a455788

High Performance Computing for Computational Biology

Du Zhihua

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in FULFILMENT of the requirement for the degree of Doctor of Philosophy

2005

QH
324.2
D812
2005

HIGH PERFORMANCE COMPUTING FOR COMPUTATIONAL BIOLOGY

By

Du Zhihua

This report is presented for Degree of
DOCTOR OF PHILOSOPHY
AT
NANYANG TECHNOLOGICAL UNIVERSITY
NANYANG AVENUE,SINGAPORE,639798
OCTORBOR 2005

NANYANG TECHNOLOGICAL UNIVERSITY
DEPARTMENT OF
SCHOOL OF COMPUTER ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Division of Information Systems for acceptance a thesis entitled "**High performance computing for computational biology**" by **Du Zhihua** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: Octorbor 2005

External Examiner:	
External Examiner:	
Internal Examiner:	
Research Supervisor:	Lin Feng
Examing Committee:	

NANYANG TECHNOLOGICAL UNIVERSITY

Date: **Octorbor 2005**

Author:	**Du Zhihua**
Title:	**High performance computing for computational biology**
Department:	**School of Computer Engineering**

Degree: **Ph.D.** Convocation: Year: **2006**

Table of Contents

List of Figures

List of Tables

Abstract

Both sequence comparison and phylogenetic inference are of great importance to biologists; and these problems are fundamentally interdependent. Most methods for phylogenetic inference use a given sequence alignment as an input, and efficient multiple alignment procedures often take advantage of a phylogenetic relationship of the sequences. However, the algorithms used in these problems are very computationally demanding. Also, the huge increase in size of publicly available genomic data has meant that many common tasks in bioinformatics are not possible to complete in a reasonable amount of time on a single processor. For example, inferring phylogenetic trees is an enormously difficult problem because of the huge number of potential alternative tree topologies, for that number grows exponentially. For a large number of taxa, it is not possible to perform an exhaustive search of the tree space. It is therefore clear that high performance computing, which can speed up the process of sequence analysis without sacrifice the quality of the results, is necessary. The role of high performance computing has also been credited in being the only solution for two of the grand challenges in biology: the understanding of evolution and the basic structure and function of proteins (D.A.Bader 2004).

This PhD project aims to highlight the potential and effectiveness of high performance computing as a viable option for mining large datasets of genome sequences. My main contributions include proposals of parallel architecture and mechanisms which empower the algorithms of sequence comparison and phylogenetic analysis. More importantly, these solutions make it possible for biologists to analyze the datasets that were previously considered too large, often leading to memory overflow or prohibitively long time for computation.

For sequence comparison, two new algorithms are presented for pairwise alignment and multiple alignment respectively to gain parallel computing power at low cost. The first one is a "block-based wavefront" algorithm developed to take advantage of dynamic programming and parallel computing to produce optimal pairwise alignments in a reasonable time frame. The parallel alignment is executed in a block-based wavefront where computing nodes will calculate the blocks along anti-diagonals in parallel. The novelty of this algorithm is a compromise of the workload of each process and the number of communications required, which makes the communication-to-computation ratio drop dramatically. Secondly, a fast and practicable algorithm for multiple sequence alignment, known as PMSA, is designed. The proposed algorithm effectively parallelized all the three stages of the ClustalW algorithm, which outperforms previous parallel schemes. With these improvements, the execution time of sequence comparison can be greatly reduced, and it is also possible to apply the proposed algorithms in large-scale sequence projects that were previously impossible.

In order to improve the topological accuracy of phylogenetic tree, a parallel divide-and-conquer model (pPhylo) is designed, which performs a more complete search of the tree space within limited time. This model is flexible; it can reconstruct the large Maximum Parsimony(MP) and Maximum Likelihood(ML) trees comprising up to 10,000 organisms and leads to significant improvements in run-time. Furthermore, the trees computed by the proposed model are consistently better than the previously known fastest and most accurate programs for MP and ML respectively. The model includes four key steps: dividing a large tree into subtrees, optimizing the subtrees in parallel, merging the computed subtrees in the correct order imposed by the decomposition and finally rearranging the global tree. Because ML is a harder problem than MP, a parallel global search method applied on an ML tree is also designed. The global search method further improves the accuracy of the supertree and can also find global configurations that were not found by operations on smaller local subsets.

In related work, a pattern-constrained multiple sequence alignment algorithm (PCMSA) is presented to improve the accuracy of multiple sequence alignment. It begins by finding an optimal constrained pairwise alignment for each pair of sequences using a new equation. Based on these results, a center sequence is defined so that the sum of the pattern-constrained scores to other *k-1* sequences is minimized (k is the number of sequences to be aligned). Finally, a multiple alignment is constructed by the alignment of the center sequence with the rest *k-1* sequences progressively. In order to form a multiple sequence alignment, spaces are inserted into each pre-aligned pairwise sequence. The significance lies on its capability of aligning the sequences sharing the same patterns. It effectively brings the information available in the existing pattern databases into multiple sequence alignment. It is also proven that the similarity score derived from the PCMSA has the worst-case guarantee on the quality of the alignment, a bound of approximation ratio of $\frac{k}{2(k-1)}$ to the similarity score of optimal alignment.

My future work focuses on the improvement of the algorithms which appear in the popular research fields of computational biology; such as computation of the "Tree of life", protein structure alignment, multiple genome alignment, and analysis of virus evolution. I also want to integrate the grid computing technology with our current architecture, to implement all kinds of sequence analysis applications with improved efficiency.

Acknowledgements

Many people have helped this thesis work in one way or another.

First of all, I would like to express my sincere gratitude to my supervisor, Dr. Lin Feng, for his many suggestions and invaluable guidance during my research. I have benefited from his academic advices and research experience, as well as the trust and freedom he granted me for my research. Without his help, my research would not reach this stage. I am also grateful to Dr. Usman Roshan of New Jersey Institute of Technology for numerous helpful discussions as well as his invaluable guidance, expertise in research and enormous support. My appreciation for his mentorship goes beyond my words. Special thanks goes to Dr. Alexandros Stamatakis of the Institute of Computer Science in Heraklion, Greece, for inspiring collaboration, and interesting discussions.

I gratefully acknowledge all my friends for being such excellent company throughout my life.

Many thanks should go to all the members of Bioinformatics Research Center for their friendship and creating a very pleasant research environment.

Finally, great thanks to School of Computer Engineering, Nanyang Technological University for offering me the award of Research Scholarship and the first-class research facilities.

This work is dedicated to my family for their constant support over all these years.

Author's Publications

1. Du, Z. H. and Lin, F., A hierarchical clustering algorithm for MIMD architecture, Computational Biology and Chemistry, 28, 2004, 417-419.

2. Du, Z. H., Lin, F., and Usman, W.R., Reconstruction of Large Phylogenetic Tree: A Parallel Approach, Computational Biology and Chemistry, 29(4), 2005, 273-280.

3. Du, Z. H. and Lin, F., Pattern-constrained Multiple Polypeptide Sequence Alignment, Computational Biology and Chemistry, 29(4), 2005, 303-307.

4. Du, Z. H. and Lin, F., A novel parallelization approach for hierarchical clustering, Parallel Computing, 31, 2005, 523-527.

5. Du, Z. H., Lin, F. and Bertil, S., Accomplishments and Challenges in High Performance Computing for Computational Biology, Current Bioinformatics (Accepted for publication).

6. Du, Z. H. and Lin, F., pNJTree: A Parallel Program for Reconstruction of Neighbor-Joining Tree and Its Application in ClustalW, Parallel Computing, (Accepted for publication).

7. Du, Z. H., Alexandros, S., Lin, F., Usman, W. R. and Luay, N., Parallel Divide-and-Conquer Phylogeny Reconstruction by Maximum Likelihood, The 2005 International Conference on High Performance Computing and Communications, Sorrento (Naples) , ITALY – September 21-24, 2005.

8. Du, Z. H. and Lin, F., Using Blocks+ Database in Needleman-Wunsch Algorithm, The 23rd International Conference of the North American Fuzzy Information Processing Society (NAFIPS 2004), Canada, 28-31 June 2004.

9. Du, Z. H. and Lin, F., Improvement of the Needleman-Wunsch Algorithm, Fourth International Conference on Rough Sets and Current Trends in Computing 2004 (RSCTC 2004), Sweden, 1-5 July 2004.

10. Du, Z. H. and Lin, F., pClustalW: A Deployment of Parallel ClustalW, The International Conference on Bioinformatics (Incob2003), Penang, Malaysia, September, 2003.

11. Du, Z. H. and Lin, F., Parallel Computation for Multiple Sequence Alignments, The Fourth International Conference on Information, Communication & Signal Processing and Fourth PacificRim Conference on Multimedia(ICICSPCM 2003), Singapore, December, 2003

12. Lin, F., Du, Z. H. and Qi, Y. T., HPTC for Sequence Analyses in Bioinformatics, HP-CAST 2004, Australia, 28-31 March 2004.

Chapter 1

Introduction

The first chapter starts with an introduction to the relevant biological background. Following that, it presents the motivation behind conducting research in high performance computing for computational biology ranging from pairwise sequence alignment, multiple sequence alignment, to phylogenetic reconstruction. Finally, it summarizes the major contributions of the work, and describes the structure of the thesis.

1.1 Cellular biology and molecular genetics

Two important molecules in all living cells are Nucleic Acids (DNA and RNA) and proteins. The DNA, RNA and proteins of an organism are all linear chains composed of smaller molecules. Each of these macromolecules stores information that provide an insight into an organism's heredity and function. They are assembled from a fixed alphabet of well-understood chemicals. DNA is made up of four chemical bases: adenine (A) and guanine (G), which are called purines, and cytosine (C) and thymine (T), referred to as pyrimidines. Each base has a slightly different composition, or combination of oxygen, carbon, nitrogen and hydrogen. RNA is a temporary medium of genetic information because it uses specific regions of DNA as template produced in a process called transcription. RNA is very similar to DNA, but has the Thymine bases replaced by Uracil. Proteins are composed of the 20 amino acids, denoted by A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, and V in sequence analysis software.

The Central Dogma, a fundamental principle of molecular biology, states that genetic information flows from DNA to RNA to proteins. Using the genes of an organism as templates, RNA is produced. The RNA can then be used as instructions for producing proteins. The genetic code that resides in DNA is passed on from generation to generation. Figure 1.1 shows the procedure.

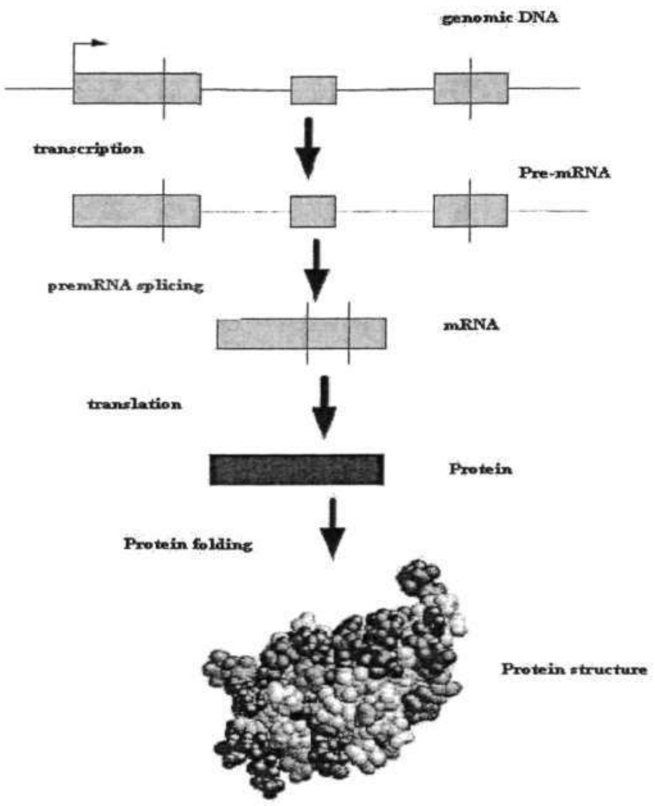


Figure 1.1: Transcription and Translation

Although DNA is a relatively stable molecule, it may be damaged by certain chemicals or UV light. Additionally, errors are introduced to DNA through the process of copying in spite of proof reading machinery. Only point mutations affecting a single spot of DNA are commonly considered, although more complex mutations like rearrangements, duplications, and inversions are possible at the chromosome level. Point mutations take the forms of substitution, deletion, or insertion. Substitution is a change of one nucleotide in DNA sequences and insertion or deletion is an addition or a removal of one or more bases from DNA sequences. Mutations are responsible for inherited disorders and diseases. However, mutations would also create new species and adapt existing ones to changing environmental conditions because they are the source of the phenotype variation on which natural selection acts.

Due to the huge volume of data flooding from biology including genome projects, proteomics, protein structure determination and the rapid expansion in digitalization of patient biomedical data, the path has been cleared for study of biological sequence data.

1.2 Sequence analysis

Analysis of biological sequence data in worldwide databases is the most commonly performed task in bioinformatics, in which the commonly used analysis techniques are sequence comparison and phylogenetic analysis.

The initial step in any phylogenetic analysis is to establish homology statements across taxa. Sequence comparison establishes the degree of similarity between two or more sequences. The more similar two gene sequences are to each other, the more closely the organisms are related. Through protein sequence alignment, it can be learned about the functionality of a protein without performing experiments: two proteins having 25% or above sequence identity will have similar structure fold, thus, a similar function (Rost 1999).

An example is the sequence alignment between cancer and uncontrolled cell growth (Doolittle, Hunkapiller, Hood, Devare, Robbins, Aaronson & Antoniads 1983). In this discovery, the sequence of cancer associated alignment with the sequence of the protein which had already been known as influencing the cell growth. The result proves the connection between cancer and cellular growth because the correlation between the two sequences is high.

Multiple sequence alignment determines the position and nature of conserved regions in each member of the group. It carries more information than a pairwise one, as a protein can be matched against a family of proteins instead of only against another one. It is also a common input used by most methods for phylogenetic inference. Conserved segments of multiple sequence alignment usually correspond to structurally and functionally important parts of proteins.

Evolution is a central concept in biology. A phylogenetic tree of a family of related nucleic acid or protein sequences is a phylogenetic hypothesis of how the family might have been derived during evolution. The relationship of different sequences is reflected by the branching; Two sequences that are very much alike will be located in neighboring outside branches and will join a common branch beneath them. Phylogenetic trees play an important role in answering many biological questions. For instance, when a gene is found in an organism or group of organisms, it is helpful to predict which ones might have an equivalent function on the basis of the phylgonetic analysis. It is also used to follow rapid changing species, such as a virus. An example for an evolutionary tree of the monkeys and the *Homo Sapiens* is provided in Figure 1.2.

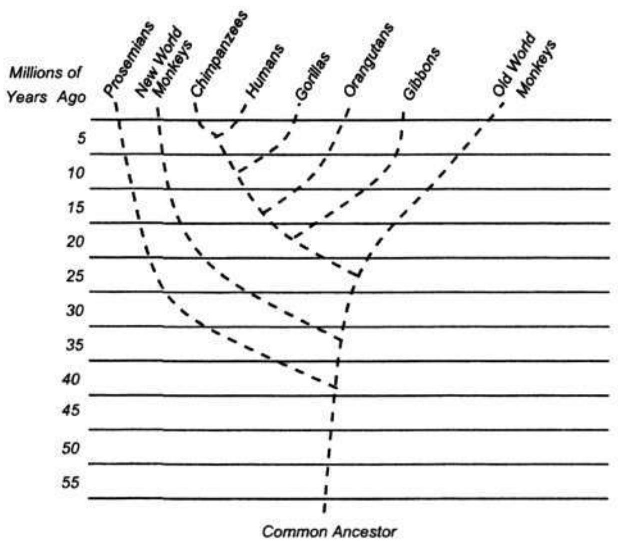


Figure 1.2: Phylogenetic tree representing the evolutionary relationship between monkeys and *Homo Sapiens*

Sequence comparison and phylogenetic inference are fundamentally interdependent because most methods for phylogenetic inference use a given multiple sequence alignment as an input. Another reason is that efficient multiple alignment procedures often take advantage of a phylogenetic relationship of the members to be aligned. Moreover, correction for biased representations (or weighting) of sampled sequences is possible only with a knowledge about interrelationship among the sequences. Therefore, ignoring this fact leads to biased and overconfident estimations. Whether the main interest is in sequence alignment or phylogeny, a major goal of computational biology is the co-estimation of both. Development of effective and efficient algorithms for these tasks has attracted researchers all over the world.

1.3 Rationale behind the research

During the last decades, research in the fields of molecular biology and biochemistry has provided the scientific community with huge amounts of sequence data. The National Center for Biotechnology Information's GenBank has more than 37 million sequence records, and this collection has nearly doubled in size every year for decades. This exponential growth rate can be observed in Figure 1.3 (GenBank growth data is available at http:// WWW.NCBI.NLM.NIH.GOV/GENBANK/ GENBANKSTATS.HTML).

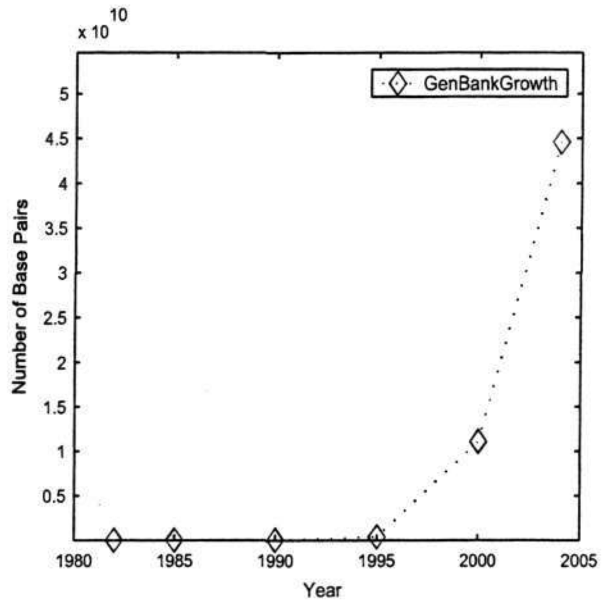


Figure 1.3: GenBank growth

As the number of DNA and protein sequences in databases grows it is increasingly important to be able to create sequence alignments and phylogenetic trees for very long or very large numbers of sequences.

The algorithms of pairwise alignment based on dynamic programming provide optimal solutions; however, they are computationally expensive, especially when comparing whole genome. The computational load often exceeds the capacity of most computing systems.

Multiple sequence alignment problems are NP-complete (An exact solution is conjectured by computer scientists to not be solvable in polynomial time, that is, an NP-complete problem requires more steps than can be grounded by a polynomial.). To solve this problem, some approximating and heuristic methods have been introduced. Some of these, called progressive methods, are developed to find near optimal solutions within reasonable lengths of time. However, they also suffer from high computational complexity when large datasets are aligned. Given that progressive methods require $O(n^2)$ steps, where n is the number of sequences to be aligned, it is not surprising that these methods take many hours to run. For instance, the alignment of a few hundred of protein sequences using the ClustalW, one of the commonly used MSA tools, requires several hours on a state-of-the-art workstation.

The amount of sequence data available to reconstruct the evolutionary history of genes and species has also increased dramatically. As a consequence, the size of phylogenetic analysis has grown as well. Additionally, inferring phylogenetic trees is an enormously difficult problem for the large number of the potential alternative tree topologies, and this number grows exponentially with the number of taxa. Constructing the "Tree of Life" is a collaborative effort by various research groups such as CIPRES (http://www.phylo.org) and ATOL (http://tolweb.org). The current techniques for reconstructing phylognies are heuristic for distance-based methods, Maximum Parsimony (MP) and Maximum Likelihood (ML) methods. The distance-based methods are fast but have high error on large datasets. MP and ML are NP-complete or even more complex. Moreover, the computation of the likelihood value for one single potential tree topology is computationally intensive. Methods that can quickly reconstruct MP and ML trees are of great benefit to the biological community.

Increasing the speed and accuracy for sequence comparison and phylogenetic inference is an imperative. While serial versions of some algorithms can use shallow computation for a reasonable result in tolerant time, serious research demands deep computing in an acceptable time frame. High performance computing (HPC) is one of the most promising methods to absorb and process the amount of data available due to its potential for delivering high computational power. Two or more processors can be used simultaneously, in parallel processing, to divide and conquer tasks that would overwhelm a single processor. It is clear that development of high performance computing technologies is necessary for tackling various bioinformatics problems. Key research issues include dividing mechanisms to break a big problem into subproblems, and to figure out how the subproblems relate to each other. The complexity of a problem is measured by the minimum number of messages that need to be sent to solve the problem and getting the maximum performance out of all the processors all the time. For example, the computations of dynamic programming based pairwise alignment are triggered by the flow of the data from neighboring elements. Elements which can be computed independently of each other are located in the anti-diagonal which “moves” across a matrix as the computation proceeds. However, such partitioning requires too much communication; it introduces inefficiencies by congesting the communication network. Therefore, a new method with good computational load balance and minimal communication is required.

1.4 Major contributions

The algorithms of sequence comparison and phylogeny reconstruction are computationally expensive. In addition, the very large databases used in computational biology give rise to serious algorithmic problems when exploring and analyzing the data contained in these databases.

The main contributions of this research are a series of algorithmic and technical solutions which empower the algorithms of pairwise and multiple sequence alignment and phylogenetic tree construction. More importantly, these solutions make it possible for biologists to analyze datasets that were previously considered too large, often leading to memory overflow or prohibitively long time for computation.

First, a new parallel algorithm, called "block-based wavefront", is presented to produce optimal pairwise alignment with reliable output and reasonable cost. The proposed algorithm takes advantage of dynamic programming and parallel computing to produce optimal results in reasonable time. It distributes the computation of a similarity matrix along block-based anti-diagonals to multiple processors, because the computation of these blocks is independent. The novelty is a compromise of the workload of each processor and the number of communications required. Even though this algorithm will increase some serial computations when computing the elements within a block, it decreases the communication load dramatically, and globally the proposed algorithm achieves a near linear speedup with high scalability on long-scale datasets.

Secondly, a fast and practicable algorithm for multiple sequence alignment is proposed. It effectively parallelized all the three stages of the ClustalW algorithm. For the first stage, the calculation of different sequence pairs is completely independent;

there is a very high degree of parallelism. For the second stage, the computation of guiding trees is parallelized. And for the last stage, we calculate profile scores in parallel in the iterative loops. Experimental results show that by distributing subroutines to multiple processors, the execution time of the ClustalW program can be significantly reduced, which outperforms previous parallel schemes. With these improvements, it is possible to apply the proposed algorithm in large-scale sequence projects that were previously impossible.

Thirdly, a novel model is developed which has been proven to be the most adequate and accurate for the inference of huge, complex trees. The proposed model is flexible enough to reconstruct both Maximum Parsimony(MP) and Maximum Likelihood(ML) trees and leads to significant improvements in run-time. It also performs a more complete search of the tree space within limited time. Therefore, the parsimony and likelihood scores of trees computed by the proposed model are consistently better than the previously known fastest and most accurate programs for MP and ML respectively.

Finally, a pattern-constrained multiple sequence alignment algorithm, PCMSA, is designed to be applied to multiple polypeptide sequence alignments. It uses pattern-constrained pairwise alignments with further assembling of these "partial" alignments into an approximate alignment of multiple sequences. It effectively brings the information available in existing pattern databases into multiple sequence alignments. It is also able to guarantee that the generated alignment satisfies the pre-defined constraints that particular patterns should be aligned together. In addition, it has the worst-case guarantee on the quality of the alignment. For example, running on the six aminoacyl-tRNA synthetase sequences, the proposed algorithm aligns the aminoacyl-transfer RNA synthetases class-I signature shared by these sequences together while the ClustalW fails to do.

I integrated these solutions into a parallel sequence analysis system on a customized multi-node cluster, which significantly improves computational performance on sequence analysis problems in computational biology. In this system, a calculation which used to take hours is now carried out in minutes, and calculations that took a week can be completed less than several hours. Parts of this PhD study have been published in six journal papers and six peer-reviewed conference papers.

1.5 Structure of the thesis

The rest of this thesis is organized as follows:

Chapter 2 presents a comprehensive survey spanning pairwise sequence alignment, multiple sequence alignment and phylogenetic tree inference; addresses problem complexity, and outlines HPC architectures.

Chapter 3 describes a dedicated system to deliver high performance computing for the problems of sequence analysis. The hardware architecture and software components are studied. In the following chapters 4-6, the details of the proposed algorithms are addressed.

Chapters 4 presents two new algorithms, for the problems of sequence comparison. The first one is focuses on dynamic programming based pairwise alignment, which produces an optimal result at the cost of huge requirements in both memory and time. The other is a fast and practicable algorithm for multiple sequence alignment.

In Chapter 5, a parallel divide-and-conquer model, pPhylo, is designed to reconstruct phylogenetic trees, which is flexible enough to work with either MP or ML methods.

Previous solutions to MSA use a substitution matrix, but they do not incorporate the knowledge of the sequences being aligned. In Chapter 6, a novel algorithm, pattern-constrained multiple sequence alignment, is presented to guarantee that the generated alignment satisfies the pre-defined constraints that some particular patterns should be aligned together.

Finally, Chapter 7 concludes this research project and addresses important aspects of future work.

Chapter 2

Research Background and Literature Survey

This chapter presents a detailed survey on the problems and methods in sequence comparison and phylogenetic analysis, addresses the complexity of the problems and outlines the architectural development of HPC systems.

2.1 Methods for biological sequence similarity comparison

2.1.1 Sequence comparison

Sequence comparison is used to tell whether two or more sequences are related and give an impression how close their relationship is in terms of sequence similarity. There are three possibilities of pairs of opposite symbols to evaluate the difference between two sequences: (i) identity, (ii) substitution or mismatch, (iii) insertion or deletion. When doing sequence alignment, identical or similar characters are put in the same column, and nonidentical characters can either be placed in the same column as a mismatch or opposite a gap in the other sequence. The goal of sequence alignment is to find an optimal alignment of sequences by bringing as many identical or similar characters as possible.

Considering the following pair of DNA sequences: **GAAGCAAT** and **GAC-CAAT**. When they are aligned one above the other:

```
G    A    A    G    C    A    A    T
G    A    C    -    C    A    A    T
```

The only differences are a change from A to C in the third position and an extra G in the first sequence. It is noted that a gap, marked with "-", is introduced in order to align the bases before and after the gap perfectly.

2.1.2 Scoring of alignments

In sequence alignment, it is necessary to use a scoring scheme to reflect the bio-chemical properties that influence the relative replaceability of amino acid or nu-cleic acid sequences in an evolutionary scenario. Some amino acids have higher matching scores than others, and some have mismatch scores as well due to their evolution and chemical properties. The degree between two letters can be repre-sented in a substitution matrix. Examples of substitution matrices, known as BLO-SUM50, BLOUSM62 (S.Henikoff & J.G.Henikoff 1992) and PAM250 (M.Dayhoff, R.M.Schwartz & B.C.Orcutt 1978), are useful for sequence alignment because of each matrix giving the changes expected for a given period of evolutionary time.

A widely used scoring matrix is given below. It is extracted from probabilities. Let a and b be two sequences, a_i and b_j be the ith symbol in a and jth symbol in b, R be a random model, w be the probability that a letter occurs in a sequence independently. The probability of the two sequences is:

$$P(a,b|R) = \prod_i w_{a_i} \prod_j w_{b_j} \tag{2.1.1}$$

Let M be a match model, z be the probability of aligned pairs of symbols, then the whole alignment probability is:

$$P(a,b|M) = \prod_i z_{a_i,b_i} \tag{2.1.2}$$

The ratio of these two likelihood is:

$$\frac{P(a,b|M)}{P(a,b|R)} = \frac{\prod_i z_{a_i,b_i}}{\prod_i w_{a_i} \prod_j w_{b_j}} = \prod_i \frac{z_{a_i,b_i}}{w_{a_i} w_{b_j}} \tag{2.1.3}$$

In order to arrive at an additive scoring system, we take the logarithm of this ratio,

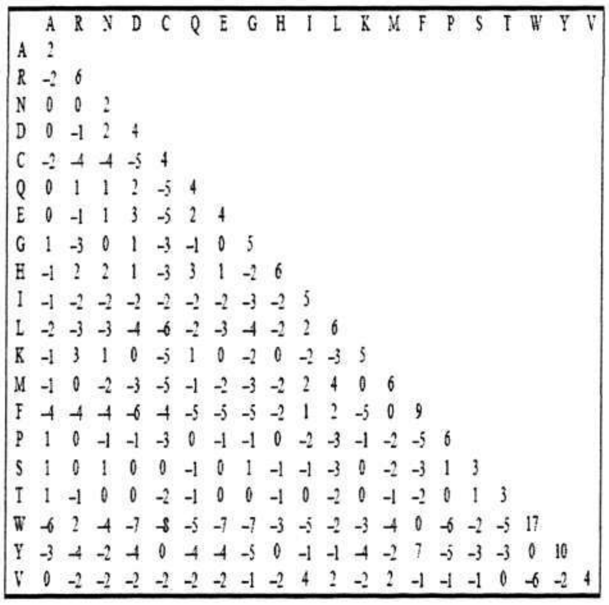
$$s(x,y) = log(\frac{z_{xy}}{w_x w_y}) \tag{2.1.4}$$

$s(x,y)$ is the log-likelihood ratio of the residue pair (x,y) occurring as a really valid aligned pair against a random pair. The alignment score T between sequences a and b is the sum of individual scores $s(x,y)$ for each aligned pair of residues.

$$T = \sum_i s(a_i, b_i) \tag{2.1.5}$$

The Dayhoff PAM matrix, shown in Figure 2.1, estimated 1572 changes in 71 groups of protein sequences that were at least 85% similar. These changes were tabulated in a 20×20 matrix, which gave an individual score s(x,y) for residue x in sequence a and y in sequence b.

The next point is gap penalty. For the alignment in the example above, there are six columns with identical characters, one column with distinct characters, and one column with a gap. Given that if a column has two identical characters, it is valued



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	2																			
R	-2	6																		
N	0	0	2																	
D	0	-1	2	4																
C	-2	-4	-4	-5	4															
Q	0	1	1	2	-5	4														
E	0	-1	1	3	-5	2	4													
G	1	-3	0	1	-3	-1	0	5												
H	-1	2	2	1	-3	3	1	-2	6											
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	5										
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
F	-4	-4	-4	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
P	1	0	-1	-1	-3	0	-1	-1	0	-2	-3	-1	-2	-5	6					
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	3				
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-2	0	1	3			
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

Figure 2.1: PAM 250 substitution matrix

+5 as a match, different characters valued -1 as a mismatch and a gap valued as -5. The cost can be computed in a straight way: 5 + 5 + (-1) + (-5) + 5 + 5 + 5 + 5 = 24.

However, the simple gap penalty model in which every gap which occurred has the same cost is not accurate enough. In order to obtain the best possible alignment between two sequences, an affine gap penalty model (O.Gotoh 1990) is introduced to define a cost function. In practice, the affine gap penalty model, Q_x, is defined as the following equation:

$$Q_x = d + e \times x \tag{2.1.6}$$

In this equation, d is an opening gap penalty for any gap, e is an extension gap penalty for each element in a gap and x is the length of gaps. The rules of thumb for gap penalties are that an opening gap penalty should be 2-3 times larger than the most negative value in the substitution matrix that is being used and an extension gap penalty should be 0.1 to 0.3 times the value of the opening gap penalty.

2.1.3 Algorithms for pairwise alignment

Pairwise alignment aims to find the best match between two DNA or protein sequences. In generally, two categories of methods have been recognized. The first category is the dynamic programming based technique, such as the Needleman-Wunsch algorithm (S.B.Needleman & C.D.Wunsch 1970) and the Smith-Waterman algorithm (Smith & Waterman 1981*b*) (Smith & Waterman 1981*a*). Dynamic programming methods assure the optimal alignment by exploring all possible alignments and choosing the best. It does this by reading in a scoring matrix that contains values for every possible residue or nucleotide match. It finds an alignment with the maximum possible score where the score of an alignment is equal to the sum of the matches taken from the scoring matrix.

The key idea is that the best alignment that ends at the positions of a given pair in two sequences is the best alignment previous to the two positions plus the score for aligning the two positions. For two sequences a=$a_1a_2...a_m$ and b=$b_1b_2...b_n$, the scoring relation is defined as the following:

$$F[i][j] = max\begin{cases} F[i-1][j-1] + s(a_ib_j) \\ F[i-1][j] - g \\ F[i][j-1] - g \end{cases} \tag{2.1.7}$$

Where $F[i,j]$ is the score at position i in the sequence a and position j in the sequence b; $s(a_ib_j)$ is the score for aligning the characters at positions i and j; g is a gap penalty. This relation defines a table F in terms of sequences a and b; which are of length m and n. Once this table is filled appropriately; $F[m, n]$ contains the optimal alignment score overall. By tracing from this element to $F[0, 0]$ and recording the steps, an optimal alignment of the two sequences is constructed.

FASTA (Pearson & Lipman 1988) and BLAST (Altschul, Gish, Miller, Myers & Lipman 1990) (Gish & States 1993) are based on the secondary category: heuristic sequence comparison. Heuristic methods can only provide sub-optimal solutions in which some good answers may be left out by trading speed for precision. They are widely used for searching large biological databases.

The FASTA algorithm sets a size k for k-tuple subwords. It proceeds through the following four steps to determine an alignment score:

1. Identify regions shared by the two sequences with k consecutive matching.
2. Re-scan the ten regions with the highest density of identities using a scoring matrix. Trim the ends of the regions to include only those residues contributing to the highest score. Each region is a partial alignment without gaps.
3. The best scoring initial regions are joined if their combined score is greater than a threshold. This initial score is used to rank the database sequences.
4. The optimized alignment is calculated around the highest scoring initial regions using a modified version of the Smith-Waterman algorithm.

BLAST (Basic Local Alignment Search Tool) is a heuristic method used to find the highest scoring locally optimal alignments, known as high-scoring segment pairs (HSP), between a query sequence and a database. The three steps involved in the BLAST algorithm are shown as following:

1. Given a length parameter w, for w-length substrings(words) in the query sequence, identify all of the substitutions of each word that have a similarity score greater than a threshold score T. These words are stored as an expanded word list.

2. Use the expanded list to identify all of the matching words in sequences of the database.

3. If two matches are found, extend each match both forwards and backwards using a scoring matrix, allowing gaps, to produce a score that is higher than a threshold score(the alignment is a HSP). Save all of the high scoring regions shared by the query sequence and each library sequence. The final gapped alignments are reported by the program.

In summary, algorithms for pairwise alignment can be solved with time complexity of $O(m \times n)$ by following dynamic programming, where m and n are the length of the sequences. The challenge lies in the practical use of this technique, especially, when long sequences are compared, the computational load exceeds the capacity of the computing system. The complexity of heuristic algorithms remains on the order of $O(m \times n)$ but the number of computations based on residue-residue comparisons is greatly reduced. They tend to be more computational efficient than dynamic programming based algorithms. However, these algorithms sacrifice sensitivity for speed and therefore more distant sequence relationships may escape from detection.

2.1.4 Algorithms for multiple alignment

Multiple sequence alignment (MSA) aims to extract the relationship among many sequences. It aligns more than two sequences to look for maximum matching of characters. As an exact alignment, in order to align several sequences, it needs to generalize dynamic programming (DP) from pairwise alignment to a multidimensional space.

Let Σ be a set of characters (residues), S be a set of sequences $S_1, S_2, ...S_k$, we define MSA of k sequences as k equal-length sequences $S' = S'_1, S'_2, ..., S'_k$ so that $|S'_1| = |S'_2| = ... = |S'_k| = n'$, and removing space characters from the x-th sequence of S' gives S_x for $1 \leq x \leq k$. The *sum-of-pairs* (SP) score of an MSA is defined as the sum of the pair-wise scores of all pairwise between the sequences:

$$\sum_{1\leq x<y\leq k} \sum_{1\leq i\leq n'} \delta(S_{x,i}, S_{y,i}) \tag{2.1.8}$$

where $S_{x,i}$ is the i-th residue in x-th sequence and $S_{y,i}$ is the i-th residue in y-th sequence. Figure 2.2 illustrates the working space in a three dimensional alignment, in which sequences A, B and C are aligned.

Direct extension of dynamic programming to MSA needs huge computational time and space. Given k sequences of length l each, the time complexity of dynamic programming solution is $O(2^k l^k)$ for computations of the number of $O(l^k)$ cells with recurrence relation $O(2^k)$. For example: 6 sequences of length 100 requires 6.4×10^{13} calculations.

It has been proven that finding an optimal alignment is an NP-complete problem, which makes the running of MSA extremely slow, if not impossible, for genome-wide sequence analysis. To solve this problem, some approximating and heuristic methods

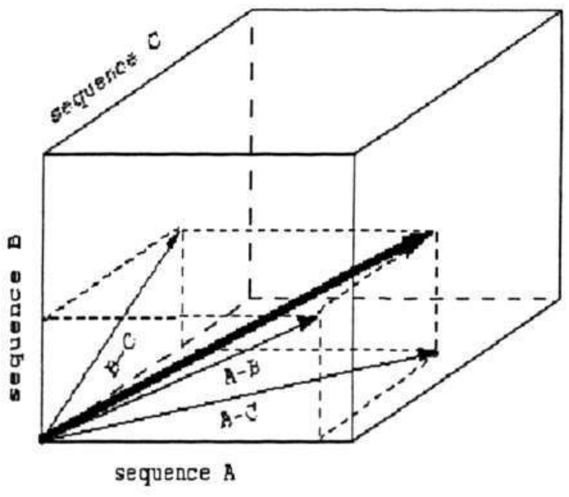


Figure 2.2: Three dimensional alignment

have been introduced. It is convenient to classify these existing methods into three main categories according to their properties.

Progressive alignment

In a progressive alignment (Feng & Doolittle 1987), it repeatedly applies the pairwise alignment algorithm instead of aligning all sequences simultaneously. The major steps are described below.

1. Determine distances between sequences.

2. Use a distance-based method to construct a phylogeneitc tree for the sequences.

3. Add sequences to the growing alignment using the order given by the tree.

This approach is by far the most widely used method for its advantage of speed and simplicity combined with reasonable sensitivity. The main shortcoming of this strategy is that once a sequence has been aligned, that alignment will never be modified even though it conflicts with sequences added later. Another problem is the difficult choice of the suitable scoring matrices and gap penalties that apply to the set of sequences (Higgins, Thompson & Gibson 1996). Long insertions or deletions can cause problem due to the intrinsic limitations of the gap penalty.

A popular tool based on progressive methods is ClustalW (Thompson, Higgins & Gibson 1994*b*). In general, the ClustalW algorithm performs better when the sequences are relative close-related. T-coffee (Notredame, Holm & HIggins 1998) is an improvement to the ClustalW algorithm. This algorithm uses a consistency-based objective function to make it possible to minimize potential errors when sequences are aligned in a progressive manner, especially in the early stages of the alignment assembly. The main difference between T-Coffee and ClustaW is that in T-Coffee, an extended library replaces a substitution matrix. Another difference is that T-Coffee's primary library is made of a mixture of global alignments(produced by ClustalW) and local alignments (produced by Lalign (Huang & Miller 1991)). This combination of local and global information enables T-Coffee to get better results than others.

Iterative alignment

Iterative alignment (Brocchieri & Karlin 1998) is to repeatedly realign subgroups of sequences and then these subgroups into a global alignment of all of the sequences. This method can reduce the initial alignment errors of most closely related sequences instead of propagating them to align more distantly related sequences. It allows for a good conceptual separation between optimization and objective functions.

Consistency-based alignment

In a consistency-based alignment (Bucka-Lassen, Caprani & Hein 1999), sequences are preprocessed so that the regions consistently conserved across the family can drive the alignment. The main advantage is to use information from structure analysis, sequence comparison, database search etc instead of a specific substitution matrix.

In Table 2.1, a large number of MSA programs developed in recent years are summarized.

Table 2.1: Some MSA programs

Program	Algorithm	Description	Reference
Align-m	Iterative/ Consistency-based	Use a non-progressive local approach to guide a global alignment, which is a new algorithm for multiple alignment of highly divergent sequences.	(VanWalle, Lasters & Wyns 2004)
ClustalW, ClustalX	Progressive	Each sequence is aligned with its closest neighbor in a guiding tree and the resulting groups are then aligned to each other in the same way until all sequences are aligned.	(Thompson et al. 1994*b*)
ComAlign	Consistency-based	Combine several multiple alignments into a single, often improved alignment.	(Bucka-Lassen et al. 1999)
DIALIGN	Consistency-based /Iterative	Align gap-free segments as a whole without introducing gaps, which is an algorithm for sequences where local homology is driving signals.	(Morgenstern, Frech, Dress & Werner 1998), (Morgenstern 1999), (Morgenstern 2004)
IterAlign	Iterative	In each iteration, sequences are locally compared to others and that every segment that shows high similarity with other proteins is replaced by a consensus.	(Brocchieri & Karlin 1998)
Praline	Iterative	Sequences are iteratively replaced with a complete profile made from a multiple alignment that only includes their closest relatives until the collection of profiles converges.	(Heringa 1999)
MAFFT	Progressive/ Iterative	Rapid group-to-group alignment by fast Fourier transformation.	(Katoh, Misawa, Kuma, & Miyata 2002)
MUSCLE	Progressive/ Iterative	Use a draft progressive step followed by an improved progressive and iterative refinement steps.	(Edgar 2004*a*) (Edgar 2004*b*)
PCMA	Progressive/ Consistency-based	Progressive method which aligns divergent groups by the T-Coffee strategy and aligns highly similar sequences as ClustalW.	(Pei, Sadreyev & Grishin 2003)
POA	Progressive/ Iterative	Directly align without the need for profiles by representing alignments as graphs.	(Lee, Grasso & Sharlow 2002)
Prrn	Progressive/ Iterative	Doubly nested randomized iterative alignment where group-to-group alignments are repeated to improve the overall score.	(Gotoh 1996)
T-coffee	Progressive/ Consistency-based	Use an alignment library to seek for maximum consistency of each residue pair with all other pairs of this library and guides the progressive step by means of this library.	(Notredame, Higgins & J. 2000)

2.2 Phylogenetic tree reconstruction

2.2.1 Phylogenetic methods

Phylogenetic trees have important applications in multiple alignment of biomolecular sequences (Thompson, Higgins & Gibson 1994*a*) (Notredame, Higgins & Heringa 2000) (Robert 2004), protein function prediction (Eisen 2003) (La, Sutch & Livesay 2005) (Lichtarge, Bourne & Cohen 1996), and drug design (Searls 2003). There are two general categories of methods for calculating phylogenetic trees: distance-based and character-based methods. The methods of both categories use sequence data, in form of an alignment, to reconstruct phylogenetic trees.

Distance-based methods calculate pairwise distances between taxa and connect those nodes that have the shortest distance into a new node replacing the old nodes, hence repeatedly reconstructing a phylogenetic tree. The analysis is based on the differences between sequences, rather than the original data. This technique is fast and quite simple for it merely groups the sequences according to their pairwise distances instead of searching the huge solution space of trees made up of all possible solutions to the sequence data. Current popular distance-based methods, such as the Neighbor joining (NJ) method (Saitou & Nei 1987) (Studier & Keppler 1988) and other distance-based methods are reported by several papers (Moret, Roshan & Warnow 2002, Nakhleh, Moret, Roshan, John & Warnow 2002, Nakhleh, Roshan, St. John, Sun & Warnow 2001*b*, Nakhleh, Roshan, St. John, Sun & Warnow 2001*a*).

Whereas distance-based methods compress all sequence information into a single number (distance data), character-based methods attempt to infer the phylogeny based on all individual characters (nucleotides or amino acids). They search through the space of all possible phylogenetic topologies i.e. trees. All trees that could possibly explain the evolutionary history of sequences are created and scored. The tree with the best score is the optimal solution for a sequence dataset. Two most popular methods are Maximum Parsimony (MP) (Camin & Sokal 1965) and Maximum Likelihood (ML) (Felsenstein 1981*a*).

MP tries to find a tree which explains data with the least mutations. For example, an input of sequences has 4 species, each of which is represented by a sequence of 3 characters: AAG, AAA, GGA and AGA. Try out different trees for these four sequences and count number of substitutions needed in each tree shown in Figure 2.3. The left tree in the figure shows the most parsimonious tree of this input, which has the parsimony length 3, is the minimum number of mutations of the input. Other right trees have the parsimony length 4 (require more substitution events).

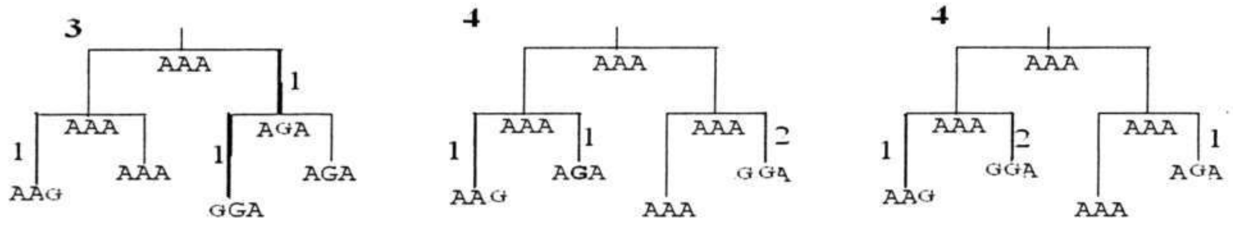


Figure 2.3: MP tree

ML searches a tree that maximizes the likelihood of data for a given evolutionary model. ML resembles MP in that the tree with the least number of changes will be most likely. However, ML evaluates trees using explicit evolutionary models. For example, in Figure 2.4, $N1$,...,$N5$ are the bases or residues observed in the extant ancestral taxa and edge length $t1$,...,$t4$ correspond to time. Let $P(Ni|Nj, tk)$ be the probability that the residue at node Nj becomes the residue at node Ni in time tk.

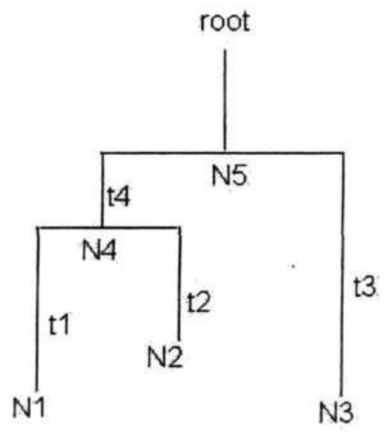


Figure 2.4: ML tree

The probability of the tree is:

$$P(N1,...,N5|T,t) = P(N1|N4,t1)P(N2|N4,t2)P(N3|N5,t3)P(N4|N5,t4)P(N5) \tag{2.2.1}$$

If we don't know the identity of the internal bases or residues at $N4$ and $N5$, the likelihood of the tree is obtained by:

$$P(N1,N2,N3|T,t) = \sum_{N4,N5} P(N1,...,N5|T,t) \tag{2.2.2}$$

The problem of ML is to find the T' which maximizes $P(N1,...,N5|T,t)$ across all trees T. If we assume independence of evolution at different sites, we can compute the probability of a given tree site by site. For ancestors at interior nodes are generally unknown in a tree, it need to generalize the likelihood by sum over all possible assignments of amino acids to the splits of the tree. ML methods can be used to explore relationships among more diverse taxa.

2.2.2 Complexity of phylogenetic analysis

The most fundamental algorithmic problem of character-based methods consists in the immense amount of potential alternative tree topologies. This number grows exponentially with the number of sequences n, e.g. for $n = 50$ organisms there already exist 2.84×10^{76} alternative topologies according to the Equation 2.2.3, a number almost as large as the number of atoms in the universe ($\approx 10^{80}$).

$$N_u = (2n-5) \times (2n-7) \times ... \times 3 \times 1 = \frac{(2n-5)!}{2^{n-3} \times (n-3)!} \qquad (2.2.3)$$

where N_u is the number of unrooted trees for n taxa.

MP problems are NP-complete which has already been demonstrated for the general version of the *perfect phylogeny* problem (Bodlaender, Fellows, Hallett, Wareham & Warnow 2000) and MP (Day, Johnson & Sankoff 1986) also known as the Steiner Tree problem in phylogenetics (Foulds & Graham 1982). Computing Maximum Likelihood trees is also commonly believed to be NP-complete (Steel 1994), though this could not be demonstrated so far because of the significantly superior mathematical complexity. Some exemplary figures are outlined in Table 2.2.

Table 2.2: Possible phylogenetic trees

Taxa(n)	Rooted tree $\frac{(2n-3)!}{2^{n-2}\times(n-2)!}$	Unrooted tree $\frac{(2n-5)!}{2^{n-3}\times(n-3)!}$
2	1	1
3	3	1
4	15	3
5	105	15
6	954	105
7	10,395	954
8	135,135	10,395
9	2,027,025	135,135
10	34,459,425	2,027,025

For constructing the Tree of Life, i.e., the evolutionary tree on all species on Earth, efficient MP and ML heuristics are required which allow for the analysis of large and complex datasets in a realistic amount of time, i.e. in the order of weeks.

2.2.3 Current softwares for phylogenetic tree reconstruction

Some of phylogenetic tree construction softwares are summarized in Table 2.3. The recently updated site by J.Felsenstein (Felsenstein 2004) provides a comprehensive list of nearly all available programs. One of the recent methods, RAxML (Stamatakis, Ludwig & Meier 2004) (Stamatakis, Ott, Ludwig & Meier 2005), is among the fastest, most accurate, and most memory-efficient ML heuristics on real biological datasets to the best of our knowledge. Furthermore, the global optimization method (fast Nearest Neighbor Interchange adapted from PHYML (Guindon & Gascuel 2003)) is not as efficient on real alignment data as RAxML. Thus, it is not suited to handle large real-data alignments of more than 1,000 sequences. Another approach which partially relies on divide and conquer has been implemented in TREE PUZZLE (Strimmer & Haeseler 1996). Although the program is very popular among biologists—mainly because it assigns confidence values to the different clades of the tree—it is too slow to handle large alignments containing more than 500 taxa. Rec-I-DCM3 (Roshan, Moret, Warnow & Williams 2004) is a new MP method that iteratively decomposes the dataset into subproblems, and solves them serially. The phylogenetic navigator (PHYNAV (S.V.Le, Schmidt & Haeseler 2004)) which is based on a zoom-in/zoom-out approach represents an interesting algorithmic alternative to Rec-I-DCM3 (Roshan et al. 2004). However, the program has a relatively high memory consumption (crashed on a 1.000-taxon tree with 1GB RAM when we ran it) compared to RAxML.

Table 2.3: This table is only a small selection of free software available for reconstruction phylogenetic tree, describing programs are recently developed or widely used.

Program	Algorithm	Description	Reference
Neighbor Joining	Distance-based	Two nearest nodes are chosen and merged recursively until all of the nodes are paired together	(Saitou & Nei 1987) (Studier & Keppler 1988)
WEIGHBOR	Distance-based	Weighbor is a weighted version of Neighbor Joining that gives significantly less weight to the longer distances in the distance matrix. The resulting trees are less perturbed by adding distant taxa compared to Neighbor Joining, and negative branch lengths are avoided	(Bruno, Socci & Halpern 2000)
BIONJ	Distance-based	Use a simple model of the sampling noise of evolutionary distances.	(Gascuel 1997)
TripleML	Distance-based/ ML	Estimate distances by local ML using a third taxon(or cluster) to improve long-distance estimation.	(Ranwez & Gascuel 2002)
PHYLIP	Distance-based/ ML/MP	A large package of free programs for parsimony, distance-based methods,ML and other methods. The ML programs have good tree searching capabilities, which also is able to apply several simple models.	(Felsenstein 1989) http://evolution.gs .washing-ton.edu/phylip.html
PAML	ML	A package of free programs for phylogenetic analysis using ML. It allows a wide variety of advanced models and some of programs are able to use gamma distributions to model heterogeneity of evolutionary rates among sequences,infer rate parameters for different genes and synonymous and nonsynonymous substitution rates etc.	(Yang 1997) http://abacus.gene.ucl .ac.uk/software /paml.html
MOLPHY	ML	A free package intended mainly for inferring phylogenetc tree by using ML and searching tree.	(Adachi & Hasegawa 1996) http://www.plantbio.uga edu/ russell/ soft-ware.html

2.3 Architectural development of HPC systems

In general, high performance computing (HPC) refers to hardware and software techniques for building computer systems to quickly perform large amount of computation. HPC includes computers, networks, algorithms and environments necessary to make such systems usable. These systems range from a departmental cluster of workstations, up to the largest super-computers. The economic benefit of these systems is increasing as computer models grow more viable and increasingly augment or replace physical experimentation. The design of HPC systems is driven by the requirements of tera-scale grand challenges for various organizations, such as bioinformatics.

A taxonomy of HPC (Parallel) architectures is shown in Figure 2.5 (Flynn.M. 1972). Single instruction, multiple data streams (SIMD) consists of multiple processors, a controller, and an interconnection network. The controller stores a program and broadcasts instructions to all processors simultaneously. Available processors perform the instructions on the contents of their own local memory. In SIMD architecture, no processor can execute a second instruction unless all processors finish the previous instruction because the system is synchronous.

A multiple instruction, multiple data streams (MIMD) machine typically consists of multiple processors and an interconnection network. This model allows each processor to store and execute its own program by providing multiple instruction streams in contrast to the SIMD. Often, all processors are executing the same program, but may be in different portions of the program at any given instant. MIMD machines are the most commonly used general-purpose parallel computers. There are mixed architectures where small number of processors are grouped together as a shared memory symmetric multiprocessor (SMP) and they are linked together in a

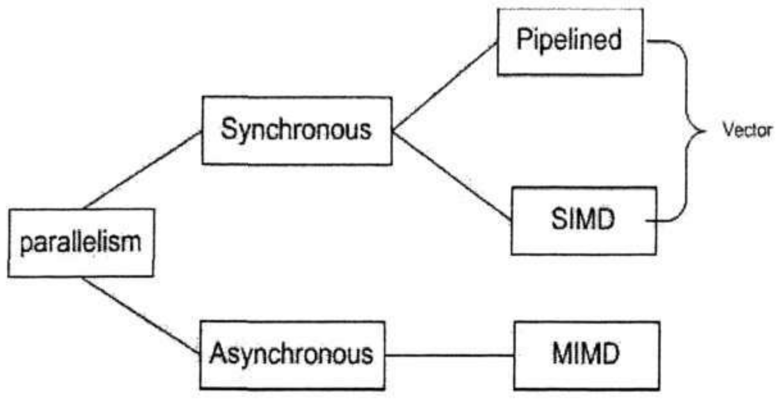


Figure 2.5: Parallel architecture

distributed memory architecture.

Synchronization is the act of bringing two or more processors to known points in their execution at the same clock time. In SIMD programming, synchronization is not an issue since every collective step is synchronized by the function of a control unit. The complexity and often the inflexibility of SIMD machines, strongly dependent on the synchronization requirements, have restricted their use mostly to special-purpose applications. In contrast, the asynchronous operation of MIMD computers makes them extremely flexible. For example, they are very well suited to the task-farming kind of parallelism, which is barely feasible at all on SIMD computers. But this asynchrony makes general programming of MIMD computers hard. It requires concurrent programming expertise and hard work, such as using message-passing to synchronize the parallel parts of an application program. Another issue on MIMD computer programming is the explicit use of synchronization primitives to control nondeterministic behavior. Nondeterminism appears almost inevitably in a system that has multiple independent threads of the control–for example, through race conditions. In the struggle for dominance between SIMD and MIMD, SIMD appears to have fallen by the wayside. Even though the SIMD can be cost effective for certain task, the more flexible and more general purpose nature of MIMD approach has prevailed.

Granularity is often used to refer to the relative size of the units of computation that executed in parallel. In particular, there is three task divisions: fine, coarse and medium. Fine-grained machines typically fall into the SIMD category. They consist of a relatively large number of small, simple processors, where all processors operate synchronously on the contents of their own memory. Coarse-grained machines consist of relatively few processors, each of which is large and powerful. They typically fall into the MIMD category, where processors operate asynchronously on the large, shared memory. There are also medium-grained machines which built from commodity microprocessors and designed in MIMD with both distributed and shared memory models.

2.3.1 Shared memory architecture

Shared memory machines have a single global image of memory that is available to all processors, typically through a common bus or switching network as shown in Figure 2.6. In this model, any processor can read or write to any part of the memory.

The single address map of a shared memory model simplifies the design of parallel programs. The interprocess communication is usually performed through shared variables. When several processors are accessing the same logical address space, blocks of code that only one processor can execute at a time are required for safety. For examples, POSIX threads (S.Kleiman, D.shah & B.Smaalders 1996) (Sum Mircosystems

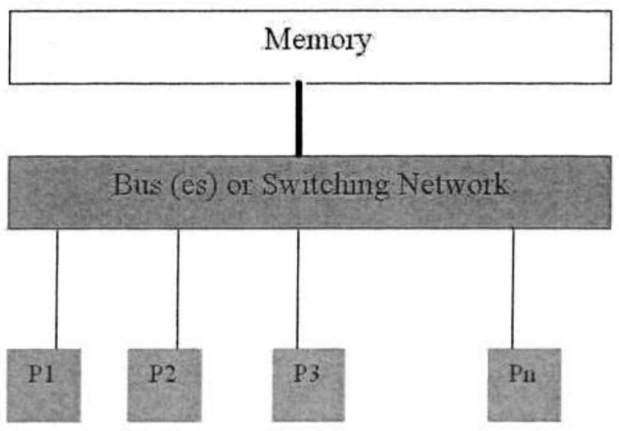


Figure 2.6: Shared Memory Architecture

1995) (of the IEEE 1996) are native threads of processing that run within a single process/application and share access to resources and memory at a fine-scale. OpenMP (R.Chandra & R.menon. 2001) (Board. 1977) (Quinn 2004) makes use of compiler directives, running systems and environment variables. The programmer provides hints to the compiler on how to parallelize sections of a correct serial implementation. More recently, UPC (Unified parallel c) (Carlson, Draper, Culler, Yelick, Brooks & Warren 1999), an extension of C that provides a shared address space and a common syntax and semantics for explicitly parallel programming in c.

These systems typically possess a relatively small number of processors (usually less than 16) which can share a common block of memory. It's easy to develop fast, efficient programs for this design because every processor has direct access to all data. However, the disadvantages of this type of arrangement are that it is not scalable beyond dozens of processors. After all, only a limited number of processors can be expected to share a block of memory. Also the technologies needed to connect the processors are rather expensive. To overcome the hardware scalability limitations of shared memory systems, massive parallel computing with scalable systems using distributed memory architectures became the center of interest.

2.3.2 Distributed memory architecture

In distributed memory machines, each processor has access only to its local memory as show in Figure 2.7. Processors communicate by sending messages to each other through interconnection network.

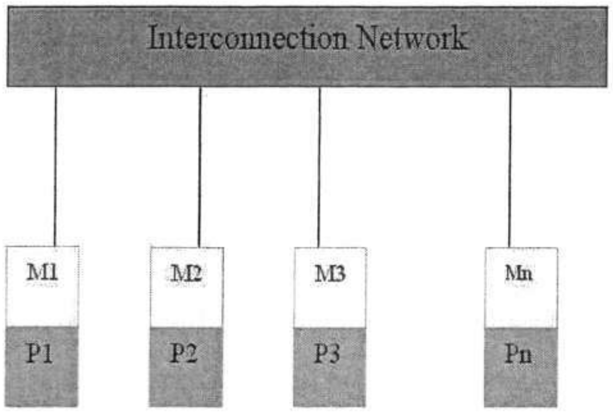


Figure 2.7: Distributed Memory Architecture

Distributed memory architectures scale very well on the other hand. But there is "time penalty" for communication between processors. In a distributed-memory architecture, communication between processors is performed by using a message-passing paradigm. There are two message passing models: Parallel Virtual Machine (PVM)(Geist, Begulin, J.Dongarra, W.Jiang, R.Manchek & V.Sunderam. 1994) and Message Passing Interface (MPI)(*MPI standard 2.0.* 1997). PVM model enable it to use a heterogeneous system containing different types of compute nodes. It could dynamically add or delete compute nodes and processors from an application program or from a system console. It supports for inter-operability both at the programming language level and the communication system. Therefore, PVM has extra capabilities to handle heterogeneous and faulty processors. MPI binds between a communication context and a group of processors. There is a unique communicator in each group and point-to-point process communication is allowed only within a group.

2.4 Summary

In this chapter, a review of problems and methods used in computational biology is presented. They are widely regarded as key technologies analyzing genomic sequence structures and functions, as well as understanding the whole genomes. The demand for computational power in this field will continue to grow as the complexity and the volume of data increase. It is clear that the development of high performance computing technologies has become crucial for deployment of the software systems to tackle the various problems.

Chapter 3

HPC System Design for Sequence Analysis

This chapter presents the hardware and software design for a generic parallel sequence analysis system, PSAS, which aims to perform sequence analysis with high accuracy within reasonable time.

3.1 Hardware architecture

A customized multi-node cluster as a hardware environment for the parallel sequence analysis system is here studied and proposed. For several reasons such a system is designed, such as its attractive performance/price ratio and adaptivity in handling scalable problem sizes. The cost can be an order of magnitude cheaper than a traditional supercomputer while providing the same computational power. In addition, the configuration of the cluster is scalable, and it can be built ranging from a few compute nodes up to hundreds of nodes; a machine comparable with supercomputers. To users, the cluster behaves like a single system, but has higher performance through redundant processors, storage, and data paths.

The building blocks are a group of customized ES45 compute nodes (Srivastava 2001), as shown in Figure 3.1 . Each node has 4 Alpha-EV68 1GHz processors and an interconnect PCI adapter capable of over 280 MB/s sustained bandwidth. It also has a crossbar as its internal network with an aggregate bandwidth of 5.2 GB/s (1.33 GB/s/processor). This is sufficient to deliver 1.0.64 byte/clock cycle to each processor in the node simultaneously.

The compute nodes are interconnected by the Quadrics 128-port interconnect switch chassis as illustrated in Figure 3.2. Multiple connections are used in each node to increase aggregate throughput and to reduce queuing delays on the 4-processor compute nodes. The fast interconnect delivers up to 500 MB/s per node, with 32 GB/s of cross-section bandwidth.

A fat tree is designed for inter-node connection, which combines the characteristics of fully connected crossbar and tree structures. First, bandwidth scales with the increasing number of processors. Secondly, the internal connection count of a fat tree grows much more slowly, at the rate of $O(nlogn)$ (where n is the number of ports), as opposed to the crossbar switch growth rate of $O(n^2)$ (Hewlett-Packard 2001). Figure 3.3 shows a quaternary fat tree of dimensions 3, which connects 64 processing nodes using 16 switches. The bidirectional nature of the links localizes traffic to a sub-tree large enough to span both nodes. The uplinks in the top stage of a switch network can either be used for expansion or can be used as additional downlinks doubling the number of nodes that can be connected without reducing the bisectional bandwidth.

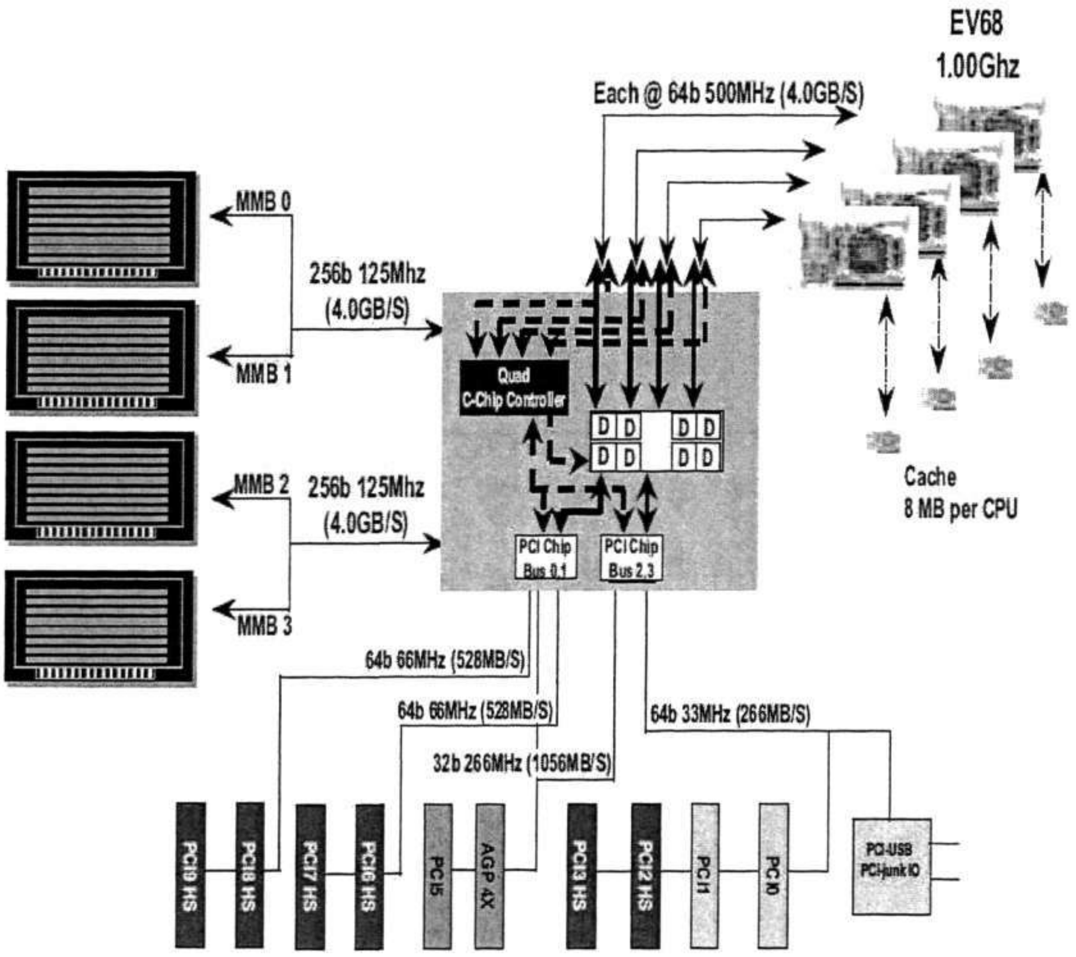


Figure 3.1: Architecture of a multi-processor compute node

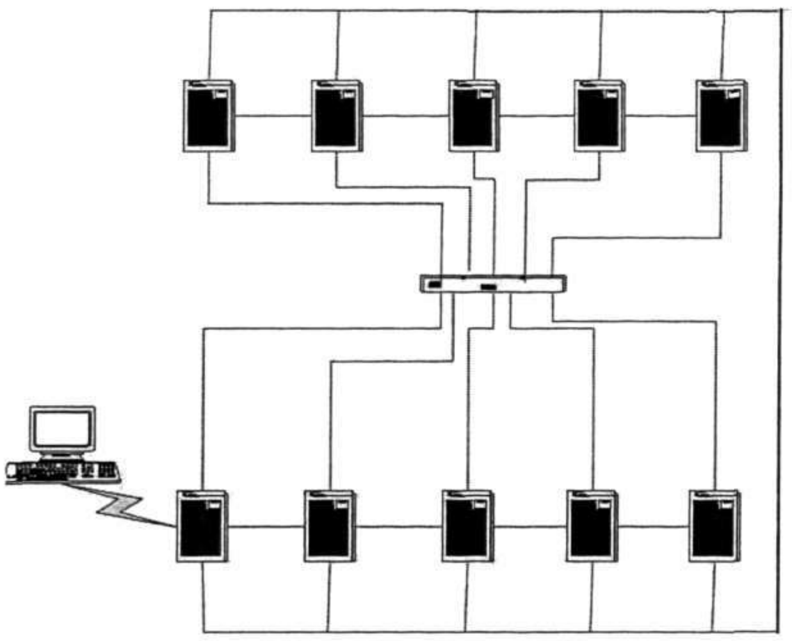


Figure 3.2: Inter-connect between compute nodes

3.2 Software system

3.2.1 Communication between processes

The proposed system exchanges information between processes through Message Passing Interface (MPI) that uses Elan hardware, software and switches to provide extremely low latency and high bandwidth communication.

