CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–21
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

An Energy-aware Performance Analysis of SWIMM: *S*mith-*W*aterman Implementation on *I*ntel's *M*ulticore and *M*anycore

Enzo Rucci[1], Carlos García[2]*, Guillermo Botella[2], Armando De Giusti[1], Marcelo Naiouf[1] and Manuel Prieto-Matías[2]

[1] *Instituto de Investigacion en Informatica LIDI, Universidad Nacional de La Plata, Buenos Aires, Argentina*
[2] *Computer Architecture Department, Complutense University of Madrid, Madrid 28040, Spain*

SUMMARY

Alignment is essential in many areas such as biological, chemical and criminal forensics. The well-known Smith-Waterman (SW) algorithm is able to retrieve the optimal local alignment with quadratic time and space complexity. There are several implementations that take advantage of computing parallelization, such as many-cores, FPGAs or GPUs, in order to reduce the alignment effort. In this research, we adapt, develop and tune the SW algorithm named SWIMM on a heterogeneous platform based on Intel's Xeon and Xeon Phi coprocessor. SWIMM is a free tool avaliable in a public git repository https://github.com/enzorucci/SWIMM. We efficiently exploit data and thread-level parallelism, reaching up to 380 GCUPS on heterogeneous architecture, 350 GCUPS for the isolated Xeon and 50 GCUPS on Xeon Phi. Despite the heterogeneous implementation obtaining the best performance, it is also the most energy demanding. In fact, we also present a trade-off analysis between performance and power consumption. The greenest configuration is based on an isolated multicore system which exploits AVX2 instruction set architecture reaching 1.5 GCUPS/Watts.
Copyright © 2010 John Wiley & Sons, Ltd.

Received ...

KEY WORDS: Bioinformatics, Smith-Waterman, HPC, Intel Xeon Phi, heterogeneous computing, power consumption.

1. INTRODUCTION

A fundamental procedure in any biological study is sequence alignment, which compares two or more biological sequences, for example those found in DNA. The purpose of this procedure is to deduce which positions within a sequence share a common evolutionary history, which equates to them being homologous. Alignment is essential in phylogenetic analysis [1], the profiling of genetic disease [2], the identification and quantification of conserved regions or functional motifs [3, 4] and ancestral sequence profiling and prediction [5]. Alignment is also important in the development of new drugs and in criminal forensics, and because of all this there is now a big research effort being made into this field of bioinformatics.

*Correspondence to: E-mail: garsanca@ucm.es
This is the peer reviewed version of the following article: [Rucci, E., Garcia, C., Botella, G., De Giusti, A., Naiouf, M., and Prieto-Matias, M. (2015) An energy-aware performance analysis of SWIMM: SmithWaterman implementation on Intel's Multicore and Manycore architectures. Concurrency Computat.: Pract. Exper., 27: 55175537. doi: 10.1002/cpe.3598.], which has been published in final form at [https://doi.org/10.1002/cpe.3598].
This article may be used for non-commercial purposes in accordance with Wiley Terms and Conditions for Use of Self-Archived Versions.

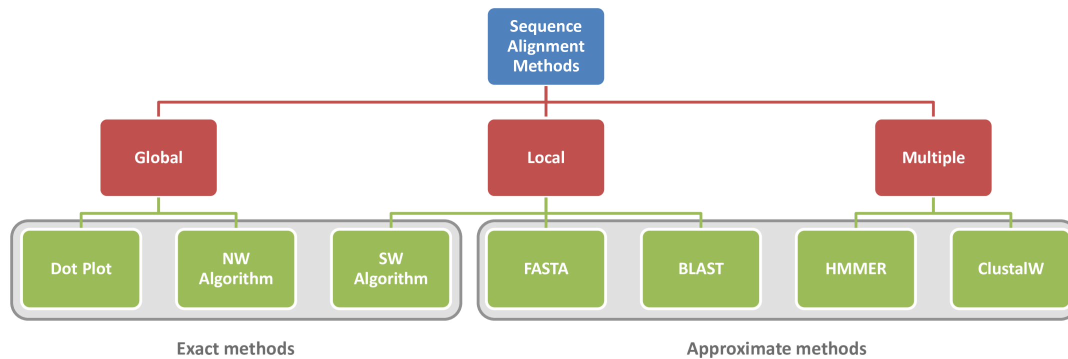

Figure 1. Sequence alignment classification.

In order to process the ever increasing quantity of data from high throughput structural genomic and genome sequencing of thousand of proteins, it is necessary to develop new computational tools that are capable of accelerating key primitives and fundamental algorithms [6].

The purpose of sequence alignment is to identify regions of similarity between two DNA, RNA or protein sequences (the query sequence and the subject or database sequence). Pairwise sequence alignment methods are classified as either global or local, where pairwise means considering only two sequences at a time.

Global methods, which include dot plot and the Needleman-Wunsch (NW) algorithm [7], aim to match as many characters as possible, from start to end, between the query sequence (q) and the database sequence (d). Both are categorized as exact methods (see Figure 1). The difference is that dot plot is based on a basic search method, whereas NW is based on Dynamic Programming (DP) [8]. Local methods, which include exact ones such as Smith-Waterman (SW) [9], as well as heuristics-based approximate methods such as FASTA [10] and Basic Local Alignment Search Tool (BLAST) [11, 12], identify short stretches of similarity between two sequences. Multiple alignment methods, such as HMMER [13] and ClustalW [14], are used to discover the similarities between a group of sequences.

Our main objective in this paper is the acceleration of the classic SW algorithm without heuristics. Almost all the applications of new sequencing technologies are based on sequence alignment [15] and SW continues to be a critical and basic primitive in many of these applications. In high-throughput sequencing, the SW algorithm is frequently used to align sequencing reads to reference sequences. Identifying the optimal alignment score using SW is however computationally expensive (linear space complexity and quadratic time complexity) since it performs an exhaustive search to find the optimal local alignment between two sequences. In spite of this, it does guarantee optimal alignment, which is essential in some applications, and SW has been the basis for many subsequent heuristic algorithms.

BLAST is one example of such heuristic algorithms, which increase speed at the cost of lower sensitivity. This algorithm keeps the position of each k-length subsequence (k-mer) of a query

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using cpeauth.cls

sequence in a hash table (k is usually 11 for a DNA sequence), with the k-mer sequence being the key, and scans the reference database sequences looking for k-mer identical matches, which are the so-called seeds. Once these seeds have been identified, BLAST performs seed extensions and joins (first without gaps), and then refines them using the classic SW algorithm once more. BLAST has been significantly improved by adding new functionalities, while maintaining the same seed-and-extend structure: some proposals have enhanced the seeding process, while others have improved the seed extension [15]. In short, accelerating SW is still a priority even though sequence alignment operations can also be speeded up using heuristic tools.

Fortunately, the alignment process exhibits inherent parallelism that can be exploited to mitigate the high cost of SW. Two well-known tools that take advantage of such parallelism for SW sequence database searches are SWIPE [16] and CUDASW++ [17]. The former focuses on CPUs with multimedia extensions such as Intel's SSE, whereas the latter targets CUDA-enabled GPUs (Graphic Processor Unit) from NVIDIA. The latest version of CUDASW++ (version 3.0) uses a hybrid implementation that is able to take advantage of both GPUs and CPUs simultaneously. More recently, Liu and Schmidt have presented SWAPHI, a highly optimized hand-tuned SW implementation for Intel Xeon Phi accelerators [18]. Moreover, these authors have also recently developed SWAPHI-LS for long DNA sequences [19]. Additionally, Wang et al. [20] have presented a novel proposal denoted XSW, which involves using an Intel Xeon Phi as coprocessor whose binary 2.0 version extends this idea to a heterogeneous architecture. There are also other proposals for SW acceleration on grid architectures [21], cloud-based systems using MapReduce [22], and even FPGA implementations [23, 24, 25].

For this paper we have used a heterogeneous Intel Xeon server equipped with an Intel Xeon Phi coprocessor. Unlike previous studies that have focused on exploiting the Xeon Phi coprocessor as far as possible using low-level optimisations [18], our aim was to evaluate a multithreaded SW code on a heterogeneous platform with SIMD extensions. Although we are able to outperform some of those previous tools, in this paper an energy consumption and performance trade-off is also considered. This paper extends the insights already offered in our previous approach [26], with the following new contributions:

- Among the main contributions is the creation of a public git repository with source code used in this paper denoted as SWIMM[†‡]. SWIMM is a software to accelerate the well-known Smith-Waterman algorithm on Intel's multicore and manycores processors. SWIMM exploits SIMD computing capabilities by means of SSE and AVX2 extensions on the Xeon and the KNC instruction set on the Xeon Phi. It also exploit the thread-level paralellism in both platforms.
- We have focused on multicore optimization by means of a lower range integer representation (8-bit and 16-bit), which enables 16 and 8 vector SIMD lanes exploitation for Intel's SSE extensions. We have extended this strategy via the exploitation of Intel's AVX2 extensions for the 256-bits (32 vector lanes) available on the latest Haswell microprocessors. To the best of the authors' knowledge, it is the first performance study in the Smith-Waterman scenario for AVX2 extensions that outperforms [§] by far the well-known SWIPE proposal.
- This idea has been extended to the Xeon Phi accelerator. However, its multimedia ISA does currently allow packaging 32-bit integer data-type. Additionally, we also make relevant comparisons with other Xeon Phi-based SW implementations, such as SWAPHI [18].
- An additional performance comparison has been made with another heterogeneous implementation, namely XSW [20].
- To enrich the discussion, we have carried out a performance comparison with the most successfully GPUs implementation known as CUDASW++ 3.1, currently the fastest SW implementation on CUDA-based GPUs.

