

FRAMEWORK FOR FASTER IMPLEMENTATION OF UNSUPERVISED CLUSTERING ALGORITHMS

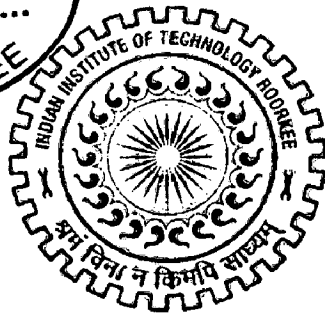
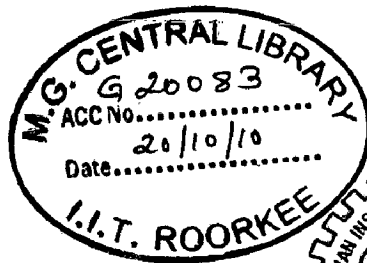
A DISSERTATION

*Submitted in partial fulfillment of the
requirements for the award of the degree*

of
MASTER OF TECHNOLOGY
in
INFORMATION TECHNOLOGY

By

ANANT BHUSHAN



DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2010

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in the dissertation entitled, "***Framework for Faster Implementation of Unsupervised Clustering Algorithms***", which is submitted in the partial fulfillment of the requirements for the award of degree of ***Master of Technology in Information Technology***, submitted in the Department of Electronics and Computer Engineering, Indian Institute of Technology Roorkee (India), is an authentic record of my own work carried out under the guidance of Dr. Kuldeep Singh, Professor, and Dr. Ankush Mittal (Ex-Faculty) Department of Electronics & Computer Engineering, Indian Institute of Technology Roorkee.

The matter embodied in the dissertation report to the best of ~~my~~ knowledge has not been submitted for the award of any other degree elsewhere.


Dated : June, 2010

Place : Roorkee


(Anant Bhushan)

CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.


Dr. Kuldeep Singh

Professor

Dept. of E&CE

IIT Roorkee


Dr. Ankush Mittal

Ex-Associate Professor

Dept. of E&CE

IIT Roorkee

ACKNOWLEDGMENTS

It gives me immense pleasure to take this opportunity to thank and express my deep sense of gratitude to my guide Dr. Kuldip Singh, Professor, Department of Electronic and Computer Engineering, Indian Institute of Technology Roorkee, for his valuable guidance. I would also like to thank my ex-guide Dr. Ankush Mittal, for his encouragement during the course of this work.

I would like to thank the Institute Computer Center for making its resources available for the purpose of this work and the staff of the Department of Electronics & Computer Engineering for their cooperation during this dissertation work.

My special sincere heartfelt gratitude to my family, whose best wishes, support and encouragement has been a constant source of strength to me during the entire work.

Finally, I would also like to thank all my friends and seniors for their support and valuable suggestions.

(Anant Bhushan)

Abstract

Clustering, particularly unsupervised clustering is central to a large number of computing application which involve machine learning and information retrieval. Algorithmic methods of improving the execution times by using filtering algorithm and kd-trees have been successful but do not provide scope for further improvements. The emergence of multi-core processors and their easy availability and low cost has made it possible to have increased computing power. The need of the hour is to have algorithms that can harness the increased computing power available at our disposal.

In this work a novel approach has been presented which can reduce the running time of the clustering algorithms by exploiting the parallel computing architectures available today. We utilize the MPI libraries for creating parallel execution threads on multicore processors. Our approach involves adding a pre-processing and post-processing step to the parallel implementation of clustering using filtering algorithm. The preprocessing step is for finding groups of dimensions which have similar characteristics and which can therefore yield better quality clusters. These sub-groups of similar dimensions are clubbed together for parallel clustering operations in the subsequent steps, based on a similarity metric. The sub-groups of dimensions are created with an overlapping dimension among adjacent groups to facilitate merging of cluster centers during the post-processing step. The parallel clustering step produces overlapping cluster centers for the sub-groups of dimensions. The post-processing step takes the clusters created by the sub-groups of dimensions and merges the cluster centers based on the overlapping dimensions.

The feasibility of the framework has been demonstrated through an implementation on multi-spectral image clustering using the filtering algorithm and significantly reduced running times were obtained. The pre-processing step involved the calculation of the kurtosis of the image data for calculating the similarity metric and grouping into sub-groups. The overhead involved in execution of the pre and post-processing steps was less than one percent of the time taken for clustering the data in parallel.

Contents

Abstract	i
Candidate's Declaration	ii
Certificate	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures	vi
List of Tables	vii
1 Introduction	1
1.1 Unsupervised Clustering	1
1.2 Parallel Computing	2
1.3 Problem Statement	2
1.4 Organisation of the Report	3
2 Unsupervised Clustering	4
2.1 The ISOCLUS Algorithm	5
2.1.1 kd-trees	8
2.1.2 The Filtering Algorithm	8
3 Parallel Processing Architectures	12
3.1 Multi-core processors	12
3.2 Multicore Architecture	12
3.3 Clusters	14
4 Message Passing Interface	15
4.1 Communication Routines	15

4.1.1	MPI point-to-point communication routines	15
4.1.2	MPI Collective communication routines	16
4.1.3	Persistent Communication Requests	16
4.2	MPI Communicators, contexts, Groups	17
4.3	Programming Paradigms	17
5	The Proposed Framework	19
5.1	Pre-processing	20
5.2	Clustering	20
5.3	Post-processing	24
5.4	The Algorithm	26
6	Experimental Results	29
6.1	Pre-Processing step	29
6.2	Parallel Clustering	32
6.3	Post-processing step	33
7	Conclusion	34
	Bibliography	35
	List of Publications	37

List of Figures

2.1	Satellite Image Bands	6
2.2	Image after combining bands	7
2.3	An example of a kd-tree of a set of points in the plane	8
2.4	Classifying nodes in the filtering algorithm	10
3.1	Multi-Core Processor Architecture	13
5.1	The Pre-processing Step	21
5.2	Flow of Data through the proposed framework	23
5.3	The Post-processing Step	25
6.1	Comparison of Serial and parallel execution time	30

List of Tables

6.1	Results of Landsat data test	30
6.2	Image Statistics of ISODATA bands	31
6.3	Bands Sorted by Kurtosis of Image data	32
6.4	Parallel clustering execution times for different number of clusters . .	33

Chapter 1

Introduction

1.1 Unsupervised Clustering

Unsupervised clustering is an indispensable tool for machine learning in the present day world. It is used for geoscience and remote sensing applications for obtaining vegetation maps of interest. Its used for analyzing trends in sales of goods from data warehouses. It is also used in the field of bioinformatics for analysis of DNA micro array expression data, it is useful in high range resolution(HRR) radar range profiles [1]. In short unsupervised clustering finds its use in many fields where system automation is required and in cases where information about the existing data is scarce, or non existent, or expensive. In such cases unsupervised clustering is helpful towards the goal of unsupervised extraction of information from data. The goal of clustering is to simplify the characterization of the data into semantically meaningful groups. In the absence of apriori knowledge, the nearest neighbor proximity constraint is fundamental to any clustering algorithm. It is based on the rationale that for a good feature representation objects that lie together in the feature space would also lie together in the reality. Attempts have been made to improve the execution times of unsupervised clustering algorithms using techniques like filtering algorithms and kd-trees. But these algorithms do not provide further scope for reducing the execution time.

1.2 Parallel Computing

Moore's law had predicted that the chip manufacturing technology would be able to double the transistors on chip roughly every two years, and the prediction has stood good so far. Microprocessor technology has been upholding this prediction to improve its frequency by various techniques. However in the recent past microprocessors have hit a frequency wall, and not been able to take advantage of the predicted exponential growth. The outcome is the emergence of multicore processors, which offer the performance benefits of multi-processors on single chip. The presence of such architectures as common desktop processors has made it possible for hitherto time-consuming algorithms to be solved on simple desktop machines. The multicore processors allow program to leverage their computing power by various means like independent threads per core, or allow user to manipulate efficient data flow between cores, or provide a layer of software which manages the scalability of the cores. With the future micro-processor trends likely to increase number of cores as the only means of their increasing computing power, it has become necessary to ensure that important algorithms be parallelized to run on next generation of micro-processors.

1.3 Problem Statement

The problem of clustering points in a multidimensional space is one of a large number of optimization problems, such as the Euclidean k -median problem. in which we have to reduce the sum of distances to the nearest center. There are no efficient solutions available for general k , and some formulations are known to be NP-Hard. [2]

In this dissertation, *an attempt has been made to create a framework for parallel implementation of unsupervised clustering algorithms that can be applied to any algorithm that processes large number of dimensions.* Typically, dimensionality has been a curse and many efforts have been made to reduce the number of dimensions before clustering of the data [3], through the use of techniques like feature extraction, feature selection, Principal Component Analysis(PCA) etc. However almost all these techniques discard some amount of information based on certain criterion, which leads to information loss. In this work an approach has been presented that adds pre-processing and post-

processing steps to an unsupervised clustering algorithm so that they can utilize the parallel architectures available today like multicore systems, clusters etc. for solving the problem by harnessing the increased computing power at our disposal in form of the parallel architectures present.

1.4 Organisation of the Report

The organisation of the Dissertation report is as follows:

- Chapter 2 starts with an overview of the unsupervised clustering algorithms, with a background on the work done so far in the field. It proceeds to give an example of unsupervised clustering and then gives a brief overview of the ISOCLUS algorithm implementation with filtering algorithm and kd-trees.
- Chapter 3 provides a detailed overview of the multicore architectures and clusters that have been used in this dissertation.
- Chapter 4 covers the Message Passing Interface(MPI) that has been employed for creating parallel threads of execution for demonstrating the working of our proposed framework.
- Chapter 5 gives a detailed explanation of the framework that the author is proposing for increasing the performance of algorithms on parallel architectures.
- Chapter 6 discusses the implementation of the framework proposed on clustering using the filtering algorithm. The results obtained are also discussed.
- Finally Chapter 7 concludes the report and gives suggestions for future work.

Chapter 2

Unsupervised Clustering

Unsupervised clustering either does not require training data to be specified in advance, or performs a form of automated selection from the input data. With these techniques it is possible to create clusters even with estimates of the number of clusters present as the algorithms are able to narrow down to the number of naturally lying clusters based on various parameters like cluster size, standard deviation among the data points in the cluster.

The classical K-means algorithm are very dependent on selection of number of clusters and their initial center locations (seed locations). This is a heuristic, greedy algorithm for minimizing SSE (Sum of Squared Errors) hence it may not converge to a global optimum. The ISOCLUS algorithm is similar to the K-means procedure but at each iteration the various clusters are examined to see if they would benefit from being combined or split, based on a number of criteria: (i) combination if two cluster centers are closer than a pre-defined tolerance they are combined and a new mean calculated as the cluster center; (ii) if the number of members in a cluster is below a given level the cluster is discarded and the members re-assigned to the closest cluster; and (iii) separation if the number of members, or the standard deviation, or the average distance from the cluster center exceed pre-defined values then the cluster may be split. ISOCLUS is an automated procedure similar to the ISODATA procedure. It produces better quality clusters than K-means as it can adapt based on the characteristics of the input data.

An application of Unsupervised clustering is in finding out vegetation maps from satellite images [4]. Figure 2.1 shows three bands of data.

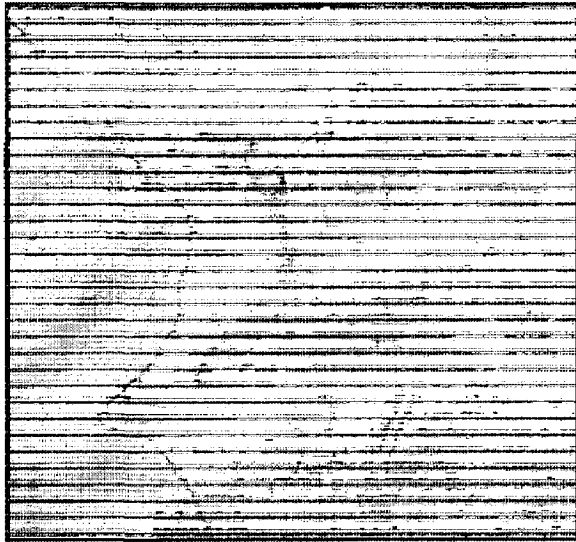
- Band 1 - Blue spectrum($0.45 - 0.52 \mu\text{m}$)[Figure 2.1(a)] and is meant for water body penetration, making it useful for coastal water mapping. Also, useful for soil/vegetation discrimination, forest type mapping and cultural feature identification.
- Band 2 - Green spectrum($0.52 - 0.60 \mu\text{m}$)[Figure 2.1(b)] is useful for measuring green reflection of vegetation. It can also be used for cultural feature identification.
- Band 3 - Red ($0.63 - 0.69 \mu\text{m}$)[Figure 2.1(c)] is sensitive to chlorophyll absorption region. It is useful for vegetation analysis and can even be used to differentiate plant types. It is also useful for cultural feature identification.

The Band data is black and white and does not contain any color information. What is done is that bands are merged by assigning them to any of the Bands of primary colors- Red, Green or Blue. These are the colors that our eyes actually see. These are called false color images because we can assign any band data to any primary color we want depending on what color we want to assign a particular property. For example, since Band 2 measures the reflection from vegetation, the value of a data at a point will indicate the amount of forest cover in the region. Thus by combining the knowledge from the various bands we can infer a large amount of information about a region, like, presence of water bodies, forest cover, kind of vegetation etc.

2.1 The ISOCLUS Algorithm

ISOCLUS is a clustering algorithm based on the ISODATA clustering algorithm [5], [6] with minor modifications. Like the k-means algorithm, ISOCLUS tries to find the best cluster centers through an iterative approach, until some convergence criteria are met. ISOCLUS uses different heuristics to determine when to merge or split clusters [5]. There are a number of user-supplied parameters. These include the following [7]:

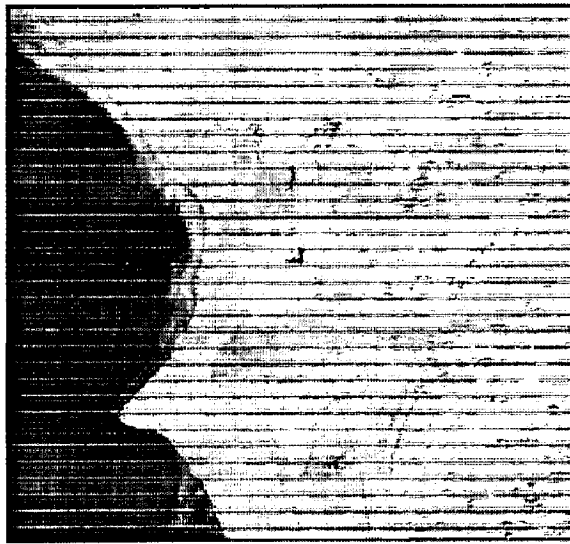
- The Desired number of clusters($Numclus$)



(a) Band 1 - Blue (0.45 - 0.52 μm)



(b) Band 2 - Green (0.52 - 0.60 μm)



(c) Band 3 - Red (0.63 - 0.69 μm)

Figure 2.1: Satellite Image Bands

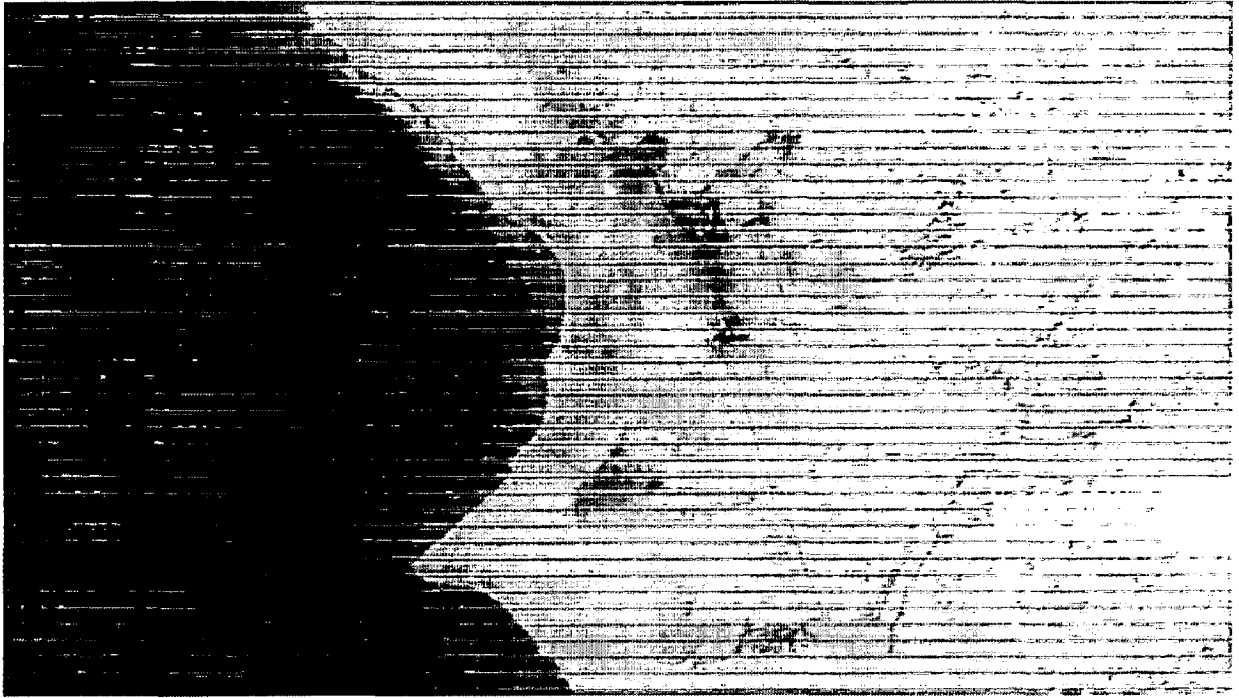


Figure 2.2: Figure obtained after merging Bands 1,2 and 3

- The minimum number of samples in a cluster($SampRm$)
- The Maximum number of Iterations($MaxIter$)
- Maximum standard deviation per cluster($StdDev$)
- The Lumping parameter($Lump$) and Maximum number of pairs that can be lumped per iteration($MaxPair$)

The algorithm can run very slowly on large data sets, hence in order to increase the speed of execution points are randomly sampled from the original data set and then randomly selects $NumClus$ centers from the samples. Depending upon the parameters passed to the algorithm the distance to the cluster centers are calculated and the points are assigned to the nearest cluster center. Next clusters with fewer than $SampRm$ are deleted. The cluster centers are then moved to the mean centroid of samples in the remaining clusters.

