FPGA Acceleration of the Pair-HMMs Forward Algorithm for DNA Sequence Analysis

Shanshan Ren[1] Vlad-Mihai Sima[1,2] Zaid Al-Ars[1,2]
[1]Computer Engineering Lab
Delft University of Technology
2628 CD Delft, The Netherlands
[2]Bluebee
Molengraaffsingel 12–14
2629 JD Delft, The Netherlands
Email: {s.ren, z.al-ars}@tudelft.nl vlad.sima@bluebee.com

***Abstract*—Many DNA sequence analysis tools have been developed to turn the massive raw DNA sequencing data generated by NGS (Next Generation Sequencing) platforms into biologically meaningful information. The pair-HMMs forward algorithm is widely used to calculate the overall alignment probability needed by a number of DNA analysis tools. In this paper, we propose a novel systolic array design to accelerate the pair-HMMs forward algorithm on FPGAs. A number of architectural features have been implemented to improve the performance of the design, such as early exit points to increase the utilization of the array for small sequence sizes, as well as on-chip buffering to enable the processing of long sequences effectively. We present an implementation of the design on the Convey supercomputing platform. Experimental results show that the FPGA implementation of the pair-HMMs forward algorithm is up to 67x faster, compared to software-only execution.**

Keywords—NGS, FPGA, pair-HMMs, hardware acceleration.

I. INTRODUCTION

NGS technology [1] provides a high-throughput and cost-effective sequencing method of DNA, creating vast opportunities for profound understanding of human disease. The analysis and interpretation of large-scale sequencing data produced by NGS is a major challenge, requiring complex statistical models and sophisticated bioinformatics tools to turning raw sequencing data into biologically meaningful information. There are a number of such tools currently available and being used widely, such as BWA, SAMtools, SOAP, VarScan and GATK.

Pair-HMMs (pair hidden Markov models) [2] are very popular for finding pairwise alignment of DNA sequences. There are two ways to use pair-HMMs in biological sequence alignment: 1) identifying optimal sequence alignment, and 2) providing the overall alignment probability. Using pair-HMMs to find the optimal sequence alignment is very popular in many different biological sequence analysis tools, such as ProbCons [3], PicXAA [4] and GLProbs [5]. Pair-HMMs identify the alignment with the largest probability as the optimal sequence alignment. The algorithm to find the optimal sequence alignment of pair-HMMs is called the Viterbi algorithm.

If the similarity of the two sequences is not strong, it is hard to find the correct alignment that gives biological meaning. Instead, pair-HMMs can then be used to calculate the probability that two sequences are related, which is referred to as the overall alignment probability [2]. The overall alignment probability is widely used in many biological sequence analysis tools. For example, [6] exploits the overall alignment probability to find the evolutionary distance between two sequences. The GATK HaplotypeCaller [7] calculates the overall alignment probability of the sequences and the candidate mutations to identify their occurrence reliability.

Pair-HMMs forward algorithm computes the overall alignment probability by summing over all possible alignments of a given pair of DNA sequences. The forward algorithm is a dynamic programming algorithm with a computational complexity of O(*nm*) (*n* and *m* are the length of two sequences), which is very large for long sequences. This drawback would influence the performance and limit the feasibility of pair-HMMs. In this paper, we investigate and propose an FPGA-based acceleration of the pair-HMMs forward algorithm with the purpose of improving its performance.

In this paper, we present the following contributions: (1) propose a novel systolic array design of the pair-HMMs forward algorithm; (2) analyze a number of optimization techniques to improve performance; and (3) present an implementation of the design on the Convey supercomputing platform. The results shows that the FPGA-based implementation is around 67x faster, compared to the software-only execution.

The rest of this paper is organized as follows. Section 2 presents a brief overview of related work. Section 3 discusses the details of the pair-HMMs forward algorithm. Section 4 discusses the design specification and optimizations of the accelerated version of the algorithm. The implementation results are discussed in Section 5. We conclude the paper and discuss future work in Section 6.

