Energy-Efficient Architecture for DP Local Sequence Alignment: Exploiting ILP and DLP*

Miguel Tairum Cruz, Pedro Tomás, and Nuno Roma

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{miguel.tairum,pedro.tomas,nuno.roma}@inesc-id.pt

Abstract. Typical approaches to solve Dynamic Programming algorithms explore data level parallelism by relying on specialized vector instructions. However, the fully-parallelizable scheme is often not compliant with the memory organization of general purpose processors, leading to a less optimal parallelism exploitation, with worse performance. The proposed processor architecture overcomes this issue by relying on a data stream loader and a set of especially designed instructions. Furthermore, to make it compliant with the strict power and energy limitations of embedded systems, it maximizes resource utilization by exploiting both data and instruction level parallelism, by statically scheduling a bundle of instructions to several vector execution units. To evaluate the proposed architecture, we compare it with two embedded processors, an ARM Cortex-A9 and an application-specific processor with SIMD extensions, using two benchmarks from the sequence alignment domain, namely Smith-Waterman and Viterbi. The obtained results show that the proposed architecture achieves up to 5.16x and 2.19x better performance-energy efficiency and up to 5.44x and 1.25x better energy efficiency than the ARM Cortex-A9 and the dedicated processor, respectively.

Keywords: Low-Power Architecture, DLP, ILP, VLIW, Dynamic Programming, Sequence Alignment.

1 Introduction

Sequence alignment applications frequently make use of Dynamic Programming (DP) algorithms to extract information from large databases [1]. As an example, the Smith-Waterman (SW) algorithm [2] is widely adopted for local sequence alignment, computing the alignment score by filling up a scoring matrix, where each cell presents a vertical, horizontal and diagonal dependency with its neighbors. Similarly, the Viterbi algorithm [3], used to find the most probable state sequence in a Hidden Markov Model (HMM), can also be represented in a matrix form, resulting in similar computations steps and dependencies as the SW algorithm. Accordingly, Data Level Parallelism (DLP) can be naturally exploited in these algorithms by using Single Instruction Multiple Data (SIMD) extensions (often present in most processors), as long as the computation is performed along the matrix anti-diagonal.

* This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT), under projects: Threads (PTDC/EEAELC/117329/2010) and project UID/CEC/50021/2013.

F. Ortuño and I. Rojas (Eds.): IWBBIO 2015, Part II, LNCS 9044, pp. 194–206, 2015.

Due to the gradual adoption of embedded systems in these type of applications (e.g. portable biochips [4]), efficient solutions are highly required, not only to guarantee the performance results required by these applications, but also to comply with strict power and energy consumption constraints. Typical processing solutions rely on General Purpose Processors (GPPs) [5,6] and dedicated hardware [7], that make use of SIMD extensions to exploit the DLP. On the other hand, the amount of memory that is required to accommodate the dependencies between states on DP algorithms is often very large, requiring the implementation of techniques to cache and reuse results from previous state computations, in order to quickly retrieve them without redundant computations. Furthermore, the previously referred anti-diagonal parallelism usually requires non-adjacent memory accesses, leading to a large performance impact in GPP implementations. To overcome this issue, GPP implementations often adopt a different processing pattern that better complies with a traditional GPP memory organization, such as vertical or horizontal simultaneous processing pattern, resulting in a better performance at the cost of introducing additional *lazy* loops to the algorithm computations. Dedicated implementations can approach this problem with special memory organization or special data management mechanisms, often resulting in better performance results [7]. However, these implementations only focus on a particular algorithm, disregarding any support for a broader class or different types of algorithms, thus limiting the range of application support.

