Parallel Approaches for SWAMP Sequence Alignment

Shannon Steinfadt and Kevin Schaffer
Department of Computer Science
Kent State University
Kent, USA
ssteinfa, kschaffe@cs.kent.edu

***Abstract*—This document is a summary and overview of several approaches to implement the local sequence alignment algorithms known as SWAMP and SWAMP+ on commercially available hardware. Using a Smith-Waterman style of alignment, these parallel algorithms have several innovative extensions that take advantage of the ASC associative computing model while maintaining speed, accuracy, and producing a richer set of results in an automated way that is not currently available.**

We consider four different hardware architectures for the realization of the ASC model. These are the ClearSpeed CSX processor, NVIDIA GPGPU graphics processors, IBM Cell Processors, and FPGAs.

Keywords-sequence alignment; parallel computing; associative SIMD; GPGPUs; Cell Processor; FPGAs

I. INTRODUCTION

To push the boundaries of current computing and alignment approaches, a library of algorithms for bioinformatics sequence alignment has been developed. The underlying alignment method uses Smith-Waterman for an exhaustive and exact result. This is useful for in-depth studies that require such sensitivity and deeper information, such as finding regulatory regions. These new algorithms are ideal for users to find and customize *subsequent* local alignments between two sequences without the intensive manual manipulation of data and parameters between different runs. Instead, a single run can return multiple results in an automated fashion not available with standard Smith-Waterman implementations.

This group of algorithms, known as SWAMP [1][2] and SWAMP+ [3] are Smith-Waterman using Associative Massive Parallelism. They are computationally intensive, parallel algorithms based on the proven and widely used Smith-Waterman algorithm but with innovative extensions. Designed for the ASC (ASsocative Computing) model [4][5], SWAMP+ takes advantage of the associative paradigm for algorithm development, ideal for development but lacking commercial hardware support.

One goal of this research is to create mainstream application software that runs the SWAMP+ suite of algorithms on commercially available hardware. To do this, ASC's associative functionality has to be efficiently emulated on commercial hardware. This is important since the functionality necessary to run ASC code should not degrade the overall performance to the point of negating the advantages gained through parallelization.

Local sequence comparison algorithms using a Smith-Waterman approach [6] are useful but have high time and space requirements. Parallelism is key to providing a reasonable running time for the high sensitivity method.

The four different hardware architectures considered for the realization of the ASC model are the ClearSpeed CSX processor, NVIDIA general-purpose graphics processing units (GPGPUs), IBM Cell Processors, and field-programmable gate arrays (FPGAs). This paper first outlines the alignment algorithms, then moves on to describe the original ASC parallel paradigm. The different hardware studied for practical implementation is then described.

II. SEQUENCE ALIGNMENT AND SWAMP +

A. Smith-Waterman Local Sequence Alignment

One way to compare two different DNA and protein sequences is with local pairwise sequence alignment. This looks for the underlying structural similarity between the nucleotide or amino acid residues of the two strings to discover gene function, evolutionary relationships, and in the future, even establish an individual's likelihood for a given disease,

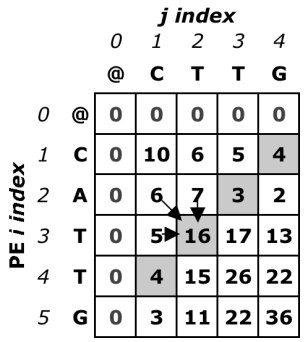

Figure 1. An example of the sequential Smith-Waterman matrix. The dependencies of cell (3, 2) are shown with arrows. While the calculated *C* values for the entire matrix are given, the shaded anti-diagonal (where all *i+j* values are equal) shows one logical parallel step since they can be computed in lockstep simultaneously. Affine gap penalties are used in this example as well as in the parallel code to produce the top scoring alignments.

978-0-7695-3685-9/09 $25.00 © 2009 IEEE
DOI 10.1109/OCCBIO.2009.12

phenotype, trait, or medication resistance.

The Smith-Waterman algorithm [6] is a dynamic programming algorithm that performs local sequence alignment on two stings, S1 and S2, of size *m* and *n*, respectively.

This algorithm allows for insertion and deletions between the two strings, referred to as indels, computing the highest scoring alignment possible. The Smith-Waterman approach tests all possible combinations of indels and alignments, resulting in a computationally and memory intensive algorithm. The dynamic programming approach uses a table or matrix to preserve values and avoid recomputations. Data dependencies exist among the different matrix values since one matrix entry cannot be computed without prior computation of its north, west and northwestern neighbors. This limits the level of parallelism that can be achieved, but nonetheless this useful and important algorithm can be sped up by using a wave front approach, computing all of the data-independent matrix entries in parallel along the anti-diagonal.

The running time of the purely sequential algorithm is O(*m***n*) using O(*m***n*) memory as well.

B. *SWAMP and SWAMP+*

Smith-Waterman using Associative Massive Parallelism, or SWAMP, has been implemented as an associative parallel local sequence alignment algorithm [2] using the ASC programming language, compiler, and simulator. The SWAMP algorithm finds the best local alignment in O(*m*+*n*) time using *m*+1 processing elements versus O(*m***n*) sequential time. The result of this work is that it finds the highest scoring subsequences that may be missed by the heuristic algorithms while retaining a reasonable running time.

SWAMP+ is a suite of innovative extensions [3] based on the underlying SWAMP algorithm. It will find the single top scoring local alignment in parallel, providing the same output as SWAMP or even the Smith-Waterman algorithm. SWAMP+ differs in that it will load the sequence data into local parallel memory and make multiple passes to seek out the top scoring non-overlapping, non-intersecting local alignments as shown in Figure 2a. Fully user controlled, the number and quality of the subsequent alignments discovered is parameterized. Users can also find multiple instances of a single subsequent as shown in Figure 2b and c. This output is somewhat similar to SIM [7] and LALIGN [8] in the FASTA package [9], and to some degree even BLAST since multiple regions of similarity are returned, but SWAMP+ does not use heuristics, is parallel, and allows for different flavors of subsequence local alignments. The capabilities of SWAMP+ do not currently exist in software that the authors are aware of, and allowing for *multiple* local alignments is not currently a process that can be automated through the use of parameters.

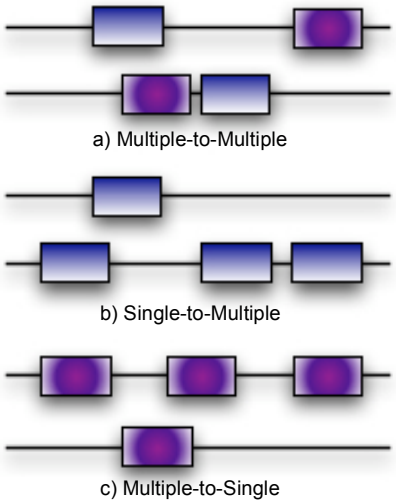

Figure 2. SWAMP+ variations where *k* is the number of local non-overlapping, non-intersecting alignments sought. In a) *k=2* and *k*=3 in b) and c).

III. ASSOCIATIVE COMPUTING MODEL (ASC)

Developed within the Department of Computer Science at Kent State University, ASC is an algorithmic model for associative computing [4][5]. The ASC model grew out of work on the STARAN and MPP, associative processors built by Goodyear Aerospace. Figure 3 shows a conceptual model of an ASC computer. There is a single control unit, also known as an instruction stream (IS), and multiple processing elements (PEs), each with its own local memory. The control unit and PE array are connected through a broadcast/reduction network and the PEs are connected together through a PE interconnection network.

ASC is a SIMD model of computation. The control unit fetches and decodes program instructions and broadcasts control signals to the PEs. The PEs, under the direction of the control unit, execute these instructions using their own local data. All PEs execute instructions in a lockstep manner, with an implicit synchronization between every instruction. SIMD computers are particularly well suited for fine-grain data parallel programming, where the same operation is applied to each element of a data set.

An ASC processor is a SIMD computer that has additional hardware, in the form of a broadcast/reduction network, to support the ASC model's specific high-speed global operations: associative search, maximum/minimum search, and responder selection/detection.

