroid cost function *(CCF)* in which only the number of related initial centroids of a cell determines the cost of a cell. *InsertCentroid (CLCF)* technique differs from *InsertCentroid (CCF)* technique in the cost function. It uses centroid and level cost function in which the level and the number of related centroids determine the cost of a cell. The detailed information about the pattern decomposition techniques and cost function was given in Chapter 3.

4.3.1 Running Time

The serial run time of an algorithm is the time elapsed between the beginning and the end of its execution on a sequential computer. The parallel run time of an algorithm is the time duration that starts with parallel computation start and ends with the last processor finishes its execution [26]. The parallel running time consists of routing steps and computational steps. In a routing step, input data travel among the processors through the interconnection network of the system or via shared memory. In a computational step, each processor executes its operations on its local data. The running time is one of the most commonly used evaluation metric that evaluates performance of the algorithm, both in parallel or sequential algorithms.

In the experiments, all timing measurements are done using *MPI* routine *"MPI_Wtime()"*. This routine returns the number of seconds since some fixed, arbitrary point of time in the past. The timing measurements are started, after the local data to be worked by the processors is already distributed among the processors. To make our results more reliable, each measurement was repeated three or more times, and each reported data is to be interpreted as an average of repeated measurements. Since *BORG* system is a multi-user system we try to pay attention that during the experiments, no other users that need to use nodes of the *BORG*, were allowed to use it.

The execution times are measured for different datasets whose structures are explained in the previous subsection. We have implemented eight different pattern decomposition techniques and done timing measurements for all of them. The execution times for *DS11* with different number of processor are given in the Table 4.2. We have measured execution times for 1,2,4,8,16,24 processors, and graphical visualization of the measurements can be seen in Figure 4.1. The *tree-based* parallel k-means has superiority over *standard* parallel k-means. As the number the processors are increases, superiority of the *tree-based* decomposition techniques still superior than the other decomposition techniques.

	EXECUTION TIME (in seconds)					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Normal Parallel K-Means	144.388	103.051	53.59	27.78	14.89	10.94
Random Decomposition	11.05	9.25	5.71	4.01	2.91	2.48
Stripwise Decomposition	10.53	7.10	4.95	3.41	2.45	2.02
UniformNumbering (SCF)	10.40	6.74	3.52	2.32	1.69	1.41
NonUniformNumbering (LCF)	10.42	6.70	3.30	2.27	1.50	1.28
InsertCentroids (CCF)	10.45	6.71	3.33	2.23	1.65	1.58
InsertCentroids (CLCF)	10.41	6.73	3.17	2.24	1.68	1.52

Table 4.2 Total execution times (in seconds) for the standart parallel k-means and tree-based parallel k-means with different pattern decomposition techniques and with different number of processor. Input dataset is DS11.

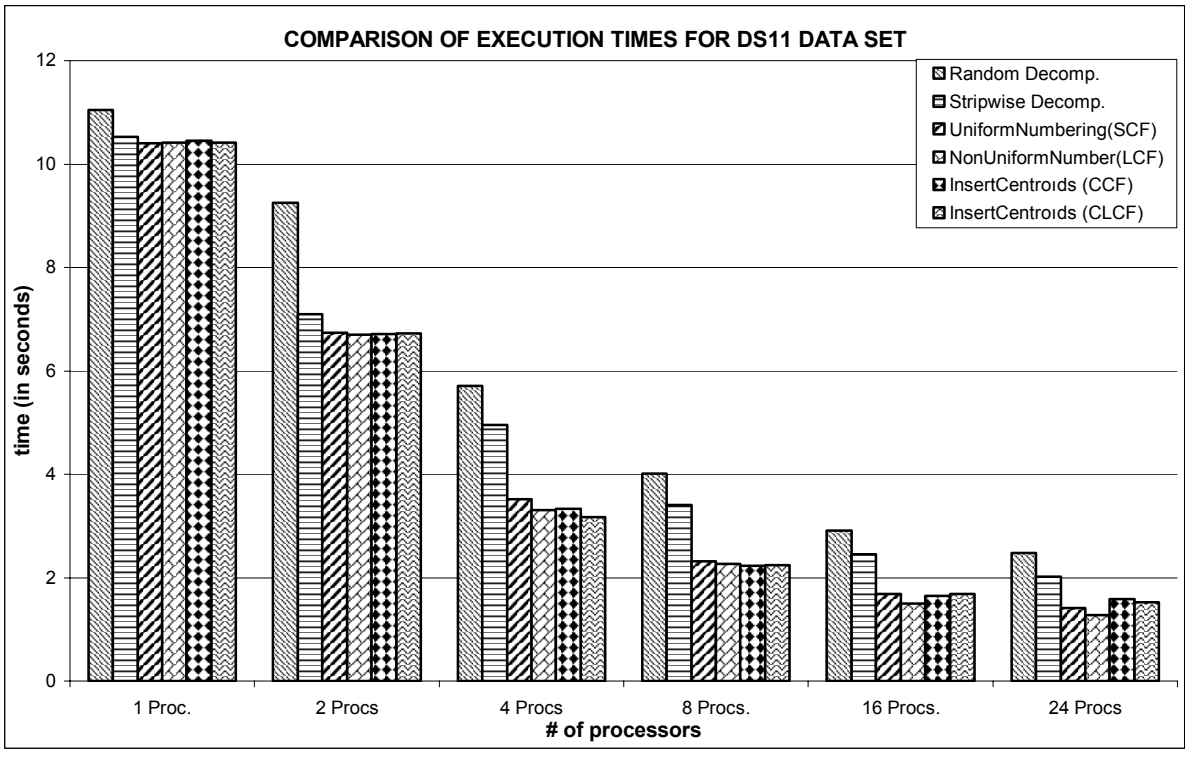


Figure 4.1 Total execution times for all of the pattern decomposition techniques for DS11 data set.

It can be easily seen that *tree-based* pattern decomposition techniques show better performance than other decomposition techniques. As we mentioned in the chapter 3, *tree-based* pattern decomposition techniques try to decompose working space into compact subspaces, which are assigned to the processors. This policy results in more pruning in the number of the candidate cluster centroids at the upper level of the pattern tree, and also less number of distance calculations in the leaf of the pattern tree. Since the main workload of the algorithm is the distance calculations between patterns and candidate clusters, the less number of distance calculations yields to less execution time, as we expected.

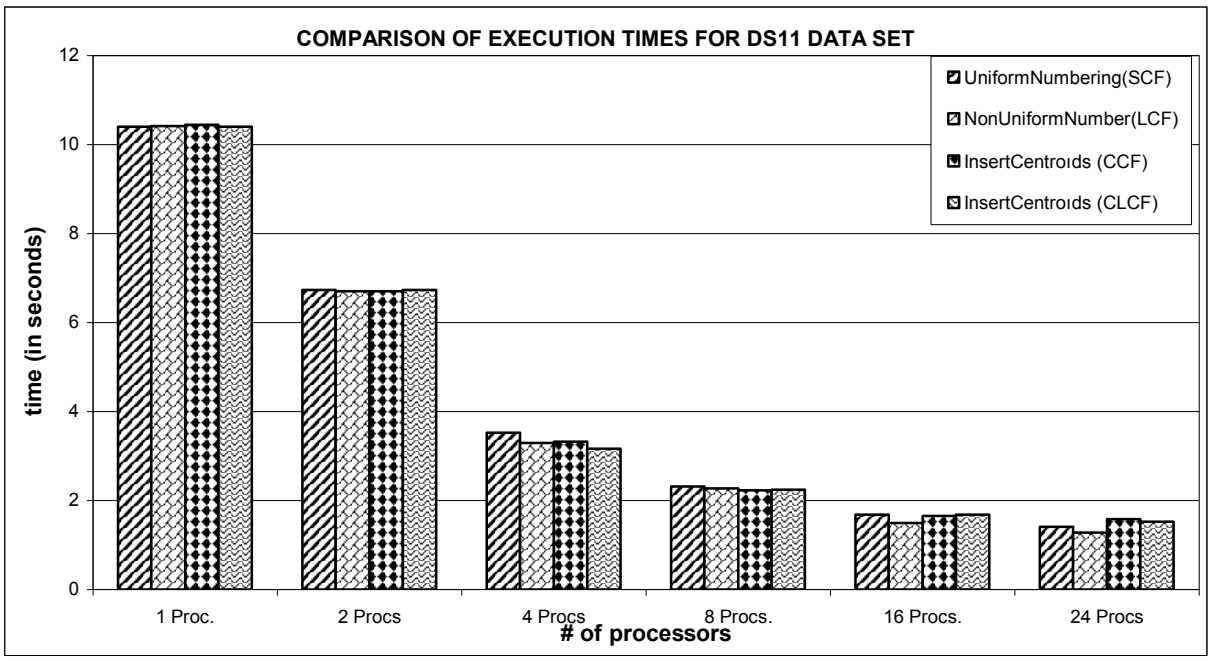


Figure 4.2: The comparison chart of the execution times of the tree-based decomposition techniques with different cost functions.

We have conducted experiments with three different tree-based decomposition techniques. And we have used four cost functions, in order to predict the workload of the pattern cell. The success of the cost functions is directly related to the predictions about the workload of the pattern cell. Figure 4.2 show that all of the tree-based techniques' performance is very near to the each other. Generally, among the tree-based decomposition techniques, *UniformNumbering (SCF)* (uniform numbered of pattern tree with simple cost function) have showed the worst performance. The reason is that *UniformNumbering (SCF)* cannot achieve physical contiguity at decomposition of the subspace, so that number of the

pruned patterns is decreased and the number of the distance calculation is increased. The rest of the experiment will be conducted with these two *tree-based* pattern decomposition technique, since there are no much differences *tree-based* pattern decomposition techniques

Figure 4.1 visualize the comparison of the execution times of pattern decomposition techniques. *Random* decomposition technique has the worst execution times. *Stripwise* decomposition technique has a better execution time than *random* decomposition but a worse execution time than all of the *tree-based* decomposition techniques. Subspaces, which are assigned to processors in the system, constructed by *random* decomposition technique and *stripwise* decomposition technique invade bigger working subspaces because of their pattern distribution policy. These bigger subspaces prevent to use advantages of the pruning process of the algorithm, so the number of the distance calculation to be performed will be higher than the *tree-based* decomposition techniques. If we examine the execution times of the *tree-based* techniques we can easily observe that, as the number of processors are increased, execution times become closer to each other(Figure 4.2). However, *NonUniformNumbering (LCF)* technique (non-uniform numbered pattern tree with level cost function) has usually showed a better performance than the *UniformNumbering (SCF)* technique because of the physical contiguity that is ensured by the non-uniform numbered pattern tree.

	EXECUTION TIME (in seconds)					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	43.10	35.75	21.02	13.25	8.70	6.93
Stripwise Decomposition	42.78	28.11	17.30	10.31	6.70	5.13
UniformNumbering (SCF)	42.51	24.61	13.56	7.70	4.57	3.59
NonUniformNumbering (LCF)	42.75	25.65	12.80	7.42	4.65	3.54

Table 4.3 Total execution times (in seconds) for different pattern decomposition techniques with different number of processorsd for DS15 data set.

	EXECUTION TIME (in seconds)					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	79.01	61.42	34.39	20.67	13.23	10.61
Stripwise Decomposition	78.69	50.85	27.58	17.44	10.80	8.30
UniformNumbering (SCF)	78.25	40.91	22.05	12.01	6.91	5.01
NonUniformNumbering (LCF)	78.45	40.33	19.54	11.79	6.77	5.06

Table 4.4 Total execution times (in seconds) for different pattern decomposition techniques with different number of processors for DS110 data set.

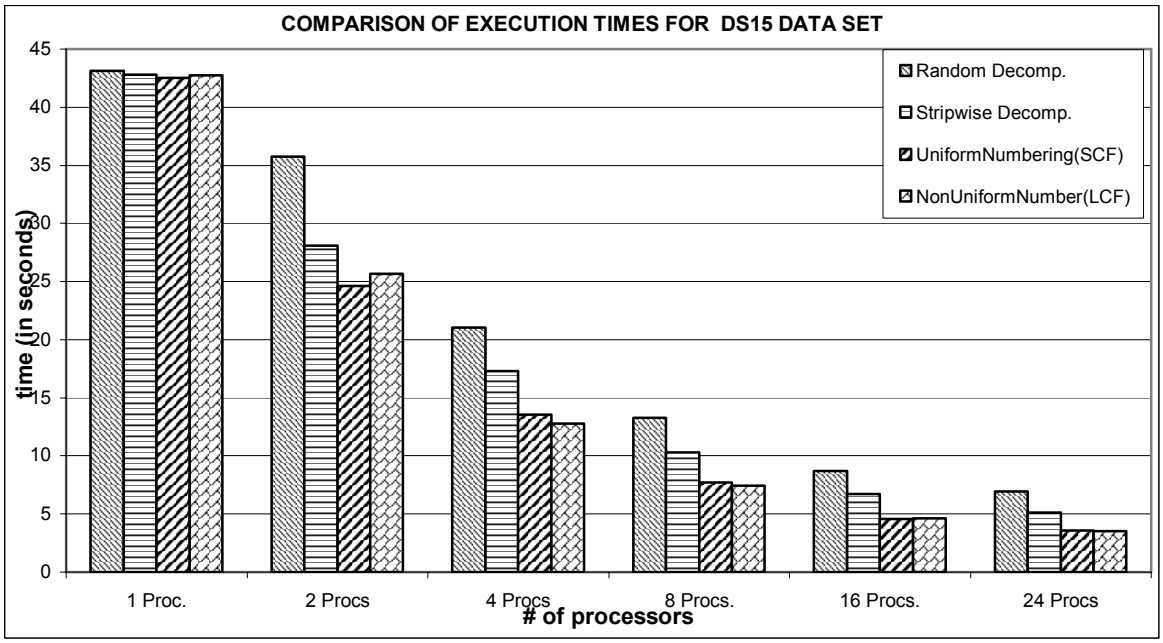


Figure 4.3: The comparison chart of the execution times of the decomposition techniques for DS15 data set. Tree-based decomposition techniques show better performance.

