Hardware Acceleration of Approximate Tandem Repeat Detection

Tomáš Martínek
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, 612 66, Czech Republic
Email: martinto@fit.vutbr.cz

Matej Lexa
Faculty of Informatics
Masaryk University
Botanická 68a, Brno, 602 00, Czech Republic
Email: lexa@fi.muni.cz

***Abstract*—Understanding the structure and function of DNA sequences represents an important area of research in modern biology. Unfortunately, analysis of such data is often complicated by the presence of mutations introduced by evolutionary processes. At the lowest scale, these usually occur in biological sequences as character substitutions, insertions or deletions (indel). They increase the time-complexity of algorithms for sequence analysis by introducing an element of uncertainty, complicating their practical usage. One class of such algorithms has been designed to search for tandem repeats with possible errors—*approximate tandem repeats*. This paper investigates the possibilities for hardware acceleration of approximate tandem repeat searching and describes a parametrized architecture suitable for chips with FPGA technology. The proposed architecture is able to detect tandems with both types of errors (mismatches and indels) and does not limit the length of detected tandem. A prototype of the circuit was implemented in VHDL language and synthesized for Virtex5 technology. Application on test sequences shows that the circuit is able to speed up tandem searching in orders of thousands in comparison with the best-known software method relying on suffix arrays.**

***Index Terms*—Approximate tandem repeat; dynamic programming; systolic array; FPGA; DNA;**

I. Introduction

A *perfect* tandem repeat is defined as a string with two identical substrings following each other, e.g. $abcdabcd$. Approximate tandem repeats are allowed to contain a limited number of differences between individual substrings, e.g. $abcdabbd$. The biological and practical aspects of both types of tandem repeats in genomic DNA of numerous organisms have been the object of scientific studies for some time now. For example, the human genome consists of two sets of chromosomes, each set containing about 3 billion nucleotide basepairs that can be represented by a linear sequence of 4 symbols (A,C,G and T). About 3 percent of the nucleotide sequence is made of protein-coding sequences and associated regulatory elements (genes). The rest used to be called junk DNA until research has shown numerous biological functions residing in these areas of the sequence, which is more appropriately referred to as intergenic DNA. Much of the intergenic regions are made of repetitive sequences. These can be dispersed repeats, where the same sequence appears at seemingly random locations of the genome, or tandem repeats, where a particular sequence is repeated in tandem, a copy follows a previous copy immediately.

Tandem repeats in human and other genomes have been classified into classes based on the average size of the repeated unit as microsatellites and minisatellites, now usually referred to as Short Tandem Repeats (STR, up to a dozen basepairs) and Variable-Number Tandem Repeats (VNTR, hundreds and thousands of basepairs) [1]. Because these repeats undergo mutations more often than protein-coding genes, and because they often differ in the number of repeats between individuals, they became a tool for typing genomes, using a number of experimental techniques to detect their periodicity, length or simply the presence of a particular repeated sequence. Tandem repeats often have specific biological functions, such as involvement in gene evolution, or creating special structures in DNA and proteins. Although difficult to sequence, many tandem repeats are known today, either as isolated sequences or as subsequences of available genomes.

We have used a database of tandem repeats associated with the UCSC Human Genome Browser [2] as well as real and artificial DNA sequences in this study to provide a spectrum of repeat-containing sequences (from real repeat-rich sequences to random sequences with very few tandem repeats).

Searching for perfect tandems is a well-studied problem. The best algorithm in this area [3] achieves linear time complexity $O(n)$, where n is the length of the input string. However, biological sequences often contain imperfect tandem repeats derived from perfect repeats by mutations. These mutations typically come as nucleotide (symbol) changes (mismatches), deletions or insertions (indels). Algorithms for tandem repeat searches in biological sequences are therefore modified to identify *approximate tandem repeats*. The metric used to evaluate similarities between substrings of tandem repeats is commonly some kind of edit distance. The simplest approaches use Hamming distance (only mismatches are considered). Levenshtein [4] distance is a much better metric for this purpose. It is defined as the minimal number of insertions, deletions and character changes needed to convert one string into another.

Because of the need to account for mismatches and indels, algorithms for approximate tandem repeat searches have higher complexity than their perfect repeat counterparts. The best direct algorithms in this area require $n.k.log(k).log(n/k)$ steps to find tandem repeats with at most k errors, or as much as $n^2.log(n)$ steps to identify approximate tandem repeats without limits to the number of errors. This level of time complexity can become prohibitive with large datasets or in interactive searches often used by biologists when exploring genomic DNA sequences. The goal of this work is to study the details of available methods for tandem repeat detection and the possibility of their hardware acceleration.

This paper is organized as follows: Section II describes state-of-the-art software methods for approximate tandem repeat searching. Related work in this area is summarized in section III. Section IV contains detailed description of our hardware architecture for acceleration of approximate tandem repeat searching. Evaluation of proposed architecture and its comparison with software implementation is given in section V. Conclusions are summarized in section VI.

II. Algorithms for Approximate Tandem Detection

Algorithms for approximate tandem repeat detection can be divided into distinct categories. In the first category, we place algorithms that do not limit the search to a certain number or to a certain proportion of errors in the input string [5], [6], [7]. As a rule, these algorithms use a dynamic programming matrix to achieve their goal. The best algorithm in this group [7] has time complexity of $n^2.log(n)$ for sequences of length n. It is practical for sequences of up to several thousand bases. The second group of algorithms represents a variation on the theme to search for tandem repeats with no more than k errors [8], [9]. The most influential work in this area [8] can find all applicable approximate tandem repeats in time $n.k.log(k).log(n/k)$, making it applicable to very long sequences. Still, when applied to entire genomes on a single processor machine, computation can take weeks or months, depending on the value of k and the length of the analyzed sequence.

The last group comprises algorithms using heuristics to simplify searches in large databases of biological sequences [10], [11], [12], [13], [14], [15]. Typical examples are: *Repeat Masker*, *Tandem Repeat Finder* [10] and *mreps* [11]. These approaches are typically associated with lower sensitivity or selectivity, they limit searches to sequences of certain lengths or find only tandems with specific types of errors (mismatches or indels) [13] or specific motifs [12].

In this work we concentrate on acceleration of the best direct technique for detection of approximate tandems with k-errors [8]. It can be used alone or in combination with appropriate heuristic methods. Details of the cited approach follow.

A. Tandem detection with k-errors

The basic principle is based on the algorithm [16] for detection of perfect tandem repeats. The input string S is seen as a concatenation of two substrings u and v, where $S = uv$, $u = s_1 \dots s_{n/2}$, $v = s_{n/2+1} \dots s_n$ (for the sake of simplicity let us consider strings with lengths equal to powers of 2). The method is designed to identify all tandem repeats that span both substrings u and v (in other words they contain the middle of S). The same calculations are applied recursively to substrings $S_1 = s_1 \dots s_{n/2}$ a $S_2 = s_{n/2+1} \dots s_n$. The recursion stops when the input string is of length 1.

Depending on whether the center of the tandem repeat lies within u or v, the tandem repeats are *left* or *right*. The procedure to find all right tandem repeats works as follows:

For all possible periods $p = 1$ to $n/2$ do:

1) $j = n/2 + p$.
2) *Forward Direction*: Find the longest prefix of $s_j \dots s_n$, that matches a prefix of $s_{n/2} \dots s_n$. Let the length of the prefix be l_1 (see figure 1).
3) *Reverse Direction*: Find the longest suffix of $s_{n/2} \dots s_{j-1}$, that matches a suffix of $s_1 \dots s_{n/2-1}$. Let the length of the suffix. be l_2.
4) If $l_1 + l_2 \geq p$ then there is at least one tandem repeat of length $2p$.

Time complexity of this basic algorithm is derived as follows: Using a suffix tree or suffix array [17], the calculation of prefix l_1 and suffix l_2 can be carried out in constant time $O(1)$. Within one iteration $n/2$ positions have to be traversed. The calculation is carried out for both left and right tandem repeats recursively in $log(n)$ steps. The overall time complexity is therefore $n.log(n)$.

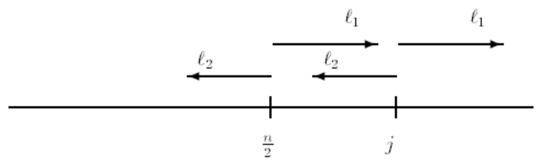

Fig. 1. Tandem repeat detection using longest common prefix l_1 and longest common suffix l_2

Landau et al. [8] elegantly extended the algorithm mentioned above to searching for tandem repeats with k mismatches. Let us assume that prefixes l_1 and suffixes l_2 contain up to k errors. Upon obtaining the prefix l_1, we find the longest common prefix k times, always from the end of the previous prefix incremented by one (see figure 2). The situation is similar for the suffix l_2. The positions of individual errors are stored in an auxiliary array. In the last step the set of positions is scanned for a subset that fulfills the conditions for a tandem repeat $l_1[h] + l_2[h'] \geq p$ and $h + h' \leq k$. Time complexity becomes $nk.log(n/k)$.

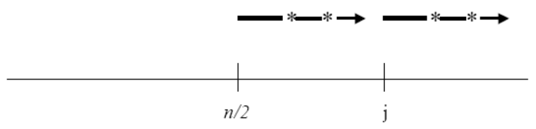

Fig. 2. Longest common prefix l_1 composed of k mismatches

Upon expanding the algorithm to indels it becomes necessary to calculate the longest common prefixes l_1 and suffixes l_2 by means of dynamic programming [18]. At each position of the tandem repeat global alignment has to be calculated in direct and reverse sense (see figure 3). However, since we only need alignments with no more than k errors, the matrix only needs to be calculated/filled up to $2k + 1$ diagonals.

Similar to the previous algorithm, we need to identify the positions $l_1[h]$ a $l_2[h']$ of common prefixes and suffixes with up to k errors in the global alignment matrix. Each diagonal of the matrix defines a different position, which together form a *wave*. The desired position can be calculated as the maximum of this wave. In the last step, we examine the array of identified positions and select a subset of positions that fulfill the tandem requirements $l_1[h] + l_2[h'] \geq p$ and $h + h' \leq k$.

The algorithm can be considerably improved by using the principle of calculating global alignment published in [19]. This approach uses only k^2 steps and its overall complexity is $nk^2.log(n/k)$.

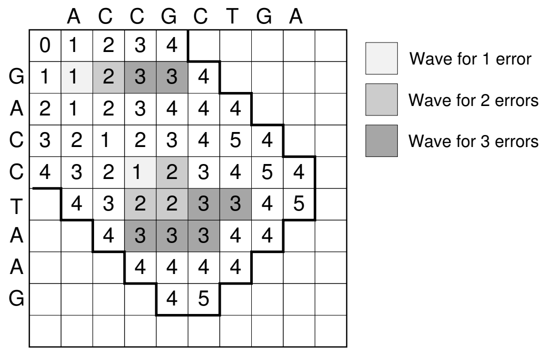

Fig. 3. Forward direction: An example of global aligment of two strings with at most 3 erros. Wawes for 1,2 and 3 errors are shown using dirrerent shade.

Landau et al. [8] improved the algorithm further. They found a procedure that uses neighboring global alignments to calculate a given tandem repeat global alignment - a so called *incremental calculation* [20]. To obtain a new global alignment, they need only k steps instead of k^2 steps. The maximum of the wave can be obtained in $log(k)$ steps, leading to an overall time complexity of $nk.log(k).log(n/k)$. This relatively complex approach has not been applied in practice so far and as such has more of a theoretical value.

Please note, that the proposed algorithm detects all tandem repeats in the input sequence. However, some occurrences can be interpreted in several ways, e.g. substring *abcabcabcabc* represents two repeats of length 6 as well as four repeats of length 3. Detailed analysis of such cases is usually performed separately, outside the main algorithm.

III. Related Works

One of the most important works in the area of hardware acceleration of approximate tandem repeat searching is [21]. The authors proposed a circuit architecture for detection of tandems with k-mismatches with an extension to detect one indel. The basic schema of the circuit (see figure 4b) is made of an array of comparator units. Each comparator unit (see figure 4a) in turn contains a pair of registers, a pair of comparators and a counter of detected errors. The input string is shifted into the array from left to right and then back again. The upper set of registers contains a substring of the input string named α_1 and the lower set of registers contains the substring α_2. The pair of comparators compares the input and output characters in α_1 and α_2. Depending on the state of the comparators, the unit adjusts the counter of mismatches as appropriate. Each comparator unit evaluates one tandem repeat length. Please, note that the length of the array limits the possible length of detected tandems.

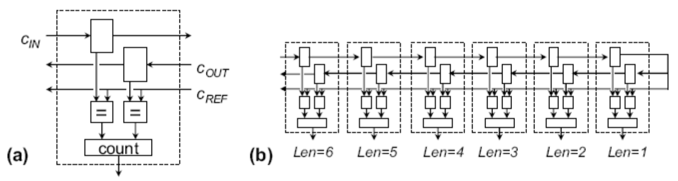

Fig. 4. Architecture of circuit for tandem repeat detection with k mismatches: a) comparison unit b) array of comparison units applied to each tandem length

The procedure to detect tandems with one indel and k mismatches is quite complicated. The comparator structure is replicated for every prefix and every length. The amount of resources used by the circuit grows rapidly, also because of the need to include the circuitry for evaluating the output of all the comparator units. For example, by extension of the circuit to 1 indel, the length of the array placed in single FPGA chip decreased from 1024 to 60. The authors themselves admit that this architecture is not ideal for detection of tandem repeats with indels.

In the presented work we focus on the creation of an architecture capable of detecting tandem repeats with k errors of any type (mismatch or indel). Furthermore, our architecture can detect tandem repeats of arbitrary length regardless of the number of available comparator units.

IV. Hardware Architecture

The proposed architecture of the circuit for tandem repeat detection is shown in figure 5. The circuit has three main parts responsible for the three phases of computation [8] as described in Chapter II:

1) Global score calculation in a systolic array of processing elements
2) Computation of wave maximum and assembling arrays $l_1[h]$ and $l_2[h']$
3) Detection of the best tandem repeat

Parts 1) and 2) occur twice in the circuit - once for the forward direction (calculation of the common prefix l_1) and once for the reverse direction (calculation of the common suffix l_2). Please note that this is only the architecture of the computation core. This architecture needs to be applied in each iteration for every position of the interval $n/2 \ldots n$. Connecting several cores to increase the parallelism of the design is left for the designer who will integrate the circuit into a specific target platform.

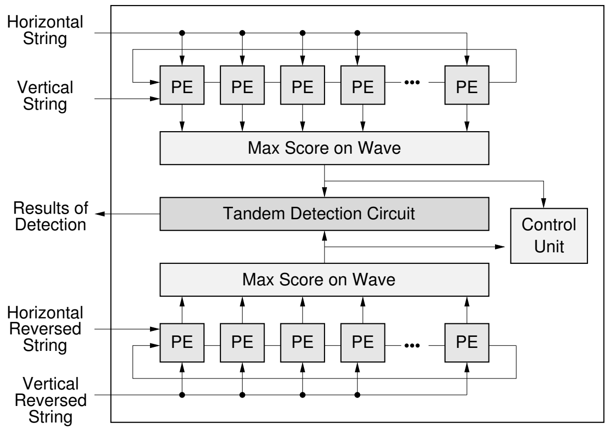

Fig. 5. Hardware architecture of a circuit for tandem repeat detection with k errors (mismatches and indels)

A. Global alignment calculation

The way calculations are carried out in hardware is similar to previously reported architectures for sequence alignment [22]. In each cycle of the clock the PE element calculates the score of one cell of the DP matrix, using the previously calculated score of three neighboring cells and two input characters (see figure 6a). Each processing element calculates the score in one column of the DP matrix and sends the intermediate results to the neighboring PE on the right. The most important difference from commonly used approaches is limiting the calculation to $2k+1$ diagonals (the array is $k+1$ elements long). At the beginning of calculations the elements are activated sequentially until they all process cells on the antidiagonal in parallel. In general, the calculations are carried out in bands and the highest parallelization is obtained on the antidiagonals of the DP matrix. As soon as a PE reaches the last diagonal, it continues calculations on the first diagonal of the next band (see figure 6b).

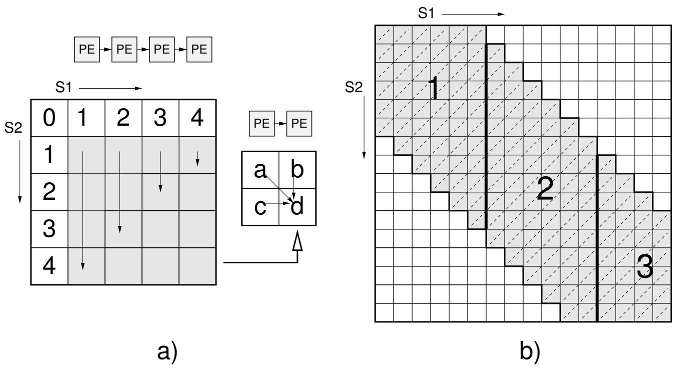

Fig. 6. Organization of PE elements and separation of computation into the stripes

Please note that while in software implementations the calculation of global alignment is accelerated via operations on a suffix array, in hardware this is achieved by sequential calculation of as many antidiagonals as are needed to detect prefixes/suffixes with up to k errors. Finding a suffix array solution in hardware is still an open problem.

B. Calculating the wave maximum

In each step the PE elements process one antidiagonal of the DP matrix. Scores generated on PE outputs therefore represent the scores on the DP matrix antidiagonal. The maximum on the wave can then be calculated as follows: (1) Calculate the minimal score on antidiagonals (minimum of the PE outputs), (2) if the minimum changes from k to $k+1$ while traversing from antidiagonal i to $i+1$, then maximum of the wave for k errors has been reached on antidiagonal i. The function of this circuit can be easily demonstrated on figure 3. The maximum of the wave for 2 errors lies on the eight antidiagonal. This maximum corresponds to the minimum of all scores on this antidiagonal. Other waves are solved in a similar manner.

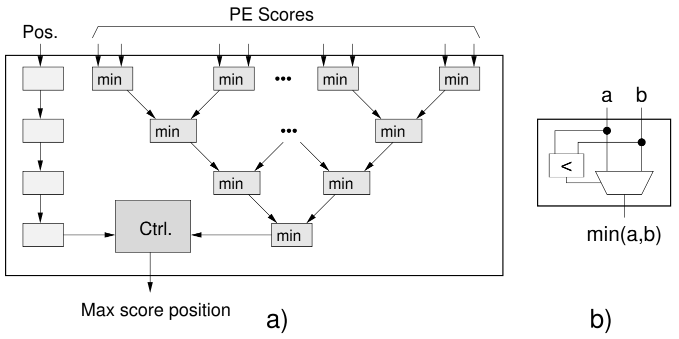

Fig. 7. Architecture of circuit for wave maximum detection: a) tree structure of units, b) detailed architecture of unit for minimum selection

The circuit architecture is made of comparators joined in a tree-like structure. They output the minimum of the two input elements (see figure 7b). There is a unit at the end of the tree-like structure that detects the change in the minimum identified by the previous step and outputs the position (antidiagonal number) that caused the change. The whole circuit can be pipelined to obtain desired working frequency.

C. Tandem detection

The role of the circuit is to detect whether there is a tandem satisfying the conditions $l_1[h] + l_2[h'] \geq p$ and $h + h' \leq k$ between the calculated prefix (l_1) and suffix (l_2). Also, if more tandems are detected at the same position, the one with the minimal number of errors is chosen. To select the tandem repeat with the lowest number of errors, the tandem condition has to be tested for positions corresponding to $0, 1, \ldots k$ errors. Table I shows all indexes of l_1 and l_2 arrays that need to be evaluated for a given number of errors.

The circuit architecture is made of an array of $k+1$ comparator units which for each pair of positions a and b evaluates whether the analyzed sequence fulfills the condition for a tandem repeat $a+b \geq p$. Input values of wave maximum in forward direction $l_1[h]$ are shifted into the first register array from left to rigth. Input values of wave maximum in the reverse direction $l_2[h']$ are shifted into the second register array in

TABLE I
LIST OF INDEXES OF l_1 AND l_2 ARRAYS TO BE COMPARED FOR DETECTION OF TANDEM WITH THE LOWEST NUMBER OF ERRORS

0	1	2	...	k
h_0, h_0‘	h_0, h_1‘	h_0, h_2‘	...	h_0, h_k‘
	h_1, h_0‘	h_1, h_1‘	...	h_1, h_{k-1}‘
		h_2, h_0‘	...	h_2, h_{k-2}‘
			...	...
				h_k, h_0‘

opposite direction. As soon as the values from both arrays meet each other (somewhere in the center), the calculation starts. Arrival of next values triggers evaluation of next number of errors. All positions referenced in Table I are compared and the tandem repeat with the lowest number of errors is labelled on the output.

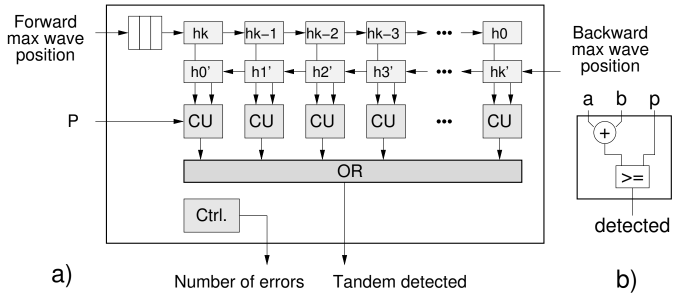

Fig. 8. Architecture of circuit for tandem detection: a) organization of comparison units, b) detailed architecture of comparison unit

The entire calculation of one iteration is finished when the prefix and suffix with at most k errors is detected (output *Max Score on Wave*). The control logic detects this state and gets ready for the next iteration.

Please note that the proposed architecture does not limit the length of the detected tandem repeat. Moreover it is capable of detecting tandem repeats with both types of errors (mismatches and indels). The proposed architecture creates a general framework for tandem repeat detection that can be fine-tuned by a set of generic parameters: the number of errors k or data width characteristics (input symbols, score values and positions in string). Using configurable FPGA technology all these parameters can be set at the time of synthesis and thus optimize the resulting circuit for the amount of resources used as well as working frequency.

V. EVALUATION AND RESULTS

The goal of this chapter is to compare the algorithm implemented in software (SW) and the proposed hardware architecture (HW) in terms of performance. As the basic metric for this comparison the number of iterations computed in unit time is chosen. In this context, a single iteration represents the three core phases of the algorithm: calculating forward and reverse prefixes/suffixes by global alignment, calculating the wave maximum and tandem repeat detection with the minimum number of errors. As the algorithm sotfware implemenentation and its performance characteritics are not available as free, we implemented the algorithm core described in Chapter II in the C programming language. To calculate the suffix array and find the longest common prefix, we used available optimized libraries [23], [24]. The program has been tested on a computer equipped with an Intel Core2 Duo E8400 3GHz processor and 2GB of RAM. To evaluate computation speed we ran 1 million iterations and calculated the average number of iterations per second. The results for different values of k are presented in Table II.

To ensure, that our SW implementation has appropriate performance characteristics, we converted the measured computation times to the number of operations per unit time, where single operation corresponds to LCP or DP rule application. This performance achieves roughly 20 millions operations per second for sufficiently high values of k (for small values of k the overhead of other algorithm parts is more noticeable). Implementations of similar algorithms, such as approximate string matching (ASM) usually achieve performance between 25 and 40 millions operation per second (for a 3GHz Xeon processor [25]). However, since ASM is based on simpler operations (only DP rules are used) we consider our implementation to be satisfactory, close enough to the theoretical maximum.

When comparing the speed of SW and HW computation, we need to take into account the use of different implementations of sequence alignment. While the SW implementation uses a suffix array, the HW architecture calculates the global alignment by dynamic programming. Please note, that SW always carries out the same number of operations to get the global alignment (dependent on k) regardless of the input sequence. The number of steps in the hardware implementation, on the other hand, will depend on the number of processed antidiagonals before k errors are reached. The number of steps in hardware will therefore be higher in sequences with a low proportion of errors. In the best case (every symbol introduces an error), the HW implementation will go through $2k+1$ steps. In the worst case (the sequence contains n symbols of the same kind), the HW implementation will carry out n steps. To evaluate the HW implementation in regards to its intended real-life use, we decided to determine the number of steps necessary on different types of real-life and model sequences. The testing set included the following sequences:

1) Random sequence of 10k bases (all four symbols generated with equal probability)
2) Common DNA sequences. 100 randomly chosen segments of 1000 bases from chromosome Y of the human genome. Chromosome Y has a higher content of repeats (about 61%)
3) DNA sequences made solely of VNTRs (period 6 and more) and STRs (period 2-5 symbols). We have randomly chosen 100 segments from chromosome Y for each type)

For each group of sequences we measured the number of steps l carried out by the HW implementation and averaged

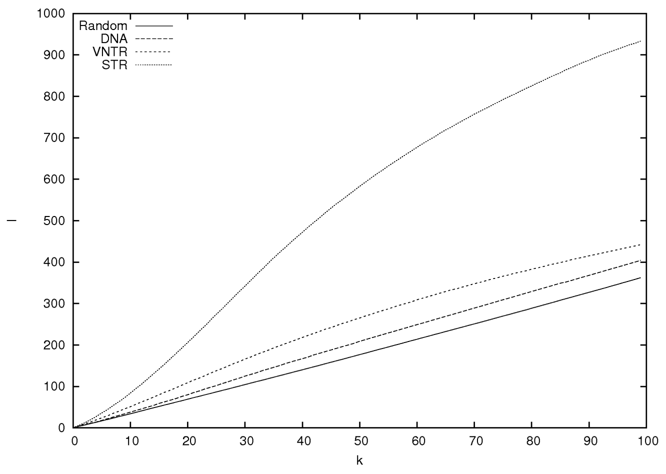

Fig. 9. Characteristics of the number of computation steps performed in the HW circuit based on the number of detected errors k, for different types of sequences

the number over all sequences in the group. The resulting graph of the dependency between k and l is shown in figure 9. As expected, the lowest number of steps was necessary for a random sequence and only fractionally more steps for common DNA. Sequences containing repeats contained longer common prefixes and suffixes causing l to be higher. This effect was more pronounced in the sequences with STR repeats. Because the dependency between l and k is very close to being linear, we only consider the slope $r = l/k$ of the relationships to evaluate HW performance. The slope is 3.6 for random sequences, 4.0 for DNA, 5.1 for VNTR and 10.4 for STR.

Let us now derive the true number of steps in HW necessary to finish one iteration. The number of clock cycles needed to calculate the global alignment will be, on average, $l = r.k$. All the other parts of the circuit work in pipelined manner, up to the moment the end of calculation is detected by the *Max wave score* unit when k errors are reached. Upon complete pipelining of the *Max wave score* unit, it will take $log(k + 1)$ cycles to detect the end of calculation, so that new iteration can begin. The average time for one iteration will therefore be $l + log(k + 1)$ cycles. Please note, that the computation can be terminated sooner, e.g. if a tandem repeat with less than k errors is presented in the input sequence, then *Tandem detection circuit* can announce such occurrence before all l steps are executed. However this optimization can be performed at SW level as well. To avoid irregularities in algorithm structure, we exclude such mechanism from our algorithm.

The proposed architecture was implemented in the VHDL language. The code is well written, but not strongly optimized. As target FPGA chip we used Virtex 5 xc5vlx330t speedgrade -2. Synthesis was carried out using Xilinx ISE tools. The resulting amount of utilized resources and the working frequency (after Place and Route process) for selected number of errors k are given in Table II. Based on the derived number of steps in HW, we calculated the number of iterations achievable per second. The amount of utilized resources clearly shows we can place several units on one chip and further increase the degree of parallelization. We therefore also present an estimate of the number of arrays that can be place on the chip to estimate the overall speedup.

With respect to the results shown in Table II , we can conclude:

- The amount of necessary resources grows linearly with increasing k. However, it contains a hidden logarithmic component introduced by the generic setting of data width for score value distribution. This is demonstrated by saving resources in the block to calculate wave maximum (see Chapter IV). From the increase in resource utilization we estimate that Virtex5 chip can hold an array for detection of tandems with about 500 errors.
- For higher values of k we can see a tendency for lower working frequency. This decrease is caused by the increasing length of busses supplying the PE array with new symbols from the computation matrix column. This effect can be eliminated by increasing the amount of resources used (for example by prefetching strings or their parts into the register array).
- Performance of the SW implementation running on the CPU has a quadratic relationship to k, since it must carry out k^2 steps to calculate the global alignments. The performance of the HW circuit decreases linearly, because all diagonals are processed in parallel and l only grows linearly with increasing k. For higher values of k, the effect of lower working frequency enters the equation.
- To compare performance of a program running on the CPU and a HW circuit in FPGA, the speedup per array lies between 80 and 873. Using the available resources on the chip we can create more arrays and thus increase the parallellism and speed of computation. The number of arrays that can be created in this manner can reach up to 271, depending on the value of k. For the sake of simplicity, we only approximated the number of arrays with respect to the amount of available resources. In real implementation, it is necessary to include circuits distributing input data to individual arrays, blocks controlling DMA operations and other characteristics of the specific target platform.
- The overall speedup decreases with the decreasing number of arrays on the chip and lies between 2 441 and 28 255. These HW performance values are based on the assumption that searches are done on DNA sequences. In the case of searching repeat-rich sequences, like those of VNTR or STR, the number of steps in HW increases and speedup needs to be adjusted according to the following table:
  The overall speedup is in the order of thousands in respect to the SW algorithm using suffix arrays

VI. Conclusions

In this work we proposed a new architecture for detection of approximate tandem repeats in DNA sequences. This ar-

TABLE II
CHARACTERISTICS OF HW IMPLEMENTATION SUCH AS AMOUNT OF CONSUMED RESOURCES AND WORKING FREQUENCY (AFTER PAR), COMPARISON OF HW AND SW PERFORMANCE AND CALCULATION OF THE RESULTING SPEEDUP

k	SW Perf. [iters/sec]	Res. util. [Slices]	HW Steps	Freq. [MHz]	HW Perf. [iters/sec]	Speedup	#Arrays	Overall Speedup
1	492,611	191	6	256	42,735,043	104	271.4	28,255
2	307,692	231	11	246	23,310,023	80	224.4	17,920
5	102,775	446	24	244	10,675,549	103	116.2	12,011
10	34,710	800	45	235	5,183,630	154	64.8	9,979
20	10,077	1,452	86	226	2,650,537	264	35.7	9,429
50	1,759	3,502	207	182	870,749	501	14.8	7,422
100	694	8,445	408	148	363,755	814	6.1	4,996
150	201	12,496	609	106	174,685	870	4.1	3,611
200	113	19,786	809	80	98,888	873	2.6	2,287

TABLE III
RANGE OF SPEEDUP FOR DIFFERENT TYPES OF INPUT SEQUENCES

Sequence Type	Speedup
DNA	2,441 - 28,255
VNTR	1,918 - 20,182
STR	945 - 11,773

chitecture allows us to search for tandem repeats with both types of errors (mismatches and indels) and does not limit the length of the tandem repeat. With the increasing number of tolerated errors, the amount of resources used by the circuit only grows linearly. We implemented the proposed architecture on a Virtex5 FPGA chip and compared the performance of this implementation to the best available solution in software, based on suffix arrays. The obtained speedup on single FPGA chip with several instances of this architecture reaches thousands, compared to software running on a 3GHz processor.

The proposed hardware architecture utilizes the advantages of FPGA technology. Configurability of the chips allows us to specify parameters such number of detected errors and data widths for input sequence, computed scores and positions before the synthesis process and thus create circuit optimized for target application. The created circuit can achieve higher working frequency, consume less resources and contain more processing elements.

Further development and improvements of HW architectures will to a certain level depend on the development of general algorithms for software implementations. For some time it has been assumed that algorithms for detection of tandem repeats cannot be improved. The most cited algorithms in this respect were the algorithm for perfect tandem repeats [16] with time complexity $n.log(n)$ and the algorithm for approximate tandem repeats [8] with time complexity $n.k.log(k).log(n/k)$. However, in 2004, a new approach has been published [3], allowing to find perfect tandems in linear time. A corresponding algorithm for approximate tandems has not been described so far. It will be interesting to follow future developments in this area and study the possibility to accelerate the new approaches using FPGA technology.

Our increasing ability to utilize hardware acceleration to manipulate and analyze huge amounts of biological sequence data for similarities and occurrence of patterns becomes important in the light of the rapid developments in sequencing technology taking place today. While it took a decade to decipher the first human genome, next-generation sequencing methods and new methods under development, (single-molecule sequencing and nanopore sequencing of DNA) are aiming at generating billions of bases per minute at a cost of hundreds of dollars at most. For example the newest sequencing machine from Helicos contains a two-processor server connected by fiber-optics to the sequencing microscope just to process the raw data in real-time and sequencing is seen as a future method of choice for personal genomics and a number of medical diagnostic procedures. Even with smaller datasets, users have an increasing need to browse the available data in real time. Hardware acceleration techniques such as the one proposed in this paper may well be the best choice to handle the speed requirements and ever increasing volume of biological sequence data generated by sequencing centers world-wide.

ACKNOWLEDGMENT

This research has been partially supported by the Grant Agency of the Czech Republic Research Grant No. 204081560 – In vitro and in silico identification of non-canonical DNA structures in genomic sequences and MSMT Research Grant No.0021630528 – Security-Oriented Research in Information Technology.

REFERENCES

[1] H. Debrauwere, C. G. Gendrel, S. Lechat, and M. Dutreix, "Differences and similarities between various tandem repeat sequences: Minisatellites and microsatellites." *Biochimie*, vol. 79, no. 9-10, pp. 577–586, 1997.

[2] B. Rhead, D. Karolchik, R. M. Kuhn, A. S. Hinrichs, A. S. Zweig, P. Fujita, M. Diekhans, K. E. Smith, K. R. Rosenbloom, B. J. Raney, A. Pohl, M. Pheasant, L. Meyer, F. Hsu, J. Hillman-Jackson, R. A. Harte, B. Giardine, T. Dreszer, H. Clawson, G. P. Barber, D. Haussler, and W. J. Kent, "The ucsc genome browser database: update 2010." *Nucleic Acids Research*, vol. 38, no. Database issue, pp. D613–D619, 1997.

[3] D. Gusfield and J. Stoye, "Linear time algorithms for finding and representing all the tandem repeats in a string," *J. Comput. Syst. Sci.*, vol. 69, no. 4, pp. 525–546, 2004.

[4] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," Tech. Rep. 8, 1966.

[5] S. Kannan and E. W. Myers, "An algorithm for locating non-overlapping regions of maximum alignment score," in *CPM '93: Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching*. London, UK: Springer-Verlag, 1993, pp. 74–86.

[6] G. Benson, "A space efficient algorithm for finding the best nonoverlapping alignment score," *Theor. Comput. Sci.*, vol. 145, no. 1-2, pp. 357–369, 1995.

[7] J. P. Schmidt, "All shortest paths in weighted grid graphs and its application to finding all approximate repeats in strings," in *Israel Symposium on Theory of Computing Systems*, 1995, pp. 67–77.

[8] G. M. Landau, J. P. Schmidt, and D. Sokol, "An algorithm for approximate tandem repeats," *Journal of Computational Biology*, vol. 8, no. 1, pp. 1–18, 2001.

[9] D. Sokol, G. Benson, and J. Tojeira, "Tandem repeats over the edit distance," *Bioinformatics*, vol. 23, no. 2, pp. e30–e35, 2007.

[10] G. Benson, "Tandem repeats finder: a program to analyze dna sequences," *Nucl. Acids Res.*, vol. 27, no. 2, pp. 573–580, January 1999. [Online]. Available: http://dx.doi.org/10.1093/nar/27.2.573

[11] "mreps: efficient and flexible detection of tandem repeats in dna," *Nucl. Acids Res.*, vol. 31, no. 13, pp. 3672–3678, July 2003.

[12] O. Delgrange and E. Rivals, "Star: an algorithm to search for tandem approximate repeats," *Bioinformatics*, vol. 20, no. 16, pp. 2812–2820, 2004.

[13] A. Krishnan and F. Tang, "Exhaustive whole-genome tandem repeats search," *Bioinformatics*, vol. 20, no. 16, pp. 2702–2710, 2004.

[14] "Rap: a new computer program for de novo identification of repeated sequences in whole genomes," *Bioinformatics*, vol. 21, no. 5, pp. 582–588, March 2005.

[15] Y. Wexler, Z. Yakhini, Y. Kashi, and D. Geiger, "Finding approximate tandem repeats in genomic sequences," in *RECOMB '04: Proceedings of the eighth annual international conference on Resaerch in computational molecular biology*. New York, NY, USA: ACM, 2004, pp. 223–232.

[16] M. G. Main and R. J. Lorentz, "An o(n log n) algorithm for finding all repetitions in a string," *J. Algorithms*, vol. 5, no. 3, pp. 422–432, 1984.

[17] D. Gusfield, "Algorithms on stings, trees, and sequences: Computer science and computational biology," *SIGACT News*, vol. 28, no. 4, pp. 197–198, 1997.

[18] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins." *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, March 1970. [Online]. Available: http://view.ncbi.nlm.nih.gov/pubmed/5420325

[19] E. Ukkonen, "Algorithms for approximate string matching," *Inf. Control*, vol. 64, no. 1-3, pp. 100–118, 1985.

[20] G. M. Landau, E. W. Myers, and J. P. Schmidt, "Incremental string comparison," *SIAM J. Comput.*, vol. 27, no. 2, pp. 557–582, 1998.

[21] A. A. Conti, T. V. Court, and M. C. Herbordt, "Processing repetitive sequence structures with mismatches at streaming rate," in *Field Programmable Logic and Application (FPL 2004)*, ser. Lecture Notes in Computer Science. Springer, 2004, pp. 1080–1083.

[22] C. W. Yu, K. H. Kwong, K.-H. Lee, and P. H. W. Leong, "A smith-waterman systolic cell." in *Field Programmable Logic and Application (FPL 2003)*, Lisbon, Portugal, September 2003, pp. 375–384.

[23] T. Kasai, G. Lee, H. Arimura, S. Arikawa, and K. Park, "Linear-time longest-common-prefix computation in suffix arrays and its applications," in *CPM '01: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching*. London, UK: Springer-Verlag, 2001, pp. 181–192, sotware library implemented by M. McIlroy and available at: http://www.cs.dartmouth.edu/˜doug/sarray/.

[24] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe, "Nearest common ancestors: a survey and a new distributed algorithm," in *SPAA '02: Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 2002, pp. 258–264, sotware library implemented by H. Bannai and available at: http://tlas.i.kyushu-u.ac.jp/˜bannai/.

[25] T. V. Court and M. C. Herbordt, "Families of fpga-based accelerators for approximate string matching," *Microprocessors and Microsystems*, vol. 31, no. 2, pp. 135 – 145, 2007, special Issue on FPGA-based Reconfigurable Computing (2).