

# Smith-Waterman Protein Search with OpenCL on FPGA

Enzo Rucci  
and A. De Giusti  
and M. Naiouf

Instituto de Investigacion en Informatica LIDI (III-LIDI)  
Universidad Nacional de La Plata  
La Plata (1900), Buenos Aires, Argentina  
Email: {erucci,degiusti,mnaiouf}@lidi.info.unlp.edu.ar

G. Botella  
and C. García  
and M. Prieto-Matias

Depto. Arquitectura de Computadores y Automatica,  
Universidad Complutense de Madrid, Madrid 28040, Spain  
Email: {gbotella,garsanca,mpmatias}@ucm.es

**Abstract**—The well-known Smith-Waterman (SW) algorithm is a high-sensitivity method for local alignments. Unfortunately, SW is expensive in terms of both execution time and memory usage, which makes it impractical in many scenarios. Previous research have shown that massive parallel architectures such as Graphic Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) are able to mitigate the computational problems achieving impressive speedups. In this paper we have explored SW acceleration on a FPGA device by means of OpenCL. We efficiently exploit data and thread-level parallelism on an Altera Stratix V FPGA card reaching upto 39 GCUPS with less than 25 watt power requirements.

**Keywords**—Bioinformatics, Smith-Waterman, FPGA, Altera, OpenCL.

## I. INTRODUCTION

High throughput structural genomic and genome sequencing have delivered to the scientific community a huge amount of data to be processed from structures and sequences of many thousand proteins. This “big data” can be interesting for the researchers to extract useful and functional insights from it.

Currently we can classify the computational approaches in two main techniques: full atom molecular dynamic simulation and bioinformatics approach. On one hand, molecular simulation uses principles from physics and physical chemistry to understand the function and folding of proteins. On other hand, bioinformatics science uses the statistical analysis of structures and protein sequences to identify the genome, to recognize their function, and additionally to anticipate structures when the only sequence information is available.

Bioinformatics is one of the most powerful technologies in life sciences nowadays, being used in researching of evolution theories, protein design among other important applications. Algorithms, methods and different findings used in these studies deal with a plethora of applications, such as functional classification of proteins, secondary structure prediction,

threading and modeling of distantly-related homologous proteins to represent its behavior through a cell’s life cycle and sequence and structure alignments.

Respecting sequence alignment, we find both -pattern searching among amino acid and nucleotide sequences- and also for the search of phylogenetic relationships among organisms. There are many models and algorithms for achieving the local alignment. Usually alignment algorithms are based on constructing indexes both for reference and read sequence. Depending on the property of the index, alignment methods can be classified into three groups: method based merge sorting [1], methods based on suffix trees [2] and algorithms based on hash tables [3].

Dynamic programming presents three different strategies to deal with this goal:

- Global strategies [4] achieve the alignment of every symbol in each sequence.
- Semiglobal strategies [5] search the best alignment between a short sequence and a long sequence, where internal and terminal gaps are penalized with different values since terminal gaps are probably caused by difference in sequence lengths.
- Local strategies comes up due to semiglobal alignment is insufficient for searching best match subsequences, since mismatching positions and gaps outside target subsequences produce non-favorable score. Smith Waterman (SW) strategy belongs to the so-called local methods. This approach is more useful when sequence to align holds significantly differences but remaining similar regions.

Focusing on the performance aspect many heuristics, such as BLAST [6] and FASTA [7] have been developed to reduce the execution time but at the expense of not guaranteeing to discover the optimal local alignments. Due to SW computational cost, the scientific community has achieved great efforts to design more efficient implementations in recent years. Most solutions proposed, find and exploit the inherent parallelism in the alignment process as:

- Intra-task parallelism approach, where the parallelism is present within a single pair of sequences. However,

©2015 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The final authenticated version is available online at <https://doi.org/10.1109/Trustcom.2015.634>

data dependencies involved in the alignment matrix, limit its scalability.

- Inter-task parallelism, based on performing several pairwise alignments concurrently. Its strength is joined with null data-dependency between alignments, which resembles the problem into an embarrassingly parallel one.