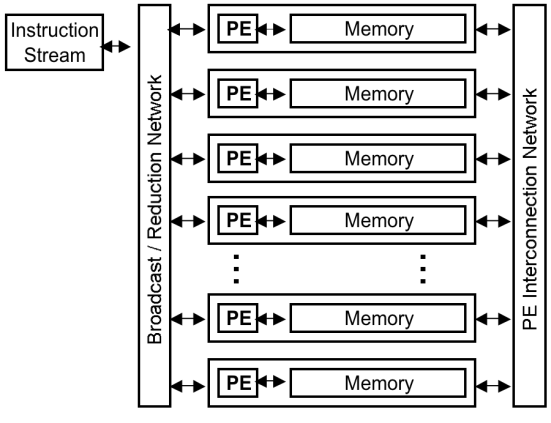

Figure 3. A high-level view of the ASC model of parallel computation.

A. Associative Search

The basic operation in an associative algorithm is the associative search. An associative search simultaneously locates all the PEs whose local data matches a given search key. Those PEs that have matching data are called *responders* and those with non-matching data are called *non-responders*. After performing a search, the algorithm can then restrict further processing to only affect the responders by disabling the non-responders (or vice versa). Performing additional searches may further refine the set of responders. Associative search is heavily utilized by SWAMP+ in selecting which PEs are active for each parallel step within every diagonal that is processed in tandem.

B. Maximum/Minimum Search

In addition to simple searches, where each PE compares its local data against a search key using a standard comparison operator (equal, less than, etc.), an associative computer can also perform global searches, where data from the entire PE array is combined together to determine the set of responders. The most common type of global search is the maximum/minimum search, where the responders are those PEs whose data is the maximum or minimum value across the entire PE array. The maximum value is used by SWAMP+ in every diagonal to track the highest value calculated so far. Use of the maximum search occurs frequently, once in logical parallel step, $m+n$ times per alignment.

C. Responder Selection/Detection

An associative search can result in multiple responders and an associative algorithm can process those responders in one of three different modes: parallel, sequential, or single selection. Parallel responder processing performs the same set of operations on each responder simultaneously. Sequential responder processing selects each responder individually, allowing a different set of operations for each responder. Single responder selection (also known as *pick one*) selects one, arbitrarily chosen, responder to undergo processing.

In addition to multiple responders, it is also possible for an associative search to result in no responders. To handle this case, the ASC model requires that an associative algorithm be able to detect whether there were any responders to a search and perform a separate set of actions in that case.

In SWAMP+, multiple responders that contain characters to be aligned are selected and processed in parallel, based on the associative searches mentioned above in Section A. Single responder selection occurs when and if there are multiple values that have the exact same maximum value when using the maximum/minimum search.

D. PE Interconnection Network

Most associative processors include some type of PE interconnection network to allow parallel data movement within the array. The ASC model itself does not specify any particular interconnection network and, in fact, many useful associative algorithms do not require one. Typically associative processors implement simple networks such as 1D linear arrays or 2D meshes. These networks are simple to implement and allow data to be transferred quickly in a synchronous manner. The 1D linear array is sufficient and ideal for the explicit communication between PEs in the SWAMP+ algorithms.

IV. Hardware Realizations for ASC

The following section describes hardware that has been evaluated for implementing the SWAMP+ application. Since there is no commercial associative hardware currently available, the necessary associative functionality described in the previous section must be emulated on the hardware for any associative algorithm to be successfully adopted. A brief description of the four parallel architectures used, the ClearSpeed CSX processor, NVIDIA general-purpose graphics processing units (GPGPUs), IBM Cell Processors, and field-programmable gate arrays (FPGAs), is given below. The mappings of the associative functionality specific to the CSX and GPGPUs are discussed. The Cell processor and FPGAs represent possible future directions for the work. At this time, not much research has been completed on mapping ASC and SWAMP+ to these architectures, so their treatment is not as in-depth.

A. ClearSpeed

1) CSX Architecture

The ClearSpeed CSX family of processors are SIMD coprocessors designed to accelerate data-parallel portions of application code [10]. ClearSpeed sells a board with two CSX processors, each with 96 PEs, which connects to a host computer through a PCI or PCI-X interface. Multiple boards can be connected to the same host in order to scale up the level of parallelism, as necessary for the application.

The CSX processor, which is based on ClearSpeed's *multithreaded array processor* (MTAP) SIMD architecture, consists of two main components: a control unit (called the *mono execution unit*) and an array of PEs (called the *poly execution unit*). These are shown are Figure 4. The control unit fetches and decodes instructions and, if the instruction is a control flow or scalar instruction, executes it as well. The remaining instructions are broadcast to the PE array for execution.