II. RELATED WORK

FPGAs are widely used to accelerate biological algorithms to achieve large speedup as many bioinformatics workloads lend themselves well to parallel execution. Examples range from commercially available implementations such as the Tera-BLAST [8], to more research oriented algorithm acceleration, such as the acceleration of SAMtools [9]. Tera-BLAST is an FGPA implementation of the BLAST aligner that achieves a 27x speedup over a 32-core CPU implementation. [9] accelerates SAMtools on FPGAs, which achieve a speedup of 2.93x over the original version of SAMtools.

As the Viterbi algorithm has been used in many topics, such as pairwise alignments, multiple sequence alignment and gene prediction, there is much research focusing on the

978-1-4673-6799-8/15/$31.00 ©2015 IEEE

acceleration of the Viterbi algorithm [10][11][12]. The acceleration of the Viterbi algorithm commonly utilizes a log-transformation of the original equations, which transform floating-point multiplication operations into floating-point addition operations. The forward algorithm, on the other hand, requires an addition operation in the probability domain, which prevents using the log-transformation to the forward algorithm. Therefore the acceleration approach used for the Viterbi algorithm cannot be applied to accelerate the forward algorithm.

III. PAIR-HMMS FORWARD ALGORITHM

Pair-HMMs have evolved from the basic HMMs. In a pair-HMM, the HMM model generates an aligned pair of sequences instead of only a single sequence. Figure 1 shows a typical pair-HMM, which is widely used in biological sequence analysis. We assume the two sequences generated by the pair-HMM are sequence X and Y. Figure 1(a) shows the state transition of the pair-HMM, which has three hidden states I_X, I_Y and M. I_X and I_Y are used to emit a single unaligned symbol only in sequence X and Y, respectively. M is used to emit an aligned pair of two symbols, where one symbol is added to sequence X and the other symbol is added to sequence Y. By traversing between the states I_X, I_Y and M, the pair-HMM generates two sequences. Figure 1(b) shows an example of an aligned pair of sequences X=ACGTC and Y=ACGAA, which are generated according to the hidden state sequence $MMMI_XI_XI_YI_Y$. The first three symbols in sequence X and sequence Y are emitted by state M. The last two symbols in sequence X are emitted by state I_X and the last two symbols in sequence Y are emitted by state I_Y. The probability of the generated alignment is the product of the state transition probabilities.

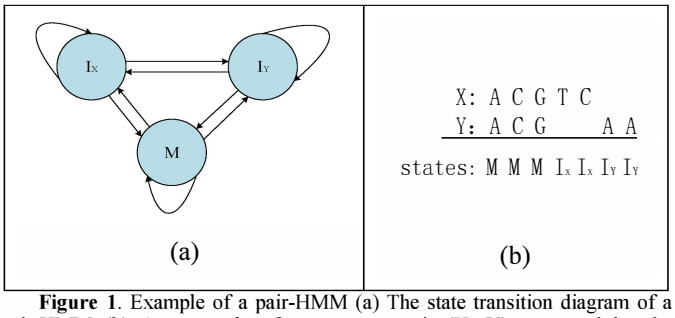


Figure 1. Example of a pair-HMM (a) The state transition diagram of a pair-HMM (b) An example of a sequence pair (X, Y) generated by the pair-HMM

From the simple example shown by Figure 1, we could see that a hidden state sequence generates an aligned pair of sequences with a specific alignment probability. If we want to find the overall alignment probability, we need to add the alignment probability of all hidden state sequences. Obviously, it is not practical to enumerate all hidden state sequences. Thus, the forward algorithm is proposed to solve this problem.

The pair-HMMs forward algorithm is implemented as a dynamic programming algorithm, as shown by Equations (1) to (3), where n and m are the length of sequence X and Y, respectively. α, β, γ, δ, ε, ζ and η are the transmission probabilities, while λ, θ, and υ are the emission probabilities. The transmission probabilities and the emission probabilities are supplied by the two sequences. In these equations, $M_{i,j}$ stands for the overall alignment probability of two sub-sequence X[1]…X[i] and Y[1]…Y[j]. $I_{i,j}$ stands for the overall alignment probability of X[1]…X[i] and Y[1]…Y[j] with X[i] aligned to gap. $D_{i,j}$ stands for the overall alignment probability of X[1]…X[i] and Y[1]…Y[j] with Y[j] aligned to gap.

Initialization:

$$\begin{cases} M_{0,0} = 1, I_{0,0} = D_{0,0} = 0 \\ M_{i,0} = I_{i,0} = 0, 0 < i \le n \\ M_{0,j} = D_{0,j} = 0, 0 < j \le m \end{cases} \quad (1)$$

Recurrence:

$$\begin{cases} M_{i,j} = \lambda \times (\alpha M_{i-1,j-1} + \beta I_{i-1,j-1} + \gamma D_{i-1,j-1}) \\ I_{i,j} = \theta \times (\delta M_{i-1,j} + \varepsilon I_{i-1,j}) \\ D_{i,j} = \upsilon \times (\zeta M_{i,j-1} + \eta D_{i,j-1}) \end{cases} \quad (2)$$

$$(1 \le i \le n, 1 \le j \le m)$$

Termination:

$$Result = M_{n,m} + I_{n,m} + D_{n,m} \quad (3)$$

As shown by these equations, three matrices are filled and the process to fill these matrices contains much inherent parallelism. Each matrix element of $M_{i,j}$, $I_{i,j}$ and $D_{i,j}$ only depends on the up-left, up and left neighbor elements of each matrix. This implies all elements on the same anti-diagonal in each matrix can be computed in parallel.

Algorithm 1 shows a pseudo code of the pair-HMMs forward algorithm. As shown by Algorithm 1, the computation complexity of the pair-HMMs forward algorithm is $O(nm)$. When the lengths of the two sequences increase, the execution time of the dynamic programming algorithm increases quadratically, which causes the high computational complexity of pair-HMMs.

Numerical underflow is a significant problem when implementing the pair-HMMs forward algorithm, as the probability of some alignments would be smaller than the smallest representable floating-point value. There are two methods to solve this problem: implementing a "log-sum" operation and rescaling [13][14]. The first method transforms the floating-point multiplications into floating-point addition in the log probability domain, but it needs to build a large look-up table containing the values used in the computation process. The second method is rescaling, which rectifies the intermediate results, however leading to many extra computations.

Some DNA sequence analysis tools, such as GATK HapltoypeCaller, implement the pair-HMMs forward algorithm directly in the probability domain without using either of these two complex methods. The reason is that, on the one hand, the numerical underflow problem does not frequently occur in DNA sequence analysis, as the length of DNA sequences is 70~250bps and numbers in the double floating-point format have an approximate range of 10^{-308} to 10^{308}. On the other hand, if a number underflows, the impact on the final result is negligible. Thus, we would implement the pair-HMMs forward algorithm in the probability domain as well.