With the recent emergence of accelerator technologies in many-core architectures, such as Field-Programmable Gate Arrays (FPGAs), Cell/Broadband Engines (Cell/BEs), Graphic Processing Units (GPUs) among others, has rise up the opportunity of accelerating life science analysis problems on commonly available hardware at an inexpensive economic cost. For large parallel systems, we remark the proposal of Qiu [8] with a hybrid implementation using cloud computing based on the well-known MapReduce model and a common cluster programmed with MPI using grid architectures based solutions [9]. Moreover, some authors [10], [11] proposed the exploitation of subword-level capabilities inside of CPU/core, where the code SWIPE is able to achieve upto 106 billion cell updates per second (GCUPS). In the field of heterogeneous computing, Farrar [12] makes use of the outdated Cell/BE processors. Also, in the hardware accelerators scenario, we highlight the software *CUDASW++* and newer versions developed by Liu [13], [14], [15] which offer a performance from 30 to 185.6 GCUPS for single and multi CUDA-enabled Graphics Processor Units (GPUs) with CPU-enabled. More recently, Liu and Schmidt have presented *SWAPHI* and *SWAPHI-LS*, highly optimized hand-tuned SW implementations on Intel Xeon Phi accelerators [16], [17] for protein and DNA sequence alignments, respectively. While *SWAPHI-LS* is able to achieve 30.1 GCUPS, *SWAPHI* reaches upto 58.8 GCUPS. Although these two studies have centered mostly on the exploitation of the accelerator, Rucci et al. [18] focus on a hybrid implementation for protein alignment that exploits both CPU and coprocessors simultaneously.

Respect to FPGAs scenario, there are several works that present SW implementations for this kind of device [19], [20], [21], [22], [23], [24]. However, most of these implementations focus on DNA alignment (which is simpler than protein alignment from algorithmic perspective) and/or cover special cases of SW alignment (for example, query and/or database sequence with limited or fixed length, embedded sequences in the design, among others). Our paper presents an approach of SW implementation by means of Open Computing Language<sup>1</sup>(OpenCL) at present supported by Altera's FPGA. Altera has recently begun to support heterogeneous OpenCL programming and therefore to promote its use. Despite Altera staff submitted a SW implementation with OpenCL in [24], it focuses on no-real RNA sequence alignment with fixed query length. Our proposal covers protein sequence alignment and is tested with real amino acid datasets, besides being fully functional for any sequence length. We would like to remark that, the use of a FPGA in order to accelerate SW is also motivated by low energy demand in these devices. This work can be considered the starting point for more exhaustive exploration of greener power choices in heterogeneous computing scenario.

The rest of the paper is organized as follows. Section II introduces the basic concepts of the Smith-Waterman algorithm. Section III describes Altera's OpenCL programming extension and Section IV details the methodology addressed to efficiently program this application through different optimization techniques. Section V analyzes the results achieved and finally Section VI presents the conclusion and future lines for this novel viability study.

## II. SMITH-WATERMAN ALGORITHM

The Smith-Waterman algorithm is used to identify the optimal local alignment between two sequences. It was proposed by Smith and Waterman and improved by Gotoh [25]. This method employs a dynamic programming approach and its high sensitivity comes from exploring all the possible alignments between two sequences.

The recurrence relations for the SW algorithm with affine gap penalties are defined below.

$$H_{i,j} = \max \begin{cases} 0 \\ H_{i-1,j-1} + SM(q_i, d_j) \\ E_{i,j} \\ F_{i,j} \end{cases} \quad (1)$$

$$E_{i,j} = \max \begin{cases} H_{i,j-1} - G_{oe} \\ E_{i,j-1} - G_e \end{cases} \quad (2)$$

$$F_{i,j} = \max \begin{cases} H_{i-1,j} - G_{oe} \\ F_{i-1,j} - G_e \end{cases} \quad (3)$$

The two sequences to be compared are defined as  $q = q_1 q_2 q_3 \dots q_m$  and  $d = d_1 d_2 d_3 \dots d_n$ .  $H_{i,j}$  represents the score for aligning the segments of  $q$  and  $d$  ending at position  $i$  and  $j$ , respectively.  $E_{i,j}$  and  $F_{i,j}$  are the scores ending with a gap involving the first  $i$  symbols of  $q$  and the first  $j$  symbols of  $d$ , respectively.  $SM$  is the *substitution matrix* which defines the substitution scores for all residue pairs. In most cases,  $SM$  rewards with a positive value when  $q_i$  and  $d_j$  are identical, and punishes with a negative value otherwise.  $G_{oe}$  is the sum of gap open and gap extension penalties while  $G_e$  is the gap extension penalty. The recurrences should be calculated with  $1 \leq i \leq m$  and  $1 \leq j \leq n$ , and must start with  $H_{i,j} = E_{i,j} = F_{i,j} = 0$  when  $i = 0$  or  $j = 0$ . The optimal local alignment score  $S$  is the maximal alignment score in the matrix  $H$ .

