pCLUSTAL - Parallel Multiple Sequence Alignment for PC Clusters

By
Sylvain Pitre

A thesis submitted to
The Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

December 2003

© Copyright
2003, Sylvain Pitre

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services

Acquisitions et
services bibliographiques

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-89861-X

Our file Notre référence

ISBN: 0-612-89861-X

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this dissertation.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de ce manuscrit.

While these forms may be included in the document page count, their removal does not represent any loss of content from the dissertation.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

One of the most common task performed by biologists are multiple sequence alignments. Biologists can align multiple sequences to infer phylogenetic trees, to highlight variable sites/regions, to predict protein structure and function and more. Years ago that task had to be done manually, making it long, arduous and prone to errors. Now biologists have access to several computer programs which will automate those alignments (i.e. CLUSTAL W).

Since biologists have this new computerized method of sequence alignments, the need grew to align more and more sequences. Traditional alignment tools available today can take hours or even days to align large numbers of sequences (800,900...). This thesis presents a parallel implementation of the CLUSTAL W alignment program using the communication library MPI for distributed PC Clusters. We will show the implementation as well as testing results on a distributed PC cluster, showing speedup and time improvements over the sequential version.

Acknowledgements

I would like to thank first and foremost my thesis supervisor Dr. Frank Dehne for all his help and knowledge without which this thesis would not have been written. His guidance and advice contributed to every step of this project from the beginning. I also want to thank Peter Taillon whose recommendations and help steered me into the correct direction. His knowledge and expertise has been invaluable and much appreciated.

A special thanks to biologists Dr. James Cheetham without whom this project would not have been started or completed. His knowledge provided many clarifications of biological subjects as well as a test user of our parallel version of CLUSTAL W. I would also like to thank Dr. Andrew Rau-Chaplin from Dalhousie University in Halifax N.S. which contributed to this project and provided access to their distributed PC cluster for testing.

Contents

Abstract	iii
Acknowledgements	iv
Contents	v
List of Tables	viii
List of Figures	ix
List of Algorithms	xi
List of Appendices	xii
1 Introduction	1
1.1 Background	1
1.1.1 Proteins and Amino Acids	1
1.1.2 DNA and RNA	3
1.2 Sequence Alignments	4
1.2.1 Definition	4
1.2.2 Local and Global Alignments	5
1.2.3 Gaps and Penalties	6
1.2.4 Scoring Matrices	6
1.3 Phylogenetic Trees	7
1.4 Why Parallel	9
1.5 Thesis Outline	9
2 Review	10
2.1 Multiple Sequence Alignment	10
2.1.1 MSA Software	11

2.1.2	Algorithm	13
2.1.3	Pairwise Alignment	14
2.1.4	Guide Tree	15
2.1.5	Progressive Alignment	16
2.1.6	Scoring	16
2.2	Previous Work	17
2.2.1	SGI	17
2.2.2	ClustalW-MPI	19
2.2.3	Other	20
3	Parallel Multiple Sequence Alignment	21
3.1	Description	21
3.2	Algorithm	21
3.2.1	Parallel Pairwise Alignment	22
3.2.2	Parallel Guide Tree	23
3.2.3	Parallel Progressive Alignment	24
3.3	Correctness	29
3.4	Performance	29
3.5	Example	30
4	Implementation	33
4.1	pCLUSTAL	33
4.2	Portal	37
5	Experimental Performance Analysis	44
5.1	Methodology	44
5.2	Data Sets	45
5.3	Experimental Results	46
5.4	Analysis	54
5.4.1	GLUCAGON	55

5.4.2	GPCR	56
5.4.3	KINASE	56
5.4.4	RELAXIN	57
5.4.5	SYNAPSIN	57
5.4.6	SYNAPTOTAGMIN	58
5.4.7	TRYPSIN	58
5.4.8	Summary & Discussion	59
6	Conclusion	61
6.1	Summary of Contributions	61
6.2	Open Problems	62
	Bibliography	63
A	Glossary	68
B	Command Line Parameters	70
C	Example Sequences	72

List of Tables

1.1	List of 20 amino acids found in proteins	2
4.1	Old and new guide tree algorithm run-times	34
4.2	Differences between CLUSTAL W and the previous pCLUSTAL version.	36
5.1	Data sets used in testing	45
5.2	Tree heights and balance coefficients (H, B)	55

List of Figures

1.1	Example of protein sequences	2
1.2	Example of a DNA sequence	3
1.3	Relationship between DNA, RNA and protein	4
1.4	A comparison of local and global sequence alignments	5
1.5	First 5 rows and columns of BLOSUM 62 scoring matrix. Using this matrix, substituting “C” with “P” will result in a -3 penalty	7
1.6	An example phylogenetic tree from a multiple alignment (from SYNAPSIN), drawn with the TreeView program	8
2.1	Screenshot of CLUSTAL X 1.83 (ClustalW for Windows)	11
2.2	Example of a distance matrix (from SYNAPSIN)	14
2.3	Example of a guide tree (from SYNAPSIN)	15
2.4	Relative Speedup using 100 and 600 GPCR protein sequences with average length of 390 amino acids [10]	18
2.5	Speedup for ClustalW-MPI for a 500-sequence data set [16]	20
3.1	Balanced (a) and unbalanced (b) guide tree	25
3.2	Studying the guide tree	26
3.3	SYNAPSIN guide tree	28
3.4	Parallel Progressive Alignment using the SYNAPSIN guide tree	29
3.5	Phases 1 and 2 of pCLUSTAL	31
3.6	Phase 3 of pCLUSTAL	32
4.1	Clustal XP Portal sequence input box and file upload	39
4.2	Clustal XP Portal simple settings	40
4.3	Clustal XP Portal Vertex Cover Parameters	40
4.4	Clustal XP Portal Pairwise and multiple alignment settings	41

4.5	Clustal XP Portal example run	42
4.6	Information flow of the Clustal XP Portal	43
5.1	Parallel Wall Clock times measured for GLUCAGON sequence set	46
5.2	Parallel Wall Clock times measured for GPCR sequence set	47
5.3	Parallel Wall Clock times measured for KINASE sequence set	47
5.4	Parallel Wall Clock times measured for RELAXIN sequence set	48
5.5	Parallel Wall Clock times measured for SYNAPSIN sequence set	48
5.6	Parallel Wall Clock times measured for SYNAPTOTAGMIN sequence set. .	49
5.7	Parallel Wall Clock times measured for TRYPSIN sequence set	49
5.8	Relative Speedups for GLUCAGON sequence set	50
5.9	Relative Speedups for GPCR sequence set	50
5.10	Relative Speedups for KINASE sequence set	51
5.11	Relative Speedups for RELAXIN sequence set	51
5.12	Relative Speedups for SYNAPSIN sequence set	52
5.13	Relative Speedups for SYNAPTOTAGMIN sequence set	52
5.14	Relative Speedups for TRYPSIN sequence set	53
5.15	Relative Speedup for the Pairwise phase for all sequence sets	53
5.16	Relative Total Speedup for all sequence sets	54

List of Algorithms

1	Sequential CLUSTAL W	13
2	Parallel CLUSTAL W	21
3	Parallel Pairwise Alignment	22
4	Parallel Guide Tree	23
5	Parallel Progressive Alignment	26
6	pCLUSTAL	36

List of Appendices

A	Glossary	68
B	Command Line Parameters	70
C	Example Protein Sequences	72

Chapter 1

Introduction

1.1 Background

To better understand the biology involved, we will first introduce required concepts and terminology. We'll define and explain proteins, amino acids and nucleic acids (DNA and RNA).

1.1.1 Proteins and Amino Acids

Proteins constitute a large part of our bodies and can exist in several different varieties. For example, structural proteins are used to build tissues and other proteins called enzymes are used as catalysts of chemical reactions. A protein can be defined as a chain of smaller molecules called amino acids. There are 20 amino acids commonly found in proteins, shown in Table 1.1 along with their respective symbol (alphabet). When amino acids join to produce a protein, what is left in the protein is what we call residues of the original amino acids. Therefore it is possible to see an author say a protein has 50 amino acids, while another will say it has 50 residues. The average length of a protein sequence is around 300 residues, but can vary from under 100 to over 5000 residues. We can see in Figure 1.1 an example of a protein sequence used in our experiments (from the RELAXIN dataset).

A	Alanine
C	Cysteine
D	Aspartic acid
E	Glutamic acid
F	Phenylalanine
G	Glycine
H	Histidine
I	Isoleucine
K	Lysine
L	Leucine
M	Methionine
N	Asparagine
P	Proline
Q	Glutamine
R	Arginine
S	Serine
T	Threonine
V	Valine
W	Tryptophan
Y	Tyrosine

Table 1.1: List of 20 amino acids found in proteins

```
>gi|1710088|sp|P51453|RELX_CAVPO Prorolaxin precursor
GFLDKVIKVCGRDLVRIKIDICGKILLGDMTTGQEKQRILGSGQSAEIMPSSINKEVDSL
NMLESIANLPEELRAMLPEKQPSSPQLQQYYPALKNSNVAVKELNKKIIRGRQEEAEDNSHSL
LKDFNLNIYSPKKRQLDMTVSEKCCQVGCTRRFIANS

>gi|1710080|sp|P51454|REL1_PANTR Prorolaxin H1 precursor
SRAVADSWMDEVIKLCGRELVRAQIAICGMSTWSKRSLSQEDAPQTPRPVAEIVPSFINKD
TETIIIMLEFIANLPELKAALSERQPSLPEPQQYYPALKDSNLSFEEFKKLIIRNRQSEAA
DSNPSELKYLGLDTHSQKQRQPYVALFEKCCCLIGCTKRSLANYC
```

Figure 1.1: Example of protein sequences.

1.1.2 DNA and RNA

DNA, or deoxyribonucleic acid, is similar to proteins since it is also a chain of simpler molecules. But like most people have already seen, DNA actually has two chains or strands. Unlike the 20 amino acids for the proteins, DNA is mainly composed of four bases: A (Adenine), T (Thymine), C (Cytosine) and G (Guanine). An example of a DNA sequence can be seen in Figure 1.2.

RNA, or ribonucleic acid, is very similar to DNA except for a few differences. Instead of containing thymine (T) it uses uracil (U). Its structure is also different from DNA; it does not form a double helix but has a more varied structure. We can use DNA or RNA sequences instead of proteins in sequences alignments.

```
>U03518 Aspergillus awamori internal transcribed spacer 1  
(ITS1)  
AACCTGCGGAAGGATCATTACCGAGTGCGGGTCCTTTGGGCCCAACCTCCCATCCGTGTCTAT  
TGTACCCTGTTGCTTCGGCGGGCCCGCCGCTTGTCGGCCGCCGGGGGGGCGCCCTGTCCCCC  
GGGCCCGTGCCCGCCGGAGACCCCAACACGAACACTGTCTGAAAGCGTGCAGCTGAGTTGAT  
TGAATGCAATCAGTTAAAACTTCAACAATGGATCTCTTGGTTCCGGC
```

Figure 1.2: Example of a DNA sequence.

We can see in Figure 1.3 the relationship between DNA, RNA and proteins. This information flow is said to be the central dogma of molecular biology. Replication is, as the name suggests, DNA duplicating itself. Transcription is the process by which the information contained in a section of DNA is transferred to a newly assembled piece of mRNA (messenger RNA). Reverse transcription (or reverse transcriptase) is an enzyme used by all retroviruses that transcribes the genetic information from the virus from RNA into DNA, which can integrate into the host genome. Finally, translation is the process of protein biosynthesis in which mRNA is used as a template to produce a specific protein according to the rules specified by the genetic code.

Figure 1.3: Relationship between DNA, RNA and protein.

1.2 Sequence Alignments

1.2.1 Definition

The alignment of DNA or protein sequences is one of the most common tasks performed in bioinformatics. The alignment of two sequences, called *pairwise* alignment, is the simplest form of alignment and is usually used for searching sequence databases. Sequences can be aligned in their entire length or only in certain regions (see Section 1.2.2). Multiple sequence alignments (MSA) are the alignment of more than two sequences and have many important uses:

- Organize data to reflect sequence homology
- Infer phylogenetic trees from homologous sequences
- Highlight conserved sites or regions
- Highlight variable sites or regions
- Uncover changes in gene or protein structure
- Homology modeling of protein structures

Traditionally, sequence alignments were done manually. While this was possible for two sequences, adding more sequences quickly becomes overwhelming and error-prone. Biologists therefore needed a way to computerize that arduous task. Several computer programs have since been developed for multiple sequence alignments (see

Section 2.1.1). Alignments are usually evaluated by a score, which is often the sum of all character matches minus the mismatches and gaps (section 1.2.3). Different programs calculate scores differently; we will go into more details about the scoring function we used in section 2.1.6.

1.2.2 Local and global alignments

Sequences can be aligned along their entire length (global alignment) or only in certain regions (local alignment). Global and local alignment can be used for both pairwise and multiple alignments. Global alignments use gaps (insertions/deletions) while local alignments only align regions between gaps (Figure 1.4). A global alignment compares the two sequences over their entire lengths, and is appropriate when comparing sequences that are expected to share similarity over their entire lengths. Global alignment maximizes regions of similarity and minimizes gaps using scoring matrices and gap parameters set by the user.

Figure 1.4: A comparison of local and global sequence alignments.

1.2.3 Gaps and penalties

As mentioned in section 1.2.2, gaps are used in global alignments where mismatches occur. When the alignment algorithm needs to insert a gap in order to match the sequences, we say it needs to *open* a gap. Then for each extra character it adds (residue alphabet) we *extend* that gap. Finally the algorithm *closes* the gap. Since adding blank or empty characters (usually a dash '-') implies the sequences were not identical, those operations need to be penalized. Therefore global alignment algorithms have user parameters to define how much penalty should be assigned.

Ideally the algorithm should introduce the minimum number of gaps in the alignment. While this is not always the case, the most expensive operation is generally opening a gap, followed by extending the gap. The closing of a gap is not penalized. Using those settings, the algorithm will tend to create fewer but longer gaps since it tries to accumulate the least penalties as possible.

1.2.4 Substitution Matrices

For protein alignments, a substitution matrix is used to determine the similarity of non-identical amino acids. Simply put, they are used during a mismatch to evaluate the possibility that an amino acid substitution is possible in nature over a certain period of evolution. Take the following example: Sequence 1 has A (Alanine) at position 1 and Sequence 2 has H (Histidine) at that same position. The algorithm will look in the scoring matrix to see if that substitution is likely (positive score) or unlikely (negative score). More common substitutions will be rewarded by a higher score in the matrix, while near impossible substitutions will be penalized heavily (Figure 1.5). The algorithm can then decide to consider that substitution or use a gap instead, whichever results in the highest overall score or smallest penalty.

There are three series of substitution matrices used in the software we used (CLUSTAL W). Each consists of several matrices which work differently at different evolutionary distances. In CLUSTAL W, several matrices are stored in memory,

spanning the full range of amino acid distance (from almost identical sequences to highly divergent ones). For very similar sequences, it is best to use a strict weight matrix which only gives a high score to identities and the most favored conservative substitutions. For more divergent sequences, it is appropriate to use more relaxed matrices which give a higher score to many other frequent substitutions. Most alignment software offers the possibility to choose a different scoring matrix to offer better alignments. The matrices available in CLUSTAL W are:

1. **BLOSUM** (Henikoff): Blosum 80, 62, 45 and 30.
2. **PAM** (Dayhoff): PAM 20, 60, 120 and 350.
3. **Gonnet**: GONNET 80, 120, 160, 250 and 350.
4. **Identity matrix**: gives a score of 1.0 to two identical amino acids and a score of zero otherwise (worst performing matrix).

	C	S	T	P	A
C	9	-1	-1	-3	0
S	-1	4	1	-1	1
T	-1	1	4	1	-1
P	-3	-1	1	7	-1
A	0	1	-1	-1	4

Figure 1.5: First 5 rows and columns of BLOSUM 62 scoring matrix. Using this matrix, substituting “C” with “P” will result in a -3 penalty.

1.3 Phylogenetic Trees

One objective of multiple sequence alignments can be the generation of fully annotated phylogenetic trees (Figure 1.6). Phylogenetic analysis is used to study evolution but involves several computational challenges. The generation of evolutionary trees is composed of two NP-complete problems: multiple sequence alignments and phylogenetic tree searches.

The quality of the phylogenetic analysis is affected by the quality of the multiple alignments. There are also other problems, such as orthologs and paralogs. Orthologs are sequences derived from a common ancestor through vertical descent (same gene in different species). Paralogs are genes within the same genome that have been generated by duplication. In order to build an accurate phylogenetic tree, we need to be able to differentiate orthologs and paralogs.

There are three common types of tree building algorithms: distance matrix, maximum likelihood and parsimony. Distance matrix methods estimate pairwise distances between the sequences (which reduces the information in the alignment to a single number). Other methods build several trees from the information in the multiple alignment and select the best tree. We will use the distance matrix method as used in CLUSTAL W.

Figure 1.6: An example phylogenetic tree from a multiple alignment (from SYNAPSIN), drawn with the TreeView program.

1.4 Why Parallel

As the number of DNA and protein sequences in databases increases, it is ever more important to be able to create multiple sequence alignments for very large numbers of sequences. Using standard multiple sequence alignment tools like CLUSTAL W [12] a set of 100 sequences can be aligned in under an hour on a typical home workstation. However, given that the underlying alignment algorithm requires $O(n^2)$ steps, where n is the number of sequences to be aligned, users quickly realize that alignments using more sequences can take hours or even days to complete.

Therefore an accessible parallel implementation of a multiple alignment program was needed. Since CLUSTAL W is one of the most used alignment software today, we decided to use it as the foundation of our parallel implementation.

1.5 Thesis Outline

In this thesis we will present a parallel implementation of CLUSTAL W for distributed PC Clusters. We will first look at the original CLUSTAL W implementation in Chapter 2 in order to give us an understanding of how it processes multiple alignments. We will look at different alignment software, describe the CLUSTAL W algorithms in more details and then in Section 2.2 we will explore the previous parallel implementations of CLUSTAL. In Chapter 3 we will present our parallel implementation of CLUSTAL W for PC Clusters. We will give an algorithm for the entire implementation as well as algorithms for individual phases. The load balancing scheme we used to distribute work in the final phase of the alignment will also be explained. We then analyze the performance of the algorithm and give an example of a multiple alignment. In Chapter 4 we will discuss the implementation issues we encountered as well as present the web portal CLUSTAL XP to allow users access to the alignment software. In Chapter 5 we will study the experimental performance analysis of the implementation. After presenting the methodology and the data sets used, we can observe the experimental result graphs and an analysis of those results.

Chapter 2

Review

2.1 Multiple sequence Alignment

As discussed in section 1.2, multiple sequence alignments (MSA) are the comparison of more than 2 protein or DNA sequences. There are several algorithms for MSA including:

- Using Dynamic Programming [24,31]
- Using the Traveling Salesman Approach [25]
- Using Evolutionary Programming [26]
- Using a Genetic Algorithm [27]
- Using the Divide and Conquer approach [32]
- Progressive Alignment (like CLUSTAL W) [30]

There are several software packages that use those algorithms (section 2.1.1), but we will limit ourselves to CLUSTAL W and its algorithms since our implementation is based on that program. In the following sections of this chapter we will explain in more detail the phases of CLUSTAL W in order to better understand the changes that needed to be implemented in the parallel version.

2.1.1 MSA Software

There are many available multiple sequence alignment software currently available and more are constantly being developed. We will present a few of the most commonly used MSA tools available.

CLUSTAL is one of the most widely used alignment software [3, 4, 5, 6, 7]. Over the past 15 years there have been several versions: CLUSTAL v [5], CLUSTAL W [12] and CLUSTAL X [11]. It is available for every platform including Linux, UNIX, Windows, Macintosh and Solaris. It uses pairwise alignments to build a guide tree using the Neighbor-Joining method [28], and a progressive alignment algorithm for multiple alignments [30]. We decided to base our implementation on CLUSTAL W since it is one of the most popular alignment software used by biologists. We will describe its algorithms in detail in the following sections.

Figure 2.1: Screenshot of CLUSTAL X 1.83 (ClustalW for Windows).

Another implementation by Florence Corpet called MultAlin [20] uses a similar approach than CLUSTAL. It also starts with a pairwise alignment of all sequences based on the conventional dynamic programming approach, but differs in the tree building process. Instead of using the Neighbor-Joining method [28], MultAlin creates an initial hierarchical clustering using the pairwise score matrix. Then it can start the multiple alignment iterations: groups of close sequences are aligned progressively together using pairwise alignments. The scores of those pairwise alignments are gathered to build a new matrix and a new hierarchical clustering. If the new clustering is different from the initial one, a new iteration can start using the new clustering. We repeat those iterations until we achieve the final multiple alignment. There is a portal online which offers a user control over some parameters to perform an alignment.

PileUp is yet another variation on the technique used in CLUSTAL [7,30]. It is part of the Genetics Computer Group (GCG) software package. It also uses pairwise alignments [34] to build a matrix, which is then used to build a dendrogram (guide tree). The guide tree however is not built using the Neighbor-Joining method [28] but using the clustering strategy called UPGMA [33], or Unweighted Pair-Group Method Using Arithmetic Averages. The multiple alignment is also completed by progressively aligning sequences following the guide tree. In the documentation it is mentioned that PileUp can align up to 500 sequences and the final alignment is restricted to 7000 characters by default but that these restrictions can be modified via parameters. The source code for this implementation cannot be found online at the present time. It can only be used as a command line on their servers or by using a web portal.

MAAFT [35], or Multiple Alignment Program for Amino Acid or Nucleotide Sequences, uses Fast Fourier Transform (FFT) to identify homologous sequences. The sequences again need to go through a pairwise comparison phase before building a guide tree. This program uses the UPGMA [33] method of building a guide tree like the PileUp program. The sequences are then aligned following the guide tree resulting in a final alignment like the other programs. But MAAFT uses an iterative refinement method to further improve the final alignment. This program also introduces a different scoring

system for multiple sequence alignments. The executable is available for Linux, UNIX, Windows and Mac. The source code however is not available to download.

DIALIGN [36,37] and DIALIGN 2 [38] are significantly different than the other programs presented above. Instead of comparing individual amino acids (or residues), it uses whole segments. Both pairwise and multiple alignments are executed using pairs of segments of equal lengths. This has the advantage of eliminating the need for gaps. This program is best suited to find local similarities in distant sequences. Since the multiple alignment is not based on a guide tree like most other algorithms, this program is not dependent on sequence order. DIALIGN first gathers all similar segments of two sequences in the set, these are called diagonals. Diagonals are given a weight relating to the probability of it occurring at random. A lower the probability will produce a higher weight. The multiple alignment is then constructed by collecting “consistent diagonals” in a greedy fashion. This program is accessible through a portal online, but the software itself is only sold as a commercial product.

2.1.2 Algorithm

As we mentioned in the previous section, we decided to use CLUSTAL W for our implementation. Therefore we will describe its algorithms in more details. Here is the basic CLUSTAL W algorithm; we will discuss each phase individually in the following sections.

Algorithm 1: Sequential CLUSTAL W

Input: A set S of n sequences. **Output:** A multiple alignment of S .

- (1) Pairwise alignment: Compute pairwise alignments for all sequences against all other sequences and store the result in a similarity matrix. The values in the sequence similarity matrix are converted to distance measures which reflect the evolutionary distance between each pair of sequences.

- (2) Guide-tree: Construct a guide-tree which defines the order in which pairs of sequences are aligned and combined with previous alignments using the sequence similarity matrix and a neighbor-joining algorithm.
- (3) Multiple alignment: Align progressively following the guide tree. Start by aligning most closely related pairs of sequences and at each step align two sequences or one to an existing sub-alignment.

— End of Algorithm —

2.1.3 Pairwise Alignment

In this phase, all n sequences are compared to each other using a global alignment algorithm based on dynamic programming, gap penalties (section 1.2.3) and scoring matrices (section 1.2.4). This amounts to $n(n-1)/2$ pairs or $O(n^2)$. For each pair, every residue is compared in both sequences in order to produce a pairwise score. That score is calculated as the number of identities in the best alignment divided by the number of residues compared (gap positions are excluded). It is initially calculated as a percent identity score and then converted to distances by dividing by 100 and subtracting from 1.0 to give the number of differences per site. After all pairs are aligned and a score is calculated, we build a distance matrix (Figure 2.2) using those score to be used in the guide tree building phase. We can notice that the distance matrix is symmetric and that the diagonal numbers are 0 since there is no distance between a sequence and itself.

	7							
1_human_Ia	0.000	0.035	0.037	0.048	0.212	0.335	0.679	
2_bovine_Ia	0.035	0.000	0.041	0.055	0.217	0.342	0.678	
3_rat_Ia	0.037	0.041	0.000	0.025	0.217	0.333	0.679	
4_mouse_Ib	0.048	0.055	0.025	0.000	0.217	0.335	0.661	
7_frog_I	0.212	0.217	0.217	0.217	0.000	0.311	0.577	
5_mouse_IIb	0.335	0.342	0.333	0.335	0.311	0.000	0.600	
6_camel	0.679	0.678	0.679	0.661	0.577	0.600	0.000	

Figure 2.2: Example of a distance matrix (from SYNAPSIN).