[†]
[‡]SWIMM is available online at https://github.com/enzorucci/SWIMM
[§]CGS_NOTA: modificar frase

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

- This paper is not only focused on the performance analysis of a heterogeneous Intel Xeon server equipped with an Intel Xeon Phi coprocessor, but it also considers energy consumption. It explores different configurations in order to find the best ratio performance/power, and also also extrapolates results for new upcoming technologies available on the market during 2015.

The rest of the paper is organized as follows: Section 2 introduces the basic concepts of the Smith-Waterman algorithm. Section 3 briefly introduces Intel's Xeon Phi architecture and in Section 4 we describe our implementation of the SW algorithm. In Section 5 we discuss performance and energy results and finally in Section 6 we present the main conclusions.

2. SMITH-WATERMAN ALGORITHM

The Smith-Waterman algorithm is used to identify the optimal local alignment between two sequences. It is based on 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 the modifications of Gotoh [27] for handling multiple sized gap penalties are shown below.

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

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

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

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 of aligning the same segments of q and d but ending with a gap in q and 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. $H_{i,j}$, $E_{i,j}$ and $F_{i,j}$ are initialized with 0 when i = 0 or j = 0. The optimal local alignment score S is the maximal alignment score in the matrix H.

It is important to note that there is a strict order of computation in matrix *H* due to the data dependences inherent to this problem. To be able to calculate the value of any cell, the value of all cells to the left and above have to be computed first, as shown in Figure 2. These dependences restrict the ways in that *H* can be computed.

For the sake of clarity, Figure 3 shows the alignment matrix between sequences ADLGRT and ADLGAVF. The scoring values considered are: *SM* rewards with 5 for identical residues and punishes with -3 for different residues, and gap insertion and extension penalties are set to 10 and 2, respectively. For this particular case, the optimal local alignment score achieved is 20.

3. INTEL'S XEON PHI

The adoption of accelerators within the HPC community keeps on growing and it is expected that new designs from Intel, NVIDIA and AMD will likely dominate most production systems in the next few years. The Intel Xeon Phi (Phi) is a many-core coprocessor with the MIC (Many

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

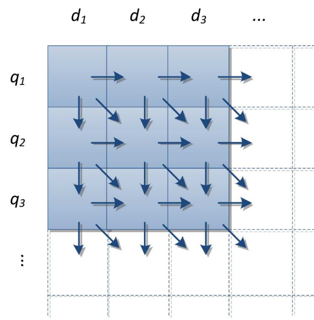

Figure 2. Data dependences in the alignment matrix H.

		A	D	L	G	A	V	F
	0	0	0	0	0	0	0	0
A	0	5	0	0	0	5	0	0
D	0	0	10	0	0	0	3	0
L	0	0	0	15	3	1	0	1
G	0	0	0	3	**20**	8	6	4
R	0	0	0	1	8	18	6	4
T	0	0	0	0	6	6	16	4

Figure 3. Alignment matrix for sequences ADLGRT and ADLGAV

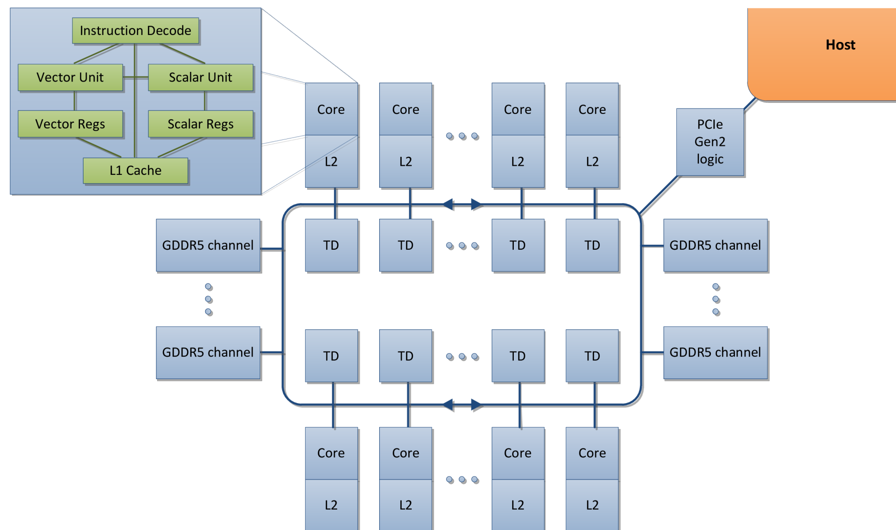

Figure 4. Xeon Phi architecture.

Integrated Cores) architecture that derived from the defunct Larrabee project [28] and the Teraflops Research Chip research project. In its current generation, the Phi features up to 61 x86 pentium cores with extended vector units (512-bit) and simultaneous multithreading (four hardware threads per core). Each core integrates an L1 cache (32 KB data + 32 KB instructions) and has an associated fully coherent L2 cache (512 KB combined data and instructions). As shown in Figure 4, a high-speed ring interconnect allows data transfer between all the L2 caches of the Phi and the memory subsystem. The Phi is able to support up to 8 memory controllers, each one with two GDDR5 channels, and is connected to the host server through a PCIe Gen2 bus.

From a programming point of view, one of the pros of this platform is the support of existing parallel programming models traditionally used on HPC systems such as the OpenMP or MPI paradigms, which simplifies code development and improves portability over other alternatives based on accelerator-specific programming languages such as CUDA or OpenCL.

Nevertheless, unlike GPUs, the Xeon Phi can be run as a completely standalone computing system, which allows running applications using solely the resources of the coprocessor. This is called the *native mode*. Building a Xeon Phi native application usually involves minimal code

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using cpeauth.cls

DOI: 10.1002/cpe

modifications. In fact, many HPC codes written for general purpose processor clusters run in this mode with ease, just recompiling them for this platform using the *-mmic* compiler flag. Nevertheless, the key to Phi performance is the efficient use of the per core vector units and the small cache. Therefore, high performance may not be achieved after direct portability migration code.

The native mode can be inefficient for applications with high input/output rates or contain mostly non parallel parts. In those cases, it is advisable to use Phi as a coprocessor device based on *offload mode*, which is the Phi's primary mode of operation. The programming model in this case is similar to other accelerators such as CUDA-enabled GPUs. The host CPU runs the sequential code of the application and invokes kernel execution on the Phi.

A code developer with expertise in the OpenMP paradigm will find this model easier to learn than other GPU-based models such as OpenCL or CUDA. Intel claims this fact to be one of the advantages over other consolidated accelerators such as GPU and FPGA. Nevertheless, the main aspects to be addressed in order to achieve high performance are still:

1. how to design the computations to efficiently map the Phi vector units.
2. how to optimally exploit the memory hierarchy, especially when handling large datasets.

Ideally, programmers would only need to introduce some directives to inform the compiler about pointer disambiguation, data alignment data dependencies and introduce minimal code modifications to allow automatic vectorization. Nevertheless, in essence, *guided auto-vectorization* is not able to achieve the best performance and programmers usually need to make an effort to hand-tune the codes by means of language intrinsics. Although intrinsics may inhibit other loop-level optimizations that improve performance, highly optimized hand-tuned codes frequently outperform their *guided* counterparts. Indeed, intrinsics are currently the only option for complex applications which suffer from data dependencies or irregular access patterns that can be hidden using specific code transformations. Unfortunately, improving performance comes at the expense of losing cross-platform portability. Most processor families, even from the same vendor, have non-compatible intrinsics which support different SIMD instruction sets. As a consequence, code developers need to write many code branches, thus increasing maintenance needs.

4. SW IMPLEMENTATION

In this section we will address the optimisations performed on Intel's Xeon and Intel's Xeon Phi platforms. Before describing them in detail, we would like to point out the algorithm flow which can be summarised in the following steps:

- *Pre-processing stage*: preprocessing of the reference database.
- *SW stage*: SW alignments.
- *Sorting stage*: sorting the alignment scores in descending order.

4.1. Parallelisation scheme

Alignments are performed as in inter-task parallelism approach [16] where several alignments are solved concurrently. This parallelisation scheme makes it possible to exploit the small-vector capabilities available on most modern microprocessors. In contrast to intra-task parallelization where the parallelisation approach is applied within a single pair of sequences, multiple database sequences are carried out simultaneously. In particular, database sequences are grouped according to a vector processing unit's (VPU) lane size. In order to balance the sequence workload between VPUs, sequences are sorted by length to be grouped as in [29] work.