MPI uses objects, called communicators and groups, to define which collection of processes may communicate with each other. Within a communicator, each process has its own unique integer identifier, called rank, assigned by the system when the process initializes. Ranks are contiguous and begin at zero. They are used to specify the source and destination of messages. For reasons of efficiency, however, message

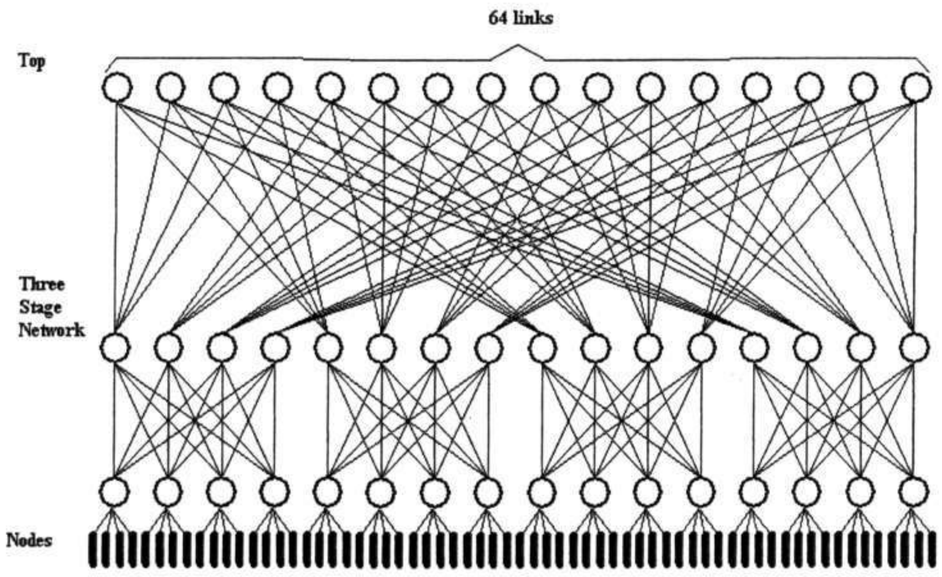


Figure 3.3: Fat tree topology, which is made up of three-stage network combining 64 nodes and the top stage with 64 links out

passing systems generally associate only one process per processor.

MPI provides a rich set of library functions for point-to-point and collective communication operations. MPI point-to-point operations typically involve message passing between two different MPI tasks: one task is performing a send operation, and the other task is performing a matching receive operation. Most of the MPI point-to-point routines can be used in either blocking or non-blocking mode. Blocking mode means that routines only return once communication completes, or when the user buffer can be used or re-used. Blocking communication is used to wait for everything to be ready, so that variables can safely be altered in order to avoid a deadlock. Process will be stopped until the send application buffer is free or the receive application buffer is written. That is, each block begins the computing only when its previous division has completed the computing. In non-blocking communication the routine returns before communication is complete. Communication continues in the background while other work is performed. The application programmer takes care of the data integrity.

Applications may require coordinated operations among multiple processes. For example, all processes need to cooperate to sum a set of numbers distributed among them. Collective communication must involve all processes in the scope of a communicator. All processes are, by default, members in the communicator MPI_COMM_WORLD.

Types of Collective Operations are as following:

- Synchronization - processes wait until all members of the group have reached the synchronization point.
- Data movement - broadcast, scatter/gather, all to all.
- Collective computation (reductions) - one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on that data.

These operations are implemented such that all processes call the same operation with the same arguments. Thus, when sending and receiving messages, one collective operation can replace multiple sending and receiving operations, resulting in lower overhead and higher performance. Two collective operations are commonly used:

MPI_Bcast (&buffer,count,data type,root,comm) –Broadcasts (sends) a message from the process with rank "root" to all other processes in the group. As illustrated in Figure 3.4, all the processes receive the minimum integer, 7, from the "root", P_1.

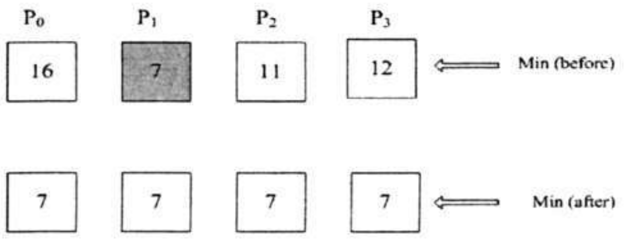


Figure 3.4: Process P_1 broadcast a message to all other processes of the group

MPI_Allreduce (&sendbuf,&recvbuf,count,data type,op,comm) – Applies a reduction operation and places the result in all processes in the group. As shown in Figure 3.5, MPI_MINLOC is used to find a global minimum value, 1, and its location, P_1.

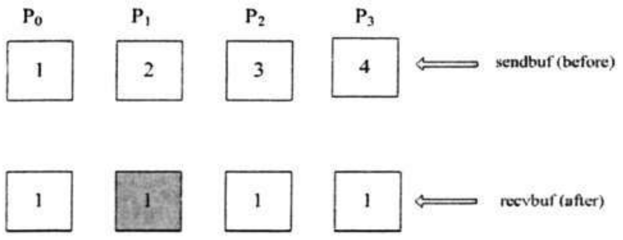


Figure 3.5: Perform and associate reduction operation, MPI_MINLOC, across all processes in the group and place the result in all tasks.

To establish a basic understanding and test on the multi-node cluster, we developed a protocol program (shown in Appendix), dealing with a typical matrix problem of Jacobi iteration (solving Laplace's equation).

3.2.2 Task schedule

In most of the parallelism, a master-worker mechanism for task scheduling is used, as shown in Figure 3.6. A processor, *P0*, works as a master process communicating with the outside and organizing jobs. The other processors, *P1,...,Pn*, work as working processes. Any jobs requested from outside will be received as a socket communication (specifying analysis methods, input data and parameters). The master process decides an arrangement for data partitioning and dispatching dynamically according to the status of working processors. Working processes repeatedly perform jobs on a small fraction of data until all jobs are finished.

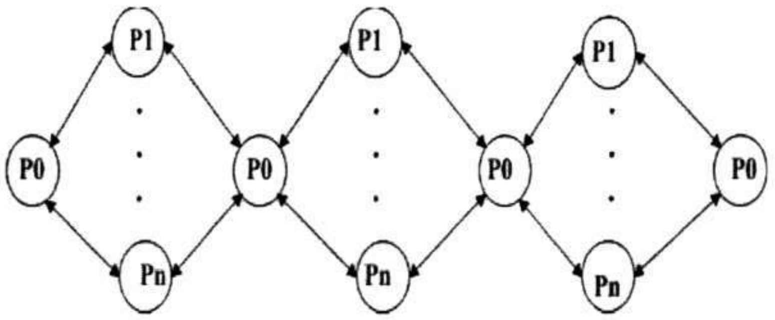


Figure 3.6: Master and worker model

3.3 Performance analysis

There are a number of criteria for measuring the performance of parallel programs. Running time measures the amount of time from the starting of a parallel program to the end when the program obtains the final result of the computation. Speedup measures the performance improvement gained through parallelization, which is the ratio between the running time using single processor and multiple processors.

When bench marking the performance improvement, it should also be noted that the maximum speedup achievable depends greatly on the degree of parallelism in a particular algorithm (Amdahl 1967). Amdahl's law states that potential program speedup is defined as follows: if n is the number of processors, S is the amount of time spent on serial parts of a program, P is the amount of time spent on parts of the program that can be done in parallel, and we assume the total time as $S+P=1$ for algebraic simplicity, then the speedup is given by:

$$\textit{speedup} = \frac{S+P}{S+P/n} = \frac{1}{S+P/n} \tag{3.3.1}$$

The speedup is limited by the fact that not all parts of code can be run in parallel. Substituting in the equation, when the number of processes goes to infinity, the speedup is still limited by $\frac{1}{S}$. This indicates that the serial fraction of code has a strong effect on speedup and helps to explain the need for large problem sizes when using parallel computers. To get good performance it is necessary to run large applications with large data array sizes and lots of computation. The reason for this is that as the problem size increases the opportunity for parallelism grows and the serial fraction shrinks; and it shrinks in its importance for speedup. The linear speedup curve is rarely achieved because parallelism entails a certain amount of communication and management overhead.

3.4 Overview of the system

Upon setting up and configuring the hardware and supporting software, I developed a set of HPC algorithms for sequence analysis and organized them into a parallel sequence analysis system. These parallel algorithms have characteristics far different from traditional serial algorithms. They break up a large task into a set of smaller tasks, assign each of the smaller tasks to a processor to work and coordinate the work between the processors periodically (problem dependent). In order to identify the parts of an algorithm that can be executed concurrently, it requires a thorough understanding of the algorithm, exploiting any inherent parallelism which may exist. Sometimes, it also needs a restructuring of the algorithm or an entirely new algorithm. Load imbalance is one of the main performance degradation factors of parallel algorithms running in the heterogeneous environments which are often used in cluster computing. The complexity of a problem is measured by the minimum number of messages that need to be sent to solve the problem while obtaining maximum performance out of all processors at all times.

The details of the proposed algorithms are addressed in the following chapters 4-6. An overview of the system is shown in the Figure 3.7.

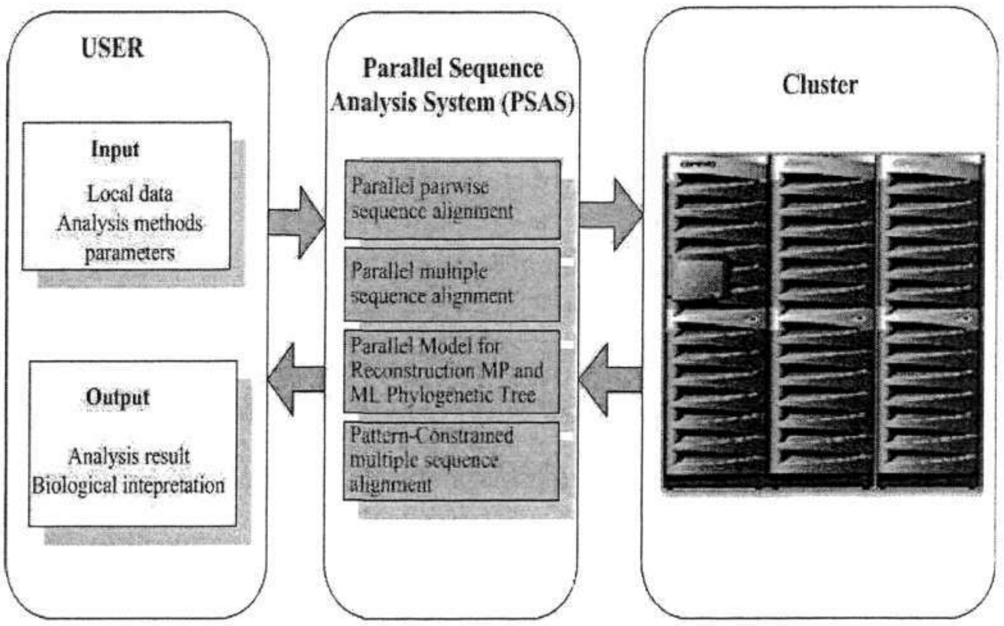

USER
Input
Local data
Analysis methods
parameters
Output
Analysis result
Biological intepretation
Parallel Sequence Analysis System (PSAS)
Parallel pairwise sequence alignment
Parallel multiple sequence alignment
Parallel Model for Reconstruction MP and ML Phylogenetic Tree
Pattern-Constrained multiple sequence alignment
Cluster

Figure 3.7: Parallel sequence analysis system

Key modules are as following:

- Parallel Pairwise Sequence Alignment – take advantage of dynamic programming and parallel computing to produce optimal results.
- Parallel Multiple Sequence Alignment –be able to align hundreds or thousands of sequences on multiple processors within reasonable time.
- Parallel Model for Reconstructing Phylogenetic Trees – be flexible enough to reconstruct both Maximum Parsimony (MP) and Maximum Likelihood (ML) phylogenetic trees and perform a more complete search of the tree space within limited time, yielding better trees in terms of final parsimony and likelihood values than comparable programs.
- Pattern-Constrained Multiple Sequence Alignment – guarantee that the generated alignment satisfies the pre-defined constraints that some particular patterns should be aligned together.

3.5 Summary

This chapter introduced our initial work on the development of a parallel sequence analysis system; both on the hardware architecture and on the software system. With such a system, a number of algorithmic and technical solutions are designed for an assorted collection of applications. In the following chapters, details of these solutions will be discussed.

Chapter 4

Parallel Computing for Sequence Comparison

The initial step in any phylogenetic analysis is to establish homology statements across taxa. Sequence comparison is, in essence, a procedure by which we can recognize and describe potential homology among nucleotide or amino acid positions. The complexity of sequence comparison problems and the necessity to deal with huge amounts of sequence data makes the development of algorithms which are fast and require little memory become a great concern. To meet these demands, two new algorithms are presented in this chapter for analyzing these biological sequences to gain parallel computing power at low cost. The first one is a "block-based wavefront" algorithm developed to speedup optimal pairwise alignment. The second one is a fast and practical algorithm for multiple sequence alignment. With these improvement, it is possible to apply the proposed algorithms in large-scale sequence projects that were previously beyond their scope.

4.1 Block-based wavefront algorithm for efficient pairwise alignment

4.1.1 Pairwise alignment via dynamic programming

Dynamic programming based technique uses previous steps for optimal alignments of smaller subsequences. For two sequences a=$a_1a_2...a_m$ and b=$b_1b_2...b_n$, an $m \times n$ matrix F is constructed, indexed by i and j, one index for each sequence, where the element, $F[i][j]$, is the score of the best alignment between $a_1, a_2, ..., a_i$ and $b_1, b_2, ..., b_j$. In order to keep track of the different values for the different gap lengths, an affine gap model is applied. For local alignment, the score value $F[i][j]$ is built recursively by the following Equation 4.1.1.

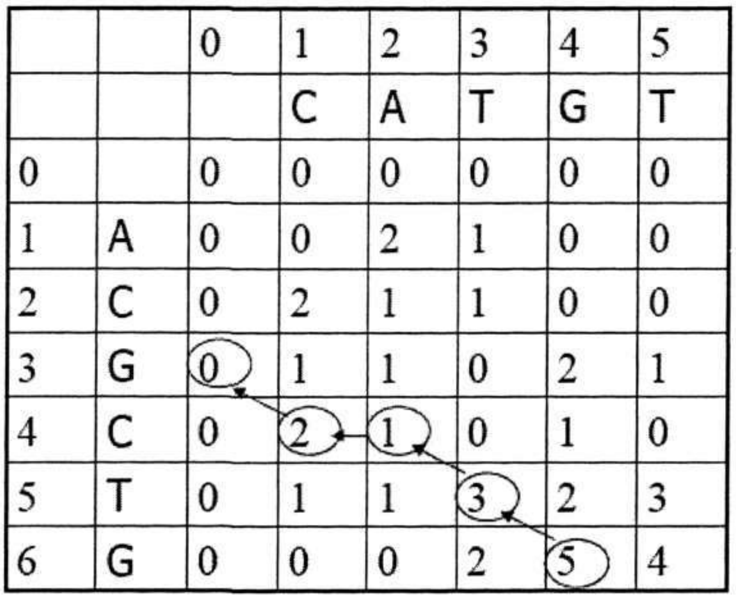
$$F[i][j] = max \begin{cases} F[i-1][j-1] + s(a_ib_j) \\ I_x[i-1][j-1] + s(a_ib_j) \\ I_y[i-1][j-1] + s(a_ib_j) \\ 0 \end{cases}$$

$$I_x[i][j] = max \begin{cases} F[i][j-1] - d \\ I_x[i][j-1] - e \end{cases}$$

$$I_y[i][j] = max \begin{cases} F[i-1][j] - d \\ I_y[i-1][j] - e \end{cases} \tag{4.1.1}$$

Where $s(a_ib_j)$ is the score for aligning the characters at positions i and j; d is an open gap penalty; e is an extended gap penalty; $F[i][j]$ is the best score given a_i with b_j; I_x is the best score given a_i aligns with a gap; I_y is the best score given b_j aligns with a gap.

For example, given two sequences a =**CATGT** and b =**ACGCTG**, the score value can be evaluated systematically by using a tabular computation. In order to obtain the alignment, it starts at the highest-scoring positions in the score matrix and follows a trace path from those positions up to an element whose score is zero, as shown in Figure 4.1. In the first column and the first row, one can see the results of the calculations of $F(a_i, -)$ and $F(-, b_j)$. In the second row, one can see the propagation of the algorithm, given that a gap and a mismatch cost -1 and a match costs +2.



		0	1	2	3	4	5
			C	A	T	G	T
0		0	0	0	0	0	0
1	A	0	0	2	1	0	0
2	C	0	2	1	1	0	0
3	G	$\boxed{0}$	1	1	0	2	1
4	C	0	$\boxed{2}$	$\boxed{1}$	0	1	0
5	T	0	1	1	$\boxed{3}$	2	3
6	G	0	0	0	2	$\boxed{5}$	4

Figure 4.1: Example of the local alignment based on dynamic programming between two sequences: CATGT and ACGCTG

4.1.2 Possible solutions

Pairwise alignment algorithms based on dynamic programming guarantee optimal results, but they impose extremely high requirements on both computer memory and execution time.

A number of fine-grained parallel architectures have been developed for dynamic programming based algorithms. P-NAC was the first such machine and computed edit distance over a four-character alphabet (D.P.Lopresti 1987). More recent examples, better tuned to the needs of computational biology, include BioScan(R.K.Singh & et al. 1996) , BISP (Chow, Hunkapiller, Peterson & Waterman 1991), and SAMBA (Guerdoux-Jamet & Lavenier 1997). An approach presented in (Schmidt, Schroder & Schimmler 2002) is based on instruction systolic arrays (ISAs). ISAs combine the speed and simplicity of systolic arrays with flexible programmability. Special-purpose hardware implementations can provide the fastest means of running a particular algorithm with very high PE (processing element) density. However, they are limited to one single algorithm, and thus cannot supply the flexibility necessary to run a variety of algorithms required for analyzing DNAs, RNAs, and proteins.

In addition to architectures specifically designed for sequence comparison, existing programmable serial and parallel architectures have been used for solving sequence alignment problems. Alpern et.al. (Alpern, Carter & Gatlin 1995), Rognes (T.Rognes & E.Seeberg 2000) and Wozniak (A.Wozniak 1997) explore instruction level parallelism in single-processor machines. Lander et. al. (E. Lander & Taylor 1988) and Brutlag et.al. (Brutlag, Dautricourt, Diaz, Fier, Moxon & Stamm 1997) discuss the implementation on a data parallel computer. Several other approaches are based on the SIMD concept, e.g. MGAP (Borah, Bajwa, Hannenhalli & Irwin 1994), Kestrel

(Blas & et. al 2005), and Fuzion (Schmidt, Schroder & Schimmler 2002). SIMD and ISA architectures are programmable and can be used for a wider range of applications, such as image processing and scientific computing. Since these architectures contain more general-purpose parallel processors, their PE density is less than the density of special-purpose ASICs. Nevertheless, SIMD solutions can still achieve significant runtime savings. However, the costs involved in designing and producing SIMD architectures are high. Therefore, none of the above solutions has a successor generation, making upgrading impossible.

Reconfigurable systems are based on programmable logic such as field-programmable gate arrays (FPGAs) or custom-designed arrays. Several solutions including Splash-2 (Hoang 1993) and Decipher (*http://www.timelogic.com* n.d.) are based on FPGAs while PIM has its own reconfigurable design (M.Gokhale & et al. 1995). In (Oliver, Schmidt & Maskell 2005), it presented a new FPGA solution that achieves a speedup of more than 100 compared to a Pentium4 using a standard off-the-shelf FPGA. Compared to the previously published FPGA solutions, it uses a new partitioning technique for varying query sequence lengths. The design presented in (Yamaguchi, Maruyama & Konagaya 2002) is closest to this approach since it also uses a linear array of PEs on a reconfigurable platform. Unfortunately, FPGAs are generally slower and have lower PE densities than special-purpose architectures. They are flexible, but the configuration must be changed for each algorithm, which is generally more complicated than writing new code for a programmable architecture.

Based on MIMD (Multiple Instruction, Multiple Data) architecture, Edmiston et.al (E.W. Edmiston & Smith 1988) presents parallel algorithms for sequence and subsequence alignment that achieve linear speedup and can use up to O(min(m,n)) processes, where m and n are the length of two sequences. But it stores the entire dynamic programming table, for example, if we want to align 2 sequences of 1 million base pairs long and that we only use one byte to store the score of each element in the score matrix, it will need $1M \times 1M$ main memory. In this way, the memory will be easily used up and it will have to wait too long for memory swapping. More recently, Rajko and Aluru (S.Rajko & S.Aluru 2004) present a space and time optimal parallel algorithm for the pairwise sequence alignment. They claim that their method requires only O((m+n)/p) space and runs in O(mn/p) time. However, there are no any practical experiments to support this theory in the paper.

4.1.3 Block-based wavefront

Mathematically, the dynamic programming based method is: to construct an $m \times n$ matrix F, where m and n are the length of the two sequences. The Equation 4.1.1 imposes data dependencies between the matrix elements in directions of left-to-right, top-to-down and main-diagonal. These dependencies imply a particular order of computation of the matrix. Following the recurrence equation, the matrix F is filled from top left to bottom right with i going from 1 to m and j from 1 to n. The order of computation of the elements in the similarity matrix is triggered by the flow of the data from neighboring elements: $F[i-1][j-1]$ (or $I_x[i-1][j-1]$ or $I_y[i-1][j-1]$), $F[i-1][j]$ (or $I_x[i-1][j]$) and $F[i][j-1]$ (or $I_y[i][j-1]$) at each step. The similarity matrix can be computed in parallel by distributing the computation along anti-diagonals because elements which can be computed independently of each other are located on a so-called *wavefront*. As illustrated in Figure 4.2 *wavefront* "moves" across the matrix as computation proceeds.

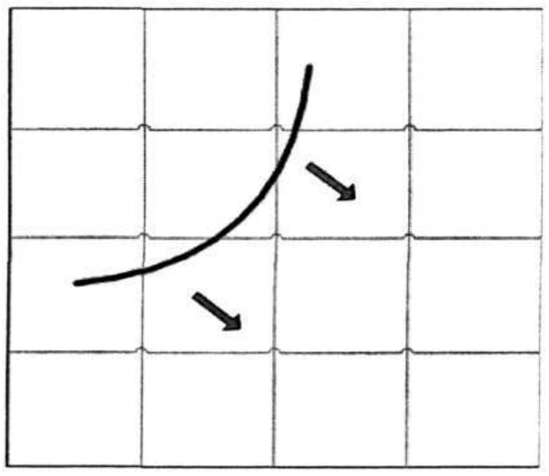


Figure 4.2: Wavefront moves across a matrix

However, such a *wavefront* computation mode has a few problems. One problem is each parallel "wavefront" leads to lots of communications among processes. For example, after process 1 computes the top-left element, it has to send the result to processes 2 and 3. Therefore, this method demands an extremely fast inter-process communication such as on systolic arrays. The other problem is that it requires a very large number of processors if real biological data is to be considered.

To solve these problems, we proposed a "block-based wavefront" algorithm to compute blocks instead of individual elements. The algorithm divides the similarity matrix by column into p groups (p is the number of processors) evenly with a number of complete columns, and assigns each processor one such group. The columns in each processor are grouped into blocks with the height of B (B is the height of block that would be adjusted according to the number of rows). Therefore, the computation of a given block requires only the column segment of the block to its immediate left, and the main-diagonal element, a total $B + 1$ elements. The parallel alignment is executed in a block-based wavefront such that computing nodes will first calculate the blocks along the first anti-diagonal in parallel, then along the second anti-diagonal in parallel, the third, the fourth,..., until the last anti-diagonal.

Figure 4.3 shows an example of computing a 16×16 matrix on 4 processors. The horizontal sequence x with 16 columns is distributed evenly to 4 processors. In each round, processes compute a 4×4 block of matrix. Initially process $p1$ starts computing block 1 in round 1. Then processes $p1$ and $p2$ can work in round 2, processes $p1$, $p2$ and $p3$ at round 3 and so on.

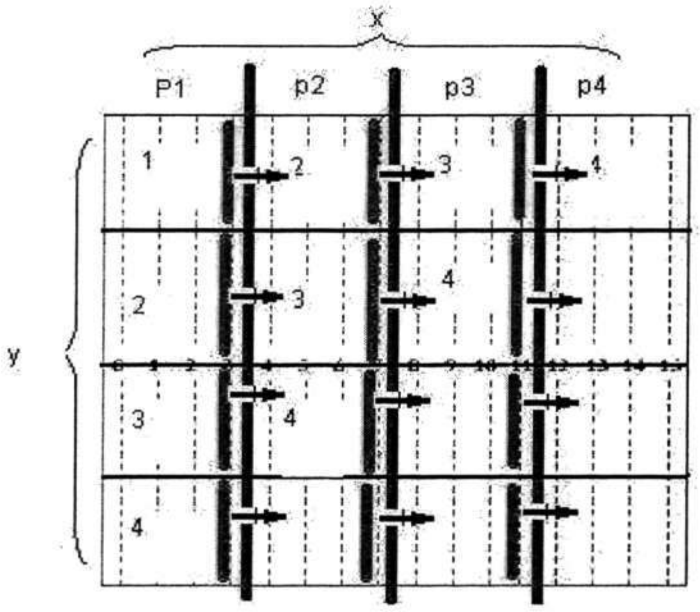


Figure 4.3: Block-based wavefront with 4×4 block size

In a *wavefront* computation mode, each element on different processes can be computed only after receiving two input values (elements of the left-to-right and the main diagonal). However, in the proposed algorithm, if a block has 4 rows and 4 columns, 16 elements will be computed after receiving 5 input values. Thus, communication-to-computation ratio drops from 2:1 to 5:16, an 84% reduction. Even though this algorithm will increase some serial computations when computing elements within a block, it decreases the communication load dramatically. The computing time complexity of each process is $\frac{m \times n}{p}$ when distributing the whole work load to p processes.

A procedural description of the proposed algorithm is shown in Table 4.1. Each process P_i receives the required elements of the neighbor block from the previous process P_{i-1}, then it computes a $m/p \times B$ block. After the computation of a block, it sends the requested elements to the next process P_{i+1}.

This algorithm does not need a global control: that is, once it is started it continues to complete the whole matrix over the cluster of compute nodes without the external intervention. We would step back from global update and ask only what each process need to do its job and what it must pass on to other processes. In order to ensure that no data value is updated at step $t+1$ until the data values in neighboring blocks have been updated at step t. We use a blocking communication mode. Process will be stopped until the **Send** application buffer is free or **Recv** application buffer is written. That is, each block will not begin computing until its previous division has completed computing.

4.1.4 Experimental results

In order to evaluate the performance of the proposed algorithm, a dataset consisting of sequences that range from 100k to 900k nucleotides was used. The algorithm was run on 4 to 48 processors to study the execution time and speedup. For uniprocessor performance, the serial version is used as a baseline. During the experiments,

Table 4.1: Parallel algorithm to compute the score matrix F

```
–Input :
    Sequence $x$ and $y$ of length $m$ and $n$
    Substitution matrix: $s(a, b)$
    Gap penalty: $g$
    Block's height: $B$
    Number of processes: $p + 1$ ($p$ is number of working processes)
–Output : Score of the optimal local alignment of $x$ and $y$

–Algorithm :
1. Connect to $p + 1$ processes and establish each process's rank: $myrank$

2. On master process (if $myrank = 0$)
       $x\_width = \frac{m}{p}$, $x$ is divided into $p$ pieces: $x = x_1, x_2, ..., x_p$
       $y\_width = B$, $y$ is divided into $n/B$ pieces: $y = y_1, y_2, ..., y_{n/B}$
       For $i = 1$ to $p$ do
           **Send** $y_j$ to next process: $myrank + 1$
       For $j = 1$ to $n/B$ do
           **Send** $y_j$ to next process: $myrank + 1$

3. On working processes (if $myrank > 0$)
       **Recv** $x_{myrank}$, part of sequence $x$ from master process
       For $j = 1$ to $n/B$ do
           If **Recv** $y_j$ and neighbor block's most right column from previous process: $myrank-1$
               Compute for a dynamic programming based alignment( $x_{myrank}$ and $y_j$)
       **Send** $y_j$ and own block's most right column to next process: $myrank + 1$
       Report the highest score in matrix $F$
```

the height of block (parameter "B") is assigned 100. In order to remove the unpredicted noise generated by the operating system, five consecutive runs for each pair of sequences were performed. The average results as from the five runs were used.

Table 4.2 lists the execution time for different problem sizes. If we use the largest data as an example, we can notice that the execution time is dramatically reduced from more than four days when running a serial program to about 2 hours when running the parallelized program on 48 processors.

Table 4.2: Execution time (sec)

Proc No.	1	4	8	12	16	20	24	28	32	36	40	44	48
$100k\times 100k$	3824	1159	605	487	318	254	210	188	168	162	156	144	132
$300k\times 300k$	38000	10200	5400	3999	2805	2200	1700	1500	1383	1222	1188	1061	952
$900k\times 900k$	390000	103660	53000	35000	27753	21400	17300	15300	13266	12600	11700	10200	8700

Figure 4.4 displays the speedup. It can be found that, for $100k \times 100K$ sequences, there is a little drop in speedup when more processors are added to the task. However, as the sequence sizes increase the speedup approaches the optimal linear speedup. The lack of speedup for the smaller dataset is a result of there not being enough jobs to fully exploit all the 48 processors' computing power. According to Amdahl's Law presented in Section 3.3, as the problem size increases, the opportunity for parallelism grows, and the serial fraction shrinks in its importance for speedup. Thus, the best speedup curve is obtained for the largest sequences that are aligned. The granularity of work is more reasonable and the speedup becomes linear for multiple processors because of the large sequence size.

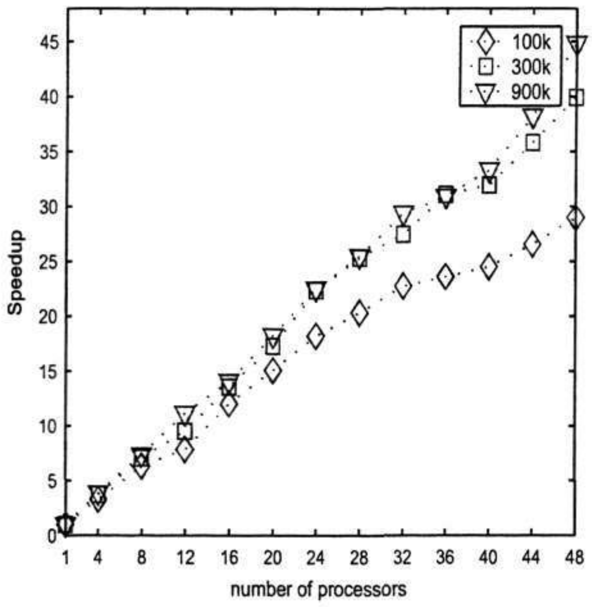


Figure 4.4: Speedup of the "block-based wavefront" algorithm for optimal pairwise alignment

4.2 Parallelized multiple sequence alignment

4.2.1 Introduction

Progressive alignments are widely used heuristic algorithms to compute multiple sequence alignment, of which the ClustalW algorithm has become the most popular one. It provides weights to sequences and adjustable parameters with reasonable defaults (Gibson, D. & Thompson 2002). It is a straightforward progressive alignment strategy where sequences are added one by one to the multiple alignment according to the order indicated by a pre-computed guiding tree. Because of the nature of the algorithm, the processing time can be considerably reduced if the computational loads are distributed over multiple processors.

Several parallel algorithms for multiple sequence alignment have been developed in recent years. The parallel version of the ClustalW program reported by Mikhailov et al. (Mikhailov, Cofer & Gomperts n.d.) is designed for shared memory multiprocessor machines. It achieves a speedup of 10 on a 16-processor shared memory machine. It parallelized the initial phase of the pairwise sequence alignments because all the pairwise alignments are independent. The guiding tree construction phase of the algorithm was also parallelized in it. However, the final phase was only partially parallelized. Usually, these machines are commodity parallel architectures and quite expensive.

Kleinjung et al. (kleinjung, Douglas & Heringa 2002) have reported a parallel progressive alignment strategy without a guiding tree. Their implementation is not a strict parallelization of the ClustalW algorithm. Recently, Li (Li 2003) and Justin Ebedes et al. (Ebedes & Datta 2004) parallelized of the ClustalW algorithm for distributed memory architectures by using MPI. Li concluded that the serial ClustalW implementation spends almost 96% running time in the first stage for pairwise alignment of the n input sequences. At the phase of building a guiding tree, Li has parallelized the procedure of constructing a guiding tree, but did not achieve any significant improvement in speedup. Justin Ebedes et al. did not parallelize this phase at all. However, when handling larger taxa sets, such as $n > 5000$, the second phase of construction a guiding tree spends more than 30% running time instead of 4%. This is one of the reasons that the speedup of the two programs decreases when employing more processors for large taxa sets.

