Genomic Data Analysis Using Grid-Based Computing

A Thesis
Presented for the
Master of Science
Degree

The University of Tennessee, Knoxville

Bhanu Prasad Rekapalli
December 2003

Dedicated to my father, mother, sister, and friends, and to the world, which gave me the opportunity to prove myself, and all who guided, motivated and supported me throughout my educational career.

Acknowledgements

I would like to sincerely acknowledge my thesis advisor Dr. Gregory Peterson for his kind and helpful supervision throughout the course of my work. I would like to thank my thesis committee members, Dr. Mohammed Ferdjallah, and Dr. Michael Langston for reviewing and directing my thesis.

I sincerely thank Webservices, UT for supporting me during my work, as a graduate assistant at Webservices this idea of thesis was generated without which the thesis would not have been existed. I am indebted to Dr. Colton Smith, John Rose, and Robert Hillhouse of Webservices for support and encouragement provided by them for this thesis. I would like to thank Dr.Gavin Sherlock of Stanford university for guiding me to pick computation intensive parts of SMD.

I am grateful to all my friends who helped in preparation of this thesis. Special thanks to my best friend Vamsi Nellore for giving moral and great support in hard times. I thank my friends Mardhav Wala for proof reading my thesis, and Venki, Rohan and my roommate Krishna.

I would like to thank University of Tennessee for giving me the opportunity for the M.S. program. Its my pleasure to thank my graduate and undergraduate professors who laid a good foundation for my thesis work.

Last but most important, I would like to thank whole hearted, my parents Lakshmi Sulochana Rekapalli and Subba Rao Rekapalli and my sister Dr. Deepthi Rekapalli who supported, loved, helped and cared for me throughout my life. I am dedicating this thesis to them.

Abstract

Microarray experiments generate a plethora of genomic data; therefore we need techniques and architectures to analyze this data more quickly. This thesis presents a solution for reducing the computation time of a highly computationally intensive data analysis part of a genomic application. The application used is the Stanford Microarray Database (SMD). SMD's implementation, working, and analysis features are described. The reasons for choosing the computationally intensive problems of the SMD, and the background importance of these problems are presented. This thesis presents an effective parallel solution to the computational problem, including the difficulties faced with the parallelization of the problem and the results achieved. Finally, future research directions for achieving even greater speedups are presented.

Table of Contents

List of Tables

List of Figures

Chapter 1

Overview

1.1 Introduction

Bioinformatics is an emerging field in biotechnology with the increasing demand and importance. Scientists and researchers predict a beginning of the “Biotechnological era” during the coming years. Bioinformatics is a developing branch in biology which is highly interdisciplinary, utilizing the concepts and techniques from informatics, statistics, mathematics, chemistry, biochemistry, physics, and linguistics[16]. To achieve a highly complex and coordinated task of creating useful microorganisms through computer aided design, a mixture of technologies must be used[17]. To refine enzymes and to analyze kinetic parameters in vitro, enzyme engineering is used. Metabolic engineering is used to analyze flux rates in vivo. Analytical chemistry is used to determine and analyze the quantity of metabolites efficiently. Genetics engineering is used to select the genes and for modifying metabolic pathways. Simulation science is used to efficiently and accurately simulate a large number of reactions. Knowledge engineering is used to construct, edit and maintain large metabolic knowledge bases. Mathematics is used to estimate and tune unknown parameters. Thus proving the highly interdisciplinary aspect of bioinformatics.

Many complex designs and tools are developed using the multi-disciplinary aspect of Bioinformatics. This arises the question, what is Bioinformatics?

1.1.1 Definitions

On July 17, 2000 the definition committee of National Institutes of Health (NIH) Biomedical Information Science and Technology Initiative Consortium (BISTIC) adopted the following definitions and distinctions[18][25].

> Bioinformatics: Research, Development, or application of computational tools and approaches for expanding the use of biological, medical, behavioral or health data, including those to acquire, store, organize, archive, analyze or visualize such data.

> Computational Biology: The development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, behavioral and social systems.

1.1.2 History

In recent times, the biological information being generated is very immense. Many projects like genome sequencing, macromolecules structures, and functional genomic experiments, led to the rise of a new field called biocomputing. Biocomputing or bioinformatics became popular because of these projects. Basically biocomputing encompass bioinformatics and computational biology. But the concept of computing in biology is old and dates back as far as 1924, when A.J. Lotka, in *"Elements of Physical Biology"* established biological laws solely from data analysis by induction [18]. In this paper he said: "It remains to enumerate the methods by which Physical Biology may be expected to develop. For the gathering of data two methods are available: observation in natural conditions, and observations under experimental (laboratory) conditions. For the

elaboration of data, the establishment of regularities (laws), there is available in this field, as every where in science, the method of induction, aided, if need be, by statistical technique."

The chronology and essence of early events that led to the development of these fields dates back, covering the period from the 1869, discovery of DNA by Miescher, to 1980-1981 and the beginning of massive sequencing[19]. Many researchers had successes in computational molecular biology in early times, like Linus Pauling, who advanced the theory of molecular evolution and Margaret Dayhoff, credited with authoring the Atlas of Protein Sequences in 1960's. This field of computing in biology took a new turn to evolve into its present-day state, when the Needleman-Wunsch algorithm for the global alignment of two biological sequences was outlined in the early 1970s. In 1977 when the first DNA genome sequenced, the interaction between biologist and computer scientist increased [20]. Programs were developed not only for simple algorithms to translate DNA sequences into protein sequences [26], but also programs were developed to detect more complicated patterns such as restriction enzyme recognition sites [27][20]. Also the early artificial intelligence approach in the field of restriction enzyme mapping [28] was successful, which revolutionized the entry of computer science community into biological society [20]. The Basic Local Alignment Search Tool (BLAST), since its introduction in 1990 revolutionized the practice of biology in virtually every lab, and the original paper of BLAST became the most cited paper.

1.1.3 Importance

The various computing techniques have evolved with the development of High Performance Computing (HPC) and the availability of vast amounts of biological and medical experimental data. There is a natural relationship between computer science and biology[16]. According to Moore's Law the processing power is doubling every 18 months, at the same time the phenomenal rate of biological data that is generated by various applications provides challenges in data storing, analyzing and accessing data. Since most of the data needed mathematical or statistical methods for processing, thus increasing the need of computation.

One of the most important part of the computing is analysis of the data generated from experiments, which is used in most of the fields. Analysis in bioinformatics focuses on genome sequencing, macromolecular structures and functional genomic experiments. Algorithmic development is an important part of bioinformatics, and techniques and algorithms are specifically developed for the analysis of biological data [16].

The GenBank is doubling every 16 months [20], and containing 2 Billion bases of DNA sequence in 1999 [29], according to this statement there should be more than ten billion of base sequences. Thus modern biology needs computational tools to store, display and analyze these sequences. The best example is human genome project which has 3 billion base pairs. This shows the great impact of bioinformatics on biological research. Hence biocomputing with help of computer may answer many biological questions, which traditional approaches could not tackle. Thus the introduction of computers in biology led to the founding of dedicated bioinformatics research groups.

1.2 Objective

An explosion of biological data, like in human genome projects, is creating a serious bottleneck in runtime of analysis, data searching and storing. This results in an immense necessity for Parallel Computing (PC) which is a dominant technique in achieving high performance.

The complexity and size of the various simulation and modelling problems, computation and manipulation problems and analysis problems is so large that it requires high processing speed and more memory, which makes parallel computing an essential and important technique in solving these humongous problems. This thesis deals with one of the biological application called Stanford Microarray Database (SMD) which stores and analyzes the gene data from the microarray experiments. This thesis focuses on the analysis part of SMD which uses K-Nearest Neighbor (Knn) impute algorithm for estimating the missing values in the experiment set and also with Hierarchical Clustering (HC) algorithm which is used to clustering genes of the experiments to specific groups based on few parameters. With this background in mind, let us define parallel computing.

Parallel computing is defined as dividing a big problem or a task into smaller tasks and distributing these smaller tasks to multiple processors and coordinating work and communication between these processors. The advantage of parallel computing is, the cost performance, which is scalable and affordable and has high reliability and fault tolerance.

There are two types of parallelism one is data parallelism and the other is functional parallelism. In data parallelism a sequential program is run on multiple machines, on different sets of data concurrently. In functional parallelism, an application or program is divided into a set of functionally independent components or tasks, which can be executed in parallel[21].

In this thesis, the Knn-impute program is data parallelized and hierarchical clustering program is partly data parallelized and partly functional parallelized. Parallel Virtual Machine (PVM) is used to calculate the bandwidth and latency of the cluster, to evaluate the performance of communication. But mostly Message Passing Interface (MPI) is used for parallelization of main programs.

The parallelization used in this thesis uses the “master-slave” concept. Master which is one of the process or a processor does most of the load balancing and coordinating work. This means the master process divides the tasks and sends them to the slave processes and the data sets on which these tasks should be performed. The slave processes performs the computation and sends back the results to the master. Then the master collects the results and generates the final output. This thesis primarily focuses on serial load balancing.

1.3 Scope of Thesis

Chapter 1 introduces the reader to the concepts, importance and the history of Bioinformatics. Chapter 2 discusses about the background research done in this field and the related work that is done on the algorithms used. Chapter 3 deals with the actual

problem and description of the SMD applications, end-to-end procedure and individual blocks of the problem which needs computation. This chapter also discusses the serial implementation and computation times of the problem, which picks up the blocks which need high performance computing. Chapter 4 describes the solution to the problem, which uses parallel computing approach to decrease the computation time of the application. Chapter 5 shows the results and graphs and the speed ups achieved. Chapter 6 is the conclusion and future work to enhance the applications further using reconfigurable computing approach.

Chapter 2

Background and Related Work

The data generated by many applications is increasing tremendously, with advancing technology. Thus data mining and analysis is becoming a important need for researchers in various fields, to solve various problems. For example, the NASA earth observatory system satellites send terabytes of data every day to be analyzed by the systems on the ground. Another example is the medical applications where a large number of radiological images are generated by hospitals. This thesis addresses processing images generated from microarray experiments to analyze genes.

To quickly analyze such humongous data sets, parallel algorithms and efficient system architectures are required. This leads to the development of high performance computing systems. For time-critical applications there is a need of efficient high performance computing architectures, where the decisions can be taken on the fly. Automatic target recognition in defense jets is a good example of a time-critical application.

Genomic data is growing explosively[1]. This data is used to diagnose diseases, or invent new drugs, and manage diseases. Analyzing this data is of great biological and medical importance. Since the time to analyze this data can be less in some cases, for example a brian tumor is detected in a patient then the images generated by the scanners should be downloaded and analyzed quickly to save the patient, thus this area provides ample opportunities for parallel processing.

The genomic application discussed in this thesis deals with microarray data.Microarray experiments are performed to examine gene expression on a genomic scale. This modern technology allows researchers to follow the expression of an organism's entire complement of genes simultaneously in a single simple experiment[2]. The Stanford Microarray Database (SMD) [22] is a web-based application that stores and analyzes microarray data. SMD has the capability to analyze the data with the help of the clustering algorithms like Hierarchical clustering, Self-Organizing Maps, and K-Means algorithms. Clustered data is then fed to the graphical viewing software, which shows the results. The problem is the number of experiments in the database are enormous, SMD at Stanford contains around 35,000 experiments. Hence it can take a huge amount of time to cluster genes according to their expression profiles. For instance, depending on how large the gene set and the number of experiments, clustering take anywhere from hours to days. Hence parallelizing the cluster program has the potential of saving enormous amounts of time, allowing researchers to efficiently mine their data.

A single DNA-microarray experiment may involve tens of microarrays; each containing thousands of spots, for which dozens of measurements are recorded resulting in millions of pieces of information. Currently, The University of Tennessee is experimenting with microarray chips, which have tens of thousands of spots. To analyze the data, the clustering program of SMD is used to cluster data. This program was written in C by SMD group of Genetics Department, Stanford University. Since the number of spots on the microarray chips is increasing and the number of experiments conducted are rising, computation times are taking hours and sometimes even days. This led to the idea of parallelizing the cluster program, for achieving the faster calculations.

Another potential problem is estimating the missing values in the experiment sets. This is done using the KNN-Impute algorithm[5], which is another time consuming part of SMD. This thesis discusses the non-parallel version of the codes. This thesis also discusses, how the data is calculated, and analyzed and how this can be parallelized and the different levels of difficulties faced to parallelize these programs.

Apart from the above application K nearest neighbors approach is used in many areas and this method is efficient in predicting the close to correct data. At the University of Minnesota researchers are developing satellite-based approaches to estimate and map forest cover types, volume and basal area on a pixel-by-pixel basis[7]. The forest services are conducting nation wide forest inventory for decades. Yet these field plot based inventories have not been able to produce precious county and more local estimation and useful operational maps. The satellite-based forest classification is not matching with ground based surveys. The K-nearest neighbors approach, offers a means of applying satellite and GIS data so as to impute forest cover type, timber volume, and many other data from field plots surrounding large or small areas on the basis of the spectral characterization of neighboring pixels. Applications to updating land use maps, crop estimates, and other resource information needs are also being investigated with KNN approach[7].

To predict the time that will be needed to traverse a certain stretch of freeway when departure is at a certain time in the future is a challenging problem. The prediction is done on the basis of the current traffic situation in combination with historical data[8]. The California Department of Transportation (Caltrans), has a Performance Measurement

system (PeMS) obtains loop detector data on flow and occupancy, aggregated over 30 seconds of intervals. The amount of data collected is around 2 giga bytes per day. KNN approach is one of the prediction method used for this.

Knn-impute approach is used in many data mining applications[9] like fault detection in industrial processes or manufactured objects or fraud detection in banking and other commercial operations, many medical and biological applications, parameter prediction in some nueral network applications, where humongous amount of data is stored in the databases all around the world. To speed up the processing of these applications parallel processing and specific architectures are required.

The hierarchical clustering is picked for the thesis because of its simplicity and efficiency and many researchers are extensively doing research to parallelize the clustering algorithms. Suppose N patterns each with M features, the sequential hierarchical clustering algorithm can be computed in $O(N^2M+N^3)$ time in a straight forward manner[10]. Li and Fang [11] proposed an O(N Log N) time parallel algorithm on the MN node hypercube and $O(n \log^2 n)$ on n-node butterfly. Li also proposed an $O(N^2)$ time parallel algorithm on the SIMD shuffle-exchange networks using N processors[12]. Tsai and Horng proposed an $O(\log^2 N)$ time parallel algorithm on the processor array with a reconfigurable bus system (PARBS) using N^3 processors[14]. Wu, Horng and Tsai also proposed a O(log N) time parallel hierarchical clustering algorithm with N^3 processors[10] on PARBS. Olson proposed many implementations of hierarchical clustering algorithms.

In his implementation he achieved O(N) time on a N-node CRCW PRAM and O(n log n) time on (n/log n) node butterfly.

All these parallel clustering algorithms have a drawback[15], they need a large number of processors about the size of the data set to achieve a reasonable performance. For instance, a data set of millions of points needs a vast number of processors which is not possible. Chapter 4 describes how the Knn-impute and hierarchical clustering programs are parallelized using few number of processors.

Chapter 3

Problem Approach

This chapter discusses about the serial implementation of the SMD application for analyzing the data. The problem is described, which shows the current implementation, the CPU utilization and some factors which led to the development of this research. This implementation is described from the point of data retrieval from the database to the point of displaying the analyzed data by the graphical user interface. The problem is tackled first block wise, and than end to end that is the entire application for analysis, based on different file sizes.

3.1 Current SMD Implementation

The SMD application is currently running on the genome server deployed in Stokeley Management Centre, UT. The database is on cluster of machines SAN1 with T3 disk array. The database uses Oracle right now, and the implementation is shown in the figure 3.1. Currently, the University of Tennessee Microarray Database (UTMD) consists of many research groups using SMD to house and analyze experiments. The UTMD currently houses around 50 experiments of 4 different organisms containing tens of thousands of genes on each chip (experiment). Since the microarray is the growing technology, the number of experiments and the genes in each experiment increase day by day. Stanford University right now houses around 35,000 experiments of around 125 different organisms and is growing rapidly every year. This same trend will be followed by UT in coming years.

.

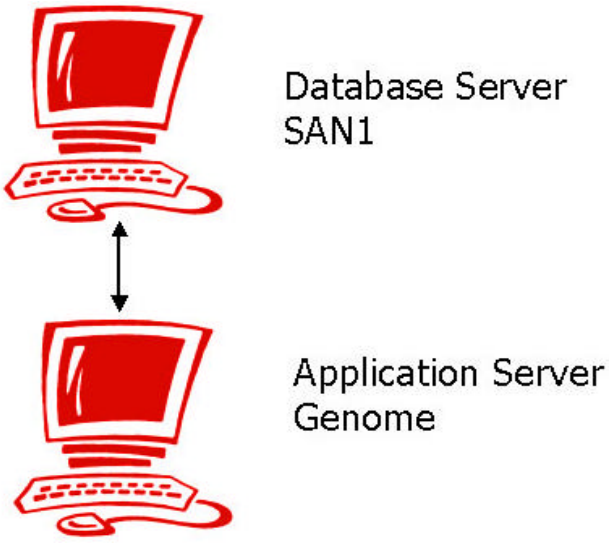

Figure 3.1: Current Implementation of SMD application at UT

The table 3.1 shows that the Knn-Impute and the clustering functions are CPU intensive operations. They take up about 100% of CPU resources thus making other operations very slow. Consider a worst case scenario, user 1 wants to cluster his data and user 2 needs to do Knn-impute on his data, since both are CPU intensive operations, this slows down the processor. This needs longer time to display the results for both the users. Hence future implementation will be to deploy clusters of computers for SMD users, so one cluster is used for clustering analysis and the other for Knn-impute, while the main application server acts like the master, distributing the work of the users to different clusters with out the knowledge of users, thus making the application faster. This thesis deals with parallelizing these programs which speeds up the application even more, thus helping researchers at UT.

Table 3.1: CPU utilization of SMD application server

Functions	% Utilizations
Cluster	100
Knn-Impute	100
Opening a Grid file	45
Loading an experiment	35
Viewing Images / Data	25
Deleting Experiments	5
Login	2

3.2 Working Procedure of SMD

The researchers at UT perform the microarray experiments, using microarray chips. The entire processes of performing experiments with the microarray chips is shown in the figure 3.2, and figure 3.3 shows the microarray chips of two different technologies, and the procedure of hybridization is shown in the figure 3.4. These chips are then scanned which gives the TIFF images of the chip, the procedure is shown in the figure 3.5. These images are fed into the Scanalyze tool and the grid image is formed which shows the actual spot location and the spot area to be considered for the data generation. The files required to load an experiment in to the database is shown in the figure 3.6. The data files, images and the grided files are loaded with the help of web interface. The advantage of SMD is that the experiments can be viewed, analyzed and quality check can be done from any where in the world as the application is web-based.