Tables 4.3 and 4.4 show execution times of the pattern decomposition techniques with larger datasets. We increase the size of the input data and fixed the number of the cluster centroids, in order to observe the behavior of pattern decomposition techniques for larger datasets. The visualization of the tables can be seen in Figures 4.3 and 4.4.

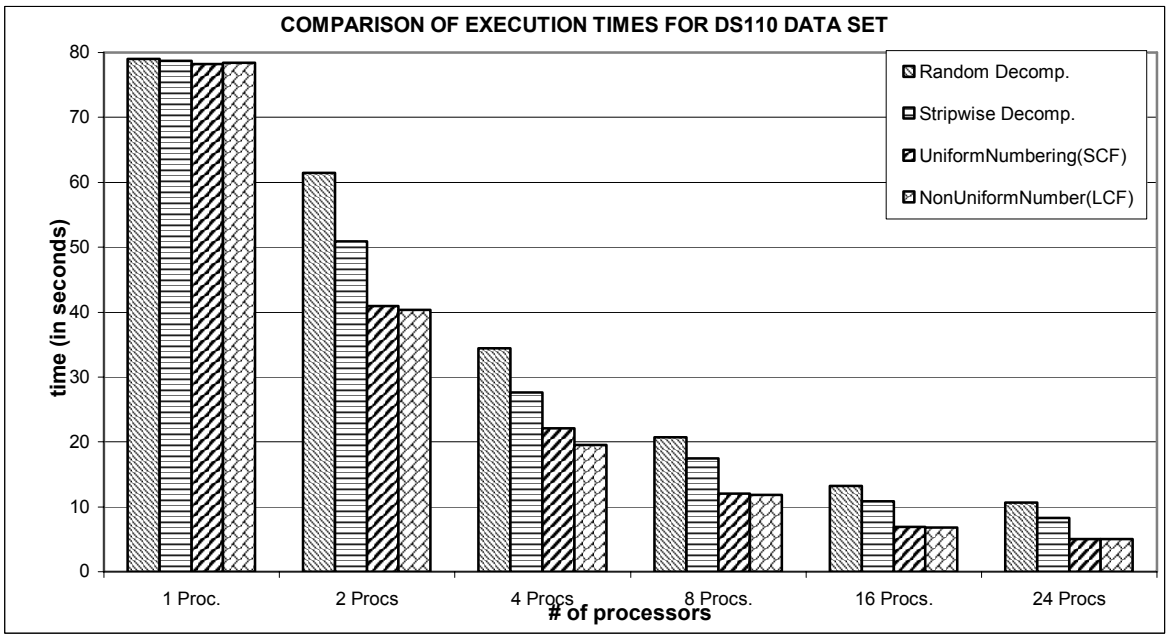


Figure 4.4: The comparison chart of the execution times of the decomposition techniques for DS110 data set. Tree-based decomposition techniques show better performance.

Increasing the number of the input patterns does not change the superiority of the *tree-based* pattern decomposition techniques. When the size of the input dataset is increased, the number of the pattern per cell is also increased for the cells of the pattern tree. In the pruning process, when the number of the candidate cluster is equal to one, all of the patterns of that cell are assigned to that unique candidate cluster centroid. Therefore, when the number of the pattern per cell is increased, by increasing the number of the input patterns, the number of the pruned patterns also increased which yields to better execution times because of the decreasing number of the distance calculations.

As we have mentioned before , increasing the size of the input datasets forces cells of the pattern tree to contain large number of patterns. Because of the structure of the *LCF*, which is explained in detail in the chapter 3, cells, with a large number of patterns will decrease the effect of the level over the cost function. When we look at the formula of *LCF*, if the number of pattern per cell is very big, *LCF* will behave like *SCF*. In other words, number of the patterns, determined by two cost functions, will not change drastically. So that, as the size of the input data is increased, execution times of the tree-based pattern decomposition techniques do not vary so much from each other.

	EXECUTION TIME (in seconds)					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	23.50	20.60	13.29	8.97	6.25	5.19
Stripwise Decomposition	23.24	17.14	10.91	7.43	5.05	4.11
UniformNumbering (SCF)	22.74	15.16	8.52	5.12	3.34	2.90
NonUniformNumbering (LCF)	22.80	15.09	8.18	5.03	3.58	2.79
InsertCentroids (CCF)	23.08	15.02	8.21	5.07	3.64	2.78
InsertCentroids (CLCF)	22.95	15.05	8.16	5.06	3.55	2.74

Table 4.5 Total execution times (in seconds) for different pattern decomposition techniques with different number of processors for DS21 data set.

In the second step of the execution times experiment, we fixed the number of the input pattern while changing the number of the input clusters in order to observe the effects of the increasing the number of the centroid clusters. In this experiment, we used three different datasets, each of which contains same number of patterns but different number of clusters. The experiments are conducted with datasets *DS21*and *DS41*. The results of the conducted experiments are shown in Tables 4.5 and 4.6.

	EXECUTION TIME (in seconds)					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	45.29	36.72	23.02	15.48	11.06	9.34
Stripwise Decomposition	44.66	31.44	19.68	12.48	8.33	7.06
UniformNumbering (SCF)	44.35	29.60	15.39	9.27	6.44	5.02
NonUniformNumbering (LCF)	44.49	30.39	15.07	8.97	6.06	5.01
InsertCentroids (CCF)	44.37	30.34	15.06	9.14	6.09	5.06
InsertCentroids (CLCF)	44.42	30.38	15.12	9.04	6.13	5.08

Table 4.6 Total Execution times (in seconds) for different pattern decomposition techniques with different number of processor for DS41 data set.

If we increase the total number of the candidate cluster, it is natural that the execution times of all the pattern decomposition techniques are also increased. Increasing the number of the input cluster decreases the performance of the pruning process. The increasing the number of the input cluster decreases the number of the pruned patterns and increases the number of the candidate clusters, thus, algorithm performs more distance calculations both for the inner nodes of the pattern tree and the leaf nodes of the pattern tree. *Random* decomposition technique, as we can see in Figures 4.5 and 4.6, has the worst execution times for both datasets. Its pruning performance is drastically decreased. Even, for the *DS41,* the number of the patterns pruned by the pruning algorithm is very small compared to the other techniques.

Tree-based pattern decomposition techniques have better execution times than the execution times of the other pattern decomposition techniques. Their execution times are very near to each other. The *UniformNumbering (SCF)* has usually the worst execution times for all datasets among the tree-based pattern decomposition techniques. *InsertCentroid (CCF)* decomposition technique has usually faster than the *UniformNumbering (SCF)* but slower than the *NonUniformNumbering (LCF).* As the number of the cluster centroids is increased, its execution times are closer to the *NonUniformNumbering (LCF)* technique's execution times. Its shows better performance as the number of the input cluster is increased, as we expected.

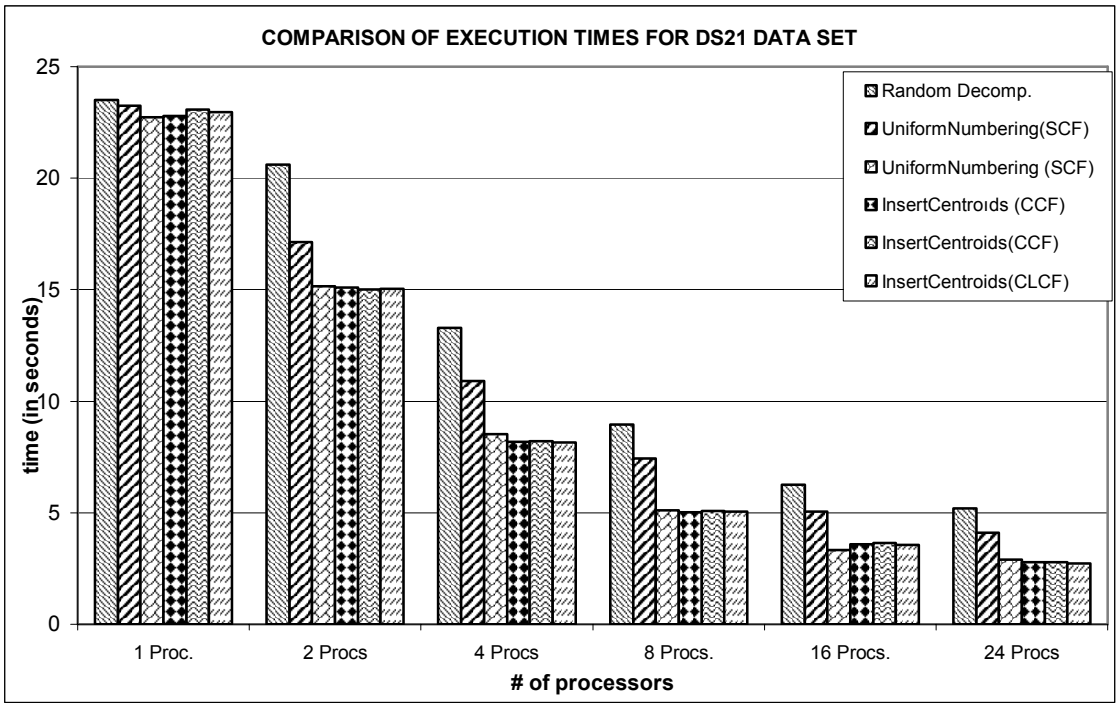


Figure 4.5: The comparison chart of the execution times of the decomposition techniques for DS21 data set. Insert Centroids(CLCF) decomposition techniques has the best execution times.

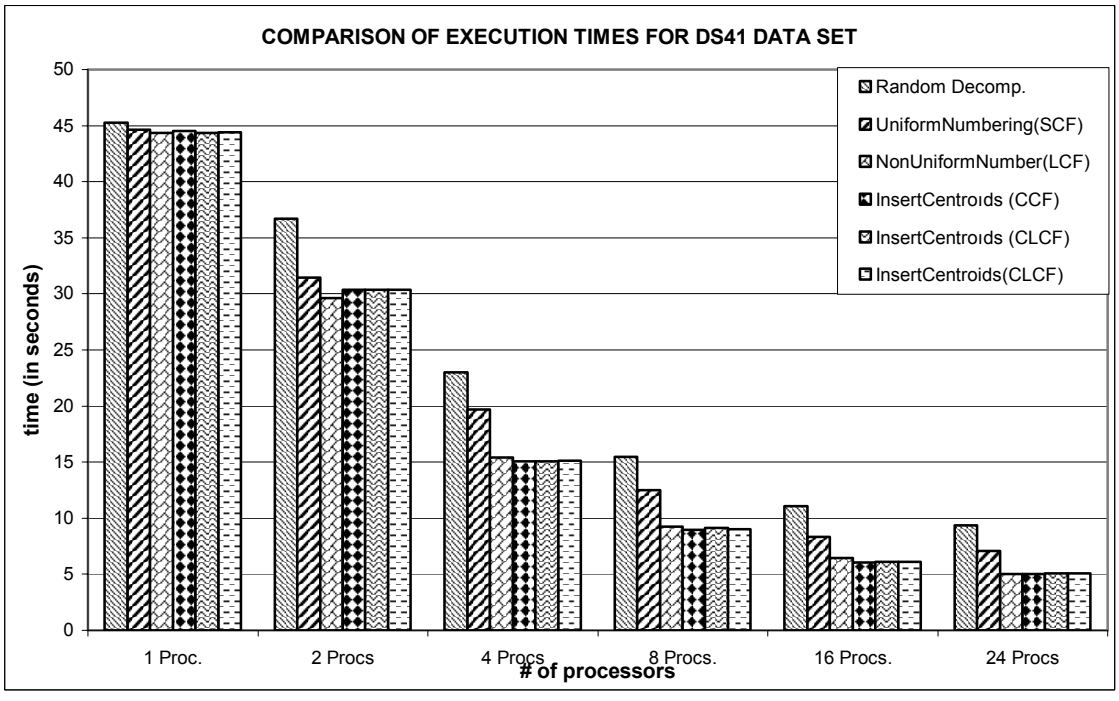


Figure 4.6: The comparison chart of the execution times of the decomposition techniques for DS41 dataset. Insert Centroids decomposition technique has better execution times.