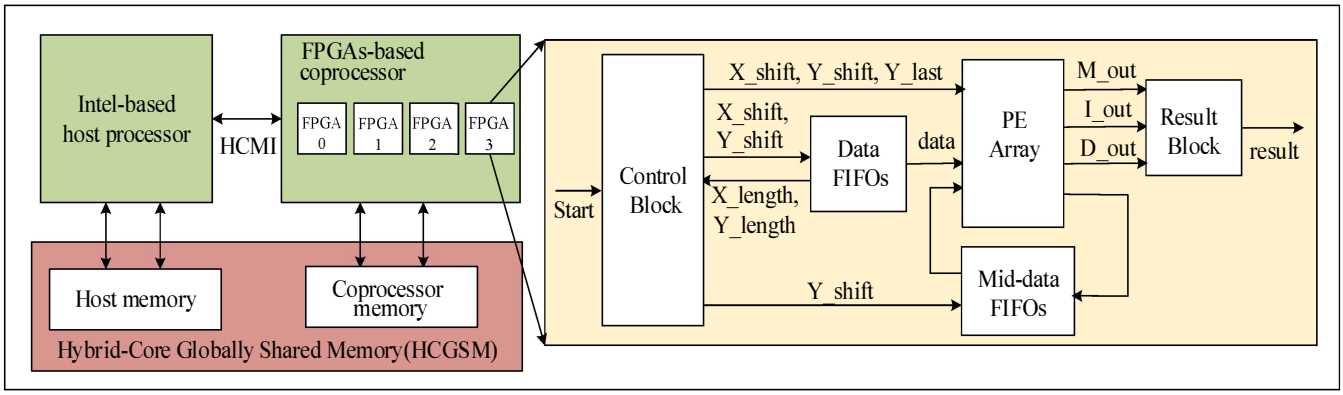


Figure 2. Block diagram of the overall architecture design

Algorithm 1. The pseudo code of the pair-HMMs forward algorithm

```
M[0][0]=1;
D[0][0]=I[0][0]=0;
for(i=1;i<=n;i++)
begin
     M[i][0]=I[i][0]=0;
end
for(i=1;i<=m;i++)
begin
     M[0][i]=D[0][i]=0;
end
for(i=1;i<=n;i++)
begin
     for(j=1;j<=m;j++)
     begin
     M[i][j]=λ×
(α×M[i-1][j-1]+β×I[i-1][j-1]+γ×D[i-1][j-1])
     I[i][j]=θ×(δ×M[i-1][j]+ε×I[i-1][j])
     D[i][j]=υ×(ζ×M[i][j-1]+η×D[i][j-1])
     end
end

result=M[n][m]+I[n][m]+D[n][m];
```

IV. DESIGN SPECIFICATION

In this section, we first present an overview of the system architecture of the FPGA implementation and then introduce the details of the hardware design.

A. Architecture overview

Convey proposed innovative hybrid-core platforms (HC-1, HC-1ex, HC-2 and HC-2ex) [15], which combine classic Intel x86 microprocessors with a coprocessor comprised of FPGAs. The platforms are useful for acceleration of computationally intensive applications, by offloading complex kernels onto the FPGA at runtime, while running the other part of the application on the host processor. We exploited HC-2ex to implement the acceleration of the pair-HMMs forward algorithm. Figure 2 shows the overall architecture of the design.

As shown in Figure 2, the host processor and coprocessor have their own physical memory. The two physical memories are mapped in the same virtual memory address space, which makes it very easy for the application developers. Efficient data transfer mechanisms are provided for transferring between the two memories. The hybrid-core memory interconnect (HCMI) is responsible for signals between the host processor and the coprocessor, such as the start signal sent by the host processor and the finish signal sent by the coprocessor.

The pair-HMMs forward algorithm is implemented on the coprocessor. The function of each block is described below:

Control Block: This block is used to control the progress of the pair-HMMs algorithm. It gets the start signal from the host processor, which indicates that the FPGAs can start their computational cycle; it gets the length of two sequences from Data FIFOs.

Data FIFOs: These FIFOs are used to store data from the memory and output data to the PE array and Control Block according to the control signals from the Control Block.

PE Array: This is used to compute the elements in the three matrixes M(i,j), I(i,j) and D(i,j). It is described in Section IV.B.2.

Mid-data FIFOs: This block is used to store the intermediate data generated by the PE array.

Result Block: This block consists of three floating-point adders and one register to calculate the result.

B. Systolic array mapping

1) Systolic arrays

The potential parallelism of the pair-HMMs forward algorithm allows us to exploit systolic arrays to accelerate the performance on FPGAs. Systolic arrays were proposed by H. T. Kung and C. E. Leiserson [16] and have since been widely used in computing matrix multiplication, LU-decomposition and dynamic programming algorithms. A systolic array is an array of identical Processing Elements (PEs), each of which gets its inputs from the previous PE and passes its outputs to the next PE. All these PEs operate in parallel. This characteristic can be used to compute the elements on the same anti-diagonal in each matrix, which can be computed in parallel.

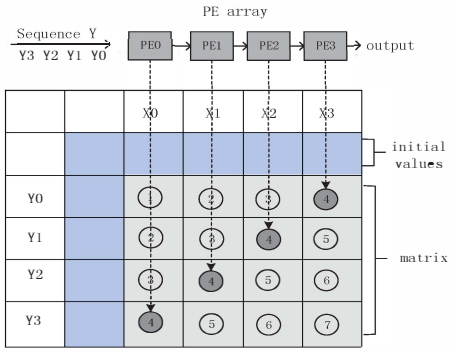
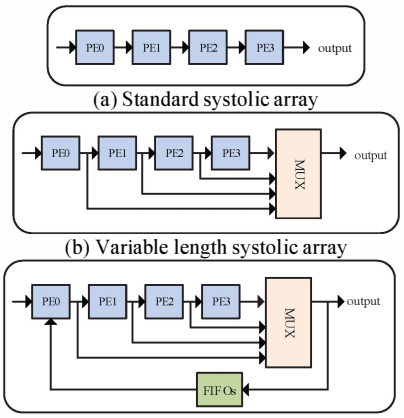


Figure 3. Mapping the pair-HMMs forward algorithm to a systolic array

We map the pair-HMMs forward algorithm to a systolic array, as shown by Figure 3. Sequence X is shifted through the array and each PE stores one of the elements of X. Then the bases of Sequence Y shift through the PEs. In each cycle, a PE calculates one element of each matrix and passes the resulting values to the next element. As the PEs run in parallel, the PE array calculates the elements in the same anti-diagonal in each cycle. For example, in the fourth clock cycle, the PE array is mapped to calculate the elements marked with 4.



(a) Standard systolic array

(b) Variable length systolic array

(c) Variable length systolic array with FIFOs

Figure 4. Systolic arrays

Figure 4(a) shows one implementation of the PE array in case the number of PEs is equal to the read length. In this case, the total computation time is shown by Equation (4).

$$T = 2 \times \text{Length}(X) + \text{Length}(Y) - 1 \quad (4)$$

If the FPGA is able to host more PEs than the needed length of Sequence X, the total computation time would be

$$T = 2 \times \text{PEnumber} + \text{Length}(Y) - 1 \quad (5)$$

In this case, the computation time is not limited by the actual length of the sequence, but by the time needed for the systolic array to complete processing all its PEs. This is caused by the fact that Sequence X and Y need to travel through the entire PE array in order to produce the final result to the output, thereby causing unnecessary latency, in addition to a reduced utilization efficiency of the systolic array. To overcome this limitation, we insert exit points after each PE in the PE array, as shown in Figure 4(b) [17]. Using this method, the bases of Sequence X are copied into the needed PEs and Sequence Y only needs to travel through the PEs where the bases are stored. Thus, the total computation time is reduced according to Equation (4).

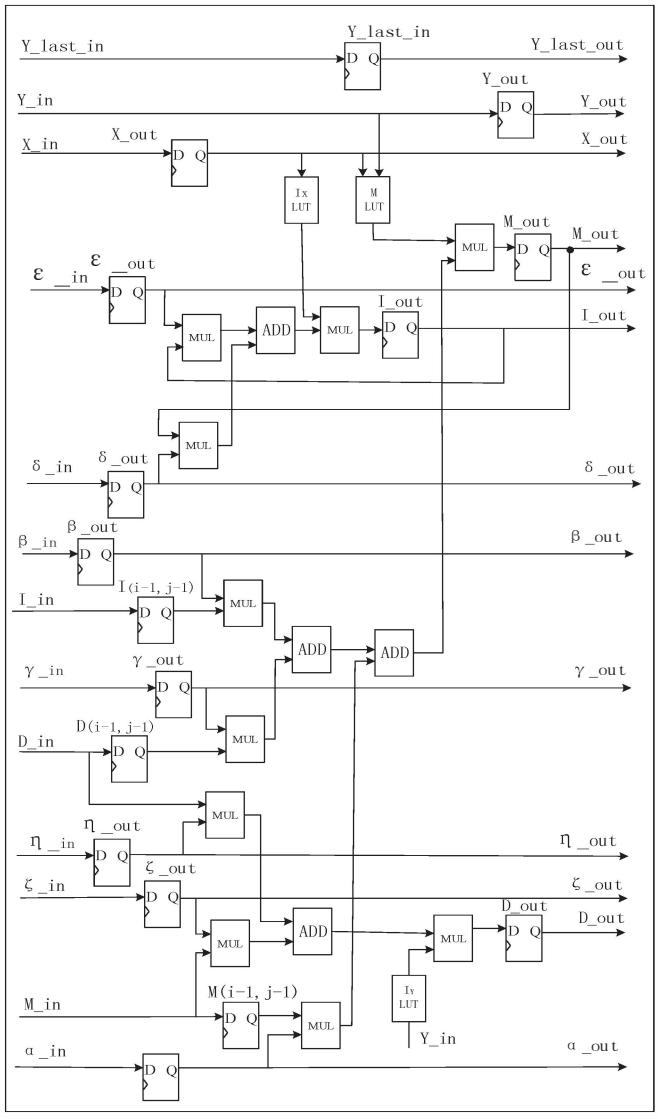


Figure 5. PE schematic for the pair-HMMs forward algorithm

If the number of PEs is smaller than the length of Sequence X, we divide the read into sub-sequences. In each iteration, one of the sub-sequences is shifted into the systolic array and the results are stored in the FIFOs. In subsequent iterations, the results in the Mid-data FIFOs and the next sub-sequence are shifted into the systolic array. This PE array is shown in Figure 4(c). The number of iterations is specified according to the length of Sequence X and the number of PEs in the systolic array. The actual systolic array implemented on the Convey FPGA is the one shown in Figure 4(c).

2) PE schematic

Figure 5 shows the PE schematic for the pair-HMMs forward algorithm. X_in and Y_in represent the base of the Sequence X and Y respectively. α_in, β_in, γ_in, δ_in, ϵ_in, ζ_in and η_in, represent α, β, γ, δ, ϵ, ζ and η respectively.

M_in, D_in and I_in respectively represent M(i,j), D(i,j) and I(i,j), which are calculated by the previous PE. Y_last_in indicates the last base of the Sequence Y is shifting into the PE

As shown by Figure 5, there are 10 floating-point multipliers and 4 floating-point adders. In order to avoid the high area costs of floating point arithmetic units, we use the DSP DSP48E1 components on the FPGAs to implement these arithmetic computations.

C. Transfer overhead

When the host processor has a pair of sequences to compute its overall alignment probability, the first step is that the host processor sends a start signal to the coprocessor and the coprocessor start to execute. Then the Control Block sends request to the memory to transfer data and store data in FIFOs on FPGAs. When the data transfer finishes, the Control Block sends signal to the PE array and FIFOs to start to compute. When the computation finishes, the Control Block writes results back to memory.

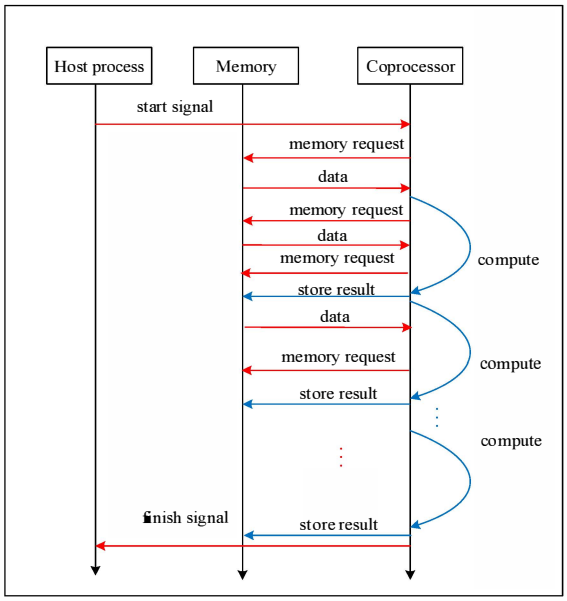


Figure 6. The flow chart of the execution for the pair-HMMs forward algorithm.

If we have several pairs of sequences to deal with, the steps described above are not an efficient solution as the data transfer is very time consuming and it makes the execution time very large. We propose a flow chart of the execution of the forward algorithm, as shown by Figure 6. For the first pair of sequences, the coprocessor waits for the data transfer from memory. As the coprocessor is computing the result of the first pair of sequences, it continues to load data from memory to be stored in the Data FIFOs. The size of Data FIFOs decides the size of the preload data. Based on profile of the FPGA implementation we determined that the most efficient is that the Data FIFOs holds 4 pairs. When the coprocessor finishes the computation of the first pair, it does not need to wait for data transfer of the following pairs of sequences and starts to compute immediately. In this way, data transfer and computation work in parallel, reducing the total execution time by hiding the data transfer overhead.

V. Experimental Results

A. Experimental setup

All tests were run on the Convey HC-2EX platform. The platform has two Intel Xeon E5-263 processors (four cores each, HyperThreading disabled) running at 3.3 GHz with 64 GB of DDR3 memory and four Xilinx Virtex-6 LX760 FPGA co-processors each with 64 GB of SG-DIMM of memory.

Each FPGA is programmed with a pair-HMMs forward algorithm module. All modules on each FPGA run in parallel. Each FPGA contains the same number of PEs, which is limited by the available resources on the FPGA chip. The PE array working frequency is 75MHz.

The pair-HMMs forward algorithm module is implemented using single-precision floating-point variables as it is the case in software packages use pair-HMMs. In general, this is adequate for most sequence analysis tools. If numerical underflow occurs using single precision floats, this will be signaled and recalculated by the pair-HMMs forward algorithm in double precision in the software.

We use the datasets downloaded from [18] to evaluate the performance. The dataset represents pair-HMMs inputs generated by HaplotypeCaller from GATK version 2.7, while calling the 1000 Genomes Project sample NA12878 which is publicly available in the 1000G FTP.

B. Speedup

First, we run the software-only implementation programmed in C++ with default parameters (according to the pseudo code shown by Algorithm 1) to measure the baseline performance. We also run 5 different FPGA implementations of the algorithm with various number of exit points: 1, 2, 3, 4 and all possible exit points. The actual locations of the exit points in each of these designs are listed in Table 1. Note that the increase in the number of exit points uses a larger portion of the FPGA, which results in a corresponding drop in the number of synthesizable PEs. With only one exit point, 96 PEs can be synthesized, while using all exit points reduces this number to 91 PEs.

Table 1. Location of exit points in the different FPGA designs

# exit points	Location of exit points	# synthesizable PEs
1	96	96 PEs
2	46, 93	93 PEs
3	31, 62, 93	93 PEs
4	25, 50, 75, 93	93 PEs
91	All	91 PEs

Figure 7 shows the maximum speedup attainable from the 5 different implementations with different number of exit points as compared with the software-only implementation. The implementation with 91 PEs and 91 exit points achieves the highest amount of speedup of 67x. The implementation with only one exit point results in the least amount of performance achieving a speedup of 62x. It is interesting to note that, the speedup correlates better with the number of exit points rather than the number of PEs, thereby increasing the performance

with the increasing number of exit points. This can be explained by the fact that PEs are relatively large in size and therefore limited in number, which means that reducing the number of PEs by a limited amount can free enough resources to connect all PEs to an exit point.

This also indicates that the performance depends on the length of the sequences in the input dataset, which causes the performance to change with a changing data set.

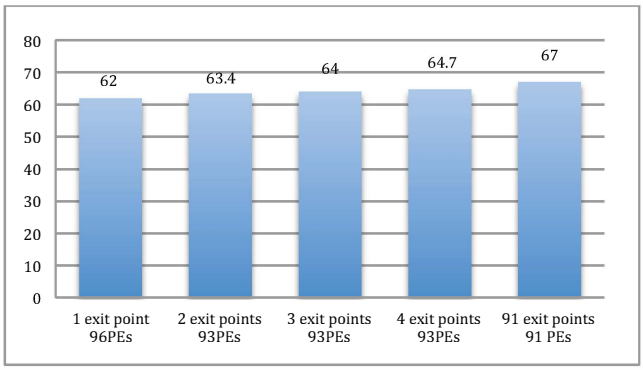


Figure 7. Speedup of the different number of exit points implementation

Table 2 lists the hardware resource utilization of the various designs used in the analysis. The table shows that the implementations are mainly limited by the number of used DSPs (used for single-precision floating-point calculations) and the occupied slices. The table also shows that registers are under utilized in all designs.

