High Performance Streaming Smith-Waterman Implementation with Implicit Synchronization on Intel FPGA using OpenCL

Houtgast, Ernst; Sima, Vlad; Al-Ars, Zaid

DOI
10.1109/BIBE.2017.000-6

Publication date
2017

Published in
2017 IEEE 17th International Conference on BioInformatics and BioEngineering (BIBE)

2017 2017 IEEE 17th International Conference on Bioinformatics and Bioengineering

High Performance Streaming Smith-Waterman Implementation with Implicit Synchronization on Intel FPGA using OpenCL

Ernst Joachim Houtgast[1,2], Vlad-Mihai Sima[1]
[1]*Bluebee Research & Development*
Bluebee BV
Rijswijk, The Netherlands
E-mail: {ernst.houtgast, vlad.sima}@bluebee.com

Zaid Al-Ars[2]
[2]*Department of Computer Engineering*
Delft University of Technology
Delft, The Netherlands
E-mail: {e.j.houtgast, z.al-ars}@tudelft.nl

***Abstract*—The Smith-Waterman algorithm is widely used in bioinformatics and is often used as a benchmark of FPGA performance. Here we present our highly optimized Smith-Waterman implementation on Intel FPGAs using OpenCL. Our implementation is both faster and more efficient than other current Smith-Waterman implementations, obtaining a theoretical performance of 214 GCUPS. Moreover, due to the streaming, implicit synchronizing nature of our implementation, which streams alignments and places no restrictions on the number of alignments in flight, it achieves 99.8% of this performance in practice, almost three times as fast as previous implementations. The expressiveness of OpenCL results in a significant reduction in lines of code, and in a significant reduction of development time compared to programming in regular hardware description languages.**

***Keywords*-FPGA; OpenCL; Smith-Waterman; systolic array**

I. Introduction

The Smith-Waterman algorithm [1] can be used to find the optimal pairwise alignment between two (sub)sequences of symbols, which in the context of bioinformatics usually means sequences of amino acids (for protein sequences) or nucleotides (for DNA sequences). Given a certain scoring scheme that awards matching symbols and penalizes differences or missing symbols, it uses a dynamic programming approach to calculate the optimal alignment between the sequences. The continued growth of bioinformatics data sets makes optimized and/or accelerated implementations of key algorithms of vital importance.

Field-Programmable Gate Arrays (or FPGAs), with their flexible and reprogrammable substrate, are a natural fit for a computationally intensive algorithm such as the Smith-Waterman algorithm. However, programming FPGAs through hardware description languages such as VHDL or Verilog is difficult, being somewhat comparable to writing software in assembly language. The rise of higher level programming languages such as OpenCL makes FPGA programming a much more accessible venture.

In this paper we present the following contributions:

- An OpenCL FPGA Smith-Waterman implementation that, limited development complexity notwithstanding, outperforms other implementations almost threefold;
- A streaming systolic array architecture that eliminates a key design issue: low utilization of the systolic array.

The remainder of this paper is organized as follows. We review related work in Section II. In Section III, we explain the Smith-Waterman algorithm. In Section IV, the two key features of our implementation, streaming and implicit synchronization, are discussed. Methods and results are presented in Section V and VI, respectively. A discussion follows in Section VII. Section VIII concludes the paper.

II. Related Work

As the Smith-Waterman algorithm [1] is a common algorithm in bioinformatics, it has received much attention to optimize the algorithm's performance, resulting in numerous accelerated implementations. The fastest software-only Smith-Waterman version is SSW [2], which extends the striped Smith-Waterman approach of Farrar [3]. These implementations make pervasive use of SIMD instructions to attain their excellent performance. However, accelerator-based implementations are still able to significantly outperform software-only implementations. For example, the GPU-based CUDASW++ 3.0 [4] is able to attain a performance of 119.0 GCUPS on a GeForce GTX 680. The highest performing FPGA-based implementation is the implementation from Sirasao[5], which attains a performance of 135.4 GCUPS using an AlphaData board with a Xilinx Virtex-7. Moreover, their FPGA-based design is significantly more efficient compared to most GPU-implementation, requiring an order of magnitude less power compared to a GPU-based approach. They measured a power-efficiency of 2.8 GCUPS/Watt compared to 0.24 GCUPS/Watt on the GPU.

