Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2013, Article ID 138037, 8 pages
http://dx.doi.org/10.1155/2013/138037

Research Article

An Effective Heuristic-Based Approach for Partitioning

Xibin Zhao,[1] Hehua Zhang,[1] Yu Jiang,[1,2] Songzheng Song,[3] Xun Jiao,[4] and Ming Gu[1]

[1] *School of Software, Tsinghua University, TNLIST, KLISS, Beijing 100084, China*
[2] *School of Computer Science and Technology, Tsinghua University, Beijing 100084, China*
[3] *School for Integrative Sciences and Engineering, National Univerisity of Singapore, Kent, Singapore 119077*
[4] *International school, Beijing university of post and telecommunication, Beijing 100876, China*

Correspondence should be addressed to Yu Jiang; jiangyu198964@126.com

Received 7 March 2013; Accepted 22 March 2013

Academic Editor: Xiaoyu Song

As being one of the most crucial steps in the design of embedded systems, hardware/software partitioning has received more concern than ever. The performance of a system design will strongly depend on the efficiency of the partitioning. In this paper, we construct a communication graph for embedded system and describe the delay-related constraints and the cost-related objective based on the graph structure. Then, we propose a heuristic based on genetic algorithm and simulated annealing to solve the problem near optimally. We note that the genetic algorithm has a strong global search capability, while the simulated annealing algorithm will fail in a local optimal solution easily. Hence, we can incorporate simulated annealing algorithm in genetic algorithm. The combined algorithm will provide more accurate near-optimal solution with faster speed. Experiment results show that the proposed algorithm produce more accurate partitions than the original genetic algorithm.

1. Introduction

Embedded systems [1–3] are becoming more and more important because of the wide applications. They consist of some hardware and software components. This is beneficial, because hardware will lead to faster speed with more expensive cost, while software will lead to lower speed with cheaper cost. So, critical components can be implemented in hardware and noncritical components can be implemented in software. This kind of hardware/software partitioning can find a good tradeoff between system performance [4] and power consumption [5]. How to find an efficient partition has been one of the key challenges in embedded system design. Traditionally, partitioning is carried out manually. The target system is usually given in the form of a task graph, which is usually assumed to be a directed acyclic graph describing the dependencies among the components of embedded system. Recently, many research efforts have been undertaken to automate this task. Those efforts can be classified by the feature of the partitioning architecture and algorithm aspects.

On the target architecture aspect of the partitioning problem, some are assumed to consist of a single software and a single hardware unit [6–9]; parallelism among components is another assumed limitation, while others do not impose these limitations. The target system is usually given in the form of a task graph, a directed acyclic graph describing the dependencies between the components of the system.

The family of exact algorithm includes branch and bound [10–12], integer linear programming [6, 7, 13], and dynamic programming [14–16]. Those algorithms are used for partitioning problems with small inputs successfully. When applied to problems with inputs of large size, they tend to be quite slow. The reason is that most formulations of the partitioning problem are NP hard [17], and these exact algorithms have exponential runtime.

Corresponding to the exact algorithms, there are more flexible and efficient heuristic algorithms. Right now, most of the researches focus on heuristic algorithms. Traditional heuristic algorithms are software oriented and hardware oriented. The hardware oriented heuristic algorithms start with a complete hardware implementation and then iteratively move component to software until the given constraints are satisfied [18, 19]. The software oriented algorithms start with a complete software implementation and iteratively move component to hardware until the speedup time constraints are met [20, 21]. Many general-purpose heuristic algorithms are also utilized to solve the system partitioning problem. Simulated annealing-related algorithms [22–24], genetic algorithms [8, 9, 25, 26], tabu search, and greedy algorithms [25, 27, 28] have been extensively used to solve partitioning problem.

In addition to the general-purpose heuristic algorithms, some researchers have constructed heuristic algorithms that leverage problem-specific domain knowledge and can find high-quality solution rapidly. For example, authors define two versions of the original partitioning problem and propose two corresponding algorithms in [29]. In the first algorithm, the problem is converted to find a minimum cut in the corresponding auxiliary graph. The second algorithm is to run the first algorithm with several different parameters and select the best partition from this set that fulfills the given limit. Another example is presented in [30]. Authors reduce the partitioning problem to a variation of knapsack problem and solve it by searching one-dimension solution space with three greedy-based algorithms, instead of searching two-dimension solution space in [29]. This strategy reduces time complexity without loss of accuracy. Some researchers address the issue that we cannot accurately determine the cost and time of system components in the design stage. Some people think that they are a subjective probability and make use of this theory in system level partitioning [31–33].

Most of the algorithms work perfectly within their own codesign environment. In this paper, we construct a communication graph, in which the implementation cost, execution time, and communication time are all taken into account. We construct a mathematical model based on this communication graph and solve the model by an enhanced heuristic method. The proposed heuristic method incorporates simulated annealing into genetic algorithm to improve the accuracy and speed of original genetic algorithm. Simulation results show that the new algorithm provides more accurate and faster partitions than that of original genetic algorithm.

This paper is organized as follows. Some background on the genetic algorithm and simulated annealing is introduced in Section 2. The constructed communication graph and the proposed mathematical model definition for partitioning problem are presented in Section 3. Section 4 presents the method which incorporates simulated annealing in genetic algorithm, for the partitioning model. Experiment results about the comparison of the original genetic method and the combined method are given in Section 5. Finally, we conclude the paper in Section 6.

2. Background

This section provides some detailed notations and definitions of genetic algorithm and simulated annealing algorithm.

2.1. Simulated Annealing. Simulated annealing algorithm is a generic probabilistic metaheuristic for the global optimization problem, locating a good approximation to the global optimum of a given function. It is proposed by Kirkpatrick et al. [34], based on the analogy between the solid annealing and the combinatorial optimization problem. In condensed matter physics, annealing involves materials' heating and controlled cooling.

Before the implementation of simulated annealing algorithm, we need to choose an initial temperature. After the initial state is generated, the two most important operations *Generation* and *Acceptation* can be performed.

Then, the algorithm will reduce the value of the temperature. The iteration process will stop until certain condition is met; for example, a good approximation to the global optimum of the given function has been found. The algorithm is shown in the Algorithm 1.

2.2. Genetic Algorithm. A genetic algorithm is a search heuristic that mimics the process of natural evolution. The basic principles of genetic algorithm were laid down by Holland [35] and have been proved useful in a variety of search and optimization problems. Genetic algorithms are based on the survival-of-the-fitness principle, which tries to retain more genetic information from generation to generation. A genetic algorithm is composed of a reproductive plan that provides an organizational framework for representing the pool of genotypes of a generation. After the successful genotypes are selected from the last generation, the set of genetic operators such as crossover, mutation, and inversion is used in creating the offspring of the next generation. Whenever some individuals exhibit better than average performance, the genetic information of these individuals will be reproduced more often.

Before the implementation of genetic algorithm, we need to generate an initial population and define a fitness function. Each individual of the initial population is a binary string which corresponds to a dedicated encoding. The initial population is usually generated randomly. We will evaluate each individual with the fitness function. The fitness of each individual is defined as $f_i/\overline{f}$, where f_i is the evaluation of individual i and $\overline{f}$ is the average evaluation of all individuals. Then, the most important three operators *Selection*, *Crossover*, and *Mutation* can be performed on the current generation.

Then, we can evaluate individuals of the next generation with the fitness function, deciding whether to stop or go on performing the three operations. The evolution process will stop until certain condition is met; for example, the fitness of individual will not be improved any more. Finally, the algorithm will return the best individual of the latest generation as the solution. The algorithm is shown in the Algorithm 2.

3. Problem Formulation

This section provides the formal definition of the partitioning problem, including the constructed communication graph structure, formal notations, and mathematical model.

3.1. Problem Definition. While preserving the dependencies among the system task modules, we build a graph structure to represent the real-world system. The communication graph can be constructed through the following steps.

```
(1) Initialize the parameters of the annealing algorithm;
(2) Randomly generate an initial state as the current_state;
(3) $K := 1$;
(4) while (system has been frozen) do
(5)     while (system equilibrium at $T_k$) do
(6)         call generation strategy for the next_state_j;
(7)         $\Delta E_{ij} = \mathbf{cost}(next\_state\_j) - \mathbf{cost}(current\_state)$;
(8)         $P_r = A_{ij}$;
(9)         if ($P_r > random[0, 1)$) then
(10)            current_state := next_state_j;
(11)        end if
(12)    end while
(13)    $T_{k+1} := T_k \cdot \alpha$;
(14) end while
(15) return current_state;
```

Algorithm 1: Annealing algorithm.

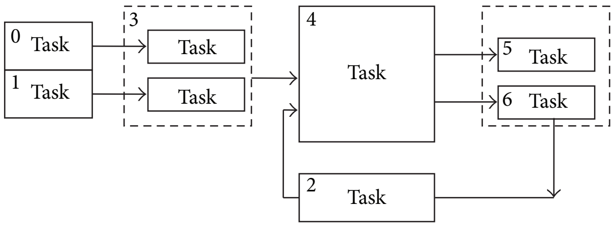

Figure 1: Constructed task module of a given system.

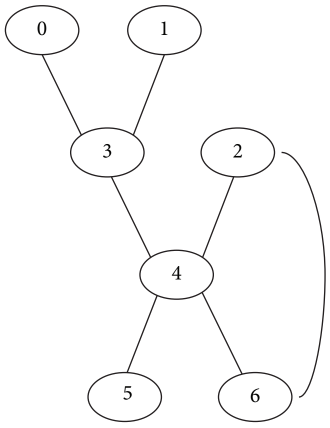

Figure 2: Constructed graph structure for the system to be partitioned.

(i) Determine the boundary of the system to be partitioned, identify the main task modules in this boundary, and describe the data signal flow through these task modules. We can accomplish this by referring to the design document, designer, implementer, and deployer of the system. A simple example is shown in Figure 1.

(ii) Construct the communication graph structure for the presented system. We map a node to each basic task module. Edges presented in step 1 are regarded as causal or dependency correlations caused by data communication. An arc is added between two nodes if the represented basic task modules are connected. This can be easily finished based on the model constructed in the previous step. The constructed communication graph structure for the system model is shown in Figure 2.

Based on the communication graph structure, we can formalize the problem as follows. The communication graph is denoted as $G(V, E)$, where V is the set of nodes $\{v_1, v_2 \ldots, v_n\}$ and E is the set of edges $\{e_{ij} \mid 1 \leq i, j \leq n\}$. We need to add cost values and execution time to each node as well as communication cost to each edge. The following notations are defined on V and E.

(i) h_i denotes the cost of node i in hardware implementation, and s_i denotes the cost of node i in software implementation.

(ii) t_i^h denotes the execution time of node i in hardware implementation, and t_i^s denotes the execution time of node i in software implementation.

(iii) c_{ij} denotes the communication time between node i, j. The value of c_{ij} is given in the context that the two nodes are implemented in different way.

The partitioning problem is to find a bipartition P, where $P = (V_h, V_s)$ such that $V_h \bigcup V_s = V$ and $V_h \bigcap V_s = \emptyset$. The partitioning problem can be represented by a decision vector $\mathbf{x}(x_1, x_2 \ldots, x_n)$, representing the implementation way of the n task modules. There are three kinds of optimization and decision problems defined on the software/hardware partitioning.

Q_1: H_0 is the given hardware constraint. Find a HW/SW partition P such that $H_X \leq H_0$ and T_X is the minimal execution time.

Q_2: T_0 is the given execution time constraint. Find a HW/SW partition P such that $T_X \leq T_0$ and H_X is the minimal hardware cost.

```
(1) Initialize the parameters of the genetic algorithm;
(2) Randomly generate the old_population;
(3) generation := 1;
(4) while (generation ≤ max_generation) do
(5)     clear the new_population;
(6)     compute fitness of individuals in the old_population;
(7)     copy the individual with the highest fitness;
(8)     while (individual_number < population_size) do
(9)         Select two parents from the old_population;
(10)        Perform the crossover to produce two offsprings;
(11)        Mutate each offspring based on mutation_rate;
(12)        Place the offspring to new_population;
(13)    end while
(14)    Replace the old_population by the new_population;
(15) end while
(16) return new_solution with the best fitness;
```

Algorithm 2: Genetic algorithm.

```
(1) Encode the parameters for the partitioning problem;
(2) Initialize the first generation $P_0$;
(3) Calculate the fitness of each individual in $P_0$;
(4) Copy the individual with the highest fitness to the solution;
(5) while (termination conditions) do
(6)     while (number of individuals ≤ generation size) do
(7)         Select two individuals $(g_1, g_2)$;
(8)         Perform crossover on $(g_1, g_2) \rightarrow (g_1', g_2')$;
(9)         if (max{fitness($g_1'$), fitness($g_2'$)} ≤ max{fitness($g_1$), fitness($g_2$)}) then
(10)            Reject the crossover with $g_1' = g_1$, $g_2' = g_2$;
(11)        else
(12)            Accept the crossover;
(13)        end if
(14)        Perform mutation on $g_1'$ to produce $ng_1$;
(15)        if (fitness($ng_1$) ≤ fitness($g_1'$)) then
(16)            Reject the mutation, $ng_1 = g_1'$;
(17)        else
(18)            Accept the mutation;
(19)        end if
(20)        Perform the above steps on $g_2'$ to produce $ng_2$;
(21)    end while
(22)    Calculate the fitness of each individual;
(23)    if (the highest fitness ≥ fitness(solution)) then
(24)        Copy the individual with the highest fitness;
(25)    end if
(26)    increase the generation number;
(27) end while
(28) return solution: $x[i]$, $i \in [1, n]$;
```

Algorithm 3: Heuristic algorithm.

Q_3: H_0 and T_0 are the given hardware constraint and execution time constraint, respectively. Find a HW/SW partition P such that $H_X \leq H_0$ and $T_X \leq T_0$.

It has been proved that Q_1, Q_2 are NP hard and Q_3 is NP complete [36]. In this paper, HW/SW partitioning is performed according to the Q_2 type.

3.2. Mathematical Model. As described in Section 1, a partition is characterized by two metrics: cost and time. The cost includes hardware cost and software cost. It represents the resource consumption to achieve the hardware and software implementation of each task module. The time includes the execution time of each task module and the communication time between task modules.

```
(1) Encode the parameters and solution for the partitioning problem;
(2) Initialize the first generation $P_0$, temperature $T_0$, annealing ratio $\alpha$;
(3) Calculate the fitness of each individual in $P_0$;
(4) Copy the individual with the highest fitness to the solution;
(5) while (termination conditions) do
(6)     while (number of individuals ≤ number of the generation size) do
(7)         Select two individuals $(g_1, g_2)$ from the current generation;
(8)         Perform crossover on $(g_1, g_2)$ to produce two new individuals $(g_1', g_2')$;    /∗ start of annealing-crossover∗/
(9)         if (max{fitness($g_1'$), fitness($g_2'$)} ≤ max{fitness($g_1$), fitness($g_2$)}) then
(10)            $\Delta C$ = max{fitness($g_1'$), fitness($g_2'$)}− max{fitness($g_1$), fitness($g_2$)};
(11)            if (min{1, exp($-\Delta C/T_k$)} ≥ random[1, 0)) then
(12)                Accept the crossover;
(13)            else
(14)                Reject the crossover with $g_1' = g_1$, $g_2' = g_2$;
(15)            end if
(16)        else
(17)            Accept the crossover;
(18)        end if                                                                             /∗ end of annealing-crossover ∗/
(19)        Perform mutation on $g_1'$ to produce $ng_1$;                                      /∗ start of annealing-mutation∗/
(20)        if (fitness($ng_1$) ≤ fitness($g_1'$)) then
(21)            $\Delta C$ = (fitness($ng_1$) − fitness($g_1'$));
(22)            if (min{1, exp($-\Delta C/T_k$)} ≥ random[1, 0)) then
(23)                Accept the mutation;
(24)            else
(25)                Reject the mutation, $ng_1 = g_1'$;
(26)            end if
(27)        else
(28)            Accept the mutation;
(29)        end if                                                                             /∗ end of annealing-mutation∗/
(30)        Perform step (19)–(29) on $g_2'$ to produce $ng_2$;
(31)    end while
(32)    Calculate the fitness of each individual in current generation;
(33)    if (the highest fitness of the current generation ≥ fitness(solution)) then
(34)        Copy the individual with the highest fitness to the solution;
(35)    end if
(36)    Reduce the temperature and increase the generation number;
(37) end while
(38) return solution: $x[i]$, $i \in [1, n]$;
```

ALGORITHM 4: Combined heuristic algorithm.

Based on the definition of previous subsection, hardware cost $H(\mathbf{x})$ of the partition $P(\mathbf{x})$ and the total time metric $T(\mathbf{x})$ can be formalized as follows:

$$H(\mathbf{x}) = \sum_{i=1}^{n} h_i (1 - x_i),$$
$$T(\mathbf{x}) = \sum_{i=1}^{n} t_i^s x_i + t_i^h (1 - x_i) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left[(x_i - x_j)^2 \right]. \tag{1}$$

Based on the formalization of the two metrics and the given constraint M on execution time, the partitioning problem can be modeled as the following optimization problem:

$$\begin{aligned} &\text{minimize} \quad H(\mathbf{x}), \\ &\text{subject to} \quad T(\mathbf{x}) \le M \quad \mathbf{x} \in \{0,1\}^n, \end{aligned} \tag{P_1}$$

which can be simplified as the problem (P_2) presented later:

$$\begin{aligned} &\text{maximize} \quad \sum_{i=1}^{n} h_i x_i, \\ &\text{subject to} \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left[(x_i - x_j)^2 \right] \\ &\qquad + \sum_{i=1}^{n} (t_i^s - t_i^h) x_i \le M - \sum_{i=1}^{n} t_i^h, \quad \mathbf{x} \in \{0,1\}^n. \end{aligned} \tag{P_2}$$

4. Algorithm

In this section, we propose two algorithms to solve the partitioning problem (P_2) based on genetic algorithm and simulated annealing algorithm. The basic principles of genetic algorithm were laid down by Holland [35] and have been proved useful in a variety of search and optimization problems. Genetic algorithm simulates the survival-of-the-fitness principle of nature. The principle provides an organizational reproductive framework: starting from an initial population, proceeding through some random selection, crossover, and mutation operators from generation to generation, and converging to a group of best environment-adapted individuals. Simulated annealing algorithm is a generic probabilistic metaheuristic for the global optimization problem, locating a good approximation to the global optimum of a given function. It is proposed by Kirkpatrick et al. [34], based on the analogy between the solid annealing and the combinatorial optimization problem.

4.1. Initial Algorithm. We apply the genetic algorithm to the uncertain partitioning problem to find the approximate optimal solution of the problem (P_2). The pseudo code in the Algorithm 3 shows the description of the algorithm. The steps (1)–(4) are the initialization of parameters and solution of the partition problem. The step (5) is used to check whether the termination condition of the propagation is met or not. The step (6) is used to ensure that the number of individuals of the next generation is not reduced. The crossover and mutation operations are performed in the iteration block to produce individuals of the next generation. The fitness function is defined on the object function of the problem (P_2). We choose the crossover and mutation strategy from [36].

4.2. Improved Algorithm. We note that the genetic algorithm has a strong global search capability, while the simulated annealing algorithm will fail in a local optimal solution easily. Hence, we can incorporate simulated annealing algorithm in genetic algorithm. We hope that the combined algorithm will provide more accurate near-optimal solution with faster speed. The pseudo code in the Algorithm 4 shows the algorithm.

The steps (8)–(18) are the original crossover operation incorporated to the Metropolis of annealing algorithm. The key idea is that when the original crossover operation produces better individuals, the crossover operation is accepted. Otherwise, we will accept the new individuals as the candidates of next generation in the Metropolis criterion. The steps (9)–(29) are the original crossover operation incorporated with the Metropolis of annealing algorithm. The key idea is the same as annealing crossover. The modified genetic operators ensure that the next generation is better than the current generation with the accepted rules based on fitness and Metropolis criterion. Those accepted rules speed up the convergence of the solution process without loss of accuracy. The steps (32)–(36) are the update of solution, generation number, and temperature.

5. Empirical Results

The proposed two algorithms are heuristics; the model is constructed from the communication graph. We have to determine the performance and the quality of the model and the solution. We have implemented them in C and test the

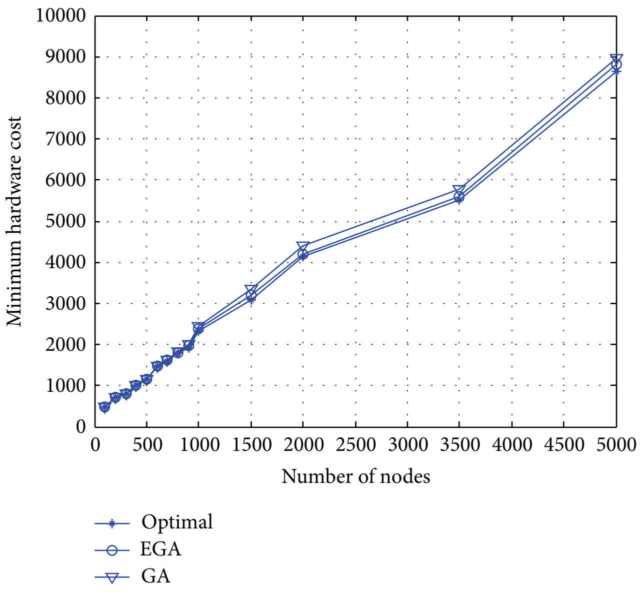

FIGURE 3: Minimum cost comparison of the partition.

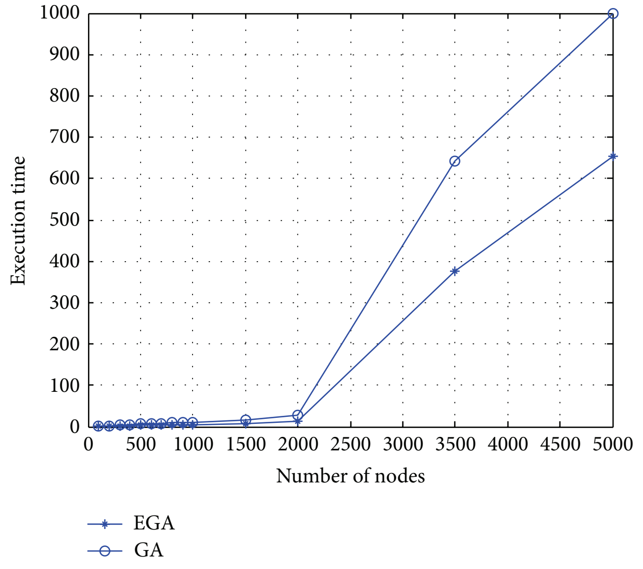

FIGURE 4: Runtime comparison of the two algorithms.

algorithms on Intel i5 2.27 GHZ PC. In order to demonstrate the effectiveness of the proposed algorithm, we compare it with original genetic-algorithm-based partitioning [36]. For testing, several random instances with different nodes and metrics are utilized. The parameters of the partitioning problem are generated with the following rules.

(i) h_i is randomly generated in $[1, 100]$.

(ii) t_i^h is randomly generated in $[1, 100]$, and t_i^s is randomly generated in $[t_i^h, 200 + t_i^h]$.

(iii) c_{ij} is randomly generated in $[1, 20]$.

(iv) M is a given time constraint and randomly generated in $[\sum_1^n t_i^h, \sum_1^n t_i^s]$.

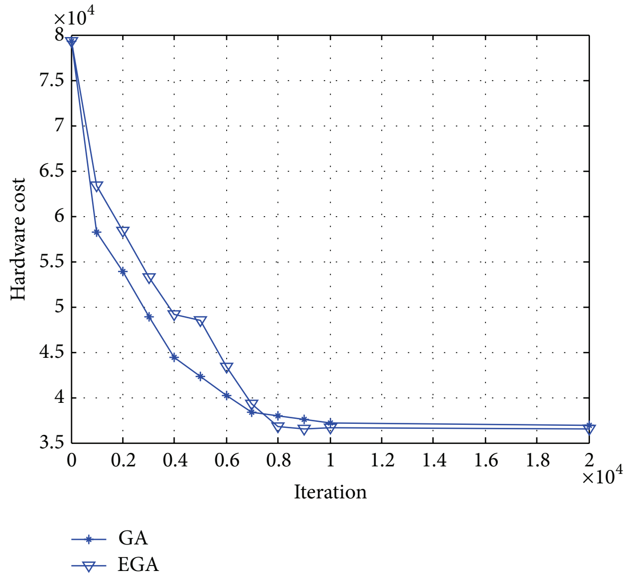

FIGURE 5: Convergence track for number of nodes equals 1000.

The simulation results of the proposed algorithms as well as the original genetic algorithm are presented in Figures 3 and 4. Each instance is tested for 100 times and the average values are presented. The first graph demonstrates the accuracy of the proposed algorithm and the second graph demonstrates the efficiency of the proposed algorithm. Furthermore, we collect the convergence track and the run time of the two algorithms.

The values about the cost value are shown in Figures 3 and 4 for different parameters configurations. For those random graphs with the small size of nodes, the results of *EGA* and *GA* are almost the same. The two algorithms yield similar results. For bigger random graphs, *EGA* outperforms *GA*. *EGA* can always find smaller values than Algorithm 1. With the increase of the size, the deviation between the two algorithms grows bigger. The improved algorithm will keep better population size, and the local search will be more universal and accurate.

We also store the convergence track of the two algorithms, as presented in Figure 5. At the beginning of the iteration procedure, *GA* drops faster than *EGA*. But *EGA* can find the near optimal solution faster than *GA* in the convergence process. The iteration number grows with the size of the nodes, which means more time to go into the stable state. We also collect the minimum expectation cost value of the two algorithms. The appearance times of the minimum value of the two algorithms demonstrate that the *EGA* performs better than the *GA*, even for a small number of nodes.

As shown in the experiment results, we can find that the original genetic algorithm needs more time, which means more iterations to meet the termination conditions. Furthermore, the accuracy of the near-optimal solution got by the incorporated algorithm is higher. From the experiments, it is reasonable to draw the conclusion that our proposed algorithm produces high-quality approximate solution and generates the solution with faster speed.

6. Conclusion

In this paper, we construct a communication graph for the partitioning problem, in which the implementation cost, execution time, and communication time are all taken into account. Then, we propose a heuristic based on genetic algorithm and simulated annealing to solve the problem near optimally, even for quite large systems. The proposed heuristic method incorporates simulated annealing in genetic algorithm. Those incorporated accepted rules based on fitness and Metropolis criterion speed up the convergence of the solution process without loss of accuracy. Experiment results show that the proposed model and algorithm produce more accurate partitions with faster speed.

Acknowledgments

This work was supported by the National Medium and Long-term Development Plan (Grant no. 2010ZX01045-002-3), the 973 Program of China (Grant no. 2010CB328000), the National Natural Science Foundation of China (Grant nos. 61073168, 61133016, and 61202010) and by the National 863 Plan of China (Grant no. 2012AA040906).

References

[1] J. B. Wang, M. Chen, X. Wan, and C. Wei, "Ant-colony-optimization-based scheduling algorithm for uplink CDMA nonreal-time data," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 1, pp. 231–241, 2009.

[2] Y. Jiang, H. Zhang, X. Song et al., "Bayesian network based reliability analysis of plc systems," *IEEE Transactions on Industry Electronics*, 2012.

[3] J. B. Wang, M. Chen, and J. Wang, "Adaptive channel and power allocation of downlink multi-user MC-CDMA systems," *Computers and Electrical Engineering*, vol. 35, no. 5, pp. 622–633, 2009.

[4] D. 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.

[5] J. Henkel, "Low power hardware/software partitioning approach for core-based embedded systems," in *Proceedings of the 36th Annual Design Automation Conference (DAC)*, pp. 122–127, ACM, June 1999.

[6] R. Niemann and P. Marwedel, "An algorithm for hardware/software partitioning using mixed integer linear programming," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 165–193, 1997.

[7] Z. Mann and A. Orbán, "Optimization problems in system-level synthesis," in *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, Tokyo, Japan, 2003.

[8] R. P. Dick and N. K. 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.

[9] J. I. Hidalgo and J. Lanchares, "Functional partitioning for hardware-software codesign using genetic algorithms," in *Proceedings of the 23rd EUROMICRO Conference*, pp. 631–638, September 1997.

[10] K. S. 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.

[11] J. Wang and X. Xie, "Optimal odd-periodic complementary sequences for diffuse wireless optical communications," *Optical Engineering*, vol. 51, no. 9, Article ID 095002, 2012.

[12] W. Jigang, B. Chang, and T. Srikanthan, "A hybrid branch-and-bound strategy for hardware/software partitioning," in *Proceedings of the 8th IEEE/ACIS International Conference on Computer and Information Science (ICIS '09)*, pp. 641–644, June 2009.

[13] 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, 2006.

[14] P. V. Knudsen and J. Madsen, "PACE: a dynamic programming algorithm for hardware/software partitioning," in *Proceedings of the 4th International Workshop on Hardware/Software Co-Design (Codes/CASHE '96)*, pp. 85–92, IEEE Computer Society, March 1996.

[15] J. Madsen, J. Grode, P. V. Knudsen, M. E. Petersen, and A. Haxthausen, "Lycos: the lyngby co-synthesis system," *Design Automation for Embedded Systems*, vol. 2, no. 2, pp. 195–235, 1997.

[16] 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.

[17] A. Kalavade, *System-level codesign of mixed hardware-software systems [Ph.D. thesis]*, University of California, Berkeley, 1995.

[18] R. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computers*, vol. 10, no. 3, pp. 29–41, 1993.

[19] R. Niemann and P. Marwedel, "Hardware/software partitioning using integer programming," in *Proceedings of the European Design & Test Conference*, pp. 473–479, IEEE Computer Society, March 1996.

[20] F. Vahid and D. D. Gajski, "Clustering for improved system-level functional partitioning," in *Proceedings of the 8th International Symposium on System Synthesis*, pp. 28–33, September 1995.

[21] F. Vahid, J. Gong, and D. D. Gajski, "Binary-constraint search algorithm for minimizing hardware during hardware/software partitioning," in *Proceedings of the European Design Automation Conference*, pp. 214–219, IEEE Computer Society, September 1994.

[22] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test of Computers*, vol. 10, no. 4, pp. 64–75, 1993.

[23] 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.

[24] L. Li, Y. Song, and M. Gao, "A new genetic simulated annealing algorithm for hardware-software partitioning," in *Proceedings of the 2nd International Conference on Information Science and Engineering (ICISE '10)*, IEEE Computer Society, December 2010.

[25] L. Y. Li and M. Shi, "Software-hardware partitioning strategy using hybrid genetic and Tabu search," in *Proceedings of the International Conference on Computer Science and Software Engineering (CSSE '08)*, pp. 83–86, December 2008.

[26] S. Zheng, Y. Zhang, and T. He, "The application of Genetic Algorithm In embedded system hardware-software partitioning," in *Proceedings of the International Conference on Electronic Computer Technology (ICECT '09)*, pp. 219–222, February 2009.

[27] 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.

[28] T. Wiangtong, P. K. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware-software codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, pp. 425–449, 2002.

[29] P. Arató, Z. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10, no. 1, pp. 136–156, 2005.

[30] W. Jigang, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: 1D search algorithms," *IEEE Transactions on Computers*, vol. 59, no. 4, pp. 532–544, 2010.

[31] J. Albuquerque, C. Coelho Jr., C. F. Cavalcanti, D. C. da Silva Jr., and A. O. Fernandes, "System-level partitioning with uncertainty," in *Proceedings of the 7th International Conference on Hardware/Software Codesign (CODES '99)*, pp. 198–202, May 1999.

[32] J. Albuquerque, "Solving hw sw partitioning bystochastic linear programming with management of teams uncertainty".

[33] Y. Jiang, H. Zhang, X. Jiao et al., "Uncertain model and algorithm for hardware/software partitioning," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI '12)*, pp. 243–248, August 2012.

[34] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

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

[36] P. Arató, S. Juhasz, Z. Mann, A. Orbán, and D. Papp, "Hardware-software partitioning in embedded system design," in *Proceedings of the IEEE International Symposium on Intelligent Signal Processing*, pp. 197–202, September 2003.