Acceleration of the Smith-Waterman algorithm for DNA sequence alignment using an FPGA platform

Barry Strengholt Matthijs Brobbel

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Computer Engineering

June 27, 2013

Abstract

With the sequencing of DNA becoming cheaper and the resulting stack of data growing bigger, there is a big challenge for both engineer and biologist. Researchers are limited by their computational power.

In this thesis, first an overview of sequence alignment algorithms will be given. Then a method to store the values of the similarity score matrix of the Smith-Waterman algorithm differentially will be presented. And finally, a description of the system approach used to design an accelerator, which implements this method, will be given.

Implementation of the system design on an Artix-7 XC7A200T-2C FPGA, could lead to a total performance of 94 GCU/s. This is a speed up of 5x compared to conventional CPUs.

Preface

This bachelor thesis was written as part of the bachelor end project of the Electrical Engineering bachelor at the University of Technology in Delft, for the department of Computer Engineering.

During this project we developed a low-cost hardware accelerator for the Smith-Waterman algorithm, which can be used for DNA sequence alignment.

A group of five students worked on this project, and two theses were written. This thesis will describe the work we did on the project, namely the literature study on the algorithms, acceleration techniques, hardware platforms and interconnect alternatives. The other thesis, written by Matthijs Geers, Fatih Han Çağlayan and Rolf Heij, describes the work of the other subgroup, namely the implementation of the accelerator design in VHDL [1].

We would like to thank Zaid Al-Ars, our supervisor, for all his support and suggestions. And we would also like to thank Anton Wallén for his inspiring work that kept us motivated throughout the project.

Barry Strengholt & Matthijs Brobbel
June 27, 2013

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction to the thesis

In this thesis an implementation of Smith-Waterman on an FPGA will be designed. This will be done with a top-down approach. To do this first some background theory about biology and sequence alignment will be discussed. Then some acceleration techniques, hardware platforms and interconnect alternatives will be discussed. After this the thesis will get to an implementation with a system design approach. The theory discussed in the first chapter will be used to underpin the choices made for the system design and implementation.

In this chapter three subjects are discussed. First, information is given about some biology topics after which genetic research is discussed, and finally a problem definition will be made.

1.2 Biology

In this section a bottom-up approach is used to give some information about biology, which is the study of life and living organisms.

1.2.1 Proteins

Proteins are large biological molecules that contain carbon, hydrogen, oxygen and nitrogen. Some proteins also contain sulphur. Proteins have many roles inside a living organism. They are responsible for a lot of functions within a living organism. Enzymes, for example, are proteins that speed up biochemical reactions.

The function of a protein is primarily defined by its shape and sequence of amino acids. Amino acids are the monomers of proteins. Each of the 20 different amino acids has a different side chain which gives the amino acid its distinctive chemical identity. Amino acids can be represented by a letter, thus proteins can be represented by a sequence of letters.

The process in which proteins are assembled is called synthesis. Information encoded in genes defines which amino acids are used to assemble a protein. The sequence of amino acids is specified by a nucleotide sequence. Nucleotides are

composed of nucleobases and they are the building blocks of nucleic acids i.e. nucleotides are the monomers of nucleic acids [2].

1.2.2 Nucleic acids

Deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) are nucleic acids. They consist of a nucleobase, a pentose sugar and a phosphate group. The nucleobases found in DNA are Cytosine (C), Guanine (G), Adenine (A) and Thymine (T). The nucleobases found in RNA are Cytosine (C), Guanine (G), Adenine (A) and Uracil (U) [2], see Figure 1.1.

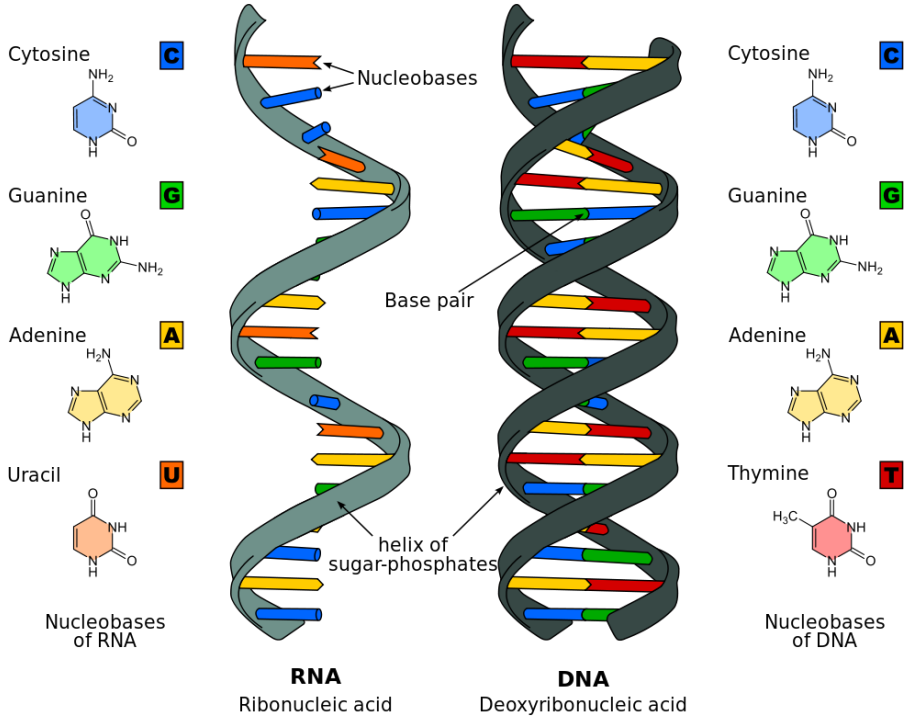

Figure 1.1: RNA a DNA, including their nucleobases.

DNA

DNA forms the inherited genetic material inside each cell of a living organism. Each segment in DNA which encodes for a protein is called a gene. Genes control the protein synthesis and regulate most of the activities inside a living organism. All the information is copied when a cell divides.

In DNA two nucleobases pair to form a base pair, which, together with the phosphate groups and deoxiribose sugars, form a spiral ladder, also called a double helix.

Every time DNA is copied the two strands unwind. When a change occurs in the base sequence of a DNA strand, it is called a mutation. These mutation can lead to diseases or the death of a cell. If, after mutation, no disease or death

occurs, there is a chance that the mutation will be passed through on offspring [2].

RNA

RNA relays information from genes to guide each cell's synthesis of proteins from amino acids. Three types of RNA can be distinguished, messenger RNA, ribosomal RNA and transfer RNA. Each of these types RNA carry out a special role in the synthesis process.

RNA differs from DNA in several aspects. RNA is single stranded in humans whereas DNA is double stranded. Another difference is the nucleobases of which they are formed, as discussed earlier [2].

1.2.3 Cells

Cells are the fundamental units of structure and function in every living organism. Some organisms exist out of only one cell while other organisms consist of trillions of cells. Cells multiply by dividing themselves.

Inside a cell there are many subcellular components. An organelle is a specialised structure inside a cell, that has a characteristic shape, and which performs a specific function.

It is possible to distinguish three main parts of a cell. The nucleus, which is a large organelle, holds most of a cell's genetic information, the plasma membrane, which forms the flexible outer surface of a cell, and the cytoplasm, which consist of all the cellular components between the plasma membrane and the nucleus [2], as shown by 1, 2 and 3 in Figure 1.2.

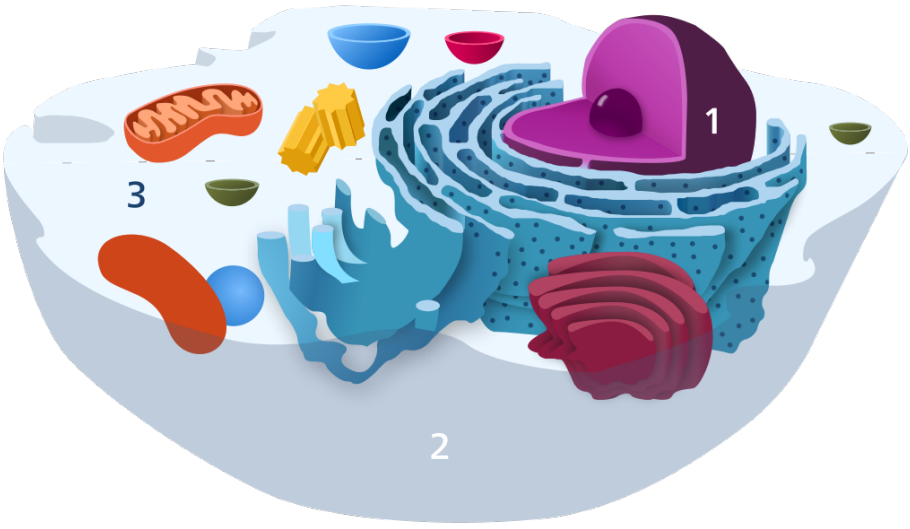

Figure 1.2: Schematic of an animal cell, showing subcellular components.

Nucleus

The Nucleus is the core of a cell. Inside the nucleus are most of the cell's hereditary units, called genes. Genes control the cellular structure and control cellular activities. Genes are arranged along chromosomes. The total genetic information carried inside a cell or living organism is called the genome [2].

1.3 Genetic research

Nowadays a lot of genetic research is going on. In this section DNA sequencing and DNA sequence analysis is discussed. Other topics of genetic research are not discussed here.

1.3.1 DNA sequencing

DNA sequencing is a process in which the order of nucleotides is determined for a DNA molecule. Nowadays the order of the nucleobases is determined by a lot of different techniques. All techniques have a different read length, which determines the maximal sequence length that can be read in one run.

Sequencing can be used to determine the sequence of nucleobases of a complete genome, a chromosome or a certain gene.

Next-generation sequencing (NGS) are promising new techniques to sequence DNA [3]. With one of these new techniques the DNA is reproduced multiple times as small parts of RNA [4]. These small sequences are then read with silicon based technology.

1.3.2 DNA sequence analysis

The sequences found by DNA sequencing can be used by researchers, interested in molecular biology and genetics, forensics and medical personnel. Researchers, for example, can use the information to learn and better understand certain processes in cells. Medical personnel could try to find genes or expressions in the DNA sequence which correspond to certain diseases or disorders. Forensics use DNA sequencing to prove if someone committed a crime by comparing the DNA sequence of a sample found at a crime scene, and the DNA sequence of a suspect.

The comparison of DNA sequences is possible with sequence alignment algorithms. These algorithms will be discussed in Chapter 2. By aligning two sequences of DNA it is possible to find where the differences and similarities are. This might be useful, for example, to determine which expression may cause a certain disease or disorder, or to determine which gene or genes encode for a certain protein.

1.4 Problem definition

In this section a problem definition will be made. First some aspects of the problem will be discussed.

1.4.1 Big data

The price for which DNA can be sequenced is dropping, whereas the rate, in which DNA can be sequenced, is growing at a slower rate. A human genome is normally stored in 23 chromosome pairs. These chromosomes contain all the genes, for a human about 20000 genes.

The total human genome consists of 3.2×10^9 base pairs [5]. However the data from a NGS run on one persons DNA gives 120 to 600 Gigabytes of data [6]. With DNA sequencing getting cheaper and faster, the total amount of available DNA sequences grows and so does the amount of data. Having a big unstructured stack of data is referred to as a big data problem.

1.4.2 Moore's law

In 1965 Moore predicted that the number of transistors on integrated circuits double every 18 months [7]. This is important because currently the cost of DNA sequencing is dropping faster than the cost of processing the resulting data on computers [8][9]. This can also be seen in Figure 1.3.

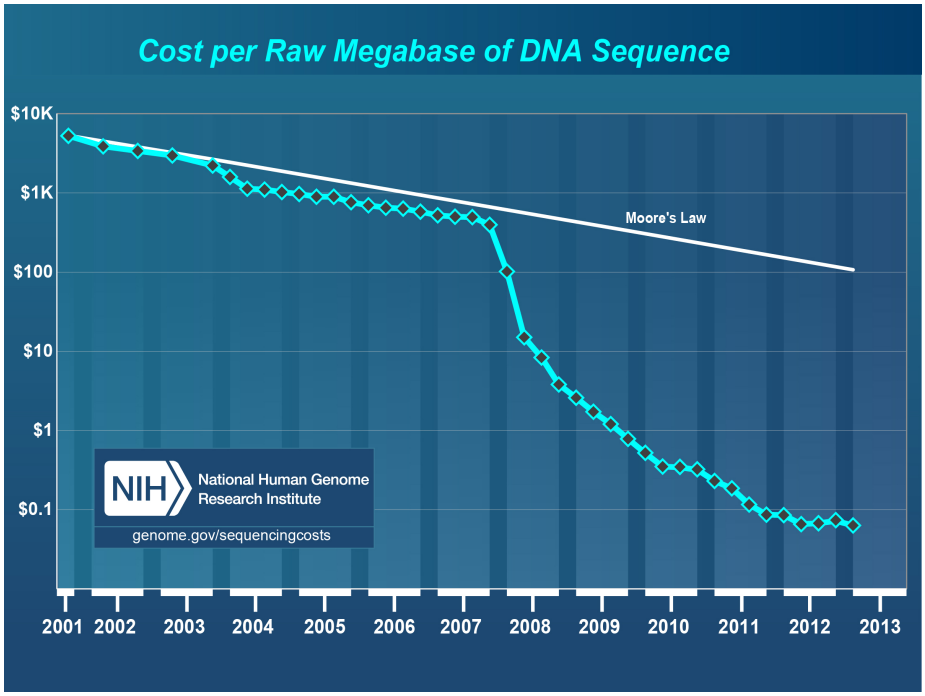

Figure 1.3: DNA sequencing costs: data from the NHGRI Genome Sequencing Program (GSP), cost per raw Megabase of DNA sequence [9].

