Multicore SIMD ASIP for Next-Generation Sequencing and Alignment Biochip Platforms

Nuno Neves, *Member, IEEE*, Nuno Sebastião, *Member, IEEE*, David Matos, *Member, IEEE*, Pedro Tomás, *Member, IEEE*, Paulo Flores, *Senior Member, IEEE*, and Nuno Roma, *Senior Member, IEEE*

***Abstract*— Targeting the development of new biochip platforms capable of autonomously sequencing and aligning biological sequences, a new multicore processing structure is proposed in this manuscript. This multicore structure makes use of a shared memory model and multiple instantiations of a novel application-specific instruction-set processor (ASIP) to simultaneously exploit both fine and coarse-grained parallelism and to achieve high performance levels at low-power consumption. The proposed ASIP is built by extending the instruction set architecture of a synthesizable processor, including both general and special-purpose single-instruction multiple-data instructions. This allows an efficient exploitation of fine-grained parallelism on the alignment of biological sequences, achieving over 30× speedup when compared with sequential algorithmic implementations. The complete system was prototyped on different field-programmable gate array platforms and synthesized with a 90-nm CMOS process technology. Experimental results demonstrate that the multicore structure scales almost linearly with the number of instantiated cores, achieving performances similar to a quad-core Intel Core i7 3820 processor, while using 25× less energy.**

***Index Terms*— Application-specific instruction-set architecture, biochip platforms, biological sequences alignment, multicore architecture, single-instruction–multiple-data (SIMD).**

I. Introduction

BIOCHIPS are probably one of the most important biotechnology contributions from the last decade, being considered a highly viable and prominent means to identify and detect different biological analytes, such as deoxyribonucleic acid (DNA), proteins, toxins, hormones, bacteria, and so on. One particular application of these lab-on-chip devices is the detection of specific genetic biomarkers (e.g., DNA sequences that cause a disease or are associated with susceptibility to a given disease), being recognized as a highly useful means in several application domains, such as personalized medicine, preliminary diagnosis and prognosis devices [1], [2]. In these applications, the sequence data, corresponding to the specific biomarker that is aimed to be identified, are stored in an on-chip memory, at configuration time, and usually correspond to a specific part of a genome or gene related with the disease/mutation that is being diagnosed (e.g., identification of cancer susceptibility genes, cardiological pathologies, etc.).

However, these prominent applications have been posing new challenges, requiring the anticipation of several common and often required postprocessing analysis steps to the biochip level. Among them, biological sequence alignment, close to or even within the biochip [3], has been regarded as a very promising challenge. Accordingly, the presented research envisages the fulfillment of the current processing gap in embedded sequence alignment platforms, thus foreseeing the development of fast, portable, disposable, and fully autonomous biological sequence analysis systems [4] at the point-of-care [5], instead of being restricted to large centralized facilities. To achieve this objective, the usage of programmable solutions is regarded as highly necessary to comply with adaptability and flexibility requirements, and contrasts with other dedicated architectures that have also been proposed in the last years [6]–[9].

Moreover, autonomous and portable devices are highly limited in what concerns their application requisites, contrary to what happens with conventional high-performance computing (HPC) solutions. Besides the imposed constraints concerning compactness, cost-efficiency, high production yields, and robust functionality, the most stringent constraint is usually concerned with their low-power consumption [10]. However, the involved amount of data and the complexity of the required computations rarely permit any significant lightening of the computational capabilities. As such, the main motivation of the presented research is the development of an efficient architecture for biochips with highly limited power constraints (up to a few watts) and whose performance should be as close as possible to that of a state-of-the-art general purpose processor (GPP). To verify the fulfillment of such a promising goal, the processing performance offered by the proposed low-power biological sequence alignment processor will be compared with that of an Intel Core i7 processor, which, despite its high-processing performance, is completely unsuited for embedded biochip purposes, due to its much greater power requisites (minimum 38 W with a single-thread execution).

The biochip under development is a heterogeneous multicore system-on-chip (SoC), composed of two types of highly optimized processor elements (PEs): 1) Type A PEs implement a preliminary heuristic filtration phase, using indexed exact search algorithms (e.g., k-mers, FM-index), where fragments

Manuscript received July 29, 2013; revised April 4, 2014; accepted June 9, 2014. Date of publication July 18, 2014; date of current version June 23, 2015. This work was supported in part by the National Funds through the Fundação para a Ciência e a Tecnologia through the Project entitled Heterogeneous Multi-Core Architecture for Biological Sequence Analysis (PTDC/EEA-ELC/113999/2009), and in part by the Project entitled Threads: Multitask System Framework With Transparent Hardware Reconfiguration (PTDC/EEA-ELC/117329/2010) and by Project PEst-OE/EEI/LA0021/2013.

The authors are with the Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento em Lisboa, Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal (e-mail: nuno.neves@inesc-id.pt; nuno.sebastiao@inesc-id.pt; david.matos@inesc-id.pt; pedro.tomas@inesc-id.pt; paulo.flores@inesc-id.pt; nuno.roma@inesc-id.pt).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2014.2333757

1063-8210 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

of the patient's sample are looked up in the biomarker-indexed sequence (stored in memory) to identify a partial match and to determine its approximate location(s) in the biomarker sequence; and 2) Type B PEs implement the subsequent computational demanding sequence alignment phase (allowing insertions, deletions, and substitutions) of the subset of samples identified in the first phase, using an exhaustive dynamic programming (DP) approach applied to the delimited region(s) of the biomarker sequence identified in the first phase.

Among the several design options, the development of an efficient application-specific instruction-set processor (ASIP) architecture for Type B PEs, specifically adapted for optimal biological sequence alignment algorithms, is regarded as a particularly suited approach to comply with all the performance and design requisites enumerated above [11]. The attained processing throughput is achieved as a result of a twofold contribution: 1) inclusion of multiple specialized single-instruction multiple-data (SIMD) vector instructions in the processor instruction-set architecture (ISA), to extensively exploit fine-grained parallelism; and 2) support for an extensive multicore computational structure, composed of multiple instantiations of the designed ASIP, to efficiently exploit coarse-grained parallelism. The cumulative result of these two important contributions was demonstrated with different field-programmable gate array (FPGA) prototypes of the proposed multicore SIMD ASIP, which proved capable of offering speedup values as high as 720×.

To further demonstrate that the proposed multicore ASIP complies with the performance and efficiency requirements of the target application domain, it was also implemented in a 90-nm CMOS process technology. Experimental results indicate that the proposed solution achieves a performance similar to that of HPC processors (e.g., an Intel Core i7) while using 20× less energy.

The manuscript is organized as follows. After the introductory motivation presented above, Section II presents a brief overview on biological sequences alignment algorithms, as well as on their current state-of-the-art SIMD implementations. In Section III, the new and adapted ISA for this specific application domain is presented, while its architecture implementation is presented in Section IV. In Section V, the developed multicore processing structure, composed of multiple instantiations of the designed ASIP is presented. Section VI presents the obtained experimental results and Section VII concludes the presentation with the enumeration of the most important contributions.

II. Biological Sequences Alignment

The alignment strategy that was adopted in the developed biochip platform follows the same approach that is usually applied in conventional HPC solutions. The patient's samples are firstly applied to a suboptimal heuristic-based method [12], [13], which uses an index data structure of a specific biomarker (previously stored in on-chip memory) to identify a potential partial match and to determine its approximate location(s) in the biomarker sequence. These suboptimal methods offer fast and preliminary solutions at the cost of a reduced sensitivity. Such preliminary indexed search is then refined by applying an optimal DP alignment method, such as the Needleman–Wunsch [14] and Smith–Waterman (SW) [15] algorithms.

A. Optimal Alignment Algorithm

The SW algorithm [15], characterized by an $\mathcal{O}(nm)$ time complexity, is a widely established dynamic programming (DP) algorithm used to obtain the local alignment between a query sequence (q) and a reference/database 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 typically filled using an affine gap penalty model [16], given by the following equation:

$$H(i,j) = \max \begin{cases} H(i-1,j-1) + Sbc(q[i], d[j]) \\ E(i,j) \\ F(i,j) \\ 0 \end{cases}, \quad \text{with}$$

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

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

where α and β represent the cost of gap opening and extension, and $\mathrm{Sbc}(q[i], d[j])$ denotes the substitution score value when aligning character $q[i]$ against character $d[j]$. The initial conditions are given by $H(i,0) = H(0,j) = E(i,0) = F(0,j) = 0$. The strict data dependencies of this algorithm require a prior processing of the neighboring upper, left, and upper-left cells, before calculating the value of a given cell.

Several solutions have been proposed to accelerate the execution of the SW algorithm. The processing structures that offer the highest performance typically adopt the form of dedicated accelerators like those proposed in [6]–[9]. However, these solutions lack the necessary flexibility to allow the execution of other similar algorithms (e.g., Needleman–Wunsch [14]) or even variations of the same algorithm (e.g., banded SW algorithm used in FASTA [13]).

