jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2011, VOL. 57, NO. 4, PP. 489–496
Manuscript received October 17, 2011; revised November 2011. DOI: 10.2478/v10177-011-0069-9

An Efficient Hardware Implementation of Smith-Waterman Algorithm Based on the Incremental Approach

Andrzej Pułka and Adam Milik

***Abstract*—The paper presents optimized hardware structure applied to genome alignment search. The proposed methodology is based on dynamic programming. The authors show how starting from the original Smith-Waterman approach, the algorithm can be optimized and the evaluation process simplified and speeded-up. The main idea is based on the observations of growth trends in the adjacent cells of the systolic array, which leads to the incremental approach. Moreover various coding styles are discussed and the best technique allowing further reduction of resources is selected. The entire processing unit utilizes fully pipelined structure that is well balanced trade-off between performance and resource requirements. The proposed technique is implemented in modern FPGA structures and obtained results proved efficiency of the methodology comparing to other approaches in the field.**

***Keywords*—DNA-tiles, pattern routing, pipelining, FPGA synthesis, parallelism and concurrency, reconfigurable systems, dynamic programming, systolic arrays.**

I. Introduction

RAPID development of natural sciences covering microbiology, molecular biology and computational biology observed for last few decades has faced researchers in front of new problems. First of all the results of these investigations have delivered huge amount of information that need to be processed and efficiently analyzed. GenBank [1] – an open access database collects data coming from various scientific centers. In this database we can find genome sequences of human beings. One of the branches of modern computational biology deals with the problem of searching for some characteristic properties of the DNA sequences [2], [3]. Genome patterns alignment belongs to such tasks. We would like to find if a given pattern (short sequence) could be localized within the long genome sequence (for human being it reaches 3GB). In other words the genome pattern alignment search task can be formulated as a mathematical (logical) problem.

The searching techniques based on classical software solutions have proved their correctness [4], but also showed their ineffectiveness when the amount of data gain GB. So, it is necessary to look for other, hardware-dedicated and effective solutions. The presented work proposes such a solution.

The limitations of modern computers and constraints of data processing cause that 'pure' software-based approaches to the genome searching are not sufficient in general. However there exist some software approaches based on Smith and Waterman (S-W) algorithm [5] combined with heuristics [6], [7] (BLAST, FASTA) or using parallel processors working in accelerated graphical environments [8], [9] (CUDA). The presented work focuses on hardware, reconfigurable solutions that seem to be more efficient and faster solutions in the field of pattern search alignment.

Some works based on programmable hardware structures have been proposed recently like, for example [7], [10]–[14]. Many efforts have been done into optimizing the hardware structures and making them more regular (systolic arrays) [15]–[18]. A very rapid development of FPGA devices allows further improving the performance and speeding up the data processing [19], [20].

This paper presents such a methodology implemented in the optimized reconfigurable hardware structure, which is much more effective solution than software based approaches. The paper is organized as follows: after a short recall of the Smith-Waterman methodology, the main idea of the proposed optimization based on the appropriate penalty values selection is given. Then we discuss the growth trends in the matrix and describe the algorithm by the simplified truth tables. In the next step we show how to reduce the logical equations describing the processing path by appropriate selection of codes. The presented methodology also considers trace back path allowing final localization of the found alignment. The experimental results and comparisons to other approaches in the field conclude the paper.

II. Problem Formulation

We can formulate the problem as follows: investigate a given, very long chain of symbols (here DNA chain) and try to localize best alignments between this chain and a set of query short sequences with assumed accuracy expressed by the penalty function. In case of DNA sequences, both chains, the reference sequence (long) $Ref = \{R_1, ..., R_n\}$ and the query sequence (short) $Qry = \{Q_1, ..., Q_m\}$ consist of symbols from 5th element set {A, C, G, T and X}, which stand for 4 DNA symbols and un unknown symbol. We need to stress up that by an alignment we not necessarily look for exact match between sequences, but we allow some differences (distance) expressed by the penalty function (1).

A. Smith-Waterman Algorithm

We can distinguish offline (indexed) and online searching techniques in the field. The hashing methods, investigated and

A. Pułka and A. Milik are with the Institute of Electronics, Silesian University of Technology, Gliwice, Poland (e-mails: apulka@polsl.pl, amilik@polsl.pl).

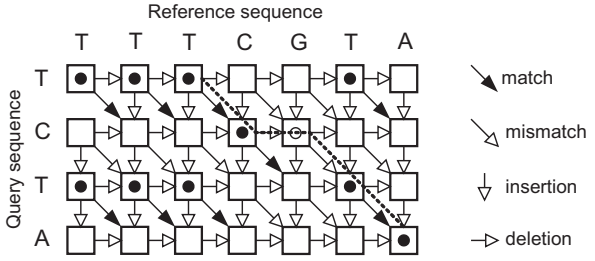

Fig. 1. Example of an array filled by S-W algorithm. The dotted line shows the path for best alignment.

