Cluster Computing (2019) 22:953–964
https://doi.org/10.1007/s10586-018-2884-6

Multi-objective constraint task scheduling algorithm for multi-core processors

Ying Xie[1,2,3,4] · Jinzhao Wu[5]

Received: 12 March 2018 / Revised: 27 September 2018 / Accepted: 3 December 2018 / Published online: 11 December 2018

Abstract
A task scheduling algorithm is an effective means to ensure multi-core processor system efficiency. This paper defines the task scheduling problem for multi-core processors and proposes a multi-objective constraint task scheduling algorithm based on artificial immune theory (MOCTS-AI). The MOCTS-AI uses vaccine extraction and vaccination to add prior knowledge to the problem and performs vaccine selection and population updating based on the Pareto optimum, thereby accelerating the convergence of the algorithm. In the MOCTS-AI, the crossover and mutation operators and the corresponding use probability for the task scheduling problem are designed to guarantee both the global and local search ability of the algorithm. Additionally, the antibody concentration in the the MOCTS-AI is designed based on the bivariate entropy. By designing the selection probability in consideration of the concentration probability and fitness probability, antibodies with high fitness and low concentration are selected, thereby optimizing the population and ensuring its diversity. A simulation experiment was performed to analyze the convergence of the algorithm and the solution diversity. Compared with other algorithms, the MOCTS-AI effectively optimizes the scheduling length, system energy consumption and system utilization.

Keywords Multi-core processor · Multi-objective constraint · Artificial immune · Task scheduling

1 Introduction

Multi-core processors, which have replaced single-core processors, are widely used in various fields, such as image processing, data warehousing, and digital signal processing [1]. Optimization of a task scheduling algorithm is an effective means to ensure multi-core processor system efficiency and to improve the performance of a multi-core processor system [2, 3].

Intelligent algorithms perform probabilistic searches for global optimization by simulating natural phenomena or processes and offer global, parallel and efficient optimization performance. Such algorithms are a common approach for the multi-objective optimization of task scheduling in multi-core processor environments. Genetic algorithms, simulated annealing algorithms, artificial immune algorithms, particle swarm optimization algorithms, ant colony algorithms and other intelligent search algorithms are emerging to address multi-objective optimization problems. Among them, artificial immune algorithms function based on the principles of natural defense,

✉ Ying Xie
xieying33@163.com

Jinzhao Wu
himrwujzhao@aliyun.com

1 The Key Laboratory for Computer Systems of the State Ethnic Affairs Commission, Southwest Minzu University, Chengdu 610041, China

2 School of Computer Science and Technology, Southwest Minzu University, Chengdu 610041, China

3 Chengdu Institute of Computer Application, Chinese Academy of Sciences, Chengdu 610041, China

4 University of the Chinese Academy of Sciences, Beijing 100049, China

5 Guangxi Key Laboratory of Hybrid Computational and IC Design Analysis, Guangxi University for Nationalities, Nanning 53006, Guangxi, China

namely, the generation of antibodies by vaccine extraction and vaccination to eliminate antigens, and they possess various beneficial capabilities of evolutionary learning methods, such as anti-noise capabilities, unsupervised learning, memory and self-organization [4, 5].

Artificial immune algorithms have the characteristics of multi-peak search and global optimization for multi-peak functions. In the large-scale, complex applications typical of multi-core processor systems, a task scheduling algorithm is a multi-objective optimization problem in which one must consider the scheduling length of the set of tasks, load balancing, system energy consumption, scheduling costs and other factors. This paper uses the principles of artificial immunity to solve the task scheduling problem; based on artificial immune theory, an antigen is defined to represent the optimization objectives and constraints of the multi-core processor task scheduling problem, an antibody is defined to represent the task allocation matrix, and each step of the artificial immune algorithm is designed and implemented to solve the task scheduling problem.

The first artificial immune algorithm, which inspired by the biological immune system, was proposed by De Castro et al. [6]. The NSGA [7], the NPGA [8], and the MOGA [9] constitute the first generation of artificial immune algorithms; they are characterized by an individual selection method based on the Pareto rank and a population diversity maintenance strategy based on a fitness sharing mechanism.

Algorithms characterized by elite retention mechanisms have also been proposed. Based on the level of non-inferior individuals to guide the search to the Pareto optimal solution, NSGA-II [10] is an improved version of the NSGA algorithm, and it serves as the performance benchmark for other algorithms applied to multi-objective optimization problems. NNIA [11, 12] selects non-dominated individuals based on a non-dominated neighborhood selection technique and clones these non-dominated individuals according to the individual crowding distance. Due to its low computational complexity, NNIA is considered representative of multi-objective optimization immune algorithms. NNCA [13] divides non-dominated individuals into elite and common populations based on crowded distance and avoids premature convergence by leading the search for elite populations and elite individuals from the general population of elite population immigrants. However, when the maximal non-dominated sequence of antibodies in the initial population is small, the NSGA-II algorithm may appear to converge early or converge locally, and the degrees to which the NNIA algorithm and the NNCA algorithm approach the optimal solution when the sampling space is small can be greatly reduced.

To solve high-dimensional multi-objective optimization problems more effectively, artificial immune algorithms have been combined with ε dominance, related entropy, principal component analysis, population evolution, cultural genes and other mechanisms. Based on the theory of artificial immunity and population evolution, [14] proposes a multi-objective optimization task scheduling algorithm that considers the task execution time, the communication time between tasks and the scheduling overhead. However, the crossover and mutation operators designed by this algorithm cannot guarantee the diversity of the population in the early evolution. Moreover, the problem of low local search ability occurs in the late evolutionary stage. By combining the cultural gene algorithm and immune algorithm, MMO [15] achieves multi-objective optimization task scheduling in heterogeneous environments considering the computational performance and system energy consumption. The addition of cultural genes increases the local optimization ability of the algorithm, but the analysis of knowledge must be solved after many iterations to obtain the solution of the problem. AISD [16] integrates the clonal selection principle and task replication technology in the immune system into the task scheduling process in heterogeneous environments to reduce the time complexity of the algorithm, but the AISD algorithm is more suitable for communication-intensive applications and requires improvement for computationally intensive applications.

In terms of a multi-core processor environment, this paper takes the task-dependent relationship as the constraint condition and defines the multi-objective constraint problem of task scheduling algorithm by minimizing the task scheduling length, minimizing the system energy consumption and maximizing the system utilization. Finally, a multi-objective constraint task scheduling algorithm based on artificial immune theory (MOCTS-AI) is proposed. According to the basic characteristic information that the solution of the objective functions should satisfy, the MOCTS-AI algorithm extracts the vaccine and applies vaccination to add prior knowledge to the problem, which promotes the evolution of the population. MOCTS-AI defines the antibody concentration by the bivariate entropy. Then, MOCTS-AI uses the concentration probability and fitness probability to promote and inhibit the population. The promotion and inhibition of antibodies optimize the current population while ensuring diversity. MOCTS-AI designs the crossover operator, mutation operator and the use probability of the corresponding operator for the task scheduling problem. Crossover and mutation guarantee the global search ability and local search ability of the algorithm. MOCTS-AI performs vaccine selection and population update based on the Pareto optimality and further accelerates the convergence of the algorithm by preserving non-dominant antibodies. The simulation experiment achieves the optimal population size by analyzing the convergence of the algorithm and the diversity of the

solution. Compared with other algorithms, the proposed algorithm effectively optimizes the scheduling length, system energy consumption and system utilization.

In Sect. 2, we construct the task model and system model. Then, we define the multi-objective constrained task scheduling problem of the multi-core processor that we use throughout the paper. On the basis of the process of the artificial immune algorithm, Sect. 3 introduces the details of the MOCTS-AI algorithm. The simulation of Sect. 4 compares the advantages and disadvantages of the MOCTS-AI algorithm and other algorithms. Finally, Sect. 5 concludes the paper.

2 The definition of the task scheduling problem

2.1 Task model

A directed acyclic graph (DAG) is used to describe the set of dependent tasks $DAG = (T, D)$, where $T = \{T_1, T_2, \cdots, T_n\}$ represents the set of task nodes. The weight of a node $Calc(T_i)$ is abstracted as the amount of computational data for the task T_i. $Pred(T_i)$ is composed of T_i's predecessor tasks. $D = \{d_{i,j} \mid T_i \rightarrow T_j, i \leq n, j \leq n\}$ is a set of constrained edges between nodes. $d_{i,j}$ indicating that task T_i is the direct predecessor of task T_j and that task T_j needs to wait for all its predecessor tasks to finish running before it can run. The weight of the edge $Comm(d_{i,j})$ is abstracted as the amount of communication from task T_i to task T_j. To simplify the model, it is assumed that there is no interference among co-running tasks.

2.2 System model

A multi-core processor system is represented by a two-tuple $ME = (C, I)$. Here, $C = \{core_1, core_2, \cdots, core_m\}$ represents a set of processor cores. $Rate(core_i)$ represents the rate of processor core $core_i$. $RunPower(core_i)$ and $IdlePower(core_i)$ represent the average runtime power and idle power, respectively, of processor core $core_i$. $I = (I_{i,j})_{m \times m}$ represents the interconnection matrix of processors, and $CommRate(I_{i,j}) > 0$ is the rate of link communication between $core_i$ and $core_j$. When two tasks are scheduled on the same processor core $core_i$, the $CommRate(I_{i,i})$ tends to be infinite; thus, we do not consider the amount of communication between two such tasks. $Power(I_{i,j})(i \neq j)$ represents the average link power between $core_i$ and $core_j$.

2.3 Scheduling goals

A scheduling algorithm can be described by the task allocation matrix $\Psi = (\psi_{i,j})_{n \times m}$. $\psi_{i,j} = 1$ means task T_i is dispatched to $core_j$; otherwise, $\psi_{i,j} = 0$. The goal of task scheduling based on a multi-core processor environment is to find a suitable task allocation matrix Ψ that can minimize the objective functions.

The scheduling length $SL(T)$ of task set T can be obtained based on the earliest start time $EST(T_i)$ and the earliest completion time $EFT(T_i)$.

$$EST(T_i) = \begin{cases} 0, & Pred(T_i) = \emptyset \\ \max\Big(\max\limits_{\substack{T_j \in Pred(T_i) \\ \psi_{j,q}=1, \psi_{i,q}=1}} \{EFT(T_j)\}, & \\ \quad \max\limits_{\substack{T_k \in Pred(T_i) \\ \psi_{i,q}=1, \psi_{k,p}=1 \\ p \neq q}} \{(EFT(T_k) + \frac{Comm(d_{k,i})}{CommRate(I_{p,q})})\}\Big), & else \end{cases} \tag{1}$$

(1)

$$EFT(T_i) = EST(T_i) + Calc(T_i)/Rate(core_p), (\psi_{i,p} = 1) \tag{2}$$

$$SL(T) = \max_{T_i \in T}\{EFT(T_i)\} \tag{3}$$

The TL of node T_i represents the length of the longest path from the root node to node T_i. No dependency exists between nodes that have the same TL. These tasks can be scheduled to different processor cores in parallel, and no additional communication overhead is generated.

$$TL(T_i) = \begin{cases} 1, & Pred(T_i) = \emptyset \\ 1 + \max\limits_{T_j \in Pred(T_i)}\{TL(T_j)\}, & else \end{cases} \tag{4}$$

The energy consumption $E(T)$ is generated when a multi-core processor performs the dependent tasks set T. $E(T)$ includes the energy consumption $CE(T)$ of the processor core

and the energy consumption $TE(T)$ of data transmission. The energy consumption $CE(T)$ is determined by the processor runtime and idle energy consumption.

Each processor core uses independent DVFS technology to dynamically adjust the voltage/frequency. The power consumption of processor core $core_i$ is approximately expressed as the sum of the runtime power consumption and the idle power consumption, which can be represented by $RunPower(core_i)$ and $IdlePower(core_i)$, respectively.

$$CE(T) = \sum_{q=1}^{m}\sum_{i=1}^{n}\psi_{i,q}\cdot\frac{Calc(T_i)}{Rate(core_q)}\cdot RunPower(core_q) + \sum_{q=1}^{m}\left(SL(T) - \sum_{i=1}^{n}\psi_{i,q}\cdot\frac{Calc(T_i)}{Rate(core_q)}\right)\cdot IdlePower(core_q) \tag{5}$$

where $IdlePower(core_q)$ is related to the factory state of the device, i.e., the power consumption in the steady state of the circuit, and can be set as a constant. $RunPower(core_q)$ is the power consumption caused by the charge and discharge of the circuit, which is determined by the circuit's load capacitance, supply voltage, and clock frequency.