This phase runs in $O(n^2m^2)$ where n is the number of sequences and m is bounded by the length of the longest sequence. This is often the most time consuming part of the sequential algorithm.

2.1.4 Guide Tree

A guide tree is a 2-Tree used in the progressive alignment; it is sometimes called a dendrogram. A 2-Tree is a binary tree that has either two children nodes or none. It is built using the distance matrix created in the pairwise phase and the Neighbor-Joining method [28]. This produces an un-rooted tree with branch lengths proportional to estimated divergence along each branch. The root is placed by a "mid-point" method [29] at a position where the means of the branch lengths on either side of the root are equal. These trees are also used to derive a weight for each sequence [29]. The weights are dependent upon the distance from the root of the tree but sequences which have a common branch with other sequences share the weight derived from the shared branch.

We can see an example of a guide tree for 7 sequences taken from the SYNAPSIN data set in Figure 2.3. This phase has a runtime of $O(n^2)$.

Figure 2.3: Example of a guide tree (from SYNAPSIN).

2.1.5 Progressive Alignment

The basic Progressive Alignment algorithm consists of aligning larger and larger groups of sequences, following the branching of the guide tree. The alignments are done in the same fashion as the pairwise alignments. We start at the leaves of the guide tree and moves towards the root. For example in Figure 2.3 the alignment order would be:

1. 4_mouse_Ib with 3_rat_Ia
2. 2_bovine_Ia with 1_human_Ia
3. Result of alignment #1 with result of alignment #2
4. Results of alignment #3 with 7_frog_I
5. Result of alignment #4 with 5_mouse_IIb
6. Result of alignment #5 with 6_drosophila finishes the alignment

Following the example we can observe that we are progressively adding more and more sequences, hence the name. This completes the multiple alignment and the result of alignment #6 is written to a file. The Progressive Alignment phase runs in $O(nm)$ since every sequences is aligned once as opposed to $n(n-1)/2$ times for the pairwise alignment.

2.1.6 Scoring

There are two main types of scoring used in CLUSTAL W: pairwise scores and multiple alignment scores. When aligning two sequences, scoring matrices (e.g. PAM250 or BLOSUM62) are used in order to determine a similarity score for matches and mismatches for each corresponding pair of amino acids or nucleotides. A penalty is incurred when gaps are inserted, as well as smaller penalties for extending those gaps. The pairwise score between a pair of sequences is calculated as the sum of similarity scores for all aligned pairs of characters minus the gap penalties introduced in either sequences. These scores are necessary for building the distance matrix used by the guide tree.

The multiple alignment score is a sum-of-pair score or SP [3]. For an alignment of N sequences, each of M columns (length), we will denote the i -th column in the alignment by $A_{i1}, A_{i2}, \dots, A_{iN}$. We define $P_{ijk} = 1$ for every pair A_{ij} and A_{ik} which are aligned with each other and $P_{ijk} = 0$ otherwise. The score S_i for the i -th column is

$$S_i = \sum_{j=1, j \neq k}^N \sum_{k=1}^N P_{ijk}$$

and the SP for the alignment is

$$SP = \frac{\sum_{i=1}^M S_i}{\sum_{i=1}^{M_r} S_{ri}}.$$

Here M_r is the number of columns in the reference alignment and S_{ri} is the score S_i for the i -th column in the reference alignment. Other attempts have been made to improve this scoring method, such as Circular tours and Traveling Salesman Problem [4].

2.2 Previous Work

2.2.1 SGI

One of the first and probably most popular parallelization of CLUSTAL was implemented by the SGI ChemBio group [10] for shared memory SGI multicomputers, like the SGI Origin 3000. It is based on SGI OpenMP shared memory compilers and runs only on shared memory SGI machines. Since it is based on CLUSTAL W it uses the same three phases: pairwise alignment, guide tree building and progressive alignment. The pairwise alignment phase is parallelized by aligning different pairs of sequences on different processors. When all the pairs are aligned, they build a distance matrix just as the one found in CLUSTAL W. In the guide tree phase they use the neighbor-joining

method [28] like the original CLUSTAL W. The most time consuming part of this algorithm is finding the minimum value of the distance matrix in each step. They handle this by assigning rows of the matrix to processors and then find the minimum in parallel. In the final progressive alignment phase, CLUSTAL W calls a scoring function $score(i,j)$ with a loop. The loop itself is not parallelizable but their implementation is said to calculate every instance of i and j in parallel. Some simple SGI pseudo-code on how they implement this is presented but without more details.

They have reported a speedup of around 10.5 for 16 processors on a SGI Origin 3000, using as input subsets of 100 and 600 of the GPCR data set with an average of 390 amino acids (Figure 2.4). The major drawback of this implementation is that it is designed to only run on shared memory SGI machines, therefore not an option for distributed memory PC clusters. SGI does not offer the source code either, but only a binary to run on their SGI machines.

Figure 2.4: Relative Speedup using 100 and 600 GPCR protein sequences with average length of 390 amino acids [10].

2.2.2 ClustalW-MPI

ClustalW-MPI is a recent parallel CLUSTAL W implementation by Kuo-Bin Li (2003, [16]). It was implemented using the MPI communication library and will run on both distributed and shared memory computers. As the SGI implementation it follows the three phases of CLUSTAL W: pairwise alignment, guide tree building and progressive alignment.

The pairwise alignment is implemented using *fixed-size chunking* [39]. They allocate a fixed number of sequence pairs to each processor. They then build the distance matrix with the results gathered. The guide tree is also done using the neighbor-joining method [28]. They mention having a parallel implementation for searching for highly divergent sequences but offer no explanation on how this is used to speedup the tree building. The progressive alignment phase is implemented using a mixed fine and coarse-grained approach. The efficiency of this algorithm is dependent on the balance of the tree. They estimate that for a well balanced tree the speedup can be estimated by $N/\log N$ where N is the number of nodes in the guide tree. They mention using the recursive parallelism paradigm [40] and the linear space profile-profile alignment algorithm [41].

This implementation is designed to run on any workstation clusters with distributed memory as opposed to SGI's shared-memory implementation.

They claim their implementation has a speedup of about 14.5 using 16 processors (eight Dual PIII 800 MHz machines, with 100Mbit Ethernet) and linear pairwise speedup using a set of 500 sequences with an average of 1100 amino acids (Figure 2.5). This paper was published after our implementation was completed.

Figure 2.5: Speedup for ClustalW-MPI for a 500-sequence data set [16].

2.2.3 Other

Very few other implementations can be found anywhere at the present time. Some websites offer portals which usually run SGI's or another shared memory implementation. There is a downloadable SMP version of CLUSTAL W 1.82 by O. Duzlevski but it is currently unpublished at this time. It uses Posix threads and is designed to run on symmetric multiprocessor computers.

Chapter 3

Parallel Multiple Sequence Alignment

3.1 Description

In this section we will discuss the parallel multiple sequence alignment algorithms used in our implementation as well as other alternatives available. We will follow the same three phases as the original CLUSTAL W: Pairwise alignment, Guide Tree building and progressive alignment.

3.2 Algorithm

The following is a general algorithm for a parallel multiple sequence alignment implementation based on the progressive alignment and guide tree:

Algorithm 2: Parallel CLUSTAL W

Input: A set S of n sequences. **Output:** A multiple alignment of S .

- (1) Pairwise alignment: Distribute all pairs of sequences to be aligned among all available processors. Gather the results on the master processor and build the distance matrix.
- (2) Guide-tree: Build the guide tree as in the sequential version using the neighbor-joining algorithm and the distance matrix.

- (3) Multiple alignment: Decide if any work can be done in parallel and execute that work. The master processor can then finish the multiple alignment following the guide tree.

— End of Algorithm —

3.2.1 Parallel Pairwise Alignment

As previously explained, this phase compares all n sequences with each other. We then have $n(n-1)/2$ pairs of sequences of length bounded by m . We assume we have p processors. There are two possible options for parallelization for this phase.

1. Every pair of sequences consists of $2m$ residues to be compared. We can divide the total number of residues to be compared by the number of available processors, which is equal to $nm(n-1)/p$, and send that amount of data to each processor. This fine grained approach has the advantage to work well for small numbers of long sequences.
2. Distribute all pairs of sequences to be aligned among the processors. We then have $n(n-1)/2p$ sequences aligned by every processor. This is a more coarse grained approach to the parallelization.

While both options are viable, only one is suited for our implementation. Option 1 would only work if a local algorithm is used for the pairwise alignments. However in the original CLUSTAL W, a global alignment is used for pairwise alignment. We therefore must use Option 2 which has the drawback of having smaller speedups for small sets of long sequences on small numbers of processors. Once every processor is done with its pairwise alignments, the results are gathered on the master processor so it can build the distance matrix.

Algorithm 3: Parallel Pairwise Alignment

Input: A set S of n sequences. **Output:** Distance matrix DM

- (1) Since we have n sequences, we have $n(n-1)/2$ pairs to send to p processors.
Choose the pairs to be sent to each processor and send them in one step.

- (2) Every processor aligns its pairs and once finished returns the scores to the master processor.
- (3) The master receives the pairwise score from all the processors, converts them to distances and inserts it into a distance matrix DM .

— End of Algorithm —

3.2.2 Parallel Guide Tree

The guide tree phase is a very small fraction of the total runtime, around 1-2%. Therefore using the standard algorithm from the sequential CLUSTAL W is acceptable. The most time-consuming part of this phase is to find the smallest value in the matrix to build the next branch. To speed up this search we use a technique similar than used in SGI's implementation [10]. The rows of the matrix are first divided between all the processors, then each processor finds the minimum values of the rows it has. The minimum values are gathered on the master processor and we then deduce the minimum value. This is effective for large numbers of long sequences, which leads to large matrices. In the following algorithm we assume we have p processors and n sequences.

Algorithm 4: Parallel Guide Tree

Input: Distance matrix DM of size n by n . **Output:** Guide tree GT

- (1) Send n/p rows to every processor.
- (2) Each processor sends the minimum of its rows to the master processor.
- (3) The master processor finds the minimum of the p values it received.
- (4) Using the smallest value, start constructing the guide tree using the Neighbor-Joining method [28] by Saitou and Nei.
- (5) Go back to (2) to find the next minimum value until we used all values of DM .
- (6) Output tree to file.

— End of Algorithm —

3.2.3 Parallel Progressive Alignment

The parallelization of the progressive alignment is a hard problem. Due to the nature of the alignment, this phase is better suited to run in sequential. The only option we have is to assign sequences that do not rely on a previous alignment to different processors. The only sequences that can be aligned in parallel at the same time are those which do not require a previous alignment. The parallel algorithm will need to work in steps, each step aligning a level of the tree. We can deduce that the level of parallelism will depend entirely on the balance of the guide tree.

For example take Figure 3.1a, a perfectly balanced tree. We can see here that pairs of leaves of the tree can easily be aligned on different processors. Going up in the tree to the next level we can also align the intermediate nodes on separate processors. Then the final root alignment can be done on one processor, preferably the master. Instead of 7 separate alignments we can accomplish the work in three steps. In this case we would get a relatively good speedup for small numbers of processors.

Conversely, let's take the worst case scenario of an unbalanced tree (Figure 3.1b). In this case we can only align the intermediate nodes after a previous alignment has been finished. In this extreme case there is no parallelization possible.

Reality, however, lies somewhere in between. In order to have a completely unbalanced tree you would need to have a set of sequences very different from one another, which is very unlikely.

In order to evaluate if we can do alignments in parallel, we first need to analyze the guide tree. An alignment can only be done in parallel on a different processor if it does not rely on the completion of another alignment. Since a guide tree is a 2-tree, each node will either have 2 leaves or none. This is due to the fact that a leaf has to be a sequence, and an internal node is an alignment between two sequences.

Figure 3.1: Balanced (a) and unbalanced (b) guide tree.

Lemma 1. *For n sequences, we will have $n-1$ alignments.*

Proof. We can prove this by induction. Since we know that internal nodes are alignments, it is sufficient to prove that a guide tree has $n-1$ internal nodes for n leaves. Lemma 1 is true for $n=1,2$. We assume that it is true for n . For $n+1$, we will have n internal nodes. The deepest internal node has two leaves and by removing those leaves the internal node will become a leaf. We removed 2 leaves, an internal node and in turn added another leaf. We now have n leaves and $n-1$ internal nodes.

