High-performance Dynamic Programming on FPGAs with OpenCL

Sean O. Settle
Altera Corporation
San Jose, CA 95134
sean.settle@altera.com

***Abstract*—Field programmable gate arrays (FPGAs) provide reconfigurable computing fabrics that can be tailored to a wide range of time and power sensitive applications. Traditionally, programming FPGAs required an expertise in complex hardware description languages (HDLs) or proprietary high-level synthesis (HLS) tools. Recently, Altera released the worlds first OpenCL conformant SDK for FPGAs. OpenCL is an open, royalty-free standard for cross-platform, parallel programming of heterogeneous systems that together with Altera extensions significantly reduces FPGA development time and costs in high-performance computing environments. In this paper, we demonstrate dynamic programming on FPGAs with OpenCL by implementing the Smith Waterman algorithm for DNA, RNA, or protein sequencing in bioinformatics in a manner readily familiar to both hardware and software developers. Results show that Altera FPGAs significantly outperform leading CPU and GPU parallel implementations by over an order of magnitude in both absolute performance and relative power efficiency.**

I. Introduction

Modern high-performance computing (HPC) systems increasingly contain a variety of multi- and many-core processor architectures, but harnessing their full processing potential has proven difficult in terms of the hardware and software expertise, and the development time required. While several proprietary standards and tools have been developed to address some of these difficulties for a limited subset of processor architectures, OpenCL is the first and most popular open royalty-free standard for general purpose parallel programming heterogeneous systems including CPUs, GPUs, FPGAs, and other accelerators and custom devices. Thanks to OpenCL, it is now possible to do reconfigurable computing with FPGAs entirely within a software development framework in a fraction of the time of a traditional hardware development cycle. In this paper, we present a case study on the acceleration of the Smith Waterman algorithm using FPGAs with OpenCL.

A. Dynamic Programming

As scientists and engineers encounter ever-increasingly complex problems, they naturally look to solve their problems by breaking them down into simpler subproblems. Dynamic programming, like divide-and-conquer, is a method for solving complex problems by exploiting an optimal substructure, often represented by a recursive relationship. However, dynamic programming differs from divide-and-conquer in that it has overlapping subproblems, which increases the fine grain communication overhead in the parallel computer architectures scientists and engineers employ to solve their problems. A common example of dynamic programming is the Smith Waterman algorithm from the field of bioinformatics for finding the optimal local alignment of two DNA, RNA, or protein sequences [1], [2].

Given a database sequence $\mathbf{a} = a_1 a_2 \dots a_m$ and a query sequence $\mathbf{b} = b_1 b_2 \dots b_n$ such that $m \geq n$, the forward pass of the Smith Waterman algorithm generates an $(m+1)\times(n+1)$ scoring matrix $\mathbf{S}$ whose non-negative elements $S_{i,j}$ quantify (higher being better) the alignment between $a_1 a_2 \dots a_i$ and $b_1 b_2 \dots b_j$. To begin, define a similarity function

$$\sigma(a_i, b_j) = \begin{cases} match, & a_i = b_j, \\ -mismatch, & a_i \neq b_j, \end{cases}$$

and non-negative affine gap function parameters for opening and extending both horizontal and vertical gaps, respectively: ogh, ogv, egh, and egv such that $ogh + egh \geq match + mismatch$ and $ogv + egv \geq match + mismatch$. Next, set $S_{i,0} = 0$ for $0 \leq i \leq m$ and $S_{0,j} = 0$ for $0 \leq j \leq n$. Similarly, set $S_{Hi,0} = -ogh$ for $1 \leq i \leq m$ and $S_{V0,j} = -ogv$ for $1 \leq j \leq n$. Then for $1 \leq i \leq m$ and $1 \leq j \leq n$, compute the scoring matrix through the following bottom-up recursion relationship:

$$S_{i,j} = \max \left\{ \begin{array}{c} 0, \\ S_{i-1,j-1} + \sigma(a_i, b_j), \\ S_{i,j-1} - ogh, \\ S_{i-1,j} - ogv, \\ S_{Hi,j-1} - egh, \\ S_{Vi-1,j} - egv \end{array} \right\}, \tag{1}$$

where

$$S_{H_{i,j}} = \max \left\{ \begin{array}{l} S_{i,j-1} - ogh, \\ S_{Hi,j-1} - egh \end{array} \right\},$$

and

$$S_{V_{i,j}} = \max \left\{ \begin{array}{l} S_{i-1,j} - ogv, \\ S_{Vi-1,j} - egv \end{array} \right\}.$$