On the other hand, it is also desirable to develop processing structures that are easily adapted to detect/identify distinct patterns, mutations, or anomalies. As an example, while some pathologies might be identified by a single detection of a given biomarker pattern, other anomalies are only identified by a successive repetition of a specific pattern [1]. Therefore, whenever flexibility is a system requirement, the usage of programmable processing solutions proves to be highly convenient. Among these, GPPs are a natural and commonly used platform to execute these algorithms, especially due to the set of SIMD instruction-set extensions that are available 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.

B. SIMD Implementations

To accelerate the alignment procedure, while still ensuring the optimal alignment, several SIMD parallelizations of the

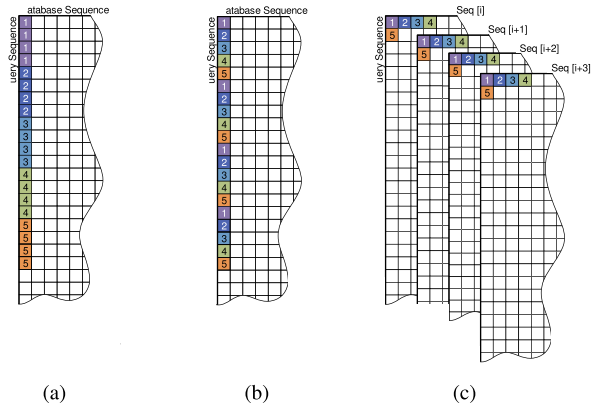


Fig. 1. SIMD implementations of the SW algorithm [15]. The first five SIMD iterations were numbered and represented with different gray levels. For simplicity, only four data elements are shown in each SIMD register. (a) Rognes and Seeberg [17]. (b) Farrar [19]. (c) Rognes [18].

SW algorithm have been presented. The most recent implementations, based on the exploitation of Intel SSE instruction set extensions, were presented in [17]–[19]. Their differences mainly lie in the adopted data processing pattern (Fig. 1).

The first implementation that was proposed in [17] precomputes a profile for the entire reference sequence. With such a technique, a vector of cells parallel to the query sequence can be simultaneously processed by each SIMD instruction [Fig. 1(a)]. The compliance with 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 an approach still implies the introduction of conditional branches in the inner loop of the algorithm, thus making the execution time-dependent on the used scoring matrix and gap penalties.

Farrar [19] also adopted a precomputed profile (of the query), but improved the processing scheme using a striped access pattern [Fig. 1(b)], 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 an SIMD register. As an example, when 128-bit SSE2 registers are considered to process 8-bit data elements, p equals 16. The length of each segment is given by $t = \lfloor (m + p - 1)/p \rfloor$, where padding zeros are inserted whenever the query size (m) is not long enough to completely fill all the segments.

This striped computation of the score matrix is fulfilled by assigning each SIMD vector element to one distinct segment. 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 an SIMD register the following query symbols: $\{q[2], q[t+2], q[2t+2], q[3t+2], \ldots, q[15t+2]\}$. 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, 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 [19] one of the fastest SIMD implementations of the SW algorithm. For this reason, this algorithm is integrated in many current high-performance alignment frameworks, such as the latest versions of SSEARCH [20].

Meanwhile, Rognes [18] 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 [Fig. 1(c)], which allows additional speedups when the underlying application aligns a specific and single query to a database of reference sequences.

When transposing this latest Rognes' approach to the application scenario under consideration, the specific biomarker sequence that was assigned to the considered biochip configuration would be the natural choice for the query sequence that is simultaneously aligned (in parallel) with several references, corresponding to the set of fractions of the patient's samples identified in the first suboptimal phase. However, this approach would also require the score profile of each identified fraction of the patient's reference to be calculated at runtime, which results in a significant performance penalty. On the other hand, with Farrar's algorithm, it is possible to calculate the score profile of the biomarker sequence in a preconfiguration step and store it in the biochip memory. Afterward, the biomarker sequence score profile can be directly used in the alignment with each of the identified fractions of the patient's reference sequence. Hence, when compared with the Rognes' approach, Farrar's saves valuable execution time and energy when aligning query sequences of smaller sizes (as is the case of the considered application domain), since the profiles do not need to be calculated at runtime. Therefore, Farrar's algorithm is not only the adequate choice for the considered application domain, but also presents itself as the solution with the broadest range of applications, thus broadening the scenarios where a system that accelerates its execution can be applied.

III. Dedicated SIMD Instruction Set for Biological Sequences Alignment

The proposed ISA was defined targeting the acceleration of the classic local and global sequence alignment procedures (such as the several SIMD implementations of the Needleman–Wunsch and SW algorithms [17], [19], [21]). Due to the high-performance and low-energy constraints of the considered biochip platform, mainly targeted for portable and autonomous biological sequence alignment of next-generation sequencing (NGS) [22] data,

Farrar's SIMD implementation [19] was considered the most suited for this specific application domain. Nevertheless, since the proposed ISA includes both general-purpose and special-purpose SIMD instructions, it can be easily used to implement a broader range of algorithms, including other operations with biological sequences (e.g., hidden Markov models (HMMs), Viterbi chains, etc.), which heavily rely on DP methods.

A. SIMD Sequence Alignment Implementation

By analyzing the original Farrar's pseudocode definition [19] [Fig. 3(a)], it is clear that the adoption of vector arithmetic instructions will potentially accelerate this algorithm implementation. These instructions should not only speed up the operations between vectors, but they may also facilitate the various 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 vector element shift instructions. Furthermore, since all these instructions will be dealing with SIMD vectors, it is also advantageous to include specialized memory access instructions, to handle vector-sized variables.

On the other hand, from a more detailed inspection of an Intel SSE2 assembly implementation [19] [Fig. 3(b)], it can also be observed that the lazy loop condition assertion requires at least five 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 on 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 should also be defined, to accommodate both the vector and scalar operands. These two aspects will be briefly covered in the following two subsections.

B. SIMD Registers

The proposed instruction set and the corresponding datapath (Section IV) can be fully parameterized in terms of the adopted register and vector-element sizes. To obtain a fair comparison with Farrar's [19] SSE2 implementation, the base parameterization that will be considered in this particular implementation uses the same register and vector-element sizes as those of the Intel SSE2 (used by Farrar [19]), i.e., 128-bit registers, with 8 or 16 elements. Nevertheless, different register and element sizes can be used, as shown in Section VI-C, where experimental results considering different register sizes are presented.

To simplify the implementation of the proposed instruction set, it is assumed that all processor registers within the register bank have the same size. Hence, any scalar (non-SIMD) instruction will only operate over the least significant part of the register, corresponding to a scalar processor word.

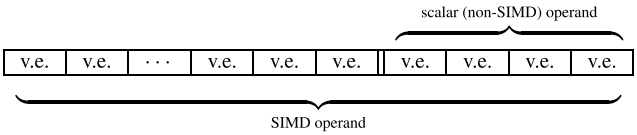


Fig. 2. Division of each processor register into a parameterizable number of SIMD vector elements (v.e.). Scalar operands coexist in the same register, occupying the least significant word.

TABLE I
PROPOSED SIMD SPECIALIZED INSTRUCTION SET FOR BIOLOGICAL SEQUENCE ALIGNMENT

ARITHMETIC			LOGIC		MEMORY	
Vector-Vector	*Vector-Scalar*	*Inner-Vector*	sllv	mtv	lv	sv
addvv	addvs	addv			lvi	svi
saddvv	saddvs		CONTROL			
rsubvv	rsubvs	rsubv	beqv	beqdv	beqiv	beqidv
srsubvv	srsubvs		bnev	bnedv	bneiv	bneidv
			bgev	bgedv	bgeiv	bgeidv
cmpvv	cmpvs	cmpv	bgtv	bgtdv	bgtiv	bgtidv
cmpuvv	cmpuvs	cmpuv	blev	bledv	bleiv	bleidv
maxvv	maxvs	maxv	bltv	bltdv	bltiv	bltidv
maxuvv	maxuvs	maxuv	bmev	bmedv	bmeiv	bmeidv

In contrast, all the proposed SIMD instructions will operate over the entire register (Fig. 2). With this design option, the critical path of the processor is confined within the datapath corresponding to the scalar (non-SIMD) operations, thus making it independent of the extended SIMD register size.

C. Proposed SIMD Instruction Set

The specialized ISA that is herein proposed defines 56 SIMD instructions for arithmetic, logic, memory access, and control operations. The set of proposed SIMD instructions, which are used in Farrar's algorithm implementation, are depicted in Table I. The instructions are subdivided into three classes: vector–vector, operating over the corresponding pairs of vector elements in each SIMD register; vector–scalar, operating between one SIMD register and the scalar 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, scalar load and store instructions will only operate over the processor scalar 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 scalar operands and only the destination (`LV`) or the origin (`SV`) are SIMD operands. There is also a special move instruction to write a scalar value to a specific SIMD vector position, keeping the remaining positions unchanged. The position and the value are provided in scalar operands.

