2008-18

Software and Hardware Acceleration of the Genomic Motif Finding Tool PhyloNet

Authors: Justin Brown

Corresponding Author: jtb1@cec.wustl.edu

Type of Report: Other

WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science

Department of Computer Science and Engineering

Thesis Examination Committee:
Jeremy Buhler, Chair
Mark Franklin
Tao Ju

SOFTWARE AND HARDWARE ACCELERATION OF THE GENOMIC MOTIF FINDING TOOL PHYLONET

by

Justin Tyler Brown, B.S.C.S

A thesis presented to the School of Engineering
of Washington University in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

August 2008
Saint Louis, Missouri

ABSTRACT OF THE THESIS

Software and Hardware Acceleration of the Genomic Motif Finding Tool PhyloNet

by

Justin Tyler Brown

Master of Science in Computer Science

Washington University in St. Louis, 2008

Research Advisor: Professor Jeremy Buhler

A major area of research in molecular and computational biology is deciphering the cis-regulatory network that governs transcriptional regulation. This task has proven to be a challenge because regulatory elements are usually short, degenerate, and hidden in very long sequences. A recently developed algorithm known as PhyloNet attempts to computationally identify conserved regulatory motifs of an organism by using alignments to related species as evidence of conservation. In this thesis, we address the problem of scaling PhyloNet to handle large genomes. We first work to improve PhyloNet in software alone, improving its speed and sensitivity. Our improved version of PhyloNet on the budding yeast genome yields 1.6x more known yeast motifs and a speedup of 20x over the originial. We then develop a streaming architecture design of PhyloNet and show how the seed matching and extension stages can be implemented using Field Programmable Gate arrays. We estimate that our FPGA design yields an additional order of magnitude speedup over our best software version of PhyloNet.

Acknowledgments

I would like to thank Dr. Jeremy Buhler for all of his advisement, patience, and invaluable feedback over the past two years. I give much thanks to the members of Dr. Buhler's lab group, Yanni Sun, Hongtao Sun, Arpith Jacob, and Josh Coats for their friendship, encouragement, and assistance.

I wish to thank the members of my research group, Mark Franklin, Roger Chamberlain, Eric Tyson, and Saurabh Gayen for their keen insights and constructive criticisms.

I am much appreciative for the support from the National Science Foundation.

Finally, I am forever grateful for the support and encouragement of my parents, sister, and Emily. Their belief in me makes all of this possible.

Justin Tyler Brown

Washington University in Saint Louis
August 2008

To Emily wherever I may find her.

Contents

Abstract . . . . . **ii**

Acknowledgments . . . . . **iii**

List of Tables . . . . . **vii**

List of Figures . . . . . **viii**

1 Introduction **1**
- 1.1 Deciphering the Cis-Regulatory Network 2
  - 1.1.1 Biology of the Cis-Regulatory Network 2
  - 1.1.2 Difficulties of Uncovering the Cis-Regulatory Network 4
- 1.2 Computational Approaches for Motif Finding 7
- 1.3 PhyloNet Algorithm 8
  - 1.3.1 PhyloNet's Input 9
  - 1.3.2 Comparing Profiles 10
  - 1.3.3 Stages of PhyloNet 11
- 1.4 Utility of Streaming Architectures 16
- 1.5 Contributions to Phylonet 18

2 Software Improvements to PhyloNet **20**
- 2.1 Software Improvements 20
  - 2.1.1 Improved Clustering 21
  - 2.1.2 Support for Gapped Profiles 23
  - 2.1.3 Other Speed Improvements 25
- 2.2 Experimental performance of PhyloNet 25
  - 2.2.1 The Data Sets 26
  - 2.2.2 Dealing with repetitive and low-complexity regions 26
  - 2.2.3 Measuring Performance of PhyloNet 27
  - 2.2.4 Output Quality in Yeast 29
  - 2.2.5 Efficiency in Yeast 30
  - 2.2.6 Scalability to Fruit Fly 31
- 2.3 Beyond Software PhyloNet 32
  - 2.3.1 Effects of Parameters on the Performance of PhyloNet 32
  - 2.3.2 Bottlenecks in software version of PhyloNet 34

3 Design of an Accelerated Seed Generation Stage **37**

3.1 Stage 1 of PhyloNet . . . 37
3.1.1 Seed Matching using Table Lookups . . . 38
3.1.2 Adapting Table Lookups for PhyloNet . . . 38
3.1.3 Table Lookup Data Structures . . . 39
3.2 Hardware Design of Stage 1 . . . 39
3.2.1 Area Estimates of Table for Stage 1 . . . 40
3.2.2 Occupancy Sizes of Tables . . . 40
3.2.3 Table Design . . . 41
3.3 Stage 1 Performance Model . . . 43
3.4 Performance Estimates for Stage 1 . . . 44
3.5 Conclusion . . . 47

4 Design of an Accelerated Seed Extension Stage . . . 49
4.1 Stage 2 of PhyloNet: Seed Extension . . . 49
4.2 Hardware Design of Stage 2 . . . 50
4.2.1 Fixed Window Extension . . . 51
4.2.2 ALLR calculation in Hardware . . . 54
4.2.3 Storage of Tables in block RAM and Representation of Values 55
4.2.4 Pipeline Design of ALLR calculation . . . 57
4.3 Synthesis of ALLR calculation in Hardware . . . 59
4.4 Performance Estimates of Stage 2 . . . 59
4.5 Overall Performance Estimation . . . 61
4.5.1 Accounting for Query Pre-processing . . . 62

5 Conclusion and Future Works . . . 64
5.1 Conclusion . . . 64
5.2 Future Works . . . 65

Appendix A Glossary . . . 66

References . . . 68

Vita . . . 71

List of Tables

1.1 Binding Site Motifs for *Abf1* 6
1.2 Example of Aligned Promoter Regions and the Profile 9

2.1 Gapped Alignment Versus Multiple Ungapped Alignments 24
2.2 Total CPU Time for *S. cerevisiae* genome 31
2.3 Statistics for Fruit Fly Data Set 31
2.4 Total Running Time of PhyloNet on the Yeast Data Set 36

3.1 Memory requirements of a direct lookup table 40
3.2 Occupancy Sizes of Tables 41
3.3 Query locations per entry for a 36 bit SRAM 42
3.4 Specifications of FPGA 45
3.5 Average Clock Cycles Needed Per Database Entry for Various Parameters 46
3.6 Output of Stage 1 47
3.7 Stage 1 Hardware versus Software Time 48

4.1 Quality of Fixed Window Filter 54
4.2 Composition of Tables for ALLR Calculation 57
4.3 Stage 4 Timing Analysis 60
4.4 Stage 1 Hardware Time with Reduced Clock Frequency 61
4.5 Hardware and software times for each stage of the optimal configuration. 62

List of Figures

1.1 Growth of GenBank (1982-2005). 2
1.2 Cis-regulatory Network 3
1.3 Distances of yeast binding sites (in base pairs) from genes. 4
1.4 A common transcription factor binding to similar but distinct binding sites in the promoters of multiple genes. 6
1.5 Basic approaches for finding motifs. Yellow boxes indicate locations of conservation. (A) Multiple Genes, Single Species. (B) Single Gene, Multiple Species. (C) Combined Approaches. 8
1.6 Phylonet pipeline. 12
1.7 Exact seed matching. 12
1.8 Mapping of profile columns to degenerate sequence. 13
1.9 Extension of Seed Hits into HSPs. 14
1.10 Clustering HSPs into motifs. Cliques in the graph (bold lines) represent clusters. (A) HSPs are mapped onto the query profile. (B) HSPs that overlap are connected. 15
1.11 Consensus sequence of a yeast motif with instances drawn from promoter alignments of four budding yeast species. 16

2.1 Clustering HSPs using an Interval Graph. 22
2.2 Performance of PhyloNet Versions parameterized by p-value cutoff. . 29
2.3 Result Quality of PhyloNet for Different User Inputs. 33
2.4 Time spent in the various stages of PhyloNet. 35

3.1 Path of Table Lookups 42
3.2 Distributions of Positions Returned from Table Lookups for Query Sizes of 1000, 2000, and 4000. 45

4.1 Variable-Length Extension in Software 50
4.2 Fixed-Length Extension in Hardware 51
4.3 Storage of two tables in one block RAM 56
4.4 Pipelined ALLR calculation. 58
4.5 Area usage of ALLR units 59

Chapter 1

Introduction

Sequence analysis encompasses a wide variety of bioinformatic methods used to infer biological information hidden within DNA and peptide sequences. Even though the most reliable way to determine the structure or function of an RNA or protein is by direct experimentation, it is far easier to obtain the DNA sequence of the corresponding gene. This provides a strong motivation to create tools that can infer structure and function by examining the sequence itself. Tasks of sequence analysis are numerous and include aligning sequences, predicting protein structures, and reconstructing gene regulatory networks. The success of sequence analysis methods is in part due to the ability to automate these methods using computers and the availability of large amounts of raw sequence provided by, for example, the Human Genome Project [19].

In order to be useful, computational methods must be able to scale to the large amounts of raw data available. Sequence databases continue to grow exponentially as entire genomes of organisms are sequenced, thus making sequence analysis a computationally demanding task. As an example, GenBank, the NIH database of all publicly available DNA sequences, has steadily doubled in size approximately every 18 months since its inception in 1982 [11] (Figure 1.1).

In this thesis, we consider the acceleration of a particular sequence analysis tool used to detect cis-regulatory sites in genomic DNA. Acceleration is important in contending with the large amounts of raw sequence available for processing. We show how this tool can be accelerated by using programmable logic on Field Programmable Gate Arrays (FPGAs). We report our methods and expected performance improvements over software.

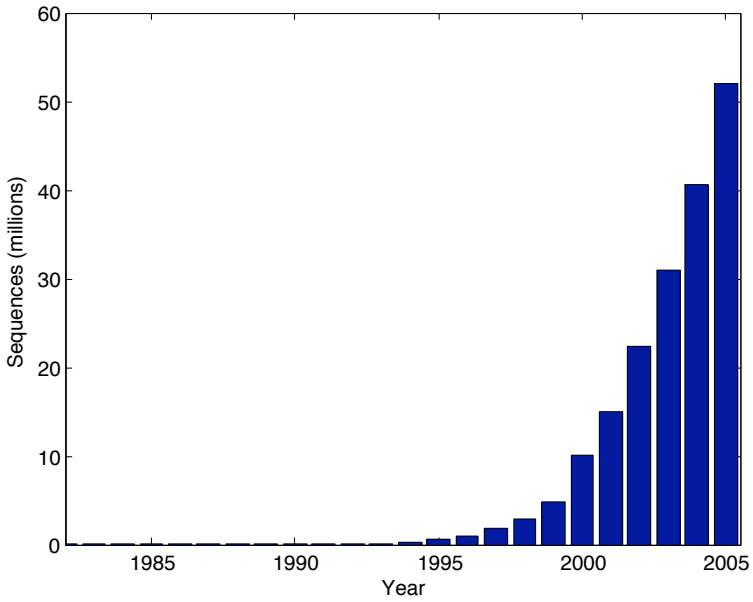

Figure 1.1: Growth of GenBank (1982-2005).

1.1 Deciphering the Cis-Regulatory Network

A key area of genomic research is understanding the mechanisms that control the regulation of gene expression levels. This regulation is governed by the cis-regulatory network which involves the interaction of *transcription factors (TF)* and their binding sites. In this network, proteins known as transcription factors bind to sites within the *promoter* regions upstream of gene-encoding sequences. These binding sites are known as *transcription factor binding sites (TFBS)*. In this thesis, we address the problem of identifying the locations of transcription factor binding sites within the genome.

1.1.1 Biology of the Cis-Regulatory Network

The TFBS are short sequences (6-12 base pairs long) located in promoter regions of the genome, which are regulatory regions of DNA located on the side of the gene where transcription begins. These binding sites act as control points for regulated gene transcription. The binding of TFs to TF Binding Sites affects the presence of

RNA polymerase, the enzyme that synthesizes the RNA from the coding region of the gene. TFs may act as activators or repressors, increasing or preventing the presence of RNA polymerase respectively, which in turn activates or represses the transcription of DNA into RNA (Figure 1.2).

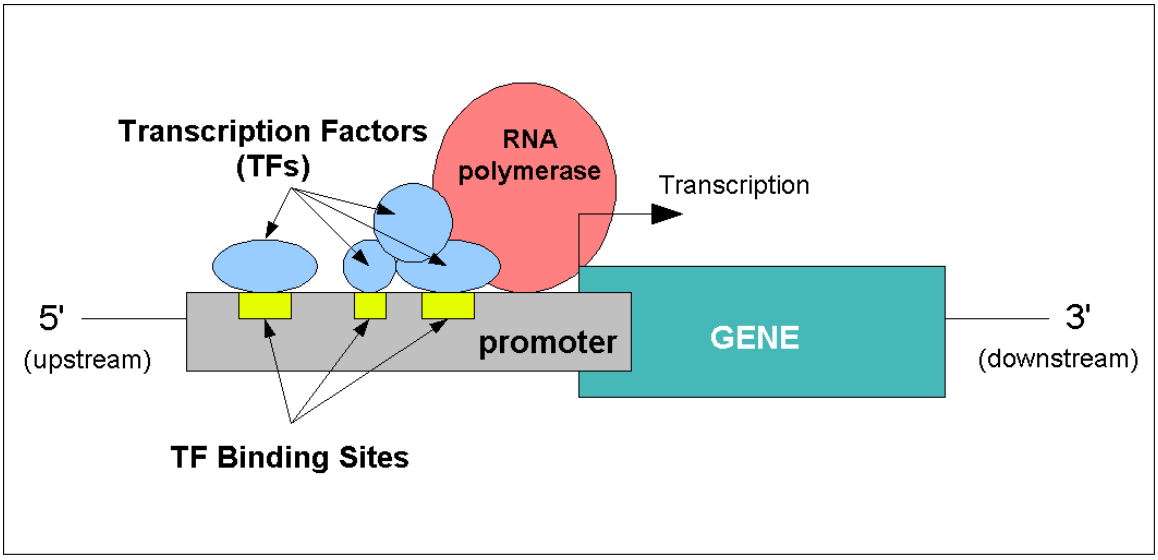

Figure 1.2: Cis-regulatory Network

In a study done on the genome of baker's yeast (*Sacchomyces cerevisiae*), it was shown that binding sites are not uniformly distributed over the promoter regions but instead show a sharply peaked distribution [15]. Very few sites are located in the region 100 base pairs (bp) upstream of protein-coding sequences. This region typically includes the transcription start site and is bound by the transcription initiation apparatus. The vast majority (74%) of the transcriptional regulator binding sites lie between 100 and 500 bp upstream of the protein-coding sequence. The number of binding sites trails off in regions more than 500 bp upstream (Figure 1.3). The yeast transcriptional regulators seem to function at short distances along the linear DNA, a property that reduces the potential for inappropriate activation of nearby genes.

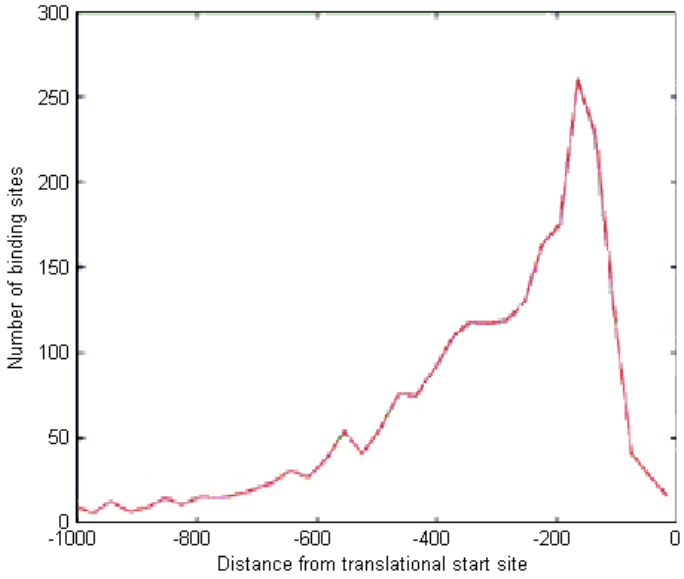

Figure 1.3: Distances of yeast binding sites (in base pairs) from genes.

1.1.2 Difficulties of Uncovering the Cis-Regulatory Network

Identifying the location of transcription factor binding sites within the genome is a key problem in genomic research. Two of the main difficulties in identifying the binding sites are the sheer number of binding sites within the entire genome and the short, degenerate nature of binding sites. Binding sites are typically 6-12 base pairs long and are hidden within promoter regions that can be on the order of 1,000 to 10,000 base pairs. Also, a single transcription factor can bind to sites that are similar but not identical to each other. This degeneracy in the binding sites for a single transcription factor makes the search for a common pattern more difficult.

Scaling of the Cis-Regulatory Network

The well-studied *S. cerevisiae* genome provides an example of the sizes of data involved. The genome of this simple eukaryotic organism is composed of about 13 million base pairs and over 4,600 genes [10]. These genes are regulated by roughly 200 transcription factors [15]. Compared to higher eukaryotes, the genes in yeast are

close together, and so the promoter regions tend to be less than 1,000 base pairs. In general, each transcription factor regulates many genes, meaning multiple binding sites are contained in each promoter region. For example, 2022 genes from *S. cerevisiae* were examined, and 4229 conserved and bound motif sites were mapped across these genes [26]. Also, the Yeastract database (*www.yeastract.com*), a comprehensive online database of the *S. cerevisiae* genome, generally lists tens of TFBS for each gene.

