

Design and Implementation of Reconfigurable Processor for Problems of Combinatorial Computations^{*}

Iouliia Skliarova, António B. Ferrari

iouliia@ua.pt, ferrari@inesca.pt

Department of Electronics and Telecommunications

University of Aveiro

3810-193 Aveiro, Portugal

Abstract

The paper analyses different techniques that might be employed in order to solve various problems of combinatorial optimization and argues that the best results can be achieved by the use of software, running on a general-purpose computer, together with an FPGA-based reconfigurable co-processor. It suggests architecture of combinatorial co-processor, which is based on hardware templates and consists of reconfigurable functional and control units. Finally the paper demonstrates how to utilize the co-processor for two practical applications formulated over discrete matrices that are satisfiability and covering problems.

1. Introduction

With the advent of reconfigurable computing it becomes possible to create rapidly custom hardware implementations of various algorithms, which cannot be solved efficiently using other known approaches (such as ASIC-based implementations, realization in general-purpose computers, etc.). Effectiveness of reconfigurable circuits in general and FPGA-based circuits in particular depends on applications, and for certain classes of problems they give many advantages from the point of view of performance, required resources, etc. [1]. This potential has been realized by many different research machines, such as DECPeRLe [2], Splash 2 [3], PRISM [4], RENCO [5], Spyder [6], etc. Many applications where an FPGA-based system offers very high performance solutions including DNA pattern recognition, RSA cryptography [7], traveling salesman problem [8], long integer arithmetic [2], signal processing

[9], neural networks [10], image processing [11], satisfiability problem [12], etc. have been developed.

Current applications used to be implemented in reconfigurable systems are mostly data processing computations based on relatively simple algorithms. The applications suitable for configurable realization usually have a large number of bit-level manipulations that are typical for image processing, cryptography, etc. The paper suggests applying the configurable computing technique to problems that invoke rather complex flow of control operations appearing in particular in the scope of combinatorial optimization. There are available a few reconfigurable engines for such type of problems [12, 13, 14]. They are mainly based on the idea of instance-specific hardware, which assumes generation of a special configuration of FPGA for each individual problem. It permits to increase performance essentially and provides a good utilization of the resources. There are available a number of special-purpose compilers that accelerate the generation of the respective configurations for FPGAs. Nevertheless the required time of hardware compilation is still rather significant and as a rule it is much higher than the time for execution of combinatorial algorithms in hardware. Thus this method can be efficiently used only for tasks with a large volume of input data [12]. This paper proposes to consider a domain-specific approach based on a Reconfigurable Combinatorial Processor (RCP) that is going to be used for solving combinatorial tasks formulated over discrete matrices [15]. In this case for each combinatorial problem it is necessary to configure only basic computational operations and the respective control algorithms. The technique considered is based on so-called hardware templates that allow to utilize the same structures of execution and control units, which will not be changed from one task to another.

^{*} This work was sponsored by the grant FCT-PRAXIS XXI/BD/21353/99

The paper is divided into 7 sections. Section 1 is this introduction. Section 2 presents typical models and methods used in the field of combinatorial optimization. Section 3 gives a brief overview of possible strategies for combinatorial computations. Section 4 discusses the architecture of the proposed RCP. Experiments are presented in section 5. Section 6 contains some proposals for future work. The conclusion is in section 7.

2. Models and methods of combinatorial optimization

The problems of combinatorial optimization have to be solved in many application areas such as logic design, technical diagnostics, artificial intelligence, etc. [15, 16]. A distinctive feature of these tasks is that they require the execution of a large number of repeated operations over a given finite set of data. The simplest way to cope with a combinatorial problem is to check exhaustively all elements of this set, which will permit either to find out the solution or to conclude that there is no solution at all. However this method cannot be used for the majority of practical problems because it requires very long execution time. That is why it is necessary to apply some optimization techniques that enable us to reduce the number of the considered elements.

A widely used approach for solving combinatorial problems is based on decision tree [17]. The root of the tree is considered to be the starting point. The other nodes correspond to situations that can be reached during the search for results. Arcs of the tree represent steps of the algorithm that have to be performed. In order to speed up the discovering of the results various tree-pruning techniques are applied. They allow to reduce the number of considered situations. Usually the pruning is based on erasing repeated variants and on avoiding those feasible solutions whose cost became higher than the cost of any already found solution. The known method of improving the effectiveness of this approach is a reduction [15] and it permits to replace the current situation with some new simpler situation. It allows to reduce the number of computations that are required for analysis of situations caused by the current algorithmic step.

However the reduction is not possible for all existing situations. That is why another method is used, which relies on a divide-and-conquer strategy [18]. It permits to consider critical situations that have to be divided into several simpler situations such that each of them has to be examined. The objective is to find out a minimal number of such variants. Very often these new situations can be ordered with the aid of some criteria [15]. Considering such preliminary ordered variants increases essentially the effectiveness of computations.

It is not always possible to find out the optimal solution for a number of practical combinatorial problems in a reasonable time with the available computational resources. That is why for such purposes approximate algorithms are widely used [18]. They reduce the number of variants by eliminating less promising branches of the decision tree. However in this procedure the optimal result might be lost. The quality of results and the time of computation define the effectiveness of approximate methods.

There are many formal models that are used in order to describe combinatorial tasks. These are sets, graphs, discrete matrices, logic functions, etc. As a rule they are convertible one into another. We have selected discrete matrices as the primary model because they can be easily coded in software and in hardware. The paper [19] shows how different combinatorial problems can be formulated over discrete matrices.

3. Strategies of combinatorial computations

There are many different ways to increase the performance of computationally intensive algorithms. For example, it is possible to construct ASICs with hardwired control and specialized functional units that have been optimized for the considered problem. However ASIC-based solutions have a number of disadvantages. For example, they are absolutely inflexible because their functionality cannot be modified after fabrication. Besides they have very high development costs, which might be reasonable only in case of large volume production.

Another solution can be based on general-purpose processors (GPP). They have a fixed architecture and instruction set and different applications are programmed at the instruction level. An advantage of this technique is a high level of flexibility. Any change in the algorithm can be easily provided in the respective software (especially if the software has been constructed on the base of object-oriented technology). However for many time-consuming procedures the performance of GPP is not sufficient.

Implementation on the basis of reconfigurable hardware allows to avoid the disadvantages of “pure software” (such as GPP) and “pure hardware” (such as ASIC) realizations. Indeed, the speed of recent FPGAs is comparable with that of ASICs and they have the flexibility of software implementations. Note that typical clock rates on FPGAs are much lower than that of GPPs of similar technology. This is due to the need to build programmable interconnects. Thus a simple mapping of an application from software to FPGA does not necessarily provide a higher performance than GPPs. To achieve much higher performance the following techniques are usually employed. First, custom functional units are constructed in such a way that they can perform

in one or few clock cycles certain operations that may require more clock cycles on a GPP (this is especially true for bit-level operations) [20]. Second, the techniques of parallel processing and pipelining are utilized.

Within the domain of configurable computing, we can distinguish between two modes of configurability: static and dynamic [21]. Static reconfiguration assumes permanent functionality after the configuration has been loaded. In fact it does not provide much flexibility but permits to increase performance by the use of hardware optimized for a given application. Dynamic reconfiguration enables us to change functionality during execution time. This paper suggests to combine this technological facility with some architectural decisions based on so-called hardware templates (HT). The primary idea of HT is the following. It is necessary to construct a parameterizable computational unit in such a way that it is composed of components that have changeable and non-changeable functionality with fixed connections in between them. Customizing the unit has to be achieved by configuring the components with alterable functionality and the number of these components is as small as possible. It allows utilizing local reconfigurability and reduces essentially configuration overhead.

It should be noted that in general it is not feasible to realize practical combinatorial algorithms entirely in an FPGA. This is because the reconfigurable logic is not so well suited for some computations, for instance, for floating point arithmetic, which might be better realized in custom mathematical co-processors. Besides some fragments of combinatorial algorithms are rarely activated and we can predict for such fragments low effectiveness of FPGA-based solutions. So GPP is more appropriate for realizing non-regular computations. On the other hand FPGAs are more suitable for regular (repeated) treatments of a large volume of data [1]. Thus the best result might be achieved by rational combination of GPP and FPGA resources. Subroutines that can benefit from a hardware implementation are mapped to the FPGA, while others are computed by the GPP.

4. Design and implementation of RCP

On the basis of the analysis of mathematical models that are used in combinatorial optimization [19], primary operations and basic techniques for combinatorial algorithms over discrete matrices [24], the following requirements for the RCP have been formulated.

The RCP has to be built on FPGAs with distributed memory cells (RAM-based cells), such as XC4000 and Virtex families of Xilinx. Such cells grouped in blocks of required sizes can be used in order to keep matrices much like general-purpose registers of GPP store operands and results of intermediate computations. Each element of any discrete matrix is coded by 2-bits and 00 represents logic

0, 01 – 1, 10 – don't care and 11 indicates that the respective element is not used at all, for example it has been deleted. Due to heterogeneity of combinatorial tasks RCP must be dynamically reconfigurable. In other words we must provide run-time alterability of processor's functionality.

In order to reduce reconfiguration time RCP has to be based on HT. In this case the alterability might be provided only for basic computational operations and the respective control algorithms. All the other components and connections between them will not be changed. In order to do this we suggest to use RAM-based HT in such a way that customizing RCP functionality is made through reloading of RAM-based FPGA cells.

Fig. 1 depicts the structure of RCP that satisfies the considered above requirements. RCP in fig. 1 consists of two parts that are a Reconfigurable Control Unit (RCU) and a Reconfigurable Function Unit (RFU).

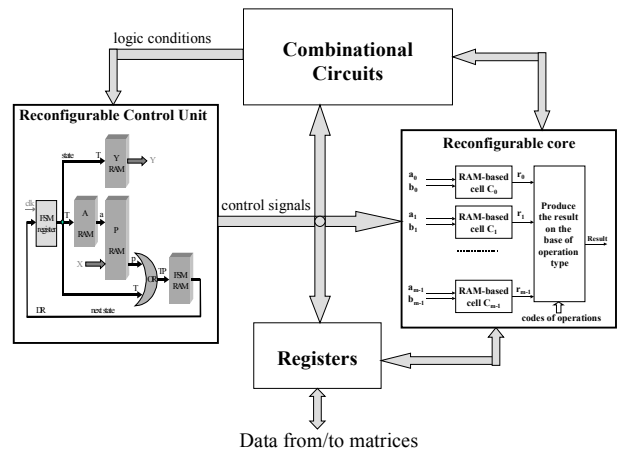


Fig. 1. The structure of RCP.

The RCU is modeled by a finite state machine (FSM) with dynamically modifiable behavior that generates the sequence of operations for the combinatorial algorithm being executed. An FSM might be presented at structural level as a composition of a combinatorial circuit, which calculates the next states and outputs, and a register that keeps the current state. We are considering so-called RAM-based FSM, i.e. such FSM for which the combinatorial circuit is constructed from RAM-based cells [22]. Any state code from the register is combined with input variables and the result forms an address of FSM RAM. Each address activates a word of the FSM RAM that contains the code of the respective next state and outputs (we are considering the Moore FSM model). In order to reduce size requirements of the RCU we employ a special fuzzy state encoding technique [22]. Thus the RCU is based on a parameterizable HT (constructed from a number of RAM-based cells) having some predefined constraints. These constraints restrict the

sizes of the respective RAMs and, consequently, specify the maximum number of FSM states, inputs, outputs and branches [22]. It is very easy to reprogram this device. For such purposes it is sufficient to reload the contents of the respective RAMs. We have developed special software tools [22] that allow to translate a given behavioral specification of control algorithms into the RAM contents assuming that the RCU is based on the predefined HT. The results of experiments [22] have shown that the proposed realization of RCU requires less area compared with circuits generated by Xilinx Foundation Software from the same specification.

The RFU is composed of memory elements and circuits needed to store and process the variables of the algorithm. Basic computations over columns and rows of discrete matrix are executed in the Reconfigurable Core (RC) shown in fig. 1. The core is composed of a number of RAM-based cells. Each cell performs an operation over one or two 2-bit values and calculates a 2-bit result. In order to implement different operations it is necessary to reload the respective group of RAM-based cells. Three groups of various operations have been proposed, which are Boolean operations, such as $\mathbf{a} \wedge \mathbf{b}$; operations that require an answer in form YES/NO, for example “test if \mathbf{a} ort \mathbf{b} is true”; and counting operations, such as calculating the number of zeros in a Boolean vector [23, 24].

The RCP has been implemented in FPGA XC4010XL and tested. Since this FPGA has very restricted resources it has been used only for verification purposes and for some experiments. We are currently working on the implementation of RCP on the base of ADM-XRC board [25] linked to PCI bus. This board contains one XCV812E Virtex Extended Memory FPGA with approximately 254K logic gates and additional block RAM [26]. Interaction with FPGA is carried out with the aid of ADM-XRC API library, which provides support for initialization, loading configuration bitstreams, data transfers, interrupt processing, clock management and error handling.

For configuring the FPGA the following model has been proposed. Basic functions of combinatorial algorithms (for instance, find-max-column, find-ort-row, find-column-covering, etc.) have to be included into a parameterized library. The system level specification has to be made with the aid of C++ programming language, i.e. a combinatorial algorithm has to be described in C++ using the library mentioned above. The assisting software tools extract the corresponding configuration from the library and download it to the FPGA when required. Note that basic HT is loaded from the very beginning and then it is necessary to reprogram only alterable components included into RCU and RFU. This essentially reduces the configuration overhead. Finally the matrix data will be passed to the FPGA. When the computation is completed the assisting software tools will store the intermediate

results and program execution will proceed, eventually reaching another hardware library function forcing similar actions, i.e. the considered above process will be repeated.

5. Experiments

Two main applications were used to test the RCP. They are satisfiability and covering problems. In the general case both are NP-complete.

5.1. Satisfiability problem

The satisfiability problem (SAT) permits to examine if a formula presented in conjunctive normal form (CNF) is satisfied by some truth assignment. A CNF consists of a conjunction of a number of clauses, where a clause is a disjunction of one or more variables or their negations. The SAT problem has great importance in the area of computer-aided hardware optimization.

For example the following formula contains 4 variables and 3 clauses, and is satisfied when $x_1=0$ $x_2=0$, $x_3=1$ and $x_4=1$:

$$(\bar{x}_1 \vee x_4)(x_3)(x_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

In order to solve this problem with the aid of RCP we have to formulate it first on a discrete matrix. Let us set a correspondence between variables and columns and between clauses and rows. Each element m_{ij} of the matrix is equal to:

- 00 – if variable x_j is included in clause c_i with negation;
- 01 – if variable x_j is included in clause c_i without negation;
- 10 – if variable x_j is not included in clause c_i ;
- 11 – if element m_{ij} is not used.

Taking into account these rules the matrix \mathbf{M} for the considered above formula can be presented in the following form:

$$\mathbf{M} = \begin{array}{cccc|c} & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & \mathbf{x}_4 & \\ \mathbf{c}_1 & \mathbf{00} & \mathbf{10} & \mathbf{10} & \mathbf{01} & \\ \mathbf{c}_2 & \mathbf{10} & \mathbf{10} & \mathbf{01} & \mathbf{10} & \\ \mathbf{c}_3 & \mathbf{01} & \mathbf{00} & \mathbf{00} & \mathbf{10} & \end{array}$$

We have implemented a deterministic solution to the SAT problem by checking every truth assignment. The respective algorithm has been described by the graph-scheme (GS) [27] depicted in fig.2.

The problem can be solved using the following sequence of operations. First we have to construct the matrix \mathbf{M} , load the control algorithm from fig. 2 into RCU and configure RFU for implementing the orthogonality checking operation.

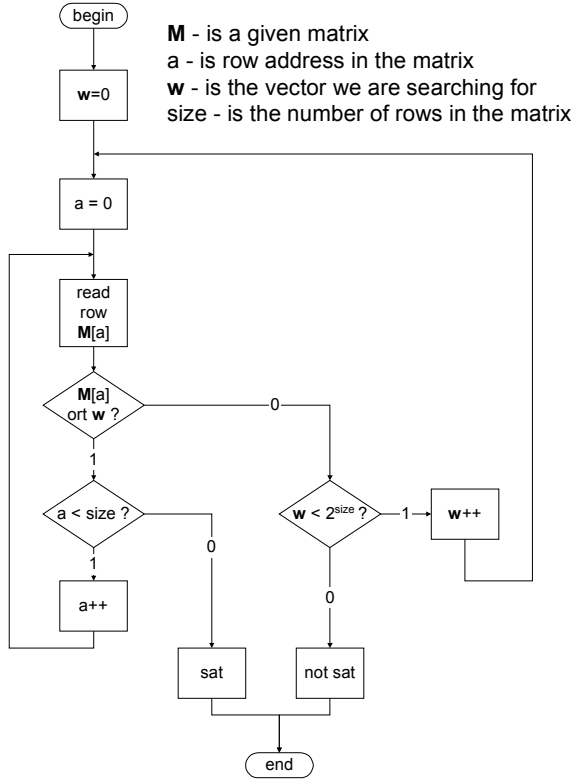


Fig. 2. GS of algorithm for solving satisfiability problem.

According to the algorithm we have to find a vector w that is orthogonal to all rows of the matrix M . If such vector cannot be found then the problem is unsatisfiable. In the opposite case the negated vector w gives the solution, i.e. all its elements having value 00 point to variables, which must be equal to 1, and all its elements with value 01 point to variables, which must be equal to 0.

The overall runtime for computing this problem in hardware includes in our case the time for configuring the matrix, RCU and RFU, and the actual hardware execution time.

5.2. Covering problem

Covering is a well-known combinatorial problem that has many useful practical applications. For a given set it requires to find out a subset having certain properties. The optimization variant of this problem assumes finding out a subset of minimal cardinality. For example, let us suppose that it is necessary to determine a minimum-size vertex cover of an undirected graph. It is known that a vertex cover of a graph $G=(V,E)$ is a subset $V' \subseteq V$ such that if $(u,v) \in E$, then $u \in V'$ or $v \in V'$ (or both) [28].

In order to solve this problem in the RCP we must formulate it again over a matrix. For this purpose let us

build the corresponding incidence matrix I , whose columns correspond to edges of the graph and rows correspond to vertices of the graph. Now in order to solve the problem we have to find out a cover of the matrix I , that is composed of the minimal number of rows having in conjunction at least one "1" in each column of I . The selected rows correspond to the vertices that must be included into minimal vertex cover. Let us consider an example. For the graph in fig. 3 the incidence matrix is presented in the following form:

	a	b	c	d	e	f	g	h	i	j	
$I =$	01	00	01	00	00	00	00	00	00	00	v_1
	01	01	00	01	01	01	00	00	00	00	v_2
	00	01	00	00	00	00	00	00	00	00	v_3
	00	00	01	01	00	00	00	01	00	00	v_4
	00	00	00	00	00	01	00	00	01	00	v_5
	00	00	00	00	00	00	00	01	00	01	v_6
	00	00	00	00	01	00	01	00	01	01	v_7
	00	00	00	00	00	00	01	00	00	00	v_8

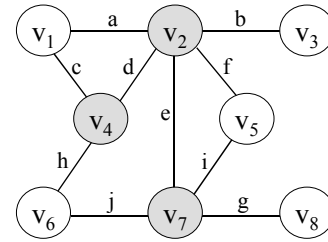


Fig. 3. Minimum size vertex cover of a graph.

In order to find out a minimal cover of the Boolean matrix we used an approximate algorithm described in fig. 4 [15]. It includes the following basic steps. First we have to construct the matrix I , load the control algorithm into RCU and configure RFU for implementing *count-number-of-ones* operation.

Applying the proposed algorithm to matrix I requires to choose first the column **a** and the row v_2 . Consequently we have to include the element v_2 to the constructing set $\{v_2\}$ and delete from the matrix I row v_2 and columns **a**, **b**, **d**, **e**, **f**. As a result matrix I will look like the following:

	c	g	h	i	j	
$I =$	01	00	00	00	00	v_1
	00	00	00	00	00	v_3
	01	00	01	00	00	v_4
	00	00	00	01	00	v_5
	00	00	01	00	01	v_6
	00	01	00	01	01	v_7
	00	01	00	00	00	v_8

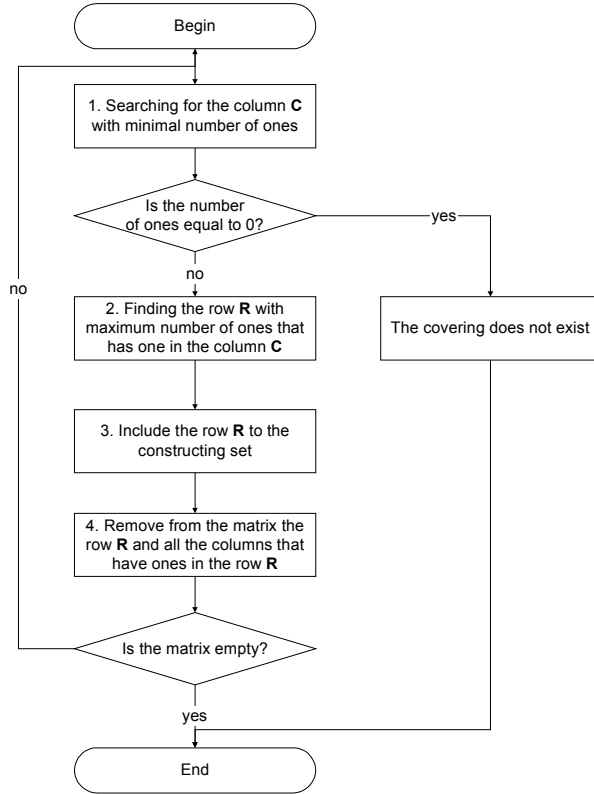


Fig. 4. An algorithm for finding the minimal row cover of a Boolean matrix.