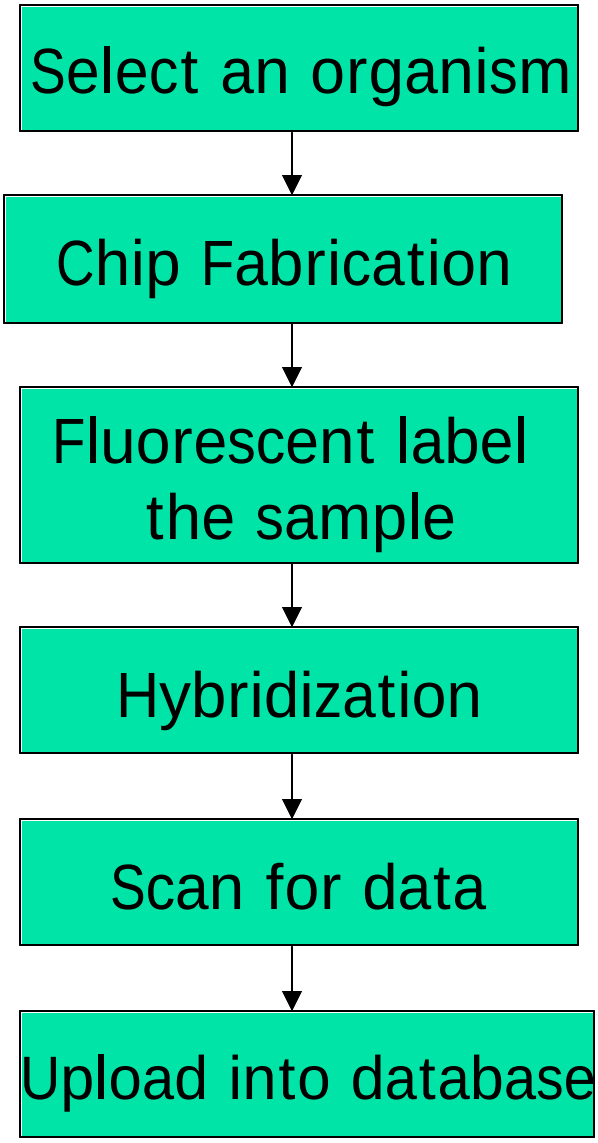

Figure 3.2: Procedure working with microarray chips

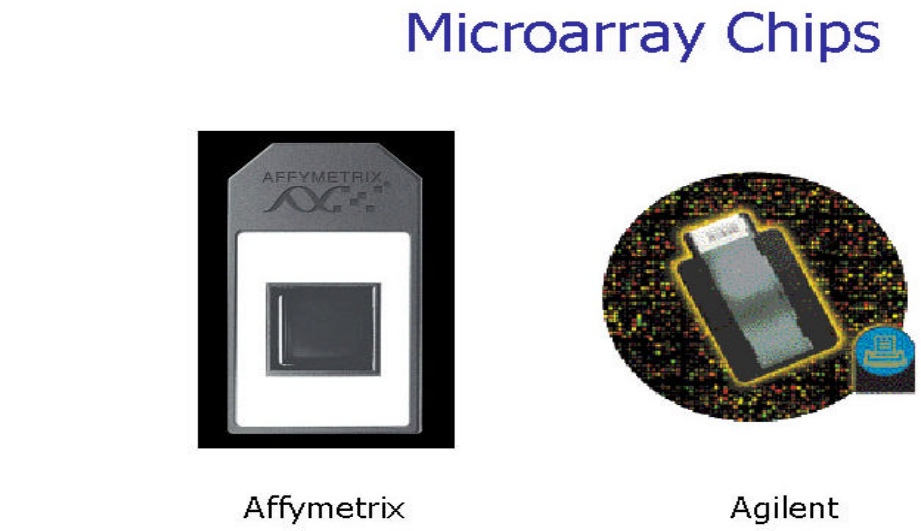

Figure 3.3: Microarray Chips, Affymetrix[23] and Spotted chips[24]

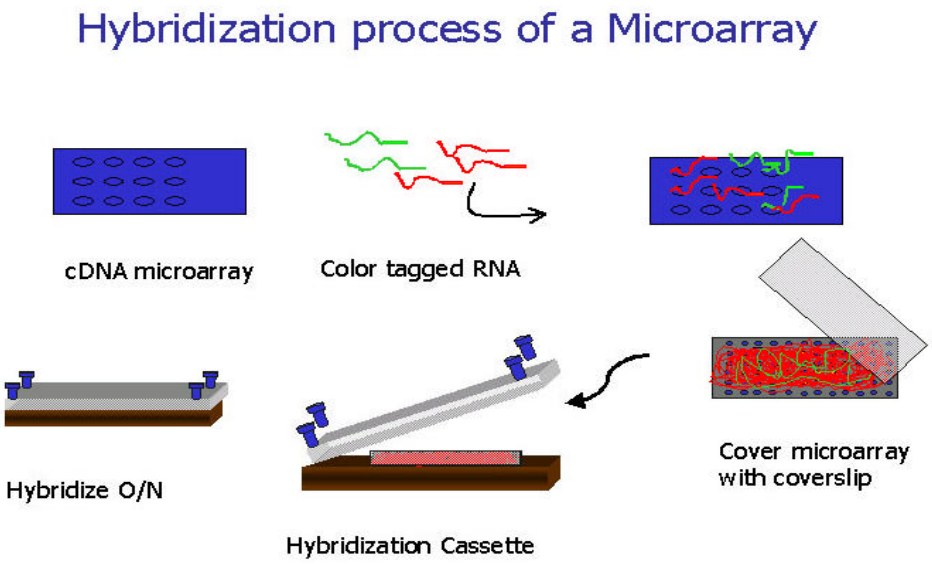

Figure 3.4: The Hybridization process [30]

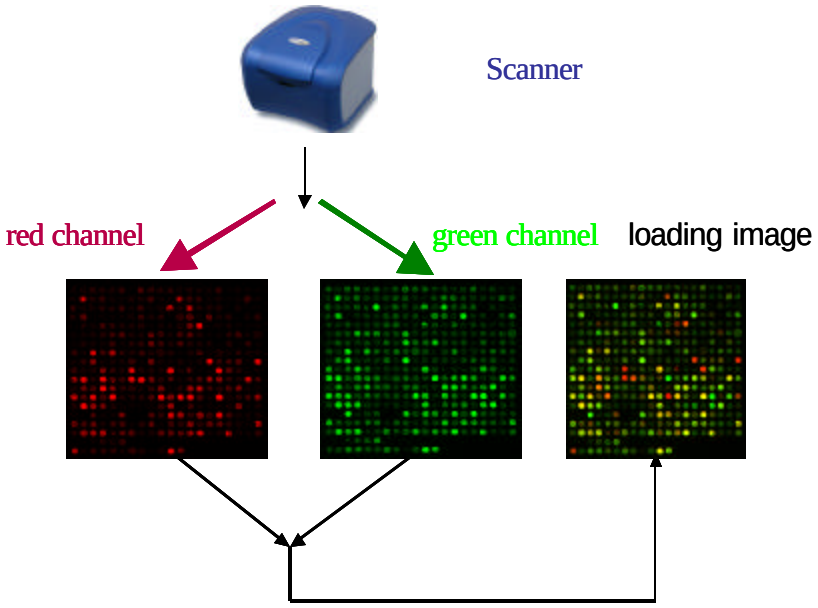

Figure 3.5: The Procedure of generating images with scanner

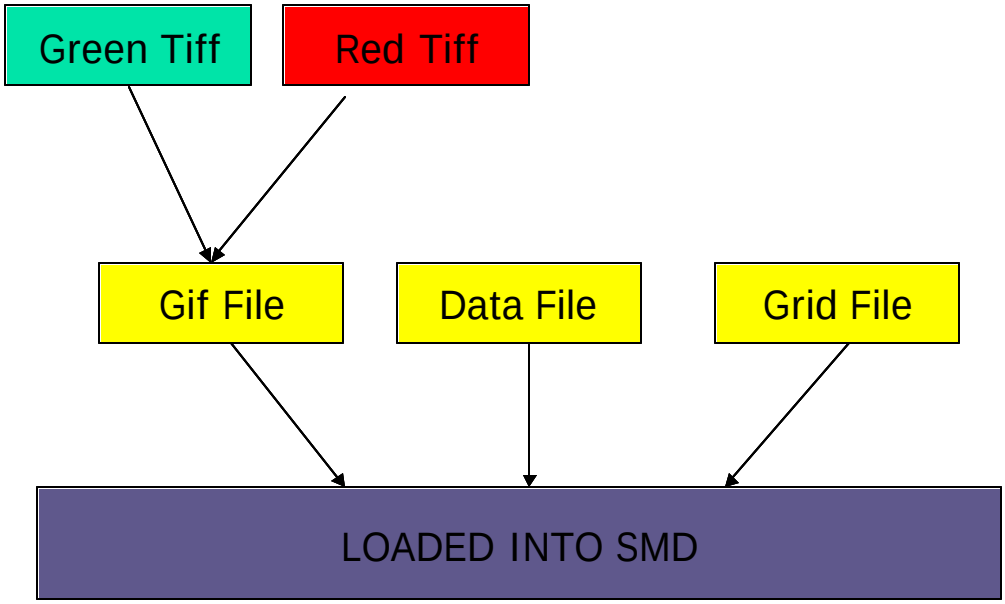

Figure 3.6: The Files required to load an experiment into SMD

3.3 Analysis Procedure of SMD

SMD provides different quality and analysis tools for the researchers to better understand the similarities and differences between various genes in the same experiment and also between the experiments. The researcher has several options to examine his data, based on various criteria the data can be filtered. The analysis is done on this filtered data file. Some times due to intensities of the image, or problems with the chips, a part data is lost, so Knn-impute is used for estimating the data, the following section describes the importance and working of Knn-impute. The output file from Knn-impute is fed into the hierarchical clustering program, the data is clustered according to the various criteria. The output from this clustering program is fed in to the graphical viewing interface and the final dendogram or tree view of the clustered data is displayed on to the screen. The description of these procedures is discussed in the following sections, with analysis times and calculation procedures etc.

3.4 K Nearest Neighbors (KNN) Impute Algorithm

However professional the researchers are, all microarray experiments are far from being perfect, hence the quality of data is not hundred percent. Since a typical microarray experiment has tens of thousands of spots on them not all these spots yield usable data. There are many reasons, like the defect in the chip, dust in the scanner or the hybridization of the chip is not done properly. This leads to flagging of majority of spots, thus losing the valuable information. On the other hand most of the analysis algorithms like hierarchical clustering and K-means clustering need complete matrix of gene array data to analyze the gene expressions, as they are not robust to missing data and lose effectiveness to few missing values[5].

The Knn-impute method imputes the missing values, by selecting the genes of similar expression profiles to that of the genes of interest from the complete matrix of data. If gene A has one missing value in experiment 1, this method would find K other genes, which have a value in experiment 1, with expressions most similar to A in experiments 2 to N where N is the number of experiments. A weighted average of values in experiment 1 from the K closest genes is than used as the estimate for the missing value of gene A[3]. In the weighted average, the contribution of each gene is weighted by similarity of its expression to that of gene A. Euclidean distance is used to calculate the gene similarity.

This method is highly accurate for the data sets up to 1%-15% of data missing. The smaller the percentage of data missing the higher is the accuracy of the estimated data. Suppose the data missing is 20% than there is a loss of 10% of accuracy. The method is relatively insensitive to the value of K in range of 10-20 neighbors. The performance declines with the lower value of K. The K value of 7 is recommended to be used on the data of UT's researchers by the Stanford. This method is not recommended for the data sets having less than 4 columns of data.

3.5 Hierarchical Clustering Algorithm

Fast and high-quality clustering algorithms are very important in analyzing and mining of data, by organizing large amounts of information into small number of meaningful clusters. Hierarchical clustering algorithm provides a view of the data at different levels of granularity, making them ideal for visualization and interactively exploring large data. Hierarchical clustering algorithms calculates in the form of trees called dendograms, which are of great interest for many applications[4]. Hierarchical trees provide a view of

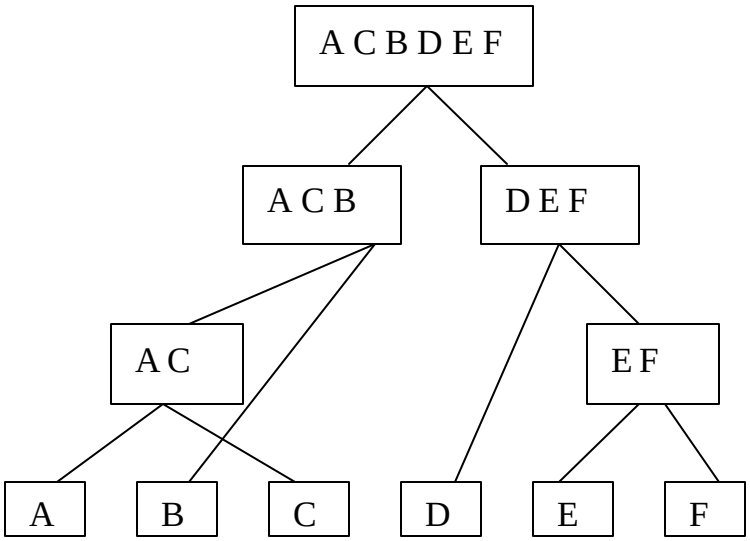

Figure 3.7: The Tree-view of the Hierarchical clustering algorithm

the data at different levels of abstraction. The clustering quality of Hierarchical clustering algorithm is superior to other partition algorithms like K-means. The figure 3.7 shows the dendrogram or the tree view which illustrates the fusion or division made at each successive stage of analysis of six data points.

In the today's world of biology, work in microarrays, sequenced genome, and bioinformatics has focused largely on algorithmic methods for processing and manipulating vast biological data sets. Hierarchical clustering has been shown to be effective in microarray data analysis for identifying genes of similar profiles and characteristics. The following chapter section's gives the description of the program used to cluster data and how it is parallelized.

3.6 Displaying Data with Graphical Viewing Interface

SMD has its own graphical viewing software, where it takes the output files generated by the clustering software and generates the images. It takes any where from few seconds to few minutes based on the size of the file. The clustering image is shown in figure 3.8.There are several options for visualizing and exploring the clustered data.

3.7 End to End Problem

The above sections describe the individual blocks of the whole problem. The figure 3.9 shows the whole problem end to end, the entire analysis part of the SMD application. The calculations are taken for the individual blocks and for the whole end to end approach.First the data is retrieved from the database by the applications server based on the filters and partitions the researcher is interested in. Then this data file which basically consist of the log transformed data which is needed for clustering to work properly is tested for the missing values and the missing values are estimated using the Knn-impute algorithm. The out put generated by the Knn-impute is the preclustered file (.pcl) is then fed into the clustering program. This program performs clustering on the input file and generates three file, clustered data table (.cdt) file which will contain the original data, but reordered to reflect the clustering. Apart from this a gene tree (.gtr) file is generated, when genes are clustered, and array tree (.atr) file is generated when the experiments are clustered. These tree file reflects the history of clusters are built. These file are then fed into the graphical viewing software and the results are displayed on the screen as an image with a list of links. Hence the researcher can view his clustered data according to the different options provided.

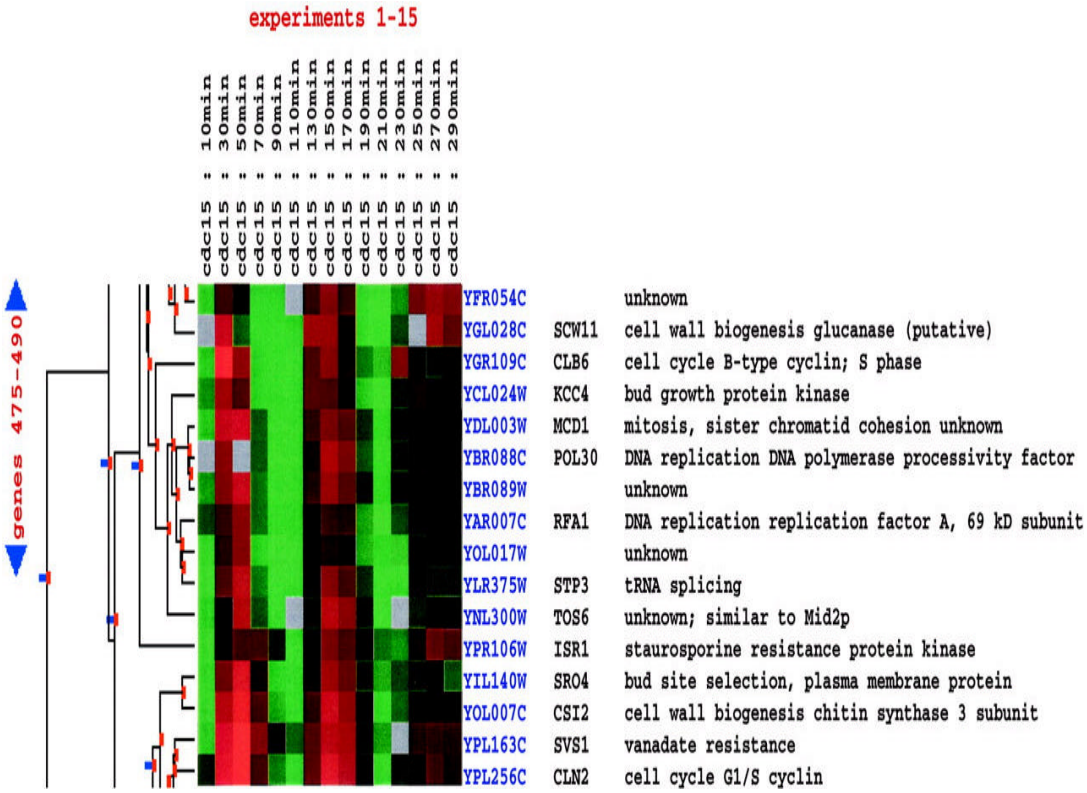

Figure 3.8: Clustered image generated by the SMD [22]

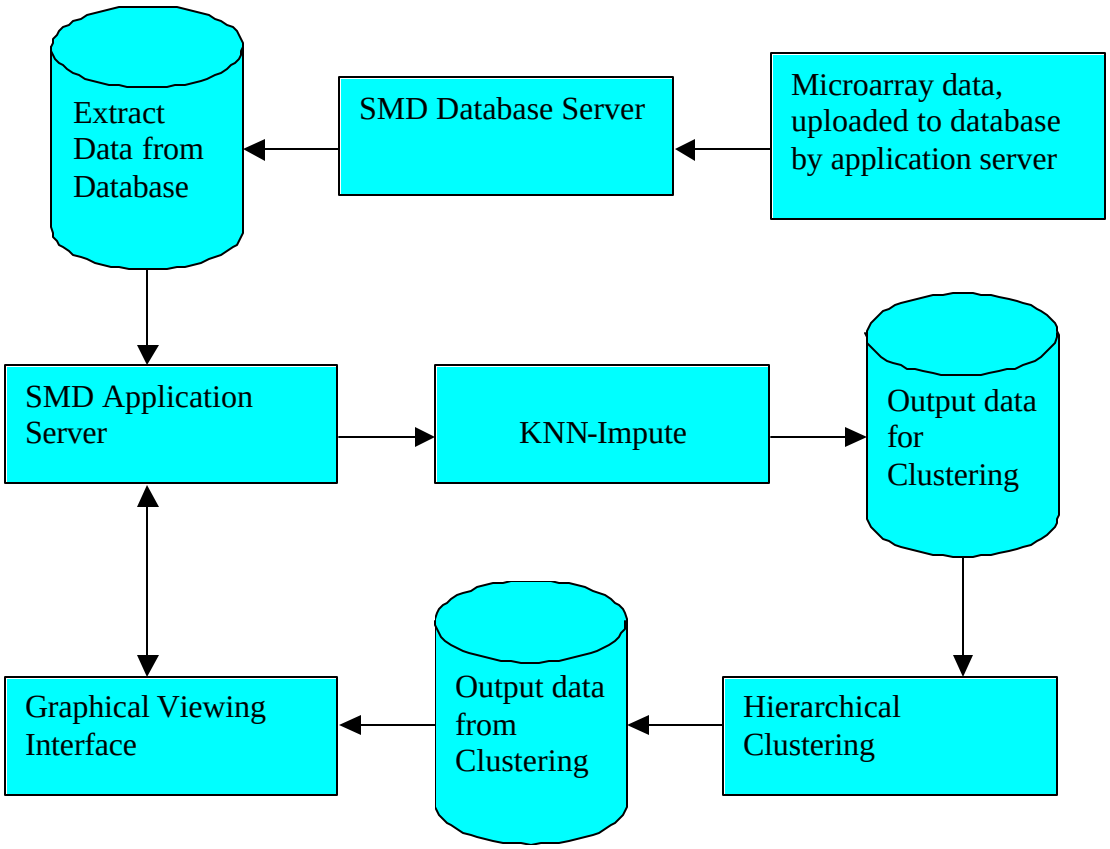

Figure 3.9: End to End Problem, design of entire application

3.8 Computation Times of Serial Implementation

The computation times for the block explained above are taken individually and as a whole end to end times of the entire application. The calculations are performed with different file sizes and on different types of clusters. The data file is generated from the database using the web. The data file description is discussed in the following section.

3.8.1 Data File Description

The data set was taken from the Stanford Microarray Database's mouse pathogen's published data, which is available for all, from the public search page. The data is published by department of Microbiology and Immunology, Stanford university school of medicine. The published article is "Distinct gene expression profiles characterize the histopathological stages of disease in Helicobacter-induced mucosa-associated lymphoid tissue lymphona"[6]. They used microarray analysis and laser microdissection to identify gene expression profiles characteristic and predictive of the various histopathological stages in a mouse model of the disease to define the cellular origin of the transcriptional responses. They showed that the disease follows a molecular progression, and identified a collection of genes whose expression level is predictive of disease stage. The reference for all arrays in this study consisted of pooled cDNA extracted from stomachs of age-matched uninfected control animals, 10 animals per time point. Data was filtered with respect to microarray spot quality like channel intensity, regression correlations, net mean, normalized mean and data distribution. The numerical data was Logbase 2 transformed. The microarray chips used where spotted 38,000-element murine cDna microarray that contains features derived from the RIKEN and NIA mouse clone sets. The total of 29 experiments data was collected for testing.

The second data set was taken from University of Tennessee, Nutritions department microarray experiments on human adiposite differentiation. The microarray chip is 19200 spotted array. The experiments where conducted on fat and lean patients. The data was filtered on above mentioned critters from 6 experiments.

3.8.2 Blockwise Computations

The table 5.1 in chapter 5 shows the computation times for serial implementation of the Knn-impute on the Linux cluster of the engineering department and the table 5.2 in chapter 5 shows the computation time for the serial implementation of hierarchical clustering algorithm on the VLSI cluster of electrical engineering department of UT. The data set used is the mouse data of 29 experiments and 38000 gene set as mentioned in section 3.8.1.

The graph in figure 5.1 shows the quadratic growth in the computation time of the Knn-impute program because of its computational complexity, with the increase in the number of genes which is proved in chapter 5.

The graph in figure 5.2 shows the squared growth in the computation time in hierarchical clustering program, with the increase in the number of genes which is proved in chapter 5.

3.8.3 End to End Computations

The time for downloading the data files is few minutes, irrespective of the file size. On the other hand the time taken for imaging the dendograms takes from few seconds to tens of

minutes shown in table 5.3 and the graph in figure 5.4 shows the growth of the computation time which is linear. From the above sections it is clear that the Knn-Impute and Hierarchical clustering combined together will occupy 90 percent of the end to end computation time. These are the potential problems to be solved. There are two approaches to solving this problem. One is parallel computing and other is reconfigurable computing. This thesis deals with the Parallelizing the Knn-impute and Hierarchical clustering programs. The next chapter deals with the Parallelizing the programs and the critical problems encountered and solutions to the problems.

Chapter 4

Parallel Approach

This chapter deals with the parallelization of the analysis programs of the SMD. This thesis shows how this approach effects the overall computation aspects of SMD. The first part of this chapter deals with the parallelization of the Knn-impute algorithm and the various difficulties experienced while parallelizing the program. The second part of the chapter deals with parallelizing the hierarchical clustering algorithm and the approach of its parallelization. The first part is comparatively easier than the second because there is no dependant data (this implies that the next data point to be calculated does not depend on the previous data point calculated). Second part is difficult because while clustering, the next nodes generated depends on the previous node, thus the dependencies are great which make it difficult to be parallelized.

4.1 Parallel Computing Architectures Used

Mainly two different clusters were used to evaluate the performance, one of the clusters, VLSI, is a UNIX platform and the other, the Engineering cluster, is a Linux platform. The VLSI cluster consists of 10 dual 450 MHz Ultra-SPARC-II RISC processors, with 1 GB RAM. These machines are connected with a Gigabit ethernet connection between them, through a Foundry Big Iron router located in Microelectronics Lab of Electrical Engineering Department, UT. The Engineering cluster located in Perkins hall consists of 7 2.5 GHz Intel pentium 4 dual processor work stations with 1 GB ram on a 100MB ethernet connection.