It is important to note that any cell of the matrix  $H$  can be computed only after the values of the left and above cells are known, as shown in Figure 1. These dependences restrict the ways in that  $H$  can be computed.

## III. OPENCL EXTENSION ON ALTERA'S FPGA

OpenCL is a framework for parallel implementation that allow to execute a parallel programs across heterogeneous platforms. It is actually supported by several hardware devices such as CPUs, GPUs, DSPs, FPGAs and other processors. The OpenCL is based on host-device model, where host is in charge of device memory management, data transferring from/to device and kernel code invocation.

<sup>1</sup>Khronos Groups. OpenCL: <https://www.khronos.org/opencl>

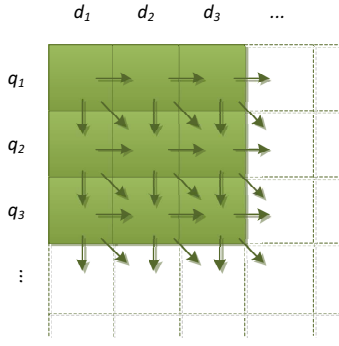


Fig. 1. Data dependencies in the alignment matrix  $H$ .

Kernel is a piece of code which expresses the parallelism of a program. OpenCL programming model divides a program workload into *work-groups* and *work-items*. *Work-items* are grouped into a *work-group*, which are executed independently respect to others *work-groups*. Data-level parallelism is regularly exploited in SIMD way, where several *work-items* are group according to lane width capabilities of the target device.

OpenCL memory model distinguishes different memory regions that are characterized by the access type, performance and scope. Global memory is read-write accessible by all *work-items* across all *work-groups* and it usually corresponds to the DRAM memory device which carries a high latency memory access. Local memory is a shared read-write memory accessible from all *work-items* of a single *work-group* and it habitually involves a low latency memory access. Constant memory is a read-only memory visible by all *work-items* across all *work-groups* and private memory as the name suggests it is only accessible by a single *work-item*.

As OpenCL is a cross platform standard for parallel programming on heterogeneous platforms, the developer can thus focus on algorithmic specifications, avoiding implementation details. The main advantage of implementing OpenCL on FPGA platforms concerns the shorter time to market and faster implementations. OpenCL Altera SDK for OpenCL supports the 1.0 specification which is a subset of the full profile with more flexible requirements and advanced features, which has been completed thanks to versions 1.1 and 1.2. The OpenCL specification defines a platform, memory and programming model which permits many add-ons that are vendor specific, cross-vendor and Khronos. There is considerable freedom in terms of implementing the platform as long as the final implementation satisfies the OpenCL specifications [26].

FPGAs present programmable networks containing logic elements, memory blocks and specific DSP blocks, and this allows the design of dynamic custom instruction pipelines in contrast with the fixed data-path architectures of CPUs and GPUs. Generally, digital design verification and creation have involved the use of Hardware Description Languages (HDLs), which are complex, error prone and affected by an extra abstraction layer as they contain the additional concept of time.

Regarding Altera's scope, FPGAs are dedicated coprocessors that obey a complex hierarchy model (see Table I particularized for the FPGA used in this research). The host processor is connected to accelerators through a peripheral interface such as PCIe.

TABLE I. OPENCL MEMORY MODEL FOR FPGAS

| OpenCL Memory | FPGA Memory | BittWare S5PHQ |
|---------------|-------------|----------------|
| global        | external    | 2x4GB DDR3     |
| constant      | cache       | 16KB DDR3      |
| local         | embedded    | 44Mbits        |
| private       | registers   | 674Kbits       |

Each Altera FPGA can have multiple in-order command queues associated with it that can execute independent commands concurrently. Kernels are compiled previously using the Altera OpenCL compiler and their content are passed at runtime to create the OpenCL program object. Regarding the execution model it is possible to use the *work-item* ordering within a pipeline, outperforming the obtained throughput thanks to this topology. The OpenCL paradigm model defines the execution of an instance of a kernel by a *work-item* using *NDRange*. Kernels are executed across a global domain of *work-items* where *work-items* are grouped into local *work-groups*. The execution model does not specify in what order the *work-items* are spread out and using the Altera implementation in a one dimensional *NDRange* *work-items* produce serial execution from 0 to the limit of global size.

