ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCE

Çağlar YILMAZ **MSc Thesis**

SYSTOLIC ARCHITECTURES FOR SEQUENCE ALIGNMENT OPERATIONS

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ADANA, 2008

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

SYSTOLIC ARCHITECTURES FOR SEQUENCE ALIGNMENT OPERATIONS

Çağlar YILMAZ

YÜKSEK LİSANS TEZİ

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Bu tez 16/01/2008 Tarihinde Aşağıdaki Jüri Üyeleri Tarafından Oybirliği İle Kabul Edilmiştir.

İmza.....	İmza.....	İmza.....
Yrd.Doç.Dr. Mustafa GÖK	Yrd.Doç.Dr. Ulus ÇEVİK	Yrd.Doç.Dr. Mutlu AVCI
DANIŞMAN	ÜYE	ÜYE

Bu tez Enstitümüz Elektrik-Elektronik Mühendisliği Anabilim Dalında hazırlanmıştır.
Kod No

Prof. Dr. Aziz ERTUNÇ
Enstitü Müdürü
İmza ve Mühür

Bu Çalışma TUBİTAK Tarafından Desteklenmiştir.

Proje No: TUBİTAK-KARİYER 105E075

ABSTRACT

MSc THESIS

SYSTOLIC ARCHITECTURES FOR SEQUENCE ALIGNMENT OPERATIONS

Çağlar YILMAZ

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
INSTITUTE OF NATURAL AND APPLIED SCIENCES
UNIVERSITY OF ÇUKUROVA

Supervisor: Assist.Prof.Dr. Mustafa GÖK

Year: 2008, Pages: 78

Jury: Assist.Prof.Dr. Mustafa GÖK
Assist.Prof.Dr. Ulus ÇEVİK
Assist.Prof.Dr. Mutlu AVCI

In this work, new and optimized algorithms and application specific processors based on these algorithms to perform sequence alignment operations, which is one of the basic operations in bioinformatics are designed. In this work, systolic architectures are developed to implement pairwise and multiple sequence alignment operations. Proposed hardware methods develop hardwares implemented on ASIC and FPGA platforms at the rate of over one hundred per cent. The designs proposed in this work provides performance gain up to 189,92% when it is implemented on FPGA. Therefore, it is provided to other research branches in bioinformatics to analyse larger amount of data in smaller time periods.

Key Words: Bioinformatics, systolic architectures, application specific processors, FPGAs, Smith-Waterman algorihtm

ÖZ

YÜKSEK LİSANS TEZİ

SYSTOLIC ARCHITECTURES FOR SEQUENCE ALIGNMENT OPERATIONS

Çağlar YILMAZ

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Danışman: Yrd.Doç.Dr. Mustafa GÖK

Yıl: 2008, Sayfa: 78

Jüri : Yrd.Doç.Dr. Mustafa GÖK
Yrd.Doç.Dr. Ulus ÇEVİK
Yrd.Doç.Dr. Mutlu AVCI

Bu çalışmada, biyoinfomatik biliminin temel işlemlerinden biri olan sıra hizalama işlemlerini çalıştırmaya yönelik, yeni ve optimize edilmiş algoritmalar ve bu algoritmaları temel alan uygulamaya yönelik işlemciler tasarlanmıştır. Bu çalışmada, ikili ve çoklu sıra hizalama işlemlerini gerçekleştirmeye yönelik sistolik donanımlar geliştirilmiştir. Önerdiğimiz tasarım yöntemleri, ASIC ve FPGA platformlarında gerçekleştirilen tasarımların alan ve zamanlarını yüzde yüzü aşan oranlarda geliştirmiştir. Bu çalışmada yer alan tasarımlar, FPGA üzerinde gerçeklendiğinde %189,92'ye kadar bir performans artışı sağlamıştır. Böylelikle biyoinformatik araştırmaları yapan diğer dallara daha büyük verileri daha kısa zaman dilimlerinde analiz etme imkanı sağlanmıştır.

Anahtar Kelimeler: Biyoinformatik, sistolik mimariler, uygulamaya özel işlemciler, FPGA'ler, Smith-Waterman algoritması

ACKNOWLEDGEMENT

First of all, I would like to thank my advisor, Assist.Prof.Dr. Mustafa GÖK, for his supervision guidance, encouragements, extremely useful suggestions, and his valuable time for this work.

I would like to thank Prof.Dr. Mehmet TÜMAY, the Head of the Computer Engineering Department, for his valuable suggestions and providing a good working environment.

I would like to thank Prof.Dr. Süleyman GÜNGÖR, the Head of the Electrical and Electronics Engineering Department, for his suggestions and providing research laboratories to finish this thesis.

I would also like to thank members of MSc thesis jury, Assist.Prof.Dr. Ulus ÇEVİK and Assist.Prof.Dr. Mutlu AVCI, for their suggestions and corrections.

I would like to thank all my colleagues for their friendship and supports to finish this thesis.

I would like to thank Turkish National Science Foundation (TUBİTAK) for financial support and for being sponsor to this project.

Finally, I would like to thank my wife, Arzu Ayşen for her endless support and encouragements for my life and career.

Çağlar YILMAZ

CONTENTS **PAGE**

ABSTRACT I
ÖZ II
ACKNOWLEDGEMENTS III
CONTENTS IV
LIST OF TABLES VI
LIST OF FIGURES VII
1. INTRODUCTION 1
2. SEQUENCE ALIGNMENT 3
2.1. Pairwise Sequence Alignment 7
2.1.1. Dot Matrix Methods 7
2.1.2. Dynamic Programming Methods 9
2.1.3. Word Methods 9
2.2. Multiple Sequence Alignment 10
2.2.1. Dynamic Programming Methods 11
2.2.2. Progressive Alignment 12
2.2.3. Iterative Methods 12
2.2.4. Hidden Markov Models 12
2.3. Software Tools for Sequence Alignment 13
2.4. Hardware Tools For Sequence Alignment 17
3. SMITH-WATERMAN ALGORITHM 19
3.1. The Algorithm 19
3.2. An Example of Smith-Waterman Algorithm 23
3.2.1. Determining the Sequences and the Parameters 23
3.2.2. Initialization the Score Matrix 24
3.2.3. Building the Matrix 24
3.2.4. Trace-Back Operation 25
3.2.5. Completing the Alignment 26
4. THE PROPOSED SMITH-WATERMAN DESIGNS 28
4.1. The Systolic Architecture 28

4.2. The Proposed Smith-Waterman Cell Designs for Pairwise Sequence Alignment 32
4.2.1. Linear Gap Penalty Cell 33
4.2.2. Affine Gap Penalty Cell 38
4.3. The Proposed Smith-Waterman Cell Designs for Multiple Sequence Alignment 44
4.3.1. Linear Gap Penalty Multiple Sequence Alignment 46
4.3.2. Affine Gap Penalty Multiple Sequence Alignment 52
5. SYNTHESIS RESULTS 59
5.1. Graphical Representations of the Syntheses Results 64
6. CONCLUSIONS AND FUTURE WORKS 70
REFERENCES 71
BIOGRAPHY 78

LIST OF TABLES **PAGE**

Table 2.1. Well known pairwise sequence alignment tools 14
Table 2.2. Popular multiple sequence alignment tools 16
Table 2.3. Hardware tools for sequence alignment operations (Hireche et. al., 2006) 17
Table 3.1. Similarity scores between FA9 proteins of some mammalian organisms 27
Table 4.1. Distribution of overall calculation time of CLUSTALW (Oliver et al., 2005) 45
Table 4.2. Truth table for the logic operation of the selector unit for linear gap penalty model 52
Table 4.3. Truth table for the logic operation of the selector unit for linear gap penalty model 57
Table 4.4. The practical results of the multiple sequence alignment hardware (Oliver et. al., 2005) 57
Table 5.1. The synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for linear gap penalty implementations 61
Table 5.2. The synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for affine gap penalty implementations 62
Table 5.3. The detailed FPGA synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for linear gap penalty implementations 63
Table 5.4. The detailed FPGA synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for affine gap penalty implementations 64

LIST OF FIGURES **PAGE**

Figure 2.1. Growth of GenBank from 1982 to 2005 (NCBI, 2007) 5
Figure 2.2. Sequence alignment between ACRO_HUMAN and ACRO_MOUSE protein sequences generated using CLUSTALX (Thompson et al, 1997) software 6
Figure 2.3. Dot matrix plot produced from the alignment of ACRO_HUMAN and ACRO_MOUSE protein sequences generated using Dotter (Sonnhammer and Durbin, 1995) software 8
Figure 2.4. Multiple sequence alignment between some mammalian organisms generated using CLUSTALX (Thompson et al, 1997) software 11
Figure 3.1. The Dayhoff PAM250 substitution matrix 21
Figure 3.2. The BLOSUM62 substitution matrix 21
Figure 3.3. Initialization the Smith-Waterman score matrix 24
Figure 3.4. Filling the score matrix 25
Figure 3.5. Trace-back operation 26
Figure 4.1. Adapting the systolic architecture to Smith-Waterman score matrix 29
Figure 4.2. Systolic architecture of Smith-Waterman algorithm 30
Figure 4.3. Linear gap penalty implementation of Smith-Waterman cell 35
Figure 4.4. Affine gap penalty implementation of Smith-Waterman cell 40
Figure 4.5. h(i,j) matrix for example 47
Figure 4.6. n(i,j) matrix for example 48
Figure 4.7. Additional signals to Smith-Waterman cell to perform multiple sequence alignment 49
Figure 4.8. Additional hardware to Smith-Waterman cell to perform multiple sequence alignment 50
Figure 4.9. Additional signals to Smith-Waterman cell to perform multiple sequence alignment 54
Figure 4.10. Additional hardware to Smith-Waterman cell to perform multiple sequence alignment 55

Figure 5.1. Graphical representation of space requirements for ASIC syntheses of linear gap penalty implementations..... 65
Figure 5.2. Graphical representation of clock frequencies for ASIC syntheses of linear gap penalty implementations..... 65
Figure 5.3. Graphical representation of space requirements for FPGA syntheses of linear gap penalty implementations..... 66
Figure 5.4. Graphical representation of clock frequencies for FPGA syntheses of linear gap penalty implementations..... 66
Figure 5.5. Graphical representation of space requirements for ASIC syntheses of affine Gap penalty implementations 67
Figure 5.6. Graphical representation of clock frequencies for ASIC syntheses of affine gap penalty implementations 67
Figure 5.7. Graphical representation of space requirements for FPGA syntheses of affine gap penalty implementations 68
Figure 5.8. Graphical representation of clock frequencies for FPGA syntheses of affine gap penalty implementations 68
Figure 5.9. Graphical representation of performance measurements for FPGA syntheses of linear gap penalty implementations..... 69
Figure 5.10. Graphical representation of performance measurements for FPGA syntheses of affine gap penalty implementations..... 69

1. INTRODUCTION

Bioinformatics is one of the biggest fund transferred research area in the world. Since humanity expects that bioinformatics can answer many questions related with the cure of diseases, minimizing global warming, lengthen the lifetime, developing agricultural production techniques to end starvation. Molecular biologist cannot analyze data without computers. The main operation for analyzing data is the sequence alignment. Therefore different algorithms have been developed to perform different types of sequence alignment. Well known algorithms for pairwise sequence alignment are FASTA (Lipman and Pearson, 1985), BLAST (Altschul et al, 1990), Smith-Waterman (Smith and Waterman, 1981), Needleman-Wunsch (Needleman and Wunsch, 1970) and for multiple sequence alignment is CLUSTALW (Thompson et al, 1994). Sequence alignment algorithms process millions of data. Therefore they must be run on high performance systems to acquire the results in a reasonable time. Considering these criteria, four common types of processing systems can be selected for sequence alignment operations. These are listed as following;

- *General purpose super computers*: They are the most flexible choice for fast sequence alignments. However, they have a very high cost and they are very hard to access.
- *Single purpose processors*: They can implement the highest performance for a single algorithm. Although they do not have a very high cost, their flexibility is very low.
- *Programmable co-processors*: They can reach almost the flexibility of reconfigurable systems and the performance of single purpose processors. Their disadvantages are; hard to design and implement, not suitable for serial production. The examples of this type of processor for bioinformatics operations are UCSC Kestrel (Di Bias et al, 2005) and GeneMatcher2 (Paracel, Inc., 2005).
- *Reconfigurable hardwares*: They are the systems which include FPGAs (Field Programmable Gate Arrays). Their processing element density is

lower than single purpose processors, but they can reach the performance of super computers and they are completely reconfigurable. FPGA and related complier and tool technologies have a huge growing speed in recent ten years. Therefore their flexibility and speed increase while their cost and size decreases rapidly.

In this thesis, new, optimized algorithms and hardwares for several types of sequence alignment operations are presented for single purpose processors and FPGAs. The content of this thesis is arranged as follows;

After this introductory section, Section 2 gives an introduction to bioinformatics and sequence alignment, sequence alignment types, and discussion of most popular software and hardware tools for sequence alignment operations. Section 3 explains the detailed Smith-Waterman algorithm for local sequence alignments, and gives an example for local pair-wise sequence alignment in the meaning of linear gap penalty model. In Section 4, the proposed algorithms and hardwares for several types of sequence alignment operations are discussed. Also in this section multiple sequence alignment is mentioned in detail. In Section 5, the syntheses results of the proposed hardware designs are given for several types of sequence alignment operations. At the end, in Section 6, conclusions and future works are explained.

2. SEQUENCE ALIGNMENT

Defining the term bioinformatics is not an easy task, since there are multiple definitions from people, organizations, comities, books, and over the internet. Roughly, all definitions from different views meet the following common idea; bioinformatics describes any use of computers to handle biological information. In formal way, The NIH (National Institutes of Health of the USA) Biomedical Information Science and Technology Initiative Consortium agreed on the following more detailed definition for bioinformatics and computational biology (NIH, 2000);

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

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

Bioinformatics is a rapidly developing branch of biology and is highly interdisciplinary, using techniques and concepts from computer sciences, informatics, statistics, mathematics, chemistry, biochemistry, physics, and linguistics. It has many practical applications in different areas of biology, medicine and more. These application areas can be listed as molecular medicine, personalized medicine, preventative medicine, gene therapy, drug development, microbial genome applications, waste cleanup, climate change studies, alternative energy sources, biotechnology, antibiotic resistance, forensic analysis of microbes, evolutionary studies, crop improvement, insect resistance, improve nutritional quality, development of drought resistance varieties, veterinary science, and comparative studies.

Because of large number of application areas and their variety, bioinformatics become a very popular science branch. Numerical representation and evidence of the growing speed of bioinformatics can be observed from the biological databases.

A biological database is a large, organized body of persistent data, usually associated with computerized software designed to update, query, and retrieve components of the data stored within the system. Currently, a lot of bioinformatics work is concerned with the technology of databases. These databases include both "public" repositories of gene data like GenBank or the Protein DataBank (the PDB), and private databases used by research groups involved in gene mapping projects or held by biotech companies.

Because of the high rate of data production and the need for researchers to have rapid access to new data, public databases have become the major medium through which genome sequence data are published. Public databases and the data services that support them are important resources in bioinformatics, and will soon be essential sources of information for all the molecular biosciences. However, successful public data services suffer from continually escalating demands from the biological community. Waterman describes the current situation in the following way: *"It is probably important to realize from the very beginning that the databases will never completely satisfy a very large percentage of the user community. The range of interest within biology itself suggests the difficulty of constructing a database that will satisfy all the potential demands on it. There is virtually no end to the depth and breadth of desirable information of interest and use to the biological community."*

EMBL and GenBank are the two major bioinformatics databases. The rate of growth of DNA (Deoxyribo Nucleic Acid) databases has been following an exponential trend, with a doubling time now estimated to be 6 months (Figure 2.1). On the other hand, recently, EMBL contained 168.352.655.393 nucleotides and 95.459.227 entries. Another importance of bioinformatics comes from growth of these databases, since datas stored in them have to be handled in manner of time and cost.

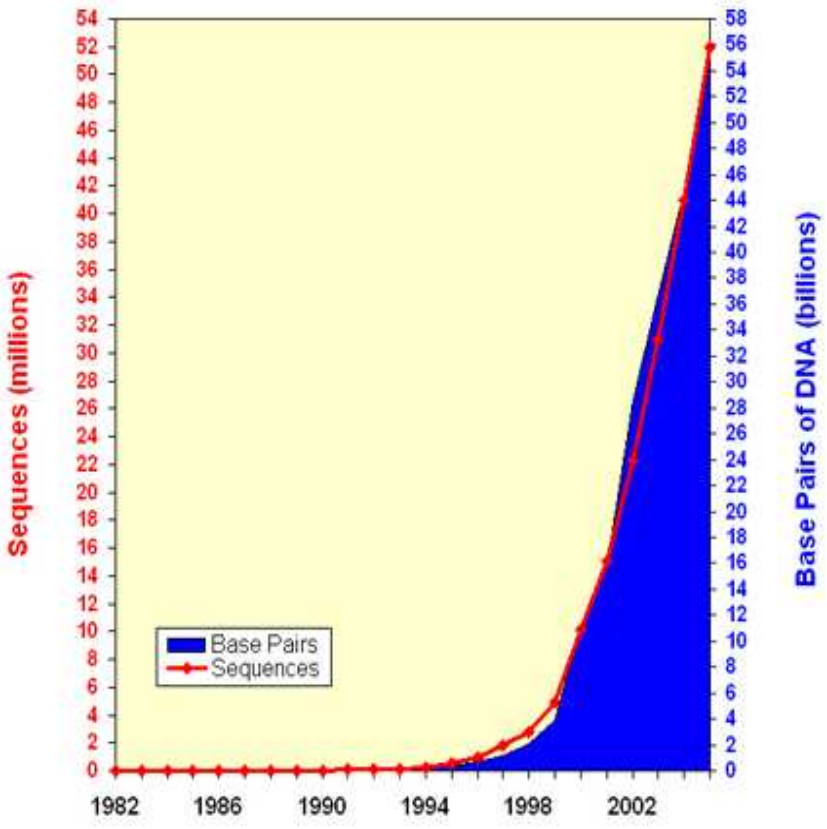

Figure 2.1. Growth of GenBank from 1982 to 2005 (NCBI, 2007)

Followings are the well known databases which serve the researches various kind of biological data; GenBank, EMBL, SwissProt, PROSITE, EC-ENZYME, RCSB PDB, GDB, OMIM, PIR-PSD, 21 Bdb: LBL's Human Chr 21 Database, MGD: The Mouse Genome Databases, ACeDB (A Caenorhabditis Elegans Database), and MEDLINE.

There are so many other public or private bioinformatics databases available on the web.

Sequence alignment is one of the most basic operations in bioinformatics. It is a way of arranging two or more DNA (Deoxyribo Nucleic Acid), RNA (Ribo Nucleic Acid), or protein codes (sequences) to identify the similar regions between them. Aligned sequences can be used to determine functional, structural, or evolutionary relationships between the corresponding organisms. Figure 2.2. shows an example of an sequence alignment operation using CLUSTALW software and

ACRO_HUMAN versus *ACRO_MOUSE* protein sequences, and similarities between them.

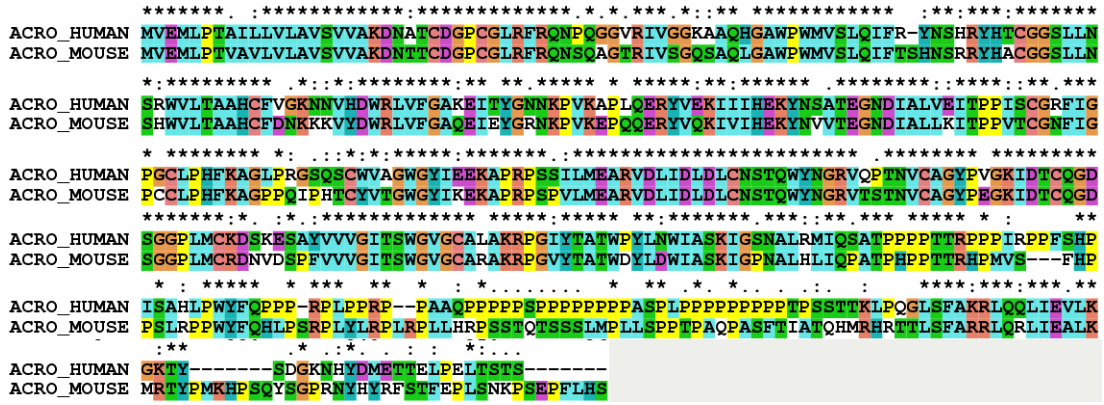

Figure 2.2. Sequence alignment between ACRO_HUMAN and ACRO_MOUSE protein sequences generated using CLUSTALX (Thompson et al, 1997) software

As Figure 2.2 shows, three operations are necessary to perform a complete sequence alignment. These operations are;

- **Match :** Indicates the corresponding column has identical DNA, RNA or protein characters.
- **Substitution :** Indicates the corresponding column has different characters and needs to be substituted.
- **Insertion or Deletion (also called gap operation) :** Places a gap to the necessary columns to slide one sequence so that next characters can be matched with the other sequence.