Furthermore, by considering the requirements that were referred to in Section III-A, particular attention was also devoted to the definition of a new subset of control instructions that can be applied in a variety of applications and are especially interesting for the SW algorithm. The need arises from the significant predominance of loop statements in the mentioned DP algorithms (generally implemented with conditional branch instructions) and takes into account the

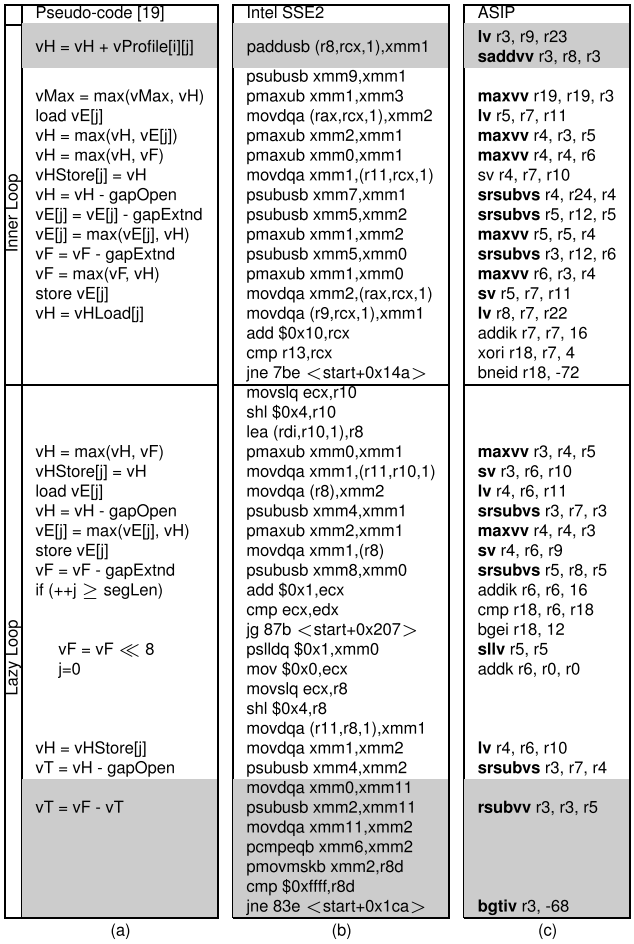


Fig. 3. Farrar's SIMD implementation [19] of the SW algorithm. (a) Pseudocode definition. (b) Intel SSE2 assembly. (c) Proposed ISA assembly code. Instructions outlined in bold face belong to the specialized ISA. Shaded areas outline the blocks of identical operations that require a different number of instructions to complete in the two implementations. Only the inner and lazy loops are illustrated in this figure.

severe penalties that these control instructions impose on deep pipeline architectures. As a consequence, the 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 will be described in Section IV. These restrictions determined the adoption of shallower pipeline structures, contrasting with modern superscalar GPP architectures, e.g., Intel and ARM. 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 taken if the condition is either verified for all of them or for at least one of them. The branch target is computed with scalar operands.

Fig. 3 presents a mapping of Farrar's [19] algorithm and the corresponding implementations with the Intel SSE2 ISA and with the proposed SIMD ISA. By comparing the two implementations, an immediate gain is observable with the proposed ISA, with more visible advantages in the lazy loop. The major contributor to this reduction is the proposed set of vectorized control instructions that significantly reduce the control overhead.

Another significant advantage of the proposed ASIP results from the adoption of a strict reduced instruction set computer (RISC) paradigm based on a shallow pipeline structure, contrasting with Intel's deep pipeline complex instruction set computer (CISC) model [23]. 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 will be demonstrated in Section VI.

IV. SIMD Processor Architecture

The MB-LITE [24] 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 [25], 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 support the proposed ISA. The reduced hardware resources required by this core were also taken into account, due to the fact that it will be used as the base for a scalable multicore processing platform, which exploits coarse-grained parallelism, as will be described in Section V.

A. MB-LITE Processor Architecture

The MB-LITE [24] 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 [25], MB-LITE has two basic types of instructions: 1) Type (register type); and 2) Type (immediate type). Consistently, all instructions have three operands (three registers, or two registers plus one immediate), except the control and shift instructions, which have two operands (two registers, or one register plus one immediate). According to the MicroBlaze architecture, the register bank is also composed of 32 general-purpose registers with three read-ports and one write-port.

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, to improve the memory writes.

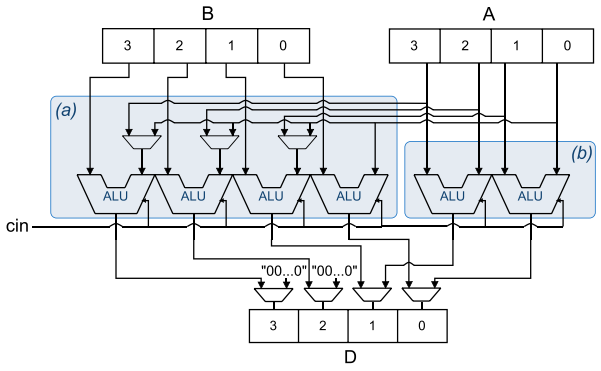


Fig. 4. Block diagram of the SIMD ALU module, for example, 4-element vector configuration, illustrating the required logic to implement. (a) Vector–vector and vector–scalar operations. (b) Inner-vector operations.

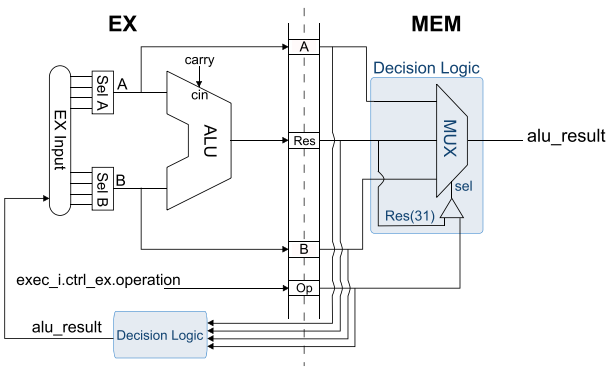


Fig. 5. Maximum decision logic is postponed to the next pipeline stage and to the pipeline forwarding lines.

Among the provided modules, MB-LITE [24] 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 the Standard output (STDOUT) can be easily used in the software development phase.

B. Modification of the Execution Unit

To support the proposed SIMD ISA, the execution unit had to be modified by extending its original ALU to include a new SIMD module.

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

The addition and subtraction operations require one adder per SIMD vector element together with some extra multiplexing logic (Fig. 4). 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. Furthermore, to reduce the complexity of the control logic and to maintain the processor's critical path, the inner-vector operations are performed by an independent set of ALU modules, as it is possible to observe in Fig. 4(b).

The SIMD maximum instruction is based on the compare instruction, comprising a subtraction followed by a signal evaluation. Therefore, the same logic is used to implement both of these instructions, requiring only a multiplexer, selected by the most significant bit 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 an 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. 5. This new maximum instruction substitutes one compare and one branch instruction, thereby eliminating the pipeline flush and gaining three or four 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 scalar ALU (Fig. 6).

Scalar 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 transparently considered by the defined SIMD load/store instructions in terms of the transferred data size so that the instructions can work with the different vector sizes. This contrasts with the scalar 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 selected just as in the store instructions, i.e., the scalar value can be multiplexed to any of the SIMD vector elements. Therefore, the resulting SIMD vector will retain the previous values in all other elements.

C. Adaptation of the Decoding Unit

Since the encoding of most original MB-LITE Type A instructions has unused bit-fields, it was decided to assign the same opcode to the new SIMD arithmetic and shift instructions as their scalar counterparts, thus using such unused fields to distinguish them in the processor control unit. With such options, it was possible to reuse 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 (Fig. 5). The new SIMD branch instructions were also encoded using unused bit-fields (co-located with the identification of the destination register), allowing an easy distinction from their scalar counterparts.