Similar to the Sirasao implementation, we use OpenCL as our implementation platform. However, our streaming implementation with implicit synchronization makes pervasive use of OpenCL features such as kernels and channels to allow for a higher performing, and more importantly, a much more efficient Smith-Waterman implementation. Utilization of our design approaches 100%, compared to 57% utilization of the Sirasao design. As a result, our implementation outperforms their implementation by almost three-fold.

2471-7819/17/31.00 ©2017 IEEE
DOI 10.1109/BIBE.2017.00089

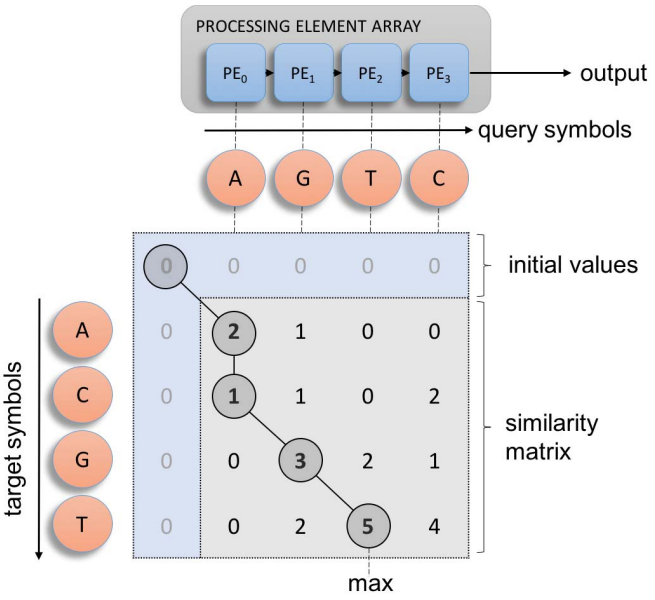

Figure 1. The Smith-Waterman algorithm operates by filling a dynamic programming-based similarity matrix. Cells inside the matrix are only dependent on their top, top-left, and left neighbor, allowing anti-diagonals of the matrix to be processed in parallel. This maps naturally onto a systolic array of Processing Elements. Each Processing Element calculates the column for one of the query symbols. The traceback phase works backwards from the highest scoring cell to produce the actual alignment.

III. Smith-Waterman Algorithm

The Smith-Waterman algorithm is guaranteed to find the optimal pairwise alignment of two sequences. It consists of two phases: first, it uses a dynamic programming approach to fill a similarity matrix, followed by a traceback phase to retrieve the optimal alignment. As the first phase is the most computationally demanding, it is the focus of this work.

The Smith-Waterman equations that govern similarity matrix score calculations are similar to the Needleman-Wunsch algorithm [6], except that disallowing negative values makes the algorithm search for optimal local alignments, as compared to optimal global alignments:

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + s(a_i, b_j) & : \text{(mis)match} \\ H_{i-1,j} - \textit{gap penalty} & : \text{insertion/deletion} \\ H_{i,j-1} - \textit{gap penalty} & : \text{insertion/deletion} \\ 0 & : \text{local alignment} \end{cases}$$

This is illustrated in Figure 1. The highest scoring cell in the similarity matrix indicates the optimal alignment score. From the above equations it is clear that each cell in the similarity matrix only depends on its top, top-left, and left neighbor. Therefore, anti-diagonals in the matrix are independent of one another and can be calculated in parallel: a wavefront of parallelism flows through the similarity matrix. This maps nicely onto a systolic array of Processing Elements, which is the typical approach when implementing the Smith-Waterman algorithm on an FPGA.

IV. Implementation Details

Our implementation uses the Intel FPGA SDK for OpenCL. Two key concepts of OpenCL are kernels and channels. An OpenCL kernel is a function executed on a compute device, so in the case of an FPGA, this is synthesized into actual hardware. Multiple identical kernels are synthesized to work in parallel to achieve higher throughput. An OpenCL channel is a mechanism that implements on-chip low-latency, high bandwidth communication between kernels. Our implementation makes pervasive use of channels, only using the on-board DDR for reading of the input sequences and writing of the output scores.