For short biological sequences, sequence alignment can be performed manually, however otherwise, some algorithms has to be used. Also as the length of the sequences increases, the computational complexity also increases, and computer technologies has to be used.

Computational approaches to sequence alignment operation generally fall into two categories. These are global sequence alignments and local sequence alignments. Global alignment is a form of sequence alignment which forces the alignment to span

the entire length of all query sequences. On the other hand, local alignment tries to identify only similarity regions within long sequences. Local alignment is computationally more challenging operation, but it is more preferable since it can present more functional, structural, or evolutionary relationships between the sequences.

Another classification can be made for sequence alignment as pairwise and multiple sequence alignments, according to the number of sequences to be aligned.

2.1. Pairwise Sequence Alignment

Pairwise sequence alignment is used to identify relationships between only two sequences. It is generally used for methods that do not require extreme precision. An example of these methods can be scanning a bioinformatics database for sequences with high homology to a known query sequence. Pairwise sequence alignment produces a score which indicates the similarity between two sequences. Therefore algorithms are developed to find this score in an efficient way.

There are three primary methods for producing pairwise sequence alignment; dot matrix methods, dynamic programming methods, and word methods.

2.1.1. Dot Matrix Methods

It is the most basic and simple algorithm to align two sequences. Its aim is to construct a plot which is based on to matches, insertions/deletions and gap operations. To construct a dot matrix plot, the two sequences are written along the top row and left most columns of a two-dimensional matrix and a dot is placed at any point where the characters in the appropriate columns match. Dot matrix methods can identify the most similar regions in the alignment visually, but it is a time consuming job to align long sequences. Figure 2.3. is an example of dot matrix plot produced from the alignment of ACRO_HUMAN and ACRO_MOUSE protein sequences using the tool “Dotter” (Sonnhammer and Durbin, 1995).

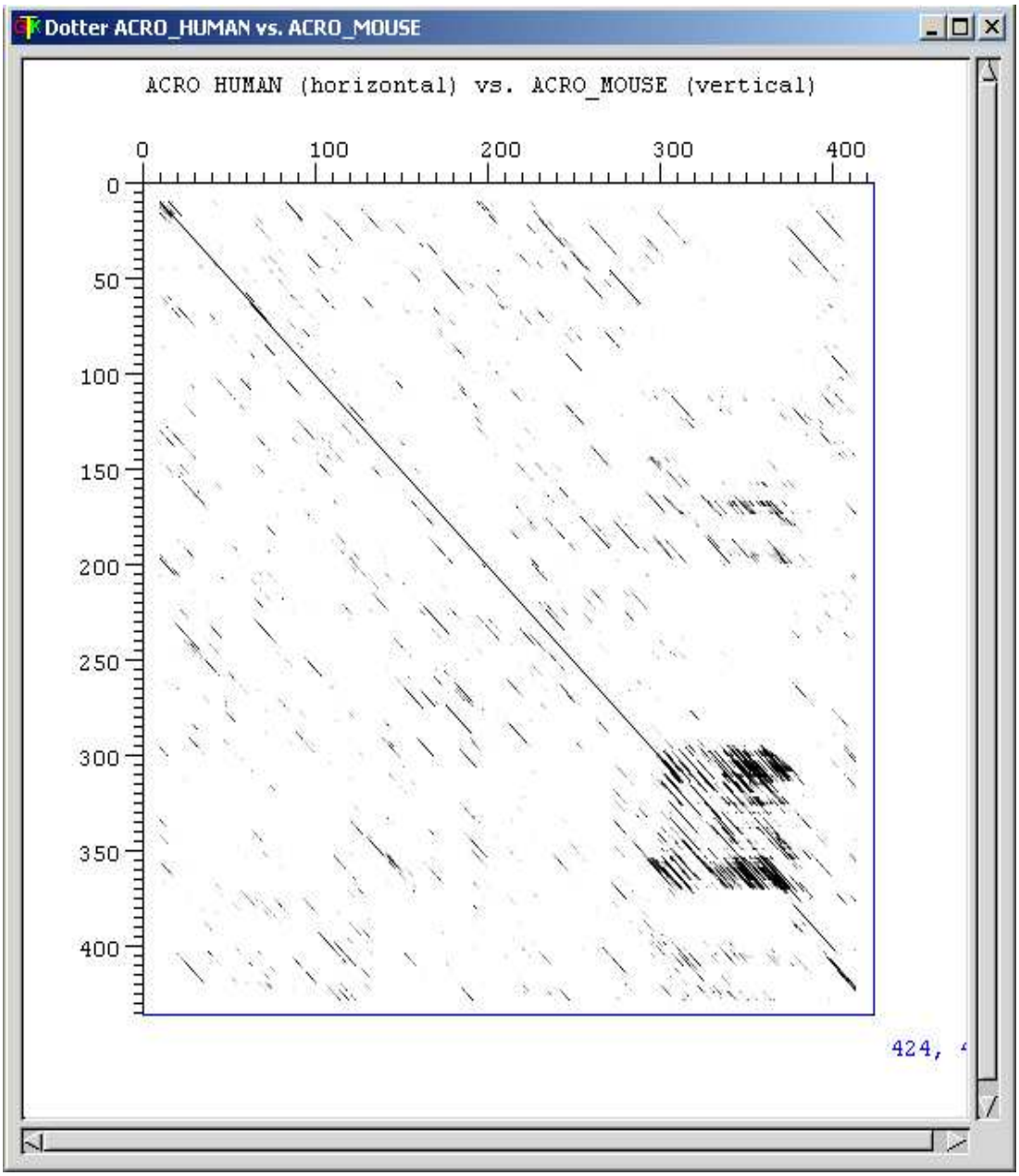

Figure 2.3. Dot matrix plot produced from the alignment of ACRO_HUMAN and ACRO_MOUSE protein sequences generated using Dotter (Sonnhammer and Durbin, 1995) software

2.1.2. Dynamic Programming Methods

Dynamic programming algorithms often aim to divide the sequence alignment problem into smaller parts. This means that every character in the alignment compared with each characters of other sequence. To perform this process, usually systolic architectures are selected. Generally dynamic programming algorithms guarantee the best optimal alignment, but it suffers from its computational complexity.

Dynamic programming methods can align all type of sequences. It uses gap penalties cost, and a score matrix to define the cost of substitution operation for each character to calculate the similarity score between the sequences. The most popular algorithms that use dynamic programming approach are Needleman-Wunsch (Needleman and Wunsch, 1970) and Smith-Waterman (Smith and Waterman, 1981) algorithms.

Needleman-Wunsch algorithm uses the dynamic programming method to align sequences to perform global sequence alignment. This algorithm is the first dynamic programming method used for biology.

Another popular algorithm is Smith-Waterman algorithm for local sequence alignment. This algorithm guarantees to find best similar region between two sequences. Smith-Waterman algorithm is mentioned in detail in Section 3.

2.1.3. Word Methods

Word Methods, also known as *k*-tuple methods, are heuristic methods. That means that they are based on to probability and statistical result, and they cannot guarantee to find the best optimal alignments. They accept the sequences as combinations of words which have *k* characters for each. As *k* increases the error rate of algorithms is increases, but the overall calculation time decreases. The reason of selecting this type of algorithms is that they are much faster than the other methods, although their results are capable of making errors. They are used in many web based

applications which serves as sequence alignment tools to scan the biological databases.

The most popular word methods implementations are FASTA (Lipman and Pearson, 1985) and BLAST (Altschul et al, 1990). In the FASTA method, the user defines a value k to use as the word length with which to search the database. The method is slower but more sensitive at lower values of *k*, which are also preferred for searches involving a very short query sequence. The BLAST family of search methods provides a number of algorithms optimized for particular types of queries, such as searching for distantly related sequence matches. BLAST was developed to provide a faster alternative to FASTA without sacrificing much accuracy; like FASTA, BLAST uses a word search of length k, but evaluates only the most significant word matches, rather than every word match as does FASTA. Most BLAST implementations use a fixed default word length that is optimized for the query and database type, and that is changed only under special circumstances, such as when searching with repetitive or very short query sequences.

2.2. Multiple Sequence Alignment

Multiple sequence alignment (MSA) is simply expanding pairwise sequence alignment to align more than two sequences at a time. Generally MSA is used to identify evolutionary conserved regions after mutations, to build phylogenetic trees, and to identify evolutionary linkage between organisms. MSA also can be local or global as mentioned in Section 2.2. Figure 2.4. shows an example of MSA performed between some mammalian organisms (EL2_MOUSE, EL2_RAT, EL2A_HUMAN, EL2B_HUMAN, EL2_BOVIN, EL2_PIG, CLCR_RAT, EL3A_HUMAN, EL3B_HUMAN, EL1_BOVIN, EL1_RAT, EL1_PIG, CFAD_RAT, ELNE_HUMAN), generated with CLUSTALX alignment tool.

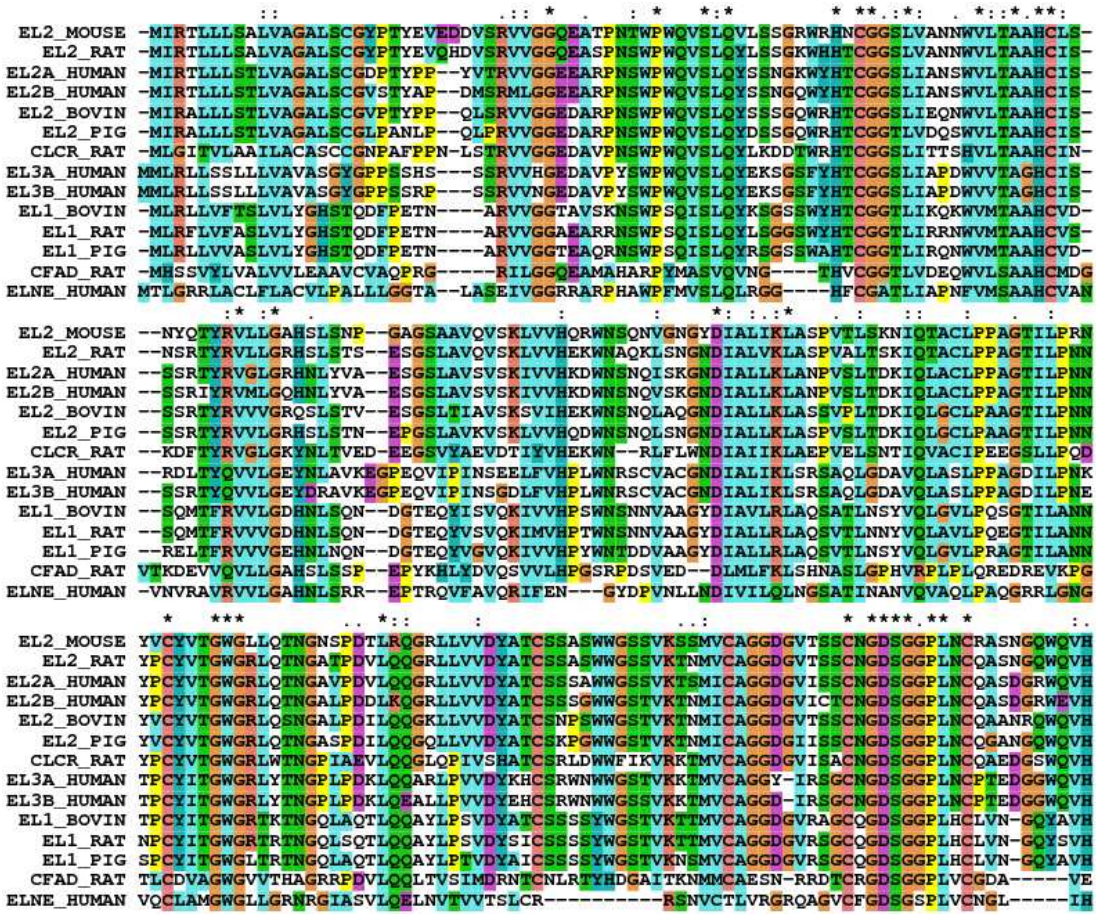

Figure 2.4. Multiple sequence alignment between some mammalian organisms generated using CLUSTALX (Thompson et al, 1997) software

Computational complexity of MSA is more challenging than the pairwise sequence alignment. The methods for producing MSA are dynamic programming methods, progressive alignment, iterative methods, and Hidden Markov Models.

2.2.1. Dynamic Programming Methods

The most direct method for producing MSA, in theory, expanding the dynamic programming approach used to generate pairwise sequence alignment. However considering time and memory limitations, this theory is not a practical solution. Because expanding the pairwise sequence alignment for n sequences, MSA has a polynomial time problem (NP-complete problem). Therefore other heuristic

methods are much more suitable for MSA. Dynamic programming methods for MSA are only used for applications which need extremely high-quality alignment or for a small number of sequences.

2.2.2. Progressive Alignment

Progressive alignment is also known as the hierarchical or tree method. It produces the final MSA by performing a series of pairwise sequence alignment on less closely related sequences. This method starts with aligning most closely related sequences. Initially, most closely related sequences are determined by an efficient clustering method such as neighbor-joining. Therefore the quality of the final MSA is dependent to the selection of most closely related sequences. Progressive alignment methods will be mentioned in detail in Section 4. The most popular algorithms that uses this method are Clustal family and T-Coffee algorithms.

2.2.3. Iterative Methods

Iterative methods are very similar to progressive methods. But iterative methods have less error rate. It based on realigning all sequences repeatedly with adding a new sequence at each aligning step to the growing MSA.

2.2.4. Hidden Markov Models

Hidden Markov Models (Rabiner, 1989) are probabilistic method that can assign likelihoods to all possible combinations of gaps, matches, and mismatches to determine the most likely MSA or set of possible MSAs. Hidden Markov Models use a cyclic graph known as a partial-order graph, which consists of a series of nodes which represent possible entries in the columns of a MSA.

2.3. Software Tools for Sequence Alignment

There are many computer applications for various types of sequence alignment. Most popular of these softwares for pairwise sequence alignment are listed in Table 2.1.

Name	Description	Reference
BLAST	*k*-tuple local search	(Altschul et al, 1990)
BLASTZ	Seeded pattern-matching	(Schwartz et al, 2003)
DNADot	Web-based dot-plot tool	(Bowen, 1998)
DOTLET	Java-based dot-plot tool	(Junier and Pagni, 2000)
Dotter	Dot matrix plot tool	(Sonnhammer and Durbin, 1995)
FASTA	*k*-tuple local search	(Lipman and Pearson, 1985)
JAligner	Open source Java implementation of Smith-Waterman	(Moustafa, 2007)
LALIGN	Local similarity (same algorithm as SIM)	(Pearson, 1991)
Matcher (EMBOSS)	Memory-optimized needleman but slow dynamic programming (based on LALIGN)	(Longden, 1999)
MCALIGN2	explicit models of indel evolution	(Wang et al, 2006)
MUMmer	Suffix-Tree based	(Kurtz et al, 2004)
Needle (EMBOSS)	Needleman-Wunsch dynamic programming	(Bleasby, 1999a)
Ngila	logarithmic and affine gap costs and explicit models of indel evolution	(Cartwright, 2007)
PatternHunter	Seeded pattern-matching	(Ma et al, 2002)
ProbA (also propA)	Stochastic partition function sampling via dynamic programming	(Mückstein et al, 2002)
REPuter	Suffix-Tree based	(Kurtz et al, 2001)
SEQALN	Various dynamic programming	(Hardy and Waterman, 1996)
SIM	Local similarity	(Huang and Miller, 1991)
SLIM Search	Ultra-fast blocked alignment	(SLIM Search, 2007)
SSEARCH	Smith-Waterman database search	(JIBRD, 2007)
Tranalign (EMBOSS)	Aligns nucleic acid sequences given a protein alignment	(Williams, 2002)
Water (EMBOSS)	Smith-Waterman dynamic programming	(Bleasby, 1999b)
Wordmatch (EMBOSS)	*k*-tuple pairwise match	(Longden, 1998)
YASS	Seeded pattern-matching	(Noe and Kucherov, 2006)

Table 2.1. Well known pairwise sequence alignment tools

Similar to pairwise sequence alignment, there are also many computer applications to generate MSA. Table 2.2 shows the well known tools for producing MSA.

Name	Description	Reference
ABA	A-Bruijn alignment	(Raphael et al, 2004)
ALE	manual alignment ; some software assistance	(Blandy and Fogel, 2007)
AMAP	Sequence annealing	(Schwartz and Pachter, 2006)
BAli-Phy	Tree+Multi alignment ; Probabilistic/Bayesian ; Joint Estimation	(Redelings and Suchard, 2007)
CHAOS/DIALIGN	Iterative alignment	(Brudno et al, 2003a)
ClustalX	CLUSTALW with graphical user interface	(Thompson et al, 1997)
ClustalW	Progressive alignment	(Thompson et al, 1994)
CodonCode Aligner	Multi alignment; ClustalW & Phrap support	(CodonCode Corp., 2007)
DNA Baser	Multi alignment	(Gabriel, 2005)
Ed'Nimbus	Seeded filtration	(Peterlongo et al, 2006)
Kalign	Progressive alignment	(Lassmann and Sonnhammer, 2005)
MAVID	Progressive alignment	(Bray and Pachter, 2004)
MAFFT	Progressive/iterative alignment	(Katoh et al, 2005)
MSA	Dynamic programming	(Lipman et al, 1989)
MULTALIN	Dynamic programming/clustering	(Corpet, 1988)
Multi-LAGAN	Progressive dynamic programming alignment	(Brudno et al, 2003b)
MUSCLE	Progressive/iterative alignment	(Edgar, 2004a) (Edgar, 2004b)
ProbCons	Probabilistic/consistency	(Do et al, 2005)
PSAlign	Alignment preserving non-heuristic	(Sze et al, 2006)
RevTrans	Combines DNA and Protein alignment.	(Wernersson and Pedersen, 2003)
SAGA	Sequence alignment by genetic algorithm	(Notredame and Higgins, 1996)
SAM	Hidden Markov model	(Hughey and Krogh, 1995)
T-Coffee	More sensitive progressive alignment	(Notredame et al, 1998)

Table 2.2. Popular multiple sequence alignment tools

2.4. Hardware Tools For Sequence Alignment

Generally hardware tools are implemented using the techniques of dynamic programming approach. There are three milestones for hardware architectures designed specifically for sequence alignment operations using Smith Waterman algorithm. These are mentioned as the followings;

- *Splash II* (Gokhale et al, 1991): It is a programmable linear logic array. The systolic architecture includes many processing elements connected in a one dimensional array. Each processing element has three components, a Xilinx FPGA, local memory, and a floating point chip.
- *Fusion 150* (Schmidt et al, 2002): it is a parallel computer with a linear SIMD (single instruction, multiple data) stream array of many processing elements on a single chip. This implementation provides a remarkable runtime saving for large scale database scanning.
- *UCSC Kestrel* (Di Bias et al, 2005): it is a single board coprocessor with a 512 element linear array of 8 bit SIMD processing elements.

Table 2.3. presents the performance and technology data for these architectures including DeCypher (Decypher Technologies, Ltd., 2007) and GeneMatcher2 (Paracel, Inc., 2005), which are commercial hardware solutions for sequence alignment operations. Performances are measured in GCUPS (Giga Cell Updates per Second).

System	Year	Technology	GCUPS	Reference
Splash II	1993	1000 nm FPGA	3	(Gokhale et al, 1991)
UCSC Kestrel	1997	500 nm ASIC	0,4	(Di Bias et al, 2005)
Fusion 150	2000	250 nm ASIC	2,5	(Schmidt et al, 2002)
DeCypher	2001	180 nm FPGA	7,5	(Decypher Technologies, Ltd., 2007)
GeneMatcher2	2001	120 nm ASIC	16	(Paracel, Inc., 2005)

Table 2.3. Hardware tools for sequence alignment operations (Hireche et. al., 2006)

The number of several academic and commercial hardware implementations for sequence alignment operations growing rapidly as the technologies of ASIC and FPGA platforms improve.

3. SMITH-WATERMAN ALGORITHM

The identification of the most similar subsequences within long sequences, that is local sequence alignment, is an important task for molecular biologist to perform molecular sequence analysis. One of the most popular algorithm to perform local sequence alignment between DNA, RNA and protein sequences is the Smith-Waterman Algorithm. The algorithm developed by T.F. Smith and M.S. Waterman (Smith and Waterman, 1981). This algorithm is developed by extending the main ideas of several studies such as Fitch (Fitch, 1966), Dayhoff (Dayhoff, 1969), Needleman-Wunsch (Needleman and Wunsch, 1970), Sankoff (Sankoff, 1972), Reichert et al (Reichert et al, 1973), Sellers (Sellers, 1974), Waterman et al (Waterman et al, 1976), Beyer et al (Beyer et al, 1979), and Smith et al (Smith et al, 1981). The algorithm simply compares all the characters of query sequence to all the characters of subject sequence to find the most similar subsequences between them. It does not use any heuristic, statistical or probabilistic method, but it uses dynamic programming approach. Therefore its results guarantee the best optimal alignment.