4.2.2 The proposed algorithm

It is known that the ClustalW algorithm involves three stages: first, pairwise alignments are made in order to calculate the divergence of each pair of sequences. For a number of n input sequences, using the symmetry of the pairwise matrix, one needs to estimate $\frac{n(n-1)}{2}$ pairs of sequences. Next, a Neighbour-Joining (NJ) tree, or Guiding Tree, is constructed. It is used to guide the final progressive multiple alignment. Finally, the sequences are progressively aligned according to the hierarchy in the guiding tree. In the progressive alignment, the most similar sequences are aligned first, then followed by the alignment of more distant sequences or groups of sequences (profile alignment). In short, for n sequences, each of the sequences with an average length of l, the execution time complexity of the three steps are $O(n^2l^2)$, $O(n^3)$ and $O(nl^2)$ respectively(Catalyurek, Stahlberg, Ferreira, Kurc & Saltz 2002).

Using different numbers of the GPCRs as input data, we have implemented the ClustalW algorithm and a typical fraction of execution time is illustrated in Figure 4.5 (where PW denotes Pairwise Alignment, GT for Guiding Tree and PA for Progressive Alignment in the figure label). The fraction of execution time has shown that, when a small number of sequences is treated, the execution time in building a guiding tree counts less than 5%. However, when the number of sequences increases, say, 800 sequences are treated, the execution time for this stage grows sharply, in this case 35%. Our experimental results are quite encouraging for parallelization of the building of a guiding tree.

We have developed a parallelized algorithm, PMSA, which has a number of differences with the existing programs. It effectively parallelizes all the three stages of the ClustalW algorithm. The details of the algorithm are given in Table 4.3.

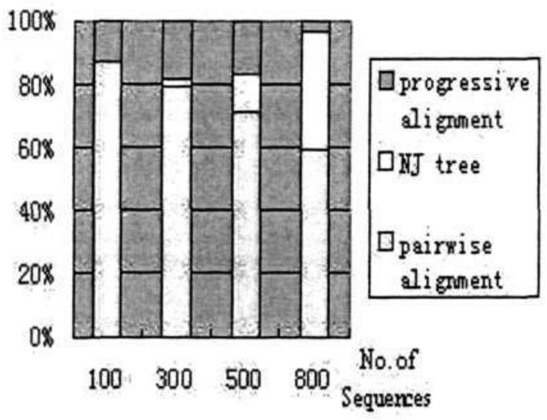


Figure 4.5: Fraction of execution time of ClustalW

Given a set of n sequences, $S = S_1, ..., S_n$, the algorithm firstly builds an $n \times n$ matrix M and each element $M[i, j]$ corresponds to the distance between object i and object j in the original dataset. Since the calculations of the distances between each pair of clusters are computationally independent tasks, a static matrix row-based partition is designed to decompose the tasks into p groups evenly (p is the number of available processors in the cluster) and assign to each processor one such group, as illustrated in Figure 4.6.

Upon the completion of the distance matrix, M, all the pairwise alignment scores are stored. Note that the matrix M is separated in p processors. In the stage of building a guiding tree, working processes find the minimum of the nearest neighbor distances locally in parallel for the number of $n-1$ iterations. The nearest nodes can be defined by minimizing the expression $Dist_{ij} = (n-1)M_{ij} - (R_i + R_j)$, where M_{ij} is the distance between node i and j shown in the distance matrix; R_x is the row sum

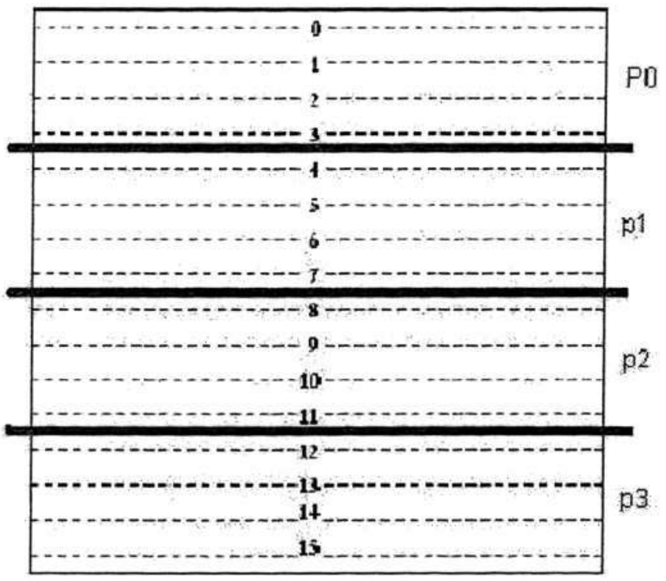


Figure 4.6: Static Row-Based Matrix Partitioning

over row x of the distance matrix; $R_x = \sum_{1\leq k\leq N} M_{xk}$; N is the remaining number of nodes adjacent to the root. The local winner communicates to other processes, and finally performs a redundant search for the global winner, $Dist_{gmini,gminj}$, by using the MPI_Allreduce and MPI_Broadcast. At the end of each iteration, a global synchronization is needed in order to update the matrix for next iteration. The master process P_0 defines a new node, y, whose three branches join $gmini$, $gminj$ and the rest of the tree. The lengths of the tree branch from y to $gmini$ and $gminj$ are defined: $L_{gmini,y} = \frac{M_{gmini,gminj}}{2} + \frac{R_{gmini}-R_{gminj}}{2(N-1)}$, $L_{gminj,y} = M_{gmini,gminj} - L_{gminj,y}$, which are the lengths of the new branches. When the nodes $gmini$ and $gminj$ are joined, they are replaced with a new node, y, with the distance to the remaining nodes given by $M_{yk} = \frac{M_{gminj,k}+M_{gmini,k}-M_{gmini,gminj}}{2}$. For elements may not be in the same process which does the update, at each step of update, it must determine if the elements needed are in the same process and which process should do the update. This non-local data, which is needed for updating, should be fetched by communicating with messages. Note that in this case, the master process $P0$ joins the computations in addition to organizing jobs, in order to reduce communication between processes.

Once a guiding tree is constructed, the master process P_0 analyzes it and identifies sequence pairings. Those which are independent of other groups, according to the guiding tree, are computed in parallel. The master process P_0 then waits for the results and serially completes the multiple alignments. In general, we only can parallelize the alignments of the same level in the tree. When we progress through higher levels of the tree, it is difficult to keep load balancing. So the PMSA parallelizes about 5-10% of the codes for this stage by calculating profile scores in parallel, gaining considerable execution time reduction depending on the size of a problem.

In our algorithm, collective communications instead of point-to-point communications are deployed, which involve all processes in a communicator. The **MPI_Allreduce** is used to combine values from all processes and distribute the result back to all processes. The **MPI_Bcast** is used to broadcast a message from the process with the global minimum to all other processes of the group.

4.2.3 Evaluation of the results

To test the overall scaling of the proposed algorithm for multiple sequence alignment(PMSA) , three datasets with 500, 800 and 1000 sequences of the GPCRs family are used. We ran five trials of the PMSA on 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44 and 48 processors and reported the average execution time. For uniprocessor performance, we used the Clustalw1.81 as a baseline. The results are shown in Figure

Table 4.3: Parallel algorithm to compute multiple alignment

```
–Input :
    A set of $n$ sequences:$S$
    Substitution matrix: $s(a,b)$
    Gap penalty: $g$
    Number of processes: $p+1$ ($p$ is number of working processes)
–Output : Multiple alignment

–Algorithm :
1. Connect to $p+1$ processes and establish each process's rank: $myrank$
2. Master process $P_0$ reads in the set of sequences $S$
3. Master Process $P_0$ sends a block of sequences, gap penalty to the working processes

/*Pairwise alignment*/
4. Each process $P_i$, $0 \leq i \leq p$, calculates $\frac{n}{p+1}$ rows of $M$, which are a block of sequences alignment : $\frac{n(n-1)}{2(p+1)}$
5. Set $N = n$

/*Building NJ-tree*/
6. For $N > 2$ do
        a. Each process computes a block of $R_x$'s for its own sub-matrix, $R_x = \sum_{k=1}^{N} d_{xk}$ (x represents the node for which we are computing now) ($M_{ii} = 0$).
        b. Working processes send back the block of $R_x$ to master process.
        c. Master process collects all the rows' $R_x$ ($1 \leq x \leq N$) and broadcast the $N$ number of $R_x$ to all working processes.
        d. Each working process computes $Dist_{ij}$ on its own matrix (for each i¡j, since the matrix is symmetric) $Dist_{ij} = M_{ij} - \frac{(R_i+R_j)}{(N-1)}$; find the minimum $Dist_{mini,minj}$
        e. Through **MPI_Allreduce** with operation **MPI_MINLOC** to find the global minimum $Dist_{gmini,gminj}$
        f. The process, the owner of the global minimum, broadcasts the $gmini$ and $gminj$ to other processes
        g. Master process defines a new node, $y$, whose three branches join $gmini, gminj$ and the rest of the tree. Define the lengths of the tree branch from $y$ to $mini$ and $minj$: $L_{gmini,y} = \frac{M_{gmini,gminj}}{2} + \frac{R_{gmini}-R_{gminj}}{2(N-1)}$, $L_{gminj,y} = M_{gmini,gminj} - L_{gminj,y}$ , which are the lengths of the new branches.
        h. Processes update the distance of its own matrix from $y$ to each other node ($k \neq gmini$ or $gminj$) as: $M_{yk} = \frac{M_{gminj,k}+M_{gmini,k}-M_{gmini,gminj}}{2}$.
        i. Remove the distances to nodes $mini$ and $minj$ from the matrix, and decrease $N$ by 1.

/*Progressive alignment*/
7. Master process $P_0$ analyzes the guide tree, identifies sequence pairings that can be evaluated independently and sends them to idle processes $P_i$, $1 \leq i \leq p$ to perform alignment.
8. Master process $P_0$ gathers resulting pairs evaluated by processes Pi, $1 \leq i \leq p$ and serially completes the multiple alignments
```

4.7

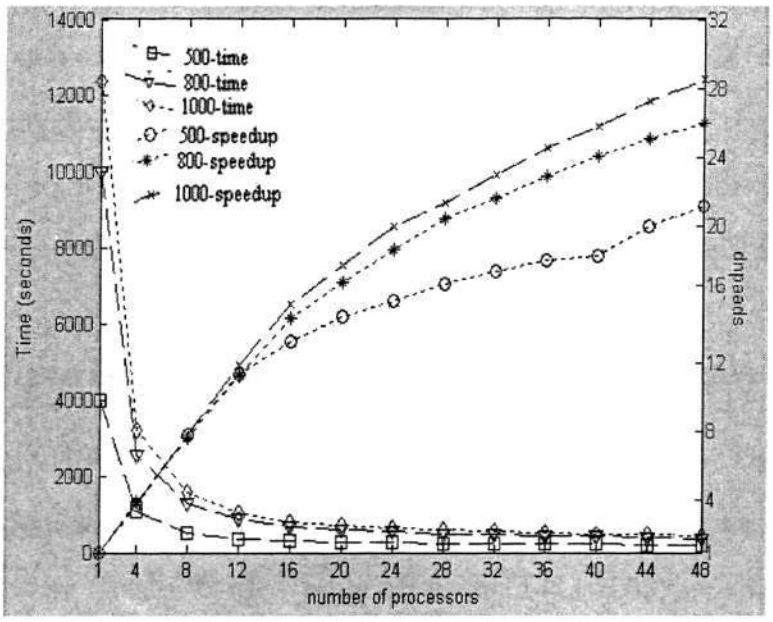


Figure 4.7: Overall scaling of parallelized multiple sequence alignment

In this experiment, speedup of more than 20 times is gained for the three datasets of GPCRs protein sequences when running on 48 processors in this application. Moreover, aligning 1000 GPCRs proteins sequences, total time is reduced from about 4 hours on one processor to about 9 minutes on 48 processors, which no doubt significantly increases research productivity.

It also can be observed from the figure that the parallel version scales up to 15.79 using 16 processors. The fairly flat curve of the speedup at the high end of processor numbers suggests that computational gain from further division of the matrix will be discounted by the overhead communication between the processes. Especially at the stage of building a guiding tree, for the number of $n-1$ iterations, an array of $R_x, 1 \leq x \leq N$, needs to be collected by the master process and broadcast to each process before utilizing it for computing the value, *Dist*. Additionally, a global synchronization is needed in order to update the matrix for next iteration. So there are the number of $n-1$ (n is the number of input sequences) rounds of communications among all the processes. Another strong effect on the speedup is the serial fraction in the stage of progressive alignment. In this stage, only the alignments of the same level in the guiding tree can be parallelized, which is about 5-10% of the codes.

4.2.4 Comparison with previous parallel schemes

As the speedup achieved in the phase of pairwise alignment and that achieved in progressive alignment are quite similar, in the following experiment we only compared the PMSA with the ClustalW-MPI program from Li (Li 2003) on the second stage, building guiding trees, to show its performance. We ran five trials of the PMSA and ClustalW-MPI on 4, 8, 16 and 32 processors on two large datasets with 3990 and 6500 sequences. We also executed the serial ClustalW 1.81 and ClustalW 1.83. As a result, the ClustalW 1.81 produced no results on the datasets with 3990 and 6550 sequences, and the ClustalW 1.83 failed to build a guiding tree on the dataset with 6550 sequences even though it improved the method of building a NJ tree based on the ClustalW 1.81.

From the average execution time presented in Table 4.4, we can see that the PMSA reduces the execution time dramatically while the ClustalW-MPI program does not make any speedup at the phase of building a tree on multiple processors. The results show that the PMSA is superior to the ClustalW-MPI in terms of parallel performance. For example, on 32 processors, the ClustalW-MPI takes more than 7 hours to build a NJ tree, while the PMSA only needed less than 1 hour on the larger dataset.

Table 4.4: Comparison of runtime (in seconds) of ClustalW-MPI_NJ to our PMSA_NJ on 4,8,16 and 32 processors.

Processors No.	3990-sequences		6500-sequences	
	ClustalW-MPI_NJ	PMSA_NJ	ClustalW-MPI_NJ	PMSA_NJ
4	4419	**1257**	26339	**5474**
8	4370	**884**	24591	**4313**
16	4409	**742**	25278	**3141**
32	4504	**697**	25418	**2888**

4.3 Summary and discussions

In this chapter, we presented and evaluated two parallel computational algorithms for sequence comparison. These algorithms allow researchers to compare biological sequences at a much higher speed than the serial methods. Moreover, they also make it possible to analyze problems that were previously considered too large. The generated alignments of these algorithms are the first step in building a phylogenetic tree on the homologous group of proteins. Once an alignment has been completed phylogenetic tree reconstruction itself presents significant computational challenges.

Chapter 5

Reconstruction of MP and ML Phylogenetic Trees

Maximum parsimony (MP) and maximum likelihood (ML) methods are most favored approaches attempted to infer phylogenetic trees for their accuracy. However, their search space is huge - there are $\frac{(2n-5)!}{2^{n-3}\times(n-3)!}$ unrooted trees, where n is the number of taxa. When n increases, there is an incredible increase in the number of possible bifurcating topologies to be evaluated. It is easy to get struck in local optimum, and there can be many optimal trees. Additionally, as sequence length increases, it requires an increased time allocation to compute cost for each topology. It can become very pronounced with likelihood methods that require more complex models of evolution while this may be a relatively minor problem for distance and parsimony methods. In addition, in some cases, there are multiple solutions, each of which is typically saved and further evaluated in order to find a more optimal solution. For instance, an enormous number of optimal trees (more than 100,000) need to be saved and evaluated. Many heuristic algorithms have been proposed to speedup the process of constructing phylogenetic trees. However some of these algorithms do not perform a sufficiently rigorous search of the tree space and often result in sub- optimal trees.

In order to perform a more complete search of the tree space within limited time limit, a parallel divide-and-conquer model (pPhylo) is designed in this chapter.

5.1 Computational tasks

5.1.1 Minimum Parsimony criterion

Given two sequences $X = x_1, x_2, ..., x_n$ and $Y = y_1, y_2, ..., y_n$ of the same length, the hamming distance H(X,Y) between them is defined as the number of different pairs (x_i, y_i). Let T be a tree whose nodes are labeled by sequences of length k over Σ; $H(e)$ be the hamming distance of each edge e in the tree T, then the parsimony length of the tree T is defined by $\sum_{e \in E(T)} H(e)$. We outline the MP algorithm as following:

Outline of Maximum Parsimony algorithm

- *Input:* Set of S of n aligned sequences, each of length k.
- *Output:* A phylogenetic tree T.
- *Algorithm:*
  - Bijectively leaf-labeled by elements from S, and additional sequences of length k labeling the internal nodes of T such that the parsimony length of T is minimized over all such possible leaf-labeled phylogenetic trees. That is $\sum_{e \in E(T)} H(e)$ is minimized.

5.1.2 Maximum Likelihood criterion

ML is a powerful technique to investigate phylogeny, however the exhaustive search for all tree topologies extremely is compute-intensive. Previous studies have shown that large datasets are challenging for MP heuristics implemented in these packages (Roshan et al. 2004) (U.Roshan, Moret, Williams & Warnow 2004*b*). We can expect the same or even worse because ML is a harder problem. The outline of maximum likelihood is shown as following:

Outline of Maximum Likelihood algorithm

- *Input:* Set of k aligned sequences at the leafs $S_1, ..., S_k$
- *Output:* A phylogenetic tree T.
- *Algorithm:*

  1. Pick a model of evolution: $P(S_i|S_j, t)$,$1 \leq i, j \leq k$
  2. Enumerate all possible tree topologies T with k leafs.
  3. For each T, maximize over all possible edge length t:

     $P(S_1, ..., S_k|T, t) = \sum_{S_{k+1},...,S_{2k-1}} P(S_1, ..., S_{2k-1}|T, t)$, (for there are $2k-1$ internal nodes).
  4. Return the tree T with the highest probability, P.

5.1.3 Models of base substitution

It is essential to have a mathematical model to understand the mechanisms of change and is required to estimate both the rate of evolution and the evolutionary history of sequences. One of the main advantages of maximum likelihood over other methods is that it permits using complex evolutionary models to estimate model parameters and thus makes inferences of evolution simultaneously.

Currently, there are a number of amino acid substitution models. These models can be classified according to the number of different substitution types they allowed for and whether they incorporate different or equal base frequencies. The most general model of a time reversible nucleotide substitution process is the General Time Reversible (GTR) model (Rodriguez, Oliver, Marin & Medina 1990) (Lanave, Preparata, Saccone & Serio 1984), in which, probabilities for each substitution are different and base frequencies are unequal. From this model, all simplified models can be obtained by further restricting the parameters.

The HKY85 model (Felsenstein 1980) allows unequal base frequencies but only transitions and transversions have different probabilities.

The K2P model is derived from the HKY85 model by setting equal base frequencies, while the Felsenstein model (Felsenstein 1981*b*) is derived from the HKY85 by setting same substitutions probabilities.

The JC model (Jukes & Cantor 1969) is the simplest model that has equal base frequencies, 0.25, and only one type substitution. There is a tradeoff in complexity between simple and elaborate models of nucleotide substitution among general tree scoring methods such as distance-based methods, MP and ML. When a model is more elaborate, more parameters have to be estimated, It will get more accurate result at cost of more modeling time. An overview of the most common models is provided in Figure 5.1.

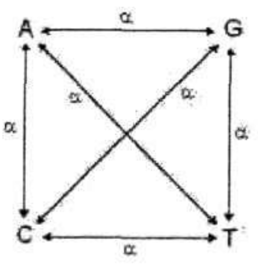
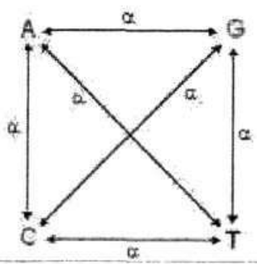
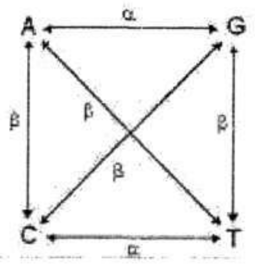
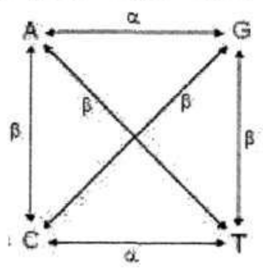
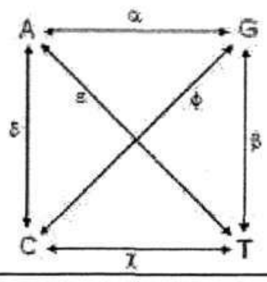
Model	Transition probability matrix	Stationary nucleotide frequencies	Number of parameters
Jukes-Cantor model		All substitutions have an equal probability and base frequencies are equal	1
Felsenstein		All substitutions have an equal probability, but there are unequal base frequencies	4
Kimura 2 parameter model (K2P)		Transitions and transversions have different probabilities, but there are equal base frequencies	3
Hasegawa, Kishino & Yano (HKY)		Transitions and transversions have different probabilities, base frequencies are unequal	5
General time reversible model (GTR)		Different probabilities for each substitution, base frequencies are unequal	9

Figure 5.1: Substitution models

5.1.4 Tree rearrangement

Hill-climbing search is the most popular technique used by biologists for finding better trees, which is shown in the following four steps:

1. Construct an initial tree.

2. Construct a set of "neighboring trees" by making small rearrangements of the initial tree.

3. If any of the neighboring trees are better than the initial tree, then select it/them and use as a starting point for new round of rearrangements. It is possible that several neighbors are equally good.

4. Repeat steps 2 and 3 until a tree that is better than all of its neighbors is found.

There are three main methods used to move from one tree topology to another:

Nearest Neighbor Interchange (NNI)– NNI rearranges two adjacent branches per internal branch. A tree with n taxa has *2(n-3)* neighbors. An example is shown in Figure 5.2.

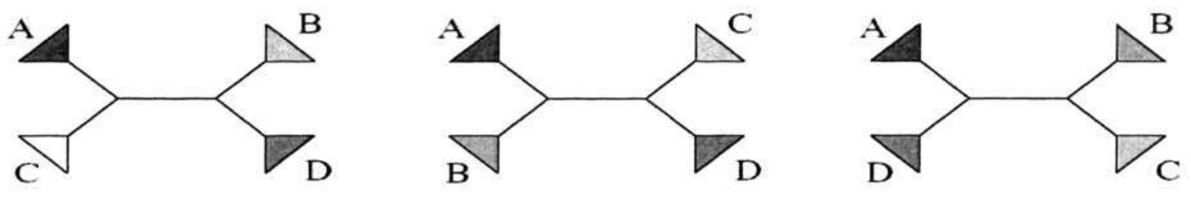


Figure 5.2: Neighbor interchange options referred to as (left to right) AC—BD, AB—CD, AD—BC, of an unrooted tree with 4 subtrees A;B;C; and D.

Subtree Pruning and Regrafting (SPR)–SPR (Maddison 1991) removes a branch with a subtree from a tree, and adds it between two nodes somewhere else in the tree, which is shown in Figure 5.3

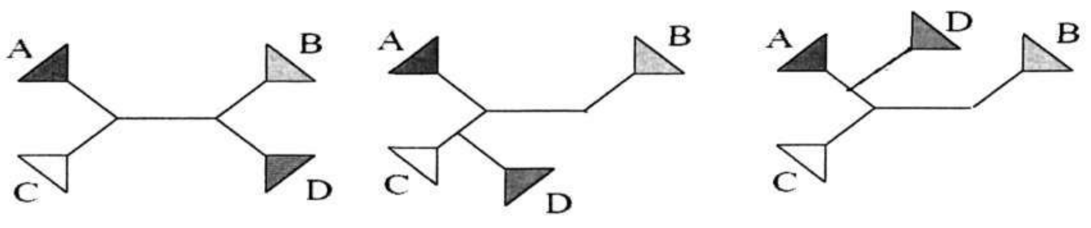


Figure 5.3: Break a branch, remove the subtree D, and attach it to one of the other branches

Tree Bisection and Reconnection(TBR)–TBR (Maddison 1991) is the most popular method. It splits a tree into two subtrees and connects both parts between all possible nodes of the other as shown in Figure 5.4. TBR could explore more trees than NNI and SPR for the neighborhood of a tree induced by them is as subset of TBR's (Maddison 1991) (Allen & Steel 2001).

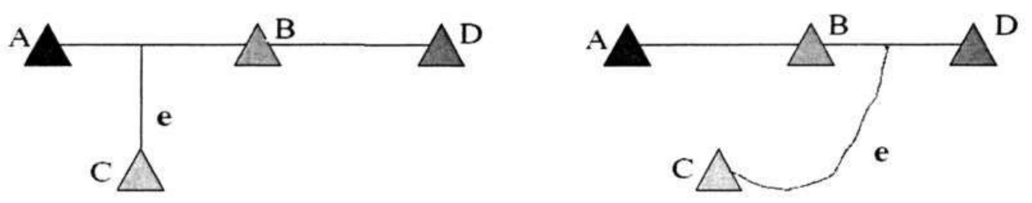


Figure 5.4: Remove an edge *e* from tree, and then reconnected by creating a new edge between the midpoints on edge in subtree C and D

NNI, SPR and TBR tree rearrangements have been compared by Roderic Page (Page 1993). From his examples, it can be concluded that TBR and SPR are superior than NNI to escape local optimum for the neighborhoods induced by TBR and SPR are larger than NNI.

5.1.5 TNT

Among the current heuristic techniques for solving MP on large datasets, TNT (Tree analysis using New Technology) performs very well ((Goloboff 1999), (U.Roshan, Moret, Williams & Warnow 2004*a*), (P.Hovenkamp 2004), (R.Meier & Ali 2005), (G.Giribet 2005)). In addition to a very fast implementation of hill-climbing heuristics, TNT implements other search strategies, such as divide-and-conquer and genetic algorithms, which allow the analysis of large datasets in a reasonable time limit (much faster than other software packages). It is better than PAUP* (a very popular software package used in the phylogenetic research community) in searching for MP trees (refer to (Roshan 2004)). In TNT package, various search heuristics of smarter local search are implemented, such as:

Parsimony ratchet(Nixon 1999)– It uses TBR (refer to 5.1.4 for more details) hill-climbing method, to search MP tree. When it reaches local optimum, it modifies the input data to move out of local optimum, then runs TBR hill-climbing on the new data. After the new data reaches local optimum, the dataset is changed back and continues to do hill-climbing.

Drift –It uses a traditional simulated annealing technique to escape local optimum. When it reaches local optimum, it will move to trees with a worse score according to a probability. Then it continues to do hill-climbing on these trees.

Sectorial search –It computes smaller subsets of a dataset, then solves them using TBR hill-climbing.

5.1.6 Disk-Covering methods

Disk Covering Methods (DCM) (Huson, Nettles & Warnow 1999) (Huson, Vawter & Warnow 1999) (Warnow, Moret & St. John 2001) (U.Roshan et al. 2004*a*) are a different class of methods for solving MP on large datasets. They divide a problem into smaller subproblems and solve them serially. DCMs are booster methods in the sense that they improve upon the base method by applying it smaller instances of the subproblems.

For different decomposition methods, there are variants of DCMs. DCM1 (Huson, Nettles & Warnow 1999) was designed for use with distance-based method and yields better results than several distance-based methods. The second DCM (Huson, Vawter & Warnow 1999), DCM2, was designed to speed-up heuristic searches for MP trees. The results have proven that DCM2 would improve MP on small datasets.

DCM1 can produce small enough subproblems in size, but the structure induced by the decomposition is often poor. DCM2 overcomes the drawback of DCM1 but the resulting subproblems are too large. Therefore, it is too slow to produce results when boosting MP on large dataset. Both the decomposition of DCM1 and DCM2 are based on a distance matrix computed on the dataset so that they can produce only one time of decomposition.

DCM3 is designed to produce smaller size of subproblems than DCM2, which decomposition is obtained on the dynamically updated *guide tree* and will produce different decomposition for different trees. Since DCM3 uses a *guide tree*, it can iteratively improve the *guide tree*.

The Rec-I-DCM3 (U.Roshan et al. 2004*a*) (U.Roshan et al. 2004*b*) method is designed to use iteration for escaping local optimum, and recursive-DCM3 to enable further localization and reduction in problem size. Their study shows that Rec-I-DCM3 convincingly outperformed the unboosted default heuristics of TNT (Goloboff 1999) on all datasets and at all the limited time allotted for computation. The Rec-I-DCM3 method is the first technique that allows a successful application of parsimony heuristics with high accuracy within reasonable time limits. The current study of it focuses mainly on techniques for MP. However, ML is a harder problem than MP because it is not known how to compute the ML score of a given tree in polynomial time.

5.2 Problems in previous parallel computing

Despite the fact that parallel implementations of MP or ML programs are technically solid in terms of performance and parallelization techniques, they significantly drag behind algorithmic development. That is, programs are parallelized that mostly do not represent the state-of-the-art algorithms any more and are out-competed by the most recent serial algorithms in terms of final tree quality and CPU time. For example, the largest tree computed with parallel fastDNAml (Stewart, Hart, Berry, Olsen, Wernert & Fischer 2001), which is based on the fastDNAml algorithm (ROlsen, H. Matsuda & Overbeek 1994) contains 150 taxa. There also exists a distributed implementation of this code (Hart, Grover, M.Liggett, Repasky, Shields, Simms, Sweeny & Wang 2003). The same holds for a technically very interesting JAVA-based distributed implementation of fastDNAml: DPRml (Keane, Naughton, Travers, McInerney & McCormack 2005). In addition to using an old search algorithm, significant performance penalties are caused by using JAVA in terms of both memory efficiency and speed of numerical calculations. Those language-dependent limitations will become more significant when trees comprising more than 417 taxa (currently largest tree with DPRml, personal communication) are computed with DPRml. The authors of DPRml are however well-aware of those limitations (personal communication) and plan to integrate algorithmic concepts from RAxML and PHYML into DPRml.

The technically challenging parallel implementation of TrExML (Wolf, S.Easteal, Kahn, McKay & L.Jermiin 2000) (Zhou, Till, Zomaya & Jermiin 2004) has been used to compute a tree containing 56 taxa. However, TrExML is probably not suited for computation of very large trees since the main feature of the program consists in a more exhaustive exploitation of search space for medium-sized alignments. Due to this exhaustive search strategy the execution time increases more rapidly than for other programs with the number of taxa. The largest tree computed with the parallel version of TREE-PUZZLE (Schmidt, Strimmer, Vingron & Haeseler 2002) contained 257 taxa due to the limitations caused by the data structures used (personal communication). As already stated, TREE-PUZZLE provides mainly advantages concerning quality-assessment for medium-sized trees. M.J. Brauer et al. (Brauer, Holder, Dries, Zwickl, Lewis & Hillis 2002) have implemented a parallel genetic tree-search algorithm that has been used to compute trees of up to 228 taxa. In addition, there exist the previous parallel and distributed implementations of RAxML (Stamatakis et al. 2004) (Stamatakis, Ott, Ludwig & Meier 2005). To the best of our knowledge, Parallel RAxML has been used to compute the largest ML-based phylogeny to date containing 10.000 organisms on a medium-sized PC cluster using approximately 3.200 accumulated CPU hours. Finally, RAxML has also been parallelized for SMPs with OpenMP exploiting fine-grained loop-level parallelism (Stamatakis, Ott & Ludwig 2005). This approach has limited scalability however, beyond 4-way CPUs and is mainly intended to solve memory problems and increase cache efficiency for the computation of very large alignments.

Therefore, in order to reconstruct, with high accuracy, phylogenetic trees at a much larger scale, further speedup and improvements are imperative.

5.3 A parallel divide-and-conquer model

In this section, a parallel divide-and-conquer model is designed for distributed memory clusters.

5.3.1 Overall structure

The task-scheduling mechanism of the model is based on a master-worker architecture, which consists of four main steps.

1. At the beginning, a master process reads in an alignment file with a starting tree (an initial guide tree) which is computed with the randomized parsimony component. Then, it performs recursively division of the main problem into smaller subproblems, and stores the merging order (subset-guidetree, *rurTree*) which is required to correctly execute the merging step. The decomposition is illustrated in function **pPhylo_divide** in full details. Some definitions used are shown as following:

   Short subtree Suppose there is a tree T with an edge e in it. Let Q_1, Q_2, Q_3 and Q_4 be the four subtrees around e; q_1, q_2, q_3 and q_4 be the set of leaves closet to e in each of the four subtrees respectively. The distance between them is measured by the hamming distances (see section 5.1.1) on the edges. The set of nodes in $q_1 \bigcup q_2 \bigcup q_3 \bigcup q_4$ is the short subtree around e.

Short subtree graph Short subtree graph is the union of cliques formed on "short subtrees" around each edge in T.

Separator Separator is the short subtree of a special edge, which would produce the most balanced bipartition of the leaves in tree T when removed.