The number of transcription factors within an organism increases with the genome size, and the number of binding sites per gene increases with larger genomes [34]. The human genome has an estimated 20,000 to 25,000 protein-coding genes [29] and over 2000 transcription factors [2]. Thus, approximately 10% of the genes in the genome code for transcription factors. In one *in vivo* analysis of the transcription factors Sp1, cMyc, and p53, an unexpectedly large number of transcription factor binding site regions were found [8]. Minimal estimates of the number of binding sites over the entire human genome were 12,000 for Sp1, 25,000 for cMyc, and 1,600 for p53.

In addition to more abundant transcription factors and binding sites, the promoter regions of higher eukaryotes tend to be longer. The genes in higher eukaryotes are generally more spread out within the genome than the genes in yeast. Thus, while binding sites in yeast are generally less than 1,000 base pairs away from the gene, binding sites in higher eukaryotes can be as much as 10,000 base pairs away from the genes. This means that instead of searching a few million base pairs as in yeast, tens to hundreds of millions of base pairs must be searched in higher eukaryotes such as human. An exhaustive comparison of each promoter sequence to all other promoter sequences is quadratic in the total length of the sequences. The orders of magnitude more sequences involved in higher-level organisms underscore the need to develop fast methods that scale to larger genomes.

Ambiguous Nature of TF Binding Sites

The other main difficulty of identifying binding sites is that they can be short and degenerate. Bindings sites are usually short (6-12 bases) and embedded in long background sequences, creating a low signal-to-noise ratio. Furthermore, binding sites can

be degenerate, meaning that two binding sites with different sequences can bind the same transcription factor (Figure 1.4).

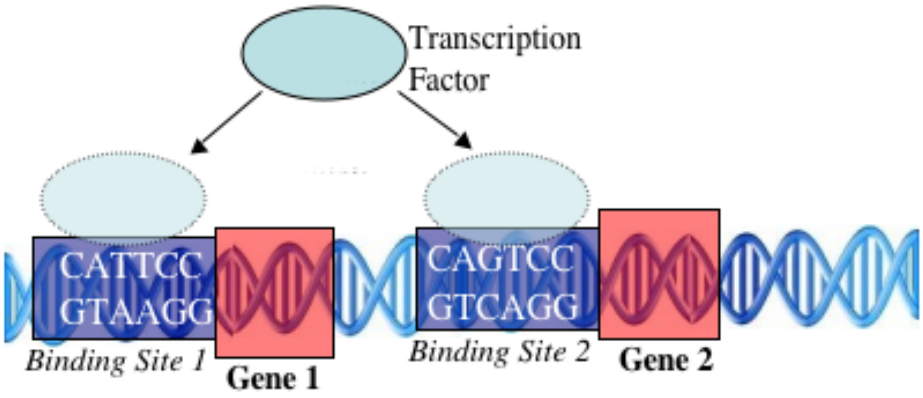

Figure 1.4: A common transcription factor binding to similar but distinct binding sites in the promoters of multiple genes.

The variability of the bases within a binding site weakens the already short, hidden signal, making it harder to distinguish it from random DNA. The discrepancy between the sequences of individual binding sites for the same transcription factor is position-specific. Certain bases are constrained by virtue of their contact with the TF, while others are free to vary [27]. Table 1.1 shows some examples from the literature of possible binding site motifs for the same transcription factor Abf1, illustraing just how dissimilar binding sites can be and yet still possess an affinity for Abf1. In the table, an n stands for no preference for a base, R represents a purine (A or G), Y represents a pyrimidine (C or T), and B represents anything but A (C, G, or T).

Table 1.1: Binding Site Motifs for *Abf1*

TnnCGTnnnnnnTGAT	[4]
TCRTnnnnnAYGA	[7]
RTCRYBnnnnACG	[9]

The middle bases of the binding site are able to mutate without preventing the Abf1 factor from binding. Only a few positions at the ends of the binding site are constrained to certain bases.

1.2 Computational Approaches for Motif Finding

Despite some of the difficulties of locating transcription factor binding sites, many computational approaches have been developed over the past two decades that have met with some success in discovering TF binding sites. Since a binding site can occur many times throughout the genome, the computational approaches generally work by identifying recurring sequence motifs within the genome.

Algorithms to recognize motifs in genomic DNA take one of two basic forms [35]. The *multiple gene, single species* approach (Figure 1.5a) recognizes motifs because they recur with few changes in the promoters of multiple genes within a single genome. Tools of this form include Gibbs Sampler [24], MEME [3], Consensus [17], and AlignAce [18]. In contrast, the *single gene, multiple species* or *phylogenetic footprinting* approach (Figure 1.5b) recognizes motifs in promoters of genes derived from a common ancestral gene (known as *orthologous* genes) across related species. Programs that use this approach include FootPrinter [6], PhyME [31], CompareProspector [25], and PhyloGibbs [30].

Wang and Stormo combined these two approaches into a single software program called PhyloCon (Phylogenetic Consensus) [35]. It is one of the first motif finding algorithms to combine the power of phylogenetic conservation and gene co-regulation to identify conserved regulatory motifs. It attempts to identify conserved regions across alignments of related species (Figure 1.5c).

PhyloCon was used along with Converge [15] to identify a comprehensive list of motifs in *S. cerevisiae* [26]. Converge and PhyloCon each identified more correct motifs than were found using the combined results of the six programs employed in an earlier study [15]. Significant sequence motifs were discovered for 36 transcription factors that were previously missed, and 636 more regulatory interactions were found than in the previous experiment.

To scale PhyloCon's ability to discover motifs across an entire genome, Wang and Stormo developed the successor program PhyloNet (Phylogenetic Network) [36]. PhyloNet implements a BLAST-like seeded alignment algorithm to accelerate detection of

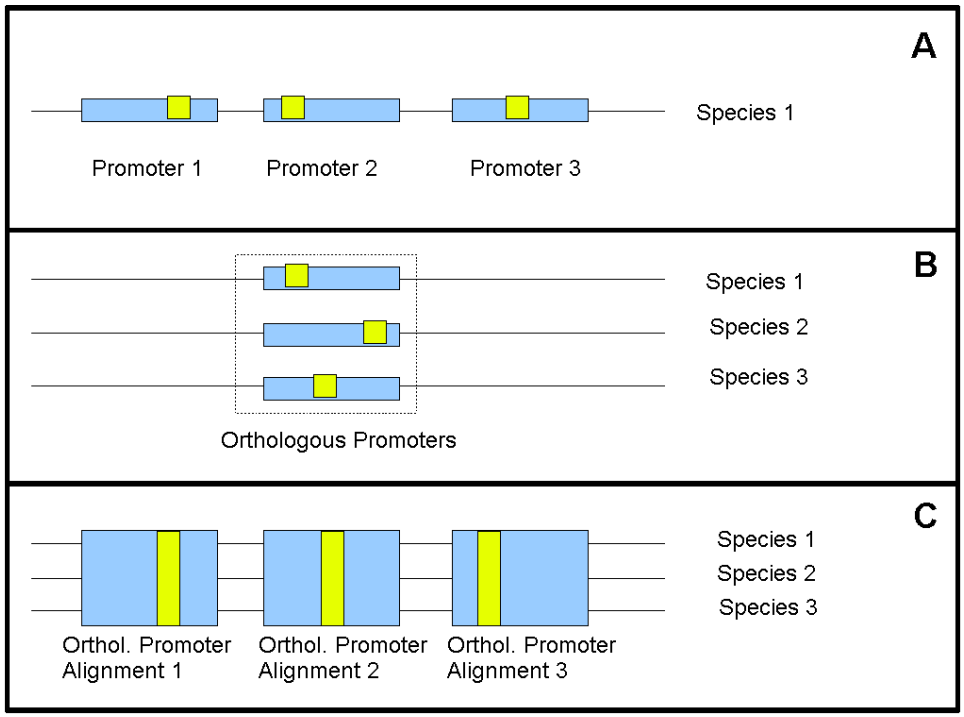

Figure 1.5: Basic approaches for finding motifs. Yellow boxes indicate locations of conservation. (A) Multiple Genes, Single Species. (B) Single Gene, Multiple Species. (C) Combined Approaches.

putative motif instances across thousands of promoters. We now describe this tool in more detail, as this is the algorithm we seek to accelerate using dedicated hardware.

1.3 PhyloNet Algorithm

PhyloNet is a software tool for detecting the regulatory site motifs, or transcription factor binding sites, involved in gene expression regulation. It identifies sites that are conserved across related species and throughout the genome. A single transcription factor can bind to many different promoters throughout the genome. As a result, one would expect to find binding sites for a particular factor in many different promoter regions within a species. Thus, a pattern that recurs often in promoter regions of a single genome is evidence that the pattern represents a TF binding site. Furthermore, since transcription factors play an important role in the transcription of genes, one

would expect that the binding sites would be under more selective pressure than the surrounding background sequence. Thus, highly conserved regions within the promoters of orthologous genes provide further evidence that the region represents a motif. Evidence of conservation across co-regulated genes and among promoter regions of orthologous genes is combined under one general framework in PhyloNet.

1.3.1 PhyloNet's Input

Phylonet takes as input a collection of multiple sequence alignments. A multiple sequence alignment is an alignment of three or more biological sequences. These sequences are generally assumed to be ortholgous. The multiple sequence alignment can be compactly represented as a *profile*, which is a matrix containing the number of each type of base found in each column of the alignment (Table 1.2).

Table 1.2: Example of Aligned Promoter Regions and the Profile

Alignment of Orthologous Promoters	YOR231W_YOR232W.aln_Cer	...TCGCATTTTCCA...
	YOR231W_YOR232W.aln_Par	...TCGCACTTGCCA...
	YOR231W_YOR232W.aln_Mik	...ATGCTTCTGTCA...
	YOR231W_YOR232W.aln_Bay	...TCGCATTTACTA
Profile	A count	...100030001004...
	C count	...030401100330...
	G count	...004000002000...
	T count	...310013341110...

In the context of PhyloNet, each profile represents an alignment of promoter regions from orthologous genes. The promoter regions are regulatory regions of DNA containing the binding sites of transcription factors located on the sides of the genes where transcription begins. In simple organisms such as yeast where the genes are close together, the promoter regions can be taken to be all the intergenic regions of the genome. Each intergenic region is typically less than 1,000 base pairs long. In higher level eukaryotes where the genes are typically more spread out, promoter regions can be defined as all the bases up to a certain distance from the transcriptional start site of each gene. The proximal promoter in higher eukaryotes is generally assumed to

be a couple hundred bases, but there are enhancer regions that could extend 10,000 bases or more from the transcriptional start site.

The set of profiles derived from multiple sequence alignments of orthologous promoter regions represent the input of PhyloNet. In its original form, PhyloNet requires the input alignments to be ungapped. An *ungapped* alignment is an alignment in which there is a one-to-one correspondence between the bases of each sequence. This means that there cannot be a base present in one sequence without a corresponding base present in the other sequence. We extend PhyloNet to be able to handle gapped multiple alignments and discuss this improvement in Chapter 2. *Gapped* alignments allow bases to be inserted into or deleted from one of the sequences, which is biologically common.

1.3.2 Comparing Profiles

Once the profiles of multiple sequence alignments are created, a method needs to be devised to compare profiles. Wang and Stormo developed a novel scoring statistic between columns of two profiles that they call the Average Log Likelihood Ratio (ALLR) [35]. For a given profile column, n_b is the number of instances of base b in the column. The profile column is modeled as an independent and identically-distriubted (i.i.d) random variable generated from a multinomial distribution, where the probability of observing base b is f_b. The probability of observing each base is approximated using the counts of the column, with pseudocounts added to account for small sample biases. Assuming the frequency vector p for all bases is known, one can compute the likelihood ratio

$$LR(f, p; n) = \prod_{b=A..T} \left(\frac{f_b}{p_b}\right)^{n_b}, \tag{1.1}$$

which measures the likelihood the observed base counts came from the approximated base distribution f rather than the background distribution p. The log likelihood ratio LLR is used in practice, which is the log of the likelihood ratio.

$$LLR(f, p; n) = n * \sum_{b=A..T} f_b \ln \frac{f_b}{p_b}, \tag{1.2}$$

where $n = \sum_b n_b$. Given two profile columns i and j with estimated parameters f_i and f_j and counts n_i and n_j respectively, $LLR(f_j, p; n_i)$ compares the log likelihoods that column i came from column j's distribution versus the background distribution p. Likewise, $LLR(f_i, p; n_j)$ compares the log likelihoods that column j came from column i's distribution versus the background distribution p. The ALLR is defined to be the weighted sum of these two log likelihood ratios:

$$ALLR(i,j) = \frac{\sum_{b=A..T} n_{bj} \ln \frac{f_{bi}}{p_b} + \sum_{b=A..T} n_{bi} \ln \frac{f_{bj}}{p_b}}{n_i + n_j}, \tag{1.3}$$

where n_{bi} is the number of occurrences of base b in column i, p_b is the background frequency for base b, and $f_{bi} = n_{bi}/n_i$ is the maximum likelihood estimate for the frequency of base b in the i-th column.

If $n_i = n_j$ as in the case with comparing ungapped multiple alignments of a set number of species, equation 1.3 reduces to

$$ALLR(i,j) = \frac{\sum_{b=A..T} f_{bj} \ln \frac{f_{bi}}{p_b} + \sum_{b=A..T} f_{bi} \ln \frac{f_{bj}}{p_b}}{2}. \tag{1.4}$$

The total ALLR score is the sum of the scores between each column comparison. Similar to the log likelihood ratio, the ALLR score assumes the number of sequences in the alignment is large, and so pseudo-counts are added to adjust for small sample biases.

1.3.3 Stages of PhyloNet

PhyloNet consists of three main stages: (1) Table generation and lookup, (2) Seed Extension, and (3) High-scoring Segment Pair (HSP) clustering. The output of each stage becomes the input of the next stage in a pipelined fashion (Figure 1.6). The amount of output of each stage decreases, but the complexity of the stage increases as the data is moved through the pipeline.

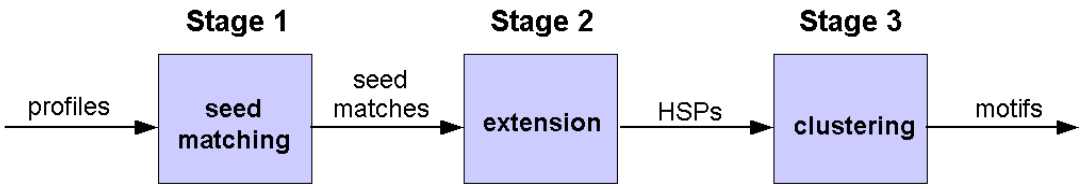

Figure 1.6: Phylonet pipeline.

Table Generation and Lookup

PhyloNet uses an inexact matching scheme to find good approximate alignments among profiles, similar to the approach used by the BLAST algorithm to find inexact matches among DNA sequences [1]. Inexact matches are found by first looking for smaller exact matches that are part of longer exact matches. By the pigeonhole principle, if two words of length l have k mismatches between them, then they will have an exact match of at least length $\lceil \frac{l-k}{k+1} \rceil$. In practice, one can expect to find longer exact matches than this.

Exact matches between two sequences can be efficiently found using lookup tables. Both sequences are divided into words of length w known as *w-mers* or *seeds*. The locations of the w-mers from one of the sequences is stored in a lookup table. The table is queried using the w-mers from the other sequence. A hit in the table corresponds to an exact match of length w between the two sequences (Figure 1.7).

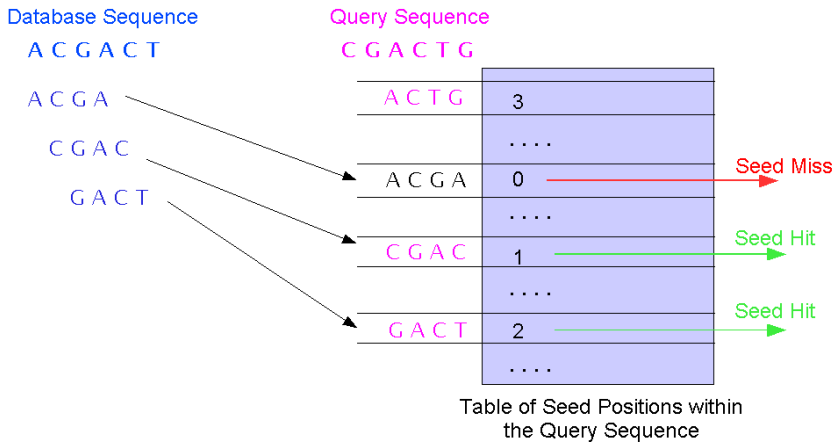

Figure 1.7: Exact seed matching.

In the example given in Figure 1.7, each sequence is divided into seeds of length 4. The positions of the seeds in the query sequence are stored in the table. The table is queried using the database seeds. A hit represents an exact match between the query and database sequences.

This approach can be generalized by searching for matches that score above some threshold rather than simply searching for exact matches. This is done by storing not only all of the w-mers from a sequence into the table but also the *neighborhoods* of those w-mers. The *neighborhood* of a w-mer is the set of all possible w-mers that match the original w-mer with a score greater than some threshold. Now when the table is queried, the lookup will return all the locations of the sequence that match the w-mer to within the threshold. This is the approach taken by BLASTP and also by PhyloNet.

