The Application of Computerized Algorithms in the Design Method of Software-Hardware Dual-Track Partitioning in an Embedded System Abstract

Wang Yuanqiang, Yang Jie, Hao Shangfu*, Zhang Xiao and Yang Jingjing

School of Information Science and Engineering, Hebei North University, Zhangjiakou 075000, Hebei, China

Abstract: It has been proved that the hardware/software partitioning problem is NP-hard. Currently, we have tried a variety of computerized algorithms to resolve it, which can be divided into two major categories: accurate algorithms and heuristic algorithms. This paper will discuss accurate algorithms and heuristic algorithms respectively. Accurate algorithms take the example of a greedy algorithm. It abstracts the hardware/software partitioning problem into 01 knapsack model and obtains the exact optimal solution by the greedy algorithm, while heuristic algorithms use a genetic algorithm as an example. It converts the hardware/software partitioning problem into a multi-constraint 0-1 knapsack problem and solves it by employing the genetic algorithm therein. By steps like "variation" and "crossover", this algorithm makes an offspring solution quickly approach an optimal solution, thereby constructing a near-optimal heuristic solution of the HW/SW partitioning problem. Experimental results demonstrate that the algorithm proposed in this paper can effectively resolve the hardware/software partitioning problem, have a good global searching capability, and the heuristic algorithm performs faster than the traditional accurate algorithm, but the heuristic algorithm only acquires a near-optimal solution, which is not perfect.

Keywords: Hardware/software partitioning, Computerized algorithm, Accurate algorithm, Heuristic algorithm, Dynamic programming, Genetic algorithm

1. INTRODUCTION

1.1. Hardware-software Co-design

Hardware/software co-design is a brand new system design idea proposed to address the problem of embedded system design. It supports concurrent engineering to shorten the design cycle, uses automatic or semi-automated design technologies and integrates reliable hardware/software methods to improve the design quality [1-3], and utilizes the validation and assessment technique to detect design errors before it is too late.

In accordance with the system objectives and requirements, through a comprehensive analysis of system hardware and software services and available resources, it farthest excavates the system concurrency between hardware and software and collaboratively designs the software/hardware architecture, so that the system can work in the best working status. This design approach can make full use of the existing hardware/software resources, shorten the systematic development cycle, lower development costs, improve system performance, and avoid the drawbacks incurred by the independent design software/hardware architecture.

The hardware/software co-design method emphasizes on coordination between hardware designers and software designers, concurrent design of systematic hardware and software [4-6], and control of the consistency of hardware/software and correctness of the system during the whole designing process.

1.2. The Hardware/Software Partitioning Problem

Hardware/software system design is the core technology for the development of modern embedded systems, while the hardware/software partitioning problem is a key part of the hardware/software co-design. In the system design stage, hardware/software partitioning determines software/hardware implementation ways for each part as per the design constraints and the characteristics of each part of the system, in order to obtain a high-performance, low-cost optimized design scheme. It determines how to divide system services into hardware and software, which has an important impact on the system performance.

The hardware/software partitioning problem is NP-hard [7-9]. In recent years, research into the automatic partitioning algorithm catches increasing attention. Currently, methods used in the partitioning problem can be summarized into three categories:

(1) Integer Linear Programming (ILP)/Mixed Integer Linear Programming (MILP) methods;

*Address correspondence to this author at the School of Information Science and Engineering, Hebei North University, Zhangjiakou 075000, Hebei, China; E-mail: r78z-yang@126.com

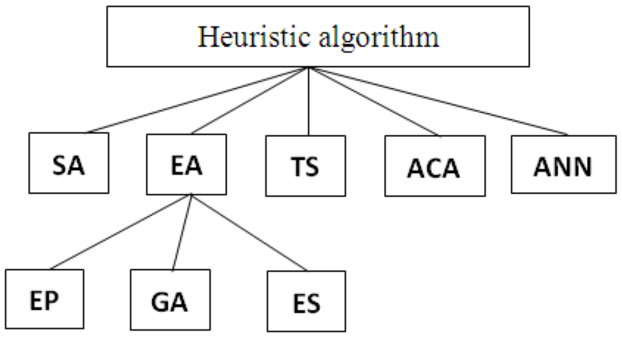

Fig. (1). Heuristic algorithm partitioning based on this method.

(2) Heuristic algorithms, such as simulated annealing algorithm, tabu search, and the genetic algorithm discussed in this paper, etc.

