Metrol. Meas. Syst., Vol. XIX (2012), No. 1, pp. 49-62.

METROLOGY AND MEASUREMENT SYSTEMS

Index 330930, ISSN 0860-8229

www.metrology.pg.gda.pl

MEASUREMENT ASPECTS OF GENOME PATTERN INVESTIGATIONS – HARDWARE IMPLEMENTATION

Andrzej Pułka, Adam Milik

Silesian University of Technology, Faculty of Automatic Control, Electronics and Computer Science, Institute of Electronics, Akademicka 16, 44-100 Gliwice, Poland (✉andrzej.pulka@polsl.pl, amilik@polsl.pl +48 32 237 1644*)*

Abstract

The work presented in the paper concerns a very important problem of searching for string alignments. The authors show that the problem of a genome pattern alignment could be interpreted and defined as a measuring task, where the distance between two (or more) patterns is investigated. The problem originates from modern computation biology. Hardware-based implementations have been driving out software solutions in the field recently. The complex programmable devices have become very commonly applied. The paper introduces a new, optimized approach based on the Smith-Waterman dynamic programming algorithm. The original algorithm is modified in order to simplify data-path processing and take advantage of the properties offered by FPGA devices. The results obtained with the proposed methodology allow to reduce the size of the functional block and radically speed up the processing time. This approach is very competitive compared with other related works.

Keywords: DNA-tiles, pattern recognition, pipelining, parallelism and concurrency, dynamic programming, systolic arrays, computational methods.

© 2012 Polish Academy of Sciences. All rights reserved

1. Introduction

The expansion of microbiology, molecular biology and computational biology, observed for last decades, has been delivering a huge amount of data. The National Center for Biotechnology Information (NCBI) as a part of the International Nucleotide Sequence Database Collaboration (INSDC) has established GenBank [1] as the open-access database. GenBank collects genome sequences produced in laboratories throughout the world from more than 100,000 distinct organisms and continues to grow at an exponential rate, doubling every 18 months (there are 126,551,501,141 bases in 135,440,924 sequence records in the traditional GenBank divisions as of April 2011). Today, researchers can collect information coming from DNA structures and it is very important to find effective and accurate techniques which allow identifying and recognizing this data, describing and understanding genome mechanisms and functions. The searching techniques based on software solutions have proved their correctness, but also showed their ineffectiveness when the amount of data reaches Gigabytes. So, it is necessary to look for other hardware-dedicated and effective solutions. The approach, which is the continuation of the previous researches of the authors [2, 3], described within the paper allows to reduce radically hardware resources and improve overall system performance.

2. Problem description and motivation

The limitations of serial (sequential) computers and constraints of data processing cause that only software-based approaches to genome searching are not sufficient and satisfying. We can distinguish many approaches in the field [4], but the most important – the milestone

Article history: received on June 7, 2011; accepted on Jan. 16, 2012; available online on March 12, 2012; DOI: 10.2478/v10178-012-0004-0.

which decided about the development of the DNA sequences searching – was the algorithm S-W of Smith and Waterman [5]. The S-W algorithm presents a heuristic search technique based on the idea of dynamic programming. Since the first introduction of the S-W method, some modifications of that approach have been proposed, but still the original S-W technique is the base for software implementations. Unfortunately, this algorithm, as every exhaustive search technique, has its natural limitations, so the problem of shortening the processing time is still very important and many researchers work on it. Some works based on hardware structures have been proposed [2, 6-8], but rapid development of FPGA devices allows improving the performance and speeding up the data processing [9]. The variety of different functional blocks and flexibility of their configuration makes these devices very useful in measuring applications [10, 11]. The presented approach proves that data alignment in fact is yet another field of the application of modern FPGA devices for measurement purposes.

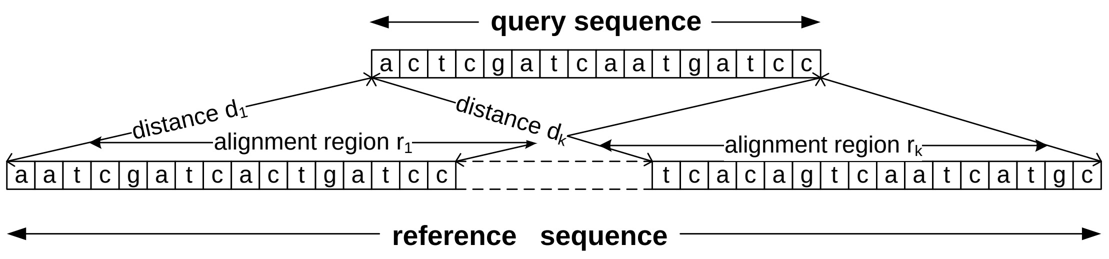

query sequence
a c t c g a t c a a t g a t c c
distance d1
distance dk
alignment region r1
alignment region rk
a a t c g a t c a c t g a t c c
t c a c a g t c a a t c a t g c
reference sequence

Fig. 1. Sequence alignment viewed as a distance measurement task.

2.1. Smith-Waterman algorithm

The original task [5] could be formulated in the following way: *find the best alignments between some query sequences within very long reference (DNA) chains*. As in [2], there are two sequences, i.e. the reference sequence and the query sequence by vectors of symbols $\boldsymbol{Ref} = \{\boldsymbol{R}_1,\ldots, \boldsymbol{R}_n\}$ and $\boldsymbol{Qry} = \{\boldsymbol{Q}_1,\ldots, \boldsymbol{Q}_m\}$ respectively. The problem of sequence alignment is in fact a problem of distance measurement between two (or even more) sequences. This distance is expressed by a special penalty function, where some modifications (insertion, deletion) between chains are allowed (Fig. 1). The Smith-Waterman algorithm [5] defines the methodology of the penalty function evaluation. This approach is based on the idea of dynamic programming [4], where the penalty values to subsequent elements of the matrix are iteratively calculated basing on the results of the alignments between elements of the reference and the query sequences (match or mismatch), i.e. :

$$P(i,j) = MIN\begin{cases} P(i-1,j-1) + \Delta(Q_i, R_j) & \textit{match / mismatch} \\ P(i,j-1) + \Delta(\_, R_i) & \textit{insertion} \\ P(i-1,j) + \Delta(Q_i, \_) & \textit{deletion} \end{cases}. \tag{1}$$

Where:

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

According to the equation (1) and the presented schemes (Fig. 2), the value of a given coefficient $P[i, j]$ depends just on values of three adjacent cells. Basing on this fact, we can arrange the calculation process of the current cell and reserve the storing resources. Only the current and the last value of calculation have to be remembered (stored) for each row of the $\boldsymbol{P}$ array. It appears (from the calculation scheme) that the entire query sequence could be calculated simultaneously, provided that all coefficients necessary for the current calculation cycle are available.