Altera's OpenCL extension also takes advantage of I/O channels and kernels channels as in OpenCL 2.0 by means of pipes [27]. Altera's channel extension allows to transfer data between work-item's in the same kernel or between different kernels. It uses a first-in, first-out (FIFO) buffer without host interaction. This feature enables work-group communication without additional synchronization and host intervention.

#### IV. SW IMPLEMENTATION

In this section we will address the programming aspects and optimizations applied to our implementation on the FPGA platform. The algorithm comprises of three stages:

- 1) Pre-processing stage: the reference database is pre-processed to adapt sequence data for FPGA processing.
- 2) SW stage: after preprocessing database, alignments among query sequences and database sequences are carried out.
- 3) Sorting stage: last, all alignment scores are sorted in descending order.

It is important to remark that stages 1 and 3 are executed in the host while stage 2 is offloaded to the FPGA.

Algorithm 1 shows the pseudo-code for the host implementation where  $Q$  corresponds to query sequences and  $vD$  is the sequence database sorted by length in order to perform  $vD$  as  $c$  chunks of similar sequence lengths. Memory management is performed in OpenCL by means *clCreateBuffer* (memory allocation), *clEnqueueWriteBuffer* and *clEnqueueReadBuffer* (memory transfer to device/host). Kernel computes the alignments among a single query and a chunk of sequence database.

##### A. Task parallel programming model

The kernel is implemented following the task parallel programming model described in OpenCL 1.0 specification,

---

**Algorithm 1** Pseudo-code for Smith-Waterman host implementation

---

```

1:  $\triangleright Q$  are the query sequences
2:  $\triangleright vD$  is the preprocessed sequence database
3:
4: clCreateBuffer's(...)  $\triangleright$  Create buffers + transfer data
5: SetKernelArg's(...)  $\triangleright$  Set kernel common arguments
6: for  $c \leq \text{get\_num\_chunks}(vD)$  do
7:    $SP_c \leftarrow \text{build\_score\_profiles}(vD_c, SM)$ 
8:   clEnqueueWriteBuffer( $SP_c$ )  $\triangleright$  Score profiles to device
9:   SetKernelArg's(...)  $\triangleright$  Args. for processing chunk  $c$ 
10:  for  $q \leq \text{get\_num\_sequences}(Q)$  do
11:    SetKernelArg's(...)  $\triangleright$  Args. for query  $q$ 
12:    clEnqueueNDRangeKernel(...)  $\triangleright$  Compute alignments among query  $q$  and chunk  $c$ 
13:  end for
14: end for
15: clEnqueueReadBuffer( $scores$ )
16:  $\text{sort}(scores)$   $\triangleright$  Sort all scores in descending order

```

---

where the kernel consists of a single *work-group* that contains a unique *work-item*. This scheme is suitable because a single *work-item* does not require any synchronization stage.

Algorithm 2 shows the pseudo-code for our kernel implementation. The alignment matrix is divided into vertical blocks and computed in a row by row manner (see Figure 2). This blocking technique not only improves data locality but also reduces the memory requirements for computing a block, which favours the use of the private low-latency memory. The inner loop is fully unrolled to increase performance. Due to data dependences between blocks (last column  $H$  and  $E$  values are needed), we employed Altera OpenCL channels to efficiently transfer values previously computed. The combination of these techniques allows to Altera OpenCL compiler generate successfully parallel pipeline execution.

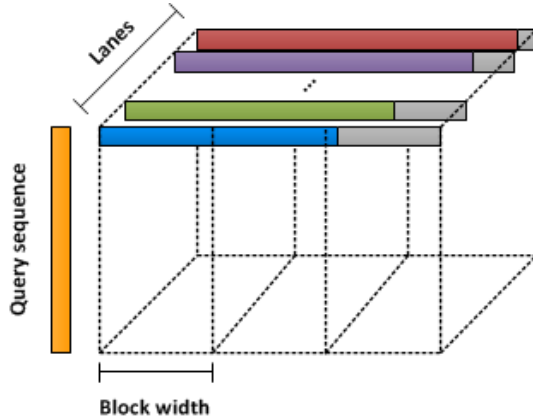


Fig. 2. Schematic representation of our OpenCL kernel implementation.