Next the algorithm considers merging or deleting clusters depending on the relationship between number of clusters and $Numclus$ and number of Iterations and $MaxIter$. For Each cluster the standard deviation along each of the coordinate axes is computed

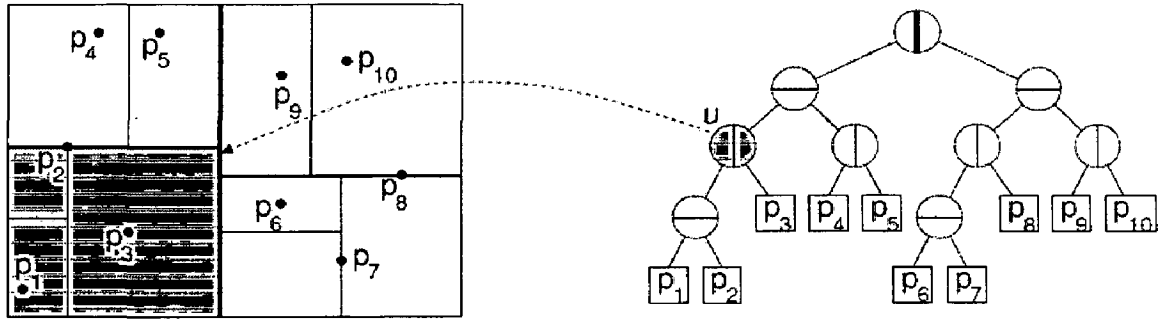


Figure 2.3: An example of a kd-tree of a set of points in the plane, showing both the associated spatial subdivision (left) and the binary tree structure (right).

and the maximum standard deviation is calculated. Based on this information and the values of *StdDev* and *Numclus* the clusters are split. And the parameters *Lump* and *MaxPair* and used to merge pairs of nearby clusters.

2.1.1 kd-trees

A kd-tree is a hierarchical decomposition of space into axis-aligned hyperrectangles called cells. Each node of the tree is implicitly associated with a unique cell and the subset of the points that lie within this cell. Each internal node of the kd-tree stores an axis-orthogonal splitting hyperplane. This hyperplane subdivides the cell into two subcells, which are associated with the left and right subtrees of the node. Nodes holding a single point are declared to be leaves of the tree. In Figure 2.3, the highlighted node u of the tree is associated with the shaded rectangular cell shown on the left side of the figure and the subset p_1, p_2, p_3 of points. It is well known that a kd-tree on n points can be constructed in $O(n \log n)$ time [8].

2.1.2 The Filtering Algorithm

The ISOCLUS algorithm is based on an enhancement of a simple and widely used heuristic for k-means clustering, sometimes called Lloyd's algorithm or the k-means algorithm [9] [10]. The Isoclus algorithm combines Lloyd's algorithm with additional mechanisms for eliminating very small clusters, splitting large clusters, and merging

nearby clusters. As with isoclus, the running time of Lloyd's algorithm is dominated by the time to compute the nearest cluster center to each data point. Naively, this would require $O(kn)$ time. Kanungo et al. presented a more efficient implementation of Lloyd's algorithm, called the filtering algorithm. Although its worst-case asymptotic running time is not better than the naive algorithm, this approach was shown to be quite efficient in practice. In the approach used by Nargess et. al. [7] points are assigned, not to their nearest neighbor, but to an approximate nearest neighbor. The filtering algorithm builds a standard kd-tree [11], augmented with additional statistical information [12].

The computational effort that is required to solve the problem depends on the time taken to compute distances and distortions among the points that need to be clustered. These steps take $O(kn)$ time in the original ISOCLUS implementation, where the implementation using the filtering algorithm has an order of execution of $O(k)$. This improvement is achieved by adapting the filtering algorithm to compute the desired information.

The implementation also reduces computation by using squared distances between points rather than euclidean distances.

Given a kd-tree for the data points S and the current set of k center points, the algorithm processes the nodes of the kd-tree in a top-down recursive manner, starting at the root [7]. Consider some node u of the tree. Let $S(u)$ denote the subset of points S that are associated with this node. If it can be inferred that all the points of $S(u)$ are closer to some center z_j than to any other center (that is, the node's associated rectangular cell lies entirely within the Voronoi cell of z_j), then we may assign u to cluster S_j . Every point associated with u is thus *implicitly assigned* to this cluster. (For example, this is the case for the node associated with cell a shown in Fig. 2.4.) If this cannot be inferred, then the cell is split, and we apply the process recursively to its two children. (This is the case for the node associated with cell b in the figure, which is split and whose two children are b_1 and b_2 .) Finally, if the process arrives at a leaf node, which contains a single point, then we determine which center is closest to the point, and assign its associated node to this center. (This is the case for the node associated with cell c of the figure.)

At the conclusion of the process, the filtering algorithm assigns the nodes of the kd-

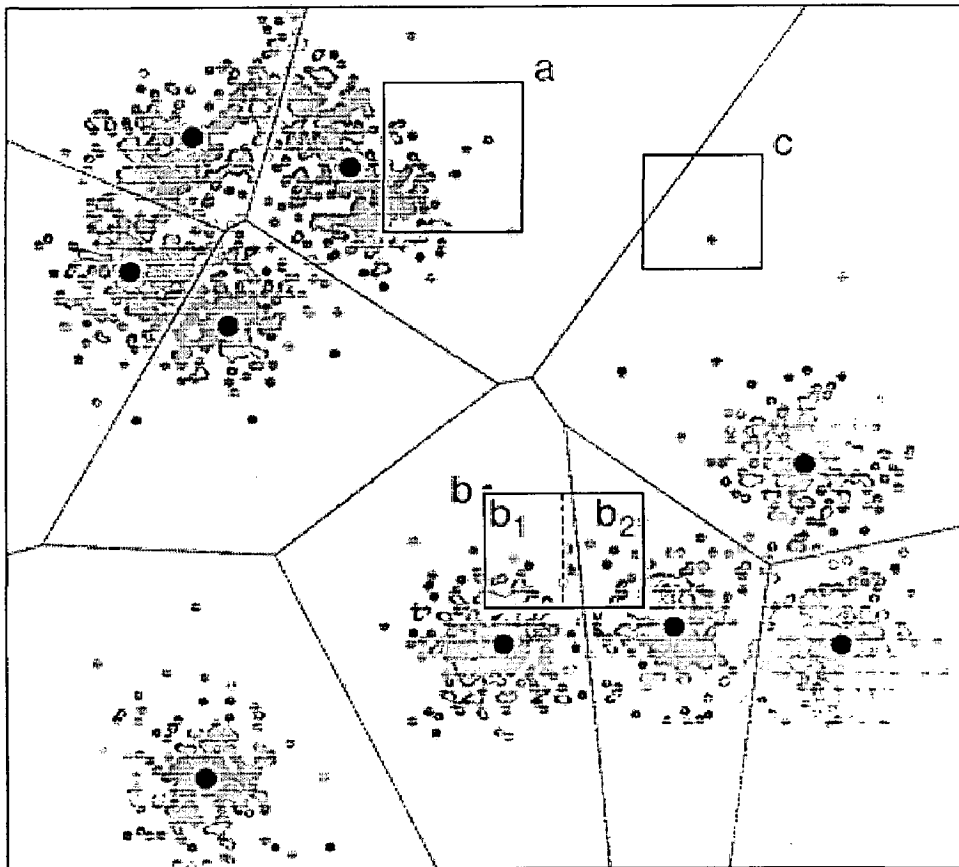


Figure 2.4: Classifying nodes in the filtering algorithm

tree to clusters in such a manner that every point of S is implicitly assigned to its closest cluster center. Furthermore, this is done so that the sets $S(u)$ assigned to a given cluster form a disjoint union of the associated cluster.

Chapter 3

Parallel Processing Architectures

3.1 Multi-core processors

Multi-core computers or systems containing Symmetric Multiprocessing(SMP) units are becoming ubiquitous and almost every new laptop, desktop or server machine is equipped with multiple cores. The shift from single-core to multi-core is mainly due to the difficulties of scaling processors to ever higher clock speeds, but the impact on the programming community is enormous. Existing applications can no longer run faster just because of a faster CPU, the programmer needs to write parallel programs in order to make use of the processing power available with the multiple cores/processing units.

3.2 Multicore Architecture

Symmetric multiprocessing or SMP involves a multiprocessor computer hardware architecture where two or more identical processors are connected to a single shared main memory and are controlled by a single OS instance. Most common multiprocessor systems today use an SMP architecture. In the case of multi-core processors, the SMP architecture applies to the cores, treating them as separate processors. Processors may be interconnected using buses, crossbar switches or on-chip mesh networks. The bottleneck in the scalability of SMP using buses or crossbar switches

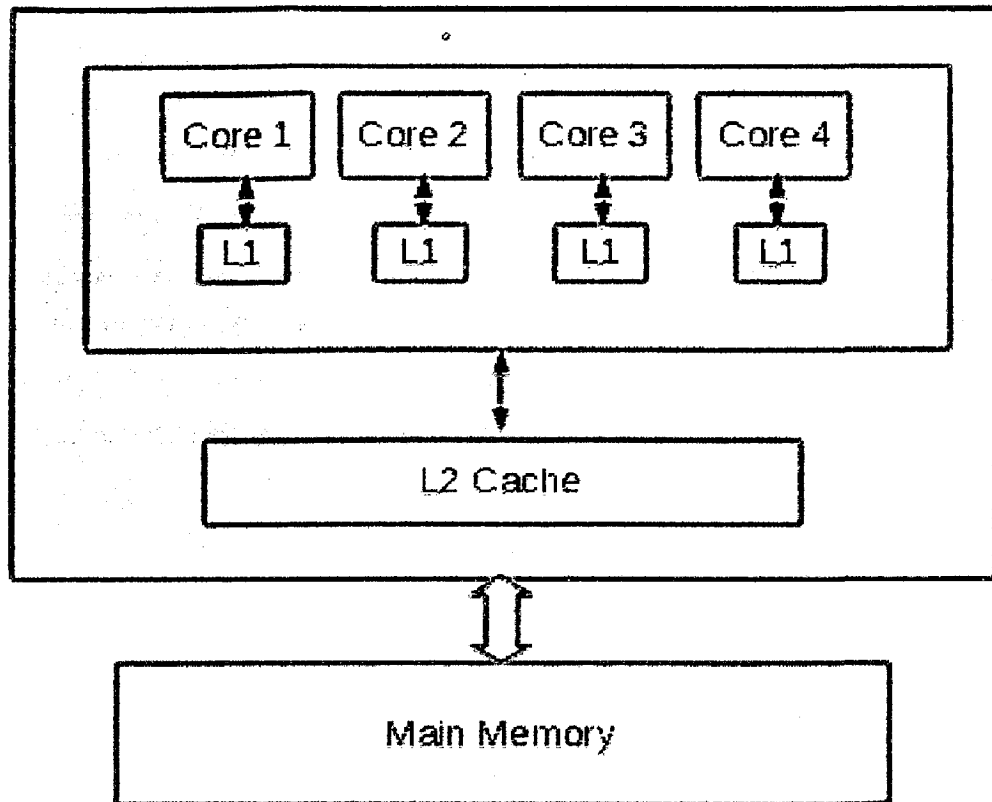


Figure 3.1: Multi-Core Processor Architecture

is the bandwidth and power consumption of the interconnect among the various processors, the memory, and the disk arrays. Mesh architectures avoid these bottlenecks, and provide nearly linear scalability to much higher processor counts [13]. The architecture of a symmetric multiprocessor is shown in Figure 3.1.

Uniprocessor and multiprocessor architectures require different programming models to utilize the power of the system completely. The amount of performance gained by the use of a multi-core processor depends very much on the software algorithms and implementation. Multicore and multithreaded CPUs have become the new approach to obtaining increases in CPU performance. Numeric applications mostly benefit from a large number of computationally powerful cores. [14]

The proximity of multiple CPU cores on the same die allows the cache coherency circuitry to operate at a much higher clock-rate than is possible if the signals have to travel off-chip. Combining equivalent CPUs on a single die significantly improves

the performance of cache snoop operations. Put simply, this means that signals between different CPUs travel shorter distances, and therefore those signals degrade less. These higher quality signals allow more data to be sent in a given time period since individual signals can be shorter and do not need to be repeated as often [15].

As shown in Figure 3.1, processors typically have two levels of cache. The level 1 cache is closer to the processor; level 2 is between level 1 and the primary memory. The level 1 cache is smaller but faster than the level 2 cache, and is often organized differently. For example, level 1 cache can be direct mapped, whereas the level 2 cache can be set-associative. And also the level 1 cache contains separate sections for instruction and data, while the level 2 cache is unified, containing both instruction and data [13]. The speed difference can be illustrated by the fact that the level 1 cache can typically be accessed in one or two clock cycles, while it takes order of 10 clock cycles to access the level 2 cache and 50 to 100 or more clock cycles to access the primary memory.

3.3 Clusters

Cluster computing is the technique of linking two or more computers into a network (usually through a local area network) in order to take advantage of the parallel processing power of those computers. MPI is a widely used library that facilitates communication between parallel programs written in C, C++, FORTRAN, Python etc.

The concept of a cluster involves taking two or more computers and organizing them to work together to provide higher availability, reliability and scalability than can be obtained by using a single system. When failure occurs in a cluster, resources can be redirected and the workload can be redistributed. The use of MPI libraries have greatly helped in making it easier to utilize the power of clusters as the same implementation that runs on a multicore system can also run on a cluster.

Chapter 4

Message Passing Interface

The generic form of message passing in parallel processing is the Message Passing Interface (MPI), which is used as the medium of communication. Most of the programming languages in parallel programming differ in view of the address space that is available to the programmer, the degree of synchronization imposed on concurrent activities and the multiplicity of programs. A proposed standard Message Passing Interface (MPI) is originally designed for writing applications and libraries for distributed memory environments. In message-passing model, the data is moved from the address space of one process to that of another by means of a cooperative operation such as a send/receive pair. This restriction sharply distinguishes the message-passing model from the shared-memory model, in which processes have access to a common pool of memory and can simply perform ordinary memory operations (load from, store into) on some set of addresses.

4.1 Communication Routines

4.1.1 MPI point-to-point communication routines

MPI has a rich set of point-to-point communication routines include the basic send and receive routines in both blocking and nonblocking forms and in four modes.

- A blocking send blocks until its message buffer can be written with a new message.
- A blocking receive blocks until the received message is in the receive buffer.
- Nonblocking sends and receives differ from blocking sends and receives in that, they return immediately and their completion must be waited for or tested for. It is expected that eventually nonblocking send and receive calls will allow the overlap of communication and computation.

4.1.2 MPI Collective communication routines

Collective communication routines are blocking routines that involve all processes in a communicator. Collective communication includes broadcasts and scatters, reductions and gathers, all-gathers and all-to-alls, scans, and a synchronizing barrier call.

4.1.3 Persistent Communication Requests

Sometimes within an inner loop of a parallel computation, a communication with the same argument list is executed repeatedly. The communication can be optimized by using a persistent communication request, which reduces the overhead for communication between the process and the communication controller. A persistent request can be thought of as a communication port or half-channel.

All MPI communication routines have a data type argument. These may be primitive data types, such as integers or floating-point numbers, or they may be user-defined, derived data types, which are specified in terms of primitive types. Derived data types allow users to specify more general, mixed, and noncontiguous communication buffers, such as array sections and structures that contain combinations of primitive data types

4.2 MPI Communicators, contexts, Groups

A distinguishing feature of the MPI standard is that it includes a mechanism for creating separate worlds of communication, accomplished through communicators, contexts, and groups.

- A communicator specifies a group of processes that will conduct communication operations within a specified context without affecting or being affected by operations occurring in other groups or contexts elsewhere in the program. A communicator also guarantees that, within any group and context, point-to-point and collective communication are isolated from each other.
- A group is an ordered collection of processes. Each process has a rank in the group; the rank runs from 0 to $n-1$. A process can belong to more than one group; its rank in one group has nothing to do with its rank in any other group. A context is the internal mechanism by which a communicator guarantees safe communication space to the group.
- Communicators provide a caching mechanism, which allows an application to attach attributes to communicators. Attributes can be user data or any other kind of information.