If we have p processors and n sequences, the optimal solution would be to send $(n-1)/p$ alignments to each processor. However, even in the best case of a balanced tree, the most alignments we can execute independently at the same time is $(n/2)/p$ which represent the number of pairs of leaves. We have to wait for the leaves to be aligned in order to align the internal nodes.

A better approach is to isolate sub-trees that can be aligned independently, and to send them to be computed on the available p processors. The problem can now be expressed as dividing the tree into p independent sub-trees, each containing roughly the same number of alignments. Furthermore, our algorithm will need to work in steps; each step combining computed sub-trees of the previous step. We therefore should try to keep all the necessary information on one processor to minimize communications. Let's take

for example Figure 3.2; a guide tree containing 6 sequences and therefore 5 alignments. The sequences are lettered from A to F and the alignments are numbered 1 to 5. Suppose we have $p=2$ processors at our disposal, P_0 and P_1 . Since we use a master/slave approach, P_0 will serve as both a master and a slave in this example.

Figure 3.2: Studying the guide tree.

First, we can observe that only alignment 1 and 2 can be done independently. Following our analysis, we would like to send $(n-1)/p$ alignments to each processor. In this case we want to send $5/2$ alignments to each processor, or 2 alignments to one and 3 to the other. Looking at the dependencies in our example guide tree we can deduce that this is not possible. Also notice that the root alignment will always have to be done last, therefore should be done by the master processor to reduce communications. The best we can do is send alignment 1 to processor P_0 and "2", "3" and "4" to P_1 . Once P_1 is finished with its computations, it will send its results to P_0 . The master processor now has everything it needs to complete alignment "5". Here is an outline of the parallel progressive alignment algorithm, including the algorithm to analyze the guide tree:

Algorithm 5: Parallel Progressive Alignment

Input: A guide tree GT with n nodes. **Output:** Final alignment. **Node:** Has a *left* and *right* pointer, a pointer to a sequence, *Count* which has the number of alignments found in the sub-tree.

- (1) Start at the root of the tree.

- (2) Recursively find the number of leaves in each sub-tree and store it in *Count*. We now know the number of alignments in each sub-tree by using *Count-1*. This step takes $O(2n-1)$ since we visit every node once.
 - (3) Go back to the root of the tree.
 - (4) Look at *right Count* and *left Count*.
 - a. If *left Count* is around $(Count-1)/p$, remove that sub-tree and assign it to an available processor. If *left Count* is larger than $(Count-1)/p$, go to (4).
 - b. If *right Count* is around $(Count-1)/p$, remove that sub-tree and assign it to an available processor. If *right Count* is larger than $(Count-1)/p$, go to (4).
 - (5) Repeat (4) until we assigned a sub-tree to every processor.
 - (6) Once every processor received and aligned its sub-tree, the results are gathered on the master we insert the new aligned sequence in the node replacing the removed sub-trees. Go to (1) with the new tree.
 - (7) Repeat (1) to (6) until we reach the final alignment.
 - (8) Output final alignment to file.
- End of Algorithm —

There are certainly caveats to this algorithm. A processor might receive more work than another or no work at all. This is especially true if we have a larger number of processors than the number of sequences. The level of parallelism also relies on the balance of the guide tree.

We have shown how the algorithm should work in a simple guide tree, but now we will take a look at a real guide tree. Figure 3.3 is the guide tree for the SYNAPSIN dataset containing 28 sequences. Following Lemma 1 we know we will have 27 alignments. Suppose we have 8 processors ($p=8$), we then have to try and send $27/8$ or around 3 to 4 alignments to each processors. First, the algorithm will calculate the number of alignments found in each sub-tree in (2). Then we try to identify 8 independent sub-trees with around 3 to 4 alignments. The algorithm is flexible with the actual number of alignments it will assign since in some cases it won't be able to assign the ideal number. We can set the variance we want it to accept, for example ± 2 . We can see that

the top sub-tree composed of sequences 1,2,6,7 fits our criteria, it has 3 alignments. This sub-tree can be sent to an available processor.

Figure 3.3: SYNAPSIN guide tree.

We repeat our search until we assigned 8 sub-trees to our 8 processors. Every time we assign a sub-tree to a processor, we cut it out of the guide tree and replace it with a node with a NULL sequence. This will identify the node as being dependent on a previous alignment and can't be used until we received the result. Once we removed all our 8 sub-trees from Figure 3.4a, we end up with Figure 3.4b. When each processor finished aligning its sub-tree it returns the result sequence to the master which places in the node where the sub-tree was removed. We then have a different guide tree with less sequences. The process starts over again, assigning sub-trees of the new guide tree to as many processors as possible. In this case we can only send two sub-trees in Figure 3.4b, resulting in Figure 3.4c. We continue in steps until we reach Figure 3.4f where we only have one alignment left to complete.

Figure 3.4: Parallel Progressive Alignment using the SYNAPSIN guide tree.

3.3 Correctness

The correctness of Algorithm pCLUSTAL follows from the correctness of the sequential CLUSTAL W. Since we did not modify the original algorithms, only execute the work in parallel, the alignments and scores are identical to the original CLUSTAL W.

3.4 Performance

We assume that n is the number of input sequences, each of which is bounded in length by m , and that p is the number of processors. An h -relation operation is a communication step in which each processor sends at most h messages and receives at most h messages.

Phase One: Step (1.0) requires $O(nm)$ time to read the sequence file and $O(n^2)$ time to map the sequence pairs to the processors. In Step (1.1), specific subsets of sequences are sent to specific processors requiring $O(1)$ h -relation operations (*MPI_AllToAllv*). In Step (1.2), each processor performs a sequential pairwise alignment on its sequence subsets, requiring $O(n^2m^2/p)$ time. Each processor sends its scores to P0 in Step (1.3), using $O(1)$ h -relation operations.

Phase Two: Building the guide tree in Step (2.0) requires $O(n^2)$ time since it is based on the neighbor-joining algorithm [28]. The rows of the matrix are distributed among the processors requiring $O(1)$ h -relation operations.

Phase Three: The analysis of TS in Step (3.0) requires $O(n)$ time. The multiple alignment operation in Step (3.1) requires $O(nm^2)$ time. The dispersal and gathering of independent candidate pairs in Steps (3.0) and (3.1) require $O(1)$ h -relation operations.

Algorithm Complexity: From the previous analysis, the total time for Algorithm *pCLUSTAL* is $O(n^2m^2/p)$ local computation and $O(1)$ h -relation operations for communication. This shows the effectiveness, in theory, of Algorithm *pCLUSTAL*. The local computation shows linear speedup, and the communication overhead consists only of a small, fixed, number of data permutations. Our performance analysis in Section 6 below shows that this property of Algorithm *pCLUSTAL* does indeed translate into very good overall performance of our *pCLUSTAL* code.

3.5 Example

Take as an example the alignment of 5 sequences: A, B, C, D and E (Figure 3.5). The master processor first inputs the sequences to be aligned. We then need to figure how many pairs of sequences need to be sent to each processor. Since we have 5 sequences, we will have $n(n-1)/2$ pairs to be aligned on p processors, or in this case 10 pairs on 4 processors. The load balancing will assign 3 pairs to processor P₁ and P₂, and 2 pairs on processors P₃ and P₄.

The data is sent to all processors in one step; then Phase 1 can begin. Every processor aligns their assigned pairs following the standard CLUSTAL pairwise algorithm. Once all the work is completed, the pairwise scores are gathered on P_0 to build the distance matrix used in the next step. In Phase 2, we distribute the rows of the distance matrix on the 4 processors and find the global minimum in parallel in each step of the Neighbor-Joining method [28].

Figure 3.5: Phases 1 and 2 of pCLUSTAL.

Now that we have the guide tree, Phase 3 can begin its progressive alignment (Figure 3.6). The guide tree is first analyzed in order to decide what can be done in parallel. Since the alignment is progressive, only independent sub-trees can be aligned in parallel. Once the parallel work is done, we gather the results on P_0 and the remainder of the alignments is done following the guide tree. The multiple alignment is now finished and written to a file.

Figure 3.6: Phase 3 of pCLUSTAL.

Chapter 4

Implementation

4.1 pCLUSTAL

The *pCLUSTAL* implementation was built using C and the MPI communication library and is based on the popular alignment software CLUSTAL W 1.82 [12]. For implementation and development we used Redhat 9.0, KDevelop 2.0 for editing and MPICH 1.2.3/1.2.4 for debugging and preliminary testing.

The implementation phase took around 8 months from start to testing. Every version had to be tested and the results analyzed by a biologist to be sure the results were correct. We had an important design requirement at the start of our implementation: the final multiple alignment and score must be identical to the ones generated by CLUSTAL W 1.82 for the same data set. The reason for this requirement is simple: since CLUSTAL is a very popular alignment tool biologists are already familiar with results using their own data sets. By modifying the result of the final alignment, a biologist would need to re-align some previous data sets to give him a reference point with the old alignment. Therefore the alignment and score must not be changed in anyway and have to be identical to the original CLUSTAL W.

After obtaining the original CLUSTAL W 1.82 source code, some cleaning up of the code was needed before any work could begin. I also used that step to familiarize myself with the code. I then began to make some tweaks to improve performance. One major improvement was implemented to the guide tree building algorithm called Neighbor-Joining Tree [30]. Either by mistake or by negligence, a part of the algorithm

was poorly coded and did some useless computations. By changing a few lines of the code, we reduced the runtime of the algorithm by a considerable margin.

As shown in Table 4.1, the new guide tree (GT) modifications are up to 75 times faster than the old version with our data sets. This flaw has recently been corrected in CLUSTAL W version 1.83.

Data Set	Number of Sequences	Old GT (seconds)	New GT (seconds)	Improvement (%)
glucagon	478	157.55	8.34	1889%
kinase	647	677.56	13.74	4931%
relaxin	426	95.22	3.81	2499%
trypsin	878	2096.34	28.07	7468%

Table 4.1: Old and new guide tree algorithm runtimes.

The first design choice we had to make when building the parallel version was to decide whether we could use the original CLUSTAL W program and write a program on top of it. This way, we would only need to divide the input sequence set over the available processors and run the original CLUSTAL W on each node. This method would create a local multiple alignment on each node which would then need to be aligned into a global multiple alignment. However, since the guide tree algorithm uses the entire input set of sequences to build a more accurate tree, the resulting final alignment would differ from the one built using the original CLUSTAL W. Since this was against our requirement, we instead decided to add our modifications inside the CLUSTAL W code.

We then needed to find an architectural model on which to base the implementation. Since we need the complete distance matrix to build the guide tree, and we need to follow the complete guide tree for the progressive alignment, the logical choice was the master/slave approach. If all three phases of the alignment were completely independent from each other, we might have used a pipelined approach. But in this case we need to complete one phase before moving on to the next. Also, since the master processor is mainly needed at the start and end of each phase, it will function as a slave node along with the other processors.

The next step was to parallelize the first phase of the alignment: the pairwise alignment. The input of sequences is included in this phase, even if it is not parallelizable. This phase is considered to be embarrassingly parallel. Since this step aligns all the possible pairs of n sequences, we have $O(n^2)$ pairs to be aligned. The best and simplest way to achieve parallelization would be to send $O(n^2/p)$ pairs to be aligned to each of the p available processors. However, this supposes that $n^2 \gg p$; otherwise increasing p will result in poor relative speedups. This also makes the runtime sensitive to the sequences' length. Having a small number of long sequences will often have a worse speedup than a large number of small sequences for large number of processors.

Once the master processor decides how many pairs will go to each processor, we execute a *MPI_AllToAllv()* call to distribute the data. After all processors have the data they need, they start doing their pairwise alignments using the CLUSTAL W pairwise algorithm and then storing the corresponding scores in a array of size $n(n-1)/2p$. Once all processors are done with the computation, we do another MPI call to gather all the calculated scores on the master processor. The master processor then builds the distance matrix as mentioned in section 3.2.1.

The second phase uses the distance matrix to build the guide tree which will be necessary in the final progressive alignment. Even in the sequential version, this phase takes a very small percentage of the whole alignment (typically around 1-2%), making it very hard to improve in parallel, especially after the modification mentioned earlier. Since the most time consuming part is looking through the matrix for the smallest element at each step of the tree building, we distribute the rows of the matrix to the processors and find the minimum in parallel (Section 3.2.2).

The third and final phase, called progressive alignment, was the trickiest part to implement in parallel. As the name suggests, this alignment is inherently sequential since it is based on the guide tree. In the sequential version, the progressive algorithm starts by aligning the most similar leaf pairs and goes up the tree eventually reaching the root. When the algorithm reaches the root, the result is the final multiple alignment. Changing the order of alignment in this step will result in a different final alignment, as well as a different final score. The first version of the parallel progressive alignment we

implemented aligned close pairs in the tree, even when not sharing a parent. This yielded different results as shown in Table 4.2.

Data sets	Number of sequences	Original Score	Parallel Score	Difference (%)
CALMODULIN	25	223999	224804	0.36%
KINASE	25	-74685	-76883	-2.94%
SYNAPTOTAGMIN	25	219455	223082	1.65%
VAMPS	22	94273	93451	-0.87%
SYNAPSIN	25	547305	557328	1.83%
SYNUCLEIN	49	543724	545581	0.34%

Table 4.2: Differences between CLUSTAL W and the previous pCLUSTAL version.

We see that the simplifications to the progressive alignment improved the scores in some cases and decreased them in other data sets. Note however that these are very small data sets, between 22 and 49 sequences. These data sets were chosen to allow the biologist to inspect the output multiple alignments by eye to detect any problems. Since the scoring is simply adding up the match score and deducting the gaps, a good score does not always imply a good alignment. To parallelize this step we first need to analyze the guide tree created in the previous phase. The progressive alignment studies the guide tree using the algorithm described in Section 3.2.3. Once it distributed the sub-trees to the processors and each processor completed its work, the master processor completes the final alignment which is then written to a file.

The following is an algorithm describing our implementation:

Algorithm 6: pCLUSTAL

Input: Set of sequences, $\{S\}$. **Output:** guide tree, TS; alignment, AS.

- (1.0) Processor P_0 reads the sequence set S . It then determines the mapping of sequence-pairs to processors, based on the number of processors, p , and the number of sequences, $n = |S|$.
- (1.1) Processor P_0 sends the pre-computed subset of sequences to the corresponding processors, P_i , $1 \leq i \leq p - 1$.

- (1.2) Each processor P_i , $0 \leq i \leq p - 1$, performs a pairwise alignment on its assigned sequences using the CLUSTAL W sequential algorithm.
 - (1.3) Each processor P_i , $1 \leq i \leq p - 1$, sends its relative pairwise alignment scores to P_0 .
 - (2.0) Processor P_0 builds the distance matrix, DM.
 - (2.1) Processor P_0 , sends the rows of DM to each processor P_i , $1 \leq i \leq p - 1$, and then starts building the guide tree using the Neighbor-Joining method, finding the minimum values in parallel.
 - (2.2) The guide tree is completed, it is written to a file.
 - (3.0) Processor P_0 analyzes TS to identify sub-trees that can be evaluated independently in Step (3.1). Sequences that can be evaluated independently are assigned to processors, P_i , $1 \leq i \leq j - 1$.
 - (3.1) Processor P_0 gathers resulting pairs evaluated in Step (3.0) and, using TS, sequentially completes the multiple alignment of the sequences, AS.
 - (3.2) The final alignment is completed, output the result to a file.
- End of Algorithm —

4.2 Portal

The goal of the implementation was to help biologists spend less time doing multiple sequences alignments so they could concentrate on more important problems. Having created a parallel version of CLUSTAL W will help achieve that goal, but of course the biologists need to be able to access that tool. Since not every biologist owns or has access to a shared or distributed PC cluster, we built an accessible web-based portal. Using the portal anyone can upload their sequence set and the alignment will be run on the parallel PC cluster at Carleton University.

Other portals currently exist using SGI's parallel CLUSTAL W implementation, but most of them have limited capabilities. For example, other portals limit the number of

sequences that can be aligned to 100 or 200 sequences. Our portal offers a high sequence limit, around 900 sequences. And unlike some portals running on only 8 or 10 processors, our portal runs on 32 processors. We chose that number of processors according to our experimental results. Using more processors will benefit the pairwise alignment phase but using more than 4 processors will not result in significant improvements of the runtime of the guide tree and progressive alignment phases. Therefore we decided 32 would be an acceptable number of processors.

Also, our CLUSTAL XP package includes a new feature: Parallel FPT Minimum Vertex Cover [2]. This feature is used to remove sequences that align badly with the rest of the data set; improving the overall alignment. By setting a threshold, users can decide what pairwise scores define a bad alignment. A graph is built using the bad pairwise scores between two sequences (nodes) as an edge. Then we calculate the Minimum Vertex Cover and remove it, which removes the bad sequences. This feature can also use a similarity threshold, which will remove sequences that have a certain score or higher. This will remove any highly similar or duplicate sequences, which tend to skew the alignment in their favor.

The first part of the Clustal XP portal (Figure 4.1) includes the names of all those involved with this project. This is also where users can paste in their sequences or upload a text file containing their sequences to be aligned (FASTA format).

Next we can see some basic CLUSTAL W parameters (Figure 4.2), such as the use of *Quick and Fast Alignment* and the format of the output. Selecting the *Quick and Fast Alignment* will compute a quicker alignment but of worse quality, it is not recommended. It is also required to select if the sequences entered are protein or nucleotide. We then have the option to leave the rest of the parameters at their default state and start the alignment by clicking the “Run ClustalXP” button.

In Figure 4.3 we have the Parallel FPT Minimum Vertex Cover parameters. Here users can chose to turn the vertex cover on or off; turning the cover off will reduce the runtime of the alignment but having it on might produce a better alignment. The *Conflict Graph: Edge Threshold* value is used to build the conflict graph before finding the cover

to remove the badly aligning sequences. Any pairwise score value equal or under this number will result in an edge in the graph.

Clustal XP

Jim Cheetham, Frank Dehne, Sylvain Pitre, Peter Taillon
Carleton University

Andrew Rau-Chaplin
Dalhousie University

Faisal Abu-Khzam, Michael A. Langston, Pushkarak Shanbhag
University of Tennessee

Please paste the sequences (FASTA format) in the box or use the *Upload file* option below.

Sequence file:

Figure 4.1: Clustal XP Portal sequence input box and file upload.

Please choose a nucleotide or protein sequence to align ☐ protein ☐ nucleotide

☐ Quick and Fast Alignment

Output in format.

Output in ☒ the input order. ☐ the aligned order.

Include Sequence Numbers? Yes ☒ No ☐

Figure 4.2: Clustal XP Portal simple settings

Similarly, the *Similarity Graph: Edge Threshold* value is used to build the similarity graph to remove very close or identical sequences. We also have the option of choosing which kernelization and tree search algorithms are used. Initially we were using the algorithms discussed in [2], but in the past few months collaboration with the University of Tennessee (Faisal Abu-Khzam, Michael A. Langston, Pushkarak Shanbhag) yielded a new version of both algorithms. The University of Tennessee kernelization and the Carleton University tree search are selected by default, each being the faster than their counterparts. The user also has the choice to skip the kernelization phase and go directly to the tree search.

Vertex Cover Parameters

Vertex Cover:	<input type="text" value="Off"/>	Kernelization:	<input type="text" value="U-Tennessee"/>
Conflict Graph: Edge Threshold	<input type="text" value="20"/>	Tree Search:	<input type="text" value="Carleton-U"/>
Similarity Graph: Edge Threshold	<input type="text" value="90"/>	Tree Search Only	☐

Figure 4.3: Clustal XP Portal Vertex Cover Parameters.

Finally we have the pairwise alignment parameters and multiple alignment parameters. The most important parameters for both the pairwise and multiple alignments are the gap opening and extension penalties (see Section 1.2).

Experienced users may want to change those settings, but usually the default will suffice.

There is also the possibility for the user to input other parameters not available on the webpage. This is done through the additional options text box (see Appendix B for complete listing). When the sequences are pasted into the text box or a sequence file is loaded and all the parameters have been chosen, the user can run the alignment by clicking the *Run Clustal XP* button. After the alignment is started, a webpage shows a description of the work being done as well as links to the relevant result files (Figure 4.5).

<u>Pairwise Alignment Parameters:</u>		<u>Multiple Alignment Parameters:</u>	
<u>For fast and approximate searches:</u>			
K-tuple(word) size	<input type="text" value="1"/>	Gap Open Penalty	<input type="text" value="10"/>
Window size	<input type="text" value="5"/>	Gap Extension Penalty	<input type="text" value="0.05"/>
Gap Penalty	<input type="text" value="3"/>	Weight Transition	☐ Yes: Value <input type="text" value="0.5"/> ☒ No
Number of Top Diagonals	<input type="text" value="5"/>	Hydrophilic Residues for Proteins	<input type="text" value="GPSNDQERK"/>
Scoring Method	<input type="text" value="percent"/> ▼	Hydrophilic Gaps	☒ Yes ☐ No
<u>For slow and accurate searches:</u>		Weight Matrix	<input type="text" value="BLOSUM"/> ▼
Gap Open Penalty	<input type="text" value="10.0"/>		
Gap Extension Penalty	<input type="text" value="0.1"/>		
Weight Matrix	<input type="text" value="BLOSUM"/> ▼		
Type <u>additional options</u> (-options for help)			
<input type="text"/>			
<input type="button" value="Run ClustalXP"/>		<input type="button" value="Reset"/>	

Figure 4.4: Clustal XP Portal Pairwise and multiple alignment settings.

Clustal XP currently works as demonstrated in Figure 4.6. First, a user can access the Clustal XP website and can then paste or upload some sequence alignments. After choosing the proper parameters, the user then presses the Run Clustal XP button. The sequences are sent and saved on the web server (A). The web server then needs to send those sequences to the PC Cluster (B) to start the alignment (C). Once the alignment is done the cluster returns the result files to the web server (D). Finally the web server can post links for the result files to the user (E). The user can then download the files which are needed. Please note that the web server is not connected with a direct link to the PC cluster but rather through an internal network.

Web Clustal Results Page

These are the results of your clustalw alignment.

Results:

sequences.txt (Input file)
sequences.dnd (Tree output file)
sequences.aln (Alignment output file)
sequences.graph (Conflict Graph file)
sequences.mvc (Minimum Vertex Cover file)

```
Running ClustalXP...
Sequence Input...
Start of Parallel Pairwise Alignments...
Finished Parallel Pairwise Alignments...
Starting to build guide tree...
Guide tree file created: [./sequences.dnd]
Start of Multiple Alignment
There are 91 groups
Aligning...
Alignment Score -79201
CLUSTAL-Alignment file created [./sequences.aln]
```

```
Timing statistics:
- Pairwise Alignment time: 0.400064 seconds
- Guide Tree creation time: 0.025665 seconds
- Multiple Alignment time: 2.224397 seconds
Total running time: 2.650252 seconds
```

```
Building conflict graph...
Guessing K... 68
Kernelization phase determined kPrime = 29.
Starting Vertex Cover...
Solution found...
Total Running Time (Wall Clock): 0.028406 seconds
Done!
```

Figure 4.5: Clustal XP Portal example run.

Figure 4.6: Information flow of the Clustal XP Portal.

Chapter 5

Experimental Performance Analysis

5.1 Methodology

For our experiments, we used a 64 node PC cluster with dual 1.8 GHz Intel Xeon processors, 1 GB RAM per node and 60 GB of disk storage per node. Every node is running Linux Redhat 7.2 with gcc 2.95.3 and MPI/LAM 6.5.6. The nodes are interconnected via a Cisco 6509 gigabit Ethernet switch. All parallel times are measured as the wall clock time between the start of the first process and the termination of the last process. We will refer to this as parallel wall clock time. All times include the time taken to read the input sequences from a file and write the solution alignment into a file. Our experiments measured the following:

- 1 – Parallel wall clock times as a function of the number of processors for all three phases of the algorithm as well as the total run time.
- 2 – Relative total speedup (ratio of total parallel wall clock on p processors) as a function of the number of processors.
- 3 – Relative speedups for all three phases of the algorithm (Pairwise alignment, Guide tree creation and Progressive alignment).

Every experiment was run a total of 20 times and the average was taken to build the graphs. We also calculated the standard deviation of the 20 runs which is displayed in the result graphs (Figures 5.1 to 5.16). Since this cluster is shared between multiple users, there is some variance between run times due to the load of the cluster.

5.2 Data Sets

While it is also possible to do multiple sequences alignments on DNA sequences we will use protein sequences exclusively. As test data, we obtained protein sequence data sets from the National Center for Biotechnology Information (NCBI). We used the following sequence sets listed in Table 5.1. Please note that the GPCR data set is only a subset of 600 sequences taken from the complete GPCR data set.

Name	Number of Sequences	Average number of residues
GLUCAGON	478	161
GPCR	600	364
KINASE	647	586
RELAXIN	426	94
SYNAPSIN	28	605
SYNAPTOTAGMIN	66	432
TRYPSIN	878	409

Table 5.1: Data sets used in testing.