### B. Parallelization approach inside the kernel

Our SW kernel employs the inter-task parallelization approach. Instead of aligning one database sequence against a query sequence at a time, multiple database sequences are aligned in parallel by means of the SIMD vector capabilities available in the FPGA. For that reason, database sequences are

---

**Algorithm 2** Pseudo-code for Smith-Waterman kernel

---

```

1: #pragma OPENCL EXTENSION cl_altera_channels : enable
2:
3: __attribute__((reqd_work_group_size(1,1,1)))
4: __attribute__((task))
5: __kernel void SW_kernel ( vD_c, q, SP_c, scores_c ) {
6:  for  $s \leq \text{get\_num\_sequences}(vD_c)$  do
7:     $d \leftarrow \text{get\_sequence}(vD_c, s)$ 
8:     $numBlocks \leftarrow \text{sizeof}(d) / BLOCK\_WIDTH$ 
9:    for  $k \leq numBlocks$  do
10:     for  $i \leq \text{sizeof}(q)$  do  $\triangleright$  each row
11:      if  $k \neq 0$  then
12:         $j = k * BLOCK\_WIDTH$ 
13:         $H_{i-1,j-1} \leftarrow \text{read\_channel}(H\_channel)$ 
14:         $E_{i,j} \leftarrow \text{read\_channel}(E\_channel)$ 
15:      end if
16:      #pragma unroll
17:      for  $jj \leq BLOCK\_WIDTH$  do
18:         $j \leftarrow k * BLOCK\_WIDTH + jj$ 
19:         $\triangleright$  Calculate current cell value
20:         $H_{i,j} \leftarrow SW\_core(H, E, F, SP_c, q)$ 
21:         $score \leftarrow \max(score, H_{i,j})$ 
22:      end for
23:      if  $k \neq numBlocks - 1$  then
24:         $j \leftarrow (k + 1) * BLOCK\_WIDTH - 1$ 
25:        write_channel( $H\_channel, H_{i-1,j}$ )
26:        write_channel( $E\_channel, E_{i,j+1}$ )
27:      end if
28:    end for
29:  end for
30:   $scores_c[s] \leftarrow score$   $\triangleright$  Save alignment scores
31: end for
32: }

```

---

processed in groups. The size of the groups are determined by the number of SIMD vector lanes. In order to maximize processing efficiency, all the sequences of the same group should be of similar length. Therefore database sequences are sorted by their lengths in ascending order and then they are padded with dummy symbols to make their lengths multiple of  $BLOCK\_WIDTH$ . Because complete unrolling requires constant loop bounds, this last step is needed in order to fully unroll kernel inner loop.

### C. Substitution score selection

Our code also implements the Score Profile ( $SP$ ) optimization technique to obtain scores from substitution matrix. This technique is based on constructing an auxiliary  $n \times l \times |\Sigma|$  two-dimensional score array, where  $n$  is the length of the database sequence,  $l$  is the number of vector lanes and  $\Sigma$  is the alphabet. Since each row of the score profile forms a  $l$ -lane score vector, its values can be loaded in parallel. To reduce FPGA hardware resource usage, the score profiles are built in the host using a set of SSE intrinsic functions as in [18] and then transferred to FPGA. However, because the size of the score profiles can become prohibitive, database sequences are processed in chunks.

#### D. Data type selection

Optimise FPGA area usage is critical to obtain high performance OpenCL applications. The alignment scores do not need a wide range data representation. *short* data type turns out to be sufficient to represent all alignment scores computed in this work.

#### E. Host-side buffers and data transfers

Host-side buffers are allocated to be 64-bytes aligned. This fact improves data transfer efficiency because Direct Memory Access (DMA) takes part to and from the FPGA. Common data to all alignments, like the queries, are transferred when creating the device buffers. All scores are transferred to the host after all alignments have been computed.

### V. EXPERIMENTAL RESULTS

#### A. Experimental platform and tests carried out

All tests have been performed on a Xeon server running CentOS (release 6.5) equipped with:

- Two Intel Xeon CPU E5-2670 8-core 2.60GHz CPUs.
- 32 GB main memory.
- An Altera Stratix V GSD5 Half-Length PCIe Board with Dual DDR3 (two banks of 4 GByte DDR3) which power consumption is less than 25 watt.

We have used the Intel's ICC compiler (version 15.0.2) with the *-O3* optimization level by default. The synthesis tool used is Quartus II DKE V12.0 2 with OpenCL SDK v14.0.