Table 2. Hardware resource utilization of different designs

# exit points	# PEs	%LUTs	%Registers	%DSPs	%Occupied slices
1	96	92%	24%	100%	98%
2 to 4	93	90%	23%	97%	98%
91	91	89%	23%	95%	98%

VI. Conclusions and Future Work

In this paper, we propose a novel systolic array design to accelerate the pair-HMMs forward algorithm on FPGAs. A number of architectural features have been implemented to improve the performance of the design, such as early exit points to increase the utilization of the array for small sequence sizes, as well as on-chip buffering to enable the processing of long sequences effectively. We implemented the design on the Convey supercomputing platform. Experimental results show that the improved pair-HMMs algorithm achieves a speedup of 67x, compared to software-only execution. The main limitation in the FPGA implementation is related to the complex floating point calculations needed by the pair-HMMs forward algorithm. This limits the design frequency to 75MHz.

In the future, we plan to accelerate the pair-HMMs forward algorithm on a GPU platform to investigate its capabilities to efficiently process floating-point operations in parallel.

References

[1] Jay Shendure and Hanlee Ji, Next-generation DNA sequencing. Nature. Biotechnology 26, 2008, 1135–1145

[2] Richard Durbin, et al., 1998. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, page90-109

[3] Chuong B. Do, Mahathi S.P. Mahabhashyam, Michael Brudno, and Serafim Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. Genome Res 2005. 15(2):330-340.

[4] Sayed Mohammad Ebrahim Sahraeian and Byung-Jun Yoon. PicXAA: greedy probabilistic construction of maximum expected accuracy alignment of multiple sequences. Nucleic Acids Res 2010, 38(15): 41-4928.

[5] Yongtao Ye, et al.. GLProbs: Aligning Multiple Sequences Adaptively. IEEE/ACM transactions on computational biology and bioinformatics, vol. 12, No.1, 2015, 1: 67-78.

[6] Bjarne Knudsen, Michael M. Miyamoto, Sequence Alignments and Pair Hidden Markov Models Using Evolutionary History. J. Mol. Biol., 2003, 333, 453-460.

[7] DePristo M, et al., A framework for variation discovery and genotyping using next-generation DNA sequencing data. 2011 NATURE GENETICS 43:491-498

[8] Accelerated BLAST Performance with Tera-BLASTTM: a comparison of FPGA versus GPU and CPU BLAST implementations," May 2013, TimeLogic Division, Active Motif Inc. [Online]. http://www.timelogic.com/documents/TimeLogic_Tera-BLAST_whitepaper_v1.0.pdf

[9] Brett Dutro, Hardware acceleration of the SAMtools variant caller, 2015. https://www.ideals.illinois.edu/bitstream/handle/2142/72860/Brett_Dutro.pdf?sequence=1

[10] Oliver TF, Schmidt B, Jakop Y, Maskell DL. High speed biological sequence analysis with hidden Markov models on reconfigurable platforms. IEEE Transactions on Infromation Technology in Biomedicine, vol. 13, no.5, pp。740-746， 2009.

[11] Yangteng Sun, et al., HMMer acceleration using systolic array based reconfigurable architecture, in Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '09), New York, NY, USA, May 2009

[12] Steven Derrien and Patrice Quinton, Parallelizing HMMER for hardware acceleration on FPGAs, in Proceedings of the International Conference on Application-specific Systems, Architectures and Processors (ASAP '07), pp. 10–17, Montreal, Canada, July 2007

[13] Sean S. Eddy, Accelerated Profile HMM Searches. *PLoS Comput. Biol.***7**, Oct. 2011. pcbi:1002195

[14] W. Kurdthongmee, A Modified HMM Forward Algorithm for an Embedded Motion Type Classification. International Journal o fSignal processing System, Vol.2, No.2, Dec., 2014, pp.84-90

[15] The Convey HC-2 ™ Computer Architectural Overview http://www.conveycomputer.com/files/4113/5394/7097/Convey_HC-2_Architectual_Overview.pdf

[16] Hsiang Tsung Kung and Charles Eric Leiserson, 1978. Systolic Arrays for VLSI, Interim report, Department of Computer Science, Carnegie Mellon University

[17] Ernst Houtgast, et al., An FPGA-based systolic array to accelerate the BWA-MEM Genomic Mapping Algorithm (September 2015). International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XV 2015, July 2015, Greece.

[18] https://github.com/MauricioCarneiro/PairHMM/tree/master/test_data