$$TE(T) = \sum_{i=1}^{n}\sum_{\substack{T_k\in pred(T_i)\\ \psi_{i,q}=1,\psi_{k,p}=1\\ p\neq q}}\frac{Comm(d_{k,i})}{CommRate(I_{p,q})}\cdot Power(I_{p,q}) \tag{6}$$

$$E(T) = CE(T) + TE(T) \tag{7}$$

The system utilization U in a multi-core processor performing the dependent task set T is defined as the average of the ratio of the run time to task schedule length for each processor core.

$$U = \sum_{q=1}^{m}\left(\frac{\sum_{i=1}^{n}\psi_{i,q}\cdot\frac{Calc(T_i)}{Rate(core_q)}}{SL(T)}\right)/m \tag{8}$$

The goal of task scheduling based on a multi-core processor environment is to find a suitable task allocation matrix $\Psi = (\psi_{i,j})_{n\times m}$ that minimizes the scheduling length, minimizes the system energy consumption and maximizes the system utilization under the premise of meeting the dependencies between tasks. Thus, the multi-core processor multi-objective constrained task scheduling problem can be defined as:

$$\begin{cases} \min: y = F(x) = (f_1(x), f_2(x), f_3(x)) \\ s.t.\ \sum_{j=1}^{m}\psi_{i,j} = 1, (i = 1,2,\ldots,n) \\ \quad TL(T_i) \geq TL(T_j) \\ \quad EST(T_i) \geq EFT(T_j), T_j \in Pred(T_i) \\ x = (\psi_{i,j})_{n\times m} \in X \subset \{0,1\}^n \times \{0,1\}^m \\ y = (y_1, y_2, y_3) \in Y \subset R^3 \end{cases} \tag{9}$$

where $\sum_{j=1}^{m}\psi_{i,j} = 1, (i = 1,2,\cdots,n)$ ensures that each task is scheduled and scheduled only once. The predecessor-successor relationship between tasks is limited by the task's TL constraint $TL(T_i) \geq TL(T_j)$ and execution time constraint $EST(T_i) \geq EFT(T_j)$. The solution of the multi-objective optimization scheduling problem is expressed as a task assignment matrix $x = (\psi_{i,j})_{n\times m} \in X \subset \{0,1\}^n \times \{0,1\}^m$, with $n \times m$-dimensional decision space. The objective functions define three optimization goals: $f_1(x) = SL(x)$, $f_2(x) = E(x)$, and $f_3(x) = -U(x)$.

3 Algorithm description

Throughout the long-term process of evolution, the biological immune system has gradually developed into a distributed system with the capabilities of self-cognition and self-organization. In the immune recognition stage, T-lymphocytes and B-lymphocytes recognize antigen and antibody cells. In the immune response stage, antibodies activate and differentiate the recognized antigens. Through complementary components that mediate the inflammatory response to promote the phagocytosis of the antigens, the antibodies finally clear the antigens. An artificial immune system simulates biological immunity; based on the principles of natural defense, it generates antibodies by vaccine extraction and vaccination to eliminate antigens.

This section uses the principles of artificial immune algorithms to solve the task scheduling problem in a way that satisfies the optimization goal. In the following subsections, we map the task scheduling problem to each step of the artificial immune algorithm. The biological meanings of terms related to artificial immune algorithms are shown in Table 1, and the flow of the intelligent search algorithm based on artificial immunity theory is shown in Fig. 1.

An antigen is defined as the optimization objectives and constraints of a multi-core processor task scheduling problem, and an antibody is defined as the task allocation matrix in Sect. 3.1. The initialization of antibody population is generated in Sect. 3.2. Section 3.3 introduces the conditions that the vaccine needs to satisfy in the task

Table 1 Biological meanings of terms related to artificial immune algorithms

Term related to artificial immune algorithms	Corresponding biological meaning
Solution represented by a binary bit string	Chromosome
Bit of a solution	Gene
Position in a string	Locus
Structure of a string	Genotype
Meaning of a string	Phenotype
Iteration	Generation
Reorganization	Crossover
Reorganization	Mutation
Selection	Survival of the fittest

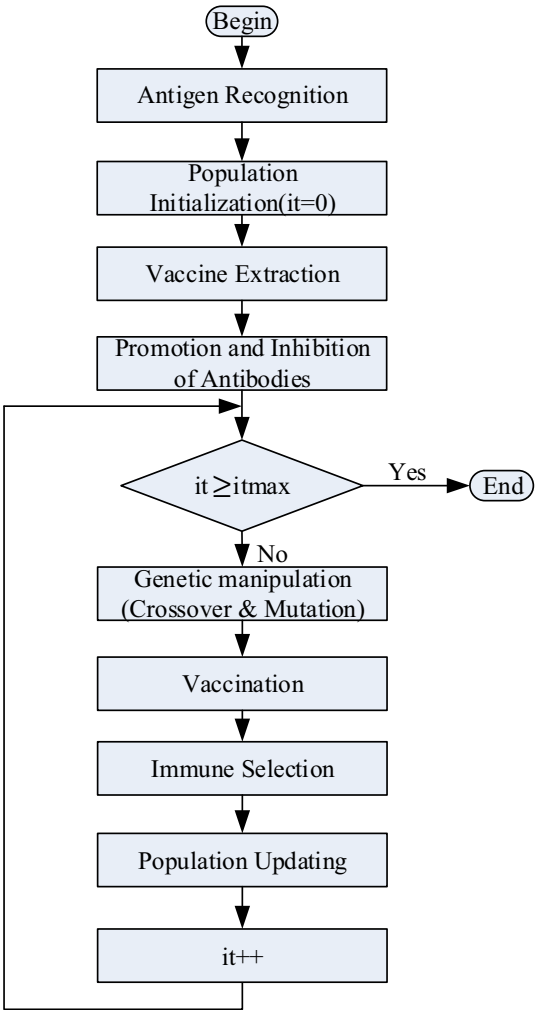

Fig. 1 The flow of the artificial immune algorithm

scheduling problem. In Sect. 3.4, the antibody concentration is defined by the bivariate entropy. Then, high-fitness antibodies are promoted, and high-concentration antibodies are inhibited. In Sects. 3.5–3.9, crossover, mutation, vaccination, immune selection, and population updating are performed iteratively n times to find solutions to the task scheduling problem that satisfy the optimization goals.