2) Implementing ASC on CSX

As ASC is based on the general SIMD model, mapping ASC to the CSX processor is a relatively straightforward process. The CSX already has hardware to broadcast instructions and data to the PEs, to enable and disable PEs, and to detect whether any PEs are currently enabled, fulfilling many of the ASC model's requirements. However, the CSX processor does not have direct support for computing a global maximum/minimum or selecting a single PE from multiple responders.

The CSX processor does have the ability to reduce a parallel value to a scalar using logical AND or OR. With this capability it is possible to use Falkoff's algorithm to implement maximum/minimum search. Falkoff's algorithm locates a maximum value by processing the values in bit-serial fashion, computing the logical OR of each parallel bitslice, and eliminating from consideration those values whose bit does not

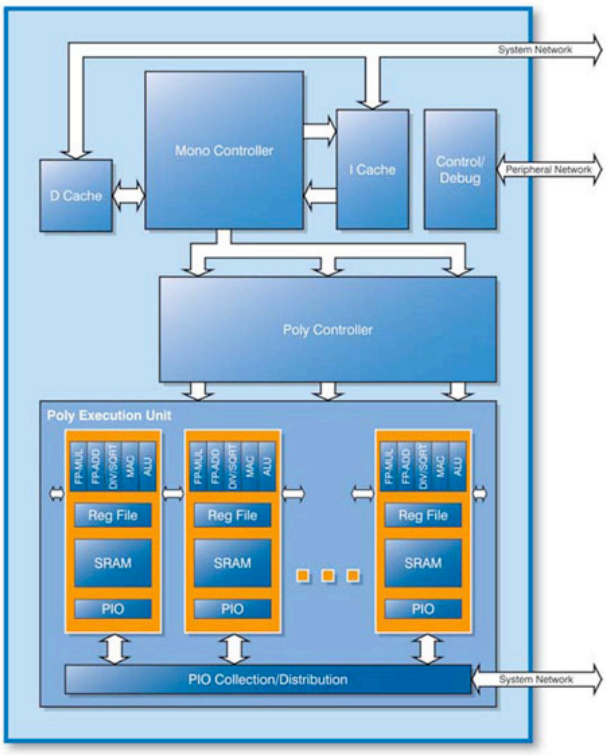

Figure 4. ClearSpeed CSX processor organization. Diagram courtesy of ClearSpeed http://www.clearspeed.com/products/csx700/.

match the sum. The algorithm is easily adapted to compute a minimum by first inverting all the value bits.

The *pick one* operation, which selects a single PE when there are multiple responders, can be implemented on the CSX processor by using the maximum/minimum operators. Each PE has a unique index associated with it and searching for the PE with the maximum or minimum index will effectively select a single PE.

With those two operators emulated in software, the CSX processor is able to fully implement the ASC model. Any ASC algorithm, including SWAMP+, can be readily adapted to the architecture.

B. GPGPUs

1) GPU Architecture

Another hardware platform to map the ASC model to is on graphics cards. Graphics cards have been used for years not only for the graphics pipeline to create and output graphics, but for other types of general-purpose computation, including sequence alignment. The advent of higher and higher powered graphics cards that contain their own processing units, known as graphics processing units or GPUs, has led to many scientific applications being offloaded to GPUs. The use of graphics hardware for non-graphics applications has been dubbed General-Purpose computation on Graphics Processing Units or GPGPU.

The graphics card manufacturer NVIDIA released the Compute Unified Device Architecture (CUDA). It provides three key abstractions that provide a clear parallel structure to conventional C code for one thread of the hierarchy [11].

CUDA is a computing architecture, but also consists of an application programming interface (API) and a software development kit (SDK). CUDA provides both a low level API and a higher level API.

The introduction of CUDA allowed for a real break from the graphics pipeline, allowing multithreaded applications to be developed without the need for stream computing. It also removed the difficult mapping of general-purpose programs to parts of the graphics pipeline. The conceptual decoupling allowed GPU programmers to no longer have values referred to as "textures" or to specifically use rasterization hardware. It also allows a level of freedom and abstraction from the hardware. One drawback with the relatively young CUDA SDK (initial release in early 2007) is that the abstraction and optimization of code is not as fully decoupled from the hardware as one might want. This causes optimization problems that can be difficult to detect and correct.

