LOAD SCHEDULING FOR BIOINFORMATICS APPLICATIONS IN LARGE SCALE NETWORKS

By

SUDHA GUNTURU

Bachelor of Technology in Computer Science

Jawaharlal Nehru Technological University

Hyderabad, Andhra Pradesh

2005

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2008

LOAD SCHEDULING FOR BIOINFORMATICS

APPLICATIONS IN LARGE SCALE

Thesis Approved:

ACKNOWLEDGMENTS

I would like to extend my deep gratitude and appreciation to my major advisor Dr. Xiaolin (Andy) Li for his support and guidance provided throughout this study. Without his help this study would not have been possible. I would like to acknowledge the help and support of Dr. Bharadwaj Veeravalli for his guidance provided throughout this study.

Likewise the support of committee members Dr. Nophill Park and Dr. George is greatly appreciated. A special note of thanks is due to the faculty members and secretarial staff of the Department of Computer Science at Oklahoma State University for their support during my time at Oklahoma State University.

Finally, I would like to express my deepest and sincerest thanks to my parents Lakshmi Gunturu and Rama Krishna Gunturu, Husband Srinivas Singampalli, Sister Sandhya Sripada, Brother-in-law Ravi Sripada family members and friends (Narsimham Singampalli, Gouri Singampalli, Srinivas Indla, Krupa Singampalli, Sailaja Sabbella, Pranav Peddi, Venkatesh Akella, Ravi Nidadavolu, Veenita Grover, Archana Patel nephews Srijan V. Sripada and Aditya Indla) for their never-ending encouragement and support, which has enabled the completion of this degree.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

NOMENCLATURE

m	Total Number of processors in the system
P_i	The ith processor in the system
E_i	The total time taken for the processor P_i compute the S and M matrices
Seq X	The first sequence that is given for the sequence alignment
Seq Y	The second sequence that is given for the sequence alignment
C	The time taken for communication
x	Length of sequence x
y	Length of sequence y
Z	Total number of iterations used to compute the S and M matrices
x_i	Total number of residues of sequence x assigned to P_i where $\sum_{i=1}^{m} x_i = x$
y_i	Total number of residues of sequence y assigned to P_i where $\sum_{j=1}^{Z} y_i = y$
$S_{i,j}$	The sub matrix of S matrix
$M_{i,j}$	The sub matrix of M matrix
T(m)	The total processing time for the alignment of the sequences

CHAPTER I

INTRODUCTION

Motivation

The interest in network based computing has grown in recent years. Many applications in scientific and engineering domains are structured as large numbers of independent tasks with low granularity. These applications are thus amenable to straightforward parallelization, typically in master-slave fashion, provided that efficient scheduling strategies are available. Such applications are called divisible load because a scheduler may divide the computation time among worker processes arbitrarily, both in terms of task and task sizes.

Scheduling the tasks of a parallel application on the resources of a distributed computing platform efficiently is critical for achieving high performance.

Over the past few decades research in the field of molecular biology has made advancement that is coupled with advances in genomic technologies. This has led to an explosive growth in the biological information generated, in turn, led to the requirement for computerized databases to store, organize, and index the data and for specialized tools to view and analyze the data.

Research Overview

The load distribution problem in distributed computing networks, consisting of a number of processors interconnected through communication links, has attracted a great deal of attention. Divisible Load Theory (DLT) is a methodology that is involved in the linear and continuous modeling of partition able computation and communication loads for parallel processing. DLT is primarily used for handling large scale processing on network based systems.

The processor partitions the load into many fractions, keeps one of the fractions for itself to process and sends the rest to its neighbors (other nodes in the network) for processing [2]. The important problem here is to decide how to achieve a balance in the load distribution between processors so that the computation is completed in shortest possible time [2]. After partitioning the load into fractions the root node or the first node that has the load with it can divide the load in two ways which can be termed as "With Front End Communication" and "Without Front End Communication". In the first case the root node processes it share of load and then will communicate the rest of the load to its children and in this case the processing time increases. In the second case, the root node communicates to other nodes and computes the load on its node simultaneously and this reduces the computation time to a great extent.

This DLT paradigm has numerous applications such as edge detection in image processing, file compression, joining operations in relational databases, graph coloring and genetic searches [8]. Some more examples of real divisible applications include searching for pattern in text, audio, graphic files, database and measurement processing, data retrieval systems, some linear algebra algorithms, and simulations [4].

The merging of the two rapid advancing technologies of molecular biology and computer science resulted in a new informatics science, namely bioinformatics. Over the past few years, the interest and research in the area of biotechnology has increased drastically. This area of study deals primarily with the methodologies of operating on molecular biological information. The present days of molecular biology is characterized by collection of large volumes of data.

The most common operations on biological data include sequence analysis, protein structures predications, genome sequence alignment, phylogeny tree construction, pathway research and sequence database placement. One of the most basic and important application of bioinformatics task is to find a set of homologies for a given sequence because the sequences are often related to the functions, if they are similar.

The different bioinformatics applications like sequence analysis, protein structures predications, genome sequence alignment, and phylogeny tree construction are distributed in different individual projects and they require high performance computational environments. Biologists use a tool called the BLAST for performing research. This tool is a database search, in other words this is described as a Google for biological sequences. This tool provides a method for searching a nucleotide and protein database. This BLAST is designed in such a way that it can detect local and global alignment.

Sequence Alignment is often used in biological analysis. This sequence alignment between any two newly discovered biological sequences can be aligned with the algorithms present in the literature and the similarity can be determined. This sequence alignment can be useful in understanding the function, structure and origin of the new gene. In sequence alignment two sequences are compared with the residues of one another while taking the positions of the residues into account. Residues in the sequence can be inserted, deleted or substituted to achieve maximum similarity or optimal alignment [8]. For example, GenBank is growing at an exponential rate , with as many as 1.2 million new sequences being appended in the year of 2000 [8] [37]. To meet the growing needs a wide variety of heuristics methods have been proposed for aligning the sequences such as FASTP, FASTA,BLAST, and FLASH [28].

Contributions

In this thesis a multiprocessor strategy is designed that exploits the computational characteristics of the algorithms that are used for biological sequence comparisons proposed in the literature. In designing the strategy the load is partitioned among the processors of the network using the DLT paradigm.

The two commonly used algorithms for sequence alignment are the Needleman-Wunsch Algorithm and Smith-Waterman Algorithm where the former is employed for Global Alignment and the latter is used for Local Alignment. The complexity of the

Needleman- Wunsch Algorithm and Smith-Waterman Algorithm to align sequence of length x is given by $O(x^2)$.

The algorithm that is employed in this study is the Needleman-Wunsch Algorithm. Communication delays are considered to be an important part of the processing time in all distributed algorithms while generating the matrix M. The way that has been adopted in this study to for parallelizing the Needleman-Wunsch Algorithm is by computing the matrix elements in diagonal fashion by using a Multiple Instruction Multiple Data Systems.

Divisible Load Theory is employed for handling the sequence alignment. The objective is to minimize the total processing time for sequence alignment. The partition of the load depends primarily on the matrix that is generated by the Needleman-Wunsch Algorithm. The network has been studied for variable link speed and constant link speed.

Outline of the Thesis

The remainder of this document contains chapters as follows. Chapter 2 contains the Literature Review related work pertinent to the problem definition. The Methodology is discussed in Chapter 3. Chapter 4 discusses the Results. Chapter 5 discusses the Conclusion and Future Work.

CHAPTER II

REVIEW OF LITERATURE

Divisible Load Theory

The origin of this Divisible Load Theory came into existence by having the desire to create intelligent sensor networks [2]. But the most recent applications include parallel and distributed computing. Divisible loads can be defined as computations which can be divided into small parts of different sizes and these can be processed independently in parallel. The paradigm of load distribution is basically concerned with a single large load which originates or arrives at one of the nodes in the network. Divisible loads can be classified into two basic categories, namely 1. Modularly Divisible Loads 2. Arbitrarily Divisible Loads. These can be defined as follows

Modularly Divisible Loads: These loads are divided into predetermined modules and are represented in the form of task graph with precedence relations [2].

Arbitrarily Divisible Loads: These loads can be arbitrarily divided into many parts and can be assigned to different processors [2].

The load is massive and requires an enormous amount of time to process, given the computing capability of the node.

This Divisible Load Theory can be applied to any kind of network. The processor partitions the load into many fractions, keeps one of the fractions for itself to process and sends the rest to its neighbors for processing. One of the main points to be taken to consideration is about how balance can be achieved while distributing the load between the processors in order to complete the computation in shortest possible time. With this we can say that the grain of parallelism is small, and there are no data dependencies. The size of the load parts should be adjusted to the speeds of communication and computation such that they have the shortest computation time for the given job. Some examples of real divisible applications include image processing applications like feature extraction, edge detection and many signal processing applications.

In this divisible load theory the manner in which the partitioning of the load depends on its *divisibility property*, which can be defined as the property which determines whether a load can be decomposed into a set of smaller loads or not as shown in figure 1.

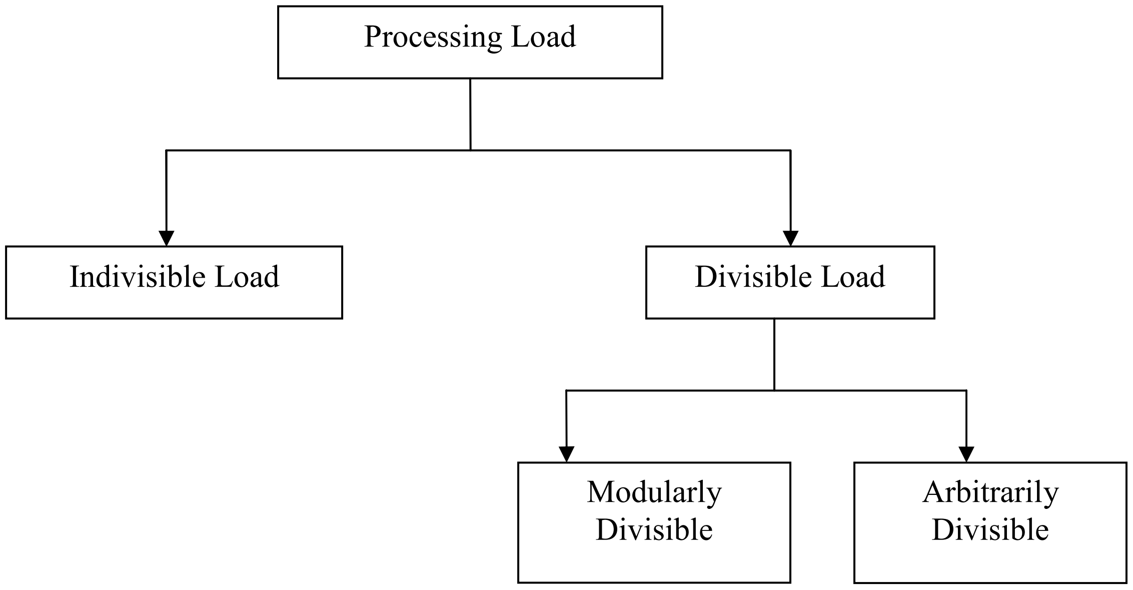


Figure 1: Classification of Processing Loads [2]

Multi-Installment Strategy

This new strategy of load distribution became an area of research interests as it minimizes the processing time. Divisible Load Theory divides the computations into parts of arbitrary sizes and these can be processed independently in parallel. To reduce the waiting time during the parallel computation the load is sent to the processors in multiple small installments. This can also be explained as the distribution of load in more than one installment in an optimal manner to minimize the processing time. The divisible load theory makes an assumption that the processor starts computation only after it has received the entire load that has been assigned to it and this gives a considerable amount of ideal time for almost all the processors because of the delay involved in communicating load from one processor to another. In order to reduce the ideal time this multi installment strategy will be used in which the load assigned to processor is sent in more than one installment.

In any network communication delays constitute an important part of the processing time. In order to reduce the initial waiting of data and for the computations to be initialized, the entire load is divided into small chunks rather than sending as one big chunk which is the divisible load theory. This way of divisible load distribution and execution is called multi-installment processing [4]. This strategy reduced the idle time of the processors in the network to a great extent, which in turn reduced the communication delays.

Bioinformatics

Introduction

Information science when applied to biology produced a field called the “Bioinformatics”. The areas of bioinformatics and computational biology involve the use of techniques and concepts including applied mathematics, informatics, statistics, computer science, artificial intelligence, chemistry, and biochemistry to solve biological problems usually on the molecular level. The terms of bioinformatics and computational biology are often interchangeable. Research in computational biology often overlaps with systems biology. Major research efforts in the field include sequence alignment, gene finding, genome assembly, protein structure alignment, protein structure prediction, prediction of gene expression and protein-protein interactions, and the modeling of evolution. The area of bioinformatics more clearly refers to the creation and advancement of algorithms, computational and statistical techniques, and also includes the theory to solve formal and practical problems arising from the management and analysis of biological data. Computational biology refers to hypothesis-driven investigation of a specific biological problem using computers, carried out with experimental or simulated data, with the primary goal of discovery and the advancement of biological knowledge. In other words, bioinformatics is concerned with the information while computational biology is concerned with the hypotheses.

The NIH Biomedical Information Science and Technology Initiative Consortium that was held on July 17, 2000 has agreed on formal definitions for bioinformatics and computational biology. They also recognized that there is no definition that could completely eliminate the overlap of the variations in interpretation by different individuals and organizations. One of the definition proposed by them are as follows:

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 [32].

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 [32].

The areas of bioinformatics and computational biology use mathematical tools to extract useful information from data produced by high-throughput biological techniques such as genome sequencing. One of the most common representative problems in bioinformatics is the assembly of high-quality genome sequences from fragmentary "shotgun" DNA sequencing. Other common problems include the study of gene regulation using data from micro arrays or mass spectrometry.

Sequence Analysis

Sequence Analysis" in biology can be explained by subjecting a DNA or peptide sequence to sequence alignment, sequence databases, repeated sequence searches, or other bioinformatics methods on a computer. Sequence analysis in molecular biology and bioinformatics is an automated, computer-based examination of characteristically fragments, for example a DNA-strand. It basically includes five biologically relevant topics:

1. This is used for the comparison of sequences in order to find similar sequences (sequence alignment)
2. In identification of gene-structures, reading frames, distributions of introns and exons and regulatory elements
3. Used for prediction of protein structures
4. Used for genome mapping
5. Comparison of homologous sequences to construct a molecular phylogeny.

