# Uncertain Model and Algorithm for Hardware/Software Partitioning

Yu Jiang†, Hehua Zhang*, Xun Jiao‡, Xiaoyu Song§, William N. N. Hung¶, Ming Gu*, and Jiaguang Sun*
*School of Software, Tsinghua university, Beijing
†School of Computer Science and Technology, Tsinghua university, Beijing
§Dept. ECE, Portland State University, Oregon, USA.
¶Synopsys Inc., Mountain View, USA.
‡school of international studies, Beijing university of post and telecommunication, Beijing

***Abstract*—Embedded systems are becoming increasingly popular due to their widespread applications. Hardware/software partitioning is becoming one of the most crucial steps in the design of embedded systems. The costs and delays of the final results of a design will strongly depend on partitioning. In this paper, we propose an uncertain programming model for partitioning problems. The delay related constraints and the cost related objective are modeled by uncertain variables with uncertainty distributions. We convert the uncertain programming model to a deterministic model and solve the converted model by an efficient heuristic method. We propose a heuristic based on genetic algorithm and simulated annealing to solve the problem near-optimally, even for quite large systems. Experiment results show that the proposed model and algorithm produce quality partitions.**

***Keywords*-embedded system; hardware/software partitioning; uncertain programming; genetic algorithm; simulated annealing.**

## I. Introduction

Embedded system typically consists of some software components, and some hardware components implemented with application specific accelerator circuits. 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, performance 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 [1] and power consumption[2]. 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, complexity of partitioning arises because more requirements on cost, power, and timing performance have to be considered, and the structure of modern embedded system is more complex.

Many efforts about how to apply search and optimization methods to automate the partitioning task have been undertaken. Some exact algorithms such as branch-and bound [3], integer linear programming [4] and dynamic programming [5] are appropriate approaches. Those algorithms have been used for partitioning problem with small inputs successfully. Because most formulations of partitioning problem are NP-hard [6], those exact algorithms tend to be quite slow for large system.

To overcome the drawback of exact algorithms, researchers turn to more flexible and efficient heuristic algorithms. Traditional heuristic algorithms are software-oriented and hardware-oriented. The hardware-oriented algorithms start with a complete hardware implementation, and then iteratively move component to software until the given constraints are satisfied [7]. The software-oriented algorithms it with the same manner [8]. Many general-purpose heuristic algorithms are also utilized to solve the system partitioning problem. Simulated annealing related algorithms [9], genetic algorithms [10], tabu search and greedy algorithms [11] have been extensively used to solve partitioning problem. In addition to the general-purpose heuristic algorithms, some custom heuristics, such as expert system [12] and GCLP algorithm [13] are also appropriate for hardware-software partitioning problem.

Most of the algorithms work perfectly within their own co-design environments. But few algorithms address the issue that we can not accurately determine the cost and time of system components in the design stage. In order to reflect real applications better, we take the above factors into account. In this paper, we construct an uncertain programming model for the partitioning problem based on a task communication graph. The implementation cost, execution time and communication time of components are modeled to be uncertain variables with uncertain distributions. We construct a mathematical model based on these uncertain variables. Then, we can convert the uncertain programming to a deterministic model and solve the converted model with a heuristic. The proposed heuristic method incorporates simulated annealing into genetic algorithm to improve the accuracy and speed of original genetic algorithm. Generally speaking, the main contributions of the work are : (1) This is the first work on hardware/software partitioning with uncertain metrics of components; (2) We propose an enhanced genetic algorithm to solve the converted uncertain mathematical model.

978-0-7695-4767-1/12 $26.00 © 2012 IEEE
DOI 10.1109/ISVLSI.2012.14

The paper is organization as follows: some background on the uncertainty theory and genetic algorithms are introduced in Section II. The proposed uncertainty model for the partitioning problem is presented in Section III. Section IV presents the solution of the proposed model, including the enhanced genetic algorithm. Empirical results are given in Section V, and we conclude in Section VI.