(3) Other methods such as a cluster-based algorithm, dynamic programming method, PACE method, bidirectional search, Kernighan/Lin algorithm and CGLP/IBS algorithm, etc.

1.3. Traditional Design Methods

The traditional embedded system design model of first hardware and then software needs repeated amendments and repetition tests. The entire design process is largely dependent on the designer's experience. The long design cycle and high development costs often depart from the requirements of original design in the process of repeated amendments.

Due to the low efficiency of traditional design methods, the computerized algorithms are introduced to settle the hardware and software partitioning problem. The typically used algorithms are principally divided into two major categories: accurate algorithms and heuristic algorithms. Below are discussed and compared the genetic algorithm and the greedy algorithm that represent heuristic algorithms and accurate algorithms respectively.

2 HARDWARE/SOFTWARE PARTITIONING OF AN EMBEDDED SYSTEM BASED ON HEURISTIC ALGORITHMS

2.1. Hardware/Software Partitioning Problem Based on a Genetic Algorithm

A heuristic algorithm means that among the random group optimization process, individuals use their own or overall experience to develop search strategies and usually obtain a near-optimal solution. Common modern heuristic algorithms include simulated annealing algorithms (SA), genetic algorithms (GA), list search algorithms (TS), evolutionary programming (ES), evolution strategies (ES), ant colony algorithms (ACA), and artificial neural network (ANN). Fig. (**1**) shows a modern heuristic algorithm partitioning scheme based on this method.

In this paper, the genetic algorithm is taken as an example to discuss the software/hardware partitioning problem by using the heuristic algorithms. The heuristic algorithm is used to address the hardware/software partitioning problem. Usually according to the characteristics of the algorithm, the hardware/software partitioning problem is first modeled.

An embedded system is a collection of function nodes. SYSTEM= {C1, C2,…Ci…Cn} is employed to represent these function nodes. Each Ci unit can be achieved in the way of software or hardware. The symbol IP is used uniformly in this paper. Each IP has parameters like cost, execution times, and execution area. The function nodes can be identified with triples.

$$C = (x, s, t) \tag{1}$$

where x□ {0,1}, 0 means the node function is realized by the software method, while 1 means the node function is realized by the hardware method.

s represents the hardware area of the node function realized by the hardware,

t = (th, ts, tim) represents the execution time of the node,

th represents the hardware execution time of the node

ts represents the software execution time of the node

tim represents the number of calls to the node

When designing each IP, designers have to sacrifice certain performance in exchange of some other performance, and ultimately optimize the performance of the whole system. In order to achieve this purpose, different implementation ways or different architectures can be made use of to design different IPs to achieve the same functionality. Each function node in the system has two implementation ways, which are hardware and software. The hardware and software partitioning problem is essentially the optimization problem in the space C1×C2×…Cn, which is to find a combination of IP {IP1i, IP2j,...IPnk}, so that the overall performance is the best and the least costly. This paper explores the shortest solution for embedded systems running under certain conditions of the hardware area.

