Multi-Core SIMD ASIP for DNA Sequence Alignment

Nuno Filipe Simões Santos Moraes Neves

IST / INESC-ID
Rua Alves Redol, 9, 1000-029 Lisboa - Portugal
Email: {nneves}@sips.inesc-id.pt

Abstract—A novel Application-Specific Instruction-set Processor (ASIP) architecture for biologic sequences alignment is proposed in this manuscript. The presented processor achieves high processing throughputs by exploiting both fine and coarse-grained parallelism. The former is achieved by extending the Instruction Set Architecture (ISA) of a synthesizable processor to include multiple specialized SIMD instructions that implement vector-vector and vector-scalar arithmetic, logic, load/store and control operations. Coarse-grained parallelism is achieved by using multiple cores to cooperatively align multiple sequences in a shared memory plain, comprising proper hardware-specific synchronization mechanisms. To ease the programming of the sequence alignment algorithms, a compilation framework based on a suitable adaptation of the GCC back-end was also implemented. The proposed system was prototyped and evaluated on a Xilinx Virtex-7 FPGA VC707 Kit, achieving a 190MHz working frequency. A vanilla and a state-of-the-art SIMD implementations of the Smith-Waterman algorithm were programmed in both the proposed ASIP and in an Intel Core i7 processor. When comparing the achieved speedups, it was observed that the proposed ISA allows to achieve a 33x speedup, which contrasts to the 11x speedup provided by SSE2 in the Intel Core i7 processor. The scalability of the multi-core system was also evaluated and proved to scale almost linearly with the number of cores. A 900-fold speedup was achieved with a 64-core processing framework.

I. INTRODUCTION

The increasing demand for computational resources in some specific and prominent application domains, coupled with severe energy and power constraints, has driven the development of highly efficient processing systems, where the processor architecture plays a major role, both on the overall performance and on the resulting efficiency of the whole processing system. In particular, the adopted Instruction Set Architecture (ISA) must be carefully tuned for the specific requirements of the considered set of algorithms. By following such approach, it is possible to achieve an efficient solution that is, at the same time, flexible enough to support algorithmic improvements.

Bioinformatics applications represent one class of algorithms with particularly high performance and efficiency requirements. Among those, protein and Deoxyribonucleic acid (DNA) sequence alignment algorithms, whose optimal solutions are usually obtained by using Dynamic Programming (DP) methods, tend to present a large runtime when executed in current General Purpose Processors (GPPs). To reduce the alignment runtime, non-optimal heuristic-based algorithms have also been developed. Examples of such heuristic algorithms include BLAST [1] and FASTA [2]. Still, although faster, such heuristic algorithms have a reduced sensitivity and many of them still execute the optimal DP algorithm in a final post-processing phase, in order to improve the quality of the results. Hence, the use of the optimal DP algorithms is always preferred, but not always used due to time restrictions.

The Smith-Waterman (SW) algorithm [3], characterized by an $O(mn)$ time complexity, is a widely established DP algorithm to obtain the local alignment between a query sequence ($q$) and a reference sequence ($d$), of sizes $m$ and $n$ respectively. It operates in two distinct phases: it starts by filling a score matrix $H$, followed by a traceback phase over this matrix. The matrix is filled by using an affine gap penalty model [4], given by the following equation:

$$H(i, j) = \max \begin{cases} \ H(i - 1, j - 1) + Sbc[q[i], d[j]], \\ E(i, j), \\ F(i, j) \end{cases}$$

$$F(i, j) = \max \begin{cases} \ H(i - 1, j) - \alpha, \\ F(i - 1, j) - \beta \end{cases}$$

$$E(i, j) = \max \begin{cases} \ H(i, j - 1) - \alpha, \\ E(i, j - 1) - \beta \end{cases}$$

where $\alpha$ and $\beta$ represent the cost of gap opening and extension, and $Sbc[q[i], d[j]]$ denotes the substitution score value obtained by aligning character $q[i]$ against character $d[j]$. The initial conditions are given by $H(i, 0) = H(0, j) = E(i, 0) = F(0, j) = 0$.

Several solutions have been proposed to accelerate the execution of the SW algorithm. The solutions that offer the highest performance typically adopt the form of dedicated accelerators, like those proposed in [5], [6], [7] and [8]. However, despite their very high throughput, these solutions lack the necessary adaptability to allow the execution of other similar algorithms (e.g., the Needleman-Wunsch algorithm for global alignment [9]) or even variations of the same algorithm (e.g., the banded SW algorithm used in FASTA [2]). Therefore, whenever adaptability is a system requirement, the use of programmable processing solutions proves to be highly necessary.
Among these, GPPs arise as a natural and commonly used platform to execute these algorithms, especially due to the set of Single-Instruction Multiple-Data (SIMD) instruction-set extensions that are already included in most current off-the-shelf processors (e.g., Intel processors). Such instructions allow to efficiently exploit fine-grained parallelism within these algorithms and thus improve the resulting performance.

In accordance with this trend, the development of an efficient multi-core Application Specific Instruction-set Processor (ASIP) is regarded as a particularly suited approach to reduce the runtime of some widely used and well-established bioinformatics applications. Equally importantly, the additional efficiency that may be offered by such ASIP will make it an adequate solution to be adopted in increasingly prominent mobile computing systems. Such efficient platforms, coupled with the development of low-power biochips with higher sensor densities [10], will allow the development of fast, portable and fully autonomous biologic sequences analysis systems [11].

To commit with all the performance and design requires enumerated above, a new ASIP specifically adapted for biological sequences alignment algorithms is proposed in this manuscript. The attained processing throughput was achieved as a result of a two-fold improvement in the original architecture: i) extension of the processor ISA to support multiple specialized SIMD vector instructions, to extensively exploit fine-grained parallelism; and ii) implementation of an extensive multi-core computational structure, composed by multiple instantiations of the designed ASIP, in order to efficiently exploit coarse-grained parallelism. The cumulative result of these two important contributions was demonstrated with an FPGA prototype of the proposed multi-core SIMD ASIP, which proved capable of offering speedup values of 900x.

The paper is organized as follows: after the introductory motivation presented above, section II describes the current state-of-the-art on SIMD implementations of DP sequences alignment algorithms. In Section III it is proposed the new and adapted ISA for this specific application domain, whose implementation in the processor architecture is presented in Section IV. In Section V it is presented the developed multi-core processing structure, composed by multiple instantiations of the designed ASIP. Section VI presents the obtained experimental results and Section VII concludes the presentation, with the enumeration of the most important contributions.

II. SIMD IMPLEMENTATIONS OF THE SW ALGORITHM

To accelerate the alignment procedure as much as possible, while still assuring the best alignment, several SIMD parallelization approaches of the SW algorithm have been presented. Three specific implementations based on the exploitation of Intel MMX/SSE instruction set extensions are particularly relevant and were presented by A. Wozniak [12], T. Rognes [13], [14] and M. Farrar [15]. Their differences mainly lie in the adopted data processing pattern (see Fig. 1).

To assure the simultaneous commitment of as many algorithm dependencies as possible, Wozniak’s implementation [12] concurrently processes a set of cells along the minor diagonal of the alignment matrix (see Fig. 1(a)). However, this straightforward approach comes at the cost of non-trivial access patterns, leading to considerable memory overheads.

To simplify and accelerate the load of the substitution scores from memory, T. Rognes and E. Seeberg [13] presented an alternative approach that pre-computes a query profile for the entire reference. With such technique, a vector of cells parallel to the query sequence can be simultaneously processed by each instruction (see Fig. 1(b)). The commitment of the data dependencies is guaranteed by defining threshold conditions relating each computed score and the insertion/extension gap penalties, allowing to disregard most comparisons related to the vertical dependencies of the algorithm. Nevertheless, such approach still implies the introduction of conditional branches in the inner loop of the algorithm, making the execution time dependent on the used scoring matrix and gap penalties.

M. Farrar [15] also adopted a pre-computed query profile, but improved the processing scheme by using a stripped access pattern (see Fig. 1(c)), where the computations are carried out in several separate stripes that cover different parts of the query sequence. Hence, the query is divided into p equal length segments, where p is given by the number of vector elements that can be simultaneously accommodated in a SIMD register. As an example, when 128-bit SSE2 registers are considered to process 8-bit data elements, p equals to 16. The length of each segment is given by \( t = \frac{(m+p-1)}{p} \), where padding zeros are inserted whenever the query size \( m \) is not long enough to completely fill all the segments (see Fig. 2).

![Fig. 1. SIMD implementations of the SW algorithm [14]. The first five SIMD iterations were numbered and represented with different gray levels. For simplicity, only 4 data elements are shown in each SIMD register.](image1)

![Fig. 2. Data dependencies between the last segment and the first segment of the next column of the score matrix \( H \). For simplicity, only 4 data elements are shown in each SIMD register (represented in a gray scale).](image2)
This stripped computation of the score matrix is fulfilled by assigning each SIMD vector element to one distinct segment (see Fig. 2). Accordingly, each matrix column, corresponding to a reference symbol \( d[j] \), is processed in \( t \) iterations, where each iteration simultaneously processes \( p \) query symbols, separated by \( t - 1 \) lines in the score matrix. As an example, when \( p = 16 \) the second iteration of the algorithm simultaneously processes in a SIMD register the following query symbols: \( \{ q[2], q[t + 2], q[2t + 2], q[3t + 2], \cdots, q[15t + 2] \} \). With this modified pattern, it is possible to move the conditional statements related to the commitment of the vertical dependencies to an independent lazy loop. This loop is executed outside the inner loop, where the vertical dependencies have to be considered only once, before starting the processing of the next reference symbol (see Fig. 2), thus reducing the impact of the vertical dependencies.

The above cumulative set of contributions and improvements led to significant speedup values of the alignment, which makes Farrar’s technique [15] one of the fastest SIMD implementations of the SW algorithm. It has even justified its integration in many current high-performance alignment frameworks, such as the latest versions of SSEARCH [16].

Meanwhile, T. Rognes [14] presented another parallelization of the algorithm that exploits other capabilities made available by modern processors. However, the considered application domain is somewhat different from the previous approaches. Instead of solving the single-reference single-query alignment problem, he simultaneously compares several different reference sequences with one single query sequence in each SIMD operation (see Fig. 1(d)), which allows additional speedups when the underlying application aligns each query to a database of reference sequences.

III. DEDICATED SIMD INSTRUCTION SET FOR BIOLOGICAL SEQUENCES ALIGNMENT

The proposed ISA was defined by targeting the acceleration of a broad range of DP algorithms, including not only the classic local and global sequence alignment procedures (such as the several SIMD implementations of the Needleman-Wunsch and Smith-Waterman algorithms [12], [13], [15]), but also other widely used DP algorithms adopted in several application domains (e.g.: Hidden Markov Models (HMMs), Viterbi chains, etc.). Nevertheless, due to its higher performance and to its prevalence in most widely established bioinformatics applications, Farrar’s SIMD implementation [15] will be herein adopted as the elected case-study, to demonstrate the advantages of the proposed ASIP.

A. Case-study: Farrar’s SIMD implementation

By analyzing Farrar’s pseudo-code definition [15] (see Fig.4(a)), it is clear that the adoption of vectorial arithmetic instructions will potentially accelerate this algorithm implementation. These instructions should not only speedup the operations between vectors, but they might also facilitate the several operations between vectors and scalars, which are particularly useful when subtracting the gap penalties. The shifting of the \( F \) and \( H \) vectors can also be efficiently implemented with a vector element shift instruction. Furthermore, since all these new instructions will be dealing with SIMD vectors, it is also advantageous to include new memory access instructions, to handle vector-sized variables.

On the other hand, from a more detailed inspection of an Intel SSE2 assembly implementation [15] (see Fig.4(b)), it can also be observed that the lazy loop condition assertion requires at least 5 instructions. Therefore, a new and specialized branch instruction to simultaneously assert a branch condition in all vector elements, without any additional processing, would significantly increase the achieved performance.

As a consequence, it is clear from the above observations that a dedicated and optimized ISA not only should include an adapted set of arithmetic SIMD operations (with a particular emphasis to the addition, subtraction, compare and maximum operations), but should also incorporate other classes of instructions, comprehending logic, memory-access and control operations. Furthermore, an appropriate register structure particularly adapted to the targeted application domains will have to be defined, to accommodate both the vectorial and scalar operands. These two aspects will be briefly covered in the following two subsections.

B. SIMD Registers

Although not limited at this respect, the proposed instruction set and the corresponding data-path (see section IV) provides support for the same register and vector-element sizes as Intel SSE2 (used by Farrar [15]), i.e. 128-bit registers, with 8 or 16 elements. Furthermore, the vector elements of each register can take any size, starting from 8 bits to the limit imposed by the register size. To obtain a fair comparison with Farrar’s [15] SSE2 implementation, only 128-bit registers with 8-bit elements will be considered in this particular case-study.

In order to simplify the implementation of the proposed instruction set and to save hardware resources, it is assumed that all processor registers within the register bank have the same size. Hence, any non-SIMD instruction will only operate over the least-significant part of the register, corresponding to a scalar processor word. In contrast, all the proposed SIMD instructions will operate over the entire register (see Fig. 3). With this design option, the critical path of the processor is confined within the data-path corresponding to the non-SIMD operations, thus making it independent of the extended SIMD register size.

Fig. 3. Division of each processor register into 8 or 16 SIMD vector elements (v.e.). Non-SIMD operands coexist in the same register, occupying the least significant word.

C. Proposed SIMD Instruction Set

The specialized ISA that is herein proposed defines 58 specialized SIMD instructions for arithmetic, logic, memory access and control operations. The most relevant subset adopted in Farrar’s algorithm implementation is depicted in Table 1. These instructions are subdivided into 3 classes: vector-vector, operating over the corresponding pairs of elements in each
TABLE I.  SUBSET OF THE PROPOSED SIMD SPECIALIZED INSTRUCTION SET FOR BINARY SEQUENCE ALIGNMENT.