For PhyloNet to take advantage of the table lookup approach, profile columns are degenerately mapped to letters of a discrete alphabet, resulting in a sequence of discrete characters. Each count vector is converted to a frequency vector such that the sum of the elements equals 1. It is then mapped to one of 15 characters (Figure 1.8). The mapping function is created by clustering columns from all matrices of known yeast transcription factors, as well as multiple sequence alignments of intergenic regions of four yeast species, into 15 subspaces via supervised learning ([36], supplementary material). This mapping of profile columns to characters allows PhyloNet to take advantage of the fast inexact matching scheme used by the family of BLAST algorithms. Each profile's degenerate sequence is in turn taken to be the *query* and is compared to all other profiles' sequences, known as the *database*. The hits resulting from the lookup are returned to stage 2, the seed extension stage.

$$\begin{bmatrix} n_A \\ n_C \\ n_G \\ n_T \end{bmatrix} \rightarrow \begin{bmatrix} f_A \\ f_C \\ f_G \\ f_T \end{bmatrix} \rightarrow \left\{ \begin{array}{ccccccc} & A & C & G & T & & \\ & a & c & g & t & & \\ R & Y & S & W & K & M & n \end{array} \right\}$$

Figure 1.8: Mapping of profile columns to degenerate sequence.

Seed Extension

Once a seed match has been found, stage 2 attempts to extend these matches into *High-scoring Segment Pairs* (HSPs). Starting at the location of the seed hit, PhyloNet compares profile columns to the left and to the right of the seed hit. Comparison between profile columns is done using the ALLR score. The seed hit is extended in both directions until the running score is less than some fixed value below the maximum score (Figure 1.9). This fixed value that decides when an extension terminates is known as the *x-drop* and is the heuristic used in NCBI BLAST for extending seed hits. The boundaries of the extension are taken to be the location of the maximum scores of the left and right extensions, and the score of the extension is simply the sum of the maximum scores in both directions. If the score is above some threshold, then the extension is deemed an HSP and is passed to stage 3, the clustering stage.

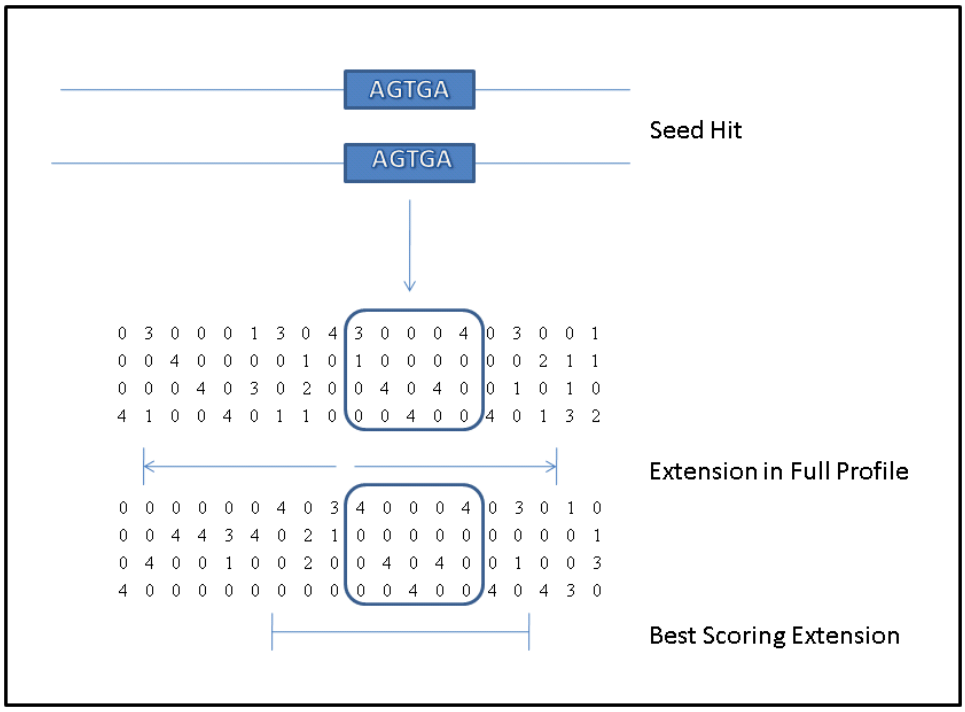

Figure 1.9: Extension of Seed Hits into HSPs.

HSP Clustering

The seed extensions from stage 2 that score high enough to become High-scoring Segment Pairs (HSPs) are passed to the HSP clustering stage. Each HSP is mapped back to the original query profile. An overlap graph is formed of the HSPs, where the nodes are the HSPs themselves and edges are drawn between HSPs that overlap each other on the original query profile. Clustering is done using a general clique finding algorithm on the overlap graph (Figure 1.10).

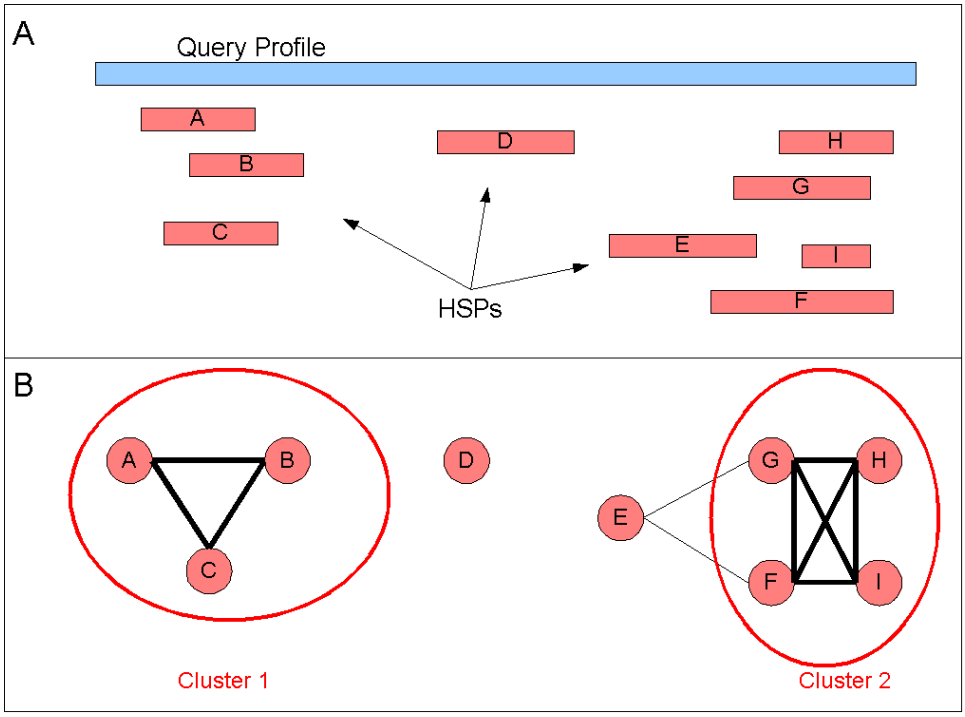

Figure 1.10: Clustering HSPs into motifs. Cliques in the graph (bold lines) represent clusters. (A) HSPs are mapped onto the query profile. (B) HSPs that overlap are connected.

Clusters are converted to motifs using a greedy heuristic approach derived from the WConsensus motif finder [17]. The motifs, which consist of a consensus sequence and a list of supporting instances, are reported as output of PhyloNet. Figure 1.11 shows an example of the consensus sequence of a short yeast motif, along with three of its instances drawn from promoter alignments of four budding yeast species.

```
MATRIX 3:
number of promoters: 3
number of sequences: 12
width = 12
tollr = 27.47
information content = 17.900 bits
ln(p-value) = -3.39865; p-value = 0.0334183

A |    0    0    0    0    0    0    2    1    0    0    0    0
C |    0    0    0   12   12    0    0    6    2   11    0    0
G |    0    0   11    0    0    0   10    1    2    0    0    1
T |   12   12    1    0    0   12    0    4    8    1   12   11

Consensus: TTGCCTGctCTT
   1|       iYAL061W_YAL060W.aln_Cer:     -1/221    TTGCCTATTCTT
   2|       iYAL061W_YAL060W.aln_Par:     -1/214    TTGCCTGCTCTT
   3|       iYAL061W_YAL060W.aln_Mik:     -1/249    TTTCCTGTCCTT
   4|       iYAL061W_YAL060W.aln_Bay:     -1/258    TTGCCTGGCCTT

   5|       iYML110C_YML109W.aln_Cer:   2713/-514   TTGCCTGCTCTT
   6|       iYML110C_YML109W.aln_Par:   2713/-518   TTGCCTATTCTT
   7|       iYML110C_YML109W.aln_Mik:   2713/-513   TTGCCTGTTCTG
   8|       iYML110C_YML109W.aln_Bay:   2713/-519   TTGCCTGCGCTT

   9|       iYNL029C_YNL027W.aln_Cer:   3016/-104   TTGCCTGCTCTT
  10|       iYNL029C_YNL027W.aln_Par:   3016/-105   TTGCCTGCTCTT
  11|       iYNL029C_YNL027W.aln_Mik:   3016/-107   TTGCCTGCTCTT
  12|       iYNL029C_YNL027W.aln_Bay:   3016/-111   TTGCCTGAGTTT
```

Figure 1.11: Consensus sequence of a yeast motif with instances drawn from promoter alignments of four budding yeast species.

1.4 Utility of Streaming Architectures

Our goal of this thesis work is to improve the quality of the results and scalability of PhyloNet. In order to be useful for large genomes such as human, the PhyloNet algorithm must be able to scale well with ever-increasing datasets. PhyloNet scales quadratically with its input size. It can process an all-vs-all comparison of the noncoding sequences of the yeast genome, but this analysis takes over five CPU days on a modern AMD Opteron workstation. The noncoding regions of higher eukaryotic genomes can contain tens to hundreds of times more sequence than that of yeast. Thus, further efforts must be made to accelerate PhyloNet.

One approach employed in accelerating biosequence comparison algorithms is developing streaming designs of these algorithms. Streaming designs are useful when the algorithm involves simple, repetitive calculations that can be pipelined. Examples of architectures that support the streaming model are chip multiprocessors, graphics

processing units (GPUs), and field-programmable gate arrays (FPGAs). We target the FPGA for our streaming design of PhyloNet.

The speedup over a traditional CPU that can be achieved by implementing a computation as a circuit on an FPGA has been reported many times in the technical literature. The inherent spatial parallelism of the logic resources on the FPGA allows for considerable compute throughput even at a sub-500 MHz clock rate. Another option for increasing throughput by exploiting parallelism is to run these algorithms on a workstation cluster. However, clusters typically have high acquisition, maintenance, and energy costs when compared to single-node solutions. One study compared the performance of image processing application programs executing in hardware on a Xilinx Virtex E2000 FPGA to that on three general-purpose processor platforms: MIPS, Pentium 111 and VLIW [12]. The study showed that the FPGA implementations are one to two orders of magnitude faster than the CPUs, even after accounting for the clock frequency differences. The authors of the study attribute this speed-up to the following factors:

1. iteration-level parallelism on FPGAs, which is limited only by device area and the available I/O and memory bandwidths.

2. better instruction efficiency in FPGAs, which is defined as the number of arithmetic and logic operations executed per unit of data.

3. the ability to stream data from memory or I/O to the datapath on the FPGA.

Several BLAST-family algorithms have been accelerated by orders of magnitudes using FPGAs. Mercury BLASTN [22] is a high-throughput and high-sensitivity BLASTN accelerator for comparing DNA-to-DNA sequences. Mercury BLASTP [21] is an FPGA implementation of BLASTP, the most popular tool to perform comparative sequence analysis of protein sequences. DeCypherBLAST [33] is a commercial product to accelerate BLASTP, utilizing FPGA-based processing engines attached to high-end CPUs. TreeBLASTP [16] is an FPGA-based accelerator for BLASTP-like computations. PhyloNet utilizes similar techniques to those in BLAST, and we thus leverage these ideas to create an FPGA implementation of PhyloNet.

1.5 Contributions to Phylonet

In this thesis work, we were able to increase both the quality of results and the speed of the PhyloNet algorithm. Through modifications of the PhyloNet algorithm, including the use of gapped alignments as input into PhyloNet, we were able to identify an average of 60% more known transcription factor binding sites in a whole-genome analysis of yeast. Utilizing a faster ALLR scoring calculation scheme and a more efficient clustering algorithm, we were able to decrease the running time of PhyloNet by 20-fold in software alone. We also devised an approach to accelerate the seed-matching and HSP generation stages of PhyloNet in hardware and got an expected speedup of close to 30-fold versus our improved software version of PhyloNet. The author's specific contributions to increase the performance of the PhyloNet algorithm are summarized below.

1. Contributions to the software analysis
   (a) A metric was devised to quantify the sensitivity and specificity of the motifs produced by Phylonet versus a gold standard data set.
   (b) Yeast and fly datasets were analyzed to determine the sensitivity of PhyloNet and to empirically locate the performance bottlenecks.
   (c) A set of filters was created to discard from the output motifs that are likely to be repetitive or low-complexity DNA.
   (d) Support for gapped profiles was introduced, and its performance was measured.
2. Contributions to the hardware design
   (a) A strategy was developed to implement stage 1 seed matching in hardware. Its performance compared to a software implementation was analyzed.
   (b) A design of the ALLR score calculation amenable for use on the FPGA was created, and its specificity and sensitivity were compared to the software scoring calculation.
   (c) A strategy was developed to implement stage 2 HSP generation in hardware. Its performance was compared to the software implementation.

(d) End-to-end throughput calculation of PhyloNet on an FPGA was studied and compared to the software performance.

The rest of the thesis is organized as follows. Chapter 2 discusses the software improvements made to PhyloNet. Chapter 3 describes the design of an accelerated seed generation stage in hardware. Chapter 4 continues the hardware design discussion with a look at the acceleration of HSP generation and concludes with end-to-end performance gains. Chapter 5 makes some concluding remarks and discusses possible future work.

Chapter 2

Software Improvements to PhyloNet

This chapter describes the software contributions we made to PhyloNet 2, the most current version of PhyloNet available. We offer a new version, PhyloNet 3, that increases both the speed and sensitivity of the program while preserving the basic structure of the computational pipeline. We address these changes throughout the chapter. We also present a way to quantity the result quality of PhyloNet and compare the performance of PhyloNet 3 to that of PhyloNet 2.

2.1 Software Improvements

In this section, we discuss the improvements made to PhyloNet 2, both in terms of its running time and its ability to discover TF binding sites within the genome. The principal algorithmic change from PhyloNet 2 to PhyloNet 3 is a complete redesign of stage 3 of the pipeline, which is the clustering of HSPs. The main change that improved sensitivity of the results was support for gapped alignments as input into PhyloNet, rather than restricting inputs to ungapped alignments as in PhyloNet 2. We discuss these in turn and also allude to some of the less drastic changes that were made.

2.1.1 Improved Clustering

In the clustering phase of PhyloNet, High Scoring Segment Pairs (HSPs) are grouped according to where they align with the profile of interest (query profile). These groups are expanded into motifs, the final output of PhyloNet. PhyloNet 2 first forms an overlap graph on the set of HSPs, where each HSP is a node in the graph, and an edge is drawn between two HSPs if they overlap each other with respect to the query profile. PhyloNet 2 uses a general clique-finding algorithm on this graph to cluster the HSPs into groups. Next, the motif is formed using a greedy heuristic by progressively adding instances to the motif and trimming regions that do not increase the score, so that the total ALLR score is maximized. The length of the motif is determined automatically by this process and does not require the motif length to be fixed *a priori*.

PhyloNet 3 uses a more efficient clustering method that offers stronger performance guarantees. We first observe that connecting HSPs with an edge when the HSPs overlap on the query profile produces an *interval graph*. All maximal cliques in an interval graph can be found in time linear in the number of HSPs and enumerated in time proportional to their total sizes [13]. Finding maximal cliques essentially involves sweeping a line across the interval graph (Figure 2.1) and counting the number of HSPs that intersect the line. Local maxima in this count correspond to maximal cliques. PhyloNet 3 uses interval clique finding to guarantee both maximality and exhaustive enumeration of clusters, with much better scalability than general clique finding. To avoid building clusters from HSPs that overlap by very little (e.g. a single base), it is desirable to enforce a minimum overlap of k positions to create an edge in the overlap graph. This criterion can be enforced by reducing each interval's right endpoint by $k - 1$ prior to clique finding.

To simplify conversion of clusters to motifs, PhyloNet 3 replaces the earlier version's greedy heuristic with the following enumerative algorithm. For each HSP H_j in the cluster, let P_c and P_j be the profiles that it aligns, and let $[\ell_j, r_j]$ and $[\ell'_j, r'_j]$ be the intervals that it aligns from P_c and P_j, respectively. Let $d_j = \ell'_j - \ell_j$ be the *diagonal* of H_j, that is, the offset of its starting indices in P_c and P_j.

