Hardware Acceleration
of Bioinformatics
Sequence Alignment Applications

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. ir. K. C. A. M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op maandag 6 juni 2011 om 12.30 uur

door

Laiq HASAN,
Master of Science in Electrical Engineering
N-W.F.P. University of Engineering and Technology, Peshawar, Pakistan

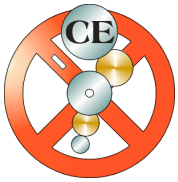
geboren te Swabi, Pakistan.

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. ir. H. J. Sips

Copromotor: Dr. ir. Z. Al-Ars

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. H. J. Sips	Technische Universiteit Delft, promotor
Dr. ir. Z. Al-Ars	Technische Universiteit Delft, copromotor
Prof. dr. W. Anheier	Universität Bremen
Prof. dr. O. Nieto-Taladriz Garcia	Universidad Politecnica de Madrid
Prof. dr. ir. C. Vuik	Technische Universiteit Delft
Prof. dr. ir. M. J. T. Reinders	Technische Universiteit Delft
Dr. ir. T. G. R. M. van Leuken	Technische Universiteit Delft
Prof. dr. ir. P. F. A. Van Mieghem	Technische Universiteit Delft, reservelid



This thesis has been completed in partial fulfillment of the requirements of Delft University of Technology (Delft, The Netherlands) for the award of the Ph.D. degree. The research described in this thesis was supported in parts by three institutions. (1) CE Lab. Delft University of Technology, (2) HEC Pakistan, (3) UET Peshawar, Pakistan.

Published and distributed by: Laiq Hasan, E-mail: laiqhasan@gmail.com

ISBN: 978-90-72298-19-5

Keywords: Bioinformatics, Sequence Alignment, Hardware Acceleration, Systolic Arrays, Recursive Variable Expansion, FPGAs, GPUs, Performance Analysis.

Printed in The Netherlands

Dedicated to:

The Sunday morning Sun, cool breeze and clear blue skies.

Summary

Biological sequence alignment is an important and challenging task in bioinformatics. Alignment may be defined as an arrangement of two or more DNA or protein sequences to highlight the regions of their similarity. Sequence alignment is used to infer the evolutionary relationship between a set of protein or DNA sequences. An accurate alignment can provide valuable information for experimentation on the newly found sequences. It is indispensable in basic research as well as in practical applications such as pharmaceutical development, drug discovery, disease prevention and criminal forensics.

Many algorithms and methods, such as, dot plot, Needleman-Wunsch, Smith-Waterman, FASTA, BLAST, HMMER and ClustalW have been proposed to perform and accelerate sequence alignment activities. However, with the ever increasing volume of data in bioinformatics databases, the time needed for biological sequence alignment is always increasing. The main aim of the research presented in this thesis is to explore and analyze the existing sequence alignment methods and come up with better and optimized solutions. The following research goals have been achieved during the course of this thesis.

1. Classification and comparison of the available sequence alignment methods with the emphasis on identifying the most optimal but computationally expensive methods that are best suited for hardware acceleration.

2. Optimized systolic array implementations of the Smith-Waterman based sequence alignment on FPGAs.

3. A novel FPGA implementation of sequence alignment based on recursive variable expansion and its performance evaluation.

4. An optimized and high performance GPU-based protein sequence alignment and its comparison with the existing GPU solutions.

5. Detailed performance analysis and optimization of the hardware-based sequence alignment, considering the limiting factors like computational resources, memory bandwidth and power consumption.

6. Introduction of a technique based on hardware partitioning to improve performance by reducing the hardware overhead cost.

Samenvatting

Biologische sequentie uitlijning is een belangrijke en uitdagende taak in bioinformatica. Uitlijning kan gedefinieerd worden als een rangschikking van twee of meer DNA- of eiwitsequenties om gelijkvormig delen te markeren. Sequentieuitlijning wordt gebruikt om de evolutionaire relatie tussen een set eiwitten of DNA-sequenties af te leiden. Een nauwkeurige uitlijning kan waardevolle informatie voor experimenten op nieuw ontdekte sequenties opleveren. Het is onontbeerlijk in basisonderzoeken evenals in praktische toepassingen zoals farmaceutische ontwikkelingen, medicijn onderzoek, ziekte preventie en forensisch onderzoek.

Vele algoritmes en methoden, zoals dot plot, Needleman-Wunsch, Smith-Waterman, FASTA, BLAST, HMMER en ClustalW zijn beoogd om de sequentieuitlijning uit te voeren en te versnellen. Echter, met het alsmaar groeiende volume van de data in bio-informatica databanken, groeit ook alsmaar de tijd benodigd voor biologische sequentie uitlijning. Het voornaamste doel van het in dit proefschrift gepresenteerde onderzoek is het verkennen en analyseren van bestaande sequentieuitlijningsmethoden en het opzetten van een betere, optimalere oplossing. De volgende onderzoeksdoelen zijn voltooid in het kader van dit proefschrift.

1. Classificatie en vergelijking van beschikbare sequentieuitlijningsmethoden met nadruk op het identificeren van de meest optimale maar mathematisch kostbaarste methoden die het best passen bij hardwareversnelling.

2. Optimalisatie van systolic-array uitvoeringen van de op Smith-Waterman gebaseerde sequentieuitlijning in FPGAs.

3. Een nieuwe FPGA-implementatie van sequentieuitlijning gebaseerd op ‘recursive variable expansion’ en zijn performance evaluatie.

4. Een geoptimaliseerde en high performance GPU-gebaseerd eiwit sequentieuitlijning en de vergelijking met bestaande GPU-oplossingen.

5. Gedetailleerde performance evaluatie en optimalisatie van hardware gebonden sequentieuitlijning, met het oog op beperkende factoren zoals computer hulpbronnen, geheugenbandbreedte en energieverbruik.

6. Introductie van een werkwijze gebaseerd op hardware partitionering om de prestatie te verbeteren door het terugbrengen van de hardware overhead kosten.

Acknowledgments

First and foremost, I would like to thank Allah SWT, as counting up His favors is inconceivable to categorize. “If you would count up the favors of Allah, never would you be able to number them: for Allah is Oft-Forgiving, Most Merciful.” (Quran, Surah Al-Nahl, Verse 18)

Next, I thank my family and friends back home, who continuously keep praying for my success, without expecting anything in return. Thanks are due to my friends and acquaintances, in The Netherlands in general and Delft in particular, whose support was necessary for my long stay here. Thanks are also due to my friends abroad, spread all over the world and whom I keep visiting from time to time. They really make me feel like a global citizen.

In the Computer Engineering Laboratory, my first thanks go to the late professor Stamatis Vassiliadis, who gave me the confidence and courage to continue with my Ph.D. God may rest his soul in peace. Next, thanks to prof. dr. ir. H. J. Sips for agreeing to be my promotor and approving the thesis. After Stamatis and H. J. Sips, the next thanks of course go to my co-supervisor, dr. ir. Zaid Al-Ars, who always remained patient and optimistic about my work and guided me to the best of his abilities throughout my Ph.D. Thanks are due to dr. Koen Bertels and dr. Georgi Gaydadjiev, who always helped and encouraged during the course of my Ph.D. Thanks to other faculty members in the laboratory, Said Hamdioui, Sorin Cotofana, Arjan van Genderen and Stephan Wong too, who have always been friendly and cooperative. Thanks are also due to my colleagues Ioannis Sourdis and Blagomir Donchev for giving me a start in VHDL, Sebastian Isaza for his friendly discussions and improving the presentation for my first colloquium talk, Bogdan Spinean for helping me out during our travel to Rabat, Morocco for DTIS’07, soon after the major surgery on my left kidney at Erasmus Medical Center in Rotterdam. Bundles of thanks to my colleagues Yao Wang and Mottaqiallah Taouil for their handy advices about my work during the later part of my Ph.D. Thanks are also due to Marijn Kentie and Erik Vermij for their contributions in the practical part of the research while doing their M.Sc. projects with me and Zaid. I also thank all other colleagues at Computer Engineering Laboratory,

for providing a friendly and research conducive environment. It would be unfair if I do not mention the technical and administrative support provided by Bert Meijs, Erik de Vries, Eef Hartman, Lidwina Tromp and Monique Tromp. Their support gave me the luxury of focusing only on my work, without caring much about the related issues.

Outside the Computer Engineering Laboratory, thanks to Franca Post from CICAT (TU Delft), Loes Minkman from NUFFIC (The Netherlands), the concerned people from the Higher Education Commission of Pakistan and University of Engineering and Technology Peshawar Pakistan. They smoothly processed all my financial and 'study leave' related issues, thus making my life relatively easy. Thanks are due to my friends Wouter van der Sluis, Weiman Chim, Saleh Safiruddin, Johan Splinter and Serge Keyser for helping me in translating the summary of my thesis and propositions into Dutch. Also, my humble thanks to all my friends and their families for taking good care of me during my necessary treatment at Erasmus Medical Center in Rotterdam in August 2007. Special thanks go to Ahmad Jan, Saad Hassan Mirza and my dear friend Waqar who remained with me for around 10 days in the hospital, my friend Malik Aleem Ahmad for taking good care for more than a month during my post hospital rest period, Mehfooz, Hamayun, Hisham, Haleem Bangash, Aqeel, Ahson Jabbar, Ahsan Shabir, Haroon, Omer, Noman, Sant Paul, Jawad, Plamen Gonchosov, Blagormir Donchev and all other friends who visited me on regular basis and kept me away from loneliness, desperation and frustration.

For the leisure times, my first thanks go to the Computer Engineering Laboratory for organizing those enjoyable social events that are an integral part of life at the laboratory. Next, I thank Saleh, Marius, George, Mihai, Chunyang, my Belgian friend Glen and my Swedish friend Henrik for their company and playing my favorite sport (tennis) with me. I also thank all my Pakistani, Indian, Chinese, Iranian, Dutch and other friends for the cricket, volleyball, squash, swimming, cycling, going to cinemas and other amazing activities in which we participated together. These activities were the best part of my stay in The Netherlands. Thanks to all my friends in Delft and outside (specially Seyab, Faisal Nadeem, Faisal Kareem, Fakhar, Mehfooz, Cheema, Hamayun, Dev, Atif, Sandilo, Hanan, Zahid Shabbir, Nadeem, Zubair, Tariq, Hisham, Sharif Ullah, Husnul Amin, Saleem, Haider, Mazhar, Amir, Sarfaraz, Samad Khan, Shakir bhai, Niaz, Imran, Adeel, Zaidi, Mafalda and others), who made my stay in The Netherlands sociable, by inviting me in so many parties and gatherings that they kept organizing from time to time. I think, I should write a memoir at some point in time to elaborate all these activities and the people involved.

Finally, I thank members of the opposing committee for devoting some of their precious time to scrutinize my thesis, give their valuable feedback and above all travel to Delft for the public defense of this dissertation.

Laiq Hasan
May 09, 2011

Contents

List of Tables

List of Figures

Abbreviations and Symbols

1D	-	1-dimensional or linear
2D	-	2-dimensional
b_f	-	blocking factor
A	-	Adenine
BLAST	-	Basic Local Alignment Search Tool
BRAM	-	Block RAM
C	-	Cytosine
CCU	-	Custom Computing Unit
CUDA	-	Compute Unified Device Architecture
CUPS	-	Cell Updates Per Second
DDR	-	Double Data Rate
DNA	-	Deoxyribonucleic Acid
DOPA	-	Database Optimized Protein Alignment
DP	-	Dynamic Programming
FASTA	-	Fast Alignment Search Tools - All
FPGAs	-	Field Programmable Gate Arrays
G	-	Guanine
GPUs	-	Graphic Processing Units
gprof	-	GNU profiler
HGP	-	Human Genome Project
HMMs	-	Hidden Morkov Models
HSPs	-	High-scoring Segment Pairs

INSDC - International Nucleotide Sequence Database Collaboration
MGAP - Micro Grained Array Processor
NCBI - National Center for Biotechnology Information
N-W - Needleman-Wunsch
PE - Processing Element
PIR - Protein Information Resource
QSP - Query Sequence Partitioning
RNA - Ribonucleic Acid
RTR - Run-time Reconfiguration
RVE - Recursive Variable Expansion
S-W - Smith-Waterman
ssearch - Smith-Waterman search
SIMD - Single-Instruction stream, Multiple-Data stream
T - Thymine
tRNA - transfer-RNA
U - Uracyl

Chapter 1

Introduction

With the ever increasing volume of data in the bioinformatics databases, the time for comparing a query sequence with the pre-existing sequences in the databases is always increasing. Researchers in various communities are working on accelerating the available methods for comparing and aligning these sequences. This thesis presents one such work. This chapter provides a brief overview of bioinformatics in general with a particular emphasis on sequence alignment.

The chapter starts with an overview of molecular biology, presented in Section 1.1. It is followed by an introduction to bioinformatics, sequence alignment, its types and applications, presented in Section 1.2. Further, it presents the acceleration approaches for sequence alignment in Section 1.3 followed by an overview of the thesis contribution. An outline of the thesis is presented in Section 1.4. The chapter concludes with a summary, presented in Section 1.5.

1.1 Molecular biology - an overview

The field of bioinformatics is the application of computer science to biology in general and molecular biology in particular. This can involve the development of algorithms and software that can analyze huge amounts of data, the automation of previously labor intensive tasks, or the creation of tools, for example with which to view 3D models of biological structures. Although no in-depth knowledge of the chemical processes involved is required from the perspective of computer scientists and hardware design experts, subjects such as *Deoxyribonucleic Acid (DNA)* and protein construction are integral to understanding the relevance of research topics like sequence alignment. In the following subsections, a recap of the basics of molecular biology is presented.

1.1.1 Cells, amino acids and proteins

All living organisms consist of one, or many more, of a basic functional unit, the cell. Classified as being 'alive' (the smallest organisms consist of a single cell), cells can process and excrete molecules (metabolism), alter their electrical potential and procreate by cell division. Many of the processes inside cells are governed by proteins. Proteins are complex chains of molecules called amino acids. Some amino acids, the 'non-essential' ones, can be synthesized by the cell. The other, essential, amino acids must be procured through the ingestion and breakdown of proteins in foods such as meat. Again, this breaking down of food products is performed by proteins, this time existing outside of any cell.

Proteins exist for a wide array of functions, for instance *actin* aids in muscle contraction while the proteins of the cytoskeleton form a cell's 'skeleton', giving it its shape and protection. Another important role of proteins is to act as a catalyst, where the proteins are called enzymes. Enzymes act as catalysts by binding to the reagents of a reaction and lowering the activation energy required for it to take place. Designed to only be compatible with those specific reagents due to their structure, enzymes are not consumed in the reaction and can be reused. Returning to the example of breaking down food into nutrients, there are enzymes that split proteins into their component amino acids, enzymes which break down fat molecules and enzymes that allow ingested nucleic acid to be reused for the construction of DNA.

1.1.2 Chromosomes and DNA

The cells require the presence of proteins, both internally and externally, to survive. In fact, the reproduction of cells relies heavily on proteins too, such as those of the aforementioned cytoskeleton facilitating the division of the cell membrane. Proteins are created, from scratch and to specification, within the cell itself. This is where the DNA comes in. DNA is stored in structures called chromosomes. Made up of the DNA molecules and a supporting protein packaging, chromosomes are 'wadded up' in the cell similar to a ball of string. Attached to these chromosomes, the DNA is protected and more compact; this way it is able to fit in the cell (nucleus). The structure and number of chromosomes varies on a per species bases, additionally the shape of chromosomes is also determined by what stage of its life cycle the cell currently resides in.

The DNA itself contains the genetic instructions that describe how the various proteins should be constructed. The structure of DNA is shown in Figure 1.1. Structurally, DNA consists of two long, coiled nucleotide polymer strands that take the well-known double-helix form. These polymers are strengthened by a skeleton of sugars and phosphate groups, connected to these sugars are the bases, pairs of molecules that specify the genetic code. Each strand of DNA has one end called the $3'$ end, the other end is called the $5'$ end. Due to the anti parallel nature of the strands, their ends are mirrored. When talking about DNA, these ends can be used to indicate in which direction a strand is being built/interpreted/etc.

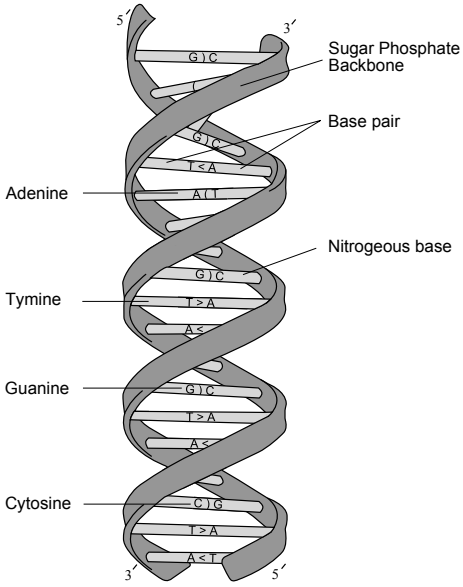


Figure 1.1: The structure of DNA

Four different bases exist, i.e. *adenine (A)*, *cytosine (C)*, *guanine (G)* and *thymine (T)*. A base on one strand will be matched by its twin base on the other strand and connected to it using hydrogen bonds. Adenine is always paired with thymine whereas cytosine and guanine form the second combination. This duplication of the bases is central to the replication of DNA, and as such, to that of the cell and the survival of the host organism.

During the replication of DNA mutations can occur, altering the genes of a host organism. This ties into the theory of evolution and the field of phylogenetics, the study of relatedness among organisms by comparing their genetic makeup.

1.1.3 RNA and transcription

The bases of DNA can be seen as letters (A, C, G and T). These letters are interpreted in words of three, called codons. Each codon describes a single amino acid, the building blocks of proteins. A portion of DNA that codes for a protein is known as a gene. As there are four bases and they appear in words of three, there exist 43 possible codons. However, only 20 different amino acids are encoded. This means that some words code for the same amino acid. Additionally, some words have special functions: the start (ATG) and stop (TAG, TAA, TGA) codons function as markers to aid in the correct interpretation of the code at the RNA stage. A sequence of codons that starts with a start codon and ends with a stop codon is called an open reading frame. The process of interpreting the genetic code and using it to synthesize proteins

is called genetic expression. The first step is the generation of *ribonucleic acid (RNA)*, which will mirror the gene in question and be transported to the cell's 'protein factory'. Genes are transcribed to RNA by an enzyme called RNA polymerase. This is bound to the correct place on the DNA by means of a promoter, which is a sequence of codons that influences the binding of RNA polymerase directly or indirectly by means of proteins. The DNA strand, the RNA will be based on, is called the coding strand. When generating RNA, the strands are separated and the complementary strand, called the template strand, is walked in the $3' \longrightarrow 5'$ direction. The strand's bases are then paired with a new strand of again complementary bases (with thymine replaced by *uracyl (U)*). This is the RNA. This strand is separated from the DNA once transcription is complete, after which the DNA's structure is restored. In effect, the created RNA is a copy of the coding strand with T replaced by U.

Example: consider a strand of DNA coding a gene:
$5'$ A T G G C C T G G A C T T C A ... $3'$ coding strand
$3'$ T A C C G G A C C T G A A G T ... $5'$ template strand

The resultant RNA will then be:
$5'$ A U G G C C U G G A C U U C A ... $3'$

Note the start codon ATG (AUG for the RNA). This RNA is transported to the ribosomes, the cell components which assemble proteins by chaining together amino acids. Here the RNA is walked and interpreted from the start to the stop codon. The codons are interpreted by means of *transfer-RNA (tRNA)*. These molecules also contain a complementary codon to match with the RNA and carry an amino acid to link up to the protein. The ribosomes themselves again consist of proteins and ribosomal RNA.

This recap of genetic expression glosses over many things, including introns/exons, the various RNA types, DNA/RNA quality control and the roles of proteins such as repressors. Although the process is more involved than described here, especially in humans, more information is not required to understand the basics of bioinformatics and sequence alignment, presented in the next section. Readers interested in further details may refer to [1] and [2].

1.2 Bioinformatics

Biology is in the middle of a major paradigm shift, driven by computing technology. Two decades before the formal inauguration of the *Human Genome Project (HGP)*, a new hybrid field (partly molecular biology and partly computer science) began to emerge. The new field was called *computational molecular biology* or *bioinformatics*, which may be defined as a discipline that generates computational tools, databases, and methods to support genomic and post genomic research. Bioinformatics is the multidisciplinary research area aimed at organizing and classifying the immense rich-

ness of sequence data, where the sequence may refer to either DNA or protein.

Bioinformatics employs a digital language for representing its information using the four basic alphabets (A, C, G, T). All the DNA molecules in an organism's cell have been represented and being identified using these alphabets. The tools of computer science, statistics and mathematics are very critical for studying bioinformatics. Some of the recent advances happened include improved DNA sequencing methods, new approaches to identify protein structure and revolutionary methods to monitor the expression of many genes in parallel. The following subsection presents major fields of bioinformatics.

1.2.1 Fields of bioinformatics

A wide variety of research topics are being explored by researchers in the diversified field of bioinformatics. Examples are, gene structure prediction, phylogenetic trees, protein structure prediction (2D, 3D), sequencing (i.e. mapping) genomes etc. Figure 1.2 gives a broad classification of major research areas in bioinformatics.

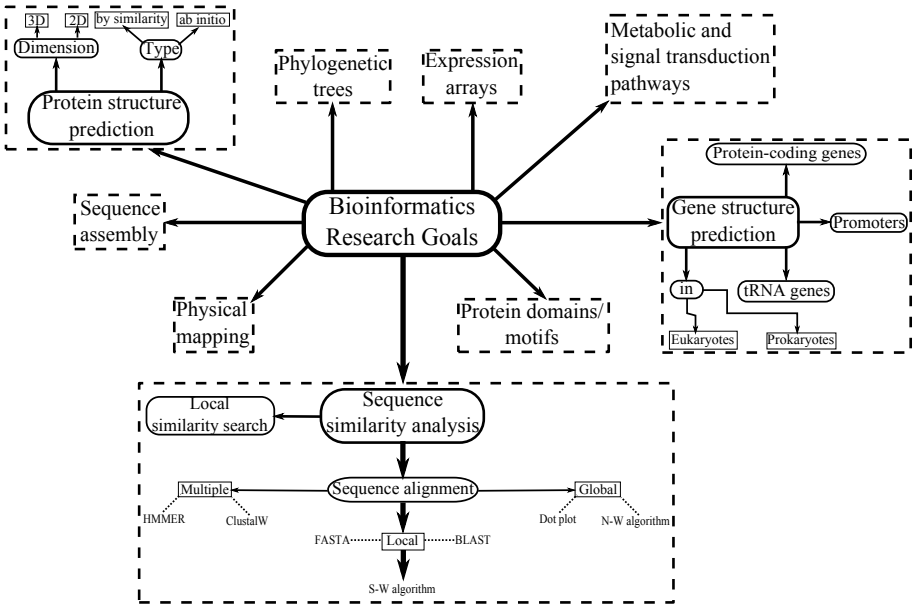


Figure 1.2: Classification of bioinformatics research areas

The relevant field among these to this thesis is the sequence similarity analysis or *sequence alignment*. A significant part of bioinformatics is the analysis of sequences, where the sequences of interest in molecular biology are those of DNA and proteins. As discussed before, DNA consists of the four bases A, C, G and T. One might say that DNA is a sequence, or string, of the alphabet {A,C,G,T}. Not surprisingly, RNA can

be looked at similarly, with the alphabet {A,C,G,U}. Whereas, proteins, too, can be viewed as strings of an alphabet [3]. In this case, the alphabet of the 20 amino acids is {A,C,D,E,F,G,H,I,K,L,M,N,P,Q,R,S,T,V,W,Y}. The amino acids corresponding to these letters are shown in Table 1.1.

Table 1.1: The 20 amino acids

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

Numerous projects for sequencing the DNA of particular organisms constantly supply new amounts of data on an enormous scale [4], with a doubling time estimated to be 9-12 months. The bioinformatics industry has grown to keep up pace with this information explosion, growing at 25-50% a year. In 2000, the US market Research company, Oscar Gruss [5] estimated that the value of the bioinformatics industry would touch $ 2 billion. With the ever increasing volume of sequence data in various bioinformatics databases from *International Nucleotide Sequence Database Collaboration (INSDC)* [6] (e.g. "public" repositories of gene data like GenBank from *National Center for Biotechnology Information (NCBI)* [7], *SwissProt* from the *Swiss Institute of Bioinformatics* [8] and *PIR* from the *Protein Information Resource* [9]), the time for comparing a query sequence with the available databases is always increasing. It could take weeks to months for a researcher to search sequences by hand in order to find related genes or proteins. Computer technology has provided the obvious solution to this problem. Not only can computers be used to store and organize sequence information into databases, but they can also be used to analyze sequence data rapidly. The evolution of computing power and storage capacity has, so far, been able to outpace the increase in sequence information being created. Theoretical scientists have derived new and sophisticated algorithms which allow sequences to be readily compared using probability theories. These comparisons become the basis for determining gene function, developing phylogenetic relationships and simulating protein models. On the other hand, hardware design experts have been working on designing and accelerating the more accurate methods of sequence alignment. In the following subsections, sequence alignment, its types and application are elaborated.

1.2.2 Sequence alignment and its types

In most common terms sequence alignment may be defined as an arrangement of two or more DNA or protein sequences to highlight the regions of their similarity. This, in turn indicates the genetic relatedness between the organisms. The similarity may be a consequence of functional, structural or evolutionary relationship between

the sequences [10]. New DNA, RNA and protein sequences develop from the pre-existing sequences rather than get invented by nature from the scratch. This fact is the foundation of any sequence analysis.

If two DNA, RNA or amino acid sequences are similar, there is a chance that they are homologous. Homologous sequences share a common ancestral sequence, their relative differences are the result of mutations. These mutations might manifest in various ways: substitutions, where one symbol is replaced by another, insertions where a new symbol is inserted into the sequence and deletions, the removal of a symbol. To establish the degree of homology, the sequences are aligned i.e. lined up in such a way that the degree of similarity is maximized. This process is known as sequence alignment, which can be classified into various types as shown in Figure 1.3. Following is a brief description of these types.

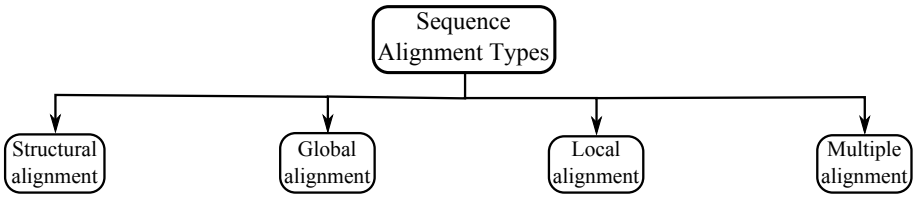


Figure 1.3: Types of sequence alignment

Structural alignment

Structural alignment [11] is an approach of attempting to infer similarity between proteins by comparing their three dimensional shapes, or tertiary structures. As a protein's shape is determined by its amino acid makeup, which, in turn, determines its function, it is obvious that structural alignment is an attractive tool for homology research. In fact, different protein letter sequences might result in similar 3D structures, where protein structure is better evolutionary conserved than sequence [12]. Unfortunately, determining the tertiary structure of proteins requires costly, time consuming procedures such as X-ray crystallography and nuclear magnetic resonance imaging (bioinformatics databases contain much less protein structures than sequences) [2]. One field of bioinformatics, i.e. protein structure prediction concentrates on unraveling the mysteries behind protein folding, the process in which an unfolded random coil amino acid takes its characteristic tertiary structure. Using computational protein folding, any of the myriad available protein sequences could be converted to a 3D representation. Then, in turn, structural alignment could be used to infer homology. Currently, however, protein folding is still an open problem and current approaches have such high computational requirements that researchers have turned to super or distributed computing [13]. Although mainly used for proteins, structural alignment is also promising for strands of RNA [14]. It is not suitable for DNA as this always takes the double-helix structure.

Global alignment

Global alignment methods operate directly on all sequence letters. The idea is to line up two (or more) sequences so that their degree of similarity is maximized. For DNA and RNA this means matching identical bases. In the case of proteins, amino acids are matched if they are identical or can be derived from one another through likely to occur substitutions [2]. Although matching two sequences directly will take into account substitution/mutations to handle insertions and deletions, the notion of *gaps* is introduced. Marked by the symbol '-', a gap can be chosen to be inserted into any of the sequences to obtain a closer match. Following is an example with the base sequences TACCAGT and CCCGTAA:

No gaps

T	A	C	C	A	G	T
C	C	C	G	T	A	A

Gaps

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

Clearly the alignment with gaps is more relevant and better exposes the similarities between both sequences. Note that other alignments are possible. An option would be:

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

As multiple alignments are possible even in this simple case, it makes sense to devise a way to rate and then select the best alignment(s). A simple method to accomplish this is to assign scores to the alignment letters. A simple scheme is 1 for a match, -1 for a mismatch and -2 for a gap. Such a scheme is said to have a *linear gap penalty*. A more advanced method is to introduce an *affine gap penalty*, which assigns different scores to the starting of a new gap and the extension of a current one. Generally, starting a new gap is given the largest penalty as this is biologically the hardest [15]. Using the aforementioned scoring system, the first gapped alignment scores (-1-2+1+1-2+1+1-2-2)=-5 and the second option does so as well with (-2-2+1+1-1+1+1-2-2)=-5. So in this case, both gapped alignments are 'as good' as one another. However, this does not automatically mean they both have the same biological relevance. To judge how relevant an alignment's score is, probabilistic methods can be used. The idea is to check whether the probability of an alignment attaining the score in question is adequately small (Chapter 7 of [15]). In case of using an affine gap penalty, the second alignment would have the best score, as it contains two gaps instead of three. The same approach can be used for amino acids as opposed to DNA bases. Instead of working with fixed scores, amino acid substitutions have been rated by their evolutionary likeliness and are available as standard 20×20 triangular substitution matrices. The two most well known matrices are the PAM and BLOSUM families. Table 1.2 shows the BLOSUM62 matrix. Example of global alignment is the *Needleman-Wunsch (N-W)* algorithm [16].

Table 1.2: The BLOSUM62 amino acid substitution matrix

	Ala	Arg	Asn	Asp	Cys	Gln	Glu	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val
Ala	4																			
Arg	-1	5																		
Asn	-2	0	6																	
Asp	-2	-2	1	6																
Cys	0	-3	-3	-3	9															
Gln	-1	1	0	0	-3	5														
Glu	-1	0	0	2	-4	2	5													
Gly	0	-2	0	-1	-3	-2	-2	6												
His	-2	0	1	-1	-3	0	0	-2	8											
Ile	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
Leu	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
Lys	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
Met	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
Phe	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
Pro	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
Ser	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
Thr	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
Trp	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Tyr	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
Val	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4

Local alignment

Local alignments are similar to global ones. The only difference is that instead of attempting to align the complete sequences to one another, portions of similarity are aligned. Following is an example with sequences GTGTACTCCAGAG and GTACCCAAG:

Global alignment

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

Local alignment

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

Looking for a local alignment will better expose 'patches' of homology in two relatively dissimilar sequences. Thus it might lead to more biologically relevant results [15]. Example of local alignment is the *Smith-Waterman (S-W)* algorithm [17].

Multiple alignment

The previous examples focused on aligning just two sequences, but in some cases it might be interesting to consider the similarities between a group of sequences. For example, if the structure of a protein is unknown, a similarity to a group of other proteins might give clues. Global and local alignment algorithms can be adapted to deal with multiple alignments, though this quickly becomes extremely computationally expensive. An alternative is to use specifically designed heuristic algorithms, for example *ClustalW* [18].

1.2.3 Applications of sequence alignment

Sequence alignment has many applications in bioinformatics. Figure 1.4 presents some examples. Following is a brief description of these applications.

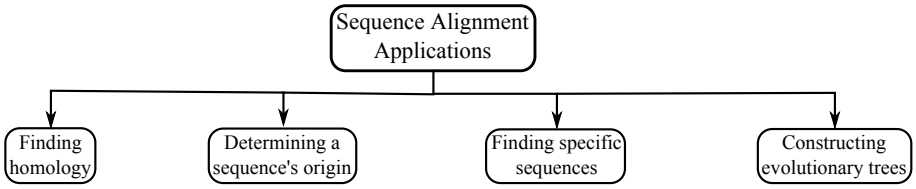


Figure 1.4: Examples of sequence alignment applications

Finding homology

One of the main uses for sequence alignment is to find homology. Homology means that two sequences share a common ancestor; evolution says that all cells must eventually trace back to the same ancestor. Finding homology between organisms might enable knowledge of one to be applied to the other, or to infer the function of one organism's gene from that of a related species.

Determining the origin of a sequence

If a DNA or protein sample is recovered but its originating species are unknown, sequence alignment can be used to find likely sources, i.e. the known sequences most closely matching the sample.

Finding specific sequences

Suppose we have discovered the function of a part of species *X's* genetic code. Then it might be attractive to search species *Y's* code for the sequence. If something similar is found, it might give clues as to the location of a similar gene in *Y*. Similarly, suppose that we might have found the piece of code that expresses a trait, such as a physical characteristic or the presence of a genetic disease, in one piece of genetic code. Searching other pieces known to either feature or lack this trait might help validate or disprove the theory.

Constructing evolutionary trees

From homology data, evolutionary (phylogenetic) trees can be constructed [15]. These trees are built using the 'genetic distance' between species and give insight into species relationships and the course of evolution. Using the concept of an evolutionary rate, the species' sequence homology can be translated into the time they took to develop from ancestral species. The actual construction of the tree can be done in many ways; examples include maximum parsony methods (building the tree such that the lowest amount of evolutionary change is required) and distance methods such as the UPGMA algorithm which builds the tree from the result matrix of a multiple alignment.

The information presented in this section is annexed by the definitions of some important bioinformatics terms given in Appendix A. In the next section, acceleration of sequence alignment is presented followed by an overview of the thesis contribution.

1.3 Acceleration of sequence alignment

This section presents a broad classification of the methods used for acceleration of sequence alignment applications. Furthermore, it provides an overview of the contributions of this thesis for acceleration, analysis and optimization of such applications.

1.3.1 Methods of acceleration

Figure 1.5 presents a broad classification for acceleration of sequence alignment applications. The figure shows that the acceleration of sequence alignment applications can either be in hardware or software. However, the main focus of the thesis is on hardware acceleration, as shaded in the figure.

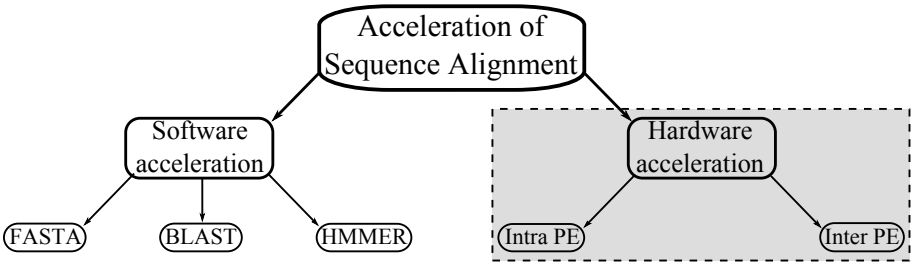


Figure 1.5: Broad classification of sequence alignment acceleration

Software acceleration

Heuristics based algorithmic modifications are done to achieve faster software implementations of the sequence alignment applications, as done in FASTA, BLAST and HMMER. These software implementations do not guarantee an optimal alignment though. Following is a brief description of such heuristics based software implementations.

FASTA: FASTA was developed in 1985 by Lipman and Pearson [19]. Unlike the N-W and S-W algorithms, FASTA approximates the optimal alignment by searching and matching k-tuples (i.e. subsequences of length k). The algorithm assumes that related proteins will have regions of identity. By searching with k-tuples, the FASTA algorithm allows small regions of local identity to be found quickly. For proteins, these k-tuples tend to be of length two.

BLAST: BLAST [20] is similar to the FASTA algorithm, however, it uses words (w) instead of k-tuples. The computational complexity of both FASTA and BLAST comes out to be $O(MN)$. The space complexity for FASTA is $O(MN)$, whereas for BLAST, it is higher than all other algorithms and it comes out to be $O(20^w + MN)$, where w is the word size.

HMMER: HMMER uses *Hidden Morkov Models (HMMs)* which are widely used in biological sequence analysis. HMMs are a probabilistic tool which can be used for sequence alignment [21], finding sequences in genetic code [2], inferring protein structure or building profiles of DNA and proteins [15]. Such profile HMMs can be used to determine whether a sequence is part of a family of DNAs or proteins. HMMs are based on the probability of a sequence adhering to certain characteristics. To determine these probabilities, machine learning principles are used. In the case of a profile HMM, the learning set is a multiple alignment. HMMs originated and still play a significant role in speech recognition [22].

Hardware acceleration

The heuristics based software implementations just described improves the performance at the cost of loosing accuracy. In contrast methods like the S-W algorithm provides accurate and optimal solution. But based on exhaustive search, the limitation of such methods is that they become too slow in practical situations, particularly for long sequences. The focus of the thesis is on hardware acceleration of such accurate methods on platforms like *Field Programmable Gate Arrays (FPGAs)* and *Graphic Processing Units (GPUs)*, which can either be related to the design of the basic *Processing Element (PE)* called *intra PE level* or related to multiple PEs organization called *inter PE level*. Following is a description of the acceleration at both levels.

Intra PE level: At this level of hardware acceleration, a basic building block, called the PE is designed for a sequence alignment application. The PE is capable of performing comparison between individual characters of the query and database sequences and is optimized for high performance and/or efficient resource utilization. In the later chapters of the thesis, such designs and optimizations are presented in detail. More specifically, high performance PE designs for systolic array and *recursive variable expansion (RVE)* based implementations are presented.

Inter PE level: At the inter PE level, the organization and interconnection of the PEs is optimized for high performance, efficient resource and bandwidth utilization. In this thesis, the PEs organized in *2-dimensional (2D)* and *1-dimensional or linear (1D)* systolic array fashion are presented. Further, 2D and 1D RVE implementations are discussed and compared with the corresponding systolic array implementations. Additionally, high performance GPU-based sequence alignment is presented that eliminates the need for inter PE communication. Also,

the issues related to bandwidth requirement and hardware redundancy are discussed in detail and performance and bandwidth optimizations are presented. Furthermore, an approach for high performance and resource efficient biological sequence alignment is presented. The succeeding chapters of the thesis give an insight to such optimizations and analysis.

