Protein Alignment Systolic Array Throughput Optimization

Protein Alignment Systolic Array Throughput Optimization

Giovanni Causapruno, Gianvito Urgese, Marco Vacca, Mariagrazia Graziano, *Member,* and Maurizio Zamboni

***Abstract*—Protein comparison is gaining importance year after year since it has been demonstrated that biologists can find correlation between different species, or genetic mutations that can lead to cancer and genetic diseases. Protein sequence alignment is the most computational intensive task when performing protein comparison. In order to speed-up alignment, dedicated processors that can perform different computations in parallel have been designed. Among them, the best performance have been achieved using Systolic Arrays. However, when the Processing Elements of the Systolic Array have an internal loop, performance could be highly reduced.**

In this work we present an architectural strategy to address this problem applying *pipeline interleaving*; this strategy is applied to a Systolic Array for Smith Waterman algorithm that we designed. Results encourage the adoption of pipeline interleaving for parallel circuits with loop based Processing Elements. We demonstrate that important benefits in terms of higher operating frequency can be derived without so relevant costs as increased complexity, area and power required.

***Index Terms*—Systolic arrays, Protein Alignment, Smith Waterman, Interleaving, CMOS.**

I. Introduction

APPROACHING the boundary limit for CMOS technological scaling [1] will not permit designers to rely any more on this scaling to increase the performance of computing systems [2]. For this reason, parallel architectures are gaining importance in these years as a solution to increase the throughput without the need to rely on technological improvements. Among them, Sysotlic Arrays (SAs) are a particular kind of parallel processing architectures that can guarantee the best performance for applications called "embarrassingly parallel" [3]. There are several applications that belong to this class and biosequence alignment is among them: for example SAs for protein alignment have been designed to achieve better performance with respect to CPUs and GPUs [4]. However, when the derived SA requires Processing Elements (PEs) with internal loops, throughput could be highly affected: in fact in a loop-based sequential circuit the result of a logic operation depends on the previous operations. As a consequence in sequential circuits it is not possible to execute a logic operation at each clock cycle, because inputs must wait many cycles to synchronize with incoming feedback signals. Throughput is therefore reduced of N times, where N is the length in clock cycles of the longest loop in the circuit [5].

In this paper we show how it is possible to apply a technique called *pipeline interleaving* to increase the throughput of loop-based SAs. Pipeline interleaving has been explored especially in the case of digital filters; here we introduce this method for Systolic Arrays, which has never been proposed before. This requires an analysis and re-design procedure of each PE, but also a careful retiming of inputs to guarantee synchronization between neighboring PEs. The SA that implements a protein alignment algorithm, presented in [6], is used as case study to show benefits in terms of increased frequency and dynamic power saved.

We introduce a systematic 3-step mechanism to adapt a SA to support pipeline interleaving. By inserting additional registers and interleaving input data it is possible to reduce the critical path and in this way increase the operating frequency of the circuit. Results achieved for the biosequence alignment SA strongly encourage the adoption of pipeline interleaving, because important benefits can be derived without relevant costs in terms of complexity and increased area. We introduce also some parameters that can be evaluated to understand what is the best area-frequency and power-frequency tradeoff and in this way choose the best level of interleaving.

The paper is organized in this way: section II gives an introduction to Systolic Arrays; section III describes briefly the problem of protein sequence alignment and the Smith Waterman algorithm; the concept of interleaving is explained in section IV. Section V describes the architecture of the Smith Waterman SA, while its optimization adopting pipeline interleaving is described in section VI. Results, obtained in ASIC with a $45\,nm$ low-power library and in FPGA, are compared and summarized in section VII.

II. Systolic Arrays

To increase the performance of a computing system, a common solution is to employ parallelism, using a large array of small processors. Systolic Arrays (SAs) were first introduced by Kung and Leiserson in 1978, who stated: "a systolic system is a network of processors which rhythmically compute and pass data through the system" [3]. Systolic Arrays are composed of Processing Elements (PEs), also called "cells", locally interconnected. Each PE receives data from neighbor cells or from outside and outputs result to the outside or to near PEs. Two are the main concepts that characterize SAs: local transmission of data (i.e. there are not global signals except from the clock) and parallel computation (i.e. all PEs work in the same way on different data).

SAs can have different shapes: for example PEs can be arranged in a bi-dimensional matrix-like shape, as is shown

Authors are with the Department of Electronics and Telecommunications, Politecnico di Torino, TO, I10129 Italy e-mail: mariagrazia.graziano@polito.it.

in Fig. 1(a), they can be linear, as in Fig. 1(b), or even have strange shapes as hexagonal or as in Fig. 1(c), with signals flowing in three different directions.

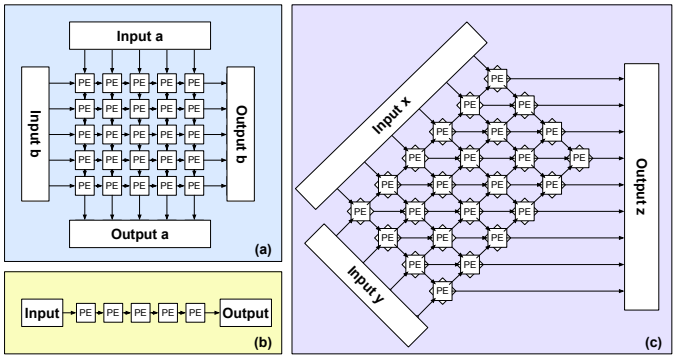


Fig. 1. (a) Matrix Systolic Array; (b) Linear Systolic Array; (c) Special Systolic Array.

In general an array processor is designed, as explained in [7], starting from the algorithm description of the problem and executing three steps: 1) Dependence Graph (DG) design, which consists in obtaining a graphical representation of an algorithm, where nodes represent computations and edges represent data dependencies. 2) Signal Flow Graph (SFG) derivation, which consists in shrinking the DG along one axis. Subsequently an SFG has a dimension less than the correspondent DG; this is due to the fact that in a DG each node represents one simple computation, while in the SFG a node is a processing unit, that should be reused in successive time steps, and for this reason nodes of a line in a DG can be mapped to a single node in the SFG. 3) Array Processor Design, that consists in designing the internal structure of each PE.

SAs have been exploited for image processing [8] [9] [10], signal processing [11] [12] [13] and video algorithms (such as those for MPEG compression). For example, in [14] and [15] a SA for logarithmic search motion estimation is presented: by exploiting a bi-dimensional systolic architecture the algorithm can be run 256 times faster than with a conventional linear array. Recently, automatic tools concerned to translate algorithms to SAs for FPGAs have been explored [16]. Also, reconfigurable arrays, that are not application-specific, have been introduced in [17].

As said before, SAs can be used also for biological sequence comparison [18] [19]: in [20] an overview of the different hardware solutions for biosequence analysis is carried out, and a NanoASIC implementation is presented to show the benefits that can be achieved in SAs adopting nanotechnologies.