Figure 2 shows the kernels and channels of a single Smith-Waterman module. The largest area is reserved for the systolic array of Processing Elements, which calculates the Smith-Waterman similarity matrix. For each alignment, each Processing Element has its unique query symbol, whereas the symbols of the target sequence flow through the array. There are two key innovations that work in unison to allow for high utilization of the systolic array:

Implicit Synchronization: Our implementation does not use a dedicated control unit. Instead, control and synchronization is implicitly arranged within each kernel. Control and data signals flow from left to the right: the Input Parser sends packets to the Target and Query Loaders, and to the Result Parser. These packets contain all the information required to know how many iterations this particular alignment requires. This allows each kernel to work independently without explicit synchronization with other kernels.

Streaming: Typically, due to the central control, a systolic array is only able to work on a single alignment at a time. However, the distributed control of our implementation allows for a streaming nature. This is illustrated in Figure 3. A key enabler is the use of Query Buffers, which for each Processing Element hold the query symbols for upcoming alignments. Whenever the current alignment is finished, a New Read token is passed through the array, signaling to a PE that it should reinitialize and load a new Query symbol.

The combined effect of both innovations is that, except for the first alignment, processing time for an alignment only depends on the length of the target sequence, and is independent of query length. The only overhead is the New Read token that is passed to indicate a new alignment. Thus, very short target sequences do slightly impact efficiency. In contrast, efficiency of the systolic array is mostly dependent on the query length as compared to systolic array size, as for shorter queries part of the systolic array will be idle. To alleviate this, it would be possible to use multiple systolic arrays of different size (refer to [7] for more details). Our designs use multiple identically sized modules to utilize all available resources on the FPGA.

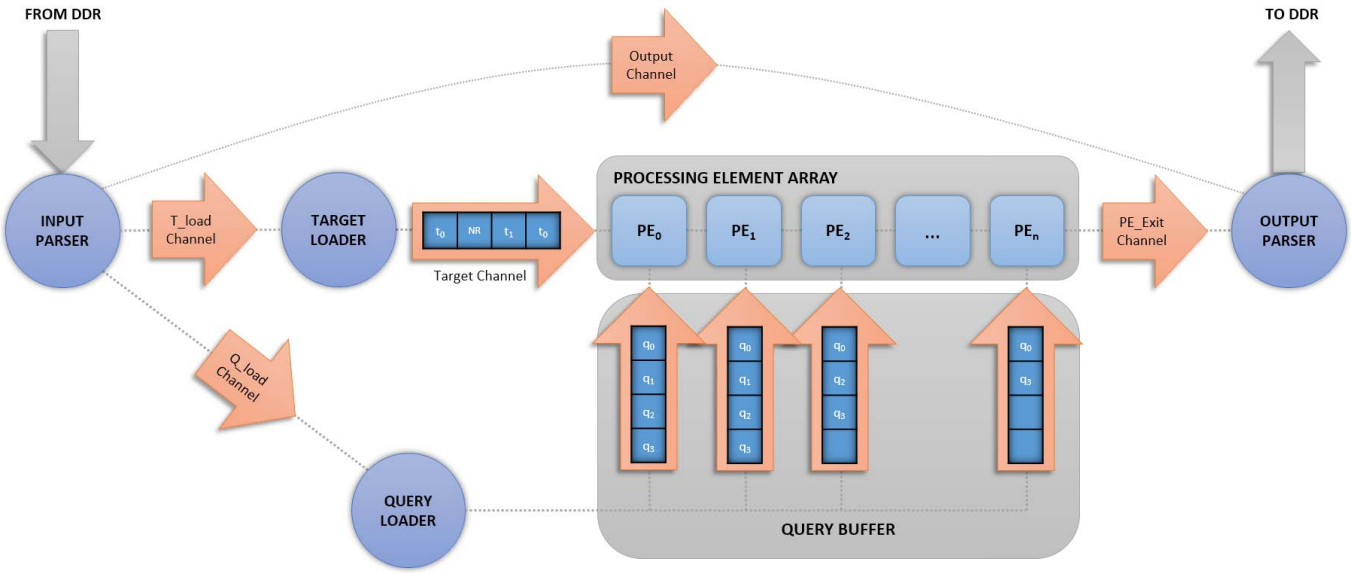

Figure 2. Overview of the kernels and channels of the Smith-Waterman module. Control and data signals flow from left to right through OpenCL channels, with no dependencies or loops, removing any limitation on the number of simultaneous alignments in flight. This way, kernels are decoupled from one another and operate asynchronously. The Query Buffer contains for each Processing Element a separate queue with query symbols for the alignments it needs to process. Whenever the Processing Element encounters the new read token, it checks against the query length to verify if it is active during this alignment; if so, it reads the next query symbol from its queue. Only the Input Parser and Output Parser communicate with the on-board DDR memory.

