Fast Matrix Covering in All Programmable Systems-on-Chip

V. Sklyarov[1], I. Skliarova[1], A. Rjabov[2], A. Sudnitson[2]
[1]*Department of Electronics, Telecommunications and Informatics/IEETA, University of Aveiro, 3810-193 Aveiro, Portugal*
[2]*Department of Computer Engineering, Tallinn University of Technology, 12617 Tallinn, Estonia*
skl@ua.pt

[1]***Abstract*—The paper suggests a technique for solving the matrix/set covering problem in all programmable systems-on-chip. A novel very fast hardware accelerator is proposed and implemented in the programmable logic (PL) of a Xilinx Zynq microchip. The accelerator is managed by software running in the processing system (ARM Cortex-A9) available on the same microchip and communicating with the PL through high-speed interfaces. The results of implementation, experiments, and comparisons demonstrate significant speedup comparing to software running in general-purpose PC and in the ARM.**

***Index Terms*—Accelerator architectures, concurrent computing, parallel processing, field programmable gate arrays, system-on-chip.**

I. INTRODUCTION

Combinatorial search algorithms are frequently involved to solve optimization problems. Examples are matrix/set covering, the Boolean satisfiability, graph coloring and many others described and reviewed in [1]–[4]. Many tasks are NP-complete and, thus, they are time consuming. We consider here the matrix/set covering which belongs to partitioning problems [1] arising in such practical applications as scheduling aircrafts, location emergency stations in urban areas, fault testing of electronic circuits, resource distribution in multi-core systems, and many others [1]. For many applications high performance is required and it may be achieved in hardware accelerators for which FPGA-based solutions are especially promising. It is shown and proved in the paper that recently appeared on the market all programmable systems-on-chip (APSoC) of Xilinx Zynq family [5] are very appropriate for implementation of combinatorial search algorithms enabling the problem to be decomposed into two sub-problems that are 1) higher-level activation of primary sub-tasks in which the algorithm has been decomposed, and 2) fast execution of the sub-tasks in the hardware accelerator. According to the proposals, the first sub-problem is assigned to a processing system (PS) implemented on the basis of industry-standard dual-core ARM Cortex-A9 in Zynq APSoC. The acceleration is done in a programmable logic - PL (Xilinx Artix-7 FPGA) that is available on the same microchip with the ARM. It is shown that such type of hardware/software co-design permits elegant and efficient solutions to be found that are faster than the best known alternatives.

The remainder of the paper is organized in six sections. Section II defines the problem and presents an example. Section III suggests architecture of the hardware accelerator. Section IV is dedicated to software/hardware co-design. Experimental setup is discussed in Section V. The results and comparisons are reported in section VI. The conclusion is given in Section VII.

Manuscript received October 28, 2013; accepted January 6, 2014.

This research was supported by EU through European Regional Development Funds, by the institutional research funding IUT 19-1 of the Estonian Ministry of Education and Research, ESF grant 9251, and by Portuguese National Funds through FCT - Foundation for Science and Technology, in the context of the project PEst-OE/EEI/UI0127/2014.

II. PROBLEM DEFINITION

The covering problem can identically be formulated on either sets [1], [2] or matrices [1]. Let $\mathbf{A} = (a_{ij})$ be a 0-1 incidence matrix. The sub-set $A_i = \{j \mid a_{ij} = 1\}$ contains all columns covered by row i (*i.e.* the row i has value 1 in all columns of the sub-set A_i). The minimal row cover is composed of the minimal number of the sub-sets $\mathbf{A}_i$ that cover all the matrix columns. Clearly, for such sub-sets there is at least one value 1 in each column of the matrix. Let us consider an example from [2] of a set S and sub-sets $S_1,\ldots,S_6$ (Fig. 1), which can be represented in the form of the following matrix **A**:

	1	2	3	4	5	6	7	8	9	10	11	12
S_1:	1	1	0	0	1	1	0	0	1	1	0	0
S_2:	0	0	0	0	0	1	1	0	0	1	1	0
S_3:	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
S_4:	*0*	*0*	*1*	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
S_5:	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
S_6:	0	0	0	1	0	0	0	1	0	0	0	0

Different algorithms have been proposed to solve the covering problem [1]–[3], such as greedy heuristic [1], [2] and a very similar method [3]. An analysis of the known algorithms has shown the following:

1. The majority of them are approximate since the problem is NP-complete;

2. The algorithms are very similar and they differ insignificantly;
3. The algorithms may equally be applied to either sets or matrices and any of such models can be chosen because they are directly convertible to each other.

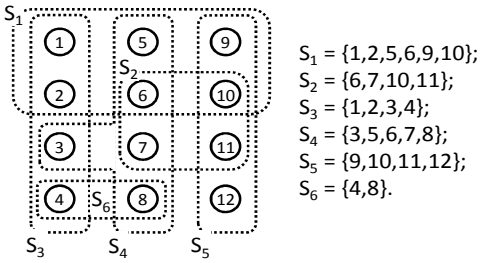

Fig. 1. An example of a set S with sub-sets S_1,...,S_6 from [2].

We consider below a slightly modified method from [3] that is applied to binary matrices exemplified above and the matrix from Fig. 1 [2] will be used to illustrate the steps of the chosen method that are the following:

1. Finding the column C_{min} with the minimum Hamming weight (HW) that is the number of ones. If there are many columns with the same (minimum) HW, selecting such one for which the maximum row is larger, where the maximum row contains 1 in the considered column and the maximum number of ones;
2. If HW = 0 then the desired covering does not exist, otherwise from the set of rows containing ones in the column C_{min} finding and including in the covering the row R_{max} with the maximum HW;
3. Removing the row R_{max} and all the columns from the matrix that contain ones in the row R_{max}. If there are no columns then the covering is found otherwise go to the step 1.

Let us apply the step 1–3 to the matrix **A** above:

1. The column 12 is chosen;
2. The row S_5 is included in the covering;
3. The row S_5 and the columns 9, 10, 11, 12 are removed from the matrix.
1. The remaining columns contain the following number of ones: 2, 2, 2, 2, 2, 3, 2, 2. The column 3 is chosen because for this column the row S_4 has the maximum HW equal to 5;
2. The row S_4 is chosen and included in the covering;
3. The row S_4 and the columns 3, 5, 6, 7, 8 are removed from the matrix.
1. The remaining matrix contains rows S_1, S_2, S_3, S_6 and columns 1, 2, 4 with the following HWs: 2, 2, 2. The column 1 is chosen;
2. The row S_3 is chosen and included in the covering;
3. After removing the row S_3 the covering is found and it includes the rows S_3,S_4,S_5 shown in italic font in the matrix above. The minimum covering is the same as in [2] that was found with a different algorithm.

III. ARCHITECTURE OF HARDWARE ACCELERATOR

This section presents the proposed architecture of the hardware accelerator executing the steps 1 and 2 from Section II. We suggest the given matrix to be unrolled in such a way that all its rows and columns are saved in the PL registers. Note that more than a hundred of thousands of such registers are available in the recent low-cost FPGAs. This technique permits all rows and columns to be accessed and processed in parallel.

Figure 2 demonstrates the unrolled matrix **A** shown above in Section II (and repeated in Fig. 2 for convenience). HW counters compute HW for all the rows/columns in parallel using combinational circuits, such as that are proposed in [6], [7]. These circuits are very fast allowing HWs to be computed in less than 20 ns even in low-cost FPGAs.

The MIN column and MAX row circuits permit to find out the minimal column C_{min} and the maximum row R_{max}. It is shown in [8] that these circuits can be built as MAX-MIN fully combinational networks producing the results faster than in 20 ns. Since all the circuits (computing HW and the maximum/minimum values) are functioning in parallel, the steps 1 and 2 may be completed faster than in 20 + 20 = 40 ns even in low-cost FPGAs. So, a very significant acceleration can be expected.

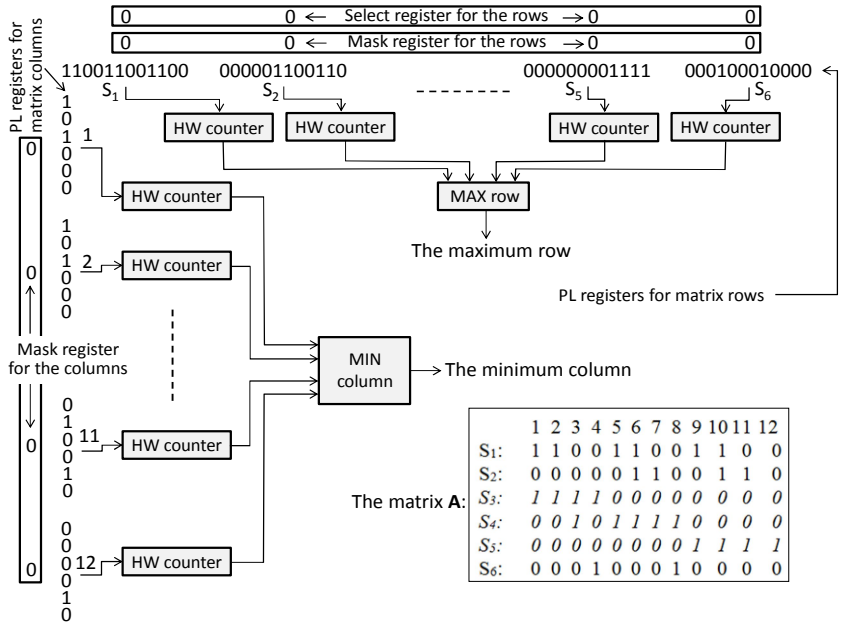

Fig. 2. Architecture of the proposed hardware accelerator on an example of unrolled matrix A from Section II.

In accordance with the proposals, the matrix is unrolled only once and any reduced matrix is formed by masking previously selected rows and columns. One select register and two mask registers (one for rows and another one for columns) shown in Fig. 2 are additionally allocated in the PL. The select register is zero-filled at the beginning of the step 1 and after the step 1 it indicates by values 1 those rows that have to be chosen by the selected column (i.e. such rows have values 1 in the selected column). The mask registers are filled in with zeroes at the beginning of the algorithm and they mask (by the values 1) those rows and columns that have been removed from the matrix in each iteration. For example, the select register contains the value 000010 after the first step in the example of Section II. The mask registers after the first iteration in the example are set to 000000001111 for the columns and 000010 for the rows. After the second iteration they are updated as 001011111111 for the columns and 000110 for the rows.

IV. SOFTWARE/HARDWARE CO-DESIGN

Figure 3 presents the proposed partitioning in software and hardware modules (assuming implementation in Zynq APSoC) of the considered algorithm that enables the minimal covering to be found.

Software in the PS is responsible for the following steps:

1. Getting from a host computer or generating the matrix, unrolling it, and saving in external DDR memory as a set of rows and a set of columns;
2. As soon as C_{min} is found, the PL generates an interrupt of type *a*. The PS receives the C_{min} and sets the select register in the PL through general-purpose ports [5];
3. As soon as R_{max} is found, the PL generates an interrupt of type *b*. The PS receives the R_{max} and sets the mask registers in the PL through general-purpose ports [5];
4. At any iteration it is checked if the solution is found or if it does not exist. If the solution is found it is indicated by the PS or transmitted to the host computer and the algorithm is completed.

Hardware in the PL implements the architecture in Fig. 2 and is responsible for the following steps:

1. Getting the unrolled matrix from external DDR through high-performance Advanced eXtensible Interface (AXI) [5] and saving the rows and columns in slice registers as it is shown in Fig. 2.
2. Getting from the PS select/mask vectors and setting/updating the select and the mask registers.
3. Finding out the value C_{min} at each iteration and as soon as the value of C_{min} is ready, generating an interrupt of type *a*.
4. Finding out the value R_{max} at any iteration and as soon as the value of R_{max} is ready, generating an interrupt of type *b*.

V. EXPERIMENTAL SETUP

Implementation was done in the Xilinx Zynq-7000 APSoC ZC702 evaluation kit [9] containing a microchip (APSoC) Zynq xc7z020. The PS is the dual-core ARM Cortex-A9 and the PL is Artix-7 FPGA from the 7th series of Xilinx. Currently only AXI, general-purpose ports and interrupts have been used (from 16 available interrupts we selected only two assigned above as type *a* and *b*). Software for the ARM was developed in C language and hardware for the PL was synthesized from specification in VHDL. Computing HW was done in LUT-based circuits from [7] that are very economical and fast. Experiments were done with two types of matrices 32×32 and 64×64. Thus, either $32 + 32 = 64$ or $64 + 64 = 128$ HW counters have been implemented in the PL section and all these circuits can run in parallel. Since C_{min} and R_{max} are found at different steps in the current implementation, only half of the HW counters work in parallel enabling either the minimal column C_{min} or the maximal row R_{max} to be found.

Because the occupied resources are indeed very small [7], we implemented all the required HW counters assuming that in future improvements all of them might function in parallel. The MIN and MAX circuits are built as combinational networks and they are described in detail in [8].

Figure 4(a) presents such a circuit for a matrix 32×32 for which the number of bits in any HW is 6 (because the maximum number of ones in a 32-bit vector is 32 that can be represented by a 6-bit code).

A particular (simplified) example for only 6 input items 3, 14, 21, 11, 14, 27 is given in Fig. 4(b). The maximum value (27) is found in a combinational circuit with only 3 gate level delays. Clearly, there is 5 gate level delay for matrices 32×32 and 6 gate level delay for matrices 64×64.

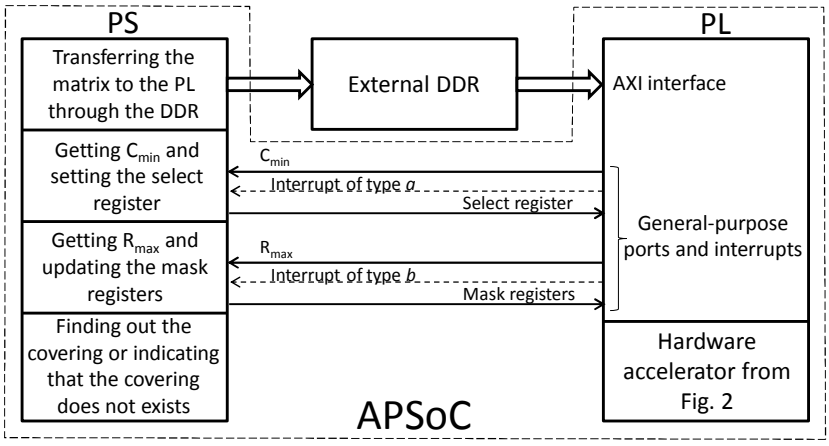

Fig. 3. Partitioning of the algorithm in software and hardware modules.

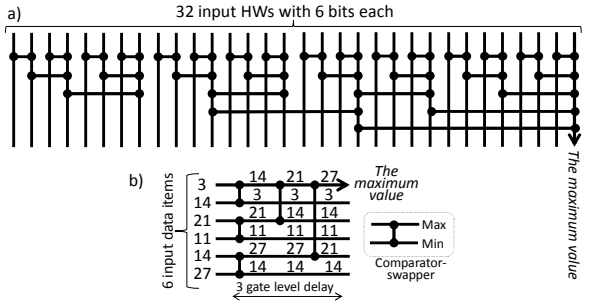

Fig. 4. MAX circuit from [8] for 32 × 32 matrix (a); an example (b).

VI. RESULTS AND COMPARISONS

We compared three different implementations in which the covering algorithm is either:

1. Described in C language program running in PC with Intel i7 2.66 GHz processor;
2. Described in C language program running in ARM Cortex-A9 (for evaluation kit [9]);
3. Implemented in the PS and in the PL of Zynq-7000 APSoC as it is shown in Fig. 3 (for evaluation kit [9]).

Initial matrices have been generated randomly using the C *rand* function and identically for all the described above implementations. The number of instances (examples) was chosen to be 100,000.

In the last case (see the point 3 above) that is the original contribution of the paper, the following results have been obtained:

1. Generating in the PS and transmitting the matrices from the PS to the PL requires about 31 μs for 32 rows and 32 columns and about 34 μs for 64 rows and 64 columns. Only one AXI 32-bit (for the matrices 32 × 32) or 64-bit (for the matrices 64 × 64) port from the 4 available ports has been used. Clearly, additional ports permit the indicated time to be reduced;
2. Each iteration in the PL is executed in about 28 ns for the matrices 64 × 64 and about 24 ns for the matrices 32 × 32;
3. Communications between the PS and the PL (through interrupts and general-purpose ports) at any iteration of the algorithm require negligible time comparing to other operations.

The covering is found significantly faster than in software. The acceleration comparing with the PS only (see point 2 above) is from 30 to 50 times and comparing with the PC (see point 1 above) is from 5 to 10 times. This is because operations of the covering algorithm in software require many cycles and frequent transmission of data between processors and memories. For example, if we consider 64 × 64 matrices then a single matrix transfer from the PS to the DDR takes 33,300 ns on average and this is the most time consuming operation. Data transfer from the DDR to the PL is done in 284 ns on average. Once the PL receives the matrix data, no more interaction with the DDR is required for further processing.

For future work we will use accelerator coherency ports available for Zynq microchips and allowing data exchange directly with the processor cache memory [10], [11]. Besides, an additional optimization technique will be provided looking for the better distribution of different sub-tasks between the PS and the PL.

VII. CONCLUSIONS

The paper presents a novel technique for implementation of matrix/set covering algorithms in hardware and software of recent all programmable systems-on-chip. A new method that permits the known approximate algorithm to be executed over suggested unrolled matrices is discussed and the relevant hardware accelerator is developed. It is shown that the covering algorithm can efficiently be partitioned in software and hardware modules that finally have been completely implemented and tested in Xilinx Zynq microchips. The results of experiments and comparisons with two different software implementations demonstrate significant speedup which is very important for various practical applications that are also mentioned in the paper.

REFERENCES

[1] K. H. Rosen, J. G. Michaels, J. L. Gross, J. W. Grossman, D. R. Shier, *Handbook of Discrete and Combinatorial Mathematics*. Boca Raton, FL: CRC Press, p. 1232, 2000.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*. USA: MIT Press, p. 1312, 2009.

[3] A. Zakrevskij, Y. Pottosin, L. Cheremisiniva. *Combinatorial Algorithms of Discrete Mathematics*. Tallinn: TUT Press, p. 193, 2008.

[4] I. Skliarova, A. B. Ferrari, "Reconfigurable hardware SAT solvers: a survey of systems", *IEEE Trans. Computers*, vol. 53, no. 11, pp. 1449–1461, 2004. [Online]. Available: http://dx.doi.org/10.1109/TC.2004.102

[5] *Zynq-7000 All Programmable SoC First Generation Architecture*, Xilinx Inc., USA, 2012. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds188-XA-Zynq-7000-Overview.pdf

[6] V. Sklyarov, I. Skliarova, "Design and implementation of counting networks", *Computing*, 2013. [Online]. Available: http://dx.doi.org/10.1007/s00607-013-0360-y

[7] V. Sklyarov, I. Skliarova, "Digital Hamming weight and distance analyzers for binary vectors and matrices", *Int. Journal of Innovative Computing, Information and Control*, vol. 9, no. 12, pp. 4825–4849, 2013. [Online]. Available: http://www.ijicic.org/ijicic-12-12021.pdf

[8] V. Sklyarov, I. Skliarova, "Fast regular circuits for network-based parallel data processing", *Advances in Electrical and Computer Engineering*, vol. 13, no. 4, pp. 47–50, 2013. [Online]. Available: http://dx.doi.org/10.4316/AECE.2013.04008

[9] *Zynq-7000 EPP ZC702 evaluation kit*, Xilinx Inc., USA, 2014. [Online]. Available: http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm

[10] M. Sadri, C. Weis, N. When, L. Benini, "Energy and performance exploration of accelerator coherency port using Xilinx ZYNQ", *Proc. 10th FPGAWorld Conf.*, Stockholm, Sweden, 2013. [Online]. Available: http://dx.doi.org/10.1145/2513683.2513688

[11] D. Mihhailov, A. Sudnitson, V. Sklyarov, I. Skliarova, "Acceleration of Recursive Data Sorting over Tree-based Structures", *Elektronika ir Elektrotechnika*, no. 7, pp. 51–56, 2011. [Online]. Available: http://dx.doi.org/10.5755/j01.eee.113.7.612