4.2 K-Nearest Neighbors Impute Algorithm Program Description

The Knn-impute program was written by Olga Troyanskaya and Michael Cantor of Stanford university. This program estimates the missing values of microarray data in a preclustered file format (PCL).The input to this program is the matrix with the missing values in pcl format, the number of neighbors to be used, the distance metric to be used is Euclidean in this case, the name of the output file. In this program the file is read and stored in to a matrix. This matrix consist of only the values, the additional columns and rows which has information such as name of the genes, heading of columns are stored in another matrix. The estimation of the missing values will be made with respect to the rows of the matrix which are not estimated before to remove the redundancy.

Once the matrix is stored and the distance to weigh the neighbors are stored the program proceeds to do imputation. The rows with the missing values are first found and after that the missing values are estimated. The k nearest neighbors are then found and compute weights of these neighbors based on 1/distance (1/d). The program avoids the division with zero. Then the program gets the sum of 1/d values for normalization and gets the weights to estimate the values based on the neighbors. Finally the copy of matrix is filled with the estimated missing values. This copy of matrix along with the matrix with names and all row and column information is used to output the final estimated file which is used for clustering. The working of the Knn-impute program is described using a flow chart which is shown in the figure 4.1.

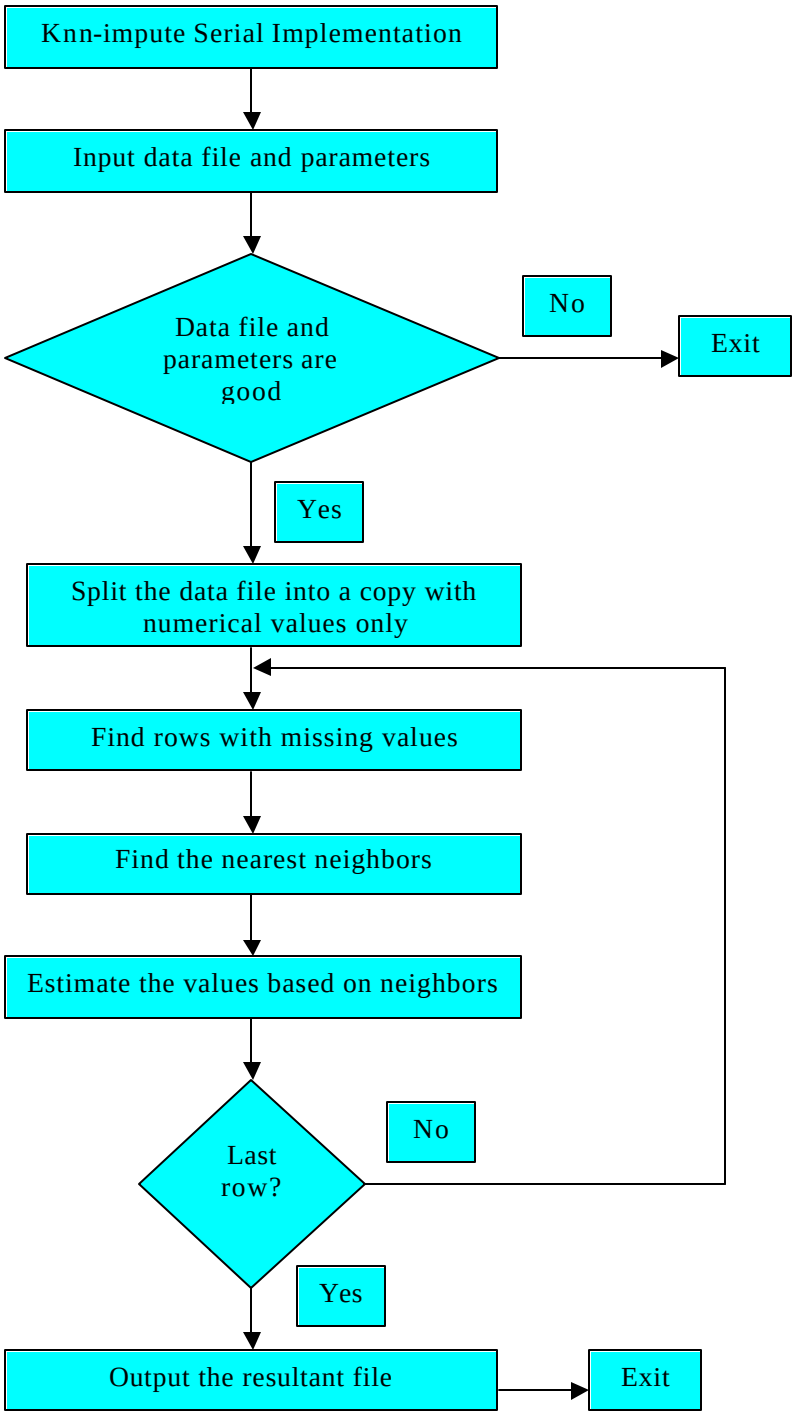

Figure 4.1: Knn-impute serial implementation

4.3 Parallelization of Knn-Impute Program

The technique used to parallelize the program was comparatively easier than the clustering program because of non dependent data. The parallelization technique used here is the master and slave concept. The master is responsible for sending the data matrix to all the slaves and collecting the estimated data from all the slaves and putting in a final file format. The master also decides the work load of each slave which is almost equal. Which implies master manages the task of dynamic load balancing and merges the results produced by the slaves. On the other hand slaves receives the data from the master, does the imputation on the data received and sends back the estimated data to the master.

The program works like this, first a copy of the matrix containing the numerical values is sent to all the slaves, along with the starting row and the ending row numbers for each slave on which the imputations should be performed. Since the data files are any where from 500KB to 15MB proper memory should be allocated for the data files both in the master and slaves. The slaves receive the data, and the starting row number and the ending row number. The slaves check for the rows with the missing values and performs the imputation for the missing data for the allotted rows. Then the estimated rows along with the other rows are sent back to the master. The master will receive the data from all the slaves and outputs the resultant file.

The parallel implementation of the Knn-impute program is shown as a flowchart in the figure 4.2.

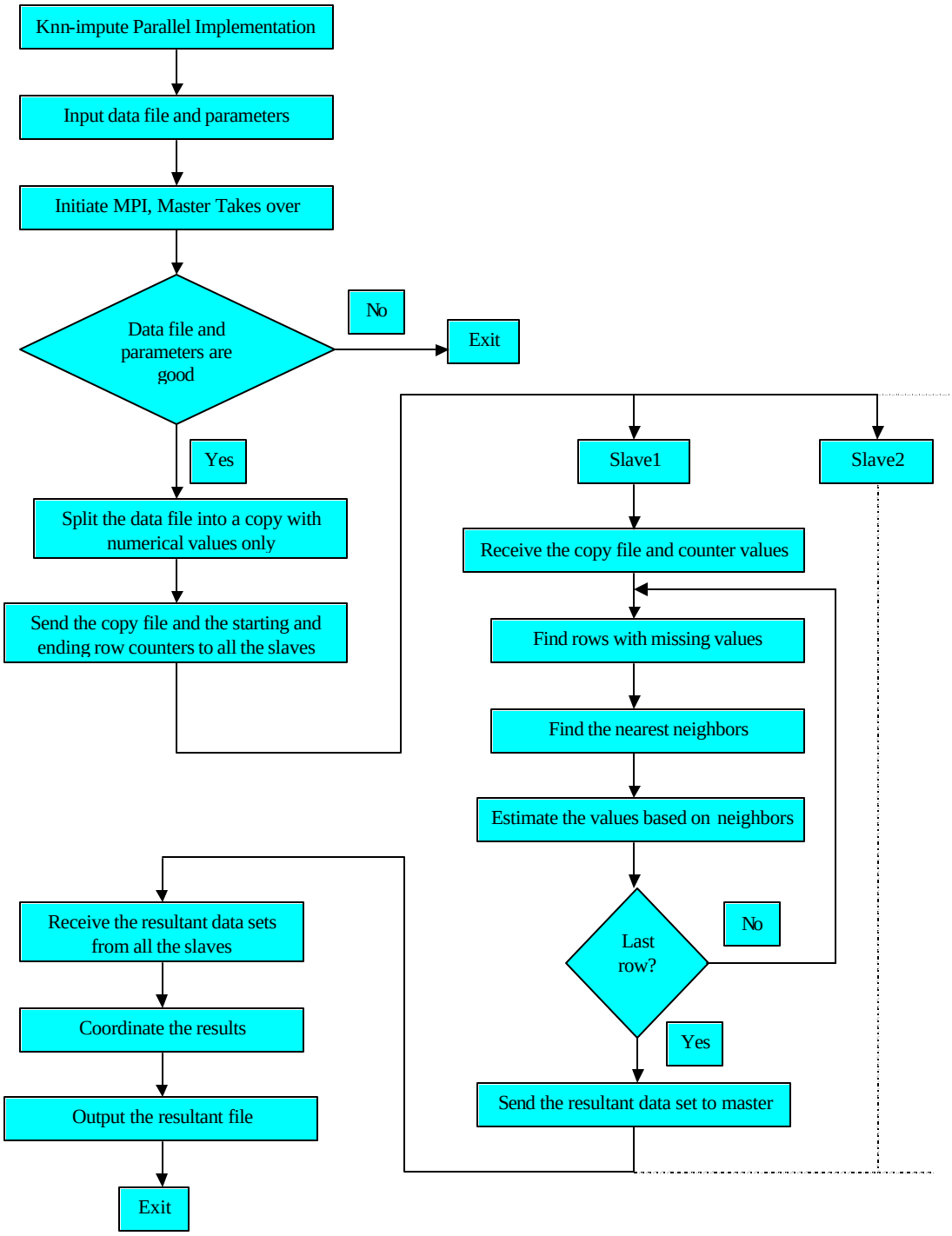

Figure 4.2: Knn-impute parallel implementation

4.4 Clustering Algorithms Program Description

To analyze the data, cluster.c program of SMD is used to cluster data, this program was written in C by Dr. Gavin Sherlock and group, Genetics Department, Stanford University. This program can cluster data using Hierarchical clustering, Self-Organizing maps and K-Means clustering algorithms. Hierarchical clustering is used for know. The program reads the input file in the text format, which has gene name, gene ID, experiment ID, experiment weight, gene weight, and the logbase ratio values. The program also checks the file for missing values and the program terminates if it finds any. The size of the data is read which determines the number of experiments, and the total number of genes in the experiment set. If the data values are not logbase values than the program also calculates the logbase values. These logbase ratio values from the data file are fed into the array matrix, which is used by all the functions of the program. As mentioned before user has three algorithms to choose from, Hierarchical, Self Organizing Maps and K-means, because of the popularity of the Hierarchical clustering algorithm, this is used to parallelize.The figure 4.3 illustrates the program flow.

4.5 HirarchicallyCluster Function Description

This function handles all the necessary calls which are required to perform the hierarchical clustering. This function opens and fills the output files. This function also determines whether the clustering is done according to the genes or experiments or both genes and experiments. This function mainly consists of two important functions one is MakeCorrelation, which calculates the correlation coefficients. The other function is ClusterNodes function which cluster the nodes according to the coefficients. The figure 4.4 illustrates the hierarchical cluster function flowchart diagram.

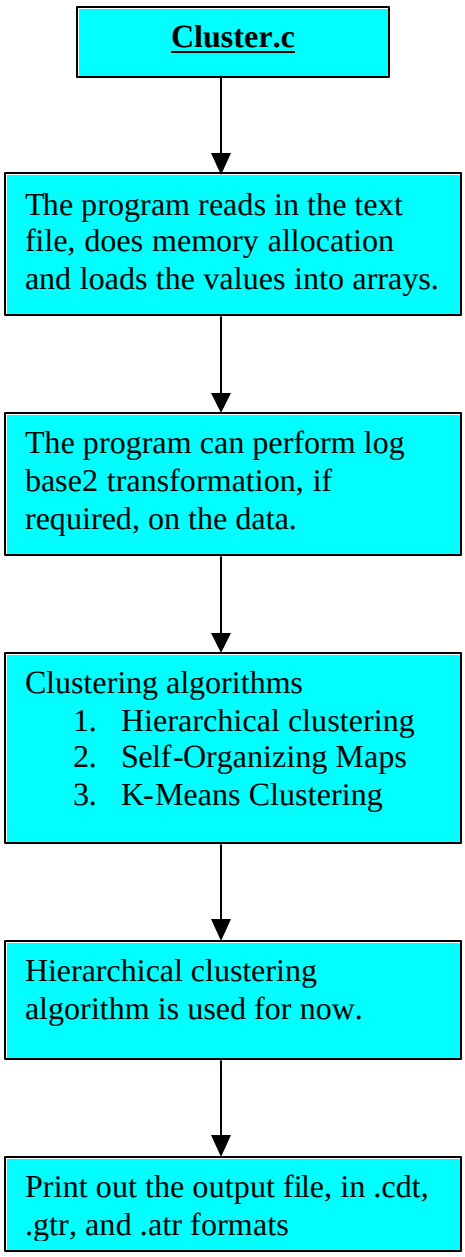

Figure 4.3: Clustering program Flow

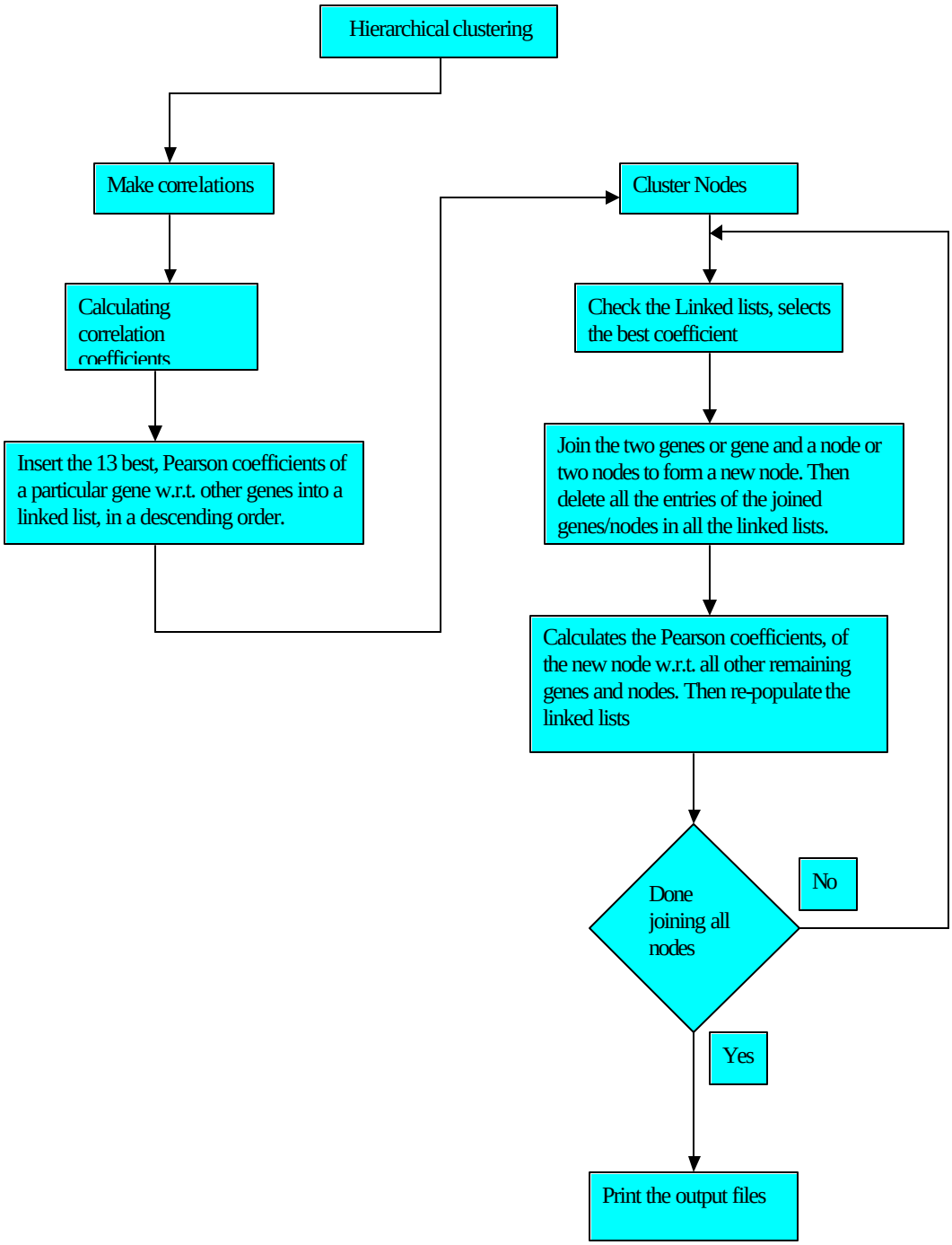

Figure 4.4: Hierarchical Clustering function flow

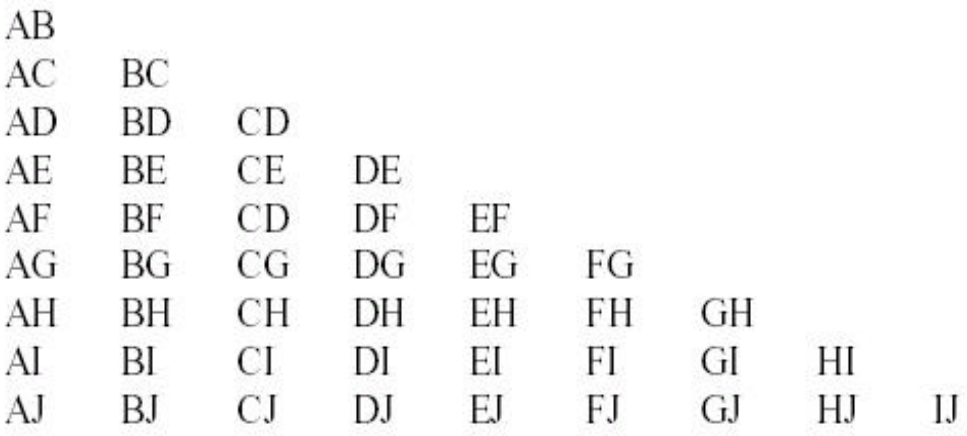

Figure 4.5: Calculation of correlation coefficients

The HierarchicallyCluster function outputs three files.cdt, .gtr, .atr files, which are fed into graphical viewing software called TreeView, which displays the clustered data. The figure 3.8 displays the tree view which is generated by the graphical viewing software.

4.6 MakeCorrelation Function Description

From the figure 4.4, it is evitable that MakeCorrelation function calculates the coefficients by comparing each profile with every other profile. This function compares gene one versus gene two but not vice versa, as shown in the figure 4.5, thus halving the total number of calculations performed. Then these coefficients are sorted and filled into the linked lists, based on the maximum number of the best values defined in the header file, for this program 13 was used.

Suppose the data set contains 10 genes A, B, .. J then the coefficients are calculated as illustrated in the figure 4.5. So, for 10 genes the total number of correlation coefficients calculated are 9+8+..+1=45. For n genes the total number of correlation coefficients will be n(n-1)/n. For example, for 19200 genes the total number of coefficients is equal to 184,310,400.

4.7 ClusterNodes Function Description

In this function genes are clustered based on the best correlation coefficients between the genes. The first entries of all the linked lists of all the genes are browsed and the best correlation is picked, to form a new node. As each node is formed, the correlation coefficients are calculated with respect to all the remaining genes or nodes, that have not themselves has been assimilate into larger nodes. Once a node or a gene is assimilated into larger node, the correlations that did exist for that node or gene are deleted from every linked list of correlations. This process is repeated till all the genes and nodes are joined and the entire tree is formed. Analyzing this huge volume of data is challenging and complex because of the interdependent factors and the uncertainty of the dependencies.

4.8 Parallelization of Clustering Program

The main functions that are parallelized are the MakeCorrelation and the ClusterNodes, as these two functions occupy 90% of the total calculation time of the entire program based on the statistical software utility GPROF. Certain measure are taken to parallelize and save time where ever it is possible, because there exists a lot of dependencies in data in this algorithm which makes it difficult to parallelize. As like the Knn-Impute the

master and slave concept is used in this program too. The work of the master is the allot work to the slaves and collect back the data and validate the data and output the files required for display. To save time as soon as the data matrix is filled the data matrix is sent to all the slaves for calculation purposes. Than the master divides the work for each slave and sends them. Which implies master manages the task of dynamic load balancing and merges the results produced by the slaves. The slaves perform calculations on their assigned work and send back the results to the master. This concept will be understood clearly from the following sections.

4.8.1 Parallelization of MakeCorrelation Function

This function occupies the 30% of the calculation time of the whole program, so this is chosen for parallelization.Figure 4.5 shows a schematic of the coefficients calculations, which are illustrated in the form of a right angled triangle, described in the figure 4.6. It has certain regular features, which can be explored for parallelization. Note the triangle of calculations can be divided between the processors according to the following reasoning.

The total number of genes = n

The number of slave processes to calculate the coefficients = p

The total number of coefficients to be calculated =

$$\frac{n(n-1)}{2}$$

The number of coefficients per slave process =

$$\frac{n(n-1)}{2p}$$

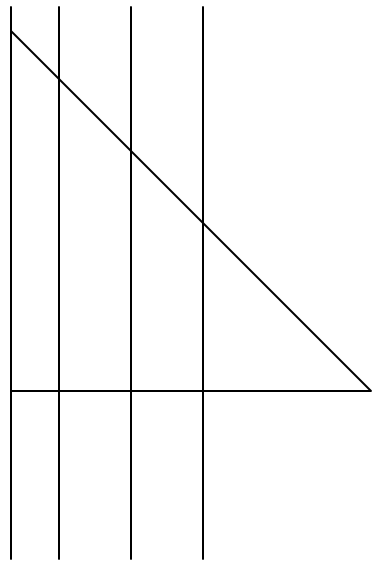

Figure 4.6: The schematic view of the correlation coefficient calculations

Dividing the above triangle into equal number of partitions based on the number of slaves, as follows

$$\frac{n(n-1)}{2p} = \frac{n_0(n_0 - 1)}{2} - \frac{n_1(n_1 - 1)}{2}$$

n_0=n,

Solving the above equation for n_1.

$$n_1(n_1 - 1) = n(n-1)(1 - \frac{1}{p}) = c$$

The above equation is in the form of the quadratic equation,

$$n_1^2 - n_1 - c = 0$$

Solving for n_1 gives,

$$n_1 = \frac{1+\sqrt{1+4c}}{2}$$

Once n_1 is found, n_2 and ... can be solved, this divide the calculations into equal partitions for each of the slave processes, thus optimizing the calculation time.

