RLDRPSO: An Efficient Heuristic Algorithm for Task Partitioning

Xiaofeng Qi(✉), Xingming Zhang, and Kaijian Yuan

National Digital Switching System Engineering and Technological R&D Center, Zhengzhou 450000, China
qxxxxf@gmail.com

Abstract. Task partitioning is the critical step in the co-design of reconfigurable embedded system. Particle Swarm Optimization (PSO) has been used for fast task partitioning. However, PSO has the problems of local extremum and low precision, leading to the poor partitioning quality and unsatisfied performance. In this paper, Reverse Learning Dynamic Radius Particle Swarm Optimization (RLDRPSO) task partitioning algorithm was proposed to solve these problems. Firstly, the fitness function was designed according to the system model. Then, DRPSO was proposed to extend the solution space and improve the accuracy. Finally, reverse learning strategy was proposed to degenerate solution periodically and solve the problem of local extremum. Experimental results show that RLDRPSO increases the partitioning quality by 20 %–45 % and the average performance of system by 7 %–9 %.

Keywords: Task partitioning · Radius particle swarm optimization · Reverse learning

1 Introduction

Task partitioning is the critical step in the co-design of reconfigurable embedded system, which has a dominant effect on the system performance. It determines which components of the system are implemented on hardware and which ones on software [1]. Traditional methods of task partitioning are made by hand. As the embedded system becomes more and more complex, it is difficult for task partitioning in embedded system. Then many researchers put attention on task partitioning algorithms, turning to use automatic methods to solve this problem [2].

Task partitioning is a problem of multi-objective optimization. Using Mixed Integer Linear Programming(MILP) [3] and Dynamic Programming(DP) [4] could find global optimal solution in theory to achieve the best system performance. However, these methods search for the whole solution space blindly. Particularly in the case of large-scale amount of tasks, these methods run for a long time.

Then, many researchers used heuristic methods [5], like Genetic Algorithm(GA) [6], Simulated Annealing(SA) [7], Ant Colony Optimization (ASO) [8]

J. Wu and L. Li (Eds.): ACA 2016, CCIS 626, pp. 118–129, 2016.
DOI: 10.1007/978-981-10-2209-8_11

and Particle Swarm Optimization(PSO) [9], as the partitioning strategy. PSO in these heuristic methods has simple parameters, executes rapidly and costs less [9]. So, we choose PSO to solve the problem of task partitioning in this paper.

While, the disadvantages of PSO are the unsatisfied approximate solution and local extremum. To solve inherent defects of PSO, early literatures proposed the optimized parameters. In the initial stage of PSO, the convergence is accelerated by the individual learning factor. In the latter part, the search is performed by the global learning factor. The improvement of this strategy is not obvious. Yang [10] proposed CLPSO task partitioning algorithm, which changed the structure of PSO and led the population to converge on the global optimal solution by a alterable leader. CLPSO is unreliable for unstable results. Luo [11] proposed ICPSO task partitioning algorithm, which combines the PSO and immune clone algorithm to extend the solution space through cloning and mutation. Hu [12] combined PSO and simulated annealing algorithm, which delays the convergence to avoid the local extremum. These methods have the problems of premature and poor accuracy and result in unsatisfied task partitioning quality.

Above all, we propose RLDRPSO task partitioning algorithm to improve the task partitioning quality system performance. RLDRPSO includes the optimized parameters, structure and combined algorithm. DRPSO algorithm is used to search for optimum solution dynamically, which expands the solution space and improves the accuracy. When judging the results falling into the local extremum, the algorithm uses reverse learning mechanism to degenerate solution in a certain extent, so as to overcome the problem of local extremum and search for better solution on this basis. Experimental results show that the proposed algorithm can effectively improve the partitioning quality and performance of the system.

This paper is organized as follow. Section 2 introduces the task partitioning model. In Sect. 3, we propose RLDRPSO task partitioning algorithm. Then the proposed algorithm is compared with typical algorithms in Sect. 4. The fifth section points out shortcomings of the proposed algorithm and the direction for further study.