Outline of pPhylo_divide

- *Input:*
  - Set of k sequences $S = S_1, S_2, ..., S_k$
  - Maximum subset size MS
  - Starting tree T
- *Output:* Set of subproblems, $allsubsets = A_1, A_2, ..., A_m$ (m is the total number of subsets)
- *Algorithm:* Recursively divide a set of k sequences S into subproblems
  - (a) Compute edge weighting for each edge by using the hamming distances.
  - (b) Compose **short subtree graph** around edges by selecting set of all leaves that are elements in a short quartet around an edge, that is $sub_1, sub_2, ..., sub_x$ (where x is the number of subsets).
  - (c) Find a **separator**, *spt*, by selecting an edge that when removed, produces the most balanced bipartition of the leaves as centroid edge, E_c. The *spt* is the leaves of the **short subtree** around E_c. The subsets are then defined to be $A_i = spt \cup sub_i, 1 \leq i \leq x$.

```
(d) For $A_i$ $(1 \leq i \leq x)$
        If ($A_i's$ size $> MS$){
            Let $T|A_i$ be the result of restricting tree T to $A_i$ for each i.
            /*Recursively compute the subsets for $A_i$ */
            **pPhylo_divide** ($A_i$, MS, $T|A_i$)
        }
        Else{
            Add $A_i$ to *allsubsets*.
            Re-build subset-guidetree, *rurTree*.
    /*Produce a subset-guidetree, *rurTree*, to keep the merge order. The
    *rurTree* is expressed in a string format that uses parenthesis to start
    and end subtree groups, commas to separate group members, and
    subproblems names to name tree leaves.*/
        }
```

The division into subproblems is executed recursively until all subproblems contain less taxa than the user-specified size of the maximum subproblem, MS. The subproblems, being smaller in size and evolutionary diameter, are easier and faster to analyze than the full dataset. These subproblems are also independent to be analyzed in multiple processes.

2. All individual subproblems are then dynamically dispatched to the working processes, which locally solve them by using the function: **pPhylo_subtree**, then return the respective subtrees to the master process. For there are different *base* methods to build MP or ML subset trees, more details in this function will be given in section 5.4.1 and 5.5.2. Here is a general description, where $T|A_i$

is the result of restricting tree T to A_i for each i.

Outline of pPhylo_subtree

- *Input:*
  - Set of y sequences in A_i
  - Starting subset tree $T|A_i$
- *Output:* MP or ML subset tree
- *Algorithm:*
  - Build a subset tree for subproblem by using a *base* method (MP or ML).

3. Once all subproblems have been solved, the subset trees are merged serially (pairwise at a time) in the order determined by subset-guidetree, *rurTree*, into the *new guide tree.* We design a **stack** structure in order to read out subset trees according to subset guide-tree. The detail of merging method, **pPhylo_merge**, is explained below:

Outline of pPhylo_merge

- *Input:*
  - set of m subset trees, $ST = st_1, st_2..., st_m$
  - subset guide-tree, *rurTree*
- *Output:* supertree, *T'*
- *Algorithm:* Use a postorder tree walk algorithm to search subset-guidetree, *rurTree*, in order to merge subset trees into a suptertree, *T'*.

```
(a) set char* $ptr$=$rurTree$;
    /*Design a stack structure to read out subset trees $st_1, st_2, ..., st_m$ according to subset guide-tree, $rurTree$.*/
(b) While(*ptr!=NULL){
    i. Switch(*$ptr$){
      A. Case '(':
            /*push '(' into the stack*/
            Push($st_i$) ;
            Break;
      B. Case '$st_i$':
            /*Push subset tree $st_i$ into the stack*/
            Push($st_i$) ;
      C. Case ')':
            Do{
               /*pop out subset trees between '(' and ')' from stack*/
               Set y=Popout();
            }while(y!='(')
         /* Merge these subset trees pairwise at a time serially.*/  $T' =$ $\mathbf{SCM}(st_i, st_y)$
            Push($T$');
      D. default:
            break; }// end of switch
    ii. $ptr$++; }// end of while
```

The Strict Consensus Merger(SCM) (Huson, Nettles & Warnow 1999, Huson, Vawter & Warnow 1999) described below, is used to do combine two subset trees into a single tree.

Strict Consensus Merger SCM method takes two trees t_1 and t_2 on possibly different leaf sets, identifies the a share leaf set X, and computes the strict consensus, t_x, of t_1 and t_2, each restricted to the leaf set X, finally adds the remaining taxa from t_1 and t_2 into t_x, so that the two trees can be merged together with strict consensus.

4. At the last phase, a hill-climbing MP or ML search on the supertree, T', is applied to do a global rearrangement as described in function **pPhlo_grearrange**. Full details of this function designed for MP and ML methods will be presented in the sections 5.4.1 and 5.5.2.

Outline of pPhlo_grearrange

- *Input:*
  - Set of k sequences, $S = S_1, S_2, ..., S_k$
  - Supertree tree, T'
- *Output:* MP or ML phylogenetic Tree, T_{best}
- *Algorithm:*
  - Apply a global search method (MP or ML) starting from T' until we reach a local optimum.

At the end, the master process verifies if the specified amount of iterations have already been executed and if so terminates the program. Otherwise, it will initiate a new round of subproblem decomposition, subproblem inference, subset trees merging, and global optimization. Note that the time required for subproblem decomposition and subproblem merging is negligible compared to MP or ML inference time of the subset trees and guide tree.

5.3.2 Dynamic distribution of tasks

In the proposed model, the master process distributes tasks dynamically. Each working process enters an infinite loop. It firstly sends a *READY* message to the master process to inform that it has ready to receive any message. When it receives an *END* message, the process cleans up and exits. Otherwise, it receives a *SUBSET* as a task to construct a subset tree by using a *base* method. If a working process finishes its task, it will send back *SUBSET TREE* result.

The master process keeps a count of how many working processes are available and sends out a *subset* to a working process when the process identifies itself as *READY*. If there are no more new tasks, this routine sends *END* message to all working processes. Finally, the master waits until all working processes are idle (that is, it receives a *READY* message from all working processes) and then sends a final message with the special value *END* to all working processes.

It is very clear that the load balancing cannot be a problem in this paradigm since each process takes a new task as soon as it is free. Note that in this model, the master process does not compute subtrees. The main framework is outlined as following:

Outline of pPhylo

- *Input:*
  - $S = S_1, S_2 ..., S_k$ of aligned biomolecular sequences, k is the number of

sequences

- n, the number of iterations
- T, starting tree
- MS, maximum subproblem size
- p+1, the number of available processes, p is the number of working processes

- *Output:* A MP or ML phylogenetic Tree, T_{best}.
- *Algorithm:*
  - Connect to $p+1$ processes and establish each process's rank No.: $myrank$
  - On master process (if $myrank = 0$)
    1. Initialize a subset-guidetree, *rurTree* and *allsubsets*, to record recursive calls as the topology for merging subtrees and to save a total set of subproblems.
    2. For n iterations do
       (a) **pPhylo_divide(S, MS, T)**
       /*Construct a recursive DCM3 decomposition using $T|S$ (a guide-tree tree on dataset S) as the guide tree to produce a total set of subproblems, $allsubsets = A_1, A_2, ..., A_m$ (m is the total number of subsets). Produce a subset-guidetree, *rurTree*, to keep the merge order.*/
       (b) For m subsets do
       Keep a count of how many working processes are available and **Send** out a *SUBSET* to a working process when the process identifies itself as *READY*.

Master process **Collect**s *SUBSET TREE*s from working processes.

(c) If there are no more new tasks, master process waits until all working processes are idle and then **Sends** a final message with the special value *END* to the all salve processes.

(d) **pPhylo_merge(ST, *rurTree*)**

/*Use a postorder tree walk algorithm to search subset-guidetree, *rurTree*, in order to merge subset trees into a suptertree, *T'*.*/

(e) **pPhylo_grearrange(*T'*, *S*)**

/* Apply a global search method starting from T' until we reach a local optimum.*/

(f) Set $T' = T\_best$.

(g) **Broadcast** the new *T'* to every available process for next iteration.

- On working processes (if $myrank > 0$)

1. Enter an infinite loop. Firstly **Send** *READY* to the master process to inform that it has ready to receive any message. Then, **Recv** *SUBSET* from the master process as tasks to construct subset trees by using:

   pPhylo_subtree(A_i, $T|A_i$).

2. If finish a task, **Send** back a *SUBSET TREE* to the master process.

3. When **Recv** an *END* message from master process, this process cleans up and exits.

5.4 MP phylogenetic tree reconstruction

Although Rec-I-DCM3 would reconstruct MP trees on large datasets of up to 14,000 taxa, a drastic improvement is still necessary in order to achieve better accuracy in less time. In this section, we show how the proposed model working with MP methods improves the performance of reconstructing trees. For convenience, we call it as pPhylo(MP). For this study we use the serial TNT program (Goloboff 1999), the best implemented MP heuristic, as the *base* method and TBR as the *global* search method (see section 5.1.5 and 5.1.4 for details regarding the use of TNT and TBR). Note that, the MP criterion returns those binary trees with the lowest parsimony score.

5.4.1 Working with MP–pPhylo(MP)

We now describe the two re-implemented functions dedicated for MP method, **pPhylo_subtre** and **pPhylo_grearrange**, in the pseudo code below for more details. Each working process computes subtrees on subproblems within a batch using the TNT method shown in **pPhylo_subtree for MP**. As the final step, **pPhylo_grearrange for MP** applies a hill-climbing MP search on the supertree until it reaches local optimum. Note that this part of the code takes a non-significant amount of time because it is doing a search on the full dataset, which is very large. It is a bottle-neck in the execution time of pPhylo(MP). However, in the next section of pPhylo boosting ML method, we improve this step and obtain better performance.

Re-implementation of pPhylo_subtree for MP

- *Input:*

  - Set of y sequences in A_i
  - Starting subset tree $T|A_i$
- *Output:* MP phylogenetic subset tree
- *Algorithm:*
  - Apply the base heuristic *basic* method, TNT, to $T|A_i$ to compute the subset tree.

Re-implementation of pPhylo_grearrange for MP

- *Input:*
  - Set of k sequences, $S = S_1, S_2, ..., S_k$
  - Supertree tree, T'
- *Output:* MP phylogenetic Tree, T_{best}
- *Algorithm:*
  - Apply TBR search starting from T' until it reaches a local optimum, let T" be the resulting local optimum
  - Set T_{best}=T"
  - Broadcast T_{best} to working processes for iterative improvement.

5.4.2 Experiment design

Methodology

To the best our knowledge, Rec-I-DCM3, is the best known technique heuristic for solving MP. Therefore, we ran the Rec-I-DCM3 to compare pPhylo(MP) against. We studied two versions of pPhylo(MP): one on 4 processes that we call P4-pPhylo(MP) and one on 10 processes which we call P10-pPhylo(MP). On each dataset we ran five trials of Rec-I-DCM3, P4-pPhylo(MP) and P10-pPhylo(MP).

Test datasets

In our experiments, we used a large variety of biological datasets ranging in size and type (DNA or RNA). All of the very large datasets are above 6K.

- Dataset1: 6,281 aligned small subunit ribosomal Eukaryotes RNA sequences (1,661 sites) (Wuyts, de Peer, Winkelmans & Wachter 2002).
- Dataset2: 6,458 aligned 16s ribosomal Firmicutes (bacteria) RNA sequences (1,352 sites) (Maidak 2000).
- Dataset3: 6,722 three-domain rRNA sequences from Robin Gutell (1122 sites) (Maidak 2000).
- Dataset4: 7,769 aligned ribosomal RNA sequences (851 sites) from three phylogenetic domains, plus organelles (mitochondria and chloroplast), obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.
- Dataset5: 11361 set of all bacteria ssu rRNA sequences from the European rRNA database (1, 360 sites) (Maidak 2000).
- Dataset6: 13921 proteobacteria 16s rRNA sequences from the RDP (1359 sites) (Maidak 2000).

The respective maximum subset sizes of pPhylo(MP) adapted to the size of each dataset are indicated in Table 5.1. In Roshan's PhD thesis (Roshan 2004), these

Table 5.1: Maximum pPhylo(MP) subproblem sizes

Dataset	Maximum subproblem size/dataset size
Dataset1–4	1/4
Dataset5–6	1/8

subset sizes were shown to perform well in comparison to other ones on these datasets. We constructed five starting trees for each trial using a randomized greedy heuristic for MP and each trial was given the same starting tree.

Implementation and platform

Our experiments were performed on a cluster of 4 customized compute nodes, each with 4 Intel Itanium 733MHz processors, PCI 66 MHz I/O bandwidth and 266MHz data bus frequency. (Note that the full-instrumented pPhylo(MP) requires a minimum of two processes).

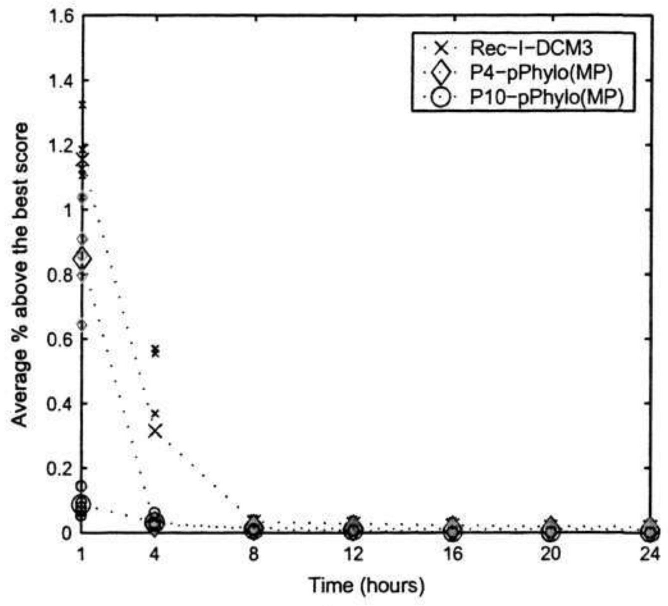
5.4.3 Experimental results

Comparing MP scores as a function of time

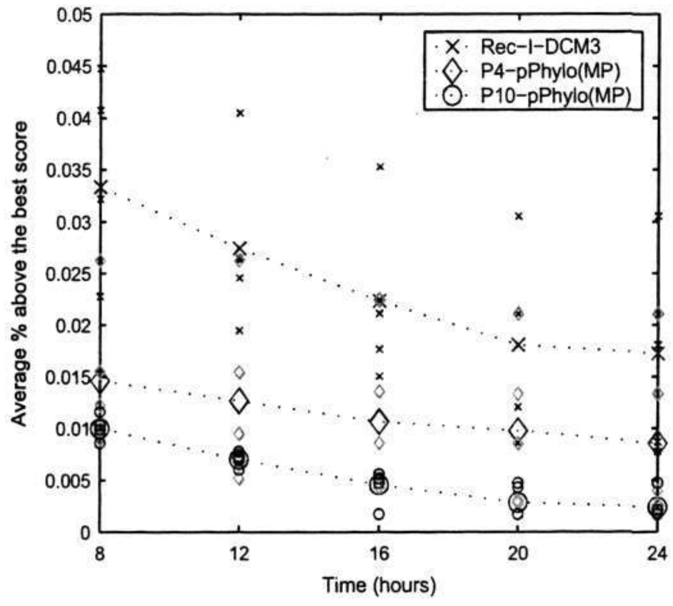
In our experiments we wanted to see which method approaches the best known score on each dataset the fastest. In general the closer the scores are to the "optimal" or best known scores on a given dataset, the more closely related we can expect trees of that score to be topologically. And the further they are from the best known score their topological divergence can increase. Refer to (T.L.Williams, B.M.E.Moret, T.Berger-Wolf, U.Roshan & T.Warnow 2004) for an experimental study of the relationship of tree topologies and MP scores. Thus, we want to get as close as possible to the optimal MP score on a given dataset.

In Figures 5.5 through 5.10 we plot the average MP score above the best known score as a percentage of the best known score on each dataset. The best known scores have been found by previously doing much longer analysis of serial Rec-I-DCM3 on these datasets. Our results in Figure 5.5 through 5.10 show the average MP scores above the best score, as a percentage of the best score on the given datasets. We first note that the average scores of P4-pPhylo(MP) are better than average scores of Rec-I-DCM3 on every point in time. However, the average score found by P10-pPhylo(MP) is better than the best score of the best trial of Rec-I-DCM3 and the average trial of P4-pPhylo(MP) at the end of 24 hours. On datasets 1 to 4, P10-pPhylo(MP) has a much lower score than Rec-I-DCM3 and P4-pPhylo(MP) at hour 1. We see an immense improvement there. However, on datasets 5 and 6, the hill-climbing search takes too long due to the much larger sizes of the datasets. Therefore we do not see this drastic drop in scores initially as we do for datasets 1 through 4.

Interestingly, on dataset 1 the best P4-pPhylo(MP) trial and the P10-pPhylo(MP) trial find a better score than the best known one so far. Achieving the best known score took about 123 hours by the serial Rec-I-DCM3, whereas the pPhylo(MP) found it within 24 hours. Since this is the smallest dataset we used, the hill-climbing search was not much of a bottleneck. As a result we see that the parallel runs were able to achieve many more iterations and thus find a very good score very quickly.

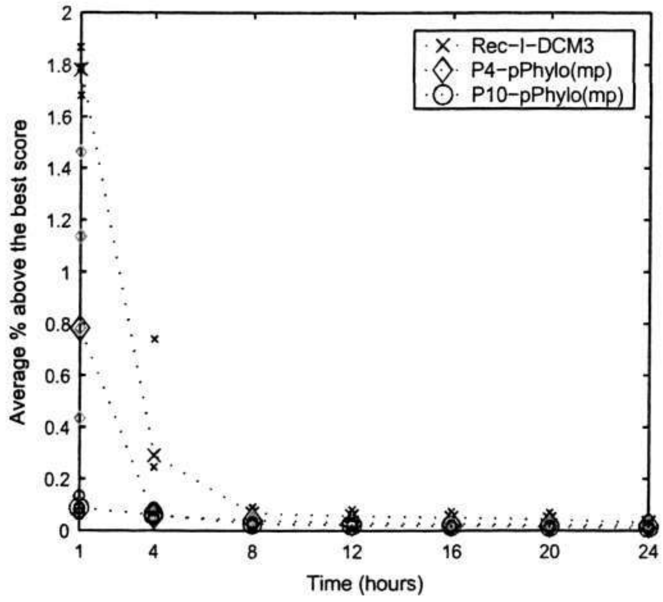


(a)

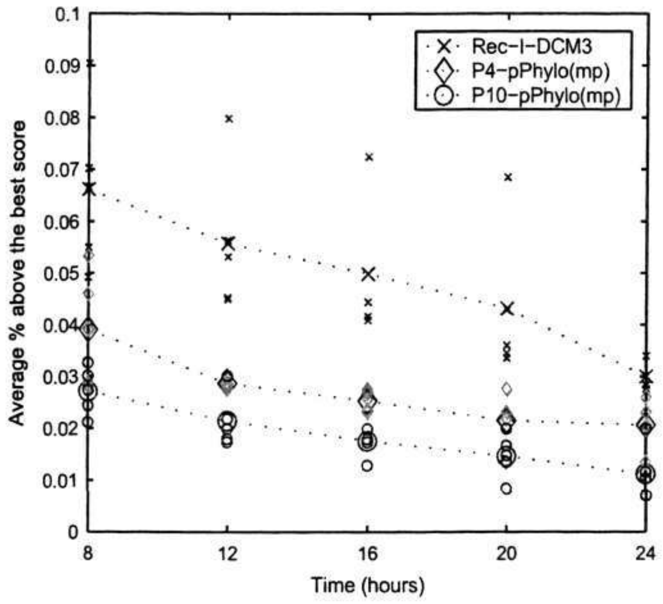


(b)

Figure 5.5: The average MP score of each method above the best known score as a percentage of the best known score on Dataset1

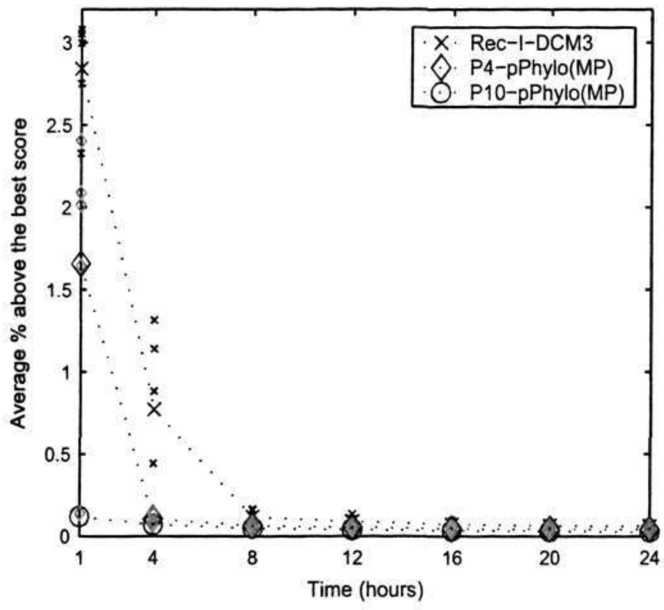


(a)

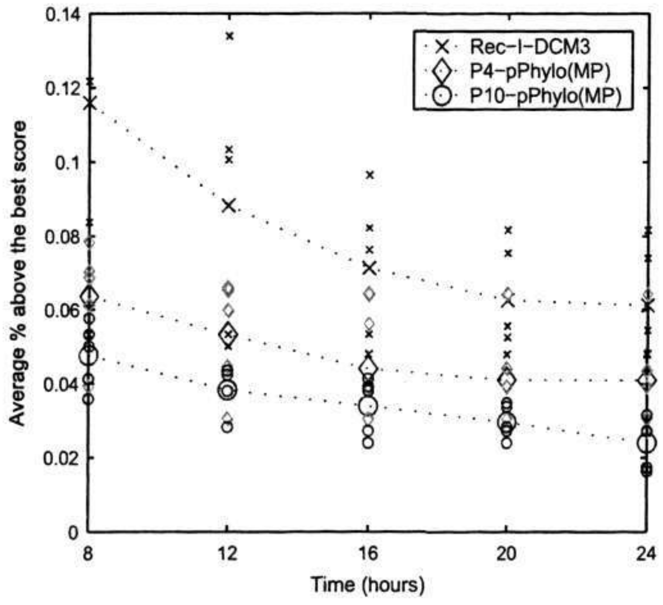


(b)

Figure 5.6: The average MP score of each method above the best known score as a percentage of the best known score on Dataset2



(a)



(b)

Figure 5.7: The average MP score of each method above the best known score as a percentage of the best known score on Dataset3

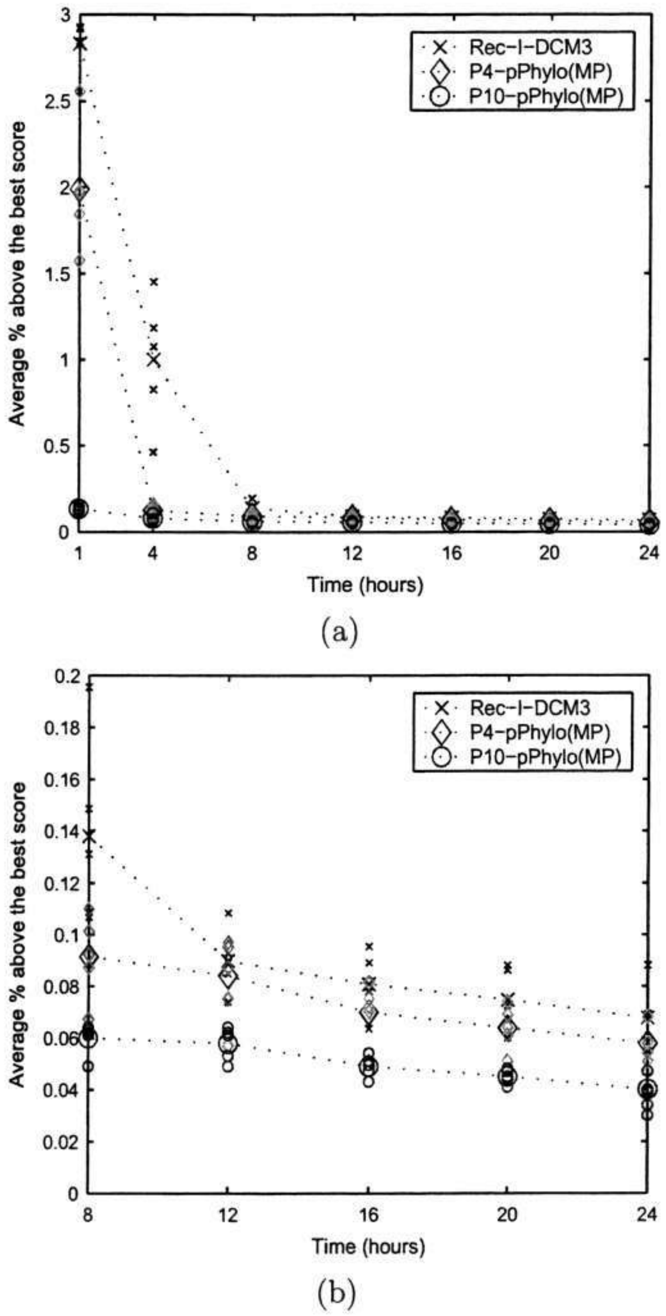
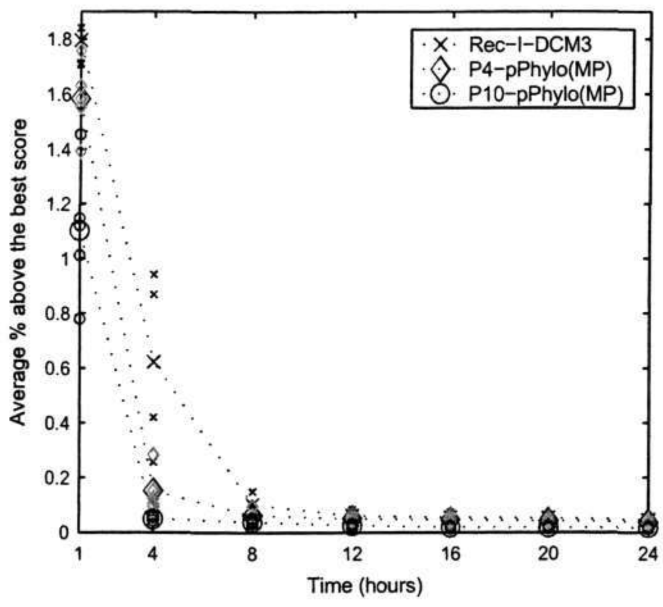
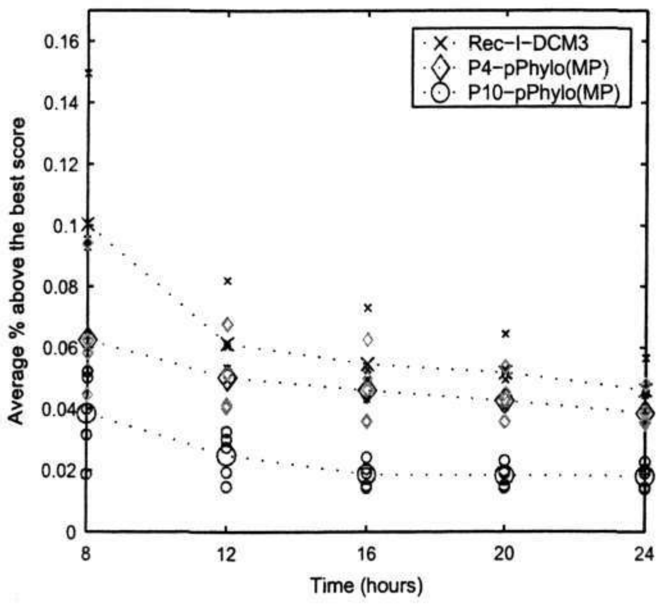


Figure 5.8: The average MP score of each method above the best known score as a percentage of the best known score on Dataset4

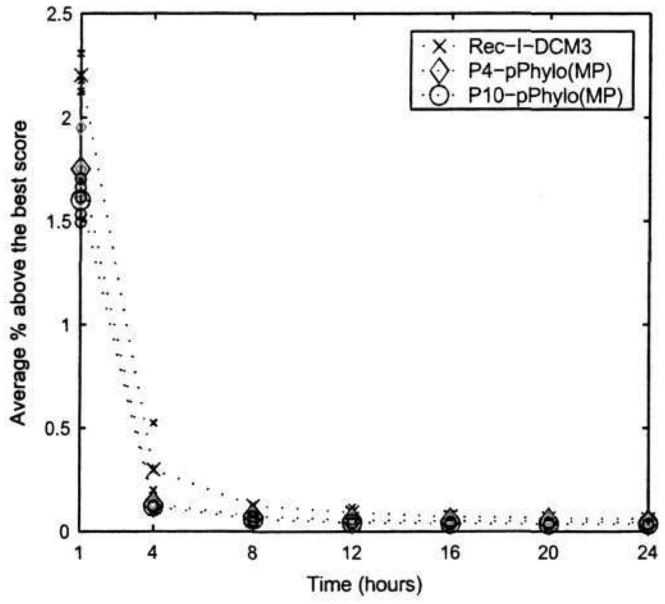


(a)

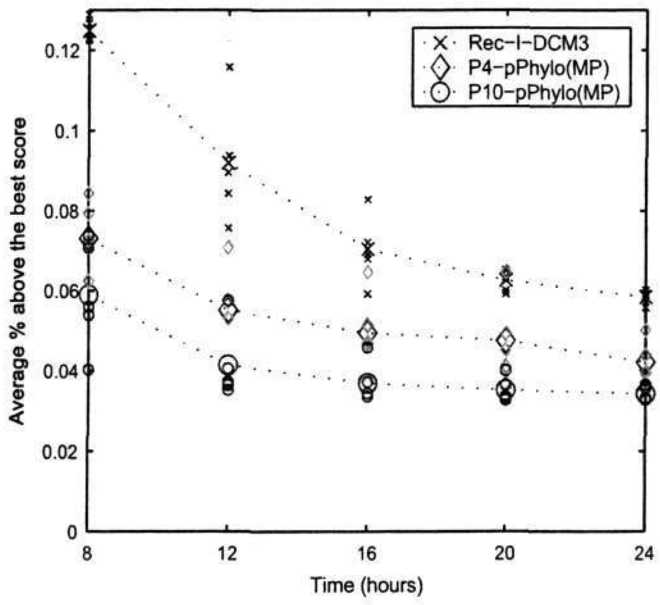


(b)

Figure 5.9: The average MP score of each method above the best known score as a percentage of the best known score on Dataset5



(a)



(b)

Figure 5.10: The average MP score of each method above the best known score as a percentage of the best known score on Dataset6

Table 5.2: The best scores found over all five trials of Rec-I-DCM3, P4-pPhylo(MP), and the single trial of P10-pPhylo(MP) at the end of 24 hours.

Dataset No.	Best known score	Rec-I-DCM3	P4-pPhylo(MP)	P10-pPhylo(MP)
1	232616	232618	232580	232580
2	156192	156235	156213	156203
3	91874	91918	91899	91890
4	99815	99870	99866	99845
5	272047	272157	272142	272085
6	240921	241064	241010	241001

Comparing the score of the best run of serial and parallel

In Table 5.2, we compare the best scores found in 24 hours by Rec-I-DCM3, P4-pPhylo(MP), and P10-pPhylo(MP). We also include the best know scores ever found on these data-sets by analyses that were ran for a week on much faster machines. The table shows that P10-pPhylo(MP) finds the best scores on larger Dataset (except dataset1) at the end of 24 hours.

Speedup of parallel over serial

- Scaling of pPhlo_MP

  We compare the time taken by one iteration of Rec-I-DCM3, P2-pPhylo(MP) (pPhylo(MP) over two processes), P4-pPhylo(MP), and P10-pPhylo(MP) on each dataset. Our results show that the parallel approach performs well and reduces the execution time shown in Figure 5.11. For example, the elapsed time of dataset 3 is reduced from 2.6 hours on a single processor to 1.37 hours on 2 processors, and less than 1 hour on 4 or more than 4 processors.

- P4-pPhylo(MP) and P10-pPhylo(MP) speedup over Rec-I-DCM3 at 24 hours

  In the Figure 5.12, we compute the time taken by the average P4-pPhylo(MP)

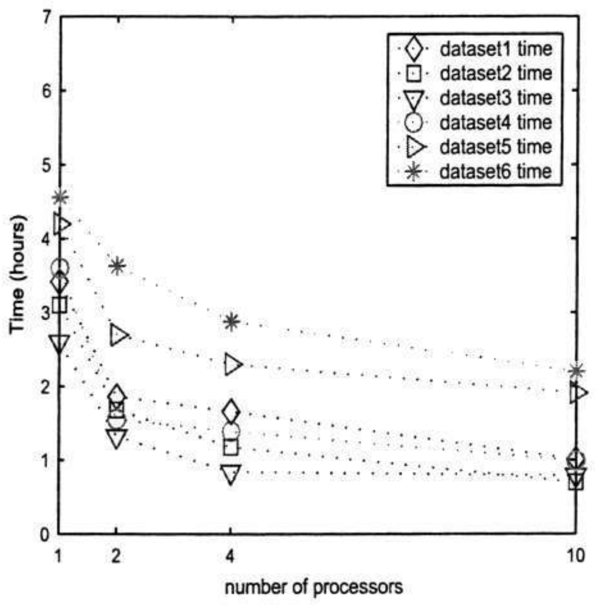


Figure 5.11: Time to complete one iteration of Rec-I-DCM3, P2-pPhylo(MP), P4-pPhylo(MP), and P10-pPhylo(MP) on datasets 1 to 6.

and P10-pPhylo(MP) trials to reach the best score of the average Rec-I-DCM3 trial and divide it by the time taken by average Rec-I-DCM3 trial to reach its best score. Thus we look at the speedup obtained by P4-pPhylo(MP) and P10-pPhylo(MP). On all the datasets P4-pPhylo(MP) and P10-pPhylo(MP) reach the best score of Rec-I-DCM3 at least in half the time. On dataset4 P10-pPhylo(MP) reaches the best Rec-I-DCM3 score three times faster, despite the bottleneck of the serial hill-climbing phase. We do not see much of an improvement of P10-pPhylo(MP) over P4-pPhylo(MP) on dataset 6 mainly because the hill-climbing phase is taking too long.