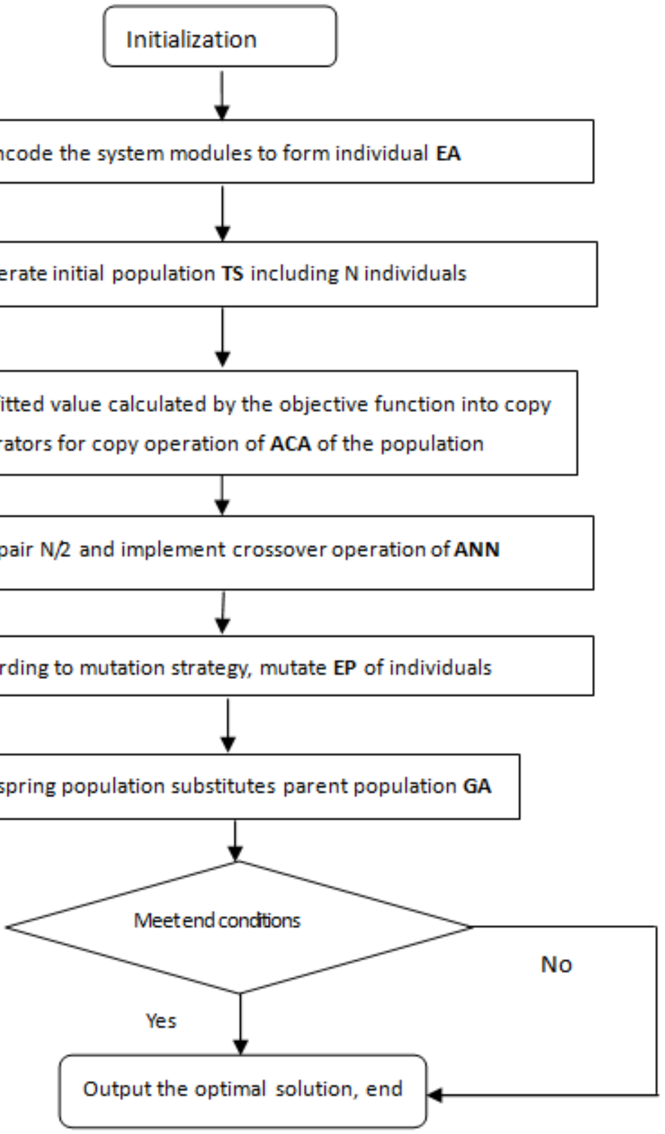
Initialization
Encode the system modules to form individual **EA**
Randomly generate initial population **TS** including N individuals
Substitute fitted value calculated by the objective function into copy operators for copy operation of **ACA** of the population
Randomly pair N/2 and implement crossover operation of **ANN**
According to mutation strategy, mutate **EP** of individuals
Offspring population substitutes parent population **GA**
Meet end conditions
No
Yes
Output the optimal solution, end

Fig. (2). Algorithm flow chart.

The partitioning problem is transformed into a classic 0-1 knapsack problem for solution:

$$\begin{cases} \textbf{minimize } \sum_{i=0}^{n} x_i * Sth_i + (1-\mathrm{x}_i) * \mathrm{Sts}_i \\ \textbf{object to } \sum_{i=0}^{n} x_i * S_i < S_{\max} \end{cases} \quad (2)$$

where

Sth_i is the total running time for the IP when implemented in the hardware $Sth_i = th_i * tim_i$; Sts_i is the total running time for the IP when implemented in the software $Sts_i = ts_i * tim_i$

The software/hardware partitioning process is grounded on the genetic algorithm principle. Adopt genetic manipulation such as reproduction, crossover and mutation, and continue to iterate until the pre-set termination condition is met, as shown in Fig. (**2**).

By analysis of the software/hardware partitioning model, it can be found that for each function node IPi in the embedded system, two implementation ways of hardware and software are respectively denoted as Ki, Ki={0,1}. The solution domain for the software/hardware partitioning problem is K, K=K1×K2×…×Km, where m is the total number of function nodes.

In this paper, binary encoding is adopted. The binary string B whose length is Li represents the implementation way of the system hardware and software; Bi corresponds to the implementation scheme of node IPi.

Bi=(bLi-1bli-2…b0), r=br;

r=1 indicates that the node is implemented in a hardware manner

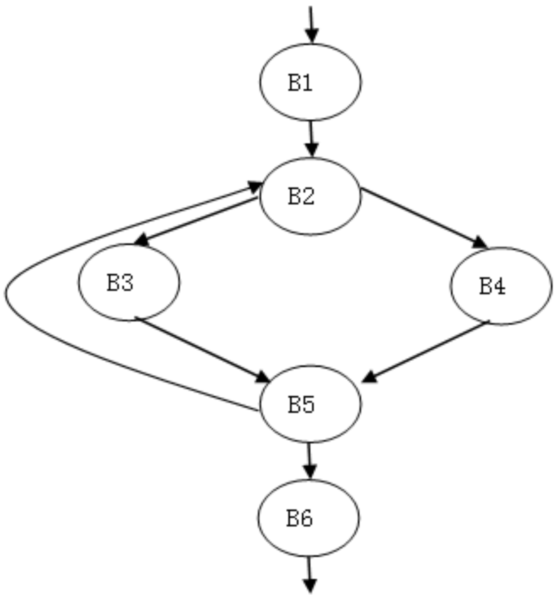

Fig. (3). Task Execution Flow Chart.

3. HARDWARE/SOFTWARE PARTITIONING OF AN EMBEDDED SYSTEM BASED ON GREEDY ALGORITHMS

A greedy algorithm is to seek the optimal solution for each sub-problem, which is to obtain the global optimal solution by getting the local optimal solution. The greedy algorithm strategy is to select the currently optimal choice every time.

The software/hardware partitioning problem is converted into the path model. B_i node represents each scheduling module. The high-frequency access path consists of $B_1, B_2, L\ , B_n$, namely $P = \{B_1, B_2, L\ , B_n\}$. For any $i \in [1, n-1]$, B_{i+1} is the successor node for B_i, similar to the previous section.

S_i s represents the hardware area of the node i function realized by the hardware,

$t = (th_i, ts_i, tim_i)$ represents the execution time of the node i,

th_i represents the hardware execution time of the node i

ts_i represents the software execution time of the node i

H and S denote the sets of hardware and software implementation, respectively. For a determined P, find a software/hardware partitioning solution that meets $P = H \cup S$, $H \cap S = \phi$. When the constraint condition of hardware area is satisfied, achieve maximum system efficiency.

Here the software/hardware partitioning problem is transformed into the shortest path problem of a directed graph. The figure has only one exit and entrance. The shortest path should meet the hardware area constraints. This graph can be represented by Fig. (**3**).

If the adjacent nodes are realized in the same way, assume that its communication cost is zero, otherwise $c_{i,i+1}$ is used to indicate the communication cost between adjacent nodes B_i and B_{i+1}. In order to conform to the assumption, the hardware implementation can fasten the running speed in relation to the software implementation. The constraints are added in the figure.

$$\forall i \in [1, n-1], Ts_i\ ?\ Th_i + c_{i,i+1} \tag{3}$$

Set xŔ {0,1}, 0 means the node function is realized by the software method, while 1 means the node function is realized by the hardware method. $(x_1, x_2, L\ , x_n)$ is a solution to the hardware/software partitioning problem. Correspondingly, set the function $T(x_1, x_2, L\ , x_n)$ to be the corresponding run-time.

$$T(x_1, x_2, \cdots, x_n) = \sum_{i=0}^{n} x_i * Th_i + (1 - x_i) * Ts_i + \sum_{i=0}^{n-1} |x_i - x_{i+1}| \bullet c_{i.i+1} \tag{4}$$

Then the maximization problem is:

$$\begin{cases} \textbf{minimize } T(x_1, x_2, L\ , x_n) \\ \textbf{object to } \sum_{i=0}^{n} x_i * S_i < S_{\max} \end{cases} \tag{5}$$

The same as a genetic algorithm in the last section, the hardware/software partitioning problem is transformed into a 01 knapsack problem.

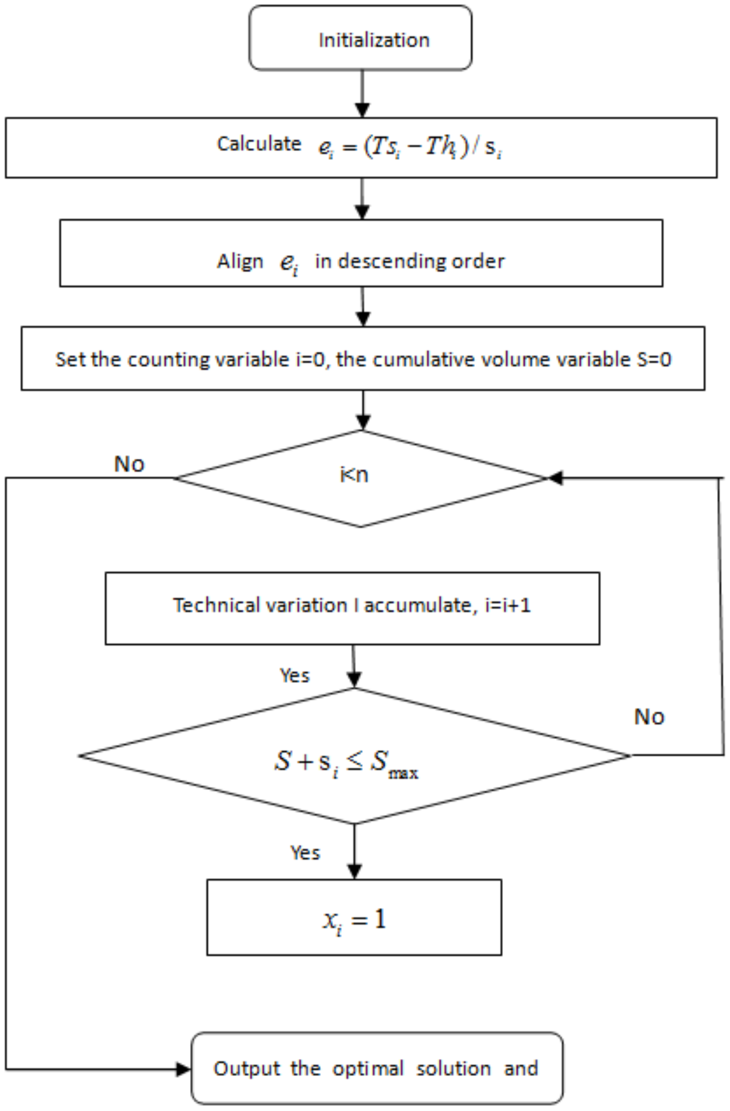
Initialization
Calculate $e_i = (Ts_i - Th_i)/s_i$
Align e_i in descending order
Set the counting variable i=0, the cumulative volume variable S=0
i<n
No
Technical variation I accumulate, i=i+1
Yes
$S + s_i \le S_{max}$
No
Yes
$x_i = 1$
Output the optimal solution and

Fig. (4). Flow chart of a greedy algorithm.

Here the communication cost between hardware and software is ignored in order to simplify the problem, then

$$T(x_1, x_2, \cdots, x_n) = \sum_{i=0}^{n} x_i * Th_i + (1 - x_i) * Ts_i \qquad (6)$$

In order to obtain the optimal solution to the knapsack, a greedy algorithm is to sort the value and weight, select the items at the top successively until the knapsack can't hold them. The greedy algorithm is used to solve the hardware/software partitioning model above. The value of the node B_i is defined as $e_i = (Ts_i - Th_i)/s_i$, which is the ratio of hardware based acceleration time and the hardware area. Sort e_i again, making e_i align in a non-increasing sequence, and then successively select B_i to fill the knapsack until there are no blocks of the suitable size that can be loaded into the knapsack. The specific program flow chart is as shown in Fig. (**4**).

The time complexity of the greedy algorithm is the time complexity for sorting, which is O (nlogn). In the running time, the greedy algorithm is very efficient, but it cannot guarantee that the solution obtained is a global optimal solution.

4. COMPARISON OF ALGORITHM PERFORMANCE

In order to evaluate the execution performance of the hardware/software partitioning algorithm based on genetic algorithms, while identifying several important parameters that determine the algorithm performance, we refer to the method proposed in Literature [10, 11] and generate a test set; each test objective in the test set contains different numbers of function nodes, node connection, and a set of input/output variables corresponding to each connection.

In addition, we also use an application as a test object. This application contains 218 EHDL description statements, and its corresponding CDFG contains 45 nodes and 57 edges.

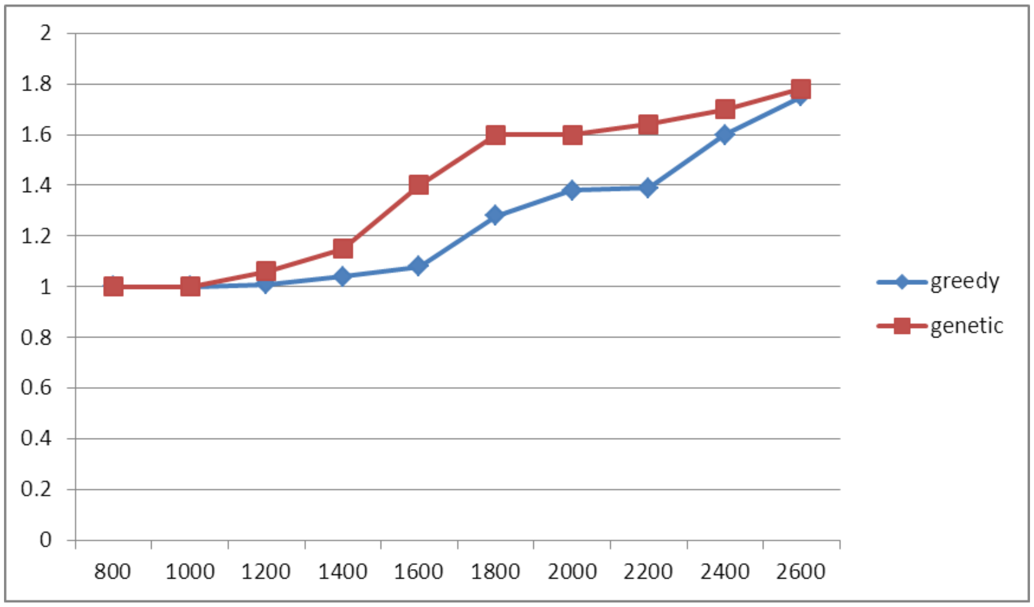

Fig. (5). Performance comparison.

Under the software implementation way of the node, the execution time is estimated by 8086 processor. Estimation is made of the execution time and the required area of nodes under different hardware schemes by using ActelACT 3 FPGA hardware library. Fig. (**5**) demonstrates a performance comparison of hardware/software partitioning schemes obtained respectively by using a genetic algorithm and a greedy algorithm under different hardware resources constraints. The ordinate is the speed-up ratio obtained by different division schemes. The reference value for the system execution speed is the system execution time under the pure software implementation mode.

CONCLUSION

The solutions to the hardware/software partitioning problem by a genetic algorithm and a greedy algorithm are compared. A heuristic algorithm represented by a genetic algorithm is usually an iterative algorithm. It starts from a set of solutions which are the initial value and continuously optimizes the initial solutions by a specific search strategy, thus approaching the optimal solution. Heuristic algorithms have high operating efficiency, but only get a near-optimal solution, so it is applicable to the large-scale hardware/software partitioning problem that has less stringent requirements for optimal efficiency.

Accurate algorithms represented by a greedy algorithm can usually acquire the global optimal solution, in which dynamic programming is more commonly used algorithm that has low operation efficiency. The time complexity is up to O (n^2). When it is of a small scale, the running efficiency will not be lower than the heuristic algorithms, and the solution acquired will be much better than heuristic algorithms. Therefore, it is suitable for solving the small-scale hardware/software partitioning problem that has stringent requirements for optimal efficiency.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This work was supported by the National Science and Technology Support Program (No. 2012BAJ18B08) of Ministry of Science and Technology, Zhangjiakou Municipal Science and Technology Bureau (No.1321019B) and Hebei North University (No. 120164, No. Q2014002).

REFERENCES

[1] W. Wolf, “A decade of hardware/software codesign,” *Computer*, vol. 36, no. 4, pp. 38-43, 2003.

[2] R. Ernst, H. Jörg, and B. Thomas, “Hardware-software cosynthesis for microcontrollers,” *Readings in Hardware/Software Co-Design,* vol. 10, no. 4, pp. 18-29, 2002.

[3] L. Séméria, and G. Abhijit, “Methodology for hardware/software co-verification in C/C++ (short paper),” In: *Proceedings of the 2000 Asia and South Pacific Design Automation Conference. ACM*, 2000.

[4] T. Wiangtong, Y.K.C. Peter, and W. Luk, “Hardware/software codesign: a systematic approach targeting data-intensive applications,” *Signal Processing Magazine, IEEE*, vol. 22, no. 3, pp. 14-22, 2005.

[5] G.S. Walia, and J.C. Carver, “A systematic literature review to identify and classify software requirement errors,” *Information and Software Technology*, vol. 51, no. 7, pp.1087-1109, 2009.

[6] H.P. Breivold, C. Ivica, and L. Magnus, “A systematic review of software architecture evolution research,” *Information and Software Technology*, vol. 54, no. 1, vol. pp. 16-40, 2012.

[7] P. Arató, Á.M. Zoltán, and O. András, “Algorithmic aspects of hardware/software partitioning,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 10, no .1, pp. 136-156, 2005.

[8] J. Du, and J.Y.T. Leung, “Minimizing total tardiness on one machine is NP-hard,” *Mathematics of Operations Research*, vol. 15, no .3, pp. 483-495, 1990,

[9] A. Kalavade, and A. Lee, “The extended partitioning problem: hardware/software mapping, scheduling, and implementation-bin

selection," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 125-163, 1997.

[10] N. Srinivas, and K. Deb, "Muiltiobjective optimization using non-dominated sorting in genetic algorithms," *Evolutionary computation*, vol. 2, no. 3, pp. 221-248, 1994.

[11] R. Günter, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks,* vol. 5, no. 1, pp. 96-101, 1994.

Received: May 26, 2015 Revised: July 14, 2015 Accepted: August 10, 2015