III. Protein Sequence Alignment

Protein comparison is a fundamental operation that allows biologists to find phylogenetic or functional correlations between different species, or genetic mutations that can lead to cancers and genetic diseases. Protein sequence alignment is the most computational intensive task when performing sequence comparison; it consists in evaluating a number, called *alignment score*, that represents the grade of similarity between the studied protein called Query (*Qry*) and the protein coming from the Database called Subject (*Sbj*). The alignment procedure is performed taking into account that the proteins are represented by long sequences of Amino Acids (AAs) identified with alphabet letters. For each couple of AAs four biological events can occur, shown in Fig. 2: a *match*, when the two AAs correspond, a *substitution*, when the AA of the *Qry* is mutated in the *Sbj*, a *deletion* or *insertion* if the *Sbj* lacks or has an additional AA in a given position, respectively. If there is a match or a substitution the value of the scoring must be updated using the value coming from a substitutional matrix: given the two AAs $Qry(i)$ and $Sbj(j)$, this matrix returns a value $s\left(Qry(i), Sbj(j)\right)$ that represents the probability that an AA is substituted with another during evolution [21].

(Qry)	ACDEFG	AC**D**EFG	AC**D**EFG	AC--EFG
(Sbj)	ACDEFG	AC**L**EFG	AC--EFG	AC**D**EFG
	match	substitution	deletion	insertion

Fig. 2. Alignment with match, substitution, insertion and deletion.

When a deletion or an insertion occurs, the alignment score must be updated using instead a gap penalty. The most used penalty is the "affine" one (γ_{aff}) in which two different values, called gap_open d and gap_extension e, are used in order to encourage one large gap rather than many small ones, since the former condition is more likely. In equation (1), g represents the length of the gap [22].

$$\gamma_{aff} = -d - (g-1) \cdot e \tag{1}$$

The Smith-Waterman algorithm (S-W) is the most famous and widely used alignment algorithm to locally align two proteins [23]. S-W is a dynamic programming algorithm that works with a score matrix $F(i,j)$, that stores values of correlation in any point, shown in the bottom part of Fig. 3. A detailed description of the S-W algorithm is out of the scope of this article. Here we describe briefly its architecture and in order to do so we introduce its working mechanism at a glance. It is important to highlight that adopting the gap affine model, each cell is required to evaluate three different values [23]. Recently this algorithm has been optimized for the Systolic Array implementation in [6], allowing faster and lighter computations. The main idea of this optimization is to use a *sel* signal that can be either 0 or 1 to choose on-the-fly between gap_open and gap_extension and for this reason it is called Dynamic Gap Selector (DGS): this substitutes the usage of two gap matrices that are provided by the original algorithm, that required a higher computation effort and an increased complexity. S-W algorithm is divided into three steps:

1) Initialization: first row and first column of the matrix, respectively $F(i,0)$ and $F(0,j)$, are initialized to 0.
2) Score Matrix filling: each cell is filled with a value of $F(i,j)$; in the case of DGS_S-W, this value is evaluated according to equations (2 - 3):

$$F(i,j) = \max \begin{cases} 0 \Rightarrow sel(i,j) = 0 \\ F(i-1,j-1) + s\left(Qry(i), Sbj(j)\right) \\ \qquad\qquad \Rightarrow sel(i,j) = 0 \\ F(i-1,j) - g \Rightarrow sel(i,j) = 1 \\ F(i,j-1) - g \Rightarrow sel(i,j) = 1 \end{cases} \tag{2}$$

$$g = \begin{cases} d & \text{if } (sel(i-1,j) \text{ or } sel(i,j-1)) = 0 \\ e & \text{if } (sel(i-1,j) \text{ or } sel(i,j-1)) = 1 \end{cases} \tag{3}$$

where d stands for gap_open and e stands for gap_extension.

3) Traceback: the maximum score represents the starting point for the best local alignment, that is found tracing back till the first 0 is found.

Fig. 3 shows the derived array for the problem. Each cell of the matrix is represented as a cube with two faces, to indicate that the value of the matrix F and the value of signal *sel* must be evaluated. The number inside these cells represents the step in which that cell is evaluated. Moreover it is shown the derivation of the linear SA, where each *Qry* AA is associated to a PE, while *Sbj* AAs are provided sequentially to the SA.

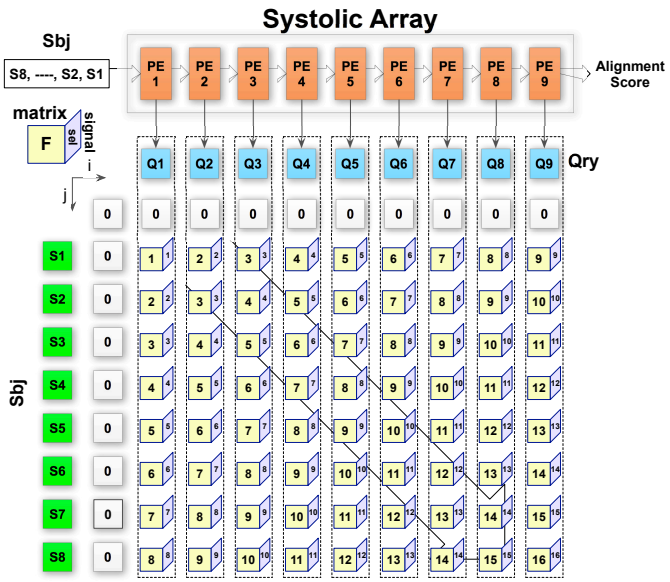


Fig. 3. Correspondence between the S-W systolic array and the S-W matrices for the DGS_S-W algorithm. The two faces of the cube represent the cells of the score matrix and the gap selection signal needed for the computation. The groups of cells are associated with the PE that executes their calculation. Each *Qry* AA is associated to a PE. The *Sbj* AAs are provided in sequential way to the Systolic array input. The numbers in the cells identify the step in which that cell is computed.

The DGS_S-W Systolic Array works in this way:

- in a first phase the matrix is initialized: each cell is associated to an AA of the *Qry*. This means that one column of the Substitutional Matrix is uploaded inside the cell and will be used as Look Up Table (LUT) during computation phase; moreover, values of gap_extension and gap_open are uploaded in dedicated registers;

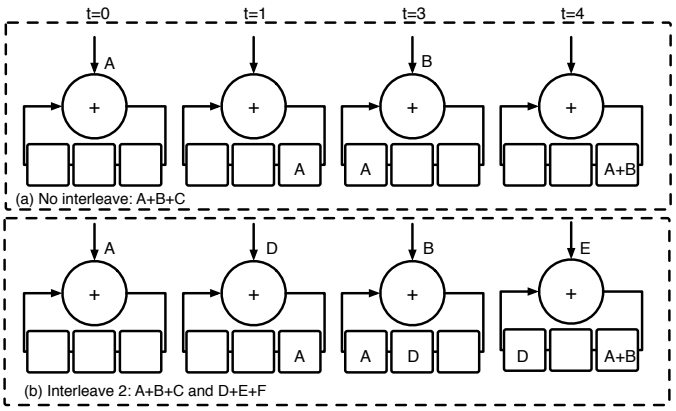


Fig. 4. Interleave example: an accumulator that contains in its internal feedback three registers. (a) the circuit without exploiting interleave requires to provide one input every 3 cycles, thus having an addition every 3 cycles. (b) the circuit with interleave 2: input values are interleaved and in this way the throughput can be simply doubled because it is possible to execute second addition $D + E$ immediately after $A + B$.

- during computation, the AAs of the *Sbj* are passed to the SA from the boundary leftmost cell. Computation is executed inside the cell and in the meanwhile the AA is passed to the neighboring PE. As said before, the AA of the *Sbj* that arrives inside a cell is used to address one element of the LUT and extract the correspondent value of the Substitutional Matrix. The maximum alignment score flows in the same direction (from left to right) and is outputted from the rightmost cell. In every PE the maximum is updated according to formula (2);
- the trace-back procedure is not implemented in this architecture because the focus here is on the most computational intensive task that is the evaluation of the maximum alignment score. What results from the usage of the SA is a list of couples (*Sbj*, result); each couple associates a maximum alignment score to each *Sbj* to be aligned.