4.3.2 Total Number of Distance Calculation

As we have mentioned in Chapter 2, the computational load is largely due to the distance calculations between input patterns and input clusters, in the k-means algorithm. The number of distance calculations performed by the standard k-means algorithm is proportional to the product of input number of patterns and number of clusters. This is computationally very expensive for very large datasets. It is also clear that execution time is directly proportional to the number of distance calculation. In the light of these observations we have examined the number of distance calculations for all of the decomposition techniques.

In the distributed k-means clustering with pruning, two main distance calculations are performed. One of them is *Cell-Centroid* calculation and the other is *Pattern-Centroid* calculation. *Cell-Centroid* calculations are performed at the upper level of the pattern tree for the determination of pruned centroids. In this calculation, distance between center of cell and cluster is calculated. *Pattern-Centroid* calculations calculate distance between pattern and cluster, like standard k-means.

In our experiments, since we have two main distance calculations to be performed, we have collected statistics about these two distance calculations, and we have determined total number of distance calculations by adding *Cell-Centroid* and *Pattern-Centroid* calculations.

	TOTAL NUMBER OF CALCULATIONS*1000					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	8,237	12,618	19,224	29,759	45,084	58,014
Stripwise Decomposition	8,237	9,416	12,254	17,140	23,506	28,376
UniformNumbering (SCF)	8,237	9,363	8,856	9,727	9,909	10,080
NonUniformNumbering (LCF)	8,237	9,319	8,789	9,491	9,495	9,604
InsertCentroids (CCF)	8,237	9,319	8,798	9,409	9,759	9,935
InsertCentroids (CLCF)	8,237	9,327	8,734	9,438	9,761	9,847

Table 4.7 : Total Number of distance calculations for DS11 data set. Total distance calculations is combination of Cell-Centroid distance calculation and Pattern-Centroid distance calculations.

Table 4.7 shows total number of distance calculations performed by the whole pattern distribution methods. *Random* decomposition method performs the highest number of distance calculations, as we expected. *Stripwise* distribution method performs less number of distance calculations than the number of the distance calculations of the random decomposition method. *Tree-based* decomposition techniques perform less number of distance calculations than the others. The numbers of the distance calculations, performed by tree-based techniques, show slight difference from each other. Generally, the least number of distance calculations is performed by the *NonUniformNumbering (LCF)* technique among all methods. Because, this technique causes more pruning activity at the upper level of the pattern tree, and these pruning activity results in decreasing the number of the *Pattern-Centroid* calculations, which are performed in leaf of the pattern tree, and *Cell-Centroid* calculations.

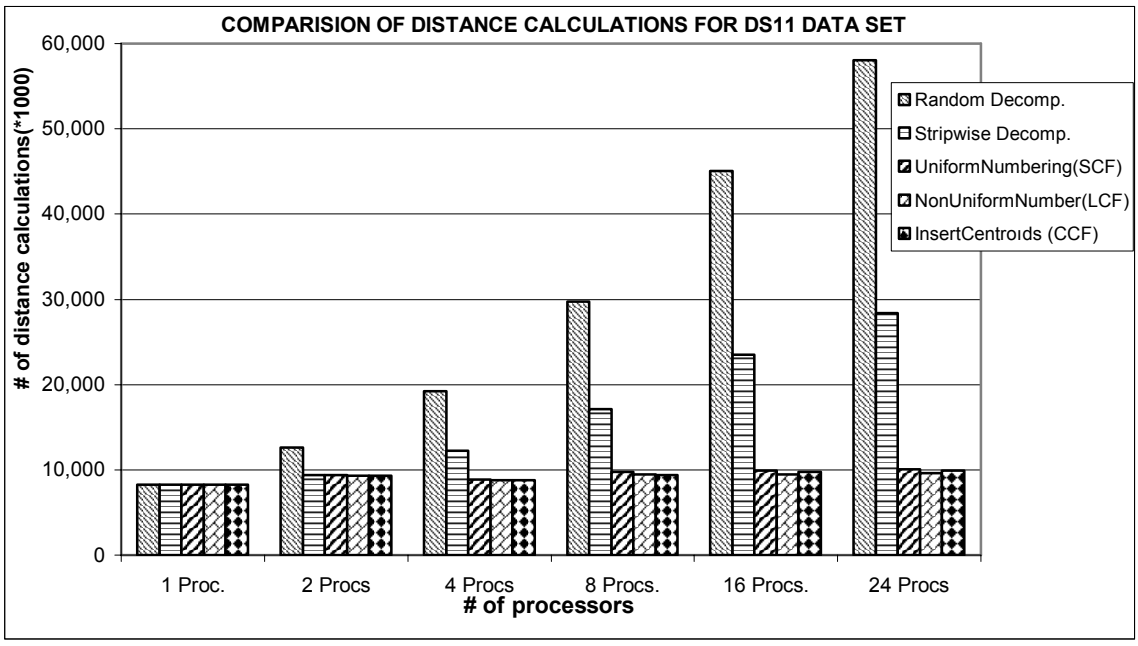


Figure 4.7 : The comparison chart of the total number of the distance calculations for the DS11 data set. Random decomposition technique performs the highest number of distance calculation.

When the number processors are increased, total number of the distance calculations is also increased. But for the *random* decomposition and *stripwise* decomposition increasing ratios are very high. For example in the *random* decomposition technique, if the number of processors is increased from one to twenty four, total number of the distance calculation is increased approximately nine times. However, in the *UniformNumbering (SCF)* technique, which generally performs the highest number of distance calculations among *tree-based*

techniques, same increase in the number of processors causes only small increase in the total number of distance calculations. In other words, tree-based decomposition techniques improve the parallelization advantages of the distributed k-means with pruning algorithm.

	TOTAL NUMBER OF CALCULATIONS*1000					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	20,742	30,505	45,930	68,171	104,006	131,851
Stripwise Decomposition	20,742	23,005	29,565	36,508	48,400	58,340
UniformNumbering (SCF)	20,742	22,784	21,798	22,695	23,101	23,386
NonUniformNumbering (LCF)	20,742	22,854	21,775	22,641	23,051	23,328

Table 4.8 : Total number of distance calculations for DS15 data set. Random decomposition technique performs the highest number of calculations. As the number of processors are increased, number of distance calculations increased drastically in random decomposition technique.

	TOTAL NUMBER OF CALCULATIONS*1000					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	27,495	39,396	58,146	85,484	128,977	164,582
Stripwise Decomposition	27,495	29,651	36,693	51,317	70,084	81,998
UniformNumbering (SCF)	27,495	29,203	28,204	28,858	29,373	29,844
NonUniformNumbering (LCF)	27,495	29,121	28,748	28,292	28,288	29,697

Table 4.9 : Total Number of distance calculations for DS110 data set. In stripwise decomposition technique, as the number of processors increases, number of distance calculations increases almost three times.

Tables 4.8 and 4.9 show the total number of the distance calculations of all of the decomposition techniques for two different dataset, *DS15* and *DS110*. *DS15* dataset has 500,000 and *DS110* has 1 million patterns. In these experiments, the input number of the patterns is increased while the number of the cluster is not changed, in order to observe to reaction of the all pattern decomposition methods. *Tree-based* decomposition techniques still have superiority over two other techniques. We have observed that, as the number of patterns increased the *random* decomposition technique's pruning performance is improved. For example for dataset *DS15*, when the number of the processors is increased from one to twenty four, total number of distance calculations increased approximately six times, this increase was nine times for dataset *DS11*, in which there are 100,000 patterns. On the other hand, increasing the number of the patterns does not affect the performance of the tree-based decomposition techniques; their ratio between the number of the distance calculations of one processor and twenty-four processors does not differ very much.

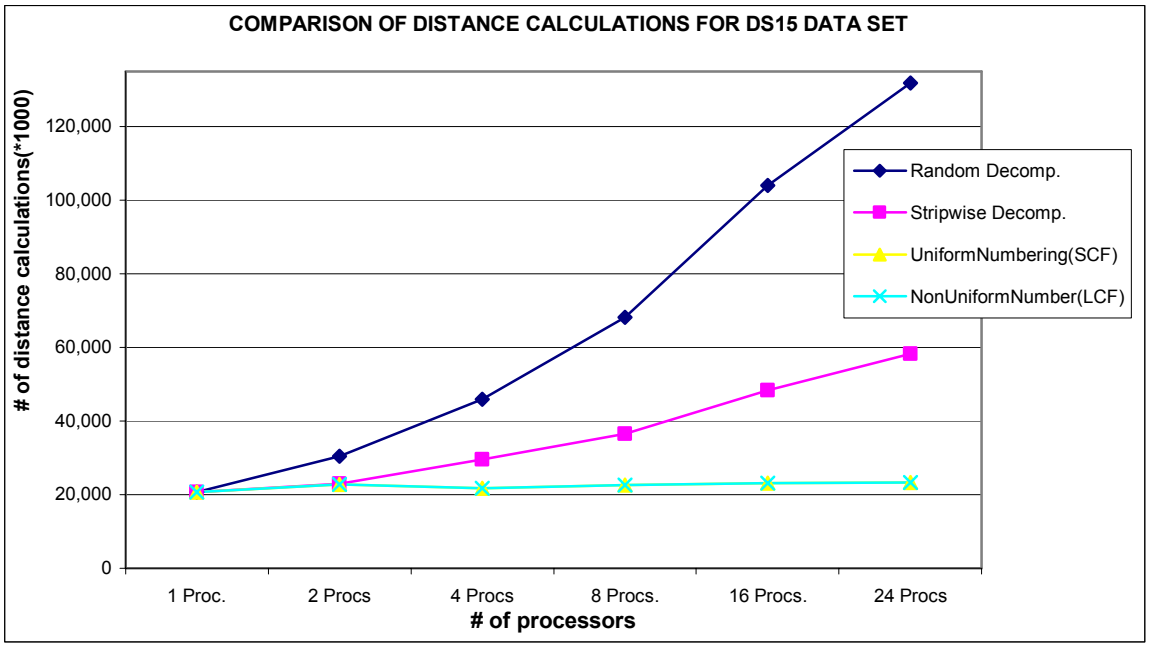


Figure 4.8 : The comparison chart of the total number of the distance calculations for the DS15 data set. Random decomposition technique performs the highest number of distance calculations.

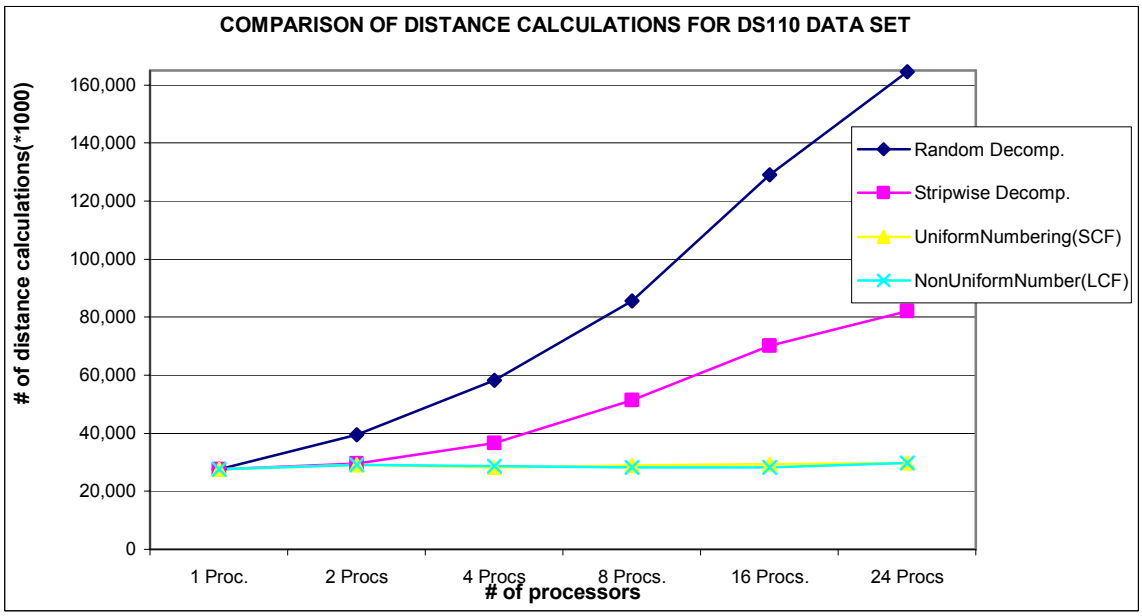


Figure 4.9 : The comparison chart of the total number of the distance calculations for the DS110 data set. The number of the distance calculations performed by the tree-based decomposition techniques are very near to each other.

There is a great difference between the total number of distance calculations of tree-based methods and the others as shown in Figures 4.8 and 4.9. *Stripwise* decomposition technique always performs approximately half of the number of the distance calculations, which is performed by *random* decomposition technique. Increasing the number of the input patterns does not affect this ratio very much. This case is valid for distance calculation ratio among all methods. Increasing the number of input does not affect the distance calculation ratio among all decomposition methods.