1.4.3 Problem definition

With DNA sequencing becoming cheaper and the stack of unsorted data growing bigger, there is a challenge for engineers and biologists to efficiently deal with the available data. Analysis of this enormous amount of data could lead to a better

understanding of life and living organisms. For the analysis of RNA and DNA sequences, sequence alignment algorithms are being used. These algorithms are time consuming, because (1) the amount of sequences can be very high, because (2) the length of these sequences can be very long, and because (3) the algorithms used to align the sequences are quadratic in time with the length of the sequences.

The problem is that some researchers, who work with long sequences, have to wait days or hours for an alignment. The lack of computational power slows their research down. The aim of this project is to analyse sequence alignment algorithms and how they can be efficiently implemented on a low-cost hardware platform.

Chapter 2

Sequence alignment algorithms

In this chapter a classification of sequence alignment algorithms will be made and five algorithms will be discussed in detail.

Sequence alignment is a way of arranging two sequences to identify regions of similarity [10]. Sequence alignment algorithms exist to find the optimal alignment of two sequences. Aligning a sequence into another sequence can be done with recursively replacing, inserting or removing an element. Each of these operations have an associated score. Sequence alignment algorithms find the optimal alignment i.e. the sequence of edits with the highest total score. An example is shown in Figure 2.1.

$$\begin{array}{cccccccccc} A & - & C & A & C & A & C & T & A \\ | & & | & | & | & | & | & & | \\ A & G & C & A & C & A & C & - & A \end{array}$$

Figure 2.1: Sequence alignment.

2.1 Classification

There are several sequence alignment algorithms. In this section a classification will be made based on properties of these algorithms. First the difference between dynamic programming algorithms and heuristics will be discussed, followed by the difference between global alignment and local alignment. After that, linear and affine gap penalty functions and divide and conquer algorithms will be discussed.

2.1.1 Dynamic programming and heuristics

Dynamic programming

Dynamic programming is a method for finding the optimal solution to problems by breaking them down into smaller subproblems [11]. The dynamic programming method can only be applied to problems with overlapping subproblems.

The solutions of the subproblems are being combined to find the overall solution. Once the solution to a subproblem has been found, it is memoized to prevent computing the same subproblem twice.

Dynamic programming algorithms find the optimal solution by examining all possible ways to solve a problem. A sequence alignment algorithm is an example of a dynamic programming algorithm. Finding all possible alignments is necessary to find the guaranteed optimal alignment of two sequences.

Heuristics

A heuristic is a problem solving method based on experiences, findings and discoveries. It gives a solution which is not guaranteed to be the optimal solution. By making assumptions about where or how to search for the optimal solution, a speedup of the original problem solving process can be achieved. Because dynamic programming algorithms examine all possible ways to solve a problem this can be a time consuming process. A reduction in computation time is possible by only examining the most probable ways to solve a problem.

An example of a heuristic approach for sequence alignment is the Basic Local Alignment Tool (BLAST) [12]. This sequence alignment algorithm is based on the Smith-Waterman algorithm, which will be discussed later in this chapter. There are lot of variations, improvements and hardware accelerate implementations available for this alignment tool [13][14][15][16][17][18][19]. The way BLAST or these other heuristic methods work will not be discussed.

2.1.2 Global and local

Given two sequences, A ={ACTAGC} and B ={TATCTGCCGT}, it is possible to align them globally 2.2 or locally 2.3.

$$\begin{array}{cccccccccc} - & A & - & C & T & A & G & C & - & - \\ & | & & | & | & : & & | & & \\ T & A & T & C & T & G & C & C & G & T \end{array}$$

Figure 2.2: Global alignment.

$$\begin{array}{cccccc} A & C & T & A & G & C \\ : & | & | & & | & | \\ T & C & T & - & G & C \end{array}$$

Figure 2.3: Local alignment.

Global

When aligning sequences globally, the optimal alignment is the alignment where the total number of matches is maximal. All characters in both sequences participate in the alignment. This method is useful when comparing closely related sequences.

Local

Local alignment is used to find related regions in two sequences. This method is much more flexible than the global alignment method. Related regions that appear in different order can still be identified as being related whereas this is not possible with the global alignment method.

2.1.3 Gap penalties

The first sequence alignment algorithms were defined with a linear gap penalty function, which represented the cost of inserting a gap into one of the sequences. Later affine gap functions were introduced to make the sequence alignment algorithms better applicable to biological sequences [20][21].

Linear gap penalty

A linear gap penalty function $w(k)$ can be defined as a function of gap length $k \geq 0$:

$$w(k) = uk \tag{2.1}$$

with $u \geq 0$ the gap penalty.

Affine gap penalty

An affine gap penalty function $w(k)$ can be defined as a function of gap length $k \geq 0$:

$$w(k) = uk + v \tag{2.2}$$

with $v \geq 0$ the gap open penalty and $u \geq 0$ the gap extension penalty.

When DNA gets replicated it is possible that the replica is not an exact duplicate. This might happen due to deletion or insertion of a base pair.

In biological sequences it is not very likely that a gap occurs, however when this happens, it is very likely that more than one base pairs are inserted. By using an affine gap function it is possible to take this effect into account. For biological sequences it makes sense to have a relatively big gap open penalty and a smaller gap extend penalty i.e. $v > u$.

2.1.4 Divide and conquer

Divide and conquer is a programming technique to solve problems by recursively dividing the problem into smaller subproblems. These smaller subproblems are divided until the solution to the problem is trivial. Then the answer to the subproblems is combined to get the answer. It is important to note that dynamic programming is a method that applies the divide and conquer technique. The difference are the memoization of solutions to subproblems and the solution being the optimal solution.

2.2 Global alignment algorithms

In this section two global alignment algorithms will be discussed in detail.

2.2.1 Needleman-Wunsch

In 1970 Needleman and Wunsch presented an algorithm to search for similarities in the amino acid sequence of two proteins [22]. The algorithm returns the alignment of two sequences for which the number of matches is maximal. With a dynamic programming approach a similarity score matrix is built. From this matrix the optimal global alignment can be found. The algorithm uses a linear gap penalty function $w(k)$ as defined earlier in Equation (2.1).

To find the optimal local alignment of two sequences A and B with:

$$\begin{array}{rcl} A & = & a_1 \cdots a_i \cdots a_n \\ B & = & b_1 \cdots b_j \cdots b_m \end{array} \tag{2.3}$$

A similarity score matrix S is built as follows:

$$S_{i,j} = \begin{cases} w(j), & i = 0, j \geq 0 \\ w(i), & i > 0, j = 0 \end{cases} \tag{2.4}$$

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(a_i, b_i) \\ S_{i,j-1} - w(1) \\ S_{i-1,j} - w(1) \end{cases} \tag{2.5}$$

with $s(a_i, b_j)$ the similarity score function defined as:

$$s(a_i, b_j) = \begin{cases} > 0, & a_i = b_j \quad \text{Match} \\ < 0, & a_i \neq b_j \quad \text{Mismatch} \end{cases} \tag{2.6}$$

The optimal alignment can be traced back starting at the highest value in the last row or last column and then iteratively selecting the highest score of the previous rows and columns until the origin is reached.

The Needleman-Wunsch algorithm runs in $O(nm)$ time and has a space complexity of $O(nm)$.

2.2.2 Hirschberg

In 1975 Hirschberg presented an algorithm to find the optimal global alignment of two sequences in quadratic time and linear space [23]. By applying this technique to the Needleman-Wunsch algorithm, a linear space requirement $O(n+m)$ is possible.

The Hirschberg technique is based on the data dependencies in the similarity score matrix S. First note that in the similarity score matrix S of Needleman-Wunsch the values in the $j - 1^{th}$ column are based on the j^{th} column i.e. to calculate the values of the j^{th} column one only needs to know the values of the $j - 1^{th}$ column. The data dependencies in the similarity score matrix will be discussed in more detail later. An example is shown in Figure 2.4.

The S matrix

	A	C	T
A	1	1	1
A	1	1	1
C	0	2	1
T	0	1	3

The first two columns

	A	C
A	1	1
A	1	1
C	0	2
T	0	1

The last two columns

	C	T
A	1	1
A	1	1
C	2	1
T	1	3

Figure 2.4: The Hirschberg technique.

2.3 Local alignment algorithms

In this section three local alignment algorithms will be discussed in detail.

2.3.1 Smith-Waterman

In 1981 Smith and Waterman presented an algorithm to find the optimal local alignment of two sequences [24]. It is a dynamic programming algorithm like the the Needleman-Wunsch algorithm of which it is a variation. The main difference between the Needleman-Wunsch algorithm and the Smith-Waterman algorithm is that negative values in the similarity score matrix are set to zero.

To find the optimal local alignment of two sequences A and B, a similarity score function $s(a_i, b_j)$ and a linear gap penalty function $w(k)$ are used, as defined in Equations (2.3), (2.6) and (2.1).

A similarity score matrix S matrix is built as follows:

$$S_{i,j} = 0, \begin{cases} 0 \leq i \leq n \\ 0 \leq j \leq m \end{cases} \tag{2.7}$$

$$S_{i,j} = \max \begin{cases} 0 \\ S_{i-1,j-1} + s(a_i, b_i) \\ S_{i-1,j} - w(1) \\ S_{i,j-1} - w(1) \end{cases} \tag{2.8}$$

To obtain the optimal local alignment a trace-back procedure is used. To do this, first find the maximum value of S. Then start moving back to the origin, choosing the highest value with each step until a zero is encountered. Finally the optimal alignment can be constructed by following this path where each step corresponds with either a replacement, insertion or deletion i.e. the element of the sequence is inserted, a gap gets inserted in sequence A or a gap gets inserted in sequence B.

The Smith-Waterman algorithm runs in $O(nm)$ time and has a space complexity of $O(nm)$.

Smith-Waterman example

Given two DNA sequences: A ={GGTGCGATAT} and B ={GCGTGGGA}, $s(a_i, b_j)$ the similarity score function defined as:

$$s(a_i, b_j) = \begin{cases} 2, & a_i = b_j \quad \text{Match} \\ -1, & a_i \neq b_j \quad \text{Mismatch} \end{cases} \tag{2.9}$$

and a linear gap penalty function as defined in Equation (2.1) with $u = 1$, a similarity score matrix S can be initialised as follows from Equation (2.7):

$S_{i,j} =$

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

Figure 2.5: The initialised similarity score matrix S.

By recursively applying Equation (2.8), the similarity score matrix S can be filled as follows:

$S_{i,j} =$

	-	G	G	T	G	C	G	A	T	A	T
-	0	0	0	0	0	0	0	0	0	0	0
G	0	2	2	1	2	1	2	1	0	0	0
C	0	1	1	1	1	4	3	2	1	0	0
G	0	2	3	2	3	3	6	5	4	3	2
T	0	1	2	5	4	3	5	5	7	6	5
G	0	2	3	4	7	6	5	4	6	6	5
G	0	2	4	3	6	6	8	7	6	5	5
G	0	2	4	3	5	5	8	7	6	5	4
A	0	1	3	3	4	4	7	10	9	8	7

Figure 2.6: The filled similarity score matrix S.

The calculation of a cell is discussed here by applying Equation (2.8) to a cell of the similarity score matrix S shown in Figure 2.6.

$$\begin{array}{ccccc} & & & A & \\ & \ddots & \vdots & \vdots & \iddots \\ & \cdots & 5 & 4 & \cdots \\ G & \cdots & 8 & S_{6,7} & \cdots \\ & \iddots & \vdots & \vdots & \ddots \end{array} \tag{2.10}$$

Applying Equation (2.8) for $S_{6,7}$ results in:

$$S_{6,7} = \max \begin{cases} 0 \\ H_{5,6} + s(G,A) \\ H_{5,7} - w(1) \\ H_{6,6} - w(1) \end{cases} = \max \begin{cases} 0 \\ 5 + -1 \\ 4 - 1 \\ 8 - 1 \end{cases} = 7 \qquad (2.11)$$

To find the local optimal alignent the trace-back procedure, as described earlier, is applied to the similarity score matrix S. This results in:

$S_{i,j} =$

	-	G	G	T	G	C	G	A	T	A	T
-	0	0	0	0	0	0	0	0	0	0	0
G	0	2	2	1	2	1	2	1	0	0	0
C	0	1	1	1	1	4	3	2	1	0	0
G	0	2	3	2	3	3	6	5	4	3	2
T	0	1	2	5	4	3	5	5	7	6	5
G	0	2	3	4	7	6	5	4	6	6	5
G	0	2	4	3	6	6	8	7	6	5	5
G	0	2	4	3	5	5	8	7	6	5	4
A	0	1	3	3	4	4	7	10	9	8	7

Figure 2.7: The trace-back procedure for the Smith-Waterman algorithm.

Reconstruction of the aligned sequences requires application of the trace-back procedure to find a path as shown in Figure 2.7. The start point of the path is the maximal score in the similarity score matrix S. The path continues in the direction of the origin of the matrix i.e. $S_{0,0}$, choosing the maximum value each step, until a 0 is encountered.

The next set of rules is applied to each step in the path to find the aligned sequences.

- A step diagonally up corresponds with a replacement i.e. a match or mismatch
- A step towards the left corresponds with a deletion i.e. a gap in A
- A step upwards corresponds with an insertion i.e. a gap in B

This results in the aligned sequences as shown in Figure 2.8.