- Comparison of iterations in 24 hours

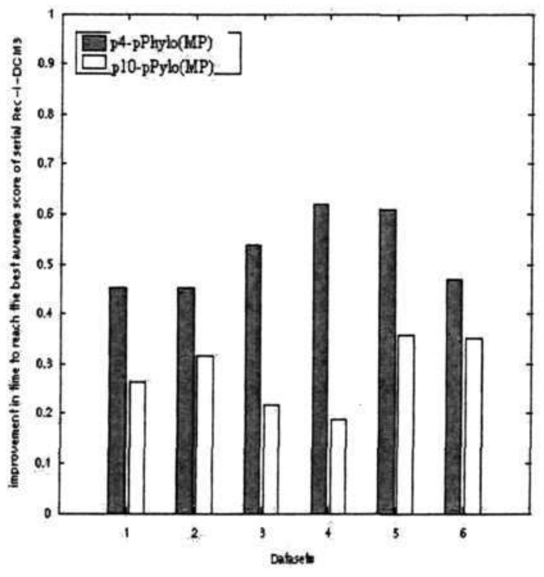


Figure 5.12: Ratio of time taken by the average P4-pPhylo(MP) and P10-pPhylo(MP) trials to reach the best Rec-I-DCM3 average score and the time taken by the average Rec-I-DCM3 trial to reach its best average score.

Table 5.3 compares the number of iterations of the serial and parallel Rec-I-DCM3 versions. P10-pPhylo(MP) is again able to do man more iterations (compared to serial) on the smaller datasets because the hill-climbing search is faster there.

5.4.4 Evaluation of performance limits

The fairly flat curve of the elapsed time at the high end of processor number shown in Figure 5.11 suggests that computational gain from further distribution of the subsets will be discounted by the overhead communication between the processes. Another factor limiting the scalability of this algorithm is the relatively few serial portions of

Table 5.3: Improvement in iterations within 24 hours

Dataset No.	Rec-I-DCM3 iterations	P4-pPhylo(MP) iterations	P10-pPhylo(MP) iterations
1	4	10	21
2	5	15	33
3	6	14	31
4	4	10	17
5	4	8	13
6	5	9	11

the program. One major portion is global rearrangement by hill-climbing heuristic search on the complete phylogenetic tree, which takes about half of time in one iteration. Clearly by reducing the time spent on the full dataset we can expect much better improvements. The Strict Consensus Merger and the actual decomposition, although both serial, take much less time, almost negligible, in comparison to the time of subtree computing and global rearrangement.

5.5 ML phylogenetic tree reconstruction

In previous section, our model has been shown to work with MP methods. We now describe how it working with ML methods, called pPhylo(ML), improves the performance of reconstructing ML trees. Note that, the ML criterion tries to maximize the probability of "evolving" the observed sequences. Whichever tree provides the maximum value for this likelihood function wins. ML is a harder problem than MP because it is not known how to compute the ML score of a given tree in polynomial time. RAxML is—to the best of our knowledge—among the currently fastest, most accurate, as well as most memory-efficient ML heuristics on real biological datasets.

However, the computation of comparatively large trees is limited by memory consumption. Thus, our parallel divide-and-conquer model is required to intelligently select overlapping sub-alignments for computing smaller subtrees in parallel.

Provided the high memory efficiency of RAxML (which the program inherited from fastDNAml) compared to other programs and the good performance on large real-world data it appears to be best-suited for use with our model. Therefore, we use a heuristic ML method which followed the idea of RAxML (Stamatakis, Ludwig & Meier 2005) as the *base method* and a new parallel version of RAxML developed in this study as global search method.

Due to the complexity of the problem and the ML criterion it is not possible to avoid global optimizations of the tree all-together. All divide and conquer approaches for ML to date execute global optimizations at some point (see Section 5.2).

We now look at RAxML and issues relating to parallelizing it.

5.5.1 Parallelizing RAxML

In this section we briefly outline the algorithm of RAxML, which is required to understand the structure and intrinsic difficulties which arise with the parallelization of the global search method.

RAxML initially computes a starting tree, which contains all sequences of the alignment using a fast greedy MP search. The MP search is performed by an appropriately modified version of Joe Felsenstein's **dnapars** program (Felsenstein 2004). One important property of the **dnapars** program is that it yields distinct starting trees depending on the input order permutation of the sequences. By randomizing the sequence input order, the program can start the optimization from different points of search space each time it is executed. Therefore, by executing several RAxML runs it is more likely to find good trees and avoid local maxima since each run will yield a distinct final tree. Thus, the confidence into the final result obtained by RAxML is higher than for strictly *deterministic* programs. However, in pPhyml(ML), the global RAxML search is initiated with a fixed starting tree (guide tree).

After the computation of the starting tree or reading in the guide tree, the likelihood of the candidate topology is improved by subsequent application of topological alterations. To evaluate and select candidate alternative topologies RAxML uses a mechanism called *lazy subtree rearrangements* (Stamatakis, Ludwig & Meier 2005). This mechanism initially performs a rapid pre-scoring of a comparatively large number of alternative topologies. After the pre-scoring step a few (20) of the best pre-scored topologies are analyzed more thoroughly. To the best of our knowledge, RAxML is currently one of the fastest *and* most accurate programs on real alignment data due to this ability to quickly pre-score a large number of alternative tree topologies and the low memory consumption.

As outlined in the example in Figure 5.13 the optimization process can be classified into two main computational phases:

1. **Difficult parallelization:** The *Initial Optimization Phase (IOP)* where the likelihood increases steeply and many improved pre-scored topologies are encountered during a single iteration of RAxML.

2. **Straight-forward Parallelization:** The *Final Optimization Phase (FOP)* where the likelihood improves asymptotically and practically all improved topologies are obtained by thoroughly optimizing the 20 best pre-scored trees.

The difficulties, which arise with the parallelization of RAxML, are mainly due

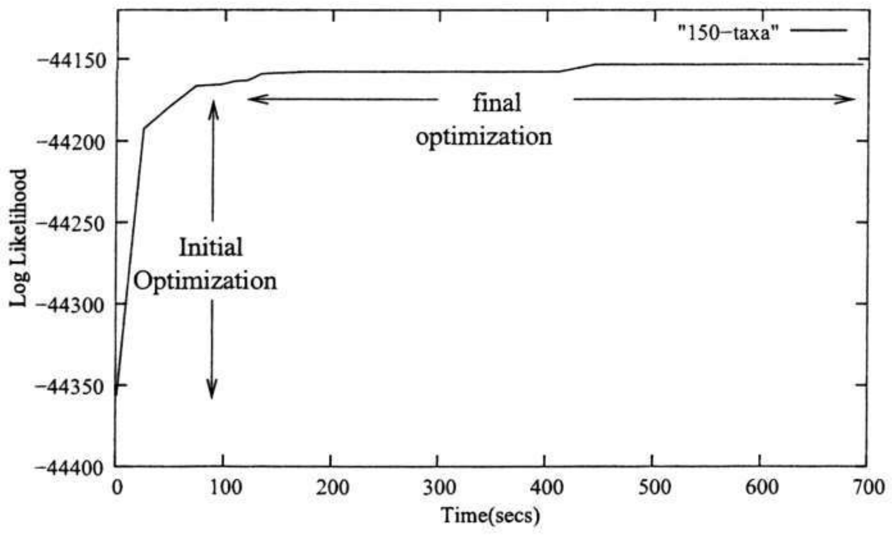


Figure 5.13: Initial and final optimization phase of RAxML for an alignment with 150 sequences

to hard-to-resolve dependencies caused by the detection of many improved trees (Stamatakis, Ludwig & Meier 2005) during the IOP. Moreover, the *fast* version of the hill-climbing algorithm of RAxML that is used for global optimization with pPhyloML further intensifies this problem, since it terminates after the IOP. During one iteration of RAxML all n subtrees of the candidate topology are subsequently removed and re-inserted into neighboring branches (subtree rearrangements). The hard-to-resolve dependency occurs when the lazy rearrangement of a subtree i yields a topology with a better likelihood than the candidate topology even though it is only pre-scored. In this case the improved topology is kept and rearrangement of subtree $i + 1$ is performed on the new topology. Especially, during the IOP when the likelihood increases steeply, improved pre-scored topologies are frequently encountered in the course of one iteration, i.e. n lazy subtree rearrangements. Since the lazy rearrangement of *one* single subtree is fast, a coarse-grained MPI-parallelization of one RAxML-iteration can only be based on assigning the rearrangement of distinct subtrees within the current candidate tree simultaneously to the workers processes. This represents a non-deterministic solution to the potential dependencies between rearrangements of subtrees i and $i + 1$. This means that when two workers w_0 and w_1 simultaneously rearrange subtrees i and $i + 1$ within the currently best candidate tree and the re-arrangement of subtree i yields a better tree, worker w_1 will miss this improvement since it is still working on the old candidate tree.

It is this frequently occurring dependency during the IOP between steps $i \rightarrow i+1$ ($i = 1...n$, n = number of subtrees) that leads to parallel performance penalties. Moreover, this causes a non-deterministic behavior since the parallel program traverses another path in search space each time (even for identical starting trees and number of processors) and might yield better or worse final tree topologies compared to the serial program. The scalability for smaller number of processors is better since every worker misses less improved trees.

The aforementioned problems have a significant impact on the IOP only, since the FOP can be parallelized more efficiently. Furthermore, due to the significantly larger proportion of computational time required by the FOP the parallel performance of the slow hill-climbing version of RAxML is substantially better. Please refer to (Stamatakis et al. 2004) for performance results of slow parallel hill-climbing and a more detailed description of the parallelization.

The necessity to parallelize and improve performance of RAxML fast hill-climbing has only been recognized within the context of using RAxML in conjunction with pPhylo(ML) and is therefore an issue of future work.

5.5.2 Working with ML–pPhylo(ML)

Following the framework of pPhylo, the overall program flow of pPhylo(ML) is outlined in Figure 5.14. Function **pPhylo_subtree** uses a heuristic ML method that followed the idea of RAxML to build ML subset trees.

Re-implementation of pPhylo_subtree for ML

- *Input:*
  - Set of y sequences in A_i
  - Starting subset tree $T|A_i$
- *Output:* A ML subset tree
- *Algorithm:*
  1. Set $T_{best}|A_i = T|A_i$
  2. For n subtrees in $T_{best}|A_i$ do:
     (a) Remove a subtree from the best tree, $T_{best}|A_i$
     (b) Reinsert the subtree into neighboring branches up to a specified distance of nodes
     (c) Optimize the three local branches adjacent to the insertion point and store the best 20 trees
     (d) Perform global branch length optimizations on those 20 best topologies only.
     (e) Update the best tree, $T_{best}|A_i$ for next loop

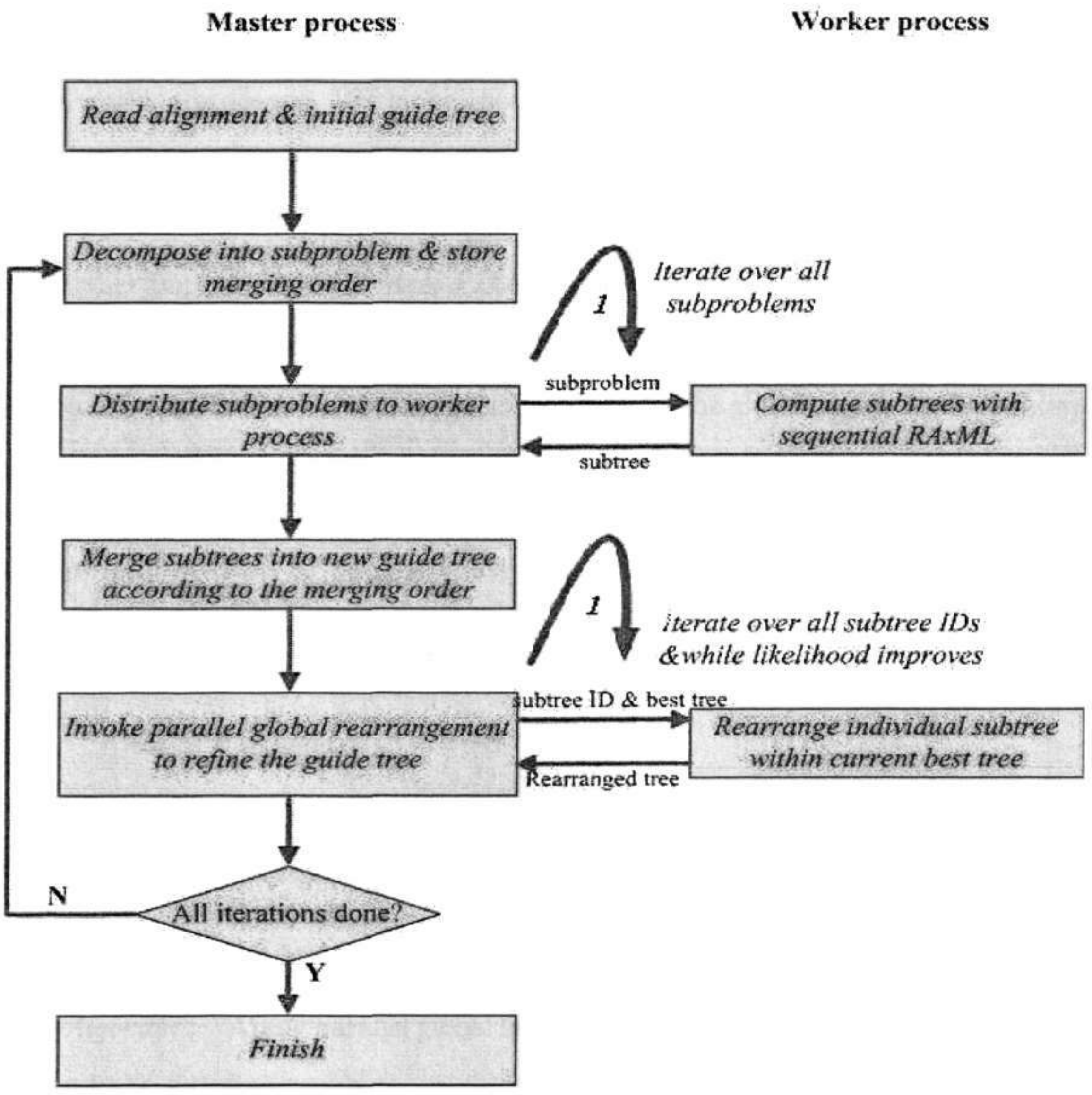

Master process
Worker process
Read alignment & initial guide tree
Decompose into subproblem & store merging order
Iterate over all subproblems
1
subproblem
Distribute subproblems to worker process
Compute subtrees with sequential RAxML
subtree
Merge subtrees into new guide tree according to the merging order
1
Iterate over all subtree IDs &while likelihood improves
subtree ID & best tree
Invoke parallel global rearrangement to refine the guide tree
Rearrange individual subtree within current best tree
Rearranged tree
N
All iterations done?
Y
Finish

Figure 5.14: pPhylo(ML) program flow

At final phase of global rearrangement, the supertree is refined (further optimized) by a parallel global search method presented in **pPhylo_grearrange for ML**. In this case, the master process distributes the IDs of the subtrees (simple integers) which have to be rearranged along with the currently best tree (only if it has been improved) to the working processes. Working processes rearrange a specified subtree within the currently best tree and return the tree topology (only if it has a better likelihood) along with a new work request. This process continues until no subtree rearrangement can further improve upon the tree. The global search method further improves the accuracy of the supertree and can also find optimal global configurations that were not found by operations on smaller—local—subsets. The global search method shown in parallel **pPhylo_grearrange** leads to significant improvements over RAxML.

Re-implementation of parallel pPhylo_grearrange for ML

- *Input:*
  - set of k sequences, $S = S_1, S_2..., S_k$
  - Supertree tree, T'
  - number of available processes, *P+1*, p is number of working processes.
- *Output:* ML phylogenetic Tree, T_{best}
- *Algorithm:* Apply parallel global search method starting from T' until we reach a local optimum.
  - On master process(if $myrank = 0$)
    1. Set an array of *flag[1...p]* for p working processes. Initially, $flag[i] = True, 1 \leq i \leq p$.

```
2. Distribute work by **Send**ing a *subtree ID* (one of the subtree to be
   arranged) along with the suptertree $T'$ to $p$ working processes
3. $T_{best}$=$T'$
4. For $n$-$(p)$ subtrees in $T_{best}$ do:
   (a) If **Recv** $READY$ from process $x$
       i. **Send** back the current score, $T_{best}$'s ML score, to process $x$
       ii. If **Recv** $W_{tree}$ from process $x$
              $T_{best}$=$W_{tree}$
              $for(1 \leq i \leq p)$
                 $flag[i] = True$,
       iii. If $flag[x] = True$
              Distribute new work by **Send**ing a **subtree ID** along with
            the currently best topology tree, $T_{best}$, to process $x$.
              Set $flag[x] = False$
            else
              Distribute work by **Send**ing a *subtree ID* (of the subtree to
            be arranged) to process $x$.
5. Send **END** message to working processes.
```

– On working processes (if $myrank > 0$)

```
1. Enter an infinite loop until it **Recv**s an $END$ message
   (a) **Recv** a *subtree ID* or *subtree ID* along with the current best tree,
       $T_{best}$, from master process.
```

(b) Rearrange the subtree within current best tree T_{best} to get a new tree W_{tree} and **Send**s back a *READY* message.

(c) **Recv** the current T_{best}'s ML score from master process.

(d) if W_{tree}'s ML score $>$ T_{best}'s ML score

Send back W_{tree} to master process.

2. **Recv** *END* message and exit.

In function parallel **pPhylo_grearrange**, in order to reduce communication between processes, we build an array: $flag[p]$, for p number of working processes. The array of *flag* controls whether the current best tree T_{best} should be distributed to each working process along with the following work requests. For example, $flag[1] = true$ means process 1 need to be updated with the currently best topology tree. If a working process already has the best tree T_{best}, the master process only need to send a *subtree ID* to it. Each working process will perform subsequent subtree rearrangements of the current step on the improved topology. Furthermore, after a working process obtains an improved tree, it will compare the score of the improved tree with the current best score on the master process. Only if the improved tree is better than the current best tree on master process, the working process sends it to master process in case other working processes have updated the best tree already. These strategies will cut half of communication time when data sets are large.

Because the sizes of individual subproblems vary significantly and the inference time per subproblem can not be predicted because it is not known a priori how many iterations will be performed by **pPhylo_subtree** before convergence. Thus, depending on the subset decomposition can lead to significant load imbalance (see Section 5.5.5 for a more detailed analysis). On the other hand, the computation of single small subproblems can not be carried out by a parallel optimized method as parallel **pPhylo_grearrange** for a supertree either due to their relatively small size and thus limited scalability. In addition—according to some experiments with a proprietary divide and conquer implementation in RAxML—a distinct method of subtree-decomposition, which yields subproblems of equal size for better load-balance does not appear promising. This is due to the fact that **pPhylo_devide** constructs subproblems intelligently with regard to closely-related taxa based on the information of the guide tree.

In the following sections of experiment design and experimental results, we initially describe the test datasets used. Thereafter, we report on serial performance improvements of Rec-I-DCM3(RAxML) over stand-alone RAxML in section and finally analyze parallel program efficiency of pPhylo(ML).

5.5.3 Experiment design

Methodology

To the best our knowledge, RAxML, is the best known technique heuristic for solving ML. Firstly, we design an experiment to show that Rec-I-DCM3(RAxML) method would improves over stand-alone RAxML on all datasets in order to demonstrate the benefits which arise from using the dividing method. Next, we assess the performance gains of our method improves Rec-I-DCM3(RAxML) dramatically.

Test datasets

In our experiments, we used a large variety of biological datasets ranging in size and type (DNA or RNA). All but one of the very large datasets (above 2K) are RNA because we were unable to find DNA alignments of that size. This is mainly due to the fact that RNA data is much more abundant since it is slower evolving and relatively easy to align as well. Some datasets have been downloaded from public databases and sites containing more than 20% gaps have been removed. Other alignments have been obtained from researchers who have manually inspected and verified the alignments.

- Dataset1: 500 *rbcL* DNA sequences (1398 sites) also known as the popular Zilla dataset (Rice, Donoghue & Olmstead 1997).
- Dataset2: 2,560 *rbcL* DNA sequences (1,232 sites) (Kallerjo, Farris, Chase, Bremer & Fay 1998).
- Dataset3: 4,114 aligned 16s ribosomal Actinobacteria RNA sequences (1,263 sites) (Maidak 2000).
- Dataset4: 6,281 aligned small subunit ribosomal Eukaryotes RNA sequences (1,661 sites) (Wuyts et al. 2002).
- Dataset5: 6,458 aligned 16s ribosomal Firmicutes (bacteria) RNA sequences (1,352 sites) (Maidak 2000).
- Dataset6: 7,769 aligned ribosomal RNA sequences (851 sites) from three phylogenetic domains, plus organelles (mitochondria and chloroplast), obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.

5.5.4 Experimental results

Serial performance

In the first set of experiments we examine the serial performance of stand-alone RAxML over Rec-I-DCM3(RAxML) in order to demonstrate the benefits which arise from using the latter dividing method.

The respective maximum subset sizes of Rec-I-DCM3 are adapted to the size of each dataset and are indicated in Table 5.4.

Table 5.4: Maximum Rec-I-DCM3 subproblem sizes

Dataset	Maximum subproblem size
Dataset1	100
Dataset2	125
Dataset3–6	500

In our experiments both methods start optimizations on the *same* starting tree. Due to the relatively long execution time on large alignments we only executed one Rec-I-DCM3(RAxML) iteration per dataset. The run time of one Rec-I-DCM3 iteration was then used as inference time limit for RAxML. Table 5.5 provides the log likelihood values for RAxML and Rec-I-DCM3 after the same amount of execution time. Note that, the apparently small differences in final likelihood values *are significant* because those are logarithmic values and due to the requirements for high score accuracy in phylogenetics (T.L.Williams et al. 2004).

The experiments clearly show that Rec-I-DCM3(RAxML) improves over stand-alone RAxML on *all* datasets, i.e. yields better likelihood values than RAxML in the same amount of time. This results serve as an argument for the choice of the divide-and-conquer model instead of stand-alone RAxML.

Table 5.5: Rec-I-DCM3(RAxML) versus RAxML log likelihood values after the same amount of inference time

Dataset	Rec-I-DCM3(RAxML) log likelihood	RAxML log likelihood
Dataset1	-99967	-99982
Dataset2	-355071	-355342
Dataset3	-383578	-383988
Dataset4	-1270920	-1271756
Dataset5	-901904	-902458
Dataset6	-541255	-541438

Parallel performance

In our second set of experiments we assess the performance gains of pPhylo(ML) over the Rec-I-DCM3(RAxML). For each dataset we executed three individual runs with Rec-I-DCM3(RAxML) and pPhylo(ML) on 4, 8, and 16 processors respectively. Once again for each individual run we used the *same* starting tree for the serial and parallel inference. Furthermore, the same subset sizes as indicated in Table 5.4 were used. In order to determine the speedup we measured the execution time of one serial and pPhylo(ML) iteration for each dataset/number of processors combination. The average serial and parallel execution time per dataset and number of processors over three individual runs are outlined in Figure 5.15.

Due to the hard-to-resolve dependencies in parallel **pPhylo_grearrange** and the size imbalance of the subproblems the overall speedup and scalability of pPhylo(ML) are moderate. As already mentioned the requirement to devise a more efficient parallelization of the *IOP* of RAxML has only recently been recognized within the context of using RAxML in conjunction with pPhylo(ML). A separate analysis of the speedup values in Table 5.6 for the parallelization of the *base*, *global*, and *whole method* shows that the parallel performance losses originate mainly from parallel

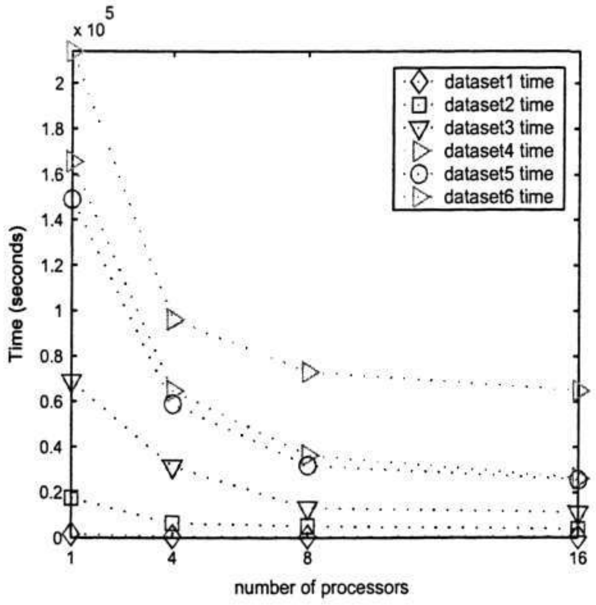


Figure 5.15: Time to complete one iteration of Rec-I-DCM3(RAxML) for datasets 1–6 and 1 up to 16 processors.

pPhylo_grearrange.

It is important to note that pPhylo(ML) executes a more thorough search, i.e. yields better trees than the serial global optimization with RAxML. Therefore, the comparison can not exclusively be based on speedup values alone but must also consider the final likelihood values attained by pPhylo(ML) which are significantly better. To demonstrate this pPhylo(ML) is granted the overall execution time of one serial Rec-I-DCM3(RAxML) iteration, i.e. the same response time. The final log likelihood values of Rec-I-DCM3(RAxML) and pPhylo(ML) (on 16 processors) after the same amount of global execution time are listed in Table 5.7. As already mentioned in the apparently small differences in final likelihood values *are significant.* Furthermore,

Table 5.6: Average base method, global method, and overall speedup values for one iteration of serial program per dataset and number of processors over three runs

Number of Processors	base method	global method	overall speedup
Dataset1			
4	4	2.4	2.6
8	4.7	2.8	3.6
16	4.85	2.78	3.5
Dataset2			
4	3	2.68	2.7
8	5.3	3.2	3.45
16	7	4.2	4.6
Dataset3			
4	1.95	2.6	2.2
8	5.5	5	5.3
16	6.7	5.7	6.2
Dataset4			
4	2.9	2.3	2.6
8	4.2	4.9	4.6
16	8.3	5.3	6.3
Dataset5			
4	2.3	2.7	2.5
8	4.8	4.4	4.7
16	7.6	5.1	5.8
Dataset6			
4	3.2	1.95	2.2
8	4.8	2.5	3
16	5.4	2.8	3.3

the computational effort to attain those improvements is not negligible due to the asymptotic increase of the log likelihood in the *FOP* (see Figure 5.13 in Section 5.5.1).

Table 5.7: Average Log likelihood scores of Rec-I-DCM3(RAxML) and pPhylo(ML) (on 16 processors) per dataset after the same amount of global execution time over three individual runs

Dataset	Rec-I-DCM3(RAxML) log likelihood	pPhylo(ML) log likelihood
Dataset1	-99967	-99945
Dataset2	-355088	-354944
Dataset3	-383524	-383108
Dataset4	-1270785	-1270379
Dataset5	-902077	-900875
Dataset6	-541019	-540334

In heuristics for hard optimization problems, 90%–95% accuracy is often considered excellent. Heuristics used in phylogenetic reconstruction must be much more accurate if it has at least 99:99% accuracy in order to produce topologically accurate trees. Thus, the log likelihood score is better, the topology of tree is more accurate.

5.5.5 Evaluation of performance limits

The general parallel performance limits of RAxML have already been outlined in Section 5.5.1. At this point we discuss the parallel performance limits of the base method by the examples of the Dataset3 and Dataset6, which initially appear to yield fairly sub-optimal speedups—especially on 16 processors—in column *base method* of Table 5.6. We show that those values are actually near-optimal.

For the sake of this analysis we measured the number of subproblems as well as the inference time per subproblem for one Rec-I-DCM3 iteration on those datasets. As already mentioned, the main problem consists in a significant load imbalance in subproblem sizes. The computations for the Dataset3 comprises 19 subproblems, which are dominated by three computations that require more than 5.000 seconds (maximum 5.569 seconds). We determined the optimal schedule of those 19 subproblems on 15 processors (since 1 processor serves as master process) and found that the maximum inference time of 5.569 seconds is the limiting factor, i.e. the minimum execution time for those 19 jobs on 15 processors is 5.569 seconds. With this data at hand we can easily calculate the maximum attainable speedup by dividing the sum of all subproblem inference time through the minimum execution time, i.e. $37353secs/5569secs = 6.71$ which corresponds to our experimental results. Note that, there is no one-to-one correspondence since the values in Table 5.6 are average values over several iterations and three runs per dataset with different guide trees and thus different decompositions.

The analysis of Dataset6 shows a similar image: there is a total of 43 subproblems which are dominated by 1 long subtree computation of 12.164 seconds and three smaller ones ranging from 5.232 to 6.235 seconds. An optimal schedule for those 43 subproblems on 15 processors shows that the large subproblem which requires 12.164 is the lower bound on the parallel solution of subproblems. The optimal speedup for the specific decomposition on this dataset is therefore $63620secs/12164secs = 5.23$.

5.6 Summary

In this chapter, a parallel divide-and-conquer model (pPhylo) is designed to flexibly reconstruct the large MP and ML trees comprising up to 10,000 organisms. The proposed model is able to perform a more complete search of the tree space within limited time. It significantly reduces response time for large trees and improves final tree quality at the same time. Experimental results show that the trees computed by the proposed model are consistently better than the previously known fastest and most accurate programs for MP and ML respectively.

Chapter 6

Pattern-Constrained Sequence Matching

The quality of inference of phylogenetic trees heavily depends on the quality of an alignment which serves as input for phylogeny programs. Therefore, a "good" multiple alignment is the most important prerequisite for conducting phylogenetic analysis. Most of the existing progressive MSA methods use a substitution matrix independent of the position, the essential problem of which is that they incorporate no external knowledge of the sequences being aligned. This chapter presents a novel algorithm to improve the accuracy of multiple sequence alignment. The significance of the proposed algorithm lies on its capability of aligning the sequences sharing the same patterns. Moreover, it is proven that the similarity score of the proposed algorithm has an approximation ratio $\frac{k}{2(k-1)}$ to the similarity score of the optimal alignment, where k is the number of sequences.

6.1 Problems with existing methods

In progressive MSA methods, one uses dynamic programming to build MSA initially with the most related sequences and then progressively adding less related sequences or groups of sequences to the initial alignment. These methods use a substitution matrix independent of the position, which has however a serious problem. They do not incorporate knowledge of the sequences being aligned and therefore cannot assure the alignment of similar structures and common patterns shared by the sequences. Comet and Henry (Comet & Henry 2002) conclude that the dynamic programming algorithm does not always align patterns in two sequences. In order to clarify it, they plot the distribution of the patterns from the databank PROSITE Rel. 14 according to the number of non-aligned patterns divided by the total number of pairwise comparisons. The statistical results prove that the dynamic programming algorithm does not align patterns when occurrences are not similar or when they are very short and do not belong to the most similar regions.

Methods have been proposed to solve this problem. In fact, two algorithms, PHI-BLAST (Pattern-Hit Initiated BLAST) (Altschul, Madden, schaffer, Zhang, Zhang, Miller & Lipman 1997) and SWP (Smith-Waterman algorithm with Patterns) (Comet & Henry 2002), attempting to combine motif with pairwise alignments have been quite successful, although they are not for MSA problems. PHI-BLAST is a searching program that combines matching of regular expressions with local alignments surrounding the match. Given a protein sequence S and a regular expression pattern P occurring in S, PHI-BLAST searches a database for sequences that include the pattern and have significant similarity to the query sequence. SWP is an algorithm designed specifically to incorporate patterns into the Smith-Waterman algorithm. It compares subsequences letter by letter as in the Smith-Waterman but attributes a supplementary bonus/reward when patterns are matched.

We have been investigating into solutions to the problem in MSA and making use of the knowledge of the sequences being aligned, including patterns in the PROSITE databank (Hofmann, Bucher, Falquet & Bairoch 1999), BLOCKS+ (Henikoff, Henikoff & Pietrokovski 1999) (Henikoff, Greene, Pietrokovski & Henikoff 2000), eBlocks database (Su, Lu & Brutlab 2002), as well as motif and structural information. As a result, we have developed a pattern-constrained algorithm for MSA (PCMSA), in particular, multiple polypeptide sequence alignment.

6.2 Patterns as constraints

Patterns are defined as multiply aligned segments corresponding to highly conserved regions of protein sequences in the proposed algorithm. These patterns can be retrieved from previously constructed databases.

In the PROSITE databank, patterns are represented by regular expressions. Each element in a pattern is separated from its neighbor by a '-'. Repetition of an element of a pattern can be indicated by following that element with a numerical value or, if it is a gap ('x'), by numerical range between parentheses. Thus, a pattern [AC]-x-V-x(4) is translated into [Ala or Cys]-any-Val-any-any-any-any. The ROSITE databank currently contains patterns and profiles specific for more than a thousand protein families or domains. Each of these signatures comes with documentation providing background information on the structures and functions of these proteins.

The BLOCKS+ database is constructed from the PROSITE databank, supplemented with additional families from the PRINTS (Attwood, Flower, Lewis, Mabey, Morgan, Scordis, Selley & Wright 1999), PfamA (Bateman, Birney, Durbin, Eddy, Finn & Sonnhammer 1999), ProDom (Corpet, Gouzy & Kahn 1998) and Domo (Gracy & Argos 1998) protein family databases. In the BLOCKS+ database, patterns are ungapped segments corresponding to the most highly conserved regions of proteins.