Then according to the algorithm (see fig. 4) we have to choose the column c and the row v_4 , which will be included in the constructing set $\{v_2, v_4\}$. The row v_4 and the columns c and h will be deleted from the matrix I :

$$\mathbf{I} = \begin{array}{ccc|l}
 \mathbf{g} & \mathbf{i} & \mathbf{j} & \\
 \hline
 00 & 00 & 00 & v_1 \\
 00 & 00 & 00 & v_3 \\
 00 & 01 & 00 & v_5 \\
 00 & 00 & 01 & v_6 \\
 01 & 01 & 01 & v_7 \\
 01 & 00 & 00 & v_8
 \end{array}$$

At the next step we have to choose the column g and the row v_7 . As a result we get the set $\{v_2, v_4, v_7\}$ and the vectors v_7, g, i and j have to be deleted from the matrix I . The matrix becomes empty and we can conclude that we found out a solution that is $\{v_2, v_4, v_7\}$.

The discovered subset is lightly shaded in the graph in fig. 3 and in initial matrix I . In this example we have found a minimum-size vertex cover of the graph but in the general case the considered algorithm does not guarantee finding the best solution.

5.3. Performance results

Table 1 contains the results obtained when we solved the two combinatorial problems considered with the aid of software running under Windows 2000 on PentiumIII-800MHz/256MB RAM PC and in the RCP implemented in XC4010XL FPGA. The rows Cov1...Cov6 show some results for the covering problem, and the rows SAT1...SAT6 – for the satisfiability problem.

The first line in Table 1 presents an example of a trivial operation that counts the number of ones in a vector. This operation requires several clock cycles in a GPP. In RCP we can realize it using much less clock cycles. This results in a significant speedup compared to software implementation.

For the covering problem the speedups actually obtained are not very significant. As we can see from fig. 4 the primary operation of the considered algorithm is the count of ones in various columns and rows of the matrix. As we mentioned above this operation is executed in RCP much faster than in software. The matrix itself is implemented in FPGA by configuring an array of CLBs as a dual-port RAM. The advantage of this strategy is that on-chip RAM is extremely fast. To read any row of the matrix we need just one clock cycle. But on the other hand to read a column we have to access each row, extract from it some bits and finally compose the column from these bits. This operation requires a number of clock cycles and since it is performed quite frequently (see fig. 4), we guess that it does not permit to get more significant acceleration relative to the software implementation.

For the satisfiability problem we have obtained more impressive speedups. This is because the considered algorithm (see fig. 2) requires only sequential reading of different rows of the matrix and checking the respective vectors for orthogonality. Each of these operations needs just one clock cycle in RCP. In software the matrix has been constructed as an array of integers. Thus checking if the vector w (see fig. 2) is orthogonal to any row of the matrix requires many memory accesses and calculations.

6. Future work

For the considered examples (see section 5) each task was completely solved in FPGA. Note that the proposed RCP limits maximum dimensions of the matrix and that is why the RCP cannot be used for dealing with an arbitrary task. The instance-specific approach [12, 13, 14] assumes generation of a new hardware circuit for each problem. If the respective circuit cannot be implemented in a single FPGA it is possible to employ several FPGAs applying special methods of multi-FPGA partitioning. Nevertheless it is not guaranteed that a given task will be solved on available reconfigurable hardware resources.

Table 1. Experimental results with RCP implemented in XC4010XL FPGA.

Problem	Matrix/vector dimensions	Software execution time (in ms)	Hardware execution time (in ms)	Speedup
Count the number of ones in a vector	8	0.0059	0.00002487	237
Cov1	8 * 8	0.284	0.0164	17.32
Cov2		0.310515	0.0202	15.37
Cov3		0.217905	0.0118	18.47
Cov4		0.17684	0.00856	20.66
Cov5		0.292075	0.0231	12.64
Cov6		0.227125	0.00521	43.59
SAT1	8 * 8	4.54248	0.02971	152.89
SAT2		5.22301	0.01508	346.35
SAT3		0.24891	0.000588	423.32
SAT4		11.75764	0.07485	157.08
SAT5		2.72465	0.008059	338.09
SAT6		0.51543	0.002647	194.72

On the other hand for very small and simple problems the use of FPGAs becomes unreasonable because it takes a significant time for reconfiguration of FPGA and HT (in our case), generation hardware circuit (in instance-specific approach), etc. Taking into account this time we will lose all the advantages of fast hardware and the software will work much faster. For complex tasks the matrix dimensions are quite large.

Because of that we suggest to apply the following strategy. As we have already discussed in section 2, a common approach to solving combinatorial problems is based on a decision tree. When we construct the tree various splitting and reduction methods are used. It enables to decrease gradually the initial matrix dimensions (traversing one of the tree branches). The RCP is based on HT with predefined constraints, such as the maximum number of rows and columns of the matrix. Taking into account these constraints we assume that each combinatorial algorithm has to be implemented in software until the step in which all the constraints will be satisfied. After that step the RCP will be responsible for dealing with the problem.

As a result we can utilize the following technique. First of all, assisting software tools will configure RFU and RCU to execute the required algorithm. After that, if the initial matrix satisfies pre-given constraints then the matrix data will be transferred to FPGA and the problem will be completely solved in the RCP. Otherwise, the software will be trying to solve the problem. During this process it will apply special splitting and reduction methods until an intermediate matrix, which has to be constructed at the current search step, will not satisfy pre-given constraints. After that the RCP will be responsible for the subsequent steps. If reconfigurable hardware finds

out a solution, the problem is considered to be solved and the result will be dispatched to the host computer. On the other hand, if the considered branch of the tree does not allow to find a solution, the control will be returned to software. The software will continue to traverse the decision tree eventually reaching another point where matrix dimensions will satisfy the RCP constraints. Then matrix data will be transferred to FPGA and the RCP will try to solve the sub-problem. These steps will be repeated until we receive either negative or positive result, i.e. we will get the solution or we will conclude that the problem does not have any solution.

As a result the decision tree will be treated in software and in hardware in a way that is shown in fig. 5.

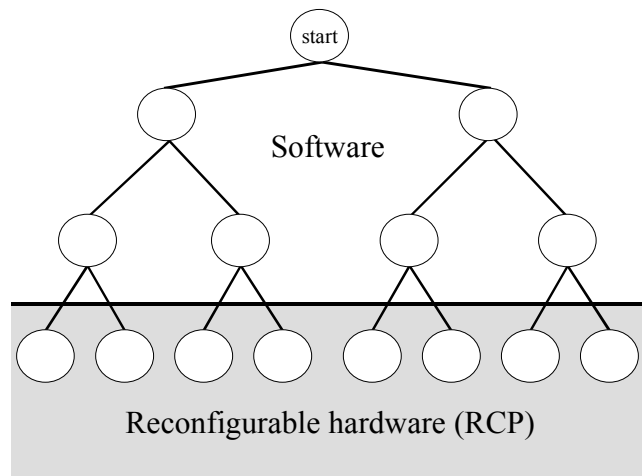


Fig. 5. Processing of the decision tree in software and in reconfigurable hardware.

7. Conclusion

The paper presents results of the design and implementation of reconfigurable processor for problems of combinatorial optimization. It proposes the structure of a reconfigurable combinatorial processor based on so-called hardware templates that are composed of non-alterable components and reprogrammable blocks. The latter include functional and control units with dynamically modifiable behavior. The advantages of RCP have been shown on experiments with two important combinatorial applications that are satisfiability and covering problems. Finally, we propose such model of computations that allows collaboration of software and reconfigurable hardware. We expect to provide further improvements on these results in future work aimed at architectural advances and implementation of the RCP on the ADM-XRC PCI board with a Virtex XCV812E FPGA.