IV. Interleaving

Aim of this paper is to show the benefits of data interleaving in Systolic Arrays through the case study of the DGS_S-W. This can provide a faster circuit, increasing the throughput. Interleaving is in general a way to arrange data in a non-contiguous way. Consider the simple circuit shown in Fig. 4: this is an accumulator based on an adder and a feedback loop with three registers. Fig. 4(a) shows the execution of $A+B+C$; there is a delay of three clock cycles between one input and the successive one, which means that one addition can be executed every three clock cycles. This is due to the fact that there is a data dependency between each addition and the successive one, which means that each addition can be executed only when the previous one has been completed and the result is ready at the input of the adder. Fig. 4(b) shows the same circuit applying interleaving: in this case it is required to have at least another operation to execute, for example $D + E + F$; *interleaving* means providing inputs in a non-continuous manner, i.e. A, D, B, E, C, F in the example, and exploiting registers to store results belonging

to different operations. This can be applied in general to any loop-based circuit. Data interleaving can be applied when there is no data-dependency between successive steps; D+E for example can be executed immediately after A+B. It could be possible to interleave also another operation to maximize the throughput, since the pipe would be in this case always full. In the example, there is no architectural reason to design an adder with 3 registers in the loop; however, it is possible to think that through a retiming procedure, these registers are inserted in the middle of the adder to reduce its critical path and increase the operating frequency. This would still mean that three clock cycles are required to execute addition and feedback of the signal and for this reason it would be possible to apply pipeline interleaving as explained.
The benefits of interleaving (often referred as *pipeline interleaving*) have been analyzed in literature, especially in the case of digital filters [24] [25]: internal feedbacks in digital filters negate the most obvious ways of improving performance, that is pipelining. In fact, recursive systems cannot be pipelined at an arbitrary level by simply inserting latches. The problem is solved by changing the internal structure of the algorithm to create additional logic delay operators inside the recursive loop, which can then be used for pipelining.

Interleaving in Systolic Arrays

In our study we want to apply the concept of interleaving to SAs, that has been never proposed. To do so, we first introduce a taxonomy of SAs and we explain how interleaving is provided at PE-level and how the concept can be extended to the whole SA.
Systolic Arrays can be divided into two main classes: those With cells that have an Internal Loop (WIL), and those WithOut Internal Loop (WOIL); the former can be further split in systolic arrays that Store results in the cells (WIL-S) and systolic arrays where the partial result is Passed Through the cells to obtain the final value (WIL-PT). The DGS S-W Systolic Array belongs to this last class.

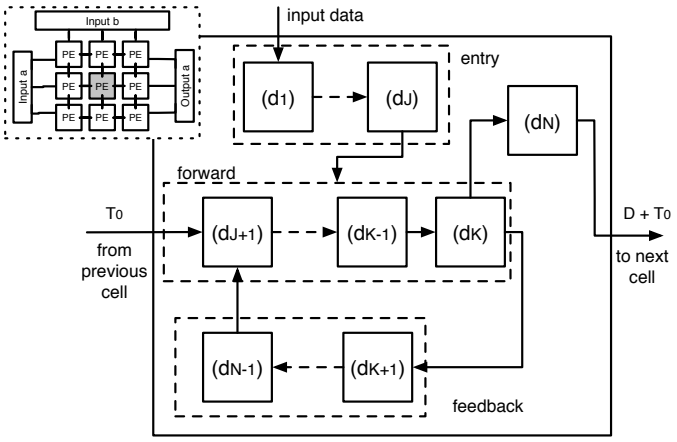


Fig. 5. WIL cell: this PE is part of an array as shown in the top-left corner. The PE is made of 4 parts: an entry section, the forward and feedback parts of the loop, and the output section. Data coming from previous PE can enter at any stage of the cell.

A cell with internal loop is shown in Fig. 5. It is made of 4 parts: an entry section, made of blocks numbered from 1 to J; the forward part of the loop, made of blocks from $J+1$ to K, the feedback part of the loop, made of blocks from $K+1$ to $N-1$; the output block, called N. Each of these blocks is associated to a delay d_i, $i = 1, 2, \ldots, N$, and cannot be pipelined internally. Let us call T_e the total delay of the entry block, defined in equation (4), T_{fo} the delay of the forward side of the loop, eq. (5), T_{fb} the delay of the feedback in the loop, eq. (6) and T_o the output delay, eq. (7):

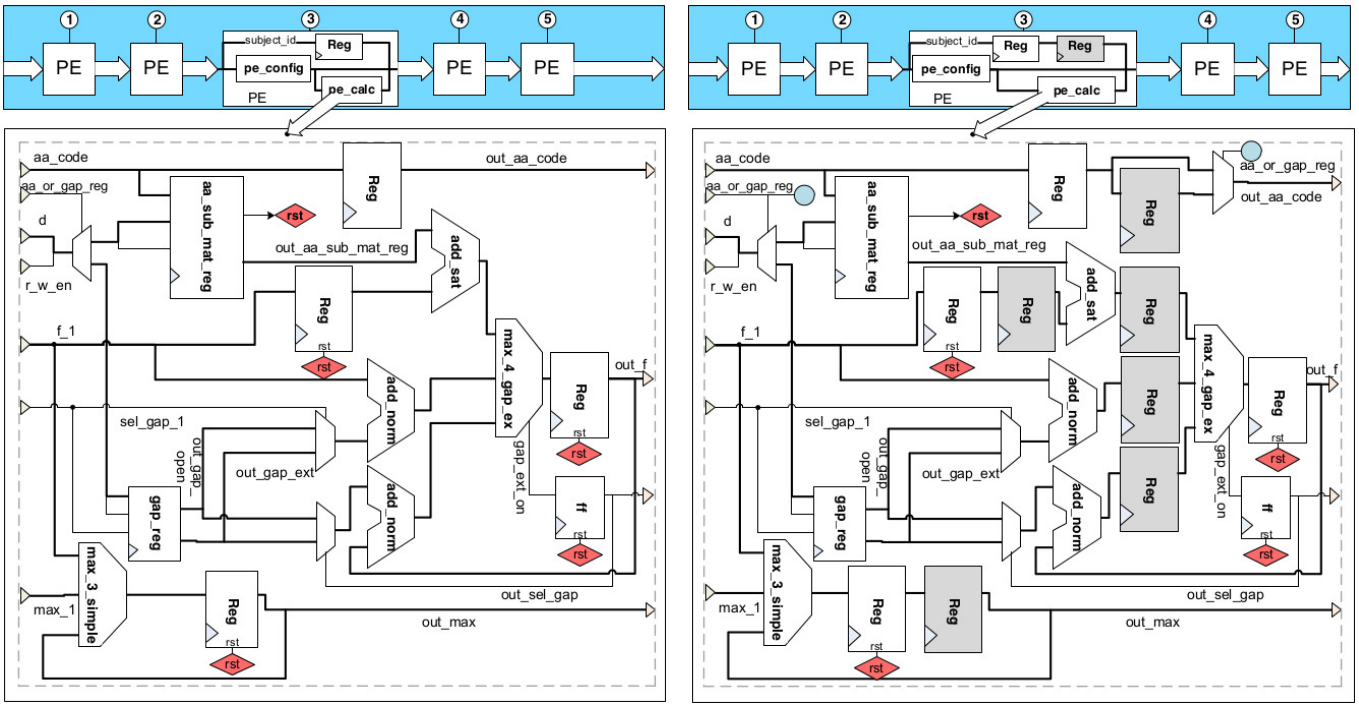
$$T_e = \sum_{i=1}^{J} d_i \quad (4) \qquad T_{fo} = \sum_{i=J+1}^{K} d_i \quad (5)$$