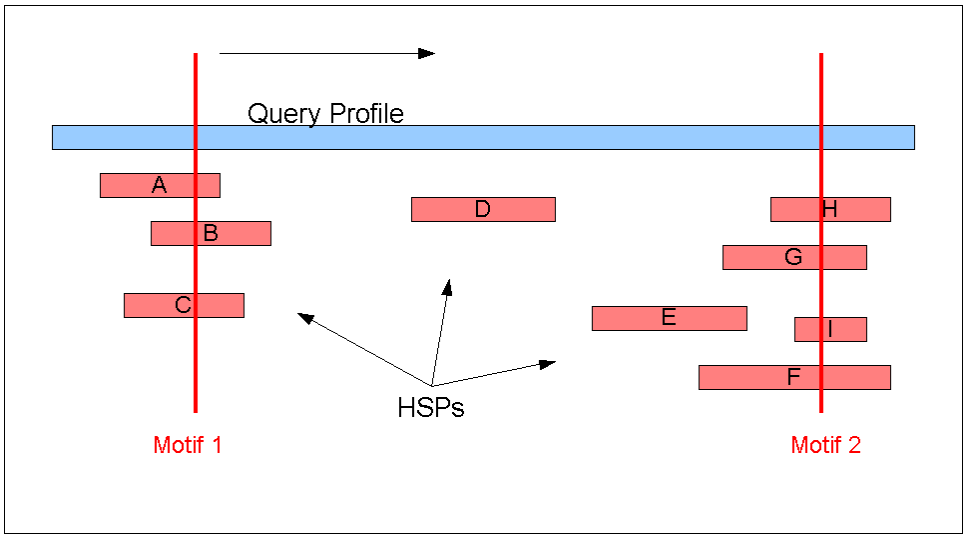

Figure 2.1: Clustering HSPs using an Interval Graph.

Suppose that the HSPs in a cluster have $\min_j \ell_j = L$, and $\max_j r_j = R$. For each left endpoint ℓ and right endpoint r, $L \leq \ell \leq r \leq R$, we find the best-scoring motif whose instance on P_c is the interval $[\ell, r]$. The instance corresponding to HSP H_j is then $[\ell + d_j, r + d_j]$. (If this instance runs off either end of P_j, then it is discarded for this choice of endpoints.) We then discard any instance whose ALLR score versus P_c is negative and retain the total score $s_{\ell,r}$ of the remaining instances. The motif with the highest total ALLR score for the cluster is the one with endpoints $\text{argmax}_{\ell,r} s_{\ell r}$ in P_c.

Our enumerative algorithm requires time $\Theta(m^2 n)$, where n is the number of HSPs in the cluster and $m = R - L + 1$. However, the ALLR scores for each column of the potential alignment between each P_j and P_c can be precomputed and stored in total time $\Theta(mn)$, so the constant factor associated with the quadratic cost in m is small in practice, consisting mostly of addition and table lookup. We also note that when the goal is instead to minimize the statistical p-value defined in [36] for the motif, the motif with best p-value for a cluster can still be found in time $\Theta(m^2 n \log n)$.

2.1.2 Support for Gapped Profiles

PhyloNet 2 takes as input multiple suboptimal ungapped alignments. Since real sequences can have insertions and deletions of bases, a single good ungapped alignment cannot in general be obtained. Thus, multiple ungapped alignments are generated with the hope that a real motif will be correctly positioned in one of these alignments. This scheme has the intuitive disadvantages that it increases the size of the data by including many alternative alignments, may not cover all of the original input sequences, and may contain many junk alignments. A gapped alignment, in contrast, can align the entire sequence at once and account for the insertions and deletions of bases that occur as species diverge from a common ancestor. Therefore, we introduce into PhyloNet 3 support for gapped alignments. Because the input now contains only a single gapped profile, rather than many profiles, per promoter region, the work of scanning the database and generating HSPs is substantially reduced, and the program runs faster. As we will see, PhyloNet 3 is also able to find more known binding sites than PhyloNet 2.

As an example of the advantage of using gapped alignments, the sequence 'CGTGTGAAGTGAT' is a binding site for the protein *Abf1* in the intergenic region of *S. cerevisiae* between the genes *YIL073C* and *YIL072W*. Results of aligning this intergenic region with the orthologous regions in three related yeast species using a gapped approach and an ungapped approach are given in Table 2.1. The region of the alignment containing the binding site is shown in the table. The multiple ungapped alignments were generated using WConsensus [17]. In total, 100 ungapped alignments were generated, 16 of which contained the binding site for Abf1 in the *S. cerevisiae* sequence. A partial, representative list of alignments containing the binding site is given in the table. Notice that the one gapped alignment shows more conservation across the sequences than any of the ungapped alignments. If a binding site lacks conservation in the other sequences, then it may not be discovered by PhyloNet. In fact, PhyloNet 3 correctly identifies this binding site as a motif, while PhyloNet 2 fails to do so.

PhyloNet 3 accepts gapped multiple alignments in Multiple Alignment Format (MAF) [28]. The MAF format stores a series of multiple alignments in a format that is human readable and straightforward to parse. MAF files include gapped alignments, the

Table 2.1: Gapped Alignment Versus Multiple Ungapped Alignments

Gapped Alignment	Multiple Ungapped Alignments
ctaCGTGTGAAGTGATa-t tt-CGTGAGAAGTGATact gttCGTGATACGTGATgct -taCGTAACAAGTGATatc	ctaCGTGTGAAGTGATata ataCACTTGTGATGTTatc gtaAATTTGCGAAGTTatc ttaCGTAACAAGTGATatc
	ctaCGTGTGAAGTGATata cttATTTTATTATCTActa cttATTTTATTATCTActg tacGTAACAAGTGATAtcc
	ctaCGTGTGAAGTGATata ccaTGTTGCAGTTTCTtat ccaTGTTGCAGTTTCTtta gaaAGGGGCAACTCTTtct

scores of the alignments, and the source genomes' positions, strand directions, and lengths of the aligned sequences.

In order to support gapped alignments, the ALLR computation must be modified to account for the presence of gaps. We considered two approaches. One approach is to treat gaps as unknown residues. An unknown residue is one in which there is no information about its identity. This introduces a new character to the possible characters in an alignment column, along with the already present 'A', 'C', 'G', and 'T' characters. The ALLR calculation would need to be only trivially modified to include this possiblity. However, a gap in the alignment really indicates that no homologous residue is present in the sequence, not that the residue is unknown. Thus, despite being straightforward to implement, this approach is not well-justified.

The second approach is simply to ignore gaps when calculating the counts of each residue in a profile column. The resulting total base count may be less than the total number of sequences, and profile columns in general may be of different sizes. Thus, we use the more general form of the ALLR calcuation which does not assume equal numbers of bases in each column:

$$ALLR(i,j) = \frac{\sum_{b=A..T} n_{bj} \ln \frac{f_{bi}}{p_b} + \sum_{b=A..T} n_{bi} \ln \frac{f_{bj}}{p_b}}{n_i + n_j}. \tag{2.1}$$

We add *pseudo-counts* to the counts in the profile columns, which has the effect of giving less weight to columns with smaller base counts. If, for example column i is different from the background distribution p but has a small number of bases, then f_{bi} tends to p_b and $\ln \frac{f_{bi}}{p_b}$ tends to 0, making the score smaller than if column i had a relatively large number of bases. In essence, gaps in an alignment do not affect the estimated frequency of a column but they affect the weight given to that estimated frequency.

2.1.3 Other Speed Improvements

The ALLR score computation between profile columns dominates the program's running time. Thus, it is important to make this computation as efficient as possible. Acceleration of this computation is done using precomputed values of logarithms, multiplication and division stored in tables. The only remaining calculations that need to be done are additions. The lookup into the tables depends on the counts of the number of bases in each of the profiles being compared. Both PhyloNet 2 and PhyloNet 3 utilize these lookup tables, but PhyloNet 3 requires one-fifth as many table lookup and additions per ALLR computation. Other changes that impact performance include more efficient memory layout of profiles, faster parsing of profile data, and a faster implementation of seed matching. These improvements were made by Dr. Jeremy Buhler.

2.2 Experimental performance of PhyloNet

In order to compare the results of PhyloNet 2 and PhyloNet 3, we devised a procedure to quantitatively compare the sensitivity and specificity of the programs. We analyzed promoter regions of two datasets, one consisting of multiple *Saccharomyces* (yeast) genomes and the other of multiple *Drosophila* (fly) genomes. We compared the outputs of the two programs to known TF binding sites in *Saccharomyces cerevisiae* and *Drosophila melanogaster*. The next few sections describe in more detail the data sets we used, some practical issues when dealing with real data sets, and the procedure used to measure the effectiveness of each of the PhyloNet programs.

2.2.1 The Data Sets

We analyzed the performance of PhyloNet 2 and PhyloNet 3 using yeast and fruit fly genomes. For the yeast dataset, we obtained both ungapped and gapped alignments. The ungapped data set consisted of 3,761 *S. cerevisiae* intergenic sequences with ortholog counterparts in *S. bayanus*, *S. paradoxus*, and *S. mikatae* (Wang, personal communication). We aligned each group using WConsensus and used these ungapped alignments as input into PhyloNet 2 and PhyloNet 3.

Our second yeast data set consisted of gapped alignments of the *S. cerevisiae* genome with DNA fragments from various related yeast species. These alignments were obtained from the UCSC Genome browser (genome.ucsc.edu). We extracted those alignments that fell within the intergenic regions of *S. cerevisiae* and used these as our promoter regions for a total of 5,769 regions. Thus, each query consisted of all the gapped multiple alignments within a particular intergenic region. These alignments were used when testing PhyloNet 3, which can handle gapped alignments.

To illustrate PhyloNet 3's abilility to scale to larger genomes, we analyzed the fruit fly genome. We obtained 15,347 gapped multiple alignments of 14 insects to *Drosophila melanogaster*. These alignments were created using MULTIZ [5] and consisted of the first 1000 bases upstream of genes from the RefSeq Genes (refGene) track that have annotated coding sequences (CDS) and untranslated regions (UTR). They are provided by the Berkeley Drosophila Genome Project on the UCSC Genome Browser.

2.2.2 Dealing with repetitive and low-complexity regions

Noncoding genomic DNA, including regions of high conservation, contains a large proportion of repetitive and low-complexity sequence. Such sequences are too promiscuous in the genome to act as specific targets for TF binding, but they are still recognized and emitted by motif finders like PhyloNet. To eliminate such "junk" motifs from our output, we devised filtering strategies to identify and discard low-complexity and short tandem repeat sequences. Firstly, all input sequences were preprocessed with Dust [14], a tool that recognizes and masks low-complexity DNA. Repetitive sequence patterns are replaced with an '*', which does not match any base.

Secondly, we implemented a set of filters to discard from the output motifs that are likely to be repetitive or low-complexity DNA. In each experiment, the first genome in the input profiles (*cerevisiae* for yeast, *melanogaster* for fly) is the most complete; we call this genome the *canonical sequence* for the input. In our experiments, we discarded any putative motif for which the query profile's canonical sequence met any of the following criteria:

- The sequence consists of 80% or more of a single base.
  Examples: AAACA, TTTATTT
- The sequence has a run of four or more of the same base.
  Examples: GCTCAAAAAG, AATTTTCGA
- The sequence contains a tandem repeat with period 2, 3, or 4 with at least three copies.
  Examples: TACACACTGG, TACGGACCACCACCGT
- The sequence consists entirely of gaps or masked bases.
  Examples: ——, *******

The same set of filters was applied to input alignments and output motifs for both PhyloNet 2b and PhyloNet 3.

We could filter more aggressively by applying our filters to the consensus sequence of the motif and/or the sequences from motif instances other than that from the query. However, visual inspection of the filtered motifs suggests that our approach misses few if any motifs that would be considered "obviously" low-complexity.

2.2.3 Measuring Performance of PhyloNet

We assessed the quality of PhyloNet's output on yeast by measuring its sensitivity and specificity. Sensitivity was measured by coverage of a set of known yeast TF binding sites, while specificity was estimated from the total amount of *S. cerevisiae* intergenic DNA labeled as being part of a motif.

To measure sensitivity, we compiled a list of experimentally supported TF binding sites from the YEASTRACT database (www.yeastract.com). YEASTRACT is a curated repository that contains regulatory associations between TFs and target genes in *S. cerevisiae* . In total, our list comprised 558 binding site locations for 66 TFs. We measured sensitivity as the fraction of this set of *S. cerevisiae* sites that were labeled by PhyloNet as being part of a motif. A site was considered found by PhyloNet when the experimentally supported site's extent overlapped a reported motif instance by over half of its length.

To assess specificity, we collected the list of *cerevisiae* sites from all motif instances output by PhyloNet. We then measured the fraction of the *cerevisiae* sequence present in the program's input that was covered by a known motif instance. Higher coverage reflects lower specificity. While this measure conflates true motif instances with falsely labeled, non-motif sequences, it provides a fair measure of specificity given that we do not know the complete set of motif instances in the *cerevisiae* genome.

PhyloNet can be manipulated to trade off sensitivity and specificity by setting the statistical p-value threshold for reported motifs. The p-value is a value from 0 to 1 that indicates the probability that a particular motif occurred by chance, with 1 meaning that it certainly occurred by chance. A lower p-value threshold reduces the amount of output and so increases specificity but may decrease sensitivity. We present a unified evaluation of output quality for many different p-values as an ROC curve.

The above evaluation assesses PhyloNet's ability to recognize all instances of a motif. A second, related question is how many distinct motifs PhyloNet can correctly recognize. To address this question, we collected sequence profiles, expressed as position-specific scoring matrices (PSSMs), for known or predicted yeast binding site motifs from the TRANSFAC database and from a set of high-throughput chromatin-IP (CHIP) experiments by Lee et al. We then compared these sets of PSSMs to those for all motifs in PhyloNet's output. The comparison was done using MatAlign (Wang, unpublished), a program to find statistically significant pairwise alignments of PSSMs from two databases. A motif from the TRANSFAC or CHIP data sets was reported as occurring in PhyloNet's output if it matched with a MatAlign conservative p-value of at most 10^{-5}.

2.2.4 Output Quality in Yeast

Figure 2.2 shows ROC curves for three versions of PhyloNet: the previous version (PhyloNet 2b), our new version (PhyloNet 3, using gapped alignments), and a more limited implementation that adds our clustering and other algorithmic improvements but still uses ungapped alignments. The set of p-values tested for each implementation included 10^{-k} for k ranging from 0 to 50.

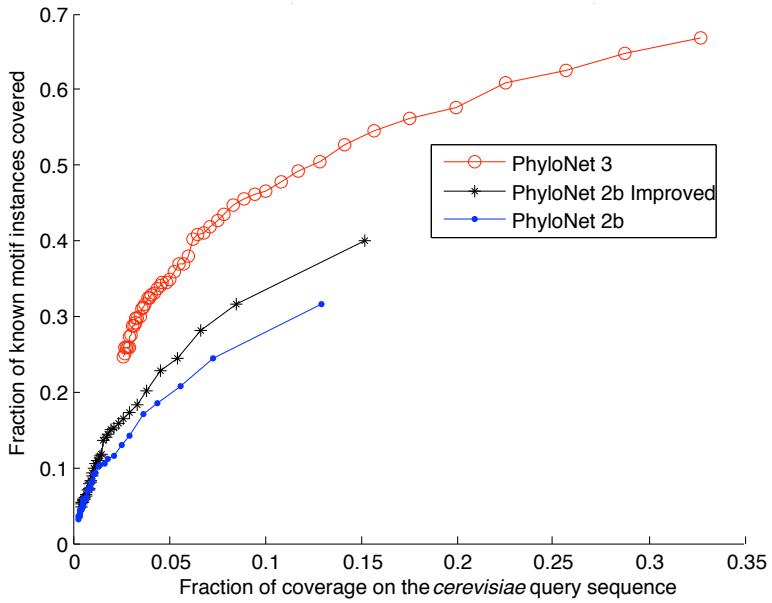

Figure 2.2: Performance of PhyloNet Versions parameterized by p-value cutoff.

We note that for the same p-value, PhyloNet 3 returns motif instances covering more of the input than does PhyloNet 2b. While it may be that these additional instances reflect more actual TF binding sites, they could also reflect an increased false positive rate in the new implementation. Hence, to fairly compare PhyloNet 2b and PhyloNet 3, we report the difference in the two programs' sensitivities at comparable specificity (i.e. coverage of the input), rather than identical p-values. Equivalently, we report the vertical distance between ROC curves at corresponding points on the horizontal axis.

Our core algorithmic improvements to ungapped PhyloNet alone increased the number of known yeast motif instances found by an average of 1.15x for the same specificity. The greatest sensitivity gains are achieved at p-values $> 10^{-5}$, i.e., at lower

levels of specificity. When we also switch to using gapped alignments, PhyloNet 3 finds an average of 1.6x more known motif instances than PhyloNet 2b for the same specificity.

To compare our results to the TRANSFAC and CHIP motif sets, we considered only those results returned by PhyloNet 3 with a p-value of at most 10^{-50}, which yielded motifs covering a total of 2.6% of the input sequence. We found that these motifs included matches to 49/50 TRANSFAC motifs and 90/102 CHIP motifs with a MatAlign p-value of at most 10^{-5}. These results indicate strong absolute sensitivity of PhyloNet 3. However, they are not substantially different from the numbers of matches obtained by PhyloNet 2b (50/50 and 88/102, respectively) for a comparable level of specificity (PhyloNet 2b p-value 10^{-6}, corresponding to 2.5% coverage of the input). Hence, the improved result quality of PhyloNet 3 in yeast is a matter of finding more motif instances, rather than more distinct motifs overall.

2.2.5 Efficiency in Yeast

While the observed improvements in PhyloNet 3's output quality are welcome, it is also important to investigate whether we achieved the goal of improved computational efficiency. We therefore compared the compute time spent by the old and new versions of PhyloNet. Our test system was a cluster of 2.4-GHz AMD Opteron processors; we report the total CPU time summed over all CPUs.