To provide the most relevant study, tests are made with the Swiss-Prot database (release 2013\_11)<sup>2</sup>. Performance evaluation is carried out with 20 protein sequences. This database comprises 192480382 amino acids in 541561 sequences being the largest sequence length 35213. The 20 query protein sequences were selected from the aforementioned database (accession numbers: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450, and Q9UKN1), ranging in length from 144 to 5478. Moreover, BLOSUM62 was used as scoring matrix and gap open and extension penalties were set to 10 and 2, respectively. Each particular test was run ten times and performance was calculated with the average of ten execution times to avoid variability.

#### B. Performance Results

*Cell updates per second* (CUPS) is a commonly used performance measure in Smith-Waterman scenario, because it allows to remove the dependency on the query sequences and the databases utilized for the different tests. A CUPS represents the time for a complete computation of one cell in matrix *H*, including all memory operations and the corresponding computation of the values in the *E* and *F* arrays. Given a query

sequence of size *Q* and a database of size *D*, the GCUPS (billion cell updates per second) value is calculated by:

$$\frac{|Q| \times |D|}{t \times 10^9} \quad (4)$$

where  $|Q|$  is the total number of symbols in the query sequence,  $|D|$  is the total number of symbols in the database and *t* is the runtime in second [14]. In this article, runtime *t* includes the device buffer creation, the transfer time of host data to FPGA, the calculation time of the SW alignments, and the transfer-back time of the scores.

In order to evaluate FPGA performance rates, we have considered different implementations according to data-parallelism degree and memory hierarchy exploitation. We detail below the main differences:

- *scalar* version is the baseline code where non optimization are performed.
- SIMD versions exploit data level parallelism by enabling vectorization at expense of a moderate increase in resource usage. Vectorial nomenclature indicate SIMD width; i.e *int4* means small vectors of 4-elements, while *int8* and *int16* uses 8 and 16 short integer packaged respectively.
- Regarding to memory exploitation, *constant* version is referred to the use of read-only constant memory in which query sequences (*constant* × 1) or both query sequences and score profiles (*constant* × 2) are allocated in it.

Table II shows FPGA resource utilization and performance achieved for the different kernel versions considered. Without using vectorization (denoted as *scalar*), our implementation achieves poor performance. The exploitation of data level parallelism by enabling vectorization allows significant performance improvements. Highest GCUPS are obtained by *int16* version, which reports a speedup with respect to *scalar* of 11.5× at the cost of 0-2.8× increase in resource usage.

Because exploiting device memory hierarchy is critical to achieve high performance, the impact of using constant memory is evaluated. Copying query sequences to constant memory (*int16* + *constant* × 1) slightly improves performance with a minor reduction in resource usage. However, this enhancement is not significant in *int16* + *constant* × 2 version, where performance is strongly downgraded. *int16* + *constant* × 1 can take advantage of this feature since constant memory is optimized for high cache hit performance. In opposite sense, *int16* + *constant* × 2 does not benefit as score profiles exhibit poor data locality principally due to their huge sizes. Because of global memory accesses incorporated an extra hardware to improve long memory latencies, better performance can be obtained if score profile data is transferred directly to this memory.

We also evaluate the impact of the query length, Figure 3 illustrates the performance of the different kernel implementations varying query length. As it can be seen, *scalar* kernel hardly improves performance. Vectorized kernels benefit from larger workloads, except for *int16* + *constant* × 2 where constant memory usage for score profiles practically cancel any performance gain from vectorization. However, *int16* +

<sup>2</sup>Swiss-Prot database available in [http://web.expasy.org/docs/swiss-prot\\_guideline.html](http://web.expasy.org/docs/swiss-prot_guideline.html)

TABLE II. PERFORMANCE AND RESOURCE USAGE COMPARISON FOR DIFFERENT OPENCL KERNEL IMPLEMENTATIONS.

| Version                     | Performance (GCUPS) | Resource Usage |      |     |      |
|-----------------------------|---------------------|----------------|------|-----|------|
|                             |                     | ALMs           | Regs | RAM | DSPs |
| scalar                      | 3.41                | 29%            | 17%  | 29% | 1%   |
| int4                        | 18                  | 39%            | 19%  | 34% | 1%   |
| int8                        | 23.66               | 54%            | 25%  | 43% | 1%   |
| int16                       | 38.9                | 81%            | 36%  | 80% | 1%   |
| int16 + constant $\times 1$ | 39.2                | 80%            | 35%  | 80% | 1%   |
| int16 + constant $\times 2$ | 3.53                | 78%            | 25%  | 77% | 1%   |