Similarity detection is often used in biological analysis. This is widely used when a new gene sequence and unknown gene sequence can give significant understanding on the function, structure and origin of the new gene. While comparing two gene sequences, which is also known as aligning two sequences residues form, one sequence is compared with the residues of the other, in which the position of the residues is taken into consideration. The different operations that can be performed are insertion, deletion and substitution of residues in other sequence.

Many algorithms have been proposed in the literature for comparing two biological sequences for similarities. The most popular algorithms in aligning the DNA are the

Needleman-Wunsch algorithm. For protein alignment it's the Smith-Waterman algorithm. In the sequence comparison a combination of the DLT approach and the algorithms are used in order to align the sequences accurately.

Needleman-Wunsch Algorithm

The Needleman–Wunsch algorithm is one of the algorithm that performs a global alignment on two sequences (called X and Y here). This algorithm finds its application in bioinformatics to align protein or nucleotide sequences. The algorithm was first proposed by Saul Needleman and Christian Wunsch in 1970 .The Needleman–Wunsch algorithm is an example of dynamic programming, and was the first application of dynamic programming to biological sequence comparison.

The algorithm can be explained in the following steps

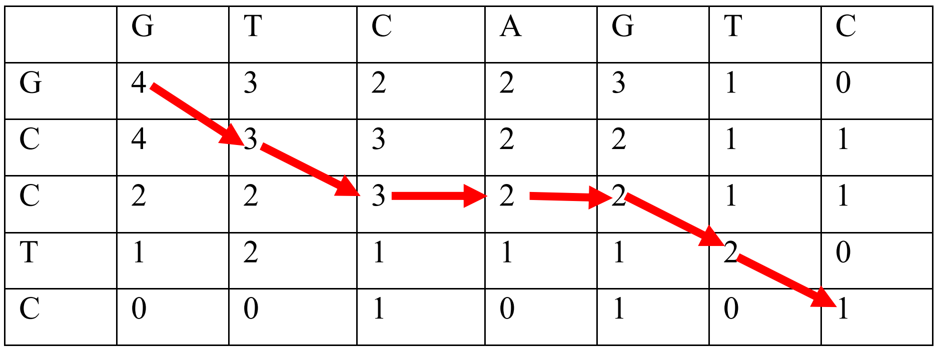
1. Initialize the matrix S=0
2. Fill in the matrix S with 1 if it is a match and 0 if it is a mismatch
3. Compute score from right hand bottom based on the formula,

   $M[i,j]=S[i, j]+Max\{M[i+1:x],M[j+1:y]\}$.
4. Trace back from the left-top corner, and select the maximum value from the adjacent column and row, and so on.

For example let us consider two sequences GTCAGTC and GCCTC. In order to align these sequences we first need to construct the matrix as shown below

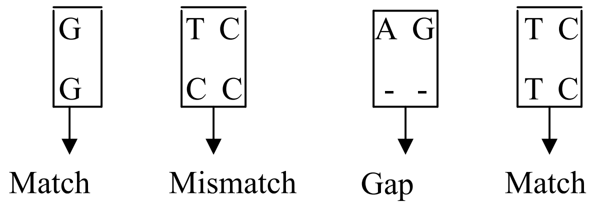
	G	T	C	A	G	T	C
G	1	0	0	0	1	0	0
C	0	0	1	0	0	0	1
C	0	0	1	0	0	0	1
T	0	1	0	0	0	1	0
C	0	0	1	0	0	0	1

Figure 2: Needleman-Wunsch Algorithm after the generation of S matrix



	G	T	C	A	G	T	C
G	4	3	2	2	3	1	0
C	4	3	3	2	2	1	1
C	2	2	3	2	2	1	1
T	1	2	1	1	1	2	0
C	0	0	1	0	1	0	1

Figure 3: Needleman-Wunsch Algorithm after the generation of M matrix



BLAST (B̲asic L̲ocal A̲lignment S̲earch T̲ool)

Background Study

In the area of bioinformatics a tool has been developed by National Center for Biotechnological Information called the BLAST. This tool is an algorithm for comparing primary biological sequence information, for amino-acid sequences of different proteins or the nucleotides of DNA sequences.

This tool primarily enables researchers to compare a query sequence with the available database of sequences and identify library sequence that resembles the query. BLAST is primarily used in research for identifying similar gene sequences. This can be explained by the following example, if a scientist discovers a previously unknown gene in a mouse, he will typically perform a BLAST search of the human genome to see if humans carry the same gene and BLAST will identify sequences with similarity in both the species.

Reasons for the wide use of BLAST

This tool is one of the most widely used bioinformatics programs as it addresses the fundamental problem of similarity detection and sequence alignment and the algorithm also emphasizes speed over sensitivity. This speed of the tool plays a vital role in searching the huge genome databases currently available. BLAST can also be used to answer these other questions in the research and is widely used in appropriate sequence matching.

Input / Output in BLAST

The input and output in BLAST are represented by FASTA format. This FASTA sequence is a text-based format for representing nucleic acid or peptide sequence in which base pair or amino acids are represented by single letter. The simplicity of FASTA format makes it easy to manipulate and parse sequences using text-processing tools and scripting languages like Python and Perl.

A sequence in FASTA format begins with a single-line description, followed by lines of sequence data. The description line is distinguished from the sequence data by a greater-than (">") symbol in the first column. The word following the ">" symbol is the identifier of the sequence, and the rest of the line is the description (both are optional). There should be no space between the ">" and the first letter of the identifier. It is recommended that all lines of text be shorter than 80 characters. The sequence ends if another line starting with a ">" appears; this indicates the start of another sequence. A simple example of one sequence in FASTA format:

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus
maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQMSFWGATVITNLF
SAIPYIGTNLVEWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPL
GLTSDSDKIPFHPYYTIKDFLGLLILILLLLLLALLSPDMLGDPDNHMPADPLNTPLHI
KPEWYFLFAYAILRSVPNKLGGVLALFLSIVILGLMPFLHTSKHRSMMLRPLSQALFWT
LTMDLLTLTWIGSQPVEYPYTIIGQMASILYFSIILAFLPIAGXIENY
```

Figure 4: Example of FASTA sequence [34]

From the figure we can see that the header line begins with '>', and this gives a name and/or a unique identifier for the sequence, and often lots of other information too. Many different sequence databases use standardized headers, which helps in automatically extracting information from the header. The header line may contain more than one header, separated by a ^A (Control-A) character.

Sequence representation

After the header and comments, one or more lines may follow describing the sequence: each line of a sequence should have fewer than 80 characters. Sequences may be protein sequences or nucleic acid sequences, and they can contain gaps or alignment characters (see sequence alignment). Sequences are expected to be represented in the standard IUB/IUPAC amino acid and nucleic acid codes, with these exceptions: lower-case letters are accepted and are mapped into upper-case; a single hyphen or dash can be used to represent a gap character; and in amino acid sequences, U and * are acceptable letters (see below). Numerical digits are not allowed but are used in some databases to indicate the position in the sequence.

The nucleic acid codes supported are:

Nucleic Acid Code	**Meaning**
A	**A**denosine
C	**C**ytidine
G	**G**uanine
T	**T**hymidine
U	**U**racil
R	G A (pu**R**ine)
Y	T C (p**Y**rimidine)
K	G T (Ketone)
M	A C (aMino group)
S	G C (Strong interaction)
W	A T (Weak interaction)
B	G T C (not A) (B comes after A)
D	G A T (not C) (D comes after C)
H	A C T (not G) (H comes after G)
N	A G C T (a**N**y)
X	Masked
-	gap of indeterminate length

Table 1: Nucleic acid codes [34]

The amino acid codes supported are:

Amino Acid Code	**Meaning**
A	Alanine
B	Aspartic acid or Asparagine
C	Cysteine
D	Aspartic acid
E	Glutamic acid
F	Phenylalanine
G	Glycine
H	Histidine
I	Isoleucine
K	Lysine
L	Leucine
M	Methionine
N	Asparagine
P	Proline
Q	Glutamine
R	Arginine
S	Serine
T	Threonine
U	Selenocysteine
V	Valine
W	Tryptophan
Y	Tyrosine
Z	Glutamic acid or Glutamine
X	Any
*	translation stop
-	gap of indeterminate length

Table 2: Amino Acid Codes [34]

Algorithm

To run, BLAST requires two sequences as input: a query sequence (also called the target sequence) and a sequence database. BLAST will find subsequences in the query that are similar to subsequences in the database. In typical usage, the query sequence is much smaller than the database, e.g., the query may be one thousand nucleotides while the database is several billion nucleotides.

BLAST searches for high scoring sequence alignments between the query sequence and sequences in the database using a heuristic approach that approximates the Smith-Waterman algorithm. The exhaustive Smith-Waterman approach is too slow for searching large genomic databases such as GenBank. Therefore, the BLAST algorithm uses a heuristic approach that is slightly less accurate than Smith-Waterman but over 50 times faster. The speed and relatively good accuracy of BLAST are the key technical innovation of the BLAST programs and arguably why the tool is the most popular bioinformatics search tool.

The BLAST algorithm can be conceptually divided into three stages.

- In the first stage, BLAST searches for exact matches of a small fixed length W between the query and sequences in the database. For example, given the sequences AGTTAC and ACTTAG and a word length W = 3, BLAST would identify the matching substring TTA that is common to both sequences. By default, W = 11 for nucleic seeds.
- In the second stage, BLAST tries to extend the match in both directions, starting at the seed. The ungapped alignment process extends the initial seed match of length W in each direction in an attempt to boost the alignment score. Insertions and deletions are not considered during this stage.

If a high-scoring ungapped alignment is found, the database sequence is passed on to the third stage.

In the third stage, BLAST performs a gapped alignment between the query sequence and the database sequence using a variation of the Smith-Waterman algorithm. Statistically significant alignments are then displayed to the user [34].

CHAPTER III

METHODOLOGY

Analysis of Needleman-Wunsch Algorithm

We will first briefly discuss about the Needleman-Wunsch Algorithm as well as some of the characteristics of the S and M matrices that are generated by the algorithm. In aligning the two biological sequences that are denoted as Seq X and Seq Y of length x and y respectively, the algorithm generates two matrices represented by S and M as mentioned above. The matrices S and M are related to each other with the equation $M[i,j]=S[i, j]+\mathrm{Max}\{M[i+1{:}\, x\,],M[j+1{:}\, y]\}$ for the range $1 \le p \le x$, $1 \le q \le y$ where $S_{p,q}$ and $M_{p,q}$ represents the p^th^ row and q^th^ column of the matrices S and M respectively.

In this computation process, residues in Seq X and Seq Y are tested recursively so that they will give the best possible alignment. S and M matrices computation gives the best possible alignment. The S matrix contains all the scores for the matches and mismatches and the M matrix give the alignment of the sequence and the data dependencies are given as shown in figure 5.

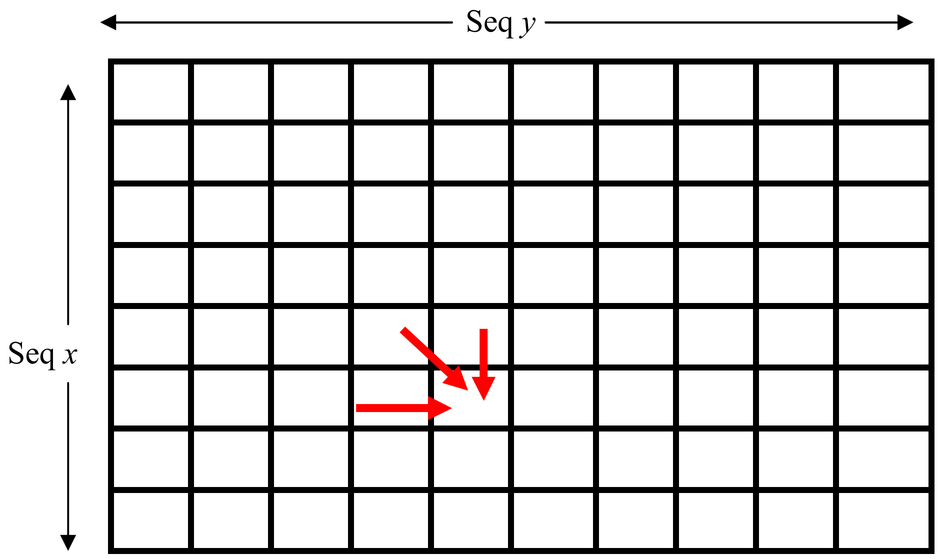


Figure 5: Illustration of the computational dependency of the element (p,q) in the M matrix

The pseudo code for the Needleman-Wunsch Algorithm is as shown below

```
For the calculation of the S matrix

for i=1 to length (x)
S(i,1)← 1  if it is a match
S(i,1)← 0  if it is a mis-match
for i=1 to length (y)
S(i,1)← 1  if it is a match
S(i,1)← 0  if it is a mis-match

This completes the calculation of S matrix

For the calculation of M matrix

for i= length (x) to 1
for j= length (y) to 1
M(i,j)=S(i,j)
Score← M(i,j)
Score_diagonal ← M(i+1,j+1)
Score_left← M(i+1,j)
Score_Right ← M(I,j+1)
While (length(x)!=0 & length(y)!=0)
If ((Score_diaognal ≥ Score_left) && (Score_diaognal ≥ Score_Right))
{
   Then there is a match
}
Else if ((Score_left ≥ Score_diagonal) && (Score_Right ≥ Score_diagonal))
{
                  Then there is a mismatch/ gap
}
Else if ((Score_left ≥ Score_diagonal) && (Score_Right ≥ Score_diagonal))
{
                  Then there is a mismatch/ gap
}
```

Figure 6: Pseudo code for the Needleman-Wunsch Algorithm

Problem Formulation

In this section, the network topology will be discussed. The network that will be considered is a simple single level tree network (SLTN) with the constraint that the root node can communicate only with one child at a time. The approach to the problem can be described in a series of steps. The first step is creating a simple SLTN with a fixed number of nodes and applying divisible load theory on the same network. Further the number of nodes in the system is increased and DLT technique is applied. The two biological sequences are given to the network and the Needleman-Wunsch algorithm gives the alignment. The final aim of this thesis report will include the computation time involved in processing the job. By the results it can be observed that by applying DLT technique the computation time decreases drastically.

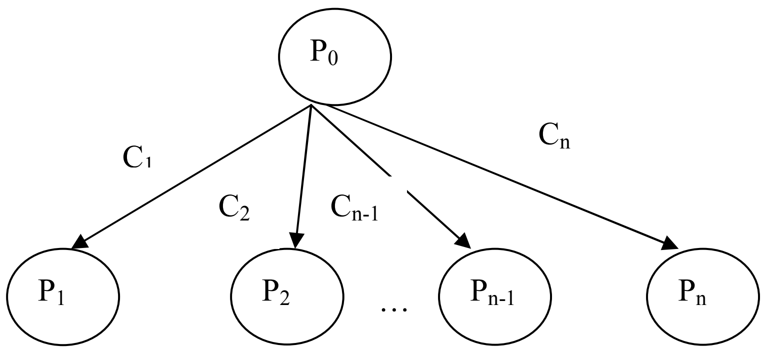


Figure 7: Single Level Tree Network

Our objective in this thesis is to design a strategy such that the processing time or the computation time for the alignment of the two biological sequences is a minimum. The two biological sequences are considered to be I*x* and *y* denoted as Sequence *x* and Sequence *y*. These sequences may vary from 1 character to 1000's of characters. In the results section however the sequences are varied from length of 100 to 1000.

We assume that all the processors in the network $P_{1,}P_{2,\ldots.}P_m$ already have Sequence *x* and Sequence *y* in their local memories, or they can be initialized in this way. To carry the process of sequence alignment in a multiprocessor environment one of the way will be by keeping a copy of the sequences in the local memory.

Design of the Load Distribution Strategy

In this section we describe the load distribution strategy. The distribution strategy for the S matrix is just a matrix of 0's and 1's so it does not have any special kind of distribution. The M matrix is partitioned into sub matrices like $M_{p,q}$ where p= 1,2,….m and q= 1,2….z where each portion of Seq *x* and Seq *y* is contained in one particular cell of the matrix M. This assignment can be explained as shown in Figure 7.

Seq y

Seq x

$M_{m,1}$	$M_{m,2}$	$M_{m,3}$	…	$M_{m,Z-3}$	$M_{m,Z-2}$	$M_{m,Z-1}$	$M_{m,Z}$
$M_{m-1,1}$	$M_{m-1,2}$	$M_{m-1,3}$	…	$M_{m-1,Z-3}$	$M_{m-1,Z-2}$	$M_{m-1,Z-1}$	$M_{m-1,Z}$
$M_{m-1,1}$	$M_{m-1,2}$	$M_{m-1,3}$	…	$M_{m-1,Z-3}$	$M_{m-1,Z-2}$	$M_{m-1,Z-1}$	$M_{m-1,Z}$
$M_{4,1}$	$M_{4,2}$	$M_{4,3}$	…	$M_{4,Z-3}$	$M_{4,Z-2}$	$M_{4,Z-1}$	$M_{4,Z}$
$M_{3,1}$	$M_{3,2}$	$M_{3,3}$	…	$M_{3,Z-3}$	$M_{3,Z-2}$	$M_{3,Z-1}$	$M_{3,Z}$
$M_{1,2}$	$M_{2,2}$	$M_{2,3}$	…	$M_{2,Z-3}$	$M_{2,Z-2}$	$M_{2,Z-1}$	$M_{2,Z}$
$M_{1,1}$	$M_{1,2}$	$M_{1,3}$	…	$M_{1,z-3}$	$M_{1,Z-2}$	$M_{1,Z-1}$	$M_{1,Z}$

Figure 8: Load Distribution Pattern

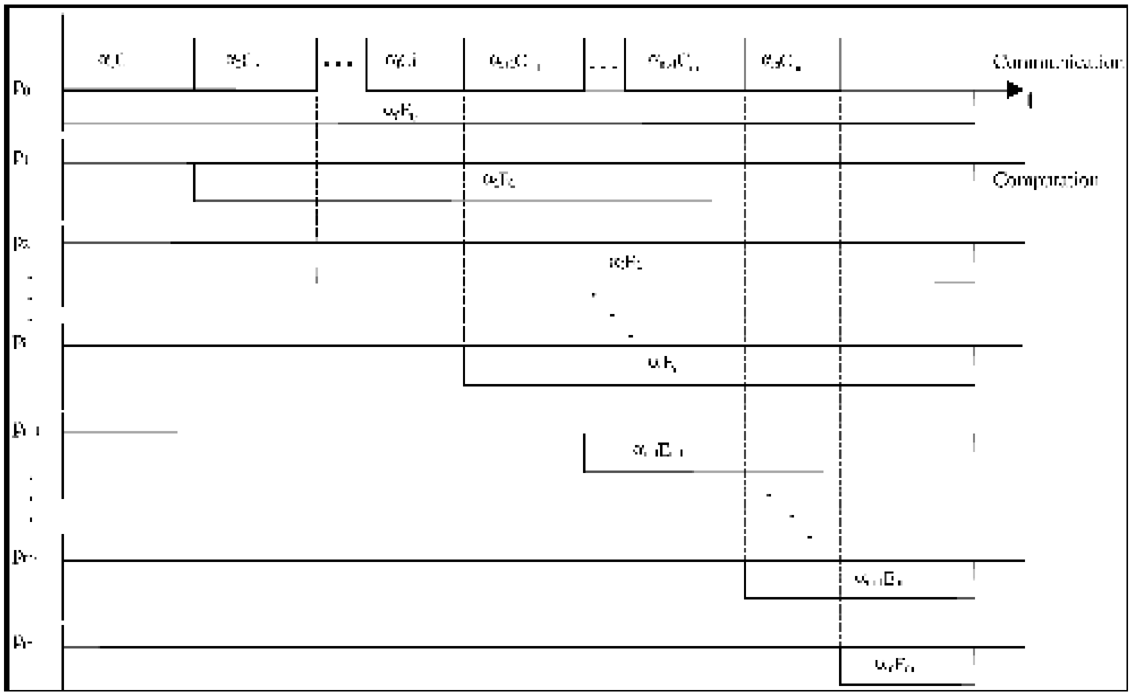


Figure 9: Timing Diagram [22]

The distribution pattern is as shown in figure 8. According to the Needleman-Wunsch Algorithm the last row will be calculated first. So the last row is given to the first processor or the root node of the system. And with respect to the data dependencies the cells with same (p+q) values can be calculated which enables parallel processing to take place in parallel during the calculation of the matrix M. This may be observed form the timing diagram. The generalized equations are as shown below. The two sequences can be fractioned into a number of smaller parts. This can be explained from the example given below. Let us consider that sequence Seq *x* and Seq *y* are where

Seq *x* = GCCTC

Seq *y* = GCTAC

The length of Seq *x* is 5 and length of Seq *y* is 5. There for the total length should be 5. From the above example we can write the generalized equations as follows

We can say that
$$x_1 + x_2 + x_3 + ... + x_{n-1} + x_n = x$$
$$y_1 + y_2 + y_3 + ... + y_{n-1} + y_n = y$$

From the timing diagram we can derive the generalized equation for the load on each processor

By generalizing the equations we can say that

$$\boxed{\sum_{n=2}^{m} x_n = x_{n-1} y_{n-1} E_{n-1} - 2C_{n-1} x_{n-1} / y_n E_n}$$

$$\boxed{x_1 = x /[1 + \sum_{n=2}^{m} [y_1 E_1 - 2C_1] / y_n E_n]}$$

The total completion time for the alignment of the two sequences is given by

$$\boxed{T(m) = xyE_1 + \Sigma\sum_{i=2}^{m} E_i + 2C(m-1)}$$

To enhance the understanding of the performance of Needleman-Wunsch Algorithm and divisible load strategy a single machine has been used [8]. Therefore the speedup, can be defined as

$$\boxed{\text{Speedup} = T(1)/ T(m)}$$

where T(m) is the processing time of our strategy on a system using m-processors. T(1) is the processing time using a single processor and is given by

$$\boxed{T(1)=xyE_1}$$

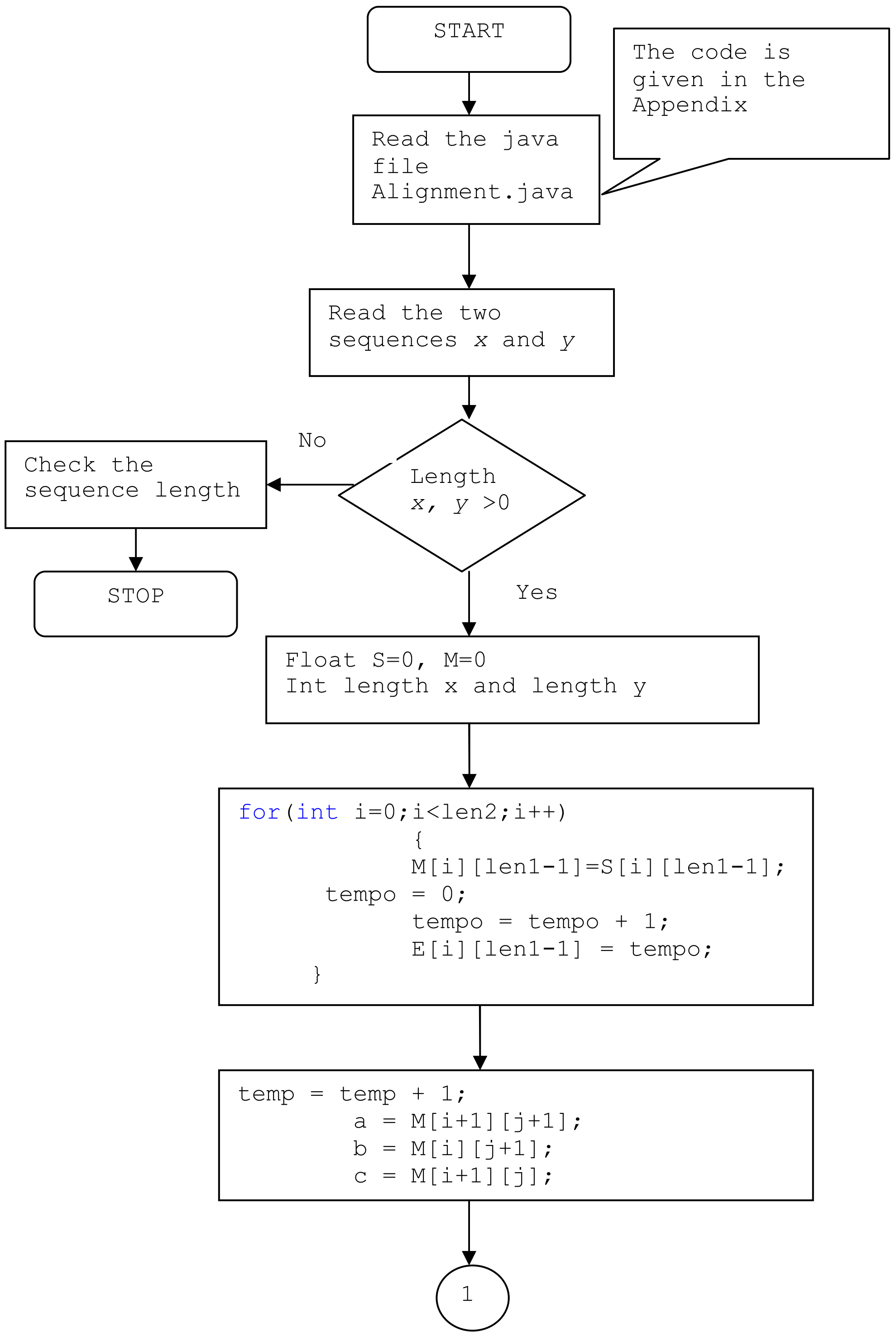
CHAPTER IV

RESULTS

This section discusses the results obtained in the study. The results have been tabulated for the network that has constant link speed and also variable link speed. The testing has been performed for sequence lengths varying from 100 to 1000 characters. The changes have been observed and the graphs have been plotted.

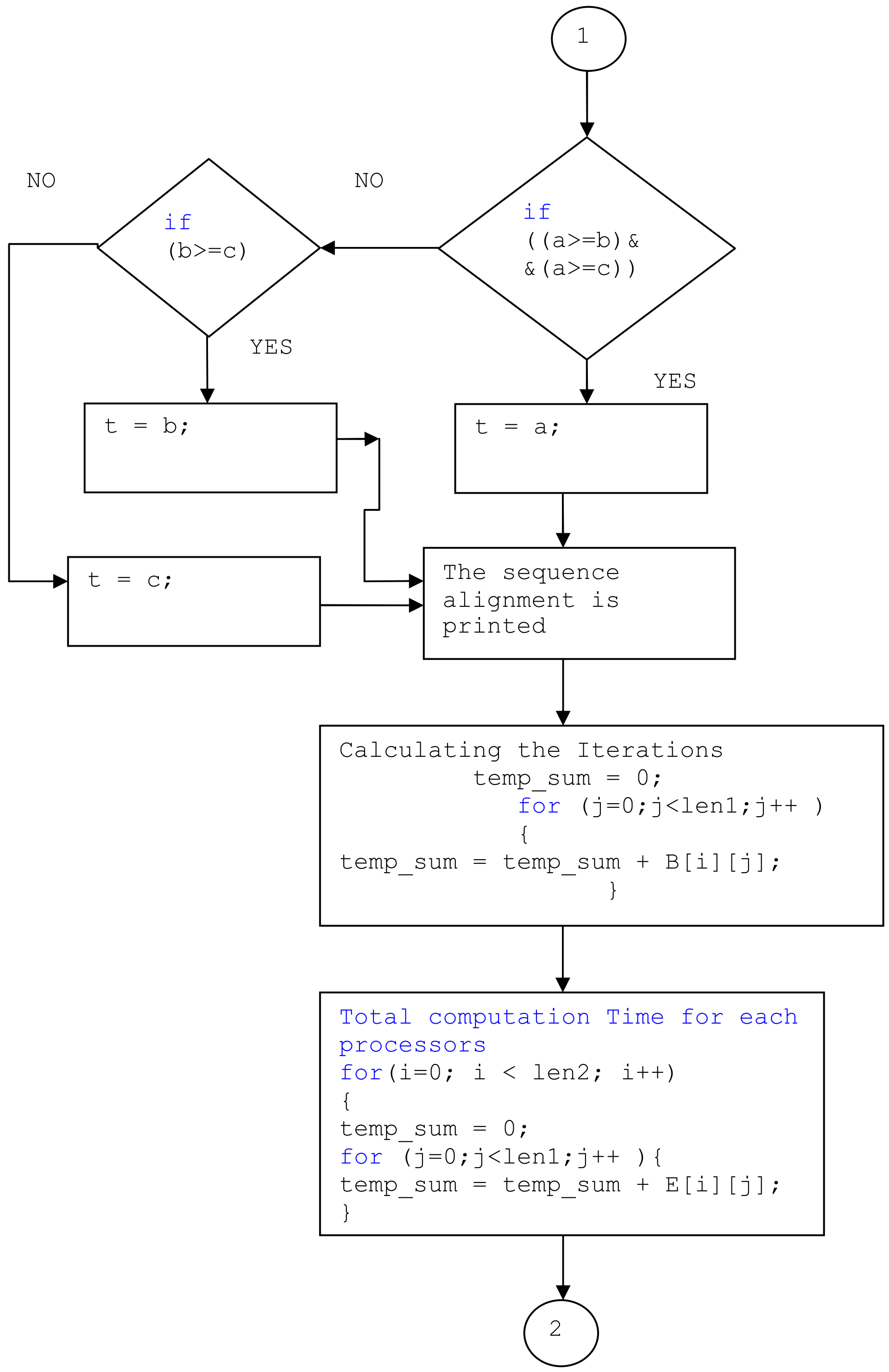
Implementation of the Algorithm

The algorithm implementation is as shown in the flowchart below and the code of is given in Appendix. The Needleman-Wunsch algorithm can be given by a simple diagrammatic representation as shown in Figure 10. The methods and classes that have been used in the coding are also given


START
Read the java file Alignment.java
The code is given in the Appendix
Read the two sequences x and y
Length x, y >0
No
Check the sequence length
STOP
Yes
Float S=0, M=0
Int length x and length y
1

```
for(int i=0;i<len2;i++)
            {
            M[i][len1-1]=S[i][len1-1];
      tempo = 0;
            tempo = tempo + 1;
            E[i][len1-1] = tempo;
     }
```

```
temp = temp + 1;
        a = M[i+1][j+1];
        b = M[i][j+1];
        c = M[i+1][j];
```


1
if ((a>=b)&&(a>=c))
NO
if (b>=c)
NO
YES
YES
t = b;
t = a;
t = c;
The sequence alignment is printed
Calculating the Iterations
temp_sum = 0;
for (j=0;j<len1;j++)
{
temp_sum = temp_sum + B[i][j];
}
Total computation Time for each processors
for(i=0; i < len2; i++)
{
temp_sum = 0;
for (j=0;j<len1;j++){
temp_sum = temp_sum + E[i][j];
}
2

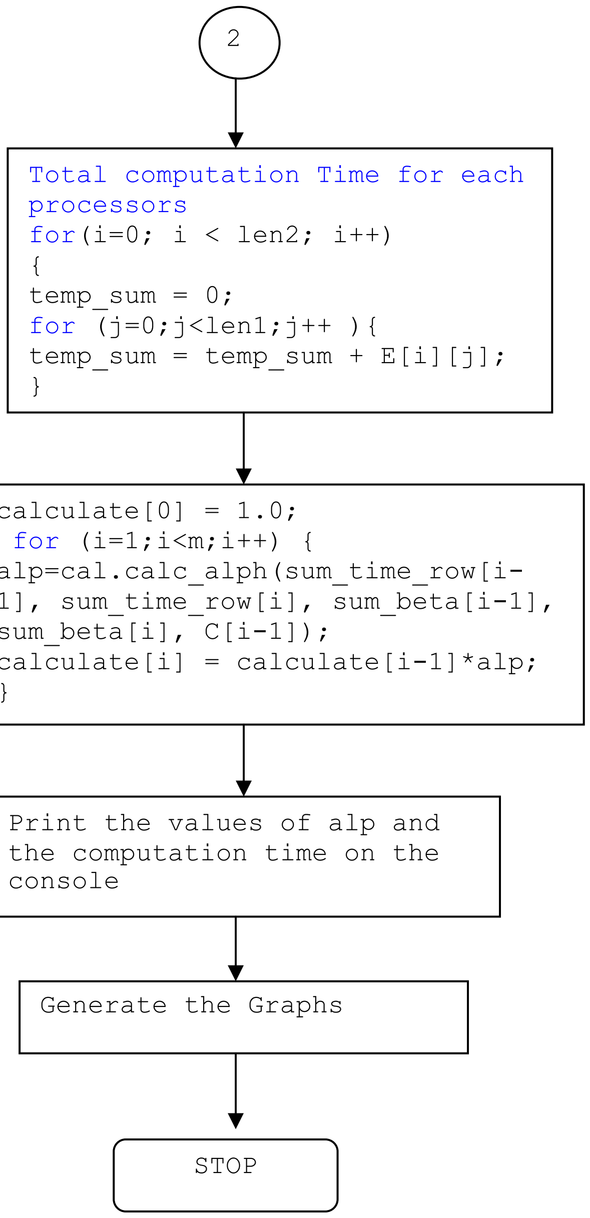

2
Total computation Time for each processors
for(i=0; i < len2; i++)
{
temp_sum = 0;
for (j=0;j<len1;j++){
temp_sum = temp_sum + E[i][j];
}
calculate[0] = 1.0;
for (i=1;i<m;i++) {
alp=cal.calc_alph(sum_time_row[i-1], sum_time_row[i], sum_beta[i-1], sum_beta[i], C[i-1]);
calculate[i] = calculate[i-1]*alp;
}
Print the values of alp and the computation time on the console
Generate the Graphs
STOP

Figure 10: Implementation of Needleman-Wunsch Algorithm

Preliminary Results

This section discusses the preliminary results that have been obtained. As defined in the problem definition a simple single level tree network has been taken as shown in Figure 7. The results have been generated in MATLAB for single level tree network. Figure 11 shows the results generated by applying the divisible load theory on the network. The x- axis of the graph represents the number of children and the Y-axis represents the computation time for the load.

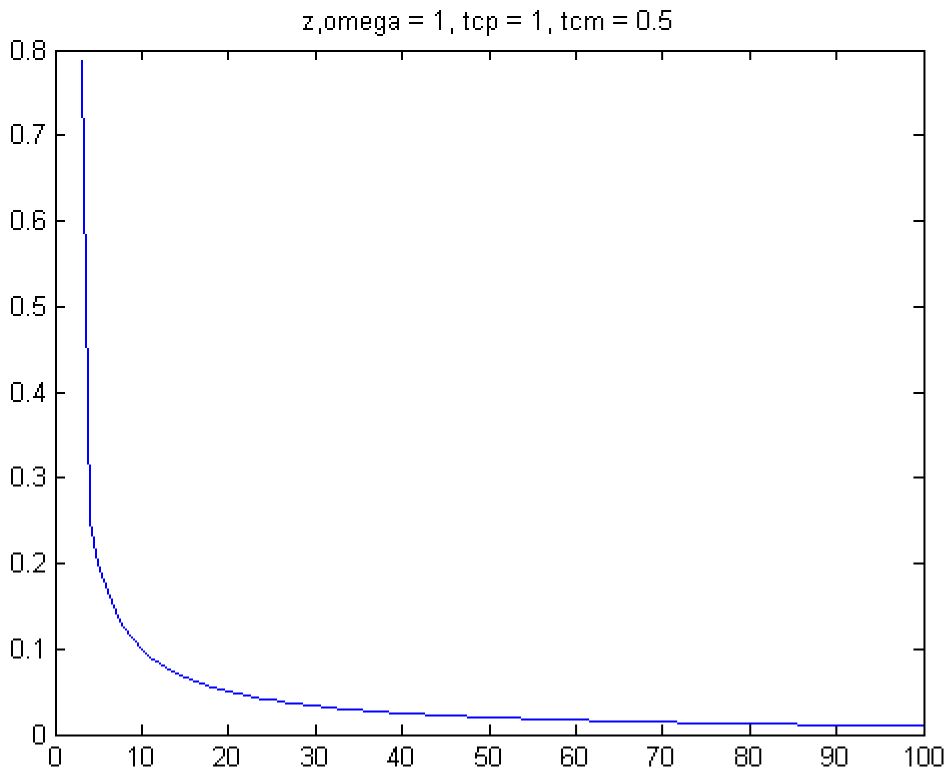


Figure 11: Number of Children (X-axis) Vs Computation Time (Y-axis) applying Divisible Load Theory

The result for the multi-installment strategy is as shown in figure 12. The X- axis represents the number of installments and the Y-axis represents the computation time. In both Figure 11 and Figure 12 we observe that that as the number of children and number of installments increase the computational time decreases.

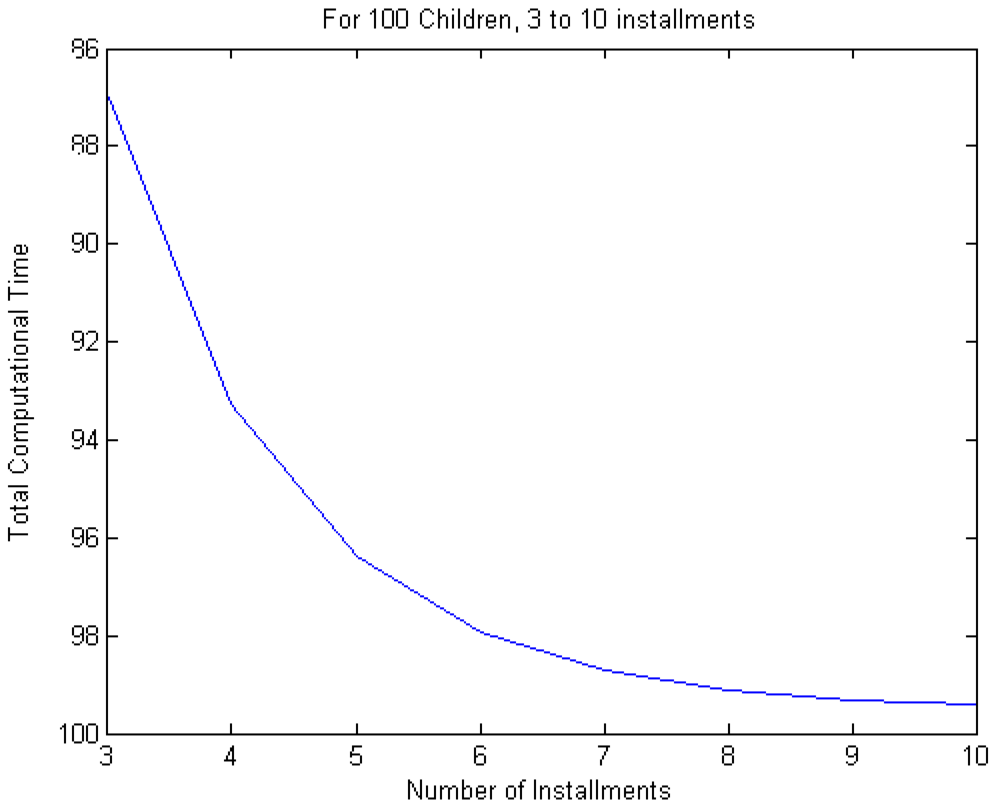


Figure 12: Number of Installments (X-axis) Vs Computation Time (Y-axis) applying Multi Installment Strategy

Final Results

Variable Link Speed

This section briefly discusses about how does the processing time changes when the link speed has been varied. The graphs have been plotted for two ranges of link speed variations. The link speed has been varied from 1-10 nano seconds and 1-100 nano seconds. In graphs (Fig 13, Fig 14, Fig 15) the link speed(C) has been varied from 1-10 nano Seconds. In the graphs (Fig 16, Fig 17) the link speed has been varied from 1-100 nano seconds. From the graphs it can be observed that the processing time depends on the communication link speed C. In other words, the higher the link speed of the network the faster is the job processing. The link speed has been varied using the Random Generator method in java. The results and the tabulated values are as shown below.

Number of processors	Processing Time (Sec)
3	166915.49
5	62671.53
7	33498.78
10	20163.56
20	11250.11
30	6804.34
40	4493.82
50	3186.41
60	2387.78
70	1868.71
80	1514.16
90	1262.08
100	1076.88

Table 3: Values for processing times for variable link speed

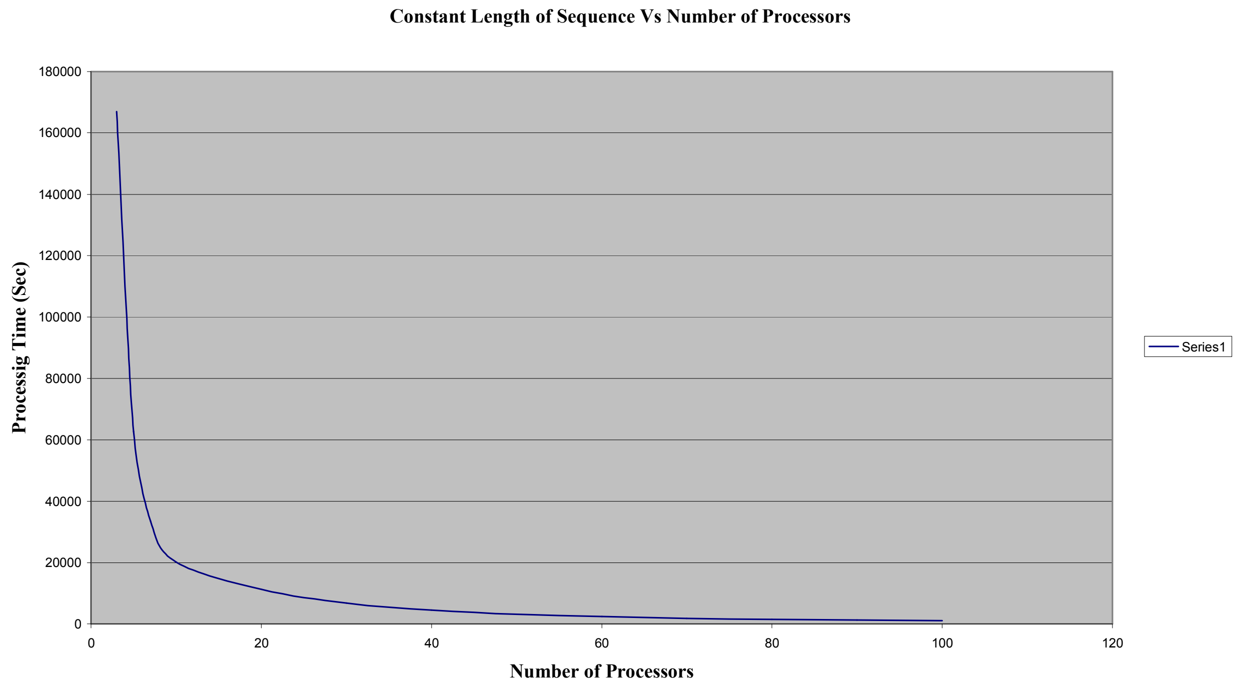


Figure 13: Number of Processors Vs Processing Time for a constant length of sequence

Figure 13 represents the graph for "Number of Processors Vs Processing time" with X-axis as the number of processors and Y-axis as the Processing Time. This graph has been plotted for a constant sequence length of 1000. From the graph it has been observed that for a constant sequence length as the number of processors increase the computation time decreases This also reemphasizes the definition of DLT that as more number of processors are added into the network the processing time decreases.

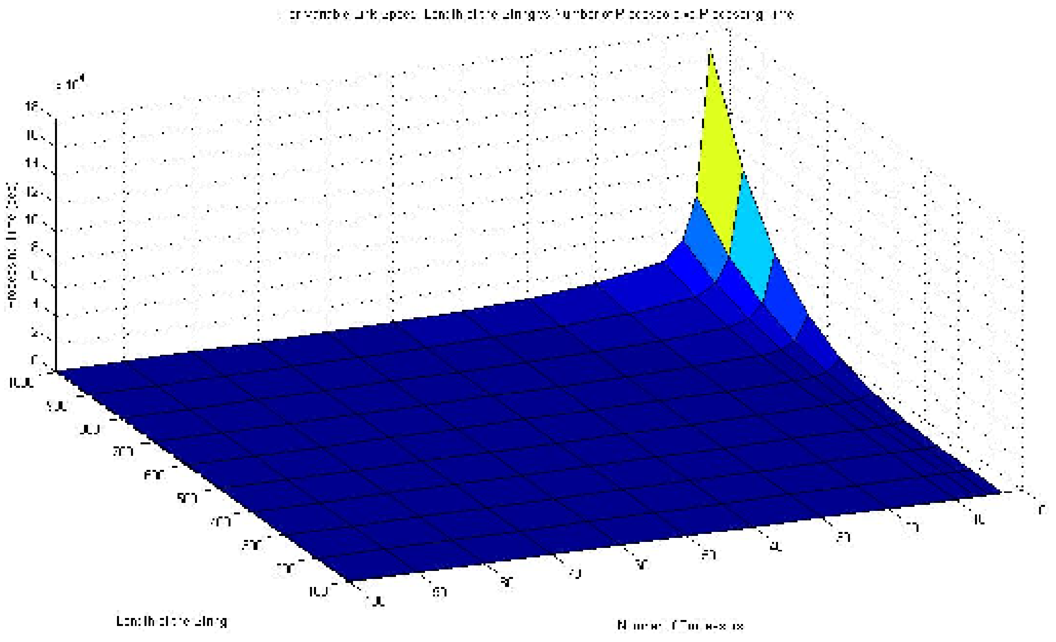


Figure 14: Number of processors Vs Length of the string Vs Processing Time

Figure 14 demonstrates the 3-D representation of how the processing time varies with respect to the length of the sequence and number of processors. From the 3-D graph of the single processor tree network it can be observed that keeping the length of the sequence constant as the number of processors increase the processing time decreases. On the other hand it can also be observed that as the numbers of processors are kept constant and the length of sequences increases and the computation time increases. As discussed in the Methodology chapter, the speedup has been calculated and the values are tabulated as shown for a constant sequence length of 1000.

Number of processors	Speedup (Sec)
3	3
5	7.99
7	14.95
10	24.84
20	44.53
30	73.62
40	111.48
50	157.23
60	209.81
70	268.09
80	330.87
90	396.96
100	465.23

Table 4: Values for speedup for variable link speed

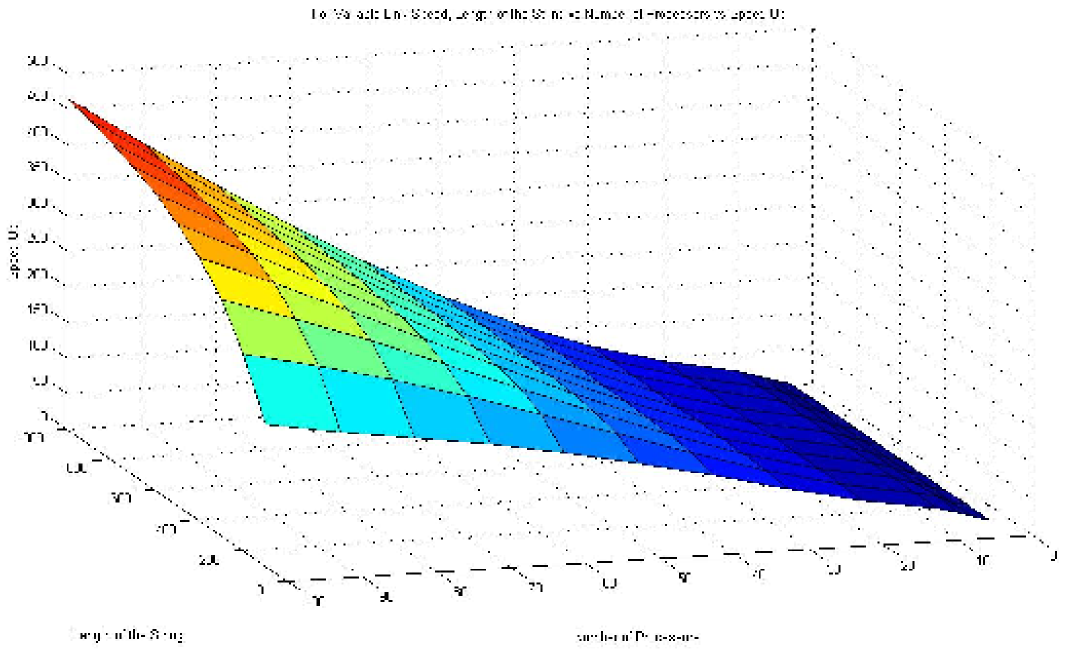


Figure 15: Speed up of the network for variable link speed

Figure 15 represents the 3-D graph for "Speed Up of the Network" with X-axis as the number of processors and Y-axis as the Processing Time. This graph has been plotted for sequence length that is being varied from 100-1000. From the graph it has been observed that speed up achieved for long sequences is greater as the time spent on computation is much larger when compared to idle time of a processor.

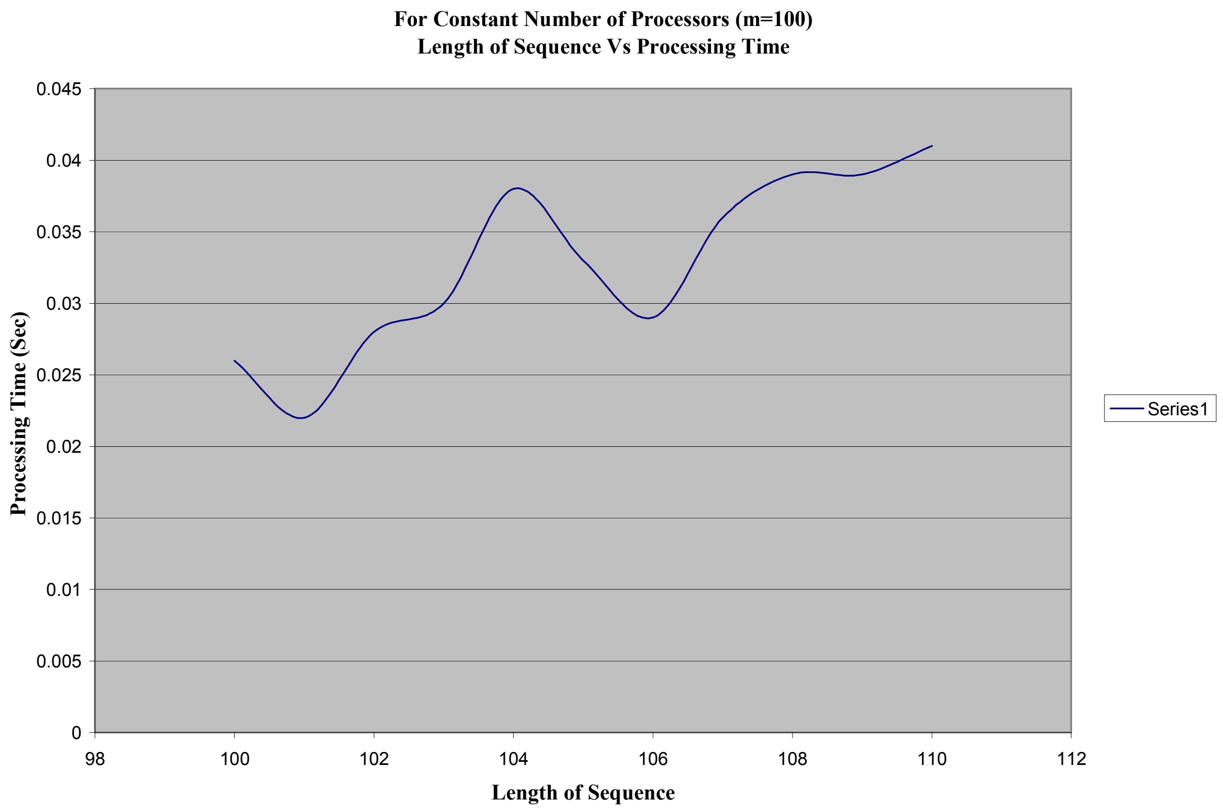


Figure 16: Length of Sequence Vs Processing Time for a constant number of processors