2.3.2 Gotoh

In 1982 Gotoh presented an improved algorithm for matching biological sequences [20][25][21]. The Gotoh algorithm is based on the Smith-Waterman algorithm and finds the local optimal alignment of two sequences with an affine gap penalty function.

$$\begin{array}{cccccccccc} - & - & G & G & T & G & C & - & G & A \\ & & & | & | & & : & & | & | \\ G & C & - & G & T & - & G & G & G & A \end{array}$$

Figure 2.8: Optimal local alignment, found with the Smith-Waterman algorithm, of the example sequences, A and B.

The algorithm uses the same sequences A and B, affine gap penalty function $w(k)$ and similarity score function $s(a_i, b_j)$ as defined earlier in Equations (2.3), (2.2) and (2.6).

The matrices D, P and Q are initialised as follows:

$$D_{i,j} = \begin{cases} 0, & i = 0, j = 0 \\ w(j), & i = 0, j > 0 \\ w(i), & i > 0, j = 0 \end{cases} \tag{2.12}$$

and for $j > 0$:

$$P_{0,j} = \infty \tag{2.13}$$

$$Q_{j,0} = \infty \tag{2.14}$$

Then the matrices D, P and Q are generated by recursion as follows:

$$D_{i,j} = \min_{i,j \geq 1} \begin{cases} D_{i-1,j-1} + s(a_i, b_i) \\ P_{i,j} \\ Q_{i,j} \end{cases} \tag{2.15}$$

where

$$P_{i,j} = \min_{1 \leq i \leq n, 1 \leq j \leq m} \begin{cases} D_{i-1,j} - w(1) \\ P_{i-1,j} - u \end{cases} \tag{2.16}$$

and

$$Q_{i,j} = \min_{1 \leq i \leq n, 1 \leq j \leq m} \begin{cases} D_{i,j-1} - w(1) \\ Q_{i,j-1} - u \end{cases} \tag{2.17}$$

Also note:

$$\begin{array}{rcl} A_i & = & a_1 a_2 \cdots a_i \\ B_j & = & b_1 b_2 \cdots b_j \end{array} \tag{2.18}$$

The D matrix holds the cost for alignment of prefixes A_i and B_j, the P matrix holds the cost for alignment of prefixes A_i and B_j that ends with a gap in B and the Q matrix holds the cost for alignment of prefixes A_i and B_j that ends with a gap in A.

To find the optimal local alignment a trace-back procedure is needed. This procedure will not be discussed.

The Gotoh algorithm runs in $O(nm)$ time and has a space complexity of $O(nm)$.

2.3.3 Miller-Myers

In 1988 Miller and Myers presented an algorithm to find the optimal local alignment of two sequences with an affine gap penalty function in linear space [26]. Miller and Myers applied the Hirschberg principal, to align sequences in $O(nm)$ time and $O(n+m)$ space, to the Gotoh algorithm, both described earlier. Improvements to this algorithm were later presented by Guan and Uberbacher [27] and Chao and Miller [28]. The details on these improved algorithms will not be discussed.

The algorithm uses the same sequences A and B, affine gap penalty function $w(k)$ and similarity score function $s(a_i, b_j)$ as defined earlier in Equations (2.3), (2.2) and (2.6). The definition for A_i and B_j as defined in Equation (2.18) is also used.

First an affine gap penalty function $gap(k)$ is defined as:

$$gap(k) = g + hk \tag{2.19}$$

where

$$\begin{array}{rcl} g & = & v \\ h & = & u + {}^1\!/_2 s(a_i, b_j)_{max} \end{array} \tag{2.20}$$

Three matrices are defined. $C_{i,j}$, which contains the minimum cost of a conversion of A_i to B_j, $D_{i,j}$ which contains the minimum cost of a conversion of A_i to B_j that deletes a_i and $I_{i,j}$ which contains the minimum cost of a conversion of A_i to B_j that inserts b_j. First the matrices are initialised as follows:

$$C_{i,j} = \begin{cases} 0, & i = 0, j = 0 \\ gap(j), & i = 0, j > 0 \\ gap(i), & i > 0, j = 0 \end{cases} \tag{2.21}$$

$$D_{i,j} = C_{0,j} + g \tag{2.22}$$

for $i = 0, j > 0$ and

$$I_{i,j} = C_{i,0} + g \tag{2.23}$$

for $i > 0, j = 0$.

Then the values in the matrices satisfy the following recurrence relations:

$$C_{i,j} = \min_{i>0,j>0} \begin{cases} D_{i,j} \\ I_{i,j} \\ C_{i-1,j-1} + s(a_i, b_j) \end{cases} \tag{2.24}$$

$$D_{i,j} = \min_{i>0,j>0} \begin{cases} D_{i-1,j} \\ C_{i-1,j} + g \end{cases} + h \tag{2.25}$$

$$I_{i,j} = \min_{i>0,j>0} \begin{cases} I_{i,j-1} \\ C_{i,j-1} + g \end{cases} + h \tag{2.26}$$

Because the values in the i^{th} rows of C and D only depend on values in the rows i and $i-1$, it is possible to use row-sized vectors instead of matrices and therefor it is possible to reach a $O(n)$ space complexity. Two vectors and three

scalars are used: CC, DD, e, c and s, the vectors and scalars respectively. They are defined as follows:

$$CC_k = \begin{cases} C_{i,k}, & k < j \\ C_{i-1,k}, & k \geq j \end{cases} \tag{2.27}$$

$$DD_k = \begin{cases} D_{i,k}, & k < j \\ D_{i-1,k}, & k \geq j \end{cases} \tag{2.28}$$

$$e = I_{i,j-1} \tag{2.29}$$

$$c = C_{i,j-1} \tag{2.30}$$

$$s = C_{i-1,j-1} \tag{2.31}$$

The divide and conquer method needed for this algorithm, to find the optimal alignment, will not be discussed because this is beyond the scope of this thesis.

2.4 Overview

In this section an overview is given for the discussed sequence alignment algorithms.

There are more sequence alignment and sequence similarity algorithms, but they all use the principles discussed in this chapter. The following algorithm were not discussed here, but they might be interesting. For example, a new algorithm by Waterman and Eggbert for best subsequence alignments with application for tRNA and rRNA was introduced in 1987 [29], a time efficient linear space local similarity algorithm was introduced by Huang was introduced in 1991 [30] and recently Zhang introduced an algorithm for the Smith-Waterman algorithm based on the divide and conquer principal [31][32].

In 1970 Needleman and Wunsch presented their algorithm to find the optimal global alignment of two sequences. Hirschberg, in 1975, presented a memory efficient way to find the optimal global alignment based on the Needleman-Wunsch algorithm. In 1981 Smith and Waterman presented their algorithm, based on the Needleman-Wunsch algorithm, to find the optimal local alignment of two sequences. An improvement came from Gotoh in 1982. He added an affine gap penalty function to the Smith-Waterman algorithm. Miller and Myers, in 1988, combined the work of Gotoh with the work of Hirschberg to present an algorithm to find the optimal local alignment of two sequences with an affine gap penalty function in a memory efficient way.

Algorithm	Classification	Time	Space
Needleman-Wunsch	Global — Linear gap penalty	O(nm)	O(nm)
Hirschberg	Global — Linear gap penalty	O(nm)	O(n+m)
Smith-Waterman	Local — Linear gap penalty	O(nm)	O(nm)
Gotoh	Local — Affine gap penalty	O(nm)	O(nm)
Miller-Myers	Local — Affine gap penalty	O(nm)	O(n+m)

Table 2.1: An overview of sequence alignment algorithms.

Chapter 3

Acceleration techniques

In this chapter some acceleration techniques will be discussed. Then memory optimisation techniques will be discussed, including a new method to store the values of the similarity score matrix differentially.

3.1 Acceleration techniques

In this section some acceleration techniques will be discussed. First parallelism and data dependencies of the Smith-Waterman algorithm are discussed to demonstrate why hardware acceleration is possible, after which linear systolic arrays and recursive variable expansion will be discussed.

Processing cores are defined here as the smallest piece of a hardware architecture which can perform an algorithm or part of an algorithm. In the literature they are sometimes being referred to as processing elements (PE).

3.1.1 Parallelism and data dependencies

Filling up the similarity score matrix S as defined for the Smith-Waterman algorithm in Equation (2.8), takes up most of the computation time when searching for the optimal local alignment of two sequences. To be able to significantly speed this process up, it should be possible to calculate different cells, of the S matrix, at the same time.

The Smith-Waterman algorithm allows for this as can be seen from the data dependencies in the similarity score matrix S. The value of a cell in the similarity score matrix S depends on the cells left, above and diagonally left above i.e. to calculate $S_{i,j}$ one needs to know $S_{i,j-1}$, $S_{i-1,j}$ and $S_{i-1,j-1}$.

$$\begin{matrix} \ddots & \vdots & \vdots \\ \cdots & S_{i-1,j-1} & S_{i,j-1} \\ \cdots & S_{i-1,j} & S_{i,j} \end{matrix} \tag{3.1}$$

This demonstrates that the Smith-Waterman algorithm allows for parallelisation. By using more than one processing core, multiple cells can be calculated at the same time [33].

However, note that it take n iterations to fully utilise n processing cores for the Smith-Waterman algorithm. This is demonstrated in Figure 3.1, where cells

with the same colour indicate cells that can be calculate at the same time e.g. all red cells can be calculated in the same iteration. Only the first 8 iterations are shown in this figure.

$S_{i,j} =$

	-	G	G	T	G	C	G	A	
-	0	0	0	0	0	0	0	0	
G	0	2	2	1	2	1	2	1	1^{st} iteration
C	0	1	1	1	1	4	3	2	2^{nd} iteration
G	0	2	3	2	3	3	6	5	3^{rd} iteration
T	0	1	2	5	4	3	5	5	4^{th} iteration
G	0	2	3	4	7	6	5	4	5^{th} iteration
G	0	2	4	3	6	6	8	7	6^{th} iteration
G	0	2	4	3	5	5	8	7	7^{th} iteration
A	0	1	3	3	4	4	7	10	8^{th} iteration

Figure 3.1: Utilisation of the processing cores for the Smith-Waterman algorithm.

It can be seen that in the first iteration only one cell can be calculated, whereas in the second iteration, two cells can be calculated.

3.1.2 Linear systolic array

A linear systolic array is an array of processing cores where each cell shares its data with the other cells in the array. Each processing core solves a subproblem and shares the solution to all the other cells in the array to prevent calculation of the same problem twice.

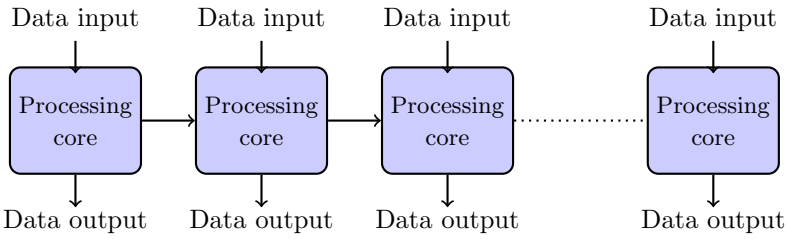

Figure 3.2: A linear systolic array of processing cores.

Linear systolic arrays are used to process dynamic programming algorithms in which solutions to subproblems depend on solutions to other subproblems. When a dynamic programming algorithm allows for parallelism, a linear systolic array can speed up the calculations significantly. Ideally, the number of subproblems solved per iterations equals the number of processing cores in the array. However, not all algorithms allow for full parallelisation from the first iteration.

From all this it can be stated that a linear systolic array can be very useful to speed solving an algorithm up, if the data dependencies of the algorithm allow

for this. Theoretically a speed up of n is reachable by using n processing cores in a linear systolic array for problems in which subproblems do not depend on solutions of other subproblems.

However, linear systolic arrays are difficult to build, because each application requires a different interconnection and structure of processing cores.

It is possible to build a linear systolic array to process the Smith-Waterman algorithm [34][35]. However, this requires to take note of the data dependencies and parallelism, discussed in the previous section. A decrease in the theoretical speedup also comes from the iterations it takes to fully utilise all the processing cores, as discussed in the previous section.

3.1.3 Recursive variable expansion

In 2007 Nawaz presented a method which allows for faster utilisation of the processing cores [36]. This method or technique is called Recursive Variable Expansion (RVE). This method makes it possible for a processing core to calculate a $n \times n$ block, of the similarity score matrix S, with $n > 1$, in one iteration. This is demonstrated here for a 2×2 block as shown in Figure 3.3.

This demonstrates that it is possible to process a 2×2 block in one iteration. It is possible to do this for $n \times n$ blocks. The downside of using this method is that the equations and therefore the processing elements grow in size. When the Smith-Waterman algorithm is used, the value of cell is determined by finding the maximum out of 4 equations, whereas this method requires to find the maximum out of 8 (more complex) equations for a 2×2 block [37].

3.2 Memory optimisation

In this section, some memory optimisations techniques will be discussed. First some insights will be given about the memory usage. Then a differential model will be discussed, and finally, data compression will be discussed.

3.2.1 Memory usage

Representation of the base pairs is possible with 2 bits, since there are 4 possible nucleobases in DNA, Cytosine (C), Guanine (G), Adenine (A) and Thymine (T). The nucleobases will represented as shown in Table 3.1.

Nucleobase	Representation
Cytosine (C)	00
Guanine (G)	01
Adenine (A)	10
Thymine (T)	11

Table 3.1: Representation of the nucleobases found in DNA.