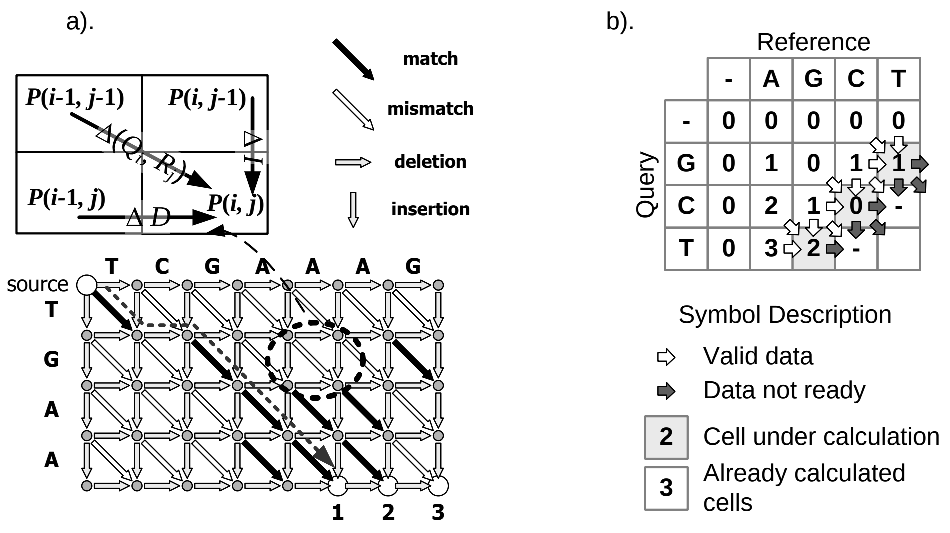


Fig. 2. Schemes of the Smith-Waterman-based optimal path search (a) and its pipeline processing (b).

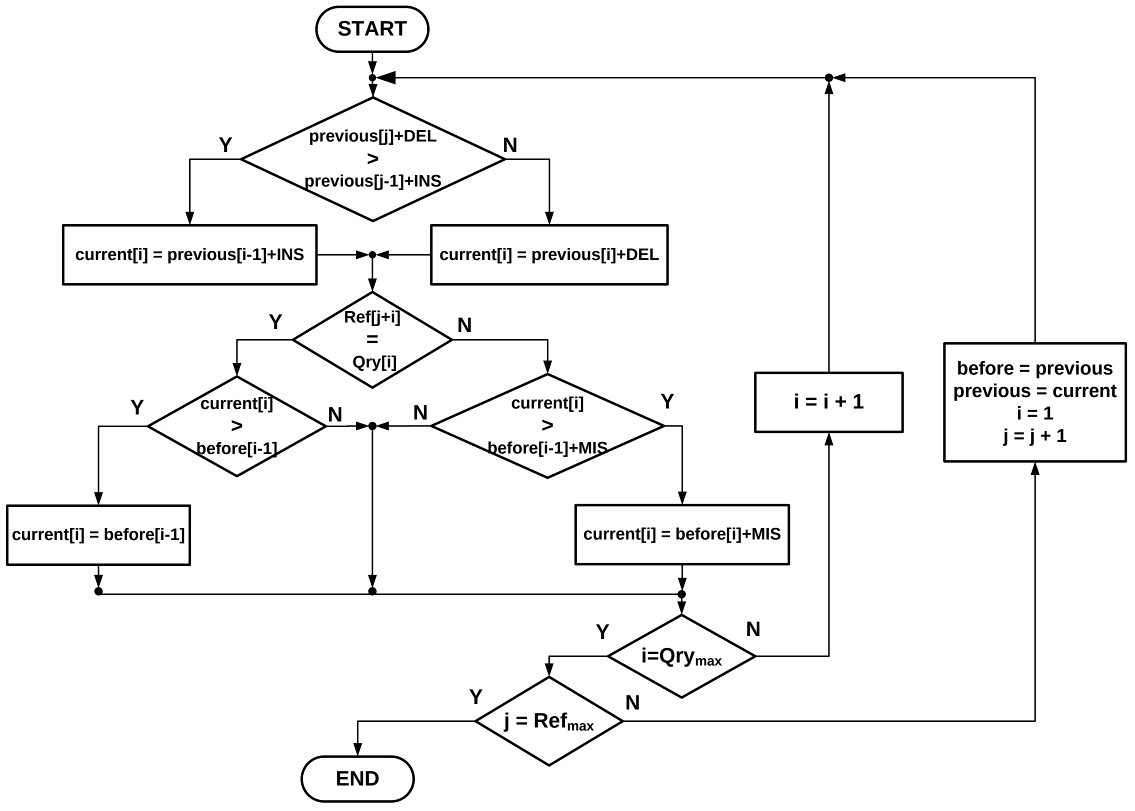


Fig. 3. The main loop of the Smith-Waterman algorithm (the evaluation of three adjacent diagonals: current, previous and before previous).