4.2. Database preprocessing

In the pre-processing stage, database sequences are sorted and grouped according to the VPU width of the target platform. In particular, for the Xeon Phi and the heterogeneous hybrid implementations, the database is divided into chunks that are offloaded to the coprocessor as it becomes idle. The

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

Xeon implementation exploits a finer grain workload in order to avoid unbalance distribution that produces idle threads. Because this process must be repeated for every search, sequence databases are preprocessed separately to avoid duplicate work. The databases are read from the FASTA format[¶] and then transformed to an internal binary format which favours faster disk access.

4.3. Multiple parallelism levels

Our implementation employs multiple parallelism levels:

- **Data-level parallelism.** As mentioned before, SIMD instructions are supported by both Intel's Xeon and Xeon Phi through the use of guided compilation or hand-tuned codification with intrinsic instructions. In this work we have explored two SIMD exploitation approaches:
  - **Guided vectorization** through the use of preprocessor directives. From OpenMP version 4.0, there is a specific directive to express data parallelism: *#pragma omp simd* enforces vectorized loop. One of the most important advantages in this type of vectorization is portability. Although lane size can differ on Intel's Xeon and Xeon Phi, the compiler can generate two different versions depending on the target platform.
  - **Intrinsic vectorization** employing the SSE and AVX extensions for the Xeon and the KNC instruction set for the Xeon Phi.
- **Thread-level parallelism.** To exploit parallelism across multiple cores, we have implemented a multi-threaded version of the SW algorithm based on the OpenMP programming model available on both the CPU and the accelerator device.

Algorithm 1 shows the pseudo-code of our SW implementation using guided-vectorization for the Xeon Phi. CPU thread-level parallelism is exploited using the OpenMP programming model: one thread is generated for each coprocessor used. The main loop that iterates over chunks of database sequences is distributed across threads with the *#omp for* pragma using a *dynamic* scheduling policy. Each CPU thread offloads the assigned chunk of database to the corresponding coprocessor to compute alignments and then waits for its completion. Inside the Xeon Phi, alignments are solved in parallel using multi-threading again. The iterations of the parallel loop are distributed using a *dynamic* scheduling policy. Despite sorting the reference database, *static* scheduling does not perform well, as has been indicated in previous research [18]. When using *guided* scheduling, performance tends to be lower. To alleviate the overhead associated with buffer allocation and deallocation, all thread buffers are pre-allocated before the SW stage and then reused in the subsequent offloads. This initial offload is used at the same time to communicate common data to all threads, like the queries and the substitution matrix. Finally, when all chunks have been processed, all the alignment scores are sorted in descending order to complete the sorting stage.

It is important to remark that the guided vectorization code for the Intel Xeon is essentially the same for the Phi, except that *#pragma offload* directives are not included. As mentioned in Section 4.2, preprocessed database is kept as one single chunk, so the code for the Intel Xeon is represented by one invocation of the SW_SEARCH function.

Regarding intrinsic vectorization implementations, both Xeon and Xeon Phi codes are structurally equal to their guided vectorization counterparts, except that *#pragma omp simd* directives are not included and implementations of SW_CORE function are composed of intrinsic instructions. Figure 5 shows the different intrinsic implementations of SW_CORE. In all cases, the alignment matrix is processed row by row. $vCur$ is the matrix cell being calculated while $vPrev$ is the previous one. vH keeps the previously processed row and its values are replaced by the current row as they are no longer needed. $vSub$ represents the substitution scores for the database sequence residues against the query residue. vE and vF are the score vectors for alignments ending in a gap in the query and the database sequence, respectively. $vGoe$ represents the vector for the sum of gap open and gap extension penalties while vGe is the vector for gap extension penalty. Last, vS keeps the current optimal alignment score.

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

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

Algorithm 1 SW_Phi(Q, vD, SM, G_o, G_e, n_mics, th_{mic})

```
1:  ▷ Q are the query sequences
2:  ▷ vD is the sequences database
3:  ▷ n_mics is the number of Xeon Phis
4:  ▷ th_mic is the number of Xeon Phi threads
5:
6:  #pragma omp parallel num_threads(n_mics)
7:  {
8:          #pragma offload target(mic:mic_no) in(Q,SM,G_o,G_e)
9:          { pre-allocate buffers }
10:         #pragma omp for schedule(dynamic)
11:         for c ≤ get_num_chunks(vD) do
12:                 #pragma offload target(mic:mic_no) in(vD_c) out(S_c)
13:                 {S_c = SW_SEARCH(Q, vD_c, SM, G_o, G_e, th_mic)}
14:         end for
15:         #pragma offload target(mic:mic_no)
16:         { de-allocate buffers }
17: }
18: scores = sort(S) ▷ in descending order
19:
20: function SW_SEARCH(Q, vD_c, SM, G_o, G_e, th)
21:
22:     #pragma omp parallel for num_threads(th) schedule(dynamic)
23:     for k ≤ |Q| * |vD_c| do
24:         S_k = 0
25:         q = get_query_sequence(Q, k)
26:         vd = get_database_sequence(vD_c, k)
27:         if score_profile then
28:             P = build_score_profile(vd, SM)
29:         end if
30:         for i ≤ |q| do
31:             if query_profile then
32:                 P = build_query_profile(q_i, vd, SM)
33:             end if
34:             #pragma omp simd
35:             for j ≤ |vd| do
36:                 [H_{i,j}, E_{i,j}, F_{i,j}] = SW_CORE(H, P, E, F, G_o, G_e)
37:                 S_k = get_max_value(S_k, H_{i,j}) ▷ save similarity scores
38:             end for
39:         end for
40:     end for
41:     return S
42: end function
```

4.4. Substitution scores

Our code also implements other well-known optimisations of the SW algorithm that have been proposed in previous works [30, 17], such as the Query Profile (*QP*) and Score Profile (*SP*) optimisations.

- The *QP* strategy is based on creating an auxiliary two-dimensional array of size $|q| \times |E|$, where q is the query sequence and E is the alphabet. Each row of this array contains the scores of the corresponding query residue against each possible residue in the alphabet. Since

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

SSE	AVX2	KNC

vCur = _mm_adds_epi8(vH[j-1], vSub);
vCur = _mm_max_epi8(vCur, vF[i]);
vCur = _mm_max_epi8(vCur, vE[j]);
vCur = _mm_max_epi8(vCur, vZero);
vS = _mm_max_epi8(vS, vCur);
vF[i] = _mm_subs_epi8(vF[i], vGe);
vE[j] = _mm_subs_epi8(vE[j], vGe);
vAux = _mm_subs_epi8(vCur, vGoe);
vF[i] = _mm_max_epi8(vF[i], vAux);
vE[j] = _mm_max_epi8(vE[j], vAux);
vH[j-1] = vPrev;
vPrev = vCur;
	vCur = _mm256_adds_epi8(vH[j-1], vSub);
	vCur = _mm256_max_epi8(vCur, vF[i]);
	vCur = _mm256_max_epi8(vCur, vE[j]);
	vCur = _mm256_max_epi8(vCur, vZero);
	vS = _mm256_max_epi8(vS, vCur);
	vF[i] = _mm256_subs_epi8(vF[i], vGe);
	vE[j] = _mm256_subs_epi8(vE[j], vGe);
	vAux = _mm256_subs_epi8(vCur, vGoe);
	vF[i] = _mm256_max_epi8(vF[i], vAux);
	vE[j] = _mm256_max_epi8(vE[j], vAux);
	vH[j-1] = vPrev;
	vPrev = vCur;
		vCur = _mm512_add_epi32(vH[j-1], vSub);
		vCur = _mm512_max_epi32(vCur, vF[i]);
		vCur = _mm512_max_epi32(vCur, vE[j]);
		vCur = _mm512_max_epi32(vCur, vZero);
		vS = _mm512_max_epi32(vS, vCur);
		vF[i] = _mm512_sub_epi32(vF[i], vGe);
		vE[j] = _mm512_sub_epi32(vE[j], vGe);
		vAux = _mm512_sub_epi32(vCur, vGoe);
		vF[i] = _mm512_max_epi32(vF[i], vAux);
		vE[j] = _mm512_max_epi32(vE[j], vAux);
		vH[j-1] = vPrev;
		vPrev = vCur;

Figure 5. Intrinsic implementations of the SW_CORE function.

each thread compares the same query residue against different ones from the database, this optimisation improves data locality. As a consequence it also increases memory requirements, but this increase is negligible because the size of the database is usually much larger than the size of the alphabet.

- The *SP* technique is based on constructing an auxiliary $n \times L$ score array, where n is the length of the database sequence and L is the number of vector lanes. Since each row of the score profile forms an L-lane score vector, an advantage is that its values can be gathered using a single vector load. However, the score profile must be re-built for each database sequence, so its suitability must be evaluated, especially for short queries.

4.5. Integer range selection

Most of alignment scores can be represented using an narrow integer representation. In the case of Intel's Xeon, alignments are computed using 8-bit integer operations. Meanwhile SSE2 instruction set allows to pack 16 element of 8-bit in a single SIMD register, AVX2 extensions doubles upto 32 elements. Additions are performed using saturated arithmetic operations to permit overflow detection. When an overflow is detected (the alignment score is equal to the maximum value of the integer representation employed), the alignment is recalculated using the next wider integer range. Nevertheless, Phi does only support 32-bit integer, so it cannot compute more than 16 alignments at the same time. This handicap could be solved in the next MIC generation with the incorporation of an Intel Atom processor with vector capabilities, denoted as AVX-512[‖], which is planned to be available in 2015.