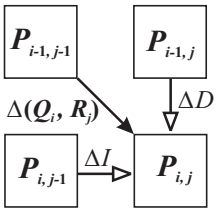

Fig. 2. Iterative evaluation of the scores from the equation (1).

improved for years, are not sufficient for this purpose and elementary online techniques are based on dynamic programming [4]. The fundamental methodology is based on Smith-Waterman algorithm [5] where the best alignment is evaluated recursively by the filling in the matrix of n+1 by m+1 cells. (see Fig. 1). The penalty function determining the movement within the array is expressed by the following equation:

$$P_{i,j} = \begin{cases} P_{i-1,j-1} + \Delta_{i,j} & match/mismatch \\ P_{i,j-1} + \Delta_{\_,j} & insertion \\ P_{i-1,j} + \Delta_{i,\_} & deletion \end{cases} \tag{1}$$

where:

$$\Delta_{i,j} = \begin{cases} 0 & when\ Q_i = R_j \quad (match) \\ \Delta M & otherwise \quad\quad\ (mismatch) \end{cases} \tag{2}$$

Cells hold partial scores (penalties) of the optimal path running through them. The first element of the matrix contains initial values and the following steps are computed recursively according to the optimal scores for the internal entries based on their neighborhood (Fig. 2).

B. Lipton Lopresti Simplification

Lipton and Lopresti [16], as first researchers, noticed that the appropriate selection of penalties has strong impact on the evaluation efficiency of the original S-W (Smith-Waterman) algorithm. We have also found that the direct approximation of the algorithm very quickly reaches the limits – we call it the saturation level of the efficiency. Our investigations of the numerical algorithm [21] have showed that it is possible to reduce the number of elements and simplify the structure by eliminating the sequential logic and increasing the number of asynchronous processing elements.

At first we have assumed the smallest integer values of penalties for the S-W algorithm [5], i.e.:

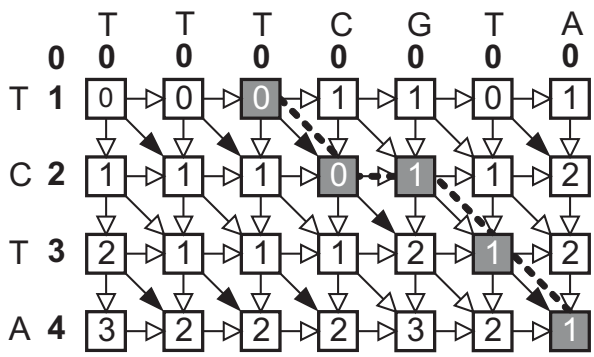

Fig. 3. Array from Fig. 1 filled with penalties evaluated according to the formula (3).

$$\begin{cases} \Delta_{i,j} = 2 & mismatch \\ \Delta I = \Delta_{\_,j} = 1 & insertion \\ \Delta D = \Delta_{i,\_} = 1 & deletion \end{cases} \tag{3}$$

And initialized the edge rows and columns of the array with values:

$$\underset{0 < i \le Q_{length}}{\forall} \quad P_{i,0} = i \tag{4}$$

$$\underset{0 < j \le R_{length}}{\forall} \quad P_{0,j} = 0 \tag{5}$$

Fig. 3 presents the fragment of the array from Fig. 1 filled with the values evaluated from the formula (3). We have denoted initial values in the upper row (above the array) (4) and on the left column (5) (just before the first column of the array) by boldfaces. Then, we have proved some interesting properties of the array [21]:

Property 1:

$$\underset{\substack{0 < i \le Q_{length} \\ 0 < j \le R_{length}}}{\forall} \begin{cases} P_{i-1,j-1} \le P_{i,j} \le P_{i-1,j-1} + 2 \\ P_{i,j-1} - 1 \le P_{i,j} \le P_{i,j-1} + 1 \\ P_{i-1,j} - 1 \le P_{i,j} \le P_{i-1,j} + 1 \end{cases} \tag{6}$$

Property 2:

$$\underset{\substack{0 < i \le Q_{length} \\ 0 < j \le R_{length}}}{\forall} \begin{cases} \text{P1: } P_{i,j} = P_{i,j-1} + \Delta x \\ \text{P2: } P_{i,j} = P_{i-1,j} + \Delta y \end{cases} \tag{7}$$

where: $\Delta x \in \{-1, 0, +1\};\ \ \Delta y \in \{-1, 0, +1\}$.

Property 3:

$$\underset{\substack{0 < i \le Q_{length} \\ 0 < j \le R_{length}}}{\forall} \quad 0 \le P_{i,0} \le i \tag{8}$$

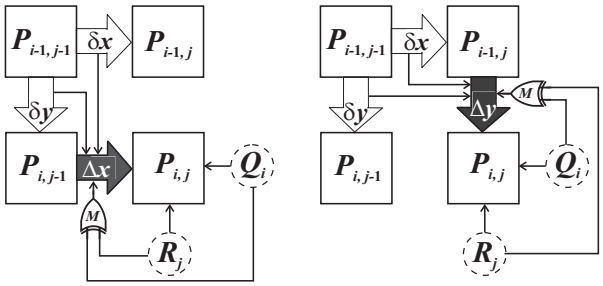

Fig. 4. Schematic diagram showing the idea of the growth trend functions.

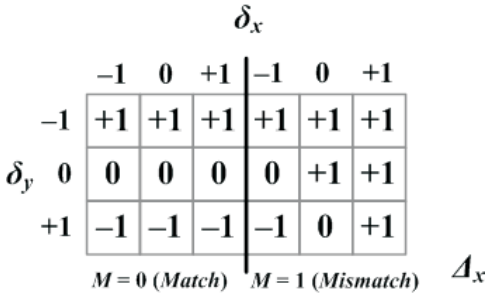

Fig. 5. Function table for the horizontal growth trend function Δx.