4.3 Programming Paradigms

The application users commonly use two types of MPI programming Paradigm: **SPMD** (Single Program Multiple Data) and **MPMD** (Multiple Program Multiple Data). In SPMD model (Single Program Multiple Data), each process runs the same program in which branching statements may be used. The statement executed by various processes may be different in various segments of the program, but one executable (same program) file runs on all processes [16].

In MPMD programming Paradigms, each process may execute different programs, depending on the rank of processes. More than one executable (program) is needed in MPMD model. The application user writes several distinct programs, which may or may not depend on the rank of the processes.

- For execution of an SPMD program, the command format used is:

```
mpirun -n <number of processes> <Executable>
```

- For execution of an MPMD program, the command format used is:

```
mpirun -n <number of processes> -host <Number of Hosts> <Master  
Executable> : -n <number of processes> -host <hosts> <Number of  
Hosts> <Slave Executable>
```

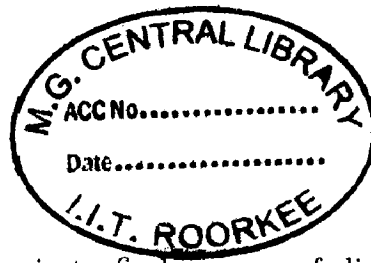
Chapter 5

The Proposed Framework

The filtering algorithm implementation significantly reduces the computation required to solve the problem. In order to further reduce the execution time, we can utilize the parallel architectures available with us for faster execution like Multi-core systems, CELL BE etc.

The main problem with utilizing parallel architectures is to decide the criterion on which the work must be divided among compute nodes that are running in parallel. The filtering algorithm implementation, i.e. our algorithm of choice is basically sequential and does not give any scope for outright parallelization. Hence we have chosen to divide the problem into smaller parts based on dimensions. Traditionally the dimensionality reduction techniques have focused on the discarding some dimensions based on importance or priorities which requires prior knowledge of the nature of the data, which is often not available immediately or is expensive.

Our approach does not cause any information loss as dimensions are not discarded. The pre-processing and post-processing steps add a small amount of overhead to the clustering time itself. However this small overhead helps us in reducing the running time of the actual clustering phase of the algorithm which is the major component of the running time as we will see in the coming sections.



5.1 Pre-processing

The purpose of the preprocessing step is to find groups of dimensions which have similar characteristics and which can therefore yield better quality clusters. The step can also help us extract information about the nature of the data from the knowledge of dimensions that have similar mathematical characteristics. The idea for this approach is derived from the work done by *Mart et. al.* [17].

Finding mathematical information about the nature of the data can also help us in deferring the use of a Subject Matter Expert(SME) for analyzing the data, along with providing the SME more information and detailed meta-data about the data to be analyzed as and when they are involved in the process of information extraction. This can help us in establishing how the different dimensions are related to each other.

Figure 5.1 explains the working of the pre-processing step for 7 dimensions. The input dimensions are passed to the code which analyses the mathematical statistics of the input dimensions. The mathematical statistic retrieved from the data can be channel depth, its minima and maxima, the mean, the standard deviation, the kurtosis and the skewness among others. In this work we have used the kurtosis of the image data for deciding on the grouping criterion. However based on our requirements we can use any of the values or a combination of values retrieved to decide on the grouping criterion without any significant impact on the time taken by the pre-processing step. Based on the similarity metric(Kurtosis in this case), we sort the dimensions and pass it on for creating sub-groups of dimensions. These sub-groups will be used by the clustering phase of the algorithm for parallel clustering.

5.2 Clustering

Based on the information obtained from the pre-processing step the root process runs the unsupervised clustering algorithm in parallel so that we can arrive upon the sub-clusters formed from the sub-groups of dimensions. We must decide upon a value (*DimMin*)for the minimum number of dimensions that must be clustered together. Our choice for the value of *DimMin* was determined by the following factors:

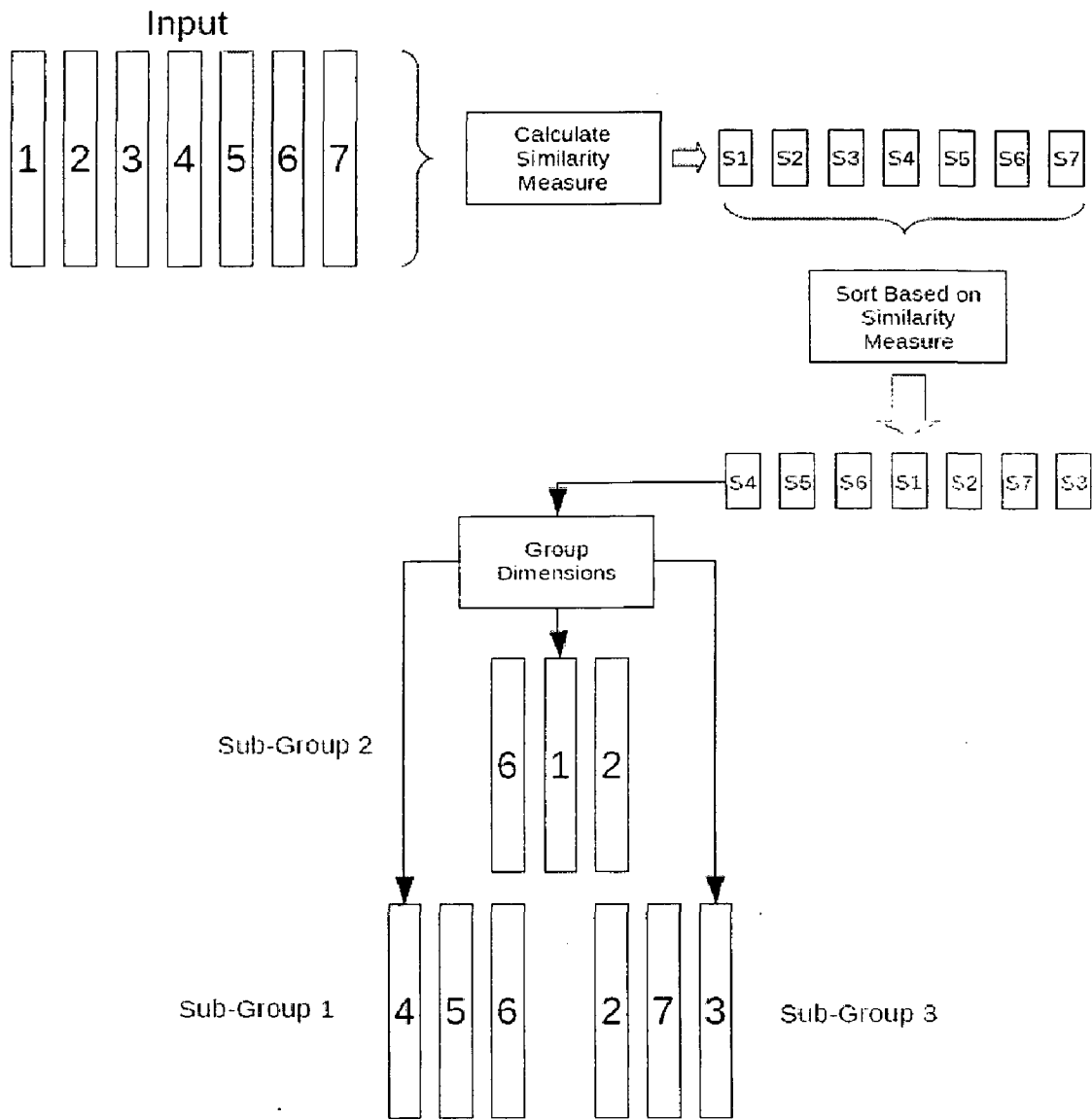


Figure 5.1: The Pre-processing Step

- *Visualisation*: Since we can better visualise data in 3-dimensional space, clustering data in three dimensions has obvious advantages for analysis of data.
- The number of threads that can run in parallel and hence the number of sub-groups of dimensions that need to be created.

In order to merge our clusters during the post-processing stage we need an intersection point between the clusters centers of two sub-groups. Hence for a data containing N dimensions and the minimum number of dimensions in a sub group $DimMin$, the number of sub-groups of dimensions is given by:

$$N_{sub} = \lfloor N / (DimMin - 1) \rfloor \quad (5.1)$$

The Figure 5.2 shows the flow of data through the proposed framework.

The MPI APIs used for communication among the threads of execution during the clustering phase are [18]:

1. `int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)`

where,

- buf* Starting address of send buffer (choice).
- count* Number of elements to send (nonnegative integer).
- datatype* Datatype of each send buffer element (handle).
- dest* Rank of destination (integer).
- tag* Message tag (integer).
- comm* Communicator (handle).

2. `int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)`

where,

- count* Maximum number of elements to receive (integer).
- datatype* Datatype of each receive buffer entry (handle).
- source* Rank of source (integer).
- tag* Message tag (integer).
- comm* Communicator (handle).