<table>
<thead>
<tr>
<th>ARITHMETIC</th>
<th>Logic</th>
<th>MEMORY</th>
<th>CONTROL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector-Vector</td>
<td>Inner-Vector</td>
<td>Vector-Vector</td>
<td></td>
</tr>
<tr>
<td>ADDVV</td>
<td>ADDDS</td>
<td>ADDV</td>
<td>SLVV</td>
</tr>
<tr>
<td>RSUBVV</td>
<td>RSUBVS</td>
<td>RSUBV</td>
<td>MTV</td>
</tr>
<tr>
<td>CMPVV</td>
<td>CMPVS</td>
<td>CMPV</td>
<td>SV</td>
</tr>
<tr>
<td>CMPSVV</td>
<td>CMPSVS</td>
<td>CMPSV</td>
<td>SV2</td>
</tr>
<tr>
<td>MAXVV</td>
<td>MAXVS</td>
<td>MAXV</td>
<td>BLTVV</td>
</tr>
<tr>
<td>MAXUVV</td>
<td>MAXUVS</td>
<td>MAXUV</td>
<td>BNEVV</td>
</tr>
</tbody>
</table>

SIMD register; vector-scalar, operating between one SIMD register and the non-SIMD operand of another register; and inner-vector, operating between the adjacent pairs of vector elements in a single SIMD register.

According to the defined register structure, non-SIMD load and store instructions will only operate over the processor word-size. As a consequence, the extended SIMD instruction set also provides the corresponding SIMD versions (LV and SV) for vectorized memory accesses. The memory address is computed with non-SIMD operands and only the destination (LV) or the origin (SV) are SIMD operands. There is also a special move instruction, to write a scalar to a specific SIMD vector position, keeping the remaining positions unchanged. The position and the scalar are provided in non-SIMD operands.

Furthermore, by considering the requirements that were referred in section III-A, a particular attention was also devoted to the definition of a new subset of optimized control instructions. This fact arises from the significant predominance of loop procedures in these DP algorithms (generally implemented with conditional branch instructions) and takes into account the severe penalties that these control instructions generally impose on deep pipeline architectures. As a consequence, such inherent losses in the attainable throughput (imposed by unavoidable pipeline flushes introduced by branch instructions) also had to be taken into account when the base processor structure was selected, as it will be described in Section IV. Such restriction determined the adoption of shallower pipeline structures, contrasting with the most recent Intel GPP architectures. To further adapt the new ISA to the targeted algorithms, the branch condition in the new set of conditional branch operations is evaluated over all the elements in the SIMD vector and the branch is only taken if the condition is valid for all of them. The branch target is computed with non-SIMD operands.

Hence, by comparing Farrar’s [15] algorithm implementation based on Intel SSE2 ISA (see Fig.4(b)) with an implementation based on the proposed ISA (see Fig.4(c)), it is observable that an immediate gain in what concerns the number of instructions is promptly achieved with the proposed ISA, with more visible advantages in the lazy-loop. The major contributor to this reduction is the new set of vectorized control instructions, that significantly reduce the control overhead.

Finally, it is important to note that the most important advantage of the proposed ASIP arises from the fact that it adopts a strict Reduced Instruction Set Computer (RISC) paradigm based on a shallow pipeline structure, contrasting with Intel’s Complex Instruction Set Computer (CISC) model. As a consequence, the observed difference in the number of executed instructions, together with the RISC single-cycle per instruction ratio (instead of CISC multiple-cycle per instruction), will significantly augment the processing gain, as it will be demonstrated in section VI.

IV. SIMD PROCESSOR ARCHITECTURE

The MB-LITE [17] soft-core was used as the base architecture for the implementation of the proposed ISA, not only due to its simple and portable processing structure, but also because it is a compliant implementation of the well known MicroBlaze ISA [18], offering the advantage of an already existing compiler that can also be extended to support the new instructions. Furthermore, the MB-LITE design is highly configurable and is relatively easy to adapt to the proposed ISA. The reduced hardware resources that are required by this core were also taken into account, prospecting the bases for a scalable multi-core processing platform to exploit coarse-grained parallelism, as will be described in section V.
A. MB-LITE processor architecture

The MB-LITE [17] processor is a 32-bit Harvard RISC based on the MIPS five-stage pipeline architecture. Accordingly, all instructions have a single cycle latency, except the branches whose latency is two or three clock cycles (with or without delay slots, respectively).

The considered MicroBlaze ISA contains the usual integer arithmetic and logic operations, conditional and unconditional branches, and load/store instructions. Some groups of instructions were left out, including the multiplication and barrel shifter operations, as well as all floating point and special register operations. As in the MicroBlaze architecture, the MB-LITE has two basic types of instructions: type A (register type) and type B (immediate type). All instructions have three operands (3 registers or 2 registers plus 1 immediate), except the control and shift instructions, which have two operands (2 registers or 1 register plus 1 immediate). Thirty two general purpose registers are available in the register bank, adopting the same reservation policy as in MicroBlaze [18].

All data hazards are identified and solved in the decode stage of the pipeline. Forwarding is provided from the latest stages, with the exception of the load instruction, which causes the introduction of a stall when the memory value is used by the following instruction. Control hazards are solved by performing static branch prediction with a predict not-taken strategy, which results in a pipeline flush whenever the branch should be taken. The processor also provides a single-line interrupt mechanism. The 32-bit data memory is organized in parallel 8-bit blocks, in order to speedup the memory access.

Among the provided modules, MB-LITE [17] integrates an address decoder to allow communication with the different peripherals in a memory mapped I/O organization. It also provides a character device, so that STDOUT can be easily used in the software development phase.

B. Modification of the execution unit

To support the proposed extension of the ISA, the execution unit had to be modified, by extending its original Arithmetic and Logic Unit (ALU) to include a new SIMD module.

Despite being fully parameterizable, the configuration of the designed SIMD module that was adopted for this specific case-study uses 128-bit registers with 16 8-bit SIMD elements. Other parameterizations could equally be used, to process values ranging from 16-bit data words to any other word size, by taking into account that increasing the word size may decrease the maximum clock frequency. The operand size can range from 2 bits (useful for DNA processing) to half the register size. By increasing the number of operands, the amount of generated logic also increases, with consequences on the processor hardware resources.

The addition and subtraction operations require one adder per SIMD vector element, together with some extra multiplexing logic (illustrated in Fig. 5). Since different types of SIMD operations are supported (vector-vector, vector-scalar and inner-vector), the required elements have to be selected from the corresponding registers and only then does the execution unit perform all the parallel arithmetic operations. The results are then chosen based on appropriate control signals.

The new maximum instruction is based on the already existing compare instruction, comprising a subtraction followed by a signal evaluation of the result. Therefore, the same logic can be used to implement these two instructions, requiring only a multiplexer, selected by the Most-Significant Bit (MSB) of the result, to choose the maximum between the two operands. Though simple to implement, this additional logic would greatly increase the critical path of the execution unit when extending to a SIMD model. To overcome this issue, the decision logic was moved to the next pipeline stage and to the pipeline forwarding lines, as described in Fig. 6. This new maximum instruction substitutes one compare and one branch instruction, therefore eliminating the pipeline flush and gaining 3 or 4 clock cycles, depending on whether the branch has delay slots or not.

The SIMD conditional branches were implemented by replicating the condition evaluation logic and by modifying it to separately evaluate all the elements in the vector, while the branch target address is calculated in the non-SIMD ALU.

Non-SIMD load/store instructions provide memory transfers with byte, halfword or word sizes. Since the SIMD vector can be greater than the processor word size (32-bits), a new vector transfer size had to be defined. This vector size is

![Fig. 5. Block diagram of the ALU SIMD module, illustrating the logic required to implement (a) the vector-vector and vector-scalar operations and (b) the inner-vector operations.](image)

![Fig. 6. The maximum decision logic is postponed to the next pipeline stage and to the pipeline forwarding lines.](image)
transiently considered by the newly defined SIMD load/store instructions, so that the instructions can work with different vector sizes. This contrasts with the non-SIMD load/store instructions, which only work with fixed transfer sizes.

Finally, the destination vector element of the new SIMD move instruction (defined as move-to-vector) is fetched just as in the store instructions, i.e. the scalar can be multiplexed to any specified position. The resulting vector will therefore retain all the other previous values.

C. Adaptation of the decoding unit

Since the encoding of most MB-LITE type A instructions has unused bit-fields, it was decided to assign the same opcode to the new SIMD arithmetic and the shift instructions as their non-SIMD counterparts, thus using such unused fields to distinguish them in the processor control unit. With such option, it was possible to re-use most of the decoding structures already implemented in the processor, except for a few control signals that had to be generated from such bit-fields. The maximum instruction, which did not exist in the original architecture, adopted the same opcode as the compare instruction, since the implemented logic operations are the same until the decision stage (see Fig. 6). The new SIMD branch instructions were also encoded by using unused bit-fields (co-located with the identification of the destination register), allowing an easy distinction from their non-SIMD counter-parts.

In contrast, unused opcodes were assigned to the SIMD load/store instructions, where the bit-field corresponding to an immediate operand was used to encode the memory address. The assigned opcodes are consecutive to those corresponding to the non-SIMD load/store, allowing for an easier decoding of the instruction. In a similar way, the move-to-vector instruction was also assigned an unused opcode, due to the absence of a similar instruction in the original ISA, and new decoding logic had to be added to the processor control unit.

D. Compiler Implementation

Three aspects are relevant concerning the compiler support: the front-end, by supporting high-level language features, in the form of special types or syntax; the middle-end, in the form of automatic program optimization; and, the back-end, by providing direct support for a specific ISA.

The conducted implementation of the compiler that was specifically developed for the proposed ASIP is based on the well-known GNU Compiler Collection (GCC) family [19]. The base GCC structure was extended to support the proposed architecture, in order to immediately allow programs to be written in the target processor’s Assembly Language (ASM). Since GCC already supports the MicroBlaze processor, adding the new mnemonics and opcodes was straightforward. At this stage, support is provided only at the back-end level. In the future, a particular attention will be also devoted to the middle-end, since it allows other optimizations using the Abstract Syntax Tree (AST). This will allow to further leverage the specific aspects of the underlying ISA.

V. MULTI-CORE PROCESSING PLATFORM

Many High Throughput Short Read (HTSR) sequencing applications require the alignment of multiple query sequences to one or more references. This requirement adds a new level of parallelism to the computation. Two possible approaches can be considered: i) a fine-grained data level parallelism, where a single core simultaneously processes different sequences using a SIMD processing model (one query/reference in each vector element); and ii) a thread level parallelism, where multiple cores perform parallel alignment of multiple query sequences on one or more references. The presented case-study followed the second of these approaches and considered the most interesting problem, where a single reference sequence exists. This requires the use of a shared memory model, similar to the one in [20], where the reference and query sequences are stored in a shared main memory (see Fig. 7). The computation is performed by: i) a work controller that is used to manage the work queue; ii) multiple processing elements that actually perform the alignment; and iii) a mechanism to gather the results from all processing elements.

To compute the alignment score for multiple query sequences, the architecture illustrated in Fig. 8(a) is proposed in this paper, which is composed of: a memory element, to store both the biologic sequences and the alignment scores; a master core, which is responsible for managing the work queue and to gather the results; multiple processing cores; a mutex circuit, to handle core synchronization. All elements are interconnected by a high-bandwidth communication interface. To reduce the amount of data that is transferred between the master and processing cores, the shared memory model studied in [21], which was developed for this specific application domain, was considered. It should be noticed that the current architecture does not include any cache level in the cores, thus alleviating the need for any explicit coherence mechanism. By using this approach, to initiate the sequence alignment between a reference sequence and a query sequence, the master core needs to communicate a minimal set of data to the target processing core, which consists of the address (in main memory) and the length of these sequences.

To perform the sequence alignment, each processing core is composed of the specialized SIMD ASIP presented in section IV, an instruction memory, a local (scratchpad) memory, a DMA controller and a network interface (see Fig. 8(b)).

Fig. 7. Processing architecture for multiple sequences alignment.

(a) General overview of the multi-core architecture.  (b) Schematic of the processing cores.

Fig. 8. Multi-core architecture for biologic sequences alignment.
A. Data communication

The proposed architecture supports several different types of interconnection mechanisms (e.g., shared bus, ring/mesh network on chip) with minimum changes in the base structure. The considered prototyping implementation, which is described in this manuscript, uses a shared bus and an arbiter to manage bus access contention. The adopted protocol is AMBA 3 AHB-Lite compatible, with multi-layer support, and requires a minimum of two clock cycles to transmit data through the bus: the first to request access to the bus and the second to transmit the data. Naturally, whenever the bus is unavailable (busy) when the access request is first made, additional cycles are required. To minimize the data transfer time, the bus arbiter also supports a burst mode, where a single bus request is used to transmit multiple data packets. In this case, a minimum of $n + 1$ clock cycles are required for transmitting $n$ data packets.

Furthermore, the processing cores were designed to reduce the contention in the system bus. For such purpose, the proposed architecture uses a local (scratchpad) memory to store temporary data and a DMA controller to handle most accesses to the main memory (e.g., prefetching the query or reference sequences to the scratchpad memory). This reduces the number of bus requests and allows hiding the communication time within computation. Although pooling is also allowed, the DMA controller can flag an interruption to the processor whenever a given copy request is finished. This allows to create an interrupt routine that handles all the data prefetching.

To ease programming (and compiler development), the processor has a memory mapped I/O organization. Thus, access to the DMA registers (for configuring DMA transfers and checking their status) or to the scratchpad memory can be performed by using the usual load/store instructions.

B. Synchronization mechanism

To allow an efficient cooperation between different cores, the multi-core architecture includes a multi-register mutex circuit, with each register supporting two states: locked by core $k$ and unlocked. The circuit works as follows: When one core tries to read from an unlocked mutex (register), a value of ‘1’ is returned and the mutex locks. After that, all other cores that read from such mutex receive the value ‘0’ until the first core unlocks it by writing the value ‘1’. All the write and read operations are atomic, assured by specific bus arbitration logic, which only allows one core to access a given mutex at a time.

VI. Experimental Results

To evaluate the proposed multi-core processing framework, a thorough performance analysis of the complete system is presented in this section. For this, it starts by analyzing the impact of the proposed SIMD instruction-set extension on the required hardware resources and on the processor’s maximum clock frequency. Afterwards, the ASIP performance speedup by is evaluated comparing the vanilla (sequential) version of the SW algorithm with Farrar’s [15] SIMD version. To further evaluate the results, the processor performance is compared with a high-performance GPP: the Intel Core i7 950, running at 3.07 GHz, with 6 GB of RAM. For a fair comparison, a single core is used to execute the same sequence alignment algorithm (Farrar’s implementation of the SW algorithm). It should be noted that no algorithmic changes were made, with the exception of those resulting from the ISA differences. Finally, the scalability of the proposed multi-core processing structure is analyzed when increasing the number of cores.

The complete system was prototyped in a Xilinx Virtex 7 FPGA (XC7VX485T), which is part of the Xilinx VC707 Evaluation Kit. To synthesize the design and perform the place-and-route, Xilinx ISE 14.4 was used. Accurate clock cycle measurements of the required time to execute the biological sequences analysis in the proposed platform was achieved by using Modelsim SE 10.0b. On the Intel architecture, cycle accurate measurements were obtained by using the PAPI library to read the processor performance counters.

A. Required hardware resources and timing analysis

To evaluate the performance and resources overhead introduced by the extended ISA, both the original MB-LITE and the proposed ASIP were implemented on a Virtex-7 FPGA. For a fair comparison, the multiplier and the barrel-shifter were deactivated in the original MB-LITE core.

The single core columns of Table II present the FPGA hardware resources and the maximum operating frequencies for both the MB-LITE and the ASIP cores. This table also presents the resources overhead of extending the MB-LITE ISA to support the new instructions. The number of LUTs increased approximately 5 times, mostly due to the extra logic that is required for the parallel arithmetic operations and the additional multiplexing logic. The number of registers also increased almost 3 times, due to the increase in the register size from 32 to 128 bits. Despite this increase in the resource usage, the total resources occupied by the ASIP on the FPGA is still less than 1%. As expected, the maximum operating frequency decreased by about 40 MHz (17%). These results demonstrate that the ISA extension had a reduced impact on the critical path of the core. It can also be ascertained by the post-place&route report that the clock period is now limited by the added multiplexing logic that is required to implement the SIMD instructions and the non-SIMD 32-bit adder.

B. SIMD instruction set evaluation

In order to demonstrate the performance improvement of the SIMD ASIP using the extended ISA, the number of clock cycles required to execute a DNA sequence alignment procedure were accurately measured. Furthermore, to validate the architecture, it was also compared against an Intel Core i7 950 processor, which is a superscalar core capable of multiple instruction issue, out-of-order and speculative execution. For this test, both the vanilla (sequential) and Farrar’s SIMD versions of the SW algorithm were considered. The sequence alignment code was compiled with GCC 4.6.2 (using the corresponding back-ends) using flags -O2 (sequential case) and -O (SIMD case), which revealed to be the most favorable parametrization for each case. For the considered benchmarking, the used DNA dataset is composed of several reference sequences ranging from 128 to 16384 elements and a query sequence of length 64. The reference sequences correspond to a random selection of sub-sequences of the Homo Sapiens
Table III presents the average execution time (in number of clock cycles), to execute the DNA sequence alignment with the proposed ASIP and the Intel Core i7. As it can be observed by analyzing the speedup columns in this table, the Intel Core i7 achieves a maximum speedup of 11.3x (average value of 10.5x), while the ASIP, with the proposed ISA, achieves a maximum speedup of 33.2x (average value of 31.2x). From this evaluation, it can be observed that the ASIP has a speedup factor about 3 times higher than the speedup obtained with the Intel Core i7, although its operating frequency is lower than 200 MHz, i.e., it runs only at 1/15 of the clock frequency of the Intel Core i7. This result demonstrates that the proposed extensions that were introduced in the ASIP are well tuned for operations commonly adopted in sequence alignment algorithms.

A second comparison can be made by analyzing the number of clock cycles per cell update for both the Intel Core i7 and the ASIP. By dividing the total number of clock cycles \( c \) by the length of the query and reference sequences \( m \) and \( n \), respectively, obtaining the number of clock cycles per cell update: \( c/(m \times n) \). For the SIMD implementations, the Intel Core i7 achieves a minimum of 1.3 cycles per cell update, whereas the ASIP achieves a best case of 2.0 cycles per cell update. While it may seem that the ASIP has a lower performance, it should be recalled that the Intel Core i7 uses macro-op fusion and multiple instruction issue to achieve up to 6 micro-ops per clock cycle [22], which results, on average, 2 Instructions Per Cycle (IPC) in the considered benchmarks. On the other hand, it is also important to recall that the ASIP SIMD register size was configured to a 128-bit width for the single purpose of ensuring a fair comparison, although it may be easily extended in order to increase the ASIP's performance. Hence, these results demonstrate that a simple RISC processor with the proposed set of SIMD instructions can be very effective to execute bioinformatics algorithms. This is especially important when considering the integration of multi-core structures in low-power embedded platforms to allow fast, portable and fully autonomous biological sequences analysis systems [11].

C. Scalability of the multi-core processing structure

The implementation of the defined scalable multi-core processing structure adopted the architecture of the ASIP, described in Section V, where all processing cores are executing Farrar’s SIMD version of the sequence alignment algorithm. To achieve such objectives, all the integrated scratchpad and shared memories were set to a 16-bit width. The considered benchmark dataset is the same as the one described in the previous subsection. In order to evaluate the shared bus contention, which constraints the multi-core scalability, the query profile used by the algorithm was stored in the shared memory, and each processing core requests access to the bus (once per iteration) to obtain the corresponding profile vector. Any other variables were stored in the scratchpad memory of each processing core.

The multi-core columns of Table II present the hardware resource usage and the maximum operating frequency of the multi-core processing structure, for different aggregates of processing cores. Due to the Block-RAM resource requirements of each core, a maximum of 38 processing cores can be instantiated on the prototyping FPGA. However, a 64-core configuration may be instantiated, without exceeding the available amount of slices, if a different memory configuration is used. The maximum operating frequency obtained for the multi-core processing structure presents a small decrease for the configurations with a greater number of processing cores.

Fig. 9 presents the obtained speedup values in what concerns the clock cycles and the total processing time of the proposed multi-core structure. Such speedup values were obtained by using a single ASIP core as the reference. The observed speedup increases almost linearly for configurations approaching the 32-ASIP cores configuration. However, the 64-ASIP cores configuration is limited by the FPGA’s resources and the obtained speedup is almost 1.5x lower than the 32-ASIP cores configuration.
up to 16 cores. With additional cores, the contention in the shared bus becomes a limiting factor, thus reducing the effectiveness of the extra cores and resulting in a sub-linear speedup increase. Still, when considering the initial non-SIMD sequential implementation as reference, the obtained results demonstrated that a 680x processing time speedup can be obtained with a 38-core parallel SIMD implementation of the proposed ASIP.

VII. CONCLUSION

The currently established biologic sequences alignment algorithms that allow the computation of the optimal alignment solutions by using DP techniques require large run-times when executed on current GPPs. In this work, it was proposed a new ASIP architecture specifically adapted for this class of algorithms that is able to achieve high processing throughputs through an optimized architecture that exploits both fine and coarse-grained parallelism.

The fine-grained parallelism is exploited by a new synthesizable processor core, featuring an extended ISA that introduces multiple specialized SIMD vector instructions. With this ASIP, a remarkable speedup of about 33x was achieved when the Farrar’s implementation (using the new ISA code generated by the developed back-end compiler based on GCC) was compared with a sequential version of the SW algorithm. This contrasts with an equivalent implementation using an Intel Core i7, based on the SSE2 instructions set, where a speedup of only 11x was achieved. These results demonstrate that the new ISA is especially fit for implementing biological alignment algorithms. Furthermore, the hardware resources overhead introduced by the new architecture is small, compared to the MB-LITE processor. Therefore, it is considered that the ASIP is a suitable RISC processor for high performance and low-power embedded systems, targeting the vast emerging set of biological data processing applications.

To exploit the coarse-grained parallelism, a multi-core computational structure composed by multiple instantiations of the proposed ASIP was developed and prototyped on a Xilinx Virtex-7 FPGA device. It was demonstrated that a linear speed up can be achieved with up to 16 processing cores, since no relevant contention on the interconnection bus exists. When the number of instantiated processors was further increased, a gradual (but expected) sub-linear behavior was observed in the attained speedup. Nevertheless, when considering the cumulative speedup resulting from using both the SIMD ISA and the muti-core architecture, the proposed system is capable of achieving significant processing time speedup values, as high as 680x with 38 cores, and clock cycle speedups up to 910x with 64 cores.

REFERENCES