Table 2.2 gives the total running time on the yeast data set for PhyloNet 2b, for our ungapped version with algorithmic improvements, and for the full gapped PhyloNet 3. Our algorithmic changes alone improved the average running time per query by almost a factor of four. Switching to the gapped data set increased this advantage to 20-fold over PhyloNet 2b. PhyloNet 3 achieves improvements in both speed and output quality over the previous version.

Table 2.2: Total CPU Time for *S. cerevisiae* genome

	Execution Time	Speedup
PhyloNet 2	130.2 hours	—
PhyloNet 2 Improved	34.2 hours	3.8x
PhyloNet 3	6.5 hours	20.0x

2.2.6 Scalability to Fruit Fly

As described above, we assessed PhyloNet 3's scalability to a larger eukaryotic genome by running it on proximal promoter sequences from *Drosophila*. This data set spans 14.4x more sequence than our gapped yeast data set, contains 12x more total alignment columns, and has a maximum alignment depth more than thrice that of the yeast alignments.

Table 2.3 gives overall running times for the fruit fly analysis. The total compute time of 7.41 CPU days was 95 times longer than that required for the gapped yeast data set, which is less than the 144x increase expected under the assumption of naive quadratic scaling with the total size of the input profiles. This improvement appears to derive from a lower density of strongly conserved regions in the *Drosophila* input and consequently less time spent in HSP extension per column of the input. We also note that the time to analyze fly was roughly comparable to the 5.54 CPU days required for the older PhyloNet 2b to process the ungapped yeast data.

Table 2.3: Statistics for Fruit Fly Data Set

No. of Profiles	15,347
Avg. Time Per Query	41.74 sec
Total CPU Time	7.41 days

We assessed the output quality of PhyloNet 3 on *Drosophila* by comparing its motifs to a set of 50 fly motifs obtained from TRANSFAC, using the same MatAlign-based protocol as in yeast. We retained only those PhyloNet motifs with p-values of at most 10^{-500}, which cover roughly 5% of the input sequence from *melanogaster*. PhyloNet's

output matched 48 of 50 (96%) TRANSFAC motifs with a MatAlign p-value of at most 10^{-5}.

2.3 Beyond Software PhyloNet

Despite the successful improvements to PhyloNet in software alone, there is still much room to improve the scalability of the PhyloNet algorithm. PhyloNet 3 manages to do an all-vs-all comparison of the promoter regions of fruit fly in over 7 CPU days. Analysis of the entire fruit fly genome would be infeasible in PhyloNet 2. In order for PhyloNet to process mammalian genomes, which in general contain longer and more complex promoter regions, in a reasonable length of time, more needs to be done to improve its processing power. This leads us to implement PhyloNet on dedicated hardware. To improve the runtime of PhyloNet, we leverage the ability to modify its parameters without changing its sensitivity to assist in our hardware design. We also identify the stages that are the software bottlenecks in order to direct our efforts in increasing performance.

2.3.1 Effects of Parameters on the Performance of PhyloNet

We will find in later chapters that the choice of the user-defined parameters of PhyloNet has a large impact on which hardware designs are feasible. The parameters that we examine impacting the design are the word length, the neighborhood threshold, and the query size.

The word length sets the length of the seeds and is 6 by default. The neighborhood threshold sets the size of the neighborhoods for each of the seeds and has a default value of 5. These parameters have an impact on area usage and throughput. Figure 2.3 plots the ROC curves of the yeast dataset for each of the different values of parameters used.

We see that these parameters are robust to at least slight deviations from the default settings. The sensitivity remains unchanged under these particular values of the

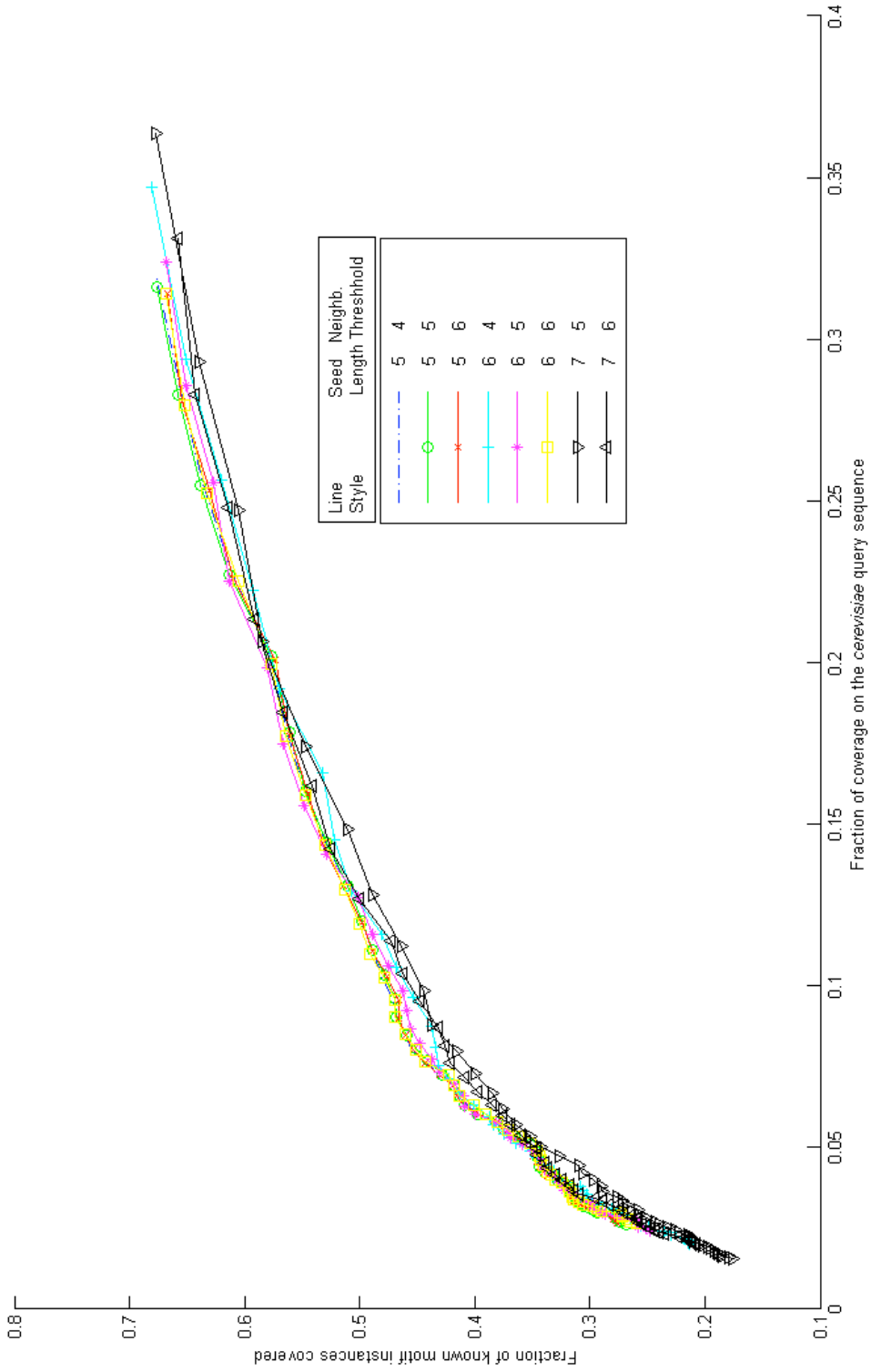

Figure 2.3: Result Quality of PhyloNet for Different User Inputs.

parameters, and so we allow these parameters to take on any of these values as we design our hardware implementation of PhyloNet.

We also examine the effects of processing more than one query simultaneously as oppossed to processing each query individually. This *bin-packing* approach has the advantage of requiring fewer passes over the database and improving the running time of PhyloNet. We double the query size until the total running time no longer improves, examining query sizes of 1000, 2000, 4000, 8000, and 16000. Table 2.4 shows the total running time of PhyloNet on the yeast data set using the different parameters.

PhyloNet took longer to run relative to the other parameters when a seed length of 7 was used. Thus, we do not include a seed length of 7 in our further analysis. The parameters that result in the fastest running time of PhyloNet on the *S. cerevisiae* genome are a word length of 6, neighborhood threshold of 6.0, and a query size of 8,000. We use this as our software baseline to compare against our hardware design. Due to resource constraints, we will see that we are limited in the sizes of our queries, and therefore only examine query sizes up to 4,000 in our hardware design.

2.3.2 Bottlenecks in software version of PhyloNet

We determine the bottlenecks of the PhyloNet pipeline by profiling the running time of PhyloNet on the yeast genome data set. Figure 2.4 shows the proportions of time spent in various stages of PhyloNet for fixed-length profiles of 1000 and the default parameters.

By far, the table setup, seed matching, and seed extension take the most amount of time, with 93% of the time spent on these items. The table setup, which involves creating the seed neighborhoods and loading the table with the query positions, can be done offline and must be done only once for each query profile. This leaves seed matching and seed extension as stages in which to concentrate our improvements. Chapter 3 is devoted to speeding up seed matching (Stage 1) in hardware, while Chapter 4 discusses the speedup of seed extension (Stage 2) in hardware and concludes

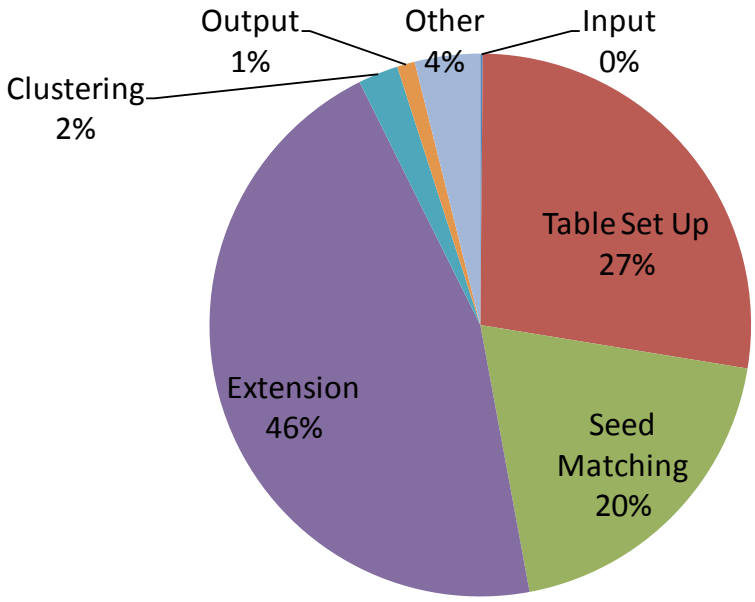

Figure 2.4: Time spent in the various stages of PhyloNet.

with an overall performance analysis and benchmarking of PhyloNet. Chapter 5 concludes our analysis and discusses possible future work.

Table 2.4: Total Running Time of PhyloNet on the Yeast Data Set

Seed Length	Neighborhood Threshold	Query Size	Time (sec)
5	4	1000	5785.5
5	4	2000	6425.0
5	5	1000	3679.9
5	5	2000	3338.3
5	5	4000	3182.4
5	5	8000	3316.3
5	6	1000	2198.0
5	6	2000	1899.5
5	6	4000	1751.7
5	6	8000	1719.1
5	6	16000	1981.1
6	4	1000	6444.9
6	4	2000	6073.0
6	4	4000	5715.5
6	4	8000	5265.6
6	4	16000	5462.4
6	5	1000	3442.9
6	5	2000	3028.1
6	5	4000	2822.9
6	5	8000	2662.6
6	6	16000	2801.6
6	6	1000	2248.1
6	6	2000	1811.8
6	6	4000	1602.0
6	6	8000	1539.9
6	6	16000	1717.3

Chapter 3

Design of an Accelerated Seed Generation Stage

In this chapter we present a design for Stage 1 of PhyloNet amenable to a hardware implementation, specifically on an FPGA. We first describe in detail the Stage 1 process. We then discuss our design of Stage 1 on an FPGA and analyze its throughput. Finally, we compare the speed of the hardware implentation of PhyloNet to that of our improved software implementation.

3.1 Stage 1 of PhyloNet

Recall that alignments of DNA sequences are called *profiles*. Each of the promoter regions of the genome are aligned to orthologous promoter regions of other species. The profiles each in turn act as the *query* profile, which is compared to all other profiles (collectively known as the *database*).

The goal of Stage 1 is to find potential locations for high-scoring regions between two profiles. This is done by searching for short regions of the database profiles that match regions of the query profile to within some threshold. Two regions that are similar will in general have short regions that match very well. This is the same idea used in the BLAST algorithm to compare DNA sequences. It is not guaranteed to find optimal local alignments but has shown good sensitivity in the context of the BLAST family of algorithms.

3.1.1 Seed Matching using Table Lookups

Finding exact matches of a fixed length between two sequences can be done in time linear in their total length using table lookups. Both sequences are divided into short substrings of fixed length known as *words*. Each word is given an address in a lookup table. The locations of the words in one sequence are stored in the table. The table is then queried using the words of the other sequence. Matches between words in the database and query are known as *seeds*. This process is guaranteed to find all exact matches of a fixed length between the two sequences.

This technique can be extended to find all substrings of a fixed length that match to within some threshold for some scoring scheme. This is done by generating a set of neighborhood words for each of the words in the stored sequence. A neighborhood word is a substring the same size as the query word that matches the query word with an alignment score greater than a user-defined threshold. A branch and bound algorithm is used to generate neighborhood words and is guaranteed to generate all words that are similar within the threshold. This is the approach used in BLASTP, an algorithm used to compare protein sequences. It is used in the context of BLASTP because evolutionary and biochemical similarities between amino acids are not well characterized by a match or mismatch of the characters as is the case for DNA sequences as some amino acids have more similar properties to each other than others.

3.1.2 Adapting Table Lookups for PhyloNet

The table lookup approach cannot be directly applied to PhyloNet because profile columns are being compared rather than simply discrete characters of sequences. The number of possible columns grows cubically with the number of sequences in the profile. Even for fairly small numbers of species, it quickly becomes impractical to compute all neighborhood profile words and store them in a lookup table. Thus, the approach used in PhyloNet is to map each of the profile columns to a discrete character that can be used in the table lookup approach. Different profiles can be mapped to the same character, and so this mapping is degenerate. PhyloNet maps the profiles to an alphabet of 15 characters. The mapping function was derived by clustering the columns of matrices of known yeast transcription factors and multiple

alignments of intergenic regions of four yeast species into subspaces via a supervised learning algorithm [36] (supplementary).

After the profiles are mapped to degenerate sequences, they can be divided into seeds and stored in the table just as in the approach used for sequence comparisons. As done for proteins, a neighborhood scheme is employed to capture approximate matches.

3.1.3 Table Lookup Data Structures

There are two data structures typically used to implement a lookup table: direct lookup and hashing. A direct lookup table contains a unique address for all possible entries. Constant-time lookup is guaranteed, but the table may be inefficient in its space usage if the number of actual entries is much less than the number of possible entries. A hash table, on the other hand, does not have a unique address for all possible entries. Thus, two entries may map to the same location, causing a collision that must be resolved. These collisions mean lookups cannot be satisfied in constant time in general. The advantage of a hash table is that it utilizes space more efficiently than a direct lookup table. The software version of PhyloNet 3 uses a direct table approach provided there is enough memory to support it and switches to a hash table approach when space is an issue.

3.2 Hardware Design of Stage 1

The main work done in Stage 1 is seed matching via the lookup table. In this section we explore the implementation of a lookup table on an FPGA. We develop a design that attempts to quickly satisfy each query of the table so that we achieve a high throughput of seed hits out of Stage 1.

3.2.1 Area Estimates of Table for Stage 1

There are two potential places to store the table on an FPGA: block RAM and SRAM. Block RAM is on-chip and so is able to take less time looking up a value than the off-chip SRAM. However, block RAM is much smaller than the SRAM. A single block RAM is at most 36 kilo*bits* while the SRAM on our target FPGA development board is 9.0 mega*bytes*. In order to accommodate storage of the locations of the query seeds and neighborhood seeds, the SRAM is needed.

For a direct lookup table, the number of entries is the number of total possible seeds of length w generated from an alphabet A. This results in a total of $|A|^w$ possible seeds and thus $|A|^w$ entries. Each entry of the SRAM can be 36 bits wide. The total area requirements of the table, then, is 36 $bits * |A|^w$. Table 3.1 shows the amount of area needed for an alphabet size of 15 and seed sizes of 5, 6, and 7.

Table 3.1: Memory requirements of a direct lookup table

Alphabet Size	Word Length	Prim Table Entries	Primary Table Size
15	5	759375	2.71 MB
15	6	11390625	44.67 MB
15	7	170859375	731.16 MB
SRAM Size		2097152	9.0 MB

If the word length is 5 or less, a direct lookup table may be used. If a word length greater than 5 is used, either more SRAMS or a hash table approach could be used.

3.2.2 Occupancy Sizes of Tables

Depending on the parameters, the number of entries in the lookup table and the number of query locations per entry will vary. The number of non-empty entries in the table divided by the total possible entries, or the *occupancy rate*, is a factor in determining the effectiveness of using a hash table. A sparse table is ideal for hashing, while an almost full table will result in many hashing collisions and yield a non-constant lookup time. The design of the table will be affected by the expected

number of query locations returned per table lookup. Both of these statistics are given in Table 3.2.