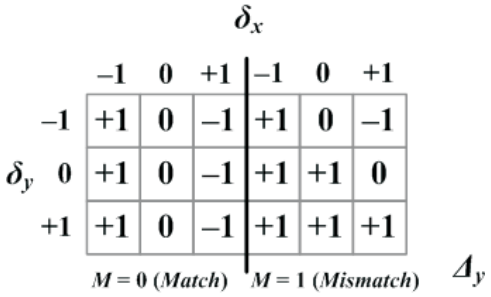

Fig. 6. Function table for the horizontal growth trend function Δy.

III. Main Idea of the Approach

The proposed approach relies on the above considerations and properties. The main idea of the hardware algorithm optimization is based on the observations of horizontal and vertical growth trends within the matrix. Fig. 4 presents both evaluation schemes for a given cell. To obtain the new value of the penalty for a cell $P_{i,j}$ we have to find the horizontal (δx) or vertical (δy) growth function, where symbols x and y denote the growth trends in horizontal and vertical direction, respectively. The third argument of these functions the variable M reflects the result of comparison between the current query symbol Q_i and the reference symbol R_j, so both functions depend on three arguments: x, y and M. Moreover, as we can see in the symbolic function tables (truth tables in Fig. 5 and 6) both growth functions: x and y are symmetrical (identical) with respect to the variables δx and δy:

$$\begin{aligned} \Delta x &= f_x(\delta x, \delta y, M) = f_y(\delta y, \delta x, M) \\ \Delta y &= f_y(\delta x, \delta y, M) = f_x(\delta y, \delta x, M) \end{aligned} \tag{9}$$

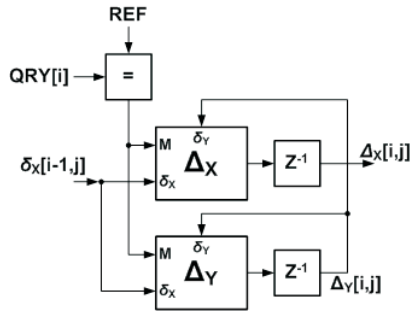

Fig. 7. The elementary unit ΔSW evaluating the growth trend function for a single cell.

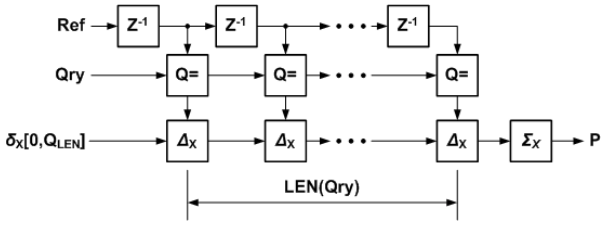

Fig. 8. The pipeline structure based on the accumulation of the horizontal growth trends.

Actually, the function tables are truth tables. The symbols from the tables have real logical meaning, i.e. they denote appropriate combinatorial functions: the symbol "+1" stands for the increment (*INC*), the symbol "−1" corresponds to the decrement (*DEC*) and "0" means no operation (*NOP*). In the next section we will discuss the problem of coding of these operations and their impact on the quality of the final implementation.

This leads to the development of the basic building block of the growth trend based Smith-Waterman (ΔSW) implementation. Fig. 7 presents the structure of the elementary unit ΔSW. It delivers growth coefficient values for both Δ_x and Δ_y calculations.

The current value of the penalty for a given (current) cell $P_{i,j}$ can be evaluated in two ways (depending on the selected growth trend function) as a result of the accumulation of the subsequent growth trends values, i.e.:

$$\begin{aligned} P_{i,j} &= P_{i,0} + \sum_{k=1}^{j} \delta x_{i,k}; \text{ and } P_{i,0} = i \\ P_{i,j} &= P_{0,j} + \sum_{l=1}^{i} \delta x_{l,j}; \text{ and } P_{0,j} = 0 \end{aligned} \tag{10}$$

IV. Main Issues of Hardware Implementation

In the previous sections we have presented how the original software (mathematical) algorithm has been converted to hardware structure. We have reduced the computational complexity thanks to the optimal selection of penalties and growth trend function. The obtained solution in comparison to the previous applications [21] and other approaches [10], [15] looks very promising, but there are several problems that should be solved

before the final hardware implementation begin to work. Otherwise the initial optimization of the algorithm could be neglected by unreasonable technology mapping. In this section we present considerations concerning details of the hardware implementation. We focus on the selection of the growth trend function, coding styles and the trace back implementation.

A. Selection of the Growth Trend Function

From the expressions (9) and (10) we can find that there are two alternative methods of the penalty calculation with the same architecture of a basic cell. We can operate in one of two possible directions: horizontal (parallel to the row) or vertical (parallel to the column) basing on functions Δ_x or Δ_y, respectively. Theoretically both functions are identical (symmetrical), however, in the case of the application to chain alignments, the more convenient implementation is the one that operates in the axis parallel to the reference sequence. In the presented example the reference pattern is placed in the row (along x axis) of the array, so the circuit should utilize Δ_x function. This approach allows calculating the final value of the elements of P systolic array by successive summation of growth coefficients (equation (10) and Fig. 8). This structure is able to process a single input symbol per clock cycle in pipelined fashion. The algorithm property (the convergence for a mismatch) allows eliminating complicated data flow control for the pipeline processing.

B. The Optimal Coding Style

Functional tables presented in Fig. 5 and Fig. 6 describe growth trends functions symbolically. To obtain final implementation we need to replace symbolic values "$+1$" and "-1" with real codes. Our first solution presented in [21] was the direct implementation of the algorithm with application of incrementing/decrementing counters (counting registers). The idea was based on the observation of the tables, which contain only few zeroes, so we can code active bit count corresponding to "±1" and the next bit responsible for counting direction (up or down). However, taking into account specificity of the hardware implementation in FPGA structures, we found this solution not optimal. So, we have decided to use semi-one-hot coding style, where every column and row of the table is coded in two bits: $\{\delta x_1; \delta x_0\}$ and $\{\delta y_1; \delta y_0\}$, respectively. The output function – the growth trend function SW of a basic building block also consists of two bits: $\{\Delta x_1; \Delta x_0\}$. This philosophy allows optimally using the LUTs of the FPGA resources.

Fig. 9 presents the full truth table for the single ΔSW block (combinations "11" are not used (no meaning), so they consist of don't cares). Logical equations for horizontal growth trend function Δ_x are following:

$$\begin{aligned} \Delta_{x1} &= \delta_{y0} \wedge (\overline{M} \vee \delta_{x1}) \\ \Delta_{x0} &= \delta_{y0} \vee (M \wedge \delta_{x0}) \vee (M \wedge \overline{\delta_{x0}} \wedge \overline{\delta_{y0}}) \end{aligned} \tag{11}$$

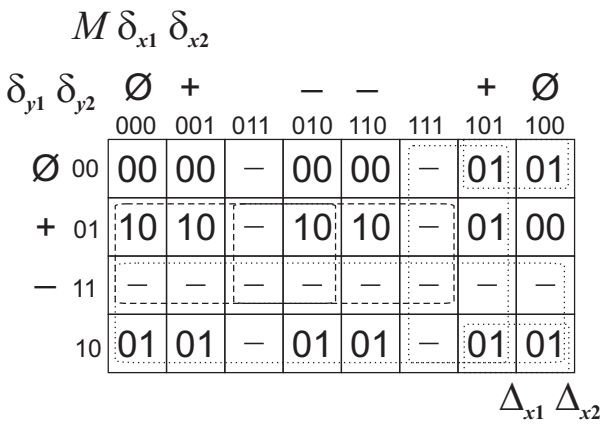

Fig. 9. Final coding for horizontal growth trend function Δx.

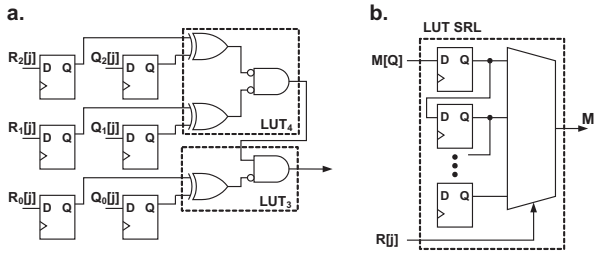

Fig. 10. Implementation of the match unit.

C. Some Problems of the Structure Mapping in Hardware

The original S-W algorithm transformations allow us to obtain the structure with extremely small calculation complexity. It is mainly based on combinatorial operations that constitute simple finite state machine (FSM). However we have identified some problems of final implementation and hardware mapping. The high level description assures only the functional correspondence that is not always optimal in the case of hardware resources. Modern FPGAs offer a lot of features that allow flexible and efficient implementation of different functions. The synthesis tool is not always able to optimally map the design as many of aspects are not described. Very often the algorithmic approach does not lead to the efficient implementation. A simplified and well implemented form of the algorithm is obtained by application of different methods of algorithm analysis and optimization. The match unit is the example of a low level mapping. It compares two symbols equality taken from the query and the reference sequences. The direct implementation of the equality operator is presented in Fig. 10a. It requires 6 D flip-flops (3 bit symbols) for the query and the reference and 2 LUT generators to implement the combinatorial part. The specific usage of a comparator assumes that before the match process begins, the query sequence is transferred to registers. The query symbol can be considered as a programmable constant value. Following this idea we can implement the match unit that is based on LUT configured as the programmable length shift register (SRL in Fig. 10b) that consumes 3 D flip-flops and a single LUT. This implementation reduces resource requirements and also the propagation delay by about 50%

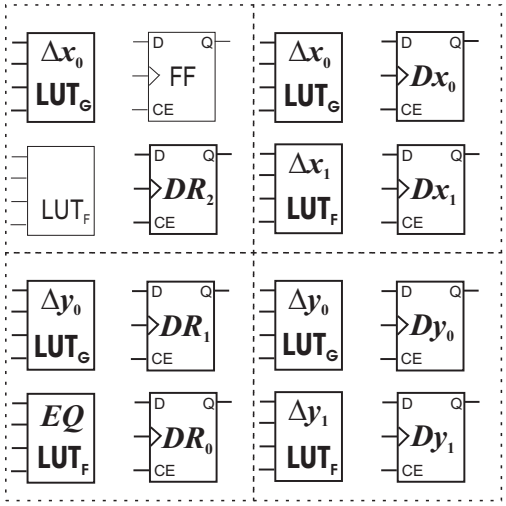

Fig. 11. Final mapping of the match unit.

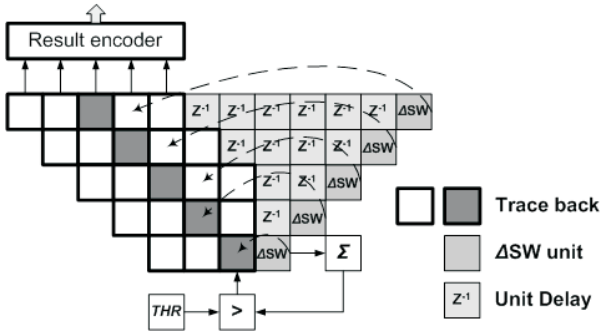

Fig. 12. The concept of the hardware trace back based on delay line.

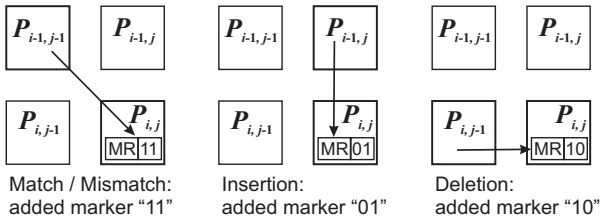

Fig. 13. Mechanism of marker registers updating.

from two layers to a single one.

Finally, the entire ΔSW cell can be implemented into 2 slices for families based on 4-input LUTs. A detailed exemplary mapping is depicted in Fig. 11. The short combinatorial path, which depth does not exceed two layers and short distant programmable connections guarantee a very short propagation time. The 6-input LUT FPGA families (Virtex-5 and later (Xilinx)) exhibit a little worse resource utilization. In general, the implementation of the basic cell requires: 2 slices, 7 flip-flops, one shift register and four 6-input LUTs or six 4-input LUTs.

D. The Trace Back Implementation

The original Smith-Waterman algorithm [5] evaluates cost of the optimal path (alignment) in a form of penalty function (1), but to obtain the exact region (to localize) of the best alignment we need to perform the backtrack. This trace back operation recreates the original path of the algorithm run. Because of the fact that the implemented pipeline based on the direct software procedure transformation into hardware implementation does not remember results of intermediate operations [21], it is necessary to introduce other mechanisms or to store the entire path. In our previous works we have introduced software-based procedure – the algorithm HIPAS [21] which is able to reconstruct the entire path (trace back) basing only on the final score. The main drawback of such a solution is necessity of stalling the pipeline for approximately $3 \times n$ clock cycles (where n is the length of the query sequence). This solution is good if we expect only few checks per sequence.

An alternative solution is additional hardware structure that should be added to basic building blocks (array cells). Fig. 12 presents the general idea of the hardware trace back solution. After each step of the systolic array evaluation process information necessary for reconstruction of the move is shifted (delayed) to be available (if needed) later. This scheme constitutes a kind of symmetrical structure to the original array (Fig. 12). If the system detects the good score (low penalty) below a given assumed threshold indicating that the region of good alignment has been localized, data from the output of this delay line is connected to the trace back structure and we can find the position of the beginning of the good alignment region.

This solution requires additional registers – marker registers (MR) responsible for remembering "moves" during the systolic array analysis (evaluation of S-W algorithm). When the procedure expressed by (3) is executed the contents of the cells are updated with a new (current) value of the penalty and the marker register is uploaded from the appropriate source (left, upper diagonal or upper cell). The contents of the register are supplemented by the current marker, i.e. the code of the last move. This code reflects the minimal value of the S-W algorithm (equation (1) and Fig. 1). Each marker is coded on two bits: "00" corresponds to "no move" (unused), "01" – vertical move (insertion), "10" – horizontal move (deletion) and "11" – diagonal move (match/mismatch). This mechanism, for ea given cell $P_{i,j}$ is illustrated in Fig. 13. In other words, when a given pipeline stream reaches the end (the final comparison of symbols is done), we have already entire trace and the location of the best alignment (Fig. 14).

The trace back mechanism is activated by penalty threshold block (THR in Fig. 12), which generates activation signal I for the first cell (the last symbol). And the activation signals I back propagate through the array reconstructing the entire path. As it was mentioned above, from each cell there are possible three moves "up" (vertical), "left" (horizontal) and "diagonal", so we need to supply each cell with the additional logic and connections between adjacent cells. We call these blocks trace back units (TBU). Each TBU is fed by delayed (not current! – see Fig. 12) contents of appropriate marker register and activation signals of previous neighbor cells (Fig. 16). TBU is responsible for generation of activation signal to only one of the adjacent cells (Fig. 15). This methodology allows for a

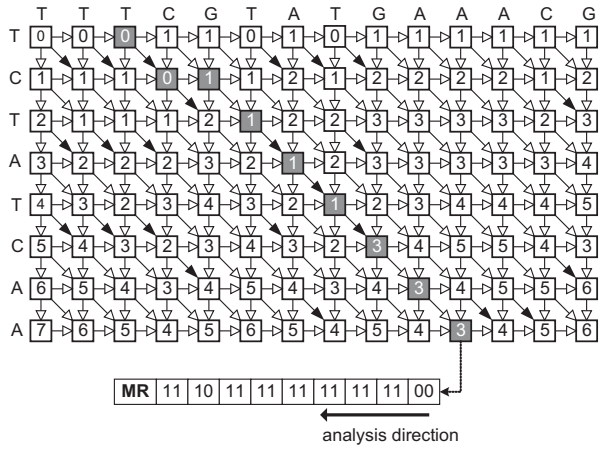

Fig. 14. Example of the good alignment score and the marker register for a sink cell with best score. Reconstructed path consists of shaded cells. (The initial edge values have been removed to increase readability).

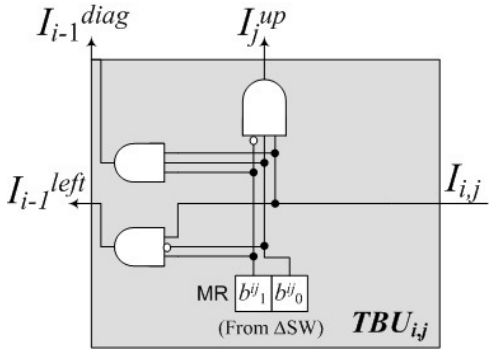

Fig. 15. Scheme of the trace back unit (TBU).

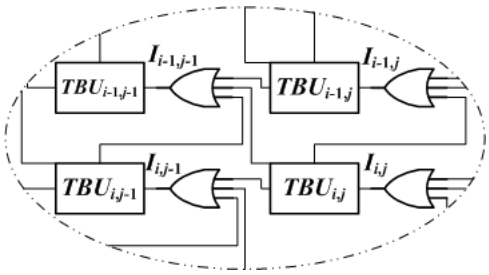

Fig. 16. Fragment of the trace structure with four adjacent TBUs.

very quick and simple reconstructing of the entire trace back path.

E. The Final Analysis of the Resource Requirements

In this subsection we summarize our considerations concerning implementation and try to determine hardware resources requirements for the entire SW system (together with the trace back path). In case of FPGA-based designs, the number of Look Up Table inputs is one of the most important factors determining the amount of required resources. For considered Xilinx families there are 4 and 6 inputs LUT generators. Let's assume that the unit is described by the query length Q_L and the maximal admitted number of insertions and deletions is not grater then d. It is assumed for rational purposes that $Q_L \gg d$.

According to the previously assumed values, the number of basic units, i.e. the number of ΔSW cells $N_{\Delta SW}$ equals to the length of the query sequence currently processed Q_L. Actually, from the practical point of view it is better to add some margin, but we can assume that the coefficient d covers this margin. So, the total number of basic cells in the trace back is expressed by:

$$N_{TB} = \sum_{i=1}^{R_L}(2 \cdot d + 1) - \sum_{i=1}^{d-1} i = \\ = R_L \cdot (2 \cdot d + 1) - \frac{d \cdot (d-1)}{2} \tag{12}$$

And the total number of 2-bit markers (MR) in the pipeline adjust shift register:

$$N_{SHR} = \sum_{i=1}^{d-1} i + \sum_{i=d}^{R_L} 2 \cdot i = \frac{R_L - d}{2} \cdot \Big[d + (d+1) \cdot \\ \cdot (R_L - d - 1)\Big] - \frac{d \cdot (d-1)}{2} \tag{13}$$

After evaluation of these numbers we can proceed to the mapping phase and the estimation of the possible implementation capabilities of a given, selected device. It is worth emphasizing that the unit requirements are quite large in comparison to other circuit's resources. The required shift register units are responsible for the implementation of the delay cycles necessary to adjust appropriate moment of the data arrival to the trace back processing unit. The implementation of the cycle adjust unit is well supported by modern FPGAs. The mentioned operation can be carried out by a LUT working in the shift register mode and the programmable delay time T_n is included in the range $\langle 1,\ 2^{L_i} \rangle$ (where L_i denotes the number of inputs of the LUT generator). To obtain the maximum processing efficiency of the system the D flip-flop, located next to the LUT, also should be utilized. For the considered families of FPGA devices, it is possible to implement up to 17-bit shift registers for 4-input LUTs or 33-bit shift register for 6-input LUTs (unfortunately technologically unit incorporates 64 stages, but only 32 of them are available in shift register mode). So the equation for shift register requirements expressed by LUT items is as follows:

$$N_{SHR} = \sum_{i=1}^{d-1} \left\lceil \frac{i}{max(T_n)} \right\rceil + \sum_{i=d}^{R_L} \left\lceil \frac{2 \cdot i}{max(T_n)} \right\rceil \tag{14}$$

Now, we can calculate the logic requirements for the desired length of the reference symbols and the desired match. To make the comparison lucid and legible in terms of FPGA devices we have used number of LUTs as a basic factor (Table I). It is justified also due to the fact that our architecture is very regular and basically only short connections between neighboring cells have been utilized (as well in terms of an FPGA architecture and the functional concept). The unit consists of three basic items that are: the **ΔSW** pipe, **SRL** –

TABLE I
RESOURCE REQUIREMENTS FOR FULL SMITH-WATTERMAN IMPLEMENTATION EXPRESSED IN LUT6

Parameters		Components Complexity			Total number of LUTs
Query length (symbols)	Max. Dist. (symbols)	ΔSW (LUTs)	TB (LUTs)	SRL (LUTs)	
512	15	4096	31504	7966	**43566**
1024	15	8192	63248	31813	**103253**
1024	31	8192	128032	31325	**167549**
1536	18	12288	113322	71409	**197019**

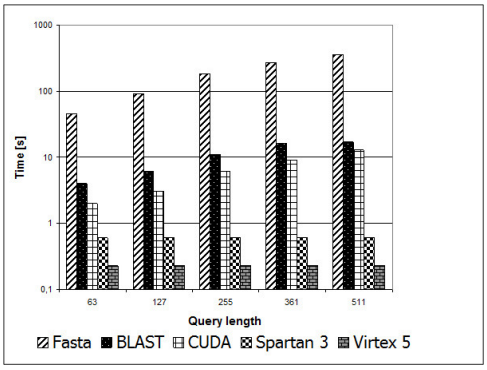

Fig. 17. Computational efficiency of different methodologies.

the pipeline result adjust register and the trace back systolic unit (**TB**). It is assumed that 85 – 90% of circuit resources can be assigned to the entire algorithm. The resource requirements have been determined for Virtex 5 VLX families. This family allows implementing the accelerator computing board thanks to PCIe embedded endpoints. As a reference, the LUT count is taken XC5VLX330T with the array 240×108 CLBs that in total gives 207360 LUTs with D-FF items.

It appears that with growing length of the reference sequence the linear growth of the ΔSW pipe and the trace back unit is observed. Similar conclusions can be drawn regarding the admitted region of the fitting described by "Max. Dist." and the trace back unit requirements. However, when the length of the reference sequence grows the requirements of the unit responsible for pipeline operation data adjust increase very quickly (two times faster than for other resources). The general logic requirements can be reduced by the implementation of RAMB36 units in a form of ring registers with the constant capacity.

V. CONCLUSIONS

We have presented a new approach to the application of modern high-density programmable hardware devices to computational biology. The main novelty of our approach, which allows optimizing the algorithm towards the final technology and modifying classical programming techniques with hardware-based solutions, is the application of the modern advanced design tools together with behavioral description of the algorithm in HDL. Moreover, we have split the searching process into two steps: a dynamic programming search and an examination of the selected patterns. The latter can be carried out by software based algorithm HIPAS [21] and/or the growth trend based trace back described in the paper. The final analysis of potential candidates can be performed effectively off-line for all queries as a single quick process. The obtained results, i.e. operating frequencies (2–3 times bigger) and density of the processing elements on the chip show that, in comparison with the previous works [8], [10], [11], [22], [23] our methodology, however not covering all possible cases so far, is a significant step forward in the field.

Designed computing accelerator has been implemented in ML-505 board. The on board FPGA device offers the embedded PCI Express communication core. This allows constructing a high bandwidth data and a control interface to transfer large amount of data. Together with the highly efficient query algorithm reconfigurable computing board can offer outstanding computation performance in genome calculation or other pattern matching tasks especially for very long input sequences.

The algorithm implementation process and its efficient mapping is a very challenging task. The HDL languages have been developed for about 30 years, however, they are not able to cover the wide range of aspects required for algorithmic approach implementation. Observations of the algorithm features bring out few interesting theorems that allow its further optimizations. The work described in [11] presents a very interesting approach based on automated generated systolic arrays (descriptions generated from C to VHDL by compilers). In that approach authors reported that it can process queries up to 1024 symbols with the maximum frequency of 174 MHz. The methodology described in our paper is based on Verilog description, which on one hand is closer to the algorithm (mimics the C-coded algorithm) and on the other hand is closer to the hardware and allows to take benefits of the hardware properties of FPGAs' structures. We are able to deal with queries consisting of up to 32400 symbols with the frequencies 400-600 MHz (within XC5VLX330T structure) under the assumption that we do not use the trace back. Actually, thanks to the regularity of the entire structure, we are able to almost evaluate any configuration of multiple queries at the same time, which gives the total number of 32400 symbols, i.e. $n \times m$ = 32400 (where: n is the number of queries and m stands for the query sequence length). Comparing to the multi-core implementations on Intel Xenon and Intel-Italium-2 applications reported in, the throughput of our system, expressed in GCUPS (giga cell-updates per second) is about 4000 faster (200 versus 0.049), so the methodology proposed here is comparable (even little better) to the one reported in [11]. As to computation efficiency the entire reference chain (genome) can be analyzed in less than 8 seconds. The proposed methodology has also been compared to other approaches: FASTA, BLAST and the multi-core processing system implemented in graphics card on general purpose computing machines [21]. Fig. 17 presents the results of these comparisons: the computation efficiency for query references of different lengths. The plot refers to search procedure in 91,694,534 amino acids chains. The diagram

contains two versions of the SW methodology, implemented in two major FPGA families Spartan 3 and Virtex 5. It is worth to emphasize that the search times of the presented programmatic approach are dependant on a query sequence length. Providing that the pipelined structure is able to hold entire sequence, it is possible to obtain results in the constant time and such a solution is limited only by the basic cell performance.

The new formula of the ΔSW algorithm radically changes data representation and reduces the complexity of the cell. Excellent operation parameters like very low hardware resource consumption per cell, very long query sequences directly processed and extremely high operation frequencies show usefulness of carried out research works. The obtained results show radical improvement in comparison to the direct and first optimized implementations of S-W algorithm. The presented methodology enables to reduce the resource utilization, simplifies the data and control path and increases the system throughput. The entire structure is very compact, neighboring cells are very close to one another and their interconnections are relatively short. We have managed to avoid the long signal distribution lines and thanks to this we have minimized the delays. Moreover, the delays are in practice reduced to LUTs generators, so the structure is very flat (one level logic).

The trace back processing based on the growth trend analysis is very simple, performed on the bit level. We have performed the worst case analysis of the trace back delays. We have found that the total delays of the LUT generator together with routing resources for the single layer (between adjacent cells) are: 590 ps and 540 ps for the family 1 and family 3 (see Virtex-5 Data Sheet [20]) respectively. So it appears that for the longest sequence with full trace back consisting of 1536 symbols (Table I) and the lower frequency (400 MHz), the entire pipeline has to be stalled for approximately 906 ns (363 clock cycles). We can conclude that for the proposed application, if the number of expected (searched) alignments is small (just a few) it is better to use HIPAS algorithm [21]. Otherwise, we have to implement the trace back hardware.

REFERENCES

[1] GenBank. (2011) The official web of national center for biotechnology information. [Online]. Available: http://www.ncbi.nlm.nih.gov/

[2] E. Pettersson, J. Lundeberg, and A. Ahmadian, "Generations of sequencing technologies," *Genomics*, vol. 93, pp. 105–111, 2009.

[3] H. S. Xu, W. K. Ren, X. H. Liu, and X. Q. Li, "Improving sequence alignment using class-specific score matrices," in *Proceedings of ICBBE 2008, The 2nd International Conference on Bioinformatics and Biomedical Engineering*, 2008, pp. 70–73.

[4] D. Gusfield, *Algorithms on strings, trees and sequences*. Cambridge University Press, 1997.

[5] T. F. Smith and M. S. Waterman, "Identification of common molecular sub-sequences," *Journal of Molecular Biology*, vol. 147, pp. 195–197, 1981.

[6] FASTA. (2011) Sequence comparison at the university of virginia. [Online]. Available: http://fasta.bioch.virginia.edu/

[7] M. C. Herbordt, J. Model, Y. Gu, B. Sukhwani, and T. VanCourt, "Single pass, blast-like, approximate string matching on fpgas," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Washington, DC, USA, 2006, pp. 217–226.

[8] Y. Liu, D. Maskell, and B. Schmidt, "Cudasw++: optimizing smith-waterman sequence database searches for cuda-enabled graphics processing units," *BMC Research Notes*, vol. 2, no. 1, p. 73, 2009.

[9] S. A. Manavski and G. Valle, "Cuda compatible gpu cards as efficient hardware accelerators for smith-waterman sequence alignment," *BMC Bioinformatics*, p. S10, 2008.

[10] K. Benkrid, Y. Liu, and A. Benkrid, "High performance biosequence database scanning using fpgas," in *Processing of 2007 ICASSP, IEEE International Conference on Acoustics, Speech and Signal*, vol. 1, april 2007, pp. 361–364.

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

[12] I. T. S. Li, W. Shum, and K. Truong, "160-fold acceleration of the smith-waterman algorithm using a field programmable gate array (fpga)." *BMC Bioinformatics*, vol. 8, p. 185, 2007.

[13] T. Oliver and B. Schmidt, "High performance biosequence database scanning on reconfigurable platforms," in *Proceedings of 18th International of Parallel and Distributed Processing Symposium 2004*, april 2004, pp. 192–199.

[14] Y. Yamaguchi, T. Maruyama, and A. Konagaya, "High speed homology search with fpgas," in *Proceedings of Pacific Symposium on Biocomputing02*, 2002, pp. 271–282.

[15] N. Hireche, J. M. P. Langlois, and G. Nicolescu, "A systolic array for sequence comparison based on two-logic-levels processing element," in *Proceedings of NEWCAS 2007, IEEE Northeast Workshop on Circuits and Systems*, aug 2007, pp. 73–76.

[16] R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," in *Proceedings of Chapel Hill Conference on VLSI*, 1985, pp. 363–376.

[17] R. Sastry and N. Ranganathan, "A systolic array for approximate string matching," in *ICCD '93, Proceedings of 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, oct 1993, pp. 402–405.

[18] F. Zhang, X.-Z. Qiao, and Z.-Y. Liu, "A parallel smith-waterman algorithm based on divide and conquer," in *Proceedings of Fifth International Conference on Algorithms and Architectures for Parallel Processing 2002*, 2002, pp. 162–169.

[19] TimeLogic. (2011) Decypher fpga biocomputing systems. [Online]. Available: http://www.timelogic.com/

[20] Xilinx. (2011) The official web site of the xilinx company. [Online]. Available: http://www.xilinx.com/

[21] A. Milik and A. Pułka, "Hardware oriented optimization of smith-waterman algorithm," in *Proceedings of ICSES 2010, IEEE International Conference on Signals and Electronic Systems*, Gliwice, Poland, sep 2010, pp. 319–322.

[22] S. Dydel, K. Benedyczak, and P. Bala, "Enabling reconfigurable hardware accelerators for the grid," in *Proceedings of PAR ELEC 2006, International Symposium on Parallel Computing in Electrical Engineering*, sep 2006, pp. 145–152.

[23] B. Phoophakdee and M. J. Zaki, "Genome-scale disk-based suffix tree indexing," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*. New York, USA: ACM, 2007, pp. 833–844.