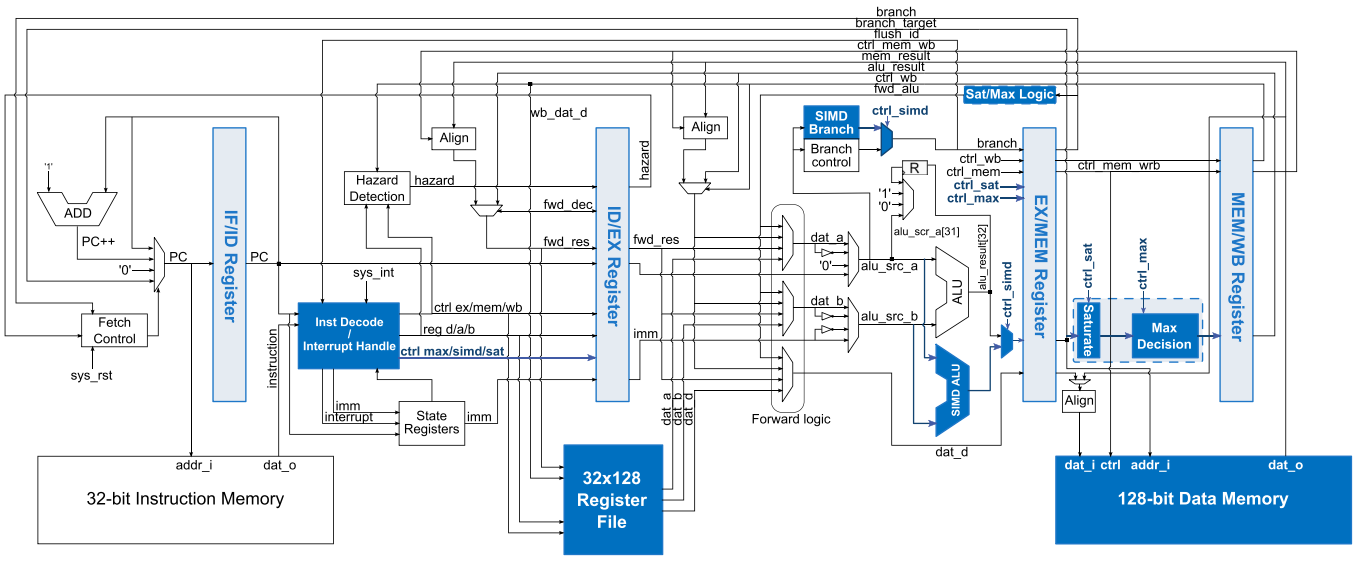


Fig. 6. Proposed ASIP pipelined architecture. The new and modified modules were highlighted in the block diagram. The decoding logic of the proposed SIMD instructions is included in the Instruction Decode/Interrupt Handle block. The SIMD addition, subtraction, compare, shift, and move instructions are implemented in the SIMD ALU block. Note the replication of the saturation detection logic and of the maximum decision logic in the forwarding lines from the MEM stage.

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 scalar load/store, allowing for an easier decoding of the instruction. The SIMD move instruction was also assigned an unused opcode, due to the absence of a similar instruction in the MB-LITE ISA. As a consequence, new decoding logic had to be added to the processor control unit.

The final ASIP pipelined architecture, with the above described modifications, is presented in Fig. 6.

D. Compiler Implementation

Writing programs directly in machine language, i.e., byte code, is not practicable and would be a serious limitation to the ASIPs usability. A compiler is thus a critical tool because it not only eases the task of writing programs, but also allows for some code optimizations and loosens the coupling between the program and the processor's implementation, ideally rendering low-level changes transparent to the programmer.

Accordingly, 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 [26]. The base GCC structure was extended to support the proposed architecture, 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. Currently, support is provided only at the back-end level. In the future, particular attention will also be given to the middle-end, since it allows other optimizations using the abstract syntax tree (AST). This will allow for further leverage of the specific aspects of the underlying ISA, including automatic vectorization. Nonetheless, support for the C/C++ programming languages can be provided by relying on intrinsic functions that directly use the SIMD instructions.

V. Multicore Processing Platform

The integration of the proposed ASIP within the biochip SoC follows the typical GPP + accelerator co-design approach. The GPP targets the coordination of the two processing steps of the biochip: 1) preliminary heuristic search phase, using exact search algorithms, where fragments of the patient's sample are looked up in the biomarker-indexed data structure (stored in memory); and 2) optimal sequence alignment phase (allowing insertions, deletions, and substitutions) of the subset of samples identified in the first phase.

Accordingly, the proposed ASIP architecture aims to accelerate the second phase, where a significant amount of fractions, preliminarily identified in the first processing phase, are efficiently processed by simultaneously exploiting both fine- and coarse-grained parallelism models: the former using the proposed SIMD ASIP, while the latter by dividing the set of patient's samples among the instantiated cores. This allows the creation of multiple jobs (coarse-grained parallelism), each executed in parallel by an instance of the proposed ASIP using the proposed SIMD instructions (fine-grained parallelism).

To apply this processing scheme, a shared memory model similar to the one in [27] is used, where both the patient's samples and the biomarker sequence are stored in a shared main memory (Fig. 7). The computation is performed using: 1) a work controller that manages the work queue, where each item corresponds to the alignment of one sample to the biomarker sequence; 2) multiple processing elements that

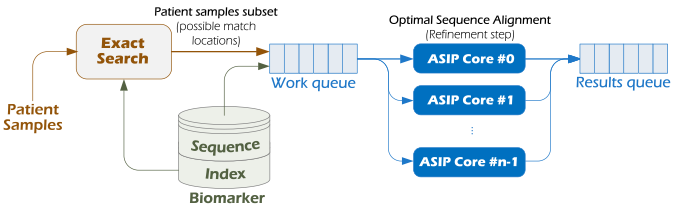


Fig. 7. Processing scheme for the alignment of multiple sequences.

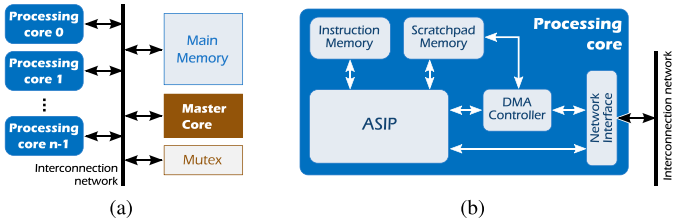


Fig. 8. Multicore architecture for biological sequences alignment. (a) General overview of the multicore architecture. (b) Schematic of the processing cores.

actually perform the alignment; and 3) a mechanism to gather the results from all processing elements.

Nevertheless, it is worth noting that other possible parallelization models could equally be applied. As an example, in a different (and more conventional) application domain, where the alignment of a single query sequence to a large reference sequence is required, the processing could be split into small stripes that would be processed in pipeline: each thread i would compute one single stripe and communicate the result to the following thread $i+1$.

The considered parallel processing structure is implemented using the multicore architecture described in Fig. 8(a). Such architecture is composed of: 1) a memory element, to store both the biological sequences and the alignment scores; 2) a master core (GPP), which is responsible for managing the work queue and to gather the results; 3) multiple processing cores, to perform the score calculations; 4) a mutex circuit, to handle core synchronization; and 5) a high-bandwidth interconnection, to handle the required communications. To reduce the amount of data that is transferred between the master and the processing cores, the shared memory model studied in [28], which was developed for this specific application domain, was considered. It should be noted that the current architecture does not include any cache level in the cores, thus alleviating the need for any explicit coherence mechanism. Using this approach, the master core only needs to communicate a minimal set of data (consisting of the addresses, in the shared memory, and length of the involved sequences) to start the alignment between a reference and a query sequences.

Each of the processing cores is composed of the specialized SIMD ASIP presented in Section IV, an instruction memory, a local (scratchpad) memory, a direct memory access (DMA) controller and an on-chip network interface [Fig. 8(b)].

A. Data Communication

The proposed architecture supports various types of interconnections (e.g., shared bus, ring/mesh network-on-chip, etc.) with minimum changes in the base structure. The considered prototyping implementation, which is described in this manuscript, uses a shared bus coupled with an arbiter to manage the access to the bus. The adopted bus protocol is AMBA 3 AHB-Lite compatible (with multilayer support), requiring a minimum of two clock cycles to transmit the data: the first to request access to the bus and the second to transmit the data. Naturally, whenever the bus is unavailable (busy), additional clock 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.

To further reduce the contention in the system bus, each processing element maintains temporary data in its local scratchpad memory, being its internal DMA controller responsible for handling accesses to the main memory (e.g., to prefetch the query or reference sequences to the scratchpad memory). This approach reduces the number of bus requests and allows to hide the communication time with the computation time. The DMA controller can also flag an interruption to the ASIP whenever a given copy request is finished. This allows the creation of an interrupt routine that handles all of the data prefetching operations.

To ease the programming task (and compiler development), the ASIP adopts a typical memory mapped I/O organization. Therefore, access to the DMA registers (to configure the DMA transfers and check their status) or to the scratchpad memory can be performed using the usual load/store instructions.

B. Synchronization Mechanism

To allow an efficient cooperation between the different cores, the multicore architecture includes a multiregister mutex circuit, with each register supporting two states: *locked by core k* and *unlocked*. This circuit works as follows: when one core attempts 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 initial core unlocks it by writing the value 1. All of the write and read operations are atomic, assured by specific bus arbitration logic, thus guaranteeing that only one core has access to a given mutex at any given time.