1.3.2 Thesis contribution

The research performed in the course of this thesis has contributed in a number of ways to the hardware acceleration, performance optimization and analysis of bioinformatics sequence alignment applications. Different ways and means have been explored to accelerate such applications. Further, detailed performance analysis has been carried out, considering the limiting factors like computational resources and memory bandwidth. Following are the details of the contribution.

- A review of hardware acceleration of sequence alignment applications and their comparisons based on various parameters [23] is presented Chapters 2 and 3.
- An accurate profiling and acceleration evaluation procedure has been proposed [24] and presented in Chapter 3.
- An efficient and high performance systolic array architecture for biological sequence alignment [25,26] is presented in Chapter 3.
- An implementation based on the RVE approach to reduce the execution time at the cost of additional hardware resource utilization [26,27] is presented in Chapter 4.
- An optimized and high performance GPU-based protein sequence alignment outperforming the existing GPU solutions [28] is presented in Chapter 5.
- A comprehensive and elaborate mathematical performance and bandwidth analysis and optimization for biological sequence alignment with particular emphasis on the S-W algorithm [29] is presented and maximum theoretical performance boundaries are investigated in Chapter 6.
- An approach based on hardware partitioning is proposed to achieve high performance and resource efficient biological sequence alignment [30]. This approach is presented in Chapter 6.
- Power consumption evaluation and its impact on performance [31] is presented in Appendix E.

1.4 Thesis outline

The rest of the thesis is organized as follows.

Chapter 2 presents a classification of various sequence alignment methods and continues with a discussion of global, local and multiple methods in detail. It describes exact methods like dot plot, Needleman-Wunsch and Smith-Waterman and approximate methods like FASTA, BLAST, HMMER and ClustalW. The chapter ends with a comparison of the presented methods followed by a brief summary of the chapter.

Chapter 3 presents a classification of the various available acceleration methods for sequence alignment applications and proposes an accurate profiling and acceleration evaluation method using the MOLEN platform. Further, it presents FPGA-based rectangular (2D) and linear (1D) systolic array implementations for sequence alignment. It continues with the discussion of an extended linear systolic array design and ends with a brief summary of the chapter.

Chapter 4 presents RVE-based approach for sequence alignment and its comparison with traditional systolic array based approaches. Further, it presents rectangular and linear FPGA-based RVE implementations for sequence alignment and a discussion of the corresponding results. It continues with the RVE performance evaluation before concluding with a brief summary of the chapter.

Chapter 5 provides an introduction to GPUs, *Compute Unified Device Architecture (CUDA)* and its programming and memory models. The chapter explores the parallelization capabilities of GPUs for sequence alignments and reviews the available GPU-based approaches. It presents an optimized GPU implementation for S-W based protein sequence alignment. Further, it evaluates the performance of the optimized GPU implementation and compares it with the fastest available similar design. The chapter concludes with a brief summary.

Chapter 6 presents a comprehensive and elaborate performance and bandwidth analysis for sequence alignment. It continues with evaluating theoretical performance boundaries for various cases and optimizing bandwidth requirement. Further, it presents a method based on hardware partitioning to carry out high performance and resource efficient biological sequence alignment. Additionally, it develops equations to show the general trend of execution time reduction, resource utilization improvement and hence performance enhancement. The chapter ends with a brief summary.

The thesis ends with Chapter 7, where chapter wise brief conclusions are given, followed by a number of recommendations intended to identify future research directions.

The main content of the thesis is annexed by five appendices organized as follows.

Appendix A defines some important terms used in bioinformatis. Appendix B provides a dot plot implementation. Appendix C gives a couple of examples to explain the N-W algorithm. Appendix D explains S-W algorithm with the help of a flow chart and two examples. Appendix E presents power consumption evaluation for sequence alignment and its impact on performance.

At the end, a list of publications related to the thesis and a brief curriculum vitae of the author are given.

1.5 Summary

This chapter served as a simple introduction to the concepts associated with molecular biology, bioinformatics and sequence alignment in general. Further, it presented the methods for acceleration of sequence alignment applications and provided an overview of the thesis contribution. The main topics discussed are as follows.

- An overview of molecular biology including a brief discussion about cells, amino acids, proteins, chromosomes, DNA, RNA and transcription.
- An introduction to bioinformatics including a discussion about its fields with a particular emphasis on sequence alignment, its types and applications.
- Classification of sequence alignment acceleration and a discussion about the acceleration methods.
- An overview of the thesis contribution with references to the papers published during the course of the thesis.
- An outline of the thesis glancing at the topics to be presented in the following chapters.

Chapter 2

Sequence Alignment Methods

This chapter introduces a taxonomy of the various sequence alignment methods found in the literature. It describes in detail, exact methods like dot plot, Needleman-Wunsch and Smith-Waterman and approximate methods like FASTA, BLAST, HMMER and ClustalW. Further, it compares the presented methods based on their complexities and parameters like alignment type and the search procedure used.

It starts with a classification of sequence alignment methods, presented in Section 2.1, followed by a discussion about global methods like dot plot and Needleman-Wunsch in Section 2.2. Section 2.3 presents local methods like Smith-Waterman, FASTA and BLAST, whereas Section 2.4 presents multiple alignment methods like HMMER and ClustalW. Section 2.5 presents a comparison of various sequence alignment methods discussed in the previous sections, whereas Section 2.6 summarizes the chapter.

2.1 Classification of sequence alignment methods

Sequence alignment aims at identifying regions of similarity between two DNA or protein sequences (the query sequence and the subject or database sequence). Traditionally, the methods of pairwise sequence alignment [32] are classified as either global or local, where pairwise means considering only two sequences at a time. Global methods [33] attempt to match as many characters as possible, from end to end, whereas local methods [34] aim at identifying short stretches of similarity between two sequences [35]. However, in some cases, it might also be needed to investigate the similarities between a group of sequences, hence multiple sequence alignment methods are introduced. Multiple sequence alignment [36] is an extension of pairwise alignment to incorporate more than two sequences at a time. Such methods try to align all of the sequences in a given query set simultaneously. Figure 2.1 gives a classifica-

tion of various available sequence alignment methods. These methods are categorized into three types, i.e. global, local and multiple, as shown in the figure. Further, the figure also identifies the exact methods and approximate methods. The methods shown in Figure 2.1 are further elaborated in the following sections.

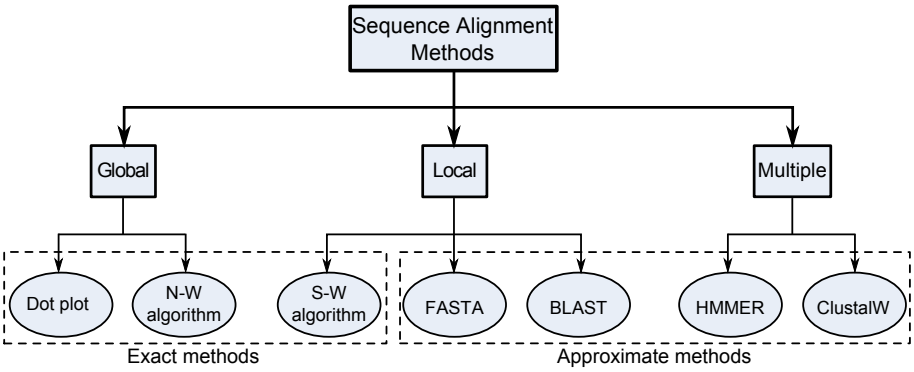


Figure 2.1: Various methods for sequence alignment

2.2 Global methods

As described earlier, global methods aim at matching as many characters as possible, from end to end between two sequences i.e. the query sequence (N_q) and the subject or database sequence (N_s). Methods carrying out global alignment include dot plot and Needleman-Wunsch algorithm. Both are categorized as exact methods. The difference is that dot plot is based on a basic search method, whereas Needleman-Wunsch on dynamic programming, as discussed in the following subsections.

2.2.1 Dot plot method

The most basic method of comparing two sequences is a visual approach known as a dot plot [37]. The sequences to be compared are arranged along the margins of a matrix. At every point in the matrix where the two sequences are identical, a dot is placed (i.e. at the intersection of every row and column that have the same letter in both sequences). A diagonal stretch of dots indicates regions where the two sequences are similar. Done in this fashion, a dot plot as shown in Table 2.1 is obtained (for clarity, dots are marked as ×s in the table).

In general two sequences are considered, i.e. the query sequence (N_q) and the subject or database sequence (N_s), whose lengths can be different, but in the ideal case are fairly similar. We proceed by creating a rectangular matrix in which the characters of N_q are mapped along the x-axis, and those of N_s along the y-axis. Initially, the

Table 2.1: Dot plot matrix

	a	c	t	g	g	a	c	t	g	g	a	c	t	g	g
a	×					×					×				
c		×					×					×			
t			×					×					×		
g				×	×				×	×				×	×
g				×	×				×	×				×	×
a	×					×					×				
c		×					×					×			
t			×					×					×		
g				×	×				×	×				×	×
g				×	×				×	×				×	×
a	×					×					×				
c		×					×					×			
t			×					×					×		
g				×	×				×	×				×	×
g				×	×				×	×				×	×

matrix is filled with zeros. Each of its cells, $x_i y_j$ (where i varies between 1 and the length of sequence N_q, and j varies between 1 and the length of sequence N_s), is considered in turn and is assigned a value indicating the level of similarity between the two residue positions (N_q and N_s). In the simplest scheme, all cells remain zero, unless $N_q = N_s$, in which case the element is assigned a value 1.

Such a matrix can be visualized quite simply for short sequences, for example by printing out the matrix in a particular font, as shown in Table 2.1 or for longer sequences, by using an appropriate graphics program. The plot is characterized by some apparently random dots (noise) and a central diagonal line, where a high density of adjacent dots indicates the regions of greatest similarity between the two sequences. For full length sequences, a plot must be reduced in size in order to be able to visualize the complete comparison and in doing so, the ×s in the magnified section shown in Table 2.1 are reduced to dots (hence the dot plot), which, at sufficiently low magnification, will ultimately merge into lines.

In contrast with identical sequences, two similar sequences will be characterized by a broken diagonal [37]. The time and space complexity of the dot plot is $O(MN)$, where M and N are the lengths of sequences N_q and N_s, respectively. This discussion is annexed by a parallel hardware design for dot plot in Appendix B.

2.2.2 Needleman-Wunsch algorithm

In 1970, Needleman and Wunsch proposed an alignment method, called Needleman-Wunsch algorithm [16] that can be obtained computationally by applying a straightforward *Dynamic Programming (DP)* [38] algorithm to find an optimal global alignment

of two sequences, i.e. the query sequence (N_q) of length M and the subject or database sequence (N_s) of length N. The algorithm is based on finding the elements of a matrix H, according to the following equation,

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \tag{2.1}$$

where $S_{i,j}$ is the similarity score of comparing N_q to N_s and d is the penalty for a mismatch. The matrix is initialized with $H_{0,j} = H_{i,0} = 0$, for all i, j.

In a simple scoring scheme, cells representing matches are scored 1 and cells representing mismatches are scored 0, assuming the penalty for a mismatch to be zero; the 2D array is thus populated with these values. An operation of successive summation of cells then commences. This process examines each cell in the matrix, the maximum score along any path leading to the cell is added to its present contents, and the summation continues. When this process has been completed, the maximum-match pathway is constructed.

The algorithm can be implemented using the following pseudo code.

Initialization:

```
H(0,j) = 0
H(i,0)= 0
```

Matrix Fill:

```
for each i,j = 1 to M,N
    {
        H(i,j) = max(H(i-1,j-1) + S(i,j),
                     H(i-1,j) - d,
                     H(i,j-1) - d)
    }
```

Traceback:

```
H(opt) = max(H(i,j))
traceback(H(opt))
```

The time complexity of the initialization step is simply $O(M + N)$. The next step is filling in the matrix with all the scores, $H_{i,j}$. For each cell of the matrix, three neighboring cells (left, above, and diagonally upper-left) must be compared. Three separate scores are calculated based on all three neighbors, and the maximum score is assigned to the cell, which is a constant time operation [39]. Thus, to fill the entire matrix, the time complexity is the number of entries, or $O(MN)$. Finally, we can traverse a maximum of N rows and M columns during the traceback, and thus the complexity of this is $O(M + N)$. Thus, the overall time complexity of this algorithm is $O(M + N) + O(MN) + O(M + N) = O(MN)$. Since this algorithm fills a single matrix of size MN, the total space complexity is $O(MN)$. Examples of N-W algorithm are given in Appendix C.

2.3 Local methods

In contrast to global methods, local methods attempt to identify short stretches of similarity between two sequences i.e. N_q and N_s. These include exact method like Smith-Waterman and heuristics based approximate methods like FASTA and BLAST, as explained in the following subsections.

2.3.1 Smith-Waterman algorithm

In 1981, Smith and Waterman described a method, commonly known as the Smith-Waterman algorithm [17], for finding common regions of local similarity. N-W algorithm described in the previous section works well for sequences that show similarity across most of their lengths. Consider, however, two sequences that are only distantly related to each other. They will, even so, exhibit small regions of local similarity, although no satisfactory overall alignment can be found. S-W algorithm solves this problem and is used for finding these common regions of similarity. Like the technique of N-W, this is a matrix-based approach, and trace back is used to reconstruct the gapped alignments. S-W method has been used as the basis for many subsequent algorithms, and is often quoted as a benchmark when comparing different alignment techniques.

When obtaining the local S-W alignment, $H_{i,j}$ for N-W algorithm is modified as follows:

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases} \tag{2.2}$$

The algorithm can be implemented using the following pseudo code.

Initialization:

```
H(0,j) = 0
H(i,0) = 0
```

Matrix Fill:

```
for each i,j = 1 to M,N
    {
        H(i,j) = max(0,
                     H(i-1,j-1) + S(i,j),
                     H(i-1,j) - d,
                     H(i,j-1) - d)
    }
```

Traceback:

```
H(opt) = max(H(i,j))
traceback(H(opt))
```

The S-W with affine gap penalties [40] is given by Equation 2.3, where $S_{i,j}$ is the similarity score and α, β are the gap opening and extension penalties, respectively. Further, $H_{0,0} = D_{0,0} = E_{0,0} = H_{i,0} = D_{i,0} = E_{i,0} = H_{0,j} = D_{0,j} = E_{0,j} = 0$, for $1 \leq i \leq M$ and $1 \leq j \leq N$, where M and N are the lengths of the sequences to be aligned.

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ D_{i,j} \\ E_{i,j} \end{cases} \tag{2.3}$$

$$\text{where} \quad D_{i,j} = \max \begin{cases} H_{i-1,j} - \alpha \\ D_{i-1,j} - \beta \end{cases} \quad \text{and} \quad E_{i,j} = \max \begin{cases} H_{i,j-1} - \alpha \\ E_{i,j-1} - \beta \end{cases}$$

N-W and S-W algorithms share many similarities. Both algorithms consist of three steps: initialization, matrix fill, and traceback. A matrix is constructed with one sequence lined up against the rows of a matrix, and another against the columns, with the first row and column initialized with a predefined value (usually zero) i.e. if the sequences are of length M and N respectively, then the matrix for the alignment algorithm will have $(M + 1) \times (N + 1)$ dimensions. The matrix fill stage scores each cell in the matrix. This score is based on whether the two intersecting elements of each sequence are a match, and also on the score of the cell's neighbors to the left, above, and diagonally upper left. Three separate scores are calculated based on all three neighbors, and the maximum of these three scores (or a zero if a negative value would result) is assigned to the cell. This is done for each cell in the matrix. Even though the computation for each cell usually only consists of additions, subtractions, and comparisons of integers, the algorithm would nevertheless perform very poorly if the lengths of the query sequences become large. The initialization and matrix fill steps for N-W and S-W algorithms are the same, so their time complexity would be $O(M + N)$ and $O(MN)$ respectively. The difference lies in the traceback step. With N-W, the traceback starts at the last cell in the matrix and traces the maximal score path back to the first cell. Whereas with the S-W, the traceback starts at the cell with the highest score in the matrix and ends at a cell when the similarity score drops below a certain predefined threshold. For doing this, the algorithm requires to find the maximum cell which is done by traversing the entire matrix, making the time complexity for the traceback $O(MN)$. It is also possible to keep track of the cell with the maximum score, during the matrix filling segment of the algorithm, although this will not change the overall complexity. Thus, the total time complexity of the S-W algorithm is $O(M + N) + O(MN) + O(MN) = O(MN)$. The space complexity is also the same as that of the N-W algorithm. This is due to the fact that the same matrix is used and the same amount of space is needed for the traceback. Thus, there is no definite space or time advantage of one algorithm over the other. However, the S-W

algorithm tends to model protein homology better, as it ignores misalignments at the ends of the proteins which are often not highly conserved.

In order to reduce the $O(MN)$ complexity of the matrix fill stage, multiple entries of the H matrix can be calculated in parallel. This is however complicated by data dependencies, whereby each $H_{i,j}$ entry depends on the values of three neighboring entries $H_{i,j-1}$, $H_{i-1,j}$ and $H_{i-1,j-1}$, with each of those entries in turn depending on the values of three neighboring entries, which effectively means that this dependency extends to every other entry in the region $H_{x,y} : x \leq i,\ y \leq j$. This implies that it is possible to simultaneously compute all the elements in each anti-diagonal, since they fall outside each others data dependency regions. Figure 2.2 shows a sample H matrix for two sequences, with the bounding boxes indicating the elements that can be computed in parallel. The bottom-right cell is highlighted to show that its data dependency region is the entire remaining matrix. The dark diagonal arrow indicates the direction in which the computation progresses. At least 9 cycles are required for this computation, as there are 9 bounding boxes representing 9 anti-diagonals and a maximum of 5 cells may be computed in parallel.

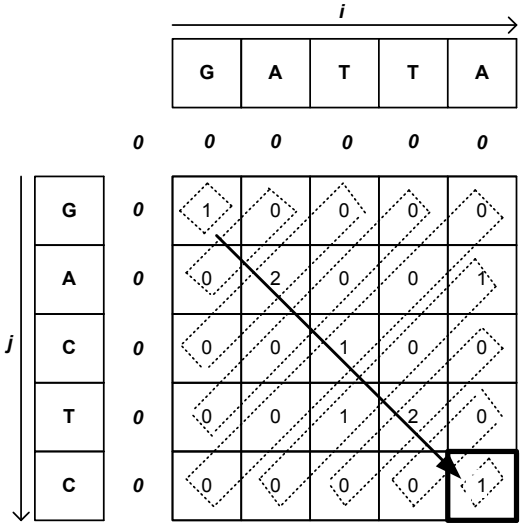


Figure 2.2: Sample H matrix, where the dotted rectangles show the elements that can be computed in parallel

The degree of parallelism is constrained to the number of elements in the anti-diagonal and the maximum number of elements that can be computed in parallel are equal to the number of elements in the longest anti-diagonal (l_d), where,

$$l_d = \min(M, N) \tag{2.4}$$

Theoretically, the lower bound to the number of steps required to calculate the entries of the H matrix in a parallel implementation of the S-W algorithm is equal to the number of anti-diagonals required to reach the bottom-right element, i.e. $M+N-1$ [41].

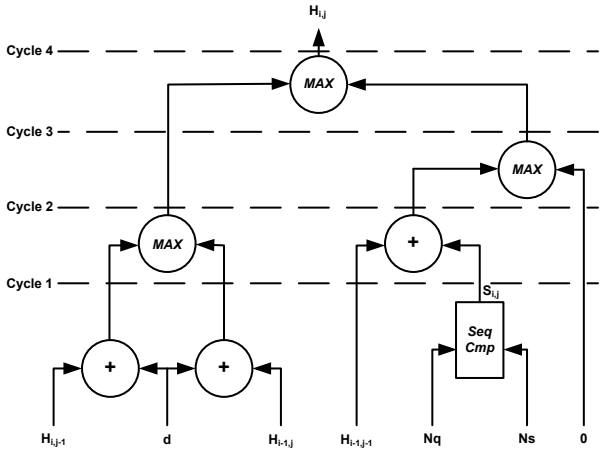


Figure 2.3: Logic to compute cells in the H matrix, where + is an adder, *MAX* is a max operator and *SeqCmp* is the sequence comparator that generates match/mismatch scores

Figure 2.3 shows the logic to compute an element of the H matrix. The logic contains three adders, a sequence comparator circuit (*SeqCmp*) and three max operators *(MAX)*. The sequence comparator compares the corresponding characters of two input sequences and outputs a match/mismatch score, depending on whether the two characters are equal or not. Each max operator finds the maximum of its two inputs. The time to compute an element is 4 cycles, assuming that the time for each cycle is equal to the latency of one add or compare operation.

For more understanding of the S-W algorithm, refer to Appendix D, where S-W examples are given in addition to its flow chart description.

2.3.2 FASTA algorithm

Fast Alignment Search Tools - All (FASTA) was developed in 1985 by Lipman and Pearson [19]. Unlike N-W and S-W algorithms, FASTA approximates the optimal alignment by searching and matching k-tuples, or subsequences of length k. The algorithm assumes that related proteins will have regions of identity, and by searching with k-tuples, FASTA algorithm allows small regions of local identity to be found quickly. For proteins, these k-tuples tend to be of length two. FASTA search process can be summarized into the following three steps.

1. In the first step of this search, the comparison can be viewed as a set of many dot plots with the query sequence on the vertical axis and each sequence in the database on the horizontal axis of its particular plot. Set a word size, where a word is a short sequence of nucleotides. A word of size 2 could be, for instance, *gg*. Place a dot wherever words of this size match. For example, if the query sequence is *ggctttcgg* and the database sequence is *aacggcttacg*, then the corresponding plot would be as shown in Figure 2.4. The two diagonal series of dots

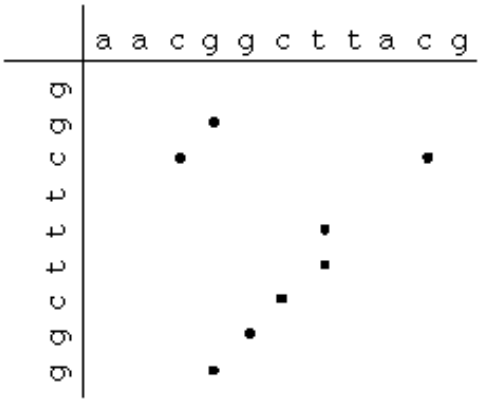


Figure 2.4: Sample plot for FASTA

in the figure indicate that the two sequences are identical over these diagonals. The purpose of this first step is to find the longest diagonals, or highest scoring regions.

2. In the second step, re-score the 10 best diagonals using a scoring matrix that allows and takes into account conservative replacements and ambiguity symbols shorter than the size of a word. This analysis finds high scoring subregions within the diagonals, called "initial regions"

3. In step three, take the initial regions whose scores are above a predetermined threshold and check to see if they can be joined together. Impose a penalty on joined regions so that they have lower scores than continuous runs. Finally, use a variant of the last part of S-W algorithm to align the sequence and calculate the optimal score. If the optimal score is above a certain threshold, place the sequence in the match list.

Like N-W and S-W algorithms, the computational (i.e. time) and space complexities of FASTA are both $O(MN)$.

2.3.3 BLAST: Basic Local Alignment Search Tool

Basic Local Alignment Search Tool (BLAST) is a heuristic method [20] to find the highest scoring locally optimal alignments between a query sequence and a database. It is similar to FASTA, however, the basis of BLAST algorithm is the use of words and *High-scoring Segment Pairs (HSPs)* instead of k-tuples. The central idea of BLAST algorithm is to pay attention only to the segment pairs that contain a word pair of length w with a score of at least T, i.e. the threshold value [42]. BLAST has three phases, described as follows.

1. **Phase 1:** Compile a list of word pairs (typically $w = 3$ for proteins) above threshold T (say 11)

   Example: For broad bean leghemoglobin (a protein that provides oxygen to bacteroids) LG**AHA**EK

   A list of words from the given sequence: LGA GAH **AHA** HAE AEK SHA AHG ...

Neighborhood word hits (AHA) above threshold T	AHA 4,8,4 16 SHA 1,8,4 13 ..... .,.,. ... AHG 4,8,0 12 ..... .,.,. ...
(T = 11)	AHI 4,8,-1 11
Neighborhood word hits below threshold	NHA -2,8,4 10

2. **Phase 2:** Scan the database for entries that match the compiled list.

3. **Phase 3:** When you manage to find a hit, extend it in either direction. Use a scoring matrix to keep track of the score. Stop when the score drops below some cutoff (default $X = 15$)

AGVVDSPKLG**AHA**EKVFG	65	LEGHEMOGLOBIN (query)
GAVMGNPKVK**AHG**KKVLH	67	BETA GLOBIN (hit)
- extend **Hit** extend	-	

In the original (1990) implementation of BLAST [20], all hits were extended in either direction. The extending step was very time consuming, as it was taking almost 90 percent of the execution time. In a 1997 refinement of BLAST [43], two independent (non-overlapping) hits are required. Extending takes place only when the two hits are within a distance A (default=40) of one another on the same diagonal. This significantly improves the performance.

The computational complexity of BLAST can be calculated and like FASTA, it also comes out to be $O(MN)$. However, using the elimination of HSPs and words, BLAST significantly lowers the number of segments that needs to be extended. This makes BLAST run faster than all the previous algorithms.

For calculating the space complexity of BLAST, we must first take into consideration the hash table, where the words and HSPs are stored. The table contains 20^w rows, one for every possible word of length w. The rows contain the locations for each of the words, and the total number of positions is of the order N. Thus, there should be of the order N seeds, which can each lead to a local alignment of a maximum of length M. The total space complexity is, $O(20^w) + O(N) + O(MN) = O(20^w + MN)$. Thus, the space complexity is slightly higher than other algorithms, however the actual space used may not be significantly larger than the dynamic programming algorithms. This is because many of the local alignments will be discarded as they do not meet the threshold.

BLAST is significantly faster than the older, slower algorithms, yet, it does not always give the optimal alignment. It is possible for BLAST to miss segments of similarity smaller than the word size, and ungapped BLAST often produces alignments which are not biologically relevant. Gapped BLAST can also produce suboptimal alignments, because when it performs the dynamic programming at the end, the best alignment may lie outside the predefined threshold. Thus, it is clear that a trade off exists between the sensitivity of an algorithm and the speed at which it runs. Some online tools are available to use BLAST, such as [44,45].

2.4 Mutiple alignment methods

The algorithms discussed so far work for pairwise alignment, but as mentioned in the previous chapter, it might be of interest in some cases to consider the similarities between a group of sequences. Multiple sequence alignment methods are introduced to handle such cases. Multiple sequence alignment is an extension of pairwise alignment and it tries to align more than two sequences in a given query set simultaneously. These include HMMER and ClustalW, as elaborated in the following subsections.

2.4.1 HMMER

Among other approaches, *profile based Hidden Markov Models (profile HMMs)* [46] have been recently used by biologists to predict the structure and function of a protein directly from its representation as an amino acid sequence [47]. Profile HMMs are used to do sensitive database searching using statistical descriptions of a sequence family's consensus [48]. The approach consists of building and providing probabilistic models of protein sequences that share similar structures or functions. As of today, there are several software implementations of this model, the HMMER software package being one of the most widely used. HMMER is an implementation of profile HMMs software for protein sequence analysis. When doing database searches

like BLAST and Smith-Waterman searches, they are based on pairwise alignments between the query sequences and the sequences in the database. HMMER uses profile HMMs instead of pairwise alignments. This means that a hit is found based on the information of many sequences instead of just one sequence. Profile HMMs can be used in different contexts. For example, when trying to discover the biological function of a given amino acid sequence, the user matches this sequence against a database of profiles such as the Pfam database [49], so as to find the best corresponding profile. The Pfam database is a large collection of multiple sequence alignments that covers approximately 8000 protein domains and protein families [50]. Another use is gene annotation, where the user wants to match a whole sequence database against one or several profile HMMs databases. However, given the computational complexity of the algorithm, such intensive comparisons lead to prohibitive execution time in the order of days or weeks [21]. Given N sequences of average length M, the time complexity of HMMER is $O(MN^2)$, whereas the space complexity is $O(MN)$ [51].

2.4.2 ClustalW

ClustalW [52] is a heuristic multiple sequence alignment tool based on the Clustal algorithm. The Clustal algorithm uses a progressive alignment method [53]. Typically, progressive alignment consists of three stages, as shown in Figure 2.5 [54]. These three stages are described as follows [55]:

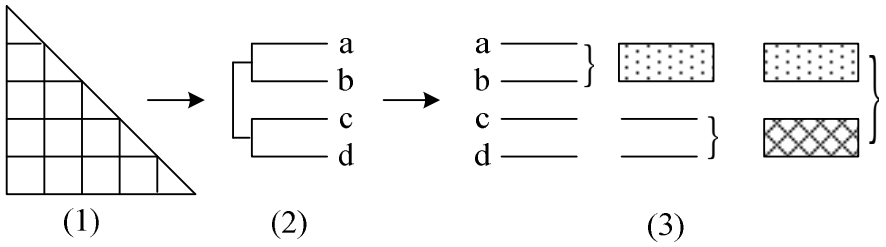


Figure 2.5: Three stages of progressive alignment: (1) similarity matrix, (2) guided tree, (3) profile-profile progressive alignment

1. **Pairwise distance computation:** Compare all pairs of sequences to obtain a similarity matrix.
2. **Guided tree generation:** Based on the similarity matrix, make a guided tree relating all the sequences.
3. **Profile-profile progressive alignment along the guided tree:** Perform progressive alignment where the order of the alignments is determined by the guided tree.

Stage 1 computes a similarity matrix comprised of the distance value between each pair of sequences using pairwise alignment. The scores of each pairwise alignment are stored in a triangular matrix as distances from 0 to 1. Stage 2 generates a guided tree from the distance matrix using distance-based phylogenetic tree reconstruction methods, where similarity tree is constructed in two steps. First an unrooted tree is constructed from the distance matrix using the Neighbor-Joining method of creating phylogenetic trees. Then the tree is transformed to a rooted version. For the Clustal algorithm all sequences get the same total weight, whereas for the newer ClustalW version a sequence's weight depends on its distance from the root and what branches it has in common with other sequences. Stage 3 performs progressive alignment of various profiles to form the final MSA along the guided tree. The progressive alignment is performed from the ends of the tree branches back to the root, using dynamic programming algorithms with some side notes. When a gap is introduced, it can not be removed at a later stage. Furthermore, if a gap is introduced within an existing gap, the full gap creation penalty is deducted. ClustalW expands on this by varying the scoring matrices used, depending on the distances between the sequences being compared and, in turn, varying the gap creation and expansion penalties depending on the current scoring matrix, sequence similarity, sequence lengths and the current position of the alignment within the sequences. Given N sequences of average length M, the time complexity of ClustalW is $O(M^2N^2)$, whereas the space complexity is $O(MN)$.

2.5 Comparison of sequence alignment methods

This section presents a comparison of the sequence alignment methods discussed in the previous sections. The comparison is based on their temporal and spatial complexities and parameters like alignment type and search procedure, as shown in Table 2.2.

Table 2.2: Comparison of various sequence alignment methods

Method	Type	Search	Time complexity	Space complexity
Dot plot	Global	Basic	$O(MN)$	$O(MN)$
N-W	Global	DP	$O(MN)$	$O(MN)$
S-W	Local	DP	$O(MN)$	$O(MN)$
FASTA	Local	Heuristic	$O(MN)$	$O(MN)$
BLAST	Local	Heuristic	$O(MN)$	$O(20^w + MN)$
HMMER	Multiple	Heuristic	$O(MN^2)$	$O(MN)$
ClustalW	Multiple	Heuristic	$O(M^2N^2)$	$O(MN)$

It is interesting to note that all the global and local sequence alignment methods essentially have the same computational complexity of $O(MN)$, yet despite this, each of the algorithms has very different running times, with BLAST being the fastest and

the dynamic programming algorithms being the slowest. In case of multiple sequence alignment methods, ClustalW has the worst time complexity of $O(M^2N^2)$, whereas HMMER has a time complexity of $O(MN^2)$. The space complexities of all the alignment methods are also essentially identical, around $O(MN)$ space, except BLAST, the space complexity of which is $O(20^w + MN)$. In the exact methods, dot plot uses a basic search method, whereas N-W and S-W use dynamic programming. On the other hand, all the approximate methods are heuristic based. It is also worthy to note that FASTA and BLAST have to make sacrifices on sensitivity to be able to achieve higher speeds. Thus, a trade off exists between speed and sensitivity and we must come to a compromise to be able to efficiently align sequences in a biologically relevant manner in a reasonable amount of time.

Being the most sensitive and optimal but computationally expensive sequence alignment method and the inherent parallelism it offers, S-W algorithm is considered as the top candidate for hardware acceleration. Furthermore, it is also used as an essential kernel in heuristics-based sequence alignment applications like FASTA, making its acceleration helpful in speeding up such applications as well. When run on a conventional PC, S-W algorithm spends most of the time on calculating elements of the H matrix, using the same matrix fill step repeatedly. This makes it well suited for parallelization on other hardware platforms like FPGAs and GPUs. Chapter 3 discusses hardware acceleration of S-W based sequence alignment applications in detail.

2.6 Summary

This chapter discussed the available sequence alignment methods, based on a classification, presented in the beginning of the chapter. The classification is followed by a discussion about each method. Following are the main topics presented in the chapter.

- Classification of various available sequence alignment methods, like global, local and multiple alignment methods.
- Discussion about global methods like dot plot and N-W algorithms.
- Detailed discussion about local methods like S-W, FASTA and BLAST.
- Discussion about multiple alignment methods including HMMER and ClustalW.
- Comparison of various sequence alignment methods presented in the previous sections, based on various parameters like complexities, alignment type and the search procedure used.

Chapter 3

Hardware Acceleration

This chapter discusses hardware acceleration of S-W based sequence alignment applications. It presents a number of hardware-accelerated design alternatives and compares them with existing implementations. Furthermore, it provides a classification of the available acceleration methods and proposes an accurate method for acceleration evaluation. It continues with an insight into systolic arrays and presents their application in sequence alignment. The chapter is organized as follows.

Section 3.1 presents a classification of the available acceleration methods and a subsequent discussion about the related work. Section 3.2 introduces an accurate acceleration evaluation approach. Sections 3.3 and 3.4 provide rectangular and linear systolic array based FPGA implementations for sequence alignment respectively. Section 3.5 summarizes the chapter.

3.1 Classification of acceleration methods

In computing, *hardware acceleration* is the use of specialized hardware to perform some function faster than is possible in software running on a general purpose CPU. The hardware that performs the acceleration, when in a separate unit from the CPU, is referred to as a *hardware accelerator*.

Work has been done on accelerating S-W based sequence alignment methods by implementing them on various available hardware platforms. A classification of this work, based on the methods of implementation is shown in Figure 3.1 and reviewed in the following subsections.

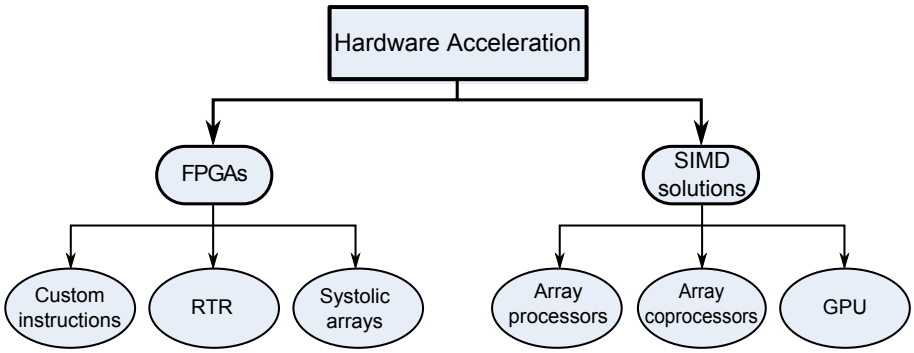


Figure 3.1: Hardware acceleration of sequence alignment methods

3.1.1 FPGAs

FPGAs are re-configurable data processing devices on which an algorithm is directly mapped to basic processing logic elements, e.g. NAND gates. To take advantage of using an FPGA, one has to implement massively-parallel algorithms on this re-configurable device. Thus they are well suited for certain classes of bioinformatics applications, such as sequence alignment.

FPGA custom instructions

In [56], the authors studied an improvement in the computational processing time of sequence alignment based on S-W algorithm using custom instructions on an FPGA board. This was done by first writing S-W algorithm in pure software and then replacing the most computationally intensive portion with an FPGA custom instruction. Finally, they compared the processing runtime between the software-only and hardware-accelerated versions to calculate the percentage of runtime improvement. The results showed that the hardware-accelerated algorithm improved the processing runtime by an average of 287%. Thus using FPGA custom instructions is a promising direction for further research in sequence alignment.

Run-time reconfiguration

One way to further exploit the reconfigurable resources of FPGAs and increase their functional density is to reconfigure them during system operation. This process is referred to as *Run-time reconfiguration (RTR)*. RTR is an approach to system implementation that divides an application or algorithm into time-exclusive operations that are implemented as separate configurations. In [57], an approach to realize high speed sequence alignment using run-time reconfiguration is proposed. With this approach, it is demonstrated that high performance can be achieved using off-the-shelf FPGA boards. The performance is almost comparable with dedicated hardware systems.

The time for comparing a query sequence of 2048 elements with a database sequence of 64 million elements using S-W algorithm is about 34 sec, which is about 330 times faster than a desktop computer with a 1GHz Pentium-III.

In [58], the performance of S-W based sequence alignment has been increased substantially by using run-time reconfiguration. The percentage of time spent on calculating the elements of $H_{i,j}$ matrix was cut by nearly a third and the absolute time spent on the algorithm was dropped from 6,461 seconds to a little over 100 seconds, approximately 64 times faster than an equivalent software-only implementation.