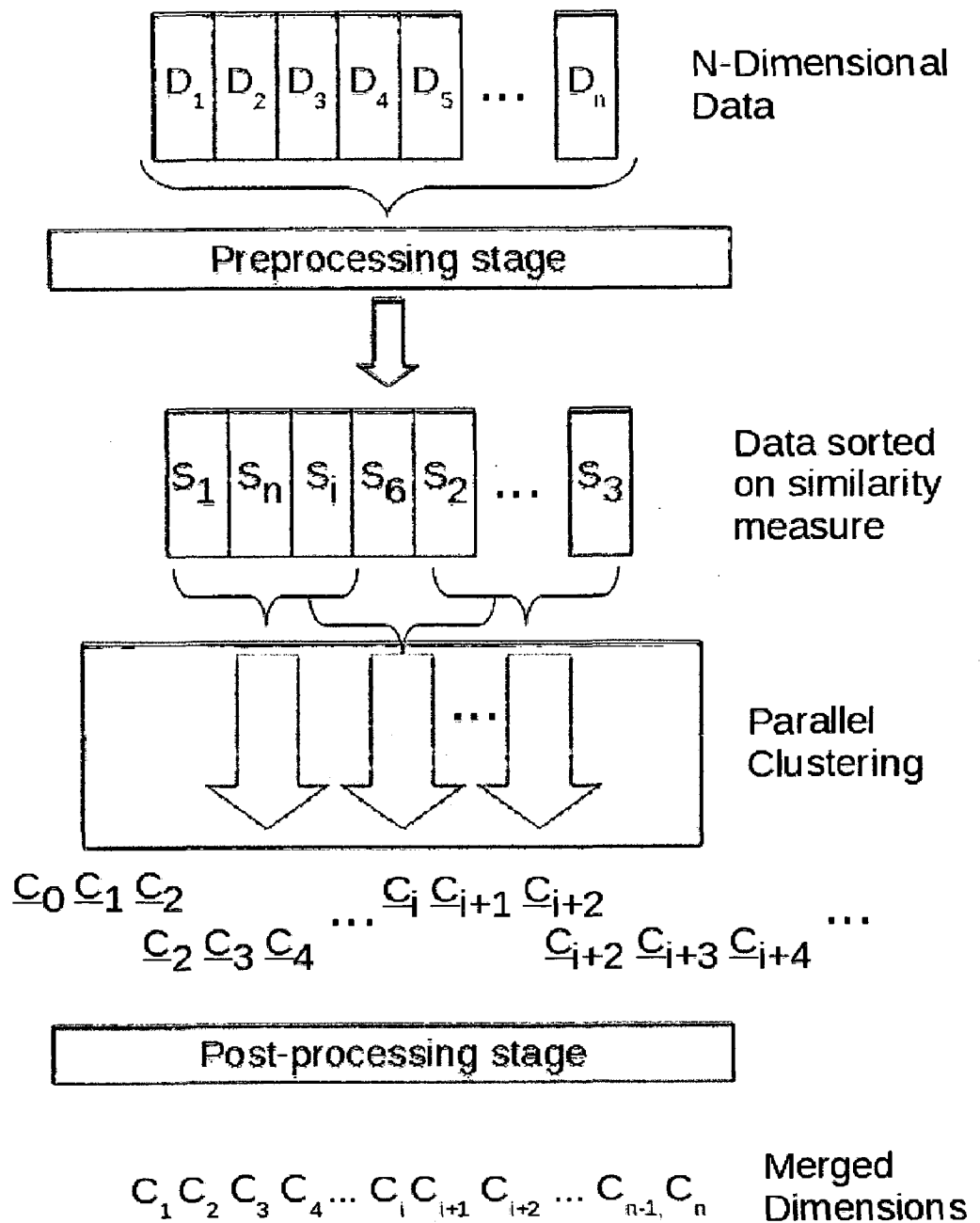


Figure 5.2: Flow of Data through the proposed framework

buf Initial address of receive buffer (choice).

status Status object (status).

3.

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
--

where,

buffer Starting address of buffer (choice).

count Number of entries in buffer (integer).

datatype Data type of buffer (handle).

root Rank of broadcast root (integer).

comm Communicator (handle).

5.3 Post-processing

The post-processing step takes the clusters created by the sub-group of dimensions and merges them based on the overlapping dimensions, with the error threshold for merging two clusters being the minimum of the Average of Squared distances of the two sub-clusters formed from the sub-groups of dimensions, being merged.

The figure 5.3 provides a schematic explanation of how the post-processing steps operate on the output of the parallel clustering step.

The new cluster centers are created based on the property that for an ideal cluster if there is a cluster center in a Data Set **D**, having **n** Dimensions, given by

$$C_0, C_1, \dots, C_i, \dots, C_n. \quad (5.2)$$

When the dimensions are divided into groups (0,1,2... i,i+1) and (i+1,i+2,i+3... n), then in case of a ideal cluster, the centers of the clusters formed in the different groups would be

$$C_0, C_1, \dots, C_{i+1} \text{ and } C_{i+1}, C_{i+2}, \dots, C_n. \quad (5.3)$$

In order to account for the fact that the real time data is noisy and the clusters are not

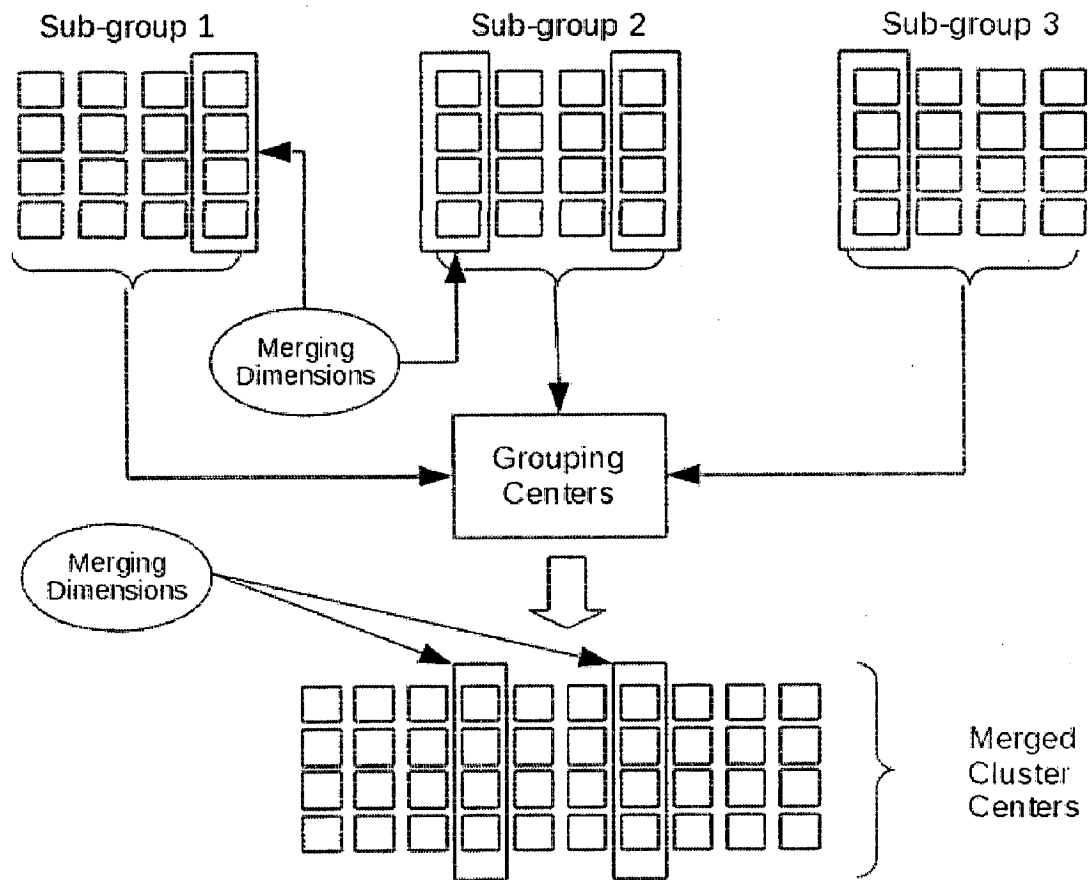


Figure 5.3: The Post-processing Step

ideal, we introduce a factor into the merging process which is very unimaginatively called the Ideal Coefficient(I_c). When $I_c = 0$, clusters centers are merged only if their centers match exactly in both the sub groups.

For cases where the value of I_c is non zero, the cluster centers of two sub-groups are merged only if they fall within the squared distance of $I_c \times \min(\text{the two cluster radii})$.

Mathematically, For two consecutive sub-groups G_1 having cluster centers C_0, C_1 and C_3 and sub-group G_2 having cluster centers C_3, C_4 and C_5 , for the sake of differentiation we denote the value of the third dimension of the group G_1 with C_{31} and first dimension of G_2 with C_{32} . The Average of Squared Distance of the cluster G_1 and G_2 is denoted by Avg_1^2 and Avg_2^2 .

