Journal Title
XX(X):1–12
©The Author(s) 2015
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

OSWALD: *O*penCL *S*mith-*W*aterman on *A*ltera FPGA for *L*arge Protein *D*atabases

Enzo Rucci[1], Carlos Garcia[2], Guillermo Botella[2], Armando De Giusti[1], Marcelo Naiouf[1] and Manuel Prieto-Matias[2]

Abstract

The well-known Smith-Waterman (SW) algorithm is a high-sensitivity method for local sequence alignments. Unfortunately, SW has quadratic time complexity, which makes this algorithm computationally demanding for large protein databases. In this paper, we present OSWALD, a portable, fully functional and general implementation to accelerate SW database searches in heterogeneous platforms based on Altera's FPGA. OSWALD exploits OpenMP multithreading and SIMD computing through SSE and AVX2 extensions on the host while it takes advantage of pipeline and vectorial parallelism by way of OpenCL on the FPGAs. Performance evaluations on two different heterogeneous architectures with real amino acids datasets show that OSWALD is competitive in comparison with other top-performing SW implementations reaching up to 442 GCUPS peak with the best GCUPS/watts ratio.

Keywords

Bioinformatics, Smith-Waterman, FPGA, Altera, OpenCL.

Introduction

High throughput structural genomic 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 for researchers in order to extract useful and functional insights. One of the main computational approaches is bioinformatics, which uses the statistical analysis of structures and protein sequences to identify the genome, recognize function, and additionally to anticipate structures when only sequence information is available. Bioinformatics has become 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.

Sequence alignment is a common task in bioinformatics, and can be considered the basis of other biological tools. This procedure is used to compare primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences. The Smith-Waterman (SW) is the most accurate method for local sequence alignment and its high sensitivity comes from exploring all the possible alignments between two sequences. This algorithm focuses on similar regions only in part of the sequences, which means that the purpose of the algorithm is finding small, locally similar regions. To calculate optimal local alignment scores, the SW algorithm has a linear space complexity and a quadratic time complexity.

Considering the performance aspect, the SW computation time may become impracticable due to its high complexity, specially with large volume datasets. For this reason, several heuristics, such as BLAST Altschul et al. (1990) and FASTA Lipman and Pearson (1985), have been developed to reduce the execution time but at the expense of not guaranteeing to discover the optimal local alignments. Because of 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 Rognes (2011).

With the recent emergence of accelerator technologies, such as Field-Programmable Gate Arrays (FPGAs), vector processing units (SIMD) in many-core architectures, Graphics Processing Units (GPUs), among others, the challenge of accelerating life science analysis problems has become more stimulating. Moreover, due the affordable cost of these devices, their exploitation is becoming an attractive solution.

As related work, we found a hybrid implementation of SW Qiu et al. (2010) which makes use of cloud computing and a cluster programmed with MPI. Moreover, there exist SW versions based on SIMD-vector exploitation Farrar (2007); Rognes (2011) that are available now on modern

[1]Instituto de Investigacion en Informatica LIDI (III-LIDI),Universidad Nacional de La Plata, Buenos Aires, Argentina
[2]Depto. Arquitectura de Computadores y Automatica, Universidad Complutense de Madrid, Spain

Corresponding author:
Carlos Garcia, Depto. Arquitectura de Computadores y Automatica Universidad Complutense de Madrid, Madrid 28040, Spain.
Email: erucci@lidi.info.unlp.edu.ar

CPUs. In the field of heterogeneous computing, Farrar Farrar (2008) makes use of the outdated Cell/BE processors. Also, in the hardware accelerators scenario, the most successful solution is the *CUDASW++* software, and its newer versions Liu et al. (2009, 2010, 2013), which offer a performance range from 30 to 185.6 GCUPS (billion cell updates per second) for single and multi CUDA-enabled Graphics Processor Units (GPUs) with concurrent CPU computing. More recently, Liu and Schmidt have released an optimized hand-tuned SW implementation for Intel Xeon Phi coprocessors Liu and Schmidt (2014); Liu et al. (2014), denoted as *SWAPHI* and *SWAPHI-LS*, for protein and DNA sequence alignment, respectively. While *SWAPHI-LS* is able to achieve 30.1 GCUPS, *SWAPHI* obtains up to 58.8 GCUPS. Besides pointing out Intel Xeon Phi exploitation, Rucci et al. Rucci et al. (2014, 2015b) have recently studied not only the performance aspect but also the energy footprint on a hybrid implementation that exploits both CPU and coprocessors simultaneously.

Eventhough, in FPGAs scenario Li et al. (2007); Dydel, Stefan and Bala, Piotr (2004); Weaver et al. (2003); Yamaguchi et al. (2011); Isa et al. (2011) present SW implementation on a FPGA. However, most of this software implements DNA alignment (which is simpler than protein alignment from an algorithmic perspective) and/or covers special cases in SW alignment (for example, query and/or database sequences of limited or fixed length, embedded sequences in the design, among others). In addition, these implementations are based on hardware description languages such as VHDL or Verilog, which limits its portability to other parallel devices. Under this premise, Altera tries to promote its FPGA usage by means of the support of the Open Computing Language[*] model (OpenCL), traditionally used in heterogeneous computing environments based on multicore and GPU. Despite Altera staff having submitted an implementation of SW with OpenCL Settle (2014), their implementation focuses on non-real RNA sequence alignment with fixed query length.

Although previous studies have focused on exploiting the FPGA, to the best of our knowledge our approach is the first high-level programming implementation on FPGAs using OpenCL with real amino acid datasets. Our implementation is a fully functional solution for any sequence length and general for FPGA-based platforms with different hardware characteristics. This paper extends the insights already offered in our previous approach Rucci et al. (2015a), with the following new contributions:

- Among the main contributions, we can highlight the creation of a public git repository with the binary executable developed for this paper, denoted as OSWALD [†]. OSWALD is a software to accelerate the well-known SW algorithm on heterogeneous platforms based on Altera's FPGA by means of high-level programming using OpenCL. OSWALD exploits OpenMP multithreading and SIMD computing through SSE and AVX2 extensions on the host while it takes advantage of pipeline and vectorial parallelism on the FPGAs.
- Regarding the original implementation, we have focused on OpenCL kernel optimization through FPGA resource stressing. The analysis includes a performance and resource usage evaluation of different kernel implementations.
- We have extended the optimized single-FPGA implementation to allow multiple FPGAs and, subsequently, to support concurrent host computation. A performance evaluation was carried out using two different protein databases over two heterogeneous architectures.
- In addition, we have compared our hybrid CPU-FPGA implementation to other reference implementations. For this purpose we have chosen the best performing CPU-based, Xeon Phi-based and GPU-based alternatives: *SWIMM* Rucci et al. (2015b) was selected for Xeon and Xeon Phi processors while CUDASW++ 3.0 Liu et al. (2013) was chosen for CUDA-compatible GPUs.
- Finally, this paper does not only focus on the performance analysis of FPGA-based architectures, but it also considers power consumption. It explores different configurations in order to find the fastest performance, the lowest power consumption and the best performance/power ratio.

The rest of the paper is organized as follows. Section 2 introduces the basic concepts of the Smith-Waterman algorithm. Section 3 introduces Altera's OpenCL programming extension and in Section 4 the methodology to efficiently program this alignment through different optimization techniques is described. Section 5 presents the results obtained, and finally Section 6 contains the conclusions and future lines of work for this novel viability study.

Smith-Waterman Algorithm

In 1981 Smith and Waterman proposed an algorithm to find the optimal local alignment of two sequences (Smith and Waterman 1981). This algorithm is based on dynamic programming and was later improved by Gotoh (Gotoh 1982). The SW method guarantees the optimal alignment because it explores all possible alignments between the pair of sequences.

To compute the optimal alignment of two sequences $q = q_1q_2q_3 \dots q_m$ and $d = d_1d_2d_3 \dots d_n$, SW fills a matrix H which keeps track of the degree of similarity between the two sequences compared. The matrix is computed according to the recurrence relations defined as follows:

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

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

[*] Khronos Groups. OpenCL: https://www.khronos.org/opencl
[†] OSWALD is available online at https://github.com/enzorucci/OSWALD

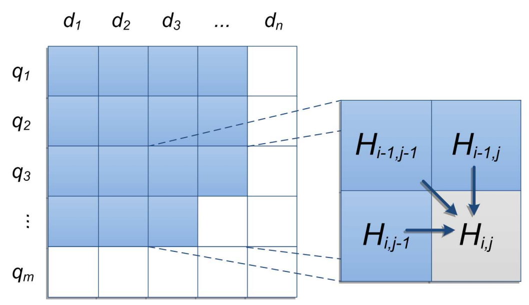

Figure 1. Data dependences in the alignment matrix *H*.

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

where $H_{i,j}$ represents the score for aligning the prefixes 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* residues of q and the first j residues of d, respectively. SM is the *substitution matrix* which defines the substitution scores for all residue pairs. Generally SM rewards with a positive value when q_i and d_j are identical or relatives, 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 $1 \leq j \leq n$, after initializing H, E and F with 0 when i = 0 or j = 0. The maximal alignment score in the matrix H is the optimal local alignment score S.

It is important to remark that any cell of the matrix H has a dependency on three cells: the one to the left, the one above and the one from the upper left diagonal, as illustrated in Figure 1. So computation must advance from top to bottom and from left to right.

OpenCL Extension on Altera's FPGA

OpenCL is a host-device-based framework for parallel implementation working across heterogeneous platforms. The language is based on the C programming language and contains extensions that allow for the specification of parallelism. Nowadays, it is supported by most hardware devices, such as CPUs, GPUs, DSPs, and FPGAs, among others. These devices (acting as coprocessors or accelerators) may have different instruction set architectures and may share memory with the host processor. OpenCL programming interfaces consider the heterogeneity between the host CPU and all connected devices.

The host-device model administers the following issues:

1. The use of different contexts for specifically available accelerators.
2. The management of memory transfers, controlling memory allocations.
3. The compilation of OpenCL codes and kernel cores to be executed on target devices.
4. The launch of the kernels on target devices, querying execution progress, and checking for errors produced.

An OpenCL kernel is the basic unit of parallel code that can be executed on a target device. OpenCL organizes a program workload into *work-groups* and *work-items*. *Work-items* are grouped into a *work-group*, which are executed independently with respect to other *work-groups*. Data-level parallelism is regularly exploited by means of the SIMD philosophy, where several *work-items* are grouped according to the lane width capabilities of the target device.

The OpenCL memory model deals with 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 but high capacity. Local memory is a shared read-write memory accessible from all *work-items* of a single *work-group*, it usually 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 it is only accessible by a single *work-item*.

Since OpenCL is a cross platform standard for parallel programming, (oriented to heterogeneity between the host and connected devices, as mentioned above), the developer can thus focus on behavioural algorithmic specifications, avoiding implementation details. On the one hand, the OpenCL specification, thus, defines a platform, memory and programming model which permits many add-ons that are vendor specific, cross-vendor and from the Khronos consortium. There is considerable freedom in terms of implementing the platform providing the final implementation satisfies the OpenCL specifications Altera Corporation (2014). On the other hand, FPGAs present programmable arrays containing logic elements, memory blocks and specific DSP blocks. This fact allows the design of dynamic custom instruction pipelines against the fixed data-path architectures of CPUs, DSPs and GPUs. The hardware Description Languages (HDLs) such as VHDL or Verilog used to develop and verify FPGA designs are complex, error prone and affected by an extra abstraction layer as they contain the additional concept of timing.

The main advantage of FPGA-based implementations using the OpenCL paradigm is the shorter time to market and faster developments in comparison with traditional FPGA developments using HDLs. FPGAs are dedicated co-processor accelerators that contain a complex hierarchy memory model (see Table 1, particularized for the FPGA used in this research). The host processor is connected to the accelerators through a peripheral interface such as a PCIe.

Table 1. 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

The OpenCL Altera (FPGA vendor) SDK supports the 1.0 specification, which is a subset of the latest current 2.1 profile (March 2015) with some flexible requirements and advanced features. As an example of these extensions, we can point to the advantage of using I/O channels and kernel channels by means of pipes Khronos Group (2014), which appeared in OpenCL 2.0. Altera's channel extension allows the transfer of data between work-item's in the same kernel or between different kernels by means of a FIFO buffer. This fact makes it possible to pass data to another work-group without additional synchronization and without host interaction.

Each Altera FPGA can have multiple in-order command queues to be executed independently and concurrently. Kernels are compiled previously and after that are passed to create the OpenCL program object at runtime. 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* up to *NDRange*. Kernels are executed across a global domain of *work-items* where *work-items* are subsequently grouped into local *work-groups*. The execution model does not specify the *work-item* execution order.

SW Implementation

In this section we will address the programming aspects and optimizations applied to our implementations on FPGA accelerated platforms. First, we present a heterogeneous implementation where alignments are carried out on a single FPGA. Then, this implementation is extended to support more than one FPGA. The final implementation concurrently exploits both host computing and FPGA devices. The algorithms comprise three stages:

1. Pre-processing stage: the reference database is preprocessed to adapt sequence data for parallel processing on multiple devices.
2. SW stage: after preprocessing the database, alignments among query sequences and database sequences are carried out.
3. Sorting stage: finally all alignment scores are sorted in descending order.

It is important to remark that stages 1 and 3 are executed on the host in all the implementations developed. Stage 2 is offloaded to the FPGA(s) and partially computed on the host in the hybrid version.

Parallelization scheme

Alignments are computed following the inter-task parallelization scheme, which takes advantage of the null data dependency between different alignments. 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 on the target platform. For this reason, database sequences are processed in groups and the size of the groups is determined by the number of SIMD vector lanes. On the host, database sequences are grouped according to the vector processing unit's (VPU) lane size. On the FPGA, it is possible to configure the number of sequences that are processed simultaneously. This aspect depends on the resources available on the FPGA.

Database preprocessing

Database sequences are sorted by their lengths in ascending order before being grouped and padded with dummy symbols. This is done to favor memory pattern access and minimize imbalances in group processing. In the FPGA implementations, the database is divided into chunks because FPGA global memory is not large and the sequence allocation space is limited. Moreover, the number of chunks should be a multiple of the number of FPGAs, and should also have the same size in order to improve workload balance between accelerators.

We would like to point out that in the hybrid implementation, the database is split into two main parts to enable a balanced workload distribution. This strategy is described in the hybrid implementation section. The host database part is performed as a single piece while on the accelerators it is divided again into several chunks, following the approach of the FPGA implementations. To avoid repeating this process, sequence databases are preprocessed separately on the host and accelerators. The databases are read from the FASTA format [‡] and then transformed into an internal binary format which favors faster disk access.

Heterogeneous single-FPGA implementation

Algorithm 1 shows the pseudo-code for the host implementation. Memory management is performed in OpenCL by means of *clCreateBuffer* (memory allocation and initialisation), *clEnqueueWriteBuffer* and *clEnqueueReadBuffer* (memory transfer to device/host). The kernel computes the alignments between a single query and a chunk of the sequence database. Kernels are invoked through the *clEnqueueNDRangeKernel* function.

The kernel is implemented following the task parallel programming model described in the OpenCL 1.0 specification, 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 favors the use of the private low-latency memory. The inner loop is fully unrolled by the compiler to increase performance. Since the compiler can perform loop unrolling, its boundaries must be constant values, so sequences are extended in the preprocessing stage to make their lengths a multiple of fixed $BLOCK\_WIDTH$ value.

Additionally, we employed Altera OpenCL channels to efficiently transfer previously computed values in order to solve data dependences between blocks (last column H and E values are needed). The combination of these techniques is essential for the Altera OpenCL compiler to successfully generate parallel pipeline execution.

Substitution score selection Our implementation is also based on the Score Profile (SP) optimization Rognes (2011)

[‡]FASTA format description: http://blast.ncbi.nlm.nih.gov/blastcgihelp.shtml

Algorithm 1 Host pseudo-code for single-FPGA exploitation

```
1: ▷ $Q$ are the query sequences
2: ▷ $vD$ is the preprocessed sequence database
3: ▷ $SP$ are the Score Profiles
4: ▷ $SM$ is the substitution matrix
5: ▷ $S$ are the alignment scores
6: ▷ $n$ are the lengths of database sequences
7: ▷ $t_h$ is the number of host threads
8:
9: clCreateBuffer's(...) ▷ Create buffers + transfer data
10: for $c \leq get\_num\_chunks(vD)$ do
11:     $SP_c = build\_SPs(vD_c, SM, t_h)$
12:     clEnqueueWriteBuffer($SP_c$) ▷ Score Profiles to device
13:     clEnqueueWriteBuffer($n_c$) ▷ Sequence lengths to device
14:     for $q \leq get\_num\_sequences(Q)$ do
15:         clEnqueueNDRangeKernel(...) ▷ Compute alignments among query $q$ and chunk $c$
16:     end for
17:     clEnqueueReadBuffer($S_c$)
18: end for
19: $S = recompute\_if\_overflow(S, Q, vD, SM, t_h)$ ▷ Recompute alignments that overflowed
20: $S = sort(S, t_h)$ ▷ Sort all scores in descending order
```

Algorithm 2 Pseudo-code for Smith-Waterman kernel

```
1: ▷ $numSequences$ is the number of sequences
2: ▷ $q$ is the query sequence
3: ▷ $m$ is the query length
4:
5: __kernel void $SW\_kernel$ ( $numSequences$, $n$, $SP$, $q$, $m$, $S$ ) {
6: for $s \leq numSequences$ do
7:     $numBlocks = n[s]/BLOCK\_WIDTH$
8:     for $k \leq numBlocks$ do
9:         for $i \leq m$ do ▷ each row
10:            if $k \neq 0$ then
11:                ▷ Receive data from previous block
12:            end if
13:            #pragma unroll
14:            for $j \leq BLOCK\_WIDTH$ do
15:                ▷ Calculate current cell value
16:            end for
17:            if $k \neq numBlocks - 1$ then
18:                ▷ Send data to next block
19:            end if
20:        end for
21:    end for
22: end for
23: }
```

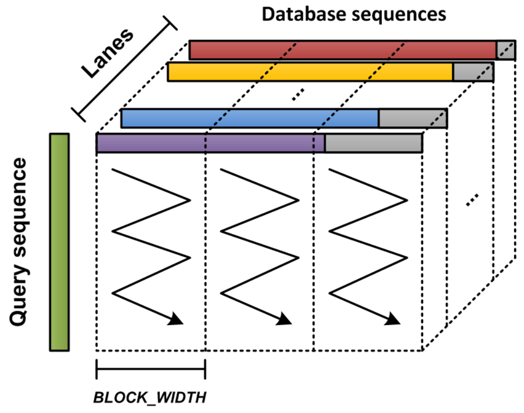

Figure 2. Schematic representation of our OpenCL kernel implementation.

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 an 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 and then transferred to the FPGA.

Data type selection Optimizing FPGA area usage is critical to obtaining high performance OpenCL applications. The alignment scores do not need a wide range data representation. For this reason we explored different integer data types to compute alignments (*char*, *short* and *int*). When the data type proves to be insufficient to represent the similarity score, i.e. overflow occurs, the alignment is recalculated on the host using the next widest integer range. The host code employs SSE instructions and is based on the open-source SWIMM tool (Rucci et al. 2015b). To allow overflow detection on the host, saturated addition is used on the FPGA kernel (in particular, the *add_sat* function).

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.

Heterogeneous multi-FPGA implementation

A simple strategy for employing multiple FPGAs at the same time consists in exploiting thread level parallelism on the host. Following this approach, an OpenMP thread is generated for each accelerator and the database chunks are distributed among the threads as soon as they become idle using a *parallel for* directive. Unfortunately, this strategy is not practical because the Altera OpenCL library is not thread-safe at host level (Altera Corporation 2014). To avoid this limitation and to allow simultaneous FPGA execution, host-device data transfers are called in a non-blocking way. Because kernels can not be invoked before data transfers are completed, the *clFinish* function is used to synchronize the host and devices. Algorithm 3 shows the pseudo-code for the host implementation. The kernel code remains invariant as in the single-FPGA implementation.

Heterogeneous hybrid implementation

By exploiting thread level parallelism, we are able to take advantage of CPU and FPGA computations. Algorithm 4 shows the pseudo-code for the host implementation. The hybrid implementation is based on a nested parallel scheme:

initially two threads are requested. The threads invoke one routine each. The $SWIMM\_search$ routine creates a nested parallel region. For the CPU alignments, our code is based on the SWIMM tool once again, which is able to take advantage of multithreading and both SSE and AVX2 extensions. The $multi-FPGA\_search$ routine computes the alignments as described in the multi-FPGA implementation of Algorithm 3.

Workload distribution strategy A key to achieving a high level of performance is the workload balance between host and accelerators. Static techniques can lead to an almost *perfect* distribution. However, these techniques involve knowing some information in advance, such as hardware features related to computational capabilities, and memory hierarchy, among others. In contrast, dynamic approaches do not need any previous information, at the expense of certain performance penalization due to imbalances and/or idling. To solve this issue, a semi-dynamic technique is performed that takes advantage of both approaches: initial tester workload (some pairwise alignments) is used to estimate the performance of any device.

In fact, the query sequences and a configurable percentage of the database residues (p) are performed, then a scheduler estimates the relative computational capabilities of both the host and the accelerators. The number of database residues assigned to the FPGAs (R_F) is evaluated according to Equation 4:

$$R_F = |D| \times \frac{n_F \times GCUPS_F}{n_F \times GCUPS_F + GCUPS_h} \tag{4}$$

where $|D|$ is the total number of residues in the database, n_F is the number of FPGA devices, and $GCUPS_F$ and $GCUPS_h$ correspond to the $GCUPS$ performance achieved by the FPGAs and the host. t_h threads are employed to estimate the compute power of the host while in the FPGA case a single accelerator is used. We assume that in a system based on multi-FPGAs each one has the same features. In an environment with different FPGAs, the scheduler should assess each FPGA's capabilities in order to distributed the workload as homogeneously as possible.

Algorithm 3 Host pseudo-code for multi-FPGA implementation

```
1: ▷ $n_F$ is the number of FPGAs
2:
3: $S = multi - FPGA\_search(Q, vD, SM, t_h, n_F)$ ▷ Compute alignments in FPGAs
4: $S = recompute\_if\_overflow(S, Q, vD, SM, t_h)$ ▷ Recompute alignments that overflowed
5: $S = sort(S, t_h)$ ▷ Sort all scores in descending order
6:
7: function $multi - FPGA\_search$ $(Q, vD, SM, S, t_h, n_F)$
8:
9:   for $d \leq n_F$ do
10:      clCreateBuffer's(...) ▷ Create buffers + transfer data
11:  end for
12:  for $(i = 0; i \leq get\_num\_chunks(vD); i+ = n_F)$ do
13:      for $d \leq n_F$ do
14:          $c = i + d$
15:          $SP_c = build\_SPs(vD_c, SM, t_h)$
16:          clEnqueueWriteBuffer($SP_c$) ▷ Score Profiles to device
17:          clEnqueueWriteBuffer($n_c$) ▷ Sequence lengths to device
18:      end for
19:      $wait()$ ▷ Block until previous transferences finish
20:      for $d \leq n_F$ do
21:          $c = i + d$
22:          for $q \leq get\_num\_sequences(Q)$ do
23:              clEnqueueNDRangeKernel(...) ▷ Compute alignments among query $q$ and chunk $c$
24:          end for
25:      end for
26:      $wait()$ ▷ Block until previous kernels finish
27:      for $d \leq n_F$ do
28:          $c = i + d$
29:          clEnqueueReadBuffer($S_c$)
30:      end for
31:  end for
32:  return $S$
33: end function
```

Experimental Results

Experimental environment and tests carried out

All tests were performed on two heterogeneous architectures running CentOS (release 6.5). The first one consists of two Intel Xeon CPU E5-2670 8-core 2.60GHz CPUs (hyper-threading enabled) and 32 GB main memory while the second one has two Intel Xeon E5-2695 v3 14-core 2.30GHz CPUs (hyper-threading enabled) and 64 GB main memory. Both architectures are equipped with:

- Two Altera Stratix V GSD5 Half-Length PCIe Boards with Dual DDR3 (two banks of 4 GByte DDR3).
- A single NVIDIA Tesla K20c GPU (2496 CUDA cores) with 5GB dedicated memory and Compute Capability 3.5.
- A single 57-core Xeon Phi 3120P coprocessor card (4 hw thread per core, 228 hw threads overall) with 6GB dedicated memory.

We used 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. OpenMP threads were bound to processor threads using *scatter* affinity.

We evaluated our application by searching 20 query protein sequences against two well-known databases: Swiss-Prot (release 2013_11)[§] and Environmental NR (release 2014_11)[¶]. The Swiss-Prot database comprises 192480382 amino acid residues in 541561 sequences, 35213 being the maximum length in amino acids. The Environmental NR database consists of 1291019045 amino acid residues in 6552667 sequences with the longest one containing 7557

[§]The Swiss-Prot database is available online at http://web.expasy.org/docs/swiss-prot_guideline.html

[¶]The Environmental NR database is available online at ftp://ftp.ncbi.nih.gov/blast/db/FASTA/env_nr.gz amino acids. The queries range in length from 144 to 5478, and they were extracted from the Swiss-Prot database (accession numbers: P02232, P05013, P14942, P07327, P01008, P03435, P42357, P21177, Q38941, P27895, P07756, P04775, P19096, P28167, P0C6B8, P20930, P08519, Q7TMA5, P33450, and Q9UKN1). Moreover, BLOSUM62 was selected as the scoring matrix, and gap insertion 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 those ten executions to avoid variability.

Since this paper considers energy consumption as well as performance, we describe the measurement environment used on hosts and accelerators:

- *Host*. Intel processors provide monitoring capabilities via hardware counters, but it is not obvious how to determine power consumption in this way. To solve this issue, Intel has developed the Intel PCM[‖] (Performance Counter Monitor) to take power measurements on the Intel Xeon processor. The Intel PCM interface allows any programmer to perform an analysis of CPU resource consumption by means of hardware counters in an easy way.
- *FPGA*. The FPGA is monitored by means of Furaxa's PCI-Express extender connected to a data acquisition device. The PCI-Express extender reports the current supplied in both 12V and 3.3V PCIe power supply lines. In particular, we use Furaxa's PCIeEXT16HOT model and the current is measured with a USB' Data Acquisition (DAQ) device that is connected to an external computer. This ad-hoc environment allows FPGA power consumption monitoring with enough sampling frequency for our experiment.
- *GPU*. Modern NVIDIA GPUs have on-board sensors for querying power consumption at runtime. This information can be obtained through the use of the NVIDIA System Management Interface (*nvidia-smi* [**]) utility, which is based on the NVIDIA Management Library (NVML) and intended to help in the management and monitorization of NVIDIA GPU devices.
- *Xeon Phi*. In a similar way to NVML for NVIDIA GPUs, Intel provides power consumption information via the Intel System Management Controller (SMC) tool (Reinders and Jeffers 2014). The coprocessor features a microcontroller located on the circuit board which monitors incoming DC power and thermal sensors. In this context, a software-based power analyzer developed by Intel makes it easy to obtain coprocessor power by means of the *micsmc* utility. Moreover, the research performed in (Igual et al. 2014) also concludes that the measurements taken by means of Intel SMC are completely reliable, observing less than 1% deviation from directly measured consumption through Xeon Phi's PCI-e channel power.

We would like to point out that the experiments of the single and multi-FPGA implementations were carried out on a system based on Xeon E5-2695 v3 processors using 28 OpenMP threads. The rest of the experiments include both architectures.

With regard to databases, the experiments with the single and multi-FPGA implementations were carried out using Swiss-Prot. However, due to its limited size, Environmental NR was used to complement the experiments in the multi-FPGA implementation and to carry out a performance comparison between the hybrid CPU-FPGA version and other SW implementations. Also, this database was used to analyze performance and power trade-off. Finally, the percentage of the database used as tester to evaluate performance capabilities on host and accelerators was fixed to 1% on the system based on Intel Xeon E5-2670 and to 2% on the system based on Intel Xeon E5-2695 v3.

Performance Results

Cell updates per second (CUPS) is a commonly used performance measure in the Smith-Waterman context, because it allows removal of 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 Q and a database D, the GCUPS (billion cell updates per second) value is calculated by:

$$\frac{|Q| \times |D|}{t \times 10^9} \tag{5}$$

where $|Q|$ is the total number of residues in the query sequence, $|D|$ is the total number of residues in the database and t is the runtime in seconds (Liu et al. 2009). In this work, 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.

Performance results of the single-FPGA implementation

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

- the *scalar* version is the baseline code where non optimization is performed.
- SIMD versions employ different integer data types and exploit data level parallelism by enabling vectorization. Vectorial nomenclature refers to SIMD width; i.e. *int4* means small vectors of 4-elements, while *int8* and *int16* use 8 and 16 integer packages respectively. On the other hand, the name prefix denotes the integer data type used; i.e. *int*, *short* and *char* represent 8, 16 and 32 bit integer data types, respectively.
- regarding memory exploitation, *constant Q* and *private Q* versions refer to the use of read-only constant memory and private memory to place query sequences, respectively.

[‖]Intel Performance Counter Monitor: http://www.intel.com/software/pcm

[**]NVIDIA System Management Interface: https://developer.nvidia.com/nvidia-system-management-interface

Table 2. Performance and resource usage comparison for OpenCL kernels with different integer data type.

Kernel	Performance (GCUPS)	Resource Usage				Performance Increase	Resource Usage Decrease
		ALMs	Regs	RAM	DSPs		
int16	17.6	76%	39%	75%	1%	-	-
short16	22.7	54%	28%	48%	1%	1.29×	0-0.36×
char16	27.0	41%	24%	41%	1%	1.53×	0-0.46×

Table 3. Performance and resource usage comparison for OpenCL kernels with different SIMD width.

Kernel	Performance (GCUPS)	Resource Usage				Performance Increase	Resource Usage Increase
		ALMs	Regs	RAM	DSPs		
scalar	4.0	28%	16%	28%	1%	-	-
char4	14.2	34%	18%	31%	1%	3.56×	0-1.21×
char8	27.7	43%	22%	38%	1%	6.95×	0-1.54×
char16	47.5	60%	30%	70%	1%	11.9×	0-2.5×

Table 4. Performance and resource usage comparison for *char16* kernel with different block width.

BLOCK_WIDTH	Performance (GCUPS)	Resource Usage				Performance Increase	Resource Usage Increase
		ALMs	Regs	RAM	DSPs		
4	11.7	40%	21%	36%	1%	-	-
8	27.0	41%	24%	41%	1%	2.31×	0-1.14×
12	37.9	53%	29%	46%	1%	3.24×	0-1.38×
16	47.5	60%	30%	70%	1%	4.06×	0-1.94×
20	52.5	75%	39%	74%	1%	4.49×	0-2.06×
24	55.0	82%	41%	81%	1%	4.7×	0-2.25×
28	57.3	92%	42%	88%	1%	4.9×	0-2.44×

Table 5. Performance and resource usage comparison for *char16* kernel with different memory exploitation.

Kernel	Performance (GCUPS)	Resource Usage			
		ALMs	Regs	RAM	DSPs
char16	57.3	92%	42%	88%	1%
char16 + constant Q	57.2	92%	41%	88%	1%
char16 + private Q	58.0	92%	42%	89%	1%

Table 2 presents FPGA resource utilization and performance achieved for OpenCL kernels with different integer data types. The same $BLOCK\_WIDTH$ value was used in these experiments and was set to 8 because *int16* resource consumption did not allow a higher value. As can be observed, the best option proves to be *char16*, not only in terms of GCUPS but also considering resource usage. *char16* reports an increase of 1.53× in performance and a reduction of 0-0.46× in resource usage with respect to *int16*. Even though *int16* does not require host recomputation, alignment scores do not need a wide range data representation. Therefore, it is convenient to compute alignments using 8-bit integers on the FPGA and recompute them on the host using wider integer types when overflow occurs.

Table 3 shows FPGA resource utilization and performance achieved for OpenCL kernels with different SIMD width. Unlike Table 2 experiments, the $BLOCK\_WIDTH$ constant could be set to 16 due to a lower resource usage from these kernels. 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 *char16* version, which reports a speedup with respect to *scalar* of 11.9× at the cost of 0-2.5× increase in resource usage.

$BLOCK\_WIDTH$ constant determines the number of vertical blocks in the alignment matrices. Table 4 exhibits FPGA resource utilization and performance achieved for *char16* kernel with different block width. Larger $BLOCK\_WIDTH$ means better performance and higher resource consumption, although the performance gain falls as $BLOCK\_WIDTH$ increases. Since sequence lengths must be a multiple of the $BLOCK\_WIDTH$ constant to permit successful parallel pipeline execution, larger values imply longer sequences, and as a consequence, the overhead in alignment computing increases. The best performance achieves 57.3 GCUPS.

The impact of using constant and private memories to place query sequences is also evaluated. Table 5 shows FPGA resource utilization and the performance achieved for the kernels used in this experiment. Copying query sequences to constant memory (*char16 + constant Q*) slightly reduces performance, contrary to the expected behavior. Constant memory is optimized for high cache hit performance. Query residues are used to index the corresponding SP and one residue is accessed for each row of a processed vertical block. Because global memory incorporates extra hardware to improve long memory latencies, better performance can be obtained if query sequences are transferred directly to this memory. However, private memory usage for query sequences effectively

Algorithm 4 Host pseudo-code for hybrid heterogeneous implementation

```
1:  ▷ $p$ is the database percentage used to compute $R_F$
2:  ▷ $R_F$ is the number of database residues assigned to FPGAs
3:  ▷ $vD_p$ is the preprocessed database chunk used to estimate relative compute power
4:  ▷ $vD_h$ is the host part of the preprocessed database
5:  ▷ $vD_F$ is the accelerators part of the preprocessed database
6:
7:      $vD_p = extract(vD, p)$   ▷ Extract database chu
        estimate relative compute power
8:
9:      $[R_F, S] = estimate\_compute\_power$ $(Q, vD_p, SM, t_h)$   ▷
        Calculate $R_F$ calling $hybrid\_search$
10:
11:     $[vD_h, vD_F] = split(vD, p, R_F, n_F)$   ▷ Split preprocessed
        database
12:
13:     $S = hybrid\_search$ ($Q$, $vD_h$, $vD_F$, $SM$, $t_h$, $n$
        Compute alignment in host and FPGA(s)
14:
15:     $S = sort(S, t_h)$ ▷ Sort all scores in descending orde
16:
17:     function $hybrid\_search$ $(Q, vD_h, vD_F, SM, t_h, n_F$
18:     #pragma omp parallel num_threads(2)
19:     {
20:         #pragma omp single nowait
21:         { $S_F = multi - FPGA\_search(Q, vD_F, SM,$
        } ▷ Compute alignments in FPGA(s)
22:         #pragma omp single
23:         { $S_h = SWIMM\_search(Q, vD_h, SM, t_h)$ } ▷ Com
        pute alignments in host
24:     }
25:     $S = recompute\_if\_overflow(S_F, Q, vD_F, SM, t_h)$   ▷
        Recompute alignments that overflowed
26:     return $S$
27:     end function
```

delivers a minor performance improvement with an insignificant increase in resource consumption, as can be seen in the *char16 + private Q* implementation.

We also evaluate the impact of the query length, and Figure 3 illustrates the performance of different kernel implementations with varying query lengths. As can be seen, the *scalar* kernel hardly improves performance while vectorized kernels benefit from larger workloads. Lastly, the *char16 + private Q* version outperforms all other kernel implementations, reaching up to 52.9 GCUPS.

Performance results of the multi-FPGA implementation Table 6 shows the performance of the multi-FPGA implementation for the two databases selected when varying the number of accelerators. As can be seen, this implementation benefits from larger workloads. It is also possible to scale its performance with good workload balance when using more than one accelerator. Due to the limited size of the Swiss-Prot database, the multi-FPGA implementation achieves a speedup of 1.85× when using two

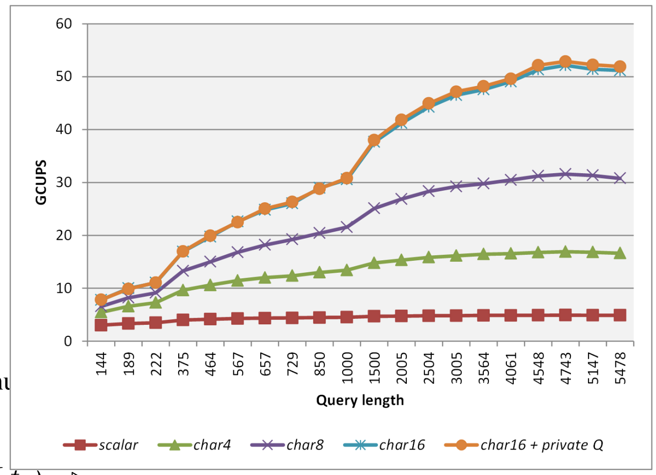

Figure 3. Performance of different OpenCL kernel implementations with queries of varying length.

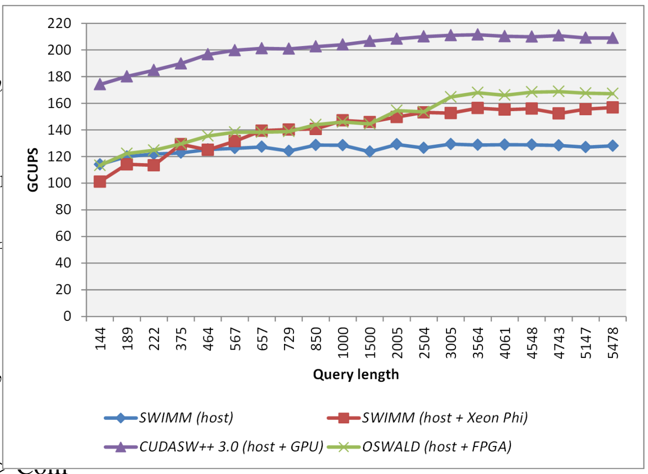

Figure 4. Performance comparison between SW implementations in system based on Intel Xeon E5-2670.

accelerators. However, the speedup goes up to 1.96× with the larger Environmental NR database.

Performance results of the hybrid implementation We have compared our hybrid version with other SW implementations on the two heterogeneous architectures used. SWIMM (v1.0.3) was selected for pure Xeon and hybrid Xeon-Xeon Phi computing (Rucci et al. 2015b). SWIMM accelerates similarity searches by exploiting multithreading and takes advantage of SSE and AVX2 extensions on the host and KNC instructions on the coprocessor. Regarding hybrid CPU-GPU computing, the fastest SW implementation on CUDA-based GPUs, CUDASW++ 3.0 (v3.1), was chosen (Liu et al. 2013). This implementation processes database sequences of short and medium length on the GPU device while long ones are carried out on the host by using the SSE instruction set as in the SWIPE approach (Rognes 2011).

Figure 4 shows the performance achieved on the heterogeneous system based on Intel Xeon E5-2670 processors. As can be observed, pure SWIMM presents an almost flat curve because of multithreading and inter-task parallelism exploitation through SSE extensions. The addition of Xeon Phi allows SWIMM to improve

Table 6. Performance of multi-FPGA implementation.

Database	FPGAs	
	1	2
Swiss-Prot	58.0	107.1
Environmental NR	58.4	114.7

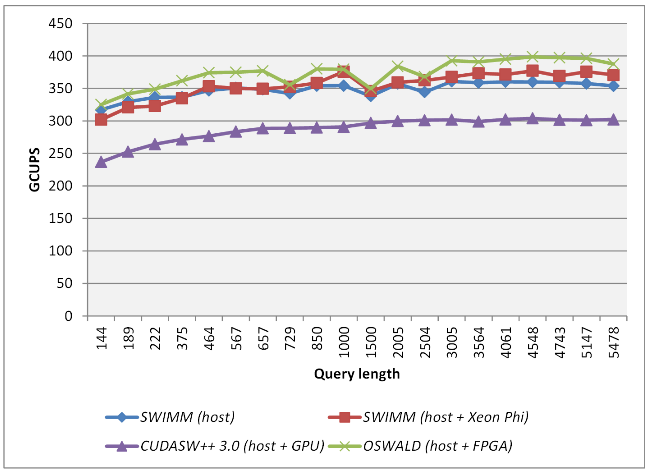

Figure 5. Performance comparison between SW implementations in system based on Intel Xeon E5-2695 v3.

its performance, except for shorter queries where this implementation is not able to take advantage of all the compute power available. The absence of low-range integers in the KNC instruction set of the Xeon Phi coprocessor is the cause of the small performance improvement. The peak performances are 129.3 and 156.7 for pure and hybrid versions, respectively. However, OSWALD's performance is always better than that of SWIMM (including for short queries), and the difference gets bigger as the query length increases. Thanks to a balanced workload distribution, OSWALD reaches up to 168.3 GCUPS. Lastly, CUDASW++ 3.0 outperforms all other implementations, achieving an impressive 210 GCUPS, principally due to NVIDIA's K20c computational power.

Figure 5 shows the performance achieved on the heterogeneous system based on Intel Xeon E5-2695 v3 processors. Unlike the other heterogeneous architecture, this system features more hardware threads and the AVX2 instruction set, which permits higher data-level parallelism. The behaviour of SWIMM is similar to the previous case. Pure SWIMM achieves a nearly flat curve, reaching 360 GCUPS. Xeon Phi incorporation decreases performance for short queries and provides little additional GCUPS for the rest. In contract to the results obtained with the previous system, CUDASW++ 3.0 presents the slowest performance. This is due to CUDASW++ 3.0 exploiting the SSE2 extension on the host and not being able to take advantage of its more powerful AVX2 counterpart. Finally, OSWALD achieves the best performance ratios, being close to 400 GCUPS peak. We would like to remark that the AVX2 exploitation and the well-balanced workload are key aspects in OSWALD's performance.

Performance and power consumption comparison

Finally, Table 7 presents a summary of the average performance and consumption achieved on the different architectures under study. As it can be seen, FPGA computing is the worst approach from performance perspective. However, it can be a good choice from the power point of view; its low power consumption (Thermal Design Power less than 25 watts) can be useful in environments with power restrictions or when power is the main concern. Also, following the same purpose, it is observed that the use of a single thread in the host is a convenient way to reduce power consumption (16%) at the cost of a smaller performance detriment (8%). In opposite sense, the use of a heterogeneous architecture based on Xeon and Xeon Phi processors is not a good option from power perspective. The incorporation of Xeon Phi coprocessor delivers low performance gain and decreases GCUPS/Watt ratio compared to host-only computing. The inability of the Xeon Phi to take advantage of low-range integer vectors prevents it from achieving better results. Moreover, the exploitation of wider vector capabilities is a key aspect to improve GCUPS/Watt ratio, as it is evidenced in the Xeon E5-2695 v3 (AVX2) compared to the Xeon E5-2670 (SSE). It is also observed that the use of hyper-threading (2 hw threads per core instead of a single thread) improves GCUPS/Watt ratio in both architectures. On the other hand, CPU combined with GPU can be a good alternative in systems that feature SSE instruction set, although AVX2 extensions are not available. In the system based on Intel Xeon E5-2670, CUDASW++ 3.0 improves GCUPS and GCUPS/Watt ratio compared to SWIMM (host-only version). However, it is not able to repeat its performance in the system based on Intel Xeon E5-2695 v3 because CUDASW++ 3.0 only exploits SSE instructions. Finally, hybrid CPU-FPGA computing stands as the best option from the performance/power point of view considering that it achieves the highest GCUPS/Watt ratio in both systems. Even more, the inclusion of an additional FPGA improves this ratio in both architectures. We would like to conclude that the use of FPGA improves significantly GCUPS/Watt ratio in both system, highlighting an improvement by 20% in the 'less' powerful system mainly due to a more homogeneous workload distribution between the host and FPGAs.

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 high computational demands scientists have developed several parallel implementations in order to reduce its response time. In addition, with the emergence of heterogeneous computing it is necessary to evaluate not only computationally scalable solutions but also

Table 7. Performance and power consumption summary.

System	Compute units	Cores	GCUPS	Power (Watt)	GCUPS/Watt
Based on Intel Xeon E5-2695 v3	Host	28	309.3	228.2	1.355
	Host	56	354.8	240	1.478
	FPGA*	1	53.5	69	0.775
	FPGA*	28	58.4	83.1	0.702
	2×FPGA*	28	114.7	169.5	0.677
	Host + Xeon Phi	56+228	450.5	380	0.843
	Host + GPU	56+2496	298.8	328.2	0.910
	Host + FPGA	56	401.1	265.6	1.510
	Host + 2 ×FPGA	56	441.6	291.2	1.516
Based on Intel Xeon E5-2670	Host	16	110.1	209.7	0.525
	Host	32	127.5	230	0.554
	Host + Xeon Phi	32+228	165.5	438.5	0.377
	Host + GPU	32+2496	206.2	303.2	0.680
	Host + FPGA	32	178.9	253.1	0.707
	Host + 2×FPGA	32	225.1	271	0.830

*Host takes part in overflow recomputation.

the energy efficiency of the system. Taking into account these considerations, this paper examines the benefits of a highly innovative technology in the form of supporting the OpenCL parallel programming model in the field of FPGAs. To the best of the authors' knowlegde, our proposal is the first high-level programming implementation on FPGAs using OpenCL with real amino acid datasets.

The main contributions of this study can be summarized as follows:

- Starting from a single-FPGA implementation, we have stressed FPGA resources to achieve a fast kernel implementation. In this sense, the exploitation of a low range integer type is a key aspect to improving performance and reducing resource usage at the same time. Data level parallelism is also critical to achieving successful performance rates at the expense of a moderate increase in resource usage. With respect to OpenCL hierarchy memory exploitation, private memory reports considerable benefits, although constant memory must be carefully studied before use. Our most successful single-FPGA implementation reaches up to 58.4 GCUPS, significantly higher than the Altera staff implementation (Settle 2014).
- We have extended the single-FPGA implementation to allow execution on multiple devices and to support concurrent host execution by means of OpenMP multithreading and SIMD computing using SSE and AVX2 extensions. Estimating the relative compute power of the host and the FPGAs to calculate SW alignments before dividing workload contributes to a well-balanced distribution. At the same time, this strategy allows us to generalize our approach to different hardware characteristics of FPGA-based platforms. Performance evaluations on two different heterogeneous architectures demonstrate that OSWALD is competitive with other top-performing SW implementations, reaching up to 442 GCUPS peak.
- Finally, we have evaluated the performance of the different implementations from an energy point of view, considering power consumption and GCUPS/Watt ratio. FPGA computing (without host concurrent execution) can be a good choice when power is the top priority. In the opposite sense, heterogeneous systems based on Xeon Phi coprocessors are not a good option for SW protein searches. The absence of low-range integer vectors on this coprocessor is the cause of its poor energy efficiency. Furthermore, taking advantage of wider vector capabilities is critical to improving the GCUPS/Watt ratio, as indicated by the Xeon E5-2695 v3 (AVX2) compared with the Xeon E5-2670 (SSE). On the other hand, GPU-based systems can lead to higher GCUPS (especially on those without AVX2 support) with acceptable GCUPS/Watt ratios. Based on our experiments, hybrid CPU-FPGA computing stands out as the best option from a performance/power perspective since it achieves the highest GCUPS/Watt ratio on both systems. Furthermore, the inclusion of an additional FPGA improves this ratio in the two architectures used.

The programming cost and the lack of portability of FPGA code have traditionally limited its applicability for SW alignments. OSWALD is a portable, completely functional and general implementation for accelerating similarity searches on FPGA-based architectures. We expect OSWALD to become an established option for accelerating SW searches in an energy-efficient way.

Acknowledgements

Enzo Rucci holds a PhD CONICET Fellowship from the Argentinian Government, and this work has been partially supported by Spanish research project TIN 2012-32180.

References

Altera Corporation (2014) Altera SDK for OpenCL Programming Guide, Version 14.0.

Altschul SF, Gish W, Miller W, Myers EW and Lipman DJ (1990) Basic local alignment search tool. *Journal of molecular biology* 215(3): 403–410.

Dydel, Stefan and Bala, Piotr (2004) Large scale protein sequence alignment using fpga reprogrammable logic devices. In: Becker J, Platzner M and Vernalde S (eds.) *Field Programmable Logic and Application*, *Lecture Notes in Computer Science*, volume 3203. Springer Berlin Heidelberg. ISBN 978-3-540-22989-6, pp. 23–32. DOI:10.1007/978-3-540-30117-2_5. URL http://dx.doi.org/10.1007/978-3-540-30117-2_5.

Farrar M (2007) Striped Smith-Waterman speeds database searches six time over other SIMD implementations. *Bioinformatics* 23 (2): 156–161.

Farrar M (2008) Optimizing Smith-Waterman for the Cell Broad-band Engine. URL http://farrar.michael.googlepages.com/SW-CellBE.pdf.

Gotoh O (1982) An improved algorithm for matching biological sequences. In: *Journal of Molecular Biology*, volume 162. pp. 705–708.

Igual FD, Jara LM, Gómez-Pérez JI, Piñuel L and Prieto-Matías M (2014) A power measurement environment for PCIe accelerators. *Computer Science - Research and Development* 30(2): 115–124. DOI:10.1007/s00450-014-0266-8. URL http://dx.doi.org/10.1007/s00450-014-0266-8.

Isa M, Benkrid K, Clayton T, Ling C and Erdogan A (2011) An fpga-based parameterised and scalable optimal solutions for pairwise biological sequence analysis. In: *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*. pp. 344–351. DOI:10.1109/AHS.2011.5963957.

Khronos Group (2014) The OpenCL Specification. version 2.0. URL https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf.

Li TI, Shum W and Truong K (2007) 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). *BMC Bioinformatics* 8:I85.

Lipman D and Pearson W (1985) Rapid and sensitive protein similarity searches. *Science* 227: 1435–1441.

Liu Y, Maskell DL and Schmidt B (2009) Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units. *BMC Research Notes* 2(1): 1–10. DOI:10.1186/1756-0500-2-73. URL http://dx.doi.org/10.1186/1756-0500-2-73.

Liu Y and Schmidt B (2014) SWAPHI: smith-waterman protein database search on xeon phi coprocessors. In: *IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, ASAP 2014, Zurich, Switzerland, June 18-20, 2014*. pp. 184–185. DOI:10.1109/ASAP.2014.6868657. URL http://dx.doi.org/10.1109/ASAP.2014.6868657.

Liu Y, Schmidt B and Maskell DL (2010) Cudasw++2.0: enhanced smith-waterman protein database search on cuda-enabled gpus based on simt and virtualized simd abstractions. *BMC Research Notes* 3(1): 1–12. DOI:10.1186/1756-0500-3-93. URL http://dx.doi.org/10.1186/1756-0500-3-93.

Liu Y, Tran TT, Lauenroth F and Schmidt B (2014) SWAPHI-LS: smith-waterman algorithm on xeon phi coprocessors for long DNA sequences. In: *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014*. pp. 257–265. DOI:10.1109/CLUSTER.2014.6968772. URL http://dx.doi.org/10.1109/CLUSTER.2014.6968772.

Liu Y, Wirawan A and Schmidt B (2013) Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC Bioinformatics* 14(1): 1–10. DOI:10.1186/1471-2105-14-117. URL http://dx.doi.org/10.1186/1471-2105-14-117.

Qiu J, Ekanayake J, Gunarathne T, Choi JY, Bae SH, Li H, Zhang B, Wu T, Ruan Y, Ekanayake S, Hughes A and Fox G (2010) Hybrid cloud and cluster computing paradigms for life science applications. *BMC Bioinformatics* 11 (Suppl12).

Reinders J and Jeffers J (2014) Front matter. In: *High Performance Parallelism Pearls*. Boston: Morgan Kaufmann. ISBN 978-0-12-802118-7, pp. i – ii.

Rognes T (2011) Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC Bioinformatics* 12(1): 1–11. DOI:10.1186/1471-2105-12-221. URL http://dx.doi.org/10.1186/1471-2105-12-221.

Rucci E, De Giusti A, Naiouf M, Botella G, Garcia C and Prieto-Matias M (2014) Smith-waterman algorithm on heterogeneous systems: A case study. In: *Cluster Computing (CLUSTER), 2014 IEEE International Conference on*. pp. 323–330. DOI:10.1109/CLUSTER.2014.6968784.

Rucci E, Garcia C, Botella G, De Giusti A, Naiouf M and Prieto-Matias M (2015a) Smith-waterman protein search with opencl on an fpga. In: *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 3. pp. 208–213. DOI:10.1109/Trustcom.2015.634.

Rucci E, García C, Botella G, Giusti AD, Naiouf MR and Prieto-Matías M (2015b) An energy-aware performance analysis of SWIMM: *S*mith-*W*aterman implementation on *I*ntel's *M*ulticore and *M*anycore architectures. *Concurrency and Computation: Practice and Experience* 27(18): 5517–5537. DOI:10.1002/cpe.3598. URL http://dx.doi.org/10.1002/cpe.3598.

Settle SO (2014) High-performance dynamic programming on fpgas with opencl. In: *IEEE High Performance Extreme Computing Conference(HPEC'13)*, volume 1.

Smith TF and Waterman MS (1981) Identification of common molecular subsequences. *Journal of Molecular Biology* 147(1): 195–197.

Weaver N, Markovskiy Y, Patel Y and Wawrzynek J (2003) 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*, FPGA '03. New York, NY, USA: ACM. ISBN 1-58113-651-X, pp. 185–194. DOI:10.1145/611817.611845. URL http://doi.acm.org/10.1145/611817.611845.

Yamaguchi Y, Tsoi H and Luk W (2011) Fpga-based smith-waterman algorithm: Analysis and novel design. In: Koch A, Krishnamurthy R, McAllister J, Woods R and El-Ghazawi T (eds.) *Reconfigurable Computing: Architectures, Tools and Applications*, *Lecture Notes in Computer Science*, volume 6578. Springer Berlin Heidelberg. ISBN 978-3-642-19474-0, pp. 181–192. DOI:10.1007/978-3-642-19475-7_20. URL http://dx.doi.org/10.1007/978-3-642-19475-7_20.