Finally, the backward pass of the Smith Waterman algorithm starts from the maximum element of $\mathbf{S}$ and retraces the path of elements that lead to the maximum element, stopping with an element equal to zero, see Fig. 1 for an example finding the optimal local alignment of two RNA sequences. Note that the elements of an anti-diagonal are dependent on elements of

978-1-4799-1365-7/13/$31.00 ©2013 IEEE

	Δ	C	A	G	C	C	U	C	G	C
Δ	0	0	0	0	0	0	0	0	0	0
A	0	0	3	0	0	0	0	0	0	0
A	0	0	3	2	0	0	0	0	0	0
U	0	0	0	2	1	0	3	0	0	0
G	0	0	0	3	1	0	0	2	3	0
C	0	3	0	0	6	4	1	3	1	6
C	0	3	2	0	3	9	5	4	3	4
A	0	0	6	2	1	5	8	4	3	2
U	0	0	2	5	1	4	8	7	3	2
U	0	0	1	1	4	3	7	7	6	2
G	0	0	0	4	0	3	3	6	10	6
A	0	0	3	0	3	1	2	2	6	9

Fig. 1. Smith Waterman algorithm results for RNA sequences **a** = AAUGCCAUUGA and **b** = CAGCCUCGC with parameters: $match = 3$, $mismatch = 1$, $ogh = 3$, $ogv = 3$, $egh = 1$, and $egv = 1$. The optimal local alignment obtained is GCCAUUG and GCC–UCG.

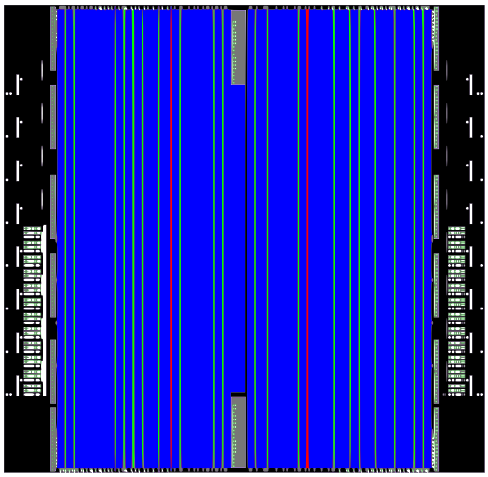

Fig. 2. A logical view of an Altera Stratix V A7 FPGA: logic array blocks (blue), embedded memory blocks (green), and DSP blocks (red)

the preceding anti-diagonal, but are mutually independent of one another within the same anti-diagonal.

B. FPGAs

FPGAs are reconfigurable integrated circuits consisting of programmable routing networks linking together logic array blocks, embedded memory blocks, and digital signal processor (DSP) blocks, see Fig. 2. In contrast to the fixed datapaths and topologies found in CPUs and GPUs that process program instructions, FPGA resources may be configured and linked together to create custom instruction pipelines through which data is processed. Dynamically creating custom pipelines to process each target application increases throughput performance and power efficiency by reducing the amount of superfluous functional units in silicon.

Traditionally, hardware developers design and verify digital circuits on FPGAs at the register-transfer level (RTL) using hardware description languages (HDLs) like Verilog and VHDL. Unfortunately, HDLs are verbose and error prone low-level languages akin to assembly language but with an extra complexity that comes with an explicit notion of time. Just as software developers gained from transitioning from assembly to C/C++ and other higher-level languages, hardware developers want the ease, portability, and productivity that comes with a standard and open higher level of abstraction.

C. OpenCL and Altera Vendor Extensions

OpenCL is an open, royalty-free standard for cross-platform parallel programming of heterogeneous systems. Therefore, unlike other proprietary HLS tools available for FPGAs, OpenCL is both vendor and architecture agnostic. The Altera SDK for OpenCL comforms to the OpenCL 1.0 specification for embedded profiles and supports several features in the OpenCL 1.1 and 1.2 specifications [3]–[5]. The OpenCL embedded profile is a subset of the full profile, but with more flexible requirements and more optional features, such as optional support for online kernel compilation.

The OpenCL specification clearly defines a platform model, memory model, execution model, and programming model, while permitting Khronos, cross-vendor, and vendor-specific extensions thereof. These models are exposed to developers through the OpenCL platform and runtime API and the OpenCL C programming language. Each vendor is allowed significant freedom to implement their platform and runtime internals as long as their implementations abide by the behavior mandated in the OpenCL specification.

In the OpenCL platform model, Altera FPGAs are dedicated OpenCL accelerators (CL_DEVICE_TYPE_ACCELERATOR) that contain a memory hierarchy, see Table I, following a relaxed consistency memory model and communicate with the host processor using a peripheral interconnect such as PCIe. Each Altera FPGA can have multiple in-order command queues associated with it that can execute independent enqueue commands concurrently. As part of the embedded profile, kernels are compiled offline with the Altera OpenCL compiler, e.g., ”aoc smith_waterman.cl -o smith_waterman.aocx.” The previous command compiles every kernel in the source file smith_waterman.cl for the default target board and saves the output in the binary file smith_waterman.aocx, the content of which is then passed to clCreateProgramWithBinary at runtime to create an OpenCL program object. Like most other C/C++ compilers, the ”-D” compiler option may be used to provide an easy and convenient means to pass compile-time literal constants, such as required work-group sizes and the loop unroll factors in our kernels to instantiate a specific number of Smith Waterman processing elements.

An important Altera implementation detail of the OpenCL execution model that we take advantage of is the work-item ordering within a pipeline. The OpenCL specification defines as part of the programming model that a work-item executes an instance of a kernel as part of a work-group that makes up an NDRange. However, the execution model doesn’t specify in what order work-items must start executing. In the Altera implementation, work-groups and work-items in an NDRange

TABLE I
OPENCL MEMORY MODEL FOR FPGAS

OpenCL Memory	FPGA Memory
global	external (e.g., DDR3-1600)
constant	cache
local	embedded (e.g., M20K)
private	registers (e.g., MLAB)

```
__attribute__ ((reqd_work_group_size(X, 1, 1)))
__kernel void worker(__global const int *input,
                     __global int *output) {
    size_t i = get_global_id(0);
    output[i] = input[i];
}
```

Fig. 3. A producer-consumer model in OpenCL

spawn such that get_group_id(dim) and get_local_id(dim) iterate faster than get_group_id(dim + 1) and get_local_id(dim + 1), respectively. Therefore in a one-dimensional NDRange, work-items spawn sequentially from 0 to get_global_size(0) - 1. Furthermore, barring id-dependent branching, work-items maintain their spawning order throughout kernel execution [6], [7].

Altera's channels extension (cl_altera_channels) provides an easy and efficient mechanism to pass built-in data types between work-items in the same kernel, work-items in different kernels, and work-items in kernels and I/O interfaces. Altera's channels are first-in, first-out (FIFO) buffers defined with a channel ID and buffer depth, see Fig. 3 and 4 for a comparison of a simple producer-consumer model in OpenCL without and with the Altera channels extension, respectively. Furthermore, a work-item will block if it attempts to write to a full channel or read from an empty channel, and thus channels may also be used as synchronization points between two work-items. This is important because the execution model in the OpenCL specification only requires two synchronization points: at explicit barriers for all work-items within a work-group while a kernel is running, and at an implied barrier upon kernel completion for all work-items in an NDRange.

II. IMPLEMENTATION

Our FPGA implementation is based roughly on the Novo-G design by George, Lam and Stitt [8]. The DNA and RNA sequence elements in our implementation are stored one element per byte. The four least significant bits store the four nucleic acids and every wildcard permutation of adenine (A), cytosine (C), guanine (G), and thymine (T) or uracil (U), and the four most significant bits store control signals. The control signals allow our implementation to efficiently stream database sequences and query sequences through our unidirectional linear systolic array via channels.

Next, note that the forward and backward pass of the Smith Waterman algorithm are $\mathcal{O}(mn)$ and $\mathcal{O}(n)$, respectively, where typically m is on the order of millions or billions and n is on the order of thousands or less. Therefore we decided

```
#pragma OPENCL EXTENSION cl_altera_channels : enable

channel int cid[2] __attribute__ ((depth(0)));

__attribute__ ((reqd_work_group_size(X, 1, 1)))
__kernel void producer(__global const int *input) {
    size_t i = get_global_id(0);
    write_channel_altera(cid[0], input[i]);
}

__attribute__ ((autorun))
__attribute__ ((reqd_work_group_size(X, 1, 1)))
__kernel void worker(void) {
    int tmp = read_channel_altera(cid[0]);
    write_channel_altera(cid[1], tmp);
}

__attribute__ ((reqd_work_group_size(X, 1, 1)))
__kernel void consumer(__global int *output) {
    size_t i = get_global_id(0);
    output[i] = read_channel_altera(cid[1]);
}
```

Fig. 4. A producer-consumer model in OpenCL using the Altera channels extensions

to offload the forward pass to the FPGA using OpenCL.