Figure 16 represents the graph for "Length of sequence Vs Processing Time" for a constant number of processors (m=100), in which X-axis represents the length of sequence and Y-axis represents the Processing Time. According to divisible load theory for a constant number of processors as the length of sequence increases the processing time increases because each processors in the system has more load on it. But from the graph we can say that the processing time is not constantly increasing. This can be attributed to the communication link, as the processing time is dependent on the communication link speed. The greater the communication link speed the lesser the processing time.

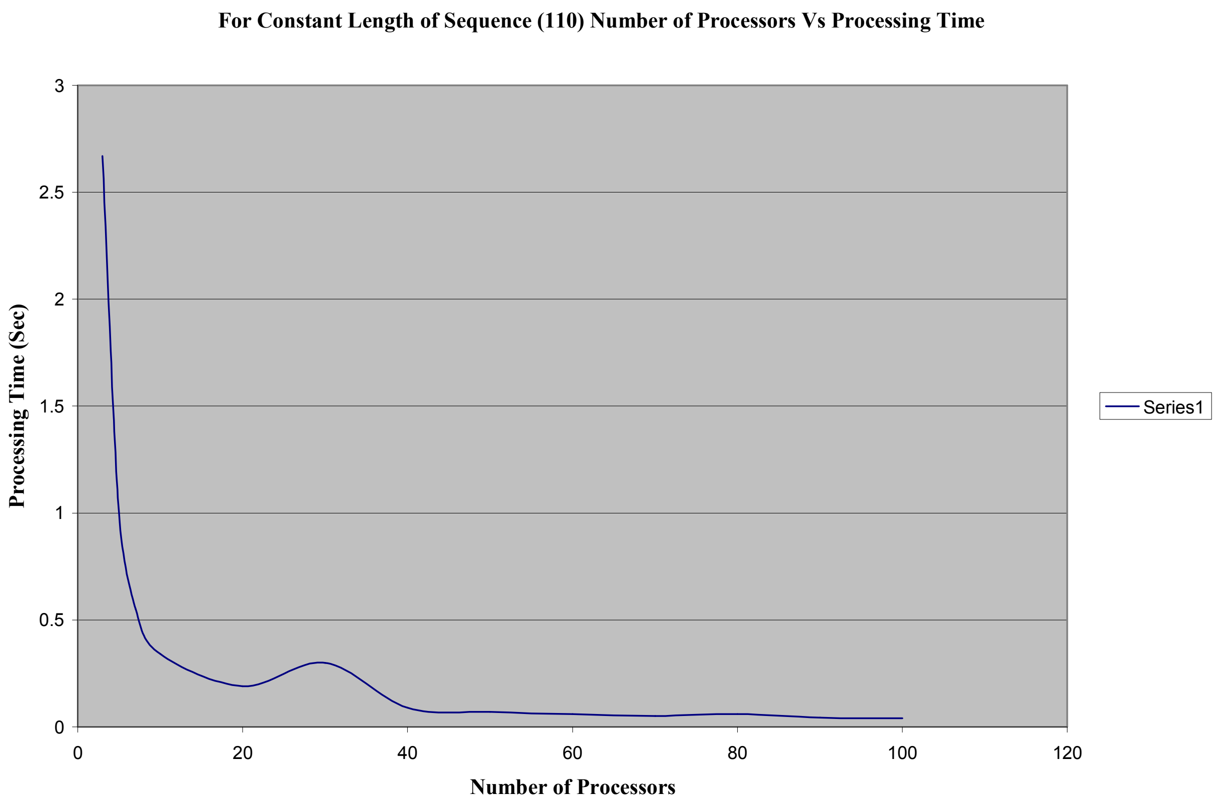


Figure 17: Number of Processors Vs Processing Time for a constant length of sequence

Figure 17 represents the graph for "Number of Processors Vs Processing Time" for a constant length of sequence, in which X-axis represents the number of processors and Y-axis represents the Processing Time. According to divisible load theory for a constant length of sequence as the number of processors increases the processing time should decrease, as more number of processors is being added to the system the load should be distributed among all of them. But from the graph we can say that the processing time is not constantly decreasing. This can be attributed to the communication link, as the processing time is dependent on the communication link speed. The greater the communication link speed the lesser the processing time.

Constant Link Speed

This section briefly discusses the about how the processing time vary when the link speed has been varied. In the graphs shown below (Fig 18, Fig 19, Fig 20) the link speed(C) has been taken as 5 nanoseconds. The results have been discussed as shown below.

Number of processors	Processing Time (Sec)
3	165000
5	62500
7	33400
10	20100
20	11200
30	6790
40	4480
50	3170
60	2380
70	1864
80	1511
90	1250
100	1074

Table 5: Values for processing times for constant link speed

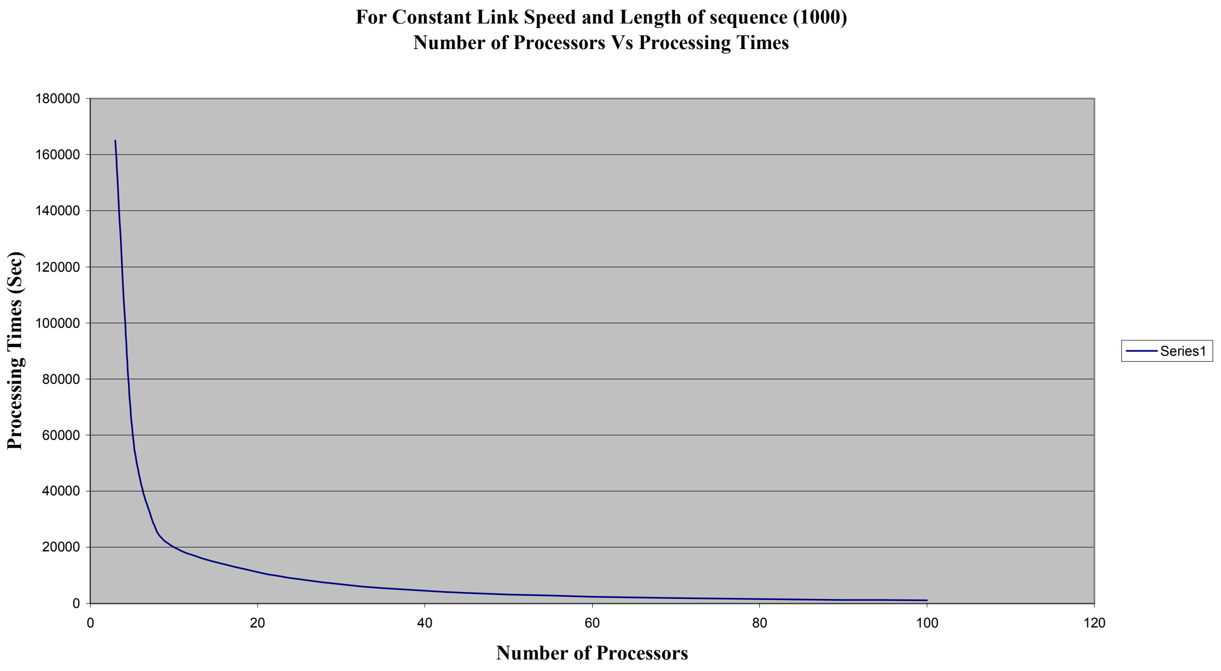


Figure 18: Number of Processors Vs Processing Time for a constant length of sequence

Figure 18 represents the graph for "Number of Processors Vs Processing time" with X-axis as the number of processors and Y-axis as the Processing Time. This graph has been plotted for a constant sequence length of 1000. From the graph it has been observed that for a constant sequence length as the number of processors increase the computation time decreases This adds strength to the definition of DLT that as more number of processors are added into the network the processing time decreases.

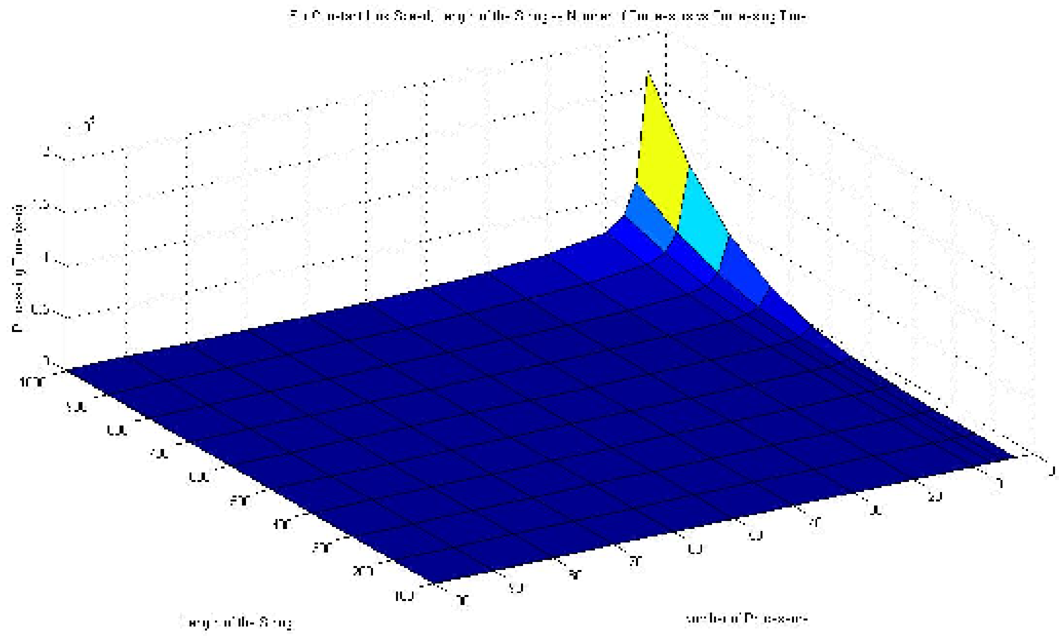


Figure 19: Number of processors Vs Length of the string Vs Processing Time

Figure 19 demonstrates the 3-D representation of how the processing time varies with respect to the length of the sequence and number of processors. From the 3-D graph of the single processor tree network it can be observed that keeping the length of the sequence constant as the number of processors increase the processing time decreases. On the other hand it can also be observed that as the numbers of processors are kept constant and the length of sequences increases the computation time increases. As discussed in the Methodology chapter, the speedup has been calculated and the values are tabulated as shown for a constant sequence length of 1000.

Number of processors	Speedup (Sec)
3	3
5	7.99
7	14.95
10	24.84
20	44.53
30	73.63
40	111.48
50	157.23
60	209.82
70	268.1
80	330.88
90	396.97
100	465.25

Table 6: Values for speedup for variable link speed

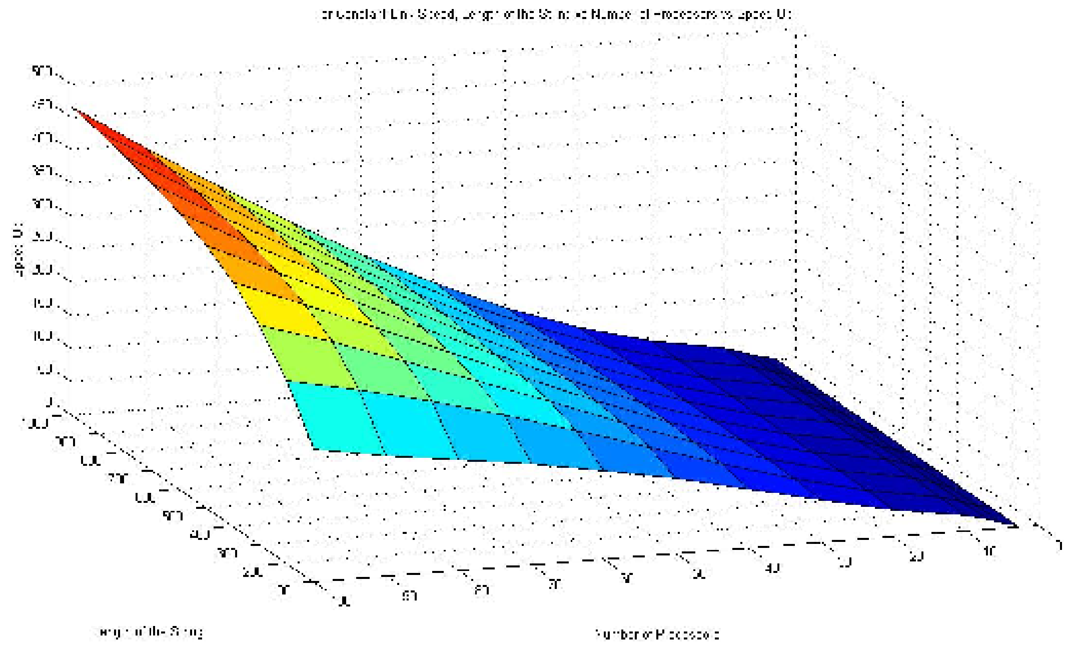


Figure 20: Speed up of the network for constant link speed

Figure 20 represents the 3-D graph for “Speed Up of the Network” with X-axis as the number of processors and Y-axis as the Processing Time. This graph has been plotted for a sequence length that has been being varied from 100-1000. From the graph it has been observed that speed up achieved for long sequences is greater as the time spent on computation is much larger when compared to idle time of a processor.

CHAPTER V

CONCLUSION AND FUTURE WORK

Conclusions

The problem that has been addressed in this research is the alignment of two biological sequences. We proposed a multiprocessor solution using a single level tree network, where communication delays are assumed to be non zero. In the design of this strategy we used the Needleman- Wunsch algorithm in the alignment of the two biological sequences. In the design we used the properties of Divisible Load Theory to determine the number of residues that should be assigned to each processor in the network.

The approach presented in this thesis is, first we had a matrix S which is a matrix of order *x* X *y* where *x* is the length of the first sequence and *y* is the length of the second sequence. Then we derived the M matrix which will give the final values and depending on that we can align the sequences. We derived the equations that will determine the size of the sub matrices according to the processor speeds where here it is assumed that all processors have equal speeds and the communication speeds are varied. With these constraints the equations have been derived and the graphs have been plotted.

From the results we observed that as the number of processors in the network increases the processing time for the job decreases and the speed up \ increases.

Future Work

Future extensions to this work can be deriving solutions that will further decrease the computation speed. This can be achieved by applying multi-installment strategy and performing the analysis using the Needleman- Wunsch Algorithm. The same problem of aligning biological sequences can be applied to various types of networks.

The alignment of biological sequences can be solved using the Sellers algorithm and the load distribution strategy. Further work can also be carried out on aligning multiple sequences with various types of clustering strategies. The same strategy of aligning sequences can be further extended to aligning multiple sequences using the algorithm like Berger-Munson algorithm

REFERENCES

[1] V. Bharadwaj, D. Ghose and V. Mani. “Multi-installment Load Distribution in Tree Network with Delays”.

[2] V. Bharadwaj, D. Ghose and Thomas G. Robertazzi. “Divisible load Theory: A New Paradigm for Load Scheduling in Distributed Systems”.

[3] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, Y. Yang . “Scheduling Divisible Load on Star and Tree Networks: Results and Open Problems”.

[4] Maciej Drozdowski, Marcin Lawenda. “ On Optimum Multi-installment Divisible load Processing in Heterogeneous Distributed Systems”

[5] Shang Mingsheng, Sun Shixin. “Optimal Multi-installments Algorithm for Divisible Load Scheduling”.

[6] Sameer Bataineh, Te-Yu Hsiung, Thomas G. Robertazzi. “Closed Form Solutions for Bus and Tree Networks of Processors Load Sharing a Divisible Job”.

[7] Tieng K. Yap, Ophir Frieder and Robert L. Martion. “Parallel Computation in Biological Sequence Analysis”.

[8] Wong Han Min, Bharadwaj Veeravalli. “Aligning Biological Sequences on Distributed Bus Networks: A Divisible Load Scheduling Approach”.

[9] David P. Anderson. “BOINC: A System for Public-Resource Computing and Storage”.

[10] David P. Anderson, John McLeod VII. “Local Scheduling for Volunteer Computing”.

[11] Jeeho Sohn, Thomas G. Robertazzi and Serge Luryi. “Optimizing Computing Costs Using Divisible Load Analysis”.

[12] Baohua Wei, Gilles Fedak and Franck Cappello. “Collaborative Data Distriution with BitTorrent for Computational Desktop Grids”.

[13] Chiou-Nan Chen, Kuan-Ching Li, Chuan Yi Tang, Yaw-Lin Lin, Hsiao-His Wang, Tsung-Ying Wu. “On Design and Implementation of a Bioinformatics Portal in Cluster and Grid Environments”.

[14] C. Germain. “ Result Checking in Global Computing Systems”. ACM Int. Conf. on Supercomputing (ICS 03).2003. P. 226-233

[15] Distributed.net, http://distributed.net

[16] BOINC, http://boinc.berkeley.edu/

[17] T.G. Robertazzi, Scheduling in Parallel and Distributed Systems. [Online]. Available: http://www.ee.sunysb.edu/~tom/dlt.html

[18] Mani, V. and Ghose, D., "Distributed Computation in a Linear Network: Closed-Form Solutions and Computational Techniques", IEEE Transactions on Aerospace & Electronic Systems, Vol. 30, No. 2, April 1994.

[19] Robertazzi, T.G., "Processor Equivalence for a Linear Daisy Chain of Load Sharing Processors", IEEE Transactions on Aerospace and Electronic Systems, Vol. 29, No. 4, Oct. 1993, pp. 1216-1221.

[20] Cheng, Y.C. and Robertazzi, T.G., "Distributed Computation for a Tree Network with Communication Delays", IEEE Transactions on Aerospace and Electronic Systems, Vol. 26, No. 3, May 1990, pp. 511-516.

[21] Cheng, Y.C. and Robertazzi, T.G., "Distributed Computation with Communication Delays", IEEE Transactions on Aerospace and Electronic Systems, Vol. 24, No. 6, Nov. 1988, pp. 700-712.

[22] Li, X., Bharadwaj, V. and Ko, C.C., "Optimal Divisible Task Scheduling on Single-Level Tree Networks with Finite Size Buffers", Accepted for publication in IEEE Transactions on Aerospace and Electronic Systems, February 2000.

[23] Li, X., Bharadwaj, V. and Ko, C.C., "Divisible Load Scheduling on Single Level Tree Networks with Buffer Constraints", IEEE Transactions on Aerospace and Electronic Systems. vol. 36, no. 4, Oct. 2000, pp. 1298-1308.

[24] V. Bharadwaj, Xiaolin Li, and, Ko Chi Chung, "On the Influence of Start-up Costs in Scheduling Divisible Loads on Bus Networks", IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 12, pp. 1288-1305, December 2000.

[25] Bharadwaj, V. and Viswanadham, N., "Sub-Optimal Solutions Using Integer Approximation Techniques for Scheduling Divisible Loads on Distributed Bus Networks", IEEE Transactions on Systems, Man, and Cybernetics: Part A, Vol. 30, No. 6, pp. 680-691, November 2000.

[26] Jones, Neil C. An introduction to bioinformatics algorithms.

[27] Aaron Elkiss. "Biological Database Normalization by Sequence Alignment"

[28] T.K. Yap, O. Frieder, and R.L. Martino, High Performance Computational Methods for biological sequence analysis. Norwell, MA: Kluwer, 1996.

[29] Bharadwaj, V., Ghose, D. and Mani, V., "An Efficient Load Distribution Strategy for a Distributed Linear Network of Processors with Communication Delays", Computer and Mathematics with Applications, Vol. 29, No. 9, May 1995, pp. 95-112.

[30] Gerrit Voss, Andre Schröder. Wolfgang Müller- Wittig, Bertil Schmidt. " Biological Sequence Alignment on Graphics Processing Units".

[31] Robertazzi, T.G. "Ten Reasons to Use Divisible Load Theory".

[32] NIH Working Definition of Bioinformatics and Computational Biology

[33] Altschul, SF, W Gish, W Miller, EW Myers, and DJ Lipman. "Basic local alignment search tool". J Mol Biol 215(3):403-10, 1990

[34] http://en.wikipedia.org/wiki/BLAST

[35] http://en.wikipedia.org/wiki/FASTA_format

[36] T.F.Smith and M.S.Waterman "Identification of Common Molecular Subsequences"

[37] D.A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, B.A. Rapp, and D.L. Wheller, "Genbank," *Nucleic Acids Research*, Vol. 28, no. 1, pp. 15-18, 2000.

[38] http://www.rff.com/flowchart_shapes.htm

APPENDICES

This gives the coding for the final results of the thesis. The algorithm Needleman-Wunsch has been coded in java and the code is as shown below.

```
/***********************************************************************
********************************
* Name: Sudha Gunturu
* Thesis Title: Load Scheduling for Bioinformatics Applications in
Large Scale Networks
* Algorithm Implemented for Sequence Alignment: Needleman-Wunsch
Algorithm
* Advisor: Dr Xiaolin (Andy) Li
* Commitee Members: Dr.Park, Dr.George
* The aim of this thesis is to prove that as the number of processors
in the
* network are incresead the total processing time for sequence
alignment
* decreases.
* The algorithm that is used in sequence alignment is "Needleman-
Wunsch"
* algorithm.
* The algorithm will align the given two sequences and then also
calculate
* the load distriburtion that should be
* given to each processor.
*
/***********************************************************************
*********************************************/
import java.io.*;
import java.util.*;
import java.lang.*;

 /*This is the main class  */

public class Alignment
{

    public static void main(String[] args)
    {
    String seq1,seq2;
    // The two strings that are to be aligned are declared as seq1 and
    // seq2

seq1="GTCAGTCUVBERTYUDJSEIOPOSMHREUNFDHIKZHFYRNJHTUWLCMNUYRFJDHTURTUTBN
BVDURUSSSMSKDUTITTPOLVREYWWIQNFHJHE";
```

```
seq2="VQPTPADHFTFGLWTVGWTGADPFGVATRANLDPVEAVHKLAELGAYGITFHDNDLIPFDATAAE
REKILGDFNQALADTGLKVPMVTTNLFSHPVFKDG";

    /* There are two matrices that are used in the Needleman-Wunsch
     * Algorithm. One matrix gives the match and mismatch an
     * other matrix gives the string alignment. In this program S is
declared
     * as an array that gives the match and mismatch
     * and the other matris is M which is also an array that gives the
final
     * alignment of the string */
     int len1=seq1.length();
     int len2=seq2.length();
     float S[][]=new float[len2][len1];
     float M[][]=new float[len2][len1];


    /* This for loop sets the value of the S matrix to 1 and 0's
depending on
     * weather it is a match or mismatch. If it is a
     * match a value of '1' is assigned in the respective cell and if
it is a
     * mismatch a value of '0' is assigned in the cell */
      for(int i=0;i<len2;i++)
             {
      for(int j=0;j<len1;j++)
       {
       if((seq2.charAt(i))==(seq1.charAt(j)))
            {
                  S[i][j]=1;
            }// End of if
            else
            {
                S[i][j]=0;
            }// end of else
      } // End of nested for loop
    } //End of for loop

  /* This loop will intialize the initial value of the second matrix M
that
   * will give the final value of the alignment to
   *  zero. Then further values are filled into the matrix depending on
the
   * comparisons which follows from Node- Left-Right*/

    for(int i=0;i<len2;i++)
       {
       for(int j=0;j<len1;j++)
      {
         M[i][j]=0;
       }//End of inner for loop
    }//End of outer for loop*/

    /* The array E is defined as the time taken to calculate each cell
in the
     * matrix which is also given as the computation time. When the row
is
```

```
     * assigned to a processor the time taken by each row is given as
the
     * total computation for each processor.*/

            float E[][]=new float[len2][len1];
            float tempo = 0;
   /* This loop performs the assignment of values in the matrix M the
last
    * row of S is made as the first row of M accordingthe implementaion
of
    * the Needleman-Wunsch Algorithm and tempo is a temporary variable
that
    * is used in the calculation of the times for each cell. This loop
    * performs the operation od storing the last row of the S matrix as
the
    * first row of the M matrix*/
     for(int j=0;j<len1;j++)
      {
            tempo = 0;
      M[len2-1][j]=S[len2-1][j];
      tempo = tempo + 1;
            E[len2-1][j] = tempo;
     }// End of for loop

/* This loop performs the operation of calculating the values of M
matrix  */
       for(int i=0;i<len2;i++)
            {
            M[i][len1-1]=S[i][len1-1];
      tempo = 0;
            tempo = tempo + 1;
            E[i][len1-1] = tempo;
     }// End of for loop

    for(int i=len2-2;i>=0;i--) {
            tempo= 0;
      for(int j=len1-2;j>=0;j--)  {
            tempo= 0;
            float low=0;
            for(int l=i+1;l<=len2-1;l++) {
                  tempo = tempo + 1;
                  for(int k=j+1;k<=len1-1;k++) {
                        tempo = tempo + 1;
                        if(low<M[l][k])
                              low=M[l][k];
                  }
            }// end of nested outer for loop
            M[i][j]=S[i][j]+low;
            E[i][j] = tempo;
      }     //end of inner for loop
    }//end of outer for loop


// ----------------------------------------------------------------------
----------------------
```

```
  /* For finding the flow in the matrix M. Computing the path and
saving the
   * row and coloumn indices which has the
   * higher values i has the row element, j has the coloumn element and
this
   * method will also give the alignment. The comparison will follow
the
   * sequence of Node- Left-Right that is it will first compare the
element
   * that is to its diaognal then to its left and then to its right*/


      int i=0,j=0,temp = 0;
    float a=0,b=0,c=0, t = -1.0f;
    int seq1_new[]=new int[len1+len2];
    int seq2_new[]=new int[len2+len1];
    for(int k=0;k<len1+len2;k++)   {
      seq1_new[k]=-1;
      seq2_new[k]=-1;
    }
    seq1_new[temp] = j;
    seq2_new[temp] = i;
    while((i<len2-1)&&(j<len1-1))   {
      temp = temp + 1;
        a = M[i+1][j+1];
        b = M[i][j+1];
        c = M[i+1][j];

        if ((a>=b)&&(a>=c))
            t = a;
            else if (b>=c)
                  t = b;
                  else
                        t = c;

        if (t==M[i+1][j+1])
        {
                   i = i+1;
                 j = j+1;
        }
        else if (t == M[i][j+1])
        {
            i = i;
            j = j+1;
        }
        else
        {
            i = i+1;
            j = j;

        }
        seq1_new[temp] = j;
      seq2_new[temp] = i;

    }
// ---------------------------------------------------------------------
------
```

```
  /* This will give the string alignment of the two strings. There are
three kinds of cases which can be termed as
   * 1. Match- if the same alphabet is found
   * 2. Mismatch- if different alphabet is found
   * 3. Gap- in order to align the sequence in the best possible way a
gap is
   *    inserted and this alignment is based upon the path trace in the
final
        M matrix. */

        int p = 0;
        String seq1_new_str,seq1_xx;
        String seq2_new_str,seq2_xx;
        seq1_new_str= seq1.charAt(seq1_new[0]) + "";
        seq2_new_str= seq2.charAt(seq2_new[0]) + "";
        p = p +1;


         while((seq1_new[p] != -1) && (seq2_new[p] != -1)) {
// if we dont represent the same row element twice get the char else
keep it as a '-'
            if (seq1_new[p]!= seq1_new[p-1])
                  {
                  seq1_xx =seq1.charAt(seq1_new[p]) + "";
                        seq1_new_str =  seq1_new_str +
seq1.charAt(seq1_new[p]) ;
                  }// end of if
            else
                  {
                        seq1_xx = "-";
                        seq1_new_str = seq1_new_str + seq1_xx;
                  }//end of else

 // if we dont represent the same row element twice get the char else
keep it as a '-'
            if (seq2_new[p]!= seq2_new[p-1])
                  {
                  seq2_xx =seq2.charAt(seq2_new[p]) + "";
                        seq2_new_str =  seq2_new_str +
seq2.charAt(seq2_new[p]);
                  }
            else {
                        seq2_xx = "-";
                        seq2_new_str = seq2_new_str + seq2_xx ;
                  }

            p = p + 1;
        }

   /* This conditions will make sure that all the characters in the
strings are visited or not. */
            if(seq1_new[p-1] < len1)
                  {
                  for (i = seq1_new[p-1] + 1; i<len1 ; i++ )
                        {
                        seq1_new_str = seq1_new_str + seq1.charAt(i);
                  }
            }

            if(seq2_new[p-1] < len2){
                  for (i = seq2_new[p-1] + 1; i<len2 ; i++ ) {
                        seq2_new_str = seq2_new_str + seq2.charAt(i);
                  }
            }


              if (seq1_new_str.length() < seq2_new_str.length()) {
                    for
(i=seq1_new_str.length();i<seq2_new_str.length();i++)  {
                          seq1_new_str = seq1_new_str + "-";
                    }
              }


              if (seq2_new_str.length() < seq1_new_str.length())
                  {
                    for
(i=seq2_new_str.length();i<seq1_new_str.length();i++)
                        {
                          seq2_new_str = seq2_new_str + "-";
                    }

              }

      // This will intilize the array B that is used to find the number
of iterations
        float B[][]=new float[len2][len1];
            for(j=0;j<len1;j++)     {
            B[len2-1][j]=1;
         }
             for(i=0;i<len2;i++) {
            B[i][len1-1]=1;
         }

// ----------------------------------------------------------------------
-------------------
 /* To calculate the iterations . This iterations is defined as the
number of times the different cells are to
   * be visited in order to copmlete the M matrix*/

            for( i=len2-2;i>=0;i--)
                  {
                  for( j=len1-2;j>=0;j--)
                        {
                        temp = 0;
                        for(int l=i+1;l<=len2-1;l++)
                              {
                              for(int k=j+1;k<=len1-1;k++)
                                    {
                                    temp = temp  + 1;
                              }
                        }
                        B[i][j]=temp;
                   }
            }



        System.out.println(" ---------------------------------------------
-----");
/*................................ calucation of Beta Values
[Iteration Time] for the each row.......................*/
           float temp_sum;
               float sum_beta[] = new float[len2];
            for(i=0; i < len2; i++)
                  {
                  temp_sum = 0;
                  for (j=0;j<len1;j++ )
                        {
                        temp_sum = temp_sum + B[i][j];
                  }
                  sum_beta[i] = temp_sum;
            }
/*............................Calucating the Total Computation Time for
each Row .................................*/

           float sum_time_row[] = new float[len2];
            for(i=0; i < len2; i++){
                  temp_sum = 0;
                  for (j=0;j<len1;j++ ){
                        temp_sum = temp_sum + E[i][j];
                  }
                  sum_time_row[i] = temp_sum;
            }



/* Calcuating the communication time of the tree. To calculate the
communication time we need the link speed which is
 * generated by the random generator [Number generated between 1 to
10]. This generated random C value is given as the
 * link speed in the calculation of alpha. Link speed is given in nano
seconds.*/


Random randomGenerator = new Random();
float C[] = new float[len2];
 for (int idx = 0; idx <len2; idx++)
      {
       //C[idx] = randomGenerator.nextInt(10) + 1;
       C[idx]=5;
 }


/* ......................Calucating Alpha -- The load on processes
..........................................*/
   /* The loads that are to be assigned to each process is defined by
alpha and with the help of the timing
    * diagram an equation is derived that gives the final value of the
alphas. */
```

```
        double calculate[] = new double[len2];
          double alpha[] = new double[len2];
          double alp,xx=0;
          calcu cal = new calcu();

         // This will take the number of processors that we are going to
test.
            int no_of_processors[] =
{3,5,7,10,20,30,40,50,60,70,80,90,100};
            int m;

            for(int tempt=0; tempt < no_of_processors.length; tempt++)
      {
                  System.out.println("------------------------------------
-----");
                  System.out.println("Processors equal to :" +
no_of_processors[tempt]);
                  m = no_of_processors[tempt];

                 calculate[0] = 1.0;
                   for (i=1;i<m;i++) {
                         alp = cal.calc_alph(sum_time_row[i-1],
sum_time_row[i], sum_beta[i-1], sum_beta[i], C[i-1]);
                         calculate[i] = calculate[i-1]*alp;
                   }

                   for (i=0;i<m;i++) {
                              xx = xx + calculate[i];
                   }

                   alpha[0] = len1/xx;

                  for (i=1;i<m;i++)  {
                       alpha[i] = alpha[0] * calculate[i];
                  }


                  System.out.println("Calucalating the values of
alpha");
                  for (i=0;i<m;i++)  {
                         System.out.println(alpha[i]);
                   }
                  System.out.println("\n");

                        // caluclating processing time
                        double proc_1, proc_2, process_time;
                        proc_1 = cal.summation(sum_time_row, len2);
                        proc_2 = cal.summation(C,len2);
                        process_time = (double)
alpha[0]*len2*sum_time_row[0] + proc_1 + 2*proc_2*(len2-1);

                        System.out.println("The Processing Time : " +
"\t" + process_time);

                        // Speed Up caluculation

                        double speedupvalue, speed_up;
```

```
                            speedupvalue = len1*len2*sum_time_row[0];
                            speed_up = speedupvalue/process_time;
                            System.out.println("The Speed up equals: " +
speed_up);
            }


 }// End of main
}// End of class

/* This class will return the alpha values and the value is calculated
from the formula derivied from the
 * timing diagarm*/
class calcu
{
       public double calc_alph(float E1, float E2, float B1, float B2,
float C1){
              double alp;
              alp = (double) ( (B1*E1) - (2*C1) )/(B2*E2);
/* This is the condition checking if alpha negative*/
              if (alp < 0)
                  {
                 System.out.println("The value of alpha goes to
negitive: chck it");
                    System.exit(0);
              }
              return alp;
        }

        public double summation(float[] A,int len){
              double xx=0;
           for (int i =0;i<len;i++){
                xx = xx + A[i];
           }
               return xx;
        }
}
```

The 3-D graphs have been implemented in Matlab and the code is as shown below

For a constant link speed:

```
clc
close all
processors = [3,5,7,10,20,30,40,50,60,70,80,90,100];
length = [100,200,300,400,500,600,700,800,900,1000];
Time3  = [1.65 53.2 404 1700 5200  12900 27900 54500 98500 165000];
Time5  = [0.63, 20, 152, 641, 1950, 4860, 10500, 20400, 36900, 62500];
Time7= [0.34, 10.8, 81.7, 343, 1040, 2600, 5620, 10900, 19700, 33400];
Time10 = [0.212, 6.59, 49.5, 207, 632, 1570, 3390, 6600, 11800, 20100];
Time20 = [0.122, 3.73, 27.9, 116, 354, 879, 1890, 3680, 6630, 11200];
Time30 = [0.0782, 2.32, 17.1, 71.4, 216, 534, 1150, 2230, 4010, 6790];
Time40 = [0.0562, 1.6, 11.6, 47.9, 144, 355, 763, 1480, 2650, 4480];
Time50 = [0.0443, 1.2, 8.55, 34.8, 103, 254, 545, 1050, 1880, 3170];
Time60 = [0.0373, 0.962, 6.68, 26.8, 79.4, 193, 411, 793, 1410, 2380];
Time70 = [0.0328, 0.808 ,5.47 ,21.6, 63.5, 153, 325, 624, 1110,1864];
```

```
Time80 = [0.0298, 0.705, 4.66, 18.1, 52.7, 126, 266, 509, 903, 1511];
Time90 = [0.0276, 0.632, 4.08, 15.6, 45, 107, 224, 427, 755, 1250];
Time100 = [0.025, 0.58, 3.67, 13.8, 39.4, 93.3, 194, 367, 646, 1074];

Time = [Time3;Time5; Time7; Time10; Time20; Time30; Time40; Time50;
Time60; Time70; Time80; Time90; Time100];

figure;
surf(length,processors,Time);
xlabel('Length of the String');
ylabel('Number of Processors');
zlabel('Processing Time (sec) ');
title('For Constant Link Speed, Length of the String vs Number of
Processors vs Processing Time');



speed3 = [3.01, 3, 3, 3, 3, 3, 3, 3, 3, 3 ];
speed5 = [7.94, 7.97 ,7.98, 7.985, 7.988, 7.99, 7.991, 7.992, 7.993,
7.994];
speed7 =[15.57, 14.7, 14.87, 14.89, 14.91, 14.92, 14.93, 14.94, 14.95,
14.95];
speed10 = [23.57, 24.25, 24.49, 24.62, 24.69, 24.74, 24.78, 24.8,
24.82, 24.84];
speed20 = [40.92, 42.8, 43.5, 43.86, 44, 44.22, 44.33, 44.41, 44.48,
44.53 ];
speed30 = [63.87, 68.7, 70.68, 71.69, 72.32, 73.75, 73.06, 73.29,
73.48, 73.63];
speed40 = [88.8, 99.7, 104.22, 106.67, 108.22, 109.28, 110, 110.65,
111.11, 111.48];
speed50 = [112.74, 132.95, 141.94, 147, 150.24, 152.49, 154.14, 155.41,
156.42, 157.23];
speed60 = [133.99, 166.22, 181.73, 190.77, 196.69, 200.86, 203.97,
206.36, 208.72, 209.82];
speed70 = [152.09, 197.86, 221.72, 236.2, 245.9, 252.86, 258.09,
262.16, 265.43, 268.1];
speed80 = [167.36, 226.86, 260.51, 281.74, 296.34, 306.99, 315, 321.48,
326.64, 330.88];
speed90 = [180.99, 252.77, 297.11, 326.19, 346.69, 361.93, 373.6,
383.04, 390.65, 396.97];
speed100 = [199.2, 275.53, 330.94, 368.66, 395.93, 416.55, 432.7,
445.68, 456.34, 465.25];

figure;
Speedup = [speed3; speed5; speed7; speed10; speed20; speed30; speed40;
speed50; speed60; speed70; speed80; speed90; speed100];
surf(length,processors,Speedup);
xlabel('Length of the String');
ylabel('Number of Processors');
zlabel('Speed Up');
title('For Constant Link Speed, Length of the String vs Number of
Processors vs Speed Up');
```

For a variable link speed:

```
clc
```

```
close all
processors = [3,5,7,10,20,30,40,50,60,70,80,90,100];
length = [100,200,300,400,500,600,700,800,900,1000];
Time3 = [1.65, 53.19, 404.29, 1704.47, 5203.03, 12948.98, 27991.44,
54578.66, 98359.84, 166915.49 ];
Time5 = [0.629, 20.07, 152.24,  641.16, 1955.99, 4865.96, 10515.45,
20498.88,  36936.05,  62671.53 ];
Time7 = [0.342, 10.82, 81.79, 343.84, 1047.8, 2604.73, 5625.94,
10962.95, 19747.62,  33498.78 ];
Time10= [0.21, 6.59, 49.59, 207.94, 632.68, 1571.14, 3390.95, 6604.03,
11890.64, 20163.56 ];
Time20 = [0.122, 3.73, 27.93, 116.73, 354.45, 879.03, 1895.33, 3688.51,
6637.35, 11250.11 ];
Time30 = [0.0782, 2.32, 17.18, 71.4,  216.03, 534.39, 1150.15,
2235.23, 4017.89, 6804.34];
Time40 = [0.056,  1.6,  11.65, 47.99, 144.37, 355.75,  763.52,
1480.69, 2657.12, 4493.82];
Time50 = [0.044, 1.2, 8.55, 34.82, 103.99, 254.95, 545.14, 1054.18,
1887.49, 3186.41];
Time60 = [0.037, 0.96, 6.68, 26.83, 79.43, 193.55, 411.99, 793.92,
1417.58, 2387.78];
Time70 = [0.032, 0.8, 5.47, 21.67, 63.53, 153.75, 325.59, 624.93,
1112.3, 1868.71];
Time80 = [0.029, 0.7, 4.66, 18.17, 52.75, 126.64, 266.68, 509.63,
903.88, 1514.16];
Time90 = [0.027, 0.63, 4.08, 15.69, 45.06, 107.42, 224.87, 427.72,
755.75, 1262.08];
Time100 = [0.025, 0.58, 3.67, 13.88,  39.46, 93.3, 194.2, 367.61,
646.98, 1076.88];

Time = [Time3;Time5; Time7; Time10; Time20; Time30; Time40; Time50;
Time60; Time70; Time80; Time90; Time100];

figure;
surf(length,processors,Time);
xlabel('Length of the String');
ylabel('Number of Processors');
zlabel('Processing Time (sec)');
title('For Variable Link Speed, Length of the String vs Number of
Processors vs Processing Time');




speed3 = [3.01, 3, 3, 3, 3, 3, 3, 3, 3, 3 ];
speed5 = [7.94, 7.97, 7.98, 7.98, 7.98, 7.99, 7.99, 7.99, 7.99, 7.99];
speed7 =[14.57, 18.78, 14.85, 14.89, 14.91, 14.92, 14.93, 14.94, 14.95,
14.95];
speed10 = [23.56, 24.25, 24.49, 24.62, 24.69, 24.74, 24.78, 24.8,
24.82, 24.84];
speed20 = [40.92, 42.8, 43.5, 43.85, 44.08, 44.22, 44.33, 48.41, 44.48,
44.53];
speed30 = [63.86, 68.77, 70.68, 71.69, 72.32, 72.75, 73.06, 73.29,
73.48, 73.62];
speed40 = [88.75, 99.7, 104.22, 106.67, 108.22, 109.28, 110.06, 110.65,
111.11, 111.4];
```

```
speed50 = [112.7, 132.95, 141.94, 147, 150.24, 152.49, 154.14, 155.41,
156.42, 157.23];
speed60 = [133.95, 166.22, 181.73, 190.77, 196.69, 200.86, 203.97,
206.36, 208.27, 209.81];
speed70 = [152.03, 197.85, 221.72, 236.19, 245.9, 252.86, 258.09,
262.16, 265.43, 268.09];
speed80 = [167.29, 226.85, 260.5, 281.74, 296.34, 306.99, 315.1,
321.48, 326.63, 330.87];
speed90 = [180.9, 252.76, 297.1, 326.18, 346.69, 361.93, 373.69,
383.04, 390.69, 396.96];
speed100 = [199.1, 275.52, 330.94, 368.65, 395.93, 416.55, 432.7,
445.68, 456.34, 465.23];

figure;
Speedup = [speed3; speed5; speed7; speed10; speed20; speed30; speed40;
speed50; speed60; speed70; speed80; speed90; speed100];
surf(length,processors,Speedup);
xlabel('Length of the String');
ylabel('Number of Processors');
zlabel('Speed Up');
title('For Variable Link Speed, Length of the String vs Number of
Processors vs Speed Up');
```

This gives the coding for the Initial results of the thesis. The graphs for the divisible load theory and multi-installment strategy have been generated using MATLAB and is as shown below.

Code for generating the graph for Divisible load theory

```
clc
clear all
z = [2 0.5 5];
omega = [2 1 1 2];
tcp = 1;
tcm = 1;
number_of_childs = length(z);

sigma = zeros(1,(length(z)));
for i = 1 : length(z)
    [sigma(i)] = compute_sigma(z(i),tcm,omega(i+1),tcp);
end

sub_alpha(1,:) = [ 0 0 0 ];
for i = 2 :length(z)+1
    [sub_alpha(i,:)] = compute_sub_alpha(sigma(i-1),number_of_childs);
end

alpha = zeros(1,length(sigma)+1);
for i = 2 : length(sigma)+ 1
    [alpha(i)] = compute_alpha(sub_alpha(i,:),sigma(i-
1),number_of_childs);
end

alpha(1) = 1 - sum(alpha)
```

```
function [alpha] =
compute_alpha(sub_alpha,sigma,number_of_installments)

xx = 0;
for i = 1 : number_of_installments
    xx = xx + sigma^(i-1);
end

alpha = sub_alpha(1)*xx;


function [sigma] = compute_sigma(z,tcm,omega,tcp)

sigma = (omega *tcp) / (z*tcm);
```

For multi installment strategy

```
clc
clear all
close all

n = 100;
total_computational_time(1) = 0;
for i = 2 : n
    z = ones(1,i);
    omega = ones(1,i+1);
    tcm = 1;
    tcp = 0.5;
    number_of_installments = 3;
    [total_computational_time(i)] =
multi_installment(z,omega,tcp,tcm,number_of_installments);
end

figure
plot(2:n,total_computational_time(:,(2:n)))
%...........................................................
...
 %   ....................


n = 7;
number_of_installments = 10;
total_computational_time = zeros(1,number_of_installments);
for i = 3 : number_of_installments
    z = ones(1,n);
    omega = ones(1,n+1);
    tcm = 1;
    tcp = 0.5;

    [total_computational_time(i)] =
multi_installment(z,omega,tcp,tcm,i);
end
figure
```

```
plot(3:number_of_installments,total_computational_time(:,(3:number_of_i
nstallments)))

%.....................................................................
....
 %   ..................

% z = [2 0.5 5 ];
% omega = [2 1 1 2];
% tcp = 1;
% tcm = 1;
% number_of_installments = 3;
% [total_computational_time] =
multi_installment(z,omega,tcp,tcm,number_of_installments);


function [sub_alpha] = compute_sub_alpha(sigma,number_of_installments)

sub_alpha(1) = 1;

for i = 2 : number_of_installments
    sub_alpha(i) = sub_alpha(i-1)*sigma;
end
```

VITA

Type Full Name Here

Candidate for the Degree of

Master of Science

Thesis: LOAD SCHEDULING FOR BIOINFORMATICS APPLICATIONS IN LARGE SCALE

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science or Arts in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December 2008.

.

Completed the requirements for the Bachelor of Technology in Computer Science at Jawaharlal Nehru Technological University, Hyderabad, India in May 2005.

Name: Sudha Gunturu Date of Degree: December 2008

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: LOAD SCHEDULING FOR BIOINFORMATICS APPLICATIONS IN LARGE SCALE NETWORKS

Pages in Study: 65 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

A load scheduling strategy with near-optimal processing time is designed to explore the computational characteristics of DNA sequence alignment algorithms, specifically, the Needleman-Wunsch Algorithm. Following the divisible load scheduling theory, we design an efficient load scheduling strategy to manage such bioinformatics applications in a large-scale network so that the overall processing time of the sequencing tasks is minimized. The row-wise and column-wise partitioning of the workload is adopted in the scheduling strategy. In this study, the load distribution depends on the length of the sequence and number of processors in the network and, the total processing time is also affected by communication link speed. We considered several cases in our study by varying the sequences, communication and computation speeds, and number of processors.

Findings and Conclusions:

Through simulation and numerical analysis, this study demonstrates that for a constant sequence length as the numbers of processors increase in the network the processing time for the job decreases and minimum overall processing time is achieved.

ADVISER'S APPROVAL: Xiaolin (Andy) Li