## II. BACKGROUND

### A. Uncertainty Theory

Uncertainty theory was proposed and refined by Liu [14] in 2007, based on his long-run research on Fuzzy theory. Uncertainty theory has become an important branch of mathematics and has been applied to scheduling, reliability analysis, data mining and system control area. The first fundamental concept of the uncertainty theory is the uncertain measure that is used to measure the belief degree of an uncertain event. The second concept is the uncertain variable used to represent imprecise quantities of uncertain events. The third concept is the uncertainty distribution that is used to describe uncertain variables in an incomplete but easy-to-use way. The three basic concepts are defined using the following notations: $\Gamma$ is a nonempty set, $L$ is a $\sigma$-algebra over $\Gamma$, each element $\Lambda$ in the $\sigma$-algebra $L$ ia called an event. It is necessary to assign each event $\Lambda$ with a number $M\{\Lambda\}$ which indicates the belief degree of the event's occurrence.

**Definition 1 (Uncertain measure):** A set function $M$ is called an uncertain measure if it satisfies the normality, self-duality, and countable sub-additivity axioms.

$$\begin{aligned} M\{\Gamma\} &= 1 && normality \\ M\{\Lambda\} + M\{\Lambda^c\} &= 1 && duality \\ M\{\bigcup_{i=1}^{\infty} \Lambda\} &\leq \sum_{i=1}^{\infty} M\{\Lambda_i\} && additivity \end{aligned}$$

**Definition 2 (Uncertain variable):** If $M$ is an uncertain measure, the triple $(\Gamma, L, \Lambda)$ is called an uncertain space. An uncertain variable $\xi$ is a measurable function from an uncertain space to the set of real numbers.

**Definition 3 (Uncertainty distribution):** The uncertainty distribution $\Phi$ of an uncertain variable $\xi$ is defined by $\Phi(x) = M\{\xi \leq x\}$ for any real number x. Many special uncertainty distributions and the corresponding properties are listed in [14].

### B. 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 [15], 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.

- **Selection** is applied to the current generation to create an intermediate generation. The probability that the individual in current generation is copied to the intermediate generation depends on its fitness. The individual with greater fitness will be copied to the intermediate generation with higher probability.
- **Crossover** takes two individuals from the intermediate generation, and recombine the two individuals with a probability $P_c$. The single point crossover method cuts the two individuals' encoding strings at some chosen position, and swaps the tail segments of the two strings.
- **Mutation** chooses an individual from the results of the previous step, and alters one bit in the individual's encoding string with a low probability $P_m$. Then, the final next generation is generated.

Then, we can evaluate individuals of the next generation with the fitness function, deciding whether to stop or go on performing the three operations. Finally, the algorithm will return the best individual of the latest generation as the solution of the problem.

### C. Simulated Annealing

Simulated annealing algorithm is a generic probabilistic meta-heuristic for the global optimization problem, locating a good approximation to the global optimum of a given function. It is proposed by Kirkpatrick [16], based on the analogy between the solid annealing and the combinatorial optimization problem. For each temperature value $T$, the probability that the particles are allowed to be in a state $r$ satisfies the Boltzmann distribution:

$$P\{\overline{E} = E(r)\} = \frac{1}{Z(T)} exp\big(-\frac{E(r)}{k_B T}\big)$$

$$Z(T) = \sum_{s \in D} exp\big(-\frac{E(s)}{k_B T}\big)$$

where $\overline{E}$ is a stochastic variable associated with the energy, $E(r)$ represents the energy, $k_B$ is the Boltzmann constant, $Z(T)$ is the normalization factor of the probability distribution, and D is the neighborhood of state s.

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 can be performed:

- **Generation** is applied to generate the candidate of the next state. There are many strategies for the selection of next state. If the next state is chosen from the neighborhood with equiprobable chance, the generation probability $G_{ij}(T)$ that state j is chosen as

the next state of i is defined as:

$$G_{ij}(T) = \begin{cases} 1/|N(i)| & j \in N(i) \\ 0 & j \notin N(i) \end{cases}$$

where $|N(i)|$ is all neighbors of state i.

- **Acceptation** decides whether to accept the new state or not, according to the Metropolis rule. The acceptance probability is defined as:

$$A_{ij} = \begin{cases} 1 & E(i) \geq E(j) \\ exp(-\bigtriangleup E_{ij}/T) & E(i) < E(j) \end{cases}$$

where $\bigtriangleup E_{ij}$ equals to $E(j) - E(i)$.

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.

## III. UNCERTAINTY MODEL AND FORMULATION

We can formalize the problem as follows. The communication graph is denoted as $G(V, E)$, where $V$ is the set of nodes $\{v_1, v_2 \cdots v_n\}$ 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 $E$:

1) $\xi_i^h$ denotes the cost of node i in hardware implementation.
2) $\Phi_{ci}^h$ is the linear uncertainty distributions of uncertain variables $\xi_i^h$, denoted by $\zeta(a_{ci}^h, b_{ci}^h)$, where $a_{ci}^h$, $b_{ci}^h$ are nonnegative real numbers. $\Phi_{ci}^h$ can be set as
3) $t_i^h$ denotes the execution time of node i in hardware implementation, and $t_i^s$ denotes the execution time of node i in software implementation.
4) $\Phi_{ti}^h$, $\Phi_{ti}^s$ are uncertain distributions of uncertain variables $T_i^h$, $T_i^s$, denoted by $\zeta(a_{ti}^h, b_{ti}^h)$, $\zeta(a_{ti}^s, b_{ti}^s)$, where $a_{ti}^h$, $b_{ti}^h$, $a_{ti}^s$, $b_{ti}^s$ are nonnegative real numbers.
5) $c_{ij}$ denotes the communication time between node i, j. The value of $c_{ij}$ is given in the context that the two nodes are implemented differently.

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 $\mathbf{x}(x_1, x_2 \cdots x_n)$, representing the implementation way of the n task modules. When the value of $x_i$ is 0, the task module will be implemented in hardware. When the value of $x_i$ is 1, the task module will be implemented in software. Then, the objective of the problem is changed to search the n-dimensional space to find the optimal value of the decision vector.

First, let us consider the cost metric. If a given node is partitioned to be in hardware implementation, the hardware cost of the node is considered. Otherwise, the software cost of the node is considered. Then, the total hardware cost with respect to a dedicated partition can be calculated as the sum of nodes in hardware implementation. The additive calculation rule for resource consumption cost is reasonable in most cases of the computation model. Based on the definition of previous subsection, hardware cost $H(\mathbf{x})$ of the partition $P(\mathbf{x})$ can be formalized as follow:

$$H(\mathbf{x}) = \sum_{i=1}^{n} \xi_i^h (1 - x_i)$$

Then, let us consider the time metric. It consists of two parts: execution time of each node, and the communication time between nodes. We give two reasonable assumptions to go on our calculation: (1) the task module can not be processed in parallel, and (2) the communication cost between node i, j is 0 when the two nodes are partitioned into the same implementation way. Based on the two assumptions, the execution time $T_e(\mathbf{x})$, communication time $T_c(\mathbf{x})$, and the total time metric $T(\mathbf{x})$ can be formalized as follows:

$$\begin{aligned} T_e(\mathbf{x}) &= \sum_{i=1}^{n} t_i^s x_i + t_i^h (1 - x_i) \\ T_c(\mathbf{x}) &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} [(x_i - x_j)^2] \\ T(\mathbf{x}) &= T_e(\mathbf{x}) + T_c(\mathbf{x}) \end{aligned} \quad (1)$$

The formulas defined above ensure that either hardware execution time or software execution time is counted once. The communication cost is accumulated when the value of $x_i$ and $x_j$ are different from each other. When the values are the same, the communication cost is ignored by multiplying 0.

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:

$$P_0 : \begin{cases} minimize & H(\mathbf{x}) \\ subject\ to & T(\mathbf{x}) \leq M \\ & \mathbf{x} \in \{0, 1\}^n \end{cases}$$

We take a look at the objective and constraint function. Both of them can be further simplified:

$$\begin{aligned} H(\mathbf{x}) &= \sum_{i=1}^{n} \xi_i^h - \sum_{i=1}^{n} \xi_i^h x_i \\ T(\mathbf{x}) &= T_c(\mathbf{x}) + \sum_{i=1}^{n} t_i^h + \sum_{i=1}^{n} (t_i^s - t_i^h) x_i \end{aligned}$$

Based on the simplified expressions, we note that minimizing the value of $H(\mathbf{x})$ is equivalent to maximizing the value of $\sum_{i=1}^{n} h_i x_i$. Hence, the solution of the problem $P_0$ is equal

to the problem $P_1$ presented below.

$$P_1 : \begin{cases} maximize & \sum_{i=1}^{n} \xi_i^h x_i \\ subject\ to & \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij}[(x_i - x_j)^2)] + \\ & \sum_{i=1}^{n} (t_i^s - t_i^h) x_i \le M - \sum_{i=1}^{n} t_i^h \\ & \mathbf{x} \in \{0,1\}^n \end{cases}$$

Then, we convert the optimal model with uncertain variables to a deterministic model. The template for the conversion is presented below:

$$\left\{ \begin{array}{l} max\ \mathbf{f}(\mathbf{x}, \xi) \\ subject\ to: \\ \quad g_j(\mathbf{x}, \xi) \le 0 \end{array} \right\} \rightarrow \left\{ \begin{array}{l} max\ E[\mathbf{f}(\mathbf{x}, \xi)] \\ subject\ to: \\ \quad M\{g_j(\mathbf{x}, \xi) \le 0\} \ge a \end{array} \right\}$$

where $a$ is the confidence level that the constraint holds. More details about the conversion model can be found in [14], [17]. Based on the details of the conversion rules and theorems proposed by Liu, the uncertain optimal problem $P_1$ can be converted. In the modeling and conversion process, we deal with most of the uncertain effects, such as different implementation cost, implementation related execution time and communication time.

## IV. ALGORITHM

In this section, we propose an algorithm to solve the partitioning problem $P_1$ based on genetic algorithm and simulated annealing algorithm. We find that the special case of problem $P_1$ (ignoring the communication cost) can be solved by efficient greedy based algorithm. Genetic algorithm has been applied to solve the general partitioning problem in [18] and the results show that the algorithm provides near-optimal solution for large size of problems.

We note that the genetic algorithm has a strong global search capability, while the simulated annealing algorithm will fail into a local optimal solution easily. Hence, we can incorporate simulated annealing algorithm into genetic algorithm. The combined algorithm will provide more accurate near-optimal solution with faster speed. The overall methodology of the proposed combined algorithm is illustrated in Fig. 1. The methodology starts with a set of initial arbitrary individuals and a certain temperature. In each iteration, fitness of each individual is evaluated. Termination conditions such as generation number limit and convergence of algorithm to a predefined fitness value are checked. If the conditions are not met, the selection, incorporated crossover with Metropolis criterion, and incorporated mutation criterion with Metropolis criterion are performed to generate the next generation. The new generation and the temperature will be reevaluated until the termination conditions are met. The detail of the algorithm is shown in Algorithm 1.

The steps 1-4 are the initialization of parameters and solution of the partition problem. The step 5 is used to check

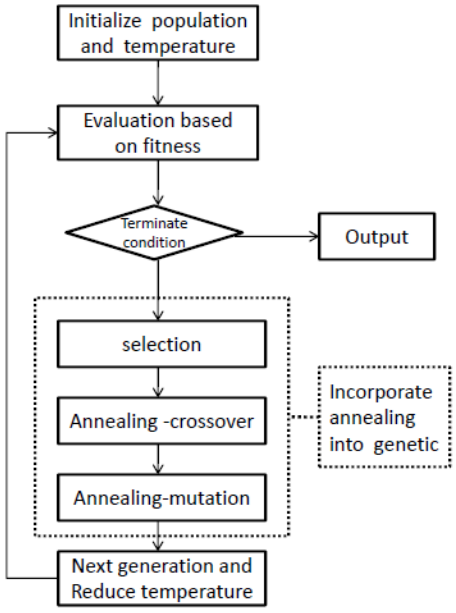


Figure 1. Main procedures for the combined algorithm

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 $\le$ 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 $(g_1^{'}, g_2^{'})$;
9: **if** (max{fitness($g_1^{'}$), fitness($g_2^{'}$)} $\le$ max{fitness($g_1$), fitness($g_2$)}) **then**
10: $\bigtriangleup$ C = max{fitness($g_1^{'}$), fitness($g_2^{'}$)} - max{fitness($g_1$), fitness($g_2$)};
11: **if** (min{1, exp(-$\bigtriangleup C/T_k$)} $\ge$ 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**
19: Perform mutation on $g_1^{'}$ to produce $ng_1$;
20: **if** (fitness($ng_1$) $\le$ fitness($g_1^{'}$)) **then**
21: $\bigtriangleup$ C = (fitness($ng_1$) - fitness($g_1^{'}$));
22: **if** (min{1, exp(-$\bigtriangleup C/T_k$)} $\ge$ 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**
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 $\ge$ 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: $x[i]$, $i \in [1, n]$;

Figure 2. ALGORITHM 1: Combined heuristic algorithm

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 annealing-crossover and annealing-mutation operations are performed in this iteration block to produce individuals of the next generation. Let us see the two operations in detail. The steps 8-18 are the original crossover operation incorporated with 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 with 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 accept rules based on fitness and Metropolis criterion. Those accept rules speed up the convergence of the solution process without lossy of accuracy. The steps 32-36 are the update of solution, generation number and temperature.

## V. EMPIRICAL RESULTS

The proposed algorithm has been implemented in C. We test the algorithm on Intel i5 2.27GHZ PC. In order to demonstrate the effectiveness of the proposed algorithm, we compare it with dynamic programming PACE [19], [20] and genetic algorithm GA [18]. For testing, several random instances with different nodes and metrics are utilized. The parameters of each node are generated with the following rules, which are consistent to [18]:

- For all $i \in [1, n]$: $a^h_{ci}$ is randomly generated in [1, 100] and $b^h_{ci}$ is randomly generated in $[a^h_{ci}, 100]$.
- For all $i \in [1, n]$: $a^h_{ti}$ is randomly generated in [1, 100] and $b^h_{ti}$ is randomly generated in $[a^h_{ti}, 100]$; $a^s_{ti}$ is randomly generated in [1, 100] and $b^s_{ti}$ is randomly generated in $[a^s_{ti}, 100]$.
- $c_{ij}$ is randomly generated in [1, 100].
- $M$ is a randomly given time constraint between $[\sum_1^n a^h_{ti},\ \sum_1^n b^s_{ti}]$.

After the parameters are initialized, the optimal solution of those instances can be obtained by the exact algorithms presented in the introduction section. Therefore, the performance of the algorithm can be easily determined. We simulate the proposed algorithm with the configure: the upper bound of the iterations equals to five times of the number of the task modules, the initial temperature equals to 10, and annealing ratio equals to 0.8, and the termination conditions is no better solution obtained. The simulation results of the proposed algorithms as well as the original genetic algorithm are presented in the TABLE I. Each instance is tested for 30 times and the average values are presented.

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. Further more, the accuracy of the near-optimal solution get by the incorporated algorithm is higher. It is reasonable to draw the conclusion that our proposed algorithm produces high quality approximate solution, and generates the solution with faster speed.

## VI. CONCLUSION

In this paper, we build a communication graph for the partitioning problem, and construct an uncertain programming model to capture real-world applications for partitioning. Since the resource cost and the execution time of components in different implementations can not be characterized exactly at the design stage, we model them as uncertain variables with uncertainty distributions. Timing constraints and cost objective functions are defined on those uncertain variables. Then, we convert the uncertain programming model to a deterministic model and solve it with a heuristic based on genetic algorithm and simulated annealing. The proposed heuristic method incorporates simulated annealing into genetic algorithm. Those incorporated accept rules based on fitness and Metropolis criterion speed up the convergence of the solution process without lossy of accuracy. We also conduct some experiments to demonstrate the effectiveness of the proposed model and algorithms. Further more, this is the first work on hardware-software partitioning problems considering uncertain metrics. And in the future, we want to focus on the algorithm aspect of the uncertain partitioning model, doing some improvements of the presented algorithm to get higher accuracy.


### ACKNOWLEDGMENT

This research is supported in part by 973 Program (No.2010CB328003) of China.


### REFERENCES


[1] 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.

[2] 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.

[3] 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.

[4] 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.

Table I
EXPERIMENT RESULTS ABOUT THE OBJECTIVE VALUE AND THE SOLUTION TIME

| problem size | minimum cost | | | solution time | |
|---|---|---|---|---|---|
| nodes | Combined Algorithm | exact value | GA | Combined algorithm | GA |
| 100 | 436 | 441 | 438 | 0.1 | 0.8 |
| 200 | 668 | 675 | 672 | 0.1 | 1.5 |
| 300 | 782 | 790 | 786 | 0.1 | 2.2 |
| 400 | 932 | 941 | 938 | 0.1 | 3.0 |
| 500 | 1008 | 1014 | 1012 | 0.1 | 3.6 |
| 600 | 1332 | 1339 | 1337 | 0.3 | 4.1 |
| 700 | 1497 | 1508 | 1504 | 0.4 | 4.3 |
| 800 | 1627 | 1638 | 1633 | 0.6 | 4.6 |
| 900 | 1875 | 1889 | 1882 | 0.8 | 5.1 |
| 1000 | 2165 | 2178 | 2173 | 0.9 | 5.4 |
| 1100 | 436 | 441 | 438 | 0.1 | 0.8 |
| 1200 | 668 | 675 | 672 | 0.1 | 1.5 |
| 1300 | 782 | 790 | 786 | 0.1 | 2.2 |
| 1400 | 932 | 941 | 938 | 0.1 | 3.0 |
| 1500 | 2876 | 2892 | 2885 | 1.6 | 8.2 |
| 1600 | 1332 | 1339 | 1337 | 0.3 | 4.1 |
| 1700 | 1497 | 1508 | 1504 | 0.4 | 4.3 |
| 1800 | 1627 | 1638 | 1633 | 0.6 | 4.6 |
| 1900 | 1875 | 1889 | 1882 | 0.8 | 5.1 |
| 2000 | 3840 | 3859 | 3852 | 2.8 | 11.3 |
| 3500 | 5263 | 5288 | 5278 | 19.5 | 703.9 |
| 5000 | 8348 | 8392 | 8378 | 30.4 | 1165.2 |


[5] 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.

[6] A.Kalavade, "System-level codesign of mixed hardware-software systems," PHDTHESIS, University of California, Berkeley, 1995.

[7] R. Niemann and P. Marwedel, "Hardware/software partitioning using integer programming," in *Proceedings of the 1996 European conference on Design and Test*. IEEE Computer Society, 1996, p. 473.

[8] F. Vahid and D. Gajski, "Clustering for improved system-level functional partitioning," in *Proceedings of the 8th international symposium on System synthesis*. ACM, 1995, pp. 28–35.

[9] 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.

[10] 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.

[11] 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.

[12] M. López-Vallejo and J. López, "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.

[13] 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.

[14] B. Liu, "Uncertainty theory," *Uncertainty Theory*, pp. 1–79, 2010.

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

[16] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, p. 671, 1983.

[17] B. Liu, "Theory and practice of uncertain programming," *Uncertainty Theory*, pp. 1–134, 2002.

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

[19] P. 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*. IEEE Computer Society, 1996, p. 85.

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