Table 3.2: Occupancy Sizes of Tables

Word Length	Neigh. Thresh.	Query Size	Occupancy Rate	Positions per Entry
5	4	1000	0.1988	2.17
5	4	2000	0.2647	3.27
5	4	4000	0.3160	5.48
5	5	1000	0.0809	1.76
5	5	2000	0.1170	2.44
5	5	4000	0.1495	3.83
5	6	1000	0.0228	1.47
5	6	2000	0.0360	1.87
5	6	4000	0.0503	2.67
6	4	1000	0.1593	1.51
6	4	2000	0.2454	1.97
6	4	4000	0.3337	2.90
6	5	1000	0.0678	1.33
6	5	2000	0.1125	1.61
6	5	4000	0.1675	2.16
6	6	1000	0.0233	1.22
6	6	2000	0.0409	1.39
6	6	4000	0.0658	1.73

In some cases, the occupancy rate is quite low, and so a hash table approach may be worthwhile to pursue. A hash table would allow for bigger tables than ones having a word size of 5, which is currently a constraint.

3.2.3 Table Design

We implement a design of the table similar to that done for Mercury BLAST [21] This table is composed of two tables, called the primary table and duplicate table (Figure 3.1). Each entry of the primary table stores a duplicate bit indicating if the duplicate table needs to be accessed. The rest of the entry contains either of the following depending on the state of the duplicate bit:

1. if the duplicate bit is set to 0, then the entry contains the positions in the query profile that matches that entry's seed; or

2. if the duplicate bit is set to 1, then the entry contains the count of the number of positions stored and a pointer to the duplicate table. The seed positions are stored in consecutive memory locations in the duplicate table starting at the address of the pointer from the primary table.

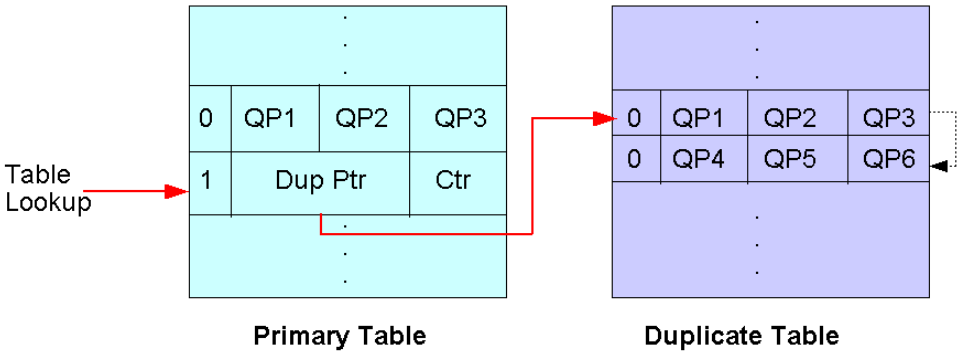

Figure 3.1: Path of Table Lookups

Since a duplicate bit is required for each entry, the number of bits available to store query locations is 35 bits. Each query location is represented with a fixed number of bits. Although using a variable number of bits to represent query locations would make the storage more compact, it would also require more logic, and thus more time, to retrieve query locations from the table. With a fixed-length represntation, the number of query positions that can be stored in a single entry is $\lceil \log_2 N \rceil$, where N is the size of the query profile. Table 3.3 gives the number of query positions possible per entry for various query lengths.

Table 3.3: Query locations per entry for a 36 bit SRAM

Query Length	Bits/Query Pos	Query Pos/Entry	Delta Encoding?
1024	10	3	No
2048	11	3	No
4096	12	3	Yes

Notice that even though a query length of 4000 requires 12 bits per query position, three query positions can be stored in 35 bits instead of 36 bits. One can use *delta*

encoding, which stores the differences between positions rather than the positions themselves [21].

3.3 Stage 1 Performance Model

Here we provide the framework to model the performance of Stage 1 given the table design of the previous section. We use this model to calculate the expected throughput of Stage 1.

The input rate onto the FPGA is the number of database entries that can be streamed in per second and is given by

$$R = I/db, \tag{3.1}$$

where I (in bits per second) is the rate at which data can be streamed to the hardware and db (in bits per position) is the space requirement of each database position. The database position consists of the counts of the number of bases from a single column and the degenerate character that represents that column. Assuming we allow up to 127 species alignments, each of the four counts of a single column needs 7 bits, for a total of 28 bits. The degenerate character can be represented with 4 bits because it comes from an alphabet of size 15. Thus, $db = 32$ bits.

The throughput of Stage 1 is the minimum of the input rate R (in database positions per second) into Stage 1 and the service rate U (in database positions per second) of Stage 1.

$$Throughput = min(R, U). \tag{3.2}$$

The service rate can be calculated as

$$U = f * \frac{1}{A} * M, \tag{3.3}$$

where f is the frequency of the SRAM, A is the expected number of clocks per database entry, and M is the number of SRAMS used in parallel. To calculate the expected number of clocks per database entry, we add up the number of clocks that each query takes, given that only 3 locations can be returned per clock, and divide

by the total number of queries. This calculation can be expressed as

$$A = \frac{1}{N} * \sum_i g(n_i), \tag{3.4}$$

where N is the number of database entries, n_i is the number of positions returned for database entry i, and $g(n_i)$ is the number of clocks needed to return n_i query locations. The number of clocks needed to return n_i query locations depends on our table design, where at most 3 query locations can be stored per table entry. Thus, queries that return 0 through 3 positions can be satisfied in one clock cycle. Queries that return 4 through 6 positions take one clock cycle to go to the duplicate table plus two more clock cycles to return all the hits for a total of three clock cycles. Queries containing 7 through 9 positions take four clock cycles, 10 through 12 positions take 5 clock cycles, and so on. The formula for $g(n_i)$ is given as

$$g(n_i) = \begin{cases} 1 & \text{if } 0 \leq n_i \leq 3 \\ \lfloor \frac{n_i - 1}{3} \rfloor + 2 & \text{if } n_i > 3 \end{cases} \tag{3.5}$$

The average amount of output generated by Stage 1 per clock cycle can be calculated by summing up the total positions returned of all database entries and dividing by the total number of clocks needed:

$$hits\_per\_clock = \frac{\sum_i i * n_i}{\sum_i g(n_i) * n_i}. \tag{3.6}$$

3.4 Performance Estimates for Stage 1

Given the performance model illustrated in the previous section, we estimate the performance of hardware Stage 1 on the yeast data set and compare it to software Stage 1. Table 3.4 gives some of the specifications for the FPGA needed in our performance model.

The input rate is 700 MB/s (Table 3.4), and the number of positions that can be streamed in per second is 700 MB/s divided by the database position size of 32 bits

Table 3.4: Specifications of FPGA

Input Rate	700 MB/s
Number of SRAMS	1
Frequency of SRAM	200 MHz

(Equation 3.1). The number of database positions that can be streamed in every second is then 183.5 million.

We empirically calculate the number of positions returned for each table lookup using the yeast dataset on various parameters. The distributions of the positions returned from the table lookups are given in Figure 3.2.

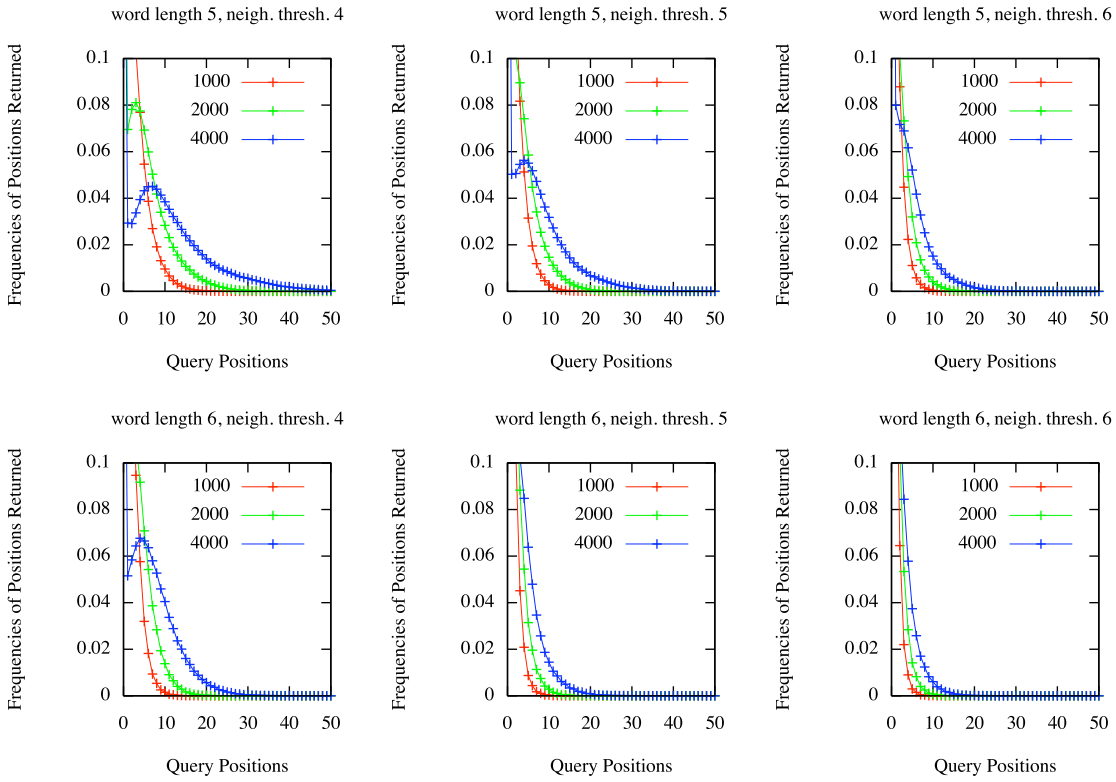

Figure 3.2: Distributions of Positions Returned from Table Lookups for Query Sizes of 1000, 2000, and 4000.

Each subplot shows the distribution of a particular word length and neighborhood threshold for query sizes of 1000, 2000, and 4000. As the query size increases, the

number of positions returned increases, shifting the distributions to the right. The distributions have peaks located either at zero or near zero with long tails. This indicates that most lookups only return a few positions, although some may return upwards of tens of positions.

Using these distributions, we can calculate the average number of clock cycles per table query based on Equation 3.4. Table 3.5 lists the average clock cycles needed per database entry for the various parameters on the yeast data set. The average number of clock cycles increases when query size increases and decreases when the neighborhood threshold increases.

Table 3.5: Average Clock Cycles Needed Per Database Entry for Various Parameters

Seed Length	Neighborhood Threshold	Query Size	Average Clks/DB
5	4	1000	1.6574
5	4	2000	2.6075
5	4	4000	4.3340
5	5	1000	1.3068
5	5	2000	1.8725
5	5	4000	2.9608
5	6	1000	1.0995
5	6	2000	1.3464
5	6	4000	1.9160
6	4	1000	1.2806
6	4	2000	1.8759
6	4	4000	3.0292
6	5	1000	1.0797
6	5	2000	1.3106
6	5	4000	1.9072
6	6	1000	1.0299
6	6	2000	1.1315
6	6	4000	1.4405

The rate at which queries can be made against the table also depends on the number of SRAMS being utilized (Equation 3.3). Even though we could potentially use more SRAMS, we used just one because we determine that this is sufficient to keep up with Stage 2 of the pipeline.

We can also calculate the amount of output produced per clock cycle based on Equation 3.6. This is given in Table 3.6. Four configurations produce less than one position per clock cycle. One out of these four has a seed length of 5. The amount of output from Stage 1 impacts our design of Stage 2, which we discuss in Chapter 4.

Table 3.6: Output of Stage 1

Seed Length	Neighborhood Threshold	Query Size	Positions Out Per Clock
5	4	1000	1.4286
5	4	2000	1.8202
5	4	4000	2.1929
5	5	1000	1.0765
5	5	2000	1.5063
5	5	4000	1.9076
5	6	1000	0.6402
5	6	2000	1.0483
5	6	4000	1.4751
6	4	1000	1.1317
6	4	2000	1.5494
6	4	4000	1.9217
6	5	1000	0.6862
6	5	2000	1.1339
6	5	4000	1.5606
6	6	1000	0.4221
6	6	2000	0.7707
6	6	4000	1.2125

Based on the number of clocks per database entry, we calculate the expected time required for Stage 1. This is simply the average clocks per database given in Table 3.4 multiplied by the size of the database divided by the clock frequency. The results are given in Table 3.7.

3.5 Conclusion

We compute the expected running time of our hardware design of Stage 1 for all of the possible parameters. The best hardware performance came when using a word length

Table 3.7: Stage 1 Hardware versus Software Time

Seed Length	Neighborhood Threshold	Query Size	Hardware Time (sec)
5	4	1000	31.4
5	4	2000	24.7
5	4	4000	20.4
5	5	1000	24.7
5	5	2000	17.7
5	5	4000	14.0
5	6	1000	20.8
5	6	2000	12.7
5	6	4000	9.0
6	4	1000	24.3
6	4	2000	17.7
6	4	4000	14.3
6	5	1000	20.4
6	5	2000	12.4
6	5	4000	9.0
6	6	1000	19.5
6	6	2000	10.7
6	6	4000	6.8

of 6, neighborhood threshold of 6, and query size of 4000, spending 6.8 seconds total. The fastest software version of PhyloNet (word size of 6, neighborhood threshold of 6, and query size of 8000) spends 141.5 seconds in the seed matching stage. If we use a direct lookup table stored on a single SRAM, we need to use a seed length of 5, which means our best hardware improvement uses word length of 5, neighborhood threshold of 6, and query size of 4000, resulting in a time of 9.0 seconds.

In general, both software and hardware run faster when the query sizes are larger, the neighborhood thresholds stricter, and the seed lengths longer. This makes sense because longer seed lengths and longer query sizes mean less lookups, and higher neighborhood thresholds mean less positions returned. We also found that the number of positions output by Stage 1 ranged from 0.4 to 2.2 positions per clock. A low neighborhood threshold, a short seed length, and a long query length result in more words in the table, meaning more seed hits returned on the average.

Chapter 4

Design of an Accelerated Seed Extension Stage

In this chapter, we discuss Stage 2 of PhyloNet, which involves the extension of seeds into high scoring segment pairs (HSPs). We develop a design for implementing this stage on an FPGA and compare the hardware design performance to the software design. We conclude with end-to-end speed improvements of PhyloNet.

4.1 Stage 2 of PhyloNet: Seed Extension

In stage 1 of PhyloNet, database seeds are compared against query seeds. Matches between the database seeds and query seeds are returned from stage 1 and are then input to stage 2. In stage 2, the seed hits are extended into high scoring segment pairs (HSPs). These HSPs are then clustered into motifs. Extension is done on the original profiles using the full ALLR calculation. Extension is performed by extending the seed hit in both directions. A running score and a maximum score are recorded for each direction. Extension terminates when the running score drops below the maximum score minus a user-defined threshold known as the *x-drop* (Figure 4.1). If the maximum score is above some threshold, it is passed to stage 3 for clustering.

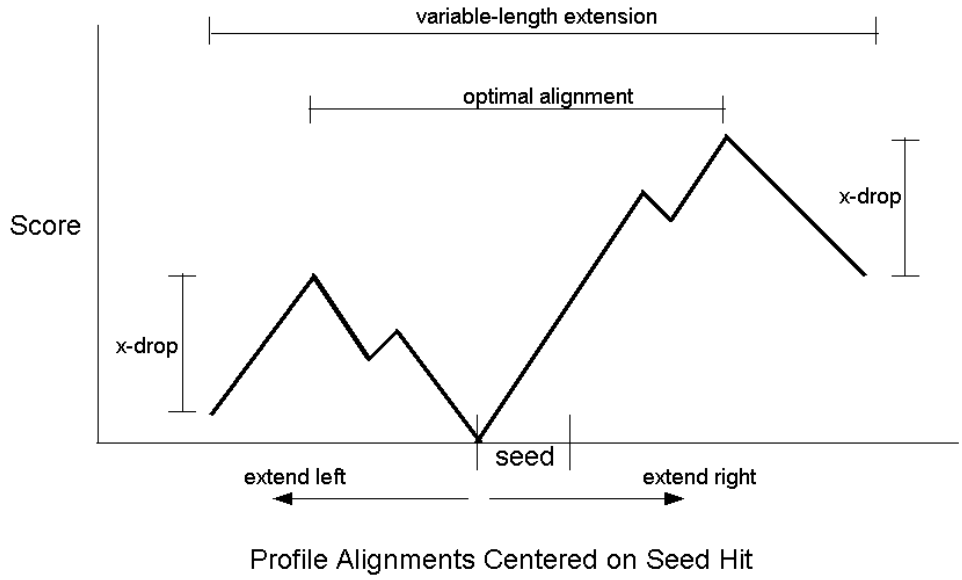

Figure 4.1: Variable-Length Extension in Software

4.2 Hardware Design of Stage 2

The stage 2 design of PhyloNet in software needs to be adapted to work in the context of a hardware design. The issues that make adapting stage 2 into hardware nontrivial are as follows:

1. Extension in software can be of variable length. It is unknown beforehand how far a seed hit must be extended.
2. The ALLR calculation involves the computation of logarithms and divisions, which are costly in hardware.
3. Software PhyloNet uses floating point arithmitic to compute the ALLR score between profile columns.
4. There are limitations in the sizes, quantities and numbers of accesses per clock cycle to tables stored on the FPGA.

We address each of these in this section as we devise a hardware design for Stage 2 of PhloNet.

