Research Article

An Effective Heuristic-Based Approach for Partitioning

Xibin Zhao, Hehua Zhang, Yu Jiang, Songzheng Song, Xun Jiao, and Ming Gu

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 University 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

Abstract
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, 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 $f_i / \bar{f}$, where $f_i$ is the evaluation of individual $i$ and $\bar{f}$ 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.
Algorithm 1: Annealing algorithm.

```plaintext
(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) \( K := K + 1 \);
(6) call generation strategy for the next state \( j \);
(7) \[ \Delta E_{ij} = \text{cost(next state}_j) - \text{cost(current state)} \];
(8) \( P_{ij} = A_{ij} \);
(9) if \( (P_{ij} > \text{random}[0,1]) \) then
(10) \( \text{current state} := \text{next state}_j \);
(11) end if
(12) end while
(13) \( T_{k+1} := T_k \cdot \alpha \);
(14) end while
(15) return current state;
```

(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 \( \{v_1,v_2,...,v_n\} \) and \( E \) is the set of edges \( \{e_{ij} | 1 \leq i,j \leq n\} \). 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) \( t^h_i \) denotes the execution time of node \( i \) in hardware implementation, and \( t^s_i \) 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 \( V_h \cup V_s = V \) and \( V_h \cap V_s = \emptyset \). The partitioning problem can be represented by a decision vector \( x(x_1,x_2,...,x_n) \), 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 $n_g_1$;
(15) if (fitness($n_g_1$) ≤ fitness($g'_1$)) then
(16) Reject the mutation, $n_g_1 = g'_1$;
(17) else
(18) Accept the mutation;
(19) end if
(20) Perform the above steps on $g'_2$ to produce $n_g_2$;
(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.
Based on the definition of previous subsection, hardware cost \( H(x) \) of the partition \( P(x) \) and the total time metric \( T(x) \) can be formalized as follows:

\[
H(x) = \sum_{i=1}^{n} h_i \left( 1 - x_i \right),
\]

\[
T(x) = \sum_{i=1}^{n} t_i^p x_i + t_i^h \left( 1 - x_i \right) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left( x_i - x_j \right)^2.
\]

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:

\[
\begin{align*}
\text{minimize} & \quad H(x), \\
\text{subject to} & \quad T(x) \leq M \quad x \in \{0, 1\}^n,
\end{align*}
\]

\((P_1)\)

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

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{n} h_i x_i, \\
\text{subject to} & \quad \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} \left( x_i - x_j \right)^2 \\
& \quad + \sum_{i=1}^{n} \left( t_i^p - t_i^h \right) x_i \leq M - \sum_{i=1}^{n} t_i^h, \quad x \in \{0, 1\}^n.
\end{align*}
\]

\((P_2)\)

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 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) \( t_{ij}^h \) is randomly generated in \([1, 100]\), and \( t_{i}^f \) is randomly generated in \([t_{ij}^h, 200 + t_{ij}^h]\).
(iii) \( c_i \) is randomly generated in \([1, 20]\).
(iv) \( M \) is a given time constraint and randomly generated in \([\sum_{i=1}^n t_{ij}^h, \sum_{i=1}^n t_{i}^f]\).
The two algorithms demonstrate that the algorithms. The appearance times of the minimum value of the two nodes, which means more time to go into the stable state. We collect the minimum expectation cost value of the two algorithms. The iteration number grows with the size of the universal and accurate.

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