The whole data set of intensity values is received by all slaves immediately after the input file is read and the data array is filled, to save time. Then the calculations are divided based on the total number of genes and total number of slaves as shown above. The starting and ending number of the counter, and the total number of coefficients to be calculated are received by the slaves. The slaves then calculates the coefficients and send back to the master. The master receives the correlation coefficients, sorts them and then inserts them into the linked lists. Receipt of the coefficients from the slaves are in the order, so that they are inserted correctly. The parallel implementation of the MakeCorrelation function is shown in the figure 4.7.

4.8.2 Parallelization of ClusterNodes Function

For a big gene set in thousands, the first few hundreds of nodes may not have any dependencies, which could be parallelized, without any problem. This implies that these nodes can be formed ‘p’ at a time, where p is the number of slave processes. Again the main time consuming task, is to calculate the correlation coefficients, hence these

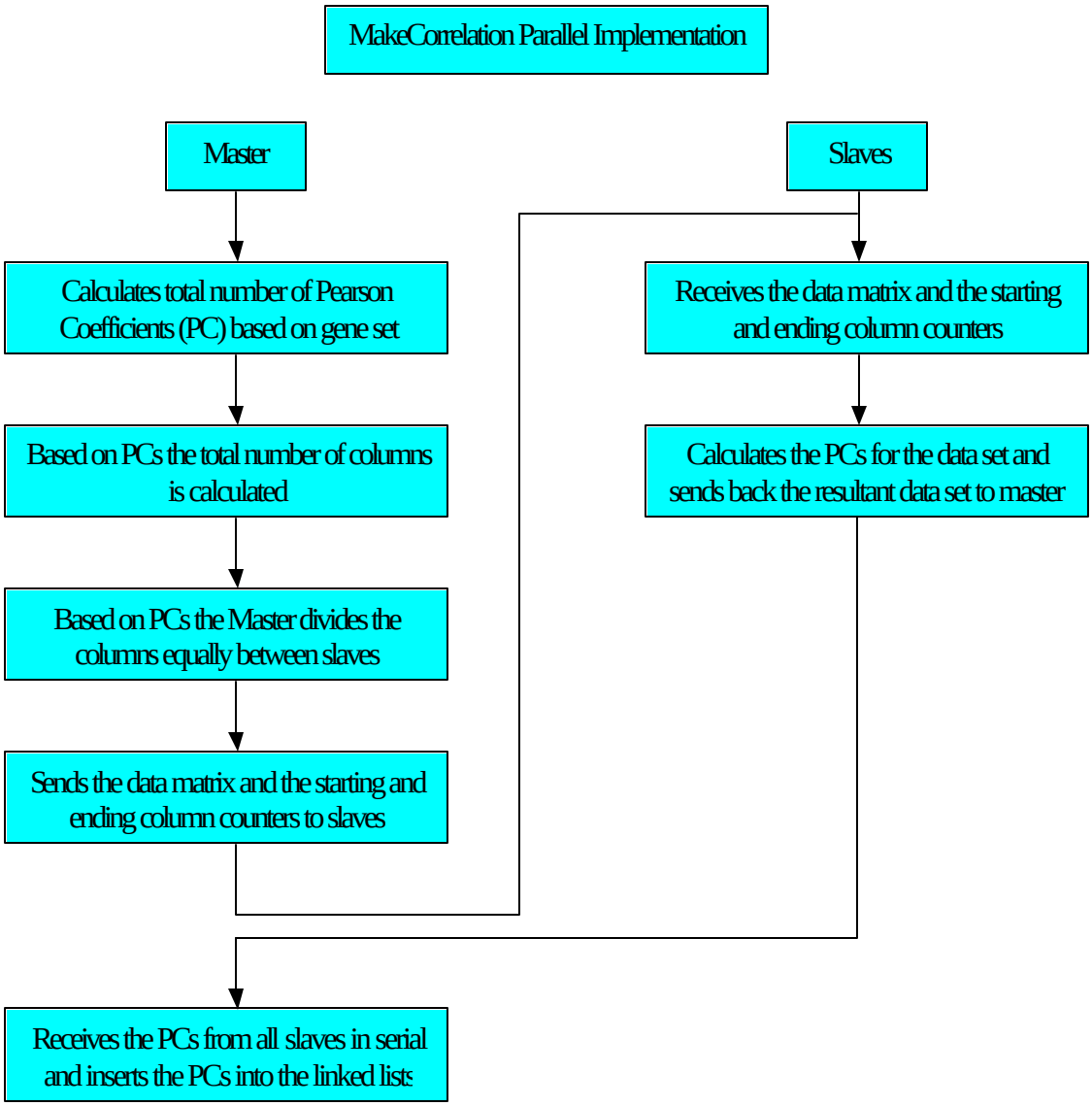

Figure 4.7: MakeCorrelation functions parallel implementation

calculations are done in slave processes. All the operations such as finding out the maximum correlation, joining the nodes, deleting the joined nodes and comparing is done in the master.

The nodes are represented as N1, N2 .. and genes as g1, g2.. joining two genes, or two nodes, or a gene and a node forms the nodes. How the dependencies occur and why they are a problem is shown below. Consider there are three slaves and so three nodes are formed at one time as illustrated below.

N1	g1 and g8	
N2	g7 and g3	these can be calculated in parallel without any problem
N3	g4 and g5	
N4	N3 and g9	
N5	g10 and N2	since N3 and N2 nodes are already calculated, this not a problem
N6	g6 and g12	
N7	g11 and N6	
N8	N7 and G14	N8 and N9 are the problems because they depend on the previous
N9	N8 and N1	calculations in the same loop.

If a node depends on the nodes that are formed in the same loop, then these calculations cannot be parallelized, these are the nodes to be dealt with. The only solution in this case, the master disregards all the calculations received from the slaves, and re-do them

in a serial fashion. Consequently this solution can be termed as conditional parallelization.

Once the calculations are performed by the slaves the master prints the .cdt, .gtr and .atr files and this completes the hierarchical clustering. The following chapter shows the results and graphs for the different calculation times and all these serial and parallelized programs.

The parallel implementation of the ClusterNode function is shown in the figure 4.8.

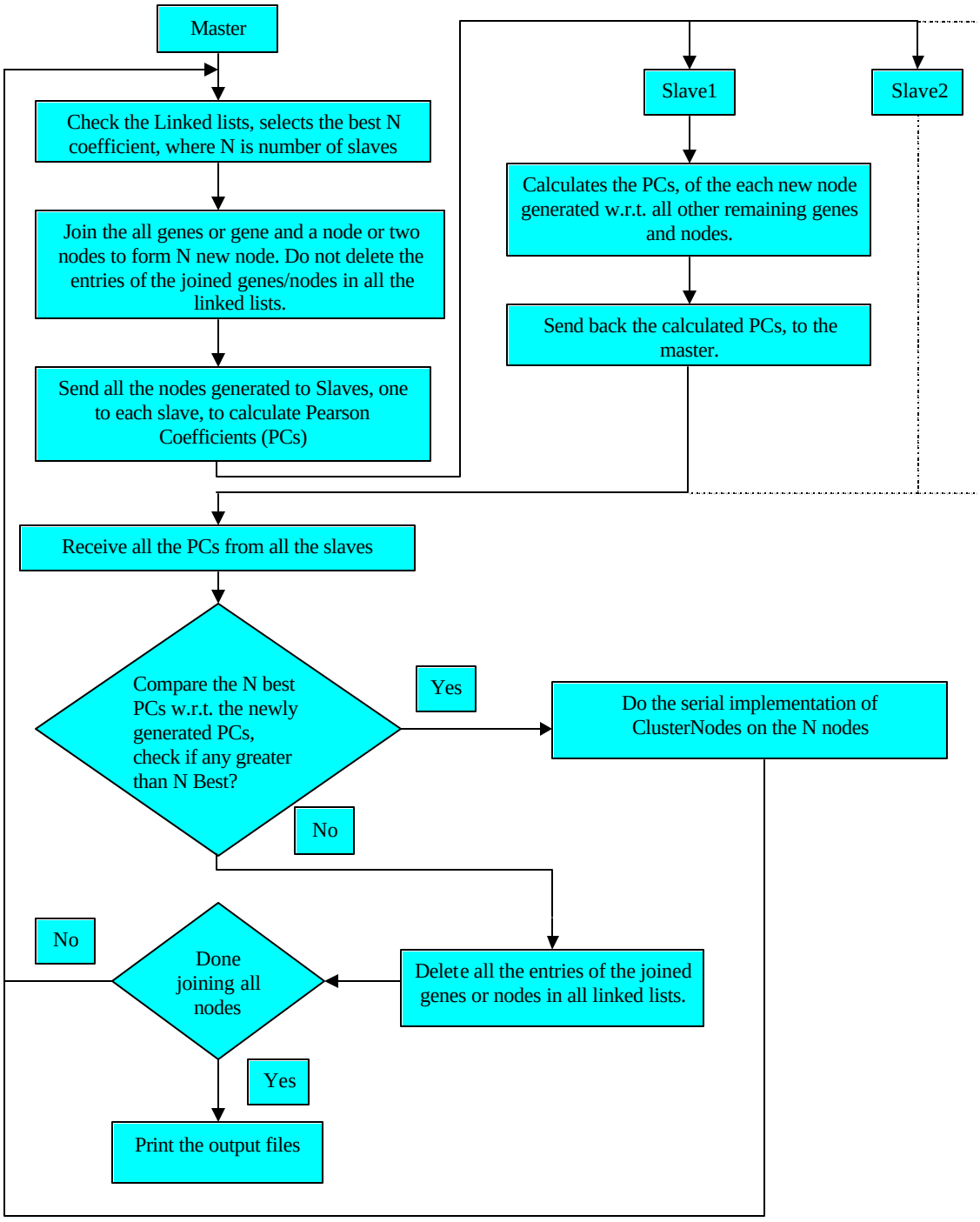

Figure 4.8: ClusterNodes functions parallel Implementation

Chapter 5

Results and Graphs

This chapter discusses about the various results obtained for serial implementation and parallel implementation. First this chapter focuses on the serial implementation of individual blocks, and the end-to-end implementation, and how the computation is growing.Next half of the chapter, shows the results achieved by parallelization. The last section displays the speedups achieved with parallelization, block wise and of the whole application.

5.1 Knn-Impute Computation of Serial Implementation

The following table 5.1 shows the time taken for implementing the Knn-impute program on a single machine of the Linux cluster which acts as a master for parallelization. The resultant graph on the data is shown in the figure 5.1.

Table 5.1: The serial implementation of Knn-Impute on Linux cluster for 29 experiments and 38000 gene set

Gene Count	Computation time(s)
1700	13
2000	17
4800	126
10000	522
14100	1074
17600	1363
21800	2285
27000	3976
30800	6214
33500	9013
36700	16493

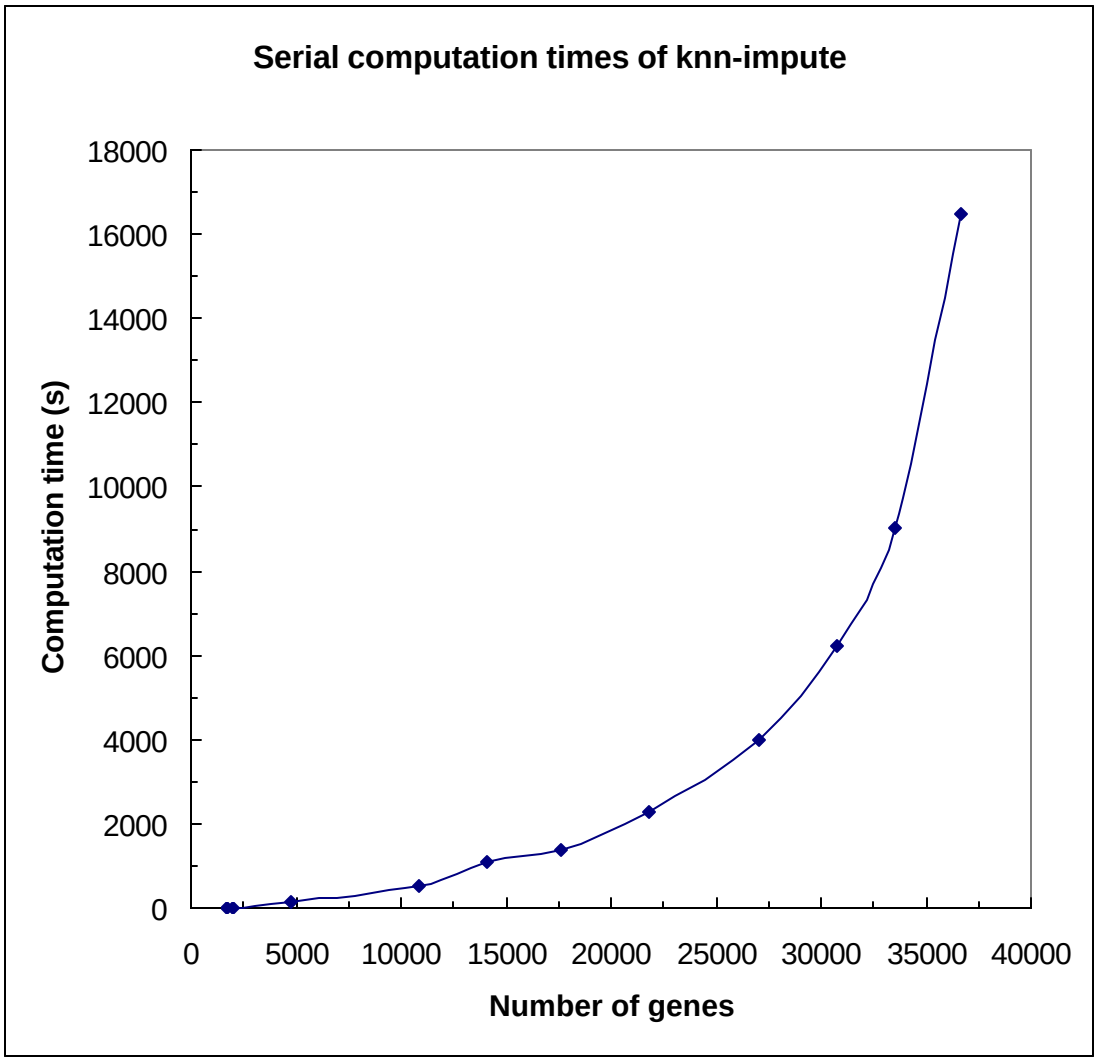

Figure 5.1: Computation time for Knn-Impute serial implementation

The data set consists of around 37000 genes and 29 experiments. The figure 5.1 shows a quadratic growth. As the computational complexity is $O(mn^2)$, where m is the number of experiments and n is the number of genes in the data set [5].

5.2 Hierarchical Clustering Computation of Serial Implementation

The clustering program was run on the VLSI cluster, the computation times are recorded on the VLSI2 machine. This machine is used as master for the parallel implementation of the clustering program. The data was recorded at non-peak time when the load on the machine is low. The computations were performed at night to get better performance. The table 5.2 shows the recorded computation times for different gene sets for the 29 experiment data. The computational complexity of the program in $O(n^2)$ where n is the number of genes in the experiment set. From the figure 5.2 we can see the squared growth of the computation times. Figure 5.3 shows both Knn-impute and hierarchical clustering computation times together for comparison.

Table 5.2: The serial implementation of clustering program on VLSI cluster for 29 experiments and 38000 gene set

Gene Count	Computation time(s)
1700	16
2000	21
4800	121
10000	641
14100	1120
17600	1715
21800	2791
27000	4306
30800	5622
33500	6713
36700	8096

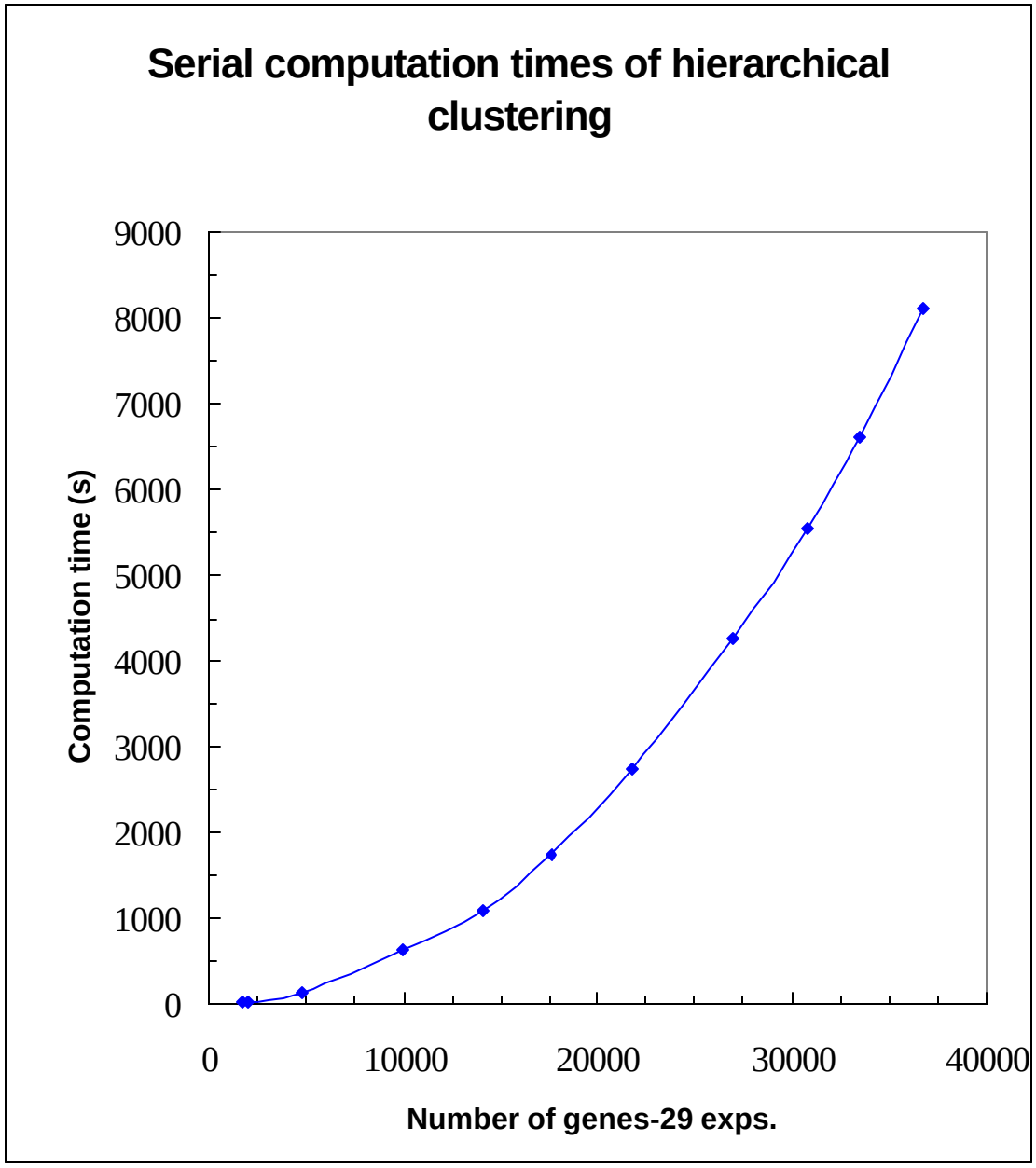

Figure 5.2: Computation time for hierarchical clustering serial implementation

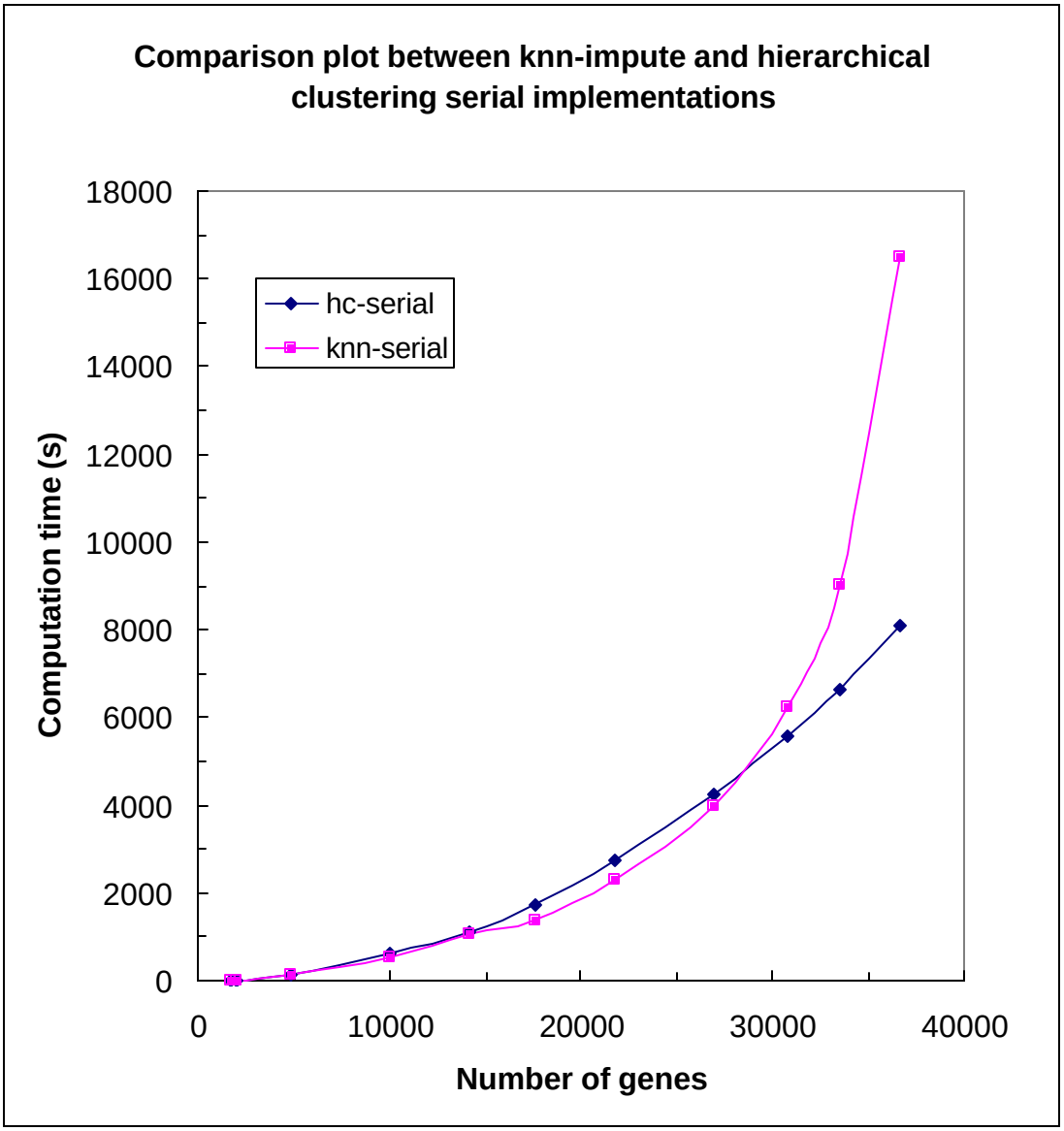

Figure 5.3: Comparison plots of Knn-impute and hierarchical clustering programs serial implementations

The growth of the Knn-impute dominates the growth of the hierarchical clustering which is seen clearly from the figure 5.3.