In this paper, a new low-power programmable processor capable of supporting different DP algorithms is proposed, focusing on the performance extraction and optimization of sequence alignment algorithms, namely the SW and the Viterbi algorithms. The objective is not only to provide a high-performance processor for DP algorithms, but also to provide an energy-efficient solution suitable for low-power embedded environments. To attain the aimed performance levels with a low-power consumption, the architecture exploits: *i)* both DLP and Instruction Level Parallelism (ILP), supporting concurrent data computation in several independent and parallel execution units, with the minimal hardware requirements to accommodate it; and *ii)* a concurrent memory access mechanism, supported on a Data Stream Unit (DSU) that accesses the memory in parallel with the algorithm computations, thus maximizing the performance obtained with the most parallel processing scheme (e.g. the anti-diagonal parallelism).

This paper is organized as follows. After a brief overview of the studied DP algorithms and corresponding parallelism, in section 2, the proposed architecture is detailed in section 3. Section 4 briefly describes the prototyping and scalability evaluation of the proposed architecture, followed, in section 5, by a performance and energy efficiency evaluation of the architecture, against different state-of-the-art architectures. Finally, the conclusions are drawn in section 6.

2 Dynamic Programming Algorithms

DP algorithms are often represented in matrix form, where each cell corresponds to a sub-problem that depends on the adjacent cells (sub-problem dependencies).

This results in a final matrix where the value of the last cell can only be determined after all the previous cells have been computed (optimal substructure property). In a 2D matrix, each cell frequently presents horizontal, vertical and diagonal dependencies, allowing for DLP extraction along the anti-diagonal of the matrix. Furthermore, since all sub-problems are similar, they will require the exact same loop to be computed, allowing for an ILP exploitation by computing different steps of the loop at the same time. Popular sequence alignment algorithms such as the SW and the Viterbi algorithms, manifest the referred proprieties of DP algorithms and thus will be considered as two particular and independent case-studies.

2.1 Smith-Waterman

The SW algorithm computes the optimal local alignment between two sequences by considering a predefined substitution score matrix and a gap penalty function [2]. Gotoh [8] subsequently improved such algorithm definition by using an affine gap penalty model, allowing multiple sized gap penalties.

Given an arbitrary pair composed by a query and a reference sequences, together with a substitution score matrix and a gap initialization and extension penalties, the score matrix can be computed by solving three recursive relations, which correspond to a diagonal, horizontal and vertical dependencies, thus resulting in an anti-diagonal processing direction to extract the maximum parallelism. After the score matrix is filled, a traceback runs over such matrix, returning the local alignment.

2.2 HMM Viterbi

The Viterbi algorithm [3] is a DP algorithm that finds the most likely sequence path of hidden states in a HMM for a given sequence of observed outputs. A HMM consists in a stochastic model, where the future states of a process depend only on the present state and not on the complete sequence of states that preceded it. Furthemore, some (or all) states are hidden from the observer, with only the sequence of outputs generated by the model visible to the observer.

HMMs can be used to model alignment profiles, thus permitting the Viterbi algorithm to solve sequence alignment problems, similarly to the SW algorithm. These profiles model a family of sequences by highlighting the common features of them. They are usually generated by an initial multiple alignment, followed by a probabilistic breakdown of the elements present in each position.

A Profile HMM contains three different main interconnected states: Match States (M), that represent each column of the profile sequence; Insertion States (I), that represent the gaps in the alignment, and Deletion States (D), that represent the portions of the profile not matched by the sequence. Additional special states are also included, to support multiple local alignments. These states consist of flanking states, which delimit the sub-regions of the local alignment (i.e., they separate an aligned region from an unmatched region, in the sequence). When using Profile HMMs, the Viterbi algorithm becomes very similar to the

SW algorithm, with diagonal, horizontal and vertical recursions, resulting in the same anti-diagonal parallel processing pattern.

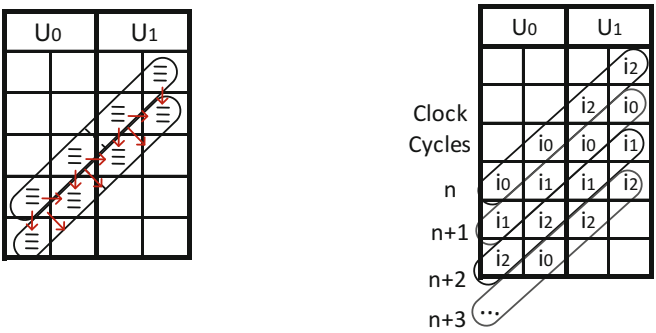
2.3 Parallelism Exploitation

The anti-diagonal processing pattern that is present in DP algorithms raises two problems to its exploitation: harder memory organization / access and a large amount of resource hardware requirements with a reduced utilization. While the former can be solved by implementing specialized memory access units to gather cell values in non-adjacent memory positions, the latter requires exploiting a different type of parallelism, namely ILP. Since the operations over a vector of elements along the anti-diagonal are independent, different parallel instructions can simultaneously compute different operations. This not only increases the potential for additional parallelism, but also reduces the hardware requirements, specially the number of FUs. The use of ILP is also supported by the common steps in a DP algorithm. Usually, these steps consist in dependency loads, followed by cell computations, and are finalized with the results storing. Hence, by assigning these different steps of the algorithms to the different elements in the vector, not only ensures ILP while maintaining data coherence (given the independence between the elements), but also improves the FUs utilization ratio.

In accordance, the proposed architecture reduces the hardware control by exploiting static ILP alongside DLP. This is achieved by issuing instructions in bundles, composed of several different parallel instructions, each operating over a vector of independent elements (DLP) (see Fig. 1(a)) in different and independent execution units (ILP). This way, instead of using a single large vector computing the same instruction as it is typical in vector architectures, the architecture has several smaller SIMD vectors, each computing their own instruction, in a similar fashion to a Very Long Instruction Word (VLIW) architecture. Parallelizing cells of a DP algorithm in different steps of the algorithm can, however, lead to data races between the cells if two conditions are not met: all cells currently being processed must be independent; and the cells that are being processed in advance must never have dependencies to the results of the other cells simultaneously being executed. The first condition is easily solved in the presented algorithms by following an anti-diagonal processing pattern. The impact caused by the irregular memory accesses referred by this parallelization scheme will be mitigated by an additional unit that performs the memory access operations concurrently to the main algorithm execution - a DSU. The second condition requires that the units operating over the down-left cells of the matrix are in advance regarding the units computing the cells at the top-right section (see Fig. 1(b)), due to the existing data dependencies.

3 Proposed Architecture

To exploit both DLP and ILP, as proposed in the previous section, the proposed architecture (see Fig. 2(a)) is composed of a bundle of vector execution units



(a) Anti-diagonal example with the respective dependencies

(b) Advancing units example. Unit 1 has a 1-instruction delay.

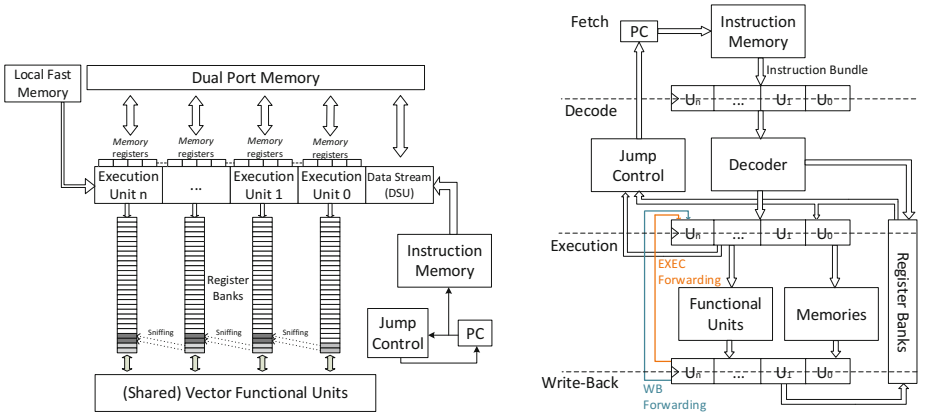
Fig. 1. Anti-diagonal DLP (left) and the adopted ILP based on an advancing units scheme (right). Both figures illustrate an example of two execution units with 2-cell vectors. 3 instructions (1 clock cycle each) are required to compute each cell: i_0, i_1 and i_2. The dependencies are represented by the arrows.

(each with an independent register bank) and a DSU (to allow autonomous data-transfers from the main shared memory). The architecture's instruction word thus consists in a bundle of several smaller SIMD instructions packed in a VLIW macro word: one for each execution units and one for the DSU. The architecture is also characterized by a pipeline structure (see Fig. 2(b)) with 4 pipeline stages, namely `FETCH`, `DECODE`, `EXECUTE` and `WRITE-BACK` stages, with data forwarding mechanisms to minimize the number of stalls due to data dependencies.

The architecture includes support for both *scalar* and *vector* functional units, a large main shared memory and a smaller local fast memory. All these blocks can be accessed by all the execution units, provided that no structural hazards occur. The DSU can only communicate with the main memory. If conflicts arise when accessing registers or FUs, a stall mechanism is implemented in a priority list manner, where the processor is stalled until all conflicts have been resolved (taking the instructions more clock cycles to compute).

The execution units, registers, memories and FUs share the same maximum vector width, with support for different word widths. This design paradigm allows for different accuracy compromises, improving algorithm performance if higher accuracy is not required (using more but smaller SIMD words), while still supporting problems that require a higher level of precision (using fewer but larger SIMD words).

The architecture can also be easily scaled in two distinct ways: by increasing the length of each execution unit and thus increasing the vector length (DLP); and by increasing the number of execution units, and thus increasing the number of parallel instructions (ILP).



(a) Proposed architecture (b) Pipeline Structure

Fig. 2. Proposed architecture scheme and respective pipeline structure

3.1 Architectural Units

As referred before, the architecture presents two distinct memories (in addition to the instructions memory): a main shared RAM memory and a local fast memory. The former can be accessed by the DSU and the execution units to store and read values, while the latter is a small memory that can only be read by the execution units, storing constant values required by DP algorithms. The existence of two memories minimizes the delay that is introduced by concurrent memory accesses and also promotes a better data organization, by separating the constant data values of the algorithms from the changing intermediary results. These memories present an access latency of 2 clock cycles: one cycle to index the correct address, and another cycle to load the value in the previous indexed address. In fact, there are two instructions to compute these two steps of a memory load, enabling a parallel usage of a memory load instruction between two units: one indexing an address, and the other effectively loading a data word. This allows achieving a throughput of 1 clock cycle with a latency of 2 clock cycles, when loading a value from memory.

The FUs in the architecture are shared by all the execution units. However, since the number of available FUs is limited, the execution units should not issue more operations of a given FU type than those available on the architecture, in order to maximize the processing performance. The FUs implemented in the architecture consist of SIMD Sum, Maximum, Shift, Logic (AND, OR, XOR) and Comparison units, with support for easily adding new FUs.

Each execution unit in the architecture has its own private register bank and a separate small set of 4 shared *memory* registers, allowing data sharing between units without memory accesses. These *memory* registers can eliminate the redundant memory loads introduced by some DP algorithms, whenever several cells (and therefore, execution units) share the same dependencies. Furthermore, these

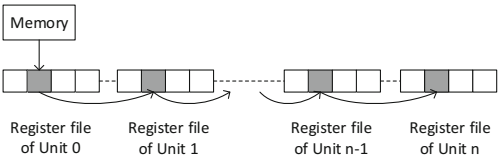


Fig. 3. Register Window mechanism: a value is loaded from memory to a register while the previous register value is shifted between adjacent units

registers will also be used by the DSU to communicate with the main memory (hence the *memory* name tag), storing and loading values from it in parallel to the algorithm computations, thus reducing the memory access latency impact. To further reduce the impact of memory accesses, the *memory* registers can also be used in a register window mechanism. This mechanism allows a memory load operation to fetch a data value to a *memory* register in one of the units on the periphery, while shifting the previous stored value on that register to the adjacent units (at the same time), as depicted in figure 3. This will update the selected *memory* registers in all units, enabling the pre-load of data values required by future iterations of the algorithms, such as sequence elements that are computed sequentially in all units.

Additionally, there is also a small subset of registers in the execution units' register banks, where a sniffing mechanism is implemented. This mechanism mirrors the sniffed registers to the register bank of the adjacent unit, effectively storing the register value in the adjacent unit. It is thus possible to share data values between adjacent units (which are very common in DP algorithms) without resorting to the main shared memory, at the same time that the *memory* registers are busy reading or storing values.

3.2 Instruction Set

While the DSU only performs memory access and shift operations over the *memory* registers (enabling the use of the referred register window mechanism), the execution units compute common arithmetic, logic, memory and control instructions. Additionally, optimized instructions for the targeted algorithms are also present, such as the `MAXMOV` instruction (useful for the SW algorithm), that not only does a maximum operation, but also adjusts some registers in the background to support the affine gap model without requiring additional clock cycles. The instruction set also presents modifiers to the implemented instructions. These modifiers can change the operand source of an instruction (e.g. use of immediate values) or add background operations to the instruction, enabling some algorithmic optimization. An example of the latter consists in a broadcast sum operation, where up to 3 parallel sums are performed in only 1 clock cycle, enabling the computation of multiple dependencies in DP algorithms.

3.3 Interface

In order to allow an efficient interconnection of the proposed architecture with different processing platforms, it was incorporated an interfacing structure aiming FPGA-based implementations. Such interfacing structure is based on the Advanced Microcontroller Bus Architecture (AMBA), structured according to its Advanced eXtensible Interface (AXI), allowing an easy interconnection between a GPP and the proposed architecture, which acts as an accelerator core.

The only communication requirement between the GPP element and the proposed architecture consists in a writing access directly to the three memories present in the proposed architecture, namely: *i)* the instructions memory; *ii)* the local fast memory; *iii)* and the RAM memory. Regarding the instructions memory, a control unit, outside the proposed architecture, controls the instructions flow from memory, with the instructions bundles being transferred in parallel to the architecture core. This control unit is further simplified when accounting that DP algorithms usually consist in one main loop that is repeated several times.

The two remaining memories require writing access from inside the proposed architecture, in addition to write access from the GPP. This way, there will be multiplexers at the entrance of the writing ports in these memories, where a control unit (external to the proposed architecture) decides which source will write to the memories. Therefore, the proposed architecture must stay idle when large sequences of data are being transferred to the memories (specially to the main memory). For the targeted sequence alignment algorithms, the idle time can be minimized by aligning a reference sequence to multiple query sequences, reducing the transfer times only to the query sequence transfers.

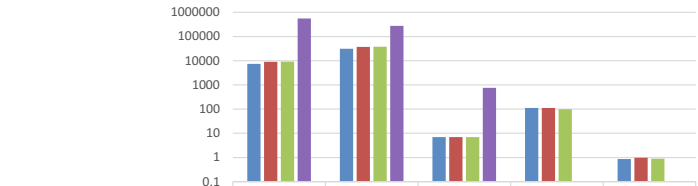
4 Prototyping

The proposed architecture was prototyped in a Zynq SoC 7100 FPGA [9]. The implemented configuration of the proposed architecture is composed of 1 DSU and 4 32-bits execution units, each using vectorial instructions to process multiple cells in parallel, resulting in a 128-bit wide VLIW. The word width was defined at 8 bits for the SW algorithm implementation and 16 bits for the Viterbi algorithm (due to higher precision requirements), resulting in 16 cells and 8 cells being computed in parallel, respectively. The register banks and memories share the same word widths.

The resources occupied in the FPGA (post place-&-route) can be seen in figure 4, accompanied by the operating frequency, power and scalability results.

4.1 Scalability

The impact on the hardware resources, frequency and power, introduced by both DLP and ILP scalability, is depicted in figure 4. The DLP scalability was evaluated by increasing the vector length from 32 bits to 40 bits, while the ILP



	Slice Registers	Slice LUTs	BRAMs	Frequency [MHz]	Power [W]
4x 32-bit units (baseline architecture)	7390	31011	7	111	0,86
4x 40-bit units	9033	37098	7	112	0,97
5x 32-bit units	9089	37635	7	98	0,89
FPGA Total	554800	277400	755		

Fig. 4. Hardware, frequency and power scalability for the proposed architecture by increasing the DLP (40-bit vectors) and by increasing the ILP (5 execution units)

scalability was achieved by increasing the number of execution units to 5. In both cases, the number of slice registers and LUTs increased approximately by 18% and 16%, respectively. The number of BRAMs remained the same, since it only changes with a greater variation in the vector width. Regarding the operating frequency and power consumption, the DLP scalability test resulted in approximately the same frequency as the baseline architecture, with an increase of 12% in the power consumption. The ILP scalability achieved a frequency decrease of 12%, with a power consumption increase of 4%.

5 Evaluation

To evaluate the proposed architecture, different metrics were used to measure both the performance and energy efficiency. The performance evaluation was measured in Cell Updates per Second (CUPS) and the energy efficiency was measured in Cell Updates per Joule (CUPJ). Additionally, the architecture was also evaluated regarding its performance-energy efficiency, measured in Cell Updates per Joule-Second (CUPJS).

5.1 Reference State-of-the-Art Architectures

To better assess the proposed architecture, it was compared with two state-of-the-art architectures representing distinct low-power domains: *i)* a mobile low-power GPP (ARM Cortex-A9) operating at 533MHz; and *ii)* a dedicated programmable ASIP (Bioblaze [7]) operating at 158MHz and implemented in the same Zynq FPGA (for a fair comparison). Additionally, the proposed architecture was also compared with an intel i7 3820 GPP, operating at 3.6GHz, for a high-performance reference evaluation (although this processor does not comply with the typical power constraints of embedded biochips). All these architectures make use of 128-bit SIMD extensions (ARM's NEON, Intel's SSE, and Bioblaze ISA), ensuring a fair comparison against the proposed architecture. The Bioblaze was used to evaluate only the SW algorithm, since it was

developed with the objective of accelerating such particular algorithm (although being programmable), while the remaining architectures were evaluated with both algorithms.

5.2 Algorithm Implementations

Both the SW and Viterbi algorithm implementations follow the anti-diagonal processing pattern, which is particularly efficiently exploited in the proposed architecture. The ILP is explored by having the left-most computing instructions in advance to the right-most cells. Hence, the SW main loop is composed of 5 instructions per execution unit (namely comparisons, sums and maximum operations), to process a complete vector of cells. The Viterbi algorithm requires a total of 23 instructions for an execution unit to compute the cells in its vector. These instructions mainly consist in simple sum and maximum operations, with additional loads and stores in the outer loop to account for the special states.

In the proposed architecture, the main memory is pre-loaded with the reference and query sequence, while both memories (main and local fast memories) are pre-loaded with all the necessary constants and cost/score values required by the evaluated algorithms. Therefore, only the algorithm steps are accounted for in the performed evaluations. Accurate clock cycle measurements of the required time to execute each biological sequence analysis in the proposed platform were obtained using Xilinx ISim.

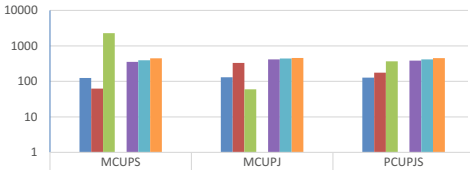
For the remaining evaluated architectures, the algorithms follow the state-of-art implementations of Farrar [5] and HMMER [6], for the SW and Viterbi algorithms, respectively. In these implementations, it is used a processing flow along the query sequence (vertical), leading to the existence of additional lazy loops in the computations.

In the Bioblaze, the clock cycle measurements were achieved by using Modelsim SE 10.0b [7]. In the ARM Cortex-A9 and Intel Core i7, 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 and the obtained values were subsequently divided by the number of repetitions and the processor clock frequency.

5.3 Smith-Waterman Evaluation

To benchmark the SW algorithm, a DNA dataset composed of several reference sequences (ranging from 128 to 16384 elements) and a set of query sequences with length ranging from 20 to 2276 elements was used. The reference sequences correspond to twenty 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`).

The evaluation results for the SW algorithm can be observed in Fig. 5, together with the scalability results introduced in section 4.1.



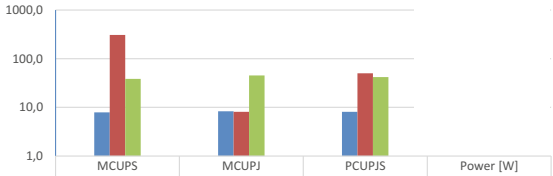
	MCUPS	MCUPJ	PCUPJS	Power
ARM Cortex-A9	124	131	128	0,98
BioBlaze	63	332	176	0,30
Intel Core i7 3820	2274	60	369	38
Baseline Proposed Architecture (4x 32-bit units)	356	416	385	0,58
Proposed Architecture (5x 32-bit units)	393	441	416	0,50
Proposed Architecture (4x 40-bit units)	448	460	454	0,56

Fig. 5. SW algorithm performance and energy-efficiency results for all evaluated architectures

When comparing with the low power architectures, namely the ARM Cortex-A9 and the Bioblaze, the proposed architecture achieves a speedup of 2.86x and 5.65x, respectively. Although the frequency of the ARM processor is 4.8x higher than the frequency of the proposed architecture, the fact that the algorithm implementation does not present lazy loops in the proposed architecture results in a better throughput and therefore in a better performance. Regarding the energy efficiency, the proposed architecture achieved a better efficiency than the ARM Cortex-A9 (3.18x) and the Bioblaze (1.25x), with the latter having a lower power consumption, thus demonstrating that the proposed architecture enables a more efficient implementation. Finally, the obtained performance-energy efficiency results show that the proposed architecture offers a significant gain over the ARM (3.02x) and the Bioblaze (2.19x).

As it was expected, in the high-performance domain the proposed architecture achieved a lower performance (0.16x) and a higher energy efficiency (6.95x) than the Intel i7, due the disparity in operating frequencies and power consumptions. Regarding the performance-energy efficiency, the proposed architecture managed to achieve better results, than the Intel i7, proving again that by efficiently exploiting the available parallelism it is possible to compensate for the difference in operating frequency.

Regarding the scalability results, the DLP scalability achieved superior performance and energy efficiency results when compared to the ILP scalability. In fact, the DLP scalability achieves a speedup of 1.26x and gains of 1.11x and 1.18x (for the performance, energy efficiency and performance-energy efficiency metrics, respectively) in comparison to the baseline architecture. As it was previously seen in section 4.1, on top of that, the DLP scalability also resulted in slightly less hardware resources and a higher operating frequency than the ILP scalability (only losing to the power consumption), proving to be the best scalability option given the amount of cell parallelism added.



	MCUPS	MCUPJ	PCUPJS	Power [W]
ARM Cortex-A9	7,9	8,3	8,1	0,95
Intel Core i7 3820	308,9	8,1	50,1	38
Proposed Architecture (4x 32-bit units)	38,6	45,2	41,8	0,584

Fig. 6. Viterbi algorithm performance and energy-efficiency results for all evaluated architectures

5.4 Viterbi Evaluation

To evaluate Viterbi's algorithm implementation, a sample of 28 HMMs from the Dfam database of *Homo Sapiens* DNA [10] was used. The adopted model lengths vary from 60 to 3000, increasing by a step of roughly 100 model states, and were created by the HMMER3.1b1 tool [6]. Query sequences (generated by the HMMER tool [6]) ranging from 20 to 10000 symbols were used to evaluate the alignment against all the above reference sequences.