Our algorithm aims to find better alignments of multiple polypeptide sequences, using the patterns from the above databases as the constraints. It uses pattern constrained pairwise alignments with further assembling of these "partial" alignments into an approximate alignment of k sequences. It also has the worst-case guarantee on the quality of the alignment.

To start with, let S be a set of k sequences $S_1, ..., S_k$, the center sequence $S_c \in S$ be the sequence such that the sum of pairwise scores constrained by patterns to the other *k-1* sequences is minimized. The sum score of MSA with respect to a center sequence S_c is the sum of pairwise scores of with any other sequences defined as following:

$$\sum_{1 \le x \le k, x \ne c} \sum_{1 \le i \le n'} \delta(S_{c,i}, S_{x,i}) \tag{6.2.1}$$

where n' is the length of the alignment; $S_{c,i}$ is the i-th residue in the center sequence S_c and $S_{x,i}$ is the i-th residue in x-th sequence.

Three main processes are developed as following:

1. Identifying all patterns shared by sequences.

It is necessary to find all occurrences of patterns shared by sequences. In this case, a list of patterns defined by regular expressions in the PROSITE databank or the BLOCKS+ database is generated for that purpose. These regular expressions are referenced to locate and recognize all patterns in sequences. The function **step** in UNIX is used to find patterns in each sequence, which returns non-zero if some substring of a sequence matches a pattern described in the list, and zero if there is no match. Then the common patterns shared by sequences are identified. There are also other cases in which users define patterns by themselves according to the conserved regions and structural information shared by sequences.

2. Defining a center sequence $S_c \in S$.

We define a center sequence $S_c \in S$ such that $\sum_{z \neq c} Sim(S_z, S_c)$ is maximized, where $Sim(S_z, S_c)$ is the similarity score of pattern-constrained alignment between sequences S_z and S_c. Care must be taken to handle the variety of pattern matches, such as inversion of the same pattern or a pattern appearing several times in the same sequence. Let S_a and S_b be two sequences of length m and n, and p be a shared pattern between the two sequences. The similarity scores of pattern-constrained alignments can be computed by using a customized dynamic programming algorithm as in Equation 6.2.2.

$$Sim_{i,j} = max \begin{cases} Sim_{i-1,j-1} + S(S_a[i], S_b[j]), 1 \leq z \leq k, z \neq c \\ Sim_{i-x,j} - w_x, x \geq 1 \\ Sim_{i,j-y} - w_y, y \geq 1 \\ Sim_{i-len,j-len} + |score(p)| \times reward + score(p) \end{cases} \tag{6.2.2}$$

Where $Sim_{i,j}$ is the similarity score obtained when aligning the beginning sequences $S_a[1...i]$ and $S_b[1...j]$ (the $Sim_{i,j}$ score is larger, the distance $D(S_a[1...i], S_b[1...j])$ is smaller); p is a shared pattern ending at position (i, j); len is the length of p; $S(S_a[i]S_b[j])$ is the score for aligning the characters at position i and j in sequence S_a and S_b; w_x and w_y are the penalties for gaps of length x and y in sequence S_a

and S_b; $score(p)$ is the score of aligning two occurrences of the pattern p. In the position (i, j), its score is determined by four paths, three of them are the same with the original algorithm: match, insertion and deletion. Another path is the match of the shared pattern beginning at the $(i - len + 1, j - len + 1)$. Figure 6.1 shows the dependency. In order to favor the alignment of patterns, it gives an additive *reward* to the score when a pattern matching. It is possible for two occurrences of a pattern to have a negative alignment score so that we use $|score(p)| \times reward + score(p)$ to ensure a positive reward of the pattern match. And it is obvious that this score also would correlate with the strength of match among patterns.

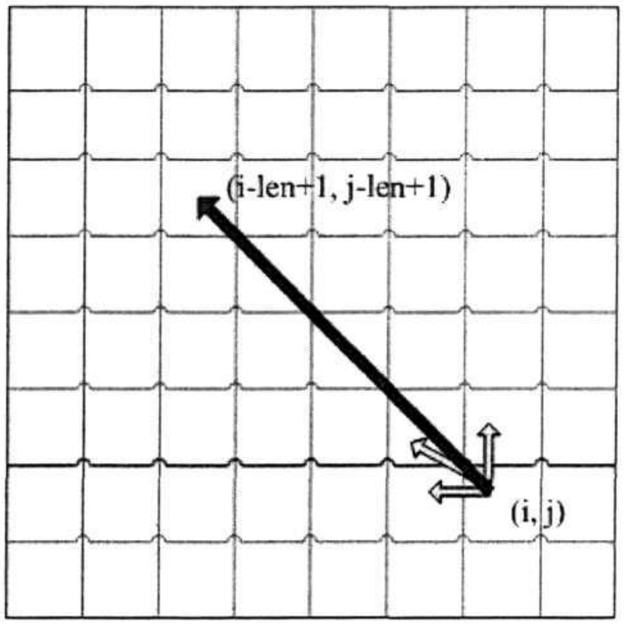


Figure 6.1: Block dependency

When doing trace back, it also needs to consider a pattern path. If the current maximum score is determined by a pattern score with *reward*, the trace back path should be along a diagonal to trace back *len* steps.

In some situation, the occurrences of patterns may have different lengths if they are obtained from regular expression databases, such as the the PROSITE databank. This is because the same PROSITE regular expression in two sequences can contain variable length of subpatterns. In order to match such patterns, we first set the area with variable length, such as $x(5,8)$, as the area of insertions/deletions in the patterns. Let position (i,j) be the current indices of aligning two occurrences (x and y) of a pattern, $(x_b...x_e)$ be the area of insertions/deletions in the occurrence x, $(y_b...y_e)$ be the area of insertions/deletions in the occurrence y. If position (i,j) does not belong to the insertion/deletion area, the letter x_i faces the letter y_j. If position (i,j) belongs to the insertions/deletions area, the Needleman-Wunsch algorithm is used align two substrings of $(x_b...x_e)$ and $(y_b...y_e)$.

For those patterns from the BLOCKS+ database, instead of a regular expression database, the occurrences of patterns are of the same length, which can reduce some hypothetical affects when doing variable length pattern matching.

3. Constructing the alignment M with the rest (k-1) sequences in S

Initially, let M $= S_c$. Sequence $S_z(1 \leq z \leq k, z \neq c)$ is added to M progressively by using Equation 6.2.2 such that the alignment of each newly added sequences, S_z with S_c is optimal. In order to form a multiple sequence alignment, spaces need to be inserted into each pre-aligned sequences.

6.3 Bound evaluation

The similarity score of the PCMSA algorithm is bounded by an approximation ratio of $\frac{k}{2(k-1)}$ to the similarity score of the optimal pattern-constrained alignment.

Given $S = S_1, ..., S_k$, $S_c \in S$ is the center sequence for which $\sum_{i \neq c} Sim(S_i, S_c)$ is maximum and a multiple alignment M.

Let M^* be the optimal pattern-constrained alignment of S, $Sim(M^*)$ is the similarity score of M^*.

Let M_c be the pattern-constrained alignment induced by the center sequence S_c, $Sim(M_c)$ is the similarity score of M_c.

$d^*(S_i, S_j)$ is the value of pattern-constrained alignment between S_i and S_j induced by M^*.

$Sim(S_i, S_j)$ is the similarity score of the pattern-constrained pairwise alignment it induces on S_i, S_j.

$d(S_i, S_j)$ is the distance between S_i, S_j, $Sim(S_i, S_j) = \frac{1}{d(S_i,S_j)}$.

$D(S_i, S_j)$ is the optimal distance between sequences S_i, S_j.

Proof:

Using the triangular inequality, we have $d(S_i, S_j) \leq d(S_i, S_c) + d(S_c, S_j)$. For each $S_i, i \neq c$, the alignment M_c induces an optimal pairwise alignment between S_c and S_i. Thus, $d(S_i, S_c) = D(S_i, S_c)$, and $d(S_i, S_j) \leq D(S_i, S_c) + D(S_c, S_j)$.

We have:

Lemma For $1 \leq i, j \leq k, i \neq j$ it holds that $d(S_i, S_j) \leq D(S_i, S_c) + D(S_c, S_j)$.

Let's show that the approximation ratio:

$\frac{Sim(M_c)}{Sim(M^*)} \geq \frac{k}{2(k-1)} > \frac{1}{2}$

Proof:

$$2Sim(M_c) = \sum_{i \neq j} Sim(S_i, S_j) = \sum_{i \neq j} \frac{1}{d(S_i,S_j)}$$

According to the **lemma**: $d(S_i, S_j) \leq D(S_i, S_c) + D(S_c, S_j)$, we have

$$\sum_{i \neq j} \frac{1}{d(S_i,S_j)} \geq \frac{1}{\sum_{i \neq j}(D(S_i,S_c)+D(S_c,S_j))} = \frac{1}{2(k-1)\sum_{i \neq c} D(S_c,S_i)}$$

Let X be $\sum_{i \neq c} D(S_c, S_i)$ and we get:

$$2Sim(M_c) \geq \frac{1}{2(k-1)X}$$

On the other hand,

$$2Sim_(M^*) = \sum_{i \neq j} Sim^*(S_i, S_j)$$

$$= \frac{1}{\sum_{i \neq j} d^*(s_i, s_j)} \leq \frac{1}{\sum_{i \neq j} D(S_i, S_j)} = \frac{1}{\sum_i \sum_{j \neq i} D(S_i, S_j)}$$

Based on the definition of S_c

$$\sum_i \sum_{j \neq i} D(S_i, S_j) \geq \sum_i \sum_{j \neq c} D(S_c, S_j) = \sum_i X = kX$$

Thus,

$$2Sim(M^*) \leq \frac{1}{kX}$$

And finally,

$$\frac{Sim(M_c)}{Sim(M^*)} \geq \frac{\frac{1}{2(k-1)X}}{\frac{1}{kX}} = \frac{kX}{2(k-1)X} = \frac{k}{2(k-1)} > \frac{1}{2}$$

6.4 Experimental results

To evaluate the performance of the PCMSA algorithm, it is compared with the ClustalW program, the most widely used method for multiple sequence alignment, on two datasets.

Test datasets

- Dataset1: six aminoacyl-tRNA synthetase sequences, SYR_PHOLL, SYR_CAEEL, COA1_POVBK, SYQ_HUMAN, CBIO2_STRAW and SP1_RARFA. The shared pattern in aminoacyl-transfer RNA synthetases is PS00178: aminoacyl-transfer RNA synthetases class-I signature.

  PS00178: P - x(0,2) - [GSTAN] - [DENQGAPK] - x - [LIVMFP] - [HT] - [LIVMYAC] - G - [HNTG] - [LIVMFYSTAGPC]

- Dataset2: eight sequences, CYTC_HUMAN, CYT_ONCMY, CYT1_MAIZE, CYTX_ONCVO, KNG1_BOVIN, ADPP_BACSU, MARAY_ENTFA and KGUA_BACFI The eight sequences share a PROSITE pattern PS00278: staphyloccocal enterotoxin/Streptococcal pyrogenic exotoxin signature 2.

  PS00278: [GSTEQKRV] - Q - [LIVT] - [VAF] - [SAGQ] - G - DG - [LIVMNK] - TK - x - [LIVMFY] - x - [LIVMFYA] - [DENQKRHSIV] ,

For the Dataset 1, the six aminoacyl-tRNA synthetase sequences are aligned in Figure 6.2. Aminoacyl-tRNA synthetases (P.Schimmel 1987) are a group of enzymes which activate amino acids and transfer them to specific tRNA molecules as the first step in protein biosynthesis. In prokaryotic organisms there are at least twenty different types of aminoacyl-tRNA synthetases, one for each different amino acid. In eukaryotes there are generally two aminoacyl-tRNA synthetases for each different amino acid: one cytosolic form and a mitochondrial form. While all these enzymes have a common function, they are widely diverse in terms of subunit size and of quaternary structure. These aminoacyl-tRNA synthetases are referred to as class-I synthetases (M.Delarue & D.Moras 1993) (P.Schimmel 1991) (G.M.Nagel & R.F.Doolittle 1991) and seem to share the same tertiary structure based on a Rossmann fold. It was found that the aminoacyl-tRNA synthetases share a region of similarity in the consensus tetrapeptide PS00178: P - x(0,2) - [GSTAN] - [DENQGAPK] - x - [LIVMFP] - [HT] - [LIVMYAC] - G - [HNTG] - [LIVMFYSTAGPC], which is well conserved. This pattern is defined as aminoacyl-transfer RNA synthetases class-I signature. Figure 6.2 indicates that the ClutalW algorithm is unable to align the pattern PS00178, but the PCMSA makes it possible to align the region of similarity in their N-terminal section.

```
sp|Q7N565|SYR_PHOLL      TPAK-PQTIVV-DYSAPNVAKQMHVGHLRSTIIGDAAVRTLEFLGHKVIR
sp|Q19825|SYR_CAEEL      PKLT-RKRVLV-DFSSPNIAKEMHVGHLRSTIIGDSICRLFEAVGFDVLR
sp|P03088|COA1_POVBK     S----AEN----DFSSDSPERKMLPCYSTARIP-------LPNLNEDLTC
sp|P47897|SYQ_HUMAN      HKPG-ENYKTP-GYVVTPHTMNLLKQHLEITGG--QVRTRFPPEPNGILH
sp|Q82B58|CBIO2_STRAW    LSGGQQQRVAI-GSVLTPHPKVLVLDEPTSALD--------PAAAEEVLA
sp|Q05308|SP1_RARFA      SKNAPSDIKNVNSWWVDPATNKVVIEARSKKAAK------AAATAAGLTA
                               .     .         :                        :

sp|Q7N565|SYR_PHOLL      ANHVGDWGTQF---------GMLIAYLEKVQNENASDMA-LSDLEAFYRE
sp|Q19825|SYR_CAEEL      VNHIGDWGTQF---------GMLIAHLYDRFPDFLKKLPDISDLQAFYKE
sp|P03088|COA1_POVBK     GN--------------------LLMWEAVTVQTEVIG---ITSMLNLHAG
sp|P47897|SYQ_HUMAN      IGHAKAINFNFGYAKANNGICFLRFDDTNPEKEEAKFFTAICDMVAWLGY
sp|Q82B58|CBIO2_STRAW    VLQRLVHDLG-----------TTVLMAEHRLERVVQYADRVALLPAPGAP
sp|Q05308|SP1_RARFA      GTYEIT----------------VSDDVIVPVRDYWGGDALSGCTLAFPVY

sp|Q7N565|SYR_PHOLL      AKKHYDEDEEFAIRARGYVVKLQGGDEYCRTMWRKLVDITMAQNQQTYDR
sp|Q19825|SYR_CAEEL      SKKRFDEDEQFKKRAYEYVVKLQSHDGDIVKAWNTICDVSKKYNQIVYNY
sp|P03088|COA1_POVBK     SQKVHEHGGGKPIQGSNFHFFAVGGE-----------------P------
sp|P47897|SYQ_HUMAN      TPYKVTYASDYFDQLYAWAVELIRRG--LAYVCHQRGEELKGHNTLPSPW
sp|Q82B58|CBIO2_STRAW    LTLGTPPEVMAASPVYPPVVDLGRLAG---------------WSPLPLTV
sp|Q05308|SP1_RARFA      GGFLTAGHCAVEGKGHILKTEMTGGQIG--------------------TV
```

(a)

```
sp|Q7N565|SYR_PHOLL    ... V---T-------P--A----K-P-Q-T---I--V---V-D-Y------S----A--PNVAKQMHVGHLRS-TI-IG-D...
sp|Q19825|SYR_CAEEL    ... IFLNTDYLRRQISLLASEGVKLP-KLTRKRV--L---V-D-F------S----S--PNIAKEMHVGHLRS-TI-IG-D...
sp|P03088|COA1_POVBK   ... L-----------P--C----Y-S---T---A--R---I----------P----L--PNLNEDLTCGNLL---M-W--E...
sp|P47897|SYQ_HUMAN    ... V---T-------PH-TMNLLK-Q-HLE---I--TGGQVRTRF------P-------PEPNGILHIGHAKAINFNFGYA...
sp|Q82B58|CBIO2_STRAW  ... T---T-------V--L----M-AEH-R---LERV---V-Q-Y------ADRV-ALLPAPGAPLTLGTP-P-EV-MA-A...
sp|Q05308|SP1_RARFA    ... L---T-------AG-TYEI-TVSDD-V---I--VP--VRD-YWGGDALSGCTLAF-PVYGGFLTAGHC-A--V-EG-K...
```

(b)

Figure 6.2: (a)ClustalW aligns 6 sequences in Dataset1 with one pattern; (b)PCMSA aligns 6 sequences in Dataset1 with one pattern

In Figure 6.3, eight sequences of inhibitors of cysteine proteases (A.J.Barrett 1987) (N.D.Rawlings & A.J.Barrett 1990)(V.Turk & W.Bode 1991) in Dataset 2 are aligned. Inhibitors of cysteine proteases are found in the tissues and body fluids of animals, in the larva of the worm Onchocerca volvulus (S.Lustigman, B.Brotman, T.Huima & A.M.Prince 1991), as well as in plants. They are grouped into three distinct but related families:

(1) Type 1 cystatins (or stefins), molecules of about 100 amino acid residues with neither disulfide bonds nor carbohydrate groups.

(2) Type 2 cystatins, molecules of about 115 amino acid residues which contain one or two disulfide loops near their C-terminus.

(3) Kininogens, which are multifunctional plasma glycoproteins.

Sequences known to belong to these families detected all have a consensus pattern, PS00278: [GSTEQKRV]-Q-[LIVT]-[VAF]-[SAGQ]-G-DG-[LIVMNK]-TK-x-[LIVMFY]-x-[LIVMFYA]-[DENQKRHSIV], in which five residues have been proposed to be important for the binding to the cysteine proteases. As shown in Figure 6.3, the PCMSA successfully works out staphyloccocal enterotoxin/Streptococcal pyrogenic exotoxin signature 2 while the ClustalW fails to align the conserved pattern.

```
sp|P01034|CYTC_HUMAN  ...  ---MAGPLRAPLLLLAILAVALAVSPAAGS------------------------------
sp|Q91195|CYT_ONCMY   ...  ---MEWKIVVP-----LFAVAFTVAN-AG-------------------------------
sp|P31726|CYT1_MAIZE  ...  KHRIVSLVAALLVLLALAAVSSTRSTQKES------------------------------
sp|P22085|CYTX_ONCVO  ...  IHLLLFSVVALVQLQGAKSARAKNPSKMES------------------------------
sp|P01044|KNG1_BOVIN  ...  VKRAQRQVVSGWNYEVNYSIAQTNCSKEEFSFLTPDCKSLSSGDTGECTDKAHVDVKLRI
sp|P54570|ADPP_BACSU  ...  EKTIAKEQIFSGKVIDLYVEDVELPNGKAS------------------------------
sp|O07107|MRAY_ENTFA  ...  MGLFFHQFTPSLLIILFILVLYGLLGYLDDFIKVFK------------------------
sp|Q64PY1|KGUA_BACFR  ...  AKVIIFSAPSGSGKSTIINYLLAQKLNLAFS-----------------------------

sp|P01034|CYTC_HUMAN       -----------SPG---KP-PRLVGGPMDASVEEE-----GVRRALDFA-VGEYNK---A
sp|Q91195|CYT_ONCMY        ----------------------LIGGPMDANMNDQ-----GTRDALQFA-VVEHNK---K
sp|P31726|CYT1_MAIZE       -----------VAD---NA-GMLAGGIKDVPANENDL---QLQELARFA-VNEHNQ---K
sp|P22085|CYTX_ONCVO       -----------KTGENQDR-PVLLGGWEDRDPKDE-----EILELLPSI-LMKVNE---Q
sp|P01044|KNG1_BOVIN       SSFSQKCDLYPVKDFVQPP-TRLCAGCPKPIPVDSP----DLEEPLSHS-IAKLNA---E
sp|P54570|ADPP_BACSU       -----------KREIVKHPGAVAVLAVTDEGKIIMVK---QFRKPLERT-IVEIPAGKLE
sp|O07107|MRAY_ENTFA       -----------KRNMGLNSRQKLIGQIFGGLVFYFVYRSEGFSDTLDLFGVAEVPLG--I
sp|Q64PY1|KGUA_BACFR       -----------ISATSRPPRGNEKHGVEYFFLSPD-----EFRQRIANNEFLEYEE---V
                                                                      . :

sp|P01034|CYTC_HUMAN       SNDMYHSRALQVVRARKQIVAGVNYFLDVELGRTTCTKTQPNLDNCPFHDQPHLKRKAFC  ...
sp|Q91195|CYT_ONCMY        TNDMFVRQVAKVVNAQKQVVSGMKYIFTVQMGRTPCRKGG-VEKVCSVHKDPQMAVPYKC  ...
sp|P31726|CYT1_MAIZE       ANALLG--FEKLVKAKTQVVAGTMYYLTIEVKDGEVKK----LYEAKVWEKPWENFKQLQ  ...
sp|P22085|CYTX_ONCVO       SNDEYHLMPIKLLKVSSQVVAGVKYKMDVQVARSQCKKSSNEKVDLTKCKKLEGHPEKVM  ...
sp|P01044|KNG1_BOVIN       HDGAFYFKIDTVKKATVQVVAGLKYSIVFIARETTCSKGS-NEELTKSCEINIHGQILHC  ...
sp|P54570|ADPP_BACSU       KGEEPEYTALRELEEETGYTAKKLTKITAFYTSPGFADEIVHVFLAEELSVLEEKRELDE  ...
sp|O07107|MRAY_ENTFA       FYGVFIIFWLVGFSNAVNLTDGIDG-LVAGLGTISFGTYAIIAWKQQFDVVIICLSVIG  ...
sp|Q64PY1|KGUA_BACFR       YTDRFYG--TLKAQVEKQLAAGQNVVFDVDVVGGCNIKKYYGERALSLFIQPPCIDELRR  ...
                                              .      :
```

(a)

```
sp|P01034|CYTC_HUMAN  ... --A----L-----Q---V------------V---RA-R---------------KQIVAGVNYFLDVE-L---------...
sp|Q91195|CYT_ONCMY   ... --Q----V-A---K---V------------V---NA-Q---------------KQVVSGMKYIFTVQ-M---------...
sp|P31726|CYT1_MAIZE  ... -------F-E---K---L------------V---KA-K---------------TQVVAGTMYYLTIE-V---------...
sp|P22085|CYTX_ONCVO  ... --P----I-----K---L------------L---KV-S---------------SQVVAGVKYKMDVQ-V---------...
sp|P01044|KNG1_BOVIN  ... FVQPPTRLCAGCPKPIPVDSPDLEEPLSHSIAKLNAEHDGAFYFKIDTVKKATVQVVAGLKYSIVFI-----------...
sp|P54570|ADPP_BACSU  ... ---------A---K---------------------------------------EQIFSGKVIDLYVEDVELPN-----...
sp|O07107|MRAY_ENTFA  ... --DF---IKVFK-KRN-MG-----------L---NSRQKLI------------GQIFGGLVFYFVYRSE---------...
sp|Q64PY1|KGUA_BACFR  ... --E----VYTD--RFYGT------------L---KA-QVE-------------KQLAAGQNVVFDVDVVGGCNIKKYY...
```

(b)

Figure 6.3: (a)ClustalW aligns 8 sequences in Dataset2 with one pattern; (b)PCMSA aligns 8 sequences in Dataset2 with one pattern

6.5 Summary and discussions

The proposed algorithm effectively brings the information available from the existing pattern databases into multiple sequence alignment. Researchers can now pay more attention to the biological characters on sequence alignment than common string sequence alignment. For the rapid increase of protein-coding sequences, the pattern-based algorithm becomes more important for interpreting the large volume of sequence data.

Chapter 7

Conclusions and Future Work

This final chapter concludes the research work in the PhD study and discusses various aspects of future work for better HPC solutions to the problems of computational biology and exploration of more applications.

7.1 Conclusions

The objective of this study is to use the power of high performance computing to speed up the processes of sequence analysis, and to avoid compromising with incomplete results, missing some optimal results and shallow computing on large datasets.

We began by studying the algorithms of sequence comparison. The complexity of sequence comparison problems and the necessity to deal with large-scale biological sequences makes the development of fast algorithms with low memory requirements becomes a great concern. To meet these demands, two new algorithms for analyzing these biological sequences are presented to gain parallel computing power at low cost. The first one is a "block-based wavefront" algorithm developed to speedup optimal pairwise alignment. Evaluation of the experimental results shows that the proposed algorithm can take the advantage of dynamic programming technique and utilize parallel computing to meet the requirements of comparing long sequences without compromising the optimal results. The second one is a fast and practicable algorithm for multiple sequence alignment. The proposed algorithm effectively parallelized the stage of building guiding trees and the other stages of the ClustalW algorithm. It results in significantly better performance than other comparable implementations.

We then proceeded to the reconstruction of phylogenetic trees. One of the "grand challenges" in HPC computational biology is the computation of the "Tree of life" containing all living organisms. Currently the most accurate methods for the reconstruction of phylogenetic trees are based on maximum parsimony (MP) and maximum likelihood (ML). Among MP and ML, Rec-I-DCM3 and RAxML are the fastest and most accurate programs respectively. In this research, a flexible parallel divide-and-conquer model was designed, which significantly outperforms REC-I-DCM3 and RAxML on all the real-world tested data sets containing from 5000 to 13,921 organisms. With respect to memory requirements and reasonable inference time, it appears to be the only model which is currently capable to handle huge alignments (over 5.000 taxa). If we take the Dataset 1 as an example, the best trial of the model finds a better score than any previous method. Achieving the best known score took about 123 hours by the Rec-I-DCM3, whereas the proposed model found it within 24 hours.

We also designed a pattern-constrained multiple sequence alignment algorithm to ensure that the generated alignment satisfies the criterion that crucial residues are aligned together. The proposed method effectively brings the information available in existing pattern databases into multiple sequence alignment. It is able to differentiate patterns which need to be aligned together from other residues, an essential property for accurate and biologically meaningful sequence alignment. It is proven that the similarity score worked out by this method has an approximation ratio of $\frac{k}{2(k-1)}$ to that of the optimal pattern-constrained alignment, where k is the number of sequences.

We integrated these solutions into a parallel sequence analysis system on a customized multi-node cluster, which significantly improves computational performance on sequence analysis problems in computational biology. It should be pointed out that such a parallel and distributed system is developed for a general MIMD architecture, which is widely available in computational biology laboratories, instead of an expensive special hardware design. Applications of these novel algorithms avoid prohibitively long computations for full datasets, which is essential for meaningful interpretation of empirical data. Researchers, working on problems involving enormous computations, complex optimizations etc, will benefit from using the system.

7.2 Future Work

In the future, we hope to further improve our system regarding the previously addressed shortfalls and to add more extensive functions for other complex problems in computational biology.

7.2.1 Computation of the "tree of life"

Despite the fact that the parallel divide-and-conquer model currently enables the computation of comparatively large trees, the scalability and efficiency of pPhylo(MP) and pPhylo(ML) still needs to be improved. We have discussed the technical as well as algorithmic problems and limitations concerning the parallelization of the *global method* and the load imbalance within the *base method* in Chapter 5. Thus, the development of a more scalable parallel algorithm for *global* optimization and a more thorough investigation of subproblem load imbalance constitute the main issue of future work.

One straight-forward technical solution to this problem is the design of a hybrid message-passing/shared-memory (MPI/OpenMP) parallelization of global rearrangement algorithm, for two reasons: *Firstly*, the fine-grained shared memory parallelization leads to super linear speedups on large datasets (Stamatakis, Ott & Ludwig 2005) due to improved cache efficiency; it will help to resolve memory shortage problems on very large and long alignments containing more than 10,000 sequences. *Secondly*, such an implementation will reduce the number of working processes which has a significant impact on the parallel efficiency of the current implementation. This reduction in the number of workers while maintaining the same amount of computational power will have a positive impact on the performance of the *base* as well as the *global* method.

7.2.2 Sequence analysis using grid computing

Parallel computing on a cluster of compute nodes has been already used successfully in sequence analysis, but it is not enough, due to the limit of the number of compute nodes. Grid computing has a potential for the expansion in computing performance by connecting a large number of computers or clusters with high performance networks. Grid system can incorporate more computational resources and help us speedup the experimental time. In the future, we want to integrate the grid technology and our system to implement all kinds of sequence analysis applications, so that we can speed up the sequence analysis time by many compute nodes of the grid system.

7.2.3 Development of more applications

We have focused mainly on boosting sequence analysis. Other problems in computational biology remain unsolved as far as parallel computing is concerned. We are interested in more challenging algorithms in computational biology and hope to extend our work in the years ahead.

Protein structure alignment

Computer programs have been created that give scientists the ability to look at the three dimensional shape of proteins. Examining a protein in three dimensions allows for greater understanding of protein functions. A complete three-dimensional structure is synthesized by tertiary structure. Unfortunately predicting protein structure from sequences is not an easy task. The common technique used is the comparison of a proposed shape to a database of known shapes, which is called structural alignment. It is of a value for the comparison of proteins in the so called "twilight zone", where the proteins are no longer recognizable because their sequences have changed too much through mutations, insertions and deletions during their evolution, but often their structures are preserved well. Protein structural alignment plays an important role in protein structure prediction, fold family classification, motif finding, and phylogenetic tree reconstruction and etc. It is analogous to sequence alignment, but more challenging because they are complex three-dimensional structures. There are some characteristics of the protein structure comparison that suggest HPC: large databases, frequent updates of the databases, high computational requirements and visualization.

Multiple genome alignment

In this area, the most striking change is the sheer volume of the available data. In 1995, the first two publications of microbial whole genome sequencing projects were published. Only several years later, there are almost 60 completed and annotated genomes available, such as eubacteria, archaebacteria, yeast and etc. Multiple genome alignment seeks to identify all similarities between a set of genome sequences. It will help us to identify differences between organisms, find functional assignment, evolution history and conserved region. However, multiple genome alignment is a challenging task due to its high demands for computational power and memory capacity. Additionally, genomes have some different natures with single protein sequence or DNA sequence containing a single gene or operon, such as rearrangements (e.g. exon shuffling or other non-syntenous regions resulting from intra-molecular recombinations), large insertions or deletions (sequences that share several regions of local similarity separated by unrelated regions), repeated elements (e.g. duplicated genes/operons, transposons, SINES,LINES etc.), tandem repeats, and inherent problems of gene regulatory elements, including their small size and relative resistance to small insertions/deletions or substitutions. Therefore, it needs a faster algorithm to align whole genomes with reliable output, reasonable cost and better display. This may present a serious challenge for efficient parallel execution.

Analyzing the evolution of virus

Phylogenetic analysis and ancestral inference of related nucleic acid or protein sequences are widely used to address problems in virus evolution, and have played a very important role in the prediction and identification of newly emerging etiological agents. Due to larger population sizes, shorter generation time and higher mutation rates, virus evolves on a faster time scale than larger organisms. Hence, the study of virus evolution often requires that existing methods of analysis are modified or that novel methods are developed. This future work is reconstructing the evolutionary history of different virus and paying particular attention to understanding their origins and the reasons for their emergence, persistence and virulence. It must involve the comparative analysis of pathogen sequence data in an evolutionary framework because these factors are the direct consequence of viral evolution. For example, the extensive sequencing and phylogenetic projects has revealed the diversity and complex reticulate evolutionary relationships of HIV. The extremely high rate of sequence change of HIV has had important implications for its control by vaccination. The urgency for effective control of HIV by vaccination is greater because more than thirty five million people currently infected. Therefore, the work is extremely important.

Bibliography

Adachi, J. & Hasegawa, M. (1996), Molphy version 2.3: programs for molecular phylogenetics based on maximum likelihood., *in* 'Computer Science Monographs', Vol. 28, Institute of Statistical Mathematics, Minato-ku, Tokyo.

A.J.Barrett (1987), 'The cystatins: a new class of peptidase inhibitors', *Trends Biochem. Sci.* **12**, 193–196.

Allen, B. & Steel, M. (2001), 'Subtree transfer operations and their induced metrics on evolutionary trees.', *Annals of combinatorics* **5**, 1–13.

Alpern, B., Carter, L. & Gatlin, K. (1995), 'Microparallellism and high performance protein matching', *In Proceeding of Supercomputing '95,San Diego, California,December3-8, ACM SIGARCH and IEEE Computer Society* .

Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D. (1990), 'Basic local alignment search tool', *J. Mol. Biol.* **215**, 403–410.

Altschul, S., Madden, T., schaffer, A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D. (1997), 'Gapped blast and psi-blast: a new generation of protein database search programs', *Nucleic Acids Res.* **25**, 3389–3402.

Amdahl, G. (1967), 'Validaity of the single processor approach achieving large-scale computing capabilities.', *In AFIPS Conference Proceedings. AFIPS Press,Reston* **30**, 483–485.

Attwood, T., Flower, D., Lewis, A., Mabey, J., Morgan, S., Scordis, P., Selley, J. & Wright, W. (1999), 'Prints prepares for the new millennium', *Nucleic Acids Res.* **27(1)**, 220–225.

A.Wozniak (1997), 'Using video-oriented instructions to speed up sequence comparison', *Computer Applications in the Biosciences* **13(2)**, 145–150.

Bateman, A., Birney, E., Durbin, R., Eddy, S., Finn, R. & Sonnhammer, E. (1999), 'Pfam 3.1: 1313 multiple alignments and profile hmms match the majority of proteins', *Nucleic Acids Res.* **27(1)**, 260–262.

Blas, A. D. & et. al (2005), 'The ucsc kestrel parallel processor', *IEEE Transactions on Parallel and Distributed Systems* **16**(1), 80–92.

Board., O. A. R. (1977), Openmp: A proposed industry standard api for shared memory programming.., *in* 'http://www.openmp.org'.