Then we combine the dimensions of G_1 and G_2 only if

$$(C_{31} - C_{32})^2 \leq \min(Avg_1^2, Avg_2^2) \quad (5.4)$$

Now in order to introduce the Ideal Coefficient(I_c) mentioned earlier into equation 5.4. For the ideal scenario the values C_{31} and C_{32} are same, and

$$(C_{31} - C_{32})^2 = 0 \quad (5.5)$$

Thus the value of I_c is *zero*.

When the value of I_c is non-zero, we have

$$(C_{31} - C_{32})^2 \leq I_c \times \min(Avg_1^2, Avg_2^2) \quad (5.6)$$

5.4 The Algorithm

An overview of the modified algorithm for parallel clustering of data is as given below.

Let $S = x_1, \dots, x_n$ denote the set of points to be clustered. And, let $\|x\|$ denote the Euclidean length of the vector x .

- 1: For each dimension in input D_1, D_2, \dots, D_n calculate the similarity metric S_1, S_2, \dots, S_n

- 2: Sort the dimensions D_i based on similarity measure S_i and divide into N_{sub} groups based on the value of $DimMin$ with the adjacent sub-groups having an overlapping dimension,

$$N_{sub} = \lfloor N / (DimMin - 1) \rfloor \quad (5.7)$$

- 3: Run steps 4 to 14 for all N_{sub} sub-groups of dimensions in parallel and then go to Step 15.
- 4: Letting $k = k_{init}$, randomly sample k cluster initial centers $Z = z_1, z_2, \dots, z_k$ from S .
- 5: Assign each point to its closest cluster center. For $1 \leq i \leq k$, let $S_i \subseteq S$ be the subset of points that are closer to z_i than to any other cluster center of Z . That is, for any $x \in S$,

$$x \in S_j \text{ if } \|x - z_j\| < \|x - z_i\|, \quad \forall i \neq j. \quad (5.8)$$

(Ties for the closest center are broken arbitrarily.) Let n_j denote the number of points of S_j .

- 6: Remove cluster centers with fewer than n_{min} points. (The associated points of S are not deleted, but are ignored for the remainder of the iteration.) Adjust the value of k and relabel the remaining clusters S_1, \dots, S_k accordingly.
- 7: Move each cluster center to the centroid of the associated set of points. That is,

$$z_j \leftarrow \frac{1}{n_j} \sum_{x \in S_j} x; \quad \text{for } 1 \leq j \leq k. \quad (5.9)$$

If any clusters were deleted in Step 6, then the algorithm goes back to Step 5.

- 8: Let Δ_j be the average distance of points of S_j to the associated cluster center z_j , and let Δ be the overall average of these distances.

$$\Delta_j \leftarrow \frac{1}{n_j} \sum_{x \in S_j} \|x - z_j\|, \text{ for } 1 \leq j \leq k. \quad \Delta \leftarrow \frac{1}{n} \sum_{j=1}^k n_j \Delta_j \quad (5.10)$$

Let $v_{j,max}$ denote the largest coordinate of v_j .

- 9: If this is the last iteration, then set $L_{min} = 0$ and go to Step 12. Also, if $2k > k_{init}$ and it is either an even numbered iteration or $k \geq 2k_{init}$, then go to Step 12.

- 10: For each cluster S_j , compute a vector $\mathbf{v}_j = (v_1; \dots, v_d)$ whose i th coordinate is the standard deviation of the i th coordinates of the vectors directed from z_j to every point of S_j . That is,

$$v_{ji} \leftarrow \left(\frac{1}{n_j} \sum_{X \in S_j} (x_i - z_{ji})^2 \right)^{1/2} \quad \text{for } 1 \leq j \leq k \text{ and } 1 \leq i \leq d. \quad (5.11)$$

- 11: For each cluster S_j , if $v_{j,max} > \Delta_{max}$ and either

$$((\Delta_j > \Delta) \text{ and } (n_j > 2(n_{min} + 1))) \text{ or } k \leq \frac{k_{init}}{2}, \quad (5.12)$$

then increment k and split S_j into two clusters by replacing its center with two cluster centers centered around z_j and separated by an amount and direction that depends on $v_{j,max}$. If any clusters are split in this step, then go to Step 5.

- 12: Compute the pairwise intercluster distances between all distinct pairs of cluster centers

$$d_{ij} \leftarrow \|z_i - z_j\|, \quad \text{for } 1 \leq i < j \leq k. \quad (5.13)$$

- 13: Sort the intercluster distances of Step 12 in increasing order, and select a subset of at most P_{max} of the closest such pairs of clusters, such that each pair has an intercluster distance of at most L_{min} . For each such pair (i, j) , if neither S_i nor S_j has been involved in a merger in this iteration, replace the two clusters S_i and S_j with a merged cluster $S_i \cup S_j$, whose associated cluster center is their weighted average

$$z_{ij} \leftarrow \frac{1}{n_i + n_j} (n_i z_i + n_j z_j). \quad (5.14)$$

Relabel the remaining clusters and decrease k accordingly.

- 14: If the number of iterations is less than I_{max} , then go to Step 5.

- 15: For $i = 1$ to $N_{sub} - 1$ repeat steps 16 to 17.

- 16: Compare the last dimension of sub-group i with first dimension of sub-group $i+1$. Merge the two groups of centers if they follow the constraint

$$(C_{31} - C_{32})^2 \leq I_c \times \min(Avg_1^2, Avg_2^2) \quad (5.15)$$

- 17: If $i = N_{sub} - 1$ then insert the merged centers into the list of Merged Cluster centers.

Chapter 6

Experimental Results

The image data we ran the tests on were 1123×1080 Landsat images of Path 146 Row 40 ($n = 1212840$). The experiments involved all 7 bands of data, running for 20 iterations and with $StdDev = Lump = 10$ and $SampRm = 100$. The results are presented in Table 6.1. The parallel version of the code using MPI was faster by a about two times for an execution with $NumClus=100$. As we can see from the results as shown in Table 6.1 and Figure 6.1, the parallel algorithm really shines when the amount of computation needed is significant. We expect even better results for problems of higher dimensions and larger number of clusters.

6.1 Pre-Processing step

For separating the dimensions into subgroups, the similarity measure that was used is kurtosis of the image data. The Kurtosis of an Image is defined as the measure of whether the data is peaked or flat relative to a normal distribution. After finding the kurtosis the dimensions are sorted based on the values obtained so that the dimensions with similar values lie together. Since kurtosis gives us an estimate of how the data varies, which is similar to what clustering tries to achieve for multiple dimensions, it can help in determining the characteristics of the data to be clustered in this case.

For finding the kurtosis of the image the APIs provided by the ImageMagick library were used. The MagickWand API is the recommended interface between the C

Table 6.1: Results of Landsat data test

NumClus	StdDev	Serial code (in secs)	Parallel Code (in secs)	Speedup
25	10	38.26	29.31	1.31
50	10	43.36	30.57	1.42
100	10	49.89	32.14	1.55
200	10	57.81	32.21	1.79
400	10	67.07	32.88	2.04