4.2.1 Fixed Window Extension

Extensions in software are of variable lengths. Extensions terminate when the current score drops below the *x-drop* cutoff. However, this approach does not lend itself well to a hardware design. Instead, we extend the seed hits over a fixed window in hardware (Figure 4.2). The seed gets extended over the entire window, and the best scoring alignment is recorded.

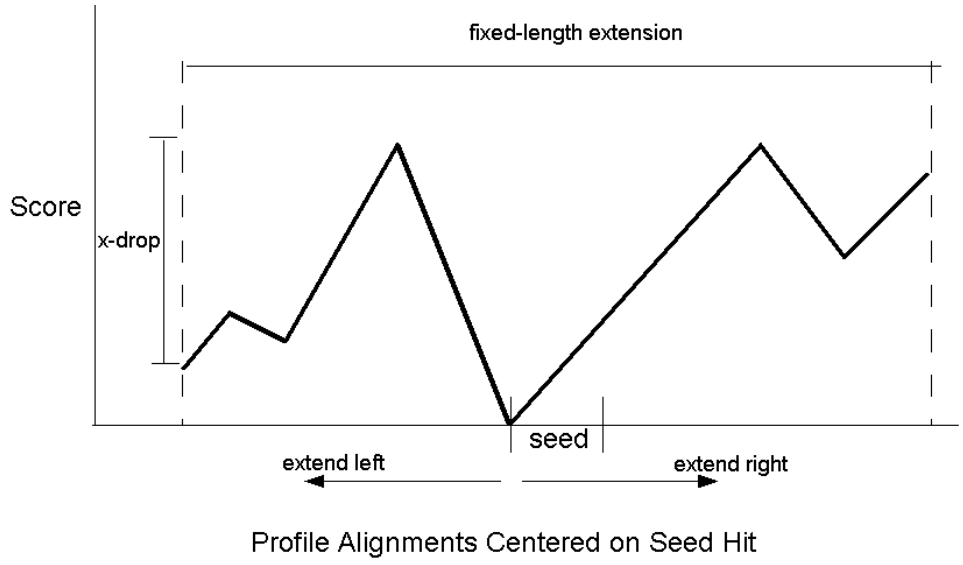

Figure 4.2: Fixed-Length Extension in Hardware

If an extension hits the edge of the window, but the current score is still within the *x-drop* of the maximum score and meets a certain threshold score T, then this extension is passed to software for the usual software extension. An extension may be deemed an HSP even if it does not get passed to the software extension, as long as its score in hardware is above the minimum score for an HSP. Algorithm 1 displays the pseudo-code for this process. Its input is the query profile (*query*), database profile(*database*), positions of the seed hit in the query and database (s_q and s_d), and the window length W. It returns true if extension in software is required. Otherwise, it returns the score of the best extension (HSP_Score) and the endpoints of the best extension (L_{max_index} and R_{max_index}).

Algorithm 1 Seed Hit Extension

1: **procedure** EXTENSION($s_q, s_d, query, database$,W)
2: $HSP\_Score \leftarrow 0$
3: EXTENDLEFT($s_q, s_d, query, database, W/2$)
4: EXTENDRIGHT($s_q, s_d, query, database, W/2$)
5: **if** $L_{current\_index} = W/2$ and $L_{current\_score} > T$ **then**
6: **return** TRUE
7: **else**
8: $HSP\_Score \leftarrow HSP\_Score + L_{max\_score}$
9: **end if**
10: **if** $R_{current\_index} = W/2$ and $R_{current\_score} > T$ **then**
11: **return** TRUE
12: **else**
13: $HSP\_Score \leftarrow ans + R_{max\_score}$
14: **end if**
15: **return** FALSE, HSP_Score, L_{max_index}, R_{max_index}
16: **end procedure**

The fixed window extension in hardware is implemented as two systolic arrays, extending $W/2$ positions to the left and to the right of the seed hit, where W is the total window length. In total, W pipelined, ALLR calculations are processed in parallel. The scores of the ALLR calculators from each extension are summed and the maximum scores (L_{max_score} and R_{max_score}) and their locations (L_{max_index} and R_{max_index}) are returned, along with the scores and indices at the termination of the extensions ($L_{current\_score}$, $R_{current\_score}$, $L_{current\_index}$, and $R_{current\_index}$). Algorithm 2 shows this process for extension in both the left and the right directions.

Another possible approach to doing the extension is to use a dynamic programming algorithm that finds the maximal-scoring extension containing the seed and uses just one systolic array of length W. Instead of extending in both the left and the right direction starting at the seed, it beings at the left-most end of the window and extends to the right-most end. This approach was used in [23]. The advantage of the approach described in the prior paragraph is that it more closely mimics the software extension, and the left and right extensions can be done in parallel.

Algorithm 2 Directional Extension

```
1: procedure ExtendLeft(s_q, s_d, query, database, W)
2: L_max_score ← L_current_score ← 0
3: L_max_index ← L_current_index ← 0
4: for i = 1 to i = W do
5:     L_current_score ← L_current_score + ALLR(query[s_q − i], database[s_d − i])
6:     L_current_index ← i
7:     if L_current_score > L_max_score then
8:         L_max_score ← L_current_score
9:         L_max_index ← L_current_index
10:    end if
11:    if L_current_score < L_max_score − x_drop then
12:        return L_max_score, L_max_index, L_current_score, L_current_index
13:    end if
14: end for
15: return L_max_score, L_max_index, L_current_score, L_current_index
16: end procedure
17: procedure ExtendRight(s_q, s_d, query, database, W)
18: R_max_score ← R_current_score ← 0
19: R_max_index ← R_current_index ← 0
20: for i = 1 to i = W do
21:    R_current_score ← R_current_score + ALLR(query[s_q + i], database[s_d + i])
22:    R_current_index ← i
23:    if R_current_score > R_max_score then
24:        R_max_score ← R_current_score
25:        R_max_index ← R_current_index
26:    end if
27:    if R_current_score < R_max_score − x_drop then
28:        return R_max_score, R_max_index, R_current_score, R_current_index
29:    end if
30: end for
31: return R_max_score, R_max_index, R_current_score, R_current_index
32: end procedure
```

This hardware extension does a good job as a pre-filter into software extension. Table 4.1 shows the fraction of the extensions that avoid being passed to software as a function of the window size. Over 99% of the extensions can be satisfied in hardware without passing them to software. We are limited in the size of the window by the

area constraints of placing multiple ALLR calculations on the hardware. This will be explained in detail in the next two sections. Even with this high filtering rate, we will see that even when using the largest window size that fits onto the FPGA, the processing done in software takes more time than the processing done in hardware. Thus, we would still like to reduce the number of extensions being passed to software or increase our hardware capacity to further reduce the time spent in Stage 2.

Table 4.1: Quality of Fixed Window Filter

Window Size	Percent Filtered
20	99.18%
24	99.51%
30	99.68%

4.2.2 ALLR calculation in Hardware

The full ALLR calculation is given as

$$ALLR(i,j) = \frac{\sum_{b=A..T} n_{bj} \ln \frac{f_{bi}}{p_b} + \sum_{b=A..T} n_{bi} \ln \frac{f_{bj}}{p_b}}{\sum_{b=A..T} n_{bi} + n_{bj}}, \tag{4.1}$$

where n_{bi} is the number of occurrences of base b in the i-th column, p_b is the background frequency for base b, and f_{bi} is the frequency of base b in the i-th column. Since the number of aligned species tends to be small, pseudo-counts are added to the score to account for small sample biases. Thus, n_i is estimated as $n_i + pc$, and f_{bi} is estimated as $\frac{n_{bi}+p_b*pc}{n_i+pc}$, where $n_i = \sum_{b=A..T} n_{bi}$ and pc is the pseudo-count, which is 0.1 by default in PhyloNet.

For the hardware implementation, we wish to reduce the logarithm and division calculations to table look-ups, while minimizing the number of additions, multiplications, and table-lookups needed to complete an ALLR calculation. By rearranging the ALLR formula, we get an equation more conducive to table lookups. We can reform

the allr calculation as

$$ALLR(i,j) = \frac{1}{n_i + n_j + 2 * pc} \quad * \quad \left[\sum_{i \neq j} (n_i + pc) * (-log\,(n_j + pc)) \right. \tag{4.2}$$
$$\left. + \quad \sum_{i \neq j} \sum_{b=A..T} (n_{bi} + pc * p_b) * log \left(\frac{n_{bj} + pc * p_b}{p_b} \right) \right].$$

Replacing the logarithms and divisions with table lookups, the above equation becomes

$$ALLR(i,j) = Table1(n_i + n_j) \quad * \quad \left[\sum_{i \neq j} Table2(n_i) * Table3(n_j) \right.$$
$$+ \quad \sum_{b=A..T} (Tableb1(n_{bi}) * Tableb2(n_{bj}))$$
$$\left. + \quad \sum_{b=A..T} (Tableb1(n_{bj}) * Tableb2(n_{bi})) \right]. \tag{4.3}$$

Each ALLR calculation only requires 9 additions and 11 multiplications and no logarithms or divisions. The logarithms and divisions are pre-computed in software and loaded onto the hardware. The number of entries in the tables depends on the number of aligned species, and the width of each table entry depends on the number of bits needed to store a value.

4.2.3 Storage of Tables in block RAM and Representation of Values

The tables required for the ALLR calculation can be stored in the block RAMs on the FPGA. In the Virtex 4 family, the number of block RAMs range from 48 to 552, and each block RAM is 18 kilobits. The block RAMs are dual-ported, meaning two simultaneous accesses can be made to a single block RAM on each clock.

The block RAMs can be configured into many aspect ratios. We utilize the ratios 1K x 18 and 512 x 36. To do so, we represent numbers using 18-bit fixed point arithmetic.

This allows us to store one value per entry in a 1K x 18-bit table and two values per entry in a 512 x 36-bit table. We find that using 18-bit fixed point numbers with 8 bits of precision maintains the same sensitivity as the full floating point numeric representation.

In equation 4.3, the indices into some tables are the same. Tables that are indexed by the same numbers can be stored side-by-side in a block RAM arranged in the ratio 512 x 36 bits. A single lookup can retrieve two values, one stored in the first 18 bits, and the other stored in the last 18 bits. Figure 4.3 shows an example of one of the tables where the number of aligned sequences is four. Two lookups are done, but four values are returned.

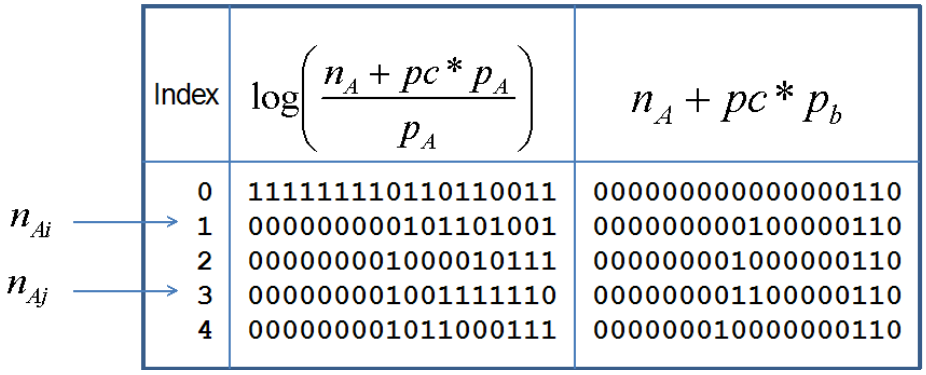

Figure 4.3: Storage of two tables in one block RAM

Based on this idea, we can consolidate the tables that are indexed by the same numbers.

$$ALLR(i,j) = Table1(n_i + n_j) \quad * \quad \left[\sum_{i \neg j} Table2_L(n_i) * Table2_U(n_j) + \sum_{b=A..T} (Tableb_L(n_{bi}) * Tableb_U(n_{bj})) + \sum_{b=A..T} (Tableb_L(n_{bj}) * Tableb_U(n_{bi})) \right]. \quad (4.4)$$

The tables subscripted with L and R return two values with each lookup. Given a table $TableX$ that returns two values, the lower half of the bits contain the value of $TableX_L$ and the upper half of the bits contain the value of $TableX_U$. Instead of needing 11 tables, one for each lookup, we need only 6 tables. These 6 tables give us the 21 values needed to compute the ALLR score. A description of each of these tables is given in Table 4.2.

Table 4.2: Composition of Tables for ALLR Calculation

Table	Index	Value 1	Value 2	Size	Max Species
Table1	$n_i + n_j$	$\frac{1}{n_i+n_j+2*pc}$	none	1024 entries x 18 bits	511
Table2	n_i	$n_i + pc$	$-log(n_i + pc)$	512 entries x 36 bits	511
TableA	n_{Ai}	$n_{Ai} + pc * p_A$	$log(\frac{n_{Ai}+pc*p_A}{p_A})$	512 entries x 36 bits	511
TableC	n_{Ci}	$n_{Ci} + pc * p_C$	$log(\frac{n_{Ci}+pc*p_C}{p_C})$	512 entries x 36 bits	511
TableG	n_{Gi}	$n_{Gi} + pc * p_G$	$log(\frac{n_{Gi}+pc*p_G}{p_G})$	512 entries x 36 bits	511
TableT	n_{Ti}	$n_{Ti} + pc * p_T$	$log(\frac{n_{Ti}+pc*p_T}{p_T})$	512 entries x 36 bits	511

4.2.4 Pipeline Design of ALLR calculation

To maximize the throughput of each ALLR calculation, we divide the computation into pipelined stages. We generate a dependency tree of the steps involved to calculate the ALLR score using the table lookups. This is illustrated in Figure 4.4.

Table 1 is queried once per clock cycle, and the other tables are queried twice, giving 11 queries of the six tables. This is possible because the block RAMs are dual-ported, allowing two simultaneous accesses. The entries returned by Tables 2, A, C, G and T actually contain two values: one value in the lower 18 bits and the other value in the upper 18 bits. The 21 values returned by the table lookups are combined through a combination of 9 additions and 11 multiplications to get the result of the ALLR calculation. Since the calculation is pipelined, the ALLR block can return one ALLR calculation every clock cycle after an initial latency of 8 clock cycles.

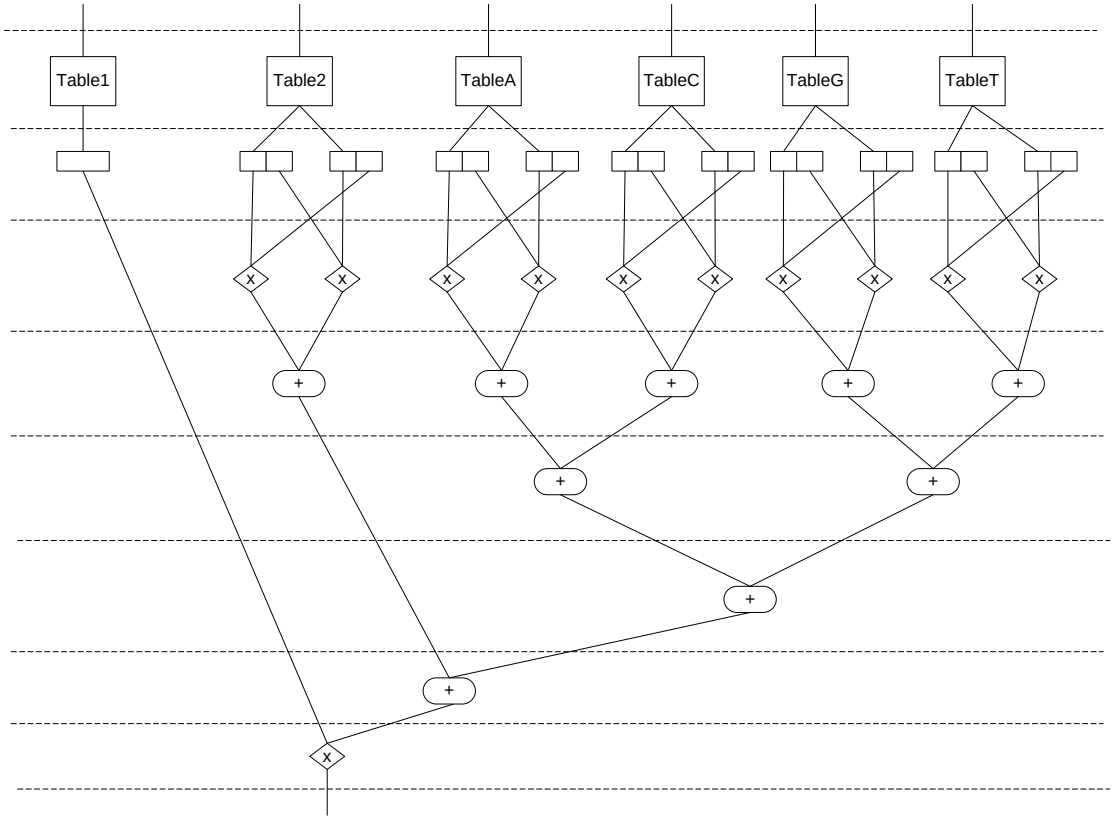

Figure 4.4: Pipelined ALLR calculation.

4.3 Synthesis of ALLR calculation in Hardware

In order to estimate how much area the ALLR units require, we simulated the ALLR calculators on the Virtex-4 XC4VLX80 platform using the Xilinx synthesis tools (www.xilinx.com). We synthesize up to 30 ALLR units using the pipelined design with table lookups and 18-bit fixed point arithmetic. The area estimates are shown in Figure 4.5.