Systolic arrays

Systolic array is an arrangement of processors in an array, where data flows synchronously across the array between neighbors, usually with different data flowing in different directions [59], [60]. Each processor at each step takes in data from one or more neighbors (e.g. North and West), processes it and, in the next step, outputs results in the opposite direction (South and East). Systolic arrays can be implemented in rectangular or 2D and linear or 1D fashion. Figure 3.2 gives a pictorial view of both implementation types.

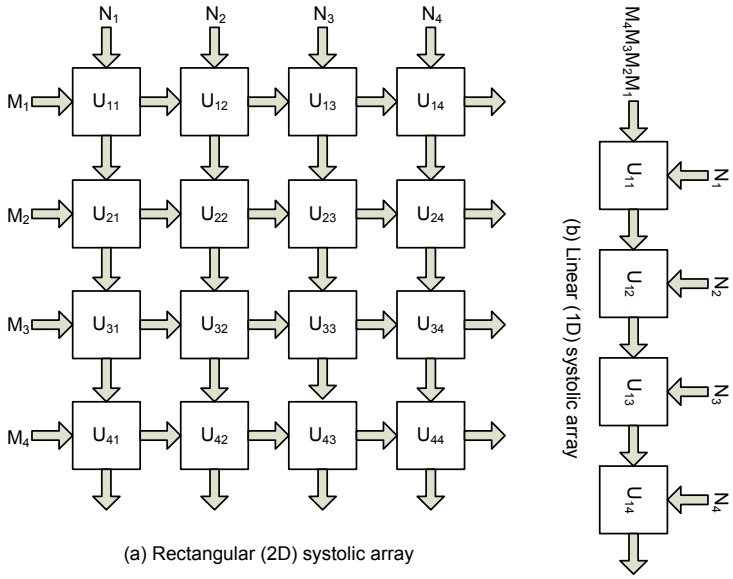


Figure 3.2: Pictorial view of systolic array architectures

In these configurations, there are two vector array inputs, M and N. The processing cells have a value, U_{ij}, that is usually a result due to a defined algorithm within the cells. Systolic array based architectures are extremely fast, easily scalable and can

do many tasks that traditional architectures can not attain. They best suit compute-intensive applications like biological sequence alignment. The disadvantage is that being highly specialized processors type, they are difficult to implement and build.

In [61], a concept to accelerate S-W algorithm on the bases of linear systolic array is demonstrated. The reason for choosing this architecture is outlined by demonstrating the efficiency and simplicity in combination with the algorithm. Nevertheless, there are two key methodologies to speedup this massively parallel system. By turning the processing from bit-parallel to bit-serial, the actual improvement is enabled. This change is performance neutral, but in combination with the drafted early maximum detection, a considerable speedup is possible. Another effect of this improvement is a data dependant execution time of the processing elements. Here, the second acceleration prevents idle times to exploit the hardware and speeds up the computation. This can be accomplished by a globally asynchronous timing representing a self-timed linear systolic array. The authors have provided no performance estimation due to the initial stage of their work, that's why it can't be compared with other related work.

3.1.2 SIMD solutions

Single-Instruction stream, Multiple-Data stream (SIMD) is a type of multiprocessor architecture in which multiple sets of operands may be fetched to multiple processing units and may be operated upon simultaneously within a single instruction cycle. Following is a discussion of several SIMD based approaches for sequence alignment.

Array processors

In [62], an implementation of S-W algorithm for sequence alignment is described on a general purpose fine-grained architecture, the *Micro Grained Array Processor (MGAP)*. The authors show that their implementation is about 5 times faster than the rapid implementation of a genetic sequence comparator using field programmable logic arrays [63]. Showing thereby that massively parallel processor arrays like the MGAP possess the capability to solve computationally intensive problems in molecular biology efficiently and inexpensively. The algorithm given in [62] takes $M + N$ steps to align two sequences. Therefore, if there are K sequences to be aligned, the entire computation would require only $M + N + K$ steps. The sequential algorithm would have taken $O(MNK)$ steps to compute K alignments.

Array coprocessors

Kestrel parallel processor is a single-board coprocessor with a 512-element linear array of 8-bit, SIMD processing elements [64]. The system was designed to analyze databases containing billions of characters from DNA, RNA, or proteins. As a case study, the authors implemented S-W algorithm on kestrel parallel processor for different query sizes and compared its performance with an implementation on a 500

MHz Ultra SPARC-II. The results of their implementations are compared with others in Table 3.1.

Graphics Processing Units (GPUs)

GPUs are single-chip processors, used primarily for computing 3D functions. This includes things such as lighting effects, object transformations, and 3D motion. GPU is a good match for bioinformatics sequence alignment applications, as it is an inexpensive and high-performance SIMD architecture.

In [65], it has been demonstrated that the streaming architecture of GPUs can be efficiently used for biological sequence database scanning. To derive an efficient mapping onto this type of architecture, the authors reformulated S-W algorithm in terms of computer graphics primitives. They claim that this is the first reported implementation of S-W algorithm on graphics hardware and its evaluation on a high-end graphics card shows a speedup of almost sixteen compared to a Pentium IV 3.0 GHz.

Table 3.1: Comparison of the work reviewed in Section 3.1

Reference	Section	Platform	Compared with	Speedup	Query size
[56]	3.1.1	FPGA	Software-only	287×	—
[57]	3.1.1	FPGA	1 GHz P-III	330×	—
[58]	3.1.1	FPGA	Software-only	64×	—
[62]	3.1.2	Array processors	SPLASH	5×	—
[64]	3.1.2	Array coprocessors	Ultra SPARC-II	17×	100
[64]	3.1.2	Array coprocessors	Ultra SPARC-II	49×	250
[64]	3.1.2	Array coprocessors	Ultra SPARC-II	99×	500
[65]	3.1.2	GPU	3.0 GHz P-IV	16×	—

Table 3.1 gives a comparison of the work reviewed in the section. It identifies that no standard comparison approach has been adapted, which makes it difficult to compare different approaches published in the literature [23]. That is why, we can only look into each implementation on individual basis to see how much improvement is achieved in comparison with the provided reference. In the following section, an accurate speedup measurement method is proposed that is independent of any specific implementation. This is done by comparing both the software-only and hardware-accelerated versions of S-W algorithm on the same platform in order to achieve an accurate profiling and acceleration evaluation. In [66], a similar approach is adapted, but the proposed implementation performs 1.27 times better.

3.2 Accurate acceleration evaluation approach

Implementing both software-only as well as hardware-accelerated versions of S-W algorithm on the same platform leads to an accurate acceleration evaluation . We have used the MOLEN platform for this purpose, since it contains both a general purpose processor in addition to a reconfigurable hardware module. The following subsections present a background about the MOLEN platform and S-W implementation on the platform.

3.2.1 MOLEN platform

Figure 3.3 shows a block diagram representation for MOLEN platform [67]. The first block on the left indicates that either the software-only or hardware-accelerated version of the algorithm arrives as input to the arbiter. For every function in the algorithm, the arbiter decides whether to send it to either the core or reconfigurable processor. The arbiter does this by using specialized instructions for calling the hardware.

The core processor is the IBM Power PC, a built-in component in Virtex-II Pro FPGA and is used for implementing the software portion of the application. There are two such components in Virtex-II Pro FPGA.

The reconfigurable processor is used for implementing the hardware portion of the application and has two parts. The main part is the reconfigurable microcode unit, responsible for the entire operation of the reconfigurable processor and an application dependant *Custom Computing Unit (CCU)*. The CCU is embedded into the reconfigurable processor using the interface provided with the MOLEN platform. The reconfigurable processor utilizes the microcode unit and the CCU to improve performance of various applications. The details of the data interface between the core processor and the reconfigurable processor are given in [67].

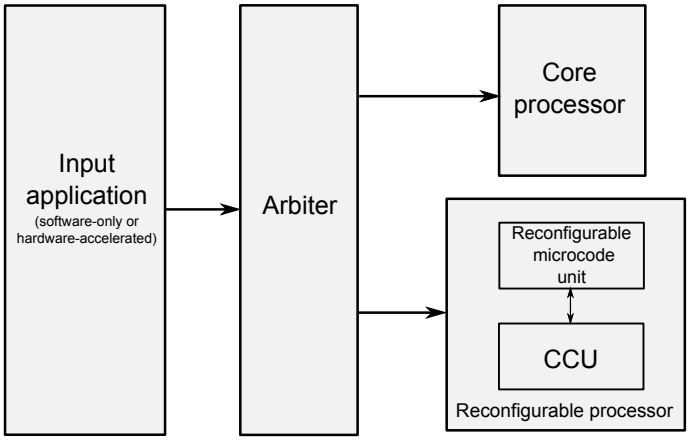


Figure 3.3: Block diagram description of MOLEN platform

The methodology for implementation on MOLEN platform comprises of the following four steps.

1. Identifying the desired function in software (code-profile).

2. Designing a CCU for the function identified in the code-profile.

3. Replacing the function identified in the code-profile by the designed CCU.

4. Comparing the cycles consumed by the software-only and hardware-accelerated versions of the identified function to measure the relative speedup.

Figure 3.4, shows a block diagram representation of MOLEN implementation approach, where the block in gray represents the desired function, identified in the code-profile. This is the function for which a CCU is designed [68].

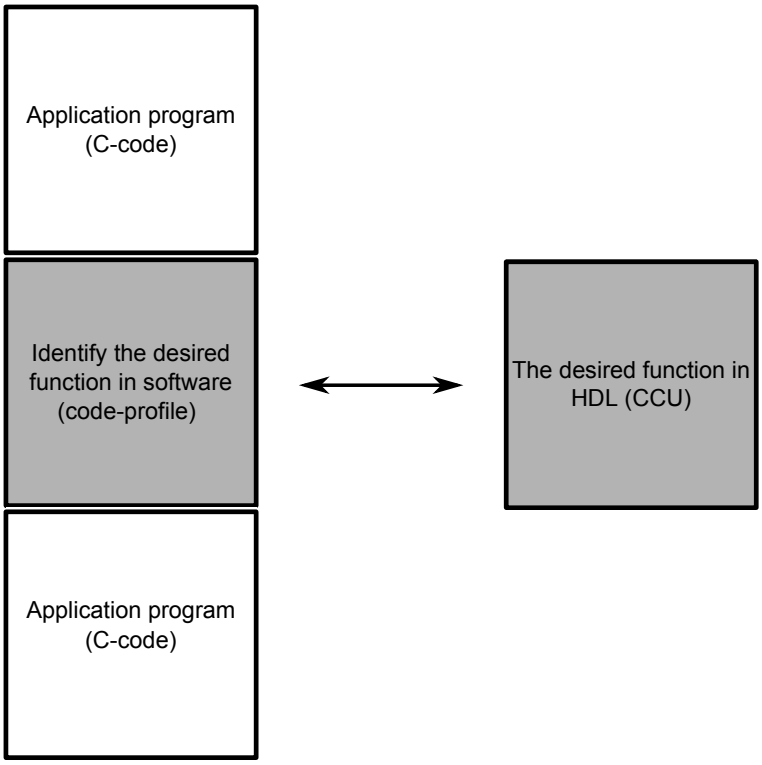


Figure 3.4: Block diagram representation of MOLEN implementation approach

3.2.2 S-W implementation on MOLEN

Following is a discussion of an implementation of the S-W algorithm on MOLEN platform. The discussion starts with details about profiling, followed by CCU design for the function, identified in the code-profile and concludes with an accurate evaluation of the speedup achieved.

Profiling

Figure 3.5 gives block diagram representation of a software-only implementation of S-W algorithm [24], where,

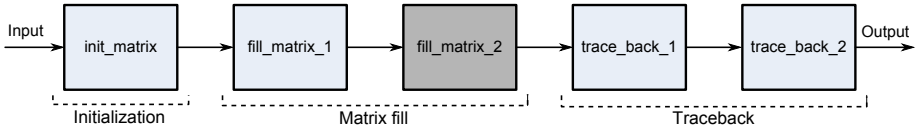


Figure 3.5: Functional description of a software implementation of S-W algorithm

- The *init_matrix* is a function used for initializing the scoring matrix.
- The *fill_matrix_1* performs two functions i.e. filling the matrix and at the same time keeping track of the maximum score in the matrix.
- The fill_matrix_2 function finds the corresponding maximum candidate for each cell in the matrix, using Equation 2.2.
- The *trace_back_1* function performs the traceback.
- The *trace_back_2* function keeps track of the direction of the traceback.

The *C-code* for this software-only implementation is compiled using *MOLEN Power PC compiler* [69]. The cycles consumed by each function in the code are evaluated, using the Power PC timer instructions. The Power PC has a clock frequency of 100 MHz, so the time period for one cycle is $\frac{1}{100}\ \mu s = 0.01\ \mu s$. Thus the time consumed by each function is equivalent to the number of cycles consumed multiplied by the time period for one cycle. The overall time consumed is the summation of time consumed by all functions, which is 172 μs whereas the % time consumed is the ratio of the time consumed by a function to the overall time consumed. This is an alternative way of profiling and is more accurate than other methods, such as using the *GNU profiler (gprof)*, used in [68]. The reason for the inaccuracy of the profiling approach adopted in [68] is that it does not account for the overhead in the computation time. This overhead is incurred by some inaccuracies in the gprof tool itself, in addition to the computational overhead in the operating system.

Table 3.2: Profiling results

Function name	No. of calls	Clock cycles	Time (μs)	% Time
init_matrix	1	753	7.53	4.33
fill_matrix_1	1	688	6.88	4.00
fill_matrix_2	**48**	**13392**	**133.92**	**78.00**
trace_back_1	1	102	1.02	0.55
trace_back_2	5	2265	22.65	13.12

Table 3.2 gives profiling results of a software-only version of S-W algorithm on the core processor of MOLEN platform i.e. IBM Power PC. It provides the function names, number of times that each function is called, the number of clock cycles consumed by each function, the amount of time consumed by each function in micro seconds and the % time consumed by each function. In Table 3.2, the fill_matrix_2 is highlighted as the most time consuming function and is the right candidate to be designed in hardware as a CCU. The fill_matrix_2 function is based on Equation 2.2, which is given as follows.

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + S_{i,j} \\ H_{i-1,j} - d \\ H_{i,j-1} - d \end{cases}$$

The table shows that the fill_matrix_2 function consumes 13392 cycles for 48 calls, so the cycles consumed for 1 call will be 13392/48 = 279. When run on Intel 3.2 GHz Pentium-IV processor, the time consumed by this function is 52.32 μs (as measured by gprof). Later in this section, a comparison is made with the Power PC.

CCU design

A hardware module, called CCU is designed in VHDL for the function of interest (fill_matrix_2), identified in the code-profile. Figure 3.6 shows the RTL schematic of the CCU.

The CCU is a synchronous comparator, which compares four 8-bit numbers and finds the maximum of them in two comparison levels. For this purpose, three similar 8-bit comparators are used, each having two 8-bit inputs *in1* and *in2*, a 1-bit reset input *rst_comp* and an 8-bit output *out_comp*. The inputs *a*, *b*, *c* and *d* in the figure, represent the four alternatives for the *max* operator in Equation 2.2. In the 1st level of the design, *a* is compared with *b* and *c* is compared with *d*, using two of the three comparators. Each comparator finds the maximum of the two numbers and provides the result to the third comparator in the second level. The third comparator finds the maximum of all four input candidates. To receive the final 8-bit output (*max_out*) at the rising edge of the clock, a D flip flop with a *clear (CLR)* input is used. The post place and route

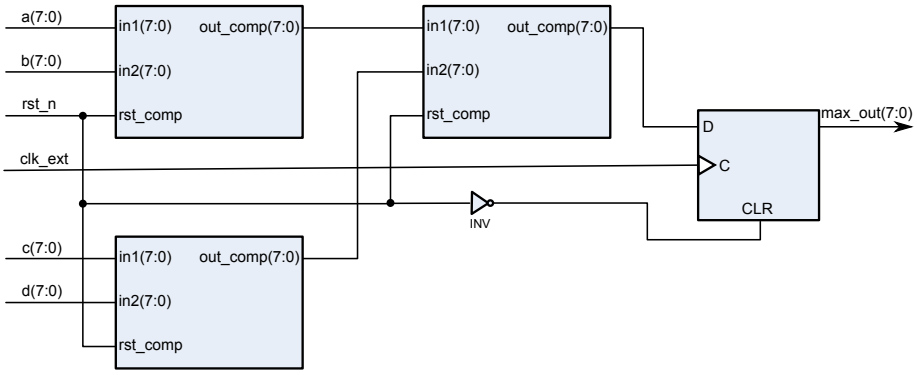


Figure 3.6: RTL schematic of the CCU for the function fill_matrix_2

simulation, as shown in Figure 3.7, exhibits that the time consumed by the CCU to compute the output (*max_out*) is 14.6 *ns* or 0.0146 μs, whereas the time consumed by its software equivalent was 52.32 μs. The figure shows that the simulation initializes the clock and reset signals (*clk_ext* and *rst_n*) with zero. The inputs *a*, *b*, *c* and *d* are also initialized with zeros, so that the output (*max_out*) stays zero during the 1st clock cycle. The clock cycle is set at 20 *ns* (i.e. 50 MHz frequency), whereas the inputs are changed after every five clock cycles. The gray blocks in Figure 3.7 highlight the time required to calculate the output (*max_out*). The time between any two gray blocks is the idle time where no computation is performed.

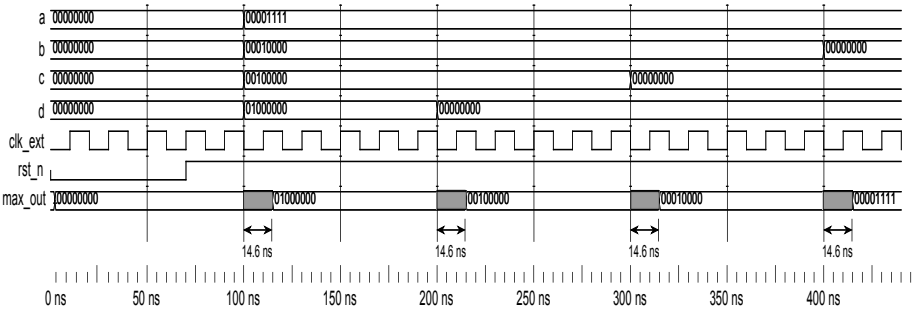


Figure 3.7: Post place and route simulation results

The speedup that a standalone CCU achieved over its software equivalent is given by the ratio of software time ($fill\_matrix\_2$ time) to hardware time (CCU time) and is calculated as follows.

$$\text{Speedup} = \frac{\text{software time}}{\text{hardware time}} = \frac{fill\_matrix\_2 \text{ time}}{\text{CCU time}} = \frac{52.32 \times 10^{-6}}{0.0146 \times 10^{-6}} = 3583$$

The device used for the implementation was Xilinx Virtex-II Pro (XC2VP30) FPGA with a speed grade of -7. It is worth mentioning that the actual speedup achieved is lower than the speedup calculated here, since gprof does not account for the overhead in the computation time. This overhead is incurred by some inaccuracies in the gprof tool itself, in addition to the computational overhead in the operating system. Secondly, the speedup shown is for the identified function only and does not represent the performance improvement for the entire application. To investigate the overall runtime improvement and overcome the indicated issues, the entire application is run on MOLEN platform. This reduces the achieved speedup, but increases the level of accuracy.

Accurate speedup evaluation

The designed CCU for the function of interest (fill_matrix_2) is modified in a way that it can be run on MOLEN platform using the provided interface. After modifying the CCU in the desired way, the definition of fill_matrix_2 function is annotated with #pragma call fpga fill_matrix_2 annotation in the *C-code*. The entire annotated *C-code* is compiled, using the MOLEN Power PC compiler. An executable file thus generated is downloaded locally. The CCU design is embedded into MOLEN using the Xilinx modular design flow. The generated bit stream is downloaded into the XUP Virtex-II Pro prototyping board by connecting a configuration cable to the prototyping board. Using the Power PC timer functions, the cycles consumed by fill_matrix_2 function in the annotated *C-code* are evaluated, which comes out to be 129 cycles. The comparison between the cycles consumed by the software-only and hardware-accelerated versions of fill_matrix_2 gives the relative speedup, given as follows.

$$\text{Speedup} = \frac{\text{Cycles in software}}{\text{Cycles with hardware acceleration}} = \frac{279}{129} = 2.16$$

This speedup is more accurate than the one presented in the previous subsection, as the software-only and hardware-accelerated versions are both implemented on the same platform. To ensure an accurate measurement of the speedup, all bottlenecks have been taken care of, such that only processing time is the limiting factor. Moreover, the approach is technology independent and can also be implemented on alternative available FPGAs, such as Virtex-IV and Virtex-V.

3.3 Rectangular (2D) systolic implementation

In this section, a basic cell design for the S-W based sequence alignment and the corresponding *rectangular (2D)* systolic array implementations are presented. It starts

with a description of the proposed cell design, followed by system design based on this cell.

3.3.1 Cell design

The performance of systolic array architectures mainly depends on, how simplified and efficient the corresponding cell design is. By efficient we mean both in terms of performance and area. Figure 3.8 shows a block diagram description of the cell design for computing $H_{i,j}$ values for the systolic array architecture, according to Equation 2.2. In Figure 3.8, *Comp1* is a comparator that compares the two input sequences and outputs the corresponding value of $S_{i,j}$, depending on the values of the match and mismatch scores, such that $S_{i,j}$ = match score, if $N_q = N_s$, otherwise $S_{i,j}$ = mismatch score. *Add1* is an adder that adds the diagonal element $H_{i-1,j-1}$ and the value of $S_{i,j}$. *Comp2* is a comparator that compares the output of the *Add1* with a constant value *0* and outputs the greater of the two numbers. *Add2* is an adder that adds the left element $H_{i-1,j}$ and $-d$, where d is the penalty for a mismatch. *Add3* is an adder that adds the upper element $H_{i,j-1}$ and $-d$. *Comp3* compares the outputs of *Add2* and *Add3* and outputs the greater of the two numbers. *Comp4* compares the outputs of *Comp2* and *Comp3* and outputs the greater of the two numbers. The output of *Comp4* is stored in a buffer, and is the corresponding $H_{i,j}$ value.

The block diagram shown in Figure 3.8 is implemented in *VHDL* and the post place and route simulations show that the asynchronous time consumed by such a cell is 10 *ns* on a Xilinx Virtex-II Pro FPGA platform.

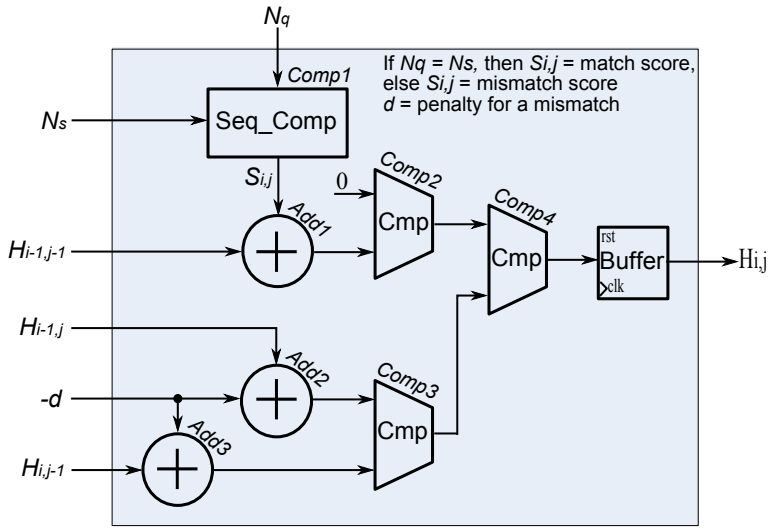


Figure 3.8: Cell design for rectangular systolic array implementation

3.3.2 System design

The cell design shown in Figure 3.8 can be used to build systolic array based systems of any size, depending on the availability of hardware resources. Figure 3.9 shows the description of such a system design, where a 4×4 systolic array architecture is chosen as an example.

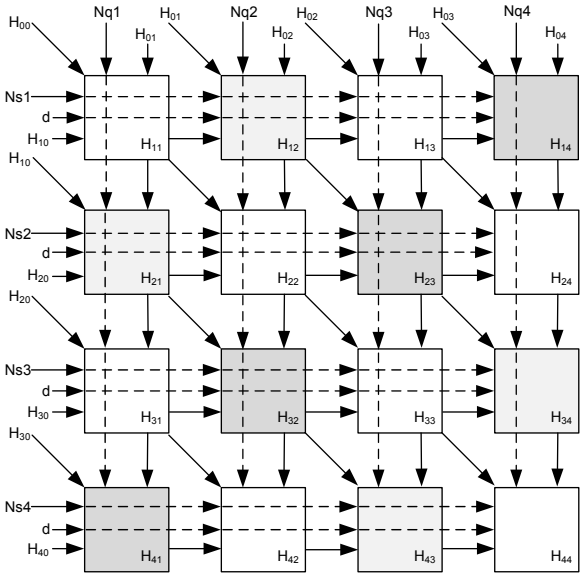


Figure 3.9: Block diagram description of a 4×4 systolic array

For computing the delay of the entire array, we run the asynchronous time simulation (post place and route simulation), which shows that the time consumed to fill a 4×4 array asynchronously = 26.4 ns. On the other hand, the time consumed to fill a 4×4 systolic array synchronously = $10 \times 7 = 70$ ns, showing thereby that the asynchronous approach is 2.6 times faster than the synchronous approach. This speedup is only significant in terms of computing the delay of the entire circuit, as the asynchronous approach only outputs the final $H_{i,j}$ value. For the intermediate $H_{i,j}$ values, we have to use the synchronous approach.

To evaluate the performance in terms of *Cell Updates Per Second (CUPS)*, we implemented the design on a Xilinx Virtex-II XC2V6000 FPGA, such that the available hardware resources were utilized to the maximum. In this way, we were able to fit a maximum of 1778 cells on the FPGA, where each cell utilized 19 slices. The clock frequency used for our implementation was 45 MHz and the performance thus achieved was,

Performance = $1778 \times 45 = 80$ GigaCUPS

This design significantly under utilizes the hardware, so we switch to a linear implementation to be discussed in the following section. However, in case of a linear

implementation, we need to keep track of the *Max* value as data is overwritten.

3.4 Linear (1D) systolic implementation

Linear systolic array is a linear arrangement of processors (hereafter called cells), connected in series, where data flows synchronously across the array between neighbors, as shown in Figure 3.10. The cells are used repeatedly during each clock cycle. In the following subsections, the cell design for linear systolic array implementation, a subsequent system design and an extended design using *Double Data Rate (DDR)* RAM are presented.

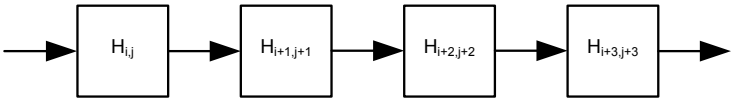


Figure 3.10: Description of a 4-element linear systolic array

3.4.1 Cell design

Figure 3.11 shows the block diagram representation of a basic cell design, for computing the elements of the *H* matrix for linear systolic array, using Equation 2.2.

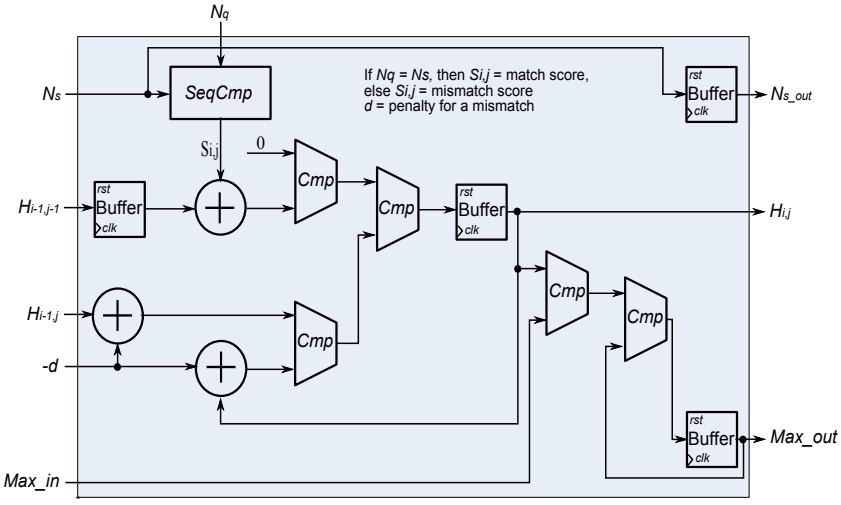


Figure 3.11: Cell design for linear systolic array implementation

In the cell design of Figure 3.11, *SeqCmp* compares the corresponding characters of the two input sequences and generates a similarity score. If the corresponding characters are the same, the similarity score is equal to a specific match score, otherwise

it is equal to a mismatch score. The diagonal input from element ($H_{i-1,j-1}$) is buffered for one clock cycle, as it is used as a diagonal element after two steps. The similarity score is added with the delayed diagonal element using an adder, the output of which is compared with a *0* using a comparator. The comparator returns *0* if the output of the adder is negative, otherwise it returns the output of the adder. The left element ($H_{i-1,j}$) and the up element (which is the current value of the cell) are added with the gap penalty using adders, the outputs of which are compared using a comparator that returns the greater of the two values. This value is then compared with the value of the previous comparator (the one that compared the sum of the diagonal element and the similarity score with a *0*) and the greater of the two values is returned. The value of the cell, stored in a buffer, is also compared with the Max_in value from the previous cell to find the global maximum. The current maximum value of the cell, stored in another buffer, is also compared with the global maximum. The maximum of the current and global maximums are compared and the greater of the two values is returned and hold in a buffer. Another buffer is used to delay the database sequence (N_s) by one clock cycle for the next element of the array. The external clock and reset lines are connected with the *clk* and *rst* inputs of all the buffers.

3.4.2 System design

The cell design shown in Figure 3.11 can be used to build a linear systolic array based systems of any size, depending on the availability of hardware resources. Figure 3.12 shows an FPGA-based 4-PE linear systolic array implementation for S-W based sequence alignment using *Block RAM (BRAM)* for intermediate data storage before transmitting the resultant data to the *PC*. In addition, there are two BRAMs for the two input sequences, i.e. (BRAM for N_q) and (BRAM for N_s). These two BRAMs are initialized with the values of the two input sequences. The input sequences are applied to the PEs in such a way that the N_q values stay fixed in their corresponding PEs, whereas the N_s values are propagated through the array in synchronism with the clock.

Figure 3.13 presents a block diagram representation of the design that works as a BRAM control unit. It consists of a BRAM data control unit, an address control unit and the BRAM itself. The address control unit generates the appropriate read/write addresses based on the request from the data control unit and the status of the read/write flag. Also it sends back an acknowledge signal to the data control unit accordingly.

Figure 3.14 shows a state machine for the BRAM Address Control Unit. It stays in the IDLE state, unless there is a request from the data control unit. It goes to SETUP state if there is a request. From the SETUP state it goes to the READ state if the read/write flag is *0*, otherwise it goes to the WRITE state. After each READ or WRITE it checks the status of the read/write flag and request signal and goes to the next state accordingly.

A parallelized S-W algorithm requires $N_s + N_q - 1$ operations for computing the entire H matrix [29]. If $Nq = Ns = N$, then the cycles required becomes $2N - 1$. In practice, a large number of PEs is required to align long sequences. The larger the

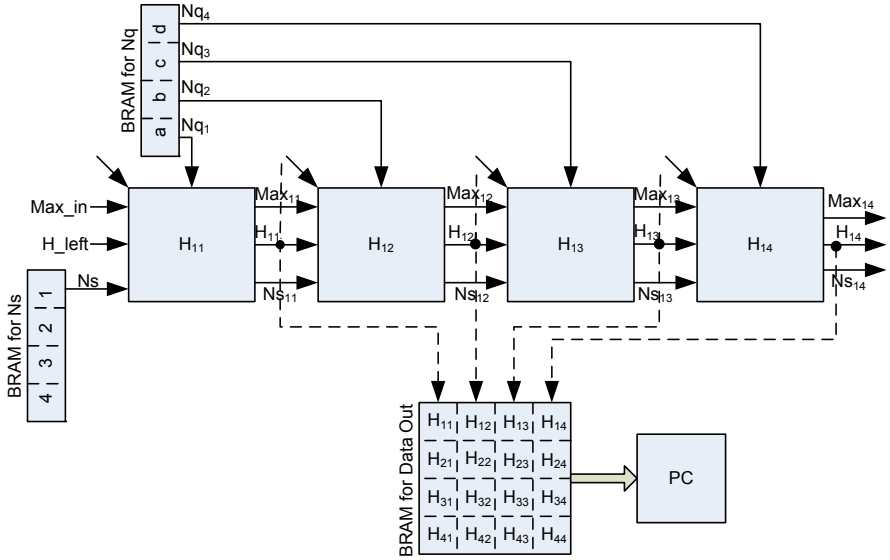


Figure 3.12: Linear systolic array design using BRAM for intermediate data storage

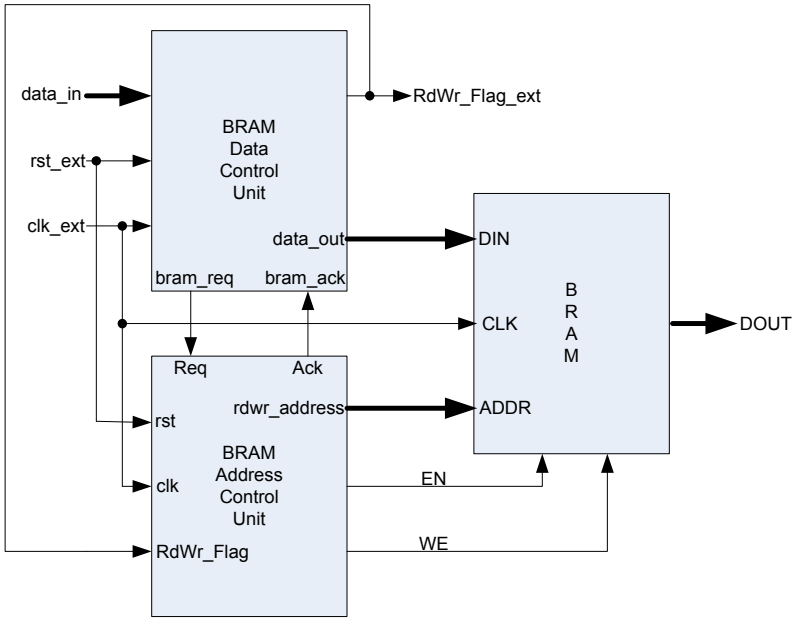


Figure 3.13: Block diagram representation of BRAM control design

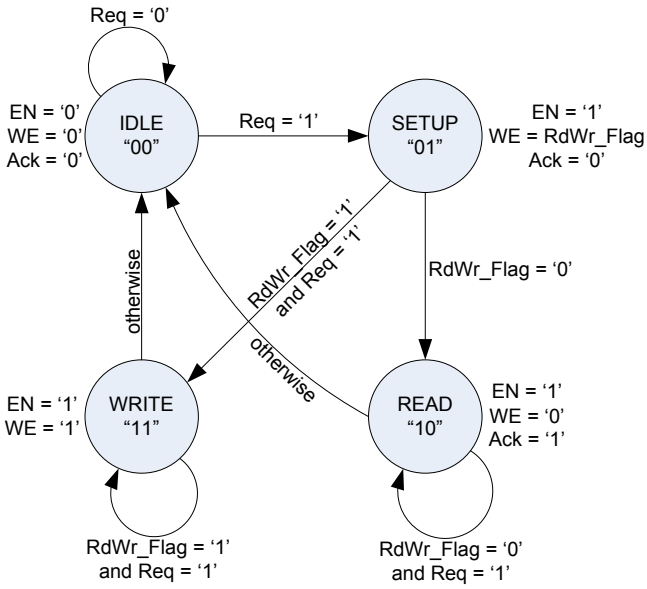


Figure 3.14: State machine for BRAM address control unit

number of PEs, the longer the query sequence that can be aligned against a database sequence and the better the performance. Table 3.3 presents performance in *Giga Cell Updates Per Second (GCUPS)* and *frequency (f)* in MHz for varying number of *PEs (N)* implemented on Xilinx Virtex-II Pro FPGA.

Table 3.3: Performance in GCUPS and frequency in MHz for various number of PEs (N)

N	f	$P = N \times f$	N	f	$P = N \times f$
20	110.26	2.21	80	110.26	8.82
40	110.26	4.41	96	110.26	10.58
60	110.26	6.61	120	110.26	13.23

The maximum number of PEs that could be implemented on a Virtex-II Pro FPGA platform are 120, but in practice, the size of the sequences may be larger than 120. The average case sequence length may be considered as 500 (74% of sequences in Swiss-Prot are ≤ 500 [8]). One possible solution to deal with this issue is to split the computation into k passes, where $k \geq 1$ is an integer.

3.4.3 Extended design with DDR RAM

Table 3.4 shows a sample H matrix for aligning two sequences of m characters each. If the precision of the aligned output data is 16 bits wide [70], then, for $m = 500$ (74% of sequences in Swiss-Prot are of length ≤ 500 [8]), the total amount of data that needs to be stored in memory is, $500 \times 500 \times 16 = 4$ Mbits. This amount increases with the increasing length of the query and database sequences.

Table 3.4: H matrix for aligning sequences of m characters each

	A	**C**	...	...	**T**	**G**
G	$H_{1,1}$	$H_{1,2}$	...	...	$H_{1,m-1}$	$H_{1,m}$
A	$H_{2,1}$	$H_{2,2}$	...	...	$H_{2,m-1}$	$H_{2,m}$
...	...	...	...	...	...	...
...	...	...	...	...	...	...
T	$H_{m-1,1}$	$H_{m-1,2}$	...	...	$H_{m-1,m-1}$	$H_{m-1,m}$
C	$H_{m,1}$	$H_{m,2}$	...	...	$H_{m,m-1}$	$H_{m,m}$

In practice, e.g. FPGA implementations, a large number of PEs is required to align long sequences. The larger the number of PEs, the longer the query sequences that can be aligned against the database sequences and the better the performance. When all the PEs are simultaneously active, the bandwidth required to store the resultant output data increases with the increasing array length. Hence, the on-chip local BRAM becomes very limited for storing all the intermediate values and can only be used as a buffer that transfers the data to an off-chip main memory, e.g. the DDR RAM. Figure 3.15 gives a block diagram description of such a system. Thus, the overall performance of the hardware system not only depends on the availability of computational resources, i.e. the number of PEs, but also on the bandwidth of the main memory (B_{main}). Both the issues are further elaborated in Chapter 6, where theoretical performance boundaries and subsequent performance and bandwidth optimization are presented.