Bodlaender, H. L., Fellows, M. R., Hallett, M. T., Wareham, T. & Warnow, T. (2000), 'The hardness of perfect phylogeny, feasible register assignment and other problems on thin colored graphs', *Theoretical Computer Science* **244**, 167–188.

Borah, M., Bajwa, R., Hannenhalli, S. & Irwin, M. (1994), A simd solution to the sequence comparison problem on the mgap, *in* 'Proc. ASAP'94', IEEE CS, pp. 144–160.

Brauer, M., Holder, M., Dries, L., Zwickl, D., Lewis, P. & Hillis, D. (2002), 'Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference.', *Molecular Biology and Evolution* **19**, 1717–1726.

Brocchieri, L. & Karlin, S. (1998), 'Asymetric-iterated multiple alignment of protein sequences.', *JMB.* **276**, 249–264.

Bruno, W., Socci, N. & Halpern, A. (2000), 'Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction', *Mol. Biol. Evol.* **17**, 189–197.

Brutlag, D., Dautricourt, J., Diaz, R., Fier, J., Moxon, B. & Stamm, R. (1997), 'Blaze: An implementation of the smith-waterman sequence comparison algorithm on a massively paralllel computer', *Computers and Chemistry* **17**, 203–207.

Bucka-Lassen, K., Caprani, O. & Hein, J. (1999), 'Combining many multiple alignments in one improved alignment.', *Bioinformatics.* **15(2)**, 122–130.

Camin, J. & Sokal, R. (1965), 'A method for deducing branching sequences in phylogeny.', *Evolution.* **19**, 311–326.

Carlson, W., Draper, J., Culler, D., Yelick, K., Brooks, E. & Warren, K. (1999), 'Aintrodction to upc and language specification.', *Technical Report CCSTR-99-157* .

Catalyurek, U., Stahlberg, E., Ferreira, R., Kurc, T. & Saltz, J. (2002), 'Improving performance of multiple sequence alignment analysis in multi-client environments', *Workshop on High Performance Computational Biology, held in conjunction with International Parallel and Distributed Processing Symposium.* .

Chow, E., Hunkapiller, T., Peterson, J. & Waterman, M. (1991), Biological information signal processor, *in* 'Proc. ASAP'91', IEEE CS, pp. 144–160.

Comet, J.-P. & Henry, J. (2002), 'Pairwise sequence alignment using a prosite pattern-derived similarity score', *Computers and Chemistry.* **26**, 421–436.

Corpet, F., Gouzy, J. & Kahn, D. (1998), 'The prodom database of protein domain families', *Nucleic Acids Res.* **26(1)**, 323–326.

D.A.Bader (2004), 'Computational biology and high-performance computing', *Communications of the ACM.* **47**, 35–40.

Day, W. H. E., Johnson, D. S. & Sankoff, D. (1986), 'he computational complexity of inferring rooted phylogenies by parsimony.', *Math. Bios.* **81**, 33–42.

Doolittle, R., Hunkapiller, M., Hood, L., Devare, S., Robbins, K., Aaronson, S. & Antoniads, H. S. s. (1983), 'virus one gene v-sis, is derived from the gene encoding a platelet-derived growth factor', *Science.* **221**, 275–277.

D.P.Lopresti (1987), 'P-nac: A systolic array for comparing nucleic acid sequences', *Computer* **20**(7), 98–99.

E. Lander, J. M. & Taylor, W. (1988), 'Protein sequence comparison on a data parallel computer', *proc. International Conference on Parallel Processing.* pp. 257–263.

Ebedes, J. & Datta, A. (2004), 'Multiple sequence alignment in parallel on a workstation cluster.', *Bioinformatics* **20(7)**, 1193–1195.

Edgar, R. C. (2004*a*), 'Muscle: a multiple sequence alignment method with reduced time and space complexity.', *BMC Bioinformatics* **5(1)**, 113.

Edgar, R. C. (2004*b*), 'Muscle: multiple sequence alignment with high accuracy and high throughput.', *Nucl. Acids Res.* **32(5)**, 1792–1797.

Eisen, J. A. (2003), 'Phylogenomics: intersection of evolution and genomics', *Science* **300**, 1706–1707.

E.W. Edmiston, N.G. Core, J. S. & Smith, R. (1988), 'Parallel processing of biological sequence comparison algorithms', *International Journal of Parallel Programming* **17**(3), 259–275.

Felsenstein, J. (1980), 'A simple method for estimating evolutionary rates of base substitutions by through comparative studies of nucleotide sequences.', *J. Mol. Evol.* **16**, 111–120.

Felsenstein, J. (1981*a*), 'Evolutionary trees from dna sequences: A maximum likelihood approach.', *J. Mol. Evol.* **17**, 368–376.

Felsenstein, J. (1981*b*), 'Evolutionary trees from DNA sequences: A maximum likelihood approach', *Journal of Molecular Evolution* **17**, 368–376.

Felsenstein, J. (1989), 'Phylip - phylogeny inference package (version 3.2).', *Cladistics* **5**, 164–166.

Felsenstein, J. (2004), 'Phylip (phylogeny inference package) version 3.6'. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle.

Feng, D. & Doolittle, R. (1987), 'Progressive sequence alignment as a prerequisite to correct phylogenetic tress', *J.Mol.Evol.* **25**, 451–360.

Flynn.M. (1972), 'Some computer organizations and their effectiveness.', *IEEE Trans. Comput.* **C-21**, 94.

Foulds, L. R. & Graham, R. L. (1982), 'The Steiner problem in phylogeny is NP-complete', *Advances in Applied Mathematics* **3**, 43–49.

Gascuel, O. (1997), 'Bionj: an improved version of the nj algorithm based on a simple model of sequence data.', *Mol. Biol. Evol.* **14**, 685–695.

Geist, A., Begulin, A., J.Dongarra, W.Jiang, R.Manchek & V.Sunderam. (1994), *PVM-Parallel Virtual Machine- A User's Guide and Tutorial for Networked Parallel computing*, MIT Press.

G.Giribet (2005), 'A review of tnt: Tree analysis using new technology', *Systematic Biology* **54(1)**, 176–178.

Gibson, T., D., H. & Thompson, J. (2002), 'On-line help for clustal w', *http://www-igbmc.ustrasbg.fr/BioInfo/ClustalW/* .

Gish, W. & States, D. (1993), 'Identification of protein coding regions by database similarity search', *Nature Genet.* **3**, 266–272.

G.M.Nagel & R.F.Doolittle (1991), 'volution and relatedness in two aminoacyl-trna synthetase families.', *Proc. Natl. Acad. Sci.U.S.A.* **88**, 8121–8125.

Goloboff, P. (1999), 'Analyzing large data sets in reasonable times: solution for composite optima.', *Cladistics.* **15**, 415–428.

Gotoh, O. (1996), 'Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments', *J.Mol.Biol.* **264**, 823–838.

Gracy, J. & Argos, P. (1998), 'Automated protein sequence database classification. ii. delineation of domain boundaries from sequence similarities', *Bioinformatics.* **14**, 174–187.

Guerdoux-Jamet, P. & Lavenier, D. (1997), 'Samba: hardware accelerator for biological sequence comparison', *CABIOS* **12**(6), 609–615.

Guindon, S. & Gascuel, O. (2003), 'A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood.', *Syst. Biol.* **52**(5), 696–704.

Hart, D., Grover, D., M.Liggett, Repasky, R., Shields, C., Simms, S., Sweeny, A. & Wang, P. (2003), Distributed parallel computing using windows desktop system., *in* 'Proceedings of CLADE'.

Henikoff, J., Greene, E., Pietrokovski, S. & Henikoff, S. (2000), 'Increased coverage of protein families with the blocks database servers', *Nucleic Acids Res* **28**, 228–230.

Henikoff, S., Henikoff, J. & Pietrokovski, S. (1999), 'Blocks+: A non-redundant database of protein alignment blocks dervied from multiple compilations', *Bioinformatics.* **15(6)**, 471–479.

Heringa, J. (1999), 'Two strategies for sequence comparison: profile preprocessed and secondary structure-induced multiple alignment', *Computational Chemistry.* **23**, 341–364.

Hewlett-Packard (2001), 'Alphaserver sc:terascale single-system-image supercomputing', *White Papers.* .

Higgins, D., Thompson, J. & Gibson, T. (1996), 'Using clustal for multiple sequence alignments', *Methods Enzymol.* **266**, 383–402.

Hoang, D. (1993), Searching genetic databases on splash 2, *in* 'Proc. IEEE Workshop on FPGAs for Custom Computing Machines', IEEE CS, pp. 185–191.

Hofmann, K., Bucher, P., Falquet, L. & Bairoch, A. (1999), 'The prosite database', *Nucleic Acids Res.* **27**, 215–219.

http://www.timelogic.com (n.d.).

Huang, X. & Miller, W. (1991), 'A time-efficient linear-space local similarity algorithm', *Adv.Appl.Math.* **12**, 337–357.

Huson, D., Nettles, S. & Warnow, T. (1999), 'Disk-covering, a fast-converging method for phylogenetic tree reconstruction', *Journal of Computational Biology* **6**, 369–386.

Huson, D., Vawter, L. & Warnow, T. (1999), Solving large scale phylogenetic problems using DCM2, *in* 'Proc. 7th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB'99)', AAAI Press, pp. 118–129.

Jukes, T. & Cantor, C. (1969), 'Evolution of protein molecules.', *In H. Munro (editor), Mammalian protein metabolism,Academic Press, New York* **III**, 21–132.

Kallerjo, M., Farris, J. S., Chase, M. W., Bremer, B. & Fay, M. F. (1998), 'Simultaneous parsimony jackknife analysis of 2538 *rbcL* DNA sequences reveals support for major clades of green plants, land plants, seed plants, and flowering plants', *Plant. Syst. Evol.* **213**, 259–287.

Katoh, K., Misawa, K., Kuma, K.-i., & Miyata, T. (2002), 'Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform.', *Nucl. Acids Res.* **30(14)**, 3059–3066.

Keane, T., Naughton, T., Travers, S., McInerney, J. & McCormack, G. (2005), 'Dprml: Distributed phylogeny reconstruction by maximum likelihood.', *Bioinformatics* **21**(7), 969–974.

kleinjung, J., Douglas, N. & Heringa, J. (2002), 'Parallelized multiple alignment', *Bioinformatics.* **18**, 1270–1271.

La, D., Sutch, B. & Livesay, D. R. (2005), 'Predicting protein functional sites with phylogenetic motifs', *Proteins: Structure, Function, and Bioinformatics* **58**(2), 309–320.

Lanave, C., Preparata, G., Saccone, C. & Serio, G. (1984), 'A new method for calculating evolutionary substitution rates.', *J. Mol. Evol.* **20**, 86–93.

Lee, C., Grasso, C. & Sharlow, M. F. (2002), 'Multiple sequence alignment using partial order graphs.', *Bioinformatics* **18(3))**, 452–464.

Li, K.-B. (2003), 'Clustalw-mpi: Clustalw analysis using distributed and parallel computing.', *Bioinformatics* **19**, 1585–1586.

Lichtarge, O., Bourne, H. R. & Cohen, F. E. (1996), 'An evolutionary trace method defines binding surfaces common to protein families', *J. Mol. Biol.* **257**, 342–358.

Maddison, D. (1991), 'The discovery and importance of multiple islands of most parsimonious trees.', *Systematic Biology.* **42(2)**, 200–210.

Maidak, B. e. a. (2000), 'The rdp (ribosomal database project) continues.', *Nucleic Acids Research.* **28**, 173–174.

M.Dayhoff, R.M.Schwartz & B.C.Orcutt (1978), 'A model of evolutionary change in proteins', *Atlas of Protein Sequence and Structure.* **30**, 345–352.

M.Delarue & D.Moras (1993), 'The aminoacyl-trna synthetase family: modules at work', *BioEssays* **15**, 675–687.

M.Gokhale & et al. (1995), 'Processing in memory: The terasys massively parallel pim array.', *Computer* **28**(4), 23–31.

Mikhailov, D., Cofer, H. & Gomperts, R. (n.d.), 'Performance optimization of clustalw: parallel clustalw, ht clustal, and multiclustal.', *White papers, Silicon Graphics, Mountain View, CA.* .

Moret, B., Roshan, U. & Warnow, T. (2002), Sequence length requirements for phylogenetic methods, *in* 'Proc. 2nd Int'l Workshop Algorithms in Bioinformatics (WABI'02)', Vol. 2452 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 343–356.

Morgenstern, B. (1999), 'Dialign2:improvement of the segment-to-segment approach to multiple sequence alignment', *Bioinformatics.* **15**, 211–218.

Morgenstern, B. (2004), 'Dialign: multiple dna and protein sequence alignment at bibiserv.', *Nucl. Acids Res.*. **32(suppl 2)**, 33–36.

Morgenstern, B., Frech, K., Dress, A. & Werner, T. (1998), 'Dialign:finding local similarities by multiple sequence alignment.', *Bioinformatics*. **14**, 290–294.

MPI standard 2.0. (1997), MPI-Forum, MPI.

Nakhleh, L., Moret, B., Roshan, U., John, K. S. & Warnow, T. (2002), The accuracy of fast phylogenetic methods for large datasets, *in* 'Proc. 7th Pacific Symp. Biocomputing (PSB'2002)', World Scientific Pub., pp. 211–222.

Nakhleh, L., Roshan, U., St. John, K., Sun, J. & Warnow, T. (2001*a*), Designing fast converging phylogenetic methods, *in* 'Proc. 9th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB'01)', Vol. 17 of *Bioinformatics*, Oxford U. Press, pp. S190–S198.

Nakhleh, L., Roshan, U., St. John, K., Sun, J. & Warnow, T. (2001*b*), The performance of phylogenetic methods on trees of bounded diameter, *in* 'Proc. 1st Int'l Workshop Algorithms in Bioinformatics (WABI'01)', Vol. 2149 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 214–226.

N.D.Rawlings & A.J.Barrett (1990), 'Evolution of proteins of the cystatin superfamily', *J. Mol. Evol.* **30**, 60–71.

Nixon, K. C. (1999), 'The parsimony ratchet, a new method for rapid parsimony analysis', *Cladistics* **15**, 407–414.

Notredame, C., Higgins, D. & Heringa, J. (2000), 'T-coffee: A novel method for multiple sequence alignments', *Journal of Molecular Biology* **302**, 205–217.

Notredame, C., Higgins, D. & J., H. (2000), 'T-coffe: A novel method for fast and accurate multiple sequence alignment.', *J. Mol. Biol.* **302**, 205–217.

Notredame, C., Holm, L. & HIggins, D. (1998), 'Coffee:an objective function for multiple sequence alignment', *Bioinformatics.* **14(5)**, 407–422.

of the IEEE, P. A. S. C. (1996), Portable applications standards committee of the ieee,information technology-portable operation system interface (posix)-part 1: System application program interface (api)., *in* 'ISO/IEC 9945-1, ANSI/IEEE Std. 1003.1.'.

O.Gotoh (1990), 'Optimal sequence alignment allowing for long gaps', *Bull. Math. Biol.* **52**(3), 359–373.

Oliver, T., Schmidt, B. & Maskell, D. (2005), Reconfigurable architectures for bio-sequence database scanning on fpgas. to ne published in IEEE Transaction on Circuits and Systems II.

Page, R. (1993), 'On islands of trees and efficacy of different methods of branch swapping in finding most parsimonious trees', *Systematic Biology* **42(1)**, 200–210.

Pearson & Lipman (1988), 'Improved tools for biological sequence comparison', *Proc.Natl.Acad.Sci.USA*. **85**, 2444–2448.

Pei, J., Sadreyev, R. & Grishin, N. V. (2003), 'Pcma: fast and accurate multiple sequence alignment based on profile consistency.', *Bioinformatics* **19(3)**), 427–428.

P.Hovenkamp (2004), 'Review of tnt tree analysis using new technology, version 1.0.', *Cladistics* **20**, 378–383.

P.Schimmel (1987), 'Aminoacyl trna synthetases: general scheme of structure-function relationships in the polypeptides and recognition of transfer rnas', *Annu. Rev. Biochem* **56**, 125–158.

P.Schimmel (1991), 'Classes of aminoacyl-trna synthetases and the establishment of the genetic code', *Trends Biochem. Sci.* **16**, 1–3.

Quinn, M. (2004), *Parallel Programming in C with MPI and OpenMP*, McGrawHill.

Ranwez, V. & Gascuel, O. (2002), 'Improvement of distance-based phylogenetic methods by a local maximum likelihood approach using triplets.', *Mol Biol Evol* **19**, 1952–1963.

R.Chandra, L.Dagum, D. D. J. & R.menon. (2001), *Parallel programming in OpenMP*, Academic Press.

Rice, K., Donoghue, M. & Olmstead, R. (1997), 'Analyzing large datasets: *rbcL* 500 revisited', *Systematic Biology* **46(3)**, 554–563.

R.K.Singh & et al. (1996), 'Bioscan: a network sharable computational resource for searching biosequence databases', *CABIOS* **12**(3), 191–196.

R.Meier & Ali, F. (2005), 'The newest kid on the parsimony block: Tnt (tree analysis using new technology)', *Systematic Entomology* **30**, 179–182.

Robert, C. E. (2004), 'Muscle: multiple sequence alignment with high accuracy and high throughput', *Nucleic Acids Research* **32**(5), 1792–1797.

Rodriguez, F., Oliver, J., Marin, A. & Medina, J. (1990), 'The general stochastic model of nucleotide substitution.', *J. Theor. Biol.* **142**, 485–501.

ROlsen, G. J., H. Matsuda, R. H. & Overbeek, R. (1994), 'fastdnaml: A tool for construction of phylogenetic trees of dna sequences using maximum likelihood.', *Computer Applications in the Biosciences (CABIOS)* **30**, 41–48.

Roshan, U. (2004), 'Algorithm techniques for improving the speed and accuracy of phylogenetic methods.', *PhD thesis* .

Roshan, U., Moret, B. M. E., Warnow, T. & Williams, T. L. (2004), 'Rec-i-dcm3: a fast algorithmic technique for reconstructing large phylogenetic trees', *Proceedings of the IEEE Computational Systems Bioinformatics conference (CSB),Stanford, California, USA* .

Rost, B. (1999), 'Twilight zone of protein sequence alignments', *Protein Eng.* **12**, 85–94.

Saitou, N. & Nei, M. (1987), 'The nigehbor-joining method: a new method for reconstructing phylogenetic tree.', *J Mol Evol* **4**, 406–425.

S.B.Needleman & C.D.Wunsch (1970), 'A general method applicable to the search for similarities in the amino acid sequence of two proteins', *J.Mol.Biol.* **48**, 443–453.

Schmidt, B., Schroder, H. & Schimmler, M. (2002), Massively parallel solutions for molecular sequence analysis, *in* 'Proc. 1st IEEE Int. Workshop on High Performance Computational Biology'.

Schmidt, H., Strimmer, K., Vingron, M. & Haeseler, A. (2002), 'Tree-puzzle: maximum likelihood phylogenetic analysis using quartets and parallel computing.', *Bioinformatics* **18**, 502–504.

Searls, D. B. (2003), 'Pharmacophylogenomics: gene, evolution, and drug targets', *Nature Reviews Drug Discovery* **2**, 613–623.

S.Henikoff & J.G.Henikoff (1992), 'Amino acid substitution matrices from protein blocks', *Proc.Natl.Acad.Sci.* **89**, 10915–10919.

S.Kleiman, D.shah & B.Smaalders (1996), Rec-i-dcm3: a fast algorithmic technique for reconstructing large phylogenetic trees, *in* 'Programming with Threads', Prentice Hall, Englewood Cliffs, NJ.

S.Lustigman, B.Brotman, T.Huima & A.M.Prince (1991), 'Characterization of an onchocerca volvulus cdna clone encoding a genus specific antigen present in infective larvae and adult worms', *Mol. Biochem. Parasitol.* **45**, 65–75.

Smith, T. & Waterman, M. (1981*a*), 'Comparison of biosequences', *Adv.Appl.Math.* **2**, 482–489.

Smith, T. & Waterman, M. (1981*b*), 'Identification of common molecular subsequences', *J.Mol.Biol.* **147**, 195–197.

S.Rajko & S.Aluru (2004), 'Space and time optimal parallel sequence alignments', *Transaction on Parallel and Distributed Systems.* **15(1)**, 1070–1081.

Srivastava, N. (2001), 'Aalphaserver sc45 system overview', *Compaq Computer Corporation, Marlboro, MA, USA* .

Stamatakis, A., Ludwig, T. & Meier, H. (2004), Parallel inference of a 10.000-taxon phylogeny with maximum likelihood., *in* 'Proceedings of 10th International Euro-Par Conference', pp. 997–1004.

Stamatakis, A., Ludwig, T. & Meier, H. (2005), 'Raxml-iii:a fast program for maximum likelihood-based inference of large phylogenetic trees.', *Bioinformatics* **21**(4), 456–463.

Stamatakis, A., Ott, M. & Ludwig, T. (2005), Raxml-omp: An efficient program for phylogenetic inference on smps, *in* 'Proceedings of of 8th International Conference on Parallel Computing Technologies (PaCT)'. Preprint available on-line at WWW.ICS.FORTH.GR/~STAMATAK.

Stamatakis, A., Ott, M., Ludwig, T. & Meier, H. (2005), Draxml@home: A distributed program for computation of large phylogenetic trees. To be published in Future Generation Computer Systems (FGCS).

Steel, M. A. (1994), 'The maximum likelihood point for a phylogenetic tree is not unique', *Systematic Biology* **43**(4), 560–564.

Stewart, C., Hart, D., Berry, D., Olsen, G., Wernert, E. & Fischer, W. (2001), Parallel implementation and performance of fastdnaml - a program for maximum likelihood phylogenetic inference., *in* 'Proceedings of SC2001'.

Strimmer, K. & Haeseler, A. (1996), 'Quartet puzzling: A maximum-likelihood method for reconstructing tree topologies.', *Mol. Biol. Evol.* **13**, 964–969.

Studier, J. & Keppler, K. (1988), 'A note on the neighbor joining method of saitou and nei.', *Mol. Biol. Evol.* **5**, 729C731.

Su, Q., Lu, L. & Brutlab, D. (2002), 'http://fold.stanford.edu/eblocks'.

Sum Mircosystems, I. (1995), Sun microsystems, inc. posix threads., *in* 'www.sun.com/developer-products/sig/threads/posix.html'.

S.V.Le, Schmidt, H. & Haeseler, A. (2004), Phynav: A novel approach to reconstruct large phylogenies, *in* 'Proceedings of GfKl conference'.

Thompson, J. D., Higgins, D. G. & Gibson, T. J. (1994*a*), 'Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice', *Nucleic Acids Res.* **22**(22), 4673–4680.

Thompson, J., Higgins, D. & Gibson, T. (1994*b*), 'Clustal w: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice', *Nucleic Acid Res.* **22**, 4673–4680.

T.L.Williams, B.M.E.Moret, T.Berger-Wolf, U.Roshan & T.Warnow (2004), 'The relationship between maximum parsi-mony scores and phylogenetic tree topologies.', *Technical Report TR-CS-2004-04, Department of Computer Science, The Univer-sity of New Mexico.* .

T.Rognes & E.Seeberg (2000), 'Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors', *Bioinformatics* **16**, 699–706.

U.Roshan, Moret, B., Williams, T. & Warnow, T. (2004*a*), 'Performance of supertertree methods on various dtaset decompositions. in o.r.p.binida-emonds, editor, phylogenetic super-trees: Combining information to reveal the tree of life', *Computational Biology.* **3**, 301–328.

U.Roshan, Moret, B., Williams, T. & Warnow, T. (2004*b*), 'Rec-i-dcm3: A fast algorithmic technique for reconstruct-ing large phylogenetic trees', *Proceedings of the IEEE Compu-tational Systems Bioinformatics conference.* .

VanWalle, I., Lasters, I. & Wyns, L. (2004), 'Align-m–a new algorithm for multiple alignment of highly divergent sequences', *Bioinformatics* **20**, 1428–1435.

V.Turk & W.Bode (1991), 'The cystatins: protein inhibitors of cysteine proteinases', *J. Mol. Evol.* **285**, 213–219.

Warnow, T., Moret, B. & St. John, K. (2001), Absolute convergence: True trees from short sequences, *in* 'Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'01)', SIAM Press, pp. 186–195.

Wolf, M., S.Easteal, Kahn, M., McKay, B. & L.Jermiin (2000), 'Trexml: A maximum likelihood program for extensive tree-space exploration.', *Bioinformatics* **16**(4), 383–394.

Wuyts, J., de Peer, Y. V., Winkelmans, T. & Wachter, R. D. (2002), 'The European database on small subunit ribosomal RNA', *Nucleic Acids Research* **30**, 183–185.

Yamaguchi, Y., Maruyama, T. & Konagaya, A. (2002), High speed homology search with fpgas, *in* 'Proc. Pacific Symposium on Biocomputing'02', pp. 271–282.

Yang, Z. (1997), 'Paml: a program package for phylogenetic analysis by maximum likelihood.', *Computer Applications in BioSciences* **13**, 555–556.

Zhou, B., Till, M., Zomaya, A. & Jermiin, L. (2004), Parallel implementation of maximum likelihood methods for phylogenetic analysis., *in* 'Proceeding of IPDPS2004'.

Appendix A

Sample Code for Jacobi Iteration

To establish a basic understanding and test on the multi-node cluster, we developed a protocol program, dealing with typical matrix problem of Jacobi iteration (solve Laplace's equation).

As illustrated in Figure A.1, the algorithm involves a main loop of an iterative solver where, at each iteration, the value at a point is replaced by the average of the North , South, East and West neighbours, as illustrated. Boundary values do not change. We focus on the inner loop, where most of the computation is done.

Since this algorithm has a simple structure, a data-parallel approach can be used to derive an equivalent parallel code. The array can be distributed across processes, and each process is assigned the task of updating the entries on the part of the array it owns.

A parallel algorithm is derived from a choice of data distribution. The data distribution should be balanced, allocating (roughly) the same number of entries to each processor; and it should minimize communication. There are two possible distributions, as shown in Figure A.2:

- 1D (block) distribution, where the matrix is partitioned in one dimension, and

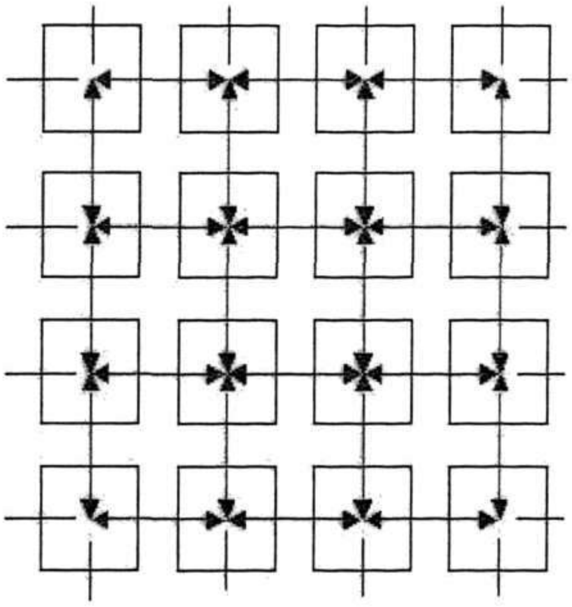


Figure A.1: Jacobi iteration, four-point stencil

- 2D (block,block) distribution, where the matrix is partitioned in two dimensions.

Since the communication occurs at block boundaries, communication volume is minimized by the 2D partition which has a better area to perimeter ratio. However, in this partition, each processor communicates with four neighbors, rather than two neighbors in the 1D partition. When the ratio of n/p (n is the problem size, p is the number of processors) is small, communication time will be dominated by the fixed overhead per message, and the first partition will lead to better performance. When the ratio is large, the second partition will result in better performance.

Suppose we use the second partition. Table A.1 gives the pseudo code of the parallel Jacobi algorithm. The **ghost points** are stored on other processors which are required for the calculation of the next iteration of the unknowns on a local processor. Communications are simplified if an overlap area is allocated at each process for storing the values to be received from the neighbor process. Essentially,

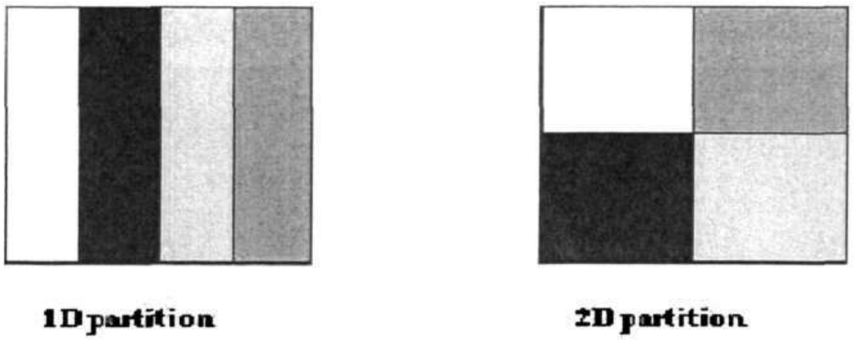


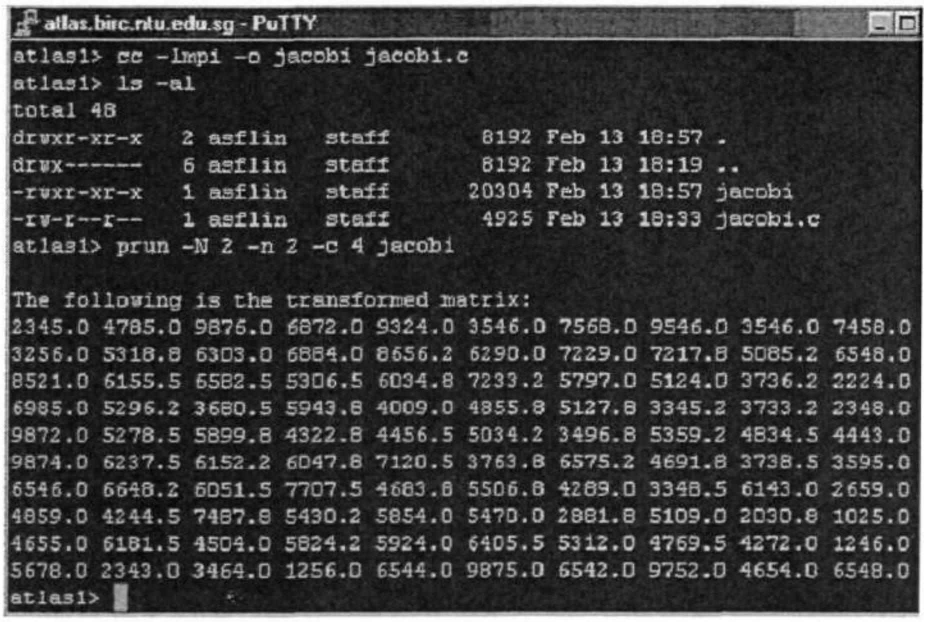
Figure A.2: Jacobi partition

storage is allocated for each entry both at the producer and at the consumer of that entry. If an entry is produced by one process and consumed by another, then storage is allocated for this entry at both processes. With such scheme there is no need for dynamic allocation of communication buffers, and the location of each variable is fixed. Such scheme works whenever the data dependencies in the computation are fixed and simple.

Table A.1: Parallelized Jacobi algorithm

```
Choose initial values for the **own mesh points** and the **ghost points**
Choose initial $Precision$ (e.g. $Precision = 10^{10}$)
While $Precision > \varepsilon$ (e.g. $\varepsilon = 10^{-5}$)
    1. Calculate next iteration for the **own domain**
    2. Send the new iteration on boundary of domain to neighboring processors
    3. Receive the new iteration for the **ghost points**
    4. Calculate $Precision$
End While
```

Figure A.3 shows the running parameters of the parallel program: resource request via prun; number of nodes (-N); number of processes (-n); number of CPUS per process (-c). Using this protocal, we tested various conditions for tuning our system and obtaining benchmarks in performance analysis.



```
atlas1> cc -lmpi -o jacobi jacobi.c
atlas1> ls -al
total 48
drwxr-xr-x   2 asflin   staff        8192 Feb 13 18:57 .
drwx------   6 asflin   staff        8192 Feb 13 18:19 ..
-rwxr-xr-x   1 asflin   staff       20304 Feb 13 18:57 jacobi
-rw-r--r--   1 asflin   staff        4925 Feb 13 18:33 jacobi.c
atlas1> prun -N 2 -n 2 -c 4 jacobi

The following is the transformed matrix:
2345.0 4785.0 9876.0 6872.0 9324.0 3546.0 7568.0 9546.0 3546.0 7458.0
3256.0 5318.8 6303.0 6884.0 8656.2 6290.0 7229.0 7217.8 5085.2 6548.0
8521.0 6155.5 6582.5 5306.5 6034.8 7233.2 5797.0 5124.0 3736.2 2224.0
6985.0 5296.2 3680.5 5943.8 4009.0 4855.8 5127.8 3345.2 3733.2 2348.0
9872.0 5278.5 5899.8 4322.8 4456.5 5034.2 3496.8 5359.2 4834.5 4443.0
9874.0 6237.5 6152.2 6047.8 7120.5 3763.8 6575.2 4691.8 3738.5 3595.0
6546.0 6648.2 6051.5 7707.5 4683.8 5506.8 4289.0 3348.5 6143.0 2659.0
4859.0 4244.5 7487.8 5430.2 5854.0 5470.0 2881.8 5109.0 2030.8 1025.0
4655.0 6181.5 4504.0 5824.2 5924.0 6405.5 5312.0 4769.5 4272.0 1246.0
5678.0 2343.0 3464.0 1256.0 6544.0 9875.0 6542.0 9752.0 4654.0 6548.0
atlas1>
```

Figure A.3: Jacobi Results