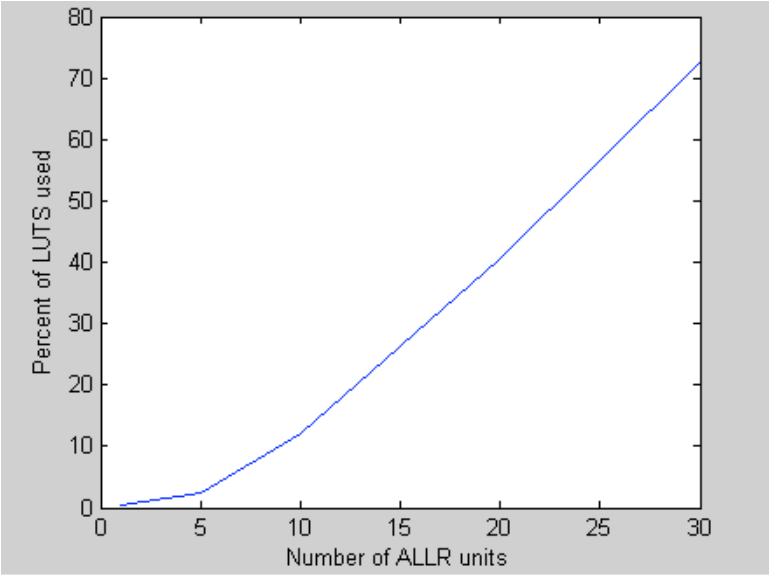

Figure 4.5: Area usage of ALLR units

Since a window size of 20 only requires 40% of the total number of LUTS and only passes 0.82% of the extensions to software, we use this window size for our hardware Stage 2 extension.

4.4 Performance Estimates of Stage 2

We compute the estimated time spent in Stage 2 of the hardware design and report speedup over the software. From our synthesis of the ALLR design in hardware, we find that we are able to process one hit per clock cycle, and our maximum clock cycle is 155 MHz. Since each hardware extension can be processed in one clock cycle using

a window size of 20, the time spent in hardware extension is the total number of seed hits times the clock period. The time spent in software is the time spent processing the seed hits that make it pass the hardware fixed window extension. Table 4.3 lists these times.

Table 4.3: Stage 4 Timing Analysis

Parameters			Hardware Filter	
Word Length	Neigh. Thresh.	Query Size	Hardware Time (seconds)	Software Time (seconds)
5	4	1000	57.7	22.4
5	4	2000	57.9	35.8
5	4	4000	57.9	58.5
5	5	1000	34.3	24.3
5	5	2000	34.4	38.5
5	5	4000	34.4	66.4
5	6	1000	17.1	21.3
5	6	2000	17.2	40.4
5	6	4000	17.2	63.3
6	4	1000	35.3	20.8
6	4	2000	35.4	32.8
6	4	4000	35.4	57.8
6	5	1000	18.0	20.9
6	5	2000	18.1	33.2
6	5	4000	18.1	57.8
6	6	1000	10.6	20.1
6	6	2000	10.6	32.2
6	6	4000	10.6	55.7

The overall time spent in hardware Stage 2 is the maximum of the times spent in hardware extension and software extension since these two steps can be done in parallel. The fastest hardware Stage 2 time is 20.1 seconds. This is achieved using word length of 6, neighborhood threshold of 6.0 and query size of 1000. The fastest software implementation spends 780.9 seconds in Stage 2, which is 38.9 times slower than hardware.

4.5 Overall Performance Estimation

We were able to substantially reduce the time spent in Stage 1 and Stage 2 of PhyloNet, which we found to be the major bottlenecks in the algorithm. Here, we present the overall performance gains with our hardware implementations of Stages 1 and 2. We found that Stage 1 could run at 200 MHz, while Stage 2 could run at 155 MHz. Thus, we run both stages at 155 MHz. Table 4.4 gives the modified time in Stage 1 using a clock frequency of 155 MHz.

Table 4.4: Stage 1 Hardware Time with Reduced Clock Frequency

Seed Length	Neighborhood Threshold	Query Size	Hardware Time (sec)
5	4	1000	40.5
5	4	2000	31.9
5	4	4000	26.3
5	5	1000	31.9
5	5	2000	22.8
5	5	4000	18.1
5	6	1000	26.8
5	6	2000	16.4
5	6	4000	11.6
6	4	1000	31.4
6	4	2000	22.8
6	4	4000	18.5
6	5	1000	26.3
6	5	2000	16.0
6	5	4000	11.6
6	6	1000	25.2
6	6	2000	13.8
6	6	4000	8.7

We have two main constraints that need to be satisfied. The first is that if we wish to use a direct lookup table, we must use a seed length of no more than 5. Also, since Stage 2 can only extend one seed hit every clock cycle, the average number of hits being produced by Stage 1 per clock cycle must be less than one. We see from our analysis in Chapter 3 that there is one set of parameters that meet both critera: seed length of 5, neighborhood threshold of 6 and query length of 1000. Stage 1 produces

0.64 positions per clock cycle with these parameters, a rate which Stage 2 can handle. Table 4.5 shows the time spent in the optimal design, split into the hardware and software time for each stage.

Table 4.5: Hardware and software times for each stage of the optimal configuration.

Stage	Hardware time (sec)	Software time (sec)
1	26.8	—
2	17.1	21.3
3	—	10.3
Final Cost	26.8 sec	31.6 sec

The hardware stages are pipelined, meaning the total time is the maximum of the times spent in each individual stage. Thus, the total time spent on the yeast data in seed matching, extension, and clustering is 31.6 seconds. This is compared to the 926.2 seconds that is required to run these stages in software alone on a single CPU. This gives an inferred speedup of 29x over the three stages in the best software version. Even if the stages of PhyloNet were pipelined on software and placed on 3 CPUs, it would still take a total time of 780.93 seconds, giving a speedup of 25x.

4.5.1 Accounting for Query Pre-processing

In addition to the the three stages of the PhyloNet pipeline, a significant amount of time is spent in the preprocessing of the queries. This preprocessing involves the neighborhood generation and table set-up for Stage 1. If this preprocessing cannot be done offline, it must be taken into account when estimating the total runtime of our hardware accelerated design of PhyloNet. In the software version of PhyloNet, preprocessing takes roughly 27% of the total time. Without improvements to this step, the most speedup that can be achieved is about 4x over the fastest software version. Improvements such as vectorization and better memory management can be made to optimize the code for query preprocessing [20]. Additionally, the sizes of the query tables grow exponentially with the word size (Table 3.1) and so using a smaller word size will cut down on the cost of query preprocessing. Since the software version

of PhyloNet uses a word length of 6 and the streaming hardware version uses a word length of 5, less work needs to be done to create the lookup tables. If the time spent in the query preprocessing step can be improved by 5 to 10x, the overall speedup over the best software version would be between 18 and 35x respectively.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

The short, degenerate nature of transcription factor binding sites coupled with the rapidly increasing genomic database makes motif-finding in DNA a demanding and time-consuming task. Recent developments in phylogenetic footprinting have resulted in more sensitive tools for locating patterns in DNA. PhyloNet is a software tool for finding patterns in DNA by combining information from conserved sequences within a single species and across many related species. The software implementation of this tool can take several days to run on large eukaryotic genomes such as mammalian.

We addressed the problem of scaling PhyloNet to large genomes by first redesigning the software and then developing a hardware/software architecture. We first made improvements to the software, most importantly reducing the complexity of HSP clustering from quadradic to linear in the number of HSPs and simplifying the input into PhyloNet by adding support for gapped alignments. We found that running PhyloNet on gapped alignments improved the tool's sensitivity on a list of known transcription factor binding sites. These improvements gave us a speedup of over 20x over the original and resulted in the same amount of output but 1.6x more known TF binding sites in the budding yeast genome.

We next developed a design of PhyloNet that is amenable to implementation on an FPGA. We organized PhyloNet as a pipeline with three stages: seed generation, seed extension, and HSP clustering. We found that the first two stages are the bottlenecks and created a design to place both these stages on hardware. The estimated speedup

of the hardware version of PhyloNet over our best software version is 18-35x, depending on how much we can reduce the time spent in the query preprocessing step. These efforts make PhyloNet a more sensitive and faster tool for finding conservation across multiple genomes.

5.2 Future Works

Immediate future work includes optimizing the query preprocessing step so that it is no longer the bottleneck and implementing and testing the hardware design put forth in this thesis. Stage 1 of PhyloNet is similar to the seed matching stage of the BLASTP algorithm, allowing us to draw upon work already done to accelerate BLASTP on FPGA hardware [21]. Possible improvements to the Stage 1 design include using Bloom Filters to pre-filter the table lookups and using hash tables to allow for larger seed lengths in hardware. We saw that using a seed length of 6 offered better performance in hardware than a seed length of 5, but a hash table design would need to be used in Stage 1 to allow for a seed length greater than 5.

Stage2, the seed extension stage of PhyloNet, deviates the most from BLAST-like algorithms because it uses a much more complicated scoring scheme between the query and database. The design used in this thesis - efficient use of table lookups and a pipeline of the computation tree - may be amenable to other complex comparisons of query to database such as HMMER, which compares sequences to a statistical model of a family of sequences. Further work needs to be done to determine how to best implement the systolic array used in seed extension. Either the two-way extension described in this thesis or the one-way extension described in [23] could be used, and the choice of implementations depends on the resources available.

Appendix A

Glossary

Alignment: A pairing of bases between two sequences highlighting the similarites and differences between them. Pairs of bases that are the same are called matches, and pairs of bases that are different are called mismatches. A base that is not paired with any base is paired with a gap.

Average Log Likelihood Ratio (ALLR): Score used by PhyloCon and PhyloNet to compare two profile columns.

Database Profiles: The set of profiles that the query profile is compared against.

Degenerate Sequence: A sequence created by mapping the columns of a profile to discrete characters. This representation is used in Stage 1 of PhyloNet.

Extension: An attempt to find the maximum score between two profile segments that contain a seed. This is part of Stage 2 of PhyloNet.

Gapped Alignment: An alignment between two sequences allowing gaps.

High Scoring Segment Pair: An extension that scores above some minimum threshold.

Motif: A recurring sequence pattern within the genome.

Multiple Alignment: An alignment of three or more sequences.

Neighborhood: For a given w-mer, the list of all w-mers that score above some threshold T when compared to that w-mer.

Orthologous sequences: Sequences that have been derived from a common ancestor.

Profile: A matrix containing the number of each type of base found in each column of the alignment.

Promoter: A regulatory region of DNA located upstream of a gene that controls initiation of gene transcription.

Query Profile: A profile that is compared against profiles from the database.

Seed: A short exact match between two profiles that acts as the starting point for an extension.

Sequence: A chain of characters from a discrete alphabet.

Transcription Factor (TF): A protein that binds to specific parts of DNA and controls the transcription of DNA into RNA.

Transcription Factor Binding Site (TFBS): A site in genomic DNA that binds transcription factors.

Ungapped Alignment: An alignment between two sequences in which all bases between sequences are paired.

***W*-mer:** A sequence of exactly w characters. Also known as a word.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J. Mol. Biol*, 215(3):403–410, 1990.

[2] M.M. Babu, N.M. Luscombe, L. Aravind, M. Gerstein, and S.A. Teichmann. Structure and evolution of transcriptional regulatory networks. *Current Opinion in Structural Biology*, 14(3):283–291, 2004.

[3] T.L. Bailey and C. Elkan. Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learning*, 21(1):51–80, 1995.

[4] R. Beinoraviciute-Kellner, G. Lipps, and G. Krauss. In vitro selection of DNA binding sites for ABF1 protein from Saccharomyces cerevisiae. *FEBS Lett*, 579(20):4535–40, 2005.

[5] M. Blanchette et al. Aligning Multiple Genomic Sequences With the Threaded Blockset Aligner. *Genome Research*, 14(4):708–715, 2004.

[6] M. Blanchette and M. Tompa. Discovery of Regulatory Elements by a Computational Method for Phylogenetic Footprinting. *Genome Research*, 12(5):739–748, 2002.

[7] AR Buchman, WJ Kimmerly, J. Rine, and RD Kornberg. Two DNA-binding factors recognize specific sequences at silencers, upstream activating sequences, autonomously replicating sequences, and telomeres in Saccharomyces cerevisiae. *Mol. Cell. Biol*, 8(1), 1988.

[8] S. Cawley et al. Unbiased Mapping of Transcription Factor Binding Sites along Human Chromosomes 21 and 22 Points to Widespread Regulation of Noncoding RNAs. *Cell*, 116(4):499–509, 2004.

[9] F. Della Seta, I. Treich, JM Buhler, and A. Sentenac. ABF1 binding sites in yeast RNA polymerase genes. *Journal of Biological Chemistry*, 265(25):15168–15175, 1990.

[10] Genes in Yeast. `http://www.yeastgenome.org/cache/genomeSnapshot.html`.

[11] Growth of GenBank Database. `http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html`.

[12] Z. Guo, W. Najjar, F. Vahid, and K. Vissers. A quantitative analysis of the speedup factors of FPGAs over processors. *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 162–170, 2004.

[13] U.I. Gupta, D.T. Lee, and J.Y.T. Leung. Efficient algorithms for interval graphs and circular-arc graphs. *Networks*, 12(4):459–467, 1982.

[14] J.M. Hancock and J.S. Armstrong. SIMPLE34: an improved and enhanced implementation for VAX and Sun computers of the SIMPLE algorithm for analysis of clustered repetitive motifs in nucleotide sequences. *Bioinformatics*, 10(1):67–70, 1994.

[15] C.T. Harbison et al. Transcriptional regulatory code of a eukaryotic genome. *Nature*, 431:99–104, 2004.

[16] M.C. Herbordt, J. Model, Y. Gu, B. Sukhwani, and T. VanCourt. Single Pass, BLAST-Like, Approximate String Matching on FPGAs. *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on*, pages 217–226, 2006.

[17] G.Z. Hertz and G.D. Stormo. Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, 15(7):563–577, 1999.

[18] J.D. Hughes, P.W. Estep, S. Tavazoie, and G.M. Church. Computational identification of Cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *Journal of Molecular Biology*, 296(5):1205–1214, 2000.

[19] Human Genome Project. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`.

[20] A. Jacob. Design and Analysis of an Accelerated Seed Generation Stage for BLASTP on the Mercury System. 2006.

[21] A. Jacob, J. Lancaster, J. Buhler, and R.D. Chamberlain. FPGA-accelerated seed generation in Mercury BLASTP. *Proc. of Symp. on Field-Programmable Custom Computing Machines*, pages 95–104, 2007.

[22] P. Krishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang, A. Jacob, and J. Lancaster. Biosequence Similarity Search on the Mercury System. *The Journal of VLSI Signal Processing*, 49(1):101–121, 2007.

[23] J. Lancaster, J. Buhler, and R.D. Chamberlain. Acceleration of ungapped extension in Mercury BLAST. *International Journal of Embedded Systems*, 2007.

[24] CE Lawrence, SF Altschul, MS Boguski, JS Liu, AF Neuwald, and JC Wootton. Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, 262(5131):208–214, 1993.

[25] Y. Liu, X.S. Liu, L. Wei, R.B. Altman, and S. Batzoglou. Eukaryotic Regulatory Element Conservation Analysis and Identification Using Comparative Genomics. *Genome Research*, 14(3):451–458, 2004.

[26] K.D. MacIsaac, T. Wang, D.B. Gordon, D.K. Gifford, G.D. Stormo, and E. Fraenkel. An improved map of conserved regulatory sites for Saccharomyces cerevisiae. *BMC Bioinformatics*, 7(1):113, 2006.

[27] A.M. Moses, D.Y. Chiang, M. Kellis, E.S. Lander, and M.B. Eisen. Position specific variation in the rate of evolution in transcription factor binding sites. *BMC Evolutionary Biology 3:19.*, 3(19), 2004.

[28] Multiple Alignment Format. `http://genome.ucsc.edu/FAQ/FAQformat.html/format5#format5`.

[29] Number of Human Genes. `http://www.ornl.gov/sci/techresources/Human_Genome/faq/genenumber.shtml`.

[30] R. Siddharthan, E.D. Siggia, and E. van Nimwegen. PhyloGibbs: A Gibbs sampling motif finder that incorporates phylogeny. *PLoS Comput. Biol*, 1(7):e67, 2005.

[31] S. Sinha, M. Blanchette, and M. Tompa. PhyME: A probabilistic algorithm for finding motifs in sets of orthologous sequences. *BMC Bioinformatics*, 5(170), 2005.

[32] J.D. Thompson, D.G. Higgins, T.J. Gibson, et al. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680, 1994.

[33] Timelogic, Inc. Timelogic DeCypher BLAST. `http://www.timelogic.com`.

[34] E. van Nimwegen. Scaling laws in the functional content of genomes. *Trends in Genetics*, 19(9):479–484, 2003.

[35] T. Wang and G.D. Stormo. Combining phylogenetic data with co-regulated genes to identify regulatory motifs. *Bioinformatics*, 19(18):2369–2380, 2003.

[36] T. Wang and G.D. Stormo. Identifying the conserved network of cis-regulatory sites of a eukaryotic genome. *Proceedings of the National Academy of Sciences*, 102(48):17400–17405, 2005.

Vita

Justin Tyler Brown

Date of Birth	October 23, 1983
Place of Birth	Springfield, MO
Degrees	B.S. Summa Cum Laude, Computer Science, May 2006

August 2008

Accelerating PhyloNet on FPGAs, Brown, M.S. 2008