2 Problem Definition

2.1 System Structure

The general structure of configurable embedded system model is presented in Fig. 1 [13].

CPU is the processor for software tasks, which contains the local memory(LM). FPGA or ASIC is the processor for hardware tasks. The communication among tasks is realized by bus and shared memory. We suppose that execution time, energy consumption, area of hardware resources and the communication time and energy consumption in CPU or FPGA are known. In this model, tasks oriented two-way division.

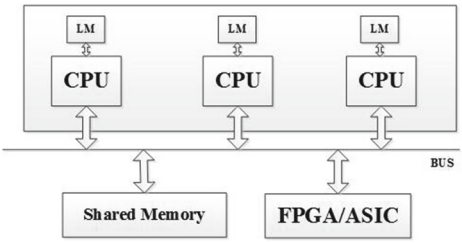

Fig. 1. Embedded system model [13]

2.2 Generalized Task Partitioning

According to the task granularity, an application needs to be decomposed into a number of tasks, which is represented by Data Flow Graph(DFG):

$$G = \{V, E\} \tag{1}$$

DFG G is a directed acyclic graph that contains a set of nodes V and edges E. Each node $v_i \in V$ represents a task that needs to be performed in processor cores. Each side $e_{ij} \in E$ indicates that the data is transferred from v_i to the task v_j via the bus. The number of tasks is n. $\forall v_i, i \in [1, n]$ contains multiple attributes:

$$v_i = \{x_i, TS_i, TH_i, ES_i, EH_i, A_i\} \tag{2}$$

x_i is the candidate partitioning location of v_i. For the task is divided into two directions, $x_i \in \{0, 1\}$. $x_i = 1$ means v_i performed in the CPU and $x_i = 0$ means v_i performed in the FPGA. TS_i, TH_i are the execution time of v_i in CPU or FPGA respectively. ES_i, EH_i are the energy consumption of v_i in CPU or FPGA respectively. A_i indicates the hardware area resources needed to perform v_i in FPGA. The hardware area resources usually mean the cost of the system design.

Generally speaking, the goal of task partitioning is to minimize the execution time and energy consumption of the system under system constraints, so as to achieve the goal of improving the system performance. For the problem of two-way partitioning in heterogeneous embedded systems, the target is to solve the optimal solution X:

$$X = \{x_1, x_2, ..., x_n\} \tag{3}$$

In G, V is divided into $\{V_s, V_h\}$. V_s represents a collection of tasks performed in CPU, and V_h represents a collection of tasks performed in FPGA or ASIC, $V_s \cup V_h = V$ and $V_s \cap V_h = \emptyset$. In the co-design of embedded system, the hardware area resource is limited. For the generalized task partitioning, the principle is to make full use of limited area resource and at the same time minimize the execution time and energy consumption. Therefore, in this paper we will not focus on tasks sensitive to time and energy, but discuss the limited area resource under this generalized principles. At the same time, the communication time and energy

consumption between each task are ignored in the bus structure. The objective function to describe the task partitioning problem is in formula (4):

$$\begin{cases} \min\left[\sum_{i=1}^{n} TH_i \cdot x_i + \sum_{i=1}^{n} TS_i \cdot (1 - x_i)\right] \\ \min\left[\sum_{i=1}^{n} EH_i \cdot x_i + \sum_{i=1}^{n} ES_i \cdot (1 - x_i)\right] \\ \sum_{i=1}^{n} A_i \cdot x_i < A_{\max} \end{cases} \tag{4}$$

$T(x)$,$E(x)$,$A(x)$ respectively represent the time, energy consumption and area overhead of a particular application. $A_{\max}$ is the largest area on the FPGA for performing tasks.