VI. Experimental Results

To evaluate the proposed ASIP as well as its integration in the multicore processing framework, a thorough performance analysis of the complete system is presented in this section. The first analysis evaluates the impact of the proposed SIMD instruction-set on the required hardware resources and on the processor's maximum clock frequency. Afterward, the attained processing speedup is evaluated by comparing the executions of the vanilla (sequential) version of the SW algorithm and of Farrar's [19] SIMD version. To assess the ASIPs performance, several implementations of the processor [in different field-programmable gate arrays (FPGAs)] will be compared with two off-the-shelf low-power processors: 1) an Intel Atom E665C, running at 1.3 GHz; and 2) an ARM Cortex-A9,

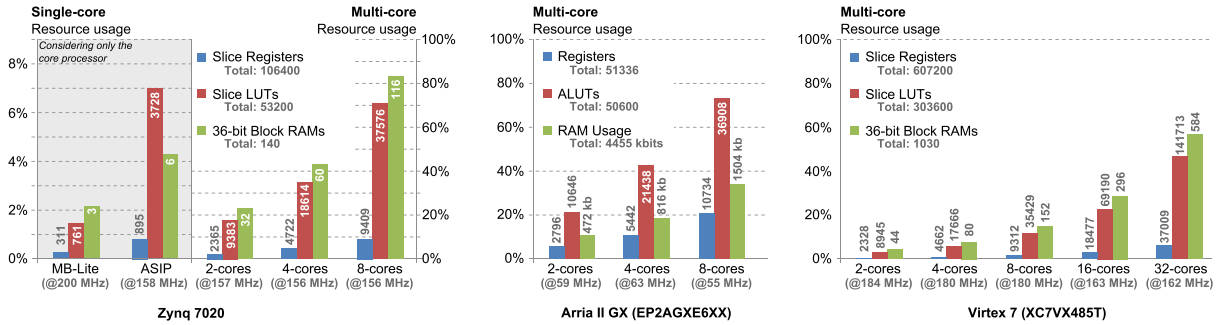


Fig. 9. Required hardware resources and corresponding operating frequency of the multicore system when implemented in different prototyping platforms.

running at 533 MHz. Furthermore, the ASIPs performance will also be compared to that of a high-performance processing solution, which, despite its high throughput, is unable to be used in low-power environments, as those of autonomous and mobile platforms. In particular, the proposed low-power multicore processing structure was synthesized targeting a 90-nm CMOS process technology and compared to a state-of-the-art Intel Core i7 3820, running at 3.6 GHz with a power consumption of 38 W for a single core. For a fair comparison, no algorithmic changes have been introduced in any cases, apart from those resulting from the ISA differences. Finally, the scalability of the proposed multicore structure with the number of instantiated processing cores is also analyzed by considering a raw performance metric [number of cell updates per second (CUPS)], a raw energy metric [number of cell updates per Joule (CUPJ)], and an energy-delay product-related metric [cell updates per Joule-second (CUPJS)].

To evaluate the performance and connectivity of the proposed multicore processing structure with the master core of the biochip platform, the complete alignment system was also prototyped in three different platforms: 1) a Xilinx Zynq 7020 SoC, comprising reconfigurable logic and a dual-core ARM Cortex-A9 processor connected through an AMBA 3 (AXI) link, making it a particularly interesting prototyping platform for developing mobile biochip systems, as stated in [29]; 2) a Kontron Microspace MSMST single-board computer composed of an Altera Arria II GX FPGA (EP2AGXE6XX) and an Intel Atom E665C connected through a 4× PCIe Gen 1 link; and 3) a personal computer composed of a Xilinx Virtex 7 FPGA (XC7VX485T) connected to an Intel Core i7 3820 through an 8× PCIe Gen 2 link. The synthesis and place-&-route for the FPGAs were performed using Xilinx ISE 14.4 and Altera Quartus II v12.0. Accurate clock cycle measurements of the required time to execute each biological sequences analysis in the proposed platform was achieved using Modelsim SE 10.0b. On the Intel Core i7, cycle accurate measurements were obtained using the precision approach path indicator (PAPI) library to read the processor performance counters. For the Intel Atom and ARM Cortex-A9, the system timing functions were used to determine the total execution time of the DNA sequence alignment. To improve the measurement accuracy, several repetitions of the same alignment were done. The obtained values were subsequently divided by the number of repetitions and the processor clock frequency.

A. Biological Benchmarking Setup

In the considered evaluation and performance analysis, the used DNA dataset is composed of several reference sequences, ranging from 128 to 16 384 base pairs, and a number of query sequences, ranging from 20 to 2276 base pairs. The reference sequences correspond to 20 indexed regions of the Homo sapiens breast cancer susceptibility gene 1 (`BRCA1` gene) (`NC_000017.11`). The query sequences were obtained from a set of 22 biomarkers for diagnosing breast cancer (`DI183511.1` to `DI183532.1`) and a fragment, with 68 base pairs, of the `BRCA1` gene with a mutation related to the presence of a Serous Papillary Adenocarcinoma (`S78558.1`).

B. Required Hardware Resources and Timing Analysis

To evaluate the attained performance and the resources overhead introduced by the proposed SIMD ISA, the original MB-LITE processor and the proposed ASIP were implemented on the previously referred prototyping devices, namely, the Zynq 7020 SoC, the Arria II GX FPGA, and the Virtex-7 FPGA. For a fair comparison, the multiplier and the barrel-shifter were deactivated in the original MB-LITE core. This way, the proposed ASIP presents maximum operating frequencies of 187, 158, and 115 MHz, on the Virtex-7, Zynq 7020, and Arria II GX devices, respectively.

The left chart of Fig. 9 presents, for both the MB-LITE and the ASIP core on the Zynq SoC, the hardware resources and the maximum operating frequencies. As it can be observed, the number of LUTs increased approximately 5×, mostly due to the extra logic that is required for the parallel arithmetic operations and the additional multiplexing logic. The number of registers and RAM/FIFO blocks duplicated because the increase in the register size, from 32 to 128 bits, requires the use of two block-RAMs per read-port, instead of just one. On the other hand, the processor maximum operating frequency decreased by about 42 MHz (21%), mostly due to the added multiplexing logic that is required to implement the SIMD instructions. Nonetheless, this decrease is by far compensated by the larger than 30× increase of the ISAs performance, as will be described in the following subsections.

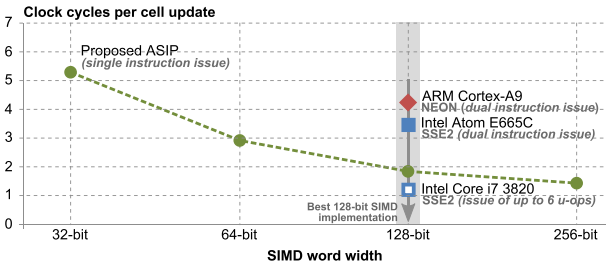
The remaining charts of Fig. 9 present the hardware resources occupied by the proposed multicore ASIP implementations. As expected, these results follow an almost linear relation with the number of instantiated cores.

Query Size [α]	Intel Atom E665C Clock Cycles [×10^6] (Sequential)	(SSE2)	Speedup	ARM Cortex-A9 Clock Cycles [×10^6] (Sequential)	(NEON)	Speedup
20	5.11	1.037	4.93	2.13	1.154	1.85
68	18.77	1.192	15.74	6.84	1.373	4.98
74	19.68	1.170	16.82	7.59	1.339	5.67
85	21.65	1.224	17.69	8.47	1.470	5.76
94	23.84	1.207	19.76	9.38	1.373	6.83
685	166.32	3.826	43.46	65.57	6.303	10.40
1861	446.73	8.616	51.85	180.15	16.262	11.08
2276	545.28	10.274	53.07	219.02	19.491	11.24
Average [β]			**18.47**			**5.09**

Query Size [α]	Intel Core i7 3820 Clock Cycles [×10^6] (Sequential)	(SSE2)	Speedup	Proposed ASIP Clock Cycles [×10^6] (Sequential)	(128-bit SIMD)	Speedup
20	2.13	0.384	5.55	6.54	0.307	21.28
68	7.12	0.606	11.75	22.12	0.555	39.85
74	7.59	0.429	17.71	24.10	0.543	44.40
85	8.01	0.480	16.70	27.62	0.631	43.78
94	9.19	0.487	18.88	30.56	0.627	48.74
685	54.90	1.504	36.50	222.62	3.375	65.95
1861	151.10	3.530	42.81	604.19	8.848	68.29
2276	189.49	4.163	45.52	739.11	10.744	68.79
Average [β]			**16.67**			**37.53**

[α] Illustrative biomarker subset regarding the speedup variation when aligning to a 4092 base pairs reference; [β] Average speedup corresponding to the alignment of the complete biomarker set with the entire reference sequence set.

(a)



(b)

Fig. 10. Evaluation of the offered SIMD ISA in the considered platforms (excluding technological differences) for sequence alignment applications. (a) Execution time (in clock cycles) and speedup variation for the alignment of different sized query sequences in the considered execution platforms. (b) Average number of clock cycles to perform one single cell update.

C. SIMD Instruction Set Evaluation

To assess the benefits of the SIMD ISA extension introduced in the proposed ASIP, and to validate whether it compensates for the additional resources and the smaller maximum operating frequency, the number of clock cycles required to execute a DNA sequence alignment procedure was accurately measured. Furthermore, the proposed architecture was also validated by comparing it against the three distinct superscalar GPPs: 1) the Intel Atom E665C processor, which is capable of dual instruction issue with in-order execution; 2) the ARM Cortex-A9 processor, which is capable of dual instruction issue with out-of-order and speculative execution; and 3) the Intel Core i7 3820 processor, capable of multiple instruction issue with out-of-order and speculative execution of up to 6 μops per clock cycle. 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 are the most favorable parameterizations for each case.

In Fig. 10, the speedup variation to execute the DNA sequence alignment on the proposed ASIP and on the three GPPs is presented. It is important to note that the measured values presented in Fig. 10(a) correspond to the alignment to a single reference size, since it was found that the reference sequence size does not affect the throughput of the implementations. By analyzing the speedup columns in Fig. 10(a), it can be observed that the proposed ISA allows the proposed ASIP to achieve an average speedup of 37.5×. In comparison, the ARM Cortex-A9 processor achieves an average value of 5.09×, the Intel Atom E665C an average of 18.47×, and the Intel Core i7 3820 achieves 16.67×.