Furthermore, since the target application should continuously align query sequences to database sequences, we decided to take advantage of the asynchronous execution between the host and OpenCL device by carving out the optimal alignment database sequence region beginning at $i = O_{max} - A_{max}$ and ending at $i = O_{max}$, which is the row containing the maximum alignment score S_{max}. We call O_{max} and A_{max} the offset and alignment indices, respectively, associated with S_{max}. The benefit to this implementation is that we can calculate the maximum alignment score on the FPGA significantly faster by not storing a prohibitively large scoring matrix to global memory. We also then need only to transfer several bytes back to the host over PCIe instead of anywhere between megabytes to terabytes of data depending on the length of the database and query sequences. Once we obtain the carved out database sequence region from the FPGA, the host can perform the Smith Waterman algorithm on a minuscule subsequence of the full database sequence while the FPGA carves out another alignment region for a new database or query sequence, effectively increasing the systems overall throughput with an efficient pipeline.

Similar to calculating a scoring matrix to obtain S_{max}, we must calculate an alignment matrix to obtain A_{max}. To begin, set $A_{i,0} = 0$ for $0 \leq i \leq m$ and $A_{0,j} = 0$ for $0 \leq j \leq n$. Next, set $A_{Hi,0} = 0$ for $1 \leq i \leq m$ and $A_{V0,j} = 0$ for $1 \leq j \leq n$. Then for $1 \leq i \leq m$ and $1 \leq j \leq n$.

$$A_{i,j} = \begin{cases} 0, & S_{i,j} = 0, \\ A_{i-1,j-1} + 1, & S_{i,j} = S_{i-1,j-1} + \sigma(a_i, b_j), \\ A_{i,j-1}, & S_{i,j} = S_{i,j-1} - ogh, \\ A_{i-1,j} + 1, & S_{i,j} = S_{i-1,j} - ogv, \\ A_{Hi,j-1}, & S_{i,j} = S_{Hi,j-1} - egh, \\ A_{Vi-1,j} + 1, & S_{i,j} = S_{Vi-1,j} - egv, \end{cases} \quad (2)$$

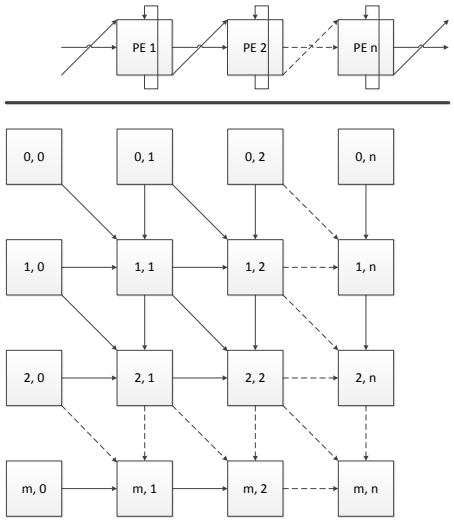

Fig. 5. A unidirectional linear systolic array of Smith Waterman processing elements and dependency graph for calculating the scoring and alignment matrices

```
__attribute__ ((reqd_work_group_size(1, 1, 1)))
__kernel void smith_waterman(int m, ...) {
    ...
    for (size_t i = 0; i < m; ++i) {
        int Sd_private[N];
        int Sh_private[N];
        int Sv_private[N];
        ...
        #pragma unroll N
        for (size_t j = 0; j < N; ++j) {
            int Sd = Sd_private[j];
            int Sh = Sh_private[j];
            int Sv = Sv_private[j];
            ...
            smith_waterman_pe(...);
            ...
            Sd_private[j + 1] = S;
            Sh_private[j + 1] = S;
            Sv_private[j] = S;
        }
    }
}
```