3.1. The Algorithm

Let $\boldsymbol{P = p_1 p_2 p_3 \dots \dots p_m}$ and $\boldsymbol{Q = q_1 q_2 q_3 \dots \dots q_n}$ are any character sequences of DNA, RNA, or protein sequences, called as subject and query sequence, respectively. The algorithm builds a score matrix using the following equations to find the most similar regions;

$$\boldsymbol{h(i,j) = max} \begin{cases} \boldsymbol{0} \\ \boldsymbol{h(i-1,j-1) + s(i,j)} \\ \boldsymbol{e(i,j)} \\ \boldsymbol{f(i,j)} \end{cases}$$

for $\boldsymbol{1 \leq i \leq m}$ and $\boldsymbol{1 \leq j \leq n}$ **(3.1)**

$$\boldsymbol{e(i,j) = max}\begin{cases}\boldsymbol{h(i-1,j) - g_o} \\ \boldsymbol{e(i-1,j) - g_e}\end{cases} \quad \text{for } \boldsymbol{1 \le i \le m} \text{ and } \boldsymbol{1 \le j \le n} \quad \textbf{(3.2)}$$

$$\boldsymbol{f(i,j) = max}\begin{cases}\boldsymbol{h(i,j-1) - g_o} \\ \boldsymbol{f(i,j-1) - g_e}\end{cases} \quad \text{for } \boldsymbol{1 \le i \le m} \text{ and } \boldsymbol{1 \le j \le n} \quad \textbf{(3.3)}$$

where $\boldsymbol{s(i,j)}$ is the substitution cost between the corresponding characters, $\boldsymbol{g_o}$ is the gap opening cost between aligning sequences, and $\boldsymbol{g_e}$ is the cost of expanding a gap already opened between aligning sequences. These parameters are determined before the sequence alignment, according to type of sequences. Generally for protein sequences, gap opening cost is equal to 5 and gap extension cost is equal to 1, and substitution score is selected using a matrix, called substitution matrix. These matrices are used to improve the alignment quality based on biological facts. Well known substitution matrices are The Dayhoff PAM250 (Dayhoff et al, 1978) matrix and The BLOSUM62 (Henikoff and Henikoff, 1992) matrix, given in Figure 3.1 and Figure 3.2, respectively.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
Ala (A)	2																			
Arg (R)	-2	6																		
Asn (N)	0	0	2																	
Asp (D)	0	-1	2	4																
Cys (C)	-2	-4	-4	-5	12															
Gln (Q)	0	1	1	2	-5	4														
Glu (E)	0	-1	1	3	-5	2	4													
Gly (G)	1	-3	0	1	-3	-1	0	5												
His (H)	-1	2	2	1	-3	3	1	-2	6											
Ile (I)	-1	-2	-2	-2	-2	-2	-2	-3	-2	-5										
Leu (L)	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	6									
Lys (K)	-1	3	1	0	-5	1	0	-2	0	-2	-3	5								
Met (M)	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	6							
Phe (F)	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	9						
Pro (P)	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	6					
Ser (S)	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	2				
Thr (T)	1	-2	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	3			
Trp (W)	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	17		
Tyr (Y)	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	10	
Val (V)	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	4

Figure 3.1. The Dayhoff PAM250 substitution matrix

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
Ala (A)	4																			
Arg (R)	-1	5																		
Asn (N)	-2	0	6																	
Asp (D)	-2	-2	1	6																
Cys (C)	0	-3	-3	-3	9															
Gln (Q)	-1	1	0	0	-3	5														
Glu (E)	-1	0	0	2	-4	2	5													
Gly (G)	0	-2	0	-1	-3	-2	-2	6												
His (H)	-2	0	1	-1	-3	0	0	-2	8											
Ile (I)	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu (L)	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys (K)	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met (M)	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe (F)	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro (P)	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser (S)	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr (T)	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp (W)	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr (Y)	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val (V)	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Figure 3.2. The BLOSUM62 substitution matrix

If gap opening, $\boldsymbol{g_o}$, and gap extension, $\boldsymbol{g_e}$, costs are selected as different then the alignment is named as ***affine gap penalty***, otherwise if they are equal to each other then the alignment named as ***linear gap penalty***, and the equations can be simplified as the following;

$$\boldsymbol{h(i,j) = max}\begin{cases} \boldsymbol{0} \\ \boldsymbol{h(i-1,j-1) + s(i,j)} \\ \boldsymbol{h(i-1,j) + g_o} \\ \boldsymbol{h(i,j-1) + g_o} \end{cases}$$

for $\boldsymbol{1 \leq i \leq m}$ and $\boldsymbol{1 \leq j \leq n}$ **(3.4)**

Biologically affine gap penalty is more meaningful then the linear gap penalty, but it requires more time and memory sources.

Because of the dependencies between the neighbor scores in the matrix, before building the matrix the following initializations has to be set;

$$\boldsymbol{h(i,0) = e(i,0) = f(i,0) = 0} \qquad \text{for } \boldsymbol{0 \leq i \leq m} \qquad \textbf{(3.5)}$$

$$\boldsymbol{h(0,j) = e(0,j) = f(0,j) = 0} \qquad \text{for } \boldsymbol{0 \leq j \leq n} \qquad \textbf{(3.6)}$$

Application of equations builds the score matrix to assist the alignment operation.

The meaning of selection for each argument can be explained as follows;

$\boldsymbol{0}$: Prevents the negative values in the matrix. Needleman-Wunsch algorithm does not prevent the negative values, since it generates global sequence alignment.

$\boldsymbol{h(i-1,j-1) + s(i,j)}$: There is no need to placing a gap, substitution or matching occurs.

$\boldsymbol{e(i,j)}$: A gap is placed or expanded in the subject sequence.

$\boldsymbol{f(i,j)}$: A gap is placed or expanded in the query sequence.

Any line includes g_o : There is a need to opening a gap in the subject or query sequence.

Any line includes g_e : There is a need to expanding a gap already opened in subject or query sequence.

After the building the score matrix an additional operation called trace-back operation has to be performed in order to complete final alignment.

Trace-back operation is following the maximum score along the matrix from right-bottom to left-top direction to create a path. Then this path declares the alignment between the subject and query sequences. However, if a biological database is scanned for similar sequences to the query sequence, then there is no need to perform trace-back operation. Because the maximum score in the matrix, which is called similarity score, will be sufficient to grab the similar sequences to the query sequence.

3.2. An Example of Smith-Waterman Algorithm

This section gives an example of Smith-Waterman Algorithm step by step in detailed explanation. In this example small subsequences of ACRO_HUMAN and ACRO_MOUSE protein sequences will be aligned and linear gap penalty model will be used.

3.2.1. Determining the Sequences and the Parameters

Let the sequences to be aligned are as the followings;

P = SLQIFRYNSH

Q = SAQLGIFTSH

The cost parameters are selected as follows; gap opening cost is 5 and the substitution matrix is BLOSUM62.

3.2.2. Initialization the Score Matrix

Because this example aligns the sequences whose lengths are 10 and 10, respectively, an empty 11x11 matrix is created and their first row and first column are filled with zeros according to the initialization equations. This empty matrix is shown in Figure 3.3.

	ϕ	**S**	**L**	**Q**	**I**	**F**	**R**	**Y**	**N**	**S**	**H**
ϕ	0	0	0	0	0	0	0	0	0	0	0
S	0										
A	0										
Q	0										
L	0										
G	0										
I	0										
F	0										
T	0										
S	0										
H	0										

Figure 3.3. Initialization the Smith-Waterman score matrix

3.2.3. Building the Matrix

After initialization step, the matrix is ready for filling with the score using the linear gap penalty model equation, from left-top to right-bottom direction.

	ϕ	S	L	Q	I	F	R	Y	N	S	H
ϕ	0	0	0	0	0	0	0	0	0	0	0
S	0	4	0	0	0	0	0	0	1	4	0
A	0	1	3	0	0	0	0	0	0	2	2
Q	0	0	0	8	3	0	1	0	0	0	2
L	0	0	4	3	10	5	0	0	0	0	0
G	0	0	0	2	5	7	3	0	0	0	0
I	0	0	2	0	6	5	4	2	0	0	0
F	0	0	0	0	1	12	7	7	2	0	0
T	0	1	0	0	0	7	11	6	7	3	0
S	0	4	0	0	0	2	6	9	7	11	6
H	0	0	1	0	0	0	2	8	10	6	19

Figure 3.4. Filling the score matrix

3.2.4. Trace-Back Operation

The next step is the performing the trace-back operation using score matrix. Trace-Back operation follows the wise direction; this means that it follows the path from right-bottom to left-top direction with guidance of maximum scores of each neighborhood. Figure 3.5 shows this direction;

	ϕ	S	L	Q	I	F	R	Y	N	S	H
ϕ	0	0	0	0	0	0	0	0	0	0	0
S	0	4	0	0	0	0	0	0	1	4	0
A	0	1	3	0	0	0	0	0	0	2	2
Q	0	0	0	8	3	0	1	0	0	0	2
L	0	0	4	3	10	5	0	0	0	0	0
G	0	0	0	2	5	7	3	0	0	0	0
I	0	0	2	0	6	5	4	2	0	0	0
F	0	0	0	0	1	12	7	7	2	0	0
T	0	1	0	0	0	7	11	6	7	3	0
S	0	4	0	0	0	2	6	9	7	11	6
H	0	0	1	0	0	0	2	8	10	6	19

Figure 3.5. Trace-back operation

3.2.5. Completing the Alignment

Trace-back operation represents the path which the alignment must follow. Using this path the alignment can be produced with placing a gap to the subject sequence if the path follows the left neighbour, or with placing a gap to the query sequence if the path follows the upper neighbour. Then the final alignment is as the following;

S L Q - - I F R Y N S H
S A Q L G I F T - - S H

Another important result obtained by Smith-Waterman algorithm is the maximum score of overall matrix, which is 19 for this example. This maximum score is called similarity score. The similarity score can indicate how much the sequences are similar to each other. Using this similarity score, molecular biologist can find the most similar sequences to a query sequence in a large database without performing the complete sequence alignment. This application is one of the most important operation in bioinformatics.

The following table shows some scores between several organisms' FA9 proteins to indicate this relationship between maximum score and biological similarity.

	Human	**Bovin**	**Canfa**	**Cavpo**	**Mouse**	**Pig**	**Rabbit**	**Rat**	**Sheep**
Human	2503								
Bovin	1945	2275							
Canfa	2139	1945	2452						
Cavpo	1141	1126	1134	1540					
Mouse	2044	1896	2050	1173	2479				
Pig	1244	1227	1242	1108	1170	1445			
Rabbit	1201	1182	1183	1090	1190	1200	1476		
Rat	1200	1170	1178	1184	1389	1167	1191	1503	
Sheep	1214	1397	1204	1105	1158	1221	1165	1147	1469

Table 3.1. Similarity scores between FA9 proteins of some mammalian organisms

According to the above figure, it can be said that FA9_HUMAN protein is most related with the FA9_CANFA and the FA9_MOUSE proteins in the several mammalian organisms.

4. THE PROPOSED SMITH-WATERMAN DESIGNS

In this thesis work, several hardware units are designed to perform various sequence alignment types using dynamic programming approach and Smith-Waterman Algorithm. These hardware designs are developed as to be reconfigurable for alignment parameters (gap opening cost, gap extension cost, substitution matrices, etc…). Also they are designed so that time and space requirements are minimum. The designed hardware designs are presented in (Gok and Yilmaz, 2006a) and (Gok and Yilmaz, 2006b)

Generally, while mapping the Smith-Waterman Algorithm into a hardware, systolic architectures are used. The reason of this is the fact that the main idea of the dynamic programming approach. In dynamic programming approach, the problem is divided into smaller parts so that every part performs the same work to solve overall problem. Similarly, in systolic architectures there are many numbers of identical hardware cells which are communicated to each other. These cells perform a specific calculation to solve the overall problem.

In this section, firstly the overall systolic architecture will be given, then the cells in this architectures will be mentioned for pair-wise sequence alignment and multiple sequence alignment.

4.1. The Systolic Architecture

To map the Smith-Waterman Algorithm into hardware, generally a systolic architecture is used. In this architecture, there are many number of hardware cells which calculate the following equations mentioned in Section 3.

$$\boldsymbol{h(i,j) = max}\begin{cases} \mathbf{0} \\ \boldsymbol{h(i-1,j-1) + s(i,j)} \\ \boldsymbol{e(i,j)} \\ \boldsymbol{f(i,j)} \end{cases}$$

for $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{m}$ and $\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n}$ **(4.1)**

$$e(i,j) = max\begin{cases} h(i-1,j) - g_o \\ e(i-1,j) - g_e \end{cases} \quad \text{for } 1 \le i \le m \text{ and } 1 \le j \le n \qquad (4.2)$$

$$f(i,j) = max\begin{cases} h(i,j-1) - g_o \\ f(i,j-1) - g_e \end{cases} \quad \text{for } 1 \le i \le m \text{ and } 1 \le j \le n \qquad (4.3)$$

$$h(i,0) = e(i,0) = f(i,0) = 0 \quad \text{for } 0 \le i \le m \qquad (4.4)$$

$$h(0,j) = e(0,j) = f(0,j) = 0 \quad \text{for } 0 \le j \le n \qquad (4.5)$$

Smith-Waterman score matrix is divided into time and space. Therefore, each row in the matrix is replaced with time, and each column replaced with space. That means, each cell is considered as a character of query sequence, and each clock cycle of a cell is considered as a character of subject sequence. This approach provides parallelization to the computing, since in every clock cycle, there many sub-calculations made in the identical cells or processing elements of the systolic architecture. These considerations are illustrated in Figure 4.1.

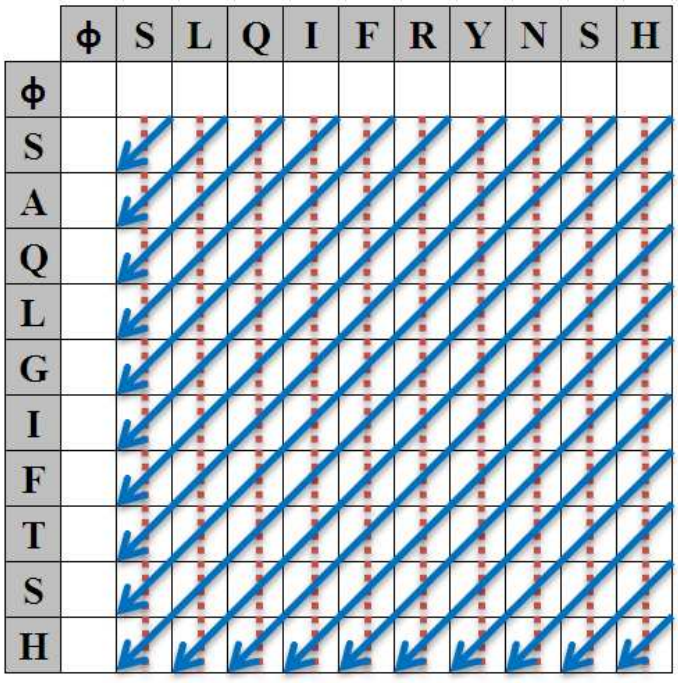

Figure 4.1. Adapting the systolic architecture to Smith-Waterman score matrix

In Figure 4.1, red dotted lines indicate a character of the query sequence (SLQIFRYNSH), and blue arrows indicate a clock cycle for the subject sequence (SAQLGIFTSH). Every single point that dotted lines and arrows crosses, a score is calculated in the Smith-Waterman cells. These scores are passed through time and cells to calculate the total similarity score.

As shown in Figure 4.1, this algorithm has the following time and space requirements. For $\boldsymbol{m}$ length query sequences and $\boldsymbol{n}$ character length subject sequence ($\boldsymbol{mxn}$ matrix), the algorithm needs;

- $\boldsymbol{n}$ Smith-Waterman cells,
- $\boldsymbol{m} + \boldsymbol{n}$ clock cycles.

The initialization step which is shown in the matrix as ϕ rows and columns does not consume time and space resources, since the all registers are cleared to zeros at the beginning and the first cell inputs are all zeros.

With this implementation each query character is compared with all subject characters so that every single possible sequence alignment is scored in the matrix. Then the path consists the maximum scores in the matrix will be the optimum alignment. When mapping this adaption into hardware, each character of query sequence is stored into a register in a cell (red dotted lines). Then the calculation will start with passing each character of subject sequence through cells by clock cycles (blue arrows). Hardware implementation of this approach is shown in the following Figure 4.2.

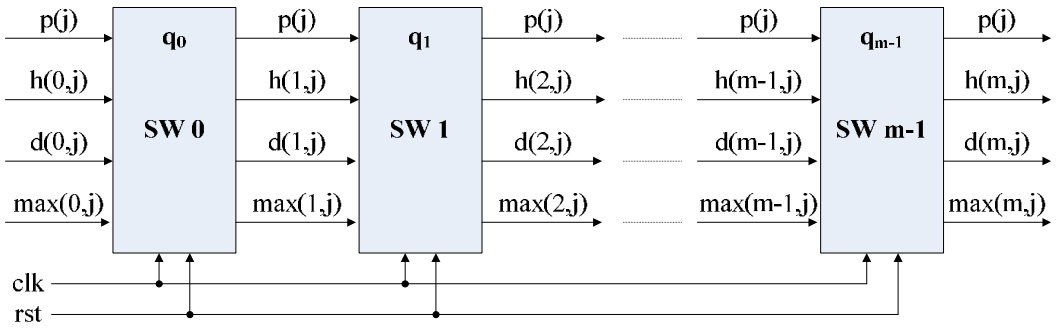

Figure 4.2. Systolic architecture of Smith-Waterman algorithm

The lines, which connect neighbor cells, carry the following signals;

- $\boldsymbol{h(i,j)}$ **:** The computed temporarily scores of the corresponding cells. The width of these bit vectors may differ as the precision of the score changes.
- $\boldsymbol{d(i,j)}$ **:** 2-bit difference values. For a Smith-Waterman cell, this bit vectors represent the difference between input scores and output scores. This signal will be explained in detailed later.
- **max(i,j) :** The calculated temporarily maximum score of the corresponding cells. The width of this signal is equal to the width of h(i,j) signal.
- $\boldsymbol{p(i,j)}$ **:** These signals are used to pass a character of the sequences for each clock cycle. The width of these signals may vary according the type of the sequences, DNA, protein etc…
- $\boldsymbol{rst}$ **:** Global asynchronous reset signal. It is set to one at the beginning of operation to clear all the registers in overall systolic architecture.
- $\boldsymbol{clk}$ **:** Global clock signal. At rising edges, all the cells compute necessary values, and pass the character of the sequences to the next cell.

After completing the sequence alignment operations, $\boldsymbol{max(m,j)}$ will be the similarity score between the subject and query sequence.

If the length of the subject sequence is larger than the number of cells, then the calculation can be completed with dividing the subject sequence. Then the intermediate values may be needed. These values are the outputs of the systolic architecture shown as $\boldsymbol{h(m,j)}$, $\boldsymbol{d(m,j)}$, and $\boldsymbol{max(m,j)}$. Therefore these signals should be used the input of the systolic array to continue the calculation for the rest of subject sequence.

4.2. The Proposed Smith-Waterman Cell Designs for Pairwise Sequence Alignment

The systolic architecture shown in Figure 4.2 is used by most of the Smith-Waterman hardware designs (Yamaguchi et al, 2002), (Yu et al, 2003). The performance differences between designs are directly related to the design of the Smith-Waterman cells. In this thesis, an optimized design is given. In general previous hardware implementations directly map the equations of the algorithm to design the Smith-Waterman cell. This approach is very inefficient because of the following reasons:

1. The direct implementation of equations is very inefficient, because as the precision of the design increases, the size of adders, registers, and maximum generators increase as well. For example if the precision is 16-bit, then all units must be 16-bit size. To overcome this complexity the following optimizations are used in the proposed design.

