Architecture model for approximate palindrome detection

Tomáš Martínek and Jan Voženílek
Faculty of Information Technology
Brno University of Technology
Božetěchova 2, Brno, 612 66, Czech Republic
Email: martinto@fit.vutbr.cz, xvozen00@stud.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. One of the interesting structures occurring in DNA is a palindrome. Biologists believe that palindromes play an important role in regulation of gene activity and other cell processes because they are often observed near promoters, introns and specific untranslated regions. Unfortunately, the time complexity of algorithms for palindrome detection increases when mutations in the form of character insertions, deletions or substitutions are taken into consideration. In recent years, several works have been aimed at acceleration of such algorithms using dedicated circuits capable of potentially large-scale searching. However, widespread use of such circuits is often complicated by varying user task details or the need to use a specific target platform. The objective of this work is therefore to create a model of hardware architecture for approximate palindrome detection and develop a technique for automatic mapping of this model to the target platform without intervention of an experienced designer. The proposed model and the mapping technique are implemented and evaluated on a family of chips with Virtex5 technology.**

I. Introduction

A palindrome is a sequence of symbols ordered in such a way, that the order is identical when read forward and backward (e.g. $abba$, $ababa$). Generally, palindromes can be written in the form $p = w.w^R$ or $p = w.c.w^R$, where w is a string, w^R is its reversed version and c is a central unpaired symbol. Depending on the presence of the symbol c, the palindrome is called *even* or *odd*.

In molecular biology, analysis of DNA molecules is one of the key subjects of research because it helps to understand functioning of living organisms at the molecular level. DNA is often represented as a long sequence of four letters A, C, G and T (corresponding to basic chemical units – nucleotides). It is believed, that structures formed by palindromic subsequences play a role in regulation of gene activity or other processes in cells. For example, hairpin and triplex palindrome-based structures are known to be present in close vicinity of genes (e.g. in promoters, introns and 3'-untranslated regions) contributing to their normal functioning, or to diseases, such as cancer. In short, the knowledge of the exact positions of palindromes in DNA is an important bit of information for molecular biologists trying to understand how entire genomes are organized and what the functions of its individual components are.

The process of searching for palindromes in biological data is often complicated by the presence of mutations introduced by evolutionary processes. These mutations occur in sequences as character insertions, deletions or substitutions. The algorithms, which search for palindromes in DNA sequences have to tolerate these changes, in order to find not only exact palindromes but also *approximate palindromes*. Unfortunately, the time-complexity of such algorithms is higher, which complicates their practical usage for large-scale or even interactive searching important for molecular biologists.

One of the ways to obtain interactivity is to implement the searching procedures in hardware. In our previous work [1] a novel hardware architecture was developed, capable to speed up the palindrome detection in orders of thousands in comparison to the best known algorithm implemented in software. However, these theoretical properties of computation core can change significantly if it is connected into a real system. The parameters of resulting circuit depend primarily on target platform characteristics and task parameters. The typical problems that users of the architecture have to address are:

- Is I/O bandwidth sufficient to feed the hundreds or thousands of processing elements (PEs) with new data and to process the output data that they generate?
- How many PEs does it really make sense to implement with respect to limited I/O bandwidth or amount of available resources?
- Then, what is the real speed-up of the system?
- How do the architecture parameters change, if a new chip with different technology is used?

The objective of this paper is to extend our previous work and create a general model of architecture for approximate palindrome detection, which would be capable to answer all previously mentioned questions and develop a method for mapping of the architecture into the target platform fully automatically without any intervention of an experienced designer.

This paper is organized as follows: Section II describes the method for approximate palindrome detection and its hardware acceleration. Related work in the area of acceleration of approximate palindrome searching is summarized in section III. Section IV contains detailed description of our hardware

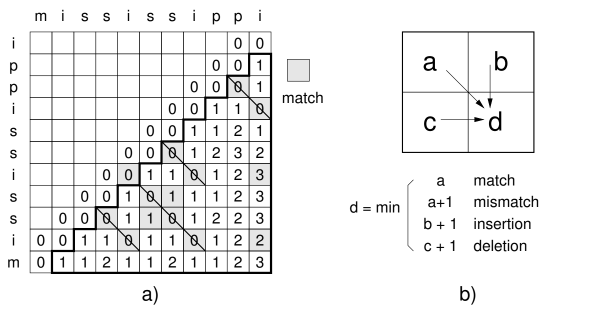

Fig. 1. Dynamic programming algorithm demonstrated on the string *mississippi*: a) the DP matrix with calculated score values b) the scheme of DP rule calculating new score d based on neighboring values a, b and c.

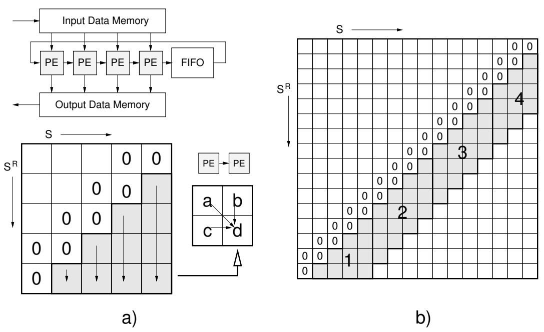

Fig. 2. Hardware architecture (a) Allocation of PEs for DP matrix computation (b) Computation of the first k-antidiagonals split into stripes

architecture model and the technique for its automatic mapping to the target platform. Evaluation of the proposed model on FPGA chips with Virtex5 technology is given in section V. Conclusions are summarized in section VI.

II. Hardware Acceleration of Approximate Palindrome Detection

Algorithms for palindrome searching have been studied intensively in the past. One of the first algorithms for finding all palindromes [2] use dynamic programming (DP) technique to calculate a two dimensional matrix of all possible palindrome alignments. Unfortunately, the algorithm time complexity converges to $O(n^2)$. Another (more practical) approach does not calculate the whole DP matrix, but only searches for palindromes with at most k errors. The best algorithms of this kind are based on suffix trees or suffix arrays [3], [4], which allow them to reduce time complexity to $O(kn)$.

Although, the suffix array-based method seems to be very promising, we do not usually search for palindromes with exactly (or up to) k errors when analyzing real sequences. Rather, we tend to tolerate more errors in longer palindromes and less errors in shorter ones. Therefore, it is more natural to define an acceptable average frequency of errors, rather than a fixed value k. This frequency can be expressed as $e = l/k$, where l is the length of the palindrome in question. If the suffix array-based method is modified accordingly, the computation has to go through $k = n/e$ cycles of diagonal extensions. The time complexity in such case will increase to $O(n^2/e)$.

Although, the suffix array-based method is very effective and calculates only the necessary number of DP matrix items, its hardware acceleration is complicated by several factors: i) entire suffix array would apparently have to reside inside the chip and thus consume significant amount of resources, and ii) during the computation the suffix array would have to be concurrently accessed from a number of chip locations corresponding to the level of parallelization. This would result in a complex interconnection system. For these reasons, we decided to accelerate the basic dynamic programing approach. Following subsections describe the DP method and its hardware acceleration in more detail.

A. Dynamic programing technique

A convenient way to search for approximate palindromes is by the way of a dynamic programming algorithm. Original algorithm can be traced back to [2]. A DP matrix is constructed so that one side represents the original sequence, while the other contains the same sequence reversed (according to the nucleotide-pairing rules for DNA sequences). With such setup, the main antidiagonal of the DP matrix represents all the n possible starting positions for odd palindromes. The neighboring antidiagonal contains the other $n-1$ possible starting sites of all the even palindromes that can exist in the sequence. Consequently, diagonals starting at any of these positions represent potential palindromes. If we fill the cells representing the starting positions with zeros, we can start filling the DP matrix along the diagonals. The numbers entered will represent the number of errors found so far in the evaluated palindromes. At each position $[i, j]$ of the DP matrix, we compare the symbols at positions i and j in the original and reversed sequences. If they are the same, no penalty is introduced. If the symbols differ, the number of errors identified so far in the particular palindrome score is incremented by one.

The necessity for a dynamic programming algorithm comes from the possibility to insert gaps into the palindromes, where symbols in some positions have no symbols to pair up with in the palindrome. In terms of the described algorithms, this means moving from one diagonal to a neighboring one when calculating the number of errors. At any position, three possibilities are evaluated: (1) Extending the existing palindrome along the diagonal - *match* or *mismatch*, (2) Inserting a gap at position i of the original sequence - *insertion*, (3) Inserting a gap at position j of the reversed sequence - *deletion*. The solution that leads to the lowest number of errors is kept, the score is recorded in the DP matrix, while the other possibilities are discarded.

B. Hardware architecture

Our architecture for the palindrome detection is similar to the one used in acceleration of approximate string matching (ASM) [5]. Nevertheless, there are some important differences between the two architectures. While the whole area of the DP matrix is computed in ASM, only the lower triangular

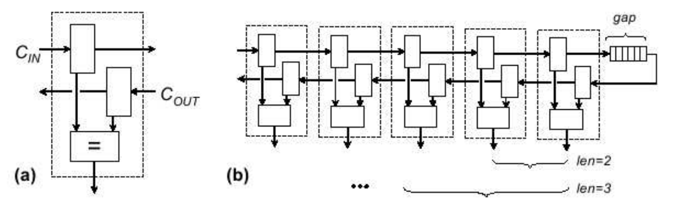

Fig. 3. Hardware architecture for palindrome searching with k-mismatches

part is significant in palindrome searching (see Fig. 2a). As the probability of palindrome occurrence decreases with its length, it makes sense to calculate only a limited number of cells (the first k antidiagonals).

Similarly to ASM, the architecture of the circuit is based on a systolic array of processing elements, where each element calculates a single column of the DP matrix. Note that the first ASM element begins computation in the upper left corner of the matrix and the next elements start their computation consequently in an ordered fashion because of data dependencies. On the other hand, palindrome detection begins on the central antidiagonal and all elements can proceed on the diagonals in parallel. Data dependency does not require the elements to wait for each other.

If the number of processing elements is lower than the length of the input string, computation is divided into bands, so that results generated by the last element are temporarily stored in a buffer (FIFO). Upon transfer of computation to the next band, the first processing element accesses the data in the buffer (see Fig. 2b). Similarly to ASM, the PE element calculates a typical DP rule and resulting score shifts to its neighbour on the right. Besides the score calculation, the PE controls the actual number of errors and palindrome length. As soon as this ratio decreases under the specified threshold (e parameter), palindrome is exported and PE stores information about palindrome position and length into the output memory.

III. Related works

Palindrome search acceleration in hardware has been studied by Conti et al. [6]. Their architecture is made of an array of processing elements connected into a loop. The input string progresses through the array from left to right, changes direction at the end of the array and continues in the opposite direction. The processing elements contain only comparators which signal palindrome-forming matches on individual positions (see Fig. 3). This architecture is able to detect approximate palindromes of all relevant lengths at every step. Only mismatches can be evaluated. Time-complexity of this approach is $O(n)$ as compared to $O(kn)$ for the best algorithms implemented in software. Using a longer array of processing elements leads to detection of longer palindromes without changes in time-complexity.

The authors also address the problem of detecting palindromes containing insertions and deletions. To do that they rely on a triplet of the above mentioned arrays. The first array operates as described for detecting palindromes with mismatches. The second array compares one symbol ahead and the third array compares one symbol behind the normal position. Analyzing the results from these three arrays the authors can detect possible insertions. Unfortunately, generalization of this approach to k-errors leads to high number of comparator arrays and an unnecessarily high use of resources on the chip.

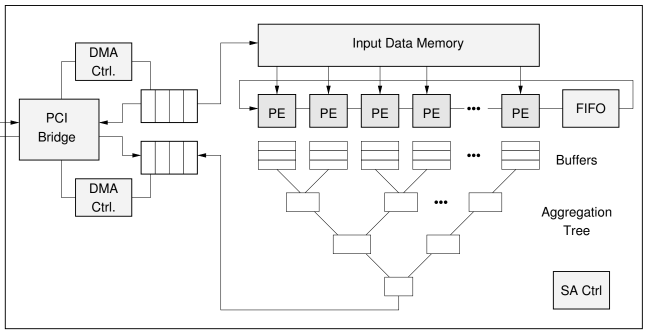

Fig. 4. System architecture including Input data memory, output aggregation tree and blocks necessary for DMA transfers

In comparison to the approach described in section II, our architecture is more scalable, because more processing elements in systolic array accelerate the computation. Moreover, amount of consumed resources is independent of amount of accepted insertions and deletions.

IV. Architecture model

The architecture described in section II shows an approach to palindrome detection parallelization and represents a basic template of circuits. However, for real use of this architecture, it is necessary to take into account the environment of the circuit and evaluate it with respect to the whole system. An example of such system is shown in Fig. 4. The architecture of the computation core is placed into the chip connected to the system bus (e.g. PCIe). This hardware accelerator usually works as a fast filter, which detects considerable fraction of the palindromes present in the sequence very quickly and remaining ones (extremely long palindromes) are left for the software part of the application.

An input sequence is transfered from the host RAM memory into the *Input Data Memory* through the system bus. On the other side, the positions of exported palindromes are aggregated from all PEs into a the single stream using a tree structure of multiplexers and comparators. The resulting stream is transfered back into the host RAM using DMA controllers and auxiliary buffers. This system architecture can be limited by the parameters of the target platform, such as available I/O bandwidth or amount of available resources. In some cases, it may not make sense to implement more than a certain number of PEs due to limited I/O bandwidth. In other cases, amount of resources may not allow to increase systolic array length and circuit performance.

A. Computation Model

The task of the model is to show architecture characteristics for selected parameters (number of PEs, number of computa-

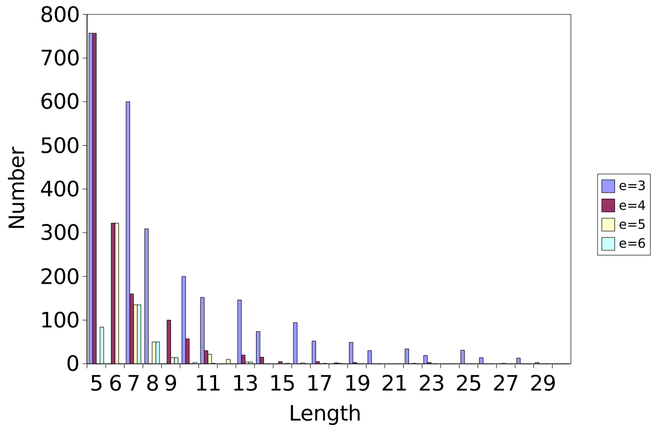

Fig. 5. Histogram of palindromes occurred in DNA sequences including four characteristics for different palindromes quality (e parameter)

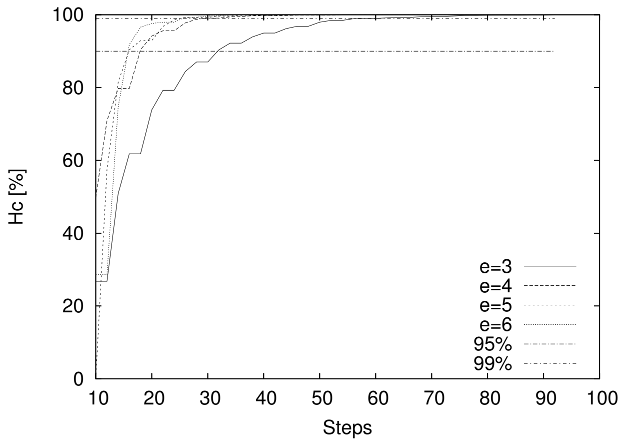

Fig. 6. Cumulative histogram representing influence of exported palindromes (in percentage) with respect to the number of computation steps

tion steps) or different surrounding conditions (I/O bandwidth, amount of resources etc.). At first, to build the model, we need to find out how the system behaves with respect to input and output. Relevant information is shown in a histogram of exported palindromes shows.

This histogram primarily depends on the type of analyzed sequence and usually it can only be obtained experimentally. As a test set, we used DNA sequences 10k characters long obtained from promoters in the first human chromosome (hg18 release of UCSC Human Genome). To eliminate the large amount of short palindrome occurrences, the algorithm was set up to search for palindromes longer than 4 characters. The test was repeated 10 times for different sequences and different requirements for palindrome quality (e parameter). Resulting histograms are depicted in Fig. 5.

The diagrams shows that with increasing length the palindromes occurrences decrease exponentially. This naturally corresponds to the probability of the palindrome occurrence in random sequences. Similarly, with increasing requirements for palindrome quality, the number of exported palindromes decreases.

Since the hardware accelerator works as a fast filter for detection of almost all palindromes (extremely long ones are left to the software), a natural question would be: How many computation steps does the circuit have to perform to cover e.g. 99% of all palindromes (number of computation steps = number of antidiagonals = 2 × length)? This information is shown by the cumulative histogram, where at each step the number of palindromes exported so far is calculated. The *cumulative histogram* can be derived from the previous histogram using the following equation:

$$H_c(n) = \sum_{i=1}^{S_{max}} H(i) \tag{1}$$

where $H(i)$ is the number of exported palindromes at step i and S_{max} is the maximal number of steps (proportional to the length of analyzed sequence). The number of necessary steps S_{opt} to cover 99% of palindromes can be taken from the cumulative histogram represented as a percentage (with respect to overall number of exported palindromes) or it can be calculated from the following inequality:

$$\frac{H_c(S_{opt})}{H_c(S_{max})} \geq 0.99 \tag{2}$$

As the cumulative histogram is monotonic, the S_{opt} value can be obtained for example using the bisection method. For the real architecture, it does not make sense to calculate more than S_{opt} computation steps.

Next, the number of computation steps influences the requirement for the system input bandwidth. All PEs of the systolic array have to be fed with new data in a timely manner. Let's assume an array with N_{PE} elements. Then for all these PEs a new input sequence characters have to be prepared during S_{opt} steps. Required input bandwidth corresponds to the equation:

$$B_{In}(N_{PE}) = \frac{N_{PE} \cdot C_{DW}}{S_{opt}} \cdot F_S \tag{3}$$

where C_{DW} is character data width in bits and F_S is the systolic array working frequency. Similarly, for output bandwidth, all palindromes exported during S_{opt} steps have to be transported back to the host RAM. The number of such exported palindromes can be taken from the cumulative histogram. Required output bandwidth corresponds to the equation:

$$B_{out}(N_{PE}) = \frac{H_c(S_{opt}) \cdot N_{PE} \cdot E_{DW}}{S_{opt}} \cdot F_S \tag{4}$$

where E_{DW} is exported item data width. As the histogram shows, the number of exported palindromes decreases exponentially with their length. For reliable transfer of all exported palindromes to the host RAM, the aggregation tree has to contain buffers of sufficient size. This buffer size is one of the parameters, that impacts the amount of consumed resources and thus the number of potential PEs placed in a chip.

The number of items that each PE generates on average can be taken from the cumulative histogram. On the other side, the number of items, which the system is able to take from each PE up to the step s, can be expressed as:

$$O_c(s) = \frac{B_{Sout}}{F_S \cdot N_{PE}} \cdot s \qquad (5)$$

where B_{Sout} is the available output system bandwidth. By subtraction of both these equations, we obtain the characteristic of utilized buffer items. Then the maximal value corresponds to the necessary buffer size.

$$Buf_{Size} = max_{i=1}^{S_{opt}}[H_c(i) - O_c(i)] \qquad (6)$$

Amount of resources needed for realization of the whole system can be calculated as a sum of all sub-components resources. General equations are shown in the following list:

$$\begin{aligned} R_{Arch} = & \quad N_{PE} \cdot R_{PE} + R_{Mem} + R_{FIFO} + \qquad (7) \\ & \quad R_{Tree} + R_{Ctrl} \\ R_{Mem} = & \quad f(N_{PE}, C_{DW}) \\ R_{Fifo} = & \quad f(S_{opt}, S_{DW}) \\ R_{Tree} = & \quad N_{PE} \cdot R_{Buf} + (N_{PE} - 1) \cdot R_{cmp} \\ R_{Buf} = & \quad f(N_{PE}, E_{DW}) \end{aligned}$$

where R_{PE} is the amount of resources needed for PEs, R_{Mem} for input data memory, R_{Fifo} for memory for intermediate results, R_{Tree} for aggregation tree, R_{Ctrl} for systolic array control logic, R_{Buf} for buffer for exported palindromes, R_{cmp} for comparator and multiplexer of aggregation tree.