5.3 Experimental Results

Figures 5.1 to 5.7 show the parallel wall clock times as a function of the number of processors for GLUCAGON, GPCR, RELAXIN, SYNAPSIN, SYNAPTOTAGMIN, KINASE, and TRYPSIN, respectively. The figures show the parallel wall clock times for the three phases of the algorithm (pairwise alignment, guide tree construction, and multiple alignment) as well as the total measured parallel wall clock time.

Figures 5.8 to 5.14 show the measured relative speedup (ratio of parallel wall clock time on one processor and parallel wall clock time on p processors), as a function

of the number of processors, for GLUCAGON, GPCR, RELAXIN, SYNAPSIN, SYNAPTOTAGMIN, KINASE, and TRYPSIN, respectively.

Figure 5.15 shows the pairwise alignment speedup for all data sets, as shown in Figures 5.1 to 5.7. And finally Figure 5.16 show the total relative speedup of all data sets also shown in Figures 5.1 to 5.7.

Figure 5.1: Parallel Wall Clock times measured for GLUCAGON sequence set.

Figure 5.2: Parallel Wall Clock times measured for GPCR sequence set.

Figure 5.3: Parallel Wall Clock times measured for KINASE sequence set.

Figure 5.4: Parallel Wall Clock times measured for RELAXIN sequence set.

Figure 5.5: Parallel Wall Clock times measured for SYNAPSIN sequence set.

Figure 5.6: Parallel Wall Clock times measured for SYNAPTOTAGMIN sequence set.

Figure 5.7: Parallel Wall Clock times measured for TRYPSIN sequence set.

Figure 5.8: Relative Speedups for GLUCAGON sequence set.

Figure 5.9: Relative Speedups for GPCR sequence set.

Figure 5.10: Relative Speedups for KINASE sequence set.

Figure 5.11: Relative Speedups for RELAXIN sequence set.

Figure 5.12: Relative Speedups for SYNAPSIN sequence set.

Figure 5.13: Relative Speedups for SYNAPTOTAGMIN sequence set.

Figure 5.14: Relative Speedups for TRYPSIN sequence set.

Figure 5.15: Relative Speedup for the Pairwise phase for all sequence sets.

Figure 5.16: Relative Total Speedup for all sequence sets.

5.4 Analysis

We will start by analyzing the performance and speedups for every dataset. Then we will look at the overall performance of the pCLUSTAL implementation.

In order to quantify the level of unbalance in the guide trees, we will be using a balance coefficient B defined as follows:

$$B = \left\lceil \left(\frac{H}{\log(2n-1)} \right)^2 \right\rceil$$

Where H is the height of the tree and n is the total number of sequences. Since we have n leaves, we will have $2n-1$ total nodes.

This coefficient will give us an idea of the imbalance of the tree by comparing the height of the guide tree H with the optimal height of a tree with n leaves ($2n-1$ nodes). A higher coefficient will mean a greater degree of imbalance. We take the square of that ratio to amplify the results and give us a clearer difference between guide trees. Otherwise, when rounding numbers, both an imbalanced and balanced tree can have very close values. We then take the ceiling of that equation.

Name	Number of Sequences (n)	$\log(2n-1)$	H	B
GLUCAGON	478	9	35	13
GPCR	600	10	28	8
KINASE	647	10	24	6
RELAXIN	426	9	42	19
SYNAPSIN	28	5	11	4
SYNAPTOTAGMIN	66	7	16	6
TRYPSIN	878	10	35	11

Table 5.2: Tree heights and balance coefficients (H, B).

The pairwise phase is expected to perform well in parallel, depending on the number and length of the sequences. The guide tree on the other hand is not expected to have good speedups, since it already runs quickly in sequential in most cases. The performance of the progressive alignment will depend on our balance coefficient B , but will probably not have good speedups even in the best cases.

5.4.1 GLUCAGON

In Figure 5.1 we can see that the pairwise phase makes good use of the available processors. The guide tree benefits from using 2 processors as opposed to one, but any more processors causes almost no decrease in runtime. The progressive alignment makes use of up to 8 processors but then using more processors does not yield better results. The

overall runtime has a nice curve up to 32 processors, using 64 processors has little improvement.

In Figure 5.8 we can see the speedups for each of the three phases. We notice that the pairwise speedup is good up to 32 processors, but drops off a little at 64 processors. Both the guide tree and progressive alignment (MSA) speedups are almost non existent. If we look in Table 5.2 we see that this data set has a rather unbalanced guide tree with $B = 13$ which explains the worse than average speedup. The overall speedup is good due in part to the progressive alignment phase. We can observe that for 16 processors the speedup is around 7.

5.4.2 GPCR

In Figure 5.2 we observe that again the pairwise makes good use of the available processors up to 32 processors. The guide tree phase is over very quickly, therefore it does not benefit from extra processors. The progressive alignment is also quick, but still can benefit from up to 4 processors. Overall the total runtime curve is quite good up to 32 processors.

In Figure 5.9 we notice that the pairwise speedups are very good up to 64 processors, being almost linear for 32 processors. The guide tree speedups are about the same as the other data sets, not very good. The progressive alignment speedups are slightly higher than most data sets. The guide tree has a coefficient $B = 8$ which is a little better than average for our data sets. The overall speedup is the best of all our data sets, being around 11 for 16 processors and 17 for 32 processors.

5.4.3 KINASE

In Figure 5.3, the pairwise runtime get a decent reduction with up to 16 processors, a little less of an improvement for 32 and 64 processors. The guide tree runtime is very small compared to the rest of the alignment, it does not benefit much from added processors. The progressive alignment on the other hand gets reduced

runtimes for up to 8 processors. The overall runtime gets improved for up to 32 processors like most data sets.

In Figure 5.10 we see the speedup curve for the pairwise alignment drops off a little at 8 processors but then stays constant up to 64 processors. The guide tree does not have much of a speedup after 2 processors. The progressive alignment has a slight speedup for up to 8 processors, but adding more processors does not help. Looking at the balance of the tree we find $B = 6$ which is a relatively balanced. The overall speedup is better than average at 9 for 16 processors and around 16 for 32 processors.

5.4.4 RELAXIN

In Figure 5.4 we see a decent curve for the pairwise runtimes for up to 16 processors, beyond that the curve is almost flat. The guide tree gets a very small improvement for 2 processors but then actually gets a little worse for 4. If you notice the scale of seconds however you will notice that this is nothing but a small fluctuation. The same goes for the progressive alignment, it does not have a significant reduction in time for any number of processors. Overall the total runtime curve is a little below average.

In Figure 5.11 we notice the pairwise speedup is near-linear for up to 32 processors but then dips a little for 64. Looking closely at the guide tree and pairwise speedup reveals a flat line for any number of processors. The guide tree in this data set has a $B = 19$ which is the worst of our data sets and explains the non-existent speedup of the progressive alignment. The overall speedup is below average with around 6.5 for 16 processors and 8 for 32 processors.

5.4.5 SYNAPSIN

In Figure 5.5 we see the pairwise runtimes do not get much improvement for over 8 processors. This was to be expected for such a small data set of 28 sequences. The guide tree does not improve at all for any number of processors, but its runtime is already very small in sequential. Here we see the progressive alignment phase gets a nice

improvement for up to 8 processors. Overall the runtime does not improve much for over 8 processors.

In Figure 5.12 we observe that the speedup for the pairwise alignment is not very good for over 8 processors. The guide tree has an almost flat line for any number of processors. The progressive alignment gets a little speedup for up to 8 processors. The guide tree has a $B = 4$ which is very good, but considering the number of sequences, not surprising. The overall speedup is below average at around 5 for 16 processors and 6 for 32 processors.

5.4.6 SYNAPTOTAGMIN

In Figure 5.6 we can see the pairwise alignment runtime curve looks good up to 16 processors. The guide tree phase is very quick since this data set only has 66 sequences. You can see the runtime increasing for over 32 processors. The progressive alignment phase has an improved runtime for up to 8 processors but then the runtimes stays almost the same up to 64 processors. Overall the runtime curve is about average for up to 16 processors.

In Figure 5.13 we notice the pairwise speedup curve is below average for more than 4 processors. The guide tree does not have much of a speedup at any number of processors. The progressive alignment phase has a noticeable speedup for up to 8 processors which is due to the good balance of the tree with $B = 6$. The overall speedup is about average with 9 for 16 processors and 12 for 32 processors.

5.4.7 TRYPSIN

In Figure 5.7 we see a nice runtime curve for the pairwise phase for up to 32 processors. The guide tree takes a very small portion of the runtime and does not benefit from extra processors. We see an improvement in runtime for the progressive alignment for up to 4 processors. Overall the runtime gets a decent improvement for up to 32 processors.

In Figure 5.14 we see that the speedup for the pairwise alignment is very good up to 32 processors but then drops a little for 64. The guide tree does not have a significant speedup for this dataset. The progressive alignment gets a small speedup for up to 4 processors with its guide tree with a balance of $B = 11$, which is average. The overall speedup is slightly below average with around 7 for 16 processors and 8 for 32. Due to the imbalance of the tree, we do not get very good results for over 4 processors.

5.4.8 Summary & Discussion

Looking at the speedups of the three phases will give us a good indication as to the total speedup. As we have already seen, the previously most time consuming part, the pairwise phase, has a good speedup for any number of processors as shown in Figure 5.15. However, the fact that the guide tree building and the progressive alignment have an almost negligible speedup brings the total speedup down. The progressive alignment quickly becomes the most time consuming part of all three phases (50-90%) for large data sets.

Despite that restriction, we still see good total speedups such as 11.0 for the GPCR dataset. In fact that speedup is comparable to the SGI implementation on a shared-memory architecture. SGI also used a subset of 600 sequences from the GPCR data set but with an average of 390 residues as opposed to an average of 364 residues for our data set. They achieved a speedup of around 10.5 on 16 processors using that data set on their SGI Origin 3000 [10].

The number of sequences and their length greatly affect the overall speedup, as can be seen in Figure 5.16, but also by the shape of the guide tree as mentioned in the previous section. The worst speedup of 5.0 for 16 processors was recorded for the smallest data set, SYNAPSIN. The other data set performed fairly well considering the progressive alignment speedup. However as we increase the number of processors, the limitation of the progressive alignment shows up even more, making the use of over 32 processors useless.

Using the height of the guide tree to calculate a balance coefficient has helped up explain why certain data sets have good or poor progressive alignment speedups. In most cases unfortunately, the parallel progressive alignment does not benefit from more than 4 processors. But even in the best case of a balance tree, we would observe speedups much less than linear, due to the fact that we have half the number of alignments at every level going up the guide tree. If we have n sequences, we have $n/2$ alignments at the leaf level. Then on the next level, we will have $n/4$ alignments and so on until we have 1 alignment at the root. If we start with $p = n/2$, each of them will have one alignment to execute at the first level. At the second level however we only have $n/4$ alignments; therefore only half the processors will be used and so on until we can only use one processor at the root level.

Chapter 6

Conclusion

6.1 Summary of contributions

In this thesis we presented the concepts of multiple sequence alignments as well as the algorithms behind them. We introduced a parallel implementation of the popular sequence alignment software CLUSTAL W, its implementation and performance analysis. Finally we presented a web portal called Clustal XP accessible online by any interested individuals.

In Chapter 1 we presented the necessary biology background relating to sequence alignments such as local and global alignments, gaps and penalties, and substitution matrices. In Chapter 2 we reviewed the topic of multiple sequences alignment by presenting different tools currently available as well as the algorithms used in the CLUSTAL W alignment software. We also discussed other available parallel implementation of multiple sequence alignment. In Chapter 3 we described our parallel multiple sequence alignment implementation called pCLUSTAL. We examine the algorithms used in details as well as offer a load balancing scheme for the parallel progressive alignment. We then analyze the theoretical performance of each part of the algorithm and give a running example of a parallel alignment. In Chapter 4 we discuss the issues and other possible methods in implementing the pCLUSTAL tool. We offer an improvement of the guide tree creation algorithm and present the improved runtimes. We

also take a look at a previous attempt implementation of the parallel progressive and show the differences we obtained compared to the original CLUSTAL W. We follow by presenting the online web portal called CLUSTAL XP that can be by anyone to align large numbers of sequences. We present the new feature called Parallel FPT Minimum vertex cover [2] and its usefulness in filtering sequence sets. Finally, in Chapter 6 we study the experimental performance analysis of pCLUSTAL. We start by describing our methodology and the data sets we used in our testing. Then after presenting the graphs of both runtimes and speedups of all the seven data sets, we study each to observe and explain performances. In order to explain the parallel progressive alignment speedups we offer a way of using the height of the tree to gage the level of unbalance.