   For $(\boldsymbol{i,j})$ indexed cell, the input score $\boldsymbol{h(i-1,j-1)}$ always varies from the output score $\boldsymbol{h(i,j)}$ by a small amount. This is because $\boldsymbol{h(i,j)}$ can only be in the range of $[\boldsymbol{h(i-1,j-1)-g_o},\ \boldsymbol{h(i-1,j-1)+\max\,(s(i,j))}]$. Since $\boldsymbol{g_o}$ is $\mathbf{5}$ for most applications, and $\boldsymbol{\max\,(s(i,j))}$ is $\mathbf{11}$ from the maximum substitution value in BLOSUM62 Matrix, 4-bit calculations is sufficient. If the calculations are made in 4-bit precision, there is a need to keep the carries or borrows. In the proposed design these intermediate values are stored in $\boldsymbol{d(i,j)}$ and $\boldsymbol{d(i,j-1)}$ vectors. Then $\boldsymbol{d(i,j)}$ and $\boldsymbol{d(i,j-1)}$ can be three different values; one "01" if there is a carry, minus one "11" if there is a barrow, or zero "00" if there is not neither carry nor barrow. The mathematical representations of these vectors are as followings;

$$\boldsymbol{d(i,j-1)=h(i,j)_{[5:4]}-h(i-1,j)_{[5:4]}} \qquad \textbf{(4.6)}$$

$$\boldsymbol{d(i,j)=h(i,j)_{[5:4]}-h(i,j-1)_{[5:4]}} \qquad \textbf{(4.7)}$$

2. For affine gap penalty model there is no need to calculate the maximums of $\boldsymbol{e(i,j)}$ and $\boldsymbol{f(i,j)}$ vectors. Because these vectors represents the scores of gap opening or extension in query and subject sequences. The maximum selection of $\boldsymbol{e(i,j)}$ and $\boldsymbol{f(i,j)}$ represents if there is an opening a gap or extending a gap already opened. Therefore instead of maximum calculation, with a multiplexer these scores can be selected properly. The selection signals of the multiplexers are generated using the previous calculations. If a gap is opened, then the selection signal is set, otherwise it is cleared.
3. The final optimization is valid for only linear gap penalty models. The substitution matrix is loaded into look up tables with values subtracted with gap opening cost. This does not affect the maximum calculations. Then there is no need to subtract the gap opening cost from the score. If the score from $\boldsymbol{h(i-1,j-1)+s(i,j)}$ is selected then gap opening cost can be added in final addition, which is already performed even there is no optimization.

Considering these optimization ideas, the Smith-Waterman cells can be designed for two different calculation methods; linear gap penalty cell, affine gap penalty cell in an efficient way.

4.2.1. Linear Gap Penalty Cell

The simplified linear gap penalty model equations are reminded as the followings;

$$\boldsymbol{h(i,j)=max}\begin{cases}\boldsymbol{0}\\ \boldsymbol{h(i-1,j-1)+s(i,j)}\\ \boldsymbol{h(i-1,j)+g_o}\\ \boldsymbol{h(i,j-1)+g_o}\end{cases}$$

for $\boldsymbol{1\le i\le m}$ and $\boldsymbol{1\le j\le n}$ **(4.8)**

$$\boldsymbol{h(i,0) = 0} \quad \text{for } \boldsymbol{0 \leq i \leq m} \tag{4.9}$$

$$\boldsymbol{h(0,j) = 0} \quad \text{for } \boldsymbol{0 \leq j \leq n} \tag{4.10}$$

where $\boldsymbol{m}$ is the length of the query sequence, and $\boldsymbol{n}$ is the length of the subject sequence.

According to the equations and the optimization ideas, the proposed linear gap penalty model Smith-Waterman cell is designed as shown in Figure 4.3.

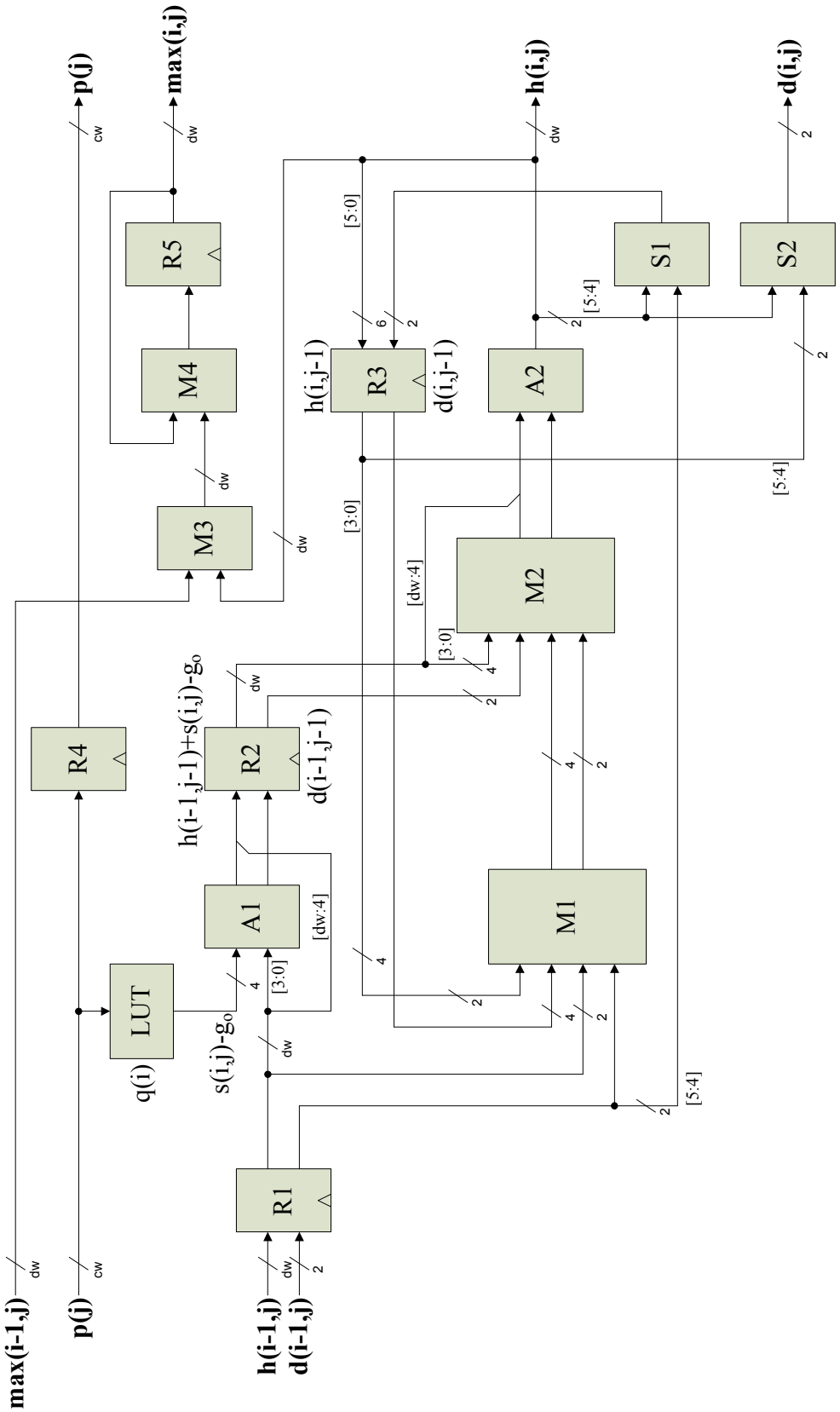

Figure 4.3. Linear gap penalty implementation of Smith-Waterman cell

In Figure 4.3, $\boldsymbol{dw}$ is the data width which determines the precision of the similarity score, $\boldsymbol{cw}$ is the character width which is used to represent the sequence code in binary numbering system. $\boldsymbol{cw}$ varies according to the alignment type. For example, when aligning protein nucleotide sequences, there are 20 types of protein characters. Therefore $\boldsymbol{cw}$ **s**hould be equal to 5. Also there may need to add some control signals to the $\boldsymbol{p(j)}$ characters of the subject sequence to control the overall systolic architecture, since $\boldsymbol{cw}$ is the only signal which passes through all Smith-Waterman cells without any change. Therefore this width may increase.

In Figure 4.3, the inputs and the outputs of the circuit are typed in bold font. In addition to the schematic of the circuit of linear gap implementation shown in Figure 4.3., there are some other signals and logics, which are not presented in the figure. These signal are $\boldsymbol{clk}$ and $\boldsymbol{rst}$, represented for clock and reset signals, respectively. Also there is a small control logic which is used to handle the loading operations. Loading operation is responsible to load the alignment parameters and subject sequence characters into the Smith-Waterman cells.

The followings are the detailed explanation of the units placed in the linear proposed gap penalty design;

- **R1:** (dw+2)-bit register. It stores $\boldsymbol{h(i-1,j)}$ and $\boldsymbol{d(i-1,j)}$ signals, which comes from the left neighbour cell, for one clock cycle duration.
- **R2**: (dw+2)-bit register. It stores $\boldsymbol{h(i-1,j-1)+s(i,j)-g_o}$ and $\boldsymbol{d(i-1,j-1)}$ values.
- **R3:** 8-bit register. It stores $\boldsymbol{h(i,j-1)_{[5:0]}}$ and $\boldsymbol{d(i-1,j-1)}$ signals, which comes from the same cell but after one clock cycle.
- **R4:** cw-bit register. It is used to pass the sequence character, that comes from the left neighbor cell to the right neighbor cell after one clock cycle.
- **R5:** dw-bit register. It stores the maximum score value for one clock cycle to calculate maximum score for the next cycle.
- **LUT:** Look up table which stores the substitution matrix for score calculations. It should be loaded before all circuit operation. In this design the substitution matrix has to be loaded with the values subtracted with $\boldsymbol{g_o}$ from

the values in the original substitution matrix because of the third optimization idea described in the previous section. Based on alignment requirements, LUT can be loaded different values, for example it can be loaded with The Dayhoff PAM250 Substitution Matrix or The BLOSUM62 Substitution Matrix for protein sequence alignments. Look up table also stores the $\boldsymbol{q(i)}$ query sequence characters.

- **A1:** 4-bit special adder. It performs addition of the $\boldsymbol{s(i,j) - g_o}$ with $\boldsymbol{h(i-1, j-1)}$. It also produces 2-bit carry output, named as $\boldsymbol{d(i-1, j-1)}$ for other calculations.
- **A2:** dw-bit special saturated adder. It performs the final calculation. It adds the selected score with an intermediate value. This intermediate value is determined using 2-bit carry or borrow input. Because the carry or borrow input is related to the bit with weight 2^5, this intermediate value can be calculated the following formula;

  $\boldsymbol{2^5.d + g_0}$ where $\boldsymbol{d}$ is the carry or borrow input.

  It is a saturated adder, which means that its output is never a negative value. This property takes place of 0 in the maximum selection in the equation.
- **S1:** 2-bit subtractor. It calculates $\boldsymbol{d(i, j-1)}$ according to the equation $\boldsymbol{d(i, j-1) = h(i, j)_{[5:4]} - h(i-1, j)_{[5:4]}}$.
- **S2:** 2-bit subtractor. It calculates $\boldsymbol{d(i, j)}$ according to the equation $\boldsymbol{d(i, j) = h(i, j)_{[5:4]} - h(i, j-1)_{[5:4]}}$.
- **M1:** 6-bit maximum generator. It calculates the maximum of $\boldsymbol{d(i, j-1)\&h(i, j-1)_{[3:0]}}$ and $\boldsymbol{d(i-1, j)\&h(i-1, j)_{[3:0]}}$. This calculation presented in the equation as the selection of the maximum of $\boldsymbol{h(i, j-1) + g_o}$ and $\boldsymbol{h(i-1, j) + g_o}$ values. There is no need to add $\boldsymbol{g_o}$ to these values because of the third optimization idea.
- **M2:** 6-bit maximum generator. It calculates the maximum of $\boldsymbol{d(i, j-1)\&h(i, j-1)_{[3:0]}}$, $\boldsymbol{d(i-1, j)\&h(i-1, j)_{[3:0]}}$, and $\boldsymbol{d(i-1, j-1)\&(h(i-1, j-1) + s(i, j) - g_o)_{[3:0]}}$. This calculation presented in the

equation as the selection of the maximum of $\boldsymbol{h(i,j-1)+g_o}$, $\boldsymbol{h(i-1,j)+g_o}$, and $\boldsymbol{h(i-1,j-1)+s(i,j)}$ values.

- **M3:** dw-bit maximum generator. It calculates the maximum of $\boldsymbol{h(i,j)}$ and $\boldsymbol{max(i-1,j)}$ to update the similarity score if the $\boldsymbol{h(i,j)}$ is larger than the current similarity score.
- **M4:** dw-bit maximum generator. It calculates the maximum of $\boldsymbol{h(i,j)}$, $\boldsymbol{max(i-1,j)}$, and $\boldsymbol{max(i,j-1)}$ to update the similarity score if the $\boldsymbol{h(i,j)}$ is larger than the current similarity score.

Finally the linear systolic array composed of these Smith-Waterman cells, calculates the overall similarity score between query and subject sequences in meaning of linear gap penalty parameters.

4.2.2. Affine Gap Penalty Cell

All affine gap penalty equations of Smith-Waterman algorithm are reminded as the followings;

$$\boldsymbol{h(i,j)=max}\begin{cases}\boldsymbol{0}\\ \boldsymbol{h(i-1,j-1)+s(i,j)}\\ \boldsymbol{e(i,j)}\\ \boldsymbol{f(i,j)}\end{cases}$$

for $\boldsymbol{1\le i\le m}$ and $\boldsymbol{1\le j\le n}$ **(4.11)**

$$\boldsymbol{e(i,j)=max}\begin{cases}\boldsymbol{h(i-1,j)-g_o}\\ \boldsymbol{e(i-1,j)-g_e}\end{cases}\quad \text{for } \boldsymbol{1\le i\le m} \text{ and } \boldsymbol{1\le j\le n} \quad \textbf{(4.12)}$$

$$\boldsymbol{f(i,j)=max}\begin{cases}\boldsymbol{h(i,j-1)-g_o}\\ \boldsymbol{f(i,j-1)-g_e}\end{cases}\quad \text{for } \boldsymbol{1\le i\le m} \text{ and } \boldsymbol{1\le j\le n} \quad \textbf{(4.13)}$$

$$\boldsymbol{h(i,0)=e(i,0)=f(i,0)=0}\qquad \text{for } \boldsymbol{0\le i\le m} \quad \textbf{(4.14)}$$

$$\boldsymbol{h(0,j) = e(0,j) = f(0,j) = 0} \qquad \text{for } \mathbf{0 \le j \le n} \tag{4.15}$$

where $\boldsymbol{m}$ is the length of the query sequence, and $\boldsymbol{n}$ is the length of the subject sequence.

According to the equations and the optimization ideas, the proposed affine gap penalty implementation of Smith-Waterman cell is designed as shown in Figure 4.4.

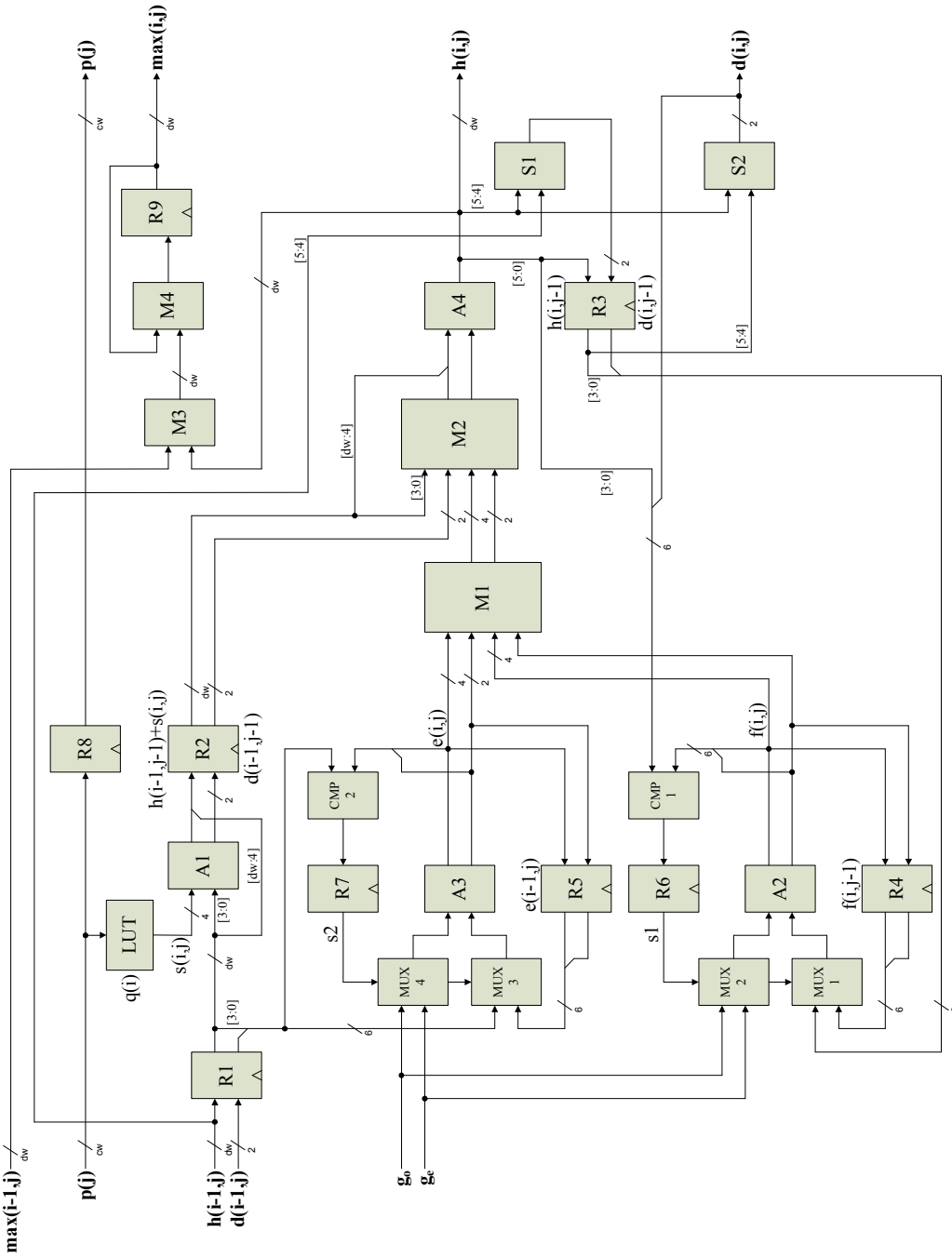

Figure 4.4. Affine gap penalty implementation of Smith-Waterman cell

All signal widths are the same with linear gap penalty cell explained before. Also additional logics and signals which are not shown in the figure is again the same with linear gap penalty cell.

The detailed explanations of the logic components are given in the followings;

- **R1:** (dw+2)-bit register. It stores $\boldsymbol{h(i-1,j)}$ and $\boldsymbol{d(i-1,j)}$ signal vectors, which comes from the left neighbour cell.
- **R2**: (dw+2)-bit register. It stores $\boldsymbol{h(i-1,j-1)+s(i,j)}$ and $\boldsymbol{d(i-1,j-1)}$ values.
- **R3:** 8-bit register. It stores $\boldsymbol{h(i,j-1)_{[5:0]}}$ and $\boldsymbol{d(i-1,j-1)}$ signals, which comes from the same cell but after one clock cycle, for one clock cycle duration.
- **R4:** 6-bit register. It stores $\boldsymbol{f(i,j-1)_{[3:0]}}$ and $\boldsymbol{df(i,j-1)}$ signals.

  R5: 6-bit register. It stores $\boldsymbol{e(i-1,j)_{[3:0]}}$ and $\boldsymbol{de(i-1,j)}$ signals.
- **R6:** A flip-flop that stores the $\boldsymbol{s1}$ control signal, which is used to determine which penalty cost will be used at the next cycle for $\boldsymbol{f(i,j)}$ calculations.
- **R7:** A flip-flop that stores the $\boldsymbol{s2}$ control signal, which is used to determine which penalty cost will be used at the next cycle for $\boldsymbol{e(i,j)}$ calculations.
- **R8:** cw-bit register. It is used to pass the sequence character, which comes from the left neighbor cell, to the right neighbor cell after one clock cycle.
- **R9:** dw-bit register. It stores the maximum score value for one clock cycle, to calculate maximum score for the next cycle.
- **LUT:** Look up table. It stores the substitution matrix for score calculations. It should be loaded before all circuit operation. According to the alignment type, LUT can be loaded different values, for example it can be loaded with the Dayhoff PAM250 substitution matrix or the BLOSUM62 substitution matrix for protein sequence alignments. Look up table also stores the $\boldsymbol{q(i)}$ query sequence characters.
- **A1:** 4-bit special adder. It performs addition of the $\boldsymbol{s(i,j)}$ with $\boldsymbol{h(i-1,j-1)}$. It also produces 2-bit carry output, named as $\boldsymbol{d(i-1,j-1)}$.
- **A2:** 4-bit special adder. It performs the addition of the $\boldsymbol{f(i,j-1)}$ or $\boldsymbol{h(i,j-1)}$ with $\boldsymbol{g_o}$ or $\boldsymbol{g_e}$ according to the control signal $\boldsymbol{s1}$ to produce $\boldsymbol{f(i,j)}$. This adder also produces the new $\boldsymbol{df(i,j)}$ carry or borrow value using the result of the addition and the input $\boldsymbol{df(i,j-1)}$ signal.

- **A3:** 4-bit special adder. It performs the addition of the $\boldsymbol{e(i-1,j)}$ or $\boldsymbol{h(i-1,j)}$ with $\boldsymbol{g_o}$ or $\boldsymbol{g_e}$ according to the control signal $\boldsymbol{s2}$ to produce $\boldsymbol{e(i,j)}$. This adder also produces the new $\boldsymbol{de(i,j)}$ carry or borrow value using the result of the addition and the input $\boldsymbol{de(i-1,j)}$ signal.
- **A4:** dw-bit special saturated adder. It performs the final calculation. It adds the selected score with an intermediate value. This intermediate value is determined using 2-bit carry or borrow input. Because the carry or borrow input is related to the bit with weight 2^5, this intermediate value is $\boldsymbol{2^5.d}$, where $\boldsymbol{d}$ is the carry or borrow input.