3. Direct software implementation of the algorithm

The software coded S-W algorithm could be represented by the block diagram given in Fig. 3. It brings out the main calculation loop consuming the most CPU time. To make the implementation more effective and closer to hardware, the entire algorithm was investigated with a special attention to the parallel processing, to obtain the most concurrent possible structure. To evaluate a penalty value for a given cell P[*i*, *j*] of the matrix, the scores of the adjacent cells should be given (Fig. 2). In programming languages this kind of operation usually corresponds to a loop statement, which assures obtaining a flexible code that is able to handle query sequences of different length. Flattening a loop into a parallel operation and applying the pipeline approach allow using all benefits offered by a hardware implementation. This implementation resembles the approach presented in [12], however it is more optimal. If a single cell complexity is moderate, the pipeline structure can hold the entire query sequence that can be processed at once. A single calculation cycle (usually executed in one clock cycle) is required to determine contents of penalty values for all query symbols. In fact, 3 adjacent diagonals: current (current), previous (previous) and before previous (before) are processed. Thanks to this fact the computational complexity is reduced to O(m+n-1).

Verilog HDL language [13] is similar to the C language and combines elements characteristic for hardware descriptions with typical software constructs. However, synthesis tools limit the allowed description styles and the set of synthesizable constructions. Generally, in any HDL (especially in Verilog), it is possible to express non-synthesizable components using a synthesizable set of constructions. The proposed Verilog model defines the hierarchical structure with high regularity, the basic cell that is handling a single symbol comparison is replicated. A detailed analysis of the block diagram (Fig. 3) shows that only two basic arithmetic operations: addition and comparison (greater or less then) could be used. The latter in the sense of high-level languages is considered as a logical operation. The magnitude comparison is completed by subtracting two values. The difference is neglected, and ‘a borrow’ and ‘a zero’ conditions are considered as required results. Both of these arithmetic operations are well supported within special components usually present in modern FPGA platforms. One of the decision blocks realizes the equality check operation that can be implemented as a non-iterative logical function (based on XOR operation). These considerations lead to the first directly translated hardware implementation, which is oriented to be as close to the original algorithm computation path as possible (Fig. 4). However, this solution requires the introduction of additional identification mechanisms of regions of a good alignment, i.e. the very exact fragment of the reference sequence with a low value of the penalty function (below the cut-off level) [5, 6, 8, 14]. In many dynamic programming-based approaches [4, 7, 15-18] the backtracking search is required to restore the best path. Here, in the pipeline approach, where the contents of the matrix constantly change, it is impossible. Because of that fact ‘*the best scorers’* are remembered in a special memory (FIFO registers). To localize the best alignment position, after the investigation of the entire reference chain we can use two trace back mechanisms addressed in section 5.

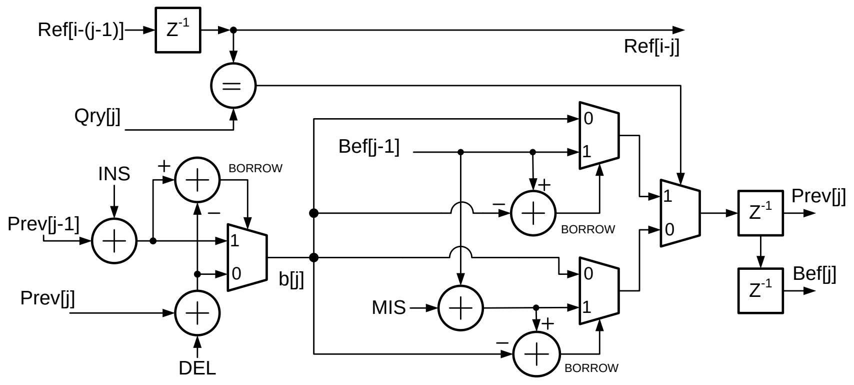


Fig. 4. Direct hardware implementation of S-W calculation units.

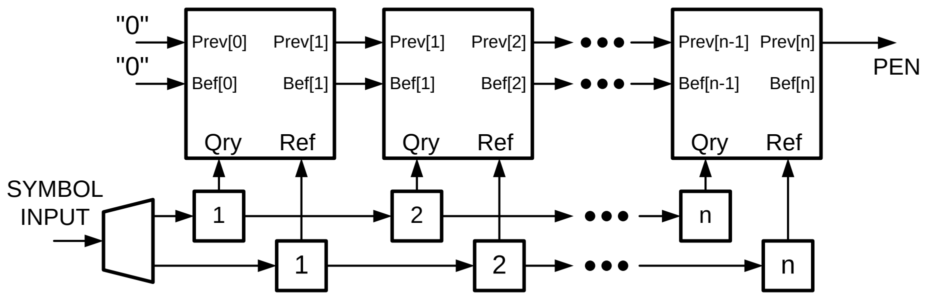


Fig. 5. Implementation of the main loop of the algorithm.

When the description of the computation path is ready, we can proceed to the indexed addressing that retrieves reference and query symbols. The overall system performance strongly depends on several factors, like the way of loop processing and data dependencies between subsequent iterations. If the impact of the second factor is strong, i.e. if the results of subsequent steps of the calculation are tightly bound, the parallel implementation (execution) is very difficult or even impossible to perform. The number of symbols in a query determines the range of the loop iterations. A current penalty value is calculated from the result obtained in two previous steps (cells with indexes *j* and *j*-1). Those variables are named *Prev* (abbreviation for a previous step) and *Bef* (abbreviation for a before previous step) on the block diagram and the query symbols are accessed by the loop index directly. The reference symbol is accessed by indexes, and each loop evaluation takes the symbol from the current point *i* toward the beginning of the sequence move up to *j* indexes *Ref*[*i*-*j*]. In other words the reference symbols are observed in the window whose length is equal to a query sequence. All the query symbols can be processed concurrently in independent processing units against respective reference symbols, so the proposed version of the dynamic programming algorithm does not depend on calculation performed in a current loop run.

The proposed architecture of the entire system exposes regular geometry. The main loop of the algorithm (Fig. 5) can be carried out as a pipeline consisting of computational blocks together with appropriate registers. The set of registers in each stage is responsible for: storage of the last and the last but one calculation result (*Prev* and *Bef* variables), remembering query symbols that are supposed to be initialized before the procedure starts, the reference symbols shift register that is fed from the data source. The length of a query sequence can be easily modified by combining the appropriate number of slices.