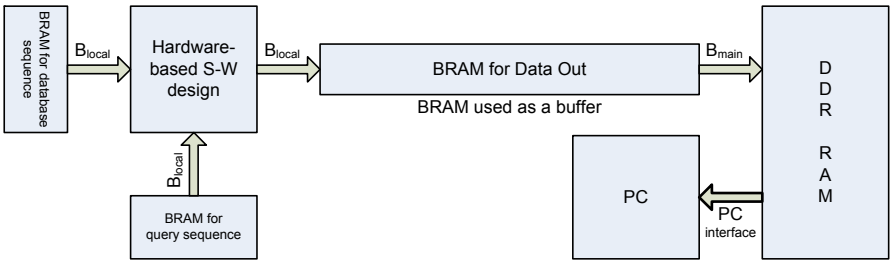


Figure 3.15: Linear systolic array design using BRAM and DDR RAM

3.5 Summary

Besides providing a classification of the acceleration methods for sequence alignment and the relevant literature review, this chapter introduced an accurate profiling and acceleration evaluation approach. Further, it presented rectangular and linear FPGA-based systolic array implementations for sequence alignment applications. The main topics presented in the chapter are as follows.

- Classification of various available acceleration methods for S-W based sequence alignment, followed by a comparison between their respective speedups relative to some baseline performance, thereby showing the need to identify a common measure for comparing different acceleration methods presented in the literature.
- Description of MOLEN reconfigurable platform and its use as an accurate profiling and acceleration evaluation platform for sequence alignment applications.
- Description of a software implementation of S-W algorithm and its profiling using both gprof and MOLEN power PC compiler. This is followed by a CCU design using the interface provided by MOLEN platform for the function identified in the profiling results.
- Implementation of both software-only and hardware-accelerated versions of S-W algorithm using MOLEN platform and an evaluation of the achieved speedup.
- *Rectangular (2D)* systolic array implementation, its corresponding cell design and a system design based on this cell.
- *Linear (1D)* systolic array implementation, its corresponding cell design and a subsequent system design.
- An extended design using DDR RAM and an insight to optimize the performance and bandwidth limitation which is further elaborated later in the thesis.

Chapter 4

RVE-based FPGA Acceleration

RVE is a kind of loop transformation that removes all the data dependencies from an algorithm, so that the algorithm can be parallelized to its maximum. In this chapter, RVE-based FPGA acceleration of sequence alignment and its comparison with traditional systolic array based acceleration is presented.

The chapter starts with an introduction to the RVE technique in Section 4.1, followed by a *rectangular (2D)* RVE implementation in Section 4.2. Section 4.3 presents a *linear (1D)* RVE implementation. Section 4.4 provides RVE performance evaluation, whereas Section 4.5 summarizes the chapter.

4.1 Introduction

This section aims at providing some insight into the RVE approach and its application in biological sequence alignment. An introduction to the RVE approach is provided in Section 4.1.1, whereas Section 4.1.2 presents an implementation procedure using the RVE approach.

4.1.1 The RVE approach

RVE [71] is a kind of loop transformation which removes all data dependencies from a program, so that the program is parallelized to its maximum. The basic idea is that if any statement G_i is dependent on statement H_j for some iteration i and j, then instead we wait for H_j to complete and then execute G_i, we will replace all the occurrences of the variable in G_i that create dependency with H_j with the computation of that variable in H_j. In this way there is no need to wait for the statement H_j to complete and statement G_i can be executed independently of H_j. This step is recursively repeated until the statement G_i is not dependent on any other statement, other than inputs or known values, which essentially means that G_i can be computed without any delays.

This transformation is explained clearly in Example 1, which adds the loop counter. Therefore after applying the RVE, we get an expression with five terms to be added, as shown in Example 2.

Example 1: A simple example which adds the loop counter

$A[1] = 1$
for $i = 2$ to 5
$A[i] = A[i\text{-}1] + i$ ——- (G_i)
end for

Example 2: After applying RVE on *Example 1*

$$\begin{aligned} A[5] &= A[4] + 5 \\ &= A[3] + 4 + 5 \\ &= A[2] + 3 + 4 + 5 \\ &= A[1] + 2 + 3 + 4 + 5 \\ &= 1 + 2 + 3 + 4 + 5 \end{aligned}$$

In this way, the whole expanded statement in Example 2 can be computed in parallel and efficiently using binary tree structure as shown in Figure 4.1 requiring 3 cycles for the entire computation. The major drawback of this technique is that the speed up is achieved at the cost of redundancy, which consumes a lot of hardware resources.

In this chapter, we present various implementations of S-W based sequence alignment applications using the RVE approach and compare the results with implementations based on the systolic array approach as discussed in the previous chapter.

4.1.2 Sequence alignment using RVE approach

To eliminate the limitation posed by the inherent data dependencies in the S-W based sequence alignment applications, we apply the RVE approach. Instead of computing an element of the H matrix at a time, as discussed in Chapter 2, we can compute a block of $k \times k$ elements in parallel, by partially applying the RVE approach. When it is applied to Equation 2.2, we get the following equations for $H_{i,j}$ in a 2×2 block.

$$H_{i-1,j-1} = \max \begin{cases} H_{i-1,j-2} - d \\ H_{i-2,j-2} + S_{i-1,j-1} \\ H_{i-2,j-1} - d \\ 0 \end{cases} \tag{4.1}$$

$$H_{i-1,j} = \max \begin{cases} H_{i-1,j-2} - 2d \\ H_{i-2,j-2} - d + S_{i-1,j-1} \\ H_{i-2,j-1} + S_{i-1,j} \\ H_{i-2,j} - d \\ 0 \end{cases} \tag{4.2}$$

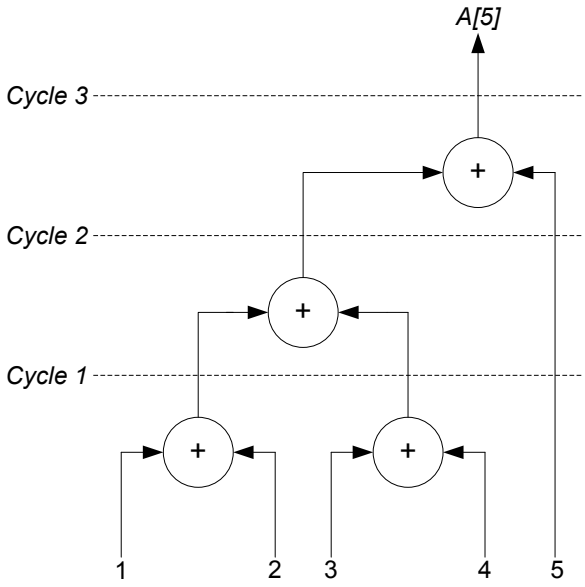


Figure 4.1: Circuit for the Example 2

$$H_{i,j-1} = \max \begin{cases} H_{i,j-2} - d \\ H_{i-1,j-2} + S_{i,j-1} \\ H_{i-2,j-2} - d + S_{i-1,j-1} \\ H_{i-2,j-1} - 2d \\ 0 \end{cases} \tag{4.3}$$

$$H_{ij} = \max \begin{cases} (H_{i,j-2}\ MAX\ H_{i-2,j}) + -2d \\ H_{i-1,j-2} - d + (S_{i,j-1}\ MAX\ S_{i,j}) \\ H_{i-2,j-2} + S_{i-1,j-1} + S_{i,j} \\ H_{i-2,j-1} - d + (S_{i-1,j}\ MAX\ S_{i,j}) \\ 0 \end{cases} \tag{4.4}$$

Figure 4.2 shows the way to fill a 2×2 H matrix using the RVE approach, as per Equations 4.1, 4.2, 4.3 and 4.4, where S is the match/mismatch score and d is the gap penalty [72]. In each case the cell to be filled is highlighted along with the cells which are required for its computation.

We define the size of RVE block as the *blocking factor* (b_f). So, for a 2×2 array, implemented using RVE, the blocking factor is $b_f = 2 \times 2$. The advantage of this approach is that all four elements in the 2×2 array are computed in parallel, without waiting for the previous elements to be computed. Thus, the data dependencies are minimized, as compared to the traditional systolic array implementation.

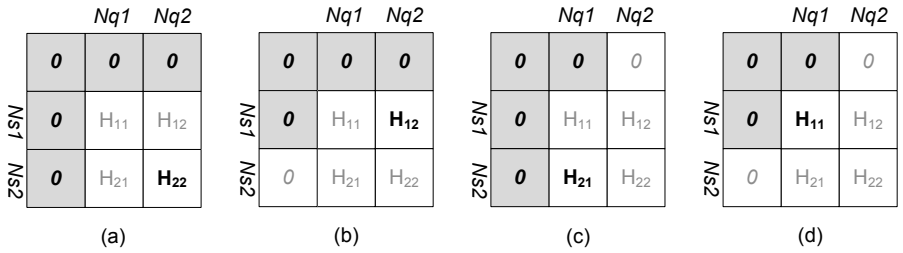


Figure 4.2: Filling a 2×2 H matrix using the RVE approach

4.2 Rectangular (2D) RVE implementation

In this section, a building block for 2-dimensional RVE implementation is described and system design based on this building block is presented. Further a discussion of the results obtained is given.

4.2.1 Building block description

Figure 4.3 shows the block diagram description of a building block for RVE implementation with $b_f = 2 \times 2$. It provides the detailed pin outs of the RVE block, where four pins are reserved for the corresponding characters of the input sequences N_s and N_q, i.e $N_{s(i)}$, $N_{s(i-1)}$, $N_{q(j)}$ and $N_{q(j-1)}$. Five pins are for the H inputs, i.e. $H_{i-2,j-2}$, $H_{i,j-2}$, $H_{i-2,j}$, $H_{i-1,j-2}$ and $H_{i-2,j-1}$. One pin is for the *gap penalty (d)* and two for the *clock (CLK)* and *reset (RST)*. The four output pins are $H_{i,j}$, $H_{i-1,j}$, $H_{i,j-1}$ and $H_{i-1,j-1}$.

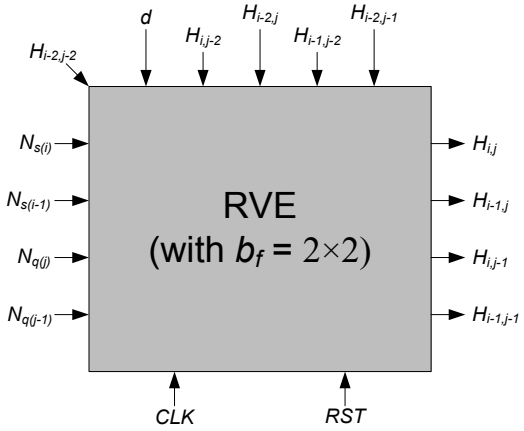


Figure 4.3: Block diagram description of a 2D RVE design with $b_f = 2 \times 2$

When implemented on a Xilinx Virtex-II Pro (XC2VP30) FPGA, the RVE block shown in Figure 4.3 consumes 30 *ns* for a clock frequency of 50 MHz. The slices utilized by the block design are 95 out 13696.

4.2.2 System design

Using the RVE block with $b_f = 2 \times 2$ as a macro design, a 5×5 blocks array is implemented. Figure 4.4 shows the block diagram representation of this implementation.

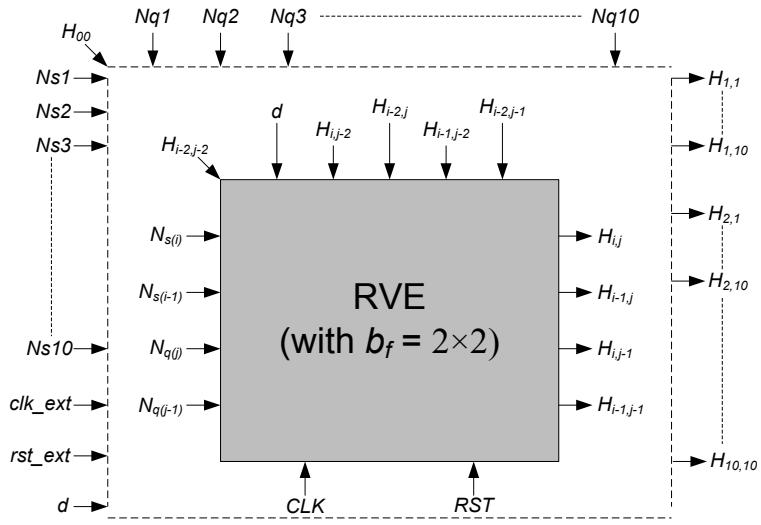


Figure 4.4: Block diagram representation of a 5×5 array using multiple RVE blocks with $b_f = 2 \times 2$

Figure 4.5 shows, how a 5×5 blocks array is constructed by using RVE blocks with $b_f = 2 \times 2$. For the blocks in the first row and first column of Figure 4.5, all the inputs come from outside, as shown by external input pins of Figure 4.4. The four outputs of each block go to the inputs of corresponding neighboring blocks, where the remaining inputs corresponding to sequence characters come from outside. The entire design consumes 2409 out of 13696 slices without considering the *IO* hardware overhead. The resources utilized with the *IO* hardware overhead are equivalent to 2630, thus a maximum of 130 PEs (RVE blocks with $b_f = 2 \times 2$) can be fitted, while implementing on a Xilinx Virtex-II Pro (XC2VP30) FPGA. Since four $H_{i,j}$ elements are calculated per PE, the maximum number of $H_{i,j}$ elements calculated is $130 \times 4 = 520$. There are 9 anti-diagonals in a 5×5 array using RVE blocks with $b_f = 2 \times 2$, represented by letters ***A***, ***B***, ***C***, ***D***, ***E***, ***F***, ***G***, ***H*** and ***I*** in Figure 4.5. Each anti-diagonal is computed in one clock cycle, so the latency is equivalent to 9 clock cycles = $9 \times 30 = 270\ ns$.

4.2.3 Discussion of results

Table 4.1 displays the results of comparing various systolic array implementation with their equivalent RVE implementations, where the first column represents the type of implementation. The second column represent the time consumed. The third column shows the speedup of each design with respect to its equivalent systolic array

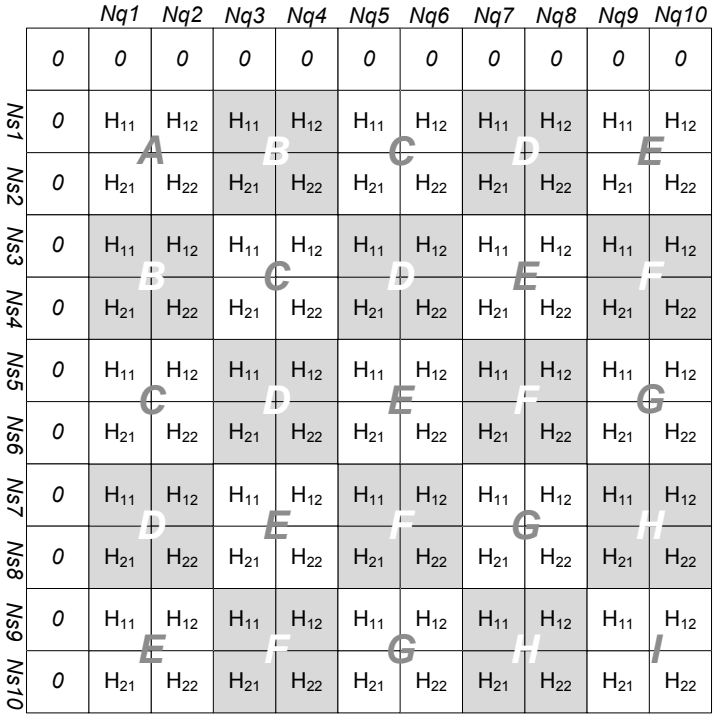


Figure 4.5: 5×5 array using RVE blocks with $b_f = 2 \times 2$

design. The fourth column gives the number of slices utilized by each implementation including the *IO* hardware overhead. The device used for implementation is Xilinx Virtex-II Pro FPGA, where the total number of available slices is 13696. The last column presents the hardware utilization cost.

The performance gain in terms of latency, achieved by 5×5 array using RVE blocks with $b_f = 2 \times 2$, as compared to its equivalent 10×10 traditional systolic array implementation (discussed in the previous chapter) = 380/270 = 1.41. This performance gain is achieved at the cost of utilizing 2630/2096 = 1.25 times additional hardware resources. In case of 14×10 array using RVE blocks with $b_f = 2 \times 2$, the performance gain in comparison with its equivalent 28×20 systolic array implementation = 940/690 = 1.36, at the cost of utilizing 13694/10751 = 1.27 times additional hardware resources. Figure 4.6 shows a graphical comparison of the results given in Table 4.1.

The performance gain achieved by the RVE approach is due to the fact that it eliminates the limitation posed by the inherent data dependencies in the S-W based sequence alignment applications. However, this performance is achieved at the cost of utilizing additional hardware resources. The speedup achieved by applying RVE increases with the increasing blocking factor (b_f), but resource utilization also increases

Table 4.1: Comparison between 2D systolic array and RVE implementations

Implementation	Time (*ns*)	Speedup	Slices	Hardware cost
2×2 systolic array	60	1	70	1
RVE block with $b_f = 2 \times 2$	30	2	95	1.36
10×10 systolic array	380	1	2096	1
5×5 array using RVE blocks with $b_f = 2 \times 2$	270	1.41	2630	1.25
28×20 systolic array	940	1	10751	1
14×10 array using RVE blocks with $b_f = 2 \times 2$	690	1.36	13694	1.27

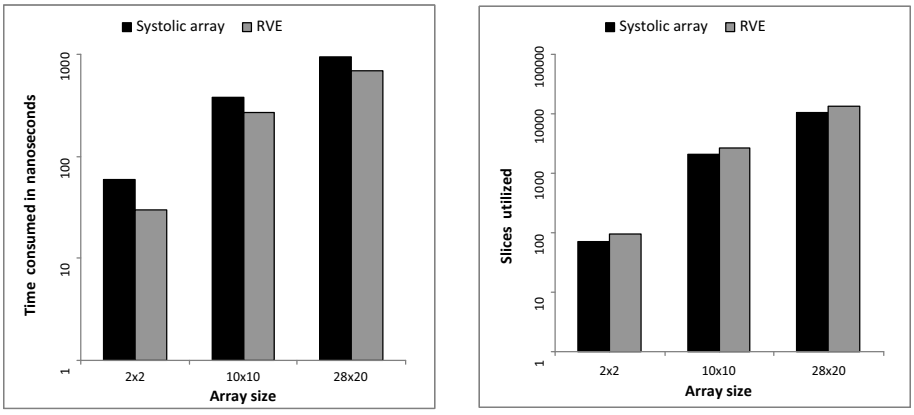


Figure 4.6: Comparison between various 2D systolic array and 2D RVE implementations on a logarithmic scale

as a consequence. Thus the limiting factor is the availability of hardware resources on the device used for implementation, Xilinx Virtex-II Pro (XC2VP30) FPGA in this case. Other major issue with this implementation is that the hardware is underutilized most of the times. To overcome this issue, we present a linear RVE design as discussed in the next section.

4.3 Linear (1D) RVE implementation

This section presents the design of a basic building block for the linear RVE implementation. Further, it presents system designs based on this building block and presents a discussion of the results achieved.

4.3.1 Building block description

Figure 4.7 shows the block diagram representation of the linear RVE design that implements a 2×2 array. This RVE block depends on the search and target sequences (i.e. the query and database sequences), the *gap penalty (d)*, *Max* input, *CLK* and *RST*, in addition to the three external elements i.e. $H_{i-2,j-2}$, $H_{i,j-2}$ and $H_{i-1,j-2}$, and two feedback elements $H_{i-2,j}$ and $H_{i-2,j-1}$. Similarly, in addition to the four elements of the H matrix i.e. $H_{i,j}$, $H_{i,j-1}$, $H_{i-1,j}$ and $H_{i-1,j-1}$, the RVE block also outputs *Max* output, N_{s1} and N_{s2}, which become inputs for the next block, when the array is extended.

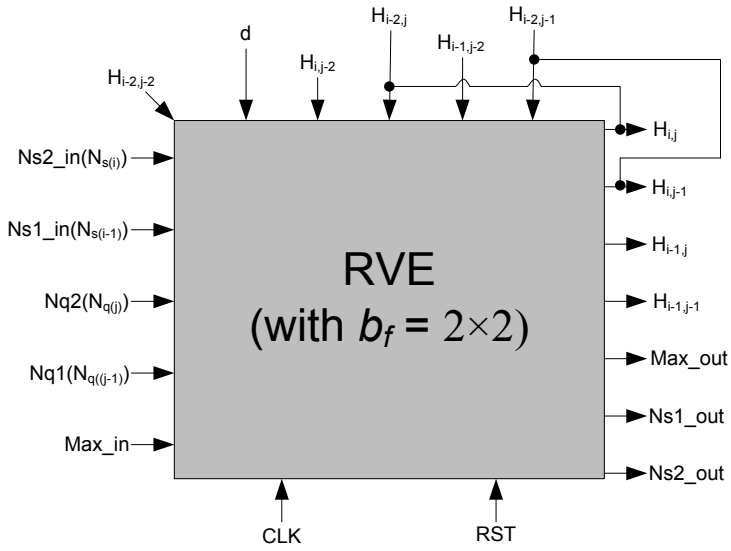


Figure 4.7: Block diagram representation of the linear RVE design with $b_f = 2 \times 2$

Figure 4.8 shows the logical description of an RVE implementation with b_f = 2×2. The comparators in the 1^{st} column of Figure 4.8 compare the corresponding characters of the input sequences and generate the similarity score accordingly. The adders in the 2^{nd} column add the gap penalty with the elements $H_{i,j-2}$, $H_{i-1,j-2}$, $H_{i,j}$ and $H_{i,j-1}$, where the 1^{st} two are external elements and the 2^{nd} two are feedback elements. The *AND* gates in the 3^{rd} column perform logic anding between the outputs of the upper 3 comparators in the 1^{st} column. The adders and comparators in the following columns perform addition and max operation on the inputs from the preceding columns. The

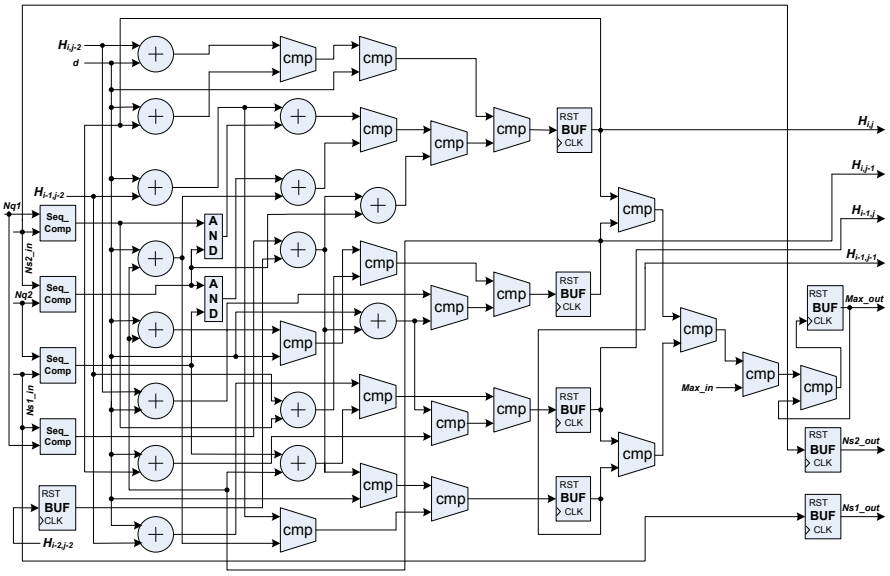


Figure 4.8: Logical description of an RVE implementation with $b_f = 2 \times 2$

values of the five outputs $H_{i,j}$, $H_{i,j-1}$, $H_{i-1,j}$, $H_{i-1,j-1}$ and *Max_out* are buffered at the output. N_{s1_in} and N_{s2_in} are also buffered in the last column to get N_{s1_out} and N_{s2_out} for the next block in the array. When implemented in VHDL, this block with $b_f = 2 \times 2$ consumes 13 *ns*, where the clock period is 30 *ns* and the frequency is 33.33 MHz. Using this block as a macro, RVE designs of various sizes can be developed depending on the availability of hardware resources.

4.3.2 System design

The basic building block for the linear RVE design shown in Figure 4.8 is used to develop RVE based systems of various sizes for sequence alignment applications. Figure 4.9 shows a 2-block linear RVE implementation as an example, where the blocks are connected in a linear systolic array fashion.

The 2-block linear RVE design shown in Figure 4.9, which is equivalent to the 4-element linear systolic array design, is implemented in VHDL and the post place and route simulation results show that the latency of the array is 300 *ns*, whereas the slices consumed are 254 out of 13696. The platform used for implementation is Xilinx Virtex-II Pro FPGA.

4.3.3 Discussion of results

Table 4.2 presents the implementation results for various linear systolic array and linear RVE designs. It demonstrates that a 4-element linear systolic array implementation

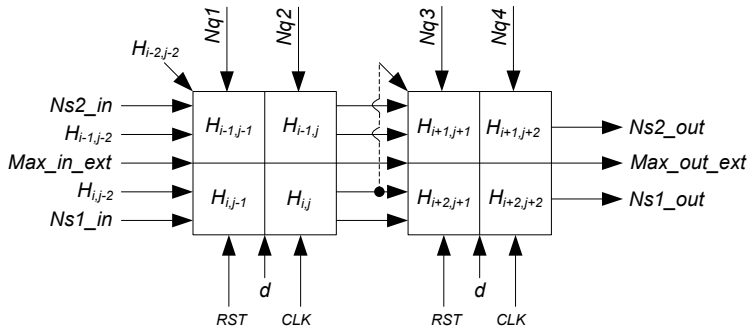


Figure 4.9: 2-block linear RVE design

consumes 700 *ns* and utilizes 127 out of 13696 slices, when implemented on a Xilinx Virtex-II Pro FPGA. Thus a maximum of 107 PEs can be implemented on the same device, thereby consuming most of the available slices on the FPGA. This implementation is used as a reference for comparison, which is traditionally used for accelerating the S-W based sequence alignment applications. The 2-block linear RVE implementation consumes 300 *ns* and utilizes 254 out of 13696 slices, when implemented on a Xilix Virtex-II Pro FPGA. Thus a maximum of 53 PEs can be implemented, using the same device. Thus in comparison with a traditional 4-element linear systolic array implementation, the 2-block linear RVE implementation improves the performance by a factor of 700/300 = 2.33, at the cost of utilizing 254/127 = 2 times additional hardware resources.

Table 4.2: Comparison between linear systolic array and linear RVE implementations

Implementation	Time (*ns*)	Speedup	Slices	Hardware cost
4-element linear systolic array	700	1	127	1
2-block linear RVE	300	2.33	254	2
10-element linear systolic array	1900	1	297	1
5-block linear RVE	900	2.11	601	2.02
200-element linear systolic array	39900	1	6350	1
100-block linear RVE	19900	2.01	12700	2

The table also shows a comparison between 10-element linear systolic array and 5-block linear RVE implementations, where the 5-block linear RVE design performs

2.11 times better than the 10-element linear systolic array implementation at the cost of utilizing 2.02 times additional hardware resources. The table further demonstrates a comparison between 200-element linear systolic array and 100-block linear RVE implementations, where the 100-block linear RVE implementation achieves 39900/19900 = 2.01 times higher performance than the linear systolic array implementation at the cost of utilizing 12700/6350 = 2 times additional hardware resources. A full scale linear systolic array implementation fits a maximum of 428 elements, whereas a full scale linear RVE implementation fits a maximum of 106 RVE blocks, where the device utilized for implementation is Xilinx Virtex-II Pro FPGA. Thus due to higher resource utilization by the linear RVE design, the full scale implementations are not comparable. From Table 4.2, it can be concluded that the linear RVE implementation is preferred in cases where high performance is desired and hardware cost is not a big concern.

The chart in Figure 4.10 shows a graphical comparison between various linear systolic array and linear RVE implementations, where the factors considered for comparison are the time consumed and the number of slices utilized. Clearly, the time consumption decreases by applying RVE, as shown in Figure 4.10(a). On the other hand, the resource utilization increases by applying RVE, as shown in Figure 4.10(b).

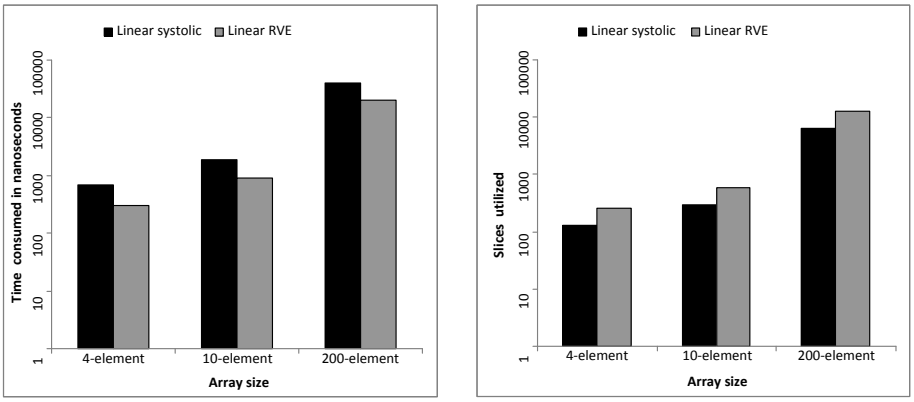


Figure 4.10: Comparison between various linear systolic array and linear RVE implementations on a logarithmic scale

In the following section, the performance of RVE implementations is evaluated for various array sizes, taking the hardware utilization (area) cost into consideration.

4.4 RVE performance evaluation

The RVE designs discussed in the previous sections focused mainly on the latency as a performance metric, which is the time it takes a character of the database sequence to travel through the design. The latency is given by Equation 4.5, where f_{opr} is the

maximum operating frequency and each RVE block consumes one cycle to compute the results.

$$\text{Latency} = \text{Number of RVE blocks} \times \frac{1}{f_{opr}} \tag{4.5}$$

In this section, we use other performance metrics to optimize the designs. RVE designs with various blocking factors, as shown in Figure 4.11 are analyzed using performance metrics like throughput and performance/area besides latency. This results in a better understanding of the RVE implications. The diagonal lines in Figure 4.11 indicate the RVE blocks that can be computed in parallel.

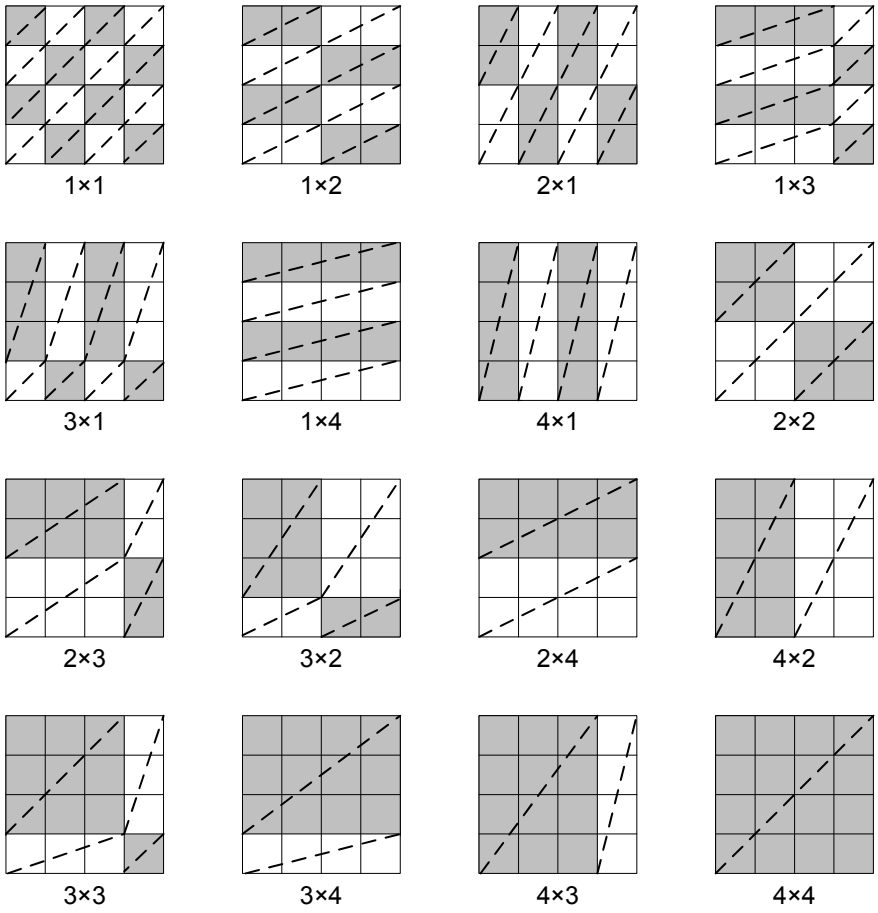


Figure 4.11: RVE designs with various blocking factors

Table 4.3 presents the performance evaluation results of linear RVE implementa-

tions with various blocking factors. The number of PEs is chosen such that a query sequence that is 36 characters long can be aligned in one run. It is clear from the 3^{rd} column of the table that for square blocking factors (i.e., 1×1, 2×2, 3×3 and 4×4), increasing blocking factors result in lower frequencies. All non-square blocking factors (e.g., 2×1, 3×1, 4×1, 3×2, 4×2 and 4×3) run on lower frequencies as well. For example, the 1×1 and 2×1 blocks run at similar frequencies, but if the blocking factor is increased from 2×1 to 3×1, the frequency drops by 37 MHz. This can partly be explained by the way the formulas expand, i.e. the number of sequential additions and the logic depth of the comparators. For the 1×1, 2×1 and 3×1 blocking factors, the number of adder stages is respectively 2, 3 and 3, whereas, the number of comparator stages is respectively 2, 3 and 3, as reported by the synthesis tool. This shows that not only the theoretical logic depth, but also the actual implementation of the circuit plays a key role in determining the maximum frequency.

Table 4.3: Performance evaluation for various RVE implementations

Blocking factor	**Number of PEs**	**f_{opr} (MHz)**	**Latency *(ns)***	**Throughput (MCUPS)**	**Performance/area (MCUPS/slice)**
1×1	36	159.0	226.4	5724	6.30
2×1	36	158.0	227.8	11376	6.81
3×1	36	121.0	297.5	13068	4.81
4×1	36	115.0	313.0	16560	4.19
2×2	18	118.0	152.5	8496	4.69
3×2	18	86.0	209.3	9288	2.61
4×2	18	67.0	268.6	9648	1.57
3×3	12	74.0	162.2	7992	1.76
4×3	12	66.8	179.6	9619	1.22
4×4	9	59.0	152.5	8496	0.81

The 4^{th} column of Table 4.3 shows that for the 2×2 square blocking factor, the latency of the design decreases significantly. But increasing the blocking factor further, the latency does not improve anymore, suggesting that going beyond 2×2 has no advantage. For the non-square blocking factors like 2×1 and 3×1, it is more complicated, as these blocks can be organized in two different ways giving rise to different latencies. This indicates that latency alone is not the best metric to evaluate the performance of an RVE design, as it only tells about how fast the end of the design is reached. To investigate the amount of work done by the design during every time unit, we compute throughput, which is defined as the number of cell updates per second. The throughput for RVE designs with various blocking factors is shown in the 5^{th} column of Table 4.3 and is calculated as per Equation 4.6, where each RVE block consumes one cycle to compute the results.

$$\text{Throughput} = \text{Number of RVE blocks} \times \text{blocking factor} \times f_{opr} \tag{4.6}$$

Like the latency, the throughput also becomes better from 1×1 to 2×2, but the improvement becomes small for larger blocking factors. This is due to the fact that the frequency decreases rapidly for larger blocking factors, as shown in Table 4.3. Only the $n \times 1$ blocks perform increasingly better. This is due to the fact that the frequency does not drop that fast here.

The last column of Table 4.3 gives performance/area in terms of MCUPS/slice. It is a useful performance metric that takes the hardware resource utilization or area cost into account and is calculated as per Equation 4.7.

$$\text{Performance per slice} = \frac{\text{Throughput}}{\text{Number of slices}} \tag{4.7}$$

It becomes clear that when taking the area into account, RVE does not perform better than the default 1×1 design for most cases. Only the RVE design with the 2×1 blocking factor gives a better performance per slice than the default case, giving rise to the conclusion that RVE designs with non-square blocking factors should also be explored for higher performance.

4.5 Summary

Besides providing an introduction to the RVE approach, this chapter presented rectangular and linear RVE implementations for biological sequence alignment applications and evaluated the performance for various RVE implementations. The main topics presented in the chapter are as follows.

- Introduction to the RVE approach and its application in biological sequence alignment.
- Rectangular (2D) RVE implementation, its corresponding building block description and the subsequent system design. Moreover, its comparison with equivalent 2D systolic array implementation and a discussion of the results.
- Linear (1D) RVE implementation, its corresponding building block description and subsequent system design. Its comparison with the equivalent linear systolic array implementation and a discussion of the results.
- A discussion about the speedups achieved by various RVE implementations and their hardware resource utilization costs.
- Performance evaluation of RVE designs with various blocking factors.

Chapter 5

GPU Acceleration

This chapter aims at exploiting the parallelization capabilities of the GPUs for biological sequence alignments. The chapter begins with a discussion of GPU as a computational platform in Section 5.1. Section 5.2 presents an optimized GPU implementation for protein sequence alignment. Section 5.3 provides a discussion of the results achieved by the optimized implementation. Section 5.4 discusses the performance limits, whereas Section 5.5 summarizes the chapter.

5.1 GPU as a computational platform

This section provides background information about GPU as a computational platform by discussing the CUDA framework, its programming and memory models. Furthermore, it explains the important phenomenon of coalescing to reduce latency of global memory. It also discusses the previous GPU implementations for biological sequence alignment.