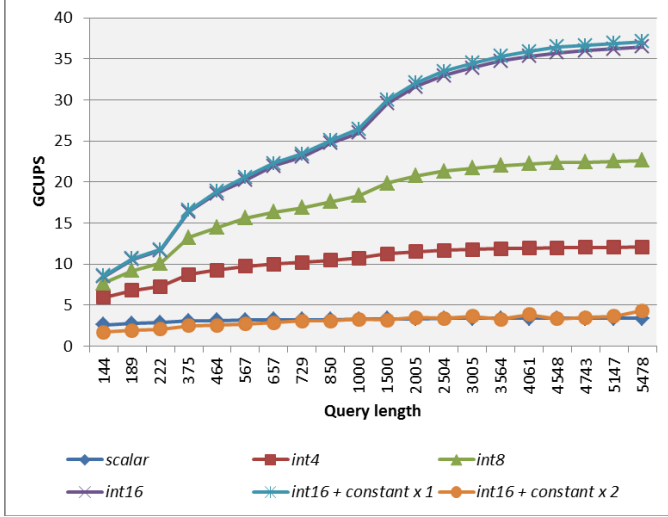


Fig. 3. Performance of the different OpenCL kernel implementations with queries of varying length.

*constant*  $\times 1$  version outperforms all other kernels implementation reaching upto 37.1 GCUPS.

## VI. CONCLUSIONS

The SW algorithm is one of the most popular algorithm in sequence alignment because it performs an exact local alignment. However, due to its computational complexity, in practice, several parallel implementations are used to reduce response time. In addition, with the emergence of heterogeneous computing and interest of the exploration not only in computationally scalable solutions but also energy efficiency, it is open the chance of considering new parallel programming paradigms and evaluate the behavior of new devices that meet these requirements. Taken into account these considerations, this paper examines the benefits of a highly innovative technology as the support of the parallel programming model OpenCL in the field of FPGAs.

Among main contribution of this research we can summarize:

- Data level parallelism is crucial to achieve successful performance rates at expense of a moderate increase in resource usage.
- OpenCL hierarchy memory exploitation such as private memory reports considerably benefits although constant memory usage hardly improves the performance.

- From performance perspective, our most successful implementation reaches upto 39 GCUPS, significantly higher than [24].

In heterogeneous computing era not only performance matters but also energy efficiency. We would like to emphasize that our SW implementation in FPGA is not only competitive in terms of performance but it also is much more power-efficient (GCUPS/Watt) than other well-known implementations as *SWIPE* [11], *CUDASW++* [14] or *SWAPHI* [16] considering the Thermal Design Power of the target devices.

According to the results obtained, we plan to analyse the scalability of SW algorithm using a hybrid system based on a multicore + FPGA, multi-FPGAs and the exploitation of the best power-efficient trade-off.

## ACKNOWLEDGMENT

Enzo Rucci holds a PhD CONICET Fellowship under Argentinian Government.

## REFERENCES

- [1] N. Malhis and S. J. M. Jones, "High quality snp calling using illumina data at shallow coverage," *Bioinformatics*, vol. 26, no. 8, pp. 1029–1035, 2010.
- [2] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, ser. FOCS '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 390–. [Online]. Available: <http://dl.acm.org/citation.cfm?id=795666.796543>
- [3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psi-blast: a new generation of protein database search programs," *Nucleic Acids Res.*, vol. 25, no. 17, pp. 3389–3402, September 1997.
- [4] 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, Mar. 1970.
- [5] M. Brudno, S. Malde, A. Poliakov, C. B. Do, O. Couronne, I. Dubchak, and S. Batzoglou, "Glocal alignment: finding rearrangements during alignment," in *ISMB (Supplement of Bioinformatics)*, 2003, pp. 54–62. [Online]. Available: <http://dblp.uni-trier.de/db/conf/ismb/ismb2003.html#BrudnoMPDCDB03>
- [6] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped blast and psiblast: a new generation of protein database search programs," *NUCLEIC ACIDS RESEARCH*, vol. 25, no. 17, pp. 3389–3402, 1997.
- [7] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 85, no. 8, pp. 2444–2448, Apr. 1988.
- [8] J. Qiu, J. Ekanayake, T. Gunarathne, J. Y. Choi, S. H. Bae, H. Li, B. Zhang, T. Wu, Y. Ruan, S. Ekanayake, A. Hughes, and G. Fox, "Hybrid cloud and cluster computing paradigms for life science applications," *BMC Bioinformatics*, vol. 11 (Suppl12), 2010.