8. References

- [1] S.Hauck, "The Roles of FPGAs in Reprogrammable Systems", Proceedings of the IEEE, vol. 86, No. 4, April 1998, pp.615-638.
- [2] J.Vuillemin, et al., "Programmable active memories: Reconfigurable systems come of age", IEEE transactions on VLSI Systems, vol.4, No. 1, March 1996, pp. 56-69.
- [3] D.A.Bell et al., "Splash 2 – FPGAs in a Custom Computing Machine", IEEE Computer Society Press, 1996.
- [4] P.M.Athanas, H.F.Silverman, "Processor Reconfiguration through Instruction-Set Metamorphosis", Computer, vol. 26, n°. 3, March 1993, pp. 11-17.
- [5] J.O.Haenni et al., "RENCO: A Reconfigurable Network Computer", Proc. 1998 ACM/SIGDA sixth international symposium on FPGA, February 22 - 25, 1998, Monterey, California, p.261.
- [6] C.Iseli, E.Sanchez, "Spyder: a Reconfigurable VLIW Processor Using FPGAs", Proc. Of the IEEE Workshop on FPGAs for Custom Computing Machines, April 1993, pp.17-24.
- [7] M.Shand, J.Vuillemin, "Fast implementation of RSA cryptography", 11th IEEE Symp. Comput. Arithmetic, Canada, 1993, pp. 252-259.
- [8] P.Graham, B.Nelson, "A Hardware Genetic Algorithm for the Traveling Salesman Problem on Splash 2", Proc. 5th International Workshop on Field Programmable Logic and Applications, August 1995, Oxford, England, pp. 352-361.
- [9] C.Dick, F.Harris, "Virtual signal processors", Microprocessors and Microsystems, 22, 1998, pp. 135-148.
- [10] P.Kolinummi, et al., "PARNEU: general-purpose partial tree computer", Microprocessors and Microsystems, 24, 2000, pp.23-42.
- [11] S.D.Haynes, et al., "Video Image Processing with the Sonic Architecture", IEEE Computer, April 2000, pp.50-57.
- [12] M.Platzner, G.Micheli, "Acceleration of Satisfiability Algorithms by Reconfigurable Hardware", Proc. 8th International Workshop on Field Programmable Logic and Applications – FPL'98, Tallin, Estonia, Springer-Verlag, 1998, pp.69-78.
- [13] P.Zhong et al., "Accelerating Boolean Satisfiability with Configurable Hardware", Proc. IEEE Symposium on FPGAs for Custom Computing Machines - FCCM98, April 1998, pp. 186-195.
- [14] M.Abramovici, J.T.de Sousa, "A SAT Solver Using Reconfigurable Hardware and Virtual Logic", Journal of Automated Reasoning, February 2000.
- [15] A.D.Zakrevski, "Logical Synthesis of Cascade Networks", Moscow: Nauka, 1981, (in Russian).
- [16] G.Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, Inc., 1994.
- [17] D.L.Kreher, D.R.Stinson, "Combinatorial Algorithms: Generation, Enumeration, and Search", CRC Press, 1999.
- [18] Z.Michalewicz, D.B.Fogel, "How to Solve It: Modern Heuristics", Springer-Verlag, 2000.
- [19] I.Skliarova, A.B.Ferrari, "Modelos matemáticos e problemas de optimização combinatoria", Electrónica e Telecomunicações, vol.3, N°3, January 2001, pp. 202-208 (in Portuguese).
- [20] D.Abramson et al., "FPGA Based Custom Computing Machines for Irregular Problems", Fourth International Symposium on High-Performance Computer Architecture, (HPCA98), February 1-4, 1998, Las Vegas, Nevada.
- [21] E.Sanchez, et al, "Static and Dynamic Configurable Systems", IEEE transactions on Computers, vol. 48, No. 6, June 1999, pp.556-564.
- [22] I.Skliarova, A.B.Ferrari, "Synthesis of reprogrammable control unit for combinatorial processor", Proc. of the 4th Int. Workshop on IEEE Design and Diagnostics of Electronic Circuits and Systems - DDECS 2001, Gyor, Hungary, April 2001, pp. 179-186.
- [23] I.Skliarova, A.B.Ferrari, "Exploiting FPGA-based Architectures and Design Tools for Problems of Reconfigurable Computations", Proc. SBCCI 2000 XIII Symposium on Integrated Circuits and System Design, Brazil, Sept. 2000, pp. 347-352.
- [24] I.Skliarova, A.B.Ferrari, "Development tools for problems of combinatorial optimization", Proc. 4th Portuguese Conference on Automatic Control (CONTROLO'2000), Portugal, Oct. 2000, pp. 552-557.
- [25] <http://www.alphadata.co.uk>
- [26] Xilinx, "The programmable Logic Data Book", Xilinx, San Jose, 2000.
- [27] S.Baranov, "Logic Synthesis for Control Automata", Kluwer Academic Publishers, 1994.
- [28] T.H.Cormat et al., "Introduction to Algorithms", McGraw-Hill, 1997.