5.1.1 CUDA framework

CUDA is the hardware and software architecture that enables NVIDIA GPUs [73] to execute programs written in C, C++, Fortran, OpenCL [74], DirectCompute [75], and other languages. A CUDA program calls kernels that run on the GPU, as shown in Figure 5.1. A kernel executes in parallel across a set of threads, where a thread is the basic unit in the programming model that executes an instance of the kernel, and has access to registers and per thread local memory. The programmer organizes these threads in grids of thread blocks, where a thread block is a set of concurrently executing threads and has a shared memory for communication between the threads. A grid is an array of thread blocks that execute the same kernel, read inputs from and write outputs to global memory, and synchronize between interdependent kernel calls.

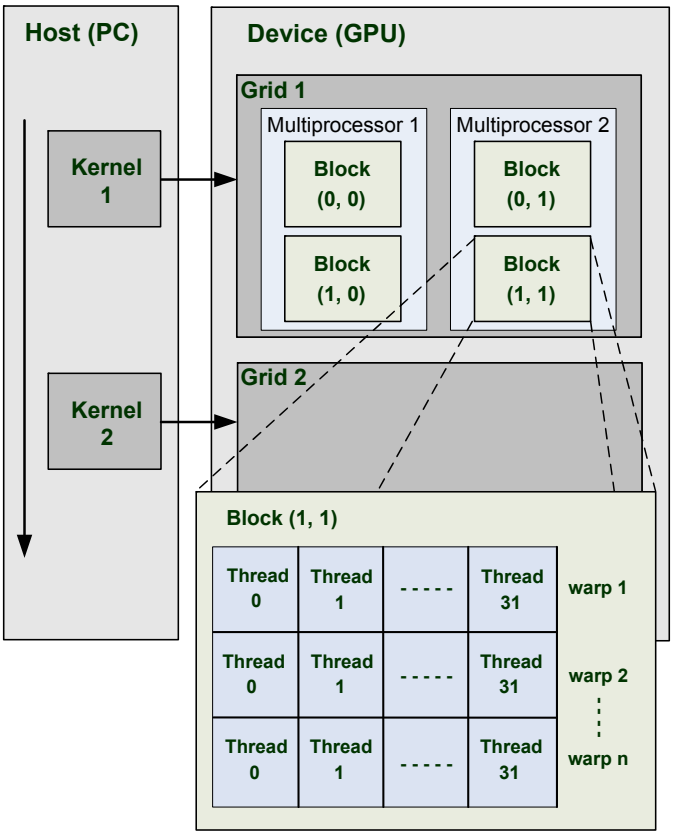


Figure 5.1: CUDA hierarchy of threads, blocks and grids

CUDA's hierarchy of threads maps to a hierarchy of processors on the GPU. A GPU executes one or more kernel grids. A GPU consists of multiprocessors that execute one or more thread blocks, as shown in Figure 5.1. Multiple thread blocks can be scheduled by the GPU to run on one multiprocessor sequentially, or in parallel by using thread switching. CUDA cores, i.e. the processing elements within a multiprocessor, execute threads in groups of 32 called *warps*. Performance on GT200-class GPUs can be optimized a great deal by having threads in a half-warp (16 threads) execute the same code path and access memory in a close vicinity.

In the CUDA parallel programming model various memory spaces exist [73]. The complete set of CUDA memory spaces is given in Figure 5.2, where global memory is the GPU's RAM. Accessing it has a high latency, which can be hidden by switching execution to other threads that are not waiting for memory accesses.

The second type of memory shown in Figure 5.2 is the texture cache. Textures are cached 'windows' into global memory, optimized for spatially local reads.

The third type of memory is the constant cache, which is a read-only portion of

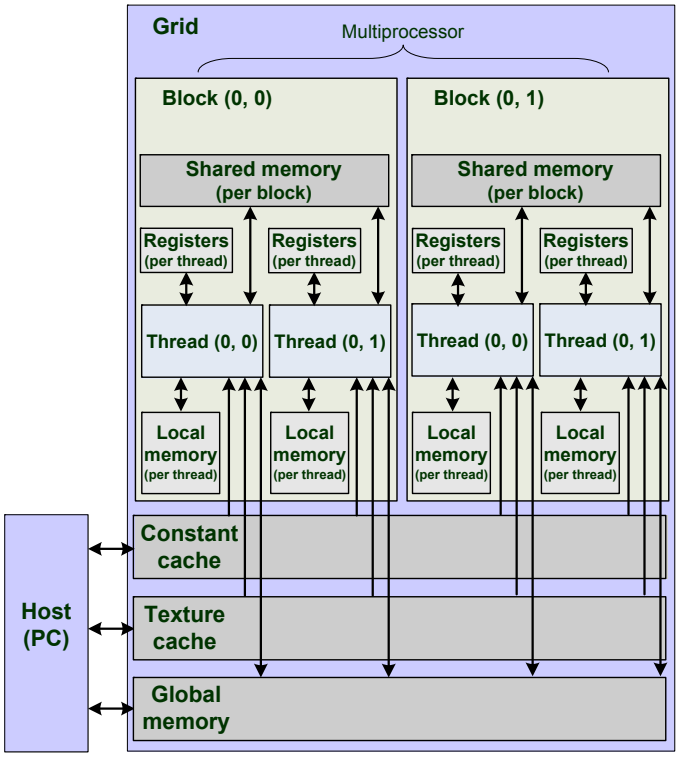


Figure 5.2: CUDA memory hierarchy

global memory. It is cached at each multiprocessor and accessing it is as fast as accessing a register.

The other types of memories are shared memory and local memory, where shared memory is a fast memory used for inter-thread communication within a thread block and local memory is a per thread portion of the global memory used for function calls and register spills. Additionally, each multiprocessor offers a bank of registers, shared between its processors.

5.1.2 Coalescing

Latency of global memory can be avoided altogether by *coalescing* memory accesses as shown in Figure 5.3, where each thread of a half-warp of 16 threads accesses a 4-byte value in global memory. The values in Figure 5.3(a) are all stored at unordered different addresses. In this case, each thread will execute a 32-byte (instead of 4-byte) memory access sequentially, since 32 bytes is the smallest memory access size supported by the GPU. Other possible access sizes are 64 and 128 bytes. This wastes 28 bytes of bandwidth per access adding to a total bandwidth wastage of $28 \times 16 = 448$

bytes for all 16 threads and as accesses take place sequentially, latency will be high.

In Figure 5.3(b), the values accessed are stored at neighboring addresses. In this case, coalescing takes place. The GPU issues a single 64-byte load, thus no bandwidth is wasted and only a single access is needed.

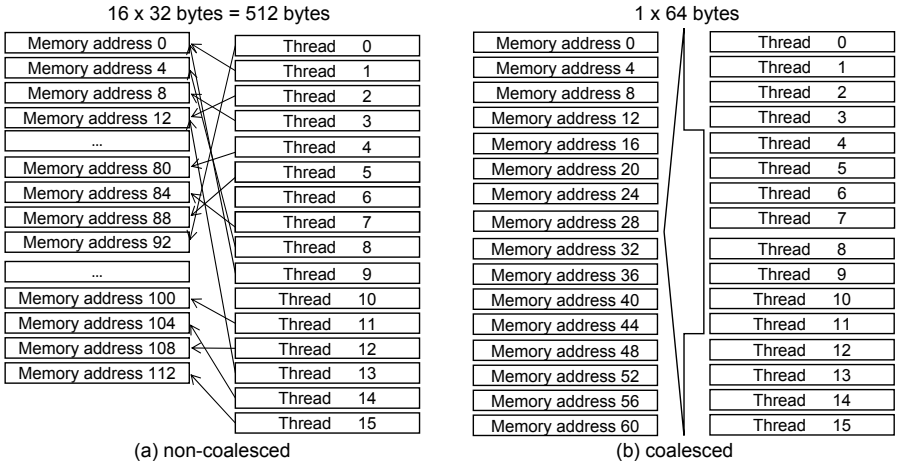


Figure 5.3: The effect of coalescing on memory reads

5.1.3 Previous implementations

The first known implementations of S-W based sequence alignment on a GPU are presented in [65] and [76]. These approaches are similar and use the OpenGL graphics API to search protein databases. First the database and query sequences are copied to GPU texture memory. The score matrix is then processed in a systolic array fashion [25], where the data flows in anti-diagonals. The results of each anti-diagonal are again stored in texture memory, which are then used as inputs for the next pass. The implementation in [65] searched 99.8% of Swiss-Prot (almost 180,000 sequences) and managed to obtain a maximum speed of 650 MCUPS compared to around 75 for the compared CPU version. The implementation discussed in [76] offers the ability to run in two modes, i.e. one with and one without traceback. The version with no traceback managed to perform at 241 MCUPS, compared to 178 with traceback and 120 for the compared CPU implementation. Both implementations were benchmarked using a Geforce GTX 7800 graphics card.

The first known CUDA implementation, 'SW-CUDA', is discussed in [77]. In this approach, each of the GPU's processors performs a complete alignment instead of them being used to stream through a single alignment. The advantage of this is that no communication between processing elements is required, thereby reducing memory reads and writes. This implementation managed to perform at 1.9 GCUPS

on a single Geforce GTX 8800 graphics card when searching Swiss-Prot, compared to around 0.12 GCUPS for the compared CPU implementation. Furthermore, it is shown to scale almost linearly with the amount of GPUs used by simply splitting up the database.

Various improvements have been suggested to the approach presented in [77], as shown in [78,79]. In the 'CUDASW++' solution presented in [79], for sequences of more than 3,072 amino acids an 'inter-task parallelization' method similar to the systolic array and OpenGL approaches is used as this, while slower, requires less memory. This 'CUDASW++' solution manages a maximum speed of about 9.5 GCUPS searching Swiss-Prot on a Geforce GTX 280 graphics card. An improved version, 'CUDASW++ 2.0' has been published recently [80]. Being the fastest Smith-Waterman GPU implementation to date, 'CUDASW++ 2.0' managed 17 GCUPS on a single GTX 280 GPU, outperforming CPU-based BLAST in its benchmarks.

5.2 Optimized GPU implementation

This section presents our high performance GPU implementation for protein sequence alignment. The implementation is called *Database Optimized Protein Alignment (DOPA)*. Section 5.2.1 outlines the design and structure of the implementation, while Section 5.2.2 details the database conversion process. Section 5.2.3 discusses the loading/storing of temporary data, whereas Section 5.2.4 demonstrates the optimization of substitution matrix accesses.

5.2.1 General design

Being the most mature GPU programming toolkit to date, NVIDIA CUDA is used for the GPU programming *(device code)* in conjunction with C++ for the PC programming *(host code)*. Like with other existing GPU implementations, protein sequences from the Swiss-Prot database [81] are considered for alignment. The reason is that protein alignment is more complex than the DNA version, which makes supporting DNA alignments later on relatively simple. Figure 5.4 shows a block diagram description of the implementation. The host code is mostly concerned with loading data structures, copying them to the GPU, and copying back and presenting the results. The query sequence, converted database and other data are copied to the GPU. Then the device code is launched, which aligns the query sequence with the database sequences using the S-W algorithm.

Like other GPU implementations, our implementation returns maximum S-W scores instead of the actual alignments. Skipping the algorithm's traceback step significantly simplifies and speeds up the implementation. Furthermore, as no data structures like pointer lists need to be kept, memory consumption is decreased as well. However, to be able to generate full alignments, a number of top-scoring sequences are exported to a new database file. The sequences in this file can then be aligned on the host PC using the *Smith-Waterman search (ssearch)* tool. This approach leads to some redun-

dancy as some sequences are aligned twice, however, the number of such sequences is relatively small. By default 20 top scoring sequences are returned, whereas the Swiss-Prot database contains more than 500,000.

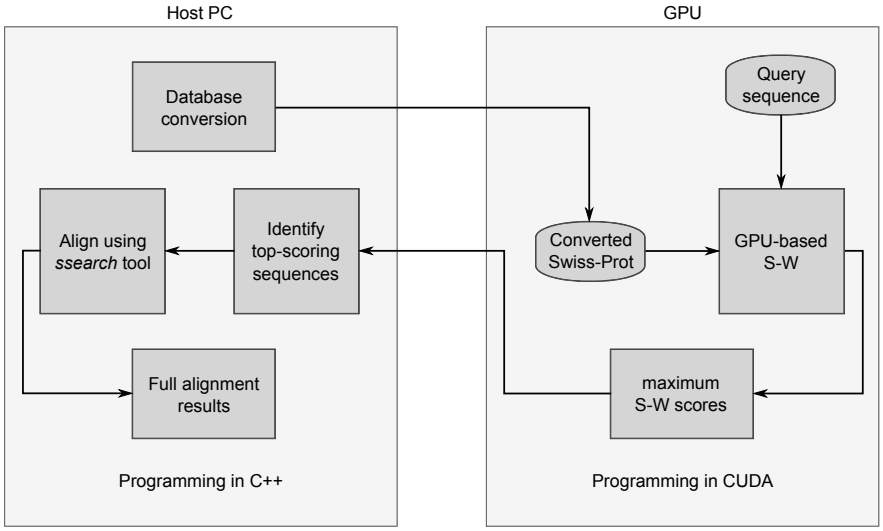


Figure 5.4: Description of the GPU implementation

Each processing element in our implementation is used to independently generate a complete alignment between a query sequence and a database sequence. This eliminates the need for inter-processor communication and results in efficient resource utilization. The GPU used for implementation (i.e. NVIDIA GTX 275) contains 240 processors, while the latest release of Swiss-Prot contains more than 500,000 sequences. Hence, it is possible to keep all processors well occupied [82].

5.2.2 Database conversion

The Swiss-Prot database is organized in FASTA format, where sequences are preceded by sequence descriptions that give names and other biological information about them. Instead of directly loading databases in FASTA format, the GPU implementation converts them to a custom GPU format to better match the device capabilities. A database only needs to be converted once, after which it is locally stored in the new format. The conversion process as shown in Figure 5.5 consists of the following steps.

Sorting

In practice the threads in a half-warp will have to wait for each other to finish their workload instead of continuing on independently. To reduce this waiting time, the

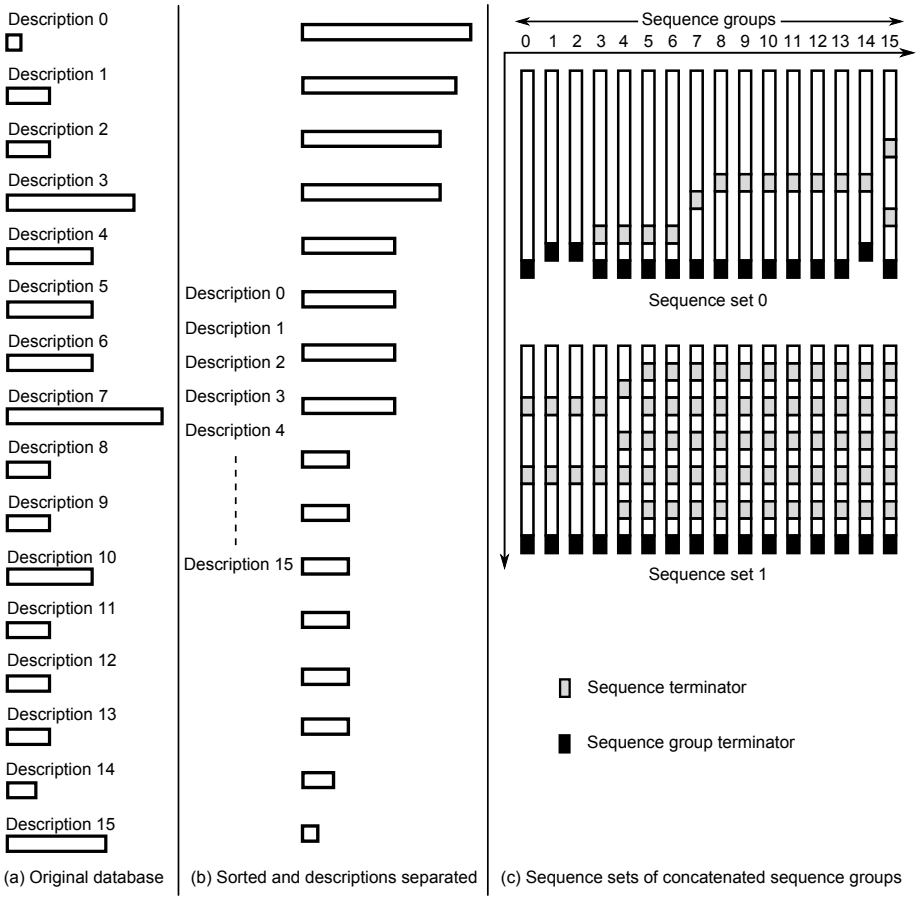


Figure 5.5: The database conversion process

database sequences are sorted by length to minimize length differences between neighboring threads, as shown in Figure 5.5(b). Sequence descriptions are stored in a separate file that is not uploaded to the GPU, saving memory and decreasing load times. Furthermore, sequence characters are replaced with numeric indexes to facilitate easier substitution matrix lookups.

Concatenation

After sorting, groups of 16 sequences are taken and processed in *sequence sets* that will have a half-warp of threads working on them, as shown in Figure 5.5(c). Even though sorting by length has somewhat equalized workload within each sequence set, various sequence sets still have different sizes. To combat this, sequences within a sequence set are concatenated with leftover sequences to form *sequence groups*. The total length of each sequence group within a sequence set nearly equals or, ideally, matches the length of the longest sequence in that set. This results in an equal workload for each thread in a half-warp processing a sequence set.

Sequence terminators are inserted between the concatenated sequences; these tell the GPU kernel to initiate a new alignment. *Sequence group terminators* are inserted at the end of each sequence group signifying the end of a group of concatenated sequences, at which point a thread will wait for the rest of the threads in the half-warp to cease execution.

Interlacing

Once all database sequences have been processed into 16-wide sets of sequence groups, they are written to file. The sequence sets are written in an interlaced fashion, as shown in Figure 5.6. Each interlaced *subset* consists of eight characters from each sequence group.

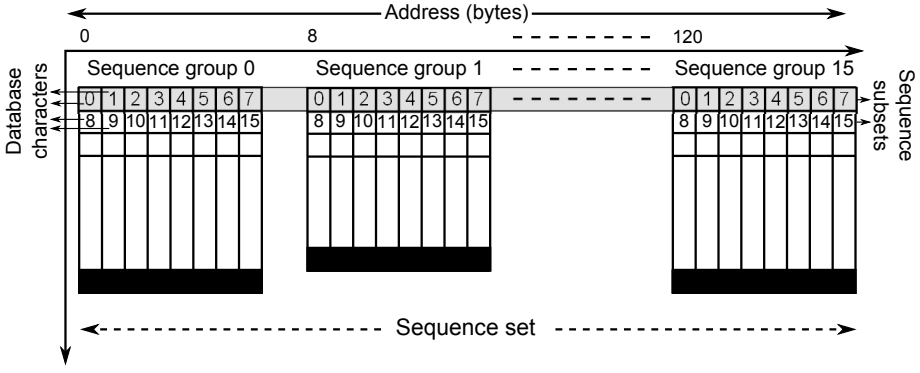


Figure 5.6: Sequence storing as interlaced subsets

Eight characters of the set's first sequence group are written, then eight characters of the set's second group and so on. As there are 16 sequence groups in each sequence set, each thread in a half-warp is now able to load 8 bytes of sequence data from neighboring addresses. As a result, 128-byte coalesced loading takes place.

Equal length sets

During code development, alignments were conducted with a synthetic (randomly generated) database, each sequence of which had the same length. The performance of this synthetic database is twice that of the Swiss-Prot database, which has sequences ranging in length from 2 to 35213 characters. The drop in performance for Swiss-Prot is the result of different workloads between different half-warps.

Though concatenation resulted in an equal workload distribution for threads within every sequence set, it still varies among different sequence sets. To resolve this, the length of each sequence group within every sequence set is made equal or nearly equal to the length of the longest sequence in the database, as shown in Figure 5.5(c). This results in an equal workload distribution for all GPU threads in general. The outcome of this is a 1.7 times increase in performance.

Evidently, equal workload across different threads improves performance; possibly a result of the GPU's thread scheduling not being optimal in the previous case. For example, the GPU thread scheduler might only schedule a new thread block once all the threads in a previous thread block have completed their execution.

5.2.3 Temporary data reads and writes

Memory bandwidth represented a serious bottleneck while developing the GPU implementation. A number of steps have been taken to optimize for high performance by reducing the number of memory accesses, the frequent temporary data accesses in particular. As no traceback is performed on the GPU, S-W matrix values do not need to be saved for the entire execution time and can be overwritten. As such, only a single column of S-W scores is kept. This score column stores values to the left of the currently processing column, i.e. $H_{i-1,1\leq j\leq N}$ in Equation 2.3. The size of this temporary data column is set to the size of the query sequence, not the database sequence, so that the column can have one fixed size for all database sequences. This usually requires less memory, as it is unlikely that the query sequence will be as long as the longest database sequence. The temporary data column is set to zero whenever a new database sequence is started. In addition to this temporary score column, variables are used to keep the values of the upper and upper-left cells required by the algorithm, i.e. $H_{i,j-1}$ and $H_{i-1,j-1}$ in Equation 2.3. To support affine gap penalties, another temporary data column is added for D values. Additionally, an upper E value is kept (see Equation 2.3).

Each S-W iteration involves reading and writing two temporary values (score and D), for four accesses in total. When both are non-coalesced, 32 byte reads/writes are

issued for each access. This means that per half-warp

$$16 \text{ threads} \times 32 \text{ bytes} \times 2 \text{ values} \times 2 \text{ read/write} = 2048 \text{ bytes}$$

of bandwidth is used, resulting in a major memory bottleneck. The optimization steps mentioned below decrease this to one 128-byte coalesced read and write for every second iteration. This is a 16 times bandwidth improvement and requires only 1 instead of 64 accesses. 128 bytes is the largest allowed coalesced access size, and is faster than multiple smaller coalesced accesses [82]. The optimizations are as follows:

- Smaller, 16-bit data type for the temporary values, cutting the theoretically required bandwidth in half and allowing for better coalescing.
- Each thread stores one data value in turn, resulting in an interlaced storage scheme. Instead of direct array accesses, a pointer into the temporary storage is started at the thread *id*, and increased by the total number of threads to move to the next element of the H matrix. Each thread in a half-warp then reads a 2-byte coalesced value, meaning that instead of two 32-byte accesses per thread, two such accesses take place per half-warp. This sixteen times bandwidth improvement results in an almost ten times net speedup.
- To again halve the number of memory accesses, the temporary score and D values are interlaced. This is done by defining a data structure consisting of these values and using it to access the score and D values for an iteration in one go. At this point, a thread accesses two 2-byte values in one read, for a total of $16 \times 2 \times 2$ bytes bandwidth per half warp. The result is a 64-byte coalesced access.
- Finally, two temporary values are interlaced to move to 128-byte accesses. This has an additional benefit of temporary reads/writes only being required for every second query sequence symbol processed.

5.2.4 Substitution matrix accesses

Aligning proteins requires the use of a substitution matrix, which is accessed every time two symbols are aligned, making its access time critical to the implementation's performance. Substitution matrix (e.g. BLOSUM 62) accesses are random and are completely dependent on the database sequence, complicating the choice of memory used. Global memory is not a good choice for such a frequent usage due to its high access time. Also the random nature of substitution matrix accesses makes coalescing very difficult. As an alternative, the substitution matrix is stored in texture memory. Texture memory is a cached window into global memory that offers lower latency and does not require coalescing for best performance. It is thus well suited for random access. Texture memory has the ability to fetch four values at a time. This mechanism can be used to fetch four substitution matrix values from a *query profile*.

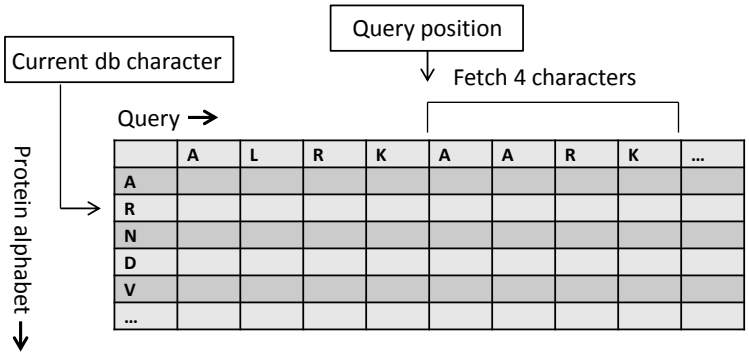


Figure 5.7: Query profile

A query profile is shown in Figure 5.7. It is a type of substitution matrix where, instead of the protein alphabet, the query sequence is used along the top row. This means that for a given database character, the substitution matrix is not random anymore: multiple substitution scores can be loaded simultaneously when aligning the query with a database character. Furthermore, query sequence lookups are not required anymore; only the current position within the query is needed to index into the profile. A query profile is generated once for every query sequence. Each query profile column stores values for 23 characters. The number of columns and hence the memory requirement for a query profile depends on the length of the query sequence. The GTX 275 GPU used for our implementation has 8KB of texture cache per multiprocessor. This means that a query sequence having more than $\lfloor 8 \times 1024/23 \rfloor = 356$ characters will result in increased cache misses, as described in [78]. Tests were performed to quantify the texture cache miss rate, which was shown to be very small. For example, aligning an 8000 character query sequence resulted in 0.009% miss rate. Using this query profile method resulted in a 17% performance improvement with Swiss-Prot [82].

5.3 Discussion of results

In this section, the performance of DOPA, the optimized GPU implementation for protein sequence alignment is evaluated and compared with other available approaches.

5.3.1 Experimental setup

The experimental setup used to test the implementation and measure its performance is as follows:

- Intel Core 2 Quad Q6600 (2.4 GHz) with 4GB of RAM

- NVIDIA Geforce GTX 275 graphics card with 896 MB of memory and clock speeds of 633, 1134 and 1404 MHz for its core, memory and shaders respectively
- 64 bit Microsoft Windows 7 Professional
- Video drivers version 257.21
- CUDA toolkit version 3.1
- Swiss-Prot release October 2010
- Substitution matrix BLOSUM62
- Gap penalty: -10 and gap extend penalty: -2 (these do not influence the execution time)

The run time is measured using the C `clock()` instruction, the accuracy of which is verified using the CUDA profiling application. Table 5.1 displays the performance results, where the execution time in seconds and the performance in GCUPS are given for query sequences of varying lengths taken from Swiss-Prot and aligned against the same database.

Table 5.1: Performance results with Swiss-Prot

Query sequence	Length	Execution time (seconds)	Performance (GCUPS)
P02232	144	1.24	21.35
P05013	189	1.65	21.06
P14942	222	1.93	21.15
P07327	375	3.24	21.28
P01008	464	3.99	21.38
P03435	567	4.89	21.32
P27895	1000	8.60	21.38
P07756	1500	12.91	21.36
P04775	2005	17.27	21.35
P19096	2504	21.54	21.37
P28167	3005	25.88	21.35
P0C6B8	3564	30.67	21.37
P20930	4061	34.97	21.35
Q9UKN1	5478	47.15	21.36

Figure 5.8(a) shows that the execution time increases linearly with sequence length, resulting in an almost constant performance of around 21.4 GCUPS, shown in Figure 5.8(b).

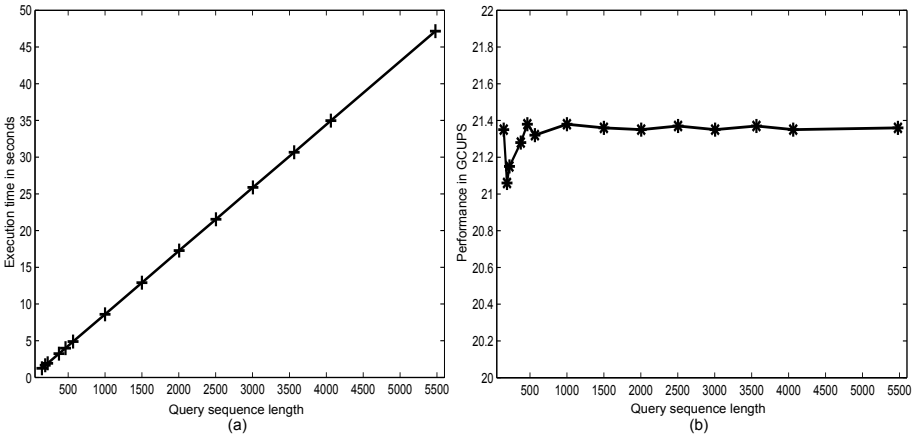


Figure 5.8: (a) Execution time (b) Performance for query sequences of varying lengths

5.3.2 Performance comparison

The optimized version of our implementation is compared with: a multi-threaded high performance *ssearch (SSE2)*; a less optimized version of our implementation with no equal length sequence sets; and with CUDASW++ 2.0 [80], the fastest GPU-based Smith-Waterman implementation to date. The comparison is shown in Figure 5.9 and described as follows.

Comparison with ssearch

Ssearch (SSE2) is an accelerated and multi-threaded version of ssearch, where ssearch is a CPU-based Smith-Waterman alignment tool that can be found in the FASTA suite of applications [83]. The SSE2 optimizations, described in [84] utilize modern CPU's vector extensions for a performance increase. The ssearch is run on the same system, using the same settings, as our GPU implementation mentioned in Section 5.3.1. The results demonstrate that our implementation performs 2.14 times better in terms of GCUPS than this accelerated and multi-threaded version of ssearch.

Comparison with a less optimized version

In the less optimized version, only some of the database optimization steps mentioned in Section 5.2.2 have been performed. In this version, sequences are only sorted, concatenated and interlaced. However, no equal length sets were used, making the length of each sequence set depend on the longest sequence within that set. When run on the same experimental setup described in Section 5.3.1, this less optimized version results in a performance of around 12.5 GCUPS. The comparison shows that our fully optimized GPU implementation performs around 1.7 times better than the less opti-

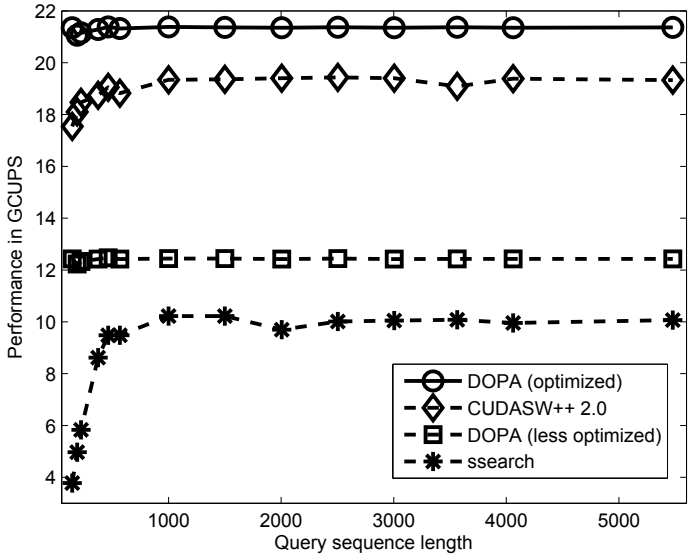


Figure 5.9: Performance comparison

mized version. This demonstrates the performance impact of the crucial optimization of equal length sequence sets, which results in an improved scheduling.

Comparison with CUDASW++ 2.0

CUDASW++ 2.0 is the fastest GPU implementation to date for S-W based protein sequence alignment. When run on the same system with the same settings, as mentioned in Section 5.3.1, CUDASW++ 2.0 achieves a performance of around 19 GCUPS. Thus our fully optimized implementation performs 1.13 times better than CUDASW++ 2.0 in terms of GCUPS. Both approaches are sensitive to the structure of the database used. Like our implementation, CUDASW++ 2.0 also uses 16-bit score values, as discussed in Section 5.2.3. Table 5.2 summarizes the optimization steps undertaken by our fully optimized implementation called DOPA in comparison with CUDASW++ 2.0.

Additionally, DOPA also brings in the following improvements:

- In comparison with CUDASW++ 2.0, DOPA is simpler, as it uses just one search kernel instead of two, requiring no inter-processor communication.
- The optimized database organization scheme used in DOPA allows an equal workload for each thread block, while CUDASW++ 2.0 uses a hand-picked point at which it switches from one kernel to the other for its work distribution.
- DOPA is complete and usable, as it exports the top scoring sequences for full

Table 5.2: A comparison with CUDASW++ 2.0

#	Optimization	DOPA	CUDASW++ 2.0
1	Database sorting	+	+
2	Concatenation into sequence groups	+	−
3	Interlacing	+	+
4	Equal length sequence sets	+	−
5	Query profile	+	+

alignment with *ssearch*. CUDASW++ 2.0 does not provide this facility. Our implementation also provides a web interface that allows it to be used conveniently and remotely.

In comparison with CUDASW++ 2.0, our less optimized implementation performs 1.52 times slower in terms of GCUPS, as shown in Figure 5.9. This is because CUDASW++ 2.0 switches to its secondary systolic array based alignment stage for long sequences. Long sequences in a database inherently have the largest length differences, specifically true for Swiss-Prot. Thus, aligning them using systolic array based approach reduces the workload differences.

5.4 Performance limits

This section explores the maximum performance limits, scalability and future prospects of our GPU-based design for protein sequence alignment.

5.4.1 Limits/bottlenecks

The optimizations mentioned in Section 5.2 eliminate various performance bottlenecks. Below, we show practical performance limits/bottlenecks to give an impression of the maximum achievable performance.

- **Database layout:** With a synthetic test database containing same length sequences, performance increases to 24 GCUPS, whereas the practical performance with Swiss-Prot stays at 21 GCUPS. This is the result of overhead factors such as processing the sequence group terminators, and the fact that the synthetic database can be created in such a way that the number of database blocks matches the number of thread half-warps.

- **Memory bandwidth:** The maximum theoretical memory bandwidth for the GTX 275 GPU is $127 GB/sec$ [85]. During benchmarking with the test database about $50 GB/sec$ of bandwidth is used in practice. This can be interpreted in two ways. The maximum possible bandwidth is not utilized; however, on the other

hand, due to proper coalescing no more data is transferred than strictly required. In any case, memory bandwidth is not a bottleneck anymore.

Not just pure bandwidth determines performance due to memory factors, but latency is an issue too. Many small sequential transfers will be slower than a single larger one. To further investigate the practical effect of memory accesses on performance, the saving and loading of temporary values (the most frequent memory accesses) are commented out. This resulted in a performance of 25.5 GCUPS with the test database, only a slight increase, verifying that memory accesses are not a limiting factor anymore.

- **Arithmetic throughput:** With memory not being a limiting factor, arithmetic performance is a likely candidate. To test this, the actual S-W formula is commented out from the kernel code. This resulted in a performance of 50 GCUPS with the synthetic test database, thereby showing that arithmetic throughput imposes the actual limit on the practical performance. Also, for the original code, the CUDA profiler reports an instruction throughput of 1, which means that instructions are issued at the maximum possible speed.

5.4.2 Scalability/future prospects

The concatenation of sequences into database blocks means that many sequences are combined to match the length of the longest sequence in the database. In other words, the longer the longest database sequence is, the fewer database blocks there will be. With the current releases of Swiss-Prot running on a GTX 275 GPU, this results in a number of launched threads having no database blocks to process.

The number of sequence blocks will increase with the Swiss-Prot database's growth, as in turn will the number of threads that have work to do. This will increase future performance effectively for free. However, two caveats apply. First of all, if the number of sequence groups grows to more than the number of GPU half-warps launched, some processing elements will have to perform multiple alignments, resulting in unequal work between half-warps and, as such, a performance penalty. However, this issue can be curtailed by increasing the size of each database sequence block by concatenating more sequences to each group, lowering the amount of sequence blocks needed. The second issue is the opposite, and arises if the longest sequence in the database were to grow. All blocks would grow larger, resulting in less blocks to spread work across. However, this happening is unlikely, as long sequences are rare; the current longest sequence is significantly longer than the second-longest one, it is not in danger of being overtaken. In support of all this, a performance increase of 3% is achieved by updating from the August to the October release of Swiss-Prot, which contains 1668 more sequences and results in one additional sequence block being created. Further evaluating the workings of the GPU's thread scheduling might allow these factors to be optimized further, decreasing the dependence on database structure.

The GPU-based S-W implementation is optimized for GT200-series GPUs. Although it will run just fine on newer GPUs such as the Geforce 400 series, performance

may not be optimal. These newer GPUs offer many more processors, which might require the workload distribution to be re-evaluated. They also run two half-warps at a time, and offer a cache hierarchy. This means that memory layouts might need to be changed, or that for example texture memory is not the best option to store a query profile anymore. Furthermore, some instructions perform differently. For example a 24-bit integer multiplication is slower, not faster, than 32-bit one on these GPUs due to architectural reasons.

5.5 Summary

Besides providing an introduction to GPUs, this chapter exploited the parallelization capabilities of GPUs for biological sequence alignments. It presented an optimized GPU implementation for S-W based protein sequence alignment and compared its performance with the best available similar design. The main topics presented in the chapter are as follows.

- General purpose computing on GPUs, a discussion about CUDA, its programming and memory models.
- A discussion about coalescing which is used to reduce latency of the global memory.
- A review of GPU-based sequence alignment.
- An optimized GPU implementation for S-W based protein sequence alignment.
- Optimization steps taken for improving performance of the implementation, such as optimizing the database conversion, temporary data reads and writes, and substitution matrix accesses.
- A discussion of results, performance evaluation and comparison with other available approaches.
- A discussion of the maximum achievable performance, practical performance limits and bottlenecks, scalability and future prospects.

Chapter 6

Performance Analysis

Performance of hardware-based sequence alignment depends on various parameters, such as computational resources and bandwidth. This chapter carries out a detailed performance analysis and proposes optimizations resulting in enhanced performance and efficient resource utilization. The chapter is organized as follows:

Section 6.1 provides theoretical performance boundaries. Section 6.2 presents performance limitations based on computational resources and bandwidth. Section 6.3 presents performance and bandwidth optimization. Section 6.4 introduces a method based on hardware partitioning to improve performance, whereas Section 6.5 generalizes the method. Section 6.6 summarizes the chapter.

6.1 Theoretical performance boundaries

Performance of the hardware-based sequence alignment depends on the available computational resources, i.e. the number of PEs. Hardware platforms like FPGAs offer abundant hardware resources, sufficient for fitting large number of PEs. Therefore, for some applications including sequence alignment, the maximum performance is limited by the available memory bandwidth. The more the memory bandwidth, the more the overall performance gain. Figure 6.1 shows a system model for the S-W based sequence alignment of two sequences, i.e. the query sequence (N_q) and the database sequence (N_s). The upper long horizontal bar in the figure represents N number of PEs, i.e. (N_{PEs}). The lower part of the figure represents the memory requirement (in terms of data width and depth), where the data width is shown for 1 PE. The width for N_q and N_s are both 5 bits, as with 5 bits we can cover alphabets for both DNA and protein sequences [3,86]. The depth for both N_q and N_s is N, based on the assumption that both the query and database sequences are of the same length, which is equal to the number of PEs N. The width for each element of the substitution ma-

trix is 5 bits, assuming that the substitution matrix under consideration is Blosum62. In Blosum62, there are both positive and negative values, with the maximum positive substitution value as 11. So 4 bits are enough for storing the magnitude and 1 bit for sign. The depth for the substitution matrix (Blosum62) is 20. The width for the scoring matrix is 16 bits [70]. The depth for the scoring matrix is $2N - 1$, as there are $2N - 1$ computational steps and the output of each one has to be written in the scoring matrix.

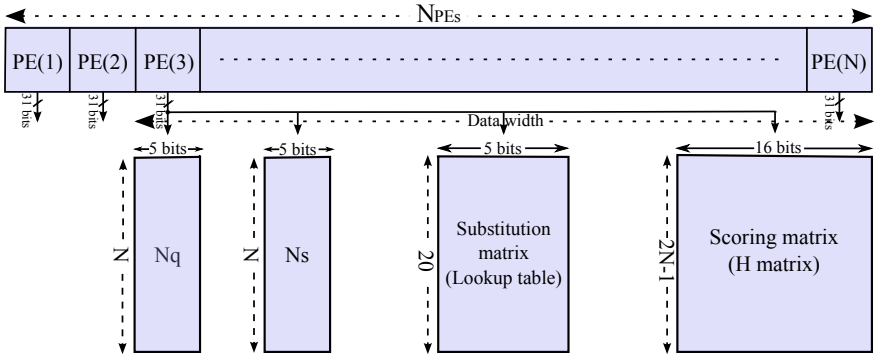


Figure 6.1: System model for the S-W based sequence alignment

The total execution time T_{exec} for the hardware-based S-W design is given by the following equation:

$$T_{exec} = T_{compute} + T_{access} \tag{6.1}$$

where $T_{compute}$ is the total computation time and T_{access} is the total time to load/store data in memory.

For a square scoring matrix (i.e. when N_q and N_s are of the same length), the number of steps to compute different matrix cells is as shown in Figure 6.2(a). Since the elements in each anti-diagonal are computed in parallel, the computation time is given by,

$$T_{compute} = (2N - 1) \times T_{PE} \tag{6.2}$$

where N is the number of PEs, such that, the total number of steps needed for the entire computation is $(2N - 1)$. T_{PE} is the computation time for 1 step, such that,

$$T_{PE} = C_{PE} \times T_{cycle}$$

where C_{PE} is the number of cycles consumed by 1 PE and T_{cycle} is the time for 1 cycle. T_{PE} is equal to the computation time for 1 PE, as the PEs utilized during each step are computed in parallel. The total time to transfer data to the main memory is given by,

$$T_{access} = \frac{D_{main}}{B_{main}}$$

where D_{main} is the total amount of data that needs to be stored in the main memory and B_{main} is the bandwidth of the main memory in bits/sec, such that,

$$D_{main} = 16N_s \times N_q$$

where the precision of each output is 16 bits wide. For a square matrix, the total data becomes,

$$D_{main} = 16N^2$$

So,

$$T_{access} = \frac{16N^2}{B_{main}} \tag{6.3}$$

Substituting Equation 6.2 and 6.3 in Equation 6.1,

$$T_{exec} = \frac{(2N-1) \times T_{PE} \times B_{main} + 16N^2}{B_{main}} \tag{6.4}$$

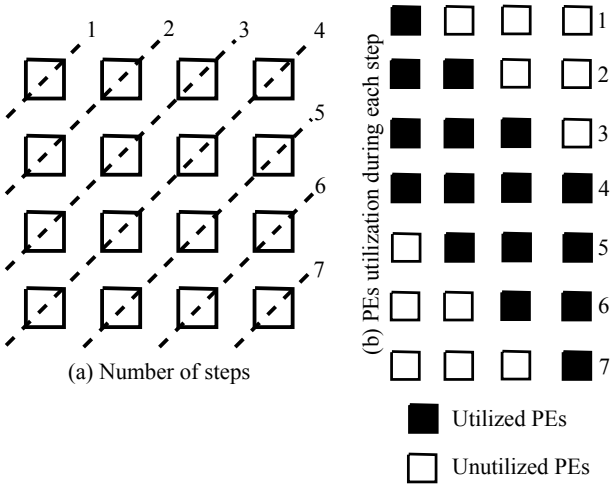


Figure 6.2: Number of steps and PEs utilization during each step for $N = N_q = N_s$

Now, the overall performance is given by the ratio of the total number of matrix fill operations to the total execution time, i.e.

$$\text{Overall Performance} = P_{overall} = \frac{\text{Total operations}}{T_{exec}}$$

$$P_{overall} = \frac{N^2}{\frac{(2N-1)\times T_{PE}\times B_{main}+16N^2}{B_{main}}}$$

$$P_{overall} = \frac{N^2 \times B_{main}}{(2N-1) \times T_{PE} \times B_{main} + 16N^2} \tag{6.5}$$

If the number of PEs is less than the size of the query sequence, then we need to partition the query sequence, so that the computation takes place in k passes, where $k \geq 1$ is an integer. In this case the total time becomes,

$$T_{total} = k \times T_{exec}, \text{ where } k = \left\lceil \frac{N_q}{N} \right\rceil$$

and the total operations become $k \times N^2$.

From Equation 6.5, the parameters that can limit the performance of hardware-based sequence alignment are:

- N : The amount of available hardware computational resources, i.e. the number of PEs.
- B_{main} : The memory bandwidth.
- T_{PE} : The computation time for a step.

6.2 Performance limitations

This section presents performance analysis based on limitations in computational resources and bandwidth.

6.2.1 Performance limited by the computational resources

Assume that we have infinite bandwidth and the performance is only limited by the computational resources, then,

$$P_{compute} = f(T_{PE}, N)$$

where $P_{compute}$ is the performance limited by the computational resources, T_{PE} is the computation time for 1 step and N is the number of PEs. In this case, $T_{access} \cong 0$, so,

$$T_{exec} = T_{compute} = N_{steps} \times T_{step} \tag{6.6}$$

where, N_{steps} is the number of anti-diagonals and T_{step} is the time taken by each anti-diagonal. Now performance, limited by the computational resources, is given by,

$$P_{compute} = \frac{N_q \times f_{op}}{C_{PE}} \times \text{Utilization ratio} \tag{6.7}$$

where C_{PE} is the number of cycles consumed by 1 PE and *Utilization ratio* is the ratio of *utilized* to *available* computational resources. Performance may also be given by,

$$P_{compute} = \frac{\text{Total operations}}{T_{exec}} \tag{6.8}$$

For the sake of clarity, the analysis of T_{exec} and utilization ratios is divided into four sub cases, as follows:

$N = N_q = N_s$

Figure 6.2 shows the number of steps and the PEs utilized during each step for this case. In Figure 6.2(a), each anti-diagonal represents a computational step, such that there is a total number of $2N - 1$ steps. Since all the PEs in each step (each anti-diagonal) are processed in parallel, therefore, $T_{step} = T_{PE}$. Hence Equation 6.6 becomes,

$$T_{exec}|_{N=N_q=N_s} = (2N - 1) \times T_{PE} \tag{6.9}$$

For the given example in Figure 6.2, $N = N_q = N_s = 4 \Rightarrow 2N - 1 = 7$. In Figure 6.2(b), each row represents the number of PEs available in each step, whereas the solid black cells represent the number of PEs utilized. So,

$$\text{Utilization ratio} = \frac{\text{PEs utilized}}{\text{PEs available}} = \frac{N_q^2}{N \times N_{steps}} \tag{6.10}$$

where, $N_{steps} = 2N - 1$

$N < (N_q = N_s)$

Figure 6.3 shows the number of steps and the PEs utilized during each step for the case where $N < (N_q = N_s)$. For example, it is shown that $N = 3$ and $N_q = N_s = 4$. In Figure 6.3(a), each anti-diagonal represents a computational step, except anti-diagonal 4, which is partitioned into 2 computational parts. The reason for this is that 4 cells can't be computed simultaneously with the three available PEs. The solid thick and tilted small line represents the partition of the steps along this anti-diagonal, such that the top gray cell is computed during the 2nd part. The total number of steps in this case is,

$$\sum_{i=1}^{2N_q-1} \left\lceil \frac{min(i,\ 2N_q - i)}{N} \right\rceil = \sum_{i=1}^{7} \left\lceil \frac{min(i,\ 8 - i)}{3} \right\rceil = 8$$

The execution time for this case is,

$$\begin{aligned} T_{exec}|_{N<(N_q=N_s)} &= N_{steps} \times T_{step} \\ &= \left(\sum_{i=1}^{2N_q-1} \left\lceil \frac{min(i,\ 2N_q - i)}{N} \right\rceil \right) \times T_{PE} \end{aligned} \tag{6.11}$$

In Figure 6.3(b), each row represents the number of PEs available in each step, whereas the solid black cells represent the number of PEs utilized.

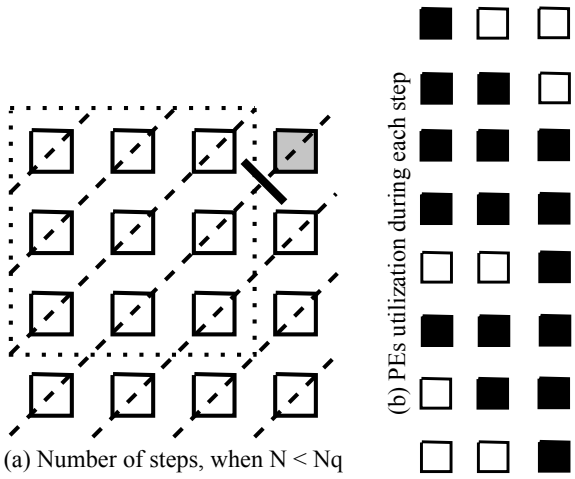


Figure 6.3: Number of steps and PEs utilization during each step for $N < (N_q = N_s)$

Resource utilization for this case is also given by Equation 6.10, where,

$$N_{steps} = \sum_{i=1}^{2N_q-1} \left\lceil \frac{min(i,\ 2N_q - i)}{N} \right\rceil$$

$N = N_q < N_s$

Figure 6.4(a) depicts a case, where the number of PEs is the same as the length of the query sequence i.e. $N = N_q = 4$ and the length of the database sequence is larger than the number of PEs i.e. $N_s = 16$. Equation 6.6 for this case becomes,

$$T_{exec}|_{N=N_q<N_s} = (N_s + N - 1) \times T_{PE} \tag{6.12}$$

The utilization ratio is given by Equation 6.13, where the utilization is dependent on the $\frac{N-1}{N_s}$ term. The lower the $\frac{N-1}{N_s}$ term, the more efficient the hardware resource utilization.

$$\begin{aligned} \text{Utilization ratio}|_{N=N_q<N_s} &= \frac{N_s \times N}{(N_s + N - 1) \times N} \\ &= \frac{1}{1 + \frac{N-1}{N_s}} \end{aligned} \tag{6.13}$$

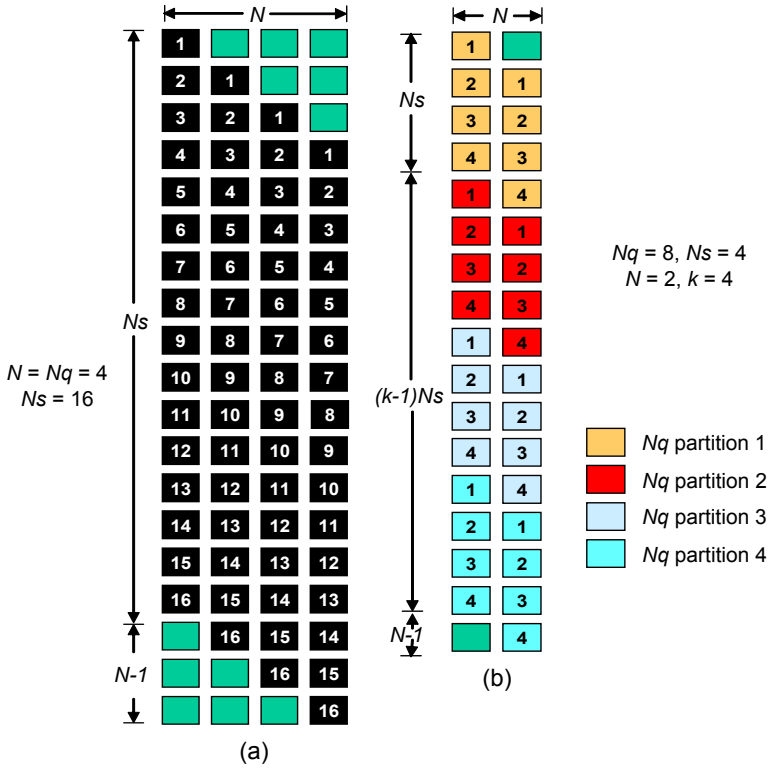


Figure 6.4: Number of steps and PEs utilization (a) $N = N_q < N_s$ (b) $N < N_s < N_q$

$N < N_s < N_q$

Figure 6.4(b) depicts the case, when $N < N_s < N_q$. In this case, we partition N_q into k parts, where the size of each part is $N_q^{'}$, such that $N_q^{'} = N$. In other words, we scale down N_q to the size of N and perform multiple (k) passes instead, where $k = \left\lceil \frac{N_q}{N_q^{'}} \right\rceil$. This approach is referred to as *Query Sequence Partitioning (QSP)*. For the given example, $N_q = 8,\ N_s = 4,\ N_q^{'} = N = 2$ and $k = \left\lceil \frac{8}{2} \right\rceil = 4$

The execution time for this case becomes,

$$\begin{aligned} T_{exec}|_{k>1,\ N<N_s<N_q} &= (N_s + (k-1) \times N_s + N - 1) \times T_{PE} \\ &= (k \times N_s + N - 1) \times T_{PE} \end{aligned} \tag{6.14}$$

The utilization ratio for this case is given by Equation 6.15, where the utilization is dependent on the $\frac{N-1}{k \times N_s}$ term. The lower the $\frac{N-1}{k \times N_s}$ term, the more efficient the hardware

resource utilization.

$$\begin{aligned} \text{Utilization ratio}|_{k>1,\ N<N_s<N_q} &= \frac{k \times N_s \times N}{(k \times N_s + N - 1) \times N} \\ &= \frac{1}{1 + \frac{N-1}{k \times N_s}} \end{aligned} \tag{6.15}$$

Table 6.1 presents the corresponding calculated values of T_{exec} for various combinations of k and N (as per Equation 6.14), where, $T_{PE} = 10\ ns$ and $N_q = N_s = 500$. The table shows that if we have the same number of processing elements as the size of the query sequence, i.e. $N = N_q = 500$, then the computation takes place in one pass, i.e. $k = 1$, which completes in 9.99 μsec. But if the number of PEs available are half the size of the query sequence, i.e. $N = \frac{1}{2}N_q$, then the computation completes in two passes, i.e. $k = 2$, that takes 12.49 μsec. Note that by halving the number of PEs, the execution time is increasing only by 25%, however, using half of the resources requires half of the bandwidth for data transfer.

Table 6.1: Execution time (T_{exec}) in μsec for various combinations of k and N

N, k	**T_{exec}**	**N, k**	**T_{exec}**	**N, k**	**T_{exec}**
$N = 1$ $k = 500$	2500	$N = 10$ $k = 50$	250.09	$N = 100$ $k = 5$	25.99
$N = 2$ $k = 250$	1250	$N = 20$ $k = 25$	125.19	$N = 125$ $k = 4$	21.24
$N = 4$ $k = 125$	625	$N = 25$ $k = 20$	100.24	$N = 250$ $k = 2$	12.49
$N = 5$ $k = 100$	500	$N = 50$ $k = 10$	50.49	$N = 500$ $k = 1$	9.99

Figure 6.5 shows the T_{exec} versus number of PEs (N) curve, limited by the computational resources, where, T_{exec} decreases with the increasing number of PEs (N).

6.2.2 Performance limited by the bandwidth

Assume that we have infinite computational resources, i.e. zero computation time, then, $T_{exec} = T_{access}$, as, $T_{compute} \cong 0$

From Equation 6.3,

$$T_{access} = \frac{16N^2 \text{ (bits)}}{B_{main} \text{ (Mbps)}} = \frac{2N^2 \text{ (Bytes)}}{B_{main} \text{ (MBps)}} \tag{6.16}$$

where, *Mbps* is the bandwidth in *Mega bits per second* and *MBps* is the bandwidth in *Mega Bytes per second*. Now, performance limited by the bandwidth is,

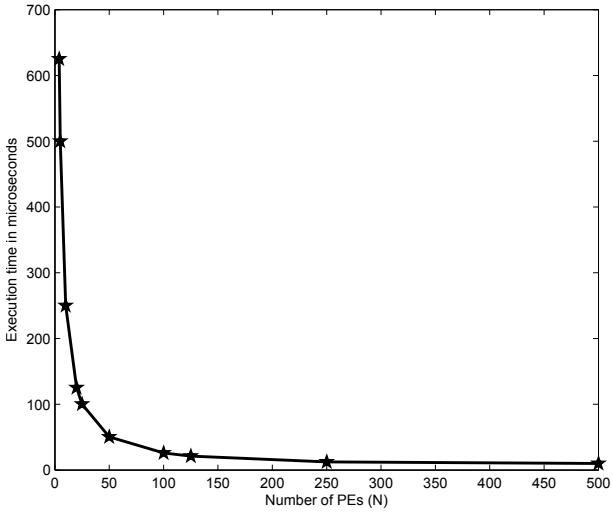


Figure 6.5: T_{exec} vs N curve, limited by the computational resources

$$P_{bandwidth} = \frac{\text{Total operations}}{T_{exec}} = \frac{N^2}{\frac{2N^2}{B_{main}}} = \frac{B_{main}}{2} \tag{6.17}$$

Table 6.2: Execution time (T_{exec}) in μ sec for various combinations of N and B_{main}

B_{main} \ N	500	250	125	100	50	25	20	10	5	4	2	1
100	5000	1250	312	200	50	12.5	8	2	0.5	0.32	0.08	0.02
200	2500	625	156	100	25	6.25	4	1	0.25	0.16	0.04	0.01
300	1667	417	104	67	17	4.17	2.7	0.67	0.17	0.1	0.03	0.007
400	1250	312	78	50	12	3.12	2	0.5	0.12	0.08	0.02	0.005
500	1000	250	62	40	10	2.5	1.6	0.4	0.1	0.06	0.016	0.004
600	833	208	52	33	8	2.1	1.3	0.33	0.08	0.05	0.013	0.0033
700	714	178	44	28	7	1.8	1.14	0.28	0.07	0.046	0.011	0.0029
800	625	156	39	25	6.2	1.6	1	0.25	0.06	0.04	0.01	0.0025
900	556	139	34	22	5.6	1.4	0.89	0.22	0.056	0.036	0.0089	0.0022
1000	500	125	31	20	5	1.25	0.8	0.2	0.05	0.032	0.008	0.002

Table 6.2 gives the execution time in (μsec), for various combinations of the number of PEs (N) and the bandwidth (B_{main}) in $MBps$. Figure 6.6(a) gives the execution time versus bandwidth curves for various values of N. The curves show that the execution time (calculated as per Equation 6.16) decreases with the increasing bandwidth, where the execution time is equal to the memory access time, as the computational time is nearly zero.

For a limited bandwidth, the execution time increases with the increasing length of the query sequence. Figure 6.6(b) shows the T_{exec} versus N curve for a case, where the limited bandwidth is 500 $MBps$ and the number of PEs (N) varies from 1 to 500. The execution time (calculated as per Equation 6.16), has a quadratic dependence on N, which causes it to increase rapidly for higher N values. In the next section, we

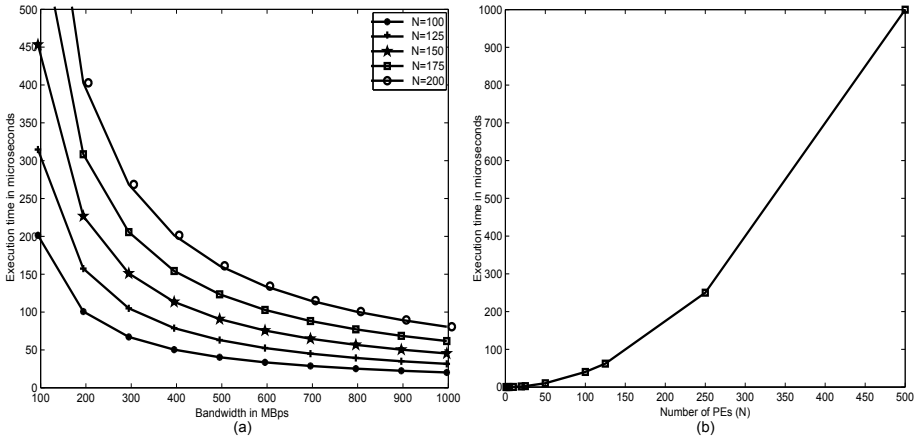


Figure 6.6: Performance limited by bandwidth (a) T_{exec} vs bandwidth (b) T_{exec} vs N

investigate the minimum execution time that gives optimum performance with reduced bandwidth requirement.

6.3 Performance and bandwidth optimization

In this section, performance gain and bandwidth requirements are optimized and a generalized equation is developed for the execution time that considers both the computational resources and bandwidth limitations. Figure 6.7 shows the T_{exec} versus N design trade off curves for the following three cases, considering $B_{main} = 500\ MBps$ and $T_{PE} = 10\ ns$.

- When performance is limited by the computational resources
- When performance is limited by the bandwidth
- When performance is limited by both the computational resources and bandwidth

T_{exec} decreases with the increasing number of N along the T_{exec} vs N curve, limited by the computational resources and based on Equation 6.14. Decreased T_{exec} results in improved performance, but the bandwidth requirement also increases as a consequence. On the other hand, for a particular available bandwidth, T_{exec} increases with the increasing number of PEs along the T_{exec} vs N curve, limited by the bandwidth and based on Equation 6.16. The T_{exec} vs N optimization curve represents the total execution time, considering both computational resources and bandwidth limitations and is based on the following equation.

$$T_{exec} = (k \times N_s + N - 1) \times T_{PE} + \frac{2N^2}{B_{main}} \tag{6.18}$$

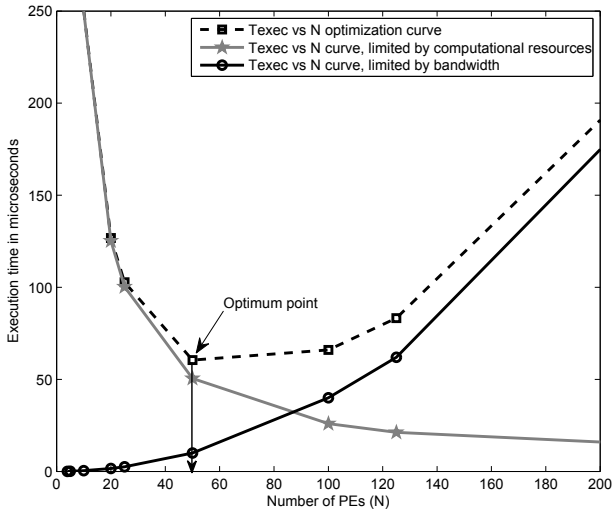


Figure 6.7: T_{exec} vs N design trade off curves

To find the N value, at which the function T_{exec} is minimum along the T_{exec} vs N optimization curve, we differentiate Equation 6.18 w.r.t. N.

$$\frac{d(T_{exec})}{dN} = \frac{d}{dN}\left[(k \times N_s + N - 1) \times T_{PE} + \frac{2N^2}{B_{main}}\right]$$

where, $k = \frac{N_q}{N'_q} = \frac{N_q}{N}$, so,

$$\begin{aligned}\frac{d(T_{exec})}{dN} &= \frac{d}{dN}\left[(\frac{N_q \times N_s}{N} + N - 1) \times T_{PE} + \frac{2N^2}{B_{main}}\right] \\ &= \frac{4N^3 + T_{PE} \times B_{main} \times N^2 - T_{PE} \times B_{main} \times N_q \times N_s}{N^2 \times B_{main}}\end{aligned}$$

Now, to find the N value, at which T_{exec} is minimum along the T_{exec} vs N optimization curve, we equate $\frac{d(T_{exec})}{dN}$ to zero, so that,

$$4N^3 + T_{PE} \times B_{main} \times N^2 - T_{PE} \times B_{main} \times N_q \times N_s = 0 \tag{6.19}$$

The discriminant of Equation 6.19 is,

$$\Delta = 4T_{PE}^2 \times B_{main}^2 \times N_q \times N_s(T_{PE}^2 \times B_{main}^2 - 108N_q \times N_s)$$

There are two cases, i.e.

1. $\Delta > 0$, if $N_q \times N_s < \frac{T_{PE}^2 \times B_{main}^2}{108}$,
   which does not take place in practice. Therefore, this case is not taken into consideration.
2. $\Delta < 0$, if $N_q \times N_s > \frac{T_{PE}^2 \times B_{main}^2}{108}$,
   which implies that Equation 6.19 has a unique positive real solution which is given as,

$$N = \frac{A^2 - 3T_{PE} \times B_{main}}{6A} \tag{6.20}$$

$$\text{where, } A = \sqrt[3]{27T_{PE}B_{main}N_qN_s + 3\sqrt{3T_{PE}^3B_{main}^3 + 81T_{PE}^2B_{main}^2N_q^2N_s^2}}$$

For a given bandwidth, Equation 6.20 gives the N value, at which the function T_{exec} is minimum along the T_{exec} vs N optimization curve. The minimum T_{exec} value guarantees an optimum performance, as any performance gain due to increasing number of PEs beyond this point is counterbalanced by the bandwidth limitation. As an example, if, $T_{PE} = 10\ ns$, $B_{main} = 500\ MBps$ and $N_q = N_s = 500$, then, the value of N, as computed per Equation 6.20 would be $N = 67.8$. This means that $N = 68$ guarantees an optimum performance for the given example. Therefore, any further increase in the number of PEs will result in subsequent performance loss due to bandwidth limitation.

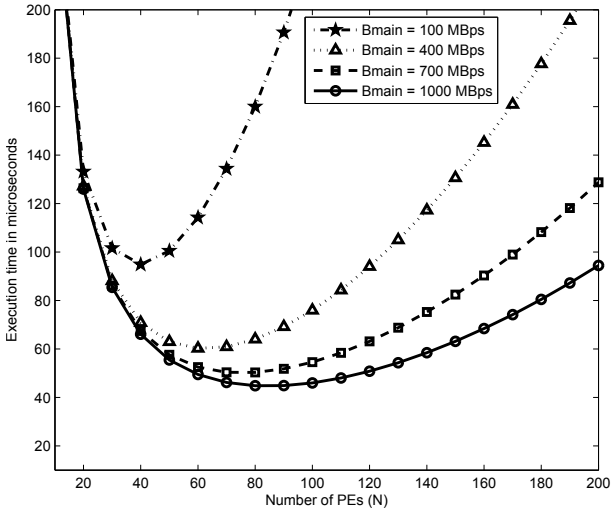


Figure 6.8: T_{exec} vs N optimization curves

Figure 6.8 shows the optimization curves for various values of B_{main} in $MBps$ and $T_{PE} = 10\ ns$, where the optimum point shifts towards higher N values for increasing

bandwidth. This implies that for higher available bandwidth a higher value of N can be used to improve the performance further.

6.4 Hardware partitioning

In this section, we present a novel method based on hardware partitioning to reduce the execution time and improve the resource utilization of S-W based sequence alignment, resulting in a higher performance as compared to conventional approaches. The method reduces the execution time and improves the resource utilization by up to 33.3%. Further, equations are developed, showing the general trend of execution time reduction, resource utilization improvement and hence performance enhancement.

6.4.1 Theoretical concept

A parallelized S-W algorithm requires $N_s + N_q - 1$ operations for computing the entire $H_{i,j}$ matrix [29]. Since every operation performed by one S-W PE takes time T_{PE}, the total execution time is given by,

$$T_{exec} = (N_s + N_q - 1) \times T_{PE}$$

where, $T_{PE} = C_{PE} \times T_{cycle}$, such that C_{PE} is the number of cycles consumed by 1 PE and T_{cycle} is the time for 1 cycle.

If two query sequences (N_{q1} and N_{q2}) need to be aligned one after the other against the same database sequence (N_s), as shown in Figure 6.9(a),

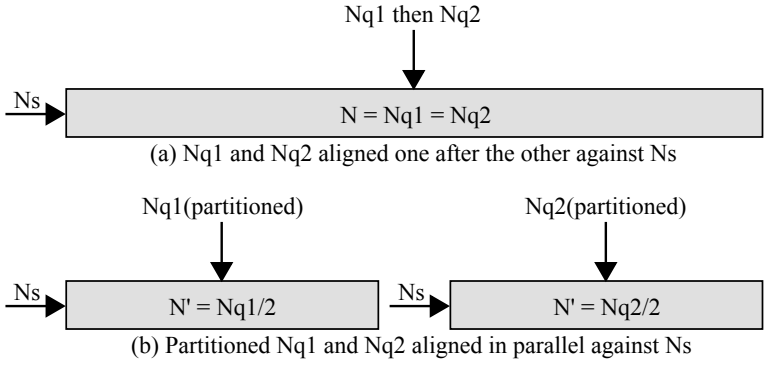


Figure 6.9: 2-sequence alignment (a) Sequential (b) Partitioned and in parallel

then the execution time becomes,

$$T_{exec}1 = (2N_s + N - 1) \times T_{PE} \tag{6.21}$$

where $N_{q1} = N_{q2} =$ The nubmer of PEs (N)

The resource utilization ratio for this case is,

$$\begin{aligned} \text{Utilization ratio} &= \frac{\text{PEs utilized}}{\text{PEs available}} \\ &= \frac{2N_s \times N}{(2N_s + N - 1) \times N} = \frac{1}{1 + \frac{N-1}{2N_s}} \end{aligned} \tag{6.22}$$

Figure 6.9(b) shows that each query sequence is partitioned into two parts and is processed in two passes, in parallel with the other. The number of PEs utilized by each query sequence is half its size, and is given as, $N^{'} = \frac{N_{q1}}{2} = \frac{N_{q2}}{2} = \frac{N}{2}$. The execution time for this case is given by,

$$T_{exec}2 = (2N_s + N^{'} - 1) \times T_{PE} \tag{6.23}$$

The resource utilization ratio for this case is,

$$\begin{aligned} \text{Utilization ratio} &= \frac{2N_s \times N^{'}}{(2N_s + N^{'} - 1) \times N^{'}} \\ &= \frac{1}{1 + \frac{N^{'}-1}{2N_s}} \end{aligned} \tag{6.24}$$

6.4.2 Example of the process

Figure 6.10 shows an example, where,

$N_{q1} = N_{q2} = N_s = N = 4$, $N^{'} = 2$, and $T_{PE} = 10ns$

Figure 6.10(a) depicts the case, where two query sequences (N_{q1} and N_{q2}) are aligned one after the other, against the same database sequence (N_s). The solid black cells in Figure 6.10(a) represent the data flow and PEs utilization for N_{q1}, whereas the light gray cells for N_{q2}. In Figure 6.10(b), the hardware is partitioned in two equal parts such that the two query sequences are aligned in parts and in parallel with each other, against the same database sequence (N_s). The solid black cells in Figure 6.10(b) represent the data flow and PEs utilization for N_{q1}, whereas the light gray cells for N_{q2}.

$$\begin{aligned} \%\ \text{time reduction} &= \frac{T_{exec}1 - T_{exec}2}{T_{exec}1} \\ &= \frac{(2N_s + N - 1) \times T_{PE} - (2N_s + N^{'} - 1) \times T_{PE}}{(2N_s + N - 1) \times T_{PE}} \\ &= \frac{N - N^{'}}{2N_s + N - 1} \end{aligned} \tag{6.25}$$

Substituting values in Equation 6.25 results in 18.18% reduction in the execution time.

Substituting values for the given example in Equation 6.22,

Utilization ratio = $\frac{1}{1+\frac{4-1}{2\times4}} = 0.73 = 73\%$

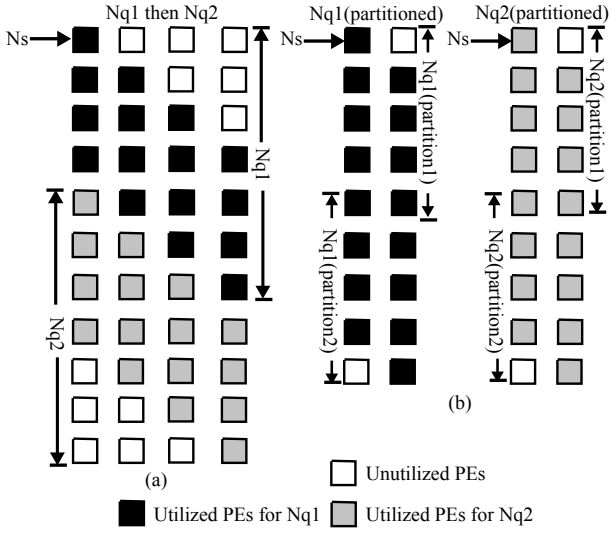


Figure 6.10: 2-sequence alignment example

Similarly, substituting the same values in Equation 6.24,

Utilization ratio = $\frac{1}{1+\frac{2-1}{2\times4}}$ = 0.89 = 89%

Thus 16% better resource utilization ratio is achieved by applying the hardware partitioning method.

In practice, the lengths of the query and subject sequences are 500 characters long in most cases [70]. To evaluate a practical case, consider $N_{q1} = N_{q2} = N_s = N = 500$, $N^{'} = 1$. Substituting these values in Equation 6.25, a 33.3% reduction in the execution time is achieved. To evaluate the resource utilization improvement, the values are substituted in Equations 6.22 and 6.24, showing thereby an improvement of 33.3% in resource utilization.

6.5 Generalizing the hardware partitioning method

To generalize the idea, consider P number of query sequences that needs to be aligned against the same database sequence (N_s), such that the length of each query sequence is equal to the number of available PEs (N). Figure 6.11(a) depicts the case, where P query sequences are aligned one after the other against N_s. Here $1 \leq P \leq i$, such that $i > 1$ is an integer. Figure 6.11(b) shows that the hardware is partitioned into Q parts such that P query sequences are aligned in parts and in parallel with the others, against the same N_s. The number of PEs utilized by each query sequence in this case is, $N^{'} = \frac{N_{q1}}{Q} = \frac{N_{q2}}{Q} = ... = \frac{N_{qi}}{Q} = \frac{N}{Q}$

The execution time becomes,

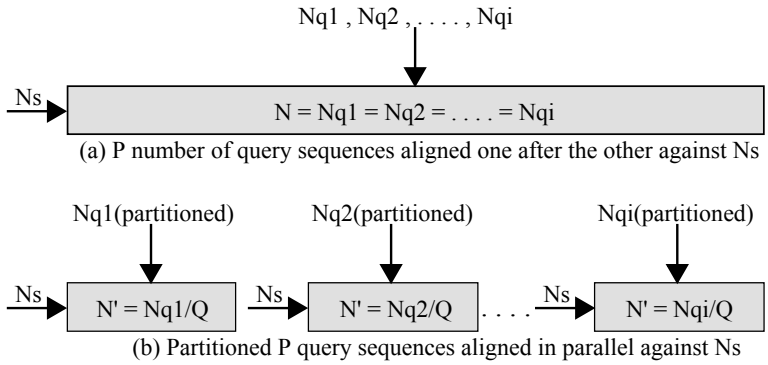


Figure 6.11: *P*-sequence alignment (a) Sequential (b) Partitioned and in parallel

$$T_{exec} = (P \times N_s + N^{'} - 1) \times T_{PE} \tag{6.26}$$

Table 6.3 shows the execution time in microseconds for various number of query sequences (Ps) and possible number of hardware partitions (Qs). The execution time is computed as per Equation 6.26, where $N_q = N_s = 500$ and $T_{PE} = 10\ ns$. The table demonstrates that the execution time decreases with the increasing number of hardware partitions (Qs), for all Ps.

Table 6.3: Execution time (T_{exec}) in μ sec for various (Ps) and (Qs)

P \ Q	1	2	3	4	5	6	8	9	10	12	18	20	24
12	65	62.5	61.6	61.2	—	60.8	—	—	—	60.4	—	—	—
18	95	92.5	91.6	—	—	90.8	—	90.5	—	—	90.3	—	—
20	105	102.5	—	101.2	101	—	—	—	100.5	—	—	100.2	—
24	125	122.5	121.6	121.2	—	120.8	120.6	—	—	120.4	—	—	120.2

Figure 6.12 shows execution time reduction by applying the hardware partitioning for various number of query sequences (Ps). The T_{exec} versus Q curves, shown in the figure for various number of Ps, demonstrate that the execution time decreases with the increasing number of hardware partitions, where T_{exec} is computed as per Equation 6.26.

The resource utilization ratio is given by,

$$\begin{aligned} \text{Utilization ratio} &= \frac{P \times N_s \times N^{'}}{(P \times N_s + N^{'} - 1) \times N^{'}} \\ &= \frac{1}{1 + \frac{N^{'}-1}{P \times N_s}} \end{aligned} \tag{6.27}$$

where the utilization ratio is dependent on the $\frac{N^{'}-1}{P \times N_s}$ term. The smaller the $\frac{N^{'}-1}{P \times N_s}$ term, the better the resource utilization. The $\frac{N^{'}-1}{P \times N_s}$ term in itself decreases with the increas-

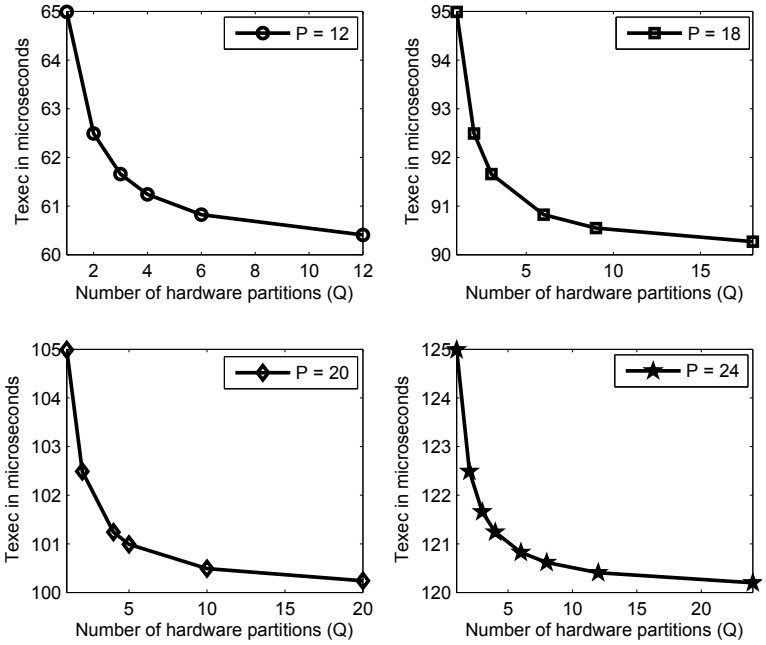


Figure 6.12: Execution time reduction by hardware partitioning

ing number of hardware partitions (i.e. decreasing $N^{'}$), so increasing the number of hardware partitions leads to a better resource utilization, as shown in Figure 6.13. The figure demonstrates that the resource utilization ratio improves with the increasing number of hardware partitions for various number query sequences (Ps), where the resource utilization ratio is computed as per Equation 6.27.

Table 6.4 shows the resource utilization ratio for various number of query sequences (Ps) and valid number of hardware partition (Qs), such that P is divisible by Q. The resource utilization ratio is computed as per Equation 6.27, where $N_q = N_s = 500$. The table demonstrates that the resource utilization ratio improves with the increasing number of hardware partitions (Qs), for all Ps.

Table 6.4: Resource utilization ratio for various (Ps) and (Qs)

P \ Q	1	2	3	4	5	6	8	9	10	12	18
6	0.8574	0.9234	0.9477	—	—	0.9733	—	—	—	—	—
8	0.8891	0.9414	—	0.9699	—	—	0.9849	—	—	—	—
10	0.9093	0.9526	—	—	0.9806	—	—	—	0.9903	—	—
12	0.9232	0.9602	0.9731	0.9798	—	0.9865	—	—	—	0.9933	—
18	0.9475	0.9731	0.9819	—	—	0.9909	—	0.9940	—	—	0.9970

The same theory applies for reducing the execution time and improving the resource utilization ratio for the case, when there is only one query sequence and the

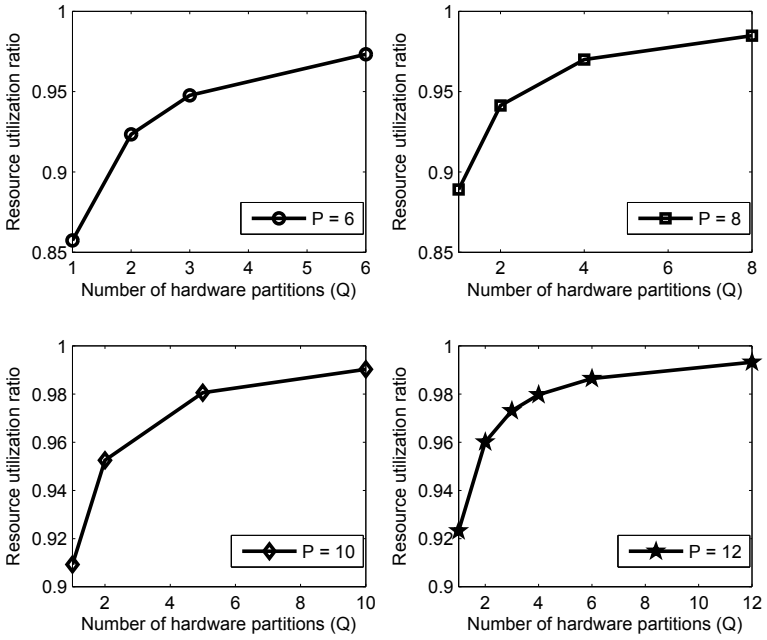


Figure 6.13: Resource utilization improvement by hardware partitioning

database is split into equal parts, such that the same query sequence is scanned against all parts of the database in parallel. This approach is adapted in GPU-based sequence alignment presented in Chapter 5, resulting in execution time reduction, resource utilization improvement and eliminating the need for inter processor communication.

To see the effect of execution time reduction and utilization ratio improvement on performance, we observe the performance equations, given as follows,

$$\text{Performance} = \frac{N_q \times f_{op}}{C_{PE}} \times \text{Utilization ratio} \tag{6.28}$$

where f_{op} is the operating frequency.

Performance may also be given by,

$$\text{Performance} = \frac{\text{Total operations}}{T_{exec}} = \frac{N_q^2}{T_{exec}} \tag{6.29}$$

Equations 6.28 and 6.29 imply that higher resource utilization ratio and lower execution time improve the performance.

In comparison with the traditional methods, the initialization process for the proposed hardware partitioning method is modified, such that the initialization input is equal to a predefined value at the start of the computation. For every succeeding array computation, the initialization input is a feed back from the last PE in the partitioned array.

6.6 Summary

This chapter provided a detailed performance and bandwidth analysis for sequence alignments. Furthermore, it introduced a method to improve performance and resource utilization. Following are the topics presented in the chapter.

- Theoretical performance boundaries.
- Performance limited by computational resources.
- Performance limited by bandwidth.
- Performance optimization when both the computational resources and bandwidth are limited.
- Hardware partitioning method for high performance and resource efficient sequence alignment.
- Execution time reduction and resource utilization improvement for various number of query sequences and hardware partitions.
- Generalization of the hardware partitioning method.

Chapter 7

Conclusions and Future Research Directions

In this chapter conclusions of the thesis and future research directions are presented. Section 7.1 gives conclusions of the work presented in the previous chapters, whereas, Section 7.2 gives an insight into the future research directions.

7.1 Conclusions

This thesis began with a discussion about molecular biology and continued with an overview of bioinformatics, a broad classification of its research areas with a particular emphases on sequence alignment, its types and applications. It proceeded further with a classification of acceleration methods for sequence alignment, followed by relevant literature review and discussion of an accurate profiling and acceleration evaluation approach. Further, it presented FPGA-based systolic array and RVE implementations for biological sequence alignment. The succeeding chapters presented GPU-based sequence alignment and a detailed performance and bandwidth analysis. Following are brief chapter wise conclusions of the thesis.

Chapter 1 presented an introduction about molecular biology by giving an overview of cells, amino acids, proteins, chromosomes, DNA, RNA and transcription. It continued with a classification of the major subfields in bioinformatics and a discussion about sequence alignment, its types and applications. Further, it presented acceleration methods for sequence alignment and details of the thesis contribution. The penultimate section provided an outline of the thesis before the summary of the chapter in the final section.

Chapter 2 gave a classification of global, local and multiple methods like dot plot, N-W and S-W algorithms, FASTA, BLAST, HMMER and ClustalW. It elaborated the difference between exact and approximate methods and continued with a comparison

of these methods based on their time and space complexities. The chapter ended by giving a brief summary of these methods.

Chapter 3 presented an overview about the hardware acceleration of sequence alignment methods and introduced a taxonomy of the various acceleration methods found in the literature. Further, it introduced an accurate speedup evaluation approach. It continued with FPGA-based systolic array implementations for sequence alignment and the discussion of an extended linear systolic array design using BRAM and DDR RAM. The chapter ended with a brief summary.

Chapter 4 presented an RVE-based approach for sequence alignment and its comparison with traditional systolic array based approaches. It presented rectangular and linear FPGA-based RVE implementations for sequence alignment and a comparison of results with corresponding systolic array implementations thereby showing a speedup at the cost of utilizing additional hardware resources. The chapter continued with the RVE performance evaluation and concluded with a brief summary of the chapter itself.

Chapter 5 provided an introduction to GPUs and exploited its parallelization capabilities for biological sequence alignments. It discussed CUDA and its programming and memory models. It provided a brief review of GPU-based sequence alignment and continued with the presentation of an optimized GPU implementation for S-W based protein sequence alignment. It presented the optimization steps undertaken for improving performance of the GPU implementation. More specifically, optimization of the database organization, temporary data reads and writes and substitution matrix accesses. Performance evaluation of the optimized GPU implementation and its comparison with the fastest available design using the same experimental setup. The chapter concluded with a brief summary.

Chapter 6 presented a detailed performance and bandwidth analysis for biological sequence alignment. It continued with developing theoretical performance boundaries for various cases and optimizing memory bandwidth requirement. Further, it introduced an approach based on hardware partitioning to reduce the execution time and improve resource utilization. The chapter also developed generalized equations for high performance and resource efficient sequence alignment. It ended with a brief summary.

Chapter 7 ends the thesis by giving chapter wise brief conclusions and providing future research directions.

7.2 Future research directions

In this thesis the main focus is on the S-W based pairwise local sequence alignment, its acceleration using different available platforms and detailed performance, bandwidth and power analysis. Similar analysis can be done for other applications such as multiple sequence alignment and global alignment. As far as the S-W algorithm is concerned, innovative techniques can be explored to reduce the amount of data storage on FPGA, so that all the data can be stored locally without the need of transferring it

to external memory. One such approach can be storing compressed data instead of the actual one. If at all it becomes possible to store all the data locally in an easily interpretable manner then the trace back step may also be carried out locally. If that happens then only the final alignment result will be required to transfer to the external memory or directly to the output interface thereby significantly improving the performance. An alternative way is the one adapted in the GPU implementation, where only the top scoring sequences are identified and fully aligned using ssearch. The optimization approach provided for the bandwidth requirement reduction can be implemented on different hardware platforms to visualize its practical impact. The hardware partitioning approach introduced in the thesis and evaluated with GPU implementation can also be applied and tested for various FPGA platforms. The GPU implementation itself can be ported and optimized for the latest available GPUs to harvest maximum possible computational power offered by the GPUs. The RVE approach for rectangular blocking factors can be explored further for performance enhancement.

Appendix A

Important Terms in Bioinformatics

1. **Nucleic acid** is a complex, high-molecular-weight biochemical macromolecule composed of nucleotide chains that convey genetic information. The most common nucleic acids are deoxyribonucleic acid (DNA) and ribonucleic acid (RNA). Nucleic acids are found in all living cells and viruses [87].

2. **Deoxyribonucleic acid (DNA)** is a nucleic acid usually in the form of a double helix that contains the genetic instructions monitoring the biological development of all cellular forms of life.

3. **Ribonucleic acid (RNA)** is a nucleic acid polymer consisting of nucleotide monomers. RNA nucleotides contain ribose rings and uracil unlike deoxyribonucleic acid (DNA), which contains deoxyribose and thymine. It is transcribed from DNA by enzymes called RNA polymerases and further processed by other enzymes. RNA serves as the template for translation of genes into proteins, transferring amino acids to the ribosome to form proteins, and also translating the transcript into proteins.

4. **Nucleotides** are the structural units of RNA and DNA. In the cell they play important roles in energy production, metabolism, and signaling.

5. **Amino acid** In chemistry, an amino acid is any molecule that contains both amine and carboxylic acid functional groups. In biochemistry, this shorter and more general term is frequently used to refer to alpha amino acids: those amino acids in which the amino and carboxylate functionalities are attached to the same carbon, the so-called α - carbon.
   An amino acid residue is what is left of an amino acid once a molecule of water has been lost (an H+ from the nitrogenous side and an OH- from the carboxylic

side) in the formation of a peptide bond. Amino acids may come in a variety of shapes and properties. They may be small or bulky, hidrophobic or hidrophyllic, electrically charged or neutral, etc, hence allowing for very complex shapes and interactions to be produced. Amino acids are commonly referred to by name or by an abbreviation, usually in three or one letter. This allows for more efficient descriptions of how they are chained together to build a protein.

6. **A protein** (from the Greek protas meaning "of primary importance") is a complex, high-molecular-mass organic compound that consists of amino acids arranged in a linear chain linked by peptide bonds.

7. **A peptide bond** is a chemical bond formed between two molecules when the carboxyl group of one molecule reacts with the amino group of the other molecule, releasing a molecule of water (H_2O).

8. **Phylogenetic tree** is a tree showing the evolutionary interrelationships among various species or other entities that are believed to have a common ancestor.

9. **Molecular phylogeny** is the use of the structure of molecules to gain information on an organism's evolutionary relationships. The result of a molecular phylogenetic analysis is expressed in a so-called phylogenetic tree.

10. **Sequence motif** In genetics, a sequence motif is a nucleotide or amino-acid sequence pattern that is widespread and has, or is conjectured to have, a biological significance.

11. **Nucleic acid codes** The most common nucleic acid codes are given below.

Nucleic acid code	Meaning
A:	Adenosine
C:	Cytidine
G:	Guanine
T:	Thymidine
U:	Uracil

12. **K-tuples** means subsequences of length k.

13. **Homology** In biology, two or more structures are said to be homologous if they share a common ancestor.

14. **A highly conserved sequence** is a sequence of nucleotides that is identical or very homologous to genes of a wide range of organisms.

Appendix B

Dot Plot Implementation

The dot plot algorithm [88] is one of the oldest computational tools for comparative genomics. It creates a pairwise comparison between two sequences and renders the results as a dot matrix. A dot matrix for two sequences 1 and 2 is simply a grid with the presence of a point at position $p = (i, j)$, if the k-tuple beginning at the *ith* position of Sequence 1 and the *jth* position of Sequence 2 coincide. For years, the quadratic running time for the dot plot algorithm was acceptable because most available sequences were short, but to make it applicable in the era of bioinformatics databases that grow exponentially, there is serious need of speeding it up. The $O(MN)$ complexity of the dot plot can be easily reduced to $O(M + N)$ by implementing it in hardware. Here, we look beyond this reduction and try to develop a hardware implementation approach that will bring the complexity further down to $O(M)$.

Figure B.1 shows the dot plot cell design, where the comparator compares the two input sequences and produces a result based on the match or mismatch. The result is a 1 if there is a match otherwise 0. The adder adds the result of the comparator with the value from the previous cell. The result of the adder is stored in a register used for storing the current value of the cell. The current value is denoted by V_N, where V stands for value and N represents the number of the cell, such that $V_N = V_{N-1} + 1$. There is another register called the backup register, which keeps a backup of the cell's maximum value. V_N is reset to 0 if there is a mismatch, but before reseting, its value is compared with the value of the backup register and if V_N ¿ V_{backup}, then V_{backup} is updated with the new V_N value. The $V_{Nmax}P_{Nmax}$ block keeps track of the maximum value and its index. Figure B.2 shows a 4-element dot plot array constructed using the cell design in Figure B.1. The operation of the array is explained with the help of an example given in Table B.1.

Table B.1 (a) gives an example, where two 4 character DNA sequences are compared using our dot plot implementation approach. The bold digits in the table represents the values in the backup registers, whereas the other digits represent the V_N values. Sequence 1 is fixed along the array, such that each character is an input to a

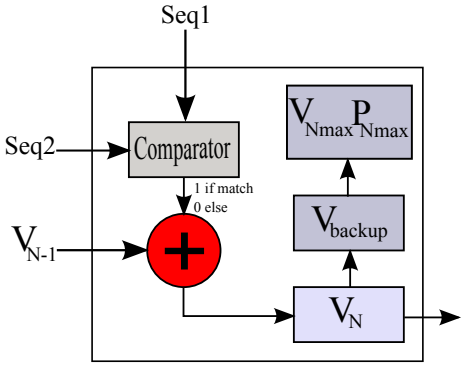


Figure B.1: Dot plot cell design

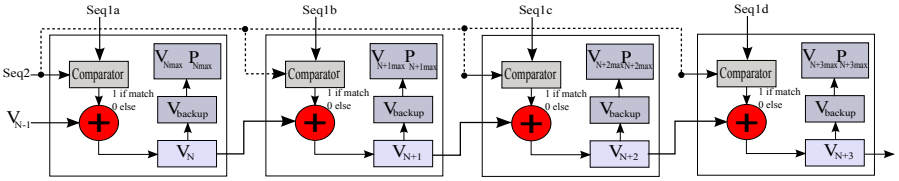


Figure B.2: 4-element dot plot array

different PE, whereas Sequence 2 is propagated through the array character by character. During the 1st clock cycle letter **A** is passed through the array and compared with the corresponding Sequence 1 letters. The resultant V_N and V_{backup} values are recorded in the 1st row of the table. The same process is repeated for all characters in Sequence 2 and the resultant values are recorded in the table. The last row in the table gives the final V_N and V_{backup} values, whereas the position or index of the maximum backup value is given by the $V_{Nmax}P_{Nmax}$ block. The maximum backup value is traced back the number of steps equal to the maximum value itself, considering the current cell as the 1st step. The result of the process is given in Table B.1 (b).

Table B.1: Example to prove our approach and its result

(a) Example

	G	**C**	**T**	**A**
A	**0**	**0**	**0**	**0**
	0	0	0	1
C	**0**	**0**	**0**	**1**
	0	1	0	0
T	**0**	**1**	**0**	**1**
	0	0	2	0
G	**0**	**1**	**2**	**1**
	1	0	0	2

(b) Result

C	**T**
C	**T**

Appendix C

N-W Examples

In this appendix, a couple of examples of global sequence alignment using the Needleman-Wunsch algorithm are presented. Section C.1 presents an example with a simple scoring scheme, whereas Section C.2 presents another example with the same sequences but an advanced scoring scheme.

C.1 Example 1

Here, the two sequences to be globally aligned using Needleman-Wunsch techniques are: G A A T T C A G T T A (query sequence), G G A T C G A (database sequence), so that, $N_q = 11$ and $N_s = 7$ (the lengths of query and database sequences, respectively). A simple scoring scheme is assumed as follows,

$$S_{i,j} = \begin{cases} 1 \text{ if } N_q = N_s & \text{(match score)} \\ 0 \text{ else} & \text{(mismatch score)} \end{cases}$$

$$\text{and} \quad d = 0 \quad \text{(gap penalty)}$$

The three steps in dynamic programming are: Initialization, Matrix fill (scoring) and Traceback.

Initialization step

The first step in the global alignment dynamic programming approach is to create a matrix H with $N_q + 1$ columns and $N_s + 1$ rows. Since this example assumes that there is no gap opening or gap extension penalty, the first row and first column of the matrix can be initially filled with 0 (as shown in Figure C.1) and thus they are considered as row 0 and column 0.

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

Figure C.1: Initialization step

Matrix fill step

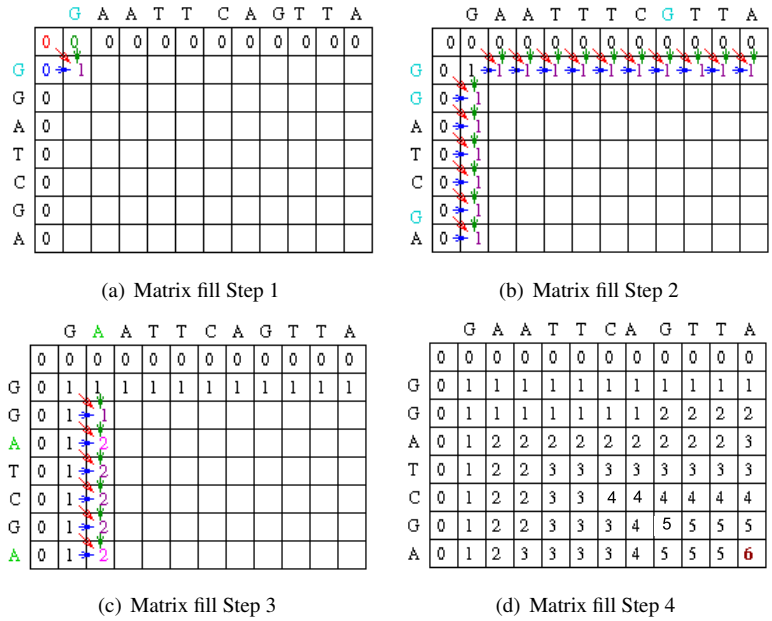
One possible solution of the matrix fill step finds the maximum global alignment score by starting in the upper left hand corner in the matrix and finding the maximal score $H_{i,j}$ for each position in the matrix. In order to find $H_{i,j}$ for any i,j (where i is assumed to be column number and j is assumed to be row number), it is important to know the score for the matrix positions to the left, above and diagonal to i, j. In terms of matrix positions, it is necessary to know $H_{i-1,j}$, $H_{i,j-1}$ and $H_{i-1,j-1}$. For each position, $H_{i,j}$ is defined to be the maximum score at position i,j; i.e.

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + S_{i,j} & \text{(match/mismatch in the diagonal)}, \\ H_{i,j-1} - d & \text{(gap in } N_q\text{)}, \\ H_{i-1,j} - d & \text{(gap in } N_s\text{)} \end{cases}$$

Using this information, the score at position (1,1) in the matrix can be calculated. Since the first character in both sequences is a G, $S_{1,1} = 1$, and by the assumptions stated at the beginning, $d = 0$. Thus, $H_{1,1} = 1$, as shown in Figure C.2(a).

Since the gap penalty d is 0, the rest of row 1 and column 1 can be filled in with the value 1. Take the example of row 1. At column 2, the value is the maximum of 0 (for a mismatch), 0 (for a vertical gap) and 1 (horizontal gap). The rest of row 1 can be filled out similarly until we get to column 8. At this point, there is a G in both sequences. Thus, the value for the cell at row 1, column 8 is the maximum of 1 (for a match), 0 (for a vertical gap) and 1 (horizontal gap). The value will again be 1. The rest of row 1 and column 1 can be filled with 1, as shown in Figure C.2(b), using the above reasoning.

Now let's look at column 2. The location at row 2 will be assigned the value of the maximum of 1 (mismatch), 1 (horizontal gap) and 1 (vertical gap). So its value is 1. At the position column 2, row 3, there is an A in both sequences. Thus, its value will be the maximum of 2 (match), 1 (horizontal gap) and 1 (vertical gap), so its value is 2. Moving along to position column 2 row 4, its value will be the maximum of 1 (mismatch), 1 (horizontal gap) and 2 (vertical gap), so its value is 2. Note that for all of the remaining positions except the last one in column 2, the choices for the value



(a) Matrix fill Step 1

(b) Matrix fill Step 2

(c) Matrix fill Step 3

(d) Matrix fill Step 4

Figure C.2: Matrix fill (a) Step 1, (b) Step 2, (c) Step 3 and (d) Step 4

will be exactly the same as in row 4, since there are no matches. The final row will contain the value 2, since it is the maximum of 2 (match), 1 (horizontal gap) and 2 (vertical gap), see Figure C.2(c).

Using the same techniques, the entire scoring matrix is filled with its corresponding values, as shown in Figure C.2(d).

Traceback step

After the matrix fill step, the maximum alignment score for the two test sequences is 6. The traceback step determines the actual alignment(s) that result in the maximum score. Note that with a simple scoring algorithm such as the one that is used here, there are likely to be multiple maximal alignments.

The traceback step begins in the last row, last column position in the matrix, i.e. the position that leads to the maximal score. In this case, there is a 6 in that location.

Traceback takes the current cell and looks to the neighbor cells that could be direct predecessors. This means it looks to the neighbor to the left (gap in N_s), the diagonal neighbor (match/mismatch), and the neighbor above it (gap in N_q). The algorithm for traceback chooses as the next cell in the sequence one of the possible predecessors. They are all equal to 5, as shown in Figure C.3(a).

Since the current cell has a value of 6 and the scores are 1 for a match and 0 for anything else, the only possible predecessor is the diagonal match/mismatch neighbor.

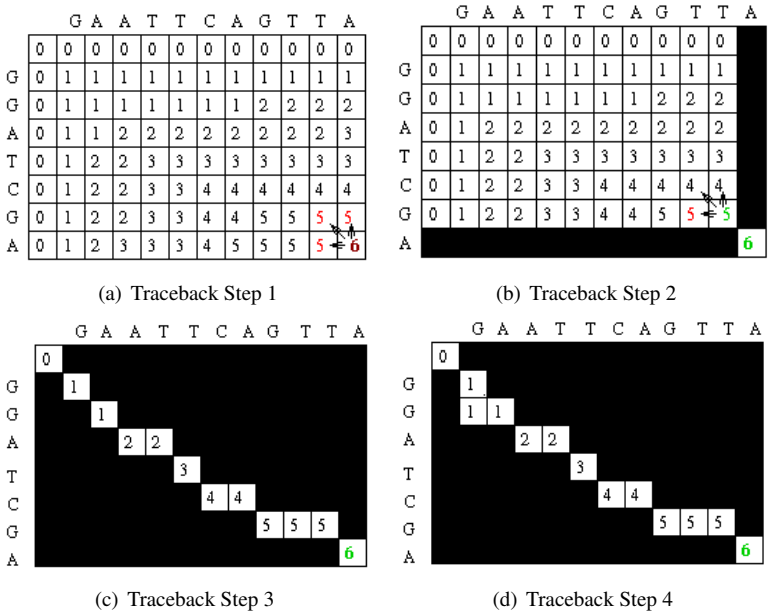


Figure C.3: Traceback (a) Step 1, (b) Step 2, (c) Step 3 and (d) Step 4

If more than one possible predecessor exists, any can be chosen.

This gives us a current alignment of:

N_q:	A
N_s:	A

Now, we look at the current cell and determine which cell is its direct predecessor. In this case, it is the cell to the left with score 5, as shown in Figure C.3(b). The alignment as described in the above step adds a gap to sequence 2,

so the current alignment is:

N_q:	T	A
N_s:	-	A

Continuing on, with the traceback step in the same way, we eventually get to a position in column 0, row 0, which indicates the completion of the traceback. One possible maximum alignment is given in Figure C.3(c),

giving an alignment of:

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

An alternate solution is given in Figure C.3(d),

giving an alignment of :

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

C.2 Example 2

Let us consider another example with the same sequences as in Example 1, but here an advanced scoring scheme is assumed where,

$$S_{i,j} = \begin{cases} 2 \text{ if } N_q = N_s & \text{(match score)} \\ -1 \text{ otherwise} & \text{(mismatch score)} \end{cases}$$
$$\text{and} \quad d = 2 \quad \text{(gap penalty)}$$

Initialization step

As in Example 1, again the first row and first column of the matrix can be initially filled with 0s and regarded as row 0 and column 0, as shown in Figure C.1.

Matrix fill step

Using the same approach as in Example 1, the score at position (1,1) in the matrix is calculated, as shown in Figure C.4(a). Note that there is also an arrow placed back into the cell $H_{0,0}$, that resulted in the maximum score, as shown in Figure C.4(b).

Moving down the first column to row 2, we can see that there is once again a match in both sequences. Thus, $S_{1,2} = 2$ and

$$H_{1,2} = \max \begin{cases} H_{0,1} + 2 \\ H_{1,1} - 2 \\ H_{0,2} - 2 \end{cases} = \max \begin{cases} 0 + 2 \\ 2 - 2 \\ 0 - 2 \end{cases} = \max \begin{cases} 2 \\ 0 \\ -2 \end{cases}$$

Hence, a value of 2 is placed in position (1,2) of the scoring matrix, as shown in Figure C.4(b) and an arrow is placed to point back to $H_{0,1}$ which led to the maximum score, as shown in Figure C.4(c).

Looking at column 1, row 3, there is no match in the sequences, so $S_{1,3} = -1$ and

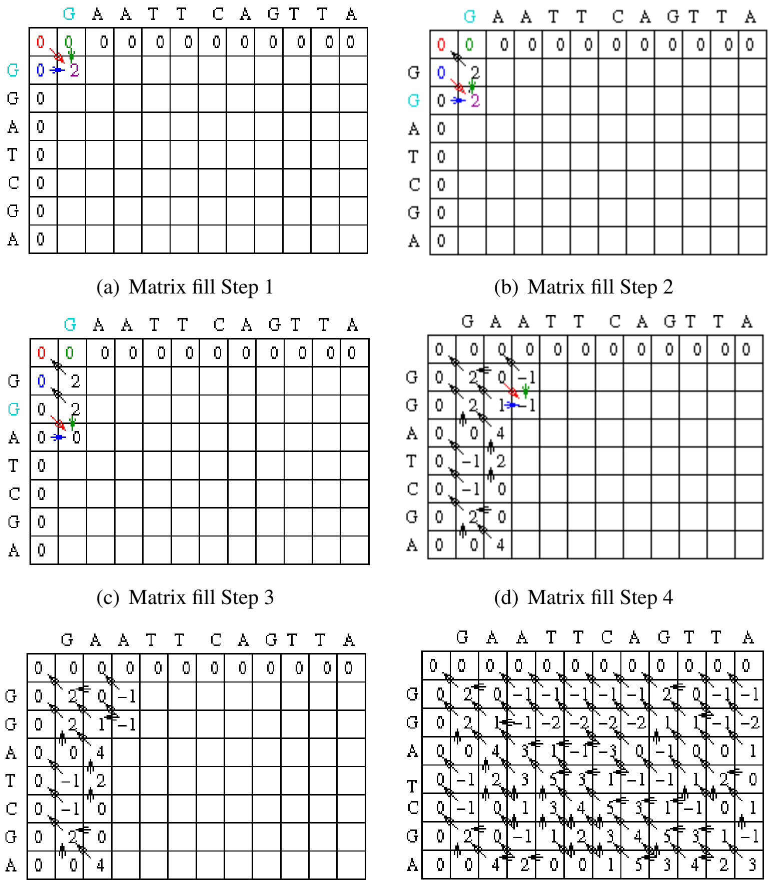
$$H_{1,3} = \max \begin{cases} H_{0,2} - 1 \\ H_{1,2} - 2 \\ H_{0,3} - 2 \end{cases} = 0$$

Thus, a value, 0, is placed in position (1,3) of the scoring matrix, as shown in Figure C.4(c) and an arrow is placed to point back to $H_{1,2}$, which led to the maximum score, as shown in Figure C.4(d).

The same procedure is continued for filling in the cells of the scoring matrix. Eventually, we get to column 3, row 2, as shown in Figure C.4(d). There is no match in the sequences at this position, so, $S_{3,2} = -1$ and

$$H_{3,2} = \max \begin{cases} H_{2,1} - 1 \\ H_{3,1} - 2 \\ H_{2,2} - 2 \end{cases} = \max \begin{cases} -1 \\ -3 \\ -1 \end{cases}$$

In this case, there are two different ways to get the maximum score. In such a case, pointers are placed back to all the cells that can produce the maximum score, as shown in Figure C.4(e). The rest of the scoring matrix is filled in the same manner. The completed scoring matrix is shown in Figure C.4(f).



(a) Matrix fill Step 1

(b) Matrix fill Step 2

(c) Matrix fill Step 3

(d) Matrix fill Step 4

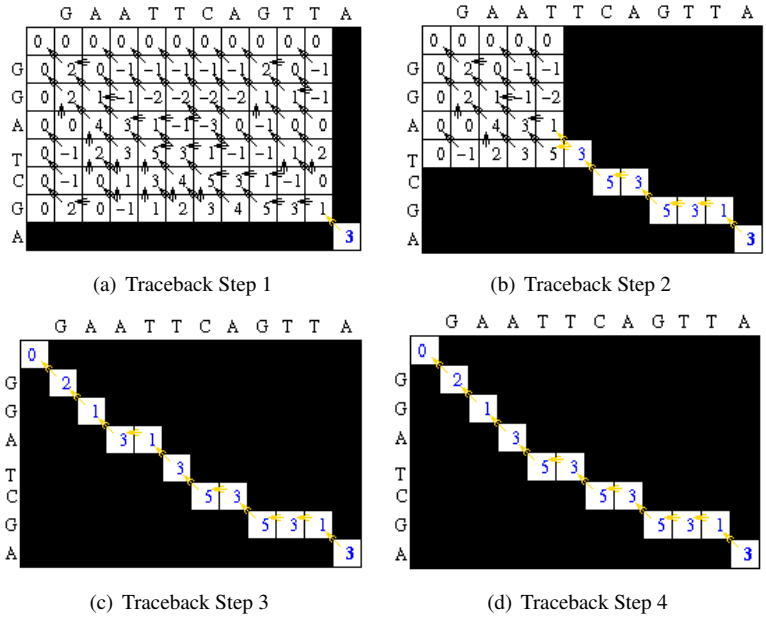
(e) Matrix fill Step 5

(f) Matrix fill Step 6

Figure C.4: Matrix fill for Example 2

Traceback step

After the matrix fill step, the maximum global alignment score for the two sequences is 3. The traceback step determines the actual alignment(s) that result in the maximum score. The traceback step begins in the last row, last column position in the matrix, i.e. the position where both sequences are globally aligned. Since, we have kept pointers back to all possible predecessors, the traceback step is simple. At each cell, we look to see, where we move next according to the pointers. To begin, the only possible predecessor is the diagonal match, as shown in Figure C.5(a).



(a) Traceback Step 1 (b) Traceback Step 2

(c) Traceback Step 3 (d) Traceback Step 4

Figure C.5: Traceback for Example 2

This gives us an alignment of:

		A
		A

We continue to follow the path using a single pointer until we get to the situation, as shown in Figure C.5(b).

The alignment at this point is:

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

Note, that there are now two possible neighbors that could result in the current score. In such a case, one of the neighbors is arbitrarily chosen. Once, the traceback is completed, it can be seen that there are only two possible paths leading to a maximal global alignment. One possible path is shown in Figure C.5(c),

giving an alignment of:

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

The other possible path is shown in figure C.5(d),

giving an alignment of:

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

Remembering that the scoring scheme is +2 for a match, -1 for a mismatch, and the gap penalty is 2, both sequences can be tested to make sure that they result in a score of 3.

G	A	A	T	T	C	A	G	T	T	A
G	G	A	-	T	C	-	G	-	-	A
+	-	+	-	+	+	-	+	-	-	+
2	1	2	2	2	2	2	2	2	2	2

2 - 1 + 2 - 2 + 2 + 2 - 2 + 2 - 2 - 2 + 2 = 3

G	A	A	T	T	C	A	G	T	T	A
G	G	A	T	-	C	-	G	-	-	A
+	-	+	+	-	+	-	+	-	-	+
2	1	2	2	2	2	2	2	2	2	2

2 - 1 + 2 + 2 - 2 + 2 - 2 + 2 - 2 - 2 + 2 = 3

Hence, both of these alignments indeed do result in the maximal alignment score.

Appendix D

S-W Examples

In this appendix, a flow chart description of the S-W algorithm and a couple of examples of local sequence alignment using the S-W algorithm are presented. It starts with the flow chart description in Section D.1, followed by a simple S-W example in Section D.2. It is concluded by another example in Section D.3 that explains the SW algorithm in comparison with Needleman-Wunsch algorithm.

D.1 Flow chart

Figure D.1 describes S-W algorithm in the form of a flow chart to elaborate its theoretical basis [23].

D.2 Example 1

In this example, the Smith-Waterman algorithm, based on the dynamic programming technique, is used to compute the optimal local alignment of two sequences,
i.e. N_q = a g g t a c and N_s = c a g c g t t g. Assume that,

$$S_{i,j} = \begin{cases} +2 & \text{if } N_q = N_s \\ -1 & else \end{cases}$$

and gap penalty (d) = 2.

The procedure consists of three steps:

1. Fill in the dynamic programming matrix.

2. Find the maximal value (score) in the matrix.

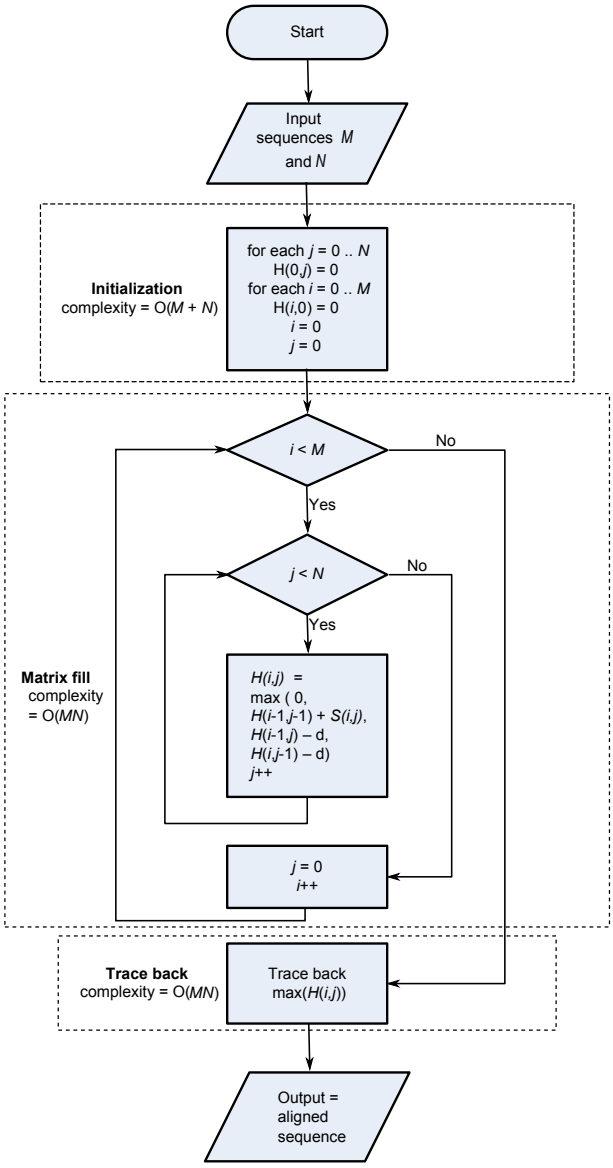


Figure D.1: Smith-Waterman flow chart

Table D.1: The dynamic programming matrix and the traceback path

		c	a	g	c	g	t	t	g
	0	**0**	0	0	0	0	0	0	0
a	0	0	**2**	0	0	0	0	0	0
g	0	0	0	**4**	**2**	2	0	0	2
g	0	0	0	2	3	**4**	2	0	2
t	0	0	0	0	1	2	**6**	4	2
a	0	0	2	0	0	0	4	5	3
c	0	2	0	1	2	0	2	3	4

3. Trace back the path that leads to the maximal score to find the optimal local alignment.

Table D.1 illustrates the calculation of the dynamic programming matrix H and the path of tracing back (shown in bold digits). The best score found in the matrix is 6 and

the corresponding optimal local alignment is:

A:	a	g	-	g	t
B:	a	g	c	g	t

D.3 Example 2

A key feature of the Smith-Waterman algorithm is that each cell in the matrix defines the end point of a potential alignment, whose similarity is represented by the value stored in the cell. The algorithm thus begins by filling the edge elements with 0.0 values, as illustrated in Table D.2, because these cells represent the ends of alignments of length zero and consequently, their similarity score is zero. Note that, here, cells in the matrix are populated with floating-point values, rather than integers, which are characteristic of the Needleman-Wunsch method; however, there is no reason why either method could not be implemented using integers or floating-point values. The symbol 'x' is used as a placeholder, as the first row and first column cannot be the endpoint of any alignment.

Table D.2: Initialization for Example 2 with floating point values

	x	A	D	L	G	A	V	F	A	L	C	D	R	Y	F	Q
x	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A	0.0															
D	0.0															
L	0.0															
G	0.0															
R	0.0															
T	0.0															
Q	0.0															
N	0.0															
C	0.0															
D	0.0															
R	0.0															
Y	0.0															
Y	0.0															
Q	0.0															

Table D.3: Calculation of first set diagonal similarity scores in the Smith-Waterman algorithm

	x	A	D	L	G	A	V	F	A	L	C	D	R	Y	F	Q
x	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A	0.0	**1.0**	0.0	0.0	0.0	1.0	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
D	0.0	0.0	**2.0**	0.0	0.0	0.0	0.7	0.0	0.0	**0.7**	0.0	1.0	0.0	0.0	0.0	0.0
L	0.0	0.0	0.0	**3.0**	0.0	0.0	0.0	0.3	0.0	**1.0**	**0.3**	0.0	**0.7**	0.0	0.0	0.0
G	0.0	0.0	0.0	0.0	**4.0**	0.0	0.0	0.0	0.0	0.0	**0.7**	0.0	0.0	**0.3**	0.0	0.0
R	0.0	0.0	0.0	0.0	0.0	**3.7**	0.0	0.0	0.0	0.0	0.0	**0.3**	**1.0**	0.0	0.0	0.0
T	0.0	0.0	0.0	0.0	0.0	0.0	**3.3**	0.0	0.0	0.0	0.0	0.0	0.0	**0.7**	0.0	0.0
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**3.0**	0.0	0.0	0.0	0.0	0.0	0.0	**0.3**	**1.0**
N	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**2.7**	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**2.3**	1.0	0.0	0.0	0.0	0.0	0.0
D	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**2.0**	**2.0**	0.0	0.0	0.0	0.0
R	0.0	0.0	0.0	**0.7**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1.7**	**3.0**	0.0	0.0	0.0
Y	0.0	0.0	0.0	0.0	**0.3**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**1.3**	**4.0**	0.0	0.0
Y	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**2.3**	**3.7**	0.0
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**2.0**	**4.7**

Table D.4: The endpoint of the Smith-Waterman algorithm after calculation of all scoring parameters. A traceback from the highest score is highlighted

	x	A	D	L	G	A	V	F	A	L	C	D	R	Y	F	Q
x	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
A	0.0	**1.0**	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
D	0.0	0.0	**2.0**	0.7	0.3	0.0	0.7	0.0	0.0	0.7	0.0	1.0	0.0	0.0	0.0	0.0
L	0.0	0.0	0.7	**3.0**	1.7	1.3	1.0	0.7	0.3	1.0	0.3	0.0	0.7	0.0	0.0	0.0
G	0.0	0.0	0.3	0.0	**4.0**	2.7	2.3	2.0	1.7	1.3	1.0	0.7	0.3	0.3	0.0	0.0
R	0.0	0.0	0.0	0.0	2.7	**3.7**	2.3	2.0	1.7	1.3	1.0	0.7	1.0	0.0	0.0	0.0
T	0.0	0.0	0.0	0.0	2.3	2.3	**3.3**	2.0	1.7	1.3	1.0	0.7	0.3	0.7	0.0	0.0
Q	0.0	0.0	0.0	0.0	2.0	2.0	2.0	**3.0**	1.7	1.3	1.0	0.7	0.3	0.0	0.3	1.0
N	0.0	0.0	0.0	0.0	1.7	1.7	1.7	1.7	**2.7**	**1.3**	1.0	0.7	0.3	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	1.3	1.3	1.3	1.3	1.3	2.3	**1.0**	0.7	0.3	0.0	0.0	0.0
D	0.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	**2.0**	0.7	0.3	0.0	0.0
R	0.0	0.0	0.0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	1.7	**3.0**	1.7	1.3	1.0
Y	0.0	0.0	0.0	0.0	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	1.7	**4.0**	2.7	2.3
Y	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3	2.7	**3.7**	2.3
Q	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	2.3	2.3	**4.7**

The next step is to populate the remaining cells in the matrix. This is achieved by evaluating three functions and choosing the maximum of the three values, or zero if a negative value would result. These functions consider the possibilities for ending an alignment at any particular cell. First, the similarity score (e.g., 1.0 for a match, -0.333 for a mismatch) for the diagonal predecessor of the cell under consideration is added to that cell's score, as shown in Table D.3; then the maximum value is calculated for a deletion represented along (a) the current row of the matrix, and (b) along the current column of the matrix. Finally, if a negative score would result, 0.0 is substituted, to indicate that there is no alignment similarity up to the current cell position. Once the matrix is complete, the highest score is located (representing the endpoint of the highest scoring alignment between the two sequences), and the other elements leading to this cell are determined using a traceback procedure, as illustrated in Table D.4. If necessary, we can search the matrix for lower-scoring local alignments simply by finding other high scores that do not form part of a previous traceback.

The essential difference between the N-W and S-W is that, in the Smith-Waterman, the matrix contains a maximum value that may not be at the N-termini of the sequences. It represents the endpoint of an alignment such that no other pair of segments with greater similarity exists between the two sequences. Hence, this is a local, rather than a global, alignment method.

Appendix E

Power Consumption Evaluation

Due to the utilization of abundant hardware resources, power consumption is becoming an important constraint for modern day sequence alignment applications. The overall power consumption consists of static and dynamic power. Static power is due to the leakage current and is technology dependent, whereas dynamic power is due to the transient current and is a consequence of the switching activity. The switching activity in turn depends on the size and type of logic and the nature of input data set. Dynamic power consumption is critical for sequence alignment applications, as it influences the performance, particularly for larger designs. In this appendix, an evaluation of dynamic power consumption for sequence alignment applications is presented and the performance per unit Watt for various number of PEs is investigated. Additionally, resource utilization and performance results are provided for implementation with a number of different platforms.

E.1 Evaluation of dynamic power consumption

The dynamic power consumed by S-W based sequence alignment implemented on a hardware platform like an FPGA is largely due to the charging and discharging activities of the capacitive elements, such as logic resources and the interconnecting fabric [89]. This can be modeled as,

$$P_i = \sum C_i V_i^2 f_i \tag{E.1}$$

where C_i, V_i and f_i are the capacitance, supply voltage and operating frequency of resource i, respectively [90].

Randomly selected input sequences from ***ssearch*** class-c benchmark of BioPerf are used to evaluate the dynamic power consumption for S-W based sequence alignment. The BioPerf suite [91] includes benchmark source codes (e.g. ***ssearch*** for the

S-W algorithm), input datasets of various sizes, and information for compiling and using the benchmarks. It contains codes from highly popular bioinformatics packages [92] and covers the major fields of study in computational molecular biology, such as sequence comparison, phylogenetic reconstruction, protein structure prediction, and sequence homology & gene finding. The benchmark considered for simulations represents the complete genome. The number of PEs are scaled according to the lengths of the input biological sequences, randomly selected from the benchmark for the evaluation of dynamic power consumption. However, sequences of lengths larger than the maximum available PEs are aligned by partitioning the query sequences [30]. For each selected length, a variety of input sequences are considered for simulations and the average dynamic power consumption is recorded. Power analyzer tool *XPower* of Xilinx *ISE 10.1* Design Suite is used for the power analysis, whereas the devices used for implementations are Xilinx *Virtex-II Pro (XC2VP30), Virtex-IV (XC4VFX12)* and *Virtex-V (XC5VTX240T)* FPGAs.

Table E.1 presents an evaluation of the dynamic power consumption for varying number of PEs, considering $XC2VP30$ FPGA for implementation. The 1st column represents the number of PEs. The 2nd column shows the power consumed by clock transitions, which increases with the increasing number of PEs. The 3rd column gives the power consumed by logic. Again, the power consumption increases with the increasing number of PEs except for the 1st row, where more power is consumed than for the succeeding higher number of PEs. The reason for this is that memories are also implemented as logic by the Xilinx *ISE* tool and no BRAMs are instantiated. The 4th column provides the power consumed by the signals, i.e. the dynamic power consumption due to the switching activity along the wires. The 5th column represents the combined power consumed by IOs and BRAMs. The last column presents the total dynamic power consumption, which is the sum of power consumed by clocks, logic, signals, IOs and BRAMs, i.e.

$$P_{Total} = P_{Clocks} + P_{Logic} + P_{Signals} + P_{IOs} + P_{BRAMs} \tag{E.2}$$

Table E.1: Dynamic power consumption in milliwatts ($XC2VP30$)

PEs	Clocks	Logic	Signals	IOs + BRAMs	Total
4	2.07	1.19	1.50	0.10	4.85
6	2.23	0.69	2.23	0.10	5.25
8	2.75	0.84	2.33	0.11	6.02
20	5.34	0.89	5.17	0.12	11.51
44	8.18	1.86	11.38	0.40	21.81
72	10.15	2.99	19.63	0.43	33.18
108	10.87	5.43	33.64	0.53	50.46

Tables E.2 and E.3 present dynamic power consumption results for implementations using $XC4VFX12$ and $XC5VTX240T$ devices, where similar trends are ob-

served, as for $XC2VP30$ device in Table E.1. The maximum number of PEs in Table E.2 is limited due to the reduced amount of resources offered by $XC4VFX12$ device.

Table E.2: Dynamic power consumption in milliwatts ($XC4VFX12$)

PEs	Clocks	Logic	Signals	IOs + BRAMs	Total
4	28.11	0.36	0.33	0.03	28.82
6	29.00	0.19	0.34	2.34	31.87
8	32.92	0.21	0.58	2.48	36.19
20	37.24	0.26	0.85	7.71	46.06
44	41.26	0.57	2.68	17.61	62.12
48	41.21	0.68	4.53	19.15	65.57

Table E.3: Dynamic power consumption in milliwatts ($XC5VTX240T$)

PEs	Clocks	Logic	Signals	IOs + BRAMs	Total
4	9.32	0.15	0.16	0.03	9.66
6	11.10	0.10	0.21	0.78	12.19
8	12.03	0.12	0.23	1.02	13.39
20	22.05	0.19	0.47	2.42	25.12
44	37.82	0.41	1.38	5.25	44.85
72	81.93	0.63	2.39	8.84	93.79
108	118.22	1.14	4.51	13.04	136.90

E.2 Resource utilization

Table E.4 presents device utilization in terms of slices and BRAMs, considering $XC2VP30$ implementation. Further, it provides the maximum frequency in MHz and performance in GCUPS for the S-W based sequence alignment. The 1st column in the table represents the number of PEs. The 2nd column provides the number of slices consumed for all given numbers of PEs. The 3rd column presents the BRAMs utilization. The reason for having no BRAMs in the 1st row is that when a limited number of memories needs to be instantiated then the Xilinx *ISE* synthesizer puts them in *Look Up Tables (LUTs)* instead of BRAMs during the synthesis process, to avoid any wastage of BRAM resources. The on-chip BRAM in FPGAs is a limited commodity and this approach saves them for other applications. The fourth column gives the maximum post place and route frequency in MHz. The last column presents the performance in GCUPs, calculated as follows:

$$\text{Performance} = N_{PE} \times f_{op} \tag{E.3}$$

where N_{PE} is the number of PEs and f_{op} is the operating frequency.

Table E.4: Device utilization and performance results ($XC2VP30$)

PEs	Slices	BRAMs	Frequency (MHz)	Performance (GCUPS)
4	646	—	110.26	0.441
6	723	3	110.00	0.660
8	975	4	109.80	0.878
20	2307	10	109.00	2.180
44	4897	24	107.20	4.717
72	7762	38	105.50	7.596
108	11737	56	103.70	11.908

Similarly, Tables E.5 and E.6 present device utilization and performance results for implementations with $XC4VFX12$ and $XC5VTX240T$ devices. Tables E.4, E.5 and E.6 indicate an increase in performance for higher number of PEs. However, a decreasing trend is observed for the maximum operating frequency due to the higher latency for larger designs.

Table E.5: Device utilization and performance results ($XC4VFX12$)

PEs	Slices	BRAMs	Frequency (MHz)	Performance (GCUPS)
4	670	—	140.64	0.563
6	816	3	140.00	0.840
8	1072	4	139.44	1.115
20	2478	10	136.32	2.726
44	4943	24	129.79	5.711
48	5359	26	128.63	6.174

E.3 Performance optimization

Figure E.1 depicts the results of performance per unit Watt for various number of PEs, considering different technologies, i.e. different FPGA platforms like $XC2VP30$, $XC4VFX12$ and $XC5VTX240T$ devices, for implementations.

The results in the figure demonstrate that the performance per unit Watt increases with the increasing number of PEs initially. It stabilizes after increasing the number of PEs beyond a certain point and eventually starts to decrease. The curve for $XC4VFX12$ is shorter than the other two curves due to a limited amount of resources offered by the device. The results are influenced by the following two factors.

1. The sub-linear increase in performance with the increasing number of PEs. The

Table E.6: Device utilization and performance results ($XC5VTX240T$)

PEs	Slices	BRAMs	Frequency (MHz)	Performance (GCUPS)
4	317	—	198.63	0.794
6	429	3	197.38	1.184
8	552	4	196.13	1.569
20	1461	10	192.31	3.846
44	3343	21	189.13	8.322
72	5479	35	186.42	13.422
108	8286	52	181.56	19.608

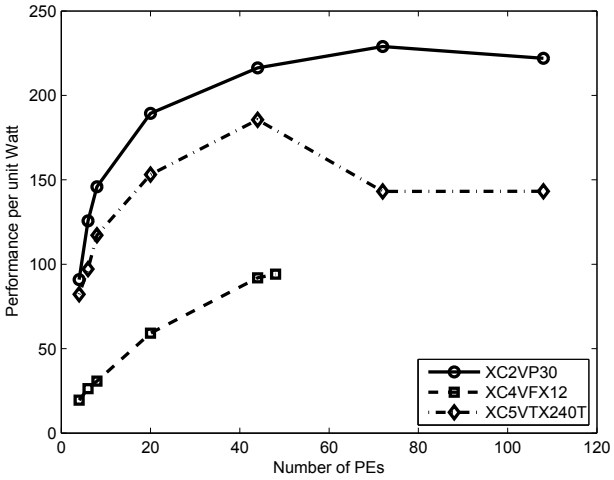


Figure E.1: Performance per unit Watt for S-W based sequence alignment

reason for this is that the maximum operating frequency decreases due to the increasing latency for larger designs.

2. The slightly super-linear increase in dynamic power consumption with the increasing number of PEs. The reason for this is that larger designs generate higher switching activity and hence consume more dynamic power.

This analysis helps in approximating the number of PEs that gives an optimized performance per unit Watt. It is observed from Figure E.1 that for achieving an optimized performance per unit Watt, the number of PEs can be approximated between 40 and 60 for $XC4VFX12$ and $XC5VTX240T$ FPGA devices. Similarly, it can be approximated between 70 and 80 for $XC2VP30$ device. Beyond these numbers, the performance per unit Watt decreases with any further increase in the number of PEs. For future work, we intend to use a larger Virtex-IV FPGA device with more resources than the device under consideration to observe the behavior of the device beyond the

current limit and better approximate the number of PEs for an optimized performance per unit Watt. Also, the $XC5VTX240T$ implementation can be scaled up for aligning longer input biological sequences in one pass to observe an onward trend for the performance per unit Watt curve based on Virtex-V FPGA.

The results are approximated by using the *MATLAB* Curve Fitting Tool and selecting a 4th degree polynomial for the curve fit, as it better resembles the experimental curves and gives a minimum *Root Mean Square Error (RMSE)*.

$$f(x) = c_1 \times x^4 + c_2 \times x^3 + c_3 \times x^2 + c_4 \times x + c_5 \tag{E.4}$$

Equation E.4 gives an approximated model where, $x = N_{PE}$, and the values of the polynomial coefficients and RMSE for various FPGA platforms under consideration are given in Table E.7.

Table E.7: Modeling coefficients for various technologies

Coefficients	**XC2VP30**	**XC4VFX12**	**XC5VTX240T**
$\mathbf{c_1}$	-1.87×10^{-005}	-5.302×10^{-006}	3.212×10^{-006}
$\mathbf{c_2}$	0.004467	0.0005742	3.455×10^{-005}
$\mathbf{c_3}$	-0.368	-0.04786	-0.09594
$\mathbf{c_4}$	12.73	3.376	6.668
$\mathbf{c_5}$	55.93	7.03	61.48
RMSE	11.78	0.8504	7.464

Bibliography

[1] A. M. Lesk, "Introduction to Bioinformatics", *Oxford University Press*, Oxford, New York, 2004.

[2] J. Cohen, "Bioinformatics: An Introduction for Computer Scientists", *ACM Computing Surveys*, vol. 36(2), pages 122–158, June 2004.

[3] L. R. Murphy, A. Wallqvist and R. M. Levy, "Simplified Amino Acid Alphabets for Protein Fold Recognition and Implication for Folding", *Protein Engineering*, vol. 13(3), pages 149–152, 2000.

[4] C. M. Keet, "Conceptual Modeling for Applied Bioscience", *School of Computing, Napier University*, Edinburgh, Scotland.

[5] Oscar Gruss BioTechnology Review, 13 March 2000.

[6] "http://www.insdc.org", *International Nucleotide Sequence Database Collaboration*, April 2010.

[7] http://www.ncbi.nlm.nih.gov.

[8] Boeckmann et al., "The SWISS-PROT Protein Knowledge Base and its Supplement TrEMBL", *Nucleic Acids Research*, vol. 31, pages 365–370, 2003.

[9] http://pir.georgetown.edu/.

[10] D. M. Mount, "Bioinformatics: Sequence and Genome Analysis", *Cold Spring Harbor Laboratory Press*, Cold Spring Harbor, NY, 2nd ed., 2004.

[11] L. Holm and C. Sander, "Protein Structure Comparison by Alignment of Distance Matrices", *Journal of Molecular Biology*, vol. 233(1), pages 123-138, 1993.

[12] C. Chothia and A. M. Lesk, "The Relation Between the Divergence of Sequence and Structure in Proteins", *The EMBO Journal*, vol. 5(4), pages 823-826, April 1986.

[13] S. M. Larson et al., "Using Distributed Computing to Tackle Previously Intractable Problems in Computational Biology", *Folding@Home and Genome@Home*.

[14] J. H. Havgaard, R. B. Lyngs, G. D. Stormo and J. Gorodkin, "Pairwise Local Structural Alignment of RNA Sequences with Sequence Similarity less than 40%", *Bioinformatics*, vol. 21(9), pages 1815-1824, 2005.

[15] A. Isaev, "Introduction to Mathematical Methods in Bioinformatics (Universitext)", *Springer*, vol. 58(1), June 2004.

[16] S. Needleman and C. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of two Proteins", *Journal of Molecular Biology*, vol. 48(3), pages 443–453.

[17] T. F. Smith and M. S. Waterman, "Identification of Common Molecular Subsequences", *Journal of Molecular Biology*, vol. 147, pages 195–197, 1981.

[18] "http://www.clustal.org", *Clustal: Multiple Sequence Alignment*, April 2010.

[19] W. R. Pearson and D. J. Lipman, "Rapid and Sensitive Protein Similarity Searches", *Science*, vol. 227, pages 1435–1441, 1985.

[20] S. F. Altschul et al., "A Basic Local Alignment Search Tool", *Journal of Molecular Biology*, vol. 215, pages 403–410, 1990.

[21] S. Derrien and P. Quinton "Hardware Acceleration of HMMER on FPGAs", *Journal of Signal Processing System*, 58, pages 53–67, 2010.

[22] L. R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition", *Proc. IEEE 77*, vol. 2, pages 257–286, 1989.

[23] L. Hasan, Z. Al-Ars and S. Vassiliadis, "Hardware Acceleration of Sequence Alignment Algorithms - An Overview", *International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'07)*, pages 96–101, Rabat, Morocco, September 2–5, 2007.

[24] L. Hasan and Z. Al-Ars, "Accurate Profiling and Acceleration Evaluation of the Smith-Waterman Algorithm using the MOLEN Platform", *International Conference on Applied Computing*, pages 188–194, Algarve, Portugal, April 2008.

[25] L. Hasan, Y. M. Khawaja and A. Bais, "A Systolic Array Architecture for The Smith-Waterman Algorithm with High Performance Cell Design", *IADIS European Conference on Data Mining*, Amsterdam, The Netherlands, July 2008.

[26] L. Hasan, Z. Al-Ars, Z. Nawaz and K. L. M. Bertels, "Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion", 3^{rd} *International Design and Test Workshop IDT08*, Monastir, Tunisia, December 2008.

[27] L. Hasan and Z. Al-Ars, "An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm", 31st *Annual International Conference of the IEEE EMBS*, pages 3845–3848, Minneapolis, Minnesota, USA, September 2009.

[28] L. Hasan, M. Kentie and Z. Al-Ars, "DOPA: GPU-based Protein Alignment Using Database and Memory Access Optimizations", *Submitted to BMC Bioinformatics*, ISSN 1471-2105, 2011.

[29] L. Hasan, Z. Al-Ars, M. Taouil and K. L. M. Bertels, "Performance and Bandwidth Optimization for Biological Sequence Alignment", 5th *International Design and Test Workshop (IDT'10)*, Abu Dhabi, UAE, December 14–15, 2010.

[30] L. Hasan, Z. Al-Ars and M. Taouil, "High Performance and Resource Efficient Biological Sequence Alignment", 32nd *Annual International Conference of the IEEE EMBS*, Pages 1767–1770, Buenos Aires, Argentina, August 31–September 4, 2010.

[31] L. Hasan and Z. Al-Ars, "Power Consumption Evaluation for Biological Sequence Alignment", 1st *STW.ICT Conference*, Pages 1–6, Veldhoven, The Netherlands, November 18–19, 2010.

[32] Waterman and Michael, "Introduction to Computational Biology", *Chapman and Hall*, 1995.

[33] I. Eidhammer, I. Jonassen and W. R. Taylor, "Pairwise Global Alignment of Sequences", *Protein Bioinformatics*, 2004.

[34] C. A. Orengo and W. R. Taylor, "A Local Alignment Method for Protein Structure Motifs", *Journal of Molecular Biology*, 233, pages 488-497, 1993.

[35] http://www.dbmi.columbia.edu/bioinformatics/.

[36] R. C. Edgar and S. Batzoglou, "Multiple Sequence Alignment", *Elsevier*, 16, pages 368–373, 2006.

[37] T. K. Attwood and D. J. P. Smith, "Introduction to Bioinformatics", *Cell and Molecular Biology in Action Series*.

[38] R. Giegerich, "A Systematic Approach to Dynamic Programming in Bioinformatics", *Bioinformatics*, vol. 16, pages 665–677, 2000.

[39] T. Ramdas and G. Egan, "A Survey of FPGA-based High Performance Computation in Molecular Biology and other Domains", *Technical Report*, MECSE, 2005.

[40] O. Gotoh, "An improved algorithm for matching biological sequences", *Journal of Molecular Biology*, vol. 162, pages 705–708, December 1982.

[41] H. Y. Liao, M. L. Yin and Y. Cheng, "A Parallel Implementation of the Smith-Waterman Algorithm for Massive Sequences Searching", 26^{th} *Annual International Conference of the IEEE EMBS"*, San Francisco, CA, USA, September 1–5, 2004.

[42] http://www.geocities.com/bioinformaticsweb/seqanalysis.html.

[43] S. F. Altschul et al., "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs", *Nucleic Acids Research*, vol. 25(17), pages 3389–3402, 1997.

[44] NCBI - BLAST (http://www.ncbi.nlm.nih.gov/BLAST/).

[45] EMBL - EBI (http://www.ebi.ac.uk/blast2/index.html).

[46] S. R. Eddy., "Profile Hidden Markov Models", *Bioinformatics*, 14(9), pages 755–763, 1998.

[47] A. Krogh, et al., "Hidden Markov Models in Computational Biology: Applications to Protein Modeling", *Journal of Molecular Biology*, 235, pages 1501-1531, 1994.

[48] "White Paper on CLC Bioinformatics Cell 2.1.2", March 27, 2009.

[49] http://www.sanger.ac.uk/Software/Pfam/.

[50] Bateman et al., "The Pfam Protein Families Database", *Nucleic Acids Research*, 32(Database issue), pages D138–D141, 2004.

[51] J. Lu, M. Perrone, K. Albayraktaroglu and M. Franklin, "HMMer-Cell: High Performance Protein Profile Searching on the Cell/B.E. Processor", *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2008)*, pages 223–232, Austin, Texas, USA, April 20–22, 2008.

[52] J. D. Thompson, D. G. Higgins and T. J. Gibson, "ClustalW: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice", *Nucleic Acids Research*, 22(22), pages 4673–4680, 1994.

[53] D. Feng and R. Doolittle, "Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees", *Journal of Molecular Evolution*, vol. 25, pages 351–360, August 1987.

[54] Y. Liu, B. Schmidt and D. L. Maskell, "MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA", 20^{th} *IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP09)*, pages 121–128, Boston MA, USA, July 7-9, 2009.

[55] Jonassen and Inge, "http://www.ii.uib.no/ inge/kb207/slides/tsld001.htm", *Multiple Sequence Alignment*, 2007.

[56] J. Chiang et al., "Hardware Accelerator for Genomic Sequence Alignment", 28^{th} *IEEE EMBS Annual International Conference*, New York City, USA, Aug 30–Sept 3, 2006.

[57] Y. Yamaguchi, Y. Miyajima, T. Maruyama and A. Konagaya, "High Speed Homology Search Using Run-Time Reconfiguration", *FPL 2002*.

[58] S. Margerm, Cray Inc, "Reconfigurable Computing in Real-World Applications", *FPGA and Structured ASIC Journal (www.fpgajournal.com)*, February 7, 2006.

[59] H. T. Kung and C. E. Leiserson, "Algorithms for VLSI Processor Arrays", *in: C. Mead, L. Conway (eds.): Introduction to VLSI Systems; Addison-Wesley*, 1979.

[60] P. Quinton and Y. Robert, "Systolic Algorithms and Architectures", *Prentice Hall International*, 1991.

[61] G. Pfeiffer, H. Kreft and M. Schimmler, "Hardware Enhanced Biosequence Alignment", *International Conference on METMBS*, 2005.

[62] M. Borah, R. S. Bajwa, S. Hannenhalli and M. J. Irwin, "A SIMD Solution to the Sequence Comparison Problem on the MGAP", *International Conference on Application Specific Array Processors*, 1994.

[63] D. P. Lopresti, "Rapid Implementation of a Genetic Sequence Comparator Using Field Programmable Logic Arrays", *Conference on Advanced Research in VLSI*, pages 138–152, 1991.

[64] A. D. Blas et al., "The UCSC Kestrel Parallel Processor", *IEEE Transactions on Parallel and Distributed Systems*, vol. 16(1), pages 80–92, 2005.

[65] A. Schroder et al., "Bio-Sequence Database Scanning on a GPU" *HICOMB*, 2006.

[66] M. Gok and C. Yilmaz, "Efficient Cell Designs for Systolic Smith-Waterman Implementation", *FPL 2006*.

[67] S. Vassiliadis et al., "The Molen Polymorphic Processor", *IEEE Transactions on Computers*, vol. 53(11), pages 1363–1375, November 2004.

[68] L. Hasan and Z. Al-Ars, "Performance Improvement of the Smith-Waterman Algorithm", *Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2007)*, Veldhoven, The Netherlands, November 29–30, 2007.

[69] E. M. Panainte, "The Molen Compiler for Reconfigurable Architectures", *Ph.D. Thesis*, Computer Engineering Laboratory, Technical University Delft, The Netherlands, 2007.

[70] T. Oliver, B. Schmidt and D. Maskell, "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs", *FPGA'05*, Monterey, California, USA, February 20–22, 2005.

[71] Z. Nawaz et al., "Recursive Variable Expansion: A Loop Transformation for Reconfigurable Systems", *International Conference on Field-Programmable Technology 2007*, Kokurakita, Kitakyushu, JAPAN, December 2007.

[72] Z. Nawaz, M. Shabbir, Z. Al-Ars and K. L. M. Bertels, "Acceleration of Smith-Waterman Using Recursive Variable Expansion", 11^{th} *Euromicro Conference on Digital System Design 2008*, Parma, Italy, September 2008.

[73] Fermi™ "NVIDIA's Next Generation CUDA™ Compute Architecture", *White paper NVIDIA Corporation*, 2009.

[74] http://www.khronos.org/opencl.

[75] http://www.nvidia.com/object/cuda_directcompute.html.

[76] Y. Liu, W. Huang, J. Johnson and S. Vaidya, "GPU Accelerated Smith-Waterman", *International Conference on Computational Science, ICCS 2006*, University of Reading, UK, May 28–31 2006.

[77] S. A. Manavski and G. Valle, "CUDA Compatible GPU Cards as Efficient Hardware Accelerators for Smith-Waterman Sequence Alignment", *BMC Bioinformatics*, vol. 9, Suppl 2:S10, 2008.

[78] A. Akoglu and G. M. Striemer, "Scalable and Highly Parallel Implementation of Smith-Waterman on Graphics Processing Unit using CUDA", *Cluster Computing*, vol. 12(3), pages 341–352, 2009.

[79] Y. Liu, D. Maskell and B. Schmidt, "CUDASW++: Optimizing Smith-Waterman Sequence Database Searches for CUDA-enabled Graphics Processing Units", *BMC Research Notes*, vol. 2(1):73, 2009.

[80] Y. Liu, B. Schmidt and D. Maskell, "CUDASW++2.0: Enhanced Smith-Waterman Protein Database Search on CUDA-enabled GPUs based on SIMT and Virtualized SIMD Abstractions", *BMC Research Notes*, vol. 3(1):93, 2010.

[81] "http://www.uniprot.org", *Universal Protein Resource*, April 2010.

[82] M.A. Kentie, "Biological Sequence Alignment Using Graphics Processing Units", *M.Sc. Thesis CE-MS-2010-35*, Computer Engineering Laboratory, Technical University Delft, The Netherlands, 2010.

[83] "UVa Fasta Server", *http://fasta.bioch.virginia.edu*, February 2011.

[84] M. Farrar, "Striped Smith-Waterman Speeds Database Searches Six Times over other SIMD Implementations", *Bioinformatics*, vol. 23(2), pages 156–161, 2007.

[85] "NVIDIA", *Nvidia cuda best practices guide 3.0*, 2010.

[86] http://www.yourgenome.org.

[87] http://en.wikipedia.org/wiki/.

[88] A. J. Gibbs and G. A. McIntyre, "The Diagram, a Method for Comparing Sequences, Its Use with Amino Acid and Nucleotide Sequences", *European Journal of Biochemistry*, vol. 16, pages 1–11, 1970.

[89] L. Shang, A. S. Kaviani and K. Bathala, "Dynamic Power Consumption in Virtex^{TM}-II FPGA Family", *FPGA'02*, Monterey, California, USA, February 24–26, 2002.

[90] G. Yeap, "Practical Low Power Digital VLSI Design", Kluwer Academic Publishers, 1998.

[91] http://www.bioperf.org/.

[92] Y. Yu, L. A. Santat and S. Choi, "Bioinformatics Packages for Sequence Analysis", *Bioinformatics*, vol. 6, pages 143–160, 2006.

Publications

Book chapter:

1. L. Hasan and Z. Al-Ars, "An Overview of Hardware-based Acceleration of Biological Sequence Alignment", *Accepted for publication as a book chapter in Bioinformatics*, 2011, ISBN 978-953-307-269-2.

Journal:

1. L. Hasan, M. Kentie and Z. Al-Ars, "DOPA: GPU-based Protein Alignment Using Database and Memory Access Optimizations", *Submitted to BMC Bioinformatics*, 2011, ISSN 1471-2105.

International conferences/workshops:

1. L. Hasan, M. Kentie and Z. Al-Ars, "GPU-Accelerated Protein Sequence Alignment", *Submitted to 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '11)*, Boston, USA, August 30–September 03, 2011.

2. L. Hasan, Z. Al-Ars, M. Taouil and K. L. M. Bertels, "Performance and Bandwidth Optimization for Biological Sequence Alignment", 5^{th} *International Design and Test Workshop (IDT'10)*, Pages 155–160, Abu Dhabi, UAE, December 14–15, 2010.

3. L. Hasan, Z. Al-Ars and M. Taouil, "High Performance and Resource Efficient Biological Sequence Alignment", 32^{nd} *Annual International Conference of the IEEE EMBS*, Pages 1767–1770, Buenos Aires, Argentina, August 31–September 4, 2010.

4. L. Hasan and Z. Al-Ars, "An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm", 31^{st} *Annual International Conference of the IEEE EMBS*, Pages 3845–3848, Minneapolis, Minnesota, USA, September 2009.

5. L Hasan, Z. Al-Ars, Z. Nawaz and K.L.M. Bertels, "Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion", 3rd *International Design and Test Workshop IDT08*, Pages 135–140, Monastir, Tunisia, December 2008.

6. L Hasan, Y.M. Khawaja and A. Bais, "A Systolic Array Architecture for The Smith-Waterman Algorithm With High Performance Cell Design", *IADIS European Conference on Data Mining*, Pages 35–42, Amsterdam, The Netherlands, July 2008.

7. L Hasan and Z. Al-Ars, "Accurate Profiling and Acceleration Evaluation of the Smith-Waterman Algorithm using the MOLEN Platform", *International Conference on Applied Computing*, Pages 188–194, Algarve, Portugal, April 2008.

8. L Hasan, Z. Al-Ars and S. Vassiliadis, "Hardware Acceleration of Sequence Alignment Algorithms – An Overview", *International Conference on Design and Technology of Integrated Systems in Nanoscale Era*, Pages 92–97, Rabat, Morocco, September 2007.

Local conferences/workshops:

1. L. Hasan and Z. Al-Ars, "Power Consumption Evaluation for Biological Sequence Alignment", 1st *STW.ICT Conference*, Pages 1–6, Veldhoven, The Netherlands, November 18–19, 2010.

2. L. Hasan and Z. Al-Ars, "Performance Comparison between Linear RVE and Linear Systolic Array Implementations of the Smith-Waterman Algorithm", *Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2009)*, Pages 451–456, Veldhoven, The Netherlands, November 2009.

3. L Hasan, Z. Al-Ars and Z. Nawaz, "A Novel Approach for Accelerating the Smith-Waterman Algorithm using Recursive Variable Expansion", *Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2008)*, Pages 40–45, Veldhoven, The Netherlands, November 2008.

4. L Hasan and Z. Al-Ars, "Performance Improvement of the Smith-Waterman Algorithm", *Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2007)*, Pages 211–214, Veldhoven, The Netherlands, November 2007.

Curriculum Vitae

Laiq Hasan was born on the 11^{th} of April, 1976 in Karnal Sher Killi, Swabi, Pakistan. He completed all his education, prior to his PhD, in Pakistan. For his primary and high schooling, he attended the Govt. Primary School Karnal Sher Killi 1981 - 1986, and Govt. High School Karnal Sher Killi 1986 - 1992. For his higher secondary school studies, he attended the prestigious Islamia College Peshawar 1992 - 1994. He did his B.Sc. in Electrical Engineering from *N-W.F.P. University of Engineering and Technology Peshawar (UET Peshawar)* 1995 - 2000 and subsequently his M.Sc. in Computer Information Systems Engineering from the same university in the period 2001 - 2003. While doing his M.Sc., he was working as a junior lecturer in UET Peshawar, 1st in the Electrical Engineering Department and then in the *Department of Computer Systems Engineering (DCSE)*. He was appointed as an Assistant Professor in DCSE in the year 2003. In the year 2005, he was awarded a scholarship by the Higher Education Commission of Pakistan for pursuing his PhD studies in The Netherlands. Since September 16, 2005, he has been in the Netherlands, pursuing his PhD studies in Computer Engineering Laboratory, at the Technical University Delft.

During his PhD, he worked on accelerating bioinformatics applications, designed FPGA and GPU based hardware accelerators for biological sequence alignment and carried out a comprehensive and elaborate theoretical analysis of crucial parameters like performance, computational resources, power and bandwidth limitations. He has presented various scientific papers in local and international conferences related to computer engineering and/or bioinformatics. Additionally, he published a journal paper and a book chapter in the same field and supervised two master thesis projects. This thesis is mainly based on the published papers relevant to the thesis topic.

He takes part in a variety of sports like cricket, volleyball, tennis and swimming. He enjoys expeditions like sailing, hiking and trekking. He also likes walking and riding his bike for hours. Reading newspapers, watching news and sports TV channels, and learning about different societies, cultures and languages are his main hobbies. Tourism and exploring new places are his passions. Making good friends, talking to diversified people and enjoying a variety of food and drinks are his additional interests.