Fig. 6. Pseudocode for a task-based Smith Waterman kernel

where

$$A_{H\,i,j} = \begin{cases} A_{i,j-1}, & S_{i,j-1} - ogh \geq S_{H\,i,j-1} - egh, \\ A_{H\,i,j-1}, & S_{i,j-1} - ogh < S_{H\,i,j-1} - egh, \end{cases}$$

and

$$A_{V\,i,j} = \begin{cases} A_{i-1,j} + 1, & S_{i-1,j} - ogv \geq S_{V\,i-1,j} - egv, \\ A_{V\,i-1,j} + 1, & S_{i-1,j} - ogv < S_{V\,i-1,j} - egv. \end{cases}$$

We processed the elements of the scoring and alignment matrices together using a unidirectional linear systolic array of Smith Waterman processing elements, see Fig. 5.

A. Task

A task kernel executes a single work-group containing a single work-item. We implemented our task kernel much like a standard single-threaded host implementation because a single work-item doesn't require any inter work-item communication or synchronization, see Fig. 6. Each iteration of the outer loop parses a database sequence element and streams it through the unidirectional linear systolic array of Smith Waterman processing elements created by fully unrolling the inner loop. We used private memory to communicate between adjacent diagonal, horizontal, vertical Smith Waterman cells. The major benefit to using a task kernel is that the Altera OpenCL compiler automatically pipelines the outer loop of our kernel based upon the dependency analysis it performs.

B. NDRange

An NDRange kernel executes one or more work-groups containing one or more work-items. We implemented our NDRange kernel by slightly refactoring our task kernel to use a one-dimensional NDRange and the Altera channels extension, see Fig. 7. Analogous to our task kernel, each work-item in the one-dimensional NDRange parses a database sequence element and streams it through the unidirectional linear systolic array of Smith Waterman processing elements created by fully unrolling the main loop. Just like in our task kernel, we used private memory to communicate between adjacent horizontal Smith Waterman cells. However, communicating between adjacent diagonal and vertical Smith Waterman cells requires communicating between work-items, possibly in different work-items, during kernel execution, which prohibits us from using local memory and explicit barriers. Therefore to efficiently communicate between adjacent diagonal and vertical cells without going to global memory, we used one depth channels to communicate between adjacent diagonal cells (zero depth channels would result in deadlocks) and zero depth channels to communicate between adjacent vertical cells.

III. Experiment

We performed the following experiments using the our task kernel executing on a Nallatech PCIe-385n board (populated with D-Link 1000Base-TX SFP modules) in an HP Z620 workstation running Windows 7 Professional 64-bit with an internal build of Altera's Quartus II 13.1 with the Altera SDK for OpenCL [9]. Our Nallatech board contains an Altera Stratix V FPGA (part number 5SGXMA7H2F35C2) and the entire board fits in a 25 watt power envelope.

We used the $m = 1259197$ elements of a megavirus's complete genome (NC_016072.1) as our database sequence and the first n elements of a mamavirus's complete genome (JF801956.1) as our query sequence, and set $match = 2$, $mismatch = 1$, $ogh = 2$, $ogv = 2$, $egh = 1$, and $egv = 1$.

We measured the performance of our Smith Waterman algorithm implementation in cell updates per second (CUPS)

```
#pragma OPENCL EXTENSION cl_altera_channels : enable

channel int Sd_private[N] __attribute__ ((depth(1)));
channel int Sv_private[N] __attribute__ ((depth(0)));

__attribute__ ((reqd_work_group_size(X, 1, 1)))
__kernel void smith_waterman(int m, ...) {
    size_t i = get_global_id(0);
    ...
    int Sh_private[N];
    #pragma unroll N
    for (size_t j = 0; j < N; ++j) {
        int Sd = read_channel_altera(Sd_channel[j]);
        int Sh = Sh_private[j];
        int Sv = read_channel_altera(Sv_channel[j]);
        ...
        smith_waterman_pe(...);
        ...
        write_channel_altera(Sd_channel[j + 1], S);
        Sh_private[j + 1], S);
        write_channel_altera(Sv_channel[j], S);
    }
}
```

Fig. 7. Pseudocode for an NDRange-based Smith Waterman kernel

given by

$$\nu = \frac{m \times n}{t} \tag{3a}$$

$$= f \times \lambda, \tag{3b}$$

where m and n are the lengths of the database and query sequences, respectively, t is the kernel time to fill in the scoring matrix, f is the frequency of the Smith Waterman compute unit synthesized in the FPGA, and λ is the number of cell updates computed concurrently. We then calculated the power efficiency as the number of CUPS per watt that the entire FPGA board consumes, which we take to be the full 25 watts.