Experiments show that, increasing the number of the input patterns does not affect superiority of the tree-based decomposition technique but all tree-based methods show very similar results because of the structure of the pattern tree. All of the cost functions results are approaching the *SCF* because of the reason that we have explained in the previous section.

To observe the effects of increasing the number of centroids, we have used dataset *DS21* and dataset *DS41*, each of which has the same number of patterns and has 200 and 400 cluster centroids respectively. The results have shown that, increasing the number of the centroids does not affect superiority of the *tree-based* decomposition techniques(Table 4.10). *Random* decomposition still performs the highest number of distance calculations. And the number of distance calculations does not vary very much among tree-based decomposition techniques.

	TOTAL NUMBER OF CALCULATIONS*1000					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	18,209	28,061	43,672	69,580	113,206	155,733
Stripwise Decomposition	18,209	21,294	28,792	42,027	58,927	71,304
UniformNumbering (SCF)	18,209	20,809	20,265	20,723	22,065	22,553
NonUniformNumbering (LCF)	18,209	20,788	19,609	20,501	21,199	21,780
InsertCentroids (CCF)	18,209	20,759	19,610	20,518	21,306	21,750
InsertCentroids (CLCF)	18,209	20,797	19,609	20,511	21,193	21,765

Table 4.10 : Total number of distance calculations for dataset DS21. Random decomposition technique performs the highest number of calculations.

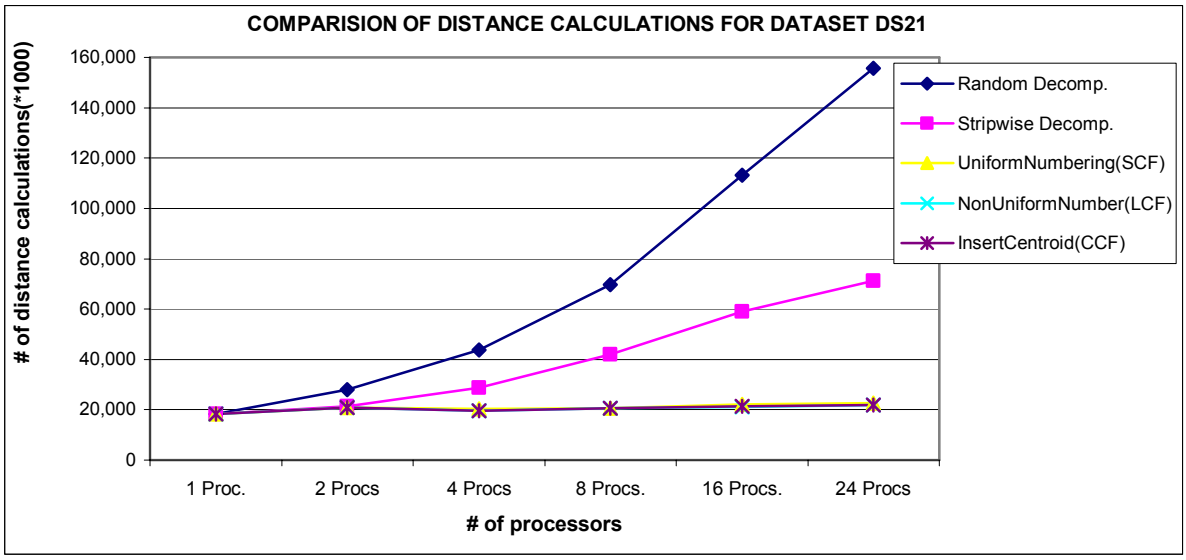


Figure 4.10 : The comparison chart of the total number of the distance calculations for DS21 data set. The number of the distance calculations performed by the tree-based decomposition techniques are very near to each other.

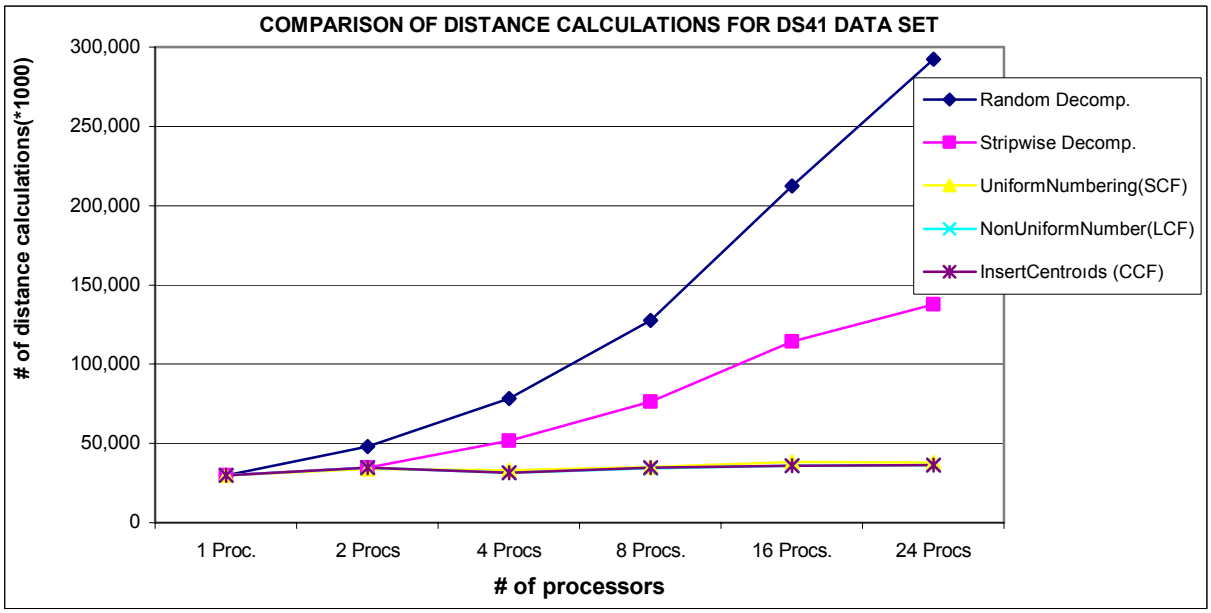


Figure 4.11 : The comparison chart of the total number of the distance calculations for DS41 data set. The number of the distance calculations performed by random decomposition technique is increasing when the number of the proceesors is increased.

Although the number of the input clusters is increased, superiority of the *tree-based* decomposition technique is not affected (Figures 4.10 and 4.11). As the number of the processor is increased, distance calculations performed by the pattern decomposition techniques also increased. But for each decomposition technique, the increase ratio related to the number of the processor is varying. For example, when we increase the number of the processors from one to twenty four, random decomposition technique performs seven times more distance calculations for dataset *DS11*. On the other hand, this increasing ratio (related to the number of processors) will be almost ten times when we increased the number of the clusters from 100 to 400, by using *DS41*. Therefore, increasing the number of the clusters decreases the performance of the *random* decomposition technique.

UniformNumbering (SCF) decomposition technique performs the highest number of distance calculations. Increasing the cluster centroids does not affect its performance very much. *InsertCentroids (CCF)* pattern decomposition technique is affected positively by the increase of the cluster centroids. In other words its performance is slightly increased as we expected. *NonUniformNumbering (LCF)* performs the smallest number of the distance calculations among all the pattern decomposition method.

Shortly, increments in the number of the distance calculations do not affect the performance of the *tree-based* pattern decomposition techniques, they still performs the least number of distance calculations. But *random* decomposition technique is affected negatively; its performance is decreased as the number of the input clusters is increased.

4.3.3 Speedup

One of the main purposes of the parallel systems is to decrease execution time by adding more processors. When we evaluate a parallel algorithm, we should know how much performance gain is achieved by parallelizing a sequential algorithm. This desirable characteristic of a parallel algorithm is measured by *speedup*.

Speedup is defined, as the ratio of the time taken to solve a problem on a single processor to the time required solving the same problem on a parallel computer with p processor [26]. If we apply this definition to our problem, we can define *speedup* as the ratio

of the execution time for *k* clusters on a single processor to the execution time for identically clustering the same data set on *p* processor. We can consider *speedup* either as a measure that represents the relative advantage of solving a problem in parallel or as a summary of the efficiency of the parallel algorithm.

If we examine the definition of the *speedup,* we can conclude that, if a parallel algorithm has high *speedup* values*,* it has achieved better parallelization. In the light of the above explanations, a parallel algorithm's speedup values can achieve at most *p*, where *p* is the number of the processor, and can never exceed it.

	SPEEDUP					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	1.00	1.19	1.93	2.75	3.80	4.45
Stripwise Decomposition	1.00	1.48	2.13	3.09	4.30	5.21
UniformNumbering (SCF)	1.00	1.54	2.95	4.49	6.15	7.38
NonUniformNumbering (LCF)	1.00	1.55	3.16	4.59	6.95	8.16
InsertCentroids (CCF)	1.00	1.56	3.14	4.69	6.33	6.61
InsertCentroids (CLCF)	1.00	1.55	3.28	4.65	6.20	6.85

Table 4.11: The speedup values of all decomposition techniques for DS11 data set.

Table 4.11 shows the speedup values of the all decomposition techniques. The input data set is *DS11*, and we can observe that the *tree-based* decomposition techniques have superiority over the *random* decomposition and *stripwise* decomposition. We know that speedup directly related with the execution time of the algorithm, since the execution time of the tree-based decomposition techniques are very near to each other, their speedup values also very near to each other.

The *NonUniformNumbering (LCF)* technique always has the best speedup values as shown in the Figure 4.12. Although, *UniformNumbering (SCF)* technique has usually the worst speedup values among tree-based techniques, its values are better than the non-tree-based decomposition techniques. The *InsertCentroid (CCF)* technique has not as good speedup values as the *NonUniformNumbering (LCF)* technique has.

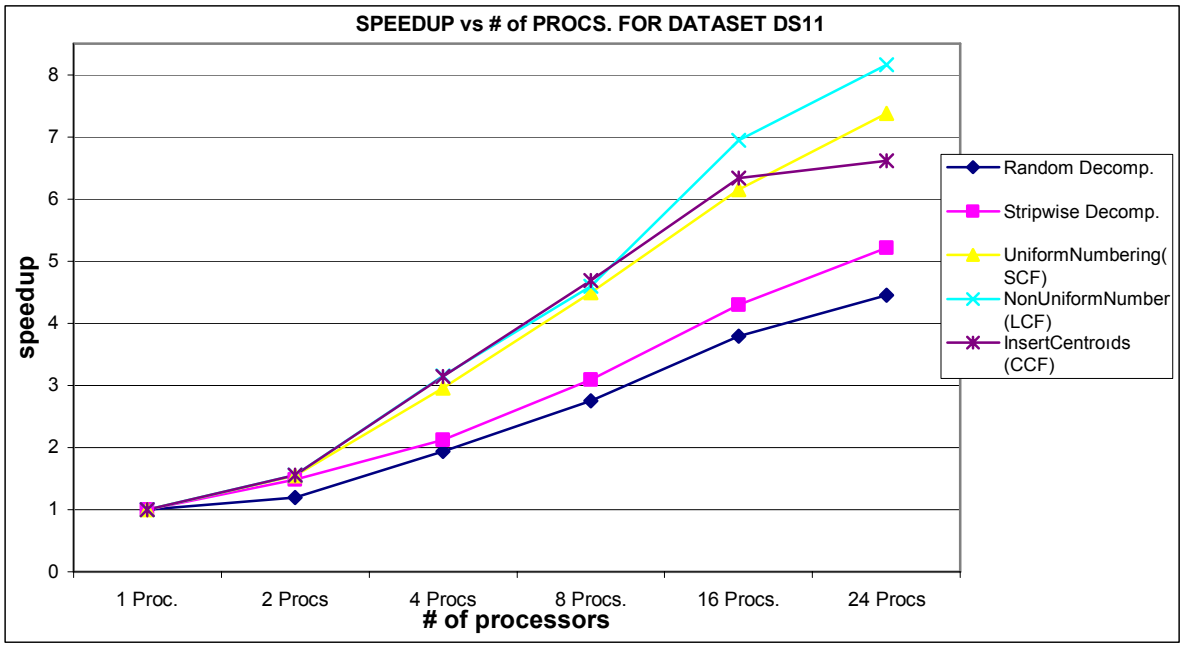


Figure 4.12 : The comparision chart of the speedup values of five decomposition techniques. The input dataset is DS11.

As we have done in the previous experiments, in order to observe the effects of the increasing input pattern number, we have repeated the experiments, with two more dataset, *DS*15 and *DS*110. The speedup values are presented in Table 4.12 and Figure 4.13.

	SPEEDUP					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	1.00	1.21	2.05	3.25	4.96	6.22
Stripwise Decomposition	1.00	1.52	2.47	4.15	6.39	8.34
UniformNumbering (SCF)	1.00	1.73	3.14	5.52	9.31	11.85
NonUniformNumbering (LCF)	1.00	1.67	3.34	5.76	9.19	12.09

Table 4.12: The speedup values of all decomposition techniques. The input dataset is DS15.