Amount of resources for memory blocks (R_{Mem}, R_{Fifo}, R_{Buf}) is expressed generally as a function of number of items and data width. Different target technology implements these blocks in different ways. For example, a FPGA allows to implement memory using embedded blocks or using basic computation gates. Characteristics of these functions are usually linear or stair function depending on used technology.

B. Automatic mapping

Based on the architecture model, it is possible to evaluate not only a specific architecture, but also to find out architecture parameters, that fits the input task and target platform the best. Moreover, this mapping process can work fully automatically without any intervention of an experienced designer.

The type of the solved task implies, that the architecture is suitable for target platform only if requirements for I/O bandwidth are satisfied and simultaneously the circuit is realizable with limited the available amount of resources. The following set of inequalities has to be satisfied:

$$D = \begin{cases} B_{Out}(N_{PE}) & \leq \quad B_{Sout} \\ B_{In}(N_{PE}) & \leq \quad B_{Sin} \\ R_{Arch}(N_{PE}) & \leq \quad R_{FPGA} \end{cases} \qquad (8)$$

Note, that the left sides of inequalities contains general relations of the architecture model, while the right side corresponds to the parameters of the specific target platform. Architectures, that satisfy the inequalities generally represent a set of acceptable architectures, sometime labeled as *design space*.

The task of the mapping process is to explore the design space and find out the best (or a group of the best) candidates based on desired criteria (computation time, performance, power consumption, etc.). In our case, the criterion is a performance and therefore the objective is to find out an architecture with as many PEs as possible.

$$max(N_{PE}|D) \qquad (9)$$

Finding of such architecture is generally an optimization problem from the area of nonlinear programming, that leads to relatively complex mathematical methods in general. Please note that in our case all the used functions are linear (B_{Out}, B_{In}) or monotonic (R_{FPGA}). Therefore, the bisection method can be used for finding the optimal number of PEs.

V. Evaluation and Results

In this section, the proposed architecture model is evaluated on the family of chips with Virtex5 LXT technology. These chips contain huge amount of computation gates in range from 7200 slices (xc5lx50t) up to 51840 slices (xc5lx50t). Moreover, all these chips include embedded IP core for connection to the PCI Express bus x8.

The proposed architecture was implemented in VHDL language and the Xilinx ISE tool was used for synthesis. The blocks for controlling of DMA transfers and access to the PCI bus were used from the NetCOPE platform [7]. Amounts of resources consumed for individual parts of the system including operation frequency and maximal available I/O bandwidth are listed in the following table.

TABLE I
HARDWARE CHARACTERISTICS AFTER SYNTHESIS PROCESS

$R_{DMA+PCI}$ [Slices]	1900
R_{PE} [Slices]	14
R_{cmp} [Slices]	6
R_{ctrl} [Slices]	23
F [MHz]	251
B_{SIn} [Gbps]	16
B_{SOut} [Gbps]	16

With respect to the small number of utilized items, the LUT gates were used for memory block implementation. The amount of consumed gates corresponds to the equation:

$$R = (Items/32) \cdot (DataWidth/2) \qquad (10)$$

where single LUT is capable to store up to 32 two-bits items. Because the pipelined architecture is used, the working frequency is approximately 251 MHz and does not change with a longer systolic array. For the purposes of connection to the PCIe bus, the whole system is synchronized at frequency 250MHz.

At first, the model was evaluated with respect to limited system I/O bandwidth (see table II). Based on the different required quality of exported palindromes (e parameter), the number of computation steps S_{opt} for covering 99% of palindromes was calculated using equation 2. As the I/O bandwidth is constantly 16Gbps for all chips of Vitex5 family, the maximal number of PEs that can be fed with the new data ($N_{PE_{BIN}}$) as well as the number of PEs that the system is able to handle at the output ($N_{PE_{BOUT}}$) can be derived from equation 3 and 4.

Please note, that with increasing requirement for palindrome quality, the number of PEs supplied with the input bandwidth decreases, because number of necessary computation steps S_{opt} decreases. On the other side, the output bandwidth is able to handle more PEs, because a smaller number of palindromes are exported (H_c). Then, the minimum of both of these values represents the maximal number of PEs, that can be implemented in the chip due to the limited I/O bandwidth (N_{PE_B}). As the table shows, this value is approximately 800 for all e parameters. More PEs will be possible only for new chips with higher I/O bandwidth or different type of connection.

TABLE II
INFLUENCE OF LIMITED I/O BANDWIDTH

E	3	4	5	6
S_{opt}	56	32	26	24
$H_c(S_{opt})$	0.28	0.15	0.06	0.03
$N_{PE_{BIN}}$	1792	1024	832	768
$N_{PE_{BOUT}}$	800	853	1867	3345
N_{PE_B}	800	853	832	768

Amount of resources for the whole system is calculated using equation 8. As the function characteristic is not linear, the number of PEs can not be derived directly, but using the bisection method. Following table III summarizes the maximal number of PEs for individual chips of Virtex5 family. The values are in range from 200 to 2000.

TABLE III
INFLUENCE OF LIMITED AMOUNT OF RESOURCES

FPGA	Slices	PEs
xc5vlx50t	7 200	215
xc5vlx110t	17 260	625
xc5vlx220t	34 560	1 332
xc5vlx330t	51 840	2 035

If we compare numbers of PEs limited by I/O bandwidth and the amount of available resources, we will arrive at the following conclusions: (1) For chips xc5vlx110t and smaller, limiting factor is the amount of available resources. (2) On the opposite side, for chips xc5vlx220t and larger, the amount of resources is sufficient, but the extension of systolic array is limited by input or output bandwidth with respect to the required palindrome quality. (3) The highest speed up of architecture is achieved with at most 800 PEs. Using equation 11 adopted from [1], we calculate maximal speed- up of hardware in comparison to the best known method implemented in software. The result shows that the architecture realized in chips with Virtex5 family is capable to achieve speed- up up to 3566 depending on the selected chip.

$$\alpha = \frac{k}{l} \cdot \frac{p_2.n_{PE}}{p_1} = 0.535 \cdot \frac{200B \cdot 800}{0.03B} = 3566 \qquad (11)$$

VI. Conclusions

In this work, a general model of architecture for approximate palindrome detection was developed. This model takes into account the real aspects of architecture use such as available I/O bandwidth, amount of resources and input task parameters (required palindrome quality, character data width, etc.). Besides the model, a technique for its mapping to a specific target platform was designed. This method allows to find a suitable parameters of the system fully automatically, without intervention of an experienced designer and thus adapt the usage of the general architecture for specific tasks with different parameters or for a new generation of chips.

The proposed model was applied on the system in the form of an accelerator connected to the PCIe bus using chips with Virtex5 technology. Evaluation of the model shows, how the I/O bandwidth and amount of available resources affects the architecture properties. For the selected chips, it is possible to implement up to 800 PEs and the resulting speed-up of the hardware is 3566 in comparison to the best known method implemented in software that uses a suffix array data structure.

Acknowledgment

This research has been partially supported by the Research Plan No. GACR, 204081560 – In vitro and in silico identification of non-canonical DNA structures in genomic sequences and Research Plan No. MSM, 0021630528 – Security-Oriented Research in Information Technology.

References

[1] T. Martnek and M. Lexa, “Hardware acceleration of approximate palindromes searching,” in *The International Conference on Field-Programmable Technology*. IEEE Computer Society, 2008, pp. 65–72.
[2] G. M. Landau and U. Vishkin, “Efficient parallel and serial approximate string matching,” Tech. Rep. Computer Science Department Technical Report #221, February 1986. [Online]. Available: citeseer.ist.psu.edu/landau86efficient.html
[3] R. de Castro Miranda and M. Ayala-Rincón, “A modification of the landau-vishkin algorithm computing longest common extensions via suffix arrays,” in *BSB*, 2005, pp. 210–213.
[4] L. Allison, “Finding approximate palindromes in strings quickly and simply,” *CoRR*, vol. abs/cs/0412004, 2004, informal publication. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr0412.html
[5] 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.
[6] 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.
[7] T. Martnek and M. Koek, “Netcope: Platform for rapid development of network applications,” in *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*. IEEE Computer Society, 2008, pp. 219–224.