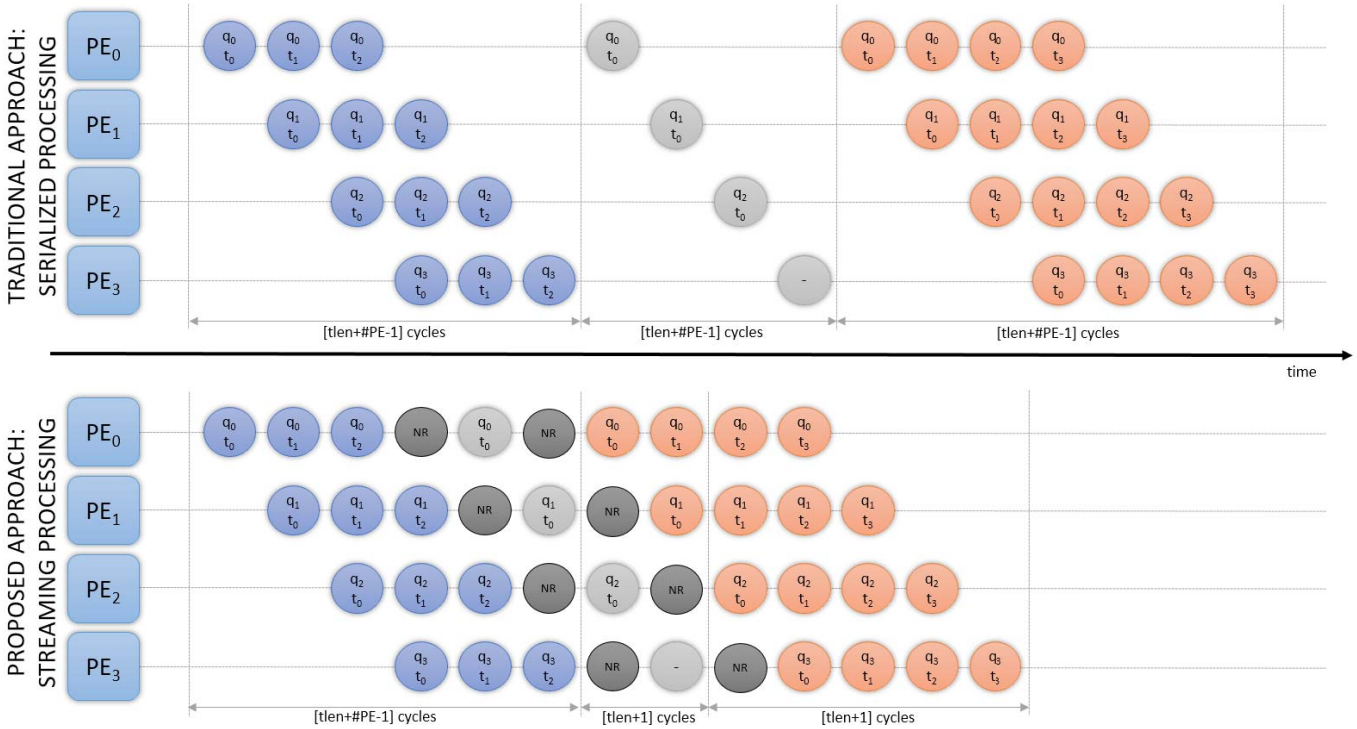

Figure 3. This example illustrates the difference in processing time for Serialized Processing compared to Streaming Processing, with three alignments being performed. A traditional Smith-Waterman systolic array performs serialized processing of the alignments. Each alignment requires [tlen+# of PE-1] cycles. The large amount of white space in the figure is indicative of the fact that large parts of the array are idle during the computation. Streaming processing results in much higher utilization of the systolic array as, except for the first alignment, each alignment only requires [tlen+1] cycles. A New Read token is inserted in-between sequences to signal to a Processing Element that is should proceed onto the next alignment. For ”real” systolic arrays consisting of tens of Processing Elements, the benefits to systolic array utilization are even more pronounced.