The results for the performance and energy efficiency metrics for the Viterbi algorithm can be observed in Fig. 6. Similarly to the SW results, the proposed architecture achieved a better performance (4.89x), energy efficiency (5.44x) and performance-energy efficiency (5.16x) than the ARM Cortex-A9. When comparing to Intel i7, the results for the performance and energy-efficiency were also similar to those obtained for the SW algorithm, with the proposed architecture achieving a lower performance (0.13x) and a higher energy efficiency (5.56x). Regarding the performance-energy efficiency, and contrary to the SW evaluation, the proposed architecture achieved a worse result than the Intel i7 (0.83x). This result shows that the higher energy efficiency of the proposed architecture, together with the optimized algorithm implementation, are not enough to surpass the higher performance of the Intel i7, even with a less efficient algorithm implementation. However, due to its higher power consumption, the Intel i7 architecture cannot be seen as a viable option to the targeted low-power embedded systems domain.

6 Conclusion

The proposed architecture exploits DLP and ILP to provide a high performance platform to DP algorithms, offering low-power consumption and high energy efficiency, to comply to the strict requisites of embedded systems (e.g. biochips).

Furthermore, the proposed architecture is scalable in two distinct fronts: at a DLP level, by increasing the vector lengths, and at an ILP level, by increasing the number of execution units. In both cases, it was shown speedup and energy efficiency gains against the baseline architecture, which demonstrates its

potential scalability. The architecture also provides an extended algorithmic support when compared with traditional dedicated processors, by implementing a instruction set with optimized instructions and modifiers. Furthermore, the proposed architecture template permits the addition of new instructions and FUs, allowing further algorithmic optimization or support.

In face of the conducted evaluations, the proposed architecture can target low power systems without showing any significant loss against commonly used GPP alternatives and dedicated architectures. According to the obtained results, it presents better performance and energy efficiency than all the low-power architectures. In terms of performance-energy efficiency, the proposed architecture achieved gains of up to 5.16x against the ARM Cortex-A9 and 2.19x against the dedicated Bioblaze. For a high-performance reference, the proposed architecture also managed to obtain a better performance-energy efficiency than the Intel i7 3820 processor.

References

1. Benson, D.A., Clark, K., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Sayers, E.W.: Genbank. Nucleic Acids Research (2014)
2. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. Journal of Molecular Biology 147(1), 195–197 (1981)
3. Viterbi, A.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory 13(2), 260–269 (1967)
4. Cardoso, F., Costa, T., Germano, J., Cardoso, S., Borme, J., Gaspar, J., Fernandes, J., Piedade, M., Freitas, P.: Integration of Magnetoresistive Biochips on a CMOS Circuit. IEEE Transactions on Magnetics 48(11), 3784–3787 (2012)
5. Farrar, M.: Striped Smith–Waterman Speeds Database Searches Six Times Over Other SIMD Implementations. Bioinformatics 23(2), 156–161 (2007)
6. Eddy, S.R.: Profile Hidden Markov Models. Bioinformatics 14(9), 755–763 (1998)
7. Neves, N., Sebastião, N., Matos, D., Tomás, P., Flores, P., Roma, N.: Multicore SIMD ASIP for Next-Generation Sequencing and Alignment Biochip Platforms. In: IEEE Transactions on Very Large Scale Integration (VLSI) Systems (in press) (2015)
8. Gotoh, O.: An improved Algorithm for Matching Biological Sequences. Journal of Molecular Biology 162(3), 705–708 (1982)
9. Xilinx.: Xilinx DS190 Zynq-7000 All Programmable SoC Overview (2013), `http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf` (last accessed on December 10, 2014)
10. Finn, R.D., Bateman, A., Clements, J., Coggill, P., Eberhardt, R.Y., Eddy, S.R., Heger, A., Hetherington, K., Holm, L., Mistry, J., et al.: Pfam: The Protein Families Database. Nucleic Acids Research, gkt1223 (2014)