5.3 End to End Computation of Serial Implementation

Apart from the Knn-impute and hierarchical clustering computation times the analysis model consist of two other part to it which contribute some time to the application completion. One is the time taken for the generation of data files from the database of SMD which is not much, it contributes only few minutes to the application at the most. On the other hand the time taken to generate the image file with the help of graphical viewing software from the output generated by the clustering programs is little more. The time taken by the image generation can effect the overall speed-up of the application. The table 5.3 gives the time taken by all the blocks of the application and the total time taken for the end-to-end of the application.

Table 5.3: The serial implementation entire application for 29 experiments and 38000 gene set

Genes	Data file generation time(s)	Knn-impute computation time(s)	Hierarchical Clustering time(s)	Image file generation time(s)	Total time(s)
1700	10	13	16	45	84
2000	12	17	21	57	107
4800	36	126	121	136	419
10000	65	522	641	298	1526
14100	120	1074	1120	417	2713
17600	140	1363	1715	522	3740
21800	196	2285	2791	648	5920
27000	220	3976	4306	804	9306
30800	280	6214	5622	918	13034
33500	300	9013	6713	998	17023
36700	340	16493	8112	1095	26040

Figure 5.4 shows almost a linear growth of the time taken for data file and image file generation combined. Figure 5.5 show the computation time for the entire application. The figure 5.5 shows the growth of the computation time of the entire application. From the figure 5.5 the growth is almost quadratic as the Knn-impute part effects the computation time of the entire application. Thus parallel computing technique is used in this thesis to answer this problem of this growth. This technique proved effective which is shown in the following sections of parallelization and speed-ups. In the figure 5.6 DIT is the time taken for the data file and image generation and KNNT is the time taken for the Knn-impute program and HCT is the time taken for the hierarchical clustering program. From the figure it is clear that the time taken by data file and image generation is low when compared to impute and clustering programs. The impute program takes up much of the time, and than the hierarchical clustering program of the entire application, which is will be clear from figure 5.7.

The impact of the Knn-impute part is the most on the entire application because of its computational complexity. The computational complexity of Knn-impute is $O(mn^2)$ where m is the number of experiments and n is the number of genes. The growth in the computation time is quadratic, it could be cubed, when m=n and this is a true case scenario. The best example is in oncology, the more samples of tumor your have the more accurate is the results. The human genome has approximately 33000 genes, and the experiments are tremendously increasing so the m>n scenario is also possible. On the other hand the computational complexity of hierarchical clustering is $O(n^2)$ and the computational complexity of data file and image generation is O(n). This shows the impact of impute part of the application is most.

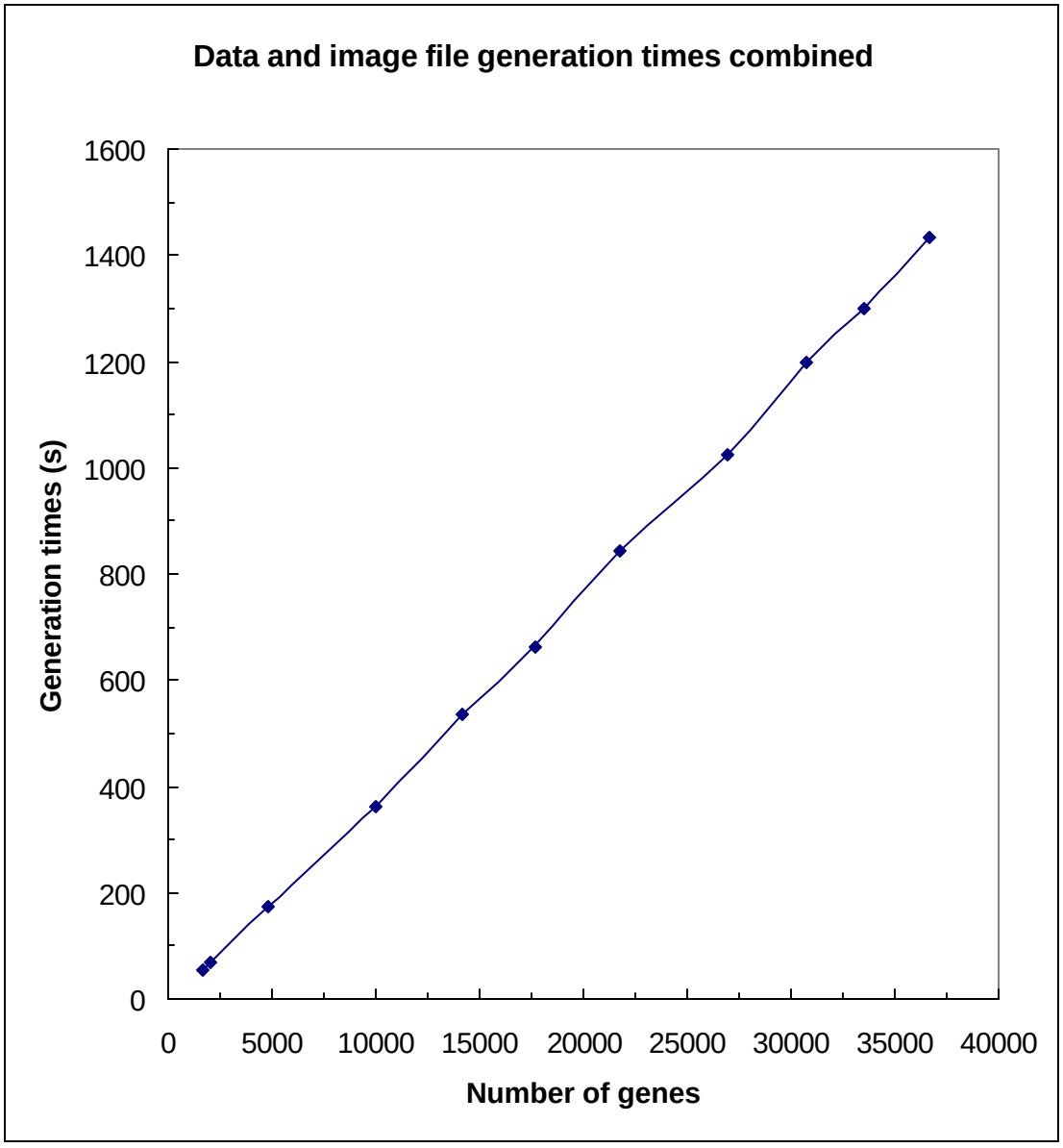

Figure 5.4: Estimated time taken for data file and image generation together for serial implementation

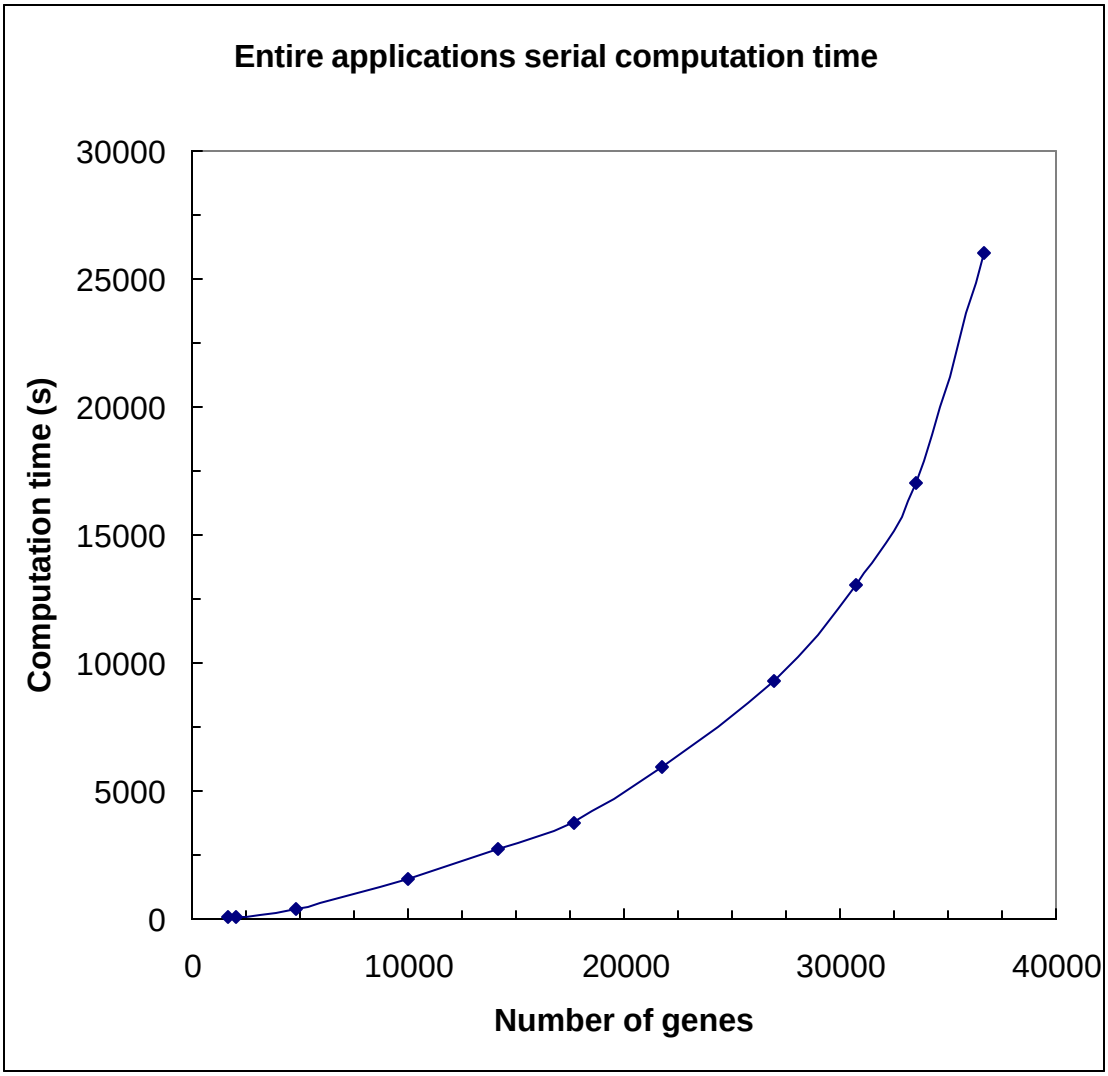

Figure 5.5: Time taken for entire application's serial implementation

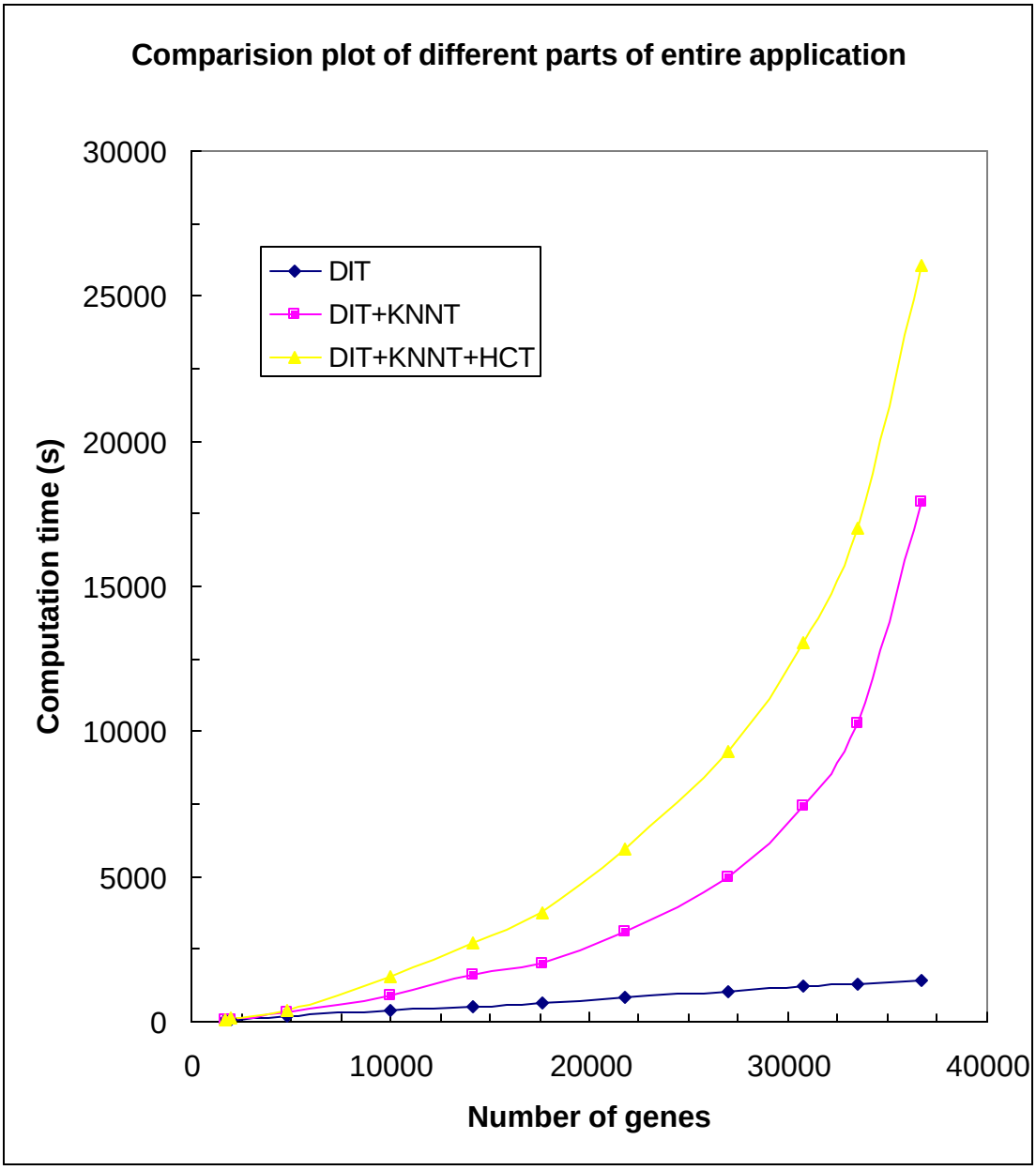

Figure 5.6: Time taken for different parts in sequential order, of entire application

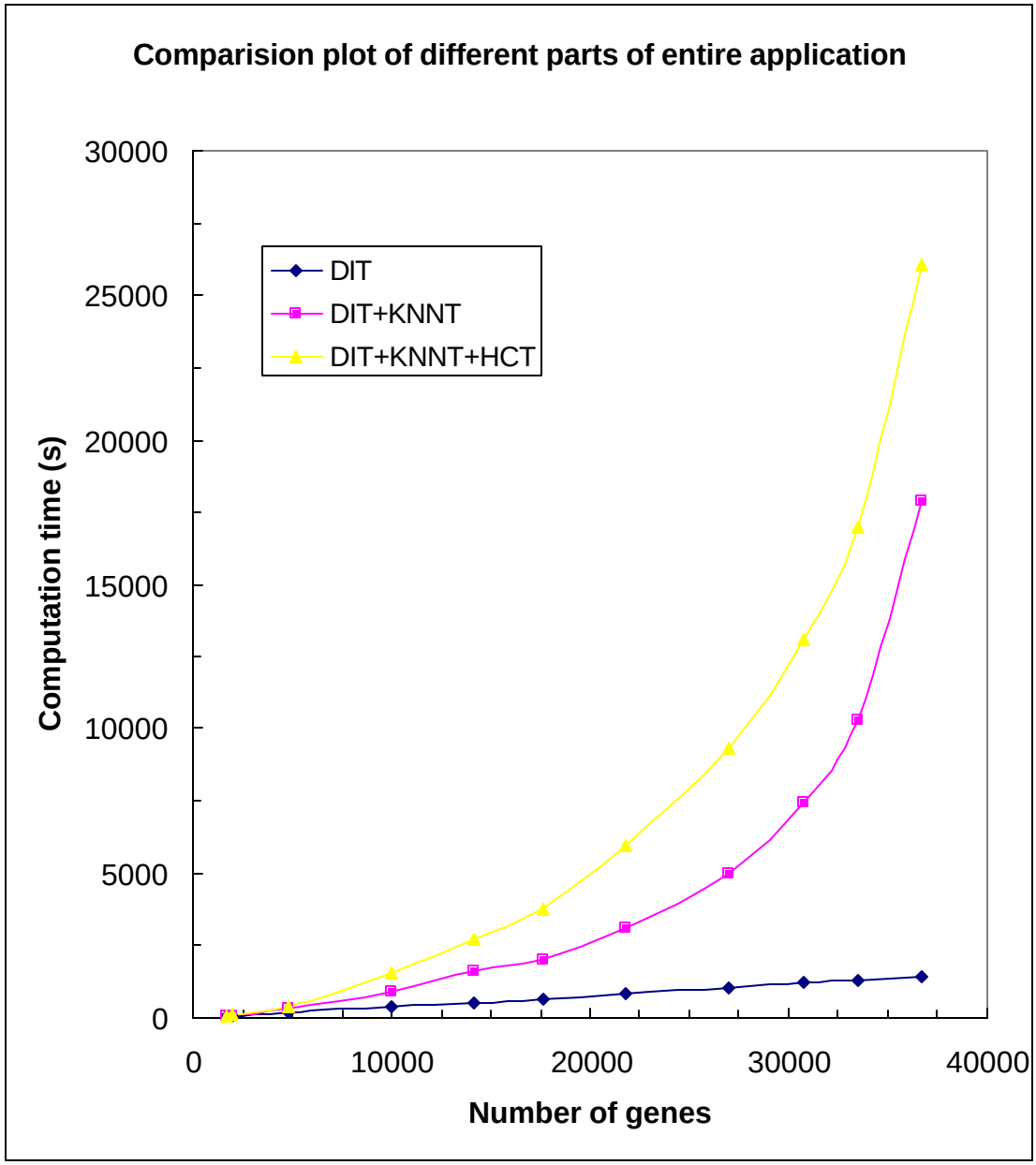

Figure 5.7: Time taken for different parts showing the growth dominant part of entire application

5.4 Knn-Impute Computation Time of Parallel Implementation

The table 5.4 shows the results of the parallel implementation of impute program. The linux cluster is used for the parallel implementation for Knn-impute. First parallel computation time is recorded on a cluster of 7 machines with one process each. To check whether there will be any improvement in computation time, three processes are used on each machine using only 6 machines of the cluster this time, the second implementation did not improve the computation time. The graphs are plotted for these values shown in figure 5.8. A comparison plot is shown in figure 5.9 between the serial implementation and parallel implementations, to check the difference in computation times and success achieved with parallelization. From the figure 5.9 it is clear that with the parallelization the speedup of almost 6 is achieved.

.

Table 5.4: The parallel implementation of Knn-Impute on Linux cluster for 29 experiments and 38000 gene set

Gene Count	**Comp. time(s) with 7 computers with 1 process each**
1700	55
2000	56
4800	77
10000	154
14100	227
17600	308
21800	460
27000	759
30800	1154
33500	1635
36700	2773

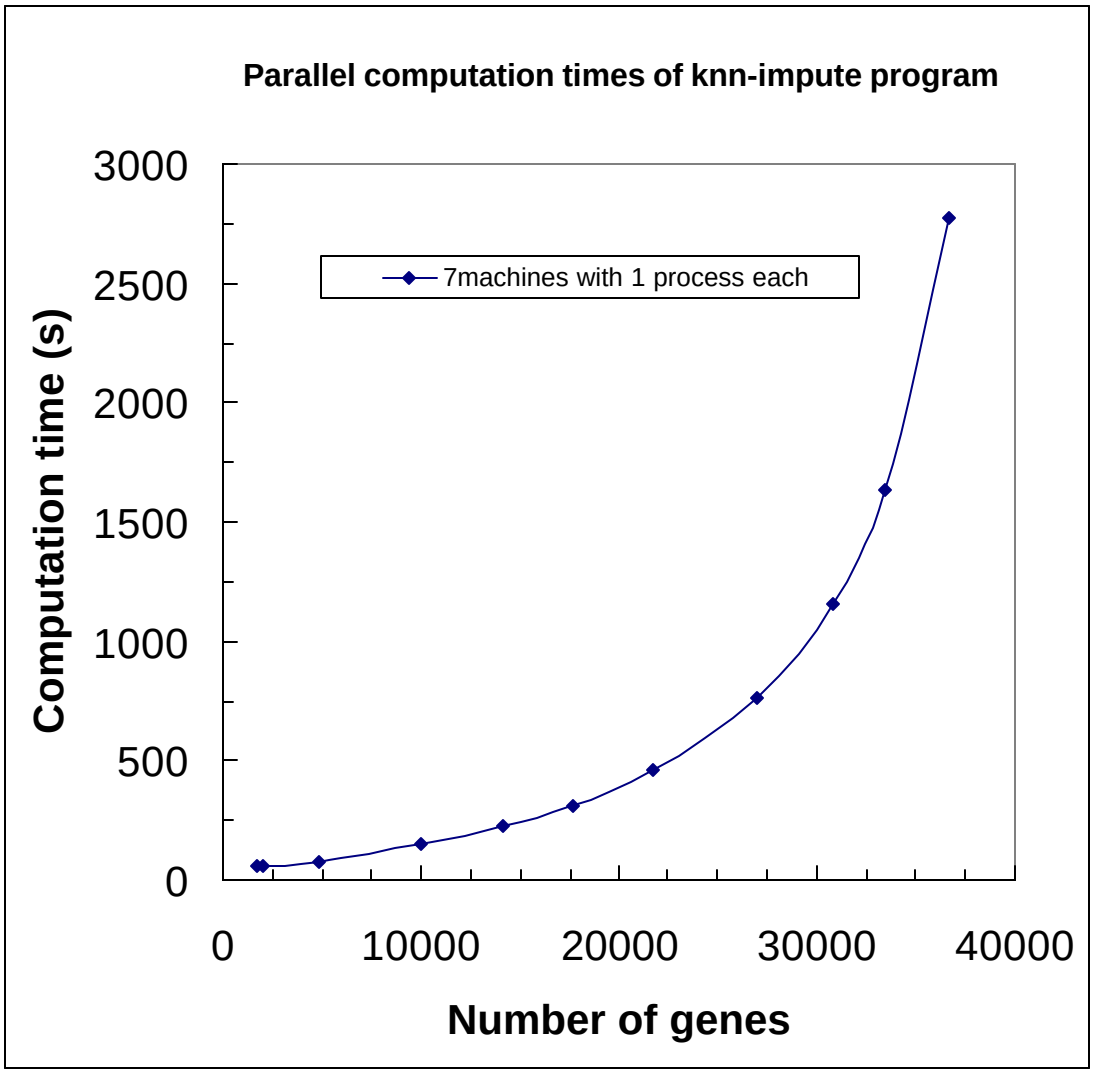

Figure 5.8: Computation time for Knn-Impute parallel implementation

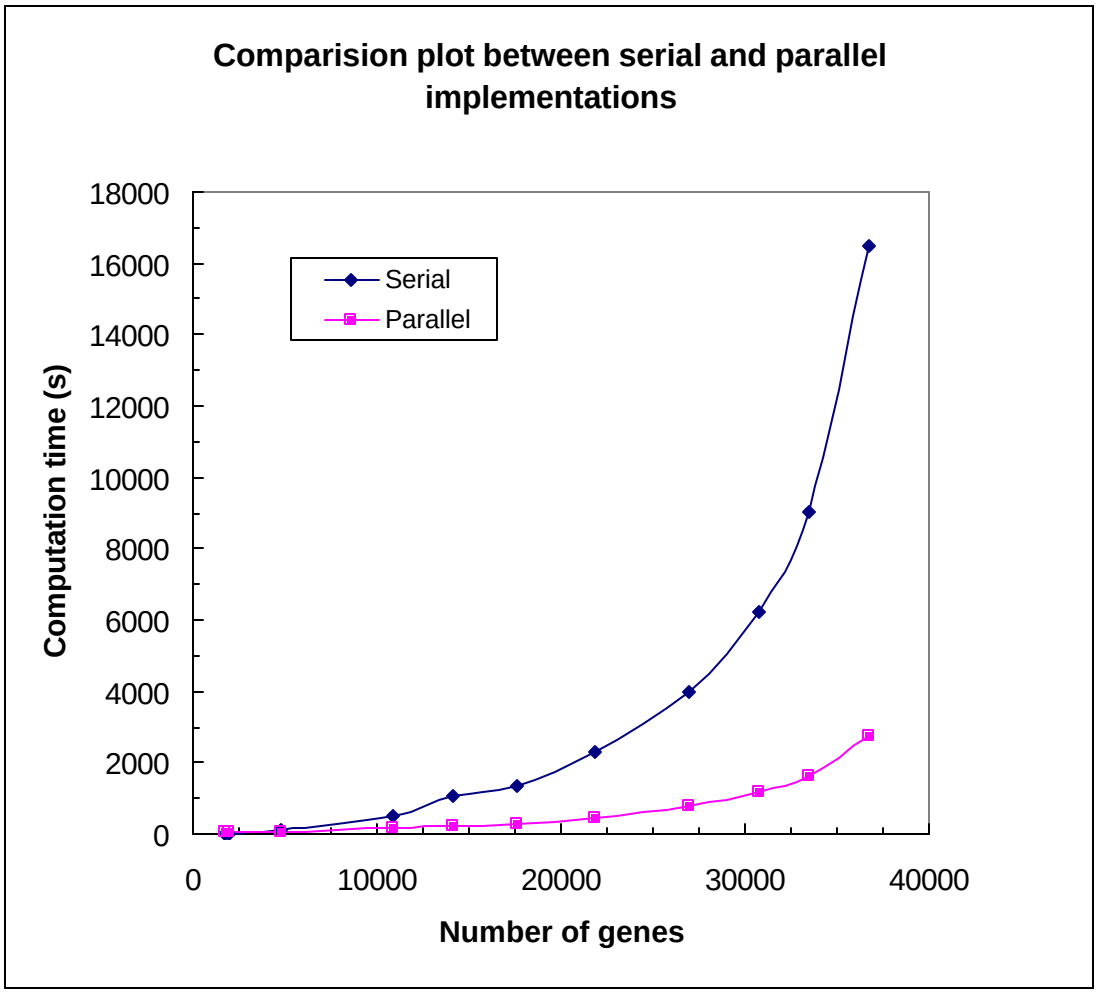

Figure 5.9: The comparison plot between Knn-Impute serial and parallel implementations

To observe the change in the computation times with the change in the number of machines used for parallelization, a set of experiments were conducted. First master with two slaves i.e. a total of three machines were used. Similarly three, four, and six slaves where used for calculations. The table below shows the computation times for different slaves. The computation times are close to serial times with low number of machines. As this program is data parallel, a very good reduction in the computation times is achieved with the increase in the machines, which can be seen from the table 5.5. The graph which shows these computation times is shown in the figure 5.10. The speed ups achieved by increasing the number of machines is discussed in the speedups section. The figure 5.11 shows the comparison plot between the serial and parallel implementations.

Table 5.5: The parallel implementation of Knn-Impute on Linux cluster with different number of slaves

29 exp.		**Machines**	**Computation**	**Time**
Genes count	2-slaves	3-slaves	4-slaves	6-slaves
1700	24	31	39	55
2000	27	32	40	56
4800	81	73	70	77
10000	286	209	176	154
14100	500	355	287	227
17600	715	501	399	308
21800	1209	808	629	460
27000	2053	1386	1069	759
30800	3211	2148	1694	1154
33500	4523	3045	2323	1635
36700	7934	5387	4071	2773

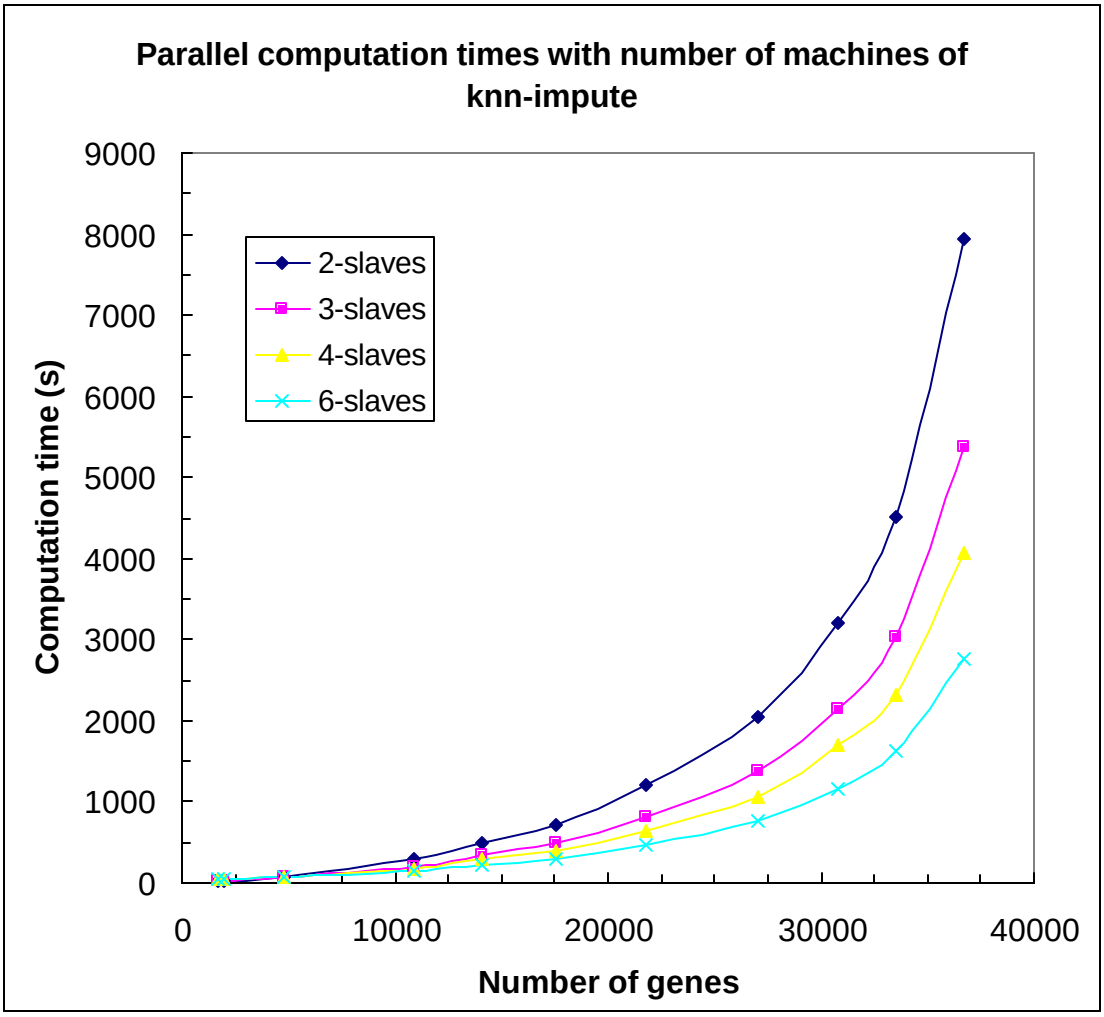

Figure 5.10: Computation time for Knn-Impute parallel implementation with number of machines

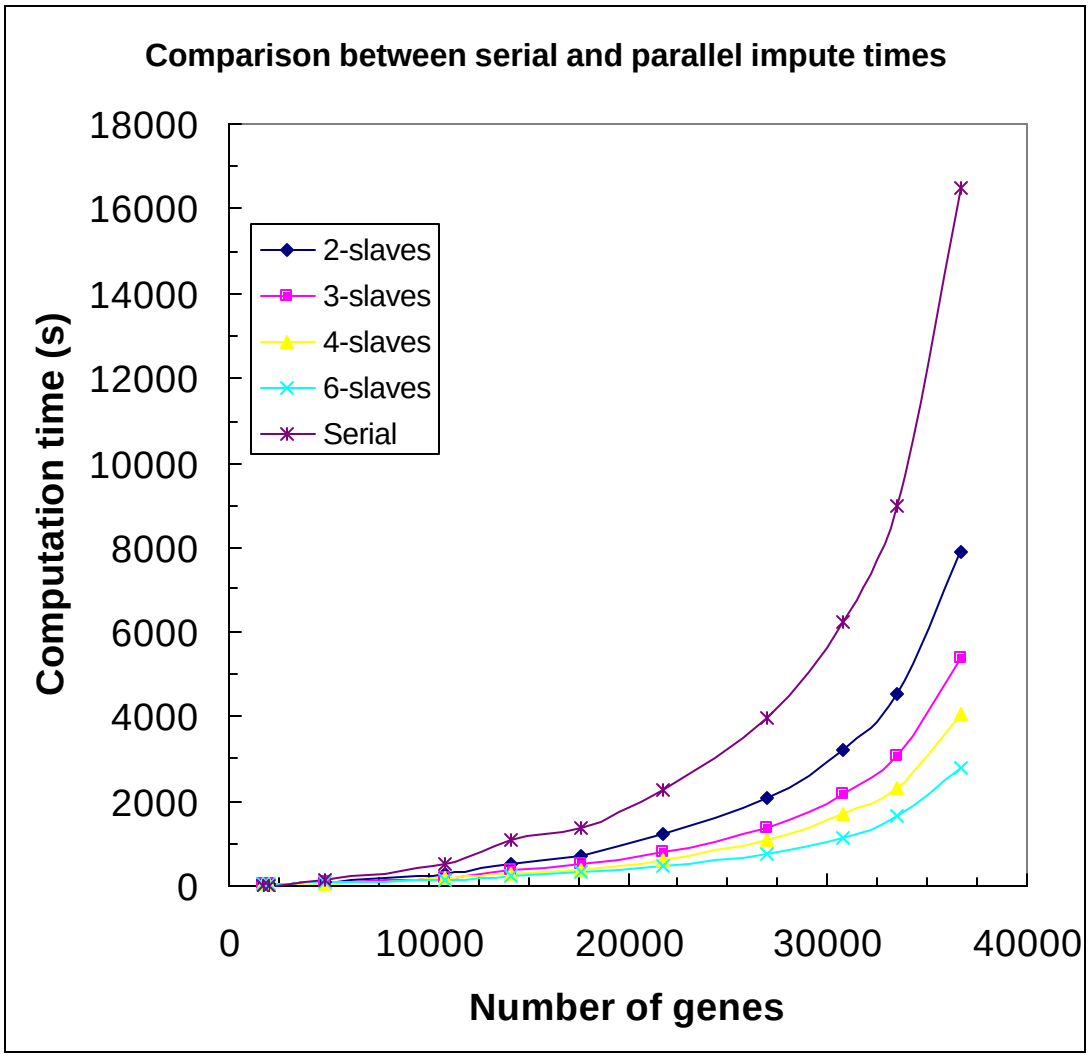

Figure 5.11: Comparison plot between Knn-Impute parallel and serial implementations

5.5 Hierarchical Clustering Computation of Parallel Implementation

The VLSI cluster is used for hierarchical clustering totally 7 machines where used for parallel program. The calculations where taken during the night when the load on the cluster is low. Still the computation time didn't decrease much because of huge data dependencies. The table 5.6 shows the computation times generated by using 7 machines. The figure 5.12 shows the plot of the parallel computation times.

The figure 5.12 shows the growth of the computation time how it increases. The computation time was calculated using different number of machines as slaves as it was described in the above section. The total number of machines used where 3, 4, 5 and 7 respectively and the parallel computation times where calculated. The table 5.7 shows the computation times obtained with number of machines. The experiments shows interesting results which are described in the following section.The growth in the computation time with the parallel implementation in not squared but less than the squared.

Figure 5.13 shows the comparison plot between the serial and parallel implementations of the hierarchical clustering program. From the figure 5.13 the comparison plot it is clear a speedup of close to 2 is achieved. Not much speedup was achieved because of the data dependencies.

Figure 5.14 shows the hierarchical clustering program parallel implementation times with different number of machines, and the figure 5.15 shows the comparison plot between the serial and parallel implementations with number of machines.

.

Table 5.6: The parallel implementation of hierarchical clustering on VLSI cluster for 29 experiments and 38000 gene set

Gene Count	**Comp. time(s) with 7 computers with 1 process each**
1700	13
2000	17
4800	59
10000	293
14100	505
17600	800
21800	1273
27000	1963
30800	2563
33500	3059
36700	3681

Table 5.7: The parallel implementation of hierarchical clustering program different number of slaves

29 exp.		**Machines**	**Computation**	**Time**
Genes count	2-slaves	3-slaves	4-slaves	6-slaves
1700	19	16	15	13
2000	24	21	19	17
4800	126	96	75	59
10000	661	504	401	293
14100	1131	863	687	505
17600	1800	1374	1089	800
21800	2864	2187	1725	1273
27000	4417	3365	2650	1963
30800	5470	4389	3437	2563
33500	6376	5237	4068	3059
36700	7493	6309	4780	3681

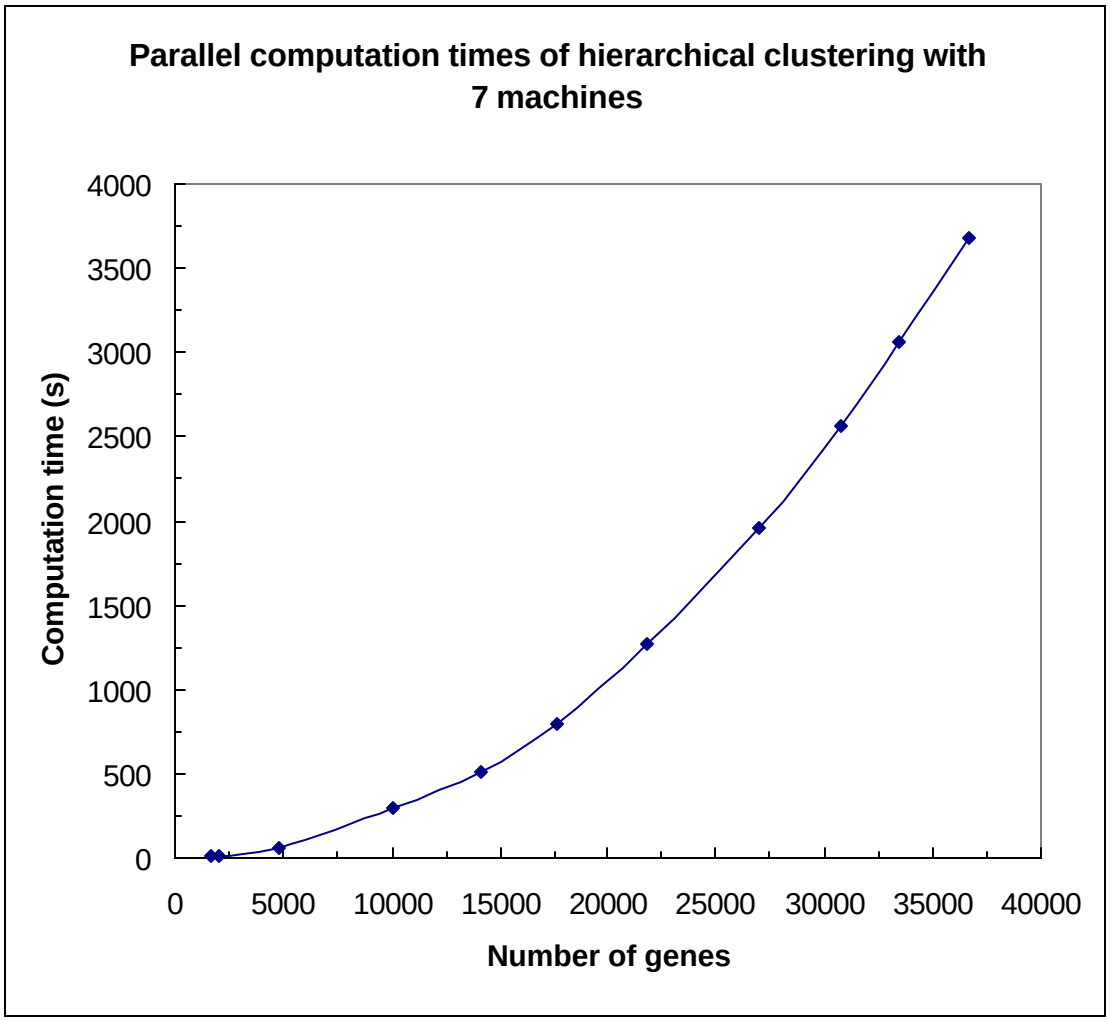

Figure 5.12: Computation time for hierarchical clustering Parallel implementation

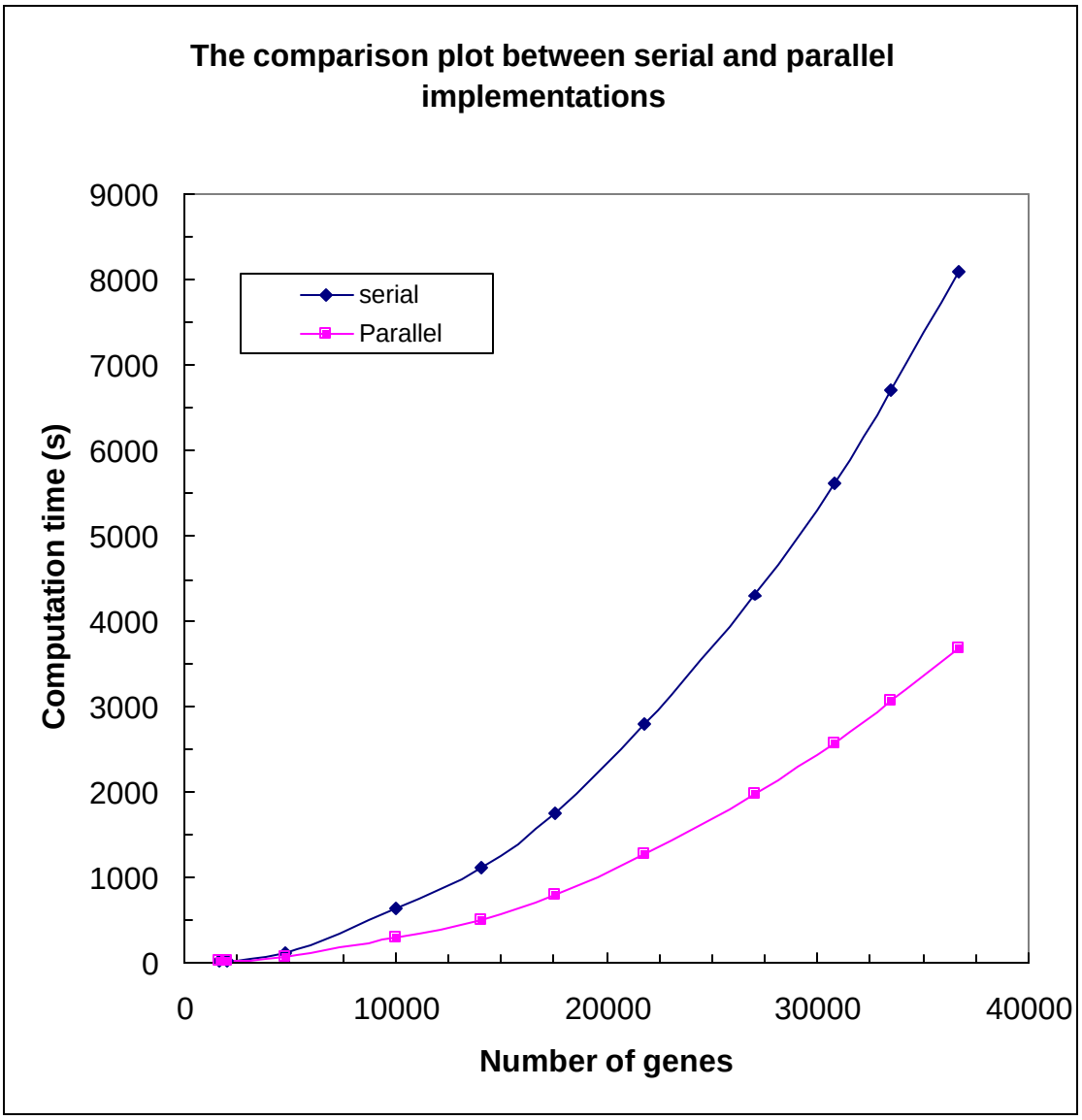

Figure 5.13: The comparison plot between the hierarchical clustering serial and parallel implementations

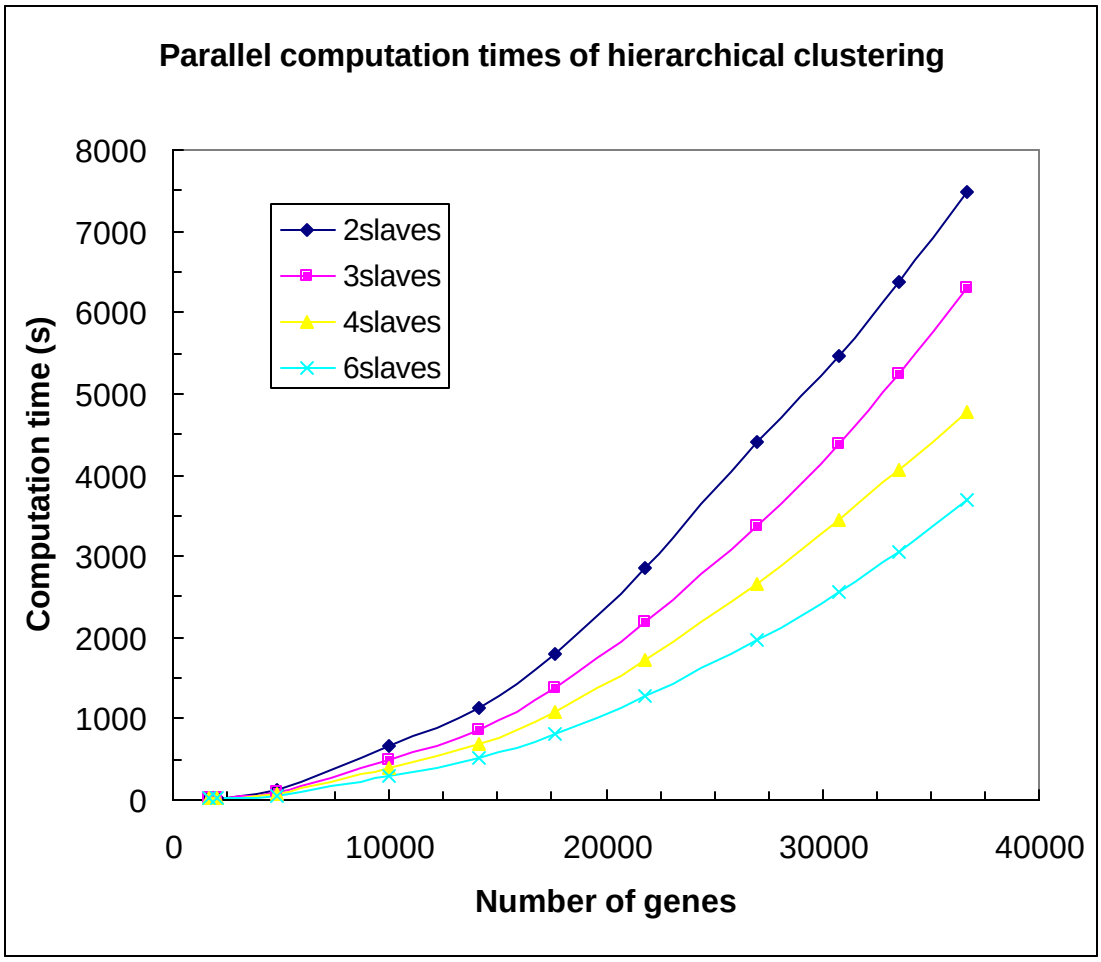

Figure 5.14: Computation time for hierarchical clustering parallel implementation with number of machines

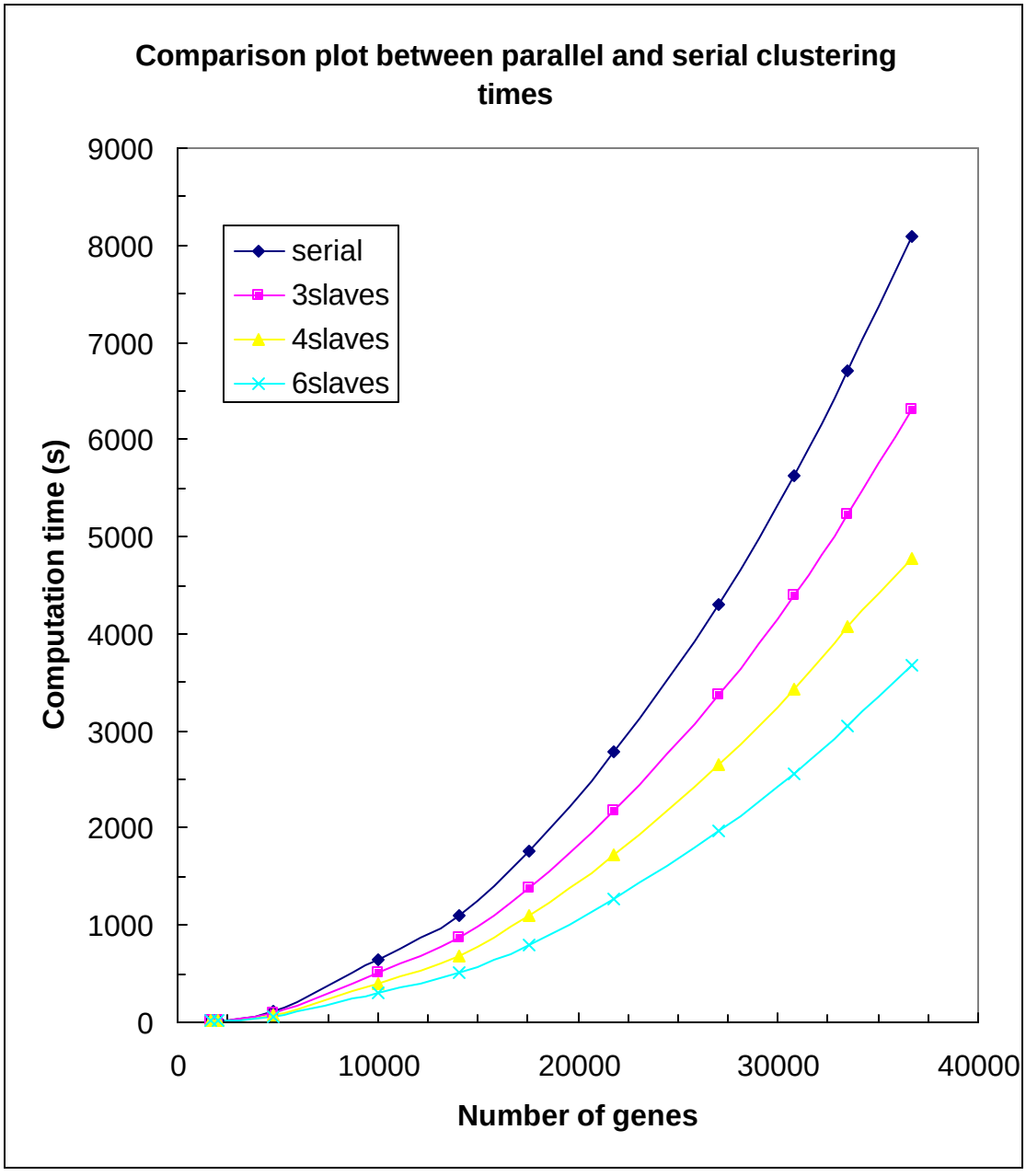

Figure 5.15: Comparison plot between hierarchical clustering parallel and serial implementations with number of machines

5.6 End to End Computation of Parallel Implementation

The table 5.8 shows the computation times achieved with parallel implementation with different number of machines. The more the number of machines the lesser is the computation time which can be seen clearly from the table 5.8. This implementation has a maximum of 7 machines and the clusters which are used for the parallelization are not dedicated to SMD this implies that the machines in these clusters perform application other than SMD which impacts the computation times. Figure 5.16 shows the entire applications parallel implementation times with different number of machines. From the figure 5.16 one can depict that more the number of machines lesser the computation time. Hence with more number of machines and dedicated clusters for the application a much higher speedups can be achieved. From the figure 5.16 we can depict that a significant achievement is accomplished. There is a noticeable reduction in the computation times. The speedups achieved by parallelization is discussed in the speed ups section.

Table 5.8: The parallel implementation of entire application with different number of slaves

29 exp.		**Machines**	**Computation**	**Time**
Genes count	2-slaves	3-slaves	4-slaves	6-slaves
1700	98	102	109	123
2000	120	122	128	142
4800	379	341	317	308
10000	1310	1076	940	810
14100	2168	1755	1511	1269
17600	3177	2537	2150	1770
21800	4917	3839	3198	2577
27000	7494	5775	4743	3746
30800	9879	7735	6329	4915
33500	12197	9580	7689	5992
36700	16862	13131	10286	7889

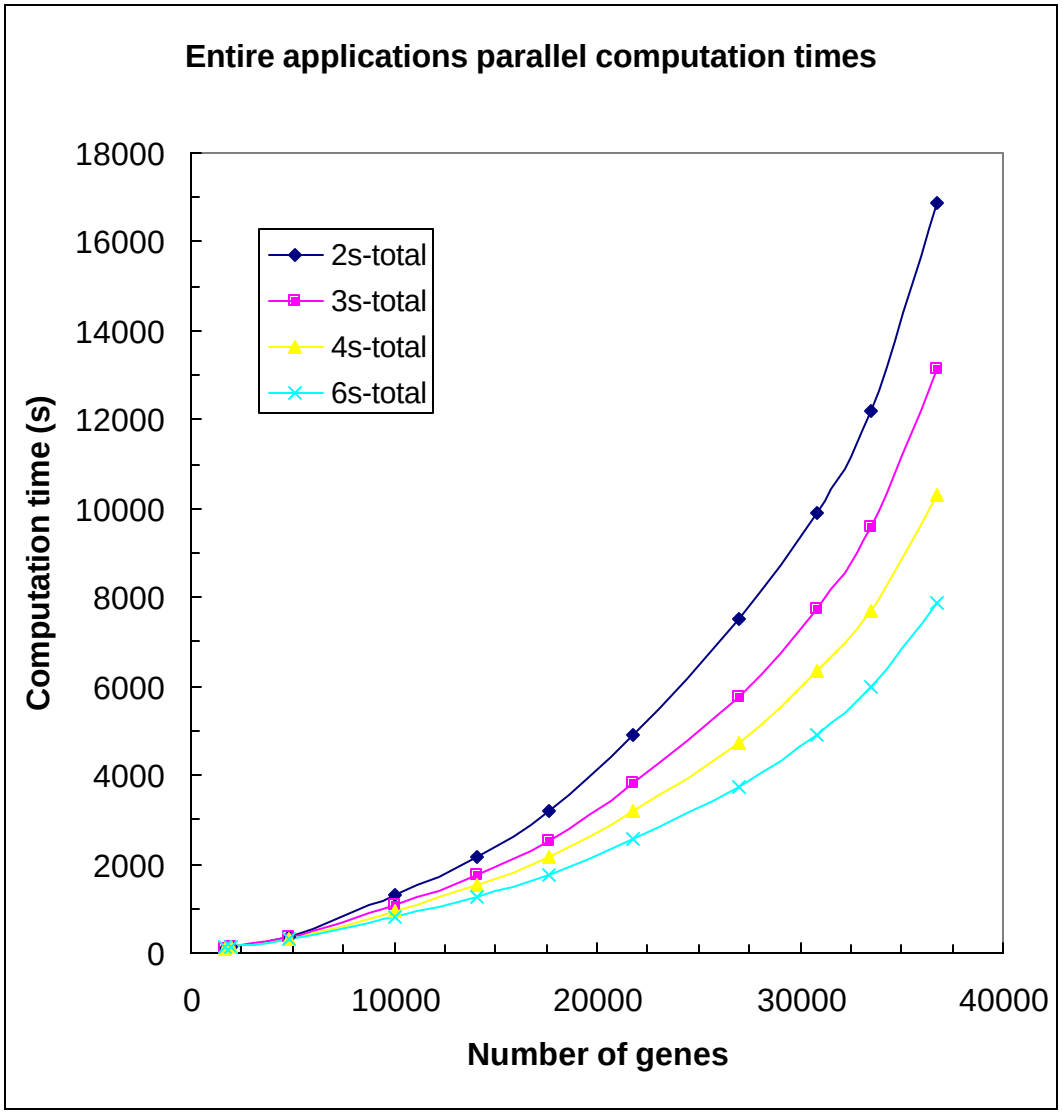

Figure 5.16: Computation time for entire applications parallel implementation with number of machines

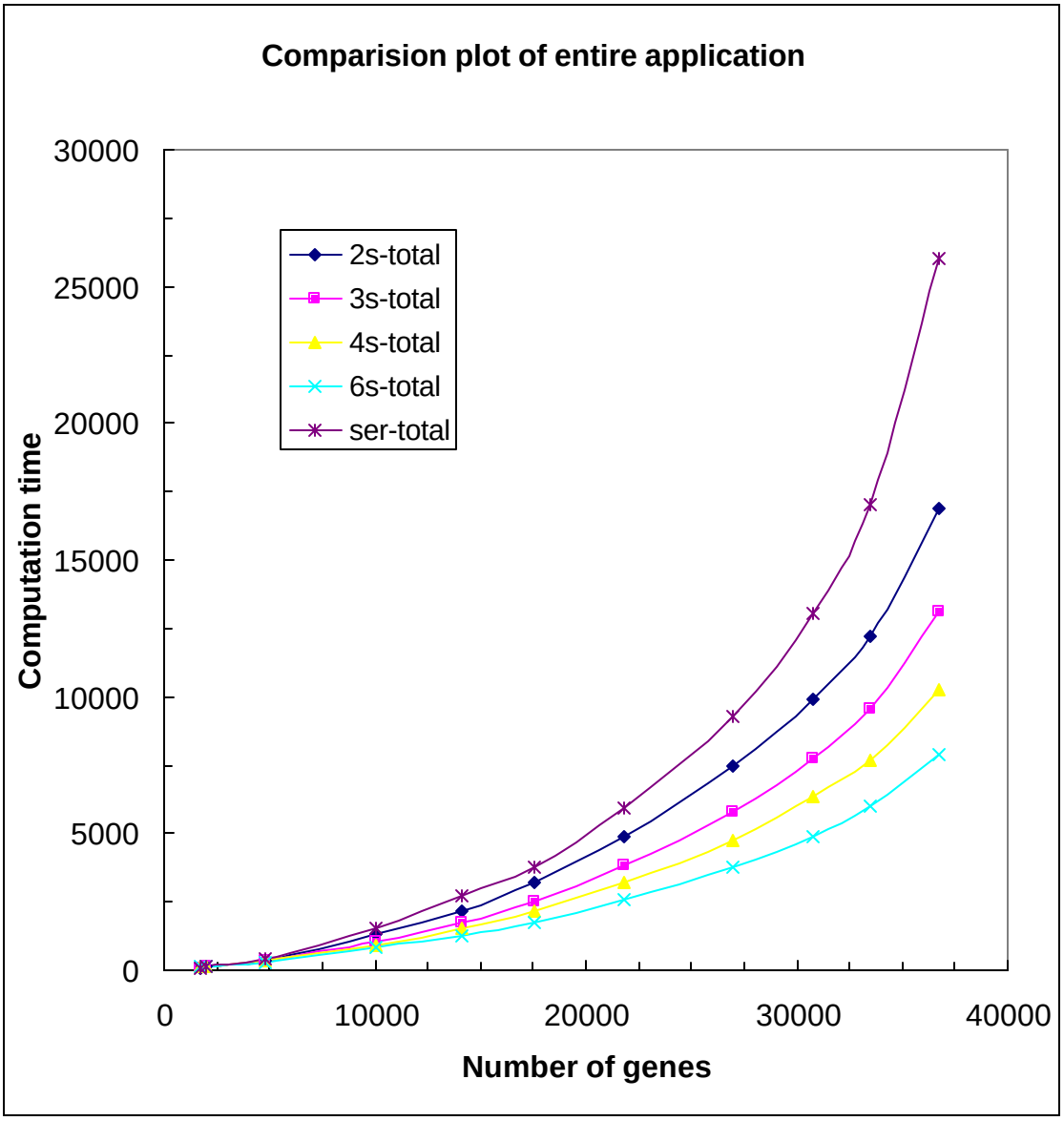

Figure 5.17: Comparison plot of entire applications parallel and serial implementations with number of machines

5.7 Speedups

The table 5.9, 5.10 shows the speedups achieved by parallelization of Knn-impute program, and hierarchical clustering program. The table 5.11 shows the speedups achieved by the entire application. From the tables the best speedup achieved by parallel implementation of Knn-impute program is close to 6 approximately 5.947. This is because there is no dependant data so the entire application can be divided between the slave doing the same functionality, thus achieving the data parallelism. On the other hand the best speedups achieved by parallelizing the hierarchical clustering program is around 2. This is because of lots of dependant data as showed in the chapter 5 in ClusterNode section. Due to this dependant data a venation parallelism approach is taken. In the entire application maximum amount of time is consumed by the combination of Knn-impute and hierarchical clustering programs. Thus a significant speedup is achieved by the entire application which is around 3. The figures 5.18, 5.19, and 5.20 shows the speedup curves. The figures show the speedups achieved with number of machines.