The GPGPUs have multiple levels of parallelism and rely on massive multithreading. Each thread has its own local memory, used to express fine-grained parallelism. Threads are organized in blocks that communicate through shared memory and are used for coarse-grained (cluster-like) parallelism [12]. Every thread is stored within a streaming processor (SP), and every SP can handle 128 threads. Eight SPs are contained within each streaming multiprocessor (SM), shown in Figure 5. While the number of SMs is scalable across the different types and generations of NVIDIA graphics cards, the underlying SM layout remains the same. This scalability is ideal as graphics cards change and are updated.

The specific compute-heavy GPGPU card with no graphics output is known as the Tesla series. The Tesla T10 has 240 SP processors that each handle 128 threads. This means that there could be a maximum of 30,720 lightweight threads processed in parallel at one time [12]. Another CUDA-enabled card may have only 128 SPs, but it can run the same CUDA code, only slower due to less parallelism.

Their overall organization is a single program (kernel), multiple data or SPMD model of computing, the same classification as MPI-based cluster computing.

Figure 5. A detail of one streaming multiprocessor (SM) is shown here. On CUDA-enabled NVIDIA hardware, a varied number of SMs exist for massively parallel processing. Each SM contains eight streaming processor (SP) cores, two special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory. One example organization is the NVIDIA Tesla T10 with 30 SMs for a total of 240 SPs..

2) Implementing ASC on GPGPUs

To map the ASC model onto CUDA, every PE would be mapped to a single thread. Due to the communication between PEs and the lockstep data movement common to SIMD and associative SIMD algorithms, communication between threads is necessary. This means that the threads need to be contained within the same logical thread block structure to emulate the PE Interconnection Network. Explicit synchronization and deadlock prevention is a necessary and difficult task for the programmer.

A second factor that limits an ASC algorithm to a single block is due to the independence requirement between blocks, where blocks can be run in any order. A thread block is limited in size to 512 threads, prematurely cutting short the level of parallelism that can be achieved on a GPGPU, effectively removing any power of scalability.

This initial mapping of the ASC functions to CUDA has been completed but a fully implemented set of associative functions is still under development. This is a more difficult mapping than ASC to the ClearSpeed CSX chip with the multiple layers of hierarchy and multithreading involved. Also, the onus of explicit synchronization is on the programmer to manage.

Regardless of the difficulties, a successful and efficient mapping of the associative functions onto the NVIDIA GPGPU hardware would be ideal. GPUs are very affordable and massively parallel. The hardware has a low cost with many current computers and laptops containing CUDA-enabled graphics cards already, and the software tools are free. This could make the SWAMP+ suite available to millions with no additional hardware necessary. While a CUDA implementation for the Smith-Waterman algorithm is described in [13], SWAMP+ differs greatly from the basic Smith-Waterman algorithm and is not really comparable to [13].

C. Cell

Developed by IBM and used in Sony's PlayStation 3 game console, the Cell Broadband Engine is a hybrid architecture that consists of a general-purpose PowerPC processor and an array of eight synergistic processing elements (SPEs) connected together through an element interconnect bus (EIB). Cell processors are widely used, not only in gaming, but as part of computation nodes in clusters and large-scale systems, such as the Roadrunner hybrid-architecture supercomputer developed by Los Alamos National Lab and IBM [14] and listed as the number one fastest computer, as listed on Top500.org, November 2008. The Cell has been used for several other bioinformatics algorithms, including sequence alignment [15], that were successfully adapted to this architecture.

D. FPGAs

A field-programmable gate array (FPGA) is a fabric of logic elements, each with a small amount of combinational logic and a register that can be used to implement everything from simple circuits to complete microprocessors. While generally slower than traditional microprocessors, FPGAs are able to exploit a high degree of fine-grained parallelism.

FPGAs can be used to implement SWAMP+ in one of two ways: pure custom logic or softcore processors. With custom logic, the algorithm would be implemented directly at the hardware level using a hardware description language (HDL) such as Verilog or VHDL. This approach would result in the highest performance as it takes full advantage of the parallelism of the hardware. Other sequence alignment algorithms have been successfully implemented on FPGAs using custom logic and shown significant performance gains [16][17]. However, a pure custom logic solution is much more difficult to design than software and tends to be highly dependent on the particular FPGA architecture used.