The above considerations do not take into account a numerical calculation problem. Unlike general purpose computers with processing words of constant length, programmable logic devices offer a flexible length of arguments. So, one of the most important properties of the hardware implementation is that it allows precise tailoring hardware resources to given data processing algorithm requirements. The determination of the required length of variables should be preceded by the detailed worst-case analysis of data processing units. After this analysis we have found that for our algorithm the maximal possible value of the penalty score can be evaluated from the following expression:

$$p[LEN]_{MAX} = (LEN + 1)\cdot \min(Ins, Del). \tag{3}$$

Where: *LEN* – length of query sequence; $p[LEN]_{MAX}$ – maximal value of penalty at *LEN* position; *Ins* – penalty for symbol insertion and *Del* – penalty for symbol deletion.

The maximal length of registers can be estimated also from the above expression (3), but it should be mentioned that a register's length changes and it depends on the position in the iterative chain of the algorithm. Partial results are accumulated from one to another stage while data is processed in the pipeline structure. For relatively short sequences it is more convenient to use a constant length of variables. On the other hand, the maximal clock frequency of the circuit operation is determined by the longest combinatorial path.

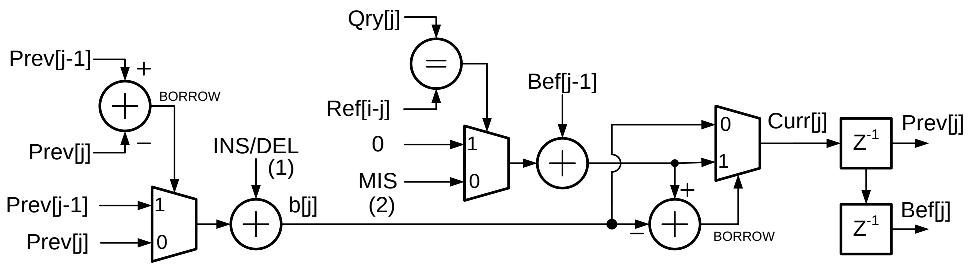


Fig. 6. Optimized Smith-Waterman algorithm cell implementation.

Fig. 6 shows a flexible implementation of the basic cell with parameterized width of the data path and registers as well as insertion, deletion and mismatch values. The proposed implementation does not contain full comparators, which in fact are not necessary for the inequality checks. Realization of a typical comparator is not optimal in terms of resources and propagation delay. Instead, the subtractors with borrow outputs are considered as the results of the comparisons have been applied. The presented approach improves circuit fitting and uses enhanced properties of FPGA devices. Such an optimized model of the circuit was extensively tested and verified against the C program prototype.

4. ΔSW – further optimization of the initial algorithm

Further considerations on the original software S–W algorithm [5] and additional assumptions allow optimizing the hardware structure and reducing the size of the circuitry. These modifications cover such problems as: integer coded and positive values of penalties, modular architecture with regular basic building blocks (processing elements) etc. Detailed discussion of these issues has been presented partially in [2] and fully in [3]. The direct implementation of the S–W algorithm very quickly has reached the limits – let us say the saturation level. Then, the numerical algorithm has been investigated and main efforts have been put into replacement of complex computational units with elementary digital elements and reduction of the sequential logic, i.e. increasing the percentage of the asynchronous processing elements. Lipton and Lopresti [15] proposed the S-W algorithm with smallest values of penalties, namely: $\Delta(Q_i, R_j) = 2$ (mismatch) and $\Delta(\_, R_j) = \Delta(Q_i,\_) = 1$ (insertion/deletion) that produces a matrix with some interesting properties summarized in theorems (see below).

4.1. Mathematical background of S-W algorithm optimization

Theorem 1 (Property 1)

Assuming that penalties meet the above (smallest integer) values and the initial rows and columns of the matrix are filled with:

$$\underset{0<i\le Q_{length}}{\forall} P[i,0]=i \quad \text{and} \quad \underset{0<j\le R_{length}}{\forall} P[0,j]=0. \tag{4}$$

The elements of array ***P*** exhibit the following properties:

$$\underset{\substack{0<i\le Q_{length}\\ 0<j\le R_{length}}}{\forall}\quad \begin{cases} P\left[i-1,j-1\right]\le P\left[i,j\right]\le P\left[i-1,j-1\right]+2\\ P\left[i,j-1\right]-1\le P\left[i,j\right]\le P\left[i,j-1\right]+1\\ P\left[i-1,j\right]-1\le P\left[i,j\right]\le P\left[i-1,j\right]+1 \end{cases} . \qquad (5a, b, c)$$

The proof by contradiction is elementary and will be skipped here.
The following conclusions are direct consequences of Theorem 1.

Theorem 2 (Conclusion)

For the matrices described in Theorem 1 the following propositions are true:

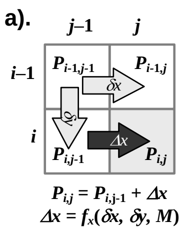
$$\underset{\substack{0\le i\le Q_{length}\\ 0\le j\le R_{length}}}{\forall}\quad \begin{array}{ll} \mathrm{P1:} & P[i,j]=P[i,j-1]+\Delta x\\ \mathrm{P2:} & P[i,j]=P[i-1,j]+\Delta y\\ \text{where:} & \Delta x\in\{-1,0,+1\};\quad \Delta y\in\{-1,0,+1\}; \end{array} \qquad (6)$$

Theorem 1 presents tight dependencies between the numerical values of ***P*** array and the correlation between query and reference patterns. Theorem 2 formulates conclusions that spring directly from Theorem 1: the differences between adjacent cells of the matrix in the horizontal and/or vertical direction do not exceed value ±1. This observation is very important and brings to mind that it is possible to use a very simple incrementing/decrementing block to implement the system functionality. Moreover, for these penalties we found that values of the matrix P elements have an upper bound, which corresponds to the row location:

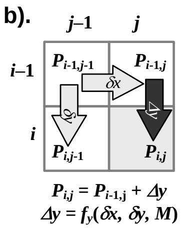
Theorem 3 (Conclusion)

$$\underset{\substack{0\le i\le Q_{length}\\ 0\le j\le R_{length}}}{\forall}\quad 0\le P[i,j]\le i\,. \qquad (7)$$

a).



b).



c).

	δ_x					
δ_y	–1	0	+1	–1	0	+1
–1	+1	+1	+1	+1	+1	+1
0	0	0	0	0	+1	+1
+1	–1	–1	–1	–1	0	+1
Δ_x	*M* = 0 (*Match*)			*M* = 1 (*Mismatch*)		

d).

	δ_x					
δ_y	–1	0	+1	–1	0	+1
–1	+1	0	–1	+1	0	–1
0	+1	0	–1	+1	+1	0
+1	+1	0	–1	+1	+1	+1
Δ_y	*M* = 0 (*Match*)			*M* = 1 (*Mismatch*)		

Fig. 7. Diagrams of the growth trends (a, b) in the S-W matrix and corresponding function tables (c, d).

4.2. Practical consequences

Basing on these theorems and observations, we can formulate the function for the next value of the matrix $P[i,j]$ expressing the relationship between the row/column increase and the reference (current) element of the matrix $P[i–1, j–1]$. Diagrams presented in Fig. 7a-b show the variables (arguments) of two functions: Δx and Δy representing the increase of the cell value that can be used alternatively for calculation of the next cell ($P[i,j]$) of the matrix ***P***. Variables δx and δy denote the growth trends in the horizontal and vertical direction, respectively. The third argument, which has an impact on the value of element $P[i,j]$ is the result of comparison between the current query symbol $Q[i]$ and the reference symbol $R[j]$. Fig. 7c-d show the tables describing functions for Δx and Δy respectively. Both functions depend on three arguments: δx; δy and M (match/mismatch). As a matter of fact, the function tables are truth tables, because the symbols used within the tables have real logical meaning, i.e. they denote appropriate combinatorial functions. So, the symbols "+1", "– 1" and "0" correspond to the increment (*INC*), the decrement (*DEC*) and no operation (*NOP*), respectively.

Functions Δx and Δy are symmetrical with respect to the variables δx and δy; i.e.:

$$\begin{aligned} \Delta x &= f_x(\delta x, \delta y, M) = f_y(\delta y, \delta x, M) \\ \Delta y &= f_y(\delta x, \delta y, M) = f_x(\delta y, \delta x, M) \end{aligned} \tag{8}$$

Each half of one table (for M=0 and M=1 respectively) is the transpose of the appropriate half of the next table. This means that combinatorial functions describing growth trends in both directions are identical. The reverse transformation from growth coefficients to current penalty value can be obtained by summation of growth factors following a selected path:

$$\boldsymbol{P}[\boldsymbol{i},\boldsymbol{j}] = \boldsymbol{P}[\boldsymbol{i},0] + \sum_{\boldsymbol{k}=1}^{\boldsymbol{j}} \boldsymbol{\delta x}[\boldsymbol{i},\boldsymbol{k}] = \boldsymbol{P}[0,\boldsymbol{j}] + \sum_{\boldsymbol{l}=1}^{\boldsymbol{i}} \boldsymbol{\delta y}[\boldsymbol{l},\boldsymbol{j}]; \quad \boldsymbol{P}[\boldsymbol{i},0] = \boldsymbol{i}; \quad \boldsymbol{P}[0,\boldsymbol{j}] = 0. \tag{9}$$

The initial value for P[i,0] is determined by (7) (Theorem 3) under the assumption that the entire chain of reference symbols processed does not match the query (initial stable state).

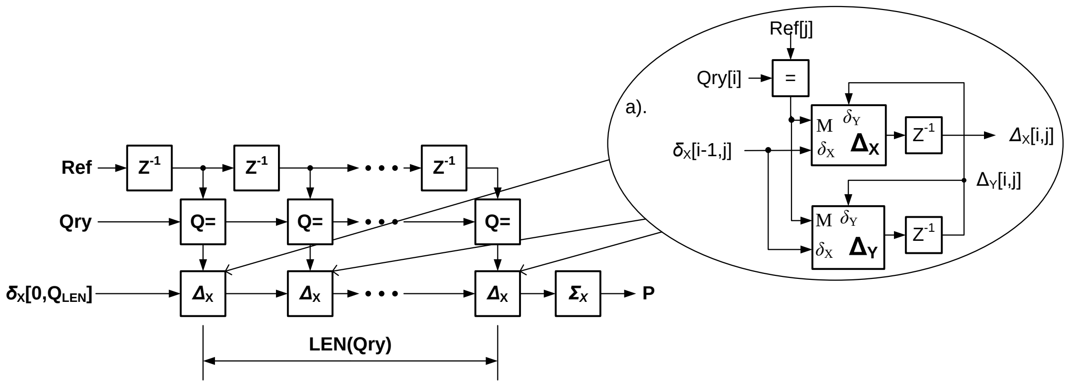


Fig. 8. Block diagram of the pipelined structure of growth function Δx with basic unit structure (a).

4.3. Implementation results

As it was mentioned in the previous sub-section, we can consider two alternative implementations with the same complexity: in the horizontal or vertical direction of the matrix ***P***, respectively. However, in the case of application to chain alignments, the more convenient implementation is the one located on the axis parallel to the reference sequence. The presented circuit executes the Δx function (the reference pattern is placed horizontally).

The block diagram of a single unit (basic building item) of the horizontal growth function Δx is depicted in Fig. 8a. The length of the query sequence determines the number of basic cells that should be connected together. The high simplicity of the cell allows implementing extremely long query sequences. The final value of P is calculated in the Σ unit. The implementation could be considered as a reversible synchronous counter. This structure is able to process a single input symbol per clock cycle in pipelined fashion.

Convergence of the algorithm allows eliminating complicated dataflow control for pipeline processing. The short combinatorial path (the depth does not exceed two Look-Up function generators) guarantees a very short propagation time. The architecture resource requirements depend on LUT size (the FPGA target platform). We have made several experiments with the Xilinx devices family [9]. Some devices (from Virtex to Virtex-4 with Spartan derivatives) are based on 4-input LUTs, and some (Virtex-5 and newer) on 6-input LUTs. The implementation benefits from the possibility to convert LUTs to shift registers (SRL) that are utilized as programmable comparators. In general, the implementation of the basic cell requires: 2 slices, 8 flip-flops, one shift register and four 6-input LUTs or six 4-input LUTs.

5. Trace back – final alignment localization

5.1. First approach – HIPAS

As it has been mentioned above, the original Smith-Waterman approach requires a backtracking search to reconstruct the optimal path (Fig. 1). Although there are some hardware techniques that propose remembering the entire path for the trace back, but such solutions seem to be rather expensive and requiring many additional ineffectively used resources. Our solution is different: we have to remember only few 'candidates' and their scores in a special memory, and analyze solutions after the entire genome search. It allows saving resources and only those regions are explored where the probability of the matching occurrence is the highest. Our **HIPAS** (**H**euristic **I**dentification for **P**atterns **A**lignment of **S**equences) could be defined by the following three phases:

```
Step 1: Initialization Phase: load the register R1 with 'found candidate' sequence; load the query register QR
        with a given query sequence and reset the register R2.
Step 2: Branch Phase: determine one of the cases (modes) of calculations.
Step 3: Execution phase:
        case 3a: if score = 0 (trivial case) then R1 → R2 and terminate.
        case 3b: if score = 1 (semi-trivial case) compare registers R1 QR and when you meet the first
         difference on the k-th position load QR as follows:
                for i < k   R1(i) → QR(i);
                for i > k   R1(i) → QR(i+1);
                QR(k) → "gap symbol"  (_).
        terminate.
        case 3c: if score > 1 (heuristic trial and error method) compare registers R1 and QR moving towards
         the beginning of the sequence and when you meet a difference try to replace it with a gap (QR(i) →
         "gap symbol) or a mismatch (R1(i) → QR(i)). Remember that two gaps one by one are not allowed and
         the penalty of mismatch is given 2 points. Each step updates the current score. When you reach the end
         of the sequence and the current score counter is equal zero,  terminate.
```

Theoretically, the algorithm requires up to $3n$ (where: n is the number of query symbols) clock cycles per a single chain check (the worst case) and an additional memory, which is usually available on the FPGA board.

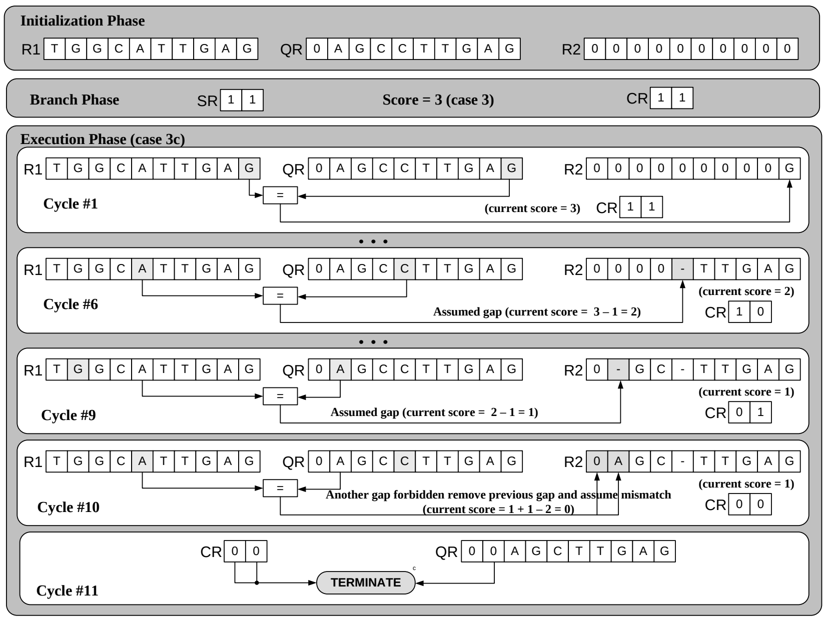
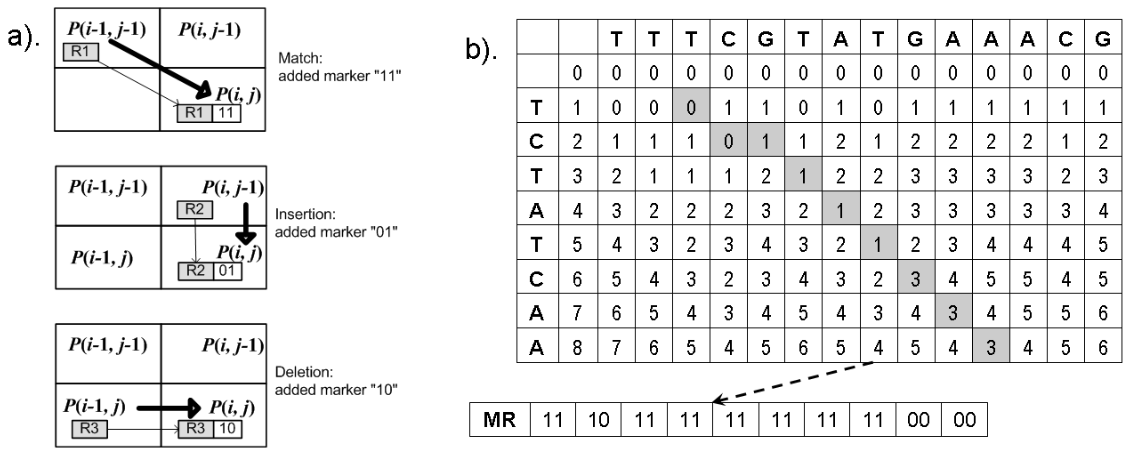


Fig. 9. An example of the HIPAS algorithm run.

5.2. Second proposal – idea of marker registers

The second more efficient mechanism uses special registers responsible for registration of the “moves” during the matrix evaluation process. This idea comes from the systolic array structure and requires additional resources attached to every cell of the structure. During the pipeline calculation process (Fig. 2) 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. 2). Each marker is coded on two bits, and the combination “00” is unused (“no move”). This mechanism, for a given cell $P[i, j]$ is illustrated in Fig. 10a. In other words, when a given pipeline stream reaches the end (the final comparison of symbols is done), the location of the best alignment is already available (Fig. 10b).



		T	T	T	C	G	T	A	T	G	A	A	A	C	G
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	1	0	0	0	1	1	0	1	0	1	1	1	1	1	1
C	2	1	1	1	0	1	1	2	1	2	2	2	2	1	2
T	3	2	1	1	1	2	1	2	2	3	3	3	3	2	3
A	4	3	2	2	2	3	2	1	2	3	3	3	3	3	4
T	5	4	3	2	3	4	3	2	1	2	3	4	4	4	5
C	6	5	4	3	2	3	4	3	2	3	4	5	5	4	5
A	7	6	5	4	3	4	5	4	3	4	3	4	5	5	6
A	8	7	6	5	4	5	6	5	4	5	4	3	4	5	6

MR	11	10	11	11	11	11	11	11	00	00

Fig. 10. The idea of marker registers (a) and the trace-back reconstructing the entire path (shaded cells).

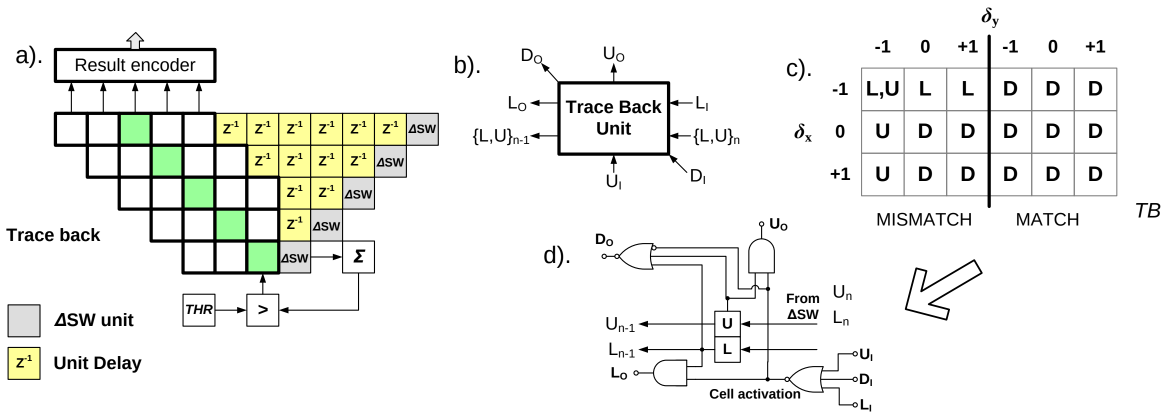


Fig. 11. The concept block diagram of the tracing back through the entire matrix (a), the trace-back cell (b), the symbolic truth table (c) and schematic diagram of the trace-back unit cell (d).

5.3. Trace back and finding the best alignments

The proposed incremental S–W (ΔSW algorithm) implementation also facilitates the trace-back procedure. Fig. 11a shows the concept block diagram of the tracing back procedure through the entire unit. Finding the optimal sequence alignment does not require storing numerical values but rather remembering data necessary for restoring the path. As we mentioned above, only two bits are needed for this purpose. The direction of the trace back can be determined basing on the numerical values in the matrix. There are three possible return paths (Fig. 11b) marked with symbols: U, L and D for “up”, “left” and “diagonal”, respectively. The procedure is initiated by the comparator that detects the score falling under a given penalty threshold (THR block). The output of the comparator activates the trace back combinatorial chain. The first activated cell is located on the diagonal. This means that the preceding sequence got an exact match. The iterative circuit cell does not need to be aware of its location in the chain. The activation from the first cell is passed to other cells according to the determined trace-back path. In order to adjust the {L,U} argument pair, each cell is equipped with registers that holds the found L and U values for their location that responds to the appropriate symbol of the reference pattern. The matched sequence potentially can contain insertion and deletion. The number of cells that are located around the diagonal in the up and left directions restricts the number of insertions and/or deletions of symbols in the sequence, respectively. Finally the active cells are observed at the top row of the unit. The signal from an active cell is encoded to determine its location that corresponds to the beginning of the matched sequence. The basic cell concept of an iterative trace-back unit is shown in Fig. 11b. The cell is fed by the activation inputs that come from three neighboring cells marked as L_I, D_I, U_I. If the cell is activated from any input it responds with an activation signal to successor cells according to the determined trace-back path. The growth trend can be mapped into return path determination (the symbolic truth table for return path in Fig. 11c). Clearly, the table shows that U and L are active only for mismatches between symbols corresponding to the reference and the query sequences (column and row coordinates). The signal U is active only if the vertical growth trend (δy) is negative (-1), while L is asserted for negative horizontal growth trend (δx). If there is no negative dominance of δx and δy, the return path goes through D. The code assignment for symbols L, U and D requires distinguishing only 4 different situations while D depends on L and U. L and U are mutually independent. Reducing the amount of information for the return path is crucial while those data must be stored and adjusted in time to properly feed the information to trace back the systolic combinatorial unit (Fig. 11d).

6. Final considerations and conclusions

A new formula of the ΔSW algorithm has been developed that 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 the usefulness of carried-out research work. Usually, when the resource consumption of a given general-purpose application mapped in a FPGA structure grows, the performance of the circuit rapidly goes down. It is mainly because of the propagation delays introduced by long-chained connections between multiple segments. Such connections traverse many routing matrices and automated "*place & route*" procedures have problems with routing signals. However, the application described in the paper is very regular and repeatable, and the entire structure is very compact. It allows packing search queries consisting of 32400 symbols. The basic blocks of ΔSW and TB units consist of neighboring cells, which are close to one another and utilize short direct connections. So, the presented methodology and the developed synthesizable model, which considers the properties of FPGA structures, enable reducing the resource utilization, simplify data and control paths and increase the system throughput even with 90% resources utilization of the XC5VLX50T board. The basic cell performance is about 220 MHz/4.5 ns (Frequency/clock cycle) for Virtex/Spartan-2 and about 600 MHz/1.6 ns for Virtex-5.

Table 1. Resource requirements for full Smith-Waterman implementation expressed in LUT6.

Parameters		**Components complexity / LUTs distribution**			**Total nr of LUTs**
Query length (nr of symbols)	**Max. Dist. (nr of symbols)**	**ΔSW (nr of LUTs)**	**TB (nr of LUTs)**	**SRL (nr of 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**

The unit consists of three basic items: the **ΔSW** pipe, **SRL** – 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. Table 1 gathers the required resources divided into functional groups determined for Virtex 5 VLX families, which allows implementing the accelerator computing board thanks to PCIe embedded endpoints. As a reference, the LUT count is taken XC5VLX330T with the array 240x108 CLBs. The second column denoted by symbol "**Max. Dist.**" contains the maximal admitted region of fitting for the sequence.

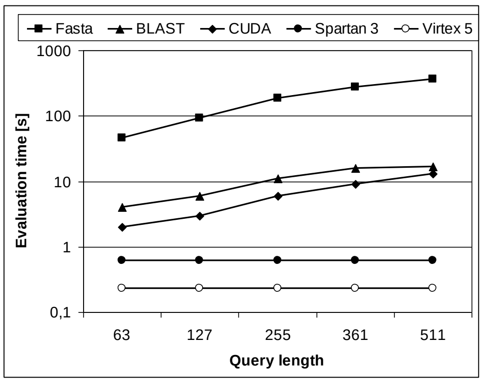


Fig. 12. Comparison of the computation efficiency of different methodologies.

Actually, thanks to the regularity of the entire structure, we are able to evaluate almost 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-Itanium-2 applications reported in [12], the throughput of our system, expressed in GCUPS (giga cell-updates per second) is about 4000 times faster, i.e. taking into account the operating frequency 600 MHz and the query length consisting of 32400 symbols we can obtain the speed 200 GCUPS (versus 0.049 in multi-core implementation). Authors of [12] improved the Smith-Waterman algorithm and they managed to reach the performance level of 178.65 GCUPS, so the methodology proposed here is comparable (even a little better). As to computation efficiency the entire reference chain (genome) can be analyzed in less than 8 seconds. Fig. 12 compares the results obtained for two implementations for Spartan3 and Virtex5 with results reported in the literature for FASTA [19], BLAST [20] methodologies and applications on parallel graphical processors [21, 22].

Acknowledgements

Dr Andrzej Pułka would like to express many thanks to Prof. Yun S. Song from the University of California Berkeley for the opportunity to join the group working on a project on novel applications for reconfigurable computing in computational biology. These experiences inspired mostly the presented contribution.

References

[1] GenBank (2010). http://www.ncbi.nlm.nih.gov/.

[2] Pułka, A., Milik, A. (2008). A New Hardware Algorithm for Searching Genome Patterns. *Proceedings of IEEE ICSES 2008*, Kraków, Poland, 177-180.

[3] Milik, A., Pułka, A. (2011). On Efficient Implementation of Search for Genome Patterns. *PAK*, 57(1), 15-18. (in Polish)

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

[5] Smith, T.F., Waterman, M.S. (1981). Identification of Common Molecular Sub-sequences. *Journal of Molecular Biology*, 147, 195-197.

[6] Yamaguchi, Y., Maruyama, T. (2002). High Speed Homology Search with FPGAs. *Proceedings of Pacific Symposium on Biocomputing*, 271-282.

[7] Zhang, F., Qiao, X., Liu, Z. (2002). A Parallel Smith-Waterman Algorithm Based on Divide and Conquer. *Proceedings of IEEE ICA3PP'02*, 162-169.

[8] Benkrid, K., Liu, Y., Benkrid, A. (2007). High Performance Biosequence Database Scanning Using FPGAs. *Proceedings of IEEE ICASSP*, Honolulu, Hawaii, USA, 361-364.

[9] Xilinx, The official Web site of the Xilinx Company, http://www.xilinx.com/.

[10] Zieliński, M. (2009). Review of Single-Stage Time-Interval Measurement Modules Implemented in FPGA Devices. *Metrology and Measurement Systems*, 16(4), 641-648.

[11] Zhang, Ming, Li, Kaicheng, Hu, Yisheng. (2010). DSP-FPGA Based Real-Time Power Quality Disturbances Classifier. *Metrology and Measurement Systems*, 17(2), 205-216.

[12] Buyukkurt, B., Najjar, W.A. (2008). Compiler generated systolic arrays for wavefront algorithm acceleration on FPGAs. *Proceedings of IEEE ICFPLA 2008*, 655-658.

[13] Ashenden, P.J. (2008). Digital Design – An Embedded Systems Approach Using VERILOG. *Morgan Kaufman Publishers*.

[14] Oliver, T., Schmidt, B. (2004). High Performance Biosequence Database Scanning on Reconfigurable Platforms. *Proceedings of IPDPS*, Santa Fe, New Mexico, USA, 192-199.

[15] Lipton, R., Lopresti, D. (1985). A systolic array for rapid string comparison. *Chapel Hill Conference on VLSI*, 363-376.

[16] Hoang. D.T. (1992). A Systolic Array for the Sequence Alignment Problem. *Brown University, Providence, RI, Technical Report CS-92-22*.

[17] Li, T., Shum, W., Truong, K. (2007). 160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA), *BMC Bioinformatics* 2007, 8, I85.

[18] Hai Song Xu, Wen Ke Ren, Xiao Hui Liu, Xiao Qin Li (2008). Improving Sequence Alignment using Class-Specific Score Matrices, *Bioinformatics and Biomedical Engineering, 2008. ICBBE 2008. The 2nd International Conference on*, 70-73.

[19] FASTA (2011). Sequence comparison at the University of Virginia. *http://fasta.bioch.virginia.edu/*.

[20] BLAST (2011). Basic Local Alignment Search Tool. *http://blast.ncbi.nlm.nih.gov/Blast.cgi*.

[21] Manavski, S.,A., Valle, G. (2008). CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. *BMC Bioinformatics* 2008, 9(Sup 2):S10.

[22] Liu, Y., Maskell, D.L., Schmidt, B. (2009). CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. *BMC Research Notes* 2009, 2:73.