2.3 PSO Task Partitioning Algorithm

In the classical PSO task partitioning algorithm, each particle represents a candidate partition solution. PSO randomly generated m candidate partition solutions, $X_1, X_2, ..., X_m$. Number of tasks is n. $X_i = \{x_{i1}, x_{i2}, ..., x_{id}, ..., x_{in}\}$ is n-dimensional vector in Euclidean space R^n and x_{id} is the dth element. PSO updates partitioning solutions by iteration. The number of iterations is $K \in N^*$. The current iteration number is $k \in [1, K]$. Each x_{id} corresponds to an updating rate of v_{id}. Updating speed vector is $V_i = \{v_{i1}, v_{i2}, ..., v_{id}, ..., v_{in}\}$, meaning the updating speed of candidate partition solution, which should be controlled within a certain range $|v_{id}| < v_{\max d}$. Velocity threshold vector is $V_{\max} = \{v_{\max 1}, ..., v_{\max d}, ..., v_{\max n}\}$. PSO uses the fitness function $f(X)$ to evaluate the candidate partitioning solution, so as to find the optimal partitioning solution. $f(X)$ is determined by the target of the system, as shown in formula (4). It is used to measure the degree of the candidate solution to adapt to the system constraints. $X_1^*, X_2^*, ..., X_n^*$ are the optimal partition solutions in the updating processes of X_i. $X_g^* = \{X_{g1}^*, X_{g2}^*, ..., X_{gn}^*\}$ is the optimal partition solution in $X_1^*, X_2^*, ..., X_n^*$. In each iteration, the candidate partition solution is guided by X_i^* and X_g^*. In the kth generation, x_{id} and v_{id} correspond to x_{id}^k and v_{id}^k. The updating formula is as follow:

$$\begin{cases} v_{id}^{k+1} = \omega \cdot v_{id}^k + c_1 r_1 (X_{id}^* - x_{id}^k) + c_2 r_2 (X_{gd}^* - x_{id}^k) \\ x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1} \end{cases} \tag{5}$$

The updating formula of PSO task partitioning algorithm is composed of three parts. The first part is the inertia part, ω is the inertia factor, which is used to measure the effect of the current updating rate. The second part is the part of self-cognition, which reflects the memory of the particle's own historical experience. The third part is the social cognitive part, which reflects the historical experience of the cooperation and knowledge sharing among the particles. c_1 and c_2 are learning factors, which respectively indicate the effect of the individual optimal solution and the global optimal solution on the direction of population updating. r_1 and r_2 are random number between [0,1].

3 RLDRPSO Task Partitioning Algorithm

In RLDRPSO task partitioning algorithm, we first design learning factors and the fitness function. With the idea of "divide and conquer", DRPSO selects regional optimal value and global optimal solution from the regional optimal solution. In the latter part of the algorithm, if the candidate partitioning solution falls into local optimal solution, this algorithm triggers reverse learning mechanism to degenerate partitioning solution, jumping out of local extremum. On the basis, the optimal solution is solved.

3.1 Learning Factors and Fitness Function

ω, c_1 and c_2 in PSO are in linear form, expressed as follows:

$$\omega = \omega_{\min} + \frac{(\omega_{\max} - \omega_{\min})(K - k)}{K} \tag{6}$$

$$c_1 = (c_{1\max} - c_{1\min}) \cdot \frac{k}{K} + c_{1\min} \tag{7}$$

$$c_2 = (c_{2\max} - c_{2\min}) \cdot \frac{K - k}{K} + c_{2\min} \tag{8}$$

In the initial stage of the algorithm, ω and c_1 are larger and c_2 is smaller. V_i and X^* have a great influence on solution updating, which not only ensures the solution space but also increases the speed of updating. At the later stage of the algorithm, c_2 increases, which means the impact of X_g^* on the updating process is larger. ω and c_1 decrease, which means the impact of X^* on the updating process is small. As a result, the updating speed is reduced and the accuracy of the solution increases.

According to formula (4), we evaluate candidate partitioning solutions from three aspects, the hardware area, task execution time and energy consumption. After that, we normalize the processing and design fitness function. Generally, if the hardware area of partitioning solution needed is larger than the area system provided, the task will not be performed. So, the fitness function is designed in the formula (9):

$$f(x) = a \cdot e^{\frac{A(x) - A_{\max}}{\delta_A} \cdot m_a} \cdot \left| \frac{A(x) - A_{\max}}{\delta_A} \right| + b \cdot \frac{T(x)}{\delta_T} + c \cdot \frac{E(x)}{\delta_E} \tag{9}$$

$A(x)$, $T(x)$ and $E(x)$, respectively, are the hardware area, execution time and energy consumption in one partitioning solution. δ_A, δ_T and δ_T, respectively, are the normalization factor of area, execution time and energy consumption. $\delta_A = \max\{\max A - A_{\max}, A_{\max} - \min A\}$, $\delta_T = \max T - \min T$, $\delta_E = \max E - \min E$. a, b and c are influence factors. Punishment factor is shown as follow:

$$m_a = \begin{cases} 1, A(x) \le A_{\max} \\ k, A(x) > A_{\max} \end{cases} \tag{10}$$

In this penalty mechanism, if the candidate partitioning solution is not in excess of the hardware area system provided, the fitness value decreases in the form of the exponential function. Otherwise, the fitness value increases rapidly according to current iteration number. When the iteration is small, the punishment is light, offering an opportunity for the candidate partitioning solution correcting the error. When the number of iterations is larger, the punishment is large. Penalty mechanism no longer tolerates the wrong candidate partitioning solution. The fitness function is in good agreement with the actual situation.

3.2 Reverse Learning

In order to solve the problem that PSO algorithm is easy to fall into the local extremum, which makes the system cannot reach the prospective performance, we design a reverse learning mechanism (RL) for the task partitioning algorithm. The idea of reverse learning is to jump out of the local extremum when algorithm predicates solution falling into local extremum. The reverse learning mechanism first initializes x_i and v_i of each partitioning solution, the initial location of the worst solutions W_i^0 and the worst location of each individual solution W_i^k. After Lth reverse learning, the result jumps out of local extremum. In the reverse learning mechanism, l indicates the number of the current reverse learning iteration. RL's updating formula is as shown in (11):

$$\begin{cases} v_{id}^{l+1} = \omega \cdot v_{id}^{l} + c_3 r_3 (x_{id}^{l} - W_{id}^{l}) + c_4 r_4 (x_{id}^{l} - W_{id}^{0}) \\ x_{id}^{l+1} = x_{id}^{l} + v_{id}^{l+1} \end{cases} \tag{11}$$

The distance between solutions is in the Euclidean distance. The distance between W_i^0 must be large enough in order to ensure that the W_i^0 can pull the result out the local extreme value. When choosing W_i^0, their distance is greater than the preset distance $R = \sqrt{n}$.

In the reverse learning mechanism, the updating speed threshold changes to $RV_{\max} = 2 \cdot V_{\max}$, making the candidate partitioning solution jumping out of local extremum rapidly under the leading of W_i^0 and W_i^k (Table 1).

Table 1. RL mechanism

Algorithm 1. RL mechanism
INPUT: X_i^k, W_i^k, W_i^0 and L
OUTPUT: X_i^k
1. initialize $l = 0$, $X_i^l = X_i^k$, $V_i^l = V_i^k$
2. **for** $l<L$
3. randomly select initial worst partition scheme from W_i^0
4. update X_i^{l+1} and V_i^{l+1}
5. calculate fitness value $f(X_i^{l+1})$
6. select max $f(X_i^{l+1})$, updating $W_i^{l+1} = X_i^{l+1}$
7. **endfor**
8. **return** $X_i^k = X_i^L$

3.3 RLDRPSO

In this paper, we improve RPSO [14] algorithm to solve the problem of task partitioning. Generally speaking, at the initial stage of the algorithm, X_i is far from the optimal solution. Using a larger radius to search X_i is conducive for fast convergence. In the late stage of the algorithm, X_i is near from the optimal solution. Reducing radius can improve the search accuracy. The basic idea of DRPSO is to dynamically adjust the region size of each candidate partition solution according to the number of iteration. The global optimal solution is selected from these region optimal solutions. Set radius r. r varies linearly with the number of iteration, as shown in formula (12):

$$r = \frac{K-k}{K} \cdot n \tag{12}$$

The distance between solutions is in the Euclidean distance. Firstly, Choose the optimal individual solution location as the center of the circle, whose radius is r. Then randomly select neighbor solutions in this circle. Find the region optimal solution $X_i^R = \{X_{i1}^R, ..., X_{id}^R, ..., X_{id}^R\}$. DRPSO's updating formula is shown as follow:

$$\begin{cases} v_{id}^{k+1} = \omega \cdot v_{id}^k + c_1 r_1 (X_{id}^R - x_{id}^k) + c_2 r_2 (X_{gd}^* - x_{id}^k) \\ x_{id}^{k+1} = x_{id}^k + v_{id}^{k+1} \end{cases} \tag{13}$$

Combined with the reverse learning mechanism mentioned in the last subsection, when the solution falls into local extremum, RL helps the result to jump out of local extremum (Table 2).

Table 2. RLDRPSO task partitioning algorithm

Algorithm 2. RLDRPSO task partitioning
INPUT: K
OUTPUT: X_g^*
1. Initialize X_i^0,W_i^0
2. **for** k<K
3. calculate current X_i^k ś fitness value $f(X_i^k)$, select X_i^*
4. initialize dynamic radius r
5. randomly select X_i^Rensuring $\|X_i^* - X_i^R\| < r$
6. updating X_i^{k+1} and V_i^{k+1}
7. updating X_i^*, W_i^{k+1}, X_g^*
8. **if** X_g^* falling into local extremum
9. trigger Algorithm 1
10. **endif**
11. **endfor**
12. **return** X_g^*

4 Experimental Results

4.1 DFG Data and RLDRPSO Parameters

Researchers usually use TGFF [15] to generate DFG and attributes as the test-bench for task partitioning algorithm. According to the research [10], we use TGFF to generate the DFG and the hardware area, execution time and power consumption of each task. The range of the specified task attributes is shown in Table 3.

Table 3. Task attributes' value [8,10]

	Task attribute	Range
FGPA	area/unit	[50,150]
	power consumption/W	[0.15,0.55]
	time/ns	[75,225]
CPU	area/unit	0
	power consumption/W	[0.25,0.65]
	time/ns	[200,400]

Task partitioning algorithms run in Visual Studio 2010, Xeon Intel 2680 CPU 2.8 GHz, RAM 4 GB, Window7. Besides RLDRPSO task partitioning algorithm, we also select the standard PSO task partitioning algorithm [9], ICPSO task partitioning algorithm [8] and CLPSO task partitioning algorithm [10] for comparison. In order to ensure the comparability, the experiment uses the same PSO parameters. The parameters are shown in Table 4.

Table 4. RLDRPSO constant parameters

DRPSO		RL	
K	100	R	10
$\omega_{\max}$	0.9		
$\omega_{\min}$	0.4	ω	0.7
$c_{1\,\max}$	2	c_3	0.7
$c_{1\,\min}$	1.85		
$c_{2\,\max}$	2	c_4	0.3
$c_{2\,\min}$	1.85		
$v_{\max}$	0.8	$rv_{\max}$	1.6

4.2 Results

To evaluate the performance of the system, the algorithm is designed to punish the task sensitive to area. So we first study the performance of the algorithm

under different area. HAR refers to the ratio of the hardware area restriction to the total hardware area. MR is the probability that the task partitioning solution cannot satisfy the system constraints. When the task partitioning solution cannot meet the requirements of the system, it needs to be resolved, which increases the execution time of the algorithm. The smaller MR is, the higher the probability of the algorithm finding the optimal solution is, that is, the space of the solution is larger. The smaller the fluctuation is, the higher the reliability of the algorithm is. Figure 2 shows that algorithms have a certain MR, especially the CLPSO task partitioning algorithm is not reliable. When the HAR is more than 80 %, the MR of RLDRPSO is 0, and the fluctuation is small. It reflects that the RLDRPSO can effectively expand the space of the candidate partitioning solutions.

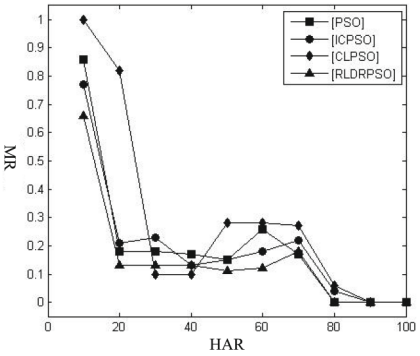

Fig. 2. MR in different HARs

In the next experiment we discuss the influence of HAR on algorithm performance. In order to compare the performance of each algorithm, the fitness value of the optimal solution is normalized to the fitness value of the general processor. From Fig. 3, in the conditions of presence of a certain MR, the performance improvement of all algorithms is almost same. When the HAR is more than 70 %, the performance of the RLDRPSO algorithm is still growing compared to other algorithms. Therefore, RLDRPSO task partitioning algorithm has a significant effect on the improvement of the system performance under large HAR.

Figure 4 is the relation of the fitness value with the iteration number, reflecting the partitioning process of algorithms. It can be seen from the figure that PSO, ICPSO and CLPSO algorithm have converged in the initial stage of the algorithm, falling into the local extremum. While RLDRPSO algorithm with reverse learning mechanism can jump out of the local extremum, converging toward the global optimal solution. In the figure, the steep part in RLDRPSO algorithm curve indicates the reverse learning process. Each time the reverse learning mechanism is triggered, the partitioning process has a significant improvement effect. In the case that the number of tasks is 500, RLDRPSO

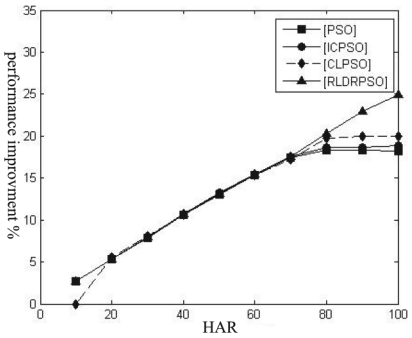

Fig. 3. Performance in different HARs

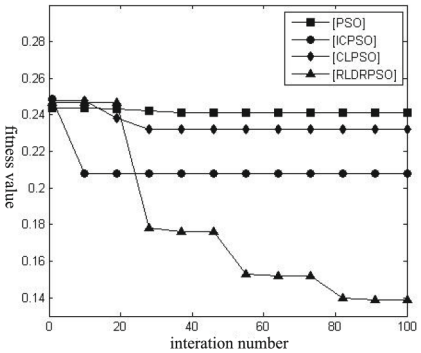

Fig. 4. Iterative process of different algorithms

improves the partitioning quality of 45 % compared to the PSO task partitioning algorithm and 39 % compared to the ICPSO task partitioning algorithm and 20 % compared to the CLPSO task partitioning algorithm. The reverse learning mechanism can effectively solve the local extremum and RLDRPSO algorithm enhance the accuracy of the solution. Finally, RLDRPSO task partitioning algorithm can improve the partitioning quality. As a result, the system performance is improved.

Figure 5 shows the effect of proposed algorithm in different number of tasks. RLDRPSO task partitioning algorithm improved the performance of the system significantly. Compared to PSO task partitioning algorithm, the average performance of the system is improved by 7 %. Compared to the ICPSO task partitioning algorithm, the average performance of the system is improved by 6.3 %. Compared to the CLPSO task partitioning algorithm, the average performance of the system is improved by 4.9 %.

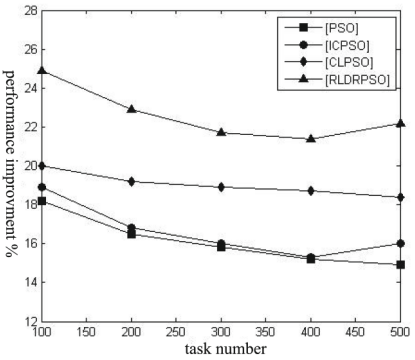

Fig. 5. Performance in different task numbers

5 Conclusions

In this paper, an integrated heuristic algorithm RLDRPSO is proposed, which can improve the task partitioning quality and the performance of system. Dynamic radius strategy and the reverse learning mechanism are proposed to solve the problem of the current task partitioning algorithm. Experimental results show that the algorithm can effectively improve the performance of the system.

The algorithm still has some defects, such as the existence of MR and the blindness of the reverse learning mechanism. Blindness means that the global optimal value which is resolved by reverse learning has the probability of being worse than the former global optimal value. Next research will conduct the discrimination and retention mechanisms to overcome these defects.

Finally, tasks can be executed simultaneously in different processing cores in heterogeneous system. Emerging reconfigurable system changes the existing system model. There is another problem of task scheduling. The combination of partitioning and scheduling will be the trend of future research.

References

1. Pu, W.: Efficient heuristic and tabu search for hardware/software partitioning. J. Supercomput. **66**(1), 118–134 (2013)
2. Mann, Z., Orb, A.: Optimization problems in system-level synthesis. In: Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications (2003)
3. Ma, Y., Liu, J., Zhang, C.: HW/SW partitioning for region-based dynamic partial reconfigurable FPGAs. In: 2014 32nd IEEE International Conference on Computer Design (ICCD), pp. 470–476. IEEE (2014)
4. Jemai, M., Ouni, B.: Hardware software partitioning of control data flow graph on system on programmable chip. Microprocess. Microsyst. **39**(4), 259–270 (2015)
5. Ernst, R., Henkel, J., Benner, T.: Hardware-software cosynthesis for microcontrollers. IEEE Des. Test Comput. **4**, 64–75 (1993)

6. Liang, H., Sinha, S., Warrier, R., et al.: Static hardware task placement on multi-context FPGA using hybrid genetic algorithm. In: International Conference on Field Programmable Logic and Applications. IEEE (2015)
7. Liang, Z., Cheng, X., Zheng, T., et al.: Hardware/software partitioning based on greedy algorithm and simulated annealing algorithm. J. Comput. Appl. **7**, 030 (2013)
8. Wang, D., Li, S., Dou, Y.: Collaborative hardware/software partition of coarse-grained reconfigurable system using evolutionary ant colony optimization. In: Asia and South Pacific Design Automation Conference, pp. 679–684. IEEE Computer Society Press (2008)
9. Bhattacharya, A., Konar, A., Das, S., et al.: Hardware software partitioning problem in embedded system design using particle swarm optimization algorithm. In: International Conference on Complex, Intelligent and Software Intensive Systems, pp. 171–176. IEEE Computer Society (2008)
10. Yang, D.: Research on SoC Hardware/Software Partitioning Methodology Based on Particle Swarm Algorithm. Xidian University (2014)
11. Luo, L., He, H., Liao, C., et al.: Hardware/Software partitioning for heterogeneous multicore SOC using particle swarm optimization and immune clone (PSO-IC) algorithm. In: 2010 IEEE International Conference on Information and Automation (ICIA), pp. 490–494. IEEE (2010)
12. Hu, D.: The Research of Hardware/Software Partitioning Based on Partical Swarm Optimazation and Simulated Annesling and Clustering Algorithm. East China Normal University (2014)
13. Sha, E., Wang, L., Zhuge, Q., et al.: Power efficiency for hardware/software partitioning with time and area constraints on MPSoC. Int. J. Parallel Program. **43**(3), 1–22 (2013)
14. Anantathanvit, M., Munlin, M.A.: Radius particle swarm optimization for resource constrained project scheduling problem. In: 2013 16th International Conference on Computer and Information Technology (ICCIT), pp. 24–29. IEEE (2014)
15. Dick, R., Rhodes, D., Wolf, W.: TGFF: task graphs for free. In: International Workshop on Hardware/software Codesign, pp. 97–101. IEEE (1998)