Increasing the number of input patterns has improved the speedup values of all decomposition techniques, as we expected. The speedup improvement ratio of the decomposition techniques does not differ from each other. For example, *NonUniformNumbering (LCF)* technique's speedup values improved approximately two times when the number of the input pattern is increased like the *random* decomposition technique.

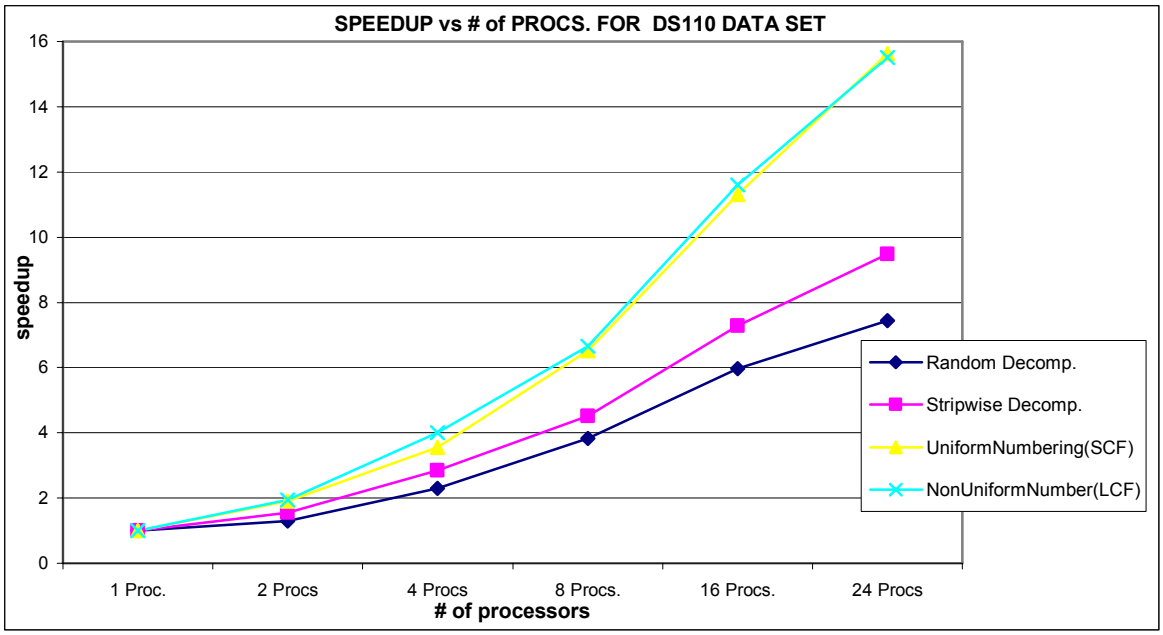


Figure 4.13 : The comparison chart of the speedup values of five decomposition techniques for DS110 data set.

Finally we study the speedup behavior when the number of the predefined cluster is varied. In this part of the experiment we fixed the input number of the patterns and increased the predefined number of clusters. We have used two data sets *DS210* and *DS410*. As we have seen in Figure 4.14 and Table 4.13, increasing the number of the cluster does not change speedup values of the pattern decomposition techniques very much. But *tree-based* decomposition techniques still have superiority over the *random* decomposition technique and *stripwise* decomposition technique.

	SPEEDUP					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	1.00	1.14	1.77	2.62	3.76	4.53
Stripwise Decomposition	1.00	1.36	2.13	3.13	4.60	5.65
UniformNumbering (SCF)	1.00	1.50	2.67	4.45	6.80	7.84
NonUniformNumber(LCF)	1.00	1.51	2.79	4.53	6.37	8.19

Table 4.13: The speedup values of all decomposition techniques for DS210 data set.

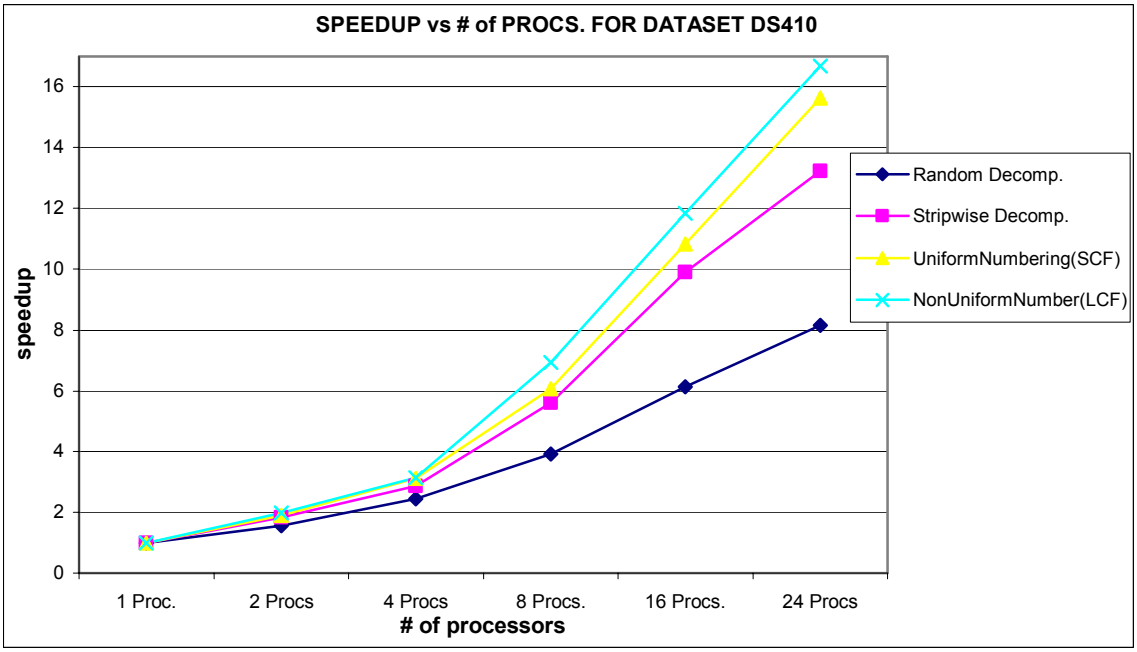


Figure 4.14 : The comparison chart of the speedup values of five decomposition techniques for DS410 data set.

4.3.4 Load Imbalance

In many parallel applications, it is very hard to predict the size of the work assigned to various processors. If different processors have different workloads, some processors may be idle during the part of the time while others working on the problem. In our application, load imbalance is very hard to deal with because of the characteristic of the pruning algorithm. We know that for standard parallel k-means the distribution of overall load is straightforward. But, in tree-based k-means algorithm it is hard to deal with.

Load imbalance can be defined as the ratio of maximum load difference to the average load. In the calculation of the load imbalance, average computational load of the system is calculated, and then maximum workload of the whole system is found. The ratio between these two values reflects the load imbalance.

	TOTAL IMBALANCE					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	0.00	0.01	0.01	0.05	0.05	0.04
Stripwise Decomposition	0.00	0.06	0.22	0.29	0.29	0.32
UniformNumbering (SCF)	0.00	0.06	0.12	0.17	0.31	0.34
NonUniformNumber(LCF)	0.00	0.06	0.09	0.23	0.30	0.37
InsertCentroids (CCF)	0.00	0.06	0.09	0.22	0.33	0.43
InsertCentroids (CLCF)	0.00	0.06	0.05	0.23	0.32	0.38

Table 4.14: The load imbalance values caused by all decomposition techniques for DS11 data set. As the number of the processor is increased, load imbalance increases with tree-based decomposition technique.

We observed that, *random* decomposition has the best-balanced computational workload. Increase in the number of the processors does not change this fact. *Tree-based* decomposition techniques suffer from the load imbalance, when the number of processor is increased. But, it has better computational workload than the *stripwise* decomposition technique when the number of the processors is less than sixteen processors. *InsertCentroid (CFC)* technique has the worst balanced load, because, it does not consider the update of the cluster which will be explained in the next section of the chapter.

In the second step of the load imbalance experiments, we again increase the number of the input patterns while fixing the number of the cluster, in order to observe that how load imbalance of the decomposition techniques will change.

	TOTAL IMBALANCE					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	0.00	0.00	0.00	0.01	0.02	0.03
Stripwise Decomposition	0.00	0.07	0.19	0.24	0.34	0.28
UniformNumbering (SCF)	0.00	0.03	0.23	0.30	0.29	0.36
NonUniformNumbering (LCF)	0.00	0.04	0.15	0.16	0.40	0.47

Table 4.15: The load imbalance values caused by all decomposition techniques for DS15 data set. As the number of the input patterns is increased, load imbalance of the tree-based decomposition technique is increased.

As shown in Table 4.15 and Figure 4.15, increasing number of the input patterns makes *tree-based* decomposition techniques more sensitive to load imbalance. Their load imbalance increases. *Stripwise* decomposition technique also suffers from the load imbalance, and like *tree-based* decomposition techniques, as the number of the input patterns is increased, *stripwise* decomposition technique causes more load imbalance. On the other hand, *random* decomposition technique is not affected by the increment of the number of the input patterns.

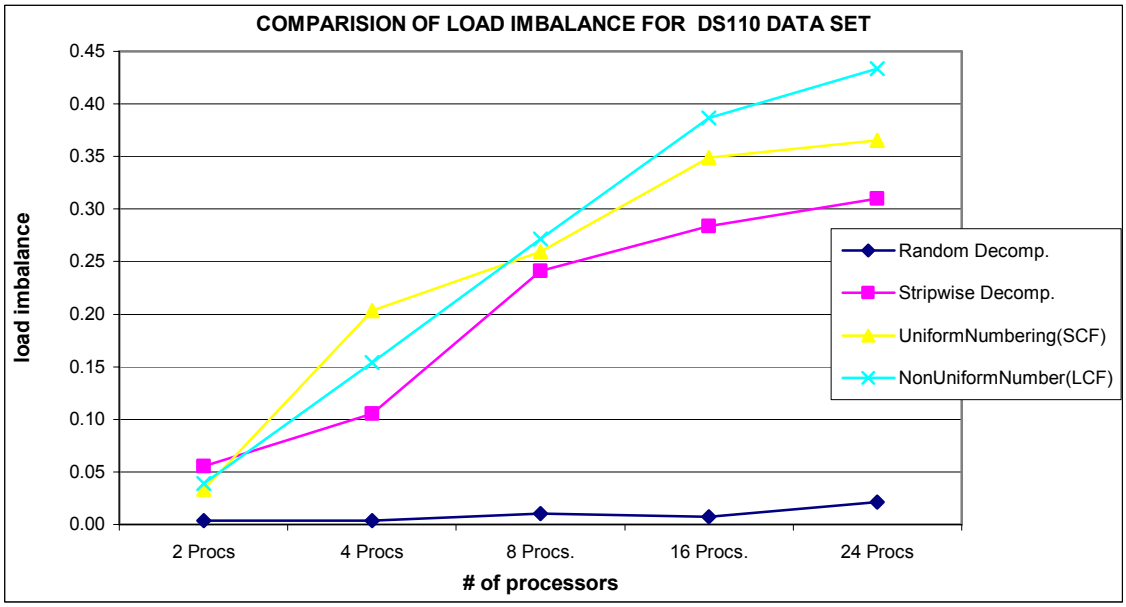


Figure 4.15: The load imbalance values caused by all decomposition techniques. The input DS110 data set. As the number of the input patterns is increased, load imbalance of the tree-based decomposition technique increased.

In the second phase of the load imbalance experiments, we have changed the number of the cluster while fixing the number of the input patterns. We have used two datasets with the same number of patterns but different number of clusters. Tables 4.16 and 4.17 show the results of the experiments. By the increase in the number of the cluster centroids, the computational load balances of the *tree-based* decomposition techniques are affected negatively. The experiments have shown that, *random* pattern decomposition technique is not affected by the increase in the number of the pattern decomposition technique.

	TOTAL IMBALANCE					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	0.00	0.00	0.02	0.02	0.04	0.04
Stripwise Decomposition	0.00	0.03	0.10	0.18	0.24	0.25
UniformNumbering (SCF)	0.00	0.05	0.12	0.20	0.20	0.40
NonUniformNumbering (LCF)	0.00	0.06	0.08	0.19	0.27	0.33
InsertCentroids (CCF)	0.00	0.06	0.08	0.19	0.27	0.34
InsertCentroids (CLCF)	0.00	0.06	0.08	0.19	0.27	0.33

Table 4.16: The load imbalance values caused by all decomposition techniques for DS21 data set. As the number of the input patterns is increased, load imbalance of the tree-based decomposition technique increased.

	TOTAL IMBALANCE					
	1 Proc.	2 Procs	4 Procs	8 Procs.	16 Procs.	24 Procs
Random Decomposition	0.00	0.00	0.01	0.03	0.03	0.03
Stripwise Decomposition	0.00	0.01	0.13	0.11	0.13	0.17
UniformNumbering (SCF)	0.00	0.01	0.07	0.19	0.29	0.30
NonUniformNumbering (LCF)	0.00	0.00	0.10	0.13	0.25	0.34
InsertCentroids (CCF)	0.00	0.00	0.10	0.18	0.24	0.34
InsertCentroids (CLCF)	0.00	0.00	0.09	0.13	0.24	0.34

Table 4.17: The load imbalance values caused by all decomposition techniques for DS41 data set. As the number of the input patterns is increased, load imbalance of the tree-based decomposition technique increased.

