Proceedings of Engineering & Technology (PET)
pp. 24-28

Optimization Algorithms for Hardware/Software Partitioning

Sonia Dimassi, Mehdi Jemai, Bouraoui Ouni and Abdellatif Mtibaa

***Abstract*—This paper presents a comparative study between two algorithms of hardware/software partitioning which aim to minimize the logic area of System on a Programmable Chip (SOPC) while respecting a time constraint. The first algorithm is based on the genetic algorithm (AG), the second one is our proposed algorithm which is based on the principle of Binary Search Trees (BST) and genetic algorithms (AG). The two algorithms aim to define the tasks that will run on the Hardware (HW) part and those that will run on the Software (SW) part. They seek to find the efficient hardware/software partition that minimize the number of tasks used by the HW and increase the number of tasks used by the SW, in order to balance all the design parameters and have a better trade-off between the logic area of the application and its execution time.**

***Keywords*—SOPC; Hardware/software partitioning; Genetic algorithms; Binary search trees**

I. INTRODUCTION

Systems on a Programmable Chip (SOPC) are increasingly used in embedded applications. They include multiple functions such as one or more processors, one or more reconfigurable areas, a signal processor DSP (Digital Signal Processor), various peripherals and memory or analog parts. In fact they have small size and their costs are reduced compared to the various circuits which used for performing the same function. Many hardware and software techniques seek to minimize a criterion (or set of criteria) given, as the surface, the execution time and consumption for SOPC. Thus, in this paper, we present two hardware-software partitioning algorithms to minimize the logic area. The proposed algorithm [1] incorporates the binary search trees into genetic algorithm to improve the complexity and the run time of the original genetic algorithm. The purpose of these algorithms is to find the efficient hardware/software partition that minimize the tasks used by the HW and increase the number of tasks used by the SW, which aims to minimize the area. In fact the implementation of a software module requires more flexibility and less cost, but more executing time, while the hardware is faster but it is more expensive and requires less time. So, we can implement performance critical components in hardware and noncritical components in software. This kind of hardware/software partitioning can lead to a good tradeoff between system performance [2] and power consumption [3]. One of the key challenges in embedded system design is how to find an efficient hardware/software partition.

Sonia Dimassi e-mail: sdimassi@yahoo.com
Mehdi Jemai e-mail: jmehdie@gmail.com
Bouraoui Ouni e-mail: ouni_bouraoui@yahoo.fr,
Abdellatif Mtibaa e-mail: abdellatif.mtibaa@enim.rnu.tn

Laboratory of Electronic and Microelectronic, University of Monastir, Monsatir 5000, Tunisia.

This paper is structured in five parts. After the introduction, we give an overview of the related work; in the third section, we present the principle of the genetic algorithm, the binary search trees and our proposed algorithm. The fourth section shows the experiments and their results. Finally, we end up with a conclusion.

II. RELATED WORK

In the early 1990s, a new technique appeared for the design of integrated circuits and systems, it was the co-design. The software/hardware co-design became necessary to meet the requirements of the embedded systems market. In fact, the emergence of multimedia systems resulted in a greater complexity of the electronics and economic competition requires a shorter design time. In addition, the large variety of technological targets motivates the designer to explore all possible solutions to have an efficient software/hardware co-design.

Traditionally, partitioning is carried out manually. The target systems are usually presented as a task graph, which describe the dependencies among the components of embedded system. Today, several factors led to the need for co-design, such as the complexity of the structure of modern embedded systems, the requirements on cost, power, and timing performance. Many approaches have addressed the problem of software/hardware partitioning [4], [5], [6], [7], [8] and they tried to find optimization methods to automate the task of partitioning. In this context, we quote the exact algorithms such as branch-and bound [9], integer linear programming [10] and dynamic programming [11]. Those algorithms have been used for partitioning problem with small inputs successfully. Most problems of partitioning are NP hard [12], this is why the exact algorithms may not be suitable for large systems, because they are quite slow. To overcome the drawback of these algorithms, researchers are moving towards to more flexible and efficient heuristic algorithms. Among these heuristic algorithms, we quote simulated annealing related algorithms [13], genetic algorithms [14], tabu search and greedy algorithms [15]. They have been extensively used to solve partitioning problem. Some custom heuristics, such as expert system [16] and GCLP algorithm [17] are also appropriate for hardware-software partitioning problem.

In this paper, we have proposed a comparative study between two algorithms of hardware/software partitioning which aim to minimize the logic area of SOPC while respecting a time constraint. The two algorithms are based on the genetic algorithms.

ISSN: 2356-5608

III. BACKGROUND

A. Genetic algorithms (AG)

Genetic algorithms are a useful optimization method in the nonlinear case. They mimic the process of natural evolution. The basic principles of genetic algorithms were fixed by Holland [18]. They simulate the survival-of-the-fitness principle of nature. It stated that the most likely individuals to survive (" best") reproduce more often and will have more descendants. Thus, the quality of the gene pool of the population will be increased, the most effective genes become more frequent and the population improves. By the same principle, a genetic algorithm starts from a population of initial solutions, makes them breed (the best solutions are more likely to reproduce), creating a new generation of solutions. By repeating this cycle several times, we obtain a population of best solutions. Genetic algorithms are generally used to find a solution, the best solution after a certain number of generations.

This is the main iteration body of a genetic algorithm:
1. Evaluate the quality (fitness) of individuals and their chances of survival.
2. Select individuals for reproduction.
3. Perform reproduction.
4. Replace the old population with the new population.

This iteration is repeated as many times as required. The evaluation of the quality (fitness) of an individual can illustrate with a numerical value, the quality of the genes that make up the individual. More the quality of an individual is higher, more it will have chance to be selected for reproduction. The reproduction is made by crossing two individuals. Indeed, we applied generic operators to the two selected individuals, usually cross-over and mutation. The reproduction provides two children (offspring) that are placed in the new population. Reproduction is repeated until we have completed the new population (the population size should remain constant). Then, we replace the old population by the new, and the process is repeated according to the needed number of generations. Finally, the algorithm will return the best individual of the latest generation as the solution of the problem.

B. Binary search trees (BST)

The trees are mainly the data structure used to store ordered data. They are the largest non-linear structure involved in the computer science. This structure can be adapted to the natural representation of organized and homogeneous information, and it has a great speed and a handling convenience. The trees are used in many computing areas, such as compilation (syntax trees to represent expressions or possible productions of language), imaging (quaternary trees), algorithmic (for example it is the support of sorting methods or management information in tables), or in the fields of artificial intelligence (game trees, decision trees, resolution trees).

The binary trees are used to storage and retrieve information. They are interesting because they optimize the access time to information. Our purpose behind using Binary Search Trees (BST) is to reduce our search space and to have an optimized data access time. In computer science, a BST, sometimes called an ordered or sorted binary tree, it is a node-based binary tree data structure which has the following properties:
1. Hierarchical data structure with a single reference to root node.
2. Each node has at most two child nodes (a left and a right child).
3. Nodes are organized by the Binary Search Property:
   - Every node is ordered by some key data field(s)
   - For every node in the tree, its key is greater than its left child's key and less than its right child's key.

Otherwise, the label of each node is greater than any node in its left sub-tree and less than each node of the right sub-tree.

C-Proposed algorithm

As mentioned, the proposed algorithm is based on the genetic algorithm and the binary search trees. To reduce the logic area on SOPC, it assigns the small modules to the Left Sub-Tree (LST) and large modules to the Right Sub-Tree (RST). In this way, we will have a hardware/software partitioning that reduce the area, in fact, the left sub-tree is assigned to the hardware part and the right sub-tree is assigned to the software part of architecture. To improve the hardware/software partitioning obtained, the genetic algorithm will be applied on the left sub-tree or on the right sub-tree according to the time constraint, instead of performing the search in the whole binary tree. The proposed algorithm aim to find the tasks that will migrate from the software part to the hardware part of architecture or to the contrary, to get a best hardware/software partitioning. The detail of our proposed algorithm is shown in Fig.1.

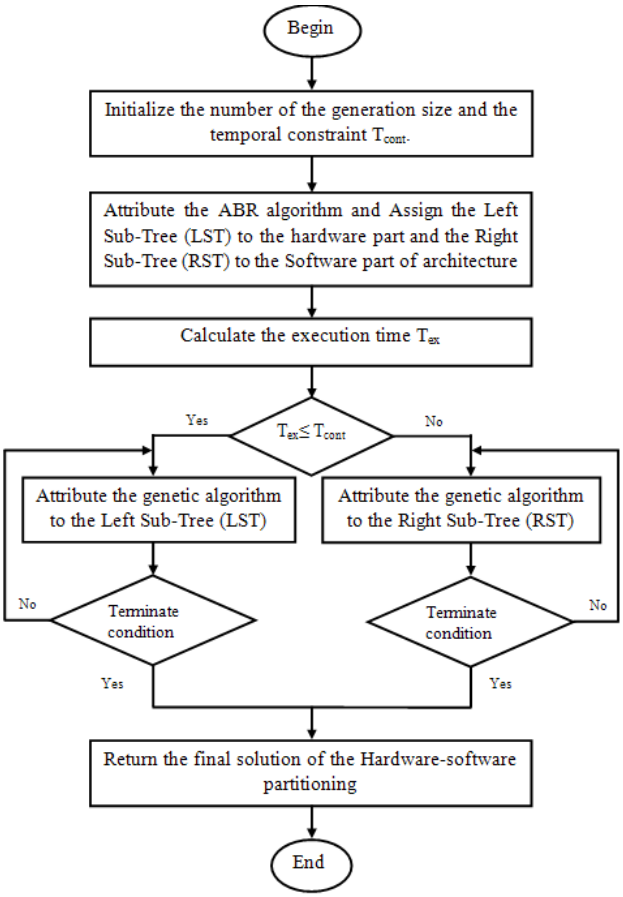

Figure 1. Main procedures for the proposed algorithm

IV. Experiments and Results

To confirm our approach, we have implemented the 16-DCT task graph on FPGA Xilinx Virtex®-5. The Xilinx Virtex®5 development kit enables a high performance embedded design in Xilinx FPGAs.

In our approach, the software resource is the PowerPC and the hardware resources are configurable logic blocs (CLBs). Hence, to compute the parameters of each node and to access to the PowerPC, we have used Xilinx ISE tool and Xilinx EDK tool. These Xilinx design tools provide resources and timing report incorporates timing delay and resources to provide a comprehensive area and timing summary of the design. Our algorithm has been written in JAVA language and executed under Windows-7 on Acer-PC (Intel Core 2 Duo T5500; 1.66 GHz; 1GB of RAM). In order to demonstrate the effectiveness of the proposed algorithm compared to genetic algorithm. The simulation results are presented in table 1.

TABLE 1: DESIGN RESULTS

Algorithm	Run time (ms)	Latency (ns)	Area (Slice)
Proposed algorithm	766	2664	2202
Genetic algorithm	40375	2928	2274

To evaluate the design results shown in table 1, we have introduced the following metric **α**

$$\alpha = \frac{L}{T_{Area} - G_{Area}} \qquad (1)$$

$\mathbf{T_{Area}}$: all nodes of the graph are implemented to the hardware part of the architecture.

$\mathbf{G_{Area}}$: the logic area consumed by the graph

L: the whole latency of the graph

Therefore, based on the above equation, a partitioning algorithm is classified to be good if it decreases the value of **α**.

TABLE 2: DESIGN RESULTS

	Proposed algorithm	Genetic algorithm
α	1.153	1.308

The above design results in table 2 show that our algorithm is the best one in terms of **α** value. Indeed, our algorithm provides a gain reaching 11.85 % compared to the Genetic algorithm.

Our proposed algorithm has reduced the search space to m nodes (m, is the number of nodes in the right sub-tree or the left sub-tree) according to the execution time of the application, while the genetic algorithm uses all the nodes of the studied graph (n nodes). Indeed, the number of reduced nodes leads to a reduction of execution time of the algorithm. In fact, our algorithm is faster 52 times than the genetic algorithm

V. Conclusion

Contrary to a large number of optimization methods, the genetic algorithms operate on a population of potential solutions allowing it to explore several areas of space configurations at the same time and they avoid focusing on a local extremum. Based on these characteristics, we proposed an algorithm that incorporates the binary search trees into genetic algorithm to address the problem of software/hardware partitioning, in order to minimize the logic area of System on a Programmable. In this paper we have made a comparative study between our proposed algorithm and the genetic algorithm. Compared to the genetic algorithm, the proposed algorithm reduces the search space, in fact, instead of performing the search in the whole binary trees, it will be done, on the left sub-tree or on the right sub-tree according to the time constraint. As a result, the run time is reduced, because the genetic algorithms consume a lot of computing time. Our Proposed algorithm is faster 52 times than the genetic algorithm, and it has provided the better design results in term of the logic area. Eventually, we can admit that the proposed algorithm improves the complexity and the run time of the original genetic algorithm.

References

[1] Sonia Dimassi, Mehdi Jemai, Bouraoui Ouni and Abdellatif Mtibaa,"Hardware-software partitioning algorithm based on Binary Search Trees and Genetic Algorithm to optimize logic area for SOPC", Published in Journal of Theoretical and Applied Information Technology (JATIT), Vol.66, No.3, August 2014.

[2] D. Gajski, F. Vahid, S. Narayan, and J. Gong, "Specsyn: An environment supporting the specify-explore-refine paradigm for hardware/software

system design," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 6, no. 1, pp. 84–100, 1998.

[3] J. Henkel, "A low power hardware/software partitioning approach for core-based embedded systems," in Proceedings of the 36th annual ACM/IEEE Design Automation Conference. ACM, 1999, pp. 122–127.

[4] Bouraoui Ouni, Ramzi Ayadi and Abdellatif Mtibaa, "Combining Temporal Partitioning and Temporal Placement Techniques for Communication Cost Improvement" Advances in Engineering Software, Elsevier Publishers, Volume 42, no 7, July 2011, pp : 444-451.

[5] Bouraoui Ouni. , Ramzi Ayadi, and Abdellatif Mtibaa. "Temporal partitioning of data flow graph for dynamically reconfigurable architecture", Journal of Systems Architecture, vol 57, no 8, September 2011, pp 790-798

[6] Bouraoui Ouni and Abdellatif Mtibaa, "Optimal placement of modules on partially reconfigurable device for reconfiguration time improvement", Microelectronics International published by Emerald Group Publishing Limited, volume 29, Issue 2, 2012, Pages 101-107

[7] Ramzi Ayadi, Bouraoui Ouni and Abdellatif Mtibaa, "A Partitioning Methodology that Optimizes the Communication Cost for Recongurable Computing Systems" International Journal of Automation and Computing (IJAC), Institute of Automation and Springer-Verlag Publishers, Volume 9, N° 3, June 2012, pp 280-287.

[8] Mehdi Jemai, Sonia Dimassi, Bouraoui Ouni and Abdellatif Mtibaa, "Optimization of logic area for System on Programmable Chip based on hardware-software partitioning", International Conference on Embedded Systems and Applications (ICESA) Hammamet-Tunisia,March 2014.

[9] K. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 10, no. 3, pp. 193–208, 2002.

[10] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integrating physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 14, no. 11, pp. 1189 –1202, nov. 2006.

[11] J. Wu and T. Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning," Information processing letters, vol. 98, no. 2, pp. 41–46, 2006.

[12] A.Kalavade, "System-level codesign of mixed hardwaresoftware systems," PHDTHESIS, University of California, Berkeley, 1995.

[13] J. Henkel and R. Ernst, "An approach to automated hardware/ software partitioning using a flexible granularity that is driven by high-level estimation techniques," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 9, no. 2, pp. 273–289, 2001.

[14] R. Dick and N. Jha, "Mogac: a multiobjective genetic algorithm for hardware-software cosynthesis of distributed embedded systems," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 10, pp. 920–935, 1998.

[15] P. Eles, Z. Peng, K. Kuchcinski, and A. Doboli, "System level hardware/software partitioning based on simulated annealing and tabu search," Design Automation for Embedded Systems, vol. 2, no. 1, pp. 5–32, 1997.

[16] M. L´opez-Vallejo and J. L´opez, "On the hardware-software partitioning problem: System modeling and partitioning techniques," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 8, no. 3, pp. 269–297, 2003.

[17] A. Kalavade and P. Subrahmanyam, "Hardware/software partitioning for multifunction systems," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 17, no. 9, pp. 819–837, 1998.

[18] J. Holland, "Genetic algorithms," Scientific American, vol.267, no. 1, pp. 66–72, 1992.