2015 IEEE Trustcom/BigDataSE/ISPA

# Smith-Waterman Protein Search with OpenCL on an FPGA

E. 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

C. García
and G. Botella
and M. Prieto-Matias
Depto. Arquitectura de Computadores y Automatica,
Universidad Complutense de Madrid, Madrid 28040, Spain
Email: {garsanca,gbotella,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 has shown that massively parallel architectures such as GPUs and FPGAs are able to mitigate the computational problems and achieve impressive speedups. In this paper we explore SW acceleration on an FPGA with OpenCL. We efficiently exploit data and thread-level parallelism on an Altera Stratix V FPGA, obtaining up to 39 GCUPS with less than 25 watt of power consumption.**

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

## I. Introduction

High throughput structural genomics and genome sequencing have provided the scientific community with a huge amount of data to be processed from structures and sequences of many thousands of proteins. This "big data" can be interesting as researchers can extract useful and functional insights from it.

Bioinformatics is one of the most powerful technologies in life sciences nowadays, and it is being used in research into evolution theories and protein design, among other important applications. Algorithms, methods and different findings used in these studies offer a plethora of applications, such as the functional classification of proteins, secondary structure prediction, threading and modeling of distantly-related homologous proteins to represent their behavior throughout a cell's life cycle, and sequence and structure alignments.

Many bioinformatics operations and analyses are partly held by sequence alignment, both for pattern searching among amino-acid and nucleotide sequences, and for the search of phylogenetic relationships among organisms. The Smith-Waterman (SW) algorithm for local alignment is one of these methods; it focuses on similar regions only in part of the sequences, which means that the purpose of the algorithm is finding small, locally similar regions. This method has been used as the basis for many subsequent algorithms and is often used as basic pattern to compare different alignment techniques. To calculate optimal local alignment scores, the SW algorithm has a linear space complexity and a quadratic time complexity.

Biological sequence database searches require calculating optimal alignment scores many times. Due to the high complexity of SW algorithm, the computation time may become impracticable. For this reason, several heuristics, such as BLAST [1] and FASTA [2], have been developed to reduce the execution time but at the expense of not guaranteeing to discover the optimal local alignments. Due to the computational cost of SW, the scientific community has made great efforts to design more efficient implementations in recent years. Most of the solutions proposed find and exploit the inherent parallelism in the alignment process as intra-task and inter-task parallelism [3].

With the recent emergence of accelerator technologies, such as Field-Programmable Gate Arrays (FPGAs), and Graphics Processing Units (GPUs), has come the opportunity of speeding up life science analysis problems on commonly available hardware at a relatively low cost. For large parallel systems, we highlight the proposal of Qiu [4] with a hybrid implementation based on the MapReduce model and a cluster programmed with MPI using grid-architecture-based solutions. Some authors have proposed the exploitation of subword-level capabilities of CPU/core [5], [3]. Also, in the hardware accelerator scenario, we can point to the CUDASW++ software and newer versions developed by Liu [6], [7], which offer a performance from 30 to 185.6 GCUPS (billion cell updates per second) for single and multi Graphics Processor Units (GPUs). More recently, Liu and Schmidt have presented SWAPHI and SWAPHI-LS, which are highly optimized hand-tuned SW implementations on Intel Xeon Phi accelerators [8], [9] for protein and DNA sequence alignments, respectively. While SWAPHI-LS is able to achieve 30.1 GCUPS, SWAPHI obtains up to 58.8 GCUPS. Although these two studies have focused mostly on the exploitation of the accelerator, Rucci et al. [10], [11] focus on a hybrid implementation for protein alignment that exploits both CPU and coprocessors simultaneously.

There are also some previous works that studied the implementation of SW on FPGAs [12], [13], [14], [15], [16], [17]. However, most of these implementations focus on DNA alignment (which is simpler than protein alignment from an algorithmic perspective) and/or cover special cases of SW alignment (for example, query and/or database sequences of limited or fixed length, embedded sequences in the design, among others). Our paper presents an approach for SW im-

978-1-4673-7952-6/15 $31.00 © 2015 IEEE
DOI 10.1109/Trustcom.2015.634

plementation using the Open Computing Language[1] (OpenCL). Altera has recently begun to support heterogeneous OpenCL programming, and therefore to promote its use. Despite Altera staff having submitted an SW implementation with OpenCL in [17], it focuses on non-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 point out that the use of an FPGA to accelerate SW is also motivated by the low energy demands of these devices. This work can be considered the starting point for more exhaustive exploration of greener power choices.