Tree-based decomposition techniques cannot balance computational workload, because, they try to predict workload of a cell, and their cell distribution policy is determined by this prediction. The most important elements of work prediction of a cell are level of the cell on the pattern tree and number of centroids related with that cell. Because of the some constraints these two prediction parameters lose their effectiveness so that, the number of the pruning patterns varies from processor to processor. The load imbalance problem will be analyzed in the next section in detail.

4.4 Comparisons and Analysis of the Experimental Results

We will compare the results of the whole experiments in this section. All of the evaluating metrics will be examined, and all of the main decomposition techniques will be compared with each other. The reasons behind the results will be explained. In all experiments, we have changed three parameters, which are the number of the input clusters, the number of the input patterns and the number of the processors. The effects of the changes are observed.

First metric is execution time. When we examine the execution time tables, we can observe that *tree-based* decomposition techniques are superior to the others. *Tree-based* decomposition techniques try to decompose working space into the compact subspaces. There will be more pruning of candidate cluster centroids at the upper levels of the local tree in case of compact subspace, because many cluster centroids will be apart from the compact space assigned to a processor. The more pruning of candidate cluster centroid at the upper level of the pattern tree will result with less number of distance calculations, in other words smaller execution times. On the other hand, *random* decomposition technique decomposes sparse and larger subspaces, in this case, the pruning will shift towards to the leaves of the pattern tree, which will result with more distance calculations. Because of the more distance calculations, the execution times achieved by *random* pattern decomposition technique are worse than other decomposition techniques. *Stripwise* decomposition technique has better execution times than the *random* decomposition technique, because it decomposes more compact subspaces than *random* decomposition technique, by disturbing patterns stripwise manner. Because of the reasons that we have explained in the chapter 3, subspaces decomposed by the *stripwise* decomposition, are not as compact as subspaces that are decomposed by *tree-based* decomposition techniques so that number of the distance calculations performed by *stripwise* decomposition technique are more than the tree-based pattern decomposition techniques.

Tree-based decomposition techniques are apart from each other in two points. One of these points is the pattern tree child numbering style the other one is the cost functions. We have used two child numbering styles, *non-uniform* and *uniform numbering*, and four cost functions, each of which determines the number of local patterns with different formulas. *Tree-based* decomposition techniques do not differs from each other very much especially if the number of the input pattern is increased. Generally, The *NonUniformNumbering* technique is better execution times than the others. The *NonUniformNumbering* technique numbers a cell's children by looking its parent numbering style and by looking which child of its parent. This numbering style ensures physical contiguity of the decomposed subspace so that, this technique generally yields more compact subspaces. In *UniformNumbering* all of the children of a cell are numbered with a chosen numbering style such as clockwise numbering. This uniform style may divide subspace into two sub subspaces. This apartness of the sub subspaces yields with less pruning of the candidate clusters, and higher number of distance calculations.

We have explained cost function in details in the previous chapter. They try to predict the cost of a cell by taking into account different parameters. *Tree-based* techniques with *LCF* have better execution times and it shows that the level of the pattern tree can be considered as the indicator of the cost of a cell. *CCF* and *CLCF* have also better execution times than the *SCF*. Their performances are directly related with the initial cluster centroids. In the each iteration of the algorithm, cluster centroids are changed and these changes lead different number of pruning of the centroids so that these functions cause the load imbalance since they do not consider the update cluster centroids..

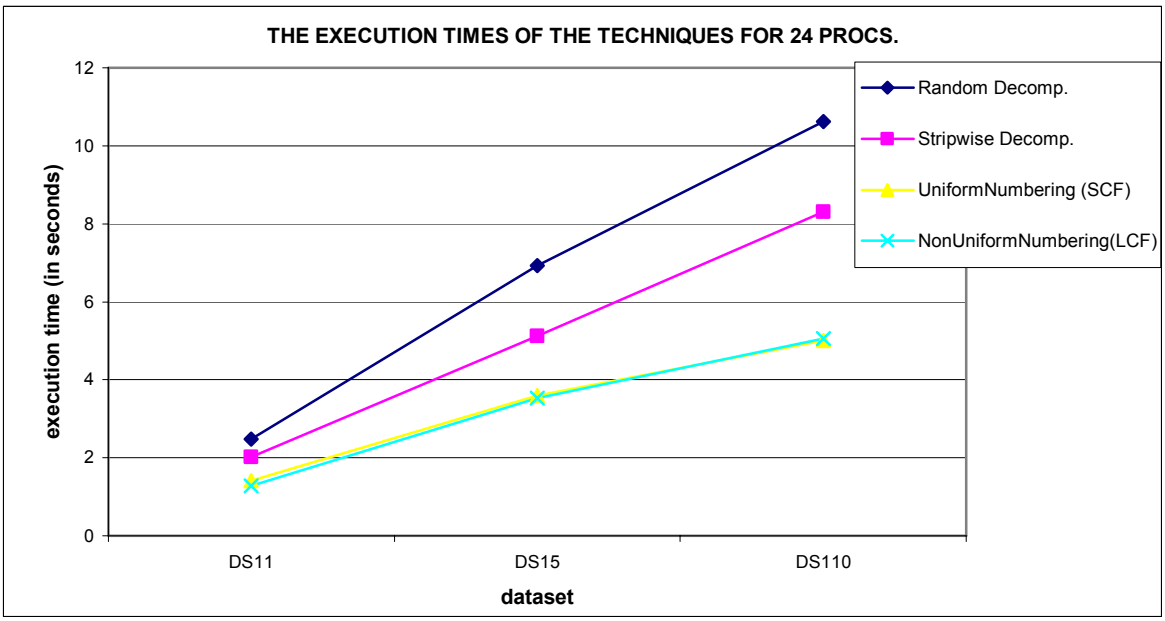


Figure 4.16: The comparison chart of the execution times of the decomposition techniques for three data sets each of which has different number of patterns. The number of the processors in the system is 24.

In the first step of the execution time experiments, we have increased the number of the input patterns. As we increased the number of the input patterns, it is natural that execution times of the all of the distribution methods are increased. But *tree-based* pattern decomposition techniques still have the best execution times (Figure 4.16). *Random* decomposition technique shows the worst execution times among all of the pattern decomposition techniques.

Another observation is that, when the number of the pattern is increased, execution times of the *tree-based* decomposition techniques do not differ very much. This case is related with the structure of the algorithm. The pattern tree is constructed with a predefined depth value, so two pattern trees whose depths are same but constructed with a different number of patterns will have the leaf cells with different pattern density. For example, if a tree is constructed with a higher number of patterns, its leaf cells might contain higher number of the patterns. *Tree-based* decomposition techniques distribute pattern tree cell by cell, each cell's cost is determined by the cost functions. All of the cost functions have some parameters such as level of the cell, number of the related centroids. When the number of the pattern per cell is increased so much, the effect of the parameters decreased, so that all of the cost functions approach the simple cost function. Therefore, as the number of the input patterns is increased, execution times of the *tree-based* techniques become closer to each other.

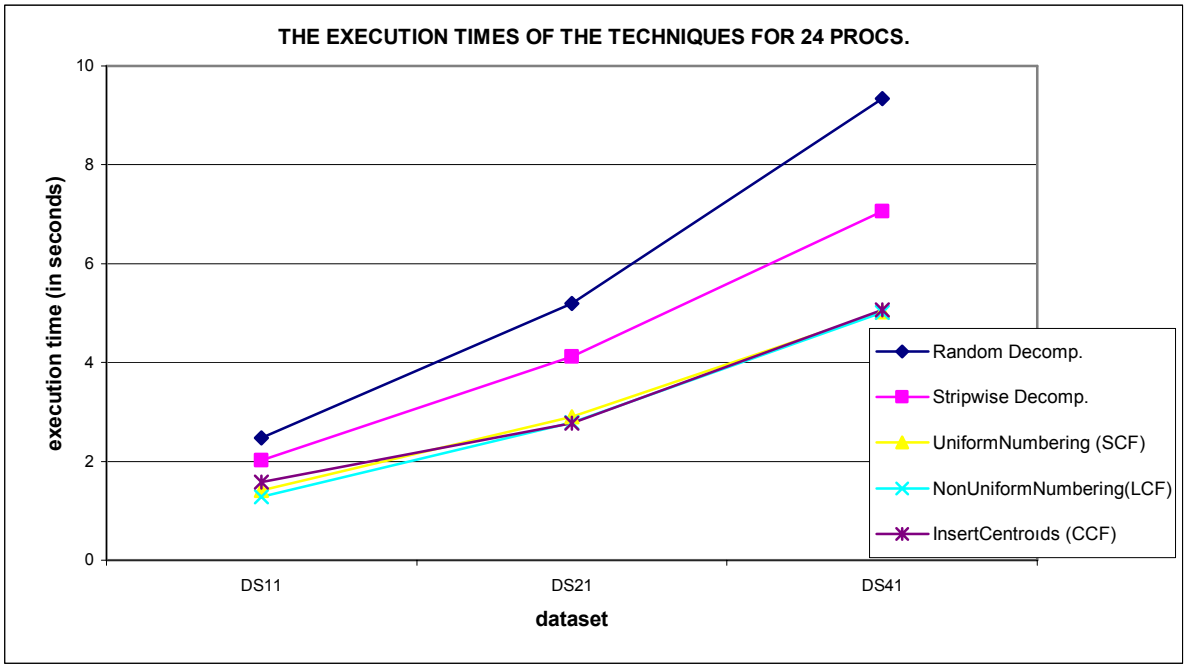


Figure 4.17: Comparison of the execution times of the decomposition techniques for three data sets each of which has different number of clusters. The number of the processors in the system is 24.

In the next step of the execution time experiments, the number of the cluster centroids is increased. The increment in the number of the cluster centroids causes higher execution times as we expected. *Stripwise* decomposition technique has better execution times than the *random* decomposition technique, but has worse execution times than the *UniformNumbering (SCF)* decomposition technique, which has usually the worst execution times among all

tree-based decomposition techniques. We have also observed that, the number of the cluster centroids is increased, the *InsertCentroid (CCF)* decomposition technique's performance slightly increased. Even it has the best execution time for data set *DS15* (Figure 4.17).

The number of the clusters is increased at most four times and the number of the input patterns is increased at most ten times. When the number of the clusters is increased four times, execution times increased almost linearly. When the number of the input patterns is increased ten times, execution times increased four times. If the number of the patterns is increased, pruning performance of the algorithm improved. The higher number of the patterns is assigned to a cluster without any distance calculations when there is only one candidate cluster centroid. On the other hand, if we increase the number of the cluster centroids, since there will be more candidate cluster centroids, performance of the pruning algorithm decreases while the number of the distance calculations is increasing. Therefore, the algorithm will have higher execution times.

In the k-means algorithm, the number of the distance calculations determines the execution time of the algorithm. The number of the distance calculations is directly related with the pattern decomposition technique. In our experiments, total number of distance calculations consists of *Cell-Centroid* and *Pattern-Centroid* distance calculations, which are explained in the previous sections.

It is natural that, *tree-based* decomposition techniques perform less number of distance calculations, because their execution time better than the other techniques. The *NonUniformNumbering (LCF)* decomposition technique performs the least number of distance calculations among all of the decomposition techniques. Although, UniformNumbering (*SCF)* performs the highest number of the distance calculations among *tree-based* decomposition techniques, there is a great difference in distance calculations with the other decomposition techniques.

We observed that, if the number of the processors is increased, total number of the distance calculations increased. Important point of the observation is that when we increased the processor number from one to twenty four, total number of distance calculation, performed by *random* decomposition, is increased almost seven times, but this increasing ratio is less than two times for the *tree-based* decomposition techniques. This ratio is another

indicator of the successful pruning process achieved by the *tree-based* decomposition technique. When *random* decomposition technique assigns subspaces to processors, each processor may have local working space as big as the whole working space, since assigned patterns are chosen randomly. As we explained before, sparse subspaces cause higher number of distance calculations. The processor, whose local patterns are assigned by the *random* decomposition, performs more number of distance calculations than the processor whose local patterns are assigned by one of the *tree-based* decomposition techniques. So, increasing the number of processor will affect total distance calculations of *random* decomposition very much.

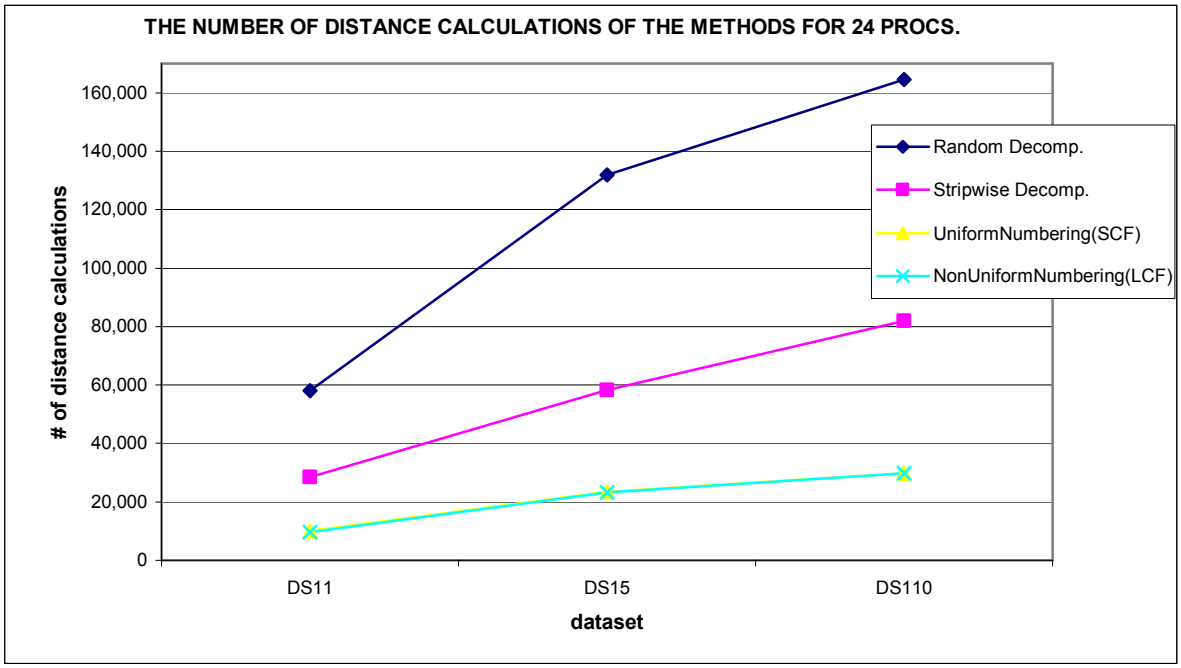


Figure 4.18: The comparision chart of the total distance calculations of the decomposition techniques for data sets DS11, DS15 and DS110. The number of the processors in the system is 24. Each of the data sets contains different number of patterns.

We have repeated the distance calculation experiments with three different datasets while we are using the same number of initial centroids. As the number of the patterns is increased, total number of the distance calculations, performed by all decomposition techniques, also increased. As shown in Figure 4.18, increasing the number of the input patterns is not changing the superiority of the *tree-based* decomposition techniques.

In the next step of the distance calculations experiments, the number of the input clusters is increased. *Random* decomposition technique performed the highest number of the distance calculations for all datasets. Although, *stripwise* decomposition technique performed

less number of distance calculations than the *random* decomposition technique, its values almost four times higher than the values of the *tree-based* decomposition techniques (Figure 4.18). The *NonUniformNumbering (LCF)* performed the least number of distance calculations among all decomposition techniques, but its values do not differ from other *tree-based* decomposition techniques very much.

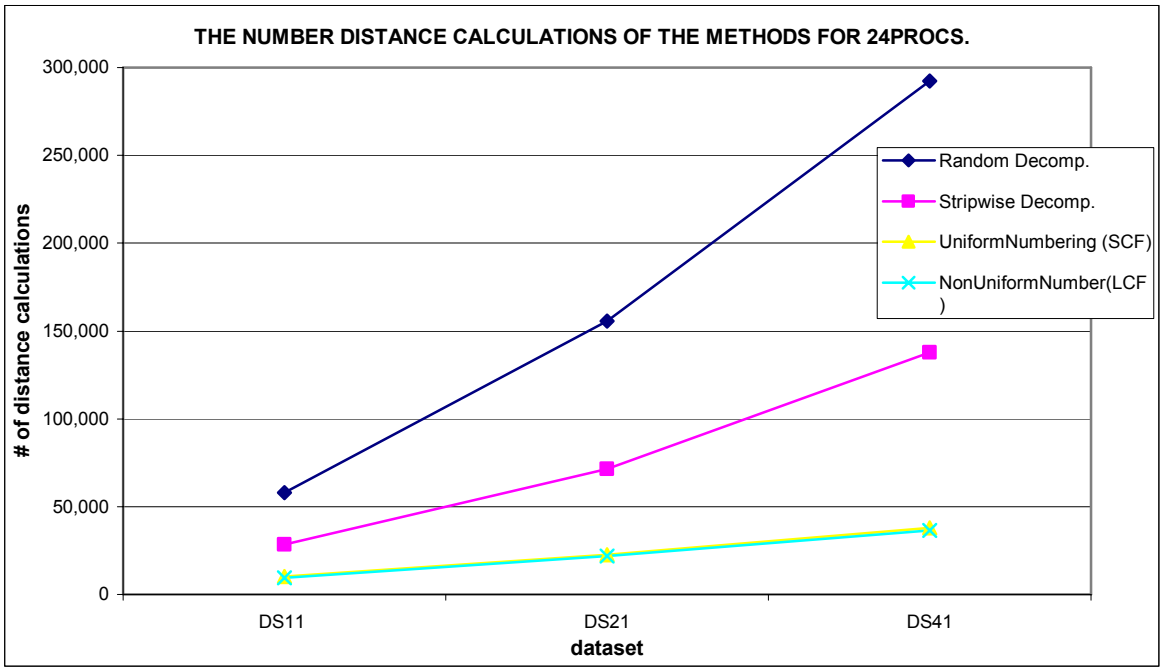


Figure 4.19: The comparison chart of the total distance calculations of the decomposition techniques for data sets DS11, DS21 and DS41. The number of the processors in the system is 24. Each of the datasets contains different number of cluster centroids.

We can summarize the performance of the parallel system by examining the *speedup* values of it. Speedup is directly related with execution times and indirectly related with the number of the distance calculations. *Tree-based* decomposition techniques have superiority in execution times and number of distance calculations so, it is natural that their speedup values will be better than the other techniques. Speedup values of the all of the *tree-based* pattern decomposition techniques do not differ from each other very much. *UniformNumbering (LCF)* technique has slightly better values. The *Random* decomposition has the worst speedup values.

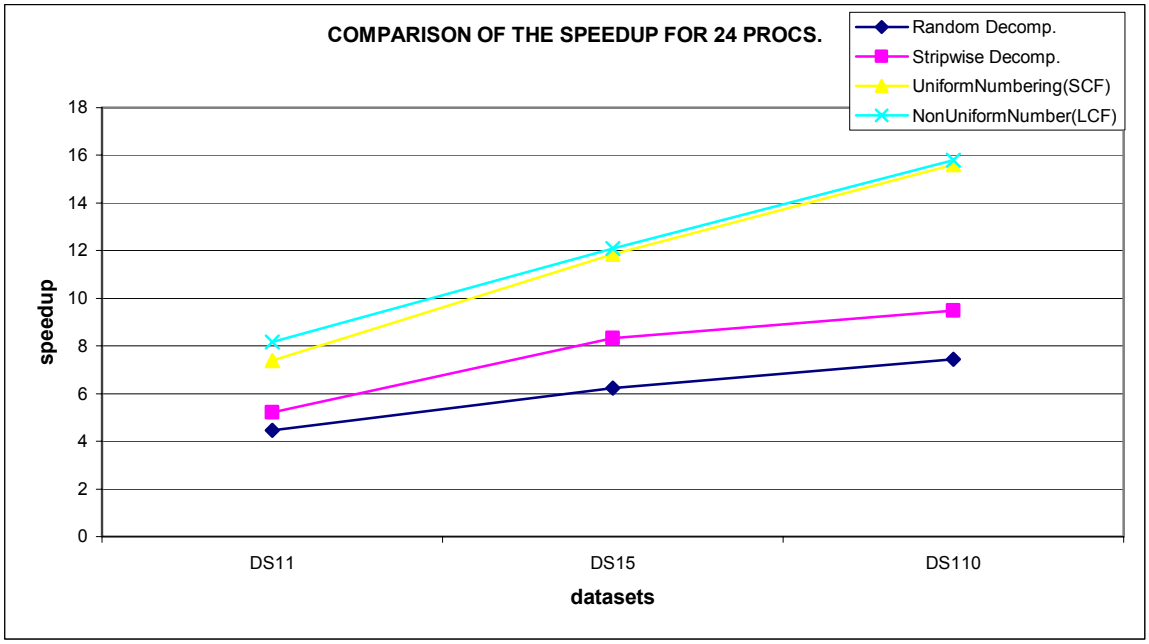


Figure 4.20: The comparison chart of the speedup values of four decomposition technique for data sets DS11, DS15 and DS110. The best speedup values are achieved by the tree-based decomposition techniques.

Theoretically, larger instance of the same problem yields higher speedup for the same number of the processors. Figure 4.20 is comparison chart of the speedup values of the four decomposition techniques. If we examine the chart, as the number of the input patterns is increased, speedup is also increased, as the theory implies. It is clear that *tree-based* decomposition techniques have better speedup values than the others. Although, *UniformNumbering (SCF)* has the worst speedup values among all of the *tree-based* decomposition techniques, its speedup values are better than both *stripwise* and *random* decomposition techniques. Increasing the number of the patterns, improving the speedup values of all decomposition techniques.

The last examined metric is load imbalance. The load balance can be defined as the ratio of maximum load difference to average load. In the experiments, computational load balance is calculated by using total distance calculations, which is performed by each processor. The average number of distance calculation of the parallel system is calculated, and

the maximum number of the distance calculation is found. The ratio between these two values represents the load imbalance of the system.

Random decomposition technique has the best-balanced computational workload. Its load balance is not effected by neither increasing number of the patterns nor increasing the number of the cluster, whereas the other decomposition techniques has been effected very much by increasing two input parameters. *Random* decomposition technique, as we explained previous section, assigns to processors equal number of patterns that are scattered approximately same working area. Every processor, which is assigned a subspace with same number of patterns scattered the approximate size of area, will perform same number of pruning activity which results with almost the same number of distance calculations. When the number of distance calculations performed by processors does not vary very much, the parallel system has better balanced computational work.

The experiments have shown that *tree-based* decomposition and *stripwise* decomposition technique are associated with load imbalance problem. So far, *tree-based* decomposition techniques have shown better performance, which depends on the successful pruning activity achieved by decomposition policy. The *tree-based* decomposition techniques assign different number of patterns to each processor. The local number of the patterns of the processor is determined by a cost function. Also, local patterns of processors are scattered over varying size of area. Every processor, being assigned a subspace with different number of patterns scattered to different size of area, will perform different number of pruning activity and different number of distance calculations, which will result in computational load imbalance. Increasing the number of patterns will cause increasing pruning activity. The difference between numbers of distance calculations of two processors might increase since each of the processor may have different pruning ratios. Because of the changing pruning ratios of two processors, distance calculation difference increased, which causes increasing load imbalance.

In *tree-based* decomposition technique, the number of the local patterns for a processor is determined by four cost functions. The detailed information about cost functions is given in the previous chapter. Experiments have shown that, there is a slight difference between all of the cost functions, because all of them are static cost functions, and cannot reflect the load changes to current situation. *SCF* has usually the worst load imbalance among

all cost functions. Because giving equal number of local pattern to each processor does not guarantee the load balance. *CCF* has better load balance than the *SCF*. *CCF* tries to predict to load imbalance by using initial cluster centroids. But, the location of the centroids may change for every iteration, which will lead changes in the cost of the cells. The prediction, made by the *CCF*, is static so that cost changes occurring with the changes of the cluster centroids, is not taken into account thus *CCF* is not as successful as *CLCF*. *CLCF* predicts cost of a cell by two parameters, level of cell and number of related centroids. Because of the reason explained above, taking into account the number of the centroids may decrease its success. But prediction, made by two parameters, can be better than the other cost function's predictions. *LCF* sometimes better than the *CLCF*, but as the number of the input parameters increase their results get closer to each other

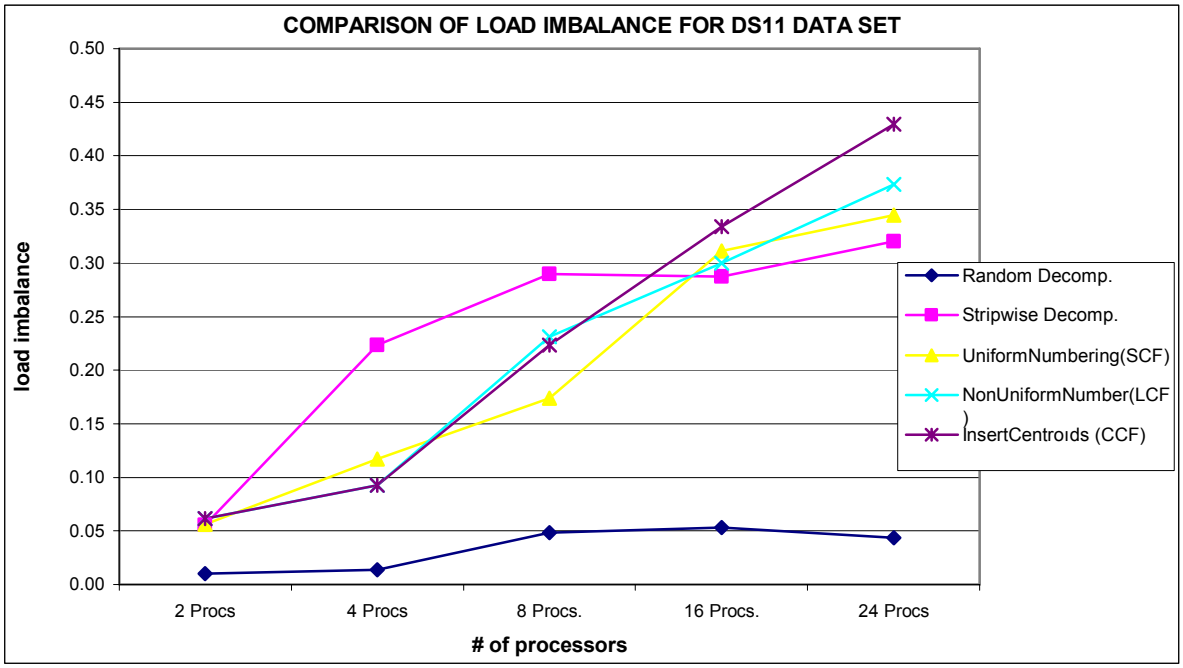


Figure 4.21: The comparison chart of the load imbalance of four methods as the number of the processors increases. Input data set is DS11. The random decomposition technique is not effected with increasing number of the pocessors.