- [9] F. Chichizola, M. Naiouf, L. D. Giusti, I. Rodriguez, and A. D. Giusti, "Overhead Analysis in Parallel Processing DNA Sequences on Grid Architectures," in *Proceedings of the LAGrid08 (2nd International Latin American Grid Workshop 2008)*, 2008.
- [10] M. Farrar, "Striped Smith-Waterman speeds database searches six time over other SIMD implementations," *Bioinformatics*, vol. 23 (2), pp. 156–161, 2007.
- [11] T. Rognes, "Faster Smith-Waterman database searches with inter-sequence SIMD parallelization," *BMC Bioinformatics*, vol. 12:221, 2011.
- [12] M. Farrar. Optimizing Smith-Waterman for the Cell Broad-band Engine. [Online]. Available: <http://farrar.michael.googlepages.com/SW-CellBE.pdf>
- [13] Y. Liu, W. Huang, J. Johnson, and S. Vaidya, "GPU Accelerated Smith-Waterman," *Lecture Notes in Computer Science*, vol. 3994, pp. 188–195, 2006.
- [14] Y. Liu, D. L. Maskell, and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Research Notes*, vol. 2:73, 2009.
- [15] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions," vol. 14:117, 2013.
- [16] Y. Liu and B. Schmidt, "Swaphi: Smith-waterman protein database search on xeon phi coprocessors," in *25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2014)*, 2014.
- [17] Y. Liu, T.-T. Tran, L. Felix, and B. Schmidt, "SWAPHI-LS: Smith-waterman Algorithm on Xeon Phi Coprocessors for Long DNA Sequences," in *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER 2014)*, September 2014.
- [18] E. Rucci, A. D. Giusti, Marcelo, G. Botella, C. Garcia, and M. Prieto-Matias, "Smith-Waterman Algorithm on Heterogeneous Systems: A Case Study," in *Proceedings of the IEEE Cluster 2014*, 2014.
- [19] T. I. Li, W. Shum, and K. Truong, "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)," *BMC Bioinformatics*, vol. 8:185, 2007.
- [20] S. Dydel and P. Bala, "Large scale protein sequence alignment using fpga reprogrammable logic devices," in *Field Programmable Logic and Application*, ser. Lecture Notes in Computer Science, J. Becker, M. Platzner, and S. Vernalde, Eds. Springer Berlin Heidelberg, 2004, vol. 3203, pp. 23–32. [Online]. Available: [http://dx.doi.org/10.1007/978-3-540-30117-2\\_5](http://dx.doi.org/10.1007/978-3-540-30117-2_5)
- [21] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzyniek, "Post-placement c-slow retiming for the xilinx virtex fpga," in *Proceedings of the 2003 ACM/SIGDA Eleventh International Symposium on Field Programmable Gate Arrays*, ser. FPGA '03. New York, NY, USA: ACM, 2003, pp. 185–194. [Online]. Available: <http://doi.acm.org/10.1145/611817.611845>
- [22] Y. Yamaguchi, H. Tsoi, and W. Luk, "Fpga-based smith-waterman algorithm: Analysis and novel design," in *Reconfigurable Computing: Architectures, Tools and Applications*, ser. Lecture Notes in Computer Science, A. Koch, R. Krishnamurthy, J. McAllister, R. Woods, and T. El-Ghazawi, Eds. Springer Berlin Heidelberg, 2011, vol. 6578, pp. 181–192. [Online]. Available: [http://dx.doi.org/10.1007/978-3-642-19475-7\\_20](http://dx.doi.org/10.1007/978-3-642-19475-7_20)
- [23] M. Isa, K. Benkrid, T. Clayton, C. Ling, and A. Erdogan, "An fpga-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, June 2011, pp. 344–351.
- [24] S. O. Settle, "High-performance dynamic programming on fpgas with opencl," 2014.
- [25] O. Gotoh, "An improved algorithm for matching biological sequences," in *Journal of Molecular Biology*, vol. 162, 1981, pp. 705–708.
- [26] A. Cooperation, "Altera SDK for OpenCL Programming Guide, Version 13.0sp1," 2012.
- [27] K. Group, "The OpenCL Specification. version 2.0," 2014. [Online]. Available: <https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf>