HARDWARE/SOFTWARE PARTITIONING ALGORITHM BASED ON THE COMBINATION OF GENETIC ALGORITHM AND TABU SEARCH

G. Li[1*] – J. Feng[1] – C. Wang[1] – J. Wang[2]

[1]Aeronautics and Astronautics Engineering College, Air Force Engineering University, Xi'an 710038; China
[2]Aviation University of Air Force, Changchun 130022, China

ARTICLE INFO

***Article history*:**
Received: 18.12.2013.
Received in revised form: 8.2.2014.
Accepted: 18.2.2014.

Keywords:
Hardware/software partitioning
Hardware orientation
Genetic algorithm
Tabu search

Abstract:

To solve the hardware/software (HW/SW) partitioning problem of a single Central Processing Unit (CPU) system, a hybrid algorithm of Genetic Algorithm (GA) and Tabu Search(TS) is studied. Firstly, the concept hardware orientation is proposed and then used in creating the initial colony of GA and the mutation, which reduces the randomicity of initial colony and the blindness of search. Secondly, GA is run, the crossover and mutation probability become smaller in the process of GA, thus they not only ensure a big search space in the early stages, but also save the good solution for later browsing. Finally, the result of GA is used as initial solution of TS, and tabu length adaptive method is put forward in the process of TS, which can improve the convergence speed. From experimental statistics, the efficiency of proposed algorithm outperforms comparison algorithm by up to 25% in a large-scale problem, what is more, it can obtain a better solution. In conclusion, under specific conditions, the proposed algorithm has higher efficiency and can get better solutions.

1 Introduction

HW/SW partitioning technology is a crucial step in System on Chip (SoC) HW/SW codesign and embedded system realization, that is, deciding which components of the system should be realized in hardware and which ones in software, finally providing the best compromise for the system while satisfying the design constrains. Clearly, this step has a dramatic impact on the cost and performance of the whole system.

Most formulations of the HW/SW partitioning problem have proven to be NP-hard [1], so exact algorithms tend to be quite slow for bigger inputs, hence for larger partitioning problem, heuristic algorithms comprise the majority of the research and much significant research has been done such as Genetic Algorithm(GA) [2], Particle Swarm Optimization(PSO)[3],[4], Tabu Search(TS) [5],[6], Ant Algorithm(AA) [7],[8], Simulated Annealing(SA) [9] as well as some improved schemes.

Researchers have combined two of these algorithms and designed hybrid algorithms for optimal solution of a partitioning problem, for instance, paper [10] proposed a hybrid algorithm of GA and AA which

[*] Corresponding author. Tel.: +86 029 8478 7514; fax: + 86 029 8478 7514

utilized the advantages of the two algorithms to overcome their disadvantages; it achieved good results in a HW/SW partitioning problem; paper [11] proposed an idea of combining GA and PSO, which took advantage of their respective merits; PSO has fast convergence speed and GA is easy to express in solving a combinatorial optimization problem. Obviously, both the partitioning result and execution time of the algorithm have all been improved.

In particular, the combination of GA and TS can take advantage of the global search capability of GA and local search capability of TS while avoiding their defects [12][13],[14]. Up to now, there have been two kinds of hybrids of GA and TS [15][16]. The former, GA calls TS for local searching for every current solution every time after crossover and mutation; in this case, the frequent call of TS greatly prolongs computation time. The latter, GA is firstly run to produce a hypo-optimal solution that is used as initial solution of TS. On this basis, TS is run to find an optimal solution, and this condition brings some advantages to GA and TS that can be fully used, moreover, they can yield algorithms with smaller complexity.

To improve the partitioning quality and algorithm efficiency, we propose a partitioning algorithm based on GA and TS. In this paper, the concept of hardware orientation is put forward and used in the process of producing initial colony and mutation, avoiding thus not just the blindness of creating initial colony through a random method but also controlling the mutation direction. Furthermore, we design an adaptive change method for crossover/mutation probability and tabu length. Experimental results demonstrate the superiority of the proposed approach over the existing algorithm in terms of efficiency and solution quality.

This paper is organized as follows. In section 2 we introduce the HW/SW partitioning problem and provide the objective function of this paper. In section 3, we propose the concept of hardware orientation and its calculation. In section 4, we describe the details of the proposed hybrid algorithm. In section 5, we show the experimental results and analyses, and then compare the proposed algorithm with existing algorithms. Finally, section 6 draws the conclusions about our work and makes some prediction for our future work.

2 Problem description

HW/SW partitioning is one of the most crucial steps in the design of embedded systems. Before partitioning, it is important to identify the construction of the implementation platform. This paper discusses platforms with single CPU, that is, the system consists of one CPU and FPGA or other reconfigurable logic modules. The assignment of HW/SW partitioning is to distribute the tasks among CPU and hardware under certain constraints.

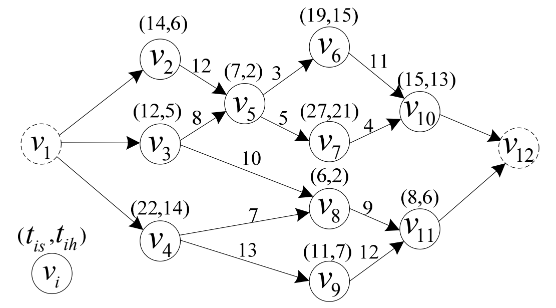

Figure 1. DAG model-based system.

We now formalize the problem as follows. The system to be partitioned is given in the form of a directed acyclic graph (DAG) just as it is shown in Fig. 1, an undirected graph $G=(V,E)$, $V=\{v_1, v_2, ..., v_n\}$ denotes the node aggregate, while v_i denotes the i-th task node, E is the edge aggregate, c_{ij} is the communication cost between nodes i and j, t_{is} and t_{ih} are execution time of task through software and hardware respectively, s_{is} and s_{ih} denote the area cost of v_i through software and hardware, S_s represents software area constraint, S_h hardware area constraint, C is communication constraint, T_{cost} is execution time that is defined to be the sum of processing time of all tasks. Assume that the communication cost between two adjacent nodes carried out through hardware or software can be overlooked, the objective function can be formulated as the following minimization problem using the method in paper [11]: n denotes the number of node, x_i denotes how the node v_i is realized, s_{is} and s_{ih} are area cost of v_i realized through software and hardware, S_s and S_h are constraints of software area and hardware area, respectively.

$$\begin{cases} \min\left(T_{\cos t} = \sum_1^n \left(t_{ih}x_i + t_{is}\left(1-x_i\right)\right)\right) \\ s.t.\ \ \sum_1^n \left(s_{is}\left(1-x_i\right)\right) \le S_s \\ \sum_1^n \left(s_{ih}x_i\right) \le S_h \\ \sum_1^n \left(\sum_{k\in V_s} c_{ik}x_i + \sum_{k\in V_h} c_{ik}\left(1-x_i\right)\right) \le C \end{cases} \quad (1)$$

3 Calculation of hardware orientation

In this paper, hardware orientation is defined as superiority of task implementation through hardware over software, characterized by three metrics: Area, Time and Communication.

3.1 Area-hardware orientation

$$S_{iorien} = \begin{cases} A\left(1-B\right)+B & ,\ S_{cons} < S_{sum} \\ 1 & ,\ S_{cons} \ge S_{sum} \end{cases} \quad (2)$$

Because of the requirements for typesetting, in Equation (2), we use A instead of $(S_{max}-S_i)/(S_{max}-S_{min})$, B instead of $1-S_{cons}/S_{sum}$ Accordingly, S_i denotes the additional area of s_{ih} being greater than s_{is}, while S_{max} and S_{min} are respectively the maximum and minimum values of S_i, and S_{sum} is the total area cost when the whole of the nodes are realized through hardware, in addition to this, S_{cons} denotes area constraint.

3.2 Time-hardware orientation

$$T_{iorien} = \begin{cases} \dfrac{\left(t_{is}-t_{ih}\right)/t_{is} - T_{1\min}}{T_{1\max} - T_{1\min}} & ,\ t_{is} > t_{ih} \\ 1 - \dfrac{\left(t_{ih}-t_{is}\right)/t_{ih} - T_{2\min}}{T_{2\max} - T_{2\min}} & ,\ t_{is} \le t_{ih} \end{cases} \quad (3)$$

Here t_{is} and t_{ih} denote the time cost of node i realized through software and hardware respectively, while $(t_{is} - t_{ih})/t_{ih}$ and $(t_{ih} - t_{is})/t_{ih}$ are performance ratios, T_{1max}, T_{1min}, T_{2max} and T_{2min} are the maximum and minimum values of the performance ratio.

3.3 Communication-hardware orientation

$$C_{isrorien} = \sum_j \left[\left(1-D\right)\times t_{sr} + D\times t_{hr}\right]c_{ij} \quad (4)$$

$$C_{isworien} = \sum_j \left[\left(1-D\right)\times t_{sw} + D\times t_{hw}\right]c_{ij} \quad (5)$$

$$C_{ihrorien} = \sum_j \left[\left(1-D\right)\times t_{hr} + D\times t_{sr}\right]c_{ij} \quad (6)$$

$$C_{ihworien} = \sum_j \left[\left(1-D\right)\times t_{hw} + D\times t_{sw}\right]c_{ij} \quad (7)$$

$$C_{iorien} = \frac{C_{isrorien} + C_{isworien}}{C_{ihrorien} + C_{ihworien}} \quad (8)$$

t_{sr}, t_{sw} , t_{hr} and t_{hw} are read and write delay of hardware and software, $D = S_{iorien} \times T_{iorien}$.
Consequently, the composite factor of hardware orientation can be described as：

$$Z_{iorien} = \frac{\alpha S_{iorien} + \beta T_{iorien} + \eta C_{iorien}}{S_{iorien} + T_{iorien} + C_{iorien}} \quad (9)$$

$\alpha + \beta + \eta = 1, \alpha > 0, \beta > 0, \eta > 0.$

4 Algorithm

The necessity and feasibility of hybrid mechanism of GA and TS have been analysed comprehensively [2]. Researchers have proposed and applied some hybrid methods [14][17],[18]. Here we mainly consider the following aspects: the creation of initial colony, the operation of selection, the operation of crossover and mutation, neighborhood structure, dynamic tabu length and tabu selection strategy.

4.1 Proposed GA process

(1) Coding. In a number of related articles, there are two familiar coding methods for GA, the binary and the decimal ones. By contrast, the binary-biased genetic algorithms have higher searching efficiency, less time-consuming for convergence, wider selecting domain of crossover and mutation probability and stronger robustness of optimized value than decimal-biased genetic algorithms [19]. Moreover, considering that the state of node here includes hardware and software realization, we choose binary as a coding mechanism. $X=(x_1, x_2, ..., x_n)$ denotes a partitioning plan, $x_i=1(x_i=0)$ means node, and v_i is carried out through hardware (software), $1 \le i \le n$.
(2) We make the reciprocal of objective function $F(X)=1/T_{cost}$ as fitness function in this paper.

(3) Creation of initial colony. In this paper, we create initial colony on the base of hardware orientation. Therefore, the bigger the hardware orientation is, the higher the probability of node is from initialization to hardware realization, and vice versa. Specifically, we first generate a random number $r_i \in (0,1)$, if $r_i < Z_{iorien}$, the node v_i is initialized through hardware or software. In addition, Hamming distance $H(X^i, X^j) = = \sum_{k=1}^{n} \left| X_k^i - X_k^j \right|$ is adopted for the difference among individuals and $H\left(X^i, X^j\right) > 4$, repeat the above operation until we have N_X individuals.

(4) Operation of selection. In order to prevent the precociousness phenomenon, the proposed algorithm selects individuals adaptively according to the change of fitness, and consequently, the selection probability of x_i can be defined using the method mentioned in [20]:

$$p_i = f'(x_i) \Big/ \sum_{i=1}^{n} f'(x_i) \tag{10}$$

$$f'(x_i) = af(x_i) + \frac{(f_{\max} - f_{\min})\left(e - e^{g/g_{\max}}\right)}{e + e^{g/g_{\max}}} \tag{11}$$

$f(x_i)$ expresses the fitness value of x_i, while $f_{\max}$ denotes the maximal value of $f(x_i)$ in current colony and $f_{\min}$ the minimal one, g is the number of iteration and $f_{\min}$ is its maximal value, a is a constant that is greater than zero and $a = 0.75$ in this paper.

According to this selection strategy, selection probability of individuals with big fitness values can be greatly reduced at the beginning of algorithm, which is beneficial for global searching; as regards the running of the algorithm, selection probability of individuals with big fitness values gradually grows bigger, which is beneficial for the convergence of algorithm.

(5) Crossover and mutation. In this process, some individuals ($N_x/2$) are selected for crossover using the two-point crossover method [11]. The selection of crossover probability P_c and mutation probability P_m, will influence the whole process of genetic algorithm. In other words, provided that there is the bigger difference between colony and fitness of individuals, the smaller P_c and P_m can help to protect individuals with bigger fitness; meanwhile, the convergence speed can also be improved. In case the difference between colony and fitness of individual is smaller, the bigger P_c and P_m can help to produce excellent individuals and prevent the algorithm from entering local optimum. Experiments have shown that an adaptive change of P_c and P_m can improve algorithm performance better than fixed value [21]. On the other hand, since the purpose of GA is to provide TS with global optimal solution, the value of P_c and P_m can be a little bit bigger. Thus, we put forward an adaptive method for crossover and mutation probability.

$$P_c = \begin{cases} p_{c1} \dfrac{f_{\max} - \max(f_i, f_j)}{f_{\max} - f_{avg}}, & \min(f_i, f_j) > f_{avg} \\ p_{c2}, & \min(f_i, f_j) \le f_{avg} \end{cases} \tag{12}$$

$$P_m = \begin{cases} p_{m1} \dfrac{f_{\max} - f}{f_{\max} - f_{avg}}, & f > f_{avg} \\ p_{m2}, & f \le f_{avg} \end{cases} \tag{13}$$

f_{avg} is average fitness of all the current individuals, while f is fitness of individual waiting for mutation, f_i and f_j represent fitness of individuals that are to be crossed, p_{c1}, p_{c2}, p_{m1} and p_{m1} are constants. If the individual fitness is smaller than f_{avg}, bigger P_c and P_m should be selected for promoting melioration of its fitness, whereas, smaller P_c and P_m should be selected for preserving the individual with bigger fitness. Considering that the individuals with bigger fitness should have smaller crossover probability, P_c uses $\min(f_i, f_j)$ as boundary.

(6) Termination criterion. For the sake of ending GA and running TS at the right time, the proposed algorithm uses dynamic termination criterion. Actually, we define the largest number of iteration $Gene_{\max}$ and the minimum evolution rate $GeneImproRat_{\min}$= 4%; if the evolution rate in three successive colonies is not larger than $GeneImproRat_{\min}$ or $g = g_{\max}$, GA would terminate.

The pseudo code of GA is denoted as shown in Table1.

Table 1. GA process

```
Input:
    Task graph $G$ and constrains $S_s$, $S_h$, $C$.
Output:
    The HW/SW partitioning result $X = (x_1, x_2, ..., x_n)$ and runtime.

1: begin
2: Calculate the comprehensive factor of hardware orientation $Z_{iorien}$ and set termination criterion;
3: Create initial colony on basis of $Z_{iorien}$, make $g = 0$, $N_{GeneImproRat,\min} = 0$; // $N_{GeneImproRat,\min}$ denotes the
   successive generations that the evolution rate is smaller than $GeneImproRat_{\min}$.
4: Calculate fitness of individuals in $P(0)$ and the average fitness;
5: Perform the operation between 6 and 21 again and again before meeting the termination criterion;
6: Calculate the selection probability $p_i$ for every individual in $P(g)$;
7: for($k = 0$; $k < N_x$; $k = k + 2$)
8: {
9: Select two individuals on basis of $p_i$;
10: Create a random number $0 < u < 1$;
11: if($u < P_m$)
12: Perform mutation operation for selected individuals, if $Z_{iorien} > 0.9$, no matter what state of the node
          is, the state is set to 1, otherwise, perform the routine mutation operation, the result is put into
          next colony;
13: elseif($u < P_m + P_c$)
14: Perform crossover operation and put the result into next colony;
15: else
16: put the individuals into next colony without change;
17: }//end for
18: g = g + 1, calculate the fitness of individuals in $P(g)$ and the average fitness;
19: if($GeneImproRat_{cur} \leq GeneImproRat_{\min}$)
20: $N_{GeneImproRat,\min} = N_{GeneImproRat,\min} + 1$;
21: end if
22: Output $X_{GA}$ which has the biggest fitness value in $P(g)$ and runtime;
23:end
```

4.2 Proposed TS process

(1) The initialization of TS. We use X_{GA} as initial input.
(2) Neighborhood structure. It is evident that the neighborhood structure has a significant impact on the quality of solution, a different rule may result in a different solution with different qualities [22]. In this work, the solution consists of a sequence of 0s and 1s; therefore, we can get the neighborhood solution by flipping two randomly selected nodes, that is to say, the state 0 is flipped into state 1, and vice versa. Obviously, every solution has C^2_n neighborhood solutions, $n > 3$, hence $C^2_n > n$. Finally, the n neighborhood solutions with bigger fitness values are selected to compose the neighborhood.

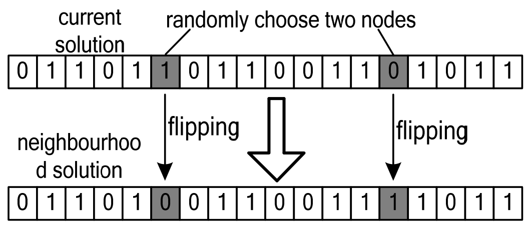

Figure 2. Neighbourhood structure method.

(3) Tabu table and tabu length. At the beginning, tabu length *TabuLen* is initialized through $\sqrt{n}$, meanwhile, a $n \times 2$ matrix $TabuFreq(n \times 2)$ is defined for recording tabu frequency of each neighborhood solution and g value when the solution has entered tabu table recently; every time a neighborhood solution enters tabu table, the corresponding value $TabuFreq(i,1)$ plus 1, and

TabuFreq(i,2) records simultaneously the g value..In addition, every time after iteration, we calculate the difference *TabuFreqDiff* for every element in *TabuFreq*(i,1) before and after iteration, then change the tabu length into *TabuLen* = *TabuLen* + *TabuFreqDiff*(*TabuLen* < n). In this article, a concept of tabu degree of $X_{neig}(i)$, denoted as *TabuDegree*(i), is defined as formula (14), because of the requirement for typesetting, we use B instead of *TabuDegree*, L instead of *TabuInteCur*, and U instead of *TabuFreq*:

$$B(i)=\frac{\eta_1\left(L-U(i,2)\right)+\eta_2 U(i,1)}{L-U(i,2)+U(i,1)} \tag{14}$$

In the equation, *TabuIterCur* denotes a current iteration number, and tabu degree is proportional to tabu frequency and the tabu interval *TabuIterCur* – *TabuFreq*(i,2). η_1 and η_2 are weighting factors, in this paper, $\eta_1 = 01.5$ and $\eta_2 = 0.85$.

(4) Selection strategy of TS. Assume that X_{cur} is a current solution whose neighborhood solution is $X_{neib}(i)$, we define $FObj(X_{neib}(i)) = F(X_{neib}(i)) - F(X_{cur})$, $F(X_{neib}(i))$ denotes fitness of $X_{neib}(i)$, while $F(X_{cur})$ denotes fitness of X_{cur}, the bigger $FObj(X_{neib}(i))$ is, the higher the quality of $X_{neib}(i)$ is.

(5) Aspiration criterion. If the neighborhood solution $X_{neib}(j)$ outperforms $X_{best\_so\_far}$ while $X_{neib}(j)$ is in tabu table, the tabu status of $X_{neib}(j)$ should be ignored and make $X_{best\_so\_far} = X_{neib}(j)$.

(6) Termination criterion of TS. The computation process will terminate either when the maximum number of iteration has been reached or when the evolution probability $TabuImproRat_{cur}$ is smaller than $TabuImproRat_{min}$ in three successive colonies, $TabuImproRat_{cur} = 0.3\%$ in this paper.

The pseudo code of TS is as shown in Table 2.

It is worthwhile pointing out that the solution refined by TS is definitely better than X_{GA}, because $X_{best\_so\_far}$ is updated only under condition that the better solution is found according to line 13 of Table 2.

5 Experiment and analysis

Creation of test set: For testing, firstly, create randomly several DAGs that have a specified node number and an average branch number, then allow every node to be associated with one function whose cost (hardware area, software area, communication cost and runtime etc.) is used to simulate task cost. Eventually we get 6 DAGs with 30, 60, 90, 120, 200, 400 nodes, respectively.

Experimental environment: (1) Pentium(R) Dual-Core 2.5GHz CPU, 2G internal storage; (2) Windows XP operating system; (3) Programming environment is Matlab R2007a.

TS has been proven to be better than GA in HW/SW partitioning [23]. To verify the effectiveness of a proposed algorithm, we choose TS [23] and GATS as comparison algorithms where TS is taken as the mutation operator [2]. Also, in order to make a fair comparison, all related parameters in our experiment are set on the same benchmark so that initial crossover probability is set to 0.8 and initial mutation probability to 0.13, what is more, TS has the same maximum number of iteration as the proposed algorithm but a constant tabu length $\sqrt{n}$.

Table 3 shows partitioning results of three algorithms. It can be observed that: (1) The proposed algorithm has higher convergence speed, because hardware orientation reduces randomicity of initial colony and affects the direction of search. To sum up, these two aspects reduce the number of iteration. Moreover, the strategies of tabu selection and adaptive tabu length also help to improve the search speed. (2) On small-scale problems when the number of node is less than 60, the proposed algorithm has lower efficiency than GATS, because the calculation of hardware orientation takes a long time. However, with an increase in the scale, when the number of nodes exceeds 90, the proposed algorithm can not only obtain a better solution but also improve operating efficiency by nearly 25%. Besides, the larger the scale is the better improvement is. The reason is that GATS must call TS in every iteration process and this will take much time, but the hardware orientation needs calculating only once.

(3) Compared with the other two algorithms, TS has the shortest runtime, whereas it only obtains solution with lowest quality, because TS has great dependence on initial solution; a good initial solution in turn could result in a good final solution, while a bad one will affect the quality of a final solution.

Table 2. TS process

```
Input:
    Task graph $G$ , constrains $S_s$, $S_h$, $C$ and $X_{GA}$;
Output:
The HW/SW partitioning result $X_{best\_so\_far}$ and runtime.

1: begin
2: Initialize the number of iteration $g_{Tabu}$ = 0, set tabu table empty, $TabuLen = \sqrt{n}$ , $TabuFreq(n,2)$ = 0, $N_{TabuImproRat,min}$ = 0; // $g_{Tabu}$ indicates the number of iteration, $N_{TabuImproRat,min}$ indicates the number of successive colonies whose evolution probability are all smaller than $TabuImproRat_{min}$.
3: $X_{cur} = X_{GA}$, $X_{best\_so\_far} = X_{GA}$;
4: Perform operation between 5 and 20 again and again before meeting the termination criterion;
5: Create the neighborhood of $X_{cur}$;
6: Calculate $FObj(X_{neib}(i))$ of every neighborhood solution;
7: (If $X_{neib}(i)$ is in tabu table and outperforms $X_{best\_so\_far}$, ignore its tabu status;)
8: If the whole neighborhood is in tabu table, then
9: $X_{cur}$ = the neighborhood solution corresponding to the smallest $TabuDegree(i)$;
10: else
11: $X_{cur}$ = the neighborhood solution corresponding to the biggest $FObj(X_{neib}(i))$;
12: end if
13: if $F(X_{cur}) > F(X_{best\_so\_far})$ then
14: $X_{best\_so\_far} = X_{cur}$
15: end if
16: Update tabu table, $TabuDegree(i)$, $TabuFreq(n\times 2)$ and $TabuLen$;
17: if($TabuImproRat_{cur} \le TabuImproRat_{min}$)
18: $N_{TabuImproRat,min} = N_{TabuImproRat,min} + 1$;
19: end if
20: $g_{Tabu} = g_{Tabu} + 1$;
21: Output $X_{best\_so\_far}$ and runtime of TS
22: end
```

To intuitively show experimental results, we run the three algorithms 30 times for the 6 DAGs respectively, then we calculate an average value of the results for each DAG, and finally we draw the comparison between proposed algorithm and the other two algorithms on partitioning results and runtime, as is shown in Fig.3 and Fig.4.

Assuming that there are 400 task nodes, we obtain initial solution using a random method and hardware orientation, respectively. The running results of GATS are shown in Fig. 5. As can be seen, an initial solution from hardware orientation can converge faster and yield better solutions. Fig. 6 shows the running results of GATS and our algorithm, intuitively illustrating the advantage of proposed algorithm on large-scale problems.

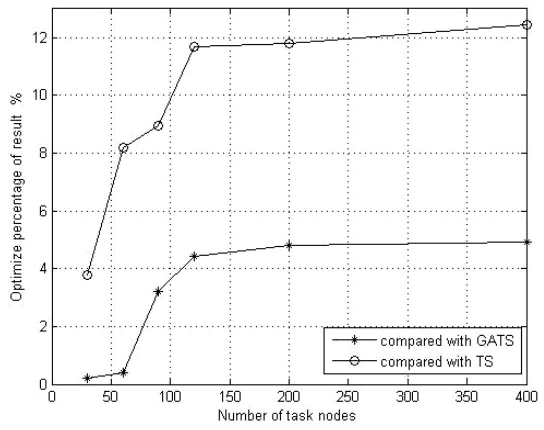

Figure 3. Optimization rate of partitioning result.

Table 3. Experimental results

Number of nodes	S_h	S_s	C	Algorithm	T_{cost}	Cost of time
30	2850	1409	591	TS	7318	92
				GATS	7086	988
				Our algorithm	7084	1035
60	5909	2943	1356	TS	14786	271
				GATS	13492	9743
				Our algorithm	13463	9862
90	8684	4308	2014	TS	22036	1361
				GATS	20795	67894
				Our algorithm	20143	58268
120	11611	5770	2637	TS	27903	1593
				GATS	26104	135102
				Our algorithm	24672	112298
200	17192	9416	3783	TS	46589	2175
				GATS	43136	330872
				Our algorithm	41065	264687
400	28154	15681	6539	TS	93961	3416
				GATS	87145	795691
				Our algorithm	82873	602338

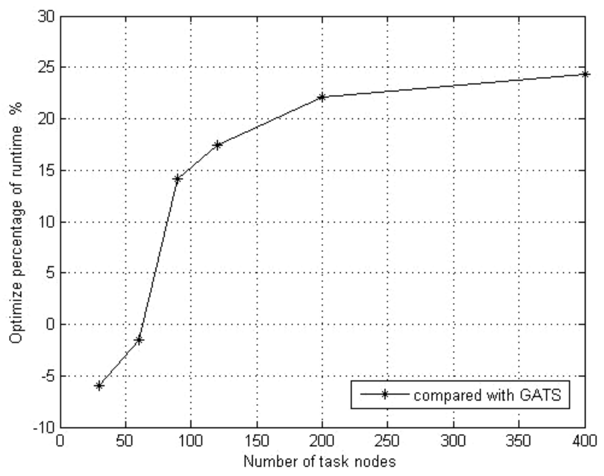

Figure 4. Optimization rate of runtime.

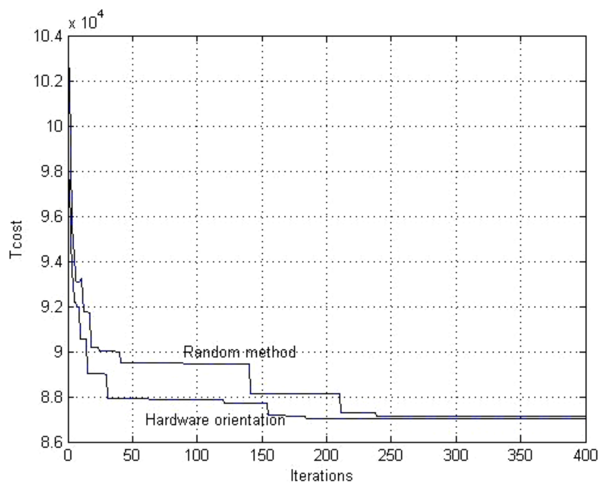

Figure 5. Running results of GATS with diffident initial solutions.

6 Conclusions

Based on GA and TS, this article presents a simple but very efficient hybrid algorithm for solving a HW/SW partitioning problem. Compared with the other two algorithms, time complexity of the proposed algorithm includes additional time for calculating hardware orientation, tabu length and tabu degree except for calculating genetic operation and fitness.

However, the application of hardware orientation and adaptive tabu length reduces an iterative number of algorithms and hardware orientation only needs to be calculated once. As a result, the proposed algorithm can improve algorithm efficiency especially on large-scale problems.

As the value of crossover and mutation probability in this paper is larger than classical GA, this may induce the blindness of search. However, the use of hardware orientation and adaptive technique prevent its occurrence. Furthermore, they increase the probability of introducing a new chromosome, which, to some extent, increases GA ability of local search.

In order to simplify problem, the proposed objective function does not take into account the power cost, which may impact partitioning accuracy. Ultimately, our on-going work will improve the objective function.

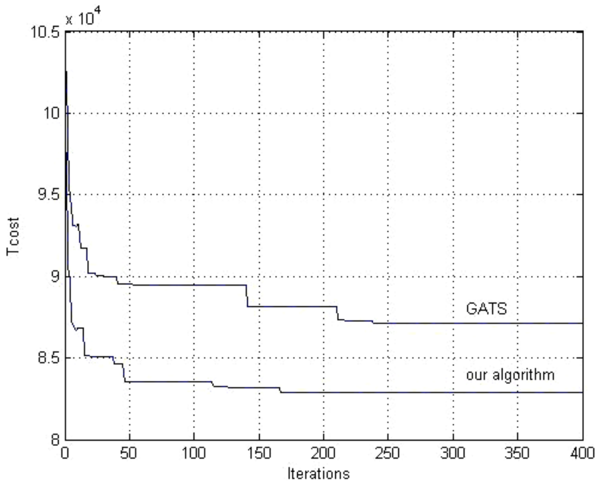

Figure 6. Running results of proposed algorithm and GATS.

References

[1] Arato P, Mann ZA, Orban A.: *Algorithmic aspects of hardware/software partitioning*, ACM Trans Des Autom Electron Syst, 10(2005), 1, 136–156

[2] Glover F, Kelly J.P., Laguna M.: *Genetic algorithms and tabu search: hybrids for optimization*, Computers and Operations Rsearch, 22(1995), 1, 111-134 .

[3] Wu, J., Srikanthan, T., Chen, G.: *Algorithmic aspects of hardware/software partitioning: 1D search algorithms*, IEEE Trans Comput, 59(2010), 4, 532–544.

[4] Li, S., Hsu, C., Wong, C., Yu, C.: *Hardware/software co-design for particle swarm optimization algorithm*, Information Sciences, (2011),181, 4582-4596.

[5] Wu, J., Thambipillai, S., Lei, T.: *Efficient heuristic algorithms for path-based hardware/software partitioning*, Mathematical and Computer Modelling. (2010), 51, 974-984.

[6] Wu, J., Wang, P., Lam, S., Srikanthan, T.: *Efficient heuristic and tabu search for hardware/software partitioning*, The Journal of Supercomputing. 66(2013), 1, 118-134.

[7] Zhang, Y., Wu, L., Wei, G., Wu, H., Guo, Y.: *Hardware/software partition using adaptive ant colony algorithm*, Control and Decision, 24(2009), 9, 1385-1389.

[8] He, T., Guo, Y.: *Power consumption optimization and delay based on ant colony algorithm in network-on-chip*, Engineering Review, 33(2013), 3, 219-225.

[9] Henkel J., Ernst R.: *An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation technique*s, IEEE Transactions on VLSI Systems, 9(2001), 2, 273–289.

[10] Xiong, Z., Li, S., Chen, J.: *Hardware/Software Partitioning Based on dynamic combination of genetic algorithm and ant algorithm*, Journal of Software, 16(2005), 4, 50-512.

[11] Liu, A., Feng, J., Liang, X., Yang, X.: *Algorithm of hardware/Software partitioning based on genetic particle swarm optimization*, Journal of Computer-Aided Design& Computer Graphic, 22(2010), 6, 927-933, 942.

[12] Li, Y.G., Abdul Ghafir, M. F., Wang, L., Singh, R.: *Improved multiple point nonlinear genetic algorithm based performance adaptation using least square method*, Journal of Engineering for Gas Turbines and Power, (2012), 134, 1-10.

[13] Fred G. , LuZH P., Hao J. K.: *Diversification-driven tabu search for unconstrained binary quadratic problems*, 4OR-A Quarterly J of Operations Research, 8(2010), 3, 239-253.

[14] Zhao, J., Zhou, H., Liang, C.: *Hybrid optimization algorithm based on genetic-tabu search for JLSP*, Systems Engineering and Electronics. 34(2012), 4, 833-838.

[15] Li, Z., Cheng, Y.: *A hybrid strategy based on genetic algorithm and tabu search importing niche*, Journal of Hunan University(Natural Sciences), 37(2010), 4, 81-84.

[16] Wang, W., Zhao, J., Wang, H.: *Design of orthogonal polyphase code for MIMO radar based on hybrid algorithm*, Systems Engineering and Electronics, 35(2013), 2, 294-298.

[17] Ji, Y., Li, L., Shi, M., Zhang, L.: *Hardware/software partitioning algorithm using hybrid genetic and tabu search*, Computer Engineering and Applications, 45(2009), 20, 81-83.

[18] Xiao, H., Lou, P., Wu, X., Qian, X.: *Unidirectional guided-path network design method based on hybrid genetic algorithm*, Computer Integrated Manufacturing Systems, 18(2012), 5, 1031-1037.

[19] Zhang, J., Li, D., Li, P. : *Comparative study of genetic algorithms encoding mechanism*, Journal of China University of Mining &Technology, 31(2002), 6, 637-640.

[20] Peng, Y., Luo, X., Wei, W.: *New fuzzy adaptive simulated annealing genetic algorithm*, Control and Decision, 24(2009), 6, 843-848.

[21] Zu, Y., Zhou, J.: *Cognitive radio resource allocation based on combined chaotic genetic algorithm*, Acta Phys.Sin, 60(2011), 7, 1-8.

[22] Wolf, W.H.: *An architectural co-synthesis algorithm for distributed embedded computing systems*, IEEE Transactions on VLSI Systems, 5(1997), 2, 218-229.

[23] Theerayod, W., Peter, Y. K.C., Wayne, L.: *Comparing three heuristic search methods for functional partitioning in hardware software codesign*, Design Automation for Embedded Systems, (2002), 6, 425-449.