Another input parameter that effects the load imbalance is the number of the processors. As shown in the Figure 4.21 that as the number of the processors is increased *random* decomposition technique is not affected very much. Because no matter how much the number of processors is increased, just number of the local pattern is decreased, but the size of their scattering area possibly very close to each other. This means that, they will perform

almost the same number of distance calculations. On the other hand, increasing the number of the processor effects *tree-based* decomposition techniques very much. It is straightforward that, if the number of the processor is increased, sizes of the subspaces and the number of patterns are decreased. Therefore, their pruning performances increased. Because of the distribution of the patterns, subspaces will be smaller in a different ratio. In other words, processors perform different number of pruning and the different number of distance calculations. As a result, *tree-based* and *stripwise* decomposition techniques are effected very much, when the number of the processor is increased.

As a summary, *tree-based* decomposition techniques have superiority on all of the metrics except load balance. *Tree-based* decomposition techniques try to decompose compact subspaces for each processor. Its pattern distribution policy decreases the number of the distance calculations. When the number of the distance calculations performed by the processors decrease, total execution times also decrease and the parallel system has the better speedup values. The *NonUniformNumbering* technique usually has better performance than the *UniformNumbering* technique, because its pattern tree child numbering style ensures physical contiguity in the subspaces, with this contiguity, processors perform better pruning activity. The *random* decomposition technique has the best-balanced computational load. For all metrics except load balance, the *stripwise* decomposition technique has superiority over *random* decomposition technique.

CHAPTER 5

CONCLUSION

In this thesis, we presented novel pattern decomposition technique to improve the performance of the parallel tree-based k-means algorithm on distributed memory machines. Our experimental results demonstrated that, proposed pattern decomposition technique ensures to parallel tree-based k-means run faster by decreasing the total number of the distance calculations performed by the algorithm.

The main computational work of the tree-based k-means is the distance calculations. If the number of the distance calculations decreases, the algorithm runs faster. The *tree-based* decomposition techniques try to decrease the total number of the distance calculations by assigning processors compact subspaces. The compact subspace improves the performance of the pruning function of the tree-based k-means algorithm. When the performance of the pruning algorithm increases, the number of the candidate centroids decreases, thus the total number of the distance calculations also decreases. The tree-based technique constructs pattern tree and decomposes the pattern tree to processors. Thus, processors are assigned smaller subspaces, then they have better execution times.

In the proposed technique pattern tree is constructed as a quadtree. We constructed pattern tree by using two different child numbering style. The child numbering style of the pattern tree related with the physical contiguity of the subspaces. The first style is the uniform numbering, in which the children cells of the pattern tree are numbered with the same ordering such as starting left-bottom children and follow the clockwise direction. In the second style, the children cells of the pattern tree are numbered with different ordering. The child numbering style of a cell is determined according to child numbering order of the parent

of the cell and which child of its parent in that ordering scheme. The nonuniform numbering style makes subspaces contiguous. The *tree-based* decomposition technique with nonuniform numbered pattern tree achieves better results, because it improves the performance of the pruning function by decomposing contiguous subspaces. The uniform numbering of the pattern tree decomposes some uncontiguous subspaces so that the performance of the owner processor of the uncontiguous subspace decreases.

The *tree-based* technique decomposes the pattern tree leaf cell by leaf cell. In order to predict cost of a cell, we have used four cost functions in the *tree-based* decomposition technique. The simple cost function predicts cost of a cell by the number of patterns belonging to the cell. The *SCF* assigns equal number of patterns to processors. The level cost function predicts the cost of a cell by considering the height and the pattern number of the cell. The centroid cost function inserts the initial centroids to the pattern tree and predicts the cost of a cell by considering the number of the related centroids and the pattern number of the cell. The level and centroid cost function predicts the cost of a cell by considering the related centroid number of the cell, the height and the pattern number of the cell. The *LCF* achieves better result than the other cost functions. Its performance is affected neither by the increase in the number of the pattern nor by the increase in the number of the candidate clusters. The *CLCF* usually gets better result than the *SCF*, but slightly worse results than the *LCF*. The successes of the *CCF* and *CLCF* are related to the initial position candidate centroids. They do not consider the centroid updates, which are done at each iteration, so that their performances decrease. As the number of the candidate clusters is increased, the *CCF* function gets slightly better result. The *SCF* usually gets worse results, because giving equal number of the pattern does not guarantee that each processor performs same number of the distance calculations.

The experimental results have shown that, *tree-based* pattern decomposition techniques always make tree-based k-means algorithm performing less number of the distance calculations. The *NonUniformNumbering (LCF)* decomposition technique usually makes the algorithm performing the least number of distance calculations. Because, it decomposes contiguous subspaces, by this way it increases performance of the pruning function, which yields to less number of distance calculations. The *UniformNumbering (SCF)* makes the *tree-based* k-means algorithm performing the highest number of the distance calculations among the *tree-based* decomposition techniques. Because, it cannot ensures the physical contiguity

of the subspaces. Although, the *UniformNumbering (SCF)* causes the highest number of the distance calculations among the *tree-based* decomposition techniques, its number of distance calculations is better than the number of the distance calculations of the *random* decomposition and the *stripwise* decomposition techniques. The *random* decomposition technique always makes the algorithm performing the highest number of the distance calculations. The *random* decomposition technique distributes patterns randomly , thus, decomposed subspaces will be large subspaces. The performance of the pruning function will decrease with a large subspaces, thus the number of the distance calculations will increase. The *stripwise* decomposition technique makes the algorithm performing less number of distance calculations than the algorithm whose subspaces are decomposed by the *random* decomposition technique.

When the number of the input patterns is increased, the number of the distance calculations performed by the *random* decomposition technique increases drastically. However the number of the distance calculation performed by the *tree-based* decomposition techniques increases slightly. The superiority of the tree-based decomposition techniques does not change, when the number of the candidate cluster centroids is increased.

The *tree-based* k-means algorithm, whose patterns are distributed by tree-based techniques, is the faster than the algorithm whose pattern decomposition technique is the *random* decomposition or the *stripwise* decomposition technique. Since the tree-based k-means algorithm with the *tree-based* pattern decomposition techniques performs the least number of the distance calculations, it is natural that the tree-based decomposition techniques make tree-based k-means running faster. Also the tree-based k-means algorithm has better speedup values with tree-based pattern decomposition techniques.

The *random* pattern decomposition technique has no imbalance problem, but *tree-based* and the *stripwise* decomposition techniques associated with the load imbalance problem. The *random* decomposition technique distributes patterns randomly, so that each processor has the same number of patterns, which are scattered over almost equal size subspaces. Thus the number of the distance calculations performed by the processors will be closer to each other. However the *tree-based* decomposition techniques decompose different number of patterns, which are scattered varying size of subspaces. This decomposition leads the varying number of the distance calculations for each processors. The increase in the

number of the patterns and the number of the candidate clusters does not effects the load balance of the random decomposition.

As a result, tree-based decomposition techniques make the tree-based k-means algorithm running faster, by decomposing the smaller and contiguous subspaces to the processors.

REFERENCES

[1]. M. V. Joshi, E. Han, G. Karypis, and V. Kumar "Parallel Algorithms in Data Mining" *University of Minnesota,1999.*

[2]. A. Gürsoy, İ. Cengiz, "Parallel Pruning for K-Means Clustering on Shared Memory Architectures" *LNCS, Vol 2150, pp. 321-325, 2001.*

[3]. A.K. Jain, M.N. Murty, and P.J. Flynn "Data Clustering: A Review" *ACM Computing Surveys, Vol. 31, No. 3, pages 79-87. , September 1999.*

[4]. T. Zhang, R. Ramakrishnan, M. Livny "BIRCH: An Efficient Data Clustering Method for Very Large Databases " *In Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD 96), pages 103-114,1996.*

[5]. P.S. Bradley, U.M Fayyad "Refining Initial Points for K-Means Clustering" *University of Minnesota, 1998.*

[6]. A.K Jain, R. C. Dubes *Algorithms for Clustering Data,* Prentice-Hall Inc., Upper Saddle River, NJ, 1988.

[7]. U Fayyad,, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.) *Advances in Knowledge Discovery and Data Mining,* MIT Press, 1996.

[8]. J. Banfield and A. Raftery, "Model-based Gaussian and non-Gaussian Clustering", *Biometrics, vol. 49, pp. 15-34, 1993.*

[9]. D. Fisher. "Knowledge Acquisition via Incremental Conceptual Clustering", *Machine Learning, 2:139-172, 1987.*

[10]. K. Fukunaga, "Introduction to Statistical Pattern Recognition", *Academic Press, San Diego,CA, 1990.*

[11]. S. Z. Selim and M. A. Ismail, "K-Means Type Algorithms: A Generalized Convergence Theorem and Characterization of Local Optimality",*IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 1, 1984.

[12]. B.W. Silverman, *Density Estimation for Statistics and Data Analysis*, London: Chapman & Hall, 1986.

[13]. D.Fasulo, "An Analysis of the Recent Work on Clustering Algorithms", *Department of Computer Science & Engineering Technical Report, University of Washington, Seattle, 1999.*

[14]. R.C. Dubes, "How many clusters are best? – An experiment", *Pattern Recognition, 20, 6, 645-663, 1987.*

[15]. K. Alsabti, S. Ranka, and V. Singh, "An Efficient K-Means Clustering Algorithm. " h*ttp://www.cise.ufl.edu/~ranka*/, 1997.

[16]. D. Foti, D. Lipari, C. Pizzuti, and D. Talia, "Scalable Parallel Clustering for Data Mining on Multicomputers", *ISI-CNR c/o DEIS, UNICAL Rende (CS), Italy, 1999.*

[17]. B. Monien, R. Feldmann, R. Klasing, R. Luling, "Parallel Architectures: Design and Efficient Use", *Proc of Symposium on Theoretical Aspects of Computer Science, 1993.*

[18]. M.V. Joshi, S. Ham, G. Karypis, V. Kumar, "Parallel Algorithms in Data Mining", University of Minnesota, Minneapolis, 1999.

[19]. L.Pangfeng ," The Parallel Implementation of N-body Algorithms " *DIMACS center challenges Rutgers University, Piscataway, New Jersey,1999.*

[20]. R. Lüling, B. Monien, F. Ramme, "Load Balancing in Large Networks: A Comparative", *Proc 3rd IEEE Symp. Parallel and Distributed Processing, 686-689, 1991.*

[21]. R. Lüling, B. Monien, "Load Balancing for Distributed Branch & Bound Algorithms" *Proc. 6th IPPS, pp. 543-549, 1992.*

[22]. M.S. Chen, J. Han, and P.S. Yu. "Data mining: An overview from database perspective", *IEEE Transactions on Knowledge and Data Eng., 8(6):866-883, December 1996.*

[23]. P.S. Jaswinder, C. Holt, T.Tokai, "Load Balancing and Data Locality in Adaptive Hierarchical N-Body Methods: Barnes-Hut, Fast Multiple, and Radiosity." *Stanford University, Stanford,1994.*

[24]. H. Samet, *The Design and Analysis of Spatial Data Structures,* Addison-Wesley, 1989.

[25]. LAM/MPI Parallel Computing, http://www.lam-mpi.org.

[26]. V. Kumar, A. Grama, A. Gupta, K. George. *Introduction to Parallel Computing Design and Analysis of Algorithms* The Benjamin/Cumming Publishing Company, 1994.

[27]. D. Judd, P. McKindley, A. Jain. "Large-Scale Parallel Data Clustering." *Proc. Int'l Conference on Pattern Recognition, pp 78-86, August 1996.*