3.1 Antigen recognition and antibody definition

In the artificial immune algorithm, the antigen is an important metric. The antigen is defined as the objective functions and their constraints, that is, Formula 9. The antibody against the antigen is the solution of the objective functions, that is, the task allocation matrix $(\psi_{i,j})_{n\times m}$. To adapt to the artificial immune algorithm and to better handle the data in the solution space, the task allocation matrix is transformed into a one-dimensional array by row priority, and an antibody is represented by binary code. Antibody 'a' represents a candidate solution with $n \times m$ genes, and the character set used on each locus is $\Phi = \{0, 1\}$.

$$a = (s_1, s_2, \cdots, s_n)$$
$$s_i = (\psi_{i,1}, \psi_{i,2}, \cdots, \psi_{i,m})$$

The population is expressed as follows:

$$A = (a_1, a_2, \cdots, a_N)$$

The fitness function of antibody 'a' is as follows:

$$\begin{aligned} fit(a) &= \frac{1}{f_1(a) + f_2(a) + f_3(a)} \\ &= \frac{1}{SL(a) + E(a) - U(a)} \end{aligned} \tag{10}$$

When the body is first exposed to an antigen, the artificial immune system produces a new vaccine by learning the antigen and then forms an antibody to destroy the antigen by vaccination. In the multi-core processor task scheduling problem, different antigens are formed according to different scheduling objectives, and the corresponding solution is the antibody that is generated by the artificial immune process.

3.2 Population initialization

The initial antibody population $A(it)$ of size $N(it)$ is generated randomly.

$$A(it) = \{a_1(it), a_2(it), \cdots, a_{N(it)}(it)\}$$

where 'it' is the population iteration number, which is initialized to 0.

The matrix of target values obtained from the initial antibody population is expressed as follows:

$$F(A(it)) = \begin{pmatrix} f_1(a_1(it)) & f_1(a_2(it)) & \cdots & f_1(a_{N(it)}(it)) \\ f_2(a_1(it)) & f_2(a_2(it)) & \cdots & f_2(a_{N(it)}(it)) \\ f_3(a_1(it)) & f_3(a_2(it)) & \cdots & f_3(a_{N(it)}(it)) \end{pmatrix}_{3\times N(it)}$$

3.3 Vaccine extraction

The vaccine corresponds to the basic characteristic information of the solution that the objective functions satisfy. Antibodies generated based on this characteristic information are one solution to the problem. Correct extraction of the vaccine promotes the evolution of the population, which accelerates the convergence of the algorithm to the global optimal solution.

According to the definition of the multi-objective constrained task scheduling problem in a multi-core processor, the vaccine generated by extracting feature information satisfies the following conditions:

$$\begin{cases} \psi_{i,p} = 1 \wedge \sum_{k=1}^{m} \psi_{i,k} = 1 \\ \psi_{j,q} = 1 \wedge \sum_{k=1}^{m} \psi_{j,k} = 1 \end{cases} \tag{11}$$

$$i \neq j, TL(T_i) = TL(T_j), p \neq q$$

Vaccines indicate that tasks with the same TL are executed in parallel on different processor cores and that each task is assigned only once.

3.4 Promotion and inhibition of antibodies

By promoting individuals with high fitness and inhibiting individuals with high concentrations, a diversity of antibodies is guaranteed while optimizing the current populations. Promotion and inhibition of antibodies improve the defect that traditional genetic algorithms tend to converge prematurely [17].

Locus $\psi_{i,j}$ can only be 0 or 1. In the population of size N', the probability $p_{i,j} = \sum_{k=1}^{N'} \psi_{i,j}^k / N'$ represents the ratio of the number of antibodies for which locus $\psi_{i,j}$ is 1 to the population size N'. $1 - p_{i,j}$ is the ratio of the number of antibodies for which locus $\psi_{i,j}$ is 0 to the population size N'. $\psi_{i,j}^k$ represents the value of the kth gene's locus $\psi_{i,j}$. The bivariate entropy function $H_{i,j}(N')$ of locus $\psi_{i,j}$ of this population can be expressed as follows:

$$\begin{aligned} H_{i,j}(N') = & \\ & -\sum_{k=1}^{N'} \psi_{i,j}^k / N' \cdot \log_2 \left(\sum_{k=1}^{N'} \psi_{i,j}^k / N' \right) \\ & - \left(1 - \sum_{k=1}^{N'} \psi_{i,j}^k / N' \right) \cdot \log_2 \left(\sum_{k=1}^{N'} \psi_{i,j}^k / N' \right) \end{aligned} \tag{12}$$

The bivariate entropy $H_{a_k(it)}(N(it))$ of antibody $a_k(it)$ is as follows:

$$H_{a_k(it)}(N(it)) = -\sum_{i=1}^{n} \sum_{j=1}^{m} H_{i,j}(N(it)) \tag{13}$$

The smaller the bivariate entropy of the antibody is, the higher the similarity between the two antibody genes is. The concentration $D_{a_k(it)}$ of antibody $a_k(it)$ is defined as the ratio of the number of antibodies whose bivariate entropy difference from antibody $a_k(it)$ is less than the ratio of ϵ to the population size $N(it)$, where $\epsilon \in [0, 0.05)$ is the similarity constant.

The selection probability of antibody $a_k(it)$ is $p_{a_k(it)}$.

$$p_{a_k(it)} = \alpha \cdot p^f_{a_k(it)} + (1 - \alpha) \cdot p^d_{a_k(it)} \tag{14}$$

where α is the adjustment factor, $p^d_{a_k(it)} = fit(a_k(it)) / \sum_{i=1}^{N(it)} fit(a_i(it))$ is the fitness probability, and $p^d_{a_k(it)} = e^{-D_{a_k(it)}}$ is the concentration probability.

The higher the individual's fitness is, the closer it is to the optimal solution, so the probability of being selected should be be greater. The higher the individuals concentration is, the more similar the individuals are, which is not conducive to the diversification of the population. The population $A(it)$ is promoted and inhibited according to the selection probability $p_{a_k(it)}$, and the diversity of the population is guaranteed while improving the fitness of the individual.

3.5 Adaptive crossover

Crossover ensures the diversity of the population and expands the global search space. The two parental antibodies recombine to form new progeny antibodies with the crossover probability. The crossover probability p_{cross} determines the number of crossovers and hence the size of the offspring population.

$$p_{cross}(it) = \begin{cases} \left(\frac{it_{max} - it}{it_{max}} \right) \cdot p^{max}_{cross} + \frac{it}{it_{max}} \cdot p^{min}_{cross}, & fit' > fit_{avg} \\ p^{max}_{cross}, & fit' \leq fit_{avg} \end{cases} \tag{15}$$

where p_{cross}^{max} and p_{cross}^{min} are the upper and lower bounds of the crossover probability, 'it' is the current iteration number, it_{max} is the maximum number of iterations, fit' is the average fitness of the parents, and fit_{avg} is the average fitness of the current population $A(it)$. This formula ensures a larger crossover probability for vulnerable individuals with lower fitness and a larger crossover probability when the number of iterations is smaller, which ensures the diversity of populations in the early evolutionary stage and avoids premature convergence.

To ensure the characteristic information of the antibody, the crossover operation use a single-point crossover operation with row priority. The specific steps are given in Algorithm 1.

Algorithm 1:

(1) Randomly select two antibodies $a_i(it)$ and $a_j(it)$ from population $A(it)$ and generate a random number k_1 over the interval (0, 1).
(2) If $k_1 > p_{cross}(it)$, go to step 1; else, $a_i(it)$ and $a_j(it)$ perform crossover operation as parents.
(3) Randomly select locus k_2 and generate intersection point $k_3 = \lceil k_2/m \rceil \times m$.
(4) Exchange all loci of $a_i(it)$ and $a_j(it)$ after the intersection point k_3 to generate progeny antibodies $a_i'(it)$ and $a_j'(it)$.

The single-point crossover operation with row priority exchanges the task assignment scheme after the intersection point.

3.6 Adaptive mutation

Mutation is an ancillary operation in the evolution of antibodies used to improve the algorithm's local search capability. A mutation operation avoids the possibility of losing important genes during evolution. The two antibodies form a new offspring antibody by mutation with mutation probability p_{muta}.

$$p_{muta}(it) = \begin{cases} \left(\frac{it_{max} - it}{it_{max}}\right) \cdot p_{muta}^{min} + \frac{it}{it_{max}} \cdot p_{muta}^{max}, & fit' > fit_{avg} \\ p_{muta}^{min}, & fit' \leq fit_{avg} \end{cases} \tag{16}$$

where p_{muta}^{max} and p_{muta}^{min} are the upper and lower bounds of the probability of mutation.

As the number of iterations increases, the mutation probability increases gradually. In the early evolutionary stage, the mutation probability is low to ensure global searching in the initial stage of evolution. When the optimal solution is approached in the late stages of evolution, local searching is mainly used to improve the accuracy of the solution.

Single-point mutation is used to conduct mutation on population $A(it)$. The specific steps are shown in Algorithm 2.

Algorithm 2:

(1) Randomly select an antibody $a_i(it)$ from population $A(it)$ and generate a random number k_1 over interval (0, 1).
(2) If $k_1 > p_{muta}(it)$, go to step 1; else, antibody $a_i(it)$ undergoes the mutation operation.
(3) Randomly select locus k_2 and generate intersection point $k_3 = \lfloor k_2/m \rfloor$, and $k_4 = k_2 \% m$.
(4) If locus ψ_{k_3,k_4} is 0, then set locus $\psi_{k_3,k_4} = 1$ and clear loci $\{\psi_{k_3,j} | (j = 1, 2, \cdots, m) \wedge (j \neq k_4)\}$; else, clear locus $\psi_{k_3,k_4} = 0$, randomly select locus $\psi_{k_3,j'} (j' = 1, 2, \cdots, m \wedge j' \neq k_4)$, and let $\psi_{k_3,j'} = 1$. Progeny antibody $a_i'(it)$ is generated through this step.

Single-point mutation ensures that each task is assigned and assigned only once, consistent with the antibody characteristic information.

3.7 Vaccination

Vaccination randomly selects a certain proportion of antibodies in the population and modifies the gene at its locus according to the characteristic information of the vaccine so that the individual has higher fitness with greater probability. The vaccination process of population $A(it)$ is shown in Algorithm 3.

Algorithm 3:

(1) According to the proportion β, randomly choose N' antibodies to form a set $R_v = \{a_1(it), a_2(it), \cdots, a_{N'}(it)\}$.
(2) For antibody $a_k(it) \in R_v$, randomly generate an integer x $(1 \leq x \leq \max_{i=1}^{n}\{TL(T_i)\})$. For all tasks in which TL is equal to x, form a subset $T^x = \{T_{x_1}, T_{x_2}, \cdots, T_{x_{n_1}}\}$.
(3) Calculate $\min_{i=1}^{m}\{\sum_{j=1}^{n} \psi_{j,i} \cdot Calc(T_j)/Rate(core_i)\}$ to obtain the processor core $core_{y_1}$ that has the minimum load. Then, assign task subset T^x to core $core_{y_1}$: $\forall i = 1, 2, \cdots, n_1, \quad \psi_{x_i,z} = 0 (z = 1, 2, \cdots, m), \quad \psi_{x_i,y_1} = 1$.
(4) Calculate $\min_{i=1}^{m}\{(t_i \cdot P_i^{run} + (SL(T) - t_i) \cdot P_i^{idle})\}$ and $\max_{i=1}^{m}\{(t_i \cdot RunPower(core_i) + (SL(T) - t_i) \cdot IdlePower(core_i))\}$ to obtain the processor core $core_{y_2}$, which has the lowest energy consumption, and processor core $core_{y_3}$, which has the largest

energy consumption, where $t_i = \sum_{j=1}^{n} \psi_{j,i} \cdot Calc(T_j)/Rate(core_i)$. Additionally, randomly select tasks in subset T^x that are allocated to $core_{y_3}$ and redistribute them to $core_{y_2}$.

Based on satisfying the vaccine characteristic information, the locus of some antibodies is modified to improve the fitness according to the optimization objectives during vaccination.

3.8 Immune selection

Immune selection in multi-objective optimization problems selects the Pareto excellent antibodies set to enter the next iteration.

For any antibody $a_i(it), a_j(it) \in A(it)$, compared with $a_j(it)$, $a_i(it)$ is Pareto dominant if and only if:

$$(\forall k_1 = 1,2,3 \quad f_{k_1}(a_i(it)) \leq f_{k_1}(a_j(it))) \wedge (\exists k_2 = 1,2,3 \quad f_{k_2}(a_i(it)) \leq f_{k_2}(a_j(it)))$$

The notation $a_i(it) \succ a_j(it)$ indicates antibody $a_i(it)$ dominates antibody $a_j(it)$.

Antibody $a^*(it) \in A(it)$ is called a Pareto excellent antibody if and only if:

$$\neg\exists a_i(it) \in A(it) : a_i(it) \succ a^*(it)$$

Pareto excellent antibodies set $P^*(it)$ is as follows:

$$P^*(it) = \{a^*(it) | \neg\exists a_i(it) \in A(it) : a_i(it) \succ a^*(it)\}$$

3.9 Population updating

When the cardinality of the Pareto excellent antibody set is $|P^*(it)| \leq N(it)$, antibodies from the current Pareto excellent antibodies set $P^*(it)$ are randomly selected, cloned and added to the next iteration. The algorithm generates a Pareto excellent antibodies set in each iteration, which accelerates the evolution of the antibodies to the optimal antibodies.

4 Simulation experiment and results analysis

The simulation experiment uses TGFF [18, 19] to randomly generate a *DAG* by setting the task set attributes. The total number of task nodes is $n \in [25, 200]$, the maximum in-degree of the task nodes is $id \in [1, 10]$, and the maximum out-degree of the task nodes is $od \in [1, 10]$. MPSoCBench [20] is used as the simulation platform. MPSoCBench simulates the multi-core processor environment by randomly combining different types of processor cores and sets the processor core number to $m \in \{4, 8, 16, 32, 64\}$.

In the simulation experiment, *GD* measures the degree to which an antibody approaches the optimal antibody, and *MS* is used to evaluate the diversity of the population.

$$GD = \sqrt{\frac{1}{|P^*(it_{max})|} \cdot \sum_{i=1}^{|P^*(it_{max})|} dist_i^2} \tag{17}$$

where it_{max} is the maximum number of iterations, $|P^*(it_{max})|$ is the number of antibodies in the it_{max}th generation of the Pareto optimal antibodies set $P^*(it_{max})$, and $dist_i$ is the Euclidean distance between the antibody in $P^*(it_{max})$ and the nearest solution in the ideal Pareto front PF_{best}.

$$MS = \sqrt{\frac{1}{\rho} \cdot \sum_{i=1}^{\rho} \left(\frac{min(F(A(it_{max}))_i^{max}, F_i^{max}) - min(F(A(it_{max})))_i^{min}, F_i^{min})}{F(A(it_{max}))_i^{max} - F(A(it_{max}))_i^{min}} \right)^2} \tag{18}$$

where ρ is the number of objective functions, and $\rho = 3$. $F(A(it_{max}))_i^{max}$ and $F(A(it_{max}))_i^{min}$ represent the maximum and minimum of the ith objective function in the target value matrix of the it_{max}th generation. F_i^{max} and F_i^{max} are the maximum and minimum of the ith objective function in PF_{best}.

Crossover is used to generate new individuals. The greater the crossover probability is, the faster the new individuals will be produced. However, if the crossover probability is too large, the possibility that the genetic model will be destroyed will increase. If the crossover probability is too small, the search process will become slow and even stagnate. Therefore, to ensure population evolution, the crossover probability is generally between 0.55 and 0.95. The upper and lower bounds of the crossover probability are $p_{cross}^{max} = 0.95$ and $p_{cross}^{min} = 0.55$.

If the mutation probability is too large, the genetic algorithm becomes a purely random search algorithm. If the mutation probability is too small, it will be difficult to generate new individual structures. Therefore, to ensure the diversity of the individual structures, the crossover probability is generally between 0.01 and 0.2. The upper and lower bounds of the mutation probability are $p_{muta}^{max} = 0.2$ and $p_{muta}^{min} = 0.01$.

In this paper, adaptive crossover and adaptive mutation are proposed. The crossover probability and mutation probability can be automatically changed with the fitness and the number of the iterations. Adaptive crossover sets a larger crossover probability for vulnerable individuals with lower fitness and a larger crossover probability when the number of iterations is smaller, which ensures the diversity of the populations in the early evolutionary stage and

avoids premature convergence. When the number of iterations is small, a smaller mutation probability is used, and when the number of iterations is larger, a larger mutation probability is used. This method ensures that the population performs mainly global search in the early stages of evolution, whereas in the later stages of evolution, mainly local searching is performed to improve the accuracy of the solution.

With a maximum number of iterations of $it_{max} = 120$ and $it_{max} = 150$, as the population size N continues to increase, the MOCTS-AI algorithm is run 50 times; the resulting mean and variance of *GD* and *MS* are shown in Figs. 2 and 5.

Figures 2 and 3 show that the size of the population has a significant effect on the degree to which the algorithm approximates the ideal solution. As the population size increases, *GD* decreases linearly, indicating that the larger the population size is, the better are the antibodies obtained.

Figures 4 and 5 show that the size of the population directly affects the diversity of the population. As the size of the population increases, *MS* gradually increases. The larger the size of the population is, the wider the distribution of the obtained antibodies at the Pareto front, ensuring the diversity of the antibodies.

As shown in Figs. 2 and 5, the amplitudes of the changes in *GD* and *MS* gradually decrease after the population size reaches $N = 150$ for the MOCTS-AI algorithm, indicating that the antibodies have approximated the optimal solution and cover the Pareto front. Compared with the case of $it_{max} = 120$, the algorithm is closer to the optimal solution and the distribution is more uniform in the case of $it_{max} = 150$. However, in the case of $it_{max} = 120$, the number of function evaluations performed by the algorithm lies in the interval [240 000, 360 000], whereas

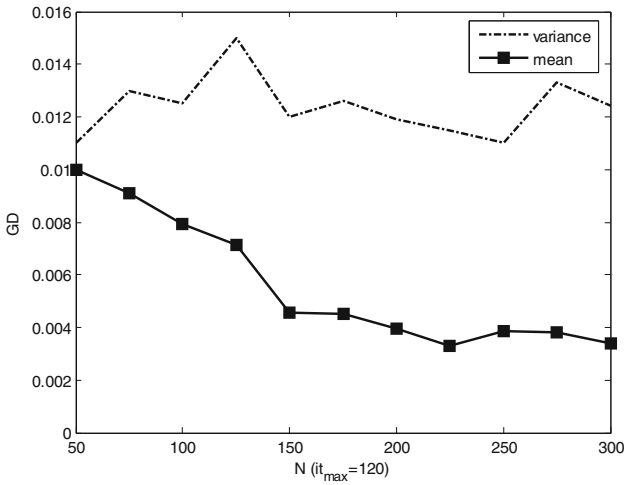

Fig. 2 *GD* values under different population sizes ($it_{max} = 120$)

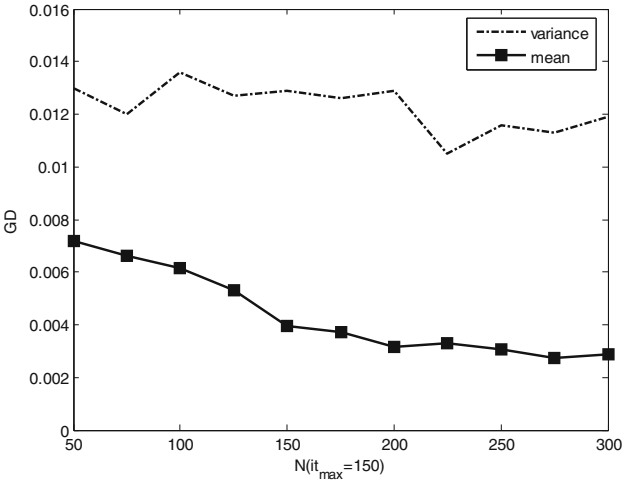

Fig. 3 *GD* values under different population sizes ($it_{max} = 150$)

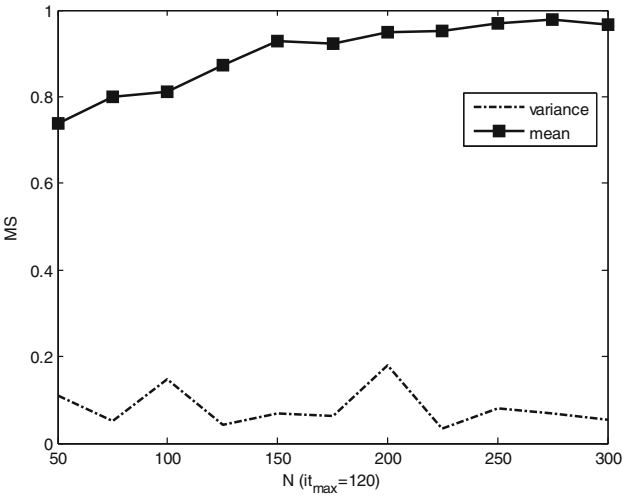

Fig. 4 *MS* values under different population sizes ($it_{max} = 120$)

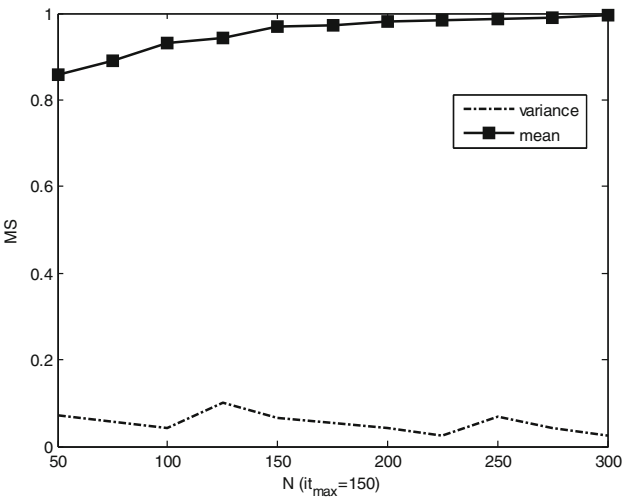

Fig. 5 *MS* values under different population sizes ($it_{max} = 150$)

in the case of $it_{max} = 150$, the number of function evaluations is increased to 1 433 752. These findings show that calculation cost of the algorithm increases greatly when $it_{max} = 150$. Therefore, in the subsequent experiments, the population size is set to $N = 150$, and the maximum number of iterations is set to $it_{max} = 120$. This is done to reduce the calculation cost of the algorithm while guaranteeing the solution quality.

Tables 2 and 3 show comparisons of the average scheduling length and the standard deviation when the MOCTS-AI, MMO, NSGA-II and NNIA algorithms are each run 50 times with increasing number of task nodes n. As shown in Table 2, when the number of tasks is small, the average scheduling length obtained by the MOCTS-AI is not substantially different from those of MMO, NSGA-II and NNIA. However, when the number of tasks is large, the average task scheduling length of the MOCTS-AI is better. As shown in Table 3, the standard deviation of the MOCTS-AI is smaller than those of the other algorithms. The simulation results show that the MOCTS-AI has a more uniformly distributed solution.

To compare the energy consumption of the four algorithms, AWR is introduced as the ratio of the average task scheduling length to the worst-case task scheduling length. When the number of task nodes is $n = 100$, the energy consumption of the system normalized by the energy consumption of the MOCTS-AI is as shown in Fig. 6. As shown in Fig. 6, because the energy consumption is not controlled in NSGA-II and NNIA, the energy consumptions of these algorithms are higher than those of MMO and the MOCTS-AI under the same circumstances. When AWR is small, the MOCTS-AI is more energy efficient than the MMO algorithm, and as the value of AWR increases, although the energy saving effect of the MOCTS-AI is reduced, it remains superior to the other algorithms.

Figure 7 compares the system utilization of the MOCTS-AI, MMO, NSGA-II and NNIA when the population size is $N = 150$ and the number of task nodes is $n = 100$. The MOCTS-AI uses the system utilization as

Table 2 Comparison of the average scheduling length

Tasks number	MOCTS-AI	MMO	NSGA-II	NNIA
25	162	165	169	164
50	358	361	365	360
75	579	601	611	607
100	849	924	927	925
150	1398	1558	1596	1581
200	1961	2184	2194	2189

Table 3 Comparison of the standard deviation of the average of scheduling length

Tasks number	MOCTS-AI	MMO	NSGA-II	NNIA
25	8.9	9.1	9.0	8.9
50	8.3	8.6	9.3	8.7
75	7.0	7.4	7.7	7.0
100	6.2	6.9	6.3	6.8
150	6.3	6.6	7.0	6.4
200	6.2	6.9	6.4	6.2

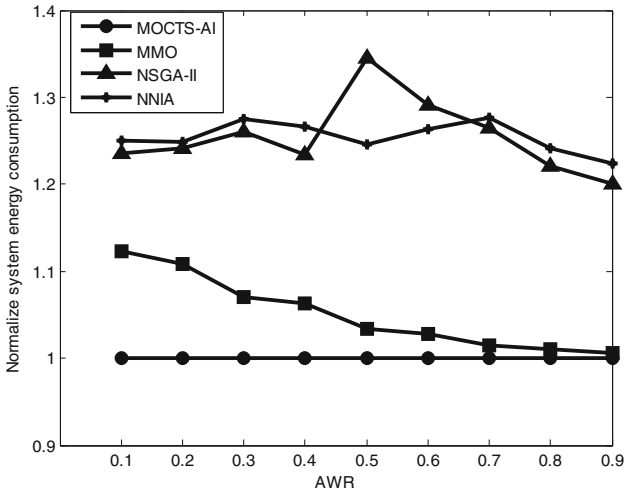

Fig. 6 Comparison of the system energy consumption

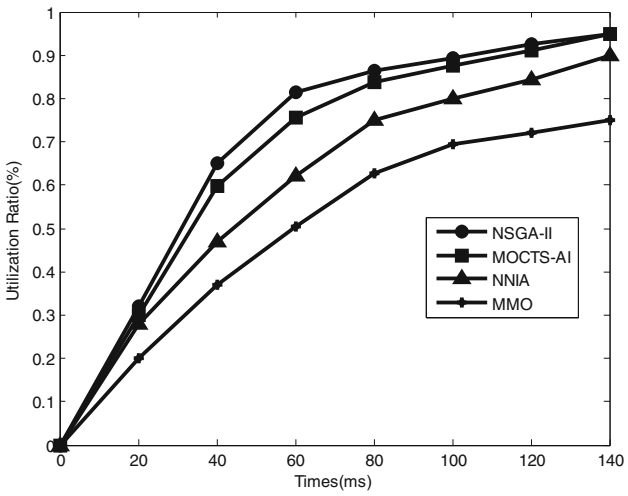

Fig. 7 Comparison of the system utilization

one of the objective functions in the task scheduling process; therefore, the system utilization is improved compared with those of the MMO and NNIA.

5 Conclusion

The multi-objective constrained task scheduling problem in a multi-core processor environment is defined in this paper. The objective functions of the multi-objective task scheduling problem include the task scheduling length, system energy consumption and system utilization. All objective functions and constraints are transformed into antigens in an artificial immune algorithm. In addition, the mapping of tasks to processor cores is transformed into antibodies in artificial immune algorithms. The proposed algorithm accelerates the convergence of the algorithm and ensures the diversity of the solutions by designing the targeted artificial immune operators and the corresponding operation probabilities. The simulation experiment analyzes the convergence of the algorithm and the diversity of the solutions. The results of the comparative analysis show that the proposed algorithm optimizes the scheduling length, system energy consumption and system utilization.

Acknowledgements The funding was provided by The Fundamental Research Funds for the Central Universities, Southwest University for Nationalities (Grant No. 2015NZYQN28), National Natural Science Foundation of China (Grant Nos. 11461006 and 11371003) and Special Fund for Scientific and Technological Bases and Talents of Guangxi (Grant No. 2016AD05050).

References

1. Baruah, S., Bertogna, M., Buttazzo, G.: Multiprocessor Scheduling for Real-Time Systems. Springer, Berlin (2015)
2. Lee, J.: Improved schedulability analysis using carry-in limitation for non-preemptive fixed-priority multiprocessor scheduling. IEEE Trans. Comput. **66**(10), 1816–1823 (2017)
3. Carlos, A.R.C., Zou, X., Cheng, A.M.K.: Real-time multiprocessor scheduling algorithm based on information theory principles. IEEE Embed. Syst. Lett. **9**(4), 93–96 (2017)
4. Wang, J., Gong, B., Liu, H.: Heterogeneous computing and grid scheduling with hierarchically parallel artificial immune optimization algorithms. ICIC Express Lett. Part B **5**, 917–923 (2014)
5. Lin, S.W., Ying, K.C.: Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. Omega **41**(2), 383–389 (2013)
6. Castro, L.R.D., Timmis, J.: Artificial Immune Systems: A New Computational Intelligence Paradigm. Springer, New York (2002)
7. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evolut. Comput. **2**(3), 221–248 (1994)
8. Horn, J., Nafpliotis, N., Goldberg, D.E.: A Niched Pareto genetic algorithm for multiobjective optimization. In: Fogarty, T.C. (ed.) Proceedings of the 1st IEEE Congress on Evolutionary Computation, pp. 82–87. IEEE, Piscataway (1994)
9. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: formulationdiscussion and generalization. In: International Conference on Genetic Algorithms, pp. 416–423. Morgan Kaufmann Publishers Inc., Burlington (1993)
10. Deb, K., Pratap, A., Agarwal, S.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evolut. Comput. **6**(2), 182–197 (2002)
11. Gong, M., Jiao, L., Yang, D.: Corrections on the box plots of the coverage metric in multiobjective immune algorithm with nondominated neighbor-based selection. Evolut. Comput. **17**(1), 131 (2009)
12. Gong, M.G., Jiao, L.C., Du, H.F.: Multi-objective immune algorithm with nondonminated neighbor-based selection. Evolut. Comput. **16**(2), 225–255 (2008)
13. Mu, C., Jiao, L., Liu, Y.: Multiobjective nondominated neighbor coevolutionary algorithm with elite population. Soft Comput. **19**(5), 1329–1349 (2015)
14. Meng, X.F., Xie, W.L.: Research on P2P task scheduling with multi-objective constraints based on immune algorithm. Acta Electron. Sin. **39**(1), 101–107 (2011)
15. Li, Z.Y., Chen, S.M., Yang, B.: Multi-objective memetic algorithm for task scheduling on heterogeneous cloud. Chin. J. Comput. **39**(2), 377–390 (2016)
16. Lee, Y.C., Zomaya, A.Y.: An artificial immune system for heterogeneous multiprocessor scheduling with task duplication. In: International Parallel and Distributed Processing Symposium. DBLP, pp. 1–8 (2013)
17. Zhong, Y., Zhang, L., Li, P.: Sub-pixel mapping based on artificial immune systems for remote sensing imagery. Pattern Recognit. **46**(11), 2902–2926 (2013)
18. Chou, C.L.: Marculescu. FARM: fault-aware resource management in NoC-based multiprocessor platforms. In: IEEE Design, Automation and Test in Europe Conference and Exhibition, pp. 1–6 (2011)
19. Dick, R.P., Rhodes, D.L., Wolf, W.: TGFF: task graphs for free. In: IEEE Proceedings of the Sixth International Workshop on Hardware/Software Codesign, 1998 (CODES/CASHE '98), pp. 97–101 (1998)
20. Duenha, L., Guedes, M., Almeida, H., et al.: MPSoCBench: a toolset for MPSoC system level evaluation. In: IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 164–171 (2014)

Ying Xie received the Ph.D. in computer software and theory from Chendu Institute of Computer Application, Chinese Academy of Science, China. She is a lecturer at SouthWest Minzu University, China. Her research interests cover the fields of embedded system, formal verification, model checking, distributed computing.

Jinzhao Wu received the Ph.D. of science from the System Science Institute of China Science Academy, China, in 1994. Currently, he is a professor with Chinese Academy of Sciences, China. His research areas include hybrid system, symbolic computation, theorem proving and formal verification.