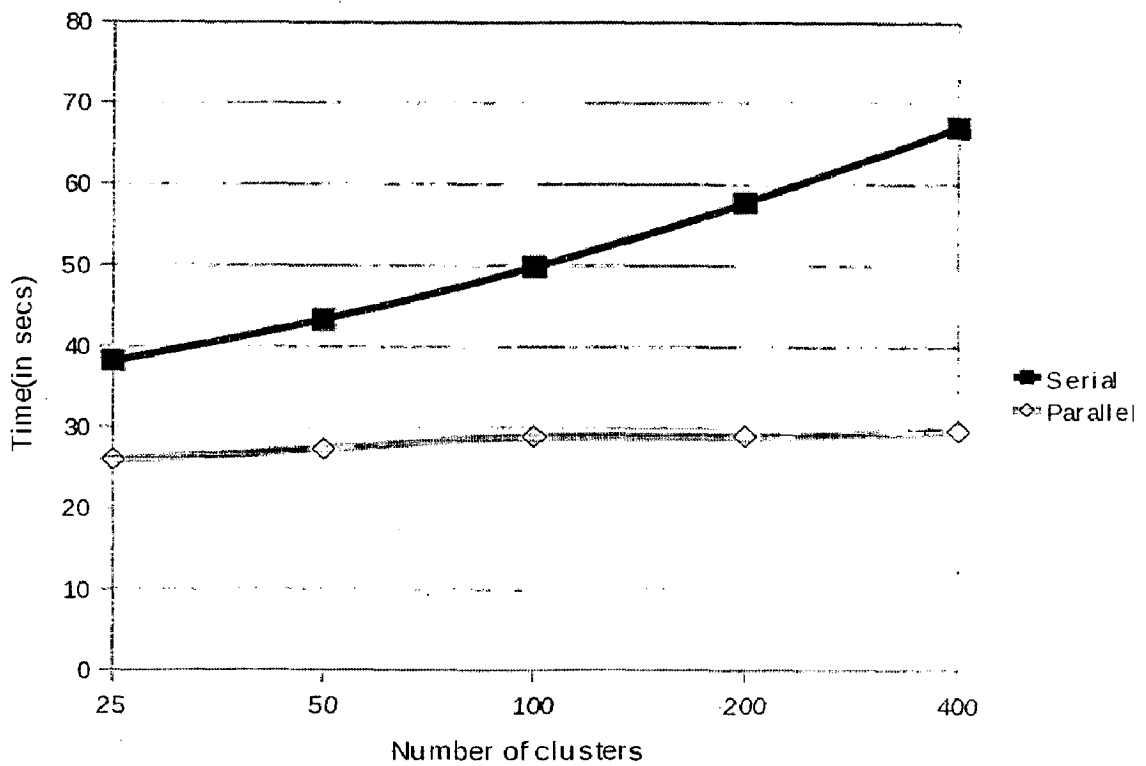


Figure 6.1: Comparison of Serial and parallel execution time

programming language and the ImageMagick image processing libraries. The function used for finding the image statistics is `MagickGetImageChannelStatistics()`, The function returns statistics for each channel in the image. The statistics include the channel depth, its minima and maxima, the mean, the standard deviation, the kurtosis and the skewness.

The format of the `MagickGetImageChannelStatistics` method is:

```
ChannelStatistics *MagickGetImageChannelStatistics(MagickWand *wand)
```

where,

wand: is the Magic Wand Object created on passing an Image file to it

The pre-processing step gives us the values of the kurtosis among others for each band of LANDSAT image data. The values are shown in the table 6.2

Table 6.2: Image Statistics of ISODATA bands

Band	Mean	Maxima	Std.Dev	Kurtosis	Skewness
1	13722.447870	65535.000000	8603.656996	-0.842767	-0.807476
2	6282.232393	51143.000000	4026.154022	-0.559548	-0.647161
3	6227.028588	63736.000000	4254.461937	0.089190	-0.213133
4	14034.895148	58339.000000	8829.011116	-1.072793	-0.841465
5	13209.891964	65535.000000	8762.995117	-1.027934	-0.517764
6	27627.177277	51914.000000	16835.649467	-0.938688	-1.011677
7	5537.834025	42662.000000	4190.343359	-0.232981	0.241447

We sort the bands based on the kurtosis (Table 6.3), its important to note that we can use any of the mathematical measures obtained from the analysis of the image data or a combination of them, without any significant variation in the amount of time taken by the pre-processing stage.

Table 6.3: Bands Sorted by Kurtosis of Image data

Band	Mean	Maxima	Std.Dev	Kurtosis	Skewness
4	14034.895148	58339.000000	8829.011116	-1.072793	-0.841465
5	13209.891964	65535.000000	8762.995117	-1.027934	-0.517764
6	27627.177277	51914.000000	16835.649467	-0.938688	-1.011677
1	13722.447870	65535.000000	8603.656996	-0.842767	-0.807476
2	6282.232393	51143.000000	4026.154022	-0.559548	-0.647161
7	5537.834025	42662.000000	4190.343359	-0.232981	0.241447
3	6227.028588	63736.000000	4254.461937	0.089190	-0.213133

6.2 Parallel Clustering

In order to run parallel clustering of the subgroups, we modified the implementation of clustering algorithm to include the parameter *DimMin*. Message Passing Interface(MPI) libraries were used to create parallel threads of execution.

The root process first reads the Directive file which contains parameters for running the filtering algorithm and broadcasts it using the *MPI_Bcast*. After obtaining the sorted dimensions from the pre-processing step the root process reads the dimension data into memory and transmits the data to slave processes based on the value of the *DimMin* using *MPI_Send*, the slave processes receive the data using *MPI_Recv*. In our experiments we have kept the value of *DimMin* as 3. Therefore as per equation 5.1 on page 22, for 7 dimensions we form 3 sub-groups because of the overlapping dimensions that are required for post-processing of sub-clusters. While creating subgroups of clusters if there are not enough dimensions left to form a different sub-group of dimensions, then we add the remaining dimensions to the last sub-group created. The final value of the number of dimensions being processed by each thread of execution is passed to the corresponding thread from the root thread.

The time taken for parallel clustering is given in Table 6.4. We can see from table 6.4 and table 6.1, that the rate of increase of computation time with the increase in

Table 6.4: Parallel clustering execution times for different number of clusters

Number of Clusters(NumClus)	Execution Time (in sccs)
25	26.093
50	27.351
100	28.922
200	28.99
400	29.66

computation required for solving the problem does not increase super linearly as in the case of the serial implementation of the algorithm.

6.3 Post-processing step

After we have obtained the centers for the sub-clusters through the parallel clustering step. The post-processing step merges the cluster centers for the different clusters to obtain the final cluster centers. The amount of time taken by the pre-processing and post-processing steps together adds very little overhead to the actual clustering process. The running times of the pre-processing and the post-processing steps together did not exceed more 1 percent of the time taken for clustering.

The number of sub-clusters from the three subgroups formed from the dimensions (4,5,6), (6,1,2), and (2,7,3) were $N_{456}=110$, $N_{612}=80$ and $N_{456}=78$ respectively. On the merging the sub-clusters we obtained 126 clusters with a value for Coefficient of Imperfection(I_C) as 1 .

Chapter 7

Conclusion

This work has proposed a framework through which we can strive for faster implementations of clustering algorithms. The proposed framework adds a pre-processing and a post-processing step so that the computation can be spread across multiple threads of execution. The feasibility of the approach has been demonstrated by modifying the implementation of clustering of LANDSAT image data using the filtering algorithm. The results were encouraging and it was possible to reduce the execution time of the algorithm to *half* its original execution time. An encouraging fact observed during the course of experimentation was that the rate of increase of the execution time is not as steep as in the original implementation, i.e. the running time of the algorithm does not increase non-linearly as in the original implementation using filtering algorithm. Hence even higher speedups are expected for clustering of larger data sets.

The work has introduced a new parameter in the post-processing stage called the Ideal Coefficient(I_c) through theoretical analysis of the clustering algorithms. One possible direction of future research can be to take the information about the data available in the pre-processing step and using it to determine the value of I_c .

Bibliography

- [1] John A. Richards and Xiuping Jia. *Remote Sensing Digital Image Analysis: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [2] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamlesh Munagala, and Vinayaka Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004.
- [3] Imola Fodor. A survey of dimension reduction techniques. Technical report, 2002.
- [4] NASA. Multispectral bands. http://rst.gsfc.nasa.gov/Sect3/Sect3_1.html, 2007. [Online; accessed March 13-14, 2010].
- [5] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.
- [6] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 10–18, New York, NY, USA, 2002. ACM.
- [7] Nargess Memarsadeghi, David M. Mount, Nathan S. Netanyahu, and Jacqueline Le Moigne. A fast implementation of the isodata clustering algorithm. *Int. J. Comput. Geometry Appl.*, 17(1):71–103, 2007.
- [8] Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, 1977.

- [9] Edward Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–780, 1965.
- [10] Léon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *NIPS*, pages 585–592, 1994.
- [11] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.
- [12] Olvi L. Mangasarian. Mathematical programming in data mining. *Data Min. Knowl. Discov.*, 1(2):183–201, 1997.
- [13] Greg R. Andrews. *Foundations of Parallel and Distributed Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
- [14] Samuel Williams, Leonid Oliker, Richard Vuduc, John Shalf, Katherine Yelick, and James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM.
- [15] Chiu-Pi Shih Tian Tian. Software techniques for shared-cache multi-core systems. <http://software.intel.com/en-us/articles/software-techniques-for-shared-cache-multi-core-systems>, 2009. [Online; accessed March 10-11, 2010].
- [16] Michelle Mills Strout, Barbara Kreaseck, and Paul D. Hovland. Data-flow analysis for mpi programs. In *ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing*, pages 175–184, Washington, DC, USA, 2006. IEEE Computer Society.
- [17] José Martínez Sotoca and Filiberto Pla. Supervised feature selection by clustering using conditional mutual information-based distances. *Pattern Recognition*, 43(6):2068–2081, Feb 2008.
- [18] The Open MPI Project. Multispectral bands. <http://www.open-mpi.org/doc/v1.4/>, 2010. [Online; accessed February 11-12, 2010].

List of Publications

- Anant Bhushan, Dr. Kuldip Singh and Dr. Ankush Mittal, "Framework for faster implementation of unsupervised clustering algorithms," *In Proceedings of International Conference on Advances in Communication, Network, and Computing*, Calicut, India, 04-05 Oct 2010 [Accepted for publication]