Evaluation of an FPGA acceleration method for DNA sequence pattern matching

Interim Report

Vytautas Mizgiris

4th Year Project Interim Report
Computer Science
School of Informatics
University of Edinburgh

2018

1 Overview

My project focuses on implementing and evaluating a method of DNA matching on an FPGA-processor driven device, supplied by the University. FPGA synthesizing is a very popular way to test a custom architecture for specific tasks for a later permanent transfer to highly scalable and fast application-specific circuits (ASICs). A number of methods are described in scientific literature, and my aim is to implement a suitably scalable, parallel method of matching two DNA sequences in hardware, and evaluate the efficiency compared to traditional general purpose CPU running an equivalent algorithm for the same task.

The interim report is going to brief through the following topics of the project progress:

- **Background and research.** The ground for any evaluation, implementation and, possibly, improvements. I have went through a number of articles and chose suitable candidate papers to implement the logic described, and present results.
- **Progress on learning and implementation**
- **Current plan and objectives, time evaluation.** The next steps that need to be taken to finish the project on time as well as objectives at the time of writing this interim report.
- **Project report outline.** Detailed section outline of the final project.

2 Background and research

I have defined an introductory paragraph in my main report, describing, in short, the applications of DNA matching. It is meant to make the reader be aware of the importance of the topic being explored:

> The DNA, or deoxyribonucleic acid, is an important part of any living organism that essentially defines the way it grows and functions. It is code that is supplied to create genes in cells, and provide vital information on how the cell should reproduce. Due to DNA's uniqueness, it is one of the most important defining features of every human being. It is very commonly used in the fields of computational biology, or bioinformatics, where DNA analysis is done by examining the cells and, most often, finding patterns and / or aligning the extracted code to another set of DNA. A few fields that such analysis helps significantly are: proving the evolution of organisms and determining the origin and history of species, paternity tests and genealogy tree building, identifying criminals by observing the DNA from fingerprints left at a crime scene and comparing that DNA to DNA found previously. All of the latter involve DNA pattern matching, for which

> an efficient solution is needed, due to it being expensive both in time and space. For example, in the case of global alignment, the time required to align each character from one sequence to another grows exponentially as the length of search sequences increases. ⟨...⟩

The task of DNA matching is highly computationally expensive: considering algorithms such as the Needleman-Wunsch dynamic programming approach, the problem can have a cubic runtime (reduced down to quadratic with a few optimizations and alterations). To speed up the process of matching, a variety of papers describe approaches of implementing a suitable algorithm on an FPGA platform:

> A field programmable gate array, which will be referred to as "FPGA" in this paper, is a processor that uses a user configured architecture. In most cases, it is a separate processor chip or a part of a chip that has a pre-configured multi-purpose processor on the side. The chip is usually attached to a board that has various types of input and output ports, used for communicating with a host interface to transfer data and debug. ⟨...⟩

2.1 Identifying possible approaches

Early into the development I have identified the main approaches that I will be looking at to implement a method of DNA matching on the provided Zynq-7010 processor. These are described in the following articles by Khan et al. and Shah et. al:

- DNA Sequence Matching System Based on Hardware Accelerators Utilized Efficiently in a Multithreaded Environment by Fahad A. Khan, Aurangzeb and Zubair A. Khan, 2009 (`https://doi.org/10.1109/ICEE.2009.5173172`)
- An Optimized and Low-cost FPGA-based DNA Sequence Alignment A Step towards Personal Genomics by Hurmat Ali Shah, Laiq Hasan, Nasir Ahmad, 2013 (`https://doi.org/10.1109/EMBC.2013.6610096`)

Both articles describe the use of Smith-Waterman algorithm[1] being deployed on different FPGA platforms, exhibiting optimizations to linear DNA matching, such as parallelism exploitation using pipelining or otherwise.

A number of other proposed implementations are published. I will be going into detail comparing the advantages and drawbacks of each to my chosen approach, taking in account the resources and learning / implementation time available, scalability and the level of parallelism of the proposed architecture.

My main goal is to be able to recreate DNA matching logic on the Zynq-7010 processor, interface it with a Linux system running on a general purpose CPU,

[1] Smith-Waterman biological sequence alignment algorithm - https://en.wikipedia.org/wiki/Smith-Waterman_algorithm

and evaluate the performance, identify problems, propose fixes. I present more on that in the next section.

3 Progress on learning and implementation

I have started off with setting up the environment required to develop an architecture on the FPGA along with learning the base terms in the field of sequence matching of bioinformatics. The University has lent me a Digilent ZYBO Zynq-7000 Development Board, which I have fitted with a distribution of embedded Linux using Xillybus[2]. The package provides a template interface for a starter FPGA design with FIFO memory access from Linux using C code. I have explored a number of other options beforehand, mostly open-source projects[3], but it proved to be too cumbersome to deal with building my own kernel: most of the customizable kernel parts were not going to be used, and there was too little time to focus just on learning how to set up a custom environment to implement a single architecture. Upon launched, the Xillybus Linux distro provides a graphical interface along with a starter Verilog FPGA project, sample code with documentation on how to communicate with the FPGA processor.

3.1 Dynamic programming

Dynamic programming is a term that refers to an optimization of a larger problem by splitting it into several sub-problems. For string matching, dynamic programming is defined more precisely as: building a score matrix for two given sequences, calculating the score for every point in the matrix given the previous alignment sub-problem score, and tracing back to locate the global optimum alignment, by following along the highest neighboring scores at each step. The recursive score formula in it's bare form is defined as the following:

$$S(i,j) = max \begin{cases} S(i-1,j-1) + \sigma(x_i, y_j) & match/mismatch \\ S(i-1,j) + \gamma & deletion \\ S(i,j-1) + \gamma & insertion \end{cases}$$

where i and j are the respective subsequences, σ is the alignment score and γ is the gap penalty.

The Smith-Waterman alignment algorithm[4] is slightly modified general DP technique, built upon one of the first implementations (Needleman-Wunsch), used to exploit the fact that optimal alignments between two DNA sequences are just

[2]An FPGA IP core for easy DMA over PCIe with Windows and Linux - http://xillybus.com

[3]https://github.com/PyHDI/zynq-linux, https://github.com/MarioLizanaC/Linaro-O.S.-for-Zybo/

[4]Smith-Waterman algorithm - https://en.wikipedia.org/wiki/Smith-Waterman_algorithm

$$S = \left\{\begin{array}{cccccccccc}
 & - & A & C & A & C & A & C & T & A \\
- & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\
G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\
A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\
C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\
A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\
C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\
A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12
\end{array}\right\}$$

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

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

Figure 1: Example best (red) an arbitrary n-th best (blue) local alignment using Smith-Waterman algorithm. The alignment is found by following coloured entries bottom-up.

in parts of the whole sequences (local alignment). The formula is defined as following:

$$S(i,j) = max \begin{cases} S(i-1,j-1) + \sigma(x_i, y_j) & match/mismatch \\ S(i-1,j) + \gamma & deletion \\ S(i,j-1) + \gamma & insertion \\ 0 \end{cases}$$

given that $S(0,j) = 0$ and $S(i,0) = 0$. If a fixed gap penalty is defined, the algorithm runs in quadratic time, therefore for highly scalable systems optimizations need to be implemented.

Example: consider two DNA sequences, $\{x = ACACACTA, y = AGCACACA\}$, match score $+2$, mismatch penalty -1, gap penalty -1. Suppose a few best optimal alignments are found using the Smith-Waterman dynamic programming equation, the results are then shown in Figure 1.

The Smith-Waterman dynamic programming approach is a technique that can be made suitably parallel and exploit the features of pipelined execution on an FPGA. The articles that I have chosen as my core implementation descriptors are discussed in the following subsections.

3.2 Shah et al. approach

The first steps I tooks were familiarising with the logic described by Shah et al. I was convinced that a fresh look at the problem of DNA matching, especially taking in account the proposed use case (personal genomics), is an interesting approach:

- Using block RAM memory as an integral part of the architecture. Eliminating the need to load sequences into the programmable logic from dedicated

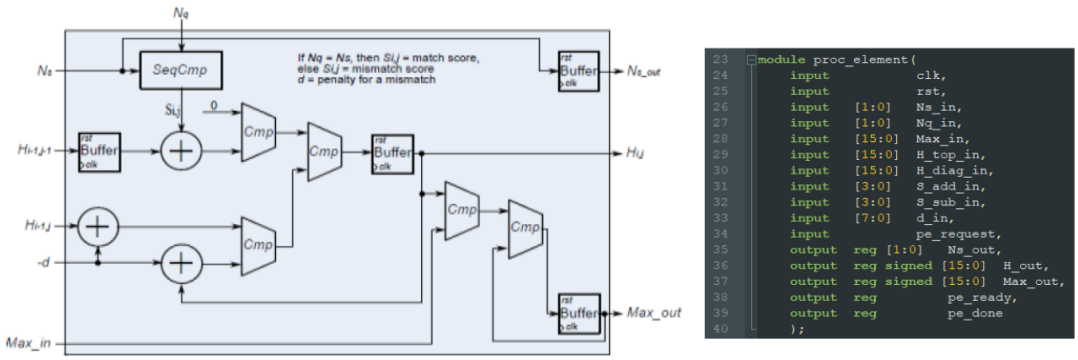

Figure 2: The Shah et al. alignment processing cell along with wrapping Verilog code describing inputs and outputs

RAM saves resources needed for wrapping logic as well as reduces the overhead of time spent loading sequences by employing a faster memory. Most modern day FPGAs, including Digilent ZYBO, have a considerable amount of block RAM which can be utilized.

- Scalability. The processing elements are connected and operate in such a fashion that, adding more units (having a device with more resources) increases the alignment speed linearly.

I have therefore tried to implement and simulate the logic for a processing element using Shah et al. method. The logic design of the processing element is depicted in Figure 3 along with a code excerpt.

Unfortunately, upon further exploration and recreation of the logic I have stumbled upon a few problems:

- Given the time available, I found myself struggling to implement the vaguely described memory management unit for block and distributed RAM address computation. I am currently considering working with a slightly less consuming task, that is, using just the block RAM entirely for the task, as ZYBO board provides enough block RAM for sufficiently long one-time alignments (240KB of block RAM).

- The use of a matrix to store scores in between multiple passes and communicating with block RAM every pass is, as described, quite an overhead for the alignment process, since the PEs remain idle during the write of score matrix columns. This is a limit to the overall parallelism that can be achieved.

  It is pointed out in the article that improvements can be made indeed just in terms of reducing the space required to store the alignment sub-scores and increasing the speed in which the alignment score can be transferred to the client.

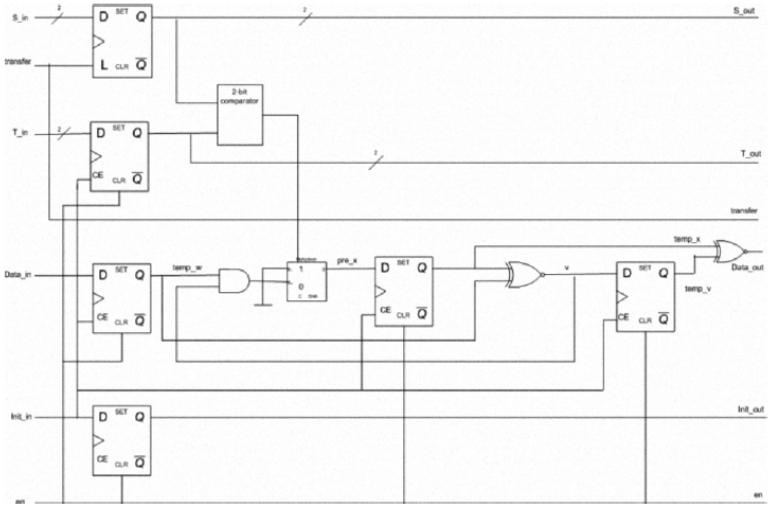

Figure 3: The Khan et al. alignment processing cell

3.3 Khan et al. approach

Now that I have mentioned the approach of Khan et al., let's dive deeper into the details. I have made a sketch of the logic in the latter paper and evaluated the possibility of recreating it in Verilog and deploying on the ZYBO development board. The team describes a way of implementing the Smith-Waterman algorithm for DNA matching and uses findings of Lipton et al.[5] to reduce the runtime from quadratic to linear and minimize the amount of data stored in each processing element, which I find is the biggest win for the described method.

At any point in the alignment procedure, the space complexity is equivalent to the length of the base sequence, as it is stored in the linearly processing elements. It is important to mention that, as opposed to the previously described approach, there is no need to have a score matrix unless the sequence being searched is longer than the amount of processing elements. Of course, this again is a constrained by the amount of hardware resources present.

3.4 Current progress

As of now, I have started to write simulation code for the processing element developed from the method described. I have come up with a piece of Python code for module testing, that converts a DNA sequence provided as pipe input, usually using the `cat` command, into a binary representation, saving it into a binary (machine code) file. The Python code parses the given data, that is, a sequence of DNA bases, in a single line or across multiple lines. The bases are then converted into a 2-bit representation, such that $\{A \equiv 00_2, C \equiv 01_2, G \equiv 10_2, T \equiv 11_2\}$ The

[5] J. Lipton, Richard & Lopresti, Daniel. (1985). A Systolic Array for Rapid String Comparison

tcccatagtcgcctgc...	00000000	03 01 01 01 00 03 00 02 03 01 02 01 01 03 02 01
...cacagggaaccaggga...	00000010	01 00 01 00 02 02 02 00 00 01 01 00 02 02 02 00
...actttatcccatacta...	00000020	00 01 03 03 03 00 03 01 01 01 00 03 00 01 03 00
...acaatttctgtaacaa	00000030	00 01 00 00 03 03 03 01 03 02 03 00 00 01 00 00

Figure 4: Sample DNA sequence conversion to binary results, obtained by running `hexdump -C` on the binary DNA file.

output is given in Figure 4, with the sequence being converted shown on the left and the corresponding binary entries on the right.

3.5 Potential for improvement

I see a possibility of using the fairly big block RAM capacity that is present in the ZYBO development board, by replacing the DRAM units to store the sequences to be aligned. If the signal in the FPGA logic design is propagated without using a clock (dependent on data changes), a clear improvement would be observed in terms of loading sequences into the processing elements. Otherwise, the clock, if used, would be able to operate at a high rate. More on this in the next sections.

4 Plan and objectives

Some of the ways of implementing required logic, described in literature, are too extensive for the time given, using old methods, exploit limited or no parallelism and / or suited to be implemented only on bigger architectures and more expensive FPGA devices. In addition to that, the core task of implementation during the time for the project requires extensive reading and learning.

There is also the need to consider the fact that regular coursework is involved, and time needs to be left for writing-up. Based on the considerations described, I have put up a plan for the rest of the project:

- The current plan is to recreate the logic needed to implement the method described by Khan et al. That is going to require research of a couple more areas, including properly interfacing with the client (Linux, using Xillybus), coupling the alignment logic with reading the sequence off DRAM FIFO and writing the score back. This needs to be completed by the *middle of February, 2018* at the latest.
- Upon having a working prototype, I will be looking into the performance of the prototype, by running tests cases with sample DNA. To achieve that, I will be looking into DNA databases of suitable problem size to evaluate the performance with as little bias as possible. The evaluation *should not take more than two weeks* to complete, including time for potential improvement I have described in Section 3.3.

- Software implementation of Smith-Waterman algorithm or a suitably equivalent procedure to draw comparisons between performance of a general purpose CPU and an FPGA. A benchmark in C++ or other language that would be suitable to communicate with the FPGA without overhead needs to be written to draw conclusions. To achieve this, *another week* will be spent.

The main objective is to successfully adapt a parallel implementation of Smith-Waterman dynamic programming algorithm on the fabric of an FPGA, measure performance improvement over a traditional CPU, identify pitfalls, potentially suggest solutions.

5 Final report outline

By now I have already laid out a preliminary outline of the final project report:

- The report will start off with presenting the basics of sequence matching in computational biology, as a part of chapter the first chapter. A range of terms, mostly focusing on DNA sequence alignment such as global and local alignment, dynamic programming and algorithms, will be presented to familiarise the reader with the content that is coming up next. In the same chapter, there will be an overview of currently employed methods, their limitations and proposals made in the recent years to optimize DNA alignment applications. The final sections will cover the structure of the project, and a list of contributions (own work).
- In the second chapter, I will be talking about the FPGA processor and introduce the ways in which it can be used to exploit fine-grained parallelism for various modern day problems. I will then shift my focus to the DNA matching problem and how an FPGA can come into help. Methods currently in use and recent proposals will be discussed, including reviews of the few that I have selected beforehand. There is still ambiguity as to whether I will set out my goal for the implementation in this section or a later one.
- The next one or two chapters I will be presenting the stages of setting up and implementation of a DNA matching acceleration architecture. This will include details of data retrieval, computation and results output. The chapter is anticipated to be the most technical of all, therefore a generous amount of visual information will be added to complement the description of progress, including excerpts of Verilog code, logic design flows, pipeline diagrams, etc.
- In the last chapter, I will discuss the results obtained by evaluating speed of DNA matching using an FPGA versus a general purpose CPU on a computer using an equivalent algorithm in software. I will also talk about the possible improvements of the system: cover articles that have used the

core methods I used and comment on the improvements made on their behalf compared to my own work. In one of the final sections (or as a part of the discussion part) I will explain how I have organized my time for the project, what difficulties arose that (potentially) restricted me from doing a better job, and a general evaluation of the approach I took and what could be done in the future, building upon my solution.