The second measurement of the ISA efficiency was conducted by comparing the average number of clock cycles required to perform a single cell update of the sequence alignment matrix in the considered processors. These values were obtained by dividing the total number of clock cycles (c) by the product of the lengths of the reference and query sequences (m and n, respectively)—$c/(m \times n)$—and are presented in Fig. 10(b). For the particular case of the proposed ASIP, the SIMD word size was also varied. From the presented chart, it can be concluded that, even though the off-the-shelf Intel Atom and ARM Cortex-A9 processors can issue two instructions per clock cycle, the proposed ASIP offers a 2 and 2.5× speedup when compared with them, respectively. On the other hand, the Intel Core i7 uses a complex control structure capable of simultaneously executing multiple instructions out-of-order (issuing up to six micro-ops per clock cycle [23]) to achieve an average of two instructions per cycle for the considered datasets. This allows reducing the number clock cycles per cell update to 1.35 (on average), which compares with a value of 1.70 cycles per cell update for the proposed ASIP (using a 128-bit vector) issuing only one instruction per clock cycle. Naturally, while the Intel architecture achieves a higher instruction throughput, it does so at the expense of additional hardware resources used to control the processor. As a consequence, and as it will be shown later in this section, the Intel i7 processor requires more energy to perform the same operation.

From the previous ISA evaluation, it is possible to conclude that a simple RISC processor with the proposed set of SIMD instructions can be very effective to execute bioinformatics algorithms, allowing it to simultaneously fulfill both the low-power and performance requirements of the NGS mobile platforms [4], [10]. Furthermore, important speedups can also be achieved by parallelizing the sequence alignment in a multicore approach.

D. Scalability of the Multicore Processing Structure

To evaluate the performance gains of a multicore alignment structure, the coarse-grained parallel architecture described in Section V was implemented. All processing cores were based on the proposed ASIP, exploiting the ISA benefits that were presented in the previous subsection. The integrated scratchpad and shared memories were set to an address-width of 15 bits. To evaluate the shared bus contention, which constrains the multicore scalability, the reference sequence used by the alignment algorithm was stored in the shared memory, and each processing core requests access to the bus (once per iteration) to obtain the corresponding symbol. All other variables were stored in the scratchpad memory of each processing core.

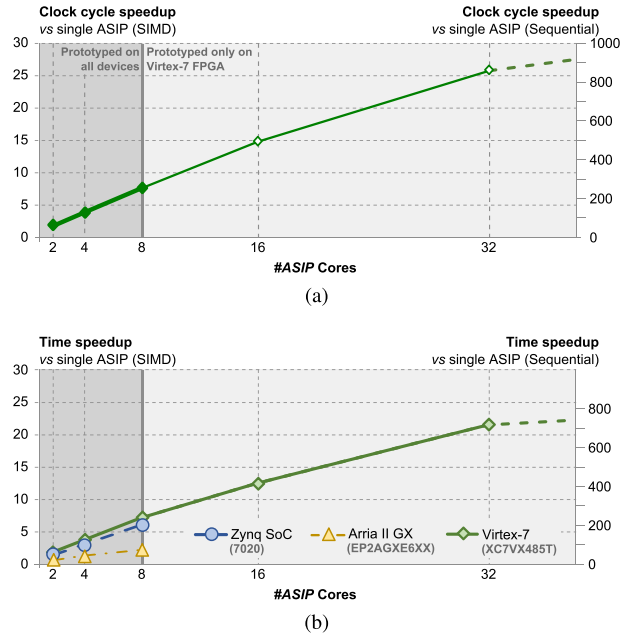


Fig. 11. Multicore processing structure performance regarding the original MB-Lite and ASIP cores; each line in (b) corresponds to the scaling of (a), considering the device maximum operating frequency of each individual core configuration. (a) Clock cycle speedup variation with the number of ASIP cores. (b) Time speedup variation with the number of ASIP cores.

Fig. 9 also presents the occupied hardware resources and the maximum operating frequency of the multicore processing structure for different aggregates of processing cores on different prototyping platforms. A maximum of eight processing cores can be prototyped in the Zynq SoC and in the Arria II GX FPGA due to the limitations imposed by the available Block-RAM and logic resources, respectively. Regarding the Virtex-7 FPGA, it has sufficient resources to implement the 32 cores of the multicore prototype system. In all cases, the operating frequency remains practically constant as the number of cores increases to eight. On the Virtex-7 FPGA, the frequency has a slight reduction with the number of cores. It can also be observed that the Zynq SoC achieves almost the same operating frequency as the larger Virtex-7 and an almost 3× higher frequency than the Aria II GX FPGA, which mainly results from the different technology of the latter FPGA.

Fig. 11 presents the clock cycle and the absolute time speedups for configurations with different numbers of cores, compared to a single 128-bit SIMD ASIP core, when processing the considered benchmark dataset. Fig. 11(a) depicts the architecture scalability in terms of the attained clock cycle speedup. This analysis allows to verify the multicore architecture scalability independently of the implementation platform. It is possible to observe that the speedup increases almost linearly for configurations of up to 16 cores. With additional cores, the contention in the shared bus becomes the limiting factor, thus reducing the effectiveness of the extra cores and resulting in a sub-linear speedup increase. Nonetheless, when considering the SIMD (or the sequential) implementation as reference, a maximum clock cycle speedup of 26× (or 830×) is achieved in a 32-core configuration.

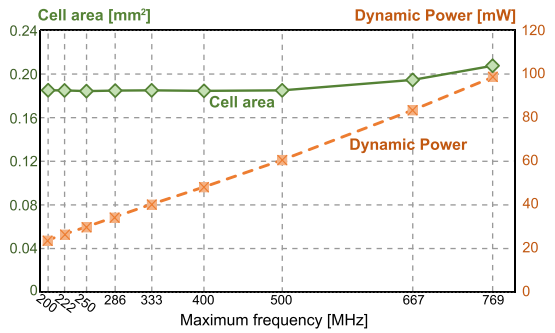


Fig. 12. Frequency, area and power scaling of the synthesized ASIC. The bold line represents the area, while the dashed line represents the dynamic power consumption, both in relation to the operating frequency.

Fig. 11(b) presents the achieved processing time speedup by taking into account the maximum operating frequencies obtained for the considered multicore configurations, implemented on different technologies. When considering the SIMD (or the sequential) implementation as the reference, the obtained results demonstrated that a 22× (or 720×) processing time speedup can be obtained with a 32-core implementation of the proposed ASIP in the Virtex-7 FPGA, with a maximum operating frequency of 157.3 MHz. Maximum processing time speedups of 6× (or 210×) and 2× (or 70×) were obtained for an 8-core configuration in the Zynq and Arria II GX FPGA devices, respectively, with maximum operating frequencies of 156.4 and 54.7 MHz. The lower processing time speedup obtained in the Arria II GX FPGA results from the significant decrease in the system's maximum operating frequency. Such decrease results from the inclusion of the shared bus, which in turn increases the amount of routing inside the FPGA.

E. ASIC Synthesis

The proposed ASIP was also synthesized using the Faraday Standard Cell Library targeting the UMC 90-nm CMOS process (library FSD0A_A), and an exhaustive performance and energy evaluation was performed. Fig. 12 presents the obtained results in terms of the total circuit area, operating frequency, and dynamic power consumption for different period constraints, from 5 ns (200 MHz) to the minimum achievable period constraint for the 90-nm technology.

As expected, the main drawback of operating the circuit with higher frequencies is the increase in dynamic power, which goes from 24.2 mW (200 MHz) to a maximum of 98 mW (769 MHz) for a single core system. Despite its increase, this power consumption response respects the low-power requirements of autonomous NGS biochip platforms. This is especially relevant considering that, when embedded in a diagnosis system, the proposed ASIP may only work during reduced amounts of time, after the sample biological sequence is acquired and while the sequence alignment is being performed. Therefore, the significant performance gains provided by the proposed ASIP structure are offered with a very small impact in terms of the power consumption of the whole biochip platform.

Finally, the complete multicore structure was also synthesized targeting the same 90-nm CMOS process technology. From the synthesis results, presented in Table III

(Section VI-F), it is possible to observe that a maximum operating frequency of 685 MHz is attained up to a 16-core configuration, while the 32-core configuration presents a decrease of 16%.

F. Performance and Energy Efficiency Evaluation

Besides the presented evaluation in terms of the resulting speedup values, several efficiency metrics are also used to study the computational and energy efficiency of the proposed multicore platform. In particular, three different metrics are used to characterize the multicore ASIP: 1) the attained raw throughput; 2) the energy efficiency; and 3) the performance-energy efficiency. These metrics were also determined for the three GPPs: 1) the Intel Atom E665C processor; 2) the ARM Cortex-A9 processor; and 3) the Intel Core i7 3820 processor. Among these, the Intel Atom and the ARM Cortex-A9 may potentially cope with the energy constraints of the target mobile platform, whereas the Intel Core i7 was only considered to compare the performance achieved by the proposed multicore platform with that of a GPP.

To compare the attained raw throughput, the CUPS metric was adopted, which is typically used in this application domain. This metric is based on the number of processed query sequences (q) in all cores, the length of the query and reference sequences (m and n, respectively), and the corresponding runtime, (t), in seconds. Hence, the CUPS metric is obtained as $(q \times m \times n)/t$ (considering that all queries have the same length). The obtained throughputs are shown in Fig. 13(b) and account for the maximum operating frequency of each implementation platform for the corresponding number of cores.

As it can be concluded from Fig. 13(b), the single-core ASIP implementations in the Xilinx FPGAs achieve throughputs very close to those of the ARM Cortex-A9 processor. On the other hand, the two considered Xilinx devices with an 8-core configuration and running at less than 200 MHz attain throughputs similar to those of the Atom processor, running at 1.3 GHz. Finally, a 32-core CMOS implementation, which is the most suited to the presented application, is able to achieve a performance similar to that of a quad-core Intel Core i7 running at an around 5× lower clock frequency and consuming around 230× less power [Fig. 13(a)].

An energy efficiency study was also performed using: 1) the power estimation tools of the Xilinx ISE and Quartus II software frameworks to obtain the values for the Virtex-7 FPGA, the Zynq SoC, and the Arria II GX FPGA; 2) the GPP thermal dissipation power (TDP) values depicted in the corresponding data-sheets (Table II); and 3) the Intel energy performance counters. Table III presents the obtained power consumption for the FPGAs and the ASIC, considering the worst-case power estimation for the used hardware resources at the maximum operating frequencies. For the Intel Atom and the ARM Cortex-A9 GPPs, the TDP was divided by the number of available cores in each processor, whereas for the Intel Core i7, the performance counters were used to accurately measure the power consumption. In Fig. 13(a), the power consumption of the considered platforms is presented, where the division between HPC and

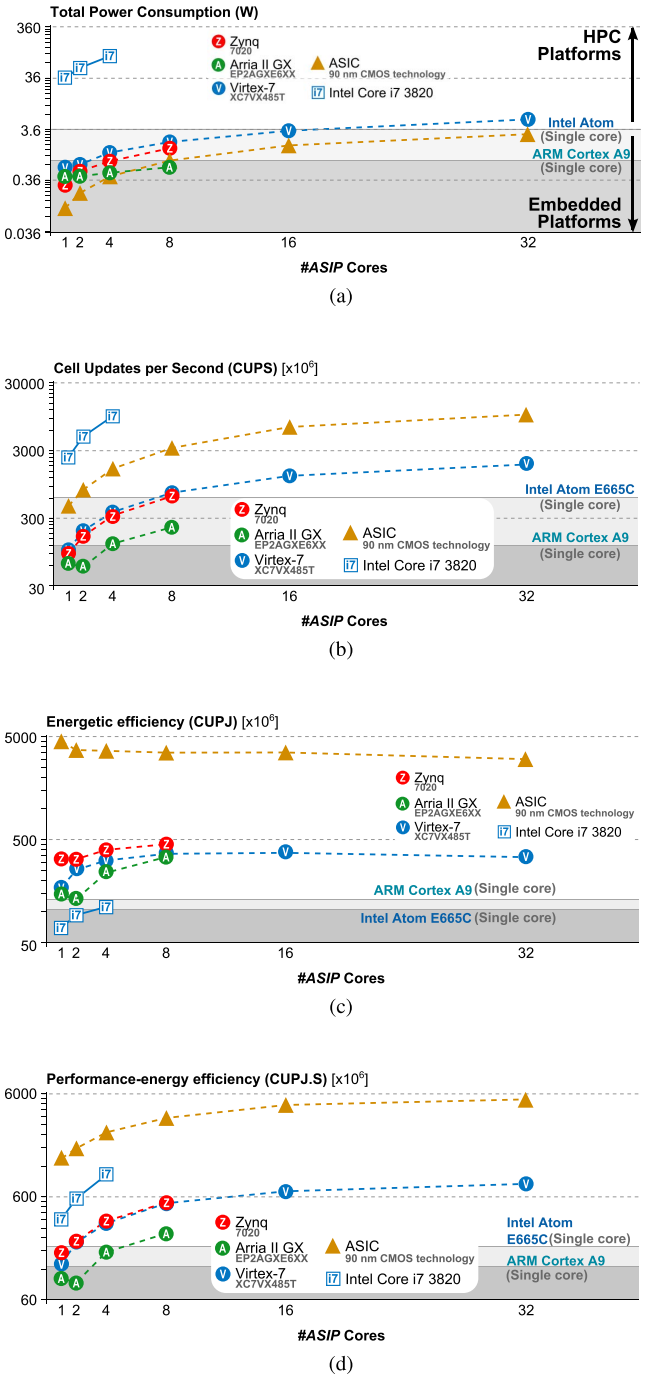


Fig. 13. Multicore system performance in comparison with the embedded ARM Cortex-A9 and Intel Atom E665C (depicted by the horizontal lines) and the high-end Intel Core i7 3820. (a) Total Power consumption for the embedded and HPC platforms (lower is better). (b) Raw throughput, given in cell updates per second (CUPS) (higher is better). (c) Energy efficiency, using the cell updates per Joule (CUPJ) metric (higher is better). (d) Performance-Energy efficiency, using an inverted energy-delay product (EDP) metric, the cell updates per Joule-second (CUPJS) (higher is better).

embedded platform was purposely set to include the Intel Atom processor.

After obtaining the energy consumption, given by the product of the execution time with the total supplied power, a performance efficiency metric based on the number of cell updates can be obtained to study the efficiency of the different configurations. The adopted CUPJ metric is given by the total

TABLE II

GPP OPERATING FREQUENCIES AND POWER ESTIMATION PARAMETERS. POWER ESTIMATE FOR THE GPPS IS THERMAL DISSIPATION POWER DIVIDED BY THE NUMBER OF CORES

	#Cores	Max. freq. [MHz]	TDP [W]	Single core power estimate [W]
Intel Core i7 3820	4	3600	130	$38^{(a)}$
Intel Atom E665C	1	1300	3.6	3.6
ARM Cortex-A9	2	533	1.9	$0.95^{(b)}$

$^{(a)}$ Actual value measured by using the Intel performance counters.
$^{(b)}$ Value estimated by dividing the TDP by the number of cores (2) and further validated using the Xilinx Power Estimation Tool for the Zynq SoC.

TABLE III

POWER CONSUMPTION MEASUREMENTS FOR EACH FPGA AND ASIC IMPLEMENTATION, WITH DIFFERENT CORE CONFIGURATIONS

	Multi-core power supply (mW)						
	# ASIPs	1	2	4	8	16	32
Zynq-7000 $^{(a)}$	Dynamic Power	209	435	753	1343	-	-
	Static Power	79.5	81.1	83.2	87.6	-	-
Arria II GX $^{(b)}$	Dynamic Power	47	107	196	350	-	-
	Static Power	322	322	325	330	-	-
Virtex-7 $^{(c)}$	Dynamic Power	445	516	1002	1772	3044	5554
	Static Power	210	211	216	224	240	275
90nm CMOS ASIC $^{(d)}$	Frequency [MHz]	769	685	685	685	685	588
	Dynamic Power	98	197	398	810	1580	2691
	Cell Leakage Power	0.36	3.06	5.67	10.99	21.42	41.28

(a) and (c): worst-case estimate using the Xilinx Power Estimation Tool; (b): worst-case estimate using the Altera Quartus II; (d): worst-case estimate using the Synopsys Design Compiler.

number of processed cells divided by the total consumed energy. Fig. 13(c) represents the average CUPJ evolution for different core configurations. From these results, it is possible to observe that the FPGA and the ASIC implementations of the proposed multicore ASIP clearly surpass the energy efficiency of all of the considered GPPs. It can also be ascertained that, with configurations of up to eight cores, the energy efficiency of the FPGA implementations increases up to a steady state value, being the implementations on the Zynq FPGA, the most efficient among the three. This can be explained by the lower dynamic power values required by these configurations, coupled with the exponential growth of the number of processed cells and with the reduced shared bus contention, as it was observed in the speedup analysis. For the Virtex-7 FPGA, it can be observed that the 16-core configuration presents the highest energy efficiency, hence corresponding to the best tradeoff between energy consumption, maximum operating frequency, amount of hardware resources, number of processed cells, and shared bus contention. As expected, the ASIC implementation outperforms all other implementations by a factor of over $4\times$.

The adopted performance-energy efficiency metric, given in cell updates per Joule-second (CUPJS), evaluates the number of cells each system can compute such that its corresponding EDP is not greater than 1 Js. It can be demonstrated that this metric is equivalent to the application of the geometric mean $\text{CUPJS} = \sqrt{\text{CUPJ} \times \text{CUPS}}$. Fig. 13(d) depicts the calculated performance-energy efficiency in CUPJS for the considered platforms. By comparing only the used FPGAs, it is possible to observe that the implementations on both Xilinx devices are almost $4\times$ more efficient than those on the Altera Arria II GX FPGA. On the other hand, when comparing the performance-energy efficiency of the low-power ARM Cortex-A9 GPPs, it is possible to observe that its efficiency is very close to the efficiency offered by a single-core ASIP implementation on the Virtex-7 FPGA, only attaining a higher efficiency than the single and 2-core ASIP implementations on the Arria II GX FPGA (the worst, among the used FPGAs platforms). When looking at the other low-power GPP (the Intel Atom processor), it is possible to observe that its performance-energy efficiency is higher than that presented by the ARM processor, by the ASIP implementations on the Arria II GX FPGA and by the single ASIP implementations on both Xilinx devices (Virtex-7 and Zynq). Nevertheless, the ASIP implementations for 4, 8, 16, and 32 cores on both Xilinx devices are capable of achieving a much higher performance-energy efficiency. In what concerns the Intel i7 processor, it is possible to observe that its performance-energy efficiency is higher than that of the ASIP implementations up to 8-cores in the FPGA devices. Only the 16 and 32-core implementations of the ASIP on the Virtex-7 FPGA outperform the single-core Intel i7. Finally, the ASIC implementation of the proposed multicore ASIP presents the highest performance-energy efficiency, for the corresponding number of cores, among all of the considered platforms (the single-core ASIC, providing 58×10^{12} CUPJS, is almost $3\times$ as efficient as the Intel Core i7 processor, with 21×10^{12} CUPJS).

In another perspective, it is also possible to compare the performance of the programmable solutions with that of dedicated hardware alignment solutions. The latter are capable of raw throughputs as high as 67×10^9 CUPS [30] (higher than all of the analyzed programmable solutions), but lack the necessary flexibility that is highly advantageous in the targeted embedded platforms, which address different types of analysis and biomarkers. Thus, considering the presented results, it is possible to conclude that the proposed multicore ASIP complies with all the requisites to be embedded on low-power autonomous platforms, while simultaneously providing HPC capabilities that are usually only offered by state-of-the-art GPPs.

VII. CONCLUSION

This paper presented a new high performance and low power ASIP architecture especially suited for DNA sequence alignment at the biochip level. To exploit an algorithm's fine-grained parallelism, the proposed ASIP features an extended SIMD ISA that, while general, is especially interesting to bioinformatics algorithms. In the particular case of the Farrar's SW implementation, the in-order single-instruction issue implementation of the proposed SIMD ISA was able to achieve speedups of about 2.5 and $2\times$, when compared with equivalent SIMD implementations on dual-instruction issue NEON and SSE implementations of the ARM Cortex A9 and of the Intel Atom E665C, respectively. In fact, the obtained experimental results show that the architecture of the proposed ASIP, based on a single-instruction issue and a five-stage pipeline implementation, can achieve a performance comparable to that of an out-of-order Intel Sandy Bridge microarchitecture.

To exploit the coarse-grained parallelism model, a multicore platform was developed and prototyped on different FPGA devices. It was demonstrated that a linear speedup 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 multicore architecture, the proposed system is capable of achieving speedup values as high as 800×, with 32 cores.

Finally, and targeting an integrated biochip platform, a 90-nm CMOS implementation of the proposed multicore processing structure was considered. Experimental results show that a performance level similar to that of an Intel Core i7 processor can be achieved using 25× less energy. This demonstrates the viability of the proposed system on NGS biochip platforms.

REFERENCES

[1] K. K. Jain, *The Handbook of Biomarkers*. New York, NY, USA: Humana Press, 2010.

[2] R. J. Leary *et al.*, "Development of personalized tumor biomarkers using massively parallel sequencing," *Sci. Transl. Med.*, vol. 2, no. 20, p. 20ra14, Feb. 2010.

[3] F. A. Cardoso *et al.*, "Integration of magnetoresistive biochips on a CMOS circuit," *IEEE Trans. Magn.*, vol. 48, no. 11, pp. 3784–3787, Nov. 2012.

[4] J. Germano *et al.*, "A portable and autonomous magnetic detection platform for biosensing," *Sensors*, vol. 9, no. 6, pp. 4119–4137, May 2009.

[5] F. Haque, J. Li, H.-C. Wu, X.-J. Liang, and P. Guo, "Solid-state and biological nanopore for real-time sensing of single chemical and sequencing of DNA," *Nano Today*, vol. 8, no. 1, pp. 56–74, Feb. 2013.

[6] N. Sebastião, N. Roma, and P. Flores, "Integrated hardware architecture for efficient computation of the *n*-best bio-sequence local alignments in embedded platforms," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 7, pp. 1262–1275, Jul. 2012.

[7] K. Benkrid, Y. Liu, and A. Benkrid, "Design and implementation of a highly parameterised FPGA-based skeleton for pairwise biological sequence alignment," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2007, pp. 275–278.

[8] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun, "A reconfigurable accelerator for Smith-Waterman algorithm," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 54, no. 12, pp. 1077–1081, Dec. 2007.

[9] Z. Nawaz, M. Nadeem, J. van Someren, and K. Bertels, "A parallel FPGA design of the Smith-Waterman traceback," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2010, pp. 454–459.

[10] R. Singh, B. Li, A. Elligton, and A. Hassibi, "A CMOS Σ-Δ photodetector array for bioluminescence-based DNA sequencing," in *Proc. Symp. Very Large Scale Integr. Circuits (VLSIC)*, 2011, pp. 96–97.

[11] N. Neves *et al.*, "BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment," in *Proc. Int. Conf. Appl.-Specific Syst., Archit. Process. (ASAP)*, Jun. 2013, pp. 241–244.

[12] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool (BLAST)," *J. Molecular Biol.*, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[13] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proc. Nat. Acad. Sci.*, vol. 85, no. 8, pp. 2444–2448, 1988.

[14] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Molecular Biol.*, vol. 48, no. 3, pp. 443–453, Mar. 1970.

[15] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Molecular Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 1981.

[16] O. Gotoh, "An improved algorithm for matching biological sequences," *J. Molecular Biol.*, vol. 162, no. 3, pp. 705–708, Dec. 1982.

[17] T. Rognes and E. Seeberg, "Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors," *Bioinformatics*, vol. 16, no. 8, pp. 699–706, Mar. 2000.

[18] T. Rognes, "Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation," *BMC Bioinform.*, vol. 12, no. 1, p. 221, 2011.

[19] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, 2007.

[20] W. R. Pearson, "Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms," *Genomics*, vol. 11, no. 3, pp. 635–650, Nov. 1991.

[21] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," *Comput. Appl. Biosci.*, vol. 13, no. 2, pp. 145–150, 1997.

[22] M. L. Metzker, "Sequencing technologies—The next generation," *Nature Rev. Genet.*, vol. 11, no. 1, pp. 31–46, Jan. 2010.

[23] *Intel 64 and IA-32 Architectures Software Developer's Manual*. Santa Clara, CA, USA: Intel, Feb. 2013.

[24] T. Kranenburg and R. van Leuken, "MB-LITE: A robust, light-weight soft-core implementation of the MicroBlaze architecture," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2010, pp. 997–1000.

[25] *MicroBlaze Processor Reference Guide*. San Jose, CA, USA: Xilinx, Jan. 2009.

[26] *GCC, the GNU Compiler Collection*, document GNU Project, Feb. 2013.

[27] F. S. Castaño, A. Ramirez, and M. Valero, "Quantitative analysis of sequence alignment applications on multiprocessor architectures," in *Proc. ACM Conf. Comput. Frontiers*, 2009, pp. 61–70.

[28] N. Roma and P. Magalhaes, "System-level prototyping framework for heterogeneous multi-core architecture applied to biological sequence analysis," in *Proc. IEEE Int. Symp. Rapid Syst. Prototyping (RSP)*, Oct. 2012, pp. 156–162.

[29] J. Leitao, J. Germano, N. Roma, R. Chaves, and P. Tomas, "Scalable and high throughput biosensing platform," in *Proc. 23rd Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2013, pp. 1–6.

[30] N. Sebastião, N. Roma, and P. Flores, "Configurable and scalable class of high performance hardware accelerators for simultaneous DNA sequence alignment," *Concurrency Comput., Pract. Exper.*, vol. 25, no. 10, pp. 1319–1339, Jul. 2013.

Nuno Neves received the M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 2013, where he is currently pursuing the Ph.D. degree.

He is a Junior Researcher with the Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His current research interests include computer architectures and parallel computing.

Nuno Sebastião received the M.Sc. degree in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 2007, where he is currently pursuing the Ph.D. degree.

He is a Junior Researcher with the Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His current research interests include architectures for biological sequences processing.

David Matos received the Ph.D. degree in information systems and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal, in 2005.

He is an Assistant Professor with the Department of Computer Science, IST, and a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His current research interests include intelligent systems, computational music processing, and high-performance computing.

Pedro Tomás received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal, in 2009.

He is an Assistant Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His current research interests include computer architectures and parallel computing.

Paulo Flores received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal, in 2001.

He is an Assistant Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His current research interests include computer architecture and algorithms, embedded systems, and EDA.

Nuno Roma received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico (IST), Technical University of Lisbon, Lisbon, Portugal, in 2008.

He is an Assistant Professor with the Department of Electrical and Computer Engineering, IST, and a Senior Researcher with the Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His current research interests include computer architectures and specialized structures for high performance computing.