4.6. Data locality

Data locality is one key element to achieve high performance, especially on the Phi, where blocking is also necessary to reduce the number of cache misses [31]. Furthermore, data structures have also been aligned to avoid the overhead of misaligned memory accesses (64-byte aligned for the Phi and 32-byte aligned for the Xeon).

4.7. Heterogeneous hybrid implementation

Using the code for both processors, we have developed a heterogeneous hybrid version which takes advantage of both Intel Xeon and Xeon Phi simultaneously. As shown in Algorithm 2, we just need to introduce an additional conditional statement to select the chunk to be processed. The implementation is based on a nested parallel scheme: initially $n_mics + 1$ threads are requested (n_mics corresponds to the number of accelerators) that invoke the routine SW_SEARCH which creates a nested parallel region (a single thread is cloned as many times as number of cores of the target platform). The n_mics threads are in charge of offloading their database chunks to their

[‖] AVX-512 Extensions: https://software.intel.com/en-us/blogs/additional-avx-512-instructions

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

corresponding coprocessor and solve their alignments, while the last CPU thread only solves the alignments of its own database chunk.

Algorithm 2 SW_het(Q, vD, SM, G_o, G_e, n_mics, th_{mic}, th_{cpu})

```
1:  ▷ th_cpu is the number of second level Xeon threads
2:
3:  #pragma omp parallel num_threads(n_mics + 1)
4:  {
5:      if mic_thread then
6:          #pragma offload target(mic:mic_no) in(Q,SM,G_o,G_e)
7:          { pre-allocate buffers }
8:      end if
9:      #pragma omp for schedule(dynamic)
10:     for c ≤ get_num_chunks(vD) do
11:         if mic_thread then
12:             #pragma offload target(mic:mic_no) in(vD_c) out(S_c)
13:             {S_c = SW_SEARCH(Q, vD_c, SM, G_o, G_e, th_mic)} ▷ MIC execution
14:         else
15:             {S_c = SW_SEARCH(Q, vD_c, SM, G_o, G_e, th_cpu)} ▷ host execution
16:         end if
17:     end for
18:     if mic_thread then
19:         #pragma offload target(mic:mic_no)
20:         {de-allocate buffers }
21:     end if
22: }
23: scores = sort(S)▷ in descending order
```

5. EXPERIMENTAL RESULTS

5.1. Experimental environment

All tests have been performed on two heterogeneous architectures running CentOS 6.5:

- The first one is equipped with:
  - Two Intel Xeon E5-2670 8-core 2.60GHz CPUs with hyper-threading enabled and 32 GB main memory.
  - A single 57-core Xeon Phi 3120P coprocessor card (4 hw thread per core, 228 hw threads overall) with 6GB dedicated memory.
- The second one is equipped with:
  - Two Intel Xeon E5-2695 v3 14-core 2.30GHz CPUs with hyper-threading enabled and 64 GB main memory.
  - 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 have used Intel's ICC compiler (version 14.0.2.144) with the *-O3* optimization level by default. Auto-vectorization has been enabled with the *-vec* compiler flag. OpenMP threads were binded to processor cores using *scatter* affinity.

The experiments used to assess performance are similar to those in previous work [16, 29, 32]. We have evaluated our application by searching 20 query protein sequences against two well-known

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

databases: Swiss-Prot (release 2013_11)** and Environmental NR (release 2014_11)††. The Swiss-Prot database consists of 192480382 amino acid residues in 541561 with the longest sequence containing 35213 amino acids. The Environmental NR database comprises 1291019045 amino acid residues in 6552667 sequences, 7557 being the maximum length. The queries have been 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), ranging in length from 144 to 5478. The scoring matrix selected was BLOSUM62, and gap insertion and extension penalties were set to 10 and 2, respectively. Performance results are in GCUPS.

As mentioned before, this paper considers energy consumption besides performance. We describe the measurement environment used on hosts and accelerators:

- On Xeon side, Intel has developed Intel PCM‡‡ (Performance Counter Monitor) to take power measurements on the Intel Xeon processor. We chose PCM because since version 2.0 it is supported by the Xeon E5 processors used in this work. In particular, Intel processors already incorporate monitoring capabilities via hardware counters, but it was not obvious to determine power consumption in this way. The Intel PCM interface allows any programmer to perform an analysis of CPU resource consumption by means of hardware counters in an easy way.
- On Xeon Phi side, this coprocessor already provides power consumption information via the Intel SMC (System Management Controller) tool. The SMC tool [33] accesses 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. Furthermore, the paper [34] also concludes that the measurements taken by means of Intel SMC are completely reliable, observing less than 1% deviation from directly measuring consumption through Xeon Phi's PCI-e channel power.
- On GPU side, NVIDIA has presented the NVIDIA System Management Interface (*nvidia-smi* *) utility based on top of the NVIDIA Management Library (NVML) and intended to help in the management and monitoring of NVIDIA GPU devices. Latest NVIDIA GPUs, have on-board sensors for querying power consumption at runtime, and this information can be obtained through the use of the *nvidia-smi* utility.

We would like to point out that the experiments of sections 5.2 and 5.3 were carried out using the Swiss-Prot database. However, due to its limited size, the experiments in the heterogeneous system were performed using the larger Environmental NR database in order to carry out analyses related with GPU performance comparison and the effect of varying workload distribution on performance and consumption (sections from 5.4 to 5.7).

Regarding to the system used, subsections 5.2, 5.3, 5.4 and 5.6 are performed on the first system based on Xeon E5-2670 (except the performance analysis of AVX2 intrinsic). Sections 5.5 and 5.7 are performed on the second one equipped with more powerful Intel Xeon E5-2695 v3 processor.

5.2. *Performance results on the Intel Xeon*

Figure 6 shows the performance on the system based on Intel Xeon E5-2670 for the different approaches under evaluation with increasing number of OpenMP threads. Without enabling vectorization (denoted as *no-vec* in the figure), our implementation hardly improves performance from 0.5 to 3 GCUPS. Automatic-vectorization does not only improve performance significantly (*simd* label in Figure 6), but it also allows our codes to scale with the number of threads. The hand-tuned codes based on SSE2 and SSE3 intrinsics (denoted as *intrinsic*) outperform their guided vectorization counterparts. The performance gap between the two approaches is noticeable (around

**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
‡‡Intel Performance Counter Monitor: http://www.intel.com/software/pcm
*NVIDIA System Management Interface: https://developer.nvidia.com/nvidia-system-management-interface

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

2×). Overall, the Score Profile (denoted as *SP*) performs better than the Query Profile (denoted as *QP*). *SP* achieves almost linear speedup, reaching 113.7 GCUPS with 32 OpenMP threads.

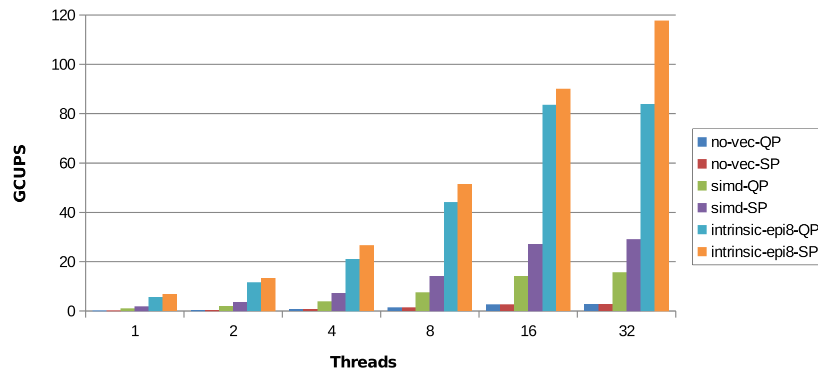

Figure 6. Scalability on the system based on Intel Xeon E5-2670.

We would like to highlight a big difference with respect to previous work [26], in which 32-bit floating data type and AVX intrinsic vectorization were used. The SW algorithm does not require a wide range data representation. In this paper, the SW implementation uses a smaller data type as signed integer representation with 8, 16 or 32 bits. This fact allows a greater level of data parallelism exploitation, although with smaller SSE intrinsic width, 128-bits in SSE instead of 256-bits in AVX. In particular, the integer SSE intrinsic could pack 16 elements of 8-bit integers into a vector, 8 elements with 16-bit range or 4 integers for 32-bit. Figure 7 shows the gain achieved using different data type ranges on the Intel Xeon when varying the number of threads. As expected, the use of smaller data types increases performance rates significantly, from 29.9 and 57.5 GCUPS for 32-bits and 16-bits version (denoted as *epi32* and *epi16* respectively) to 113.7 GCUPS on the 8-bit implementation (bar labeled as *epi8*).

The versions using 8-bit and 16-bit score ranges also consider overflow detection, as mentioned in section 4.5. Saturation arithmetic is used in order to check overflow for a single $H_{i,j}$ score. When the saturation value is detected, the alignment is recalculated using the next widest range. It has been observed that the additional overhead related with alignment re-computation using larger data types is negligible. Performance is only affected by vector size. In fact, using the Swiss-Prot database, overflow occurs in only 0.5% of alignments for 8-bit representation and none in 16-bits. Moreover, similar overflow rates have also been observed in the Environmental NR database (0.75% for 8-bits and none for 16-bits), so it is worth using 8-bit data in order to exploit data parallelism as much as possible.

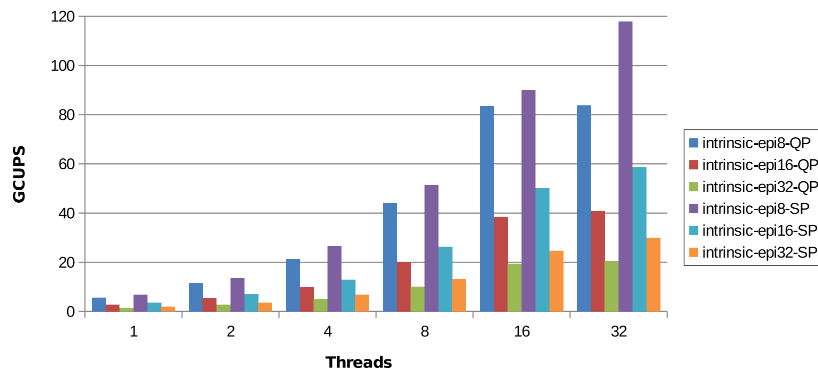

Figure 7. Scalability achieved when varying data type range.

Figure 8 illustrates the performance with queries of varying length using 32 OpenMP threads. Most implementations do not experience large variations in behavior since our approaches exploit

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

inter-task parallelism. However, we observe a gradual drop in performance for shorter queries, which is more noticeable for the hand-tuned versions. Although *SP* calculates the substitution scores for all residues in the database sequence, is more efficient respect to memory access than the *QP* variant with longer query sequences. Memory access is the key to achieving satisfactory performance rates.

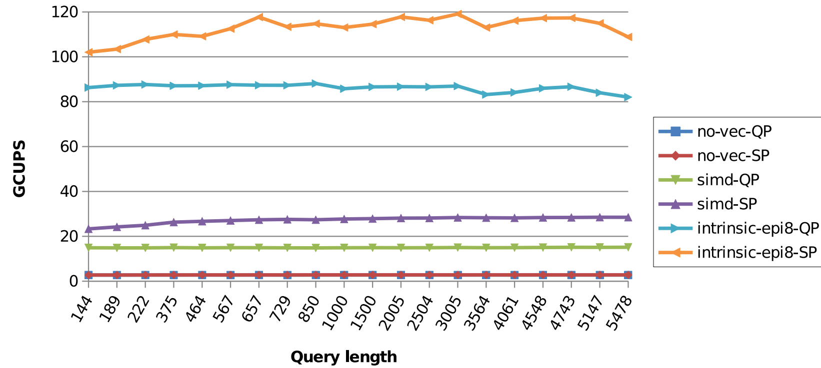

Figure 8. Performance on the Intel Xeon with queries of varying length.

Because data level exploitation is critical to achieving satisfactory performance, we have carried out a performance study using AVX2 in order to take advantage of 256-bit vector capabilities with integer arithmetic. AVX2 instructions are available on Intel's latest Haswell microprocessor. We would like to emphasize that according to the authors' knowledge, SWIMM is the first SW implementation using AVX2 extensions. Figure 9 compares the performance rates achieved with the AVX2 instruction set in comparison with the well-known SWIPE code on a system based on Intel Xeon E5-2695 v3. Both substitution scores schemes with AVX2 outperform SWIPE, but it is *SP* who is far better than this optimised SW version, achieving speed-ups of upto $1.4\times$. Running at full system, the *SP* scheme reaches the impressive performance of 342.3 GCUPS. Moreover, the acceleration observed with the use of AVX2 extensions with respect to SSE2 is about $2\times$, as is to be expected due to the double width vector size in AVX2, so one can expect to find even greater performance ratios in systems with wider vector capabilities, as in the announced AVX-512[†], which will be available in the next Xeon generation, codenamed Skylake.

[†]AVX-512 Extensions: https://software.intel.com/en-us/blogs/additional-avx-512-instructions

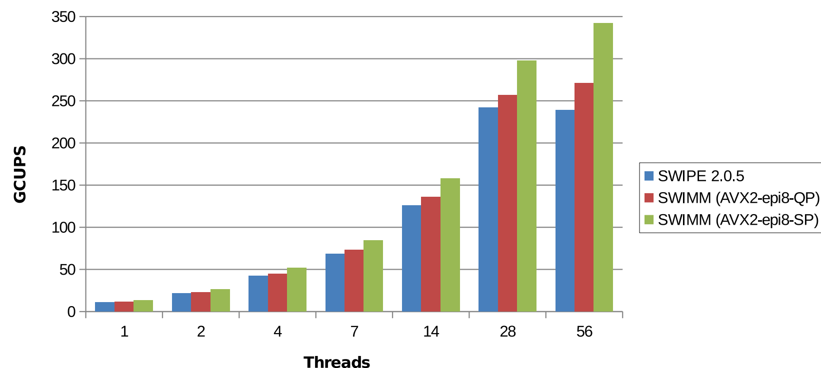

Figure 9. SWIMM performance on the Intel Xeon E5-2695 v3 using AVX2 vector capabilities in comparison with SWIPE version.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using cpeauth.cls

5.3. Performance results on the Intel Xeon Phi

Figure 10 shows the performance on the Intel Xeon Phi for the different approaches when varying the number of threads (from 57 to 228 hardware threads). Phi's implementation involves a significant difference with respect to Xeon. The current accelerator incorporates vector capabilities for integer, but accepted data is limited to 32-bit integers. We can state that this fact radically limits performance, as has been demonstrated in previous analyses where the exploitation of data-level parallelism is paramount.

Again, without enabling vectorization, our implementation hardly improves performance. As with Intel's Xeon system, the non-vectorized versions barely exhibit performance differences. Both guided vectorization implementations labeled *simd* present similar behavior, achieving a maximum of 12.3 and 14.3 GCUPS for *QP* and *SP*, respectively. The hand-tuned codes based on MIC intrinsics also outperform their guided vectorization counterparts: 38.8 and 43.9 for *QP* and *SP* approaches. However, in this case, the performance gap between the two approaches is slightly lower (an average of $1.1\times$). Indeed, for query lengths shorter than 375 residues, *QP* outperforms *SP*. A similar behavior has been observed in previous research for the Xeon Phi [18]. We also observe that the hand-tuned *SP* achieves the best performance (43.9 GCUPS with 228 threads), scaling relatively well with the number of hardware threads. We would like to remark that the use of integer arithmetic instead of floating-point arithmetic [26] leads to an average acceleration of $1.4\times$, although performance would be significantly better with more SIMD capacity.

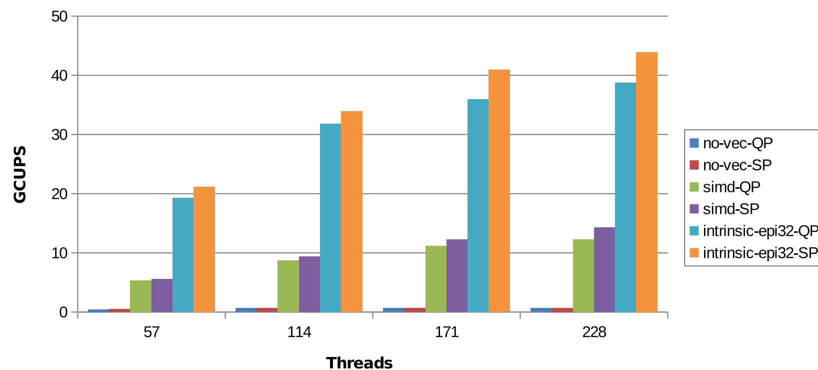

Figure 10. Scalability on the Intel Xeon Phi.

Figure 11 illustrates the performance with queries of varying length using 228 hardware threads on the Phi. There is a noticeable drop in performance with queries shorter than 567 residues. This phenomenon can be explained by the additional overhead incurred by the previous construction of the *SP*, which does not compensate for the indexation benefits in shorter queries. In fact, the *QP* scheme is more efficient on Phi because the ISA incorporates a single multimedia permutation operation, while on the Xeon it is conducted with several shuffle instructions. For this reason, and to avoid the divergences observed depending on the query length, we developed an adaptive implementation that combines *SP* and *QP* approaches depending on query size (denoted in figure as *AP*).

As has been mentioned above, SIMD exploitation is the key to achieving satisfactory throughput. Although 8-bit packing data is not available on Phi yet, its incorporation is expected for the next Phi's generation, known as Knights Landing, by means of AVX-512 extensions. Software emulation could be developed, packing two 16-bit integers into a single 32-bit integer, but the masking operations necessary to maintain algorithm coherence and the software overflow checking do not provide any improvement.

Figure12 shows a performance comparison of the binary SWAPHI [18] with our implementation. As shown, our *AP* proposal is competitive for medium and large query sizes. As the SWAPHI source code is not yet available, we cannot determine any reason for the performance drop for small

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

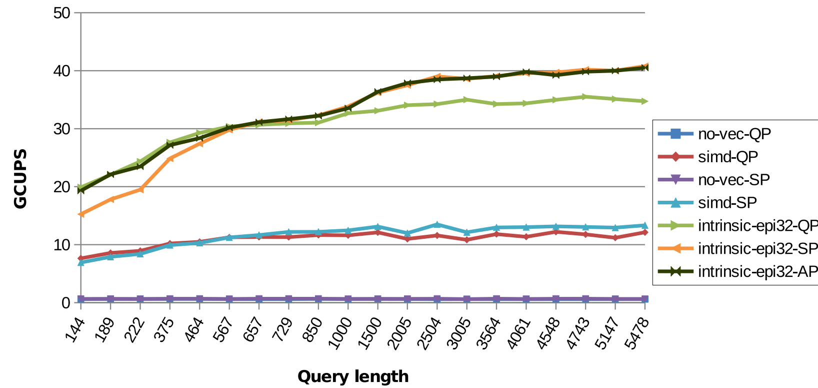

Figure 11. Performance on the Intel Xeon Phi with queries of varying length.

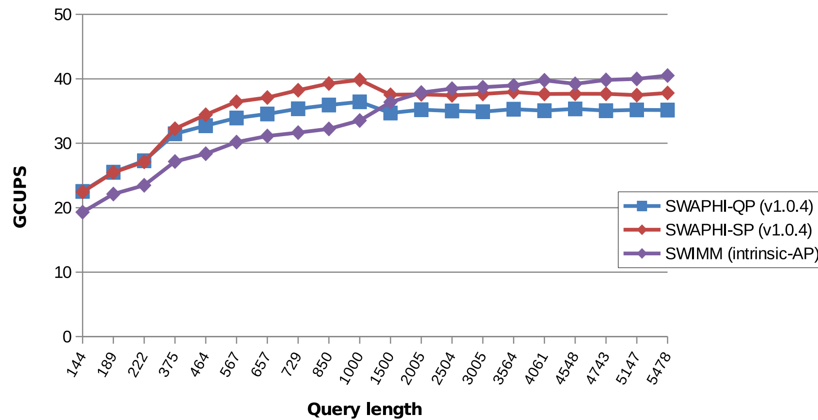

Figure 12. Performance comparison between our SWIMM implementation and SWAPHI on Intel Xeon Phi.

queries. However, better performance is expected for the future Knights Landing architecture, in which many more SIMD options will be available.

Finally, Figure 13 analyzes the impact of the blocking optimizations. We have chosen the most successful configuration in terms of performance in both systems under study: 32 hardware threads + *SP* approach on Xeon and 228 hardware threads with *AP* scheme on Phi. The blocking technique achieves a consistent improvement of $1.7\times$ in performance rates, even for the shorter query lengths on the Phi. However, blocking benefits are much lower on Xeon (an average of $1.1\times$), since the Xeon has a larger cache memory than the Phi.

5.4. Performance results of the hybrid implementation

In this section we analyze the behavior of a hybrid implementation using the Xeon and Phi simultaneously by means of static workload distribution. The key to achieving better performance is the workload balance between the two processors. For that reason, we have adapted the hybrid implementation to support a simple static distribution of database sequences. Figure 14 analyses workload distribution using our best hand-tuned implementations on both processors: *SP* on the Xeon and *AP* on the Phi. The abscissa focuses on the percentage of the sequence pairs that are aligned on the Xeon, while the rest are aligned on the Phi. The maximum performance is achieved with a static workload distribution of 75% on the Xeon and 25% on the Phi. Although a relatively simple workload distribution scheme is used, the overall performance is upto 160 GCUPS. We would like to emphasize that this figure almost corresponds to the ideal sum of the individual performances achieved on the Xeon and Phi (117.8 and 41.9 GCUPS, respectively). Therefore, the additional overhead caused by a static sequence distribution is almost negligible.

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using cpeauth.cls

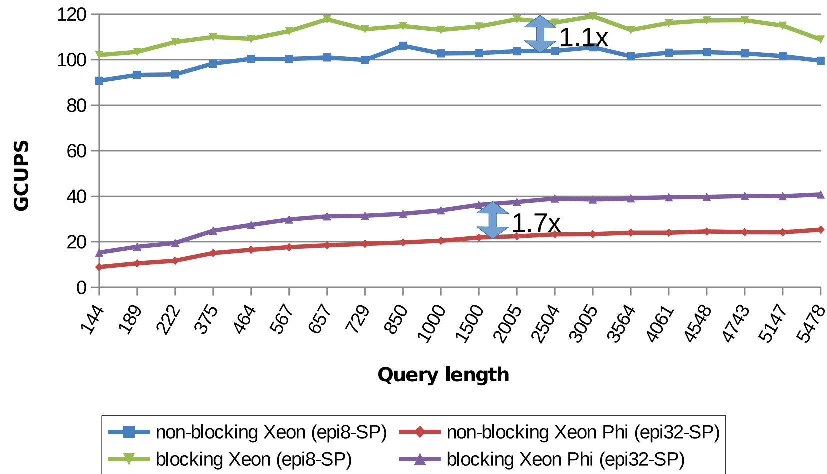

Figure 13. Performance impact of the blocking optimizations with queries of varying length.

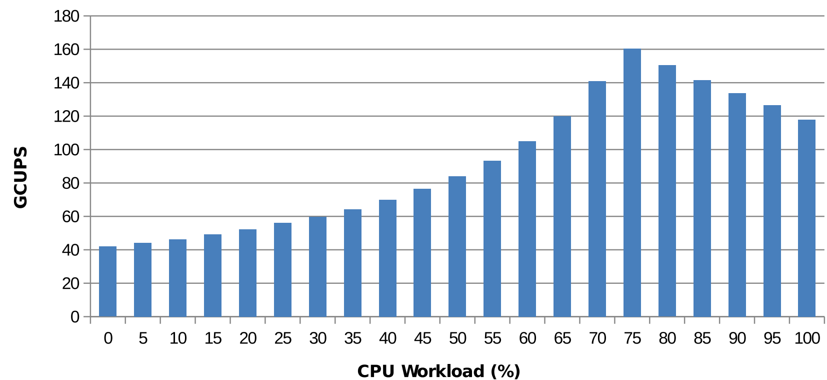

Figure 14. Performance of the hybrid implementation when varying the percentage of sequence pairs that are aligned on the Xeon.

Figure 15 illustrates a performance comparison of our hybrid approach respect to XSW v2.0 [‡] implementation. As in the previous experiment, *SP* and *AP* schemes were used on the Xeon and the Phi, respectively, but in this case workload distribution is based on the approach described in section 4.7. Although version XSW v1.0 [20] develops a SW implementation on Xeon Phi in native mode, there exists a binary v2.0 that extends it to a heterogeneous system. Both experiments were carried out using all available computational resources: 32 threads on Xeon E5-2670 and 228 on Phi. As shown, our SWIMM implementation improves [§]by far XSW implementation for all queries length considered.

5.5. *Performance comparison with other SW implementation*

This subsection comparatively analyses the performance achieved by other SW implementations in comparison with our SWIMM proposal. Regarding to the multicore exploitation, it is chosen SWIPE [16] implementation already used in previous subsection. Regarding heterogeneous computing, this comparative study includes CUDASW++ v3.0 [17] and XSW [20]. Whereas CUDASW++ version 3.1 performs medium and short queries into a GPU card, long queries are

[‡]XSW v2.0 is available online at http://sdu-hpcl.github.io/XSW/
[§]CGS_NOTA: mejorar estilo

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using cpeauth.cls

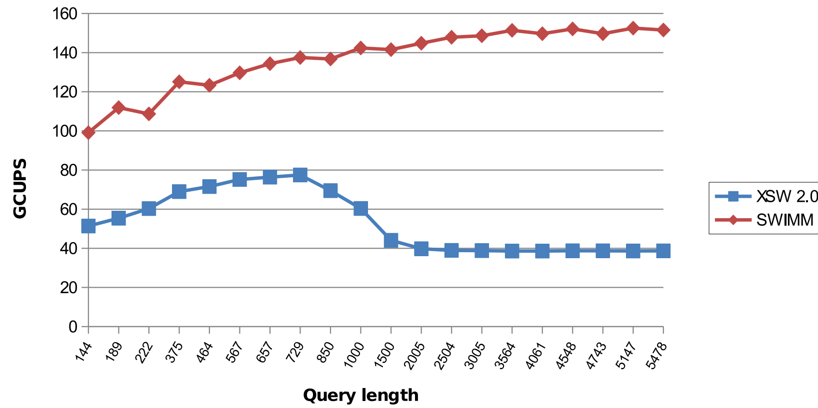

Figure 15. Performance comparison between our SWIMM implementation and XSW varying queries length.

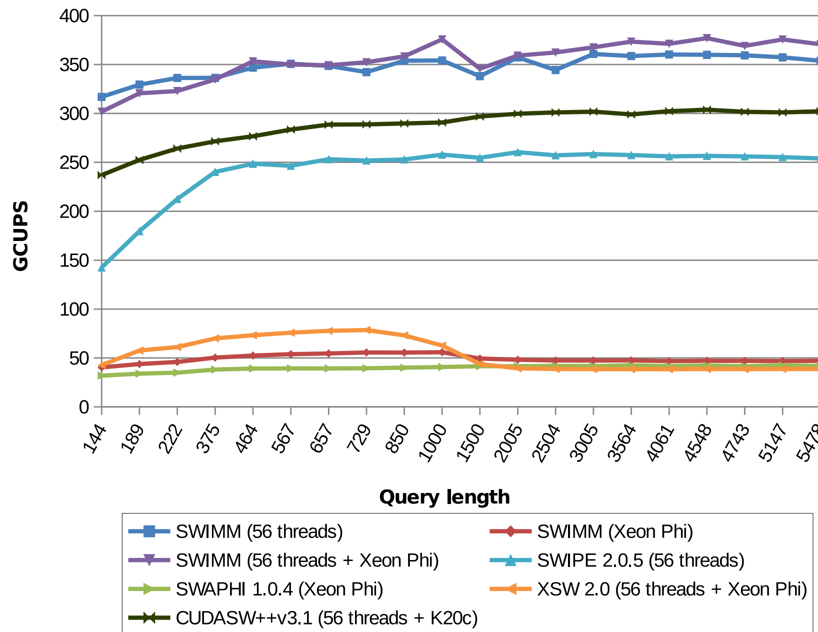

Figure 16. Performance comparison between SW implementations varying queries length.

carried out in the host by using the SSE instruction set as in SWIPE approach. Finally, XSW uses a coprocessor based on Xeon Phi as it was described in previous subsections.

Figure 16 shows the performance achieved in a heterogeneous system equipped by Intel Xeon E5-2695 v3 as host and two devices based on NVIDIA-K20c card and Xeon Phi 3120P coprocessor. As was expected, SWIMM implementation achieved the best performance ratios (300-380 GCUPS), although most of the acceleration comes from the data-level parallelism exploitation by means of AVX2 (315-360 GCUPS). We would like to note that although the use of heterogeneous computing expects to report better performance rates, the huge GCUPS difference between the Xeon and the Phi supposes a serious unbalance workload (CPU-threads are idle) which is translated into a significant performance detriment.

Moreover, while CUDASW++3.1 achieves an important thoughtput upto 300 GCPUS, SWIPE gets a peak of 220 GCUPS and XSW reports poor performance ratios. It also observed the limited performance on Phi coprocessor, just reaching a 50 GCUPS peak.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe

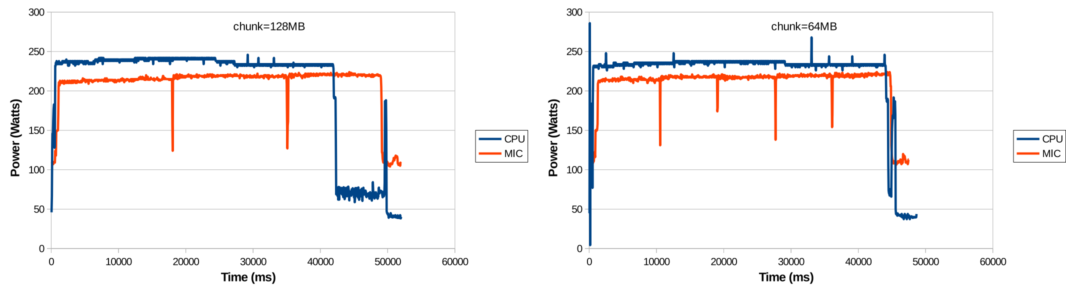

Figure 17. Consumption profile of hybrid SW implementation for 128MB and 64MB chunks.

5.6. *Power results of the hybrid implementation*

Understanding the behaviour of an architecture not only from the angle of performance but also considering the energy consumption has begun to be a necessary task in the heterogeneous computing field. In this section we focus on the energy consumption aspect for the SW algorithm on the systems used in this paper. In the previous section, it was observed that workload balance between host and accelerator is the key to achieving good performance in a heterogeneous architecture. Following this premise, the consumption detailed for the heterogeneous system when searching the largest query sequence against the Environmental NR database is shown in Figure 17. The algorithm scheduler distributes the workload between host and coprocessor. In particular, database chunks are sent to each device as soon as it becomes free. Chunk size determines the workload balance, as is shown in this figure (sizes of 64 MB and 128 MB). Lower granularity (smaller chunks) means better balancing, which means less execution time. This fact is evident in the last consumption valley on the host when it is waiting until the accelerator job is completed.

In addition, Figure 17 clearly shows several differentiated execution phases on the Xeon Phi. Coprocessor consumption falls are related to the in/out data transfers involved in any chunk database load to be processed. This aspect is not perceptible in the host because there is no job submission as such, and one can observe a maintained consumption over time. Finally, the last peak consumption on the host is due to the final sort stage once all the alignments have been processed.

Figure 18 shows the consumption profile measured in Joules when varying workload distribution between the two processors. Blue bars correspond to the power consumption on the Xeon, while red ones correspond to Phi. As expected, the consumption profile varies substantially with respect to the performance observed in Figure 14. The low performance rates achieved on the Phi (in term of GCUPS) severely penalize its consumption. In fact, the most reasonable configuration is the one that disables the accelerator, when one can observe not only the minimum energy consumption but also the best ratio GCUPS/Watts. In particular, a configuration of 100% on the host, which means the coprocessor is disabled, achieves a ratio of 0.52 GCUPS/Watts, while the best configuration for the heterogeneous architecture reaches only 0.368 GCUPS/Watts (75% workload is performed on the host).

5.7. *Performance and power summary*

Finally, Table I shows a summary of the average performance and consumption achieved on the different architectures studied. As mentioned above, the use of a heterogeneous architecture based on Xeon and Xeon Phi processors is not the best approach in terms of consumption, as is reflected by the worst GCUPS/Watts ratio (Xeon Phi column). However, although this kind of hybrid system is not appropriate nowadays for aligning sequences using the SW algorithm, this is expected to change in the next Knights Landing architecture. This poor performance (in terms of GCUPS) is due to the absence on the Phi of low-range vector capabilities, which will be corrected in the next generation with the addition of the AVX-512 extensions. In fact, this aspect is key to improving the GCUPS/Watts ratio, as it is evidenced on the Xeon E5-2695. Higher performance is derived from the exploitation of wider AVX2 vectors, which stands out as the best system in terms of

Copyright © 2010 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls
DOI: 10.1002/cpe

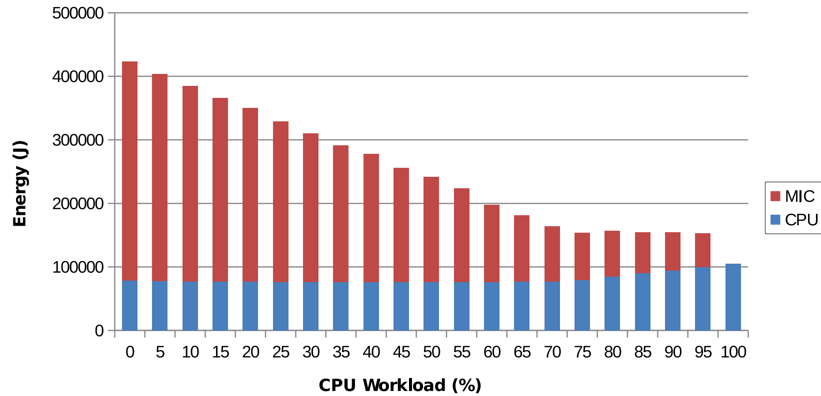

Figure 18. Overall consumption in hybrid implementation.

GCUPS/Watts, even better than the hybrid CPU-GPU case. The gain in performance by the GPU incorporation does not compensate the increase in energy consumption. Furthermore, it is observed that the exploitation of hyper-threading (2 hw threads per core instead of a single thread) is not only beneficial in performance terms with an extra 15% GCUPS in the AVX2 version, but also in terms of consumption, with only an 5% power increment.

Table I. Performance and consumption rates summary.

System	Based on Intel E5-2670			Based on Intel E5-2695 v3			
	2×Xeon	Xeon Phi 3120P	2×Xeon + Phi	2×Xeon	1×Xeon	2×Xeon + Phi	2×Xeon + K20c
Cores	32	228	32 +228	56	28	56 + 228	56 + 2496
Power (Watts)	228.2	268.2	436.5	240	228.2	450.5	328.2
GCUPS	117.8	41.9	160	354.8	309.3	380	298.8
GCUPS/Watts	**0.516**	**0.156**	**0.367**	**1.479**	**1.356**	**0.844**	**0.910**

6. CONCLUSIONS

The SW algorithm performs an exact local sequence alignment. Nevertheless, in practice, several parallel implementations are used due to its computational complexity. Moreover, Intel has recently designed a coprocessor known as Xeon Phi for HPC. Among its main advantages is the ease of programming, and it is largely compatible with Intel Xeon codes. To obtain successful performance rates on both devices, two levels of parallelism exploitation are needed: thread-level parallelism by means of OpenMP and data-level parallelism using SIMD instructions.

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

- From a performance perspective:
  - ¶SIMD exploitation is crucial to achieving competitive performance rates. Guided vectorization hardly offers performance improvements in comparison with hand-tuned vectorization. Therefore, the smallest data type (8-bits) is used in order to exploit as much data level parallelism as possible. In contrast, the Xeon E5-2670 running with 32 threads obtained up to 118 GCUPS using SSE (16×8bit lane vectors), and the Xeon E5-2695 with 56 threads reaches up to 355 GCUPS using AVX2 (32×8bit lane vectors). However, the current Xeon Phi includes only 32-bit integer vector capabilities,

¶CGS_nota: indicar que es con la base de datos swissprot

Copyright © 2010 John Wiley & Sons, Ltd.
Prepared using cpeauth.cls

DOI: 10.1002/cpe

which limits the performance to 44 GCUPS for 228 threads. Intel has announced the incorporation of AVX-512 that supports 8-bit lane vectors for the second half of 2015. A notable performance increment is expected by using our methodology. To the best of the authors' knowledge, our implementation is the first SW implementation using AVX2, outperforming the well-known SWIPE by upto $1.4\times$.
  - In the Xeon, the *SP* strategy outperforms its *QP* counterpart. In the Phi, the leading scheme depends on the query length. The *AP* scheme solves this problem taking the best of both approaches.
  - Regarding scalability, our approach achieves an efficiency of 82% on the Xeon E5-2670 processor with 16 threads and of 80% on the Xeon E5-5695 with 28 threads, which drops to 32% in the Xeon Phi with 228 threads.
  - Our Phi implementation outperforms the current SWAPHI for medium and large query sizes.
  - The impact of blocking optimizations is more significant on Xeon Phi, as the cache size is small.

- From a heterogeneous perspective:
  - Huge GCUPS difference between the Xeon and the Phi can lead to serious unbalanced workload, which translates into significant performance detriment. The workload balance is critical in order to achieve successful performance rates. The most successful configuration achieves up-to 380 GCUPS.
  - Our approach also outperforms the XSW implementation.
- From a power consumption perspective:
  - Considering energetic efficiency as GCUPS/Watt, heterogeneous computing based on Xeon and Xeon Phi processors is not the best choice. In fact, according to our power measurements, is worth disabling Xeon Phi due to its poor performance in its current architectural version (Knights Corner). It is basically caused by non-supporting vector capabilities for narrow data types.
  - Because GPUs provide low range integer operations, heterogeneous computing using this kind of accelerators stands as a better option, achieving in this case 0.91 GCUPS/Watts.
  - Finally, the most efficient configuration is on the Xeon E5-2695, since it has AVX2 extensions. It obtains 1.479 GCUPS/Watts.

ACKNOWLEDGMENTS

Enzo Rucci holds a PhD CONICET Fellowship from the Argentinian Government. This work has been partially supported by the Spanish research project TIN 2012-32180 and the CAPAP-H4 network (TIN2011-15734-E).

REFERENCES

1. Felsenstein J. *Inferring phylogenies*. Sinauer Associates, 2003.
2. Miller MP, Parker JD, Rissing SW, Kumar S. Quantifying the intragenic distribution of human disease mutations. *Annals of Human Genetics* 2003; **67**(6):567–579.
3. Thomas JW, Touchman JW, Blakesley RW, Bouffard GG, Beckstrom-Sternberg SM, Margulies EH, Blanchette M, Siepel AC, Thomas PJ, McDowell JC, *et al.*. Comparative analyses of multi-species sequences from targeted genomic regions. *Nature* 2003; **424**(6950):788–93.
4. Kirkness EF, Bafna V, Halpern AL, Levy S, Remington K, Rusch DB, Delcher AL, Pop M, Wang W, Fraser CM, *et al.*. The Dog Genome: Survey Sequencing and Comparative Analysis. *Science* Sep 2003; **301**(5641):1898–1903.
5. Hall BG. Simple and accurate estimation of ancestral protein sequences. *Proceedings of the National Academy of Sciences of the United States of America* Apr 2006; **103**(14):5431–5436.
6. Higgs P, Attwood T. *Bioinformatics and Molecular Evolution*. Wiley-Blackwell, 2005.
7. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* Mar 1970; **48**(3):443–453.

Copyright © 2010 John Wiley & Sons, Ltd.

8. Giegerich R. A systematic approach to dynamic programming in bioinformatics. *Bioinformatics* 2000; **16**(8):665–677.
9. Smith TF, Waterman MS. Identification of common molecular subsequences. *Journal of Molecular Biology* March 1981; **147**(1):195–197.
10. Lipman D, Pearson W. Rapid and sensitive protein similarity searches. *Science* 1985; **227**:1435–1441.
11. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. *Journal of molecular biology* Oct 1990; **215**(3):403–410.
12. Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res* September 1997; **25**(17):3389–3402.
13. Eddy SR. Profile hidden markov models. *Bioinform* 1998; **14**:755–763.
14. Thompson J, Higgins D, Gibson T. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res* 1994; **22**:4673–4680.
15. Li H, Homer N. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics* 2010; **11**(5):473–483.
16. Rognes T. Faster Smith-Waterman database searches with inter-sequence SIMD parallelization. *BMC Bioinformatics* 2011; **12:221**.
17. Liu Y, Wirawan A, Schmidt B. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. 2013.
18. Liu Y, Schmidt B. Swaphi: Smith-waterman protein database search on xeon phi coprocessors. *25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2014)*, 2014.
19. Liu Y, Tran TT, Felix L, Schmidt B. SWAPHI-LS: Smith-waterman Algorithm on Xeon Phi Coprocessors for Long DNA Sequences. *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER 2014)*, 2014.
20. Wang L, Chan Y, Duan X, Lan H, Meng X, Liu W. XSW: Accelerating Biological Database Search on Xeon Phi. *Fourth International Workshop on Accelerators and Hybrid Exascale Systems 2014*, 2014.
21. Chichizola F, Naiouf M, Giusti LD, Rodriguez I, Giusti AD. Overhead Analysis in Parallel Processing DNA Sequences on Grid Architectures. *Proceedings of the LAGrid08 (2nd International Latin American Grid Workshop 2008)*, 2008.
22. Qiu J, Ekanayake J, Gunarathne T, Choi JY, Bae SH, Li H, Zhang B, Wu T, Ruan Y, Ekanayake S, *et al.*. Hybrid cloud and cluster computing paradigms for life science applications. *BMC Bioinformatics* 2010; **11 (Suppl12)**.
23. Yamaguchi Y, Tsoi KH, Luk W. *ARC*, 0001 AK, Krishnamurthy R, McAllister J, Woods R, El-Ghazawi TA (eds.), Springer; 181–192.
24. Yu CW, Kwong KH, Lee KH, Leong PHW. A smith-waterman systolic cell. *In Proceedings of the 13th International Workshop on Field Programmable Logic and Applications FPL 2003*, Springer, 2003; 375–384.
25. Li TI, Shum W, Truong K. 60-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA). *BMC Bioinformatics* 2007; **8:I85**.
26. Rucci E, Giusti AD, Marcelo, Botella G, Garcia C, Prieto-Matias M. Smith-Waterman Algorithm on Heterogeneous Systems: A Case Study. *Proceedings of the IEEE Cluster 2014*, 2014.
27. Gotoh O. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, vol. 162, 1981; 705–708.
28. Seiler L, Carmean D, Sprangle E, Forsyth T, Dubey P, Junkins S, Lake A, Cavin R, Espasa R, Grochowski E, *et al.*. Larrabee: A many-core x86 architecture for visual computing. *IEEE Micro* 2009; **29**(1):10–21.
29. Liu Y, Maskell DL, Schmidt B. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes* 2009; **2:73**.
30. Farrar M. Striped Smith-Waterman speeds database searches six time over other SIMD implementations. *Bioinformatics* 2007; **23 (2)**:156–161.
31. Rucci E. Computación eficiente del alineamiento de secuencias de adn sobre cluster de multicores. Master's Thesis, Universidad Nacional de La Plata, Argentina. Available online at: http://hdl.handle.net/10915/27737 2013.
32. Liu Y, Huang W, Johnson J, Vaidya S. GPU Accelerated Smith-Waterman. *Lecture Notes in Computer Science* 2006; **3994**:188–195.
33. Front matter. *High Performance Parallelism Pearls*, Jeffers JR (ed.). Morgan Kaufmann: Boston, 2015; i – ii.
34. Igual FD, Jara LM, Gomez-Perez JI, Piuel L, Prieto-Matias M. A power measurement environment for pcie accelerators. *Computer Science - Research and Development* 2014; :1–10.

Copyright © 2010 John Wiley & Sons, Ltd.

DOI: 10.1002/cpe