6.2 Open Problems

One of the open problems in this thesis is to create an optimal parallel multiple alignment algorithm to replace the current near-sequential progressive alignment algorithm. Such an algorithm would need to have its sequences aligned independently from one another. Most of the alternative multiple alignment algorithms to progressive alignment are NP-complete and would be non-trivial to implement in parallel. We were bound by our original restrictions to use the algorithms found in CLUSTAL W and had to be sure the resulting alignment would be identical. If we remove that restriction, then we are open other possibilities. However, for a parallel implementation to be correct it would have to be studied and accepted by biologists. An algorithm can be fast but also be worthless in practical use.

Bibliography

- [1] James Cheetham, Frank Dehne, Sylvain Pitre, Andrew Rau-Chaplin, Peter J. Taillon, "Parallel CLUSTAL W For PC Clusters," in Proc. *International Conference on Computational Science and Its Applications (ICCSA 2003)*, Montreal, Canada, 2003, Springer Lecture Notes in Computer Science, Vol. 2667, Part II, pp. 300-309, 2003.
- [2] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, P. J. Taillon, "A Parallel FPT Application For Clusters," in Proc. *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, pp. 70-77, 2003.
- [3] H. Carillo. and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math*, 48(5):1073–1082, 1998.
- [4] G.H. Gonnet, C. Korostensky, and S. Benner. Evaluation measures of multiple sequence alignments. *J Comput Biol.*, 7(1-2):261–276, 2000.
- [5] H.G. Higgins, A.J. Bleasby, and R. Fuchs. Clustal v: improved software for multiple sequence alignment. *CABIOS*, 8:189–191, 1992.
- [6] H.G. Higgins and P.M. Sharp. Clustal: a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73:237–244, 1988.
- [7] H.G. Higgins and P.M. Sharp. Fast and sensitive multiple sequence alignments on a microcomputer. *CABIOS*, 5:151–153, 1989.

- [8] H.G. Higgins, J.D. Thompson, and T.J. Gibson. Using Clustal for multiple sequence alignments. *Methods Enzymol*, 266:383–402, 1996.
- [9] F. Jeanmougin, J.D. Thompson, M. Gouy, D.G. Higgins, and T.J. Gibson. Multiple sequence alignment with Clustal X. *Trends Biochem Sci*, 23:403–405, 1998.
- [10] Mikhailov, D., Cofer, H. and Gomperts, R. Performance optimization of Clustal W: Parallel Clustal W, HT Clustal, and MULTICLUSTAL, White papers, Silicon Graphics Inc., Mountain View, CA, USA, 2001.
- [11] J.D. Thompson, T.J. Gibson, F. Plewniak, F. Jeanmougin, and H.G. Higgins. The clustalx windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools. *Nucleic Acids Research*, 24:4876–4882, 1997.
- [12] J.D. Thompson, H.G. Higgins, and T.J. Gibson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.
- [13] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, 1997.
- [14] D. Gusfield, *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, 1997.
- [15] Yuan, J., Amend, A., Borkowski, J., DeMarco, R., Bauiley, W., Liu, Y., Xie, G., and Blevins, R. MULTICLUSTAL: a systematic method for surveying Clustal W alignment parameters. *Bioinformatics*, 15[10], 862, 1999.

- [16] Li, Kuo-Bin, ClustalW-MPI: ClustalW Analysis Using Distributed and Parallel Computing, , *Bioinformatics*, 19(12), 1585—1586, 2003.
- [17] Trelles O., On the parallelization of bioinformatic applications, Briefings in Bioinformatics (vol.2), 2001.
- [18] Higgins, D.G., Bleasby, A.J. and Fuchs, R. CLUSTAL V: improved software for multiple sequence alignment. Computer Applications in the Biosciences (CABIOS), 8(2):189-191, 1992.
- [19] Fogel, G.B., Krink, T., Thiemo, K., A Clustal Alignment Improver using Evolutionary Algorithms, 2002.
- [20] Corpet, F., "Multiple sequence alignment with hierarchical clustering", Nucl. Acids Res., 16 (22), 10881-10890, 1988.
- [21] E. Sobel and M. Martinez. *A multiple sequence alignment program*. Nucleic Acids Research, 14:363--374, 1986. 235.
- [22] H. Carrillo, and D.J. Lipman, "The multiple sequence alignment problem in biology," *SIAM Journal of Applied Mathematics*, vol. 48, pp. 1073-1082, 1988.
- [23] S.F. Altschul, and D.J. Lipman, "Trees, stars, and multiple biological sequence alignment," *SIAM Journal of Applied Mathematics*, vol. 49, pp. 197-209, 1989.
- [24] D.J. Lipman, S.F. Altschul, and J.D. Kececioglu, "A tool for multiple sequence alignment," *Proceedings of the National Academy of Sciences USA*, vol. 86, pp. 4412-4415, 1989.
- [25] C. Korostensky, and G. H. Gonnet, "Near Optimal Multiple Sequence Alignments Using a Traveling Salesman Problem Approach," SPIRE, 105-114, 1999.

- [26] K. Chellapilla, and G.. B. Fogel, "Multiple sequence alignment using evolutionary programming", Congress on Evolutionary Computation, 445-452, 1999.
- [27] M. Isokawa, M. Wayama, and T. Shimizu, "Multiple sequence alignment using a genetic algorithm," *Genome Informatics*, 7, 176-177, 1996.
- [28] Saitou, N., Nei, M., The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.* 4, 406-425, 1987.
- [29] Thompson, J. D., Higgins, D. G. & Gibson, T. J. Improved sensitivity of profile searches through the use of sequence weights and gap excision. *CABIOS* 10, 19-30., 1994.
- [30] Feng, D.F. and Doolittle, R.F., Progressive sequence alignment as a prerequisite to correct phylogenetic trees.. *Journal of Molecular Evolution* 25:351-360, 1987.
- [31] Taylor, W., Multiple protein sequence alignment using double-dynamic programming. *Computers & Chemistry*, Volume 24, Issue 1, Pages 3 – 12., 2000.
- [32] Stoye, J., Multiple sequence alignment with the divide-and-conquer method, *Gene* 211 (2), GC45-GC56, 1998.
- [33] Sneath, P.H.A. and Sokal, R.R. *Numerical Taxonomy* (pp; 230-234), W.H. Freeman and Company, San Francisco, California, USA, 1973.
- [34] Needleman, S. B., and Wunsch, C. D. An Efficient Method Applicable to the Search for Similarities in the Amino Acid Sequences of Two Proteins. *Journal of Molecular Biology* 48, 444-453, 1970.

- [35] Katoh,K., Misawa,K., Kuma,K. and Miyata,T. (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Res.*, 30, 3059–3066.
- [36] B. Morgenstern, A. Dress, T. Werner: Multiple DNA and protein sequence alignment based on segment-to-segment comparison *Proc. Natl. Acad. Sci. USA* 93, pp. 12098-12103, 1996.
- [37] B. Morgenstern, K. Frech, A. Dress, T. Werner: DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics* 14, 290-294, 1998.
- [38] B. Morgenstern, W.R. Atchley, K. Hahn, A. Dress: Segment-based scores for pairwise and multiple sequence alignments. *Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB 98)*, in press, 1998.
- [39] Hagerup, T., Allocating independent tasks to parallel processors. *J. Parallel Distrib. Comput.*, 47, 185-197, 1997.
- [40] Andrews, G.R., *Foundations of multithreaded, parallel, and distributed programming*. Addison-Wesley, Reading, MA., 2000.
- [41] Myers, E.W. and Miller, W. Optimal alignments in linear space. *Comput. Appl. Biosci.*, 4, 11-17, 1988.

Appendix A

Glossary

Alignment Comparing two sequences to achieve the maximum level of identity and conservation in order to find the degree of similarity of homology.

Amino Acid Simple molecules composed of a central carbon atom, a hydrogen atom, an amino group, a carboxy group and a side chain. The differences between the 20 common amino acids are found in the side chain.

BLOSUM **B**locks **S**ubstitution **M**atrix. A popular substitution matrix.

DNA Deoxyribonucleic acid, double strand of molecules, also called double helix. It's main function is to encode information. The bases of DNA are adenine (A), cytosine (C), guanine (G) and thymine (T).

Distance Matrix A matrix in which a cell A_{ij} represents a distance between two sequences i and j . That distance is usually calculated on a pairwise score. The higher the pairwise score, the smaller the distance.

Global Alignment Alignment of two sequences along their entire length.

Guide Tree A 2-Tree used in progressive alignments. A node in a guide tree either has two children or none. Two leaves share a parent when they are relatively close to one another (based on a distance matrix).

H-Relation Each processor sends at most h messages and receives at most h messages.

Homologous protein Two proteins that share common evolutionary history (or an ancestor). Two proteins that have common biological or chemical composition are often referred as homologous.

Local alignment Alignment of some segments of two sequences.

Nucleic acid A chain of nucleotides (RNA or DNA molecule).

Nucleotide Adenine (A), cytosine (C), guanine (G) and thymine (T) in DNA, adenine (A), cytosine (C), guanine (G) and uracil (U) in RNA.

PAM Percent Accepted Mutation. A substitution matrix introduced by Dayhoff et al.

Phylogenetic Tree Also known as Phylogeny. Trees in which leaves represent different species and the intermediate nodes represent hypothesized ancestors.

Protein A polypeptide, a chain of linked amino acids.

Residue An amino acid as found in a protein.

RNA Ribonucleic acid, single-stranded nucleic acid similar to DNA but having ribose sugar rather than deoxyribose sugar and uracil rather than thymine as one of the pyrimidine bases. Its role includes the transmission of genetic information from DNA (transcription) and into protein (translation).

Substitution Matrix A matrix of size 20×20 (reflecting the number of amino acids) used to quantify the amount of evolutionary change in a protein sequence. A substitution matrix will give a score for the probability that an amino acid A would mutate into amino acid B . The simplest version would be an identity matrix, when 0 is a mismatch and 1 is an identical match. There are several use substitution matrix, such as PAM, BLOSUM and Gonnet, as well as variants of each (e.g. BLOSUM62).

Appendix B

Command Line Parameters

DATA

/INFILE=file	input sequences
/PROFILE1=file	profiles (old alignment)
/PROFILE2=file	

VERBS

/OPTIONS	list the command line parameters
/HELP or /CHECK	outline the command line params
/ALIGN	do full multiple alignment
/TREE	calculate NJ tree
/BOOTSTRAP(=n)	bootstrap a NJ tree (n= number of bootstraps; def. = 1000)
/CONVERT	output the input sequences in a different file format

GENERAL SETTINGS

/INTERACTIVE	read command line, then enter normal interactive menus
/QUICKTREE	use FAST algorithm for the alignment guide tree
/NEGATIVE	protein alignment with negative values in matrix
/OUTFILE=file	sequence alignment file name
/OUTPUT= *	* = GCG, GDE, PHYLIP or PIR
/OUTORDER= *	* = INPUT or ALIGNED
/CASE= *	* = LOWER or UPPER (for GDE output only)
/SEQNOS	OFF or ON (for Clustal output only)

FAST PAIRWISE ALIGNMENTS

/KTUPLE=n	word size
/TOPDIAGS=n	number of best diags
/WINDOW=n	window around best diags
/PAIRGAP=n	gap penalty
/SCORE	PERCENT or ABSOLUTE

SLOW PAIRWISE ALIGNMENTS

/PWMATRIX=	Protein weight matrix=BLOSUM, PAM, GONNET, ID or file
/PWDNAMATRIX=	DNA weight matrix=IUB, CLUSTALW or file
/PWGAOPEN=f	gap opening penalty
/PWGAPEXT=f	gap extension penalty

MULTIPLE ALIGNMENTS

/NEWTREE=	file for new guide tree
/USETREE=	file for old guide tree
/MATRIX=	Protein weight matrix=BLOSUM, PAM, GONNET, ID or file
/DNAMATRIX=	DNA weight matrix=IUB, CLUSTALW or file
/GAOPEN=f	gap opening penalty
/GAPEXT=f	gap extension penalty
/ENDGAPS	no end gap separation pen
/GAPDIST=n	gap separation pen. range
/NOPGAP	residue-specific gaps off
/NOHGAP	hydrophilic gaps off
/HGAPRESIDUES=	list hydrophilic residues
/MAXDIV=n	% identity for delay
/TYPE=	PROTEIN or DNA
/TRANSWEIGHT	transitions weighted

PROFILE ALIGNMENTS

/PROFILE	Merge two alignments by profile alignment
/NEWTREE1=	file for new guide tree for profile1
/NEWTREE2=	file for new guide tree for profile2
/USETREE1=	file for old guide tree for profile1
/USETREE2=	file for old guide tree for profile2

SEQUENCE TO PROFILE ALIGNMENTS

/SEQUENCES	Sequentially add profile2 sequences to profile1 alignment
/NEWTREE=	file for new guide tree
/USETREE=	file for old guide tree

STRUCTURE ALIGNMENTS

/NOSECSTR1	do not use secondary structure/gap penalty mask for profile 1
/NOSECSTR2	do not use secondary structure/gap penalty mask for profile 2
/SECSTROUT=	STRUCTURE/MASK/BOTH/NONE output in alignment file
/HELIXGAP=n	gap penalty for helix core residues
/STRANDGAP=n	gap penalty for strand core residues
/LOOPGAP=n	gap penalty for loop regions
/TERMINALGAP=n	gap penalty for structure termini
/HELIXENDIN=n	# of residues inside helix to be treated as terminal
/HELIXENDOUT=n	# of residues outside helix to be treated as terminal
/STRANDENDIN=n	# of residues inside strand to be treated as terminal
/STRANDENDOUT=N	# of residues outside strand to be treated as terminal

TREES

/OUTPUTTREE=	nj OR phylip OR dist
/SEED=n	seed number for bootstraps
/KIMURA	use Kimura's correction
/TOSSGAPS	ignore positions with gaps

Appendix C

Example Protein Sequences

These are the sequences that were used in the examples in this thesis (from SYNAPSIN).