  It is an adder, that never produces a negative value. This property takes place of 0 in the maximum selection in the equation.
- **S1:** 2-bit subtractor. It calculates $\boldsymbol{d(i,j-1)}$ according to the equations represented as $\boldsymbol{d(i,j-1) = h(i,j)_{[5:4]} - h(i-1,j)_{[5:4]}}$.
- **S2:** 2-bit subtractor. It calculates $\boldsymbol{d(i,j)}$ according to the equations represented as $\boldsymbol{d(i,j) = h(i,j)_{[5:4]} - h(i,j-1)_{[5:4]}}$.
- **CMP1:** 6-bit comparator. It generates the control signal $\boldsymbol{s1}$ using the result of the subtraction of $\boldsymbol{df(i,j)}\&f(i,j)$ from $\mathbf{d(i,j)}\&h(i,j)$. Performing this operation produces the following selection; If a new gap is opening in the subject sequence, $\boldsymbol{s1}$ will be '0'. Otherwise, an existing gap is extending in the subject sequence, then $\boldsymbol{s1}$ will be '1'. This control signal will be used to select between $\boldsymbol{f(i,j)}$ and $\boldsymbol{h(i,j)}$.
- **CMP2:** 6-bit comparator. It generates the control signal $\boldsymbol{s2}$ using the result of the subtraction of $\boldsymbol{de(i,j)}\&e(i,j)$ from $\boldsymbol{d(i,j)}\&h(i,j)$. Performing this operation produces the following selection; if a new gap is opening in the query sequence, then $\boldsymbol{s2}$ will be '0'. Otherwise, an existing gap is extending in the query sequence, $\boldsymbol{s2}$ will be '1'. This control signal will be used to select between $\boldsymbol{e(i,j)}$ and $\boldsymbol{h(i,j)}$.
- **MUX1:** 6-bit multiplexer. It selects $\boldsymbol{f(i,j-1)}$ or $\boldsymbol{h(i,j-1)}$ according to the $\boldsymbol{s1}$ control signal.

- **MUX2:** 4-bit multiplexer. It selects $\boldsymbol{g_o}$ or $\boldsymbol{g_e}$ according to the $\boldsymbol{s1}$ control signal.
- **MUX3:** 6-bit multiplexer. It selects $\boldsymbol{e(i-1,j)}$ or $\boldsymbol{h(i-1,j)}$ according to the $\boldsymbol{s2}$ control signal.
- **MUX4:** 4-bit multiplexer. It selects $\boldsymbol{g_o}$ or $\boldsymbol{g_e}$ according to the $\boldsymbol{s2}$ control signal.
- **M1:** 6-bit maximum generator. It calculates the maximum of $\boldsymbol{d(i,j-1)\&h(i,j-1)_{[3:0]}}$ and $\boldsymbol{d(i-1,j)\&h(i-1,j)_{[3:0]}}$. This calculation presented in the equation as the selection of the maximum of $\boldsymbol{e(i,j)}$ and $\boldsymbol{f(i,j)}$ values.
- **M2:** 6-bit maximum generator. It calculates the maximum of $\boldsymbol{d(i,j-1)\&h(i,j-1)_{[3:0]}}$, $\boldsymbol{d(i-1,j)\&h(i-1,j)_{[3:0]}}$, and $\boldsymbol{d(i-1,j-1)\&(h(i-1,j-1)+s(i,j)-g_o)_{[3:0]}}$. This calculation presented in the equation as the selection of the maximum of $\boldsymbol{e(i,j)}$, $\boldsymbol{f(i,j)}$, and $\boldsymbol{h(i-1,j-1)+s(i,j)}$ values.
- **M3:** dw-bit maximum generator. It calculates the maximum of $\boldsymbol{h(i,j)}$ and $\boldsymbol{max(i-1,j)}$ to update the similarity score when $\boldsymbol{h(i,j)}$ is larger than the current similarity score.
- **M4:** dw-bit maximum generator. It calculates the maximum of $\boldsymbol{h(i,j)}$, $\boldsymbol{max(i-1,j)}$, and $\boldsymbol{max(i,j-1)}$ to update the similarity score when $\boldsymbol{h(i,j)}$ is larger than the current similarity score.

The linear systolic array composed of these Smith-Waterman cells, calculates the overall similarity score between query and subject sequences in meaning of affine gap penalty parameters. Although affine gap penalty models always more complex than linear gap penalty models, their results are more useful in biological research.

4.3. The Proposed Smith-Waterman Cell Designs for Multiple Sequence Alignment

To perform multiple sequence alignment, the previous Smith-Waterman cell designs can be modified so that they can assist to produce final multiple sequence alignment. In this work, the designed hardwares are used to speed up CLUSTALW software which is a very popular software tool used to build multiple sequence alignment.

CLUSTALW is based on progressive sequence alignment approach for multiple sequence alignment. Progressive sequence alignment procedure consists of three stages. These stages are explained briefly as the followings;

- Distance Matrix: A distance value is computed using the pair-wise sequence alignment between each pair of sequences. The computed values are stored in a matrix, called distance matrix.
- Guided Tree: This step uses the distance matrix obtained from the first step and forms a guided-tree using the neighbor-joining method. The guided-tree is used to find out closely related sequences or a group of sequences that are aligned progressively in the last step to form the final multiple sequence alignment.
- Progressive Alignment: In the final step, pair-wise sequence alignment is performed between most related sequences to get final multiple sequence alignment.

Oliver et al reported that the most time consuming task is the first step (Oliver et al, 2005a). The following table indicates the distribution of overall calculation time along the steps.

Number of Sequences	Distance Matrix	Guided Tree	Progressive Alignment
100	87,10%	0,03%	12,80%
200	93,10%	0,07%	6,80%
400	92,10%	0,25%	7,70%
800	92,30%	0,54%	7,20%

Table 4.1. Distribution of overall calculation time of CLUSTALW (Oliver et al., 2005)

Therefore it is obvious that speeding up the first step is very important to decrease the overall calculation time.

To build the distance matrix the following formula is used for a set of sequences $\boldsymbol{S} = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \dots\dots, \boldsymbol{S_n}\}$.

$$\boldsymbol{d(S_i, S_j) = 1 - \frac{nid(S_i, S_j)}{min(l_i, l_j)}} \qquad \textbf{(4.16)}$$

where $\boldsymbol{d(S_i, S_j)}$ is the distance value between two sequences $\boldsymbol{S_i}$ and $\boldsymbol{S_j}$, $\boldsymbol{nid(S_i, S_j)}$ is the number of identical characters in the optimal pair-wise alignment between two sequences $\boldsymbol{S_i}$ and $\boldsymbol{S_j}$, and $\boldsymbol{min(l_i, l_j)}$ is the minimum length of two sequences $\boldsymbol{S_i}$ and $\boldsymbol{S_j}$.

The computational complexity comes from obtaining $\boldsymbol{nid(S_i, S_j)}$ values, since it has to be computed after performing pair-wise sequence alignment between each pair of sequences. In (Oliver et al, 2005a), Oliver et al reported an equation set which computes $\boldsymbol{nid(S_i, S_j)}$ values without performing the actual alignment. These equations and their hardware implementations will be mentioned in the following sections. Multiple sequence alignment can be performed with using either linear or affine gap penalty model parameters.

4.3.1. Linear Gap Penalty Multiple Sequence Alignment

To compute $\boldsymbol{nid}(\boldsymbol{S_i}, \boldsymbol{S_j})$ values without performing the actual alignment the following equations are used for linear gap penalty.

$$\boldsymbol{h(i,j) = max}\begin{cases} \mathbf{0} \\ \boldsymbol{h(i-1,j-1) + s(i,j)} \\ \boldsymbol{h(i-1,j) + g_o} \\ \boldsymbol{h(i,j-1) + g_o} \end{cases}$$

for $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{m}$ and $\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n}$ **(4.17)**

$$\boldsymbol{n(i,j)} = \begin{cases} \mathbf{0} & if\ \boldsymbol{h(i,j)} = \mathbf{0} \\ \boldsymbol{n(i-1,j-1) + m(i,j)} & if\ \boldsymbol{h(i,j)} = \boldsymbol{h(i-1,j-1) + s(i,j)} \\ \boldsymbol{n(i-1,j)} & if\ \boldsymbol{h(i,j)} = \boldsymbol{h(i-1,j) + g_o} \\ \boldsymbol{n(i,j-1)} & if\ \boldsymbol{h(i,j)} = \boldsymbol{h(i,j-1) + g_o} \end{cases}$$

for $\mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{m}$ and $\mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n}$ **(4.18)**

$$\boldsymbol{m(i,j)} = \begin{cases} \mathbf{1} & if\ \boldsymbol{s(i) = s(j)} \\ \mathbf{0} & otherwise \end{cases} \quad \text{for } \mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{m} \text{ and } \mathbf{1} \leq \boldsymbol{j} \leq \boldsymbol{n} \quad \textbf{(4.19)}$$

$$\boldsymbol{h(i,0) = n(i,0) = 0} \qquad \text{for } \mathbf{0} \leq \boldsymbol{i} \leq \boldsymbol{m} \quad \textbf{(4.20)}$$

$$\boldsymbol{h(0,j) = n(0,j) = 0} \qquad \text{for } \mathbf{0} \leq \boldsymbol{j} \leq \boldsymbol{n} \quad \textbf{(4.21)}$$

where $\boldsymbol{m}$ is the length of the query sequence, and $\boldsymbol{n}$ is the length of the subject sequence.

First part of these equations are the same with pair-wise sequence alignment. The new equations are added to compute number of identical characters. Performing these equations build a new matrix, called $\boldsymbol{n(i,j)}$ matrix. This maximum value of this matrix is equal to number of identical characters of the alignment, $\boldsymbol{nid}(\boldsymbol{S_i}, \boldsymbol{S_j})$. Therefore, this value can be computed while performing the alignment, instead of

producing the actual alignment. This saves time for multiple sequence alignment. An example of computing $\boldsymbol{nid}(\boldsymbol{S_i}, \boldsymbol{S_j})$ value is given in the followings;

Consider the alignment between two sequences P=SLQIFRYNSH and Q=SAQLGIFTSH, mentioned in the Section 3. The $\boldsymbol{h}(\boldsymbol{i}, \boldsymbol{j})$ matrix of these sequences is given in the Figure 4.5.

	ϕ	**S**	**L**	**Q**	**I**	**F**	**R**	**Y**	**N**	**S**	**H**
ϕ	0	0	0	0	0	0	0	0	0	0	0
S	0	4	0	0	0	0	0	0	1	4	0
A	0	1	3	0	0	0	0	0	0	2	2
Q	0	0	0	8	3	0	1	0	0	0	2
L	0	0	4	3	10	5	0	0	0	0	0
G	0	0	0	2	5	7	3	0	0	0	0
I	0	0	2	0	6	5	4	2	0	0	0
F	0	0	0	0	1	12	7	7	2	0	0
T	0	1	0	0	0	7	11	6	7	3	0
S	0	4	0	0	0	2	6	9	7	11	6
H	0	0	1	0	0	0	2	8	10	6	19

Figure 4.5. h(i,j) matrix for example

According to the matrix, the final alignment is shown in the following;

S L Q - - I F R Y N S H
S A Q L G I F T - - S H

Therefore the $\boldsymbol{nid}(\boldsymbol{S_i}, \boldsymbol{S_j})$ value is 6 for this alignment, since there are 6 identical characters with the same column in the alignment. According to these sequences, $\boldsymbol{n}(\boldsymbol{i}, \boldsymbol{j})$ matrix is built as the following figure;

	ϕ	S	L	Q	I	F	R	Y	N	S	H
ϕ	0	0	0	0	0	0	0	0	0	0	0
S	0	1	0	0	0	0	0	0	0	1	0
A	0	0	1	0	0	0	0	0	0	0	1
Q	0	0	0	2	2	0	0	0	0	0	0
L	0	0	1	2	2	2	2	0	0	0	0
G	0	0	0	1	2	2	2	0	0	0	0
I	0	0	0	0	3	2	2	2	0	0	0
F	0	0	0	0	2	4	3	2	2	0	0
T	0	0	0	0	0	3	4	3	2	2	0
S	0	1	0	0	0	3	3	4	4	5	3
H	0	0	1	0	0	0	3	3	3	3	6

Figure 4.6. n(i,j) matrix for example

Hardware implementation of these equations can be built with placing some addition signals and subcircuits to the linear gap penalty model used for pair-wise sequence alignment. These addition signals and circuits are shown in the following Figure 4.7. and Figure 4.8. Additional signals to linear gap penalty pairwise sequence alignment hardware are colored as red.

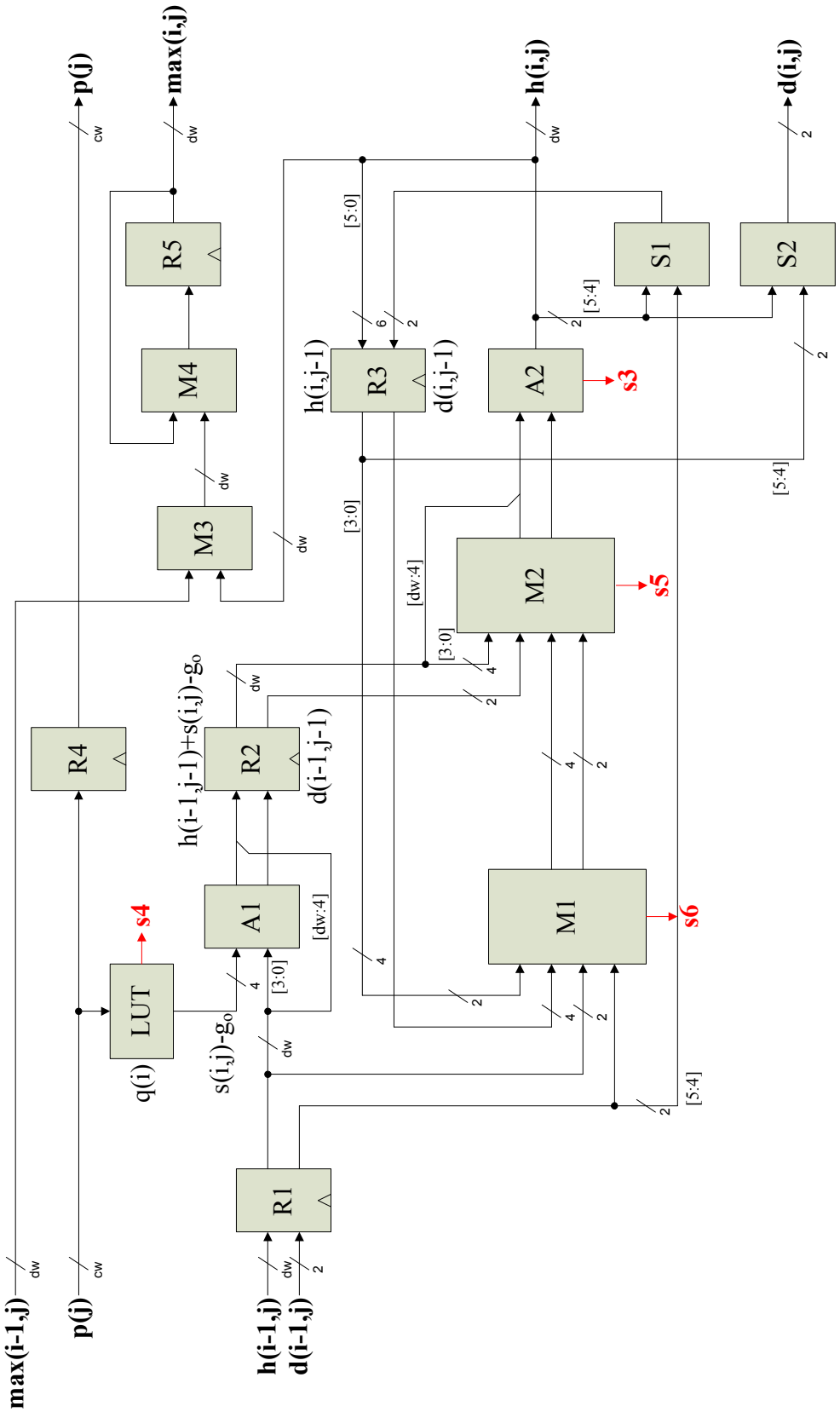

Figure 4.7. Additional signals to Smith-Waterman cell to perform multiple sequence alignment

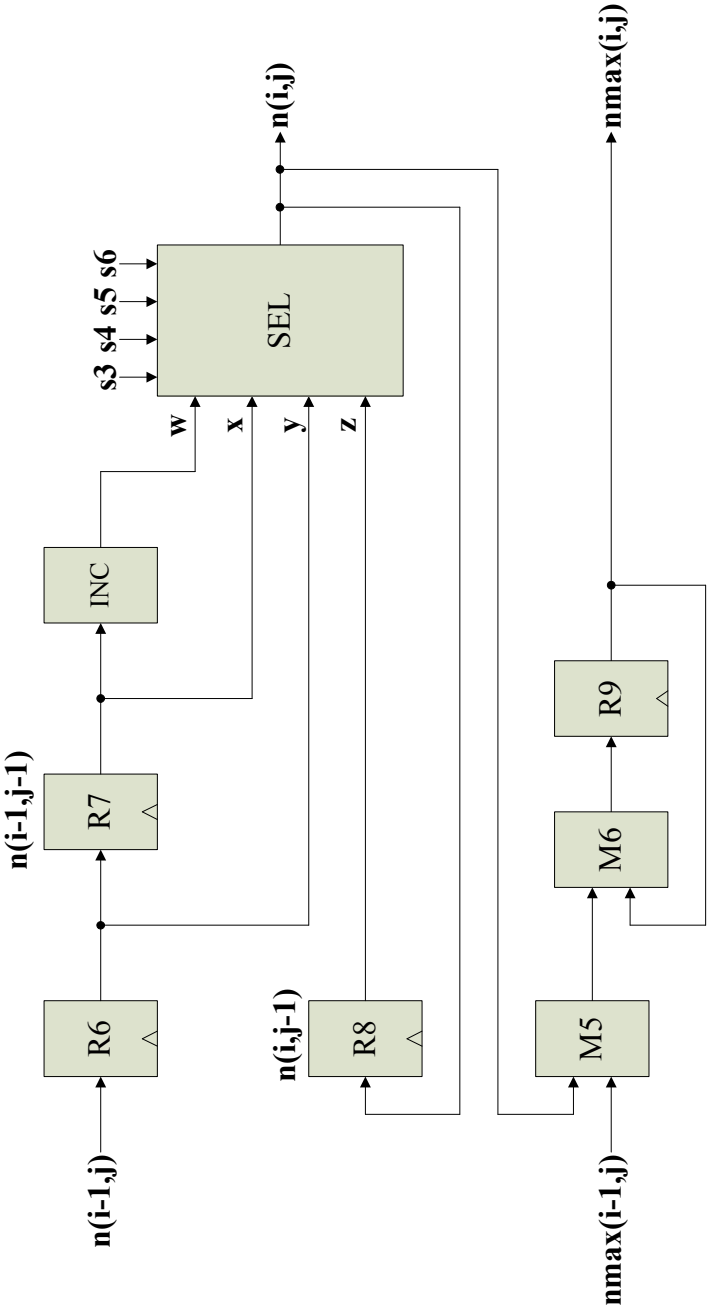

Figure 4.8. Additional hardware to Smith-Waterman cell to perform multiple sequence alignment

The following informations describe the functionality of the selective signals $s3$, $s4$, $s5$, and $s6$;

- **$s3$:** This signal indicates $h(i,j)$ signal is whether '0' or not. If the result of A2 is '0', then this signal is '1'. Otherwise it is '0'.

- $\boldsymbol{s4}$**:** This signal tests the identical sequence characters. If the query sequence character is the same character with the subject sequence character, then this signal is '1', otherwise it is '0'.
- $\boldsymbol{s5}$**:** This signal is used to find out if $\boldsymbol{h(i,j) = h(i-1,j-1) + s(i,j)}$ is true. If $\boldsymbol{h(i-1,j-1) + s(i,j)}$ term is selected, then this signal is '1', otherwise it is '0'.
- $\boldsymbol{s6}$**:** This signal is used to find out if $\boldsymbol{h(i-1,j)}$ is greater than $\boldsymbol{h(i,j-1)}$. If it is true, then this signal is '1', otherwise it is '0'.

These four signals are used to perform the selections in the Equation (4.18) and Equation (4.19). The hardware shown in Figure 4.8 builds the $\boldsymbol{n(i,j)}$ matrix according to the datas indicated by these selection signals. The followings describe the functionality of the units in the hardware to perform multiple sequence alignment, which is shown in the Figure 4.8.

- **R6:** dw-bit register. It stores $\boldsymbol{n(i-1,j)}$ signal vector, which are comes the left neighbor cell.
- **R7**: dw-bit register. It stores $\boldsymbol{n(i-1,j-1)}$ value.
- **R8**: dw-bit register. It stores $\boldsymbol{n(i-1,j-1)}$ value.
- **R9**: dw-bit register. It stores $\boldsymbol{nmax(i,j)}$ value for the calculations to be performed in the next clock cycle.
- **INC**: dw-bit incrementer which increments the input value by one.
- **M5**: dw-bit maximum generator. It is used to find the maximum of the signals $\boldsymbol{n(i,j)}$ and $\boldsymbol{nmax(i-1,j)}$.
- **M6**: dw-bit maximum generator. It is used to find the maximum of the signals $\boldsymbol{n(i,j)}$, $\boldsymbol{nmax(i-1,j)}$ and $\boldsymbol{nmax(i,j-1)}$.
- **SEL**: This unit is the final selector. It uses the selective signals generated by the main hardware and possible results of the $\boldsymbol{n(i,j)}$ values stored in the additional hardware to select the actual $\boldsymbol{n(i,j)}$ value. It performs the logical operations which are represented in the truth table which is shown in the Table 4.2.

s3	s4	s5	s6	n(i,j)	
1	X	X	X	0	
0	1	0	X	n(i-1,j-1)+1	(w)
0	0	0	X	n(i-1,j-1)	(x)
0	X	1	1	n(i-1,j)	(y)
0	X	1	0	n(i,j-1)	(z)

Table 4.2. Truth table for the logic operation of the selector unit for linear gap penalty model

With this truth table and required hardware, the Smith-Waterman cell with systolic array can now calculate and build the $\boldsymbol{n(i,j)}$ matrix to calculate $\boldsymbol{nid(S_i, S_j)}$ value, and speed up the CLUSTALX to perform multiple sequence alignment.

4.3.2. Affine Gap Penalty Multiple Sequence Alignment

For affine gap penalty multiple sequence alignment, each step of the CLUSTALX software is as the same with linear gap penalty multiple sequence alignment. The only difference between them is the equations of the algorithm. Similar to the pair-wise sequence alignment, in affine gap penalty case, equations are more complex than linear gap equations. The equations needed in affine gap penalty multiple sequence alignment are given as the followings;

$$\boldsymbol{h(i,j) = max}\begin{cases} \boldsymbol{0} \\ \boldsymbol{h(i-1,j-1) + s(i,j)} \\ \boldsymbol{e(i,j)} \\ \boldsymbol{f(i,j)} \end{cases}$$

for $\boldsymbol{1 \le i \le m}$ and $\boldsymbol{1 \le j \le n}$ **(4.22)**

$$\boldsymbol{e(i,j) = max}\begin{cases} \boldsymbol{h(i-1,j) - g_o} \\ \boldsymbol{e(i-1,j) - g_e} \end{cases} \text{ for } \boldsymbol{1 \le i \le m} \text{ and } \boldsymbol{1 \le j \le n} \quad \textbf{(4.23)}$$

$$\boldsymbol{f(i,j) = max\begin{cases} h(i,j-1) - g_o \\ f(i,j-1) - g_e \end{cases}} \quad \text{for } \mathbf{1 \le i \le m} \text{ and } \mathbf{1 \le j \le n} \qquad \textbf{(4.24)}$$

$$\boldsymbol{n(i,j) = \begin{cases} 0 & if\ h(i,j) = 0 \\ n(i-1,j-1) + m(i,j) & if\ h(i,j) = h(i-1,j-1) + s(i,j) \\ n(i - ne(i,j), j) & if\ h(i,j) = e(i,j) \\ nf(i, j - nf(i,j)) & if\ h(i,j) = f(i,j) \end{cases}}$$

for $\mathbf{1 \le i \le m}$ and $\mathbf{1 \le j \le n}$ **(4.25)**

$$\boldsymbol{ne(i,j) = \begin{cases} 1 & if\ e(i,j) = h(i-1,j) - g_o\ or\ i = 1 \\ ne(i-1,j) + 1 & if\ e(i,j) = e(i-1,j) - g_e \end{cases}}$$

for $\mathbf{1 \le i \le m}$ and $\mathbf{1 \le j \le n}$ **(4.26)**

$$\boldsymbol{nf(i,j) = \begin{cases} 1 & if\ f(i,j) = h(i,j-1) - g_o\ or\ j = 1 \\ nf(i,j-1) + 1 & if\ f(i,j) = e(i-1,j) - g_e \end{cases}}$$

for $\mathbf{1 \le i \le m}$ and $\mathbf{1 \le j \le n}$ **(4.27)**

$$\boldsymbol{m(i,j) = \begin{cases} 1 & if\ s(i) = s(j) \\ 0 & otherwise \end{cases}} \qquad \text{for } \mathbf{1 \le i \le m} \text{ and } \mathbf{1 \le j \le n} \qquad \textbf{(4.28)}$$

$$\boldsymbol{h(i,0) = n(i,0) = 0} \qquad \text{for } \mathbf{0 \le i \le m} \qquad \textbf{(4.29)}$$

$$\boldsymbol{h(0,j) = n(0,j) = 0} \qquad \text{for } \mathbf{0 \le j \le n} \qquad \textbf{(4.30)}$$

where $\boldsymbol{m}$ is the length of the query sequence, and $\boldsymbol{n}$ is the length of the subject sequence.

Although these equations seem very complex and very hard to implement, using the second optimization idea, the hardware implementation can be assimilated like pair-wise sequence alignment.

The mentioned hardwares are shown in Figure 4.9 and Figure 4.10 with additional selective signals and required hardware. Additional signals to affine gap penalty pairwise sequence alignment hardware are colored as red.

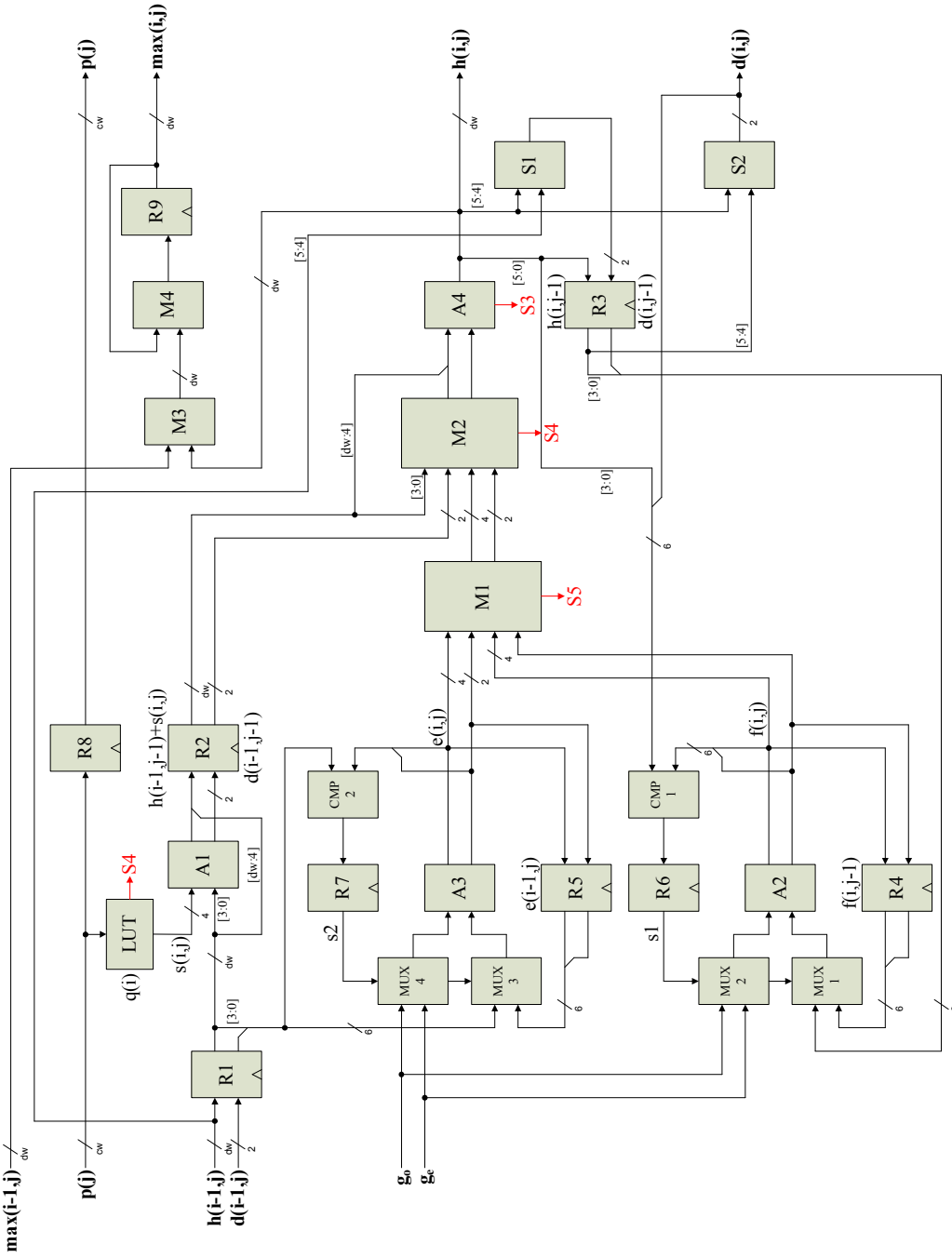

Figure 4.9. Additional signals to Smith Watermancell to perform multiple sequence alignment

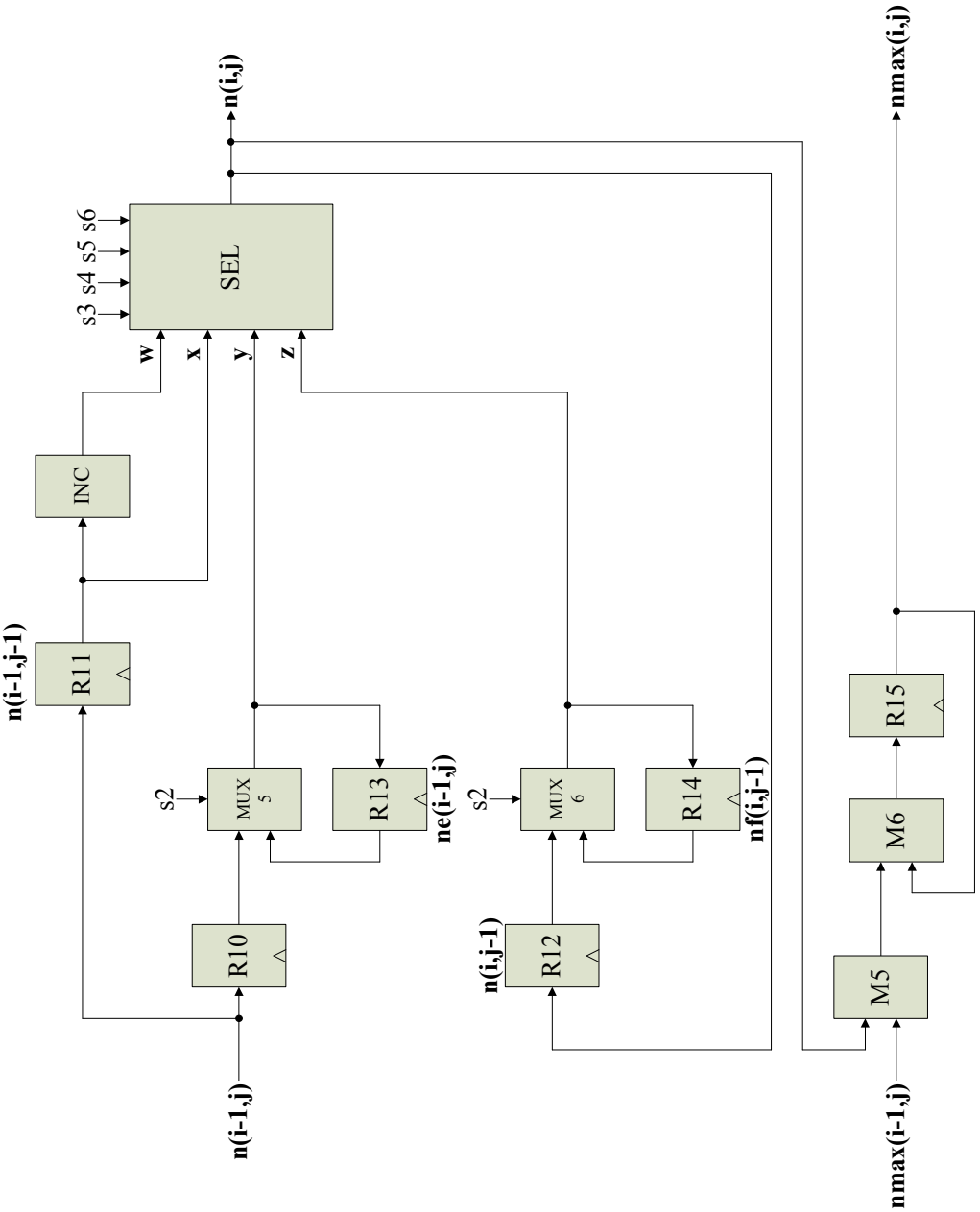

Figure 4.10. Additional hardware to Smith-Waterman cell to perform multiple sequence alignment

The detailed explanations of the selective signals and additional hardwares are represented as the followings;

- ***s3*:** This signal indicates $\boldsymbol{h(i,j)}$ signal is whether '0' or not. If the result of A2 is '0', then this signal is '1'. Otherwise it is '0'.

- $\boldsymbol{s4}$**:** This signal tests the identical sequence characters. If the query sequence character is the same character with the subject sequence character, then this signal is '1', otherwise it is '0'.
- $\boldsymbol{s5}$**:** This signal is used to find out if $\boldsymbol{h(i,j) = h(i-1,j-1) + s(i,j)}$ is true. If $\boldsymbol{h(i-1,j-1) + s(i,j)}$ term is selected, then this signal is '1', otherwise it is '0'.
- $\boldsymbol{s6}$**:** This signal is used to find out if $\boldsymbol{h(i-1,j)}$ is greater than $\boldsymbol{h(i,j-1)}$. If it is true, then this signal is '1', otherwise it is '0'.
- **R10:** dw-bit register. It stores $\boldsymbol{n(i-1,j)}$ signal vector, which are came from the left neighbor cell.
- **R11**: dw-bit register. It stores $\boldsymbol{n(i-1,j-1)}$ value.
- **R12**: dw-bit register. It stores $\boldsymbol{n(i,j-1)}$ value.
- **R13**: dw-bit register. It stores $\boldsymbol{ne(i-1,j)}$ value.
- **R14**: dw-bit register. It stores $\boldsymbol{nf(i,j-1)}$ value.
- **R15**: dw-bit register. It stores $\boldsymbol{nmax(i,j)}$ value for the calculations to be performed in the next clock cycle.
- **MUX5**: dw-bit multiplexer. It selects $\boldsymbol{ne(i-1,j)}$ or $\boldsymbol{n(i-1,j)}$ according to the value of $\boldsymbol{s2}$ signal.
- **MUX6**: dw-bit multiplexer. It selects $\boldsymbol{nf(i,j-1)}$ or $\boldsymbol{n(i,j-1)}$ according to the value of $\boldsymbol{s1}$ signal.
- **INC**: dw-bit incrementer that increments the input value by one.
- **M5**: dw-bit maximum generator. It is used to find the maximum of the signals $\boldsymbol{n(i,j)}$ and $\boldsymbol{nmax(i-1,j)}$.
- **M6**: dw-bit maximum generator. It is used to find the maximum of the signals $\boldsymbol{n(i,j)}$, $\boldsymbol{nmax(i-1,j)}$ and $\boldsymbol{nmax(i,j-1)}$.
- **SEL**: This unit is the final selector. It uses the selective signals generated by the main hardware and possible results of the $\boldsymbol{n(i,j)}$ values stored in the additional hardware to select the actual $\boldsymbol{n(i,j)}$ value. It performs the logical operations which are represented in the truth table which is shown in the Table 4.3

s3	s4	s5	s6	n(i,j)	
1	X	X	X	0	
0	1	0	X	n(i-1,j-1)+1	(w)
0	0	0	X	n(i-1,j-1)	(x)
0	X	1	0	ne(i,j)	(y)
0	X	1	1	nf(i,j)	(z)

Table 4.3. Truth table for the logic operation of the selector unit for affine gap penalty model

For both linear gap penalty and affine gap penalty multiple sequence alignment operation, the described hardwares build the $\boldsymbol{n(i,j)}$ matrix. Using the maximum value of this matrix, the $\boldsymbol{nid(S_i, S_j)}$ value is obtained and it is used to calculate the distance value between two corresponding sequences. Since CLUSTALW consumes about 90% of the operation time for calculation of distance values, calculating this value using the hardware will provide a huge speeding up rate. The practical results reported in (Oliver et al, 2005a) are represented in Table 4.4.

		Calculation Times (sec)		
		PC	FPGA	Speed up
Number of Sequences	100	42,4	1,1	37,8
	200	184	3,6	50,9
	400	833	18,1	46
	600	1697	38,2	44,4
	800	2967	65	45,6
	1000	4410	97,5	45,2

Table 4.4. The practical results of the multiple sequence alignment hardware (Oliver et. al., 2005)

As shown in the table, modifying the pair-wise sequence alignment hardwares as to calculate distance value between sequences provides about 50 times speeding up the calculation time of the CLUSTALW software. Also this hardware

can be used any algorithm which uses progressive alignment method such as PRALINE, PILEUP, etc...

5. SYNTHESIS RESULTS

This section presents synthesis results. The proposed designs are compared with reference (Oliver et al, 2005a) and (Oliver et al, 2005b) hardware designs to point out the advantages of the proposed designs. The reference hardwares are the direct implementations of the alignment equations to the Smith-Waterman cells and they are the most recent designs.

To comparison the proposed and reference designs, VHDL (Very High Scale Integrated Circuits Hardware Description Language) codes of each design is written in generic coding style, and generated for simulation and synhesis in 8, 16, 24, 32, and 64-bit data-width size. Then these VHDL codes are simulated using ModelSIM SE 6.1a tool to verify the correctness of operations of the proposed and reference hardwares. Syntheses are made for all hardware mentioned in the previous sections on two different platforms, ASIC (Application Specific Integrated Circuits) and FPGA (Field Programmable Gate Array).

ASIC implementations are designed to perform a specific operation to take advantage from space and time requirements. Generally they are much smaller and faster than a general purpose processor which is loaded with a sequence of instructions to perform the same operation. However they suffer from manufacturing issues. They are not suitable for serial production. Therefore manufacturing an ACIS chip has a high cost. Also there are some possibilities to make error during production in nanometer scales.

Another option to implement a digital design is the configuring the FPGAs. FPGAs are reconfigurable hardware platforms which allow designers to map their algorithms. Although they are relatively slower and bigger than the ASICs, their flexibility makes them very popular selection during recent years. Their reconfigurability helps the designers to detect errors and bugs in the design and allows fixing them. They have a great marketing around the world and suitable prices. Also there is another importance of FPGA for the bioinformatics. FPGA performance is increasing by 4 times every two years. The performance of

architectural improvements is increasing 5 times every two years. Thus, FPGAs have a potential for performance improvement of 20 times every two years. Biological databases are doubled every six months, which means 16 times growth over two years. Therefore it is expected that FPGA platforms can respond to the biological database growing.

In this work, proposed and reference are compared using these two different platforms. For ASIC syntheses TSMC (Taiwan Semiconductor Manufacturing Company) 0,18 micron technology cells with Leonardo Spectrum v2005a synthesis tool, and for FPGA syntheses Xilinx Virtex II XC2V6000 FPGA platform with Xilinx ISE 8.1 design tool is used.

The followings are loaded into hardwares as the parameters of sequence alignments; the substitution matrix is BLOSUM62, the gap opening cost is 5, and the gap extension cost is 1. These parameters are reconfigurable so that they can be changed according to user's needs.

The synthesis results are given in Table 5.1 and Table 5.2, for linear and affine gap penalty models, respectively. These syntheses results illustrate the requirements of a Smith Waterman Cells.

		ASIC		FPGA	
		Number of Gates	**Clock Frequency (MHz)**	**Number of Slices**	**Clock Frequency (MHz)**
8-Bit	**Proposed**	1.000	203,1	145	115,1
	Reference	1.020	192,7	160	114,7
	Gain	1,96%	5,40%	9,38%	0,35%
16-Bit	**Proposed**	1.292	176,7	179	93,1
	Reference	1.479	152,1	195	92,2
	Gain	12,64%	16,17%	8,21%	0,98%
24-Bit	**Proposed**	1.535	131,2	214	80,8
	Reference	1.927	111,3	250	78,8
	Gain	20,34%	17,88%	14,40%	2,54%
32-Bit	**Proposed**	1.826	87,5	249	76,2
	Reference	2.425	72,9	297	72,5
	Gain	24,70%	20,03%	16,16%	5,10%
64-Bit	**Proposed**	2.957	57,1	314	62,1
	Reference	4.237	39,6	485	54,9
	Gain	30,21%	44,19%	35,26%	13,11%

Table 5.1. The synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for linear gap penalty implementations

		ASIC		FPGA	
		Number of Gates	**Clock Frequency (MHz)**	**Number of Slices**	**Clock Frequency (MHz)**
8-Bit	**Proposed**	1.238	194,7	187	82,1
	Reference	1.247	190,9	204	82,0
	Gain	0,72%	1,99%	8,33%	0,12%
16-Bit	**Proposed**	1.557	138,8	215	66,1
	Reference	1.983	108,9	279	65,0
	Gain	21,48%	27,46%	22,94%	1,69%
24-Bit	**Proposed**	1.838	99,4	229	62,1
	Reference	2.730	77,2	365	59,7
	Gain	32,67%	28,76%	37,26%	4,02%
32-Bit	**Proposed**	2.152	78,1	250	59,4
	Reference	3.520	59,4	456	55,0
	Gain	38,86%	31,48%	45,18%	8,00%
64-Bit	**Proposed**	3.327	53,4	344	50,7
	Reference	6.503	32,2	820	41,8
	Gain	48,84%	65,84%	58,05%	21,29%

Table 5.2. The synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for affine gap penalty implementations

The syntheses results presented in Table 5.1 and 5.2 indicate that new proposed hardwares lead a remarkable gain of space and time requirements using the optimization ideas applied to the Smith Waterman equations. These gains are up to 35,26% of space and up to 44,19% of time for linear gap penalty implementations and up to 58,05% of space and up to 65,84% of time for affine gap penalty implementations. There is also a noticeable benefit of proposed algorithm, which is that as the precision of data-width increases, the number of the gates or the slices increases slowly according to the synthesis results. That is, because most of the units has a constant bit number, such as adders, maximum generators. Only some hardware units such as registers get larger as the data-width increases.

These hardwares are also synthesized for only FPGA platform, to get some specific results. For example, to find out how many cells can be mapped onto Xilinx

Virtex II XC2V6000 FPGA platform. This result is important, because the number of the cells determines the number of the characters to be able to align in subject sequence. If the number of the characters in the subject sequence exceeds the number of the Smith Waterman cells in the systolic architecture, some runtime configurations must be made. This configuration can include iterative steps for each part of the subject sequence. This affects the overall system performance. Therefore, increasing the number of Smith Waterman cells leads the decreasing the number of iteration for the same biological sequence.

Table 5.3 and Table 5.4 indicate the results of the detailed FPGA synthesis.

		Precisions				
		8-Bit	**16-Bit**	**24-Bit**	**32-Bit**	**64-Bit**
Number of SW Cells	**Proposed**	233	188	157	135	107
	Reference	211	173	135	113	69
	Gain	10,43%	8,67%	16,30%	19,47%	55,07%
Clock Frequency (MHz)	**Proposed**	115,1	93,1	80,8	76,2	62,1
	Reference	114,7	92,2	78,8	72,5	54,9
	Gain	0,35%	0,98%	2,54%	5,10%	13,11%
Performance (GCUPS)	**Proposed**	26,82	17,50	12,69	10,29	6,64
	Reference	24,20	15,95	10,64	8,19	3,79
	Gain	10,81%	9,73%	19,25%	25,57%	75,41%

Table 5.3. The detailed FPGA synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for linear gap penalty implementations

		Precisions				
		8-Bit	**16-Bit**	**24-Bit**	**32-Bit**	**64-Bit**
Number of SW Cells	**Proposed**	180	157	147	135	98
	Reference	165	121	92	74	41
	Gain	9,09%	29,75%	59,78%	82,43%	139,02%
Clock Frequency (MHz)	**Proposed**	82,1	66,1	62,1	59,4	50,7
	Reference	82,0	65,0	59,7	55,0	41,8
	Gain	0,12%	1,69%	4,02%	8,00%	21,29%
Performance (GCUPS)	**Proposed**	14,78	10,38	9,13	8,02	4,97
	Reference	13,53	7,87	5,49	4,07	1,71
	Gain	9,22%	31,95%	66,21%	97,03%	189,92%

Table 5.4. The detailed FPGA synthesis results for the proposed and the reference (Oliver et al, 2005b) SW designs for affine gap penalty implementations

These tables are calculated using the number of slices presented in Xilinx Virtex II XC2V6000 FPGA platform, which is 33792. These FPGA synthesis results indicate that the new implementations provide up to 75,41% for linear gap penalty model and up to 189,92% for affine gap penalty model performance gain. These performance measures have a special unit which is GCUPS (Giga Cell Updates per Second). This unit measures the performances of systolic architectures using production of numbers of cells and frequencies. It is meaning is that how many cells are updated in one second.

5.1. Graphical Representations of the Syntheses Results

This section gives graphical representations to illustrate the syntheses results. Proposed hardwares are represented in green line and reference hardwares are represented in blue lines in all graphics.

Figure 5.1 and Figure 5.2 present the graphics for ASIC syntheses of linear gap penalty implementations.

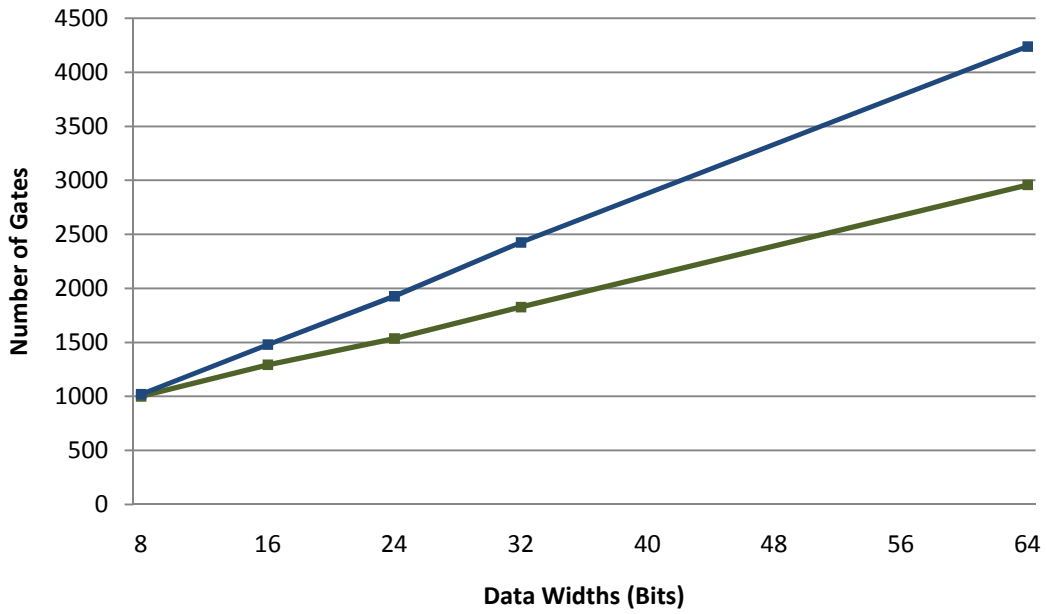

Figure 5.1. Graphical representation of space requirements for ASIC syntheses of linear gap penalty implementations

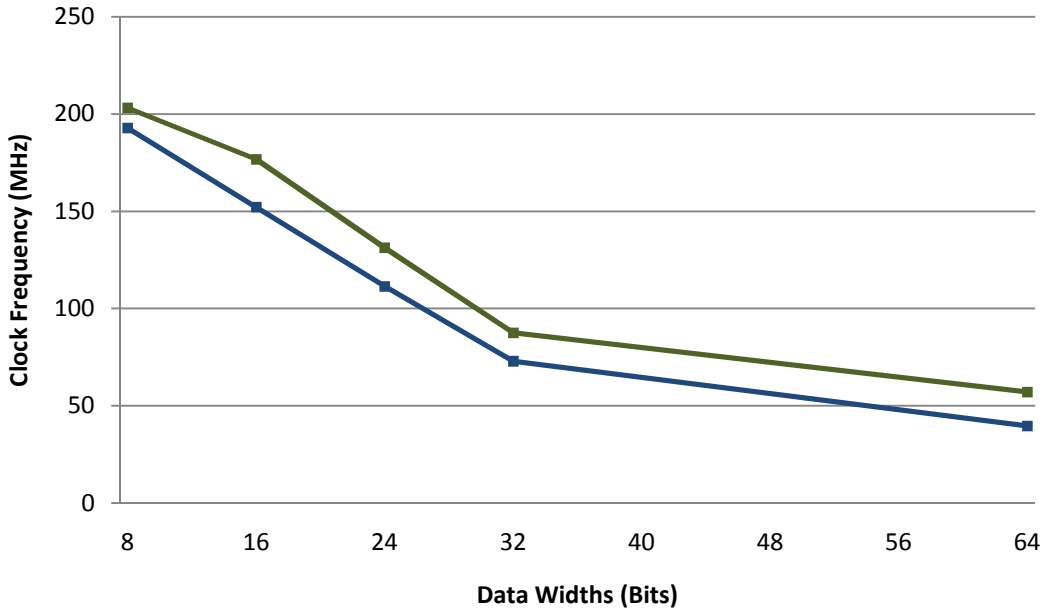

Figure 5.2. Graphical representation of clock frequencies for ASIC syntheses of linear gap penalty implementations

Figure 5.3 and Figure 5.4 present the graphics for FPGA syntheses of linear gap penalty implementations.

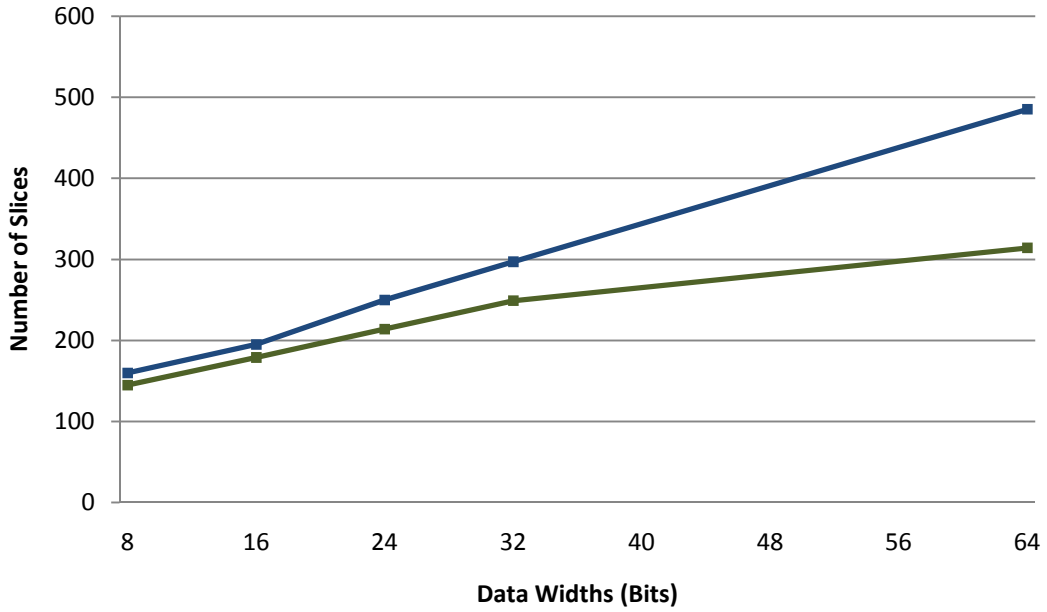

Figure 5.3. Graphical representation of space requirements for FPGA syntheses of linear gap penalty implementations

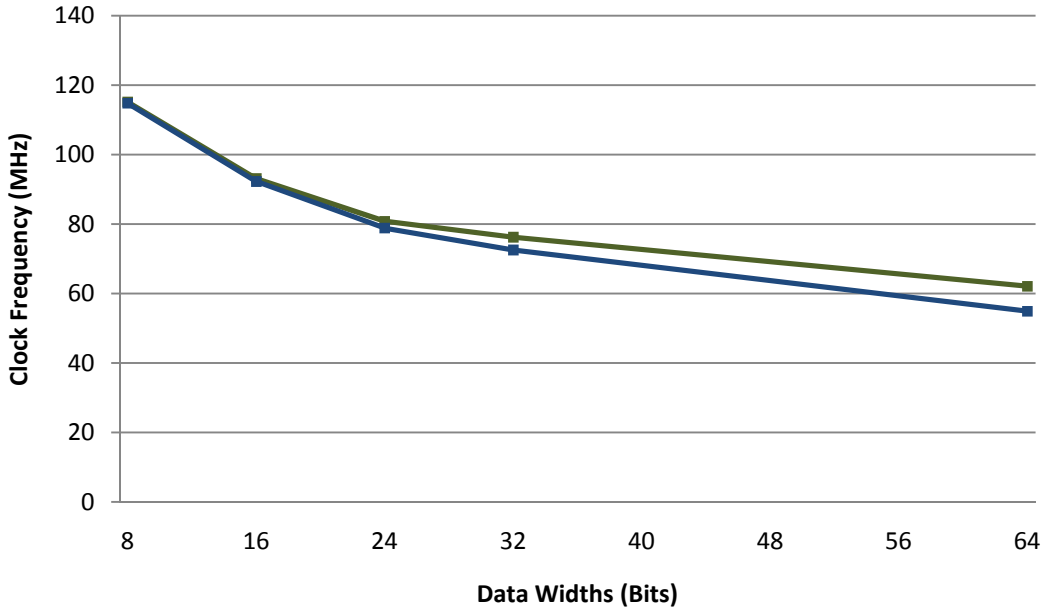

Figure 5.4. Graphical representation of clock frequencies for FPGA syntheses of linear gap penalty implementations

Figure 5.5 and Figure 5.6 present the graphics for ASIC syntheses of affine gap penalty implementations.

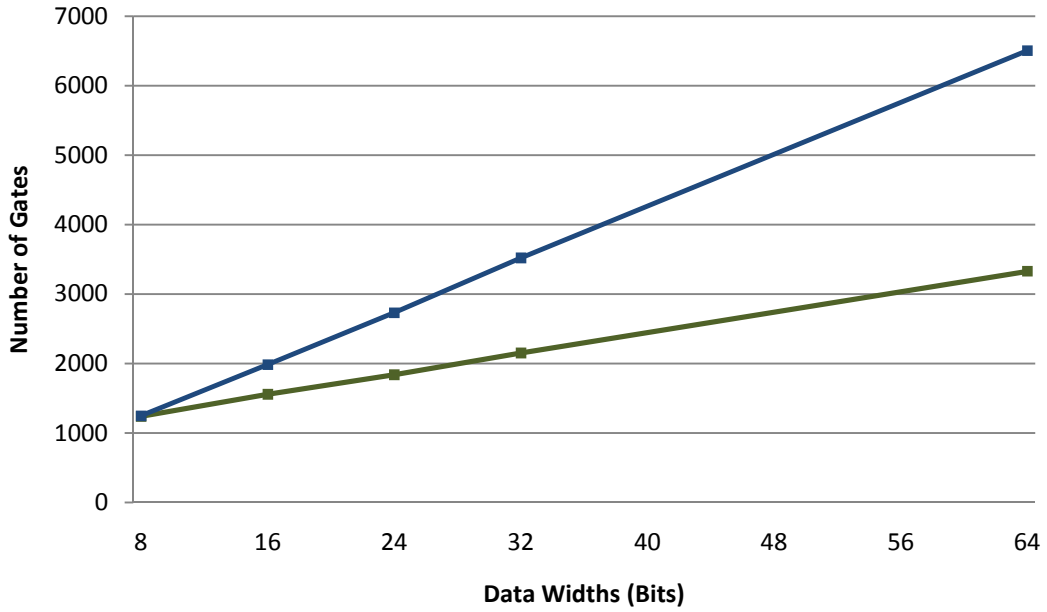

Figure 5.5. Graphical representation of space requirements for ASIC syntheses of affine gap penalty implementations

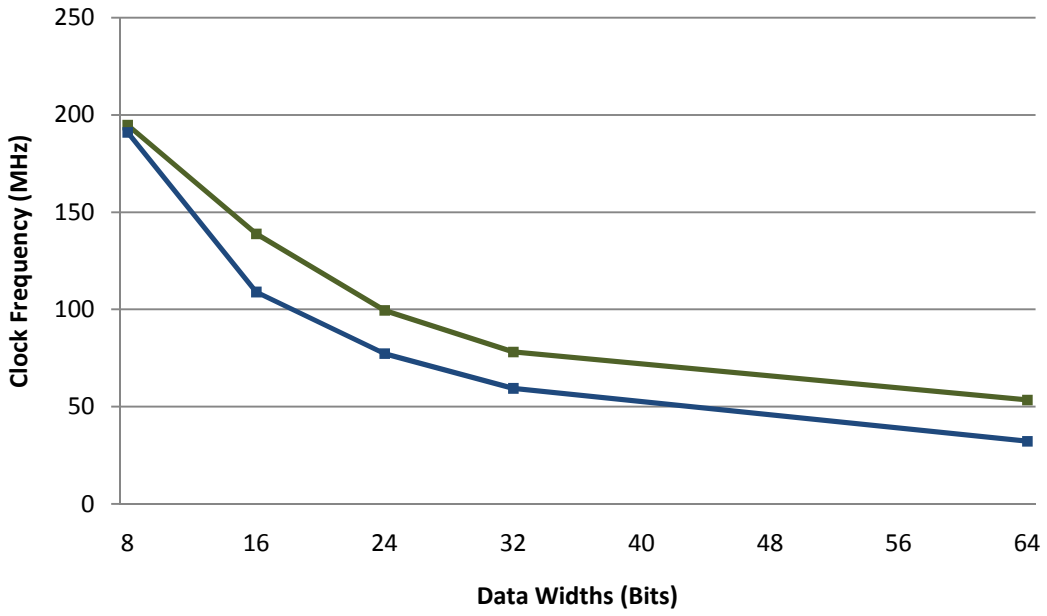

Figure 5.6. Graphical representation of clock frequencies for ASIC syntheses of affine gap penalty implementations

Figure 5.7 and Figure 5.8 present the graphics for FPGA syntheses of affine gap penalty implementations.

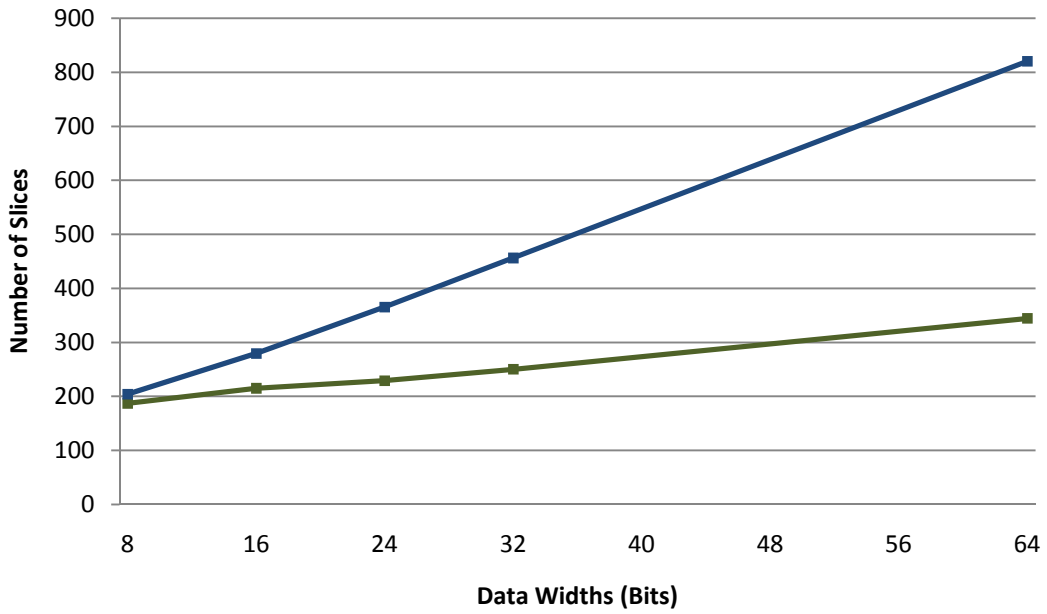

Figure 5.7. Graphical representation of space requirements for FPGA syntheses of affine gap penalty implementations

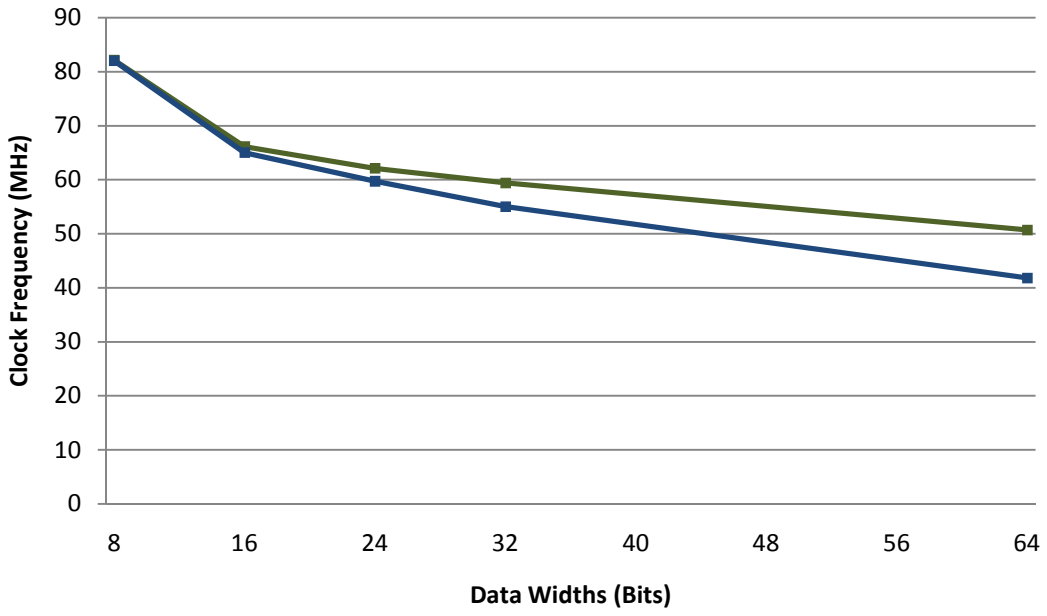

Figure 5.8. Graphical representation of clock frequencies for FPGA syntheses of affine gap penalty implementations

Figure 5.9 and Figure 5.10 present the graphics of performance measurements for FPGA syntheses of both linear and affine gap penalty implementations.