IV. Results

A. Resource Utilization

The FPGA resource utilization for our task-based Smith Waterman kernel is given in Table II. Our unidirectional linear systolic array of Smith Waterman processing elements consumes 0.23% of logic, 0.10% of registers, and 0.01% of memory blocks per processing element. There is an overall overhead of 22% for logic, 7.9% for registers, and 12% for memory blocks.

TABLE II
FPGA RESOURCE UTILIZATION

n	f	Logic	Registers	Memory Blocks
16	210 MHz	25%	9%	12%
32	195 MHz	29%	11%	13%
64	209 MHz	36%	14%	13%
128	186 MHz	51%	20%	13%
256	193 MHz	80%	32%	15%
Total	N/A	234720	938880	2560

B. Performance and Power Efficiency

The FPGA performance and power efficiency for our task-based Smith Waterman kernel is given in Table III. We achieve 0.095 GCUPS per Smith Waterman processing element with a power efficiency that reaches 0.988 GCUPS/W. Our task-based Smith Waterman kernel operates around 200 MHz with a latency of 2 clock cycles per cell update.

TABLE III
FPGA PERFORMANCE AND POWER EFFICIENCY

n	f	GCUPS	GCUPS/W
16	210 MHz	1.67	0.067
32	195 MHz	3.11	0.124
64	209 MHz	6.69	0.268
128	186 MHz	11.9	0.476
256	193 MHz	24.7	0.988

V. Conclusion

For $n = 256$, CPUs achieve 2.5 GCUPS running at 0.038 GCUPS/W in [10]. Meanwhile, GPUs have been shown to reach 9.4 GCUPS in [11], 14 GCUPS in [12], and 16 GCUPS in [13] with power efficiencies of 0.039, 0.058, and 0.067 GCUPS/W, respectively. Our OpenCL implementation of the Smith Waterman algorithm for the FPGA outperforms the best CPUs and GPUs above by a factor of 9.9 and 1.5 in absolute performance, and 26 and 15 in relative power efficiency, respectively. Furthermore, we have much room in our design for improvement. Our current implementation has an instantiation interval of 2, which means we could readily double our throughput to 49.4 GCUPS by interleaving two $n = 256$ query sequences per kernel execution. Alternatively, if we reduce the latency from 2 clock cycles per cell update to 1, we would also double our throughput to 49.4 GCUPS. Also, our current implementation can be extended to stream sequences and results to and from I/O using the Altera channels extension just like the Novo-G design by George, *et al* [8]. By streaming from I/O, we would save a significant amount of resources presently used to interface with global memory. Those resources would then be redistributed to build more Smith Waterman processing elements.

References

[1] T. F. Smith and M. S. Waterman, "The identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, pp. 195–197, 1981.

[2] O. Gotoh, "An improved algorithm for matching biological sequences," *J. Mol. Biol.*, vol. 162, pp. 705–708, 1982.

[3] *The OpenCL Specification, Version 1.0*, Khronos Group, 2009.

[4] *The OpenCL Specification, Version 1.1*, Khronos Group, 2011.

[5] *The OpenCL Specification, Version 1.2*, Khronos Group, 2012.

[6] *Altera SDK for OpenCL Programming Guide, Version 13.0sp1*, Altera Corporation, 2013.

[7] *Altera SDK for OpenCL Optimization Guide, Version 13.0sp1*, Altera Corporation, 2013.

[8] A. George, H. Lam, and G. Stitt, "Novo-G: At the forefront of scalable reconfigurable supercomputing," *Comput. Sci. Eng.*, vol. 13, no. 1, pp. 82–86, 2011.

[9] *Altera SDK for OpenCL Getting Started, Version 13.0sp1*, Altera Corporation, 2013.

[10] M. S. Farrar, "Striped Smith-Waterman database searches with inter-sequence SIMD parallelisation," *Bioinform.*, vol. 23, no. 2, pp. 156–161, 2007.

[11] Y. Liu, M. L. Douglas, and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Res. Notes*, vol. 2, no. 73, 2009.

[12] L. Ligowski and W. Rudnicki, "An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–8.

[13] Y. Liu, B. Schmidt, and M. L. Douglas, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Res. Notes*, vol. 3, no. 93, 2010.