>1_human_Ia

```
MNYLRRRLSDSNFMANLPNGYMTDLQRPQPPPPPPGAHSPGATPGPGTATAERSSGVAPAASPAAPSPGS
SGGGGFFSSLSNAVKQTAAAAAATFSEQVGGGSGGAGRGGAASRVLLVIDEPHTDWAKYFKGKKIHGEID
IKVEQAEFSDLNLVAHANGGFSVDMEVLRNGVKVVRSLKPDFVLIROHAFSMARNGDYRSLVIGLQYAGI
PSVNSLHSVYNFCDKPWVFAQMVRLHKKLGTEEFPLIDQTFYPNHKEMLSSTTYPVVVKMGHAHSGMGKV
KVDNQHDFQDIASVVALTKTYATAEPFIDAKYDVRVQKIGQNYKAYMRTSVSGNWKTN TG SAMLEQIAMS
DRYKLWVDTCEIFGGGLDICAVEALHGKDGRDHIIEVVGSSMPLIGDHQDEDKQLLIVELVNVKMAQALPR
QRORDASPGRGSHGQTPSPGALPLGRQTSQQPAGPPAQQRPPPPQGGPPQPGPGPQRQGPPLQQRPPPPQGG
QHLSGLGPPAGSPLPQRLPSPTSAPQQPASQAAPPTQGQGRQSRPVAGGPGAPPAARPPASPSQRQAGP
PQATRQTSVSGPAPPKASGAPPGQQRQGPPOKPPGPAGPTRQASQAGPVPRTGPPTTQQPRPSGPGPAG
APKPQLAQKPSQDVPPPATAAAGGPPHPQLNKSSQLTNAFNLPEPAPPRPSLSQDEVKAETIRSLRKSFA
SLFSD
```

>2_bovine_Ia

```
MNYLRRRLSDSNFMANLPNGYMTDLQRPQPPPPPPAAPSPGATTGPATATAERASSAAPVASPAAPSPGS
SGGGGFFSSLSNAVKQTAAAAAATFSEQVGGGSGGAGRGGAARVLLVIDEPHTDWAKYFKGKKIHGEID
IKVEQAEFSDLNLVAHANGGFSVDMEVLRNGVKVVRSLKPDFVLIROHAFSMARNGDYRSLVIGLQYAGI
PSINSLHSVYNFCDKPWVFAQMVRLHKKLGTEEFPLINQTFYPNHKEMLSSTTYPVVVKMGHAHSGMGKV
KVDNQHDFQDIASVVALTKTYATTEPFIDAKYDVRIQKIGQNYKAYMRTSVSGNWKTN TG SAMLEQIAMS
DRYKLWVDTCEIFGGGLDICAVEALHGKDGRDHI IQVVGSSMPLIGDHQDEDKQLLIVELVNVKMAQALPR
QRORDASPGRGSHSQTSPGALPLGRQISQQPAGPPAQQRPPPPQGGPPQPGPGPQRQGPPLQQRPTPQGG
QHLSGLGPPAGSPLPQRLPSPTSVPPQASQATPMTQGQGRQSRPVAGGPGAPPATRPASPSQRQAGP
PQATRQTSVSGQAPPKASGVPPGGQQRQGPPOKPPGPAGPTRQASQAGPMPRTGPPTTQQPRPSGPGPAG
RPTKPQLAQKPSQDVPPPATAAAGGPPHPQLNKSSQLTNAFNLPEPAPPRPSLSQDEVKAETIRSLRKSF
ASLFSD
```

>3_rat_Ia

```
MNYLRRRLSDSNFMANLPNGYMTDLQRPQPPPPPPSAASPGATPGSAAA SAERASTAAPVASPAAPSPGS
SGGGGFFSSLSNAVKQTAAAAAATFSEQVGGGSGGAGRGGAARVLLVIDEPHTDWAKYFKGKKIHGEID
IKVEQAEFSDLNLVAHANGGFSVDMEVLRNGVKVVRSLKPDFVLIROHAFSMARNGDYRSLVIGLQYAGI
PSVNSLHSVYNFCDKPWVFAQMVRLHKKLGTEEFPLIDQTFYPNHKEMLSSTTYPVVVKMGHAHSGMGKV
KVDNQHDFQDIASVVALTKTYATAEPFIDAKYDVRVQKIGQNYKAYMRTSVSGNWKTN TG SAMLEQIAMS
DRYKLWVDTCEIFGGGLDICAVEALHGKDGRDHIIEVVGSSMPLIGDHQDEDKQLLIVELVNVKMTQALPR
QRDASPGRGSHSQTSPGALPLGRQTSQQPAGPPAQQRPPPPQGGPPQPGPGPQRQGPPLQQRPPPPQGGQH
LSGLGPPAGSPLPQRLPSPTAAPQQSASQATPMTQGQGRQSRPVAGGPGAPPAARPPASPSQRQAGPPQ
ATRQASISGGPAPPKASGSPGGQQRQGPPOKPPGPAGPIRQASQAGPGPRTGPPTTQQPRPSGPGPAGRP
TKPQLAQKPSQDVPPPIIAAAGGPPHPQLNKSSQLTNAFNLPEPAPPRPSLSQDEVKAETIRSLRKSFA
```

LFS

>4_mouse_Ib

MNYLRRRLSDSNFMANLPNGYMTDLQRPQPPPPPSAASPGATPGSATASAERASTAAPVASPAAPSPGS
SGGGGFFSSLSNAVKQTAAAAATFSEQVGGSGGAGRGGAARVLLVIDEPHTDWAKYFKGKKIHGEID
IKVEQAEFSDLNLVAHANGGFSVDMEVLRNGVKVVRSLKPDFVLIRQHAFSMARNGDYRSLVIGLQYAGI
PSVNSLHVSYNFCDKPWVFAQMVRLHKKLGTEEFPLIDQTFYPNHKEMLSSTTYPVVVKMGHAHSGMGKV
KVDNQHDFQDIASVVALTKTYATAEPFIDAKYDVRVQKIGQNYKAYMRTSVSGNWKNTGTSAMLEQIAMS
DRYKLWVDTCEIFGGLDICAVEALHGKDGRDHIIEVVGSSMPLIGDHQDEDKQLIVELVVNKMTOALPR
QPORDASPRGSHSQSSSPGALTIGRQTSQQPAGPPAQORPPPQGGPPQPGPGPQROGPPLOQRPPPQGO
QHLSGLGPPAGSPLPQRLPSPTAAPQOSASQATPVTQGGGRQSRPVAGGPGAPPAARPPASPSQORQAGA
PQATRQASISGPAPTASGAPPGGQORQGGPPQKPPGPAGPTRQASQAGPGPRTGPPTTQQRPSGPGPAG
RPAKPQLAQKPSQDVPPFITAAAGGPPHPQLKASPSQAQP

>5_mouse_IIb

MMNFLRRRLSDSSFIANLPNGYMTDLQRPQPPPPAPGPGAATASAATSAASPGPERPPPAQAPAPQP
APQPAPTSPVSGSSFFSSLSQAVKQTAASAGLVDAPAPSAASRKAKVLLVVDEPHTDWAKCFRGKKILGDY
DIKVEQAEFSELNLVAHADGTAYAVDMQVLRNGTKVVRSEFRPDFVLIRQHAFGMAENEDFRHLVIGMQYAG
LPSINSLESIYNFCDKPWVFAQMVAFKTLGGEKFLIEQTYYPNHREMLTLPFTPVVVKIGHAHSGMGK
VKVENHYDFQDIASVVALTQTYATAEPFIDAKYDIRVQKIGNNYKAYMRTSISGNWKNTGTSAMLEQIAM
SDRYKLWVDACSEMFGGLDICAOKAVHKGDKDYIFEVMDCSMPLIGEHQVEDRQLITDLVISKMNQLLS
RTPALSPQRPLTTQQPQSGTLKEPDSSKTPPQRPQGGCLQYILDCNGIAGVGPQVQAS

>6_camel

MPPPPAPGQPAGAAPELSLSFGAGKTPATAAPAPPRGVSAPTSPAKSREGLLQRVQSLTGAARDQGASIL
GAAVQSATQRAPAFSKDKYFTLLVLDQNTDWSKYFRGRRLHGDFFIRGEQAEFRDITVVSSADTGPVVT
MAAYRSGTRVARSEFRPDFVLIRQPPRDGSSDYRSTILGLKYGGVPSINSLHSIYQFDKQWVFSHLLQLQ
RRLGRDGFPLIEQTFPPNPRDLFQFTKFPVSLKAGHCHGGVATARLENQALQDAAGLVSGAGNDSHCYC
TIEPYIDAKFSVHIQKIGNNYKAFMRKSITGNWKTNQGSAMLEQITLTKYKSWVDEISELFGGMEVCGL
SVVVAKDGREYIISACNSTFALIGDTQEEDRRQIADLVSGRMQNVCRPSMAQTGPGKLPSRSSVSSRAES
PTDEGVAPTPLPAGPRPAPMGPPPIPERTSPAVGSIGRLSSRSSISEVPEEPSSSGPSTVGGVRRDSQ
TSQSSTISSSVSRAGQRPPTQNSVVEDAEDTMKNLRKTFAGIFGDMEIANKKRGRRTASETSSGSGPGSV
PSSAGPGSGFSSSFLGKQFSFAGKGEVISTOPTQRPSEPPAIPPTASSAVRPESVSVSDSRNTDTLT
ERAGAGYQPVNTNIEQQERVNPFDKPSKSGSAASIHTSSSSSISSSISSRINRNGNAIKSPPPPAGPPP
PPPTNVTAVGSNANSSSGYRNSFSSSLSKDKTSYGNYGSTTSVETITRMDTNTTNIGATATEAGEASGVT
AITNISNSDGIVAPTTGTITTSVTTNDWRSAGMRSASVYSAPAAVTTVLPGDTSGYDSNSIASQGEGLN
NPDDLPSYTRPSYSRSESNAKHSDDLVIIFGDSKTPASYGNGKYTRAAGSISDADMIFGGPPSNYKTRD
FGASKSMSMTSGGVGSGNGSGSGLGGYKIYDGIQNAAFSDFSDSGSMSSIGSHTKRWSASKEEDELDLK

>7_frog_I

MMNYLRRRLSDSNFMANLPNGYMSDLQRPDPAPPPPPAAAAAPQGPPPPAVSPEPPSSGGGFFSSLSNAV
KQTTAALSEQVSTATGVSPGLXGSKVLLVIDEPHGDWGLKFKGKRINGEYDIKVEQAEFCDNLVAHSNG
SFSVDLEVLNRNGVKVVRSLKPDFVLIRQHAFSMARNGNFRSLVIGLQYAGVPSLNSLHVSYNFCDKPWVF
SQLVRLHKKLGPEEFPLIEQTYYPNHKEMLTASKFPVVIKMGHAHSGMGKVVDNQYDFQDIASVVALTK
TYATSEPFIDAKYDVRIQKIGSNYQAYMRTSISGNWKNTGTSAMLEQIAVSDRYKLWVDACADIFGGLDI
CAVEALHGKDGRDHIIEVVGSSMPLIGDQQEEEDKQLIADLVVSRMSQNVQCSPLSRPTHGTQPPAQVVS
S