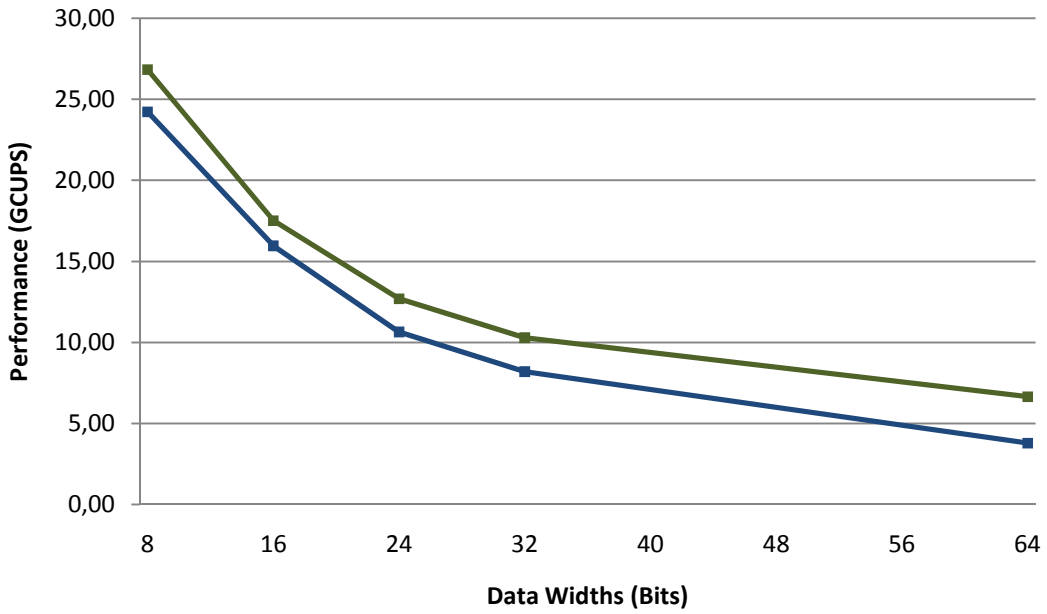

Figure 5.9. Graphical representation of performance measurements for FPGA syntheses of linear gap penalty implementations

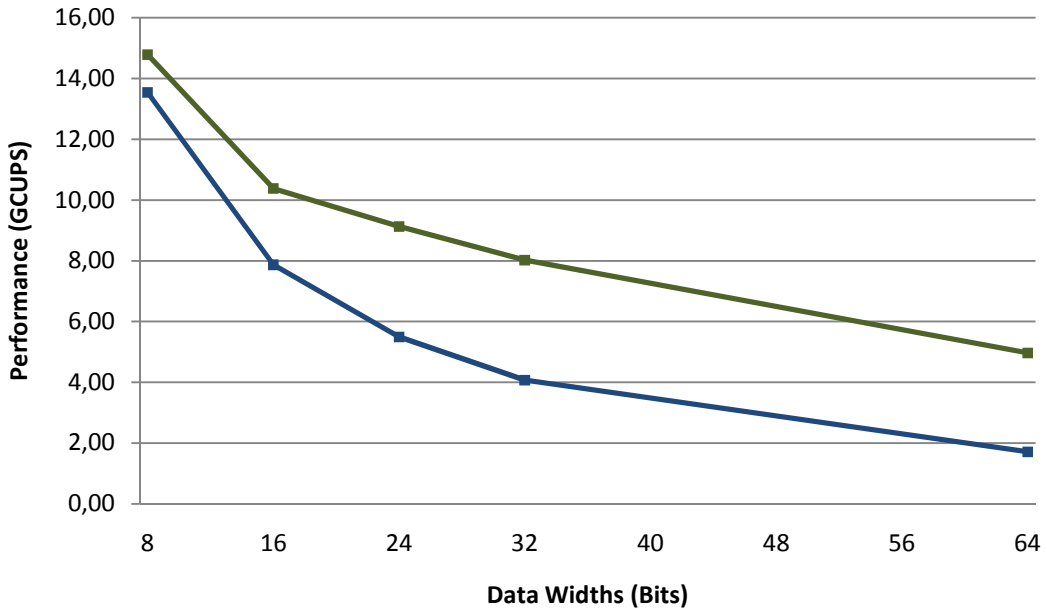

Figure 5.10. Graphical representation of performance measurements for FPGA syntheses of affine gap penalty implementations

6. CONCLUSIONS AND FUTURE WORKS

Bioinformatics research is focused on analysis of vast amount of data stored in ever growing biological databases. The main analysis tool for research is sequence alignment. In this thesis work, optimized algorithms and hardwares for sequence alignment operations are presented. The presented design techniques optimize the implementation for systolic sequence alignment hardware. The proposed designs with 8, 16, 24, 32, and 64 bit score lengths are modeled using VHDL and mapped on ASIC and FPGA platforms. The main advantage of the proposed techniques is that it keeps the hardware increase in minimum range as the score bit length increases. The proposed techniques can be very useful to map big systolic array in one platform. By this way, very long query sequences can be aligned in one iteration.

The proposed hardwares are designed in VHDL for several bit wides and simulated using ModelSIM Simulation tool. Using these hardwares provide the users up to 58,05% space, up to 65,84% time, and up to 189,92% performance gain with the comparison with the previous works. Recent days, these proposed hardwares are being implemented using the Xilinx Virtex II XC2V6000 FPGA platform.

As the future works, multiple sequence alignment algorithms mentioned in thesis will be mapped onto FPGA platform. After completing all the necessary sequence alignment operations on FPGA, a complete system will be designed. This system will include an FPGA platform card which is mounted a host PC via PCI slots, and an interface software for users to perform the sequence alignment operations on these hardwares. With this system users can scan a biological database for similar sequences with a query sequence, which might be DNA or protein sequence, align a pair or a multiple of sequences in the meaning of linear gap penalty and affine gap penalty models.

REFERENCES

ALTSCHUL, S.F., GISH, W., MILLER, W., MYERS, E.W., and LIPMAN, D.J., 1990, Basic local alignment search tool, Journal of Molecular Biology, 215, 3, 403-410.

BEYER, W.A., SMITH, T.F., STEIN, M.L., and ULAM, S.M., 1979, A molecular sequence metric and evolutionary trees, Mathematical Biosciences, 19, 9-25.

BLandY, J. and FOGEL, K., 2007, An Alignment Editor based on GNU Emacs, http://www.red-bean.com/ale/.

BLEASBY, A., 1999, Needle: Needleman-Wunsch global alignment (EMBOSS), http://bioweb.pasteur.fr/seqanal/interfaces/needle.html.

BLEASBY, A., 1999, Water : Smith-Waterman local alignment (EMBOSS), http://bioweb.pasteur.fr/seqanal/interfaces/water.html.

BOWEN, R.A., 1998, Nucleic acid dot plots, Colorado State University Animal Reproduction and Biotechnology Laboratory, http://arbl.cvmbs.colostate.edu/molkit/dnadot/index.html.

BRAY, N. and PACHTER, L., 2004, MAVID: constrained ancestral alignment of multiple sequences, Genome Research, 14, 693-699.

BRUDNO, M., CHAPMAN, M., GÖTTGENS, B., BATZOGLOU, S., and MORGENSTERN, B., 2003, Fast and sensitive multiple alignment of large genomic sequences, BMC Bioinformatics, 4, 66.

BRUDNO, M., DO, C., COOPER, G., KIM, M.F., DAVYDOV, E., GREEN, E.D., SIDOW, A. and BATZOGLOU, S., 2003, LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA, Genome Research, 13, 4, 721-731.

CARTWRIGHT, R.A., 2007, Ngila: global pairwise alignments with logarithmic and affine gap costs, Bioinformatics, 23, 11, 1427-1428.

CODONCODE CORPORATION, 2007, http://www.codoncode.com.

CORPET, F., 1988, Multiple sequence alignment with hierarchical clustering, Nucleic Acids Research, 16, 22, 10881-10890.

DAYHOFF, M.O., 1969, Atlas of Protein Sequence and Structure, National Biomedical Reserch Foundation.

DAYHOFF, M.O., SCHWARTZ, R.M., ORCUTT, B.C., 1978, A model of evolutionary change in proteins, Atlas of Protein Sequence and Structure, 5, 3, 345 – 352.

DECYPHER TECHNOLOGIES, LTD., 2007, www.decypher.com/.

DI BIAS, A., DAHLE, D.M., DIEKHANS, M., GRATE, L., HIRSCHBERG, J., KARPLUS, K., KELLER, H., KENDRICK, M., MESA-MARTINEZ, F.J., PEASE, D., RICE, E., SCHULTZ, A., SPECK, D., and HUGHEY, R., 2005, The UCSC Kestrel parallel processor, IEEE Transactions on Parallel and Distributed Systems, 16, 1, 80-92.

DO, C.B., MAHABHASHYAM, M.S.P., BRUDNO, M. and BATZOGLOU, S., 2005, PROBCONS: probabilistic consistency-based multiple sequence alignment, Genome Research, 15, 330-340.

EDGAR, R.C., 2004, MUSCLE: multiple sequence alignment with high accuracy and high throughput, Nucleic Acids Research, 32, 5, 1792-1797.

EDGAR, R.C., 2004, MUSCLE: a multiple sequence alignment method with reduced time and space complexity, BMC Bioinformatics, 5, 1, 113.

FITCH, W.M., 1966, an improved method of testing for evolutionary homology, Journal of Molecular Biology, 16, 9–13.

GABRIEL, M., 2005, DNA Baser: a sequence assembly software, http://www.dnabaser.com.

GOK, M., and YILMAZ, C., 2006, Hardware designs for local alignment of protein sequences, Lecture Notes in Computer Sciences, 4263, 277-285.

GOK, M., and YILMAZ, C., 2006, Efficient cell designs for systolic smith-waterman implementations, Proceedings of the 16th International Conference on Field Programmable Logic and Applications, 889-892.

GOKHALE, M., HOLMES, W., KOPSER, A., LUCAS, S., MINNICH, R., SWEELY, D., and LOPRESTI, D., 1991, Building and using a highly parallel programmable logic array, Computer, 24, 81-89.

HARDY, P., WATERMAN, M.S., 1996, The sequence alignment software library at USC, http://www.hto.usc.edu/software/seqaln.

HENIKOFF, S. and HENIKOFF, J., 1992, Amino acid substitution matrices from protein blocks, Proceedings of the National Academy of Sciences of the United States of America, 89, 10915- 10919.

HUANG, X. and MILLER W., 1991, A time-efficient, linear-space local similarity algorithm, Advances in Applied Mathematics, 12, 337-357.

HUGHEY, R. and KROGH, A., 1995, SAM : sequence alignment and modeling software system, Technical Report UCSC-CRL, University of California, Santa Cruz, CA, 95-97.

JAPANESE INSTITUTE OF BIOINFORMATICS RESEARCH AND DEVELOPMENT, 2007, http://www-btls.jst.go.jp/cgi-bin/Tools/SSEARCH/index.cgi.

JUNIER, T. and PAGNI, M., 2000, Dotlet: diagonal plots in a web browser, Bioinformatics, 16, 2, 178-179.

KATOH, K., KUMA, K., TOH, H., and MIYATA, T., 2005, MAFFT version 5: improvement in accuracy of multiple sequence alignment, Nucleic Acids Research, 33, 511-518.

KURTZ, S., CHOUDHURI, J. V., OHLEBUSCH, E., SCHLEIERMACHER, C., STOYE, J., and GIEGERICH, R., 2007, REPuter: the manifold applications of repeat analysis on a genomic scale, Nucleic Acids Research, 29, 22, 4633-4642.

KURTZ, S., PHILLIPPY, A., DELCHER, A.L., SMOOT, M., SHUMWAY, M., ANTONESCU, C., and SALZBERG, S.L., 2004, Versatile and open software for comparing large genomes, Genome Biology, 5, 2, 12-20.

LASSMANN, T. and SONNHAMMER, E.L.L., 2005, Kalign - an accurate and fast multiple sequence alignment algorithm, BMC Bioinformatics, 6, 298.

LIPMAN, D.J. and PEARSON, W.R., 1985, Rapid and sensitive protein similarity searches, Science, 227, 4693, 1435-1441.

LIPMAN, D.J., ALTSCHUL, S.F. and KECECIOGLU. J.D., 1989, A tool for multiple sequence alignment, Proceedings of the National Academy of Sciences of the United States of America, 86, 4412-4415.

LONGDEN, I., 1998, Wordmatch : finds all exact matches of a given size between two sequences (EMBOSS), http://bioweb.pasteur.fr/seqanal/interfaces/wordmatch.html.

LONGDEN, I., 1999, Matcher : Finds the best local alignments between two sequences (EMBOSS), http://bioweb.pasteur.fr/seqanal/interfaces/matcher.html.

MA, B., TROMP, J., LI, M., 2002, PatternHunter: faster and more sensitive homology search, Bioinformatics, 18, 3, 440-445.

MOUSTAFA, A., 2007, JAligner: Open source Java implementation of Smith-Waterman, http://jaligner.sourceforge.net.

MÜCKSTEIN, U., HOFACKER, I.L., and STADLER, P.F., 2002, Stochastic pairwise alignments, Bioinformatics, 18, 153-160.

NCBI (NATIONAL CENTER FOR BIOTECHNOLOGY INFORMATION), 2007, http://www.ncbi.nlm.nih.gov.

NEEDLEMAN, S.B. and WUNSCH, C.D., 1970, A general method applicable to the search for similarities in the amino acid sequence of two proteins, Journal of Molecular Biology, 48, 3, 443-453.

NIH (NATIONAL INSTITUTES OF HEALTH OF THE USA) BIOMEDICAL INFORMATION SCIENCE AND TECHNOLOGY INITIATIVE CONSORTIUM, 2000, NIH working definition of bioinformatics and computational biology, http://www.bisti.nih.gov/ CompuBioDef.pdf.

NOE, L. and KUCHEROV, G., 2006, YASS: DNA local alignment tool, http://bioinfo.lifl.fr/yass/yass.php.

NOTREDAME, C., HIGGINS, D.G., 1996, SAGA: sequence alignment by genetic algorithm, Nucleic Acids Research, 24, 1515-1524.

NOTREDAME, C., HOLME, L., and HIGGINS, D.G., 1998, COFFEE: a new objective function for multiple sequence alignmnent, Bioinformatics, 14, 5, 407-422.

OLIVER, T.F., SCHMIDT, B., NATHAN, D., CLEMENS, R., and MASKELL, D.L., 2005, Multiple sequence alignment on an FPGA, Proceedings of the 11th International Conference on Parallel and Distributed Systems, 2, 326-330.

OLIVER, T.F., SCHMIDT, B., and MASKELL, D.L., 2005, Hyper customized processors for biosequence database scanning on FPGAs, 13th ACM International Symposium on Field-Programmable Gate Arrays, 229-237.

PARACEL, INC., 2005, http://www.paracel.com.

PEARSON, B., 1991, LALIGN - find multiple matching subsegments in two sequences, http://www.ch.embnet.org/software/ LALIGN_form.html.

PETERLONGO, P., PISANTI, N., PEREIRA DO LAGO, A., and SAGOT, M.F., 2006, Lossless filter for long multiple repetitions with edit distance, Department of Computer Science, University of Pisa.

RABINER, L.R., 1989, A Tutorial on hidden markov models and selected applications in speech recognition, Proceedings of the IEEE, 77, 2, 257-286.

RAPHAEL, B., ZHI, D., TANG, H, and PEVZNER, P., 2004, A novel method for multiple alignment of sequences with repeated and shuffled elements, Genome Research, 14, 11, 2336-2346.

REDELINGS, B.D. and SUCHARD, M.A., 2007, Incorporating indel information into phylogeny estimation for rapidly emerging pathogens, BMC Evolutionary Biology, 7, 40-58.

REICHERT, T.A., COHEN, D.N., and WONG, A.K.C., 1973, An application of information theory to genetic mutations and the matching of polypeptide sequences, Journal of Theoretical Biology, 42, 245-261.

SANKOFF, D., 1972, Matching sequence under deletion-insertion constraints, Proceedings of National Academy of Sciences, 61, 4-6.

SCHMIDT, B., SCHRODER, H., and SCHIMMLER, M., 2002, Massively parallel solutions for molecular sequence analysis, Proceedings of the International Parallel and Distributed Processing Symposium, 186-192.

SCHWARTZ, A.S., KENT, W.J., SMIT, A., ZHANG, Z., BAERTSCH, R., HARDISON, R.C., HAUSSLER, D., and MILLER, W., 2003, Human - mouse alignments with BLASTZ, Genome Research, 13, 1, 103-107.

SCHWARTZ, A.S. and PACHTER, L., 2007, Multiple alignment by sequence annealing, Bioinformatics, 23, 24–29.

SELLER, P.H., 1974, On the Theory and Computation of Evolutionary Distances, SIAM: Journal on Applied Mathematics, 26, 787-793.

SLIM SEARCH LTD., 2007, http://www.slimsearch.com.

SMITH, T.F. and WATERMAN, M.S., 1981, Identification of common molecular subsequences, Journal of Molecular Biology, 147, 1, 195-197.

SMITH, T.F., WATERMAN, M.S. and FITCH, W.M., 1981, Comparative biosequence metrics, Journal of Molecular Evolution, 18, 38-46.

SONNHAMMER, E.L.L. and DURBIN, R., 1995, A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis, Gene, 167, 1-10.

SZE, S.H., LU, Y., and YANG, Q, 2006, A polynomial time solvable formulation of multiple sequence alignment, Journal of Computational Biology, 13, 309-319.

THOMPSON, J.D., HIGGINS, D.G., and GIBSON, T.J., 1994, CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice, Nucleic Acids Research, 22, 4673-4680.

THOMPSON, J.D., GIBSON, T.J., PLEWNIAK, F., JEANMOUGIN, F., and HIGGINS, D.G., 1997, The ClustalX windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools, Nucleic Acids Research, 25, 4876-4882.

WANG, J., KEIGHTLEY, P.D. and JOHNSON, T., 2006, MCALIGN2: faster, accurate global pairwise alignment of non-coding DNA sequences based on explicit models of indel evolution, BMC Bioinformatics, 7, 292-306.

WATERMAN, M.S., SMITH, T.F., and BEYER, W.A., 1976, Some biological sequence metrics, Advances in Mathematics, 20, 3, 367-387.

WERNERSSON, R. and PEDERSEN, A.G., 2003, RevTrans – Constructing alignments of coding DNA from aligned amino acid sequences, Nucleic Acids Res., 31, 13, 3537-3539.

WILLIAMS, G., 2002, Tranalign : align nucleic coding regions given the aligned proteins (EMBOSS), http://bioweb.pasteur.fr/seqanal/interfaces/tranalign.html.

YAMAGUCHI, Y., MARUYAMA, T., and KONAYAGA, A., 2002, High speed homology search with FPGAs, Proceedings of the Pacific Symposium on Biocomputing, 271-282.

YU, C.W., KWONG, K.H., LEE, K.H., and LEONG, P.H.W., 2003, A Smith-Waterman systolic cell, Proceedings of International Conference on Field Programmable Logic and Applications, 375-384.

BIOGRAPHY

I was born in Ankara, at 1983. I completed my elementary education at Anıttepe İlköğretim Okulu, Ankara in 1994. I went to high school at Bahçelievler Deneme Lisesi, Ankara in 1997.

I graduated from University of Çukurova, Electrical-Electronics Engineering Department in 2005. Since this date, I am a research engineer in University of Çukurova, Computer Engineering Department.

I am interested in digital design, digital arithmetic, application specific processors, FPGAs, web design and applications.