Noname manuscript No.
(will be inserted by the editor)

SWIMM 2.0: enhanced Smith-Waterman on Intel's Multicore and Manycore architectures based on AVX-512 vector extensions

Enzo Rucci · Carlos Garcia Sanchez · Guillermo Botella Juan · Armando De Giusti · Marcelo Naiouf · Manuel Prieto-Matias

Received: date / Accepted: date

Abstract The well-known Smith-Waterman (SW) algorithm is the most commonly used method for local sequence alignments, but its acceptance is limited by the computational requirements for large protein databases. Although the acceleration of SW has already been studied on many parallel platforms, there are hardly any studies which take advantage of the latest Intel architectures based on AVX-512 vector extensions. This SIMD set is currently supported by Intel's Knights Landing (KNL) accelerator and Intel's Skylake (SKL) general purpose processors. In this paper, we present an SW version that is optimized for both architectures: the renowned SWIMM 2.0. The novelty of this vector instruction set requires the revision of previous programming and optimization techniques. SWIMM 2.0 is based on a massive multi-threading and SIMD exploitation. It is competitive in terms of performance compared with other state-of-the-art implementations, reaching 511 GCUPS on a single KNL node and 734 GCUPS on a server equipped with a dual SKL processor. Moreover, these successful performance rates make SWIMM 2.0 the most efficient energy footprint implementation in this study achieving 2.94 GCUPS/Watts on the SKL processor.

Keywords Bioinformatics · Smith-Waterman · Xeon-Phi · Intel-KNL · SIMD · Intel-AVX512

Enzo Rucci, Armando De Giusti
II-LIDI, CONICET, Universidad Nacional de La Plata, Argentina
E-mail: {erucci,degiusti}@lidi.info.unlp.edu.ar

Carlos Garcia, Guillermo Botella, Manuel Prieto-Matias
Universidad Complutense de Madrid, Spain
E-mail: {garsanca, gbotella, mpmatias}@ucm.es

Marcelo Naiouf
III-LIDI, Universidad Nacional de La Plata, Argentina
E-mail: mnaiouf@lidi.info.unlp.edu.ar

The final publication is available at `https://doi.org/10.1007/s10766-018-0585-7`

1 Introduction

Disciplines such as genomics and proteomics face the challenge of processing a massive amount of data. Sequencing centers, analytical research facilities and individual laboratories are experiencing the *data explosion* phenomenon: an exponential growth in the amount of biological data produced [1]. In fact, next-generation sequencing technologies used in biomedical studies or medical practice to identify genetic variants associated with diseases involve more and more data processing. The use of computers to catalog and retrieve information requires both sophisticated hardware and complex software developments to make this task feasible.

The development of bioinformatics entails the collaboration of antagonistic actors such as professionals in the life sciences and computer experts in order to provide new tools. One of the most important challenges facing the scientific community is to perform genomic data analysis in a reasonable time, so a deep knowledge of hardware optimization and acceleration is required in most cases. A common operation in bioinformatics is to identify the similarities between two sequences. The Smith-Waterman (SW) algorithm is usually applied because of its high sensitivity. SW is based on dynamic programming and guarantees an optimal solution for a pairwise local alignment. However, SW's weakness resides in its high complexity: the alignment process has a quadratic order depending on the sequence lengths. In fact, the acceptance of SW has been limited by its prohibitively high execution times and the computational resources required. Although heuristics (such as BLAST [2] and FASTA [3]) are suitable in certain contexts because they are considerably faster, they fail due to loss of sensitivity [4].

SW can be employed to search for similar regions between two long DNA sequences or protein sequences. In both scenarios, a score matrix must be built in order to determine the best alignment. Furthermore, the matrix size depends on the sequence lengths that, at the same time, influence the parallel scalability. For long DNA sequences, data parallelism is performed across the independent matrix anti-diagonal wavefront by means of the *intra-task* scheme. In the context of proteins, a single sequence is usually compared with a sequence database, which means a large number of pairwise alignments to be computed. As protein sequences are much shorter than DNA sequences, multiple pairwise alignments can be evaluated simultaneously by using the *inter-task* scheme. It should be noted that the *intra-task* scheme can be applied to protein alignment (Farrar's approach is the fastest [5]); however, the *inter-task* scheme generally produces better performance results in this case.

In the last few years, the feasibility of using parallel computational devices to improve performance has received considerable attention in bioinformatics. In the context of SW protein alignment, the exploitation of SIMD (Single Instruction Multiple Data) capabilities on modern CPUs has been widely studied [6]. Among the proposals, we can highlight the fastest SSE-based tool *SWIPE* [7] and its evolution into AVX2 extensions *libssa* [8]. Also, the *Parasail* library [9], whose strength is the abstraction of the SIMD instruction set.

With the emergence of Graphic Processor Units (GPUs), we find the most successful software, which is known as *CUDASW++*, and its improved versions [10,11]. Additionally, for Intel's modern co-processors based on the Xeon Phi architecture, we can mention the optimized hand-tuned SW implementation denoted as *SWAPHI* [12] and its hybrid extension *LSBDS* [13]. In a similar way, the *SWIMM* [14] software is able to exploit CPUs, co-processors or both simultaneously. Moreover, the recently released *SWhybrid* framework offers the possibility of combining CPUs, GPUs and Xeon Phi's [15]. Finally, there are also studies that use Field Programmable Gate Arrays (FPGAs) as accelerators, such as linear systolic array implementations for Xilinx Virtex FPGAs [16,17]; custom instructions [18] and the novel OpenCL paradigm on Intel-Altera's FPGAs [19].

This paper presents the latest release of the SWIMM software. SWIMM 2.0 provides new contributions to SW protein database searches by using Intel's latest AVX-512 SIMD extension. Intel's AVX-512 (Advance Vector Extensions) consists of new SIMD instructions for the x86 instruction set architecture (ISA) which allows the exploitation of 512-bit vectors in an SIMD way. Although this ISA was proposed by Intel in 2013, it was initially supported by Intel's Xeon Phi Knights Landing (KNL) accelerator and recently by Intel's general purpose Skylake-X Core processor. This work can be seen as an extension of the previous one published in [20], and the main differences are detailed below:

- Among the main contributions, we can highlight the creation of a public git repository with the binary executable developed for this paper [1].
- We have focused on code optimization by means of AVX-512 exploitation. Two main versions were developed: (1) an integer 32-bit version has been optimized in order to improve instruction-level parallelism exploitation on a KNL architecture and; (2) the use of lower range integer representation (8-bit) which enables 64 SIMD lane exploitation by means of AVX-512BW ISA (only available on the latest Intel Skylake processor). To the best of the authors' knowledge, this is the first evaluation of the AVX-512BW extensions for the SW algorithm.
- Additional experiments with smaller and larger protein datasets (UniprotKB/Swiss-Prot and UniProtKB/TrEMBL) were also carried out. This aspect emphasizes the independence of the SWIMM 2.0 performance as regards to the sequence sizes.
- We have included a performance comparison with other SIMD implementations such as SWIPE [2], Parasail [3] and libssa [4].
- In addition, a comparison with GPUs and FPGAs has been made considering not only performance but also power efficiency aspect. This compara-

[1] SWIMM 2.0 is available at `https://github.com/enzorucci/SWIMM2.0`
[2] SWIPE is available at public repository: `https://github.com/torognes/swipe`
[3] Parasail is available at public repository: `https://github.com/jeffdaily/parasail`
[4] libssa is available at public repository: `https://github.com/RonnySoak/libssa`

tive study denotes that SWIMM 2.0 is the best option to reduce the energy footprint in the protein sequence alignment scenario.

Section 2 introduces the basic concepts of the Smith-Waterman algorithm. Section 3 briefly introduces Intel's AVX-512 vector extensions. We describe our implementation of the SW algorithm in Section 4. In Section 5, performance results are described and finally, in Section 6, we conclude with some ideas for future research.

2 Smith-Waterman Algorithm

Let $S = s_1, s_2..., s_N$ be the N sequences under study. Let $P = \{< a, b > | S_a, S_b \in S,\ 1 <= a < b <= N\}$ be a sequence pair (S_a and S_b are usually referred to as the query and database sequences, respectively). The goal is to evaluate all pairwise comparisons in order to achieve the maximum score, which corresponds to the most similar region between a particular couple of sequences.

Given a pairwise sequence S_a and S_b of lengths $|S_a| = m$ and $|S_b| = n$, the recurrence relations for the SW [21] algorithm with affine gap penalties [22] are defined by the following equations:

$$H_{i,j} = max\{0, H_{i-1,j-1} + SM(S_a[i], S_b[j]), E_{i,j}, F_{i,j}\} \tag{1}$$

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

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

Let $H_{i,j}$ be the score for aligning the prefixes $S_a[1..i]$ and $S_b[1..j]$. Let $E_{i,j}$ and $F_{i,j}$ be the scores of prefix $S_a[1..i]$ aligned to a gap and prefix $S_b[1..j]$ aligned to a gap, respectively. Let SM be the *scoring matrix* which defines the substitution scores for all residue pairs. In general, SM rewards $S_a[i]$ and $S_b[j]$ with a posistive value if they are similar/equal, and penalizes them with a negative value otherwise. Let G_{oe} be the sum of gap open and gap extension penalties. Let G_e be 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 maximum value in the alignment matrix H is the optimal local alignment score.

It is important to note that equations 1, 2 and 3 represent the data dependence in each cell (i, j) since for a matrix cell its left, upper, and upper left neighbors need to be previously computed, as reflected in Figure 1.

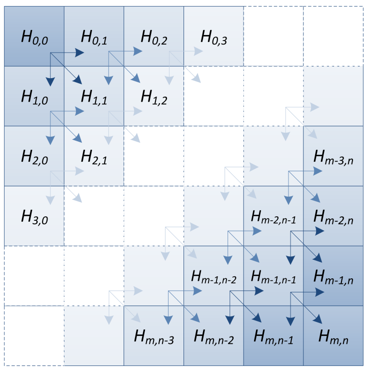


Fig. 1 Data dependences in the alignment matrix H.

3 Intel's AVX-512

Intel continues the trend of incorporating wider vector units with the announcement of the AVX-512 in 2013. AVX-512 is a set of new 512-bit SIMD x86 instruction set extensions that not only doubles the vector register width with respect to its predecessor, but also duplicates the available vector register. It is currently supported on Intel's Xeon Phi KNL and the lastest Skylake-X server. However, the incorporation of 512-bit vector extensions is not new; in fact, it was already available on Intel's Xeon KNC, but the novelty lies in the confluence support of AVX-512F (AVX-512 Foundation) on general purpose processors as well as on the accelerators.

Intel AVX-512 instructions are divided into seven categories: foundation instructions (AVX-512F), which are the base 512-bit extensions for arithmetic operations in single and double precision; conflict-detection instructions (AVX-512CD); exponential and reciprocal instructions (AVX-512ER); prefetch instructions (AVX-512PF); byte and word instructions (AVX-512BW); doubleword and quad-word instructions (AVX-512DQ); and vector length extension for operation in 128-bit and 256-bit (AVX-512VL). Future extensions are scheduled to be supported on next-generation accelerators and general purpose processors.

3.1 Intel's KNL

Accelerators are considered an alternative to general purpose systems in High-Performance Computing (HPC) for computational performance scalability with current consumption restrictions. Knights Landing [23] (KNL) is the code brand for the second generation Intel Xeon Phi product family, which is built for HPC purposes.

Intel launched the first Phi generation (KNC) with 61 x86 pentium cores (4 hardware threads per core) equipped with a 512-bit vector unit (VPU) each. In contrast to KNC co-processors connected via a PCI Express bus to the host,

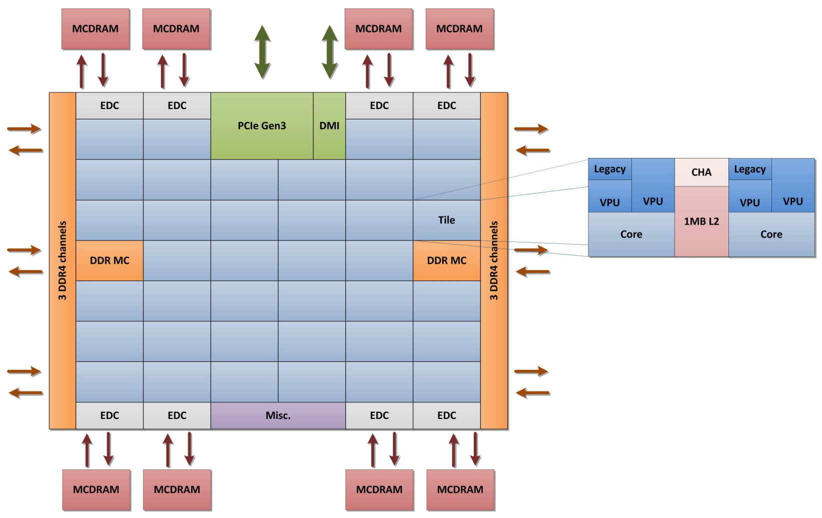


Fig. 2 Xeon Phi KNL architecture.

KNL accelerators can act as self-boot processors. KNL includes 38 physical tiles with two out-of-order core processors, two 512-bit VPUs per core and a shared cache memory. Unlike KNC, KNL also supports all legacy x86 vector ISAs such as 128-bit SSEx and 256-bit AVX, but just one of the two VPUs per core is compatible with legacy floating-point instructions, such as x87, MMX, and a subset of byte and word SSE instructions. The KNL accelerator only supports a subset of AVX-512 extensions such as AVX-512F, AVX-512CD, AVX-512ER and AVX-512PF. Another novelty at KNL is the incorporation of two on-package high-bandwidth memories (HBMs) [24]: multi-channel DRAM (MCDRAM) and improved (DDR) memory.

3.2 Intel's Skylake Server

Intel's Skylake (SKL) launched in 2015 and is the codename for the successor to the Broadwell microarchitecture. Intel distinguishes between two SKL processor versions [25]: *client* and *server*. The server core is considerably larger than the client one, featuring AVX-512 vector extensions. However, SKL servers only support a subset of AVX-512 extensions such as AVX-512F, AVX-512CD, AVX-512BW, AVX-512DQ and AVX-512VL. From a micro-architecture point of view, SKL launches 256-bit AVX or 128-bit SSEx vector instructions through ports 0 and 1. These two ports can also work in a fused way issuing a single arithmetic 512-bit vector instruction. It is important to note that some SKL server models include an additional, dedicated AVX-512 unit, which is managed through port 5.

4 SW Implementation

This section addresses the programming and optimization techniques performed for both Intel processors. Firstly, we describe the algorithm flow, which can be summarized in the following steps:

1. *Pre-processing stage*: database sequences are pre-processed to allow efficient parallel computation.
2. *SW stage*: alignments are computed.
3. *Sorting stage*: alignment scores are sorted in descending order.

We adopted the inter-task parallelism approach to take advantage of the Intel processors' VPUs. Multiple alignments are processed simultaneously since each channel computes the alignment of one database sequence and the query sequence. For this reason, database sequences must be grouped and the size of the groups is determined by the number of SIMD vector lanes. To minimize workload imbalances, database sequences are sorted by their lengths in ascending order and padded with dummy symbols before grouping.

SWIMM 2.0 accepts databases in FASTA format [5] and then turns them into a binary format. In this way, it avoids repeating the pre-processing stage for every search. Furthermore, this conversion then allows a faster reading of sequences to main memory.

4.1 Two Levels of Parallelism

SWIMM 2.0 exploits two levels of parallelism: (1) data and (2) threads. On the first level, we take advantage of SIMD instructions through the use of hand-tuned intrinsics. In particular, we have developed different versions considering the SSE4.1, AVX2, AVX-512F and AVX-512BW instruction sets. On the second level, we distribute the workload among multiple threads by leveraging the OpenMP programming paradigm.

The pseudo-code of our implementation is shown in Algorithm 1. The database sequences are dynamically assigned among the threads as soon as they become idle. As can be seen, the alignments are initially computed using 8-bit integer vectors because this range is generally enough to represent scores. This characteristic favors the exploitation of more SIMD parallelism since more data could be packed; however, it also requires the usage of saturated arithmetics to detect potential overflow cases. Therefore, when the final score is equal to the maximum possible value (i.e. potential overflow is detected), the alignment is recalculated using a 16-bit integer range. In the unlikely event of a 16-bit representation not being sufficient, 32-bit integers are used.

The Algorithm 2 presents the pseudo-code for the alignment matrix computation (for a graphic representation see Figure 3). The alignment matrix is divided into vertical blocks and computed in a row-by-row manner. As the

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

Algorithm 1 Pseudo-code for SWIMM 2.0

```
1: #pragma omp parallel for
2: for each $d$ in $vDB$ do                              ▷ $vDB$ is the preprocessed database
3:     $S_d = SW_{8bits}(q, d, SM, G_o, G_e)$            ▷ $q$ is the query sequence
4:     if $overflow_{8bits}(S_d)$ then
5:         $S_d = SW_{16bits}(q, d, SM, G_{oe}, G_e)$
6:         if $overflow_{16bits}(S_d)$ then
7:             $S_d = SW_{32bits}(q, d, SM, G_{oe}, G_e)$
8:         end if
9:     end if
10: end foreach
```

Algorithm 2 Pseudo-code for the alignment matrix computation

```
1: function $SW_{[8|16|32]bits}(q, d, SM, G_{oe}, G_e)$
2:     for $b \leq numBlocks$ do
3:         $P = buildProfile(q, d, SM)$
4:         for $i \leq m$ do                                   ▷ each row
5:             #pragma unroll
6:             for $j \leq bw$ do                              ▷ $bw$ is the block width
7:                 $s = SW\_CORE(H, E, F, P, G_{oe}, G_e)$     ▷ Calculate current cell value
8:             end for
9:         end for
10:    end for
11:    return $s$
12: end function
```

number of cache misses is reduced, this blocking technique improves data locality. Also, due to the data dependences mentioned in Section 2, each block needs the last column H and E values of the previous block. Therefore, two buffers are employed to save these data (shown as yellow bars). In addition, the inner loop is fully unrolled to increase performance.

4.2 Core Instructions

Figure 4 shows the implementation of the SW_CORE function according to all the extension sets used. First, the vector being calculated ($vCur$) is updated using the sum of the previous row block ($vPrev$) and the substitution scores for the database sequence residues against the query residue ($vSub$). Afterwards, these values are compared with the score vectors for alignments ending in a gap in the query (vE) and the database sequences (vF). The next step consists of $vCur$ with a zero vector comparison to ensure that all cells remain positive. Directly after this, the values in the current vector are then compared with the maximum score of the current alignment (vS). Finally, the vectors vE and vF are updated for the next row computations. Note that $vCur$ and $vPrev$ need to be swapped in order to maintain the correctness of the next row values.

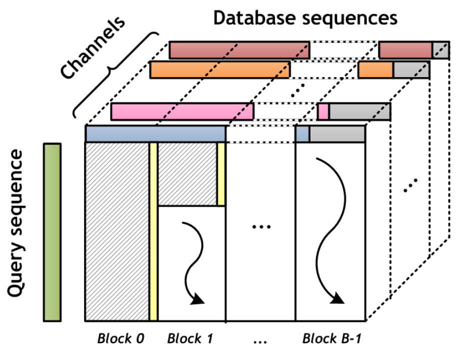
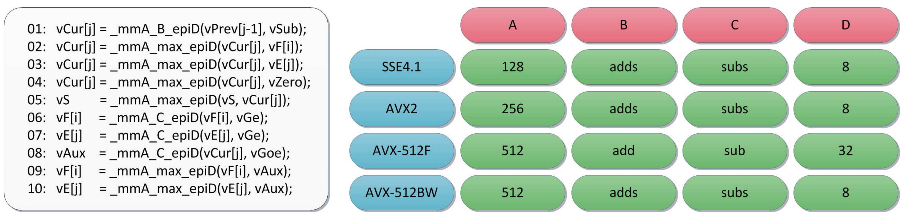


Fig. 3 Graphic representation of the alignment computation



```
01:  vCur[j] = _mmA_B_epiD(vPrev[j-1], vSub);
02:  vCur[j] = _mmA_max_epiD(vCur[j], vF[i]);
03:  vCur[j] = _mmA_max_epiD(vCur[j], vE[j]);
04:  vCur[j] = _mmA_max_epiD(vCur[j], vZero);
05:  vS       = _mmA_max_epiD(vS, vCur[j]);
06:  vF[i]    = _mmA_C_epiD(vF[i], vGe);
07:  vE[j]    = _mmA_C_epiD(vE[j], vGe);
08:  vAux     = _mmA_C_epiD(vCur[j], vGoe);
09:  vF[i]    = _mmA_max_epiD(vF[i], vAux);
10:  vE[j]    = _mmA_max_epiD(vE[j], vAux);
```

	A	B	C	D
SSE4.1	128	adds	subs	8
AVX2	256	adds	subs	8
AVX-512F	512	add	sub	32
AVX-512BW	512	adds	subs	8

Fig. 4 Implementation of the SW_CORE function

4.3 SIMD Set Selection

Taking into consideration the diversity of SIMD sets, SSEx can pack 16 elements of 8 bits in a single SIMD register while AVX2 is able to double up to 32. AVX-512 extensions are able to divide the 512-bit registers into 64 lanes by means of the AVX-512BW subset. Unfortunately, Xeon Phi KNL processors do not include AVX-512BW extensions. This fact means that the narrowest integer range on these devices is 32-bit for AVX-512. However, binary compatibility with Xeon processors allows Phi KNL to use SSEx and AVX2 instructions to exploit 8-bit integers. Xeon SKL does feature the AVX-512BW subset, besides offering compatibility with the SSEx and AVX2 sets.

4.4 Score Range Biasing

By biasing all 8-bit scores by an offset of 128 [7], the representation range is doubled and comparison with the zero vector can be avoided (line 4 in Figure 4) in saturated implementations. It is important to remark that this technique requires using saturated subtraction and converting back the final scores. In addition, the H, E and F buffers must be initialized with lower limits instead of 0.

Table 1 Experimental platforms used in the tests.

#	Host			Accelerator				
	Processor	**Memory**	**TDP (Watt)**	**Type**	**Architecture**	**Cores**	**Memory**	**TDP (Watt)**
1	Intel Xeon Phi 7250 68-core 1.40GHz (4 hw threads per core)	16GB HBW 64GB RAM	215	–				
2	2×Intel Xeon Gold 6128 6-core 3.40Ghz (2 hw threads per core)	96GB RAM	2×115	–				
3	2×Intel Xeon Gold 6138 20-core 2.0Ghz (2 hw threads per core)	96GB RAM	2×125	–				
4	2×Intel Xeon E5-2695 v3 14-core 2.30Ghz (2 hw threads per core)	96GB RAM	2×120	GPU		NVIDIA GTX1080		
					Pascal	2560	8GB RAM	180
						NVIDIA GTX 980		
					Kepler	2048	4GB RAM	165
5	Intel Xeon E5-1620 v3 4-core 3.50Ghz (2 hw threads per core)	32GB RAM	140	GPU		NVIDIA Tesla K20c		
					Kepler	2496	5GB RAM	225
6	2×Intel Xeon E5-2695 v3 14-core 2.30GHz (2 hw threads per core)	64GB RAM	2×120	FPGA		2×Altera Stratix V GSD5		
					Stratix	-	2×4GB RAM	2×25

4.5 Substitution Scores Selection

We have implemented two well-known optimizations of the SW algorithm that have been proposed in previous works, namely the Query Profile (QP) [26] and Score Profile (SP) [7] techniques for substitution scores selection:

– The QP strategy creates a matrix of size $|q| \times |\sum|$, where q is the query sequence and $\sum$ is the alphabet. Each row of this matrix stores the scores of the corresponding query residue against each possible residue in the alphabet. This optimization improves data locality because each thread compares the same query residue against different ones from the database. The cost is a negligible increment in memory requirements.

– The SP technique creates a three-dimensional score array of size $n \times L \times \sum$, where n is the length of the database sequence, L is the number of vector lanes and $\sum$ is the alphabet. This array is constructed prior to matrix computation and contains the substitution scores for each query-database residue pair. SP not only improves data locality but also reduces the number of operations in the innermost loop since its values can be gathered using a single vector load. However, the score array must be rebuilt for each database sequence group, so its suitability must be evaluated (especially for short queries).

To rearrange substitution scores, *shuffling* intrinsics are used in SSE4.1 (*_mm_shuffle_epi8*), AVX2 (*_mm256_shuffle_epi8*), and AVX-512BW (*_mm512_shuffle_epi8*) versions while permutation instructions are employed in the AVX-512F case (*_mm512_permutevar_epi32* and *_mm512_permutevar_epi32*).

5 Experimental Results

5.1 Experimental Design

All tests were carried out using the platforms described in Table 1. We have used Intel's ICC compiler with the *-O3* optimization level by default (version

18.0.0.128) and CUDA SDK 9.5. The first three servers were used to evaluate SWIMM 2.0 and the other SIMD-based alternatives, while the rest were employed to perform a comparison with GPUs and FPGAs. The performance was evaluated by carrying out similar experiments to those in previous works [19, 14, 11, 7]. We have evaluated SWIMM 2.0 by searching 20 query protein sequences against three well-known databases of different size:

- UniProtKB/Swiss-Prot (release 2016_11)[6]. This database contains 197953409 amino acid residues in 553231 sequences with a maximum length of 35213.
- Environmental NR (release 2016_11)[7]. This database contains 1384686404 amino acid residues in 6962291 sequences (maximum length of 11944).
- UniProtKB/TrEMBL (release 2016_11)[8]. This database contains 71002161 sequences comprising 23825212925 amino acid residues (the longest sequence has 36805 amino acid residues).

Environmental NR was selected as the default database. In addition, the input 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).

With regards to the scoring scheme, we configured BLOSUM62 and 10(2) as the scoring matrix and gap insertion (extension) penalty, respectively. Each particular test was run ten times and the performance was calculated as their average to avoid variability.

5.2 Performance Results on the Intel Xeon Phi 7250

GCUPS (billion cell updates per second) is commonly used as the performance metric in the context of SW, and its value is calculated by using the formula $\frac{|Q|\times|D|}{t\times10^9}$, 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 [7].

Figure 5 shows the performance of the different instruction sets and profile techniques used when varying the number of threads[9]. The best performances are achieved by the AVX2 extensions (488.3 GCUPS) followed by AVX-512F (187.2 GCUPS), with SSE4.1 (163 GCUPS) being the worst. As mentioned above, data level exploitation is critical to achieving maximum performance in this application. Even though AVX-512F doubles vectorial width with respect to AVX2 instructions, the lack of low-range integer operations imposes a strong limit on performance as almost all alignment scores can be represented using 8-bit integer data. Despite the fact that the SSE4.1 version computes

[6] Swiss-Prot: `http://www.uniprot.org/downloads`

[7] Environmental NR: `ftp://ftp.ncbi.nih.gov/blast/db/FASTA/env_nr.gz`

[8] TrEMBL: `http://www.uniprot.org/downloads`

[9] SSE4.1 and AVX2 versions using the *QP* technique were excluded from the analysis to improve figure readability since we found that the *SP* scheme always achieved the best performance, as in previous works [14]

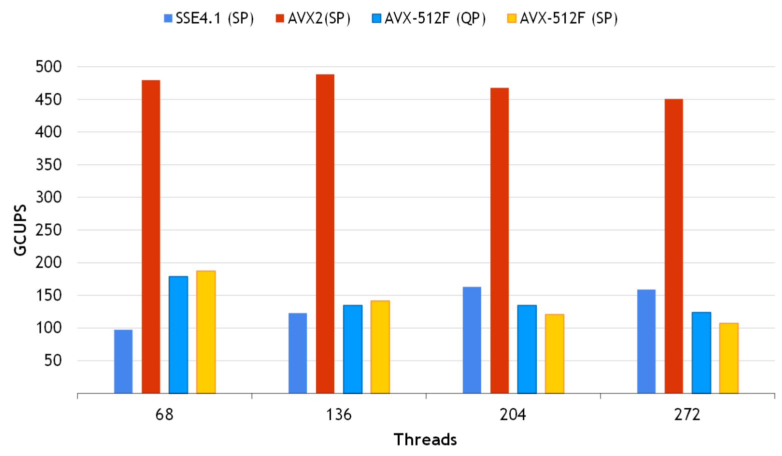


Fig. 5 SWIMM 2.0 performance for the different instruction sets and profile techniques used varying the number of threads on the Xeon Phi 7250.

16 alignments in parallel as does AVX-512F, the performance of the former is slower compared to the latter. In fact, only one of the two VPUs per core has support for the *legacy* subset of byte and word SSE instructions.

With regards to the number of threads, the AVX2 implementation performs best when using 136 threads, although its performance with 68 threads is very close (just 1% slower). A similar behavior occurs with SSE4.1 intrinsics, where using 204 threads is slightly better (1%) compared with the 272 threads configuration. However, both AVX-512F versions achieve the best results when using 68 threads, and large performance differences are observed with other thread configurations.

Figure 6 illustrates the performance obtained when varying the protein database with MCDRAM enabled (the most successful SWIMM 2.0 configuration is selected: AVX2 with 136 threads). It is important to mention that the MCDRAM exploitation does not need to modify the source code via the *numactl* utility. Regardless of the database employed, MCDRAM always provides a small performance gain (approximately 1.5%).

Figure 7 presents performance evolution when varying query length with the most favorable configuration for each implementation: 204, 136 and 68 threads for SSE4.1, AVX2 and AVX-512F intrinsics, respectively. Also, data is placed in MCDRAM memory. The SSE4.1 and AVX-512F implementations achieve an almost constant performance. As expected, this behavior is caused by the exploitation of the *inter-task* parallelism scheme. The AVX2 version's performance has a tendency to improve, something that becomes less pronounced with larger query sequences ($m \geq 2504$). For AVX-512F, the behavior of *QP* and *SP* differs, with a better performance for short sequences in *QP*. This aspect was also observed in other studies on the previous Xeon Phi KNC [12,14]. This was due to the additional overhead incurred by the *SP*

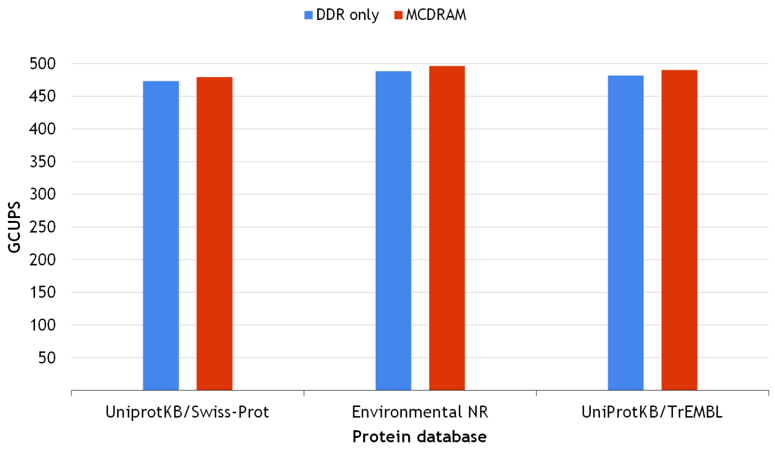


Fig. 6 SWIMM 2.0 performance for the AVX2 version varying the protein database on the Xeon Phi 7250.

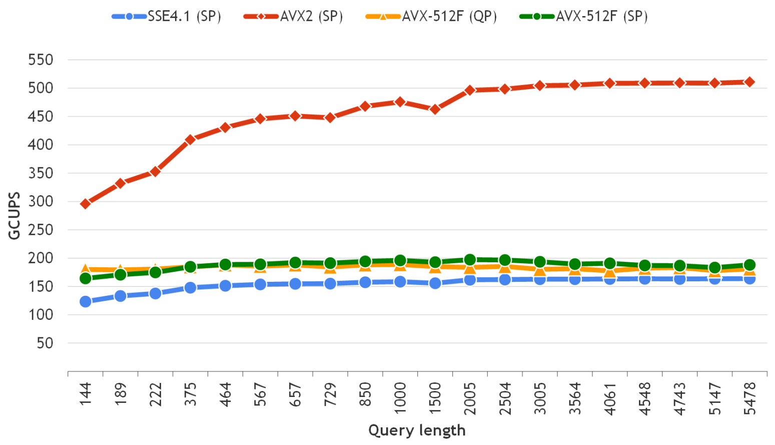


Fig. 7 SWIMM 2.0 performance for the different instruction sets used varying the query length on the Xeon Phi 7250.

construction, a fact which does not compensate for the indexation benefits in shorter queries. As a summary, the peak performances achieved are 511.1, 197.2, 188.8 and 163.7 GCUPS for the AVX2, AVX-512F (*SP*), AVX-512F (*QP*) and SSE4.1 implementations.

Lastly, we compared SWIMM 2.0 with other top-performance implementations[10]: SWIPE (v2.0.5) is the fastest SSEx-compatible tool while libssa and Parasail (v1.2) are libraries that provide a utility to take advantage of them.

[10] We have discarded the comparison with the SWhybrid framework [15] because we detected inconsistent alignment results in most of the experiments.

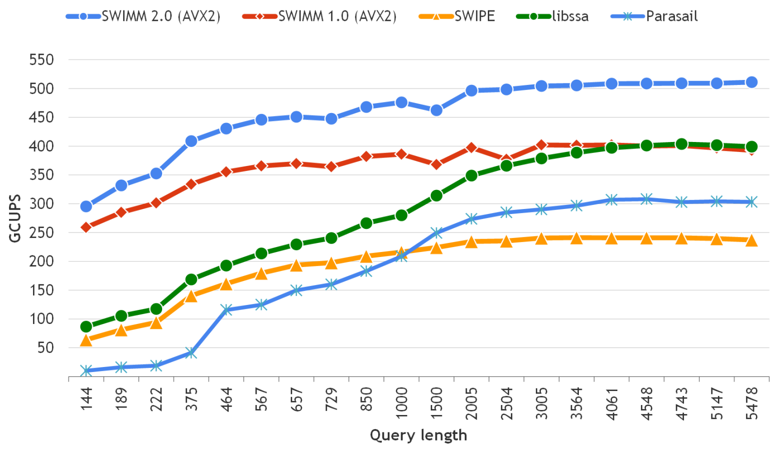


Fig. 8 Performance comparison among SWIMM 2.0, SWIMM 1.0, SWIPE, Parasail and libssa on the Xeon Phi 7250.

As both libssa and Parasail offer many different execution options, we tested all of them and selected the one that reported the best performance rates: *parasail_sw_striped_profile_avx2_256_sat* and *search_8_avx2_sw* for Parasail and libssa, respectively. Both implementations are based on the AVX2 subset using 8-bit integer data with overflow checking. Among the main differences are: (1) Parasail implements Farrar's stripped approach following *intra-task* parallelism while libssa exploits *inter-task* parallelism; and (2) Parasail performs *QP* optimization as long as libssa uses the *SP* technique. Additionally, we also considered SWIMM 1.0 [14] (v1.1.3).

Figure 8 shows the performance comparison between SWIMM 2.0, SWIMM 1.0, SWIPE, Parasail and libssa (MCDRAM enabled). It can be observed that the Parasail *intra-task* approach limits its parallel scalability for small alignments. In fact, SWIPE runs 2.7× faster than Parasail for the first half of the test set, even though the latter computes twice the cells in parallel. The *inter-task* scheme also benefits libssa, which beats Parasail for all query lengths by a factor of 2.4× on average. For its part, SWIMM 2.0 yields an average performance of 456 GCUPS with the maximum performance of 511.1 GCUPS, which means an average improvement of 1.24× against its previous version. In addition, our implementation outperforms SWIPE, Parasail and libssa by an average factor of 2.53×, 5.64× and 1.84×, respectively.

5.3 Performance Results on the Intel Xeon Gold 6128

As in the previous subsection, we assesed performance using the GCUPS metric. Figure 9 shows the performance of the different instruction sets and profile

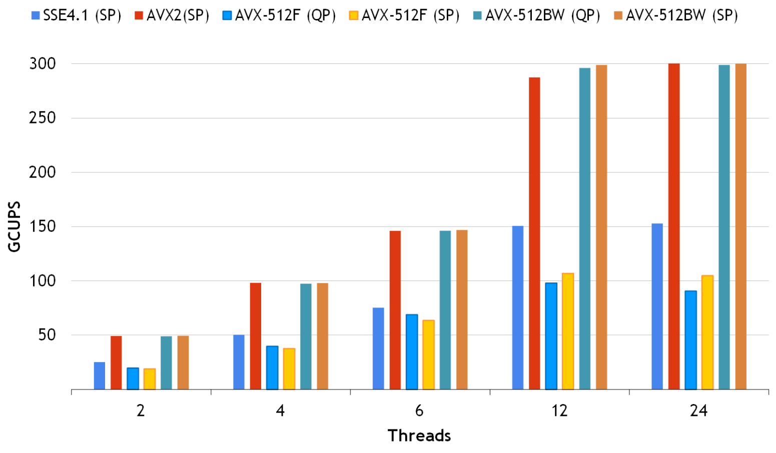


Fig. 9 SWIMM 2.0 performance for the different instruction sets and profile techniques used varying the number of threads on the Xeon Gold 6128.

techniques used when varying the number of threads [11]. Once again, we found that exploiting low-range integers is a key factor in obtaining a high-level performance. Unlike the Xeon Phi 7250 case, the SSE4.1 implementation is able to outperform AVX-512F counterparts (there is no restriction on VPU's support for SSEx instructions on this processor). In addition, the AVX2 version takes advantage of its larger vectorial width compared with the SSE4.1 implementation, doubling the latter's performance. However, despite doubling its vectorial lane, the AVX-512BW versions reported practically the same performance as the AVX2 counterpart. This contradictory aspect is due to the fact that not all Intel's Skylake models include the dedicated VPU unit that is accessible by the scheduler through port 5. In these cases, the fused VPU associated with ports 0 and 1 is the only unit for launching AVX-512BW instructions. In fact, this fused VPU works by launching a single AVX-512BW instruction or two AVX2 instructions simultaneously. Therefore, the benefit in doubling the vector capacity is compensated for doubling the number of AVX2 instructions that are simultaneously issued: one AVX-512BW with 64×8-bit integers or two AVX2 instructions with 32×8-bit integers each.

As can be seen in Figure 9, all the versions under study also take advantage of thread level parallelism. Independently of the intrinsic set used, the implementations displayed almost linear speedups with regard to the number of threads until reaching the number of cores (2×6). Furthermore, the usage of hyper-threading provided some extra GCUPS in the saturated versions.

Figure 10 shows performance evolution when varying query length for the SSE4.1, AVX2, AVX-512F and AVX-512BW set, respectively. Performance

[11] The SSE4.1 and AVX2 versions using the *QP* technique were excluded from the analysis to improve figure readability since we found that the *SP* scheme always achieved the best performance, as in previous works [14]

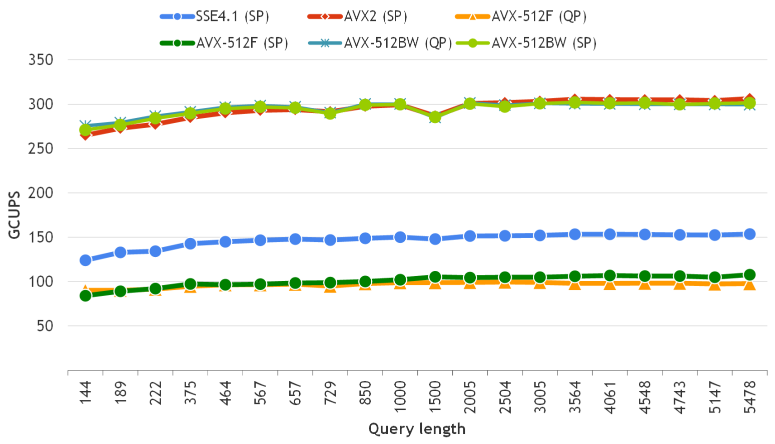


Fig. 10 SWIMM 2.0 performance for the different instruction sets used varying the query length on the Xeon Gold 6128.

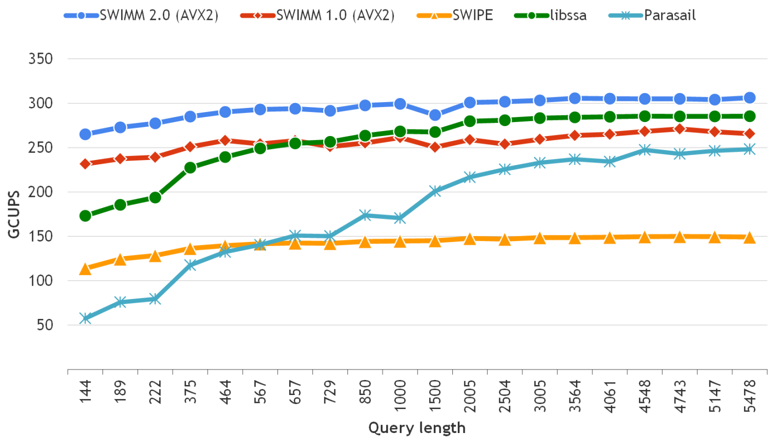


Fig. 11 Performance comparison among SWIMM 2.0, SWIMM 1.0, SWIPE, Parasail and libssa on the Xeon Gold 6128.

seems to be almost independent of the query length. The peak performances achieved are 153.7, 306.3, 99.6/107.7 and 301.4/302 GCUPS for the SSE4.1, AVX2, AVX-512F (QP/SP) and AVX-512BW (QP/SP) implementations.

Finally, we compare our newly-developed SWIMM 2.0 against SWIMM 1.0, SWIPE, Parasail and libssa in Figure 11. The behavior observed is quite similar to the Xeon Phi case: SWIPE beats Parasail for the six smallest queries while libssa outperforms both of them for all query lengths. SWIMM 2.0 outperforms all of them, running 1.15×, 1.17×, 1.94× and 2.08× faster than SWIMM 1.0, libssa, Parasail and SWIPE, respectively.

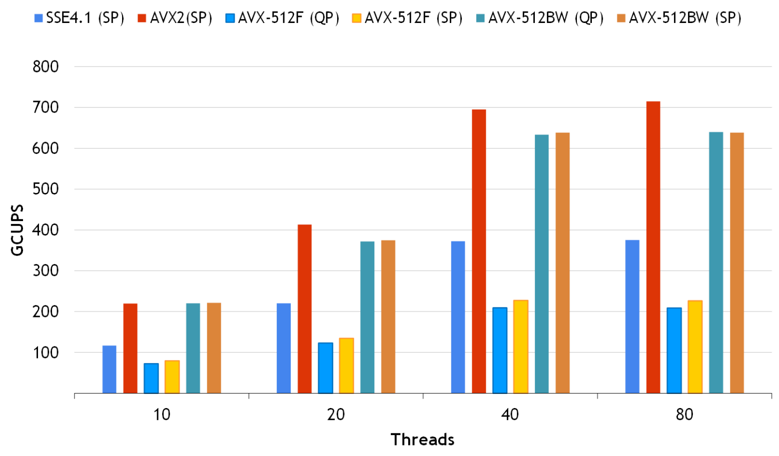


Fig. 12 SWIMM 2.0 performance for the different instruction sets and profile techniques used varying the number of threads on the Xeon Gold 6138.

5.4 Performance Results on the Intel Xeon Gold 6138

The usage of the dual Intel Xeon Gold 6138 allows us to assess scalability issues in the SKL micro-architecture taking into account that this server has a considerably larger number of cores (2×20) compared to the Xeon Gold 6128 server (2×6). Figure 12 shows the performance of the different instruction sets and profile techniques used when varying the number of threads. As in the previous Xeon Gold experiments, the SSE4.1 implementation outperforms the AVX-512F versions and again both *QP* and SP present similar performances for AVX-512BW instructions. However, in contrast to the slight performance improvement of the AVX2 version over the AVX-512BW alternatives, this gap is much more noticeable on the Intel Xeon Gold 6138. In particular, the performance difference is 1.09× for 40 threads but grows upto 1.12× when all available hardware threads are used. Using hardware counters and Intel's VTune tool, it was observed that this divergence is due to a higher pressure in the memory hierarchy of the AVX-512BW version. In fact, 19% of pipeline slots are stalled due to memory demand in AVX-512BW approach while only 13% was observed for AVX2 version. Additionally, the memory bound effect becomes even more remarkable when hyper-threading is enabled increasing by 2% for AVX-512BW versus 1.5% in AVX2. The performance rates show that AVX2 implementation beats all the rest counterparts reaching 722.9 GCUPS on average and a peak of 734 GCUPS.

Finally, we compare SWIMM 2.0 against the other SIMD-based alternatives on the Xeon Gold 6138 too (see Figure 13). SWIMM 2.0 takes advantage of the largest number of cores available in this server obtaining even greater performance differences compared to the Xeon Gold 6128 case. In particular,

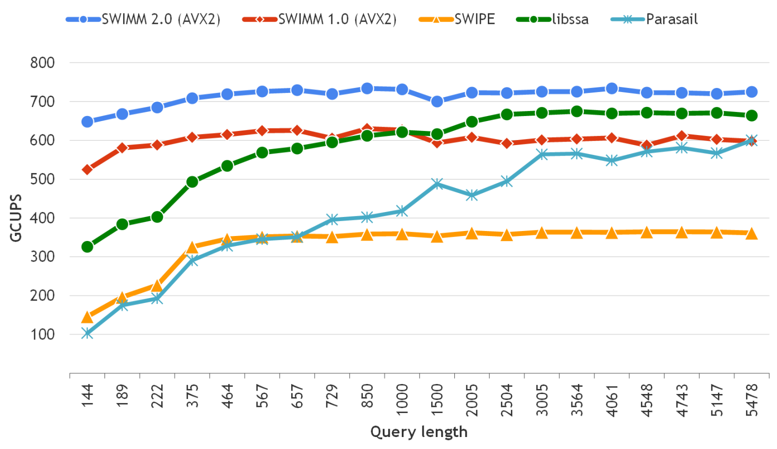


Fig. 13 Performance comparison among SWIMM 2.0, SWIMM 1.0, SWIPE, Parasail and libssa on the Xeon Gold 6138.

our SWIMM 2.0 implementation runs 1.19×, 1.26×, 2.06× and 2.26× faster than SWIMM 1.0, libssa, Parasail and SWIPE, respectively.

5.5 Performance and Power Efficiency Comparison with GPUs and FPGAs

We have also carried out a performance and power efficiency comparison with GPUs and FPGAs. For CUDA-enabled GPUs, CUDASW++ series [10,11] stand as the fastest option [12]. The latest version, CUDASW++ 3.0, is an hybrid CPU-GPU implementation optimized for Kepler architecture. This tool employs SSE intrinsics for the host and CUDA PTX video instructions (8-bit) for the accelerator. Since these video instructions were removed from subsequent NVIDIA architectures, we also selected CUDASW++ 2.0 implementation. This version computes using 32-bit integer data but allow us to include newer GPUs in the analysis. For FPGA accelerators, we have chosen the OSWALD package [19] in its hybrid configuration because it offers a satisfactory performance-power tradeoff.

Table 2 presents power efficiency ratios considering the GCUPS peak performance and the Thermal Design Power (TDP) of each platform. At the GPU side, it can be seen that newer generations improve performance-power ratio. While CUDASW++ 3.0 reaches more GCUPS than its previous version on the GTX980, the associated high power consumption results in lower GCUPS/Watt quotient. The best result is presented by CUDASW++ 2.0 on the newest GTX1080. Regarding FPGA accelerators, OSWALD achieves an

[12] Once again, we have discarded the comparison with the SWhybrid framework [15] because we detected inconsistent alignment results in most of the experiments.

[13] As CUDASW++ 2.0 is not and hybrid CPU-GPU implementation, only GPU power is considered

Table 2 Performance and power efficiency comparison with GPUs and FPGAs.

Implementation		Platform	GCUPS (peak)	Watt	GCUPS/Watt
SWIMM 2.0	1	Xeon Phi 7250	511	215	2.38
	2	2×Xeon Gold 6128	306	2×115	1.33
	3	2×Xeon Gold 6138	734	2×125	2.94
CUDASW++ 3.0	5	Xeon E5-1620 v3 + Tesla K20c	97	365	0.27
CUDASW++ 2.0	4	GTX1080	154	180[13]	0.86
	4	GTX980	81	165[13]	0.49
OSWALD	6	2×Intel Xeon E5-2695 v3 + 2×Stratix V	442	290	1.52

important performance peak but at the expense of a non-negligible power consumption. It is important to note that 83% of the power requirements correspond to the host. In an ideal situation with isolated FPGAs, the performance-power tradeoff could be increased upto 2 GCUPS/Watt according to [19].

On its behalf, SWIMM 2.0 outperforms all CUDASW++ implementations not only from performance perspective but also from power efficiency point of view. On the Xeon Gold 6138, SWIMM 2.0 obtains 2.4× more GCUPS than on the 6128 version. Because the first only requires 1.08× more Watts than the second, SWIMM 2.0 improves it power efficiency ratio from 1.33 to 2.94. Regarding the Xeon Phi 7250 processor, SWIMM 2.0 places behind the Xeon Gold 6138 but still reaching a remarkable GCUPS/Watt ratio (2.38).

6 Conclusions

The study and acceleration of the SW algorithm on different platforms has generated great interest in the scientific community due to its increasing relevance in Bioinformatics. In this paper, we have presented the recently released SWIMM 2.0 software, which incorporates new capacities to take advantage of Intel's latest AVX-512 SIMD extensions. In this regard, we have evaluated SWIMM 2.0 by using two different novel architectures: a Xeon Phi KNL accelerator and two Xeon Skylake general purpose processors.

Among the main contributions of this study we can highlight:

– The exploitation of low-range integer vectors is crucial in achieving optimal performance rates. Remarkable speedups were achieved on all platforms through enabling more SIMD parallelism.
– To the best of the authors' knowledge, this is the first evaluation of the SW algorithm with the AVX-512BW extensions. Regarding Xeon Skylake processors, AVX-512BW and AVX2 versions reported similar performances when using a small number of cores since the benefit in doubling the vector capacity of the first is compensated for doubling the number of simultaneous instructions of the latter. However, as the number of cores increases, AVX-512BW performance is penalized due to a higher memory bound.
– Multi-threading must be carefully evaluated. On KNL accelerators, different numbers of threads produced the best results for each instruction set. In

the same way, enabling hyper-threading not always improved performance on the SKL architecture.
- The usage of MCDRAM provides some additional GCUPS with practically no programmer intervention on KNL.
- The AVX2 instruction set still remains as the best choice for current HPC platforms from Intel. The peak performances observed are 511, 306.3 and 734 GCUPS on the KNL accelerator, the Xeon Gold 6128 and the Xeon Gold 6138, respectively.
- SWIMM 2.0 outperformed other state-of-the-art SIMD-based implementations on all platforms under study. In addition, SWIMM 2.0 also showed better performance than other implementation achieving less energy footprint rates: 2.38 GCUPS/Watts on the Xeon Phi 7250 and 2.94 GCUPS/Watts on the Xeon Gold 6138.

In view of the results obtained, we forecast a promising opportunity to accelerate SW database searches on future Xeon SKL processors since these devices will include more dedicated VPUs (enabling more SIMD parallelism).

Acknowledgements This work has been supported by the EU (FEDER) and the Spanish MINECO, under grant TIN2015-65277-R and the CAPAP-H6 network (TIN2016-81840-REDT).

References

1. E. Bender, Nature **527**, S19 (2015)
2. S.F. Altschul, T.L. Madden, A.A. Schffer, J. Zhang, Z. Zhang, W. Miller, D.J. Lipman, NUCLEIC ACIDS RESEARCH **25**(17), 3389 (1997)
3. W.R. Pearson, D.J. Lipman, Proceedings of the National Academy of Sciences of the United States of America **85**(8), 2444 (1988). DOI 10.1073/pnas.85.8.2444
4. P.E. Sæbø, S.M. Andersen, J. Myrseth, J.K. Laerdahl, T. Rognes, Nucleic acids research **33**(suppl_2), W535 (2005)
5. M. Farrar, Bioinformatics **23 (2)**, 156 (2007)
6. E. Rucci, C. García, G. Botella, A. De Giusti, M. Naiouf, M. Prieto-Matías, *State-of-the-Art in Smith–Waterman Protein Database Search on HPC Platforms* (Springer, 2016), pp. 197–223. DOI 10.1007/978-3-319-41279-5_6. URL http://dx.doi.org/10.1007/978-3-319-41279-5_6
7. T. Rognes, BMC Bioinformatics **12**(1), 221 (2011). DOI 10.1186/1471-2105-12-221. URL http://dx.doi.org/10.1186/1471-2105-12-221
8. J.T. Frielingsdorf, Improving optimal sequence alignments through a simd-accelerated library. Master's thesis, University of Oslo (2015)
9. J. Daily, BMC Bioinformatics **17 (81)** (2016)
10. Y. Liu, B. Schmidt, D.L. Maskell, BMC Research Notes **3**(1), 1 (2010). DOI 10.1186/1756-0500-3-93. URL http://dx.doi.org/10.1186/1756-0500-3-93
11. Y. Liu, A. Wirawan, B. Schmidt, BMC Bioinformatics **14:117** (2013)
12. Y. Liu, B. Schmidt, in *25th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2014)* (2014)
13. H. Lan, W. Liu, B. Schmidt, B. Wang, in *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (2015), pp. 503–510. DOI 10.1109/BIBM.2015.7359735
14. E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, M. Prieto-Matas, Concurrency and Computation: Practice and Experience **27**(18), 5517 (2015). DOI 10.1002/cpe.3598. URL http://dx.doi.org/10.1002/cpe.3598

15. H. Lan, W. Liu, Y. Liu, B. Schmidt, in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)* (2017), pp. 42–51. DOI 10.1109/IPDPS.2017.42
16. M. Isa, K. Benkrid, T. Clayton, C. Ling, A. Erdogan, in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on* (2011), pp. 344–351. DOI 10.1109/AHS.2011.5963957
17. T.F. Oliver, B. Schmidt, D.L. Maskell, IEEE Transactions on Circuits and Systems II: Express Briefs **52**(12), 851 (2005). DOI 10.1109/TCSII.2005.853340
18. T.I. Li, W. Shum, K. Truong, BMC Bioinformatics **8:I85** (2007)
19. E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, M. Prieto-Matas, International Journal of High Performance Computing Applications p. 1094342016654215 (2016). DOI 10.1177/1094342016654215. URL `http://dx.doi.org/10.1177/1094342016654215`
20. E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, M. Prieto-Matias, in *Algorithms and Architectures for Parallel Processing: 17th International Conference, ICA3PP 2017, Helsinki, Finland, August 21-23, 2017, Proceedings*, ed. by S. Ibrahim, K.K.R. Choo, Z. Yan, W. Pedrycz (Springer International Publishing, Cham, 2017), pp. 569–579. DOI 10.1007/978-3-319-65482-9_42. URL `https://doi.org/10.1007/978-3-319-65482-9_42`
21. T.F. Smith, M.S. Waterman, Journal of Molecular Biology **147**(1), 195 (1981)
22. O. Gotoh, in *Journal of Molecular Biology*, vol. 162 (1981), vol. 162, pp. 705–708
23. A. Sodani, R. Gramunt, J. Corbal, H.S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, Y.C. Liu, IEEE Micro **36**(2), 34 (2016). DOI 10.1109/MM.2016.25
24. R. Asai. MCDRAM as High-Bandidth Memory (HBM) in Knights Landing Processors: Developer's Guide (2016). URL `https://goparallel.sourceforge.net/wp-content/uploads/2016/05/Colfax_KNL_MCDRAM_Guide.pdf`
25. I. Corporation. Intel 64 and ia-32 architectures optimization reference manual (2017). URL `https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf`
26. T. Rognes, E. Seeberg, Bioinformatics **16**(8), 699 (2000). DOI 10.1093/bioinformatics/16.8.699. URL `+http://dx.doi.org/10.1093/bioinformatics/16.8.699`