Table I
SMITH-WATERMAN KERNEL PERFORMANCE RESULTS

	Design			FPGA Resource Utilization		Performance (GCUPS)		
	Modules	**PEs**	**Frequency**	**Logic**	**RAM Blocks**	**Theoretical**	**Actual**	**Utilization**
BittWare A10PL4	2	131	153 MHz	27%	23%	40.1	40.0	99.8%
BittWare A10PL4	4	131	150 MHz	44%	34%	78.6	78.4	99.8%
BittWare A10PL4	6	131	137 MHz	59%	54%	107.4	107.2	99.8%
Intel A10_REF	2	131	193 MHz	25%	18%	50.5	50.4	99.8%
Intel A10_REF	4	131	188 MHz	41%	40%	98.5	98.3	99.8%
Intel A10_REF	6	131	166 MHz	58%	40%	130.4	130.2	99.8%
Intel A10_REF	8	131	178 MHz	69%	98%	186.5	186.2	99.8%
Intel A10_REF	10	131	164 MHz	91%	98%	214.8	214.4	99.8%
Sirasao[5]	42	32	N/A	N/A	N/A	135.4	77.0	56.9%

V. EXPERIMENTAL SETUP

Results were obtained on a system with an Intel Xeon E5-1650 (six cores, twelve threads @ 3.2 GHz) with 64 GB of RAM. We used a BittWare A10PL4 PCIe board with an Intel Arria 10 GX FPGA (BittWare A10PL4), and the Intel Arria 10 GX FPGA Development Kit Reference Platform board (Intel A10_REF). The only relevant difference between these two boards is that the Intel Reference board uses an Arria 10 GX with higher FPGA fabric speed-grade. Both devices have the same logic density of 1150k logic elements. We used the latest Intel FPGA SDK for OpenCL, which is version 17.0.1 [8]. The results are obtained using a data set of 100'000 pairwise alignment query/target pairs, with a query length of 131 and a target length of 400.

VI. RESULTS

The standardized metric for comparing the performance of Smith-Waterman implementations is by using the GCUPS unit: giga-cell updates per seconds. This number indicates the billions of cell updates that can be performed every second on the Smith-Waterman similarity matrix. The theoretical value can be attained only when all Processing Elements are busy performing useful work. This is not often the case, usually only for very long target sequences. A key innovation of our implementation is that our efficiency is virtually independent of target sequence length. The theoretical maximum GCUPS value is calculated by:

$$\text{Max}_{\text{theo}} = \#\text{ of modules} \times \text{frequency} \times \#\text{ of PEs}$$

The results for the various designs are shown in Table I. We show a number of designs, with increasing number of modules, and for both Arria 10 FPGA boards (A10_REF and A10PL4). The largest 10-module design is able to achieve a theoretical maximum performance of 214 GCUPS. From the results, it is clear that the higher FPGA speed-grade used by the A10_REF board has a significant effect on achievable frequency, improving it by 21-26%. The larger designs show a bit reduced frequency compared to smaller designs, as the FPGA synthesis tool chain has to put more effort into generating a functional design.

We compare our results to the previously highest performing FPGA Smith-Waterman implementation of Sirasao [5]. Sirasao evaluated a variety of designs of which the ratio between number of Processing Elements and number of modules varied. Here, we included only the results for their best performing design, which includes 42 modules of 32 PEs each. Note that the total number of Processing Elements is quite similar to our largest design, with 1344 PEs for Sirasao compared to 1310 for our 10-module design. This 42-module design is able to achieve a theoretical maximum performance of 135 GCUPS. This means that our design has a +58% higher theoretical performance.

However, for traditional Smith-Waterman systolic array designs, the GCUPS value achievable in practice is substantially lower than the theoretical maximum performance. Utilization can be defined as:

$$\text{Utilization} = \frac{(\text{PEs} \times \text{cycles})_{\text{useful}}}{(\text{PEs} \times \text{cycles})_{\text{overall}}}$$

For example, the Sirasao design only achieves 56.9% utilization on their data set, for an actual performance of 77 GCUPS. In contrast, our design achieves almost full utilization, still obtaining 214 GCUPS. Peak performance is only slightly reduced, as for each alignment, one target symbol per alignment is used to indicate a new sequence, thus resulting for our data set with target sequence length 400 in an efficiency of 99.8% (=400/401). In their paper, Sirasao [5] tested with a data set with target length 256 sequences and query length 128, for this the efficiency of our design would be 97.3%. This shows that in practice, our streaming, implicit synchronizing design is almost three times as fast as the fastest previously known implementation.