$$T_{fb} = \sum_{i=K+1}^{N-1} d_i \quad (6) \qquad T_o = d_N \quad (7)$$

Input data coming from outside enter in the first block, while data coming from the neighbor processing element enter in the first block of the loop $J+1$.
In order to match timing of inputs with delay of the feedback, it can be demonstrated that inputs must be given every $T_{loop} = T_{fo} + T_{fb}$ cycles, that is the total time of the feedback loop. However, given the intrinsic pipelined nature of the structure, we can improve performance and usage of the PE providing inputs every $K = max\{d_i\}$. A new operation can be started after K cycles, and in this way N different operations can be interleaved, being $N = \lfloor T_{loop}/K \rfloor$ (integer division). When T_{loop} is not a perfect multiple of K, the remainder of the division, called R, must be taken into account: after N operations have been started, the following one must start with a delay of $K + R$ with respect to the previous, so to have synchronization with the result coming from the loop. R represents a number of "stalls" that must be inserted between the one set of N inputs and the following set. Consider the following example: $T_e = 3$, $T_{fo} = 3$, $T_{fb} = 10$; it is possible to interleave $T_{loop}/K = 13/3 = 4$ operations, inserting a stall ($R = 1$) after each set of 4 inputs.

V. DGS_S-W Architecture

The structure of each PE in the SA is shown in Fig. 6(a). Each PE is made of two sub-blocks, PE_CONFIG and PE_CALC: the first is used only during configuration phase, that consists in uploading in each PE the values of the column of the Substitutional matrix of the correspondent *Qry* AA inside the LUT and values of gap_open and gap_extension in registers, while PE_CALC is used during both phases. One register is also present to store the identification number of the incoming *Sbj* that must be aligned. Each AA is represented with a number from 1 to 23; this number is identified as **aa_code** (5-bit bus).
Two main blocks are present inside PE_CALC: one is the LUT that stores the column of the Substitutional Matrix corresponding to the AA of the *Qry* assigned to each PE, that is called AA_SUB_MAT_REG. The other block is called GAP_REG and contains two registers, storing the values of gap_open and gap_extension. The choice between these two values is



(a) No interleave (b) Interleave 2

Fig. 6. **(a)** The structure of the Systolic Array: each PE is made of a configuration part, PE_CONFIG, and a calculation one PE_CALC; the latter includes the LUT to store columns of the Substitutional matrix and the maximum evaluation blocks to find the best alignment score. **(b)** The structure of the Systolic Array supporting *pipeline interleaving* with level 2 of data interleave: notice the additional registers that split the critical path and can store two different values inside the loops. The additional registers must not be reset during computation phase, so they are represented without a **rst** control.

then executed through the multiplexers that are allocated after the GAP_REG block. Notice that while one multiplexer is controlled by the signal coming from outside **sel_gap_1**, the other is locally controlled through the signal **out_sel_gap** that is in turn transmitted to next PE.

This structure presents also three adders: two of them are named ADD_NORM while another is called ADD_SAT; the first two must execute $F(i, j-1) + (-g)$ and $F(i-1, j) + (-g)$; since the gap values to add are negative, no overflow can occur, so these are normal Ripple Carry Adders (RCA). ADD_SAT must instead execute $F(i-1, j-1) + s(Qry(i),\ Sbj(j))$; this addition could cause overflow and for this reason it is used an adder that in case of overflow saturates, i.e., every time MSBs cause a carry, the result is the maximum that can be represented with the given number of bits.

The maximum evaluation is computed in two different blocks: the first is the MAX_3_SIMPLE, that computes the maximum between the stored maximum, the maximum coming from previous PE and the $F(i, j)$ evaluated in previous cell. This is the maximum alignment score that can be obtained with a given *Sbj*; since $F(i, j)$ of cell N is taken into account in cell $N + 1$ by this architecture, there is the need for a further maximum checking after the last PE that is performed in another block allocated at the output, called PE_TERMINATOR. This block will consist in the bottom part of PE_CALC only, i.e. MAX_3_SIMPLE with the associated register.

The second maximum evaluation is the local result $F(i, j)$, according to equation (2). This is done using the MAX_4_GAP_EX block: this block evaluates the maximum between the three inputs, or returns 0 in case the maximum is negative; moreover, it dynamically selects between the gap_open and gap_extension.

PE_CALC comprises also the registers that store the value of $F(i, j)$, and the value of the gap that must be used in following cycle, and a register that stores the $F(i, j-1)$ coming from previous cell in order to obtain $F(i-1, j-1)$. Finally, one register is used to store the AA code (**aa_code**) that in following clock cycle is transmitted to neighbor PE.

VI. Systolic Arrays Interleaving Optimization

In this section the optimization procedure that should be followed in order to enhance SAs with interleaving is presented. We first describe the steps of the procedure that can be in general applied to any SA and then exploit this mechanism in order to improve the DGS_S-W performance.

The following three steps must be performed to introduce interleaving to a general WIL SA:

1) *Insert registers in the loops*: this step consists in identifying all the loops present inside each PE, finding the Critical Path (CP) inside each loop and inserting registers to split the CP. It is important to split all the loops with the same number of registers. This number will identify the level of interleave that should be applied to the circuit to guarantee maximum performance; in the following it will be referred as N.
2) *Synchronization of other signals*: it is common to have other registers that store partial results outside loop structures. These registers must be replicated in order

to guarantee synchronization of all signals. N registers must be inserted in place of each original one outside any loop structure. It is useful to check correctness of the derived circuit through simulation, by providing the same inputs of the original no-interleave structure, but with a higher delay between successive inputs: in particular, if input $i+1$ was provided k cycles after input i, to check this circuit it must be provided after $k \cdot N$ cycles (a procedure referred as *skewing* in normal pipelining procedures).

3) *Data Interleaving*: this last step allows to increase the throughput. It consists in providing inputs in an interleaved manner, considering N different operations; the new order of inputs will be: input 1 of op.1, input 1 of op.2, ...input 1 of op.N, input 2 of op.1 and so on. This guarantees the maximum throughput for the given architecture. The rules to evaluate the delays of successive inputs were described in section IV.

In order to follow this description, one should a priori decide the level of interleave. However, this choice for the best level of interleave depends on a number of factors; in general one could try to predict the possible increase in frequency derived by the reduction of the CP and the increased area due to the additional area and try to understand what is the optimum; otherwise, since this optimization mechanism is extremely simple and linear, one could adapt its circuit for different levels of pipeline and then analyze what is the best option to choose. This is what we did in our case study and results of this analysis are summarized in following section.

Interleaving of DGS_S-W Systolic Array

DGS_S-W is a WIL-PT Systolic Array. The optimization procedure described before consists of three different phases: the first is the reduction of the critical path by introducing latches in each PE (pipelining); the second is the synchronization of other signals by inserting additional registers, and the third definition of the way to provide inputs in a non-contiguous manner (interleaving), so that there is not a reduction of throughput.

In order to pipeline the PE, it is necessary to insert registers and flip-flops, increasing the latency but without affecting the behavior of the circuit. In order to find the maximum, MAX_4_GAP_EX block has three subtractors in parallel; these subtractors are built as RCA with an inverted input and initial carry $c_0 = 1$. For this reason the critical path crosses the ADD_SAT and the MAX_4_GAP_EX; the startpoint of the critical path is instead in PE_CONFIG, exactly in a flip flop that stores the value of **aa_or_gap_reg**. In fact, there is a full combinational path between this register and the one at the output of MAX_4_GAP_EX.

In order to split the critical path, a register must be inserted between ADD_SAT and MAX_4_GAP_EX block. To guarantee synchronization (step 2 of the procedure), registers must be inserted also at the outputs of the other two adders. Consider the signal **f_1**: this is used inside one ADD_NORM but also stored inside a register, in order to be used in following cycle in ADD_SAT. When moving from no-interleave to interleave 2, it is clear that the storing register must be duplicated, because it will be used after two cycles. For the same reason, it is needed to add one register in the maximum evaluation loop. Finally, one more register is inserted in the **aa_code** chain to adapt to interleave 2. This however must be used during computation phase only, while during configuration phase, when **aa_code** is used to send AA codes of the *Qry*, the chain must be made by one register only. This can be achieved using a multiplexer.

The processing element adapted for interleave 2 is shown in Fig. 6(b). Further levels of interleave can be achieved by inserting registers to split the critical path and additional registers to guarantee synchronization. For each additional level of interleave, the registers to guarantee synchronization are: one more register for **aa_code**, one to store **f_1**, and another inside the maximum evaluation loop shown at the bottom of Fig. 6(b). Moreover, one register must be added in PE block, to store the **subject_id**. In order to split the critical path, interleave 3 can be achieved by inserting registers in the middle of ADD_NORM and ADD_SAT, thus splitting the carry propagation critical path, as in Fig. 7(b). The same can be done for interleave 4 acting on the maximum evaluation blocks, i.e. MAX_4_GAP_EXT and MAX_3_SIMPLE, as in Fig. 7(c). Finally, we have achieved interleave 5 by inserting a register at the input of ADD_SAT, since the critical path propagates at this point from PE_CONFIG till the register in the middle of ADD_SAT, as in Fig. 7(d). To guarantee synchronization, registers have been inserted at the output of the two ADD_NORM blocks. Further levels of interleave would have not split the CP significantly, and for this reason we stopped at level 5. Results in next paragraph show that our choice was correct since the best results are achieved with interleave 2 and 3. Further improvements could be obtained with a change og perspective in the maximum calculation [26].

In order to have the Systolic Array working at its best, it is also necessary to interleave input data. This can be done acting on the test-bench at simulation level. The database is stored in a file in the following way: each line represents a sequence, that starts with an identification number (**subject_id**), followed by the length of the sequence, i.e. the number of AAs, and then the sequence starts (in the sequence each number corresponds to an AA). The test-bench must then be able to read N sequences at the same time, where N is the level of pipeline. Moreover, when a sequence finishes, it must restart computation for that "slot" only, and start inputting the first sequence not yet initialized. Values belonging to different evaluations coexist inside each PE. For example, as explained before, to achieve interleave 2 it is required to double the register that stores **f_1**; with interleaving the added register is filled with the value of **f_1** of the other independent operation. As said, after one sequence has finished, it is necessary to clean the memory in registers storing the values evaluated with this sequence, in order to start then the new computations. This is done through a reset signal; this signal must affect only this sequence and not the others that are still under computation, and this is done resetting only some registers, as shown in Fig. 6(b). The other registers are reset only during

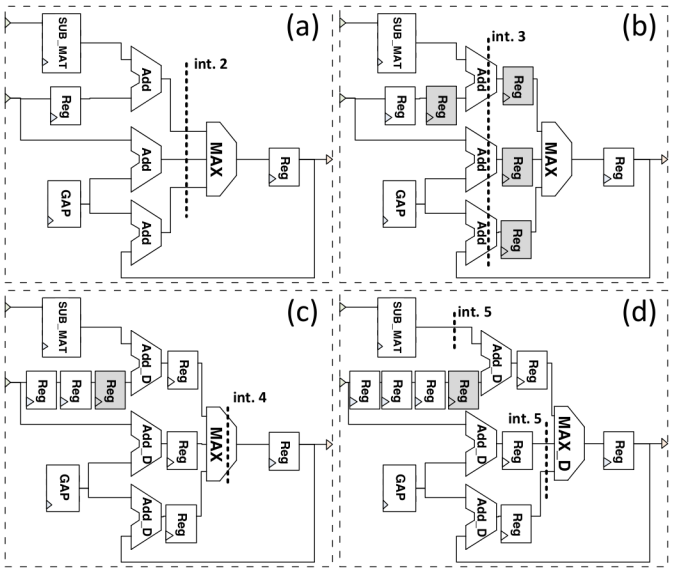


Fig. 7. A simplified structure of PE_CALC with "interleave cuts": each cut represents additional registers that must be placed to achieve a deeper level of interleave. **(a)** Original PE_CALC highlighting the place where to cut for interleave 2. **(b)** PE_CALC for interleave 2, achieved inserting additional registers highlighted in grey. The cut for interleave 3 is shown as well. **(c)** PE_CALC for interleave 3, achieved with one additional register and pipelined adders ADD_D in place of the original ones. The cut for interleave 4 is shown as well. **(d)** PE_CALC adapted for interleave 4, using an additional register, in grey, and a pipelined maximum evaluation block MAX_D in place of the original one MAX. Cuts to achieve interleave 5 are shown as well.

TABLE I
RESULTS OF THE SYNTHESIS OF AN SA WITH 30 PES IN ASIC, USING A $45\,nm$ LOW-POWER LIBRARY.

	Clock Frequency (MHz)		**Total Area** $(10^4 \mu m^2)$		**Area per *Sbj*** $(10^4 \mu m^2)$		**Total Power** (mW)		**Power per *Sbj*** (mW)	
	NO	PCO	NO	PCO	NO	PCO	NO	PCO	NO	PCO
no-int.	207.0	**383.1**	9.0	11.4	9.0	11.4	8.4	**21.7**	8.4	21.7
int. 2	349.6	625.0	10.7	13.0	5.4	6.5	10.4	35.8	5.2	17.9
int. 3	406.5	724.6	12.6	14.8	4.2	4.9	13.6	46.3	4.5	15.4
int. 4	465.1	892.8	16.9	18.8	4.2	4.7	20.7	104.5	5.2	26.1
int. 5	**534.7**	1190.5	18.8	22.2	3.8	4.4	**23.8**	241.1	4.8	48.2

configuration phase, in order to clean their content for the initialization.

VII. RESULTS

In this section the results that can be achieved adopting *pipeline interleaving* for the DSG_S-W Systolic Array case study are resumed and analyzed. Results have been carried out in CMOS technology both in FPGA and ASIC.

As target FPGA for our design we have used Xilinx Virtex-5 XC5VLX330T, which comprises 51840 slices, each containing 4 LUTs and 4 flip-flops.

In ASIC, we have synthesized the SA with a $45\,nm$ low-power library. We have two series of results: one achieved without setting constraints to the synthesizer (NO - No Optimization), and the other fixing stringent constraints on the maximum clock period but at the same time trying to reduce the dissipated power (PCO - Power Clock Optimization). The results are summarized in TABLE I.

We can notice an important trend in speed improvement with interleaving: moving from the case of no interleave to interleave 5 it is possible to achieve an improvement in frequency of 158%. If the designer has to increase the operating frequency of the circuit, he can decide to: optimize using synthesizer techniques (no interleaving with Power Clock Optimization), or adopt pipeline interleaving (n-interleaving with No Optimization). If we compare these two situations (no interleaving PCO and interleaving 5 NO) in TABLE I we notice that required power is similar, while with interleaving we can achieve an higher operating frequency. In TABLE I we report also the Power per *Sbj* consumed by the circuit; this is the dynamic power normalized by the number of parallel operations (interleaved) that are executed. It can be noticed that in NO the no interleave case is the worst one. The same consideration can be done also for the Area per *Sbj* as reported in TABLE I.

In general PCO, with respect to NO, requires an enormous increase of power, in fact the power trend in the PCO case with interleaving is exponential.

We want to highlight here the reasons that lead to an exponential trend in the increase of dynamic power. It is possible to estimate the dynamic power dissipated by a digital circuit as the sum of the powers of all the gates, which can be evaluated using equation $P_{dyn} = E_{SW}\, C_L\, {V_{DD}}^2\, f_{clk}$ [27], where E_{SW} is the switching activity, i.e. the probability that the gate output switches from 0 to 1 or from 1 to 0, C_L is the load capacitance associated to a node, V_{DD} is the supply voltage and f_{clk} is the operating frequency.

Given this relation, we have noticed that frequency increases with a linear trend and this reflects in P_{dyn} equation as a linear increase of the dynamic power of each gate. Moreover, area increases as well with a linear trend, due to the increased number of registers present in each level of pipeline, so, the number of gates increases and this reflects in the total dynamic power. Finally, synthesizer tries to optimize the speed of the circuit adopting big computational blocks that have high load capacitance C_L. It is then clear that the effect on power is exponential due to different and unrelated linear increases, in frequency, required area and load capacitance.

We want to stress the importance of *pipeline interleaving*, referring to TABLE I. If we compare the dynamic power dissipated by the optimized (PCO) no-interleave case, this is almost equal to that dissipated interleave 4 without optimization; however, the achieved clock frequency in interleave 4 is 20% higher than that of PCO no-interleave. We can thus highlight that *pipeline interleaving* allows increasing frequency without being affected by the exponential increase of power that occurs when achieving faster circuits using synthesizer optimization. This is further highlighted in Fig. 8. The relation between frequency and chip area increase is instead shown in Fig. 9.

A. CUPS evaluation

We want to evaluate the Cells Updates Per Second (CUPS), that is a very common performance indicator for systolic arrays architectures, as a measure of how many PEs are updated in a second. For a systolic array devoted to protein

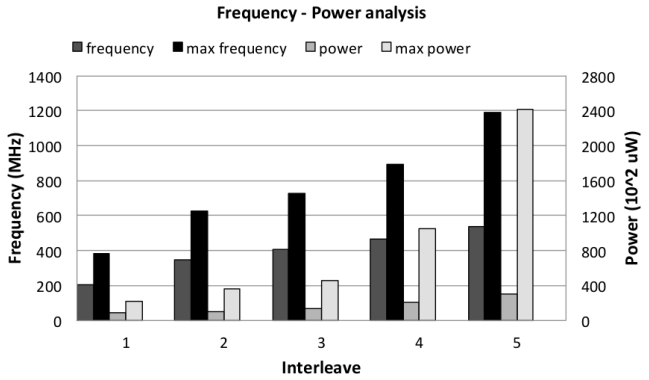


Fig. 8. Frequency - Power relation: for each level of interleave it is shown the maximum frequency and the dynamic power required in cases NO (labels "frequency" and "power") and PCO (labels "max frequency" and "max power").

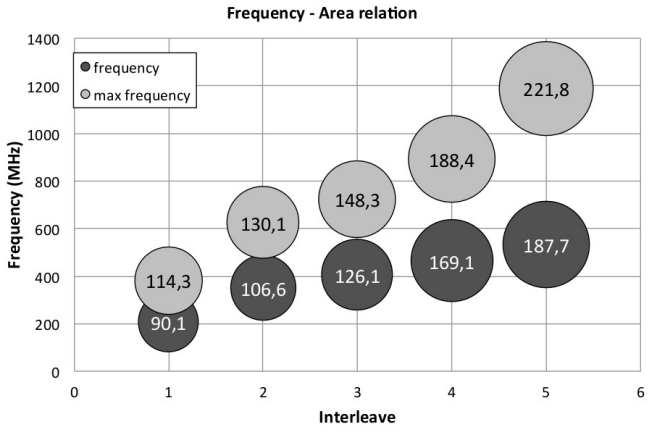


Fig. 9. Frequency - Area relation: the size of each bubble represents the area of the chip in $10^3\ mm^2$.

alignment, the peak CUPS that can be obtained are given by equation (8):

$$CUPS = f_{clk} \times \#PEs \tag{8}$$

where $\#PEs$ represents the maximum number of PEs that can be inserted in a given area. This area is well defined for FPGAs, because it represents the number of slices in the specific FPGA used. It is instead not defined for ASIC, and we must find an equivalent area that gives us comparable results. We assume for both ASIC and FPGA a linear relation between area and number of PEs, i.e. $A_{tot} = \#PEs \times A_{PE}$, where A_{tot} is the total area and A_{PE} is the area occupied by the single PE. As reference area for the ASIC we have chosen the one that is able to contain as many PEs as can be contained by the target FPGA in the case of no-interleave.
For the FPGA we have synthesized an SA with 100 cells, and we have evaluated the average area occupied by each cell assuming a linear relation as said before. Results are summarized in TABLE II, where GCUPS represents the Giga CUPS.

We can notice that the value of GCUPS increases with the increased level of interleave. This is not respected in interleave 4 for two reasons: first, it was not possible to achieve an improvement in clock frequency with respect to interleave 3, but there was an increase in required area due to increased number of registers, causing a reduction of the number of PEs that can be placed in the FPGA; second, the area estimation might be incorrect, in fact it was expected a value between those of interleave 3 and 5. This again causes less PEs to be placed in the FPGA, thus reducing the GCUPS.

TABLE II
RESULTS ACHIEVED IMPLEMENTING THE DSG S-W IN THE TARGET FPGA.

	Clock Frequency (MHz)	**Slices (100 PEs)**	**max #PEs**	**GCUPS**
no-int.	137.51	18474 (35.6 %)	280	38.50
int. 2	185.71	20958 (40.4 %)	247	45.87
int. 3	206.02	21162 (40.8 %)	244	50.27
int. 4	206.02	31624 (62.9 %)	163	33.58
int. 5	324.10	29745 (57.4 %)	174	56.39

It is possible to map 280 cells in the target FPGA without interleaving; this means that the target area for the ASIC is the one that can allocate 280 cells in the case no-interleave without optimization (NO). Results are summarized in TABLE III. As we can notice, in FPGA the best GCUPS performance in the case without optimization can be achieved with interleave 2.If we exploit instead the clock and power optimization achievable with the synthesizer, the performance increase with interleave, except for the case interleave 4.

TABLE III
ASIC EVALUATION OF GCUPS USING A $45\,nm$ LOW-POWER LIBRARY. THE NUMBER OF PES IS OBTAINED CONSIDERING AN EQUIVALENT CHIP AREA TO MAP 280 PES IN THE NO INTERLEAVE NO CASE.

	No Optimization (NO)			**With Optimization (PCO)**		
	$\mathbf{f_{clk}}$ (MHz)	**#PEs**	**GCUPS**	$\mathbf{f_{clk}}$ (MHz)	**#PEs**	**GCUPS**
no int.	207.0	280	57.971	383.1	220	84.291
int. 2	349.6	236	**82.517**	625.0	193	120.625
int. 3	406.5	199	80.894	724.6	170	123.188
int. 4	465.1	149	69.302	892.8	133	118.750
int. 5	534.7	134	71.658	1190.5	113	134.524

B. Total time evaluation

Increasing interleave level leads to a reduction of the number of PEs that can be placed in an FPGA. If the *Qry* length is greater than the number of PEs, two or more passes of the database sequences through the array are required. This means having a second initialization phase, that however can be interleaved with previous evaluation phase and thus requires only 1 cycle for each AA of the query to be loaded. During the second evaluation phase, results of first pass are loaded again into the Array for a second pass. Call i the interleave level, $L_k(i)$ the length of the query loaded at step k, n the number of *Sbj* to be scanned, of length L_s. $L_k(i)$ is equal to the number of PEs if a new pass of is required at step k, otherwise it is 0. The total time to scan the whole database is

then given by equation (9):

$$T(i) = T_{clk}(i) \sum_{k} [L_k(i)(1+i) + n\, L_s] \qquad (9)$$

For each step k, three elements are added to evaluate the total time of the step: $L_k(i) \times 1$ is the total time required to initialize the array; $L_k(i) \times i$ is the time required by each input to produce a result at the output of the array; $L_s \times n$ is the delay to provide last input to the array.

Of course the clock period is also dependent on interleave level. As an example, using values from TABLE II, with a *Qry* length of 260 AAs, a database of 300 proteins of 1000 AAs each to scan, we obtain a total time of $T(1) = 2185\,\mu s$ for the Array without interleaving, while $T(5) = 1857\,\mu s$ for the Array with interleave 5. Notice that using the array without interleaving there is not the need of a second pass, that it is required in the other cases. Yet, the total time is lower in interleave 5 case because of the higher operating frequency. Depending on the parameters of equation (9), other levels of interleave may be advantageous over the original case.

C. Global System Considerations

The encouraging performance improvement we have achieved through our pipeline interleaving technique must be sustained by the other elements of the global architecture. Usually a Systolic Array is used as an attached processor connected to the Host through a bus; the host provides data and control signals to the SA. The array processor is composed of an array controller and an interface unit: this is usually done as a chain of buffers. A memory system is connected to the bus as well, as shown in Fig. 10. In case of more complex structures with multiple processing units, also hierarchical shared memory organization could be taken into consideration [28].

In order to compare speed of components, we can introduce a new unit of measurement called *Amino Acids per second* $[AA/s]$ that represents how many AAs can be processed in one second. In the DGS_S-W an AA is represented with 5 bits, and other 3 bits must be sent for control purposes to the SA. This means that $1\,AA/s = 8\,b/s$. If we design our architecture with a PCI Express 3.0 x8 (8-bit parallelism) we are able to sustain speeds up to $8\,GAA/s$, since the speed of a line is $8\,Gbit/s$. Fast RAM can provide $3.2\,GB/s$, that in our case correspond to $3.2\,GAA/s$. FPGAs then can have a PCI Express Interface block integrated, as in the case of Xilinx Virtex 7, that can guarantee the maximum speed communication. A full analysis of the global system is out of the scope of this article, but these few considerations show that it is possible to design it supporting an SA executing up to $3.2\,GAA/s$. In case it is required to have a faster SA, then design techniques such as parallel buffering can be used.

D. Speed-Power product evaluation

One common indicator of the performance of a digital IC is the speed power product, also referred as figure of merit of a digital IC [29]. It is defined as the product of propagation delay (in nano seconds) and power dissipation (in mW) and is measured in pico joules. TABLE IV summarizes results for the ASIC; the best result (lowest value) is obtained with interleave 2 without optimization, and the second best with interleave 3; this is coherent to the results achieved for the GCUPS evaluation.

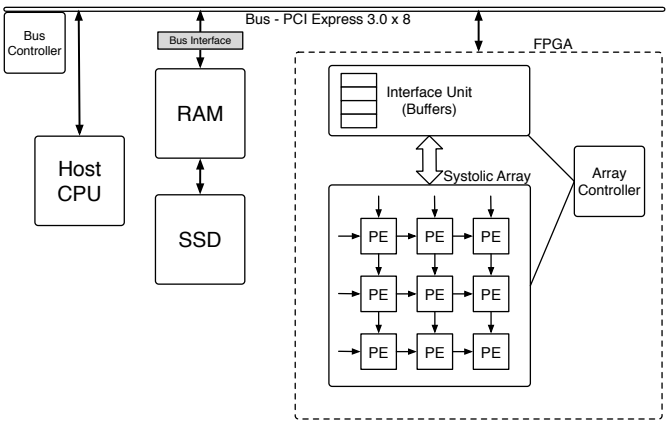


Fig. 10. Global system architecture: a bus connects the host processor with the systolic array and the memory. The Systolic Array is connected to the bus with an interface unit and controlled by an array controller.

TABLE IV
SPEED-POWER PRODUCT EVALUATION FOR THE ASIC. AMONG THE NO OPTIMIZATION (NO) CASES, INTERLEAVE 2 GUARANTEES THE BEST TRADEOFF BETWEEN OPERATING FREQUENCY AND POWER DISSIPATION.

(pJ)	no int.	int. 2	int. 3	int. 4	int. 5
NO	40.40	**29.90**	33.53	44.43	55.71
PCO	56.70	57.27	63.79	117.01	202.52

VIII. CONCLUSIONS

CMOS scaling limit will not guarantee in next years a direct increase in operating frequency; for this reason Systolic Arrays are back in the limelight to increase the performance of a computing system. In particular, they reveal to be the best choice to implement Protein Sequence Alignment algorithms, such as the Smith-Waterman. In this article we demonstrate how it is possible to increase the performance of a Systolic Array using a technique called *pipeline interleaving*: this technique requires loop pipelining inserting additional registers and data interleaving. In this way it is possible to reduce the critical path and thus increase the frequency, without having a reduction of throughput.

The Smith-Waterman SA with Dynamic Gap Selector is first introduced and then optimized to support up to interleave 5. Results, achieved both in FPGA and ASIC with a $45\,nm$ low power library, show how this technique permitted us to obtain a circuit that is 2.5 times faster than the original one without having the relevant increase of power dissipation that results when using synthesizer optimization mechanisms. It turns out that *pipeline interleaving* should be always applied when possible because it guarantees higher frequencies at a reasonable area and power dissipation cost.

Pipeline Interleaving in SAs should be also exploited with new technologies, such as Quantum-dot Cellular Automata (QCA)

because of their intrinsic pipelined nature; our future efforts will concentrate in designing a DGS S-W SA in QCA and optimize it using pipeline interleaving to explore benefits of this technique with other technologies.

REFERENCES

[1] D. Rairigh, "Limits of CMOS technology scaling and technologies beyond-CMOS," 2005.

[2] A. Pulimeno, M. Graziano, and G. Piccinini, "Udsm trends comparison: From technology roadmap to ultrasparc niagara2," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 7, pp. 1341–1346, 2012.

[3] H. Kung, C. Leiserson, and C.-M. U. D. of Computer Science, *Systolic Arrays for (VLSI)*, ser. CMU-CS. Carnegie-Mellon University, Department of Computer Science, 1978. [Online]. Available: http://books.google.it/books?id=pAKfHAAACAAJ

[4] L. Hasan and Z. Al-Ars, *An Overview of Hardware-Based Acceleration of Biological Sequence Alignment*. InTech, 2011.

[5] M. Vacca, M. Graziano, and M. Zamboni, "Asynchronous solutions for nanomagnetic logic circuits," *J. Emerg. Technol. Comput. Syst.*, vol. 7, no. 4, pp. 15:1–15:18, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043643.2043645

[6] G. Urgese, M. Graziano, M. Vacca, M. Awais, S. Frache, and M. Zamboni, "Protein alignment hw/sw optimizations," in *IEEE Int. Conf. on Electronics, Circuits, and Systems (ICECS)*, dec. 2012.

[7] S. Y. Kung, *VLSI array processors*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1987.

[8] S. B. Pan and R.-H. Park, "Unified systolic arrays for computation of the dct/dst/dht," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 7, no. 2, pp. 413 –419, apr 1997.

[9] S. Panchanathan and M. Goldberg, "A systolic array architecture for image coding using adaptive vector quantization," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 1, no. 2, pp. 222 –229, jun 1991.

[10] G. Iyengar and S. Panchanathan, "Systolic array architecture for gabor decomposition," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 5, no. 4, pp. 355 –359, aug 1995.

[11] H. Lim and J. Swartzlander, E.E., "Multidimensional systolic arrays for the implementation of discrete fourier transforms," *Signal Processing, IEEE Transactions on*, vol. 47, no. 5, pp. 1359 –1370, may 1999.

[12] H. Herzberg and R. Haimi-Cohen, "A systolic array realization of an lms adaptive filter and the effects of delayed adaptation," *Signal Processing, IEEE Transactions on*, vol. 40, no. 11, pp. 2799 –2803, nov 1992.

[13] L.-W. Chang and M.-C. Wu, "A unified systolic array for discrete cosine and sine transforms," *Signal Processing, IEEE Transactions on*, vol. 39, no. 1, pp. 192 –194, jan 1991.

[14] H. Yeo and Y. H. Hu, "A modular high-throughput architecture for logarithmic search block-matching motion estimation," *Circ. and Syst. for Video Tech., IEEE Trans. on*, vol. 8, no. 3, pp. 299 –315, jun 1998.

[15] Y.-S. Jehng, L.-G. Chen, and T.-D. Chiueh, "An efficient and simple vlsi tree architecture for motion estimation algorithms," *Signal Processing, IEEE Transactions on*, vol. 41, no. 2, pp. 889 –900, feb 1993.

[16] B. Buyukkurt and W. Najj, "Compiler generated systolic arrays for wavefront algorithm acceleration on fpgas," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, sept. 2008, pp. 655 –658.

[17] W. Jin, C. Zhang, and H. Li, "Mapping multiple algorithms into a reconfigurable systolic array," in *Electrical and Computer Engineering, 2008. Canadian Conf. on*, may 2008, pp. 001 187–92.

[18] M. Gok and C. Yilmaz, "Efficient cell designs for systolic smith-waterman implementations," in *Field Programmable Logic and Applications, 2006. FPL '06. Int. Conf. on*, aug. 2006, pp. 1 –4.

[19] E. Chow, T. Hunkapiller, J. Peterson, and M. Waterman, "Biological information signal processor," in *Application Specific Array Processors, 1991. Proc. of the Int. Conf. on*, sep 1991, pp. 144 –160.

[20] M. Graziano, S. Frache, and M. Zamboni, "A hardware viewpoint on biosequence analysis: What’s next?" *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 4, pp. 29:1–29:21, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2504774

[21] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[22] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge, UK: Cambridge University Press, 1998.

[23] T. Smith and M. Waterman, "Identification of common molecular subsequences," *J. of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[24] K. Parhi and D. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters. i. pipelining using scattered look-ahead and decomposition," *Acoustics, Speech and Signal Proc., IEEE Trans. on*, vol. 37, no. 7, pp. 1099 –1117, jul 1989.

[25] ——, "Pipeline interleaving and parallelism in recursive digital filters. ii. pipelined incremental block filtering," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 7, pp. 1118 –1134, jul 1989.

[26] L. Amaru, M. Martina, and G. Masera, "High speed architectures for finding the first two maximum/minimum values," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 20, no. 12, pp. 2342–2346, 2012.

[27] J. Rabaey, *Low Power Design Essentials*, S. Verlag, Ed., 2009.

[28] M. R. Casu, M. Ruo Roch, S. Tota, and M. Zamboni, "A noc-based hybrid message-passing/shared-memory approach to {CMP} design," *Microprocessors and Microsystems*, vol. 35, no. 2, pp. 261 – 273, 2011, ¡ce:title¿Special issue on Network-on-Chip Architectures and Design Methodologies¡/ce:title¿.

[29] A. Singh, *Digital Principles Foundation Of Circuit Design And Application*. New Age International, 2006.

Giovanni Causapruno Giovanni Causapruno received the Dr.Eng. degree in Electronics Engineering from Politecnico di Torino, Italy, in 2012, where he is currently a PhD candidate in Electronics and Communications Engineering. His research interests include parallel processing architectures for nanotechnologies.

Gianvito Urgese Gianvito Urgese received the M. Eng degree in Electronics Engineering from Politecnico di Torino, Turin, Italy, in 2012, where he is currently pursuing the Ph.D. in Computer Engineering and Automatic Control. His research interests include high performance parallel processing architectures applied on bioinformatic algorithms.

Marco Vacca Marco Vacca received the Dr. Eng. degree in electronics engineering from the Politecnico di Torino, Turin, Italy, in 2008. In 2013, he got the Ph.D. degree in electronics and communication engineering. He is currently working as an Research Assistant in the Politecnico di Torino. Since 2010, he has been teaching Design of Digital Circuits and Power Electronics. His research interests include quantum-dot cellular automata and others beyond-CMOS technologies.

Mariagrazia Graziano Mariagrazia Graziano received the Dr.Eng. degree and the Ph.D in Electronics Engineering from the Politecnico di Torino, Italy, in 1997 and 2001, respectively. Since 2002 she is a researcher and since 2005 Assistant Professor at the Politecnico di Torino. Since 2008 she is adjunct Faculty at the University of Illinois at Chicago. Her research interests include design of CMOS "beyond CMOS" devices, circuits and architectures. She is author and co-author of more than 90 published works.

Maurizio Zamboni Maurizio Zamboni got his Electronics Eng. degree in 1983 and the Ph. D. degree in 1988 at the Politecnico di Torino. He joined the Electronics Department of the Politecnico di Torino in 1983, became Researcher in 1989, Associate Professor in 1992 and Full Professor of Electronics in 2005. His research activity focuses on multiprocessor architectures design, in IC optimization for Artificial Intelligence, Telecommunication, low-power circuits and innovative beyond CMOS technologies. He is co-author of more than 120 scientific papers (three invited) and three books.