All sequence alignment algorithms compare sequences and store their alignment scores in one or several matrices. For the Needleman-Wunsch and Smith-Waterman algorithms, the space complexity is $O(nm)$. This means that the size of the matrices used for these algorithms grows quadratic with the length of the

sequences. As a result, the amount of storage needed to store these matrices can grow beyond the available memory in conventional systems. For example, the human chromosome 1 consists of 121 Megabase pairs. When two of these chromosomes are aligned and stored in cells with 32 bit integers, the amount of cells in the matrix is $(121 \cdot 10^6)^2$. Multiplication of this with the bit width of each cell gives the storage size needed for the whole matrix. For this example the amount of memory needed to store the matrix would be 468.5 Peta bits. Nowadays sequences of these sizes are not aligned with the Needleman-Wunsch or Smith-Waterman algorithms, but rather with the linear space algorithms as discussed earlier.

3.2.2 Differential model

In this subsection a space efficient way to store the similarity score matrix for the Needleman-Wunsch algorithm and the Smith-Waterman algorithm will be shown [38]. First by proving a way to store the data differentially, then by demonstrating the correctness of this model for the Smith-Waterman algorithm.

Definitions

The Needleman-Wunsch algorithm and the Smith-Waterman algorithm define a score matrix S with the value of a cell, with columns $i \in \mathbb{N}$ and row $j \in \mathbb{N}$, given by $S_{i,j}$, as can be seen in see Equation (3.1).

The Smith-Waterman algorithm initialises the similarity score matrix S as stated before in Equation (2.7).

Lets define two differential variables $\delta^J_{i,j}$ and $\delta^I_{I,j}$, with I and J describing the direction in the S matrix i.e. I corresponds to the difference between $S_{i,j}$ and the cell left of $S_{i,j}$ and J corresponds with the difference between $S_{i,j}$ and the cell above $S_{i,j}$. For $i, j \geq 1$ these differential variables can be defined as:

$$\delta^J_{i,j} = S_{i,j} - S_{i,j-1} \tag{3.2}$$

$$\delta^I_{i,j} = S_{i,j} - S_{i-1,j} \tag{3.3}$$

Representation proof

The absolute values of the similarity score matrix can be found if the similarity score matrix is represented by the differential values only. In this subsection a proof by induction will be given. More specifically phrased: proof by induction will be used to show that when the differential values between the cells are stored, the absolute values of every cell in the similarity score matrix can be calculated i.e. if Equation 3.4 is true.

$$S_{i,j} = \sum_{1=k}^{j} \delta_{i,k} \tag{3.4}$$

Proof by induction

Basis step For the basis step it is required to prove:

$$S_{i,1} = \sum_{1=k}^{1} \delta_{i,k} \tag{3.5}$$

Shuffling the definition of 3.2 gives $S_{i,j} = S_{i,j-1} + \delta^J_{i,j}$. Recall that the Smith-Waterman algorithm states $S_{i,0} = 0$, as can be seen in Equation (2.7). Combining these two equations for $j = 1$ gives:

$$S_{i,1} = S_{i,1-1} + \delta_{i,1,0} = 0 + \delta_{i,1,0} = \sum_{1=k}^{1} \delta_{i,k} \tag{3.6}$$

Thus 3.5 is true for the basis step.

Induction step For the induction step, Equation (3.4) is assumed to be true for all values up to l, the hypothesis, given in Equation (3.7). If Equation (3.4) holds for $l + 1$, the equation is proven by induction.

$$S_{i,l} = \sum_{1=k}^{l} \delta_{i,k} \tag{3.7}$$

Using the definition from Equation (3.3):

$$S_{i,l+1} = S_{i,l} + \delta_{i,l+1,0} \tag{3.8}$$

Substitution of the hypothesis in Equation (3.4) for $S_{i,l}$ gives:

$$S_{i,l+1} = \sum_{1=k}^{l} \delta_{i,k,0} + \delta_{i,l+1,0} = \sum_{1=k}^{l+1} \delta_{i,k} \tag{3.9}$$

Hereby Equation (3.4) is proven by induction.

Calculating differential values for the Needleman-Wunsch algorithm

In this subsection it will be demonstrated that a differential score value in the similarity score matrix for the Needleman-Wunsch algorithm can be calculated, when only differential values are stored. This will be done by manipulating formulae.

The Needleman-Wunsch algorithm defines a similarity score matrix S, as stated before in Equation (2.5), and a similarity score function $s_(a_i, b_j)$ as defined earlier in Equation (2.6).

Substitution of Equation (3.3) and Equation (3.2) into Equation (2.5) gives:

$$S_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(a_i, b_i) \\ S_{i-1,j-1} + \delta^I_{i,j} - w(1) \\ S_{i-1,j-1} + \delta^J_{i,j} - w(1) \end{cases} \tag{3.10}$$

This can be rewritten to:

$$S_{i,j} = S_{i-1,j-1} + \max \begin{cases} s(a_i, b_i) \\ \delta^I_{i,j} - w(1) \\ \delta^J_{i,j} - w(1) \end{cases} \tag{3.11}$$

Combining Equation (3.2) and Equation (3.3) gives:

$$S_{i-1,j-1} = S_{i,j-1} - \delta^I_{i,j-1} = S_{i,j} - \delta^I_{i,j-1} - \delta^J_{i,j} \tag{3.12}$$

Substitution of Equation (3.12) into Equation (3.11) gives:

$$S_{i,j} = S_{i,j} - \delta^I_{i,j-1} - \delta^J_{i,j} + \max \begin{cases} s(a_i, b_i) \\ \delta^I_{i,j} - w(1) \\ \delta^J_{i,j} - w(1) \end{cases} \tag{3.13}$$

$S_{i,j}$ can now be taken out of Equation (3.13), this gives:

$$\delta^J_{i,j} = -\delta^I_{i,j-1} + \max \begin{cases} s(a_i, b_i) \\ \delta^I_{i,j} - w(1) \\ \delta^J_{i,j} - w(1) \end{cases} \tag{3.14}$$

The same steps can also be applied to get the horizontal difference $\delta^I_{i,j}$. This gives:

$$\delta^I_{i,j} = -\delta^J_{i-1,j} + \max \begin{cases} s(a_i, b_i) \\ \delta^I_{i,j} - w(1) \\ \delta^J_{i,j} - w(1) \end{cases} \tag{3.15}$$

This shows that for the Needleman-Wunsch algorithm the differential score only depends on the differential values $\delta^I_{i,j}$ and $\delta^I_{i,j}$ and the output of the similarity score function $s(a_i, b_j)$. Thus the similarity score matrix S for the Needleman-Wunsch can be built by using differential values only. Note that the absolute scores for the similarity score matrix can be found by applying Equation (3.4) to these values.

Calculating differential values for the Smith-Waterman algorithm

In this subsection it will be demonstrated that a differential score value in the similarity score matrix for the Smith-Waterman can be calculated, when only differential values are stored. Calculation of the similarity score matrix differentially for the Smith-Waterman algorithm is slightly more complex than doing this for the Needleman-Wunsch algorithm because the comparison in the Smith-Waterman algorithm with a zero, as described earlier in Equation (2.8). If only the differential values are used to calculate the value of a cell in the similarity score matrix, it can not easily be determined whether Equation (3.4) should return a zero.

However, by summing all the previous differential values it is possible to determine the absolute values of the similarity score matrix for the Smith-Waterman algorithm. This method, which differs from the previous discussed method for the Needleman-Wunsch algorithm, will be demonstrated here and its correctness will be shown.

First, $S^*_{i,j}$ is defined as a part of the max case of the Smith-Waterman algorithm, as earlier defined in Equation (2.8):

$$S^*_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(a_i, b_i) \\ S_{i-1,j} - w(1) \\ S_{i,j-1} - w(1) \end{cases} \tag{3.16}$$

Then Equation (2.8) can be rewritten to:

$$S_{i,j} = \max \begin{cases} 0 \\ S^*_{i,j} \end{cases} \tag{3.17}$$

Using the definition of the max operator in Equation (3.17) gives:

$$S_{i,j} = \begin{cases} 0, & \text{if } S^*_{i,j} < 0 \\ S^*_{i,j}, & \text{else} \end{cases} \tag{3.18}$$

The difference between a cell in the similarity score matrix, as defined in Equation (3.16), and the cell left of this cell i.e. $S^*_{i,j}$ and $S_{i,j-1}$, is defined here as $\delta^{J*}_{i,j}$. Using Equation (3.16) gives:

$$\delta^{J*}_{i,j} = S^*_{i,j} - S_{i,j-1} \tag{3.19}$$

Applying the definition from Equation (3.2) for the zero case in Equation (3.18) gives:

$$S_{i,j} = 0 = S_{i,j-1} + \delta^J_{i,j}, \text{ if } S^*_{i,j} < 0 \tag{3.20}$$

Rewriting and substitution of the definition for $\delta^{J*}_{i,j}$, as defined in Equation (3.19), into Equation (3.20) gives:

$$\delta^J_{i,j} = -S_{i,j-1}, \text{ if } S_{i,j-1} + \delta^{J*}_{i,j} < 0 \tag{3.21}$$

The other case in Equation (3.18) gives:

$$S_{i,j} = S^*_{i,j}, \text{ if } S_{i,j-1} + \delta^{J*}_{i,j} \geq 0 \tag{3.22}$$

With Equation (3.2) and Equation (3.19), this can be rewritten to:

$$S_{i,j} = S_{i,j-1} + \delta^{J*}_{i,j} = S^*_{i,j} = S_{i,j-1} + \delta^{J*}_{i,j} \tag{3.23}$$

This simplifies to:

$$\delta^J_{i,j} = \delta^{J*}_{i,j}, \text{ if } S_{i,j-1} + \delta^{J*}_{i,j} \geq 0 \tag{3.24}$$

Combining Equation (3.20) and Equation (3.24) gives:

$$\delta^J_{i,j} = \begin{cases} -S_{i,j-1}, & \text{if } S^*_{i,j} < 0 \\ \delta^{J*}_{i,j}, & \text{else} \end{cases} \tag{3.25}$$

The same steps can be applied to get the horizontal difference $\delta^I_{i,j}$. This gives:

$$\delta^I_{i,j} = \begin{cases} -S_{i-1,j}, & \text{if } S^*_{i,j} < 0 \\ \delta^{I*}_{i,j}, & \text{else} \end{cases} \tag{3.26}$$

with $\delta^{I*}_{i,j}$ the difference between a cell in the similarity score matrix, as defined in Equation (3.16), and the cell above this cell i.e. $S^*_{i,j}$ and $S_{i-1,j}$.

This demonstrates that it is also possible for the Smith-Waterman algorithm to calculate the similarity score matrix by using differential values and keeping track of one absolute value.

3.2.3 Data compression

The amount of bits needed per differential value is $\log_2(2w(1) + s_{max})$. Although it is possible to use $\log_2(2w(1) + s_{max})$ bits, since the differential values are bounded, as will be shown in this subsection. An implementation with signed integers would make arithmetic operations easier. This would require $1 + \log_2(2w(1) + s_{max})$ bits per differential value. Thus, the amount of bits needed for storing signed integers with a row reference and a column reference is $2(1 + \log_2(w(1) + s_{max})$ bits per cell.

Bounded differences

In this section it will be proven that the difference between two cells is bounded. Analysis of the minimal and maximal difference between two cells will be used to do this.

Theorem All differences $\delta_{i,j}$, are bounded by $w(1)$ and $s(a_i, b_j)_{max}$, for all i and for all j:

$$s(a_i, b_j)_{max} + w(1) \geq \delta_{i,j} \geq -w(1) \tag{3.27}$$

Proof Without loss of generality, the following will be shown only for $\delta^J_{i,j}$.

Let i and j be given, then first Equation (3.27) will be proven for the lower bound and then for the upper bound.

First the lower bound will be proven, i.e. $\delta^J_{i,j} \geq -w(1)$, and then the upper bound will be proven i.e. $s(a_i, b_j)_{max} + w(1) \geq \delta_{i,j}$.

Lower bound For a value in the similarity score matrix of the Smith-Waterman algorithm there are four different possibilities as can be seen from Equation (2.8). $S_{i,j-1} - w(1) \leq S_{i,j}$ applies to all the possibilities.

From this the horizontal lower bound for $S_{i,j}$ can be found:

$$S_{i,j} \geq S_{i,j-1} - w(1) \tag{3.28}$$

Combining Equation (3.28) with the definition for $\delta^J_{i,j}$ from Equation (3.2) gives:

$$\delta^J_{i,j} = S_{i,j} - S_{i,j-1} \geq -w(1) \tag{3.29}$$

Upper bound First, the maximal value of the similarity score function is defined as:

$$s_{max} \geq s(a_i, b_j) \tag{3.30}$$

From Equation (2.7) it follows that $s(a_i, b_j) > 0$ for $a_i = b_j$. This gives:

$$s_{max} > 0 \tag{3.31}$$

The diagonal upper bound is:

$$S_{i-1,j-1} + s_{max} \geq S_{i,j} \tag{3.32}$$

This can be concluded from the four different cases for a value in the similarity score matrix of the Smith-Waterman algorithm, as can been seen from Equation (2.8).

Equation (3.30) and Equation (3.31) prove that Equation (3.32) is true for $S_{i,j} = 0$.

Equation (3.30) proves that Equation (3.32) is true for $S_{i,j} = S_{i-1,j-1} + s_{a_i,b_j}$.

By reductio ad absurdum lets assume $S_{i-1,j-1} + s_{max} < S_{i,j}$ to be true. Then a counter example can be made. If $S_{i-1,j-1} = 0$ and $w(1), S_{i-1,j} = 1$, then $S_{i,j} = S_{i-1,j} - w(1) = 0$. The assumption is false, therefore it follows that Equation (3.32) is true for $S_{i,j} = S_{i-1,j} - w(1)$.