VII. DISCUSSION

The Streaming architecture significantly improves systolic array utilization. Figure 4 and Figure 5 illustrate how utilization depends on target and query length, respectively.

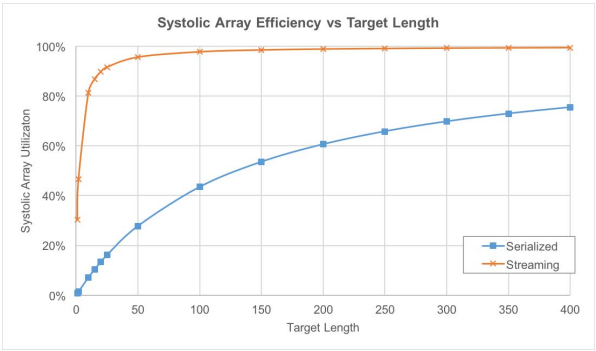

Figure 4. Systolic array efficiency dependence on target length (data set: 100 alignments with query length 131 and variable target length).

Figure 5. Systolic array efficiency dependence on query length (data set: 100 alignments with variable query length and target length 400).

Whereas efficiency of the Serialized systolic array slowly increases with target length, the Streaming systolic array obtains high efficiency almost immediately. For a data set with so few alignments, the disproportionate long cycle time of the first alignment has a large impact on efficiency; a bigger data set would mask this better. The dependence on query length is similar for both systolic array architectures, showing linear dependence. In [7], various solution are proposed to improve utilization in situations that have an imbalance between systolic array size and query length. For example, the Variable Physical Length (VPL)-systolic array contains multiple sized systolic arrays: alignments with shorter query length go to the smaller arrays. Therefore, the improvements proposed here represent the missing link to achieve a high utilization systolic array.

Compared to using normal hardware description languages, using a high-level language such as OpenCL has two main benefits. First, OpenCL is more expressive (our Processing Element kernel code in OpenCL requires 90 lines of code compared to about 450 lines of VHDL). Second, OpenCL development has more convenient testing and debugging capabilities, such as rapid testing using software emulation, and the ability to use printf statements inside kernels. The end result is a much faster development cycle.

VIII. Conclusion

We presented our OpenCL-based FPGA Smith-Waterman implementation that employs two key techniques to greatly improve the utilization of its underlying systolic array architecture. By eliminating centralized control and through the use of Query Buffers, an arbitrary number of alignments can be in flight at the same time, resulting in utilization close to theoretical maximum performance. The techniques presented here are generally applicable to any linear systolic array design, although here we only consider the Smith-Waterman algorithm. Our resulting implementation is both the fastest and most efficient, resulting in a maximum performance of 214 GCUPS and outperforming other Smith-Waterman FPGA implementations almost three-fold.

Compared to typical hardware description languages used for FPGA development, OpenCL simplifies writing code, testing and debugging. The ability to emulate and debug in software allows for a much more agile development cycle, allowing one to test many more different designs.

Acknowledgments

The authors would like to thank the kind people at Intel and OVH, for providing support on all questions regarding the Intel FPGA SDK for OpenCL, and for providing access to their cloud nodes for development and testing.

References

[1] T. Smith and M. Waterman, “Identification of Common Molecular Subsequences,” *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[2] M. Zhao, W. Lee, E. Garrison, and G. Marth, “Ssw library: an simd smith-waterman c/c++ library for use in genomic applications,” *PloS one*, vol. 8, no. 12, p. e82138, 2013.

[3] M. Farrar, “Striped smith–waterman speeds database searches six times over other simd implementations,” *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2006.

[4] Y. Liu, A. Wirawan, and B. Schmidt, “CUDASW++ 3.0: Accelerating Smith-Waterman Protein Database Search by Coupling CPU and GPU SIMD Instructions,” *BMC bioinformatics*, vol. 14, no. 1, p. 117, 2013.

[5] A. Sirasao, E. Delaye, R. Sunkavalli, and S. Neuendorffer, “Fpga based opencl acceleration of genome sequencing software,” *System*, vol. 128, no. 8.7, p. 11, 2015.

[6] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[7] E. Houtgast, V. Sima, K. Bertels, and Z. Al-Ars, “An FPGA-Based Systolic Array to Accelerate the BWA-MEM Genomic Mapping Algorithm,” in *Intl. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation*, 2015.

[8] Intel, “The Intel FPGA SDK for Open Computing Language,” https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html, last visited: 2017-08-24.