Table 5.9: The speedups achieved by Knn-impute program

29 exp.	**Machines speedups**			
Genes count	2-slaves	3-slaves	4-slaves	6-slaves
1700	0.541	0.419	0.333	0.236
2000	0.629	0.531	0.425	0.303
4800	1.555	1.726	1.8	1.636
10000	1.825	2.497	2.965	3.389
14100	2.148	3.025	3.742	4.731
17600	1.906	2.72	3.416	4.425
21800	1.889	2.827	3.632	4.967
27000	1.936	2.868	3.719	5.238
30800	1.935	2.892	3.668	5.384
33500	1.992	2.959	3.879	5.511
36700	2.078	3.061	4.051	5.947

Table 5.10: The speedups achieved by hierarchical clustering program

29 exp.	**Machines speedups**			
Genes count	2-slaves	3-slaves	4-slaves	6-slaves
1700	0.842	1	1.066	1.23
2000	0.875	1	1.105	1.235
4800	0.959	1.256	1.593	2.04
10000	0.969	1.270	1.596	2.18
14100	0.973	1.275	1.603	2.181
17600	0.974	1.275	1.609	2.19
21800	0.974	1.276	1.617	2.19
27000	0.974	1.279	1.625	2.192
30800	1.027	1.28	1.637	2.193
33500	1.052	1.281	1.649	2.194
36700	1.080	1.283	1.693	2.2

Table 5.11: The speedups achieved by entire application

29 exp.	**Machines speedups**			
Genes count	2-slaves	3-slaves	4-slaves	6-slaves
1700	0.857	0.823	0.77	0.682
2000	0.891	0.877	0.835	0.753
4800	1.105	1.227	1.317	1.360
10000	1.164	1.417	1.622	1.881
14100	1.25	1.545	1.794	2.137
17600	1.176	1.473	1.738	2.112
21800	1.203	1.541	1.851	2.296
27000	1.241	1.611	1.961	2.483
30800	1.319	1.684	2.059	2.651
33500	1.395	1.776	2.213	2.84
36700	1.544	1.983	2.531	3.301

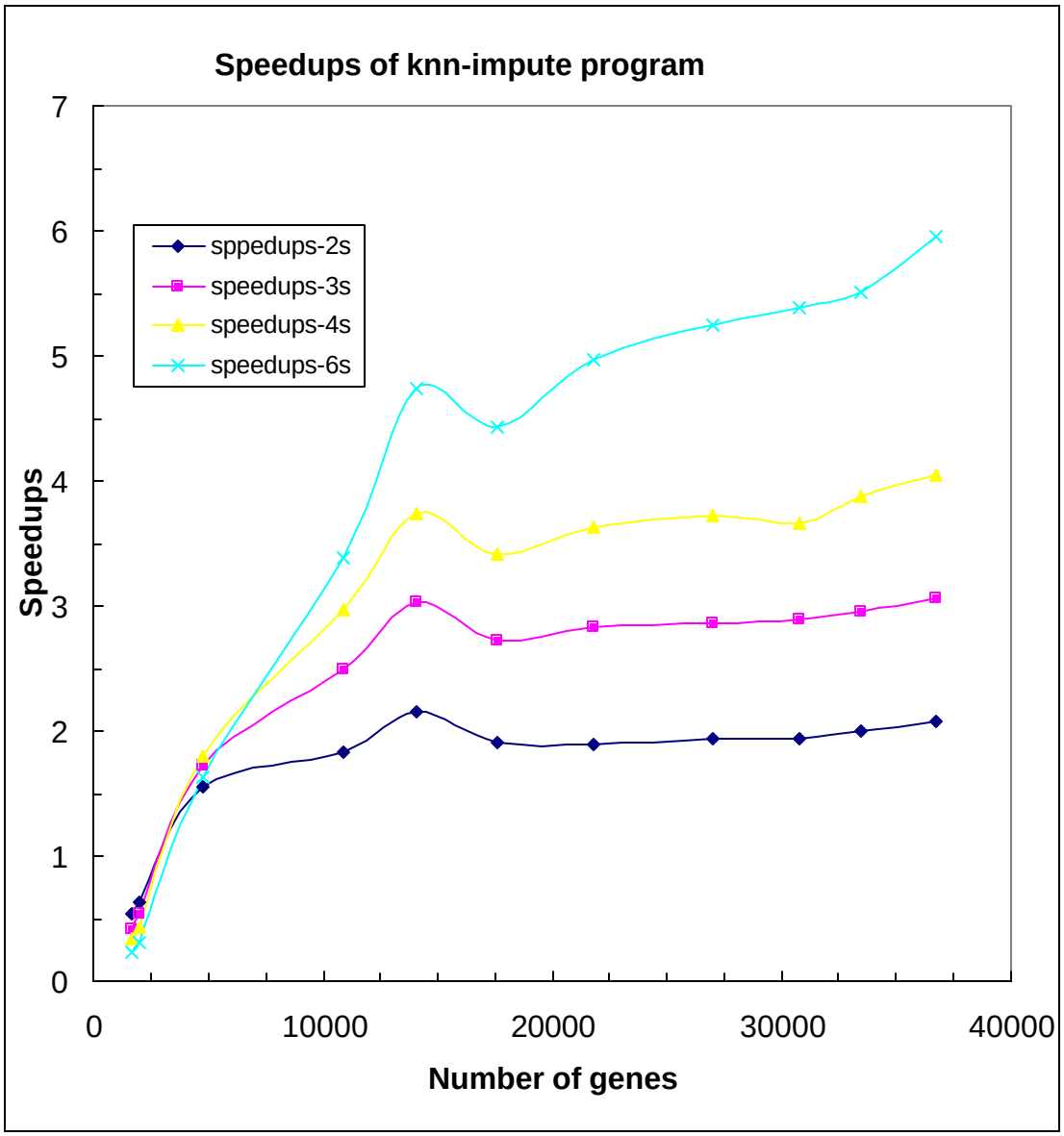

Figure 5.18: The speedups curves of Knn-impute program

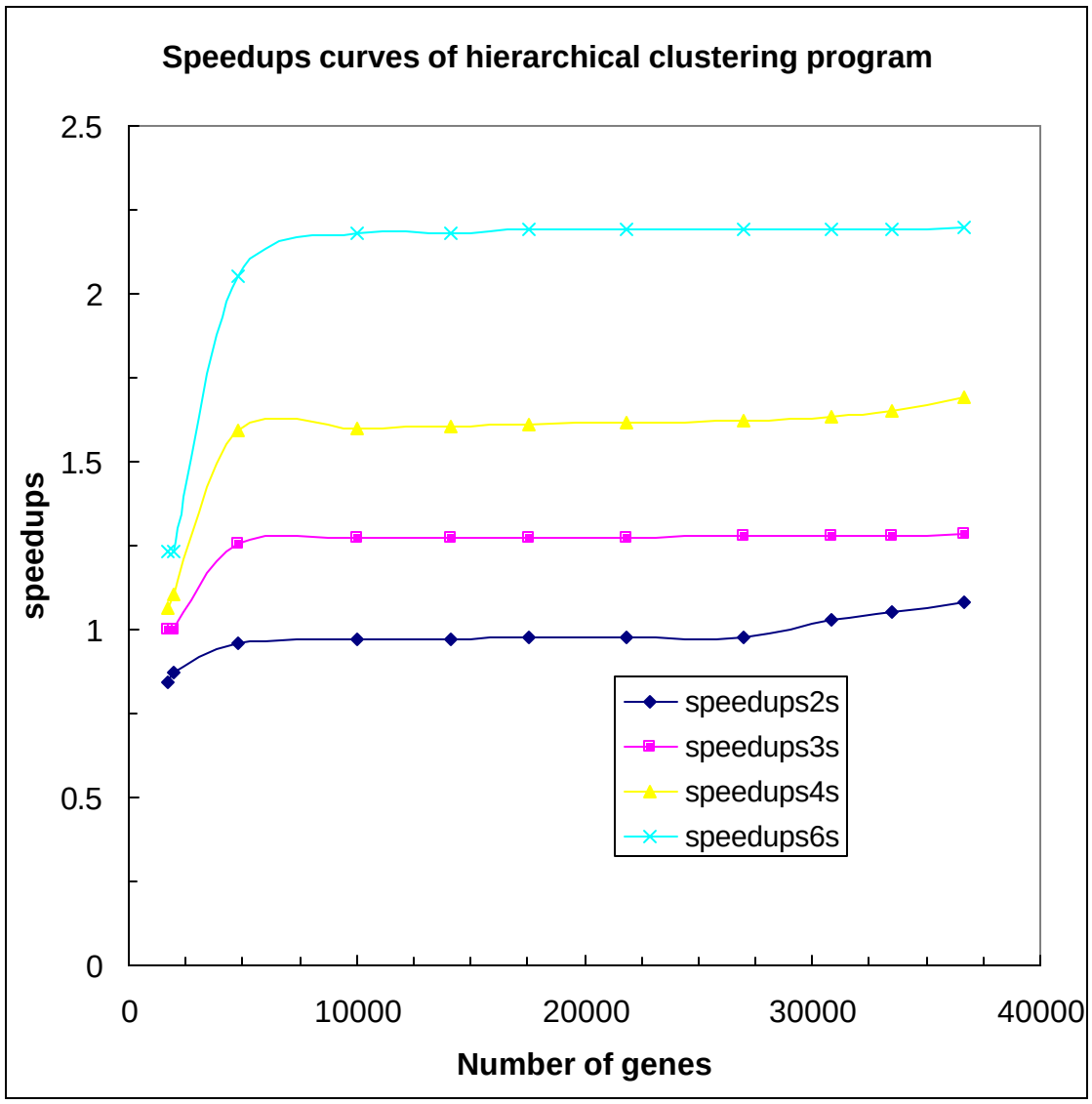

Figure 5.19: The speedup curves of hierarchical clustering program

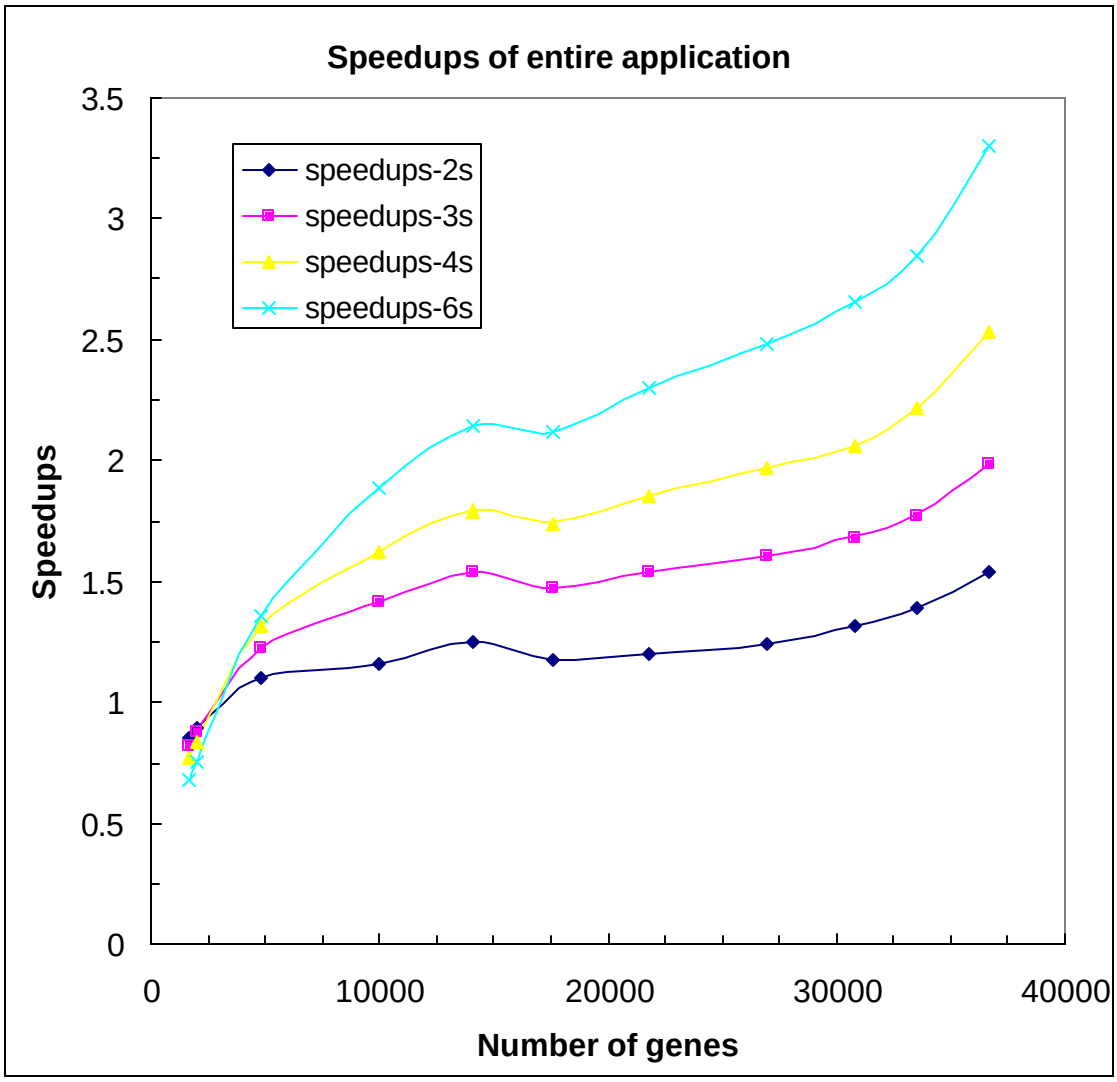

Figure 5.20: The speedups curves of entire application

The bump in the speedups seen in the Knn-impute implementation is because the percentage of missing values is less in the files which are on the down slope. So the computation time for the serial implementation is comparatively less than the neighboring files thus showing a fall in the speedups thus clarifying the bump in the speedup curves. The same trend is carried to the entire applications speedup curves from the Knn-imputes.

This shows that the impact of Knn-impute term is the highest on the whole application. The proof of this is discussed in the serial implementation section 5.3.

Chapter 6

Conclusions and Future Improvements

This thesis explains the Stanford Microarray Database application and the analysis part of the application and addresses the problem of long computation time taken by the analysis programs of SMD. The solution to the problem is parallel computing, which the thesis discusses. This chapter summarizes and compares the results of parallelizing the analysis programs of the SMD, and discusses ideas for future improvements in enhancing the processing speed of the whole application.

6.1 Conclusions

This thesis attempts to provide a solution for processing the large data generated by the genetics applications. The uniqueness of this thesis is the computation speeds achieved by using the freely available computing systems in the University of Tennessee, using parallel processing. The analysis part of the SMD application is picked because it needs lots of computation time for processing. The Knn-impute and Hierarchical clustering programs where chosen because of their popularity and wide use. The impute algorithms are used not only in this application but also in many image processing and some of the traffic estimation problems. Hierarchical clustering is used in many applications to generate dendograms for similar objects and groups etc.

Chapter 3 discusses the serial implementation of the SMD application procedure and the analysis part of the application. As the number of genes increases there is almost a

quadratic growth in the computation time of the Knn-impute and squared growth of computation time for hierarchical clustering program. There is a linear growth in the time taken to generate the files for analysis from the database and the image generation. Thus the time taken for end to end process of the application has quadratic growth. These growths are proved using the computational complexities of each part of the entire application.

Chapter 4 discusses the parallel implementation of the analysis programs, Knn-impute and the hierarchical clustering. Two types of parallelism are used one is data parallelism which is used in Knn-impute program and the other is functional parallelism which is partly used, along with data parallelism in clustering program. The results achieved with parallelism are good. For the Knn-impute with just 7 machines a computation speed-up of 6 is achieved, this is because of the data is parallelization and non dependent. On the other hand the computational speed-ups achieved with hierarchical clustering is close to 2. This is because of data dependencies, as the node generating the data, depends on the data of the previous node thus causing difficulties in parallelizing. Functional parallelism is used in this program.

The speed-ups achieved by combining both Knn-impute results and the hierarchical clustering results is close to 4. This shows the success of the thesis in achieving the speed-ups of the application. Further the end to end speed-ups achieved is around 3 which can save lot of time for the researchers analyzing their data. The future improvements will attain speed-ups of many folds with the various implementations discussed in the next section.

6.2 Future Improvements

The fast growing requirements of not only the genetics applications but also engineering, commercial and business applications, and lower time to market lead to the development of many techniques in computing. Computing based applications such as genetic analysis, simulation and modelling, weather forecasting, semiconductor modeling, drug design and medical applications require high processing speed and large amount of memory. Parallel computing technique which is used in this thesis is one of the essential and important technique in solving these complex problems. A prevalent problem in parallel computing is the ever growing technological gap between processors and communication links. The main terms that determine the performance of the network are the bandwidth and the latency of the communication link. Future work will address the issues of bandwidth and latency, and come up with improved solutions to speedup the application further.

This thesis deals with the serial load balancing by the master processor. Future work could address the dynamic load balancing of the problem. The end to end process of the SMD application can be further improved by parallelizing the image generation, will be added later.

Traditional computing is divided into general purpose computing and application specific computing. Microprocessors are used for general purpose computing and perform a wide range of applications. However, microprocessors are not efficient for data processing applications. Because the architecture of the microprocessor is fixed the software program generally determines the computation time. Application specific computing, on

the other hand, uses Application Specific Integrated Circuits (ASICs) to perform operations in hardware. ASICs are very fast and efficient in performing a specific operation for which they are designed. However, the limited flexibility and time-design cost are major disadvantages of ASICs. The flexibility and the speed of ASICs can be achieved with Reconfigurable Computing (RC). The main component of RC systems is Field-Programmable Gate Arrays (FPGAs). The recent advances in the design of the FPGAs lead to a drastic improvement in the RC systems. The SMD application and analysis programs could be deployed on a RC architecture which would result in a faster computational time.

References

1. George Karypis, University of Minnesota-Twin cities, “Data Mining for Genomics”.

2. Gavin Sherlock, Tina Hernandez-boussard, Andrew Kasarskis, Gail Binkley, John C. Matese, Selina S. Dwight, Miroslava Kaloper, Shuai Weng, Heng Jin, Catherine A. Ball, Michael B Eisen, Paul T. Spellman, Patrick O. Brown, David Botstein and J. Michael Cherry, “ The Stanford Microarray Database”, Nucleic Acids Research, 2001,vol. 29, No.1.

3. Jeremy Gollub, Catherine A. Ball, Gail Binkley, Janos Demeter, David B. Finkelstein, Joan M. Hebert, Tina Hernadez-Boussard, Heng Jin, Miroslava kaloper, John C. Matese, Mark Schroeder, Patrick O. Brown, David Botstein and Gavin Sherlock, “ The Stanford Microarray Database: data access and quality assesment tools”, Nucleic Acids Research, 2003, Vol.31, No. 1 94-96.

4. Ying Zhao and George Karypis“Evaluation of Hierarchical clustering Algorithms for Document Datasets”, Technigal Report #02-022.

5. Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein and Russ B. Altman, “Missing value estimatiom methods for DNA microarrays”, Bioinformatics vol 17 no. 6 2001 pages 520-525.

6. Anne Mueller, Jani O’Rourke, Jan Grimm, Karen Guillemin, Michael F. Dixon, Adrian Lee, and Stanley Falkow,”Distinct gene expression profiles characterize the histopathological stages of disease in helicobacter-induced mucosa-associated lymphoid tissue lymphoma”, PNAS, Feb. 2 2003, vol. 100 no. 3 1292-1297.

7. Upper Midwest Regional Earth Science Application Center, “A Whitepaper”, http:// resac.gis.umn.edu/news/resac_whitepaper.pdf pages 12-14.

8. John Rice, Erik Van Zwet, “A Simple and Effective Method for Predicticting Travel Times on Freeways”.

9. David Hand, “Principles of Data Mining”.

10. Chin-Hsiung Wu, Shi-Jinn Horng, Horng-ren Tsai, “Efficient Parallel Algorithm for Hierarchical Clustering on Arrays with Reconfigurable Optical Buses”, Journal of Parallel and Distributed Computing 60, 1137-1153 (2000).

11. X. Li and Z. Fang, “Parallel Clustering Algorithms”, Parallel Computing 11 (1989), 275-290.

12. X. Li, “Parallel Algorithms for Hierarchical clustering and cluster validity”, IEEE Trans. Pattern Anal. Mach. Intelligence 12 (1990), 391-403.

13. Olson C.F., “Parallel algorithms for Hierarchical Clustering”, Parallel Computing,21(8):1313-1325.

14. H. R. Tsai, S.J. Horng, S.S. Lee, S.S. Tsai, and T.W. Kao, “parallel hierarchical clustering algorithms on processor arrays with a reconfigurable bus system”, Pattern recognition 30 (1997), 801-815.

15. Xiaowei Xu, Jochen Jager, Hans-Peter Kriegel, “A Fast Parallel Clustering Algorithm for Large Spatial Databases”, Data Mining and Knowledge Discovery, 3, 263-290 (1999).

16. tMichael Nilges and Jens P. Linge, “Bioinformatics - a definition”, Unite de Bio-Informatique Structurale, Institut Pasteur, 25-28 rue du Doctor Roux, F-75015 Paris, France.

17. rMasaru Tomita, Editorial (Towards Computer-Aided Design of Useful Microorganisms), Bioinformatics Vol. 17 no. 12 2001, Page 1091-1092.

18. Ashok Palaniappan, “Deep Meaning in Biocomputing”, Resonance, July 2002.

19. Edward N. Trifonov, “Earliest Pages of Bioinformatics”, Bioinformatics Vol. 16 no. 1 2000, Pages 5 - 9.

20. Richard J. Roberts, “The Early Days of Bioinformatics Publishing”, Bioinformatics Vol. 16 no. 1 2000, Pages 2 - 4.

21. Kei-Hoi Cheung et al, “Graphically-Enabled Integration of Bioinformatics Tools Allowing Parallel Execution”, Center for Medical Informatics, Yale University, New Haven, CT.

22. http://genome-www.stanford.edu/microarray

23. www.affymetrix.com

24. www.agilent.com

25. www.bisti.nih.gov

26. Korn et al., 1977; McCallum and Smith, 1977.

27. Fuchs et al., 1978; Gingeras et al., 1978.

28. Stefik, 1978.

29. Benson et al., 1999.

30. Slide from presentation of Sumitra Urs, Nutrition department, UT.

31. World wide web.

Vita

Bhanu Prasad Rekapalli was born in Hyderabad, a city of great historic importance in Andrapradesh, India in 1978, son of Lakshmi Sulochana, and Subba Rao Rekapalli. He attended Siva Sivani Public school which has high standards of education which laid a strong foundation for the higher studies. After performing brilliantly in a highly competitive exam, toughest of its kind in the country, he joined Jawaharlal Nehru Technological University in electrical and electronics engineering. During his undergraduate education, he got good foundation in mathematics and basic sciences which broadened his horizon of knowledge. He was an executive member of the EE department which enhanced his leadership skills. He did his practical training during the final year of undergraduate program in VLSI division of Electronics Corporation India Limited, and the research project helped him to delve deeper into challenging fields of microelectronics systems and VLSI design. He received a B.Tech. degree in Electrical and Electronics Engineering, in June 1999 from Jawaharlal Nehru Technological University.

He came to US for graduate for his graduate education, he joined University of Tennessee, Knoxville. During his stay in UT he gained experience in different kinds of jobs. He tutored mathematics to sophomores and juniors and he was teaching assistant in EE department. The teaching experience endowed him to gain proficiency in teaching. After that he joined Webservices group, UT as a Graduate assistant, which helped him to find the idea for his thesis research. His job at Webservices with John, Colton and Bob helped him a lot with his thesis. At the same time he enhanced his computer skills in database management, website applications design and development. After receiving

the M.S. degree in Electrical Engineering from UT, he enrolled in to Ph.D. at UT. His current research interest are in the fields of bioinformatics, High performance computing and Architecture design. His hobbies and interests include working out in gym, water sports, travelling around world and dancing.