By reductio ad absurdum lets assume $S_{i-1,j-1} + s_{max} < S_{i,j}$ to be true. Then a counter example can be made: If $S_{i-1,j-1} = 0$ and $w(1), S_{i,j-1} = 1$, then $S_{i,j} = S_{i-1,j} - w(1) = 0$. The assumption is false, therefore it follows that Equation (3.32) is true for $S_{i,j} = S_{i,j-1} - w(1)$.

Combination of the diagonal upper bound, given in Equation (3.32), and the lower bound, given in Equation (3.28), gives the upper bound in the vertical direction:

$$S_{i-1,j-1} + s_{max} \geq S_{i,j} \geq S_{i-1,j} - w(1) \tag{3.33}$$

Rewriting this equation for another row by using the definition of $\delta^J_{i,j}$, given in Equation (3.2), gives:

$$S_{i,j-1} + s_{max} \geq S_{i,j} - w(1) = \delta^J_{i,j} \tag{3.34}$$

The upper bound, given in Equation (3.34), combined with the lower bound, given in Equation (3.29), results in the bounded differences as stated in Equation (3.27):

$$s_{max} + w(1) \geq \delta^J_{i,j} \geq -w(1) \tag{3.35}$$

Hereby it is proven that all differences $\delta_{i,j}$ are bounded by $w(1)$ and $s(a_i, b_j)$.

$$\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & S_{i-2,j-2} & S_{i-2,j-1} & S_{i-2,j} \\ \cdots & S_{i-1,j-2} & S_{i-1,j-1} & S_{i-1,j} \\ \cdots & S_{i,j-2} & S_{i,j-1} & S_{i,j} \end{array}$$

A 2×2 block of a similarity score matrix $S_{i,j}$

$$\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & S_{i-2,j-2} & S_{i-2,j-1} & S_{i-2,j} \\ \cdots & S_{i-1,j-2} & S_{i-1,j-1} & S_{i-1,j} \\ \cdots & S_{i,j-2} & S_{i,j-1} & S_{i,j} \end{array}$$

$S_{i-1,j-1}$ depends on $S_{i-2,j-2}$, $S_{i-1,j-2}$ and $S_{i-2,j-1}$

$$\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & S_{i-2,j-2} & S_{i-2,j-1} & S_{i-2,j} \\ \cdots & S_{i-1,j-2} & S_{i-1,j-1} & S_{i-1,j} \\ \cdots & S_{i,j-2} & S_{i,j-1} & S_{i,j} \end{array}$$

$S_{i-1,j}$ depends on $S_{i-2,j-2}$, $S_{i-1,j-2}$, $S_{i-2,j-1}$ and $S_{i-2,j}$

$$\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & S_{i-2,j-2} & S_{i-2,j-1} & S_{i-2,j} \\ \cdots & S_{i-1,j-2} & S_{i-1,j-1} & S_{i-1,j} \\ \cdots & S_{i,j-2} & S_{i,j-1} & S_{i,j} \end{array}$$

$S_{i,j-1}$ depends on $S_{i-2,j-2}$, $S_{i-1,j-2}$, $S_{i,j-2}$ and $S_{i-2,j-1}$

$$\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & S_{i-2,j-2} & S_{i-2,j-1} & S_{i-2,j} \\ \cdots & S_{i-1,j-2} & S_{i-1,j-1} & S_{i-1,j} \\ \cdots & S_{i,j-2} & S_{i,j-1} & S_{i,j} \end{array}$$

$S_{i,j1}$ depends on $S_{i-2,j-2}$, $S_{i-1,j-2}$, $S_{i,j-2}$, $S_{i-2,j-1}$ and $S_{i-2,j}$

Figure 3.3: A 2×2 block in the similarity score matrix S can be calculated in one iteration by using the Recursive Variable Expansion method.

Chapter 4

Hardware acceleration

In this chapter some hardware acceleration platforms and interconnect alternatives will be discussed.

4.1 Hardware platforms

In this section some hardware platforms will be discussed and compared.

A processing unit is defined as a piece of hardware, which consists of one or more processing cores, which can perform an algorithm or part of an algorithm by means of operations.

4.1.1 CPU

CPU is an abbreviation for Central Processing Unit. It is the most common processing unit, used in most personal computers and supercomputers. A computing system can contain more than one CPU, this is called multiprocessing. It is also possible to have more than one CPU on a single chip, these chips are called multi-core processors, where a core refers to a single CPU.

The cores in a CPU process software sequentially by performing simple arithmetical, input or output and logical operations. When multiple cores and or CPUs are present they can share their workload.

Conventional CPUs have large instruction sets, which makes them suitable for a lot of different applications. However, these large instructions sets come with a cost, most of the area of the CPU is inactive most of the time. The performance of a CPUs depends on the clock rate and the number of instructions that can be performed in one clock cycle. Most CPUs have high clock rates compared to other processing units, which makes them suitable for algorithms that allow no parallelism.

4.1.2 GPU

GPU is an abbreviation for Graphics Processing Unit. It is a processing unit highly specialised in, and used mainly for, manipulation of computer graphics. Its highly parallel structure makes it suitable for processing large blocks of data in parallel.

Nowadays there is a shift of focus by the manufacturers of GPUs from graphics to more general and parallel computations. These GPUs are also referred to as general-purpose graphical processing units (GPGPU). OpenCL is the most commonly used general-purpose GPU computing language and CUDA is the most commonly used framework.

A significant speedup of sequence alignment algorithms, compared to CPUs, is achievable with GPUs [39][40][41][42][43][44].

4.1.3 FPGA

FPGA is an abbreviation for Field Programmable Gate Array. It is a integrated circuit from which the logic ports can be configured to perform almost any logical operation. Contrary to CPU and GPU, an FPGA is not per definition a processing unit. It can be configured to be any processing unit. This makes it possible to build highly specialised processing units with or on FPGAs.

FPGAs have a lower clock rate than conventional CPUs and GPUs. FPGAs are mainly used for applications where a high throughput of data is required or where algorithms are used that allow for parallel computing, because a high number of multiple specialised processing cores can be implemented.

4.1.4 Overview

In this subsection an overview will be given for the discussed hardware platforms. The aspects, important for sequence alignment algorithms, will be compared.

Sequence alignment algorithms allow for parallel computing, and would therefore benefit from multiple processing cores. This makes the cost and power requirement per processing core important. It is also important to compare the flexibility of the different hardware platforms. Most of these aspects, including scalability, were compared by Vermij and Hasan [37][33]. These results were combined in Figure 4.1.

It demonstrates the different aspects of the different hardware platforms. It can easily be seen, for example, that a bigger number of processing cores can be implemented on an FPGA than on a CPU for the same power requirements and cost.

4.2 Interconnect alternatives

In this section some interconnect methods will be discussed.

4.2.1 RS-232

RS-232 is an old serial communication protocol, defined in 1969. With normal cables the maximal distance allowed is around 15 meters. The maximal bit rate is around 11.5 kbit/s. The protocol is relatively easy to implement, but the bit rate is very low compared to the alternatives discussed below. Previously it was used to connect peripheral hardware to personal computers. Nowadays it is sometimes used in industrial environments and for applications where the bit rate is not important.

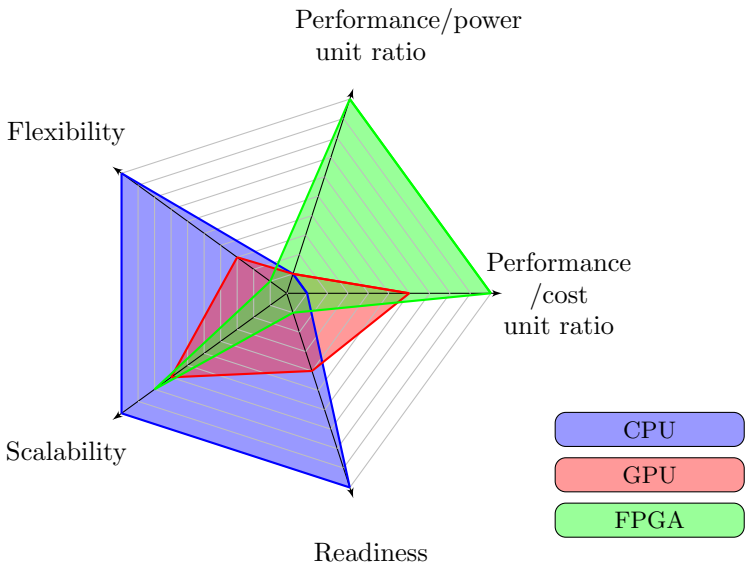

Figure 4.1: Several aspects, important for sequence alignment algorithms, compared for different hardware platforms.

4.2.2 Universal Serial Bus

Universal Serial Bus, usually abbreviated to USB, is a commonly used bus standard. The standard specifies the communication protocol, the wires and the connectors. There are several versions of the specification. Version 2.0 will be discussed here. To communicate with USB, one side acts as a host and the other side acts as a client. The client side is often the peripheral hardware [45]. The connection can be set up in a star topology with a maximal of 5 hubs. The maximal theoretical bit rate is 480 Mbit/s. USB is commonly used for communication between personal computers and peripheral hardware. The success of this specification has also led to the usage on other devices such as mobile telephones.

The versions of USB and their bit rates can be seen in Table 4.1.

Version	Bit rate
USB 1.0 low speed	1.536 Mbit/s
USB 1.0 full speed	12 Mbit/s
USB 2.0	480 Mbit/s
USB 3.0	5 Gbit/s

Table 4.1: Versions of USB and their bit rates.

4.2.3 Ethernet

Ethernet is the name for a group of standards. These standards define wired networking technologies. Ethernet enables computers to connect to other computers in a convenient way. It is convenient because the technology is in use

since the beginning of the Internet, it can be used in a star topology and the cables are flexible and not expensive. There are a lot of different versions of Ethernet. Different versions have different maximal bit rates. 10BASE5, for example, has a bit rate of 10 Mbit/s and 100GbE has a bit rate of 100Gbit/s. For the scope of this thesis, only the very commonly used versions, 100BASE-TX and 1000BASE-T, will be discussed. 100BASE-TX is the standard for connecting personal computers to networks and to the Internet. It features a maximal bit rate of 100 Mbit/s full-duplex i.e. in both directions. Two devices can be connected directly or two or more devices can be connected through hubs. 1000BASE-T is a more powerful version of 100BASE-TX. It features a maximal bit rate of 1 Gbit/s full-duplex. Most conventional computers and servers are equipped with this technology.

Some versions of Ethernet and their bit rates can be seen in Table 4.2.

Version	Bit rate
10BASE-T	10 Mbit/s
100BASE-TX	100 Mbit/s
1000BASE-T	1 Gbit/s
10GBASE-X	10 Gbit/s
40GBASE-X	40 Gbit/s
100GBASE-X	100 Gbit/s

Table 4.2: Versions of Ethernet and their bit rates.

4.2.4 PCI-express

PCI-express is an expansion bus standard to connect boards with specialised hardware to motherboards [46]. PCI is an abbreviation for Peripheral Component Interconnect. The standard specifies lanes e.g. one lane is referred to as x1 and 32 lanes are referred to as x32. One lane is the minimal bus width and thirty two lanes is the maximal bus width. The biggest bus width supported by conventional computers and servers is usually x16. The amount of lanes between a device and a motherboard can change dynamically when needed. The bit rate per lane differs for each different generation of PCI-express. Generation 3 will be discussed here because it is commonly used and it has the highest bit rate per lane. PCI-express Generation 3 features a bit rate of 8.0 Gbit/s per lane full-duplex. The protocol uses packages to communicate. These packages have headers and use a coding scheme of 128 bits data per 130 bits package. This results in an effective data rate of 7.88 Gbit/s per lane. The maximal bit rate for a standard PCI-express x16 Generation 3 card is 126.08 Gbit/s. It specifies a physical layout as an interface between the motherboard and the peripheral device. PCI-express cards are backwards compatible i.e. PCI-express will fit in any slot that has at least as many lanes as the card is wired for. The PCI-express slot also delivers the power to the peripheral device. Some generations of PCI-express and the bit rates for 1 and 16 lanes can be seen in Table 4.3.

Generation	Lanes	Bit rate
PCI-express Generation 1	x1	2.5 Gbit/s
	x16	40.0 Gbit/s
PCI-express Generation 2	x1	5.0 Gbit/s
	x16	80.0 Gbit/s
PCI-express Generation 3	x1	8.0 Gbit/s
	x16	128.0 Gbit/s

Table 4.3: Generations and bus widths of PCI-express including their bit rates.

4.2.5 InfiniBand

InfiniBand is a standard for high speed networking which defines links [47]. The possible link widths for InfiniBand are: one link, 1x; four links, 4x; eight links, 8x; and twelve links, 12x. There are five versions of InfiniBand as can be seen in Table 4.4. The InfiniBand protocol uses packages with headers to deliver data. InfiniBand uses 8B/10B coding to enable clock recovery. The actual data rate without headers and coding is thus slightly less than 80% [48]. Optical and copper cables can be used to interconnect devices. InfiniBand specifies a star topology, but unlike Ethernet it also uses a switched fabric, with multiple paths for one stream of data. The multiple paths guarantee a more continuous throughput when used in large networks. InfiniBand is optimised for low latency. There are switches available with a latency of 100ns [49].

Version	Bit rate
SDR	2.5 Gbit/s
DDR	5.0 Gbit/s
QDR	10.0 Gbit/s
FDR	14.0325 Gbit/s
EDR	25.78125 Gbit/s

Table 4.4: Versions of InfiniBand and their bit rates.

4.2.6 Overview

In this subsection an overview will be given for the discussed interconnect methods. Different requirements demand different features. RS-232 could be a very good interconnect alternative when a simple protocol is needed to transfer and receive data at a slow bit rate. USB is a good alternative when convenience to connect to personal computers is important. Ethernet is easy to use and is available in a few versions for different bit rate requirements. However, it requires a network infrastructure. PCI-express allows for high bit rates, but the boards have to be placed inside a computer or server, on a motherboard, therefore it is not easy scalable. InfiniBand has a high bit rate and it is easily scalable. However, it requires an expensive network infrastructure.

Interconnect technology	Classification	Bit rate
RS-232	Peripheral — Serial	115 kbit/s
USB (2.0)	Peripheral — Serial	480 Mbit/s
Ethernet (1000BASE-T)	Network — Serial	1 Gbit/s
PCIe (x16 Generation 3)	Inter computer — Parallel	126.8 Gbit/s
InfiniBand (FDR)	Network — Serial	56.25 Gbit/s

Table 4.5: An overview of some interconnect alternatives.

Chapter 5

System design

In this chapter the system design will be discussed. First, the targets and requirements will be defined. Then two possible system setups will be discussed and design choices will be underpinned. Finally, an overview of the system design and the choices that were made, will be given.

5.1 Conditions

In this section the targets and requirements for the system will be discussed.

5.1.1 Targets

The main target of this project is to design a system which can accelerate DNA sequence alignment. Nowadays, DNA sequence alignment is being accelerated by the use of heuristic methods or with use of hardware accelerators and supercomputers. The target of this system is to sit between the expensive hardware accelerators and supercomputers, which can be used to find optimal alignments, and the less accurate heuristic methods that are being used by researchers who do not have access to hardware accelerators and supercomputers.

A list of targets is given here, after which, in the next subsection, the requirements will be defined. The target system is a system which:

1. Accelerates DNA sequence alignment, compared to conventional CPUs, used in PCs.
2. Guarantees the optimal local alignment.
3. Connects to a PC, to transfer the sequences, and receive the alignment.
4. Is low-cost, compared to other, high-end solutions available for sequence alignment acceleration.

Before the requirements were defined, first some choices were made about the algorithm and acceleration techniques. This allowed for a better definition of the requirements and a faster implementation.

The algorithm that was chosen is the Smith-Waterman algorithm. The Smith-Waterman algorithm returns the optimal local alignment of two sequences,

as stated before in Chapter 2. This algorithm was chosen instead of the improved algorithms available i.e. Gotoh and Miller-Myers. These improved algorithms, allow for the use of an affine gap penalty function and/or require linear space, but they are both based on the Smith-Waterman algorithm. A system which can accelerate the Smith-Waterman algorithm could easily be extended to support an affine gap penalty function, and/or require linear space instead of quadratic space [32].

It was chosen instead of a heuristic method because the Smith-Waterman algorithm guarantees the optimal local alignment and is thus more accurate than the heuristic methods [33].

It is possible to use the Recursive Variable Expansion method for the Smith-Waterman algorithm, as described earlier in Chapter 3. However, it was decided that this method will not be used for the system, not because the results of implementations have not proven to be very successful [50][51][52][53], but because research has shown that the performance gain per area is not significantly bigger compared to conventional methods [37][1].

Summarised, the target system is a low-cost system that accelerates the Smith-Waterman algorithm with a linear gap penalty function, which returns the optimal local alignment of two sequences, sent from a PC, back to a PC.

5.1.2 System requirements

With the targets, as described earlier, the requirements can be defined. The MoSCoW method was used to prioritise the requirements. The MoSCoW categories are defined as follows:

- **M**ust; these requirements must be satisfied for the system design to be successful.
- **S**hould; these requirements should be included in the system.
- **C**ould; these requirements are desirable, but not necessary for the system design to be successful.
- **W**ould; it would be nice to realise these requirements in the future.

To realise a system that meets the targets that were set, the following system requirements were defined.

The system must at least:

- Be able to calculate the similarity score matrix of the Smith-Waterman algorithm, for DNA sequences, using a linear gap penalty function, as defined in Equation (2.8) and Equation (2.1).
- Connect easily to a conventional PC.
- Be able to return the value of the maximal score in the similarity score matrix to a PC.
- Accelerate the Smith-Waterman algorithm, compared to the conventional CPUs, currently used in PCs.
- Be low-cost, compared to the other available accelerators.

The system should at least:

- Be able to return the value and the index of the maximal score in the similarity score matrix to a PC.
- Be easily scalable e.g. by clustering.

The system could maybe:

- Be able to calculate the matrices of the Gotoh algorithm, for DNA sequences, using an affine gap penalty function, as defined in Equation (2.15), Equation (2.16), Equation (2.17) and Equation (2.2).
- Be able to do the trace-back procedure, to find the optimal local alignment.

For future research, it would be nice if the system:

- Is able to find the optimal local alignment, for DNA sequences, using an affine gap penalty, in linear space i.e. the Miller-Myers algorithm.

5.2 System setup

In this section, two possible system setups will be discussed.

5.2.1 Single accelerator

In this setup, a host will connect to an accelerator through an interface, see Figure 5.1. The user will send DNA sequences from the host to the accelerator, on which the similarity score matrix of the Smith-Waterman algorithm will be calculated.

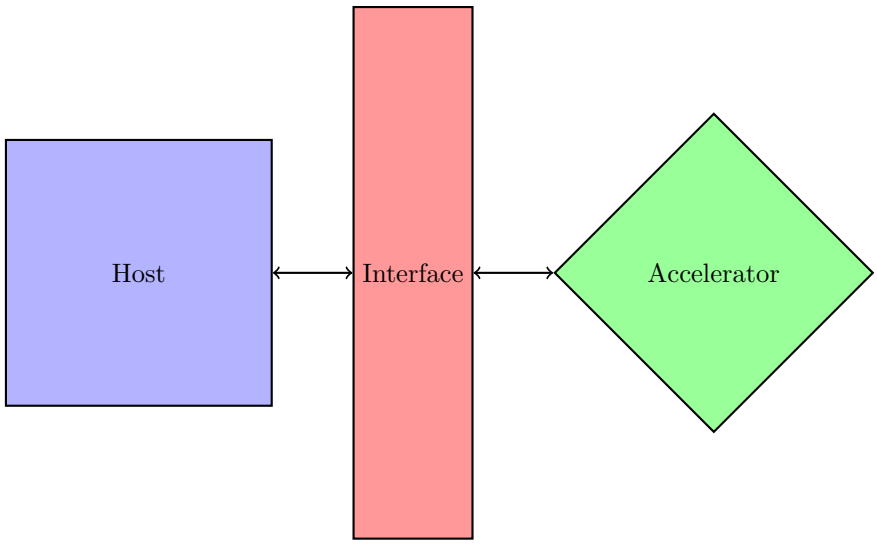

Figure 5.1: Single accelerator setup.

5.2.2 Clustering

In this setup, a host will connect to multiple accelerators through a shared interface, see Figure 5.2. A group of independent computing units, where all units perform a small part of the same task, are called clusters. The shared interface could be realised with switches.

With more accelerators, more processing cores could be implemented, and therefore the performance of the system would increase with this setup, compared to the single accelerator setup.

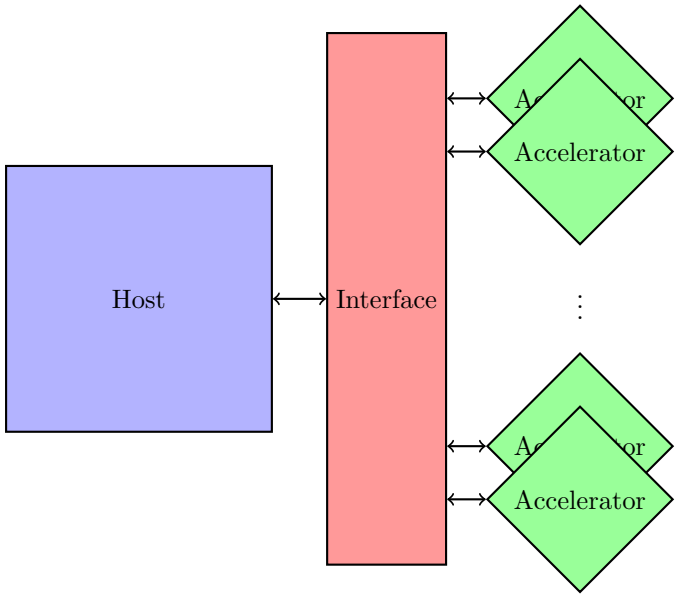

Figure 5.2: Cluster setup, multiple accelerators connected to a single host.

For this system, first the single accelerator setup was chosen, since the clustering setup would be based on the single accelerator setup. The clustering setup could then be realised by applying some changes to the single accelerator setup.

5.3 Design

In this section, the system design choices will be discussed.

5.3.1 Host

The host in the system, connects to the accelerator, via an interface, as can be seen in Figure 5.1. The host should at least be able to sent and receive information to the accelerator. One of the targets is to use a PC as host. By choosing a PC as host, the entry level for users gets lowered, and future expansions are relatively easy to implement. Another benefit from choosing a PC, is the number of interconnect options available with conventional PCs.

The host will have two main tasks, sending the DNA sequences to the accelerator, and receiving the results from the accelerator. The results can either be the similarity score matrix or the value and index of the maximal score.

Host program for the Smith-Waterman algorithm, with the similarity score matrix returned by the accelerator.

Require: A, B
Ensure: connection with accelerator
1: send(A, B)
2: receive(S)
3: alignment $\leftarrow$ trace-back(S)

Host program for the Smith-Waterman algorithm, with the value and index of the maximal score returned by the accelerator.

Require: A, B
Ensure: connection with accelerator
1: send(A, B)
2: receive($score_{max}, index$)
3: alignment $\leftarrow$ divide-and-conquer($score_{max}, index$)

Unfortunately, because the size of the similarity score matrix for long DNA sequences can grow beyond available memory and storage on conventional PCs, it is not possible for the accelerator to return the complete similarity score matrix. This is the reason that for this system, the accelerator will return the value and the index of the maximal score, as can be seen from the requirements.

It could have been possible to implement the trace-back procedure on the accelerator [54][55], but this would not be a good choice, since calculating the similarity score matrix requires the most computation time in the Smith-Waterman algorithm, and the trace-back procedure can easily be implemented on the host.

5.3.2 Accelerator

The accelerator receives the sequences from the host. With these sequences, the similarity score matrix, as defined in Equation (2.8), can be built. The accelerator in the system then returns the value and index of the maximal score to the host, via an interface, as can be seen in Figure 5.1. With these results, the host can use the divide and conquer technique to find the alignment.

A hardware platform was chosen for the accelerator. As explained in Subsection 3.1.1, the Smith-Waterman algorithm allows for parallel computation of the similarity score matrix. The system would, for that reason, benefit from a processing unit with multiple processing cores. Calculation of the similarity score matrix can be done by a linear systolic array of processing cores. The number of processing cores defines the acceleration that can be achieved. Ideally, the number of processing cores corresponds to the number of cells of the similarity score matrix that can be processed at the same iteration. Please note that utilisation of the processing cores takes some time, but this effect becomes neglectable for long DNA sequences.

An FPGA was chosen as the hardware platform for this system. This hardware platform is not very flexible, as can be seen from Figure 4.1, but the performance/power unit ratio and performance/cost unit ratio are better for this platform than the ratios of the other hardware platforms that were considered for this system [56][57][58]. An FPGA can be configured for a specific task, which means that all the area of the FPGA can be effectively used, most of the time [59]. Therefore, the number of processing cores, which corresponds almost directly to the speedup that can be achieved, that can be implemented per cost unit is the largest for the FPGA platform. This, together with the scalability options, were the two main reasons that an FPGA was chosen for this system.

5.3.3 Interface

The interface in the system connects the host to the accelerator, as can be seen in Figure 5.1. The two main tasks for the interface are to send the DNA sequences from the host to the accelerator, and return the result to the host i.e. the value and index of the maximal score.

As stated before, the size of the similarity score matrix can grow beyond available memory and storage, and therefore it is not possible to store the similarity score matrix on the accelerator or on the host. Instead, the value and index of the maximal score are returned.

The bandwidth intensive task for the interface is to send the sequences to the accelerator [60]. Each nucleobase in DNA can be represented by 2 bits, as shown before in Table 3.1. Two options were considered here. First, it would be possible to send the complete DNA sequences to the accelerator where they would be stored in memory. The other option, which is probably better because a lot of memory could be required for long DNA sequences, is to send parts of the sequences on demand to the accelerator. The number of processing cores determines how much sequences have to be send within a certain time frame, and therefore determine the bandwidth requirement, when the sequences are streamed to the accelerator [60].

However, the choice for a interconnect method, was not only based on the bandwidth requirement. Because a PC was chosen as a host, the accessibility of the system depends on the ease of connecting it to a PC.

For this system, serial and USB were considered, but the maximal bit rates of these interconnect alternatives would become the bottleneck of the system. PCI-express and Ethernet support higher bit rates. From these two, Ethernet has the advantage of easier connectivity and cluster options. InfiniBand could also be used however this method is expensive and not common for conventional PCs.

5.4 Overview

In this section, an overview will be given of the choices that were made for the system design.

A single accelerator setup was chosen because clustering is beyond the scope of this thesis. As host, a PC was chosen because nowadays they are commonly used. For the accelerator, an FPGA was chosen, because they can be configured to be specific processing units, with lots of processing cores. As interconnect

alternative, Ethernet was chosen, because it connects easily to conventional PCs and it supports relatively high bit rates.

An overview of the system design choices is shown in Table 5.1. An overview of the system setup is given in Figure 5.3.

Part of design	Choice
System setup	Single accelerator
Host	PC
Accelerator	FPGA
Interface	Ethernet

Table 5.1: An overview of the system design choices.

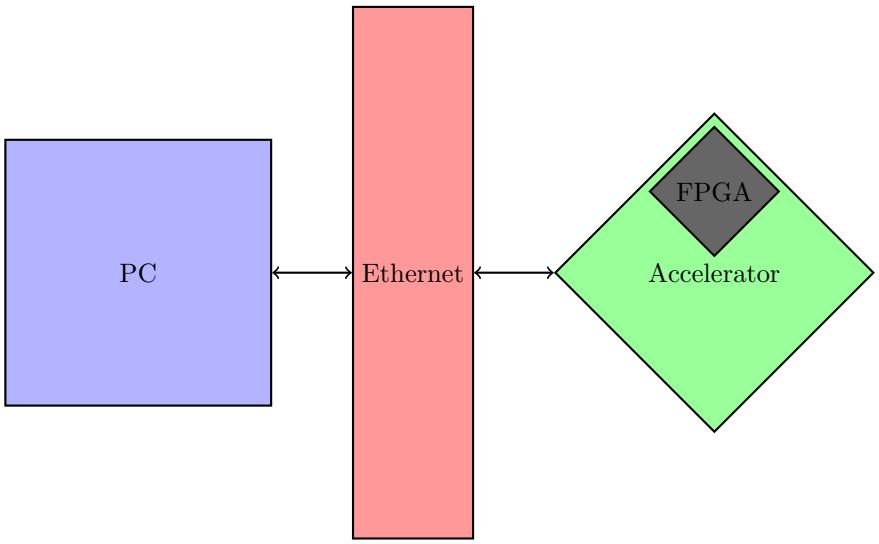

Figure 5.3: An overview of the system setup.

Chapter 6

Implementation

In this chapter a possible implementation of the design, described in the previous chapter, will be discussed. First an FPGA selection will be made. Then, the computation and communication implementations of the accelerator are discussed. Finally an overview is given, in which the performance is analysed and compared with other studies and platforms.

6.1 FPGA selection

In this section, the FPGA selection and choice will be discussed.

For the selection of the FPGA, the two most important properties were cost and power efficiency. Since the target of this project is to design a low-cost solution, a consideration between performance and cost had to be made. The goal was to find the FPGA with the best performance/cost unit ratio.

The performance of an FPGA can best be expressed by the limiting factor. The limiting factor is the number of look up tables (LUTs). LUTs are a basic hardware parts for FPGAs.

First, two manufacturers were considered; Xilinx and Altera. They both offer different families of FPGAs, for all kinds of applications. A state-of-the-art Xilinx Atrix-7 FPGA was chosen, since this family of FPGAs is optimised for low-cost and low-power. The XC7A200T was chosen, which features $33,650$ slices, and each slice contains four LUTs, which gives a total of $134,600$ LUTs [61].

The two other 7 series FPGAs from Xilinx, the Kintex-7 and the Virtex-7, offer better performance, but they were not chosen since they are not considered to be low-cost.

6.2 Computation

In this section the implementation of the computational part of the accelerator will be discussed. First, the implementation of the processing core will be demonstrated, after which an implementation of a linear systolic array of processing cores will be given.

6.2.1 Processing core

A processing core in this system will calculate the value of a cell of the similarity score matrix. The value of a cell is determined by Equation (2.8). In the implementation the differential method, introduced in Section 3.2.2, will be used to reduce the number of logic cells needed per processing core.

The processing core implementation can be seen in Figure 6.1 and Figure 6.2.

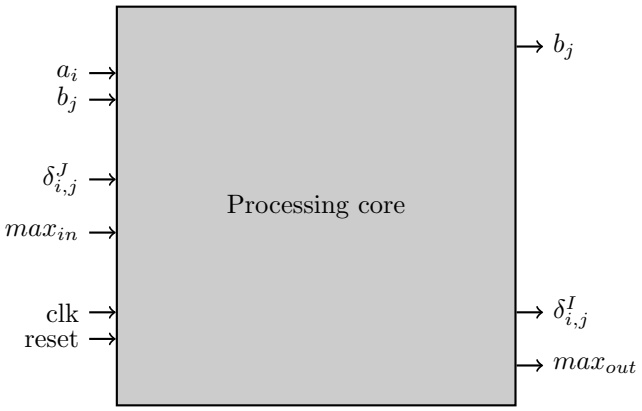

Figure 6.1: A processing core.

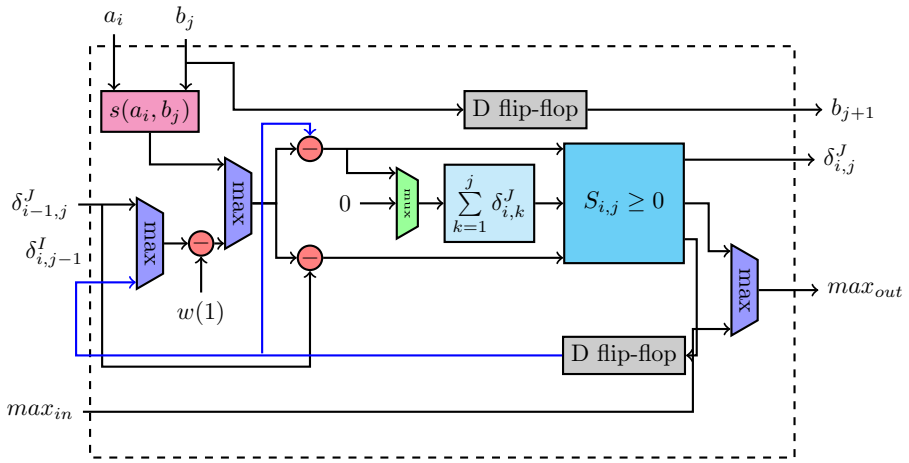

Figure 6.2: The processing core implementation by Geers, Çağlayan and Heij [1]. This core processes the differential score values from the top to the bottom of a column.

The implementation in VHDL of this design was done by Geers, Çağlayan and Heij [1]. With, $4 \geq s_{max} \geq 2$ and $8 \geq w(1) \geq 1$, one 32 bit sum operator and three 5 bit subtract operators were used for this implementation. A straight forward implementation of Smith-Waterman would need three 32 bit add operators. The use of the differential model reduces the number of logic

cells needed per processing core. For this design, one processing core uses 226 LUTs and 46 slice registers [1].

6.2.2 Linear systolic array

With the processing cores a linear systolic array will be built, as can be seen in Figure (6.3). The implementation in VHDL was done by Geers, Çağlayan and Heij [1].

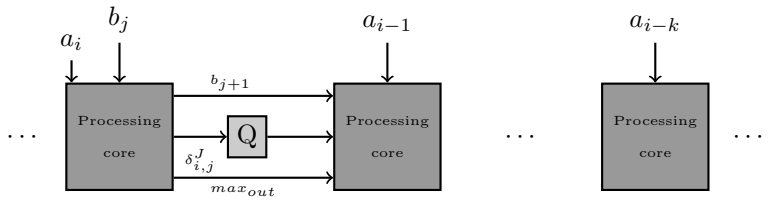

Figure 6.3: Linear systolic array implementation.

The linear systolic array calculates the similarity score matrix from the top to the bottom, and from the left to the right [34][62][52]. Some of the differential values calculated are fed back into the same processing core. This means that the calculated difference with the cell to the left will be stored for one clock pulse, and will then be used to calculate the value of the cell below. The calculated difference with the cell above is sent to the next processing core in the array, which will calculate the value of the cell to its right.

6.3 Communication

In this section, the selection of an Ethernet version and core, which will form the communication part of the accelerator, will be discussed.

6.3.1 Ethernet selection

In Chapter 4, different versions of Ethernet were discussed. In this subsection, a version will be selected, according to the requirements of the system.

Since bandwidth can easily become the limiting factor for long DNA sequences, a consideration had to be made between cost, performance and accessibility.

The fastest version of Ethernet currently available is 1000GBASE-X which supports a bit rate up to 100 Gbit/s. However, the 1000BASE-T version was chosen, which supports a bit rate up to 1 Gbit/s, because this version is supported by most conventional PCs nowadays, and it has the best performance/cost ratio.

A bit rate of 1 Gbit/s would be sufficient for alignment of two long DNA sequences, however it could be insufficient for the alignment of a short and a long sequence. This is because the amount of bits that need to be transferred in a certain time frame is relatively low for alignment of two long sequences compared to the amount of bits transferred in a certain time frame when a short sequence is aligned to a long sequence.

6.3.2 Ethernet core

Because developing an Ethernet core is beyond the scope of this thesis, an open core was selected because of the low-cost requirement. A stable 10/100/1000 Mbit/s tri-mode Ethernet core was selected [63]. The core uses 1526 LUTs.

6.4 Overview

In this section, an overview will be given of the suggested implementation given in this chapter. The performance will be analysed and compared to other solutions.

6.4.1 Theoretical performance

The performance of a linear systolic array implementation can be expressed as the number of active processing cores times the clock frequency. For long sequences, the start up and finish time become a very small part of the total sequence. The difference between the actual performance and the maximal performance is neglectable [37].

The amount of processing cores possible on an FPGA depends on the available amount of LUTs and slice registers. An Artix-7-200T has 134600 LUTs and 269200 slice registers [61]. The number of processing cores is limited by the amount of LUTs. If 80% of the LUTs can be used, which is quite ambitious, 469 processing cores could fit on an Artix-7. This can be seen in Table 6.1.

Part of the design	Cores	LUTs	Total LUTs	LUTs usage
Ethernet core	1	1,526	1,526	1.1%
Processing cores	469	226	105,994	78.7%
Total design	-	-	107,520	79.8811%

Table 6.1: The number of LUTs needed for the design.

The Artix-7 FPGA can run on a 200 MHz clock [64]. This could lead to a total performance of 93.8 GCU/s.

6.4.2 Overview

The system proposed in this thesis delivers at least 2 times more performance per euro compared to the other systems that were compared. This does not include the host and the interface.

An overview of the performance of several implemented platforms will be given here. The price per system are estimates. The proposed system will cost about 400 euro. The price of an Artix-7 XC7A200T-2C is 200 euros [65]. The accelerator, with the FPGA, will cost around 240 euros, leaving 160 euros for development and research.

A mid-range CPU was used for The Szalkowski implementation and a high-end CPU was used for the Hasan implementation [56]. The price is deducted of a Opteron 6284 system. The Convey HC-1 implementation [37] and the Rivyera by Sciengines [66] were also compared. The prices of these systems were found after contact with the companies.

Implementation	Cores	Frequency (Mhz)	Performance (GCU/s)	Price (€/system)	Price (GCU/s/€)
Proposed	469	200	93.8	400	0.234
Szalkowski [56]	4	2400	15.7	400	0.039
Hasan [56]	16	2300	64	2000	0.032
Convey [37]	691	150	691	22000	0.031
Sciengines [66]	-	-	6046	70000	0.086

Table 6.2: Comparison of different computing solution on GCU/s and price.

Chapter 7

Conclusions and recommendations

In this chapter, conclusions and recommendations for future research will be given.

7.1 Conclusions

In this section, the conclusions of the research, design and implementation will be given.

Unfortunately, the suggested system setup could not be implemented in the time that was available for this project. For testing purposes an implementation of a USB driver in VHDL was tested on an Altera DE2 FPGA Development Board. With this driver, data could be shared between a host PC and the FPGA board. However, from this implementation it could be concluded that the bit rate of USB 2.0 would be the limiting factor for the system. An Ethernet driver was not implemented, because this was beyond the scope of this thesis, and there was insufficient time.

A linear systolic array of processing cores, which returns the value of the maximal score of the similarity score matrix, was implemented in VHDL by Geers, Çağlayan and Heij [1]. Tests on an Altera DE2 FPGA Development Board demonstrated that the implementation of the processing core and the linear systolic array of processing cores worked accordingly, and could return the value of the maximal score of the similarity score matrix.

An estimation of the performance of a implementation of the system on an Artix-7 XC7A200T-2C FPGA shows that a performance of 93 GCU/s is achievable with a low-cost FPGA. The power requirement of this implementation was not analysed, nor the power requirements of the other implementation that are available, but it can easily be stated that the power demanded for this system is less than the power demanded for super computing platforms. The performance/power unit ratio for this implementation would be better than for other implementations, due to the use of a low-power FPGA and an area efficient implementation of the processing cores. As a result, this system is both low-cost in use and in purchase compared to the other systems that are available and the

performance/cost unit of this system is bigger than the other available system as can be seen from Table 6.2.

7.1.1 Research

From the research, which is the main part of this thesis, the following conclusions can be made:

- Genetic research leads to understanding how life and living organisms work, and this research field is limited by its computational power.
- The cost at which DNA can be sequenced is dropping at a higher rate than the cost at which the DNA sequences can be analysed with computation power.
- Heuristics methods are used, because optimal methods require too much computation time on conventional or available systems.
- Local sequence alignment algorithms, with an affine gap penalty function, are mainly used for DNA sequence alignment.
- There are a lot of improved variations to the Smith-Waterman algorithm however there is a lack of optimised hardware implementations for these improved algorithms.
- The similarity score matrix of the Smith-Waterman algorithm can be calculated in parallel.
- The similarity score matrix of the Smith-Waterman algorithm can be stored differentially.

7.1.2 Design

From the design the following conclusions can be made:

- There is a lack of low-cost systems which can find the optimal local alignment of DNA sequences.
- Using a host PC allows for flexible designs.
- FPGAs can be used to accelerate the Smith-Waterman algorithm, because a lot of processing cores can be implemented on a single FPGA.
- Low-cost and low-power solutions for sequence alignment algorithms are possible.

7.1.3 Implementation

From the implementation, which was mostly done by Geers, Çağlayan and Heij [1], the following conclusions can be made:

- The area needed per processing core could be reduced by calculating the values of the similarity score matrix differentially.

- The available bandwidth of the system can become the limiting factor for mapping of the sequences found by Next-Generation sequencers.
- The bandwidth requirement can be reduced by using buffers on the FPGA to keep track of the value of the maximal score.

7.2 Recommendations

The following recommendations for future research can be made:

- Implementation of the differential method, introduced in this thesis, for other hardware platforms.
- Implementation of the Gotoh algorithm with an affine gap penalty function, to make the system design, suggested in this thesis, more suitable for biotechnology applications.
- Implementation using state-of-the-art FPGA platforms.
- Implementation of the clustering system setup.
- Adding memory to the accelerator design for better performance with long DNA sequences.
- A hybrid platform of an FPGA and CPU to perform the Miller-Myers algorithm.
- On-the-fly reconfiguration of the FPGA, using the Recursive Variable Expansion method only to speed up the utilisation of the processing cores.
- Sequence alignment algorithms usages comparison study.
- Analysis of computational problems in the bio-technology industry.
- Comparison study of power requirements for hardware accelerated systems used for sequence alignment algorithms.

Bibliography

[1] M. Geers, F. H. Çağlayan, and R.W. Heij. *Low-Cost Smith-Waterman Acceleration*. June 2013.

[2] G.J. Tortora and Derrickson B.H. *Principles of Anatomy and Physiology*. 13th International Student edition. John Wiley and Sons Ltd, 2011.

[3] E.R. Mardis. "The impact of next-generation sequencing technology on genetics". In: *Trends in Genetics* 24.3 (2008), pp. 133–141.

[4] T. Tucker, M. Marra, and J.M. Friedman. "Massively Parallel Sequencing: The Next Big Thing in Genetic Medicine". In: *The American Journal of Human Genetics* 85.2 (2009), pp. 142–154.

[5] E.S. Lander et al. "Initial sequencing and analysis of the human genome". In: *Nature* 409.6822 (Feb. 2001), pp. 860–921.

[6] intrepidbio. *Next generation sequencing the data storage dilemma*. June 2011. URL: `http://www.intrepidbio.com/next-generation-sequencing-the-data-storage-dilemma/`.

[7] G.E. Moore. "Cramming more components onto integrated circuits". In: *Electronics* 38.8 (1965).

[8] L.D. Stein. "The case for cloud computing in genome informatics". In: *Genome biology* 11.5 (2010), p. 207.

[9] K.A. Wetterstrand. *DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP)*. June 2013. URL: `http://genome.gov/sequencingcosts/`.

[10] J. Kleinberg and E. Tardos. "Sequence Alignment". In: *Algorithm Design*. Addison-Wesley, 2006. Chap. 6, pp. 278–290.

[11] A. Stivala et al. "Lock-free Parallel Dynamic Programming". In: *Journal of Parallel and Distributed Computing* 70.8 (2010), pp. 839–848.

[12] S.F. Altschul et al. "Basic Local Alignment Search Tool". In: *Journal of Molecular Biology* 215.3 (Oct. 1990), pp. 403–410.

[13] S.F. Altschul et al. "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs". In: vol. 25. 17. 1997.

[14] B. Harris et al. "A Banded Smith-Waterman FPGA Accelerator for Mercury BLASTP". In: *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. 2007, pp. 765–769.

[15] S. Ishikawa, A. Tanaka, and T. Miyazaki. "Hardware Accelerator for BLAST". In: *Embedded Multicore Socs (MCSoC), 2012 IEEE 6th International Symposium on*. 2012, pp. 16–22.

[16] A. Jacob et al. “Mercury BLASTP: Accelerating Protein Sequence Alignment”. In: *ACM Transactions on Reconfigurable Technology and Systems* 1.2 (June 2008).

[17] D.J. Lipman and W.R. Pearson. “Rapid and Sensitive Protein Similarity Searches”. In: *Science* 227.4693 (Mar. 1985), pp. 1435–1441.

[18] W.R. Pearson and D.J. Lipman. “Improved tools for biological sequence comparison”. In: *Proceedings of the National Academy of Sciences of the United States of America* 85.8 (Apr. 1988), pp. 2444–2448.

[19] W.R. Pearson. “Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith-Waterman and FASTA Algorithms”. In: *Genomix* 11 (Dec. 1991), pp. 635–650.

[20] O. Gotoh. “An Improved Algorithm for Matching Biological Sequences”. In: *Journal of Molecular Biology* 162.3 (Dec. 1982), pp. 705–708.

[21] S.F. Altschul and B.W. Erickson. “Optimal Sequence Alignment Using Affine Gap Costs”. In: *Bulletin of Mathematical Biology* 48.5-6 (1986), pp. 603–616.

[22] S.B. Needleman and C.D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453.

[23] D.S. Hirschberg. “A Linear Space Algorithm for Computing Maximal Common Subsequences”. In: *Communications of the ACM* 18.6 (June 1975), pp. 341–343.

[24] T.F. Smith and M.S. Waterman. “Identification of Common Molecular Subsequences”. In: *Journal of Molecular Biology* 147.1 (1981), pp. 195–197.

[25] M.S. Waterman. “Efficient Sequence Alignment Algorithms”. In: *Journal of Theoretical Biology* 108.3 (June 1984), pp. 333–337.

[26] E.W. Myers and W. Miller. “Optimal alignments in linear space”. In: *Computer Applications in the Biosciences* 4 (1988), pp. 11–17.

[27] X. Guan and E.C. Uberbacher. *A multiple divide-and-conquer (MDC) algorithm for optimal alignments in linear space*. Tech. rep. United States, Jan. 1994, p. 13.

[28] K.-M. Chao and W. Miller. “Linear-Space Algorithms that Build Local Alignments from Fragments”. In: *Algorithmica* 13 (1995), pp. 106–134.

[29] M.S. Waterman and M. Eggert. “A New Algorithm for Best Subsequence Alignments with Application to tRNA-rRNA Comparisons”. In: *Journal of Molecular Biology* 197.4 (1987), pp. 723–728.

[30] X. Huang and W. Miller. “A time-efficient linear-space local similarity algorithm”. In: *Advances in Applied Mathematics* 12.3 (Sept. 1991), pp. 337–357.

[31] F. Zhang, X. Qiao, and Z. Liu. “A Parallel Smith-Waterman Algorithm Based on Divide and Conquer”. In: *Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*. ICA3PP ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 162–.

[32] F. Zhang, X. Qiao, and Z. Liu. “Parallel Divide and Conquer Bio-sequence Comparison Based on Smith-Waterman Algorithm”. In: *Science in China Series F: Information Sciences* 47.2 (2004), pp. 221–231.

[33] L. Hasan, Z. Al-Ars, and S. Vassiliadis. “Hardware Acceleration of Sequence Alignment Algorithms-An Overview”. In: *Design Technology of Integrated Systems in Nanoscale Era, 2007. DTIS. International Conference on*. 2007, pp. 92–97.

[34] M. Gok and C. Yilmaz. “Efficient Cell Designs for Systolic Smith-Waterman Implementations”. In: *Field Programmable Logic and Applications, 2006. FPL ’06. International Conference on*. 2006, pp. 1–4.

[35] L. Hasan, Y.M. Khawaja, and A. Bais. “A Systolic Array Architecture for the Smith-Waterman Algorithm with High Performance Cell Design”. In: *IADIS European Conf. Data Mining*. 2008, pp. 35–44.

[36] Z. Nawaz et al. “Recursive Variable Expansion: A Loop Transformation for Reconfigurable Systems”. In: *Field-Programmable Technology, 2007. ICFPT 2007. International Conference on*. Kokurakita, Japan, 2007, pp. 301–304.

[37] Erik Vermij. “Genetic Sequence alignment on a supercomputing platform”. MA thesis. Delft University of Technology, 2011.

[38] A.E. de la Serna. “Differential Scoring for Systolic Sequence Alignment”. In: *BIBE*. Dec. 2007, pp. 1204–1208.

[39] L. Hasan, M.A. Kentie, and Z. Al-Ars. “GPU-Accelerated Protein Sequence Alignment”. In: *Proc. 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. Boston, USA, Aug. 2011, pp. 2442–2446.

[40] L. Hasan, M.A. Kentie, and Z. Al-Ars. “DOPA: GPU-based Protein Alignment Using Database and Memory Access Optimizations”. In: *BMC Research Notes* 4.261 (July 2011), pp. 1–11.

[41] A. Khajeh-Saeed, S Poole, and J.B. Perot. “Acceleration of the Smith-Waterman algorithm using single and multiple graphics processors”. In: *Journal of Computational Physics* 229 (2010), pp. 4247–4258.

[42] S. Manavski and G. Valle. “CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment”. In: *BMC Bioinformatics* 9.Suppl 2 (2008), S10.

[43] E.F. de O.Sandes and A.C.M.A. de Melo. “Smith-Waterman Alignment of Huge Sequences with GPU in Linear Space”. In: *Parallel and Distributed Processing Symposium, International* 0.1530-2075 (2011), pp. 1199–1211.

[44] E. F. de O.Sandes and A.C.M.A. de Melo. “Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences Using GPU”. In: *Parallel and Distributed Systems, IEEE Transactions on* 24.5 (2013), pp. 1009–1021.

[45] Hewlett-Packard Company et al. *Universal Serial Bus 3.0 Specification*. Tech. rep. May 2011.

[46] PCISIG. *PCI Express Base Specification Revision 3.0*. Tech. rep. Nov. 2010.

[47] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 2 Release 1.3*. Nov. 2012.

[48] InfiniBand Trade Association. *InfiniBand Architecture Specification Volume 1 Release 1.2.1*. Nov. 2007.

[49] Mellanox. *Product overview*. June 2013. URL: `http://www.mellanox.com/page/switch_systems_overview`.

[50] L. Hasan et al. "Hardware Implementation of the Smith-Waterman Algorithm Using Recursive Variable Expansion". In: *Proc. 3rd IEEE International Design and Test Workshop*. Monastir, Tunisia, Dec. 2008, pp. 135–140.

[51] L. Hasan, Z. Al-Ars, and Z. Nawaz. "A Novel Approach for Accelerating the Smith-Waterman Algorithm using Recursive Variable Expansion". In: *Proc. 19th Annual Workshop on Circuits, Systems and Signal Processing*. Veldhoven, The Netherlands, Nov. 2008, pp. 40–45.

[52] L. Hasan and Z. Al-Ars. "Performance Comparison between Linear RVE and Linear Systolic Array Implementations of the Smith-Waterman Algorithm". In: *Proc. 20th Annual Workshop on Circuits, Systems and Signal Processing*. Veldhoven, The Netherlands, Nov. 2009, pp. 451–456.

[53] Z. Nawaz, H.E. Sumbul, and K.L.M. Bertels. "Fast Smith-Waterman hardware implementation". In: *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. Atlanta, USA, 2010, pp. 1–4.

[54] S. Lloyd and Q.O. Snell. "Hardware Accelerated Sequence Alignment with Traceback". In: *International Journal of Reconfigurable Computing* 2009 (Jan. 2009), 9:1–9:10.

[55] Z. Nawaz et al. "A parallel FPGA design of the Smith-Waterman traceback". In: *Field-Programmable Technology (FPT), 2010 International Conference on*. Beijing, China, 2010, pp. 454–459.

[56] L. Hasan and Z. Al-Ars. "An Overview of Hardware-Based Acceleration of Biological Sequence Alignment". In: *Computational Biology and Applied Bioinformatics* (Dec. 2011), pp. 187–202.

[57] I. Li, W. Shum, and K. Truong. "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)". In: *BMC Bioinformatics* 8.1 (2007), p. 185.

[58] T. Rognes and E. Seeberg. "Six-fold speed-up of SmithWaterman sequence database searches using parallel processing on common microprocessors". In: *Bioinformatics* 16.8 (2000), pp. 699–706.

[59] L. Hasan and Z. Al-Ars. "Performance Improvement of the Smith-Waterman Algorithm". In: *Proc. 18th Annual Workshop on Circuits, Systems and Signal Processing*. Veldhoven, The Netherlands, Nov. 2007, pp. 211–214.

[60] L. Hasan et al. "Performance and Bandwidth Optimization for Biological Sequence Alignment". In: *Proc. 5th IEEE International Design and Test Workshop*. Abu Dhabi, UAE, Dec. 2010, pp. 155–160.

[61] Xilinx. *7 Series FPGAs Overview*. Tech. rep. Nov. 2012.

[62] L. Hasan, Y.M. Khawaja, and A. Bais. "A Systolic Array Architecture for the Smith-Waterman Algorithm with High Performance Cell Design". In: *IADIS European Conf. Data Mining*. 2008, pp. 35–44.

[63] J. Gao. *10/100/1000 Mbps tri-mode ethernet MAC*. Nov. 2005. URL: `http://opencores.org/project,ethernet_tri_mode`.

[64] Xilinx. *Artix 7 data sheet*. Tech. rep. Apr. 2013.

[65] digikey.nl. *Cataloge FPGAs*. June 2013. URL: `http://www.digikey.nl/scripts/dksearch/dksus.dll?pv457=417&FV=fff40027\%2Cfff80166&k=artix+7&mnonly=0&newproducts=0&ColumnSort=0&page=1&stock=1&pbfree=1&quantity=0&ptm=0&fid=0&pageSize=25`.

[66] SciEngines GmbH. *Accelerated Smith-Waterman on RIVYERA Hardware*. Tech. rep. Kiel, Germany, 2012.