An alternative to pure custom logic is a hybrid approach using softcore processors. A softcore processor is a processor implemented entirely within the FPGA fabric. Softcore processors can be programmed just like ordinary (hardcore) processors, but they can be customized with application-specific instructions. These special instructions are then implemented with custom logic that can take advantage of the highly parallel FPGA hardware. Two companies, Mitrionics and Convey, are currently using FPGAs in this capacity. Mitrionics has bioinformatics applications as one its focuses, allowing users to download preconfigured solutions specific to sequence alignment.

V. CONCLUSIONS

The novel approach used by the SWAMP+ suite of algorithms can be applied in many ways, from finding regulatory regions to other in-depth studies between homologous sequences. The main factor barring SWAMP+ from becoming mainstream is the lack of commodity hardware supporting the ASC model upon which the algorithms are based.

As such, several hardware architectures have been considered and the associative functionality established. These platforms for the ASC model include ClearSpeed CSX, NVIDIA's CUDA, IBM Cell Broadband Engine, and FPGAs.

The more SIMD-like the underlying architecture, the easier it has been to realize the associative functionality of the ASC model on these various systems. As such, the simplest and most elegant mapping for ASC is the ClearSpeed CSX. For high availability, a CUDA mapping is the most beneficial. More information and a mailing list for the SWAMP+ algorithm suite is available at http://www.swampalign.com.

REFERENCES

[1] S. Steinfadt, M. Scherger, and J.W. Baker, "A Local Sequence Alignment Algorithm Using an Associative Model of Parallel Computation," *Proc. IASTED Computational and Systems Biology* (CASB 2006), Nov. 13-14, 2006, pp. 38-43.

[2] S. Steinfadt and J.W. Baker, "SWAMP: Smith-Waterman using Associative Massive Parallelism," *Proc. Int'l Workshop Parallel and Distributed Scientific and Eng. Computing* (PDSEC 2008), Apr. 19, 2008.

[3] S. Steinfadt and J.W. Baker, "SWAMP+: Variations for Sequence Alignment Using Associative Massive Parallelism," submitted for review.

[4] J.L. Potter, et al., "ASC: An Associative Computing Paradigm," *IEEE Computer*, vol. 27, no. 11, Nov. 1994, pp. 19-25.

[5] J.L. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers,* Plenum Publishing, 1992.

[6] O. Gotoh, "An Improved Algorithm for Matching Biological Sequences," *J. Molecular Biology*, vol. 162, 1982, pp. 705-708.

[7] X. Huang and W. Miller, "A Time-Efficient, Linear-Space Local Similarity Algorithm," *Advances in Applied Mathematics*, vol. 12, 1991, pp. 337-357.

[8] W. Pearson and D. Lipman, "Improved tools for biological sequence comparison," *Proc. Nat'l Academy of Sciences USA*, vol. 85, no. 8, 1988, pp. 2444–2448.

[9] W. Pearson, "FASTA Sequence Comparison Suite," Jan. 2008; http://fasta.bioch.virginia.edu/fasta_www2/.

[10] ClearSpeed Technology, "Products Overview," Feb. 2009; http://www.clearspeed.com/products/overview.

[11] J.I. Nickolls et al., "Scalable parallel programming with CUDA," *Queue*, vol. 6, 2008, pp. 40-53.

[12] M. Fatica, "High Performance Computing with CUDA: Introduction" tutorial slides presented at *ACM/IEEE Conf. Supercomputing* (SC 2008), Nov. 2008.

[13] S. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, vol. 9, suppl. 2, 2008, p. S10.

[14] K.J. Barker et al., "Entering the petaflop era: The architecture and performance of roadrunner," *Proc. ACM/IEEE Conf. Supercomputing* (SC 2008), 2008, pp. 1-11.

[15] M. Farrar, "Optimizing Smith-Waterman for the Cell Broadband Engine," Mar. 2009; http://farrar.michael.googlepages.com/SW-CellBE.pdf.

[16] T. Oliver, B. Schmidt and D. Maskell, "Hyper customized processors for bio-sequence database scanning on FPGAs," *Proc. ACM/SIGDA 13th Int'l Symp. Field-Programmable Gate Arrays* (FPGA 2005), 2005, pp. 229-237.

[17] I.T. Li, W. Shum and K. Truong, "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)," *BMC Bioinformatics*, vol. 8, 2007, p. 185.