The rest of the paper is organized as follows. Section II introduces the basic concepts of the Smith-Waterman algorithm. Section III introduces Altera's OpenCL programming extension and in Section IV the methodology used to efficiently program this alignment through different optimization techniques is described. Section V presents the results obtained, and finally Section VI contains 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 [18]. 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 $H$.

[1] Khronos Groups. OpenCL: https://www.khronos.org/opencl

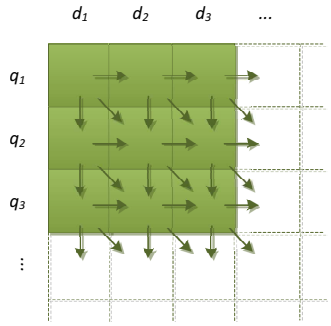


Fig. 1. Data dependences in the alignment matrix *H*.

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 which *H* can be computed.

## III. OpenCL Extension on Altera's FPGA

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

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

The 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 that is visible to all *work-items* across all *work-groups*, and private memory, as the name suggests, 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 supports the OpenCL 1.0 specification which is a subset of the full profile with more flexible requirements and advanced features, and 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, either vendor or cross-vendor specific. There is considerable

freedom in terms of implementing the platform as long as the final implementation satisfies the OpenCL specifications [19].

FPGAs provide 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.

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 |

Regarding Altera's scope, the FPGAs are dedicated co-processors 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.

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 is 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 throughput obtained 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 distributed, and using the Altera implementation in a one-dimensional *NDRange* work-items produce a serial execution from 0 to the global size limit.

Altera's OpenCL extension also takes advantage of I/O channels and kernel channels as in OpenCL 2.0 by means of pipes [20]. Altera's channel extension allows the transfer of data between work-items' 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 address the programming aspects and optimizations applied to our implementation on the FPGA platform. The algorithm comprises three stages:

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

Stages 1 and 3 are executed on the host, while stage 2 is offloaded to the FPGA.

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

```
1: ▷ Q are the query sequences
2: ▷ vD is the preprocessed sequence database
3:
4: clCreateBuffer's(...) ▷ Create buffers + transfer data
5: SetKernelArg's(...) ▷ Set kernel common arguments
6: for c ≤ get_num_chunks(vD) do
7:     SP_c ← build_score_profiles(vD_c, SM)
8:     clEnqueueWriteBuffer(SP_c) ▷ Score profiles to device
9:     SetKernelArg's(...) ▷ Args. for processing chunk c
10:    for q ≤ get_num_sequences(Q) do
11:        SetKernelArg's(...) ▷ Args. for query q
12:        clEnqueueNDRangeKernel(...) ▷ Compute alignments among query q and chunk c
13:    end for
14: end for
15: clEnqueueReadBuffer(scores)
16: sort(scores) ▷ Sort all scores in descending order
```

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 divide $vD$ as $c$ chunks of similar sequence lengths. Memory management is performed in OpenCL by means of *clCreateBuffer* (memory allocation), *clEnqueueWriteBuffer* and *clEnqueueReadBuffer* (memory transfer to device/host). The kernel computes the alignments between a single query and a chunk of sequence database.

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 improve performance. Due to data dependencies between blocks (last column $H$ and $E$ values are needed), we employed Altera OpenCL channels to efficiently transfer previously-computed values. The combination of these techniques allows the Altera OpenCL compiler to successfully generate parallel pipeline execution.

### A. Parallelism 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 this reason, database sequences are processed in groups. The size of the groups is 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 a multiple

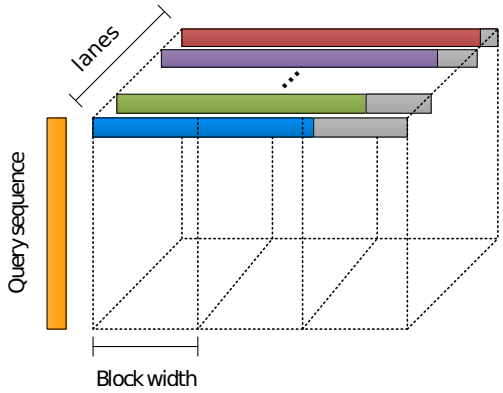


Fig. 2. Schematic representation of our OpenCL kernel implementation.

**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 ≤ get_num_sequences(vD_c) do
7:      d ← get_sequence(vD_c, s)
8:      numBlocks ← sizeof(d)/BLOCK_WIDTH
9:      for k ≤ numBlocks do
10:         for i ≤ sizeof(q) do ▷ each row
11:             if k ≠ 0 then
12:                 j = k * BLOCK_WIDTH
13:                 H_{i-1,j-1} ← read_channel(H_channel)
14:                 E_{i,j} ← read_channel(E_channel)
15:             end if
16:             #pragma unroll
17:             for jj ≤ BLOCK_WIDTH do
18:                 j ← k * BLOCK_WIDTH + jj
19:                 ▷ Calculate current cell value
20:                 H_{i,j} ← SW_core(H, E, F, SP_c, q)
21:                 score ← max(score, H_{i,j})
22:             end for
23:             if k ≠ numBlocks − 1 then
24:                 j ← (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] ← score ▷ Save alignment scores
31: end for
32: }
```

of $BLOCK\_WIDTH$. Because complete unrolling requires constant loop bounds, this last step is needed in order to fully unroll the kernel inner loop.

### B. Substitution score selection

Our code also implements the Score Profile ($SP$) optimization technique to obtain scores from the substitution matrix. This technique is based on constructing an auxiliary $n \times l \times |\sum|$ two-dimensional score array, where $n$ is the length of the database sequence, $l$ is the number of vector lanes and $\sum$ 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 on the host using a set of SSE intrinsic functions as in [10] and then transferred to the FPGA. However, because the size of the score profiles can become prohibitive, database sequences are processed in chunks.

### C. Data type selection

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

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

Host-side buffers are allocated to be 64-byte aligned. This fact improves data transfer efficiency because Direct Memory Access (DMA) takes place to and from the FPGA. Common data to all alignments, such as 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 were 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) whose power consumption is less that 25 watts.

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

To provide the most relevant study, tests were performed with the Swiss-Prot database (release 2013_11)[2]. Performance evaluation was carried out with 20 protein sequences. This database comprises 192480382 amino acids in 541561 sequences, the largest sequence being 35213 in length. 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. Furthermore, BLOSUM62 was used as the scoring matrix, and gap open and extension penalties were set to 10 and 2, respectively. Each particular test was run ten times, and the performance was calculated with the average of ten execution times to avoid variability.

[2]Swiss-Prot database available in http://web.expasy.org/docs/swiss-prot_guideline.html

### B. Performance Results

*Cell updates per second* (CUPS) is a commonly used performance measure in Smith-Waterman, because it allows the removal of the dependency on the query sequences and the databases utilized for the different tests. 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:

$$\frac{|Q| \times |D|}{t \times 10^9} \tag{4}$$

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 seconds. In this paper, the runtime $t$ includes the device buffer creation, the cost of data transfer between host and the FPGA and time of the score alignments.

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

- the *scalar* version is the baseline code without optimizations.
- SIMD versions exploit data level parallelism by enabling vectorization at the expense of a moderate increase in resource usage. Vectorial nomenclature indicates SIMD width; i.e *int4* means small vectors of 4-elements, while *int8* and *int16* use 8 and 16 short integer packages, respectively.
- Regarding memory exploitation, the *constant* version refers to the use of read-only constant memory in which query sequences (*constant* $\times$ *1*) or both query sequences and score profiles (*constant* $\times$ *2*) are allocated to it.

Table II shows FPGA resource utilization and the performance obtained for the different kernel versions considered. Without using vectorization (denoted as *scalar*), our implementation performs poorly. The exploitation of data level parallelism by enabling vectorization allows significant performance improvements. The highest GCUPS are obtained by the *int16* version, which achieves a 11.5$\times$ speedup with a negligible resource usage increment.

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

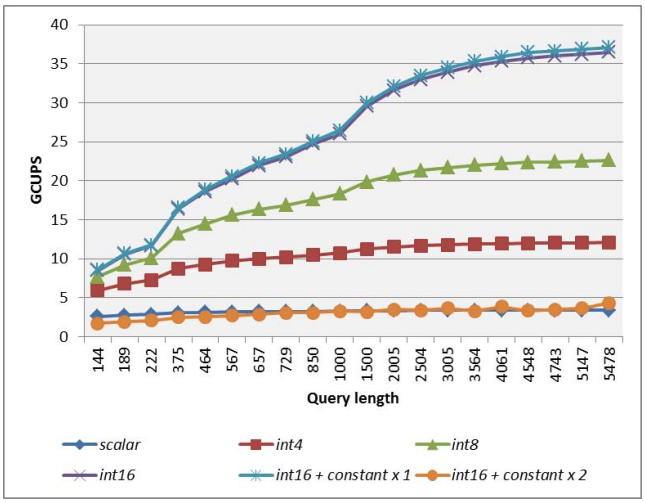


Fig. 3. Performance of the different OpenCL kernel implementations with queries of varying length.

We also evaluate the impact of query length, and Figure 3 illustrates the performance of the different kernel implementations with varying query length. As can be seen, the *scalar* kernel hardly improves performance. The vectorized kernels benefit from larger workloads, except for *int16* + *constant* $\times$ *2*, where constant memory usage for score profiles practically cancels out any performance gain from vectorization. However, the *int16* + *constant* $\times$ *1* version outperforms all other kernel implementations, reaching up to 37.1 GCUPS.

## VI. Conclusions

The SW algorithm is one of the most popular algorithms 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 the response time. In addition, with the emergence of heterogeneous computing and interest in the exploration not only of computationally scalable solutions but also energy efficiency, the chance arises of considering new parallel programming paradigms and evaluating the behavior of new devices that meet these requirements.

Among the main contributions of this research we can highlight:

- Data level parallelism is crucial to achieve successful performance rates at the expense of a moderate increase in resource usage.
- The exploitation of OpenCL memory hierarchy, such as the private memory offers considerable benefits, although constant memory usage hardly improves the performance.
- Our most successful implementation reaches up to 39 GCUPS, which is significantly higher than previous implementation [17].

In this heterogeneous computing era, not only performance matters but also energy efficiency. We would like to emphasize

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% |

that our SW implementation on an FPGA is not only competitive in terms of performance, but it is also more power-efficient (GCUPS/Watt) than other well-known implementations such as *SWIPE* [3], *CUDASW++* [6] or *SWAPHI* [8] considering the Thermal Design Power of the target devices.

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

## ACKNOWLEDGMENT

Enzo Rucci holds a PhD CONICET Fellowship under Argentinian Government and this work has been partially supported by Spanish goverment through research contract TIN 2012-32180 and CAPAP-H5 network (TIN2014-53522).

## REFERENCES

[1] S. F. Altschul, T. L. Madden, A. A. Schffer, 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.

[2] 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.

[3] T. Rognes, "Faster Smith-Waterman database searches with inter-sequence SIMD parallelization," *BMC Bioinformatics*, vol. 12:221, 2011.

[4] 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.

[5] M. Farrar, "Striped Smith-Waterman speeds database searches six time over other SIMD implementations," *Bioinformatics*, vol. 23 (2), pp. 156–161, 2007.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] E. Rucci, G. Botella, C. Garcia, A. D. Giusti, M. Naiouf, and M. Prieto-Matias, "Smith-Waterman Algorithm on Heterogeneous Systems: A Case Study," in *Proceedings of the IEEE Cluster 2014*, 2014.

[11] E. Rucci, C. Garcia, G. Botella, A. D. Giusti, M. Naiouf, and M. Prieto-Matias, "An Energy-aware Performance Analysis of SWIMM: Smith-Waterman Implementation on Intel's Multicore and Manycore Architectures," in *Concurrency and Computation: Practice and Experience*. Wiley, 2015.

[12] 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:I85, 2007.

[13] 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

[14] N. Weaver, Y. Markovskiy, Y. Patel, and J. Wawrzynek, "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

[15] 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

[16] 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.

[17] S. O. Settle, "High-performance dynamic programming on fpgas with opencl," 2014.

[18] O. Gotoh, "An improved algorithm for matching biological sequences," in *Journal of Molecular Biology*, vol. 162, 1981, pp. 705–708.

[19] A. Coorporation, "Altera SDK for OpenCL Programming Guide, Version 13.0sp1," 2012.

[20] K. Group, "The OpenCL Specification. version 2.0," 2014. [Online]. Available: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf