Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 287475, 9 pages
http://dx.doi.org/10.1155/2014/287475

Research Article

Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm

Weizhe Zhang,[1] Hucheng Xie,[1] Boran Cao,[1] and Albert M. K. Cheng[2]

[1] *School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China*
[2] *Department of Computer Science, University of Houston, Houston, TX 77204, USA*

Correspondence should be addressed to Weizhe Zhang; wzzhang@hit.edu.cn

Received 7 April 2014; Accepted 6 August 2014; Published 28 August 2014

Academic Editor: Yang Xu

Energy consumption in computer systems has become a more and more important issue. High energy consumption has already damaged the environment to some extent, especially in heterogeneous multiprocessors. In this paper, we first formulate and describe the energy-aware real-time task scheduling problem in heterogeneous multiprocessors. Then we propose a particle swarm optimization (PSO) based algorithm, which can successfully reduce the energy cost and the time for searching feasible solutions. Experimental results show that the PSO-based energy-aware metaheuristic uses 40%–50% less energy than the GA-based and SFLA-based algorithms and spends 10% less time than the SFLA-based algorithm in finding the solutions. Besides, it can also find 19% more feasible solutions than the SFLA-based algorithm.

1. Introduction

Multiple processing in heterogeneous computing platforms adapts to different types of computing needs. Using multiple processing platforms will improve the system performance and satisfy the increase in energy consumption. However, assigning real-time tasks to a multiprocessor implementation proves to be an NP-hard problem. The problems of real-time task allocation in a heterogeneous environment have been studied extensively in the existing references. However, most of the studies focus on the performance metrics of how to minimize the maximum utilization and these problems can be mapped to the traditional makespan problem [1].

Energy consumption has become a major problem in computer systems; the processor consumes most of the energy, especially in embedded systems, where the excessive energy consumption will cause serious pollution and waste of resources in the natural environment [2, 3]. Therefore, how to reduce processor energy consumption becomes a widespread concern. We need to focus on the problem from reducing the maximum utilization to energy consumption under the premise of meeting the specified task deadlines.

Although it is an NP-hard problem, there are many approximation algorithms for solving the problem of real-time task allocation in a heterogeneous processor environment, including traditional real-time task scheduling algorithms such as deadline-monotonic (DM) algorithm [4], rate-monotonic (RM) algorithm [5], least-laxity-first (LLF) algorithm [6], earliest-deadline-first (EDF) algorithm [5], and linear programming-based (LP) algorithm [7] and the swarm intelligence algorithms such as ant colony optimization (ACO) [8], genetic algorithm (GA) [9–11], and shuffled frog-leaping algorithm (SFLA) [12, 13]. In these studies, most algorithms do not consider energy consumption factors. Besides, the number of feasible solutions and energy saving are in conflict. Therefore, we need to find a new algorithm to solve this multiobjective optimization problem.

The heuristic algorithm in [14] is an adaptive algorithmic structure; it can be used to adapt to a series of relatively wide range of issues. Though many heuristic algorithms exist, the particle swarm optimization (PSO) algorithm emerges as a novel heuristic algorithm in recent years. This algorithm is inspired by the social behavior of a group of migratory birds that try to reach an unknown destination. Each bird is referred to as a particle. Each particle has a fitness value determined by the function to be optimized and a speed that determines their flight direction and distance; then the particle with the current optimal particle to search the optimal solution is chosen in the solution space. Compared with the genetic algorithm, the PSO algorithm has no processes such as reproduction, crossover, and mutation; it is only through simple operation for evolution, which is easy to achieve, and the efficiency is better.

The aim of this work is to propose a new algorithm to solve the problem of real-time task scheduling in a heterogeneous processor environment, under the premise of meeting all task deadlines to reduce energy consumption.

The main objective of the work is as follows.

(1) Formulate the real-time task scheduling problem based on energy awareness and add more constraints. Put the energy consumption as the utility into the constraint condition.

(2) Based on the PSO algorithm, propose one algorithm that can solve the problem of real-time task scheduling based on energy awareness, which can find as many feasible solutions as possible before the specified deadlines and minimize the energy consumption.

(3) Through a series of comprehensive experiments, make a comparison of the proposed algorithm with the existing traditional algorithms and the other heuristic algorithms, to improve the algorithm so as to achieve the purpose of the optimization.

This paper is organized as follows. Section 2 presents the state of the art of the task scheduling problem in multiprocessor platforms. Section 3 formulates the problem of real-time task scheduling in heterogeneous processors based on energy awareness. Overview of the PSO algorithm and the proposed energy-aware real-time task scheduling algorithm based on the PSO algorithm is introduced in Section 4. We analyze the performance and results of the proposed algorithm in Section 5. The final part gives the conclusion and summary and provides directions for future work.

2. Related Works

Baruah [15] has made a study on task scheduling in heterogeneous multiprocessor platforms, and some improvements have been made for the ACO heuristic algorithms and the improved algorithm performs very well in finding the feasible solutions under time constraint. Braun et al. [14] conclude 11 heuristic algorithms that can be applied to task scheduling in heterogeneous multiprocessor platforms to reduce the execution time. However, Braun et al. assume that each task on each machine has accurate execution time and has no time constraints. In the experimental results, the GA has a good performance. So far, there are available heuristic algorithms such as GA, ACO algorithm, SA algorithm, PSO algorithm, and SFLA algorithm [12, 13], and these algorithms have been applied to task scheduling in multiprocessor platforms.

Baruah [16] converts the task scheduling problem in multiprocessor platforms into an ILP problem and proposes an approximate polynomial time algorithm. However, the LP problem has a lot of feasible solutions; a polynomial time algorithm is not guaranteed to find the fundamental solution. Similarly, Leung and Whitehead [4] convert the task scheduling problems in multiprocessor platform, in which tasks can be divided and have priority to LP. However, they believe that each task can be arbitrarily segmented, but this assumption is limited in practice.

Baruah [17] puts forward a polynomial time algorithm for the task scheduling problem in multiprocessor platforms in which tasks are preemptive and transitive. Its purpose is to achieve task scheduling under a series of real-time tasks constraints in heterogeneous processor platforms. However, they ignore the communication overhead and that a task is dividable.

The task scheduling problem in multiprocessor platforms not only needs to solve the increasing number of the feasible solutions but also reduces the energy consumption of each found feasible solution. At present, there are a lot of researchers studying task scheduling in multiprocessor platforms to reduce energy consumption. But, in general, the PSO algorithm has not been used in these subjects.

Cheng et al. [8] propose an improved ACO algorithm in multiprocessor platforms task scheduling which can find sufficient feasible solutions, while satisfying the time constraints. But this algorithm is not PSO related and does not make a certain improvement in energy consumption. Baruah [17] in multiple processing scheduling applications describes a non-ACO algorithm. The algorithm can effectively reduce energy consumption and reduce scheduling time, but the algorithm needs a premise: all the tasks must have the same computation time. In addition, the algorithm is not PSO-based. Aydin and Yang [18] propose the worst-fit-decreasing algorithm in multiprocessor task scheduling to reduce energy consumption and meet the deadline. However, this heuristic algorithm is not a PSO-based algorithm. Zhu et al. [19] put forward the corresponding effective algorithm in multiprocessing task scheduling, but it is not PSO-based algorithm, either.

The evolutionary algorithms thick swarm intelligence optimization algorithm as the goal, such as ACO algorithm, evolution strategy, and GA, can solve the problem of multiobjective combinatorial optimization and obtain a better solution, but the algorithm is complex and has low efficiency. So, looking for a more effective task scheduling and allocation algorithm is very important.

Particle swarm optimization (PSO) algorithm [20, 21] is a new global optimization algorithm, the same with the other swarm intelligence algorithm; all belong to the group of intelligent evolutionary computation technology. Randomly initialize population and then evaluate it according to the fitness function, so as to determine whether to have further search. However, the PSO-based algorithm has no operation such as reproduction, crossover, and mutation, only works through simple arithmetic for evolution, and is simple and easy to achieve. As an important tool of optimization, the PSO-based algorithm can be applied in cloud computing and information retrieval [22, 23].

Table 1: The parameter of the PSO algorithm.

Number	Parameter	Description
1	CP	The current position of the particle
2	BP	The best position of the particle
3	V	The speed of the particle
4	C_1	The learning factor of the particle
5	C_2	The learning factor of the particle
6	ω	The inertia weight of the particle
7	Rand()	The random function between 0 and 1
8	Fitness	The quality evaluation standard of particle

3. Problem Formulation

Each task is assigned to a particular processor and does not exceed any of the computing capacity of the processor without exceeding the deadline of the task. In general, the computation time and deadline for each task are known. But for now, some real-time tasks are dynamically changed. A series of periodic tasks is assigned to the series of heterogeneous processor and does not exceed the deadline. The problem is an NP-hard problem. We solve this problem based on particle swarm optimization (PSO).

3.1. Heterogeneous Multiprocessors Platforms. HMP $= \{P_1, P_2, \ldots, P_m\}$ is a heterogeneous multiprocessor platform. P_j in each clock cycle executes only one command and determines speed according to the type of task. $S_{i,j}$ is the clock frequency and is the speed to perform a specific task T_i. $e_{i,j}$ refers to the execution time of T_i on the P_j, $e_{i,j} = c_i/s_{i,j}$, where c_i is the clock cycles needed for the execution of T_i task.

3.2. Periodic Task Set. PTS $= \{T_1, T_2, \ldots, T_n\}$ consists of n real-time tasks. T_i is made up of a binary group (e, p), where e presents WCET (it is estimated as the worst case execution time); p is the task period. T_i generates an infinite sequence of tasks; each task is at most e time units, and the interval is p time units. The deadline of each T_i is p time units after the arrival of T_i.

3.3. Real-Time Task Scheduling, Energy Utilization, and Energy Consumption. We build a task scheduling situation matrix $X_{n\times m}$ (see Table 2). Matrix element $x_{i\times j}$ indicates whether task T_i can be assigned to processor P_j. The value of element $x_{i\times j}$ is 0 or 1, respectively, which indicates that task T_i is not assigned to the processor P_j and task T_i is already assigned to processor P_j.

The energy consumption matrix in the real-time task scheduling problem on heterogeneous processors is presented by $U_{n\times m}$; its element $u_{i,j}$ is computed as $u_{i,j} = e_{i,j}/p_i$ which shows the energy consumption it takes to execute task T_i on processor P_j. $u_{i,j}$ is a real number whose range will be set $(0,1) \cup +\infty$; if task T_i cannot run on processor P_j, then $u_{i,j}$ is set $+\infty$.

Table 2: The energy utility matrix of m processors with n tasks.

	P_1	P_2	P_3	$\cdots$	P_{m-1}	P_m
T_1	$u_{1,1}$	$u_{1,2}$	$u_{1,3}$	$\cdots$	$u_{1,m-1}$	$u_{1,m}$
T_2	$u_{2,1}$	$u_{2,2}$	$u_{2,3}$	$\cdots$	$u_{2,m-1}$	$u_{2,m}$
T_3	$u_{3,1}$	$u_{3,2}$	$u_{3,3}$	$\cdots$	$u_{3,m-1}$	$u_{3,m}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_{n-1}	$u_{n-1,1}$	$u_{n-1,2}$	$u_{n-1,3}$	$\cdots$	$u_{n-1,m-1}$	$u_{n-1,m}$
T_n	$u_{n,1}$	$u_{n,2}$	$u_{n,3}$	$\cdots$	$u_{n,m-1}$	$u_{n,m}$

Energy consumption of T_i in processor P_j on each cycle is as follows:

$$E_{i,j} = \text{Power}_{i,j} \times e_{i,j} \approx \left(C_{ef} \times \frac{s_{i,j}^3}{k^2} \right) \times e_{i,j} = \frac{C_{ef}}{k^2} \times c_i \times s_{i,j}^2, \tag{1}$$

where C_{ef} and k are constants. Thus, $E_{i,j} \propto c_i \times s_{i,j}$, and the energy consumption is linear.

The total energy consumption on the m processors is

$$\text{Energy}_{n\times m} = \sum_{i=1}^{n} \sum_{j=1}^{m} E_{i,j} \times x_{i,j}. \tag{2}$$

Here we define the theoretical maximum energy consumption value as

$$\text{Max\_Energy}_{n\times m} = \sum_{i=1}^{n} \underset{j=1}{\overset{m}{\text{MAX}}} \left(E_{i,j} \right). \tag{3}$$

3.4. Constraint Model. On the basis of the defined energy consumption matrix $U_{n\times m}$, the constraint model of the real-time task scheduling problem in heterogeneous processor is given. The constraint model consists of the following three parts:

(1) $\sum_{j=1}^{m} x_{i,j} = 1$, $(i = 1, 2, 3, \ldots, n)$,

(2) $\sum_{i=1}^{n} x_{i,j} * u_{i,j} \le U$, $(j = 1, 2, 3, \ldots, m)$,

(3) $x_{i,j}$ is either 0 or 1, $(i = 1, 2, 3, \ldots, n;\ j = 1, 2, 3, \ldots, m)$,

where U represents the maximum amount of computation each processor allows and will be set to 1 in our experiments.

3.5. Calculation of the Fitness Function. The fitness function is defined as the ratio of actual energy consumption and theoretical energy consumption. By (1), we assume that C_{ef} and k are constants and are set as 1. The theoretical maximum energy consumption is calculated as MaxEC $= c_i \times s_{i,j}^2$. The actual energy consumption can be computed as dEC $= \sum_{i\le m, j\le n} \text{energy}[i][j]$. Thus, the fitness function is defined as follows:

$$\text{fitness} = \frac{\text{dEC}}{\text{MaxEC}}. \tag{4}$$

3.6. Energy-Aware Real-Time Task Scheduling Problem for Heterogeneous Multiprocessors. Given HMP $= \{P_1, P_2, \dots, P_m\}$ and PTS $= \{T_1, T_2, \dots, T_n\}$, we name eRTSP energy-aware real-time tasks scheduling problem in heterogeneous processors. eRTSP has two conflict optimization aims. The first one is to look for each task assigned to a specific processor and makes the utilization of each processor that does not exceed its maximum utilization. The second one is the energy consumption, which is to find a feasible solution to minimize the energy consumption on the corresponding processor.

4. PSO Algorithm for Energy-Aware Real-Time Task Scheduling Problem

4.1. Introduction to the PSO Algorithm. The PSO algorithm [21] was first proposed by Eberhart and Shi. It is a kind of evolutionary computation theory. The PSO algorithm is inspired by a social behavior of a group of migrants trying to reach an unknown destination. In the PSO algorithm, each solution is a group of birds and each bird is said to be a particle. All particles have a fitness value which is determined by the function to be optimized and each particle has a speed which determines its flight direction and distance and then the particle searches the optimal solution in solution space with the current optimal particle. The PSO algorithm and GA are both based on the iterative method. A particle is similar to a chromosome in the GA. But unlike the GA, an evolutionary process does not generate new members from the parent member in the PSO algorithm but only changes its own social behavior according to the process of moving towards the destination.

In fact, the PSO algorithm imitates the communication of the birds when they are flying together. Each bird moves towards a certain direction; when in communication, it determines the best position. Therefore, each bird depends on the current position at a particular speed towards the best birds. Then, each bird forms its new location to view their search space and repeats the process until the bird reaches the desired destination. It is important to note that the process also involves the interaction and intelligence in the community, in order to learn from their own experience (local search) and from the surrounding particles experience (global search).

The PSO algorithm is initialized in the initial time for a group of random particles. The ith particle is presented as the position of an S-dimensional space as a point and S is the number of variables. In the entire process of the PSO algorithm, each particle i displays three variables: the current position of the particle CP_i, the best position of the previous iteration of the loop the particle has reached BP_i, and flight speed of the particle V_i. These three variables are represented with a component form as follows:

$$\begin{aligned} \mathrm{CP} &= (cp_1, cp_2, \dots, cp_i, \dots, cp_n), \\ \mathrm{BP} &= (bp_1, bp_2, \dots, bp_i, \dots, bp_n), \\ V &= (v_1, v_2, \dots, v_i, \dots, v_n). \end{aligned} \tag{5}$$

In each time period, the best position CP_g of particles g is calculated as all of the best adaptations. Therefore, each particle updates its own speed V_i to catch up on the best particle g as follows:

$$\begin{aligned} V_i{}' = \omega \times V_i + c_1 \times \operatorname{rand}() \times (\mathrm{BP}_i - \mathrm{CP}_i) \\ + c_2 \times \operatorname{Rand}() \times (\mathrm{BP}_g - \mathrm{CP}_i). \end{aligned} \tag{6}$$

According to the above formula and making use of the new speed, we update the position of the particle as follows:

$$\mathrm{CP}_i' = \mathrm{CP}_i + V_i', \tag{7}$$

in which $-V_{\max} \le V_i \le V_{\max}$.

We called c_1, c_2 the learning factors which are two constants; rand() and Rand() are two random functions which range in $[0, 1]$; $V_{\max}$ is the maximum velocity limit of the particle; ω is an inertia weight used to affect the current speed. In the formula (6), the second component presents the thought of its current position and the best position. On the other hand, represented by the formula (1), the third component is the cooperation between the particles, comparing the current position of a particle and the best position.

4.2. Applying the PSO Algorithm to eRTSP

4.2.1. Building Energy Matrix and Time-Consuming Matrix. The eRTSP problem can be represented as a bipartite graph. There are two types of nodes: PTS and HMP. A task is mapped to a node of PTS, and a processor is mapped to a node of HMP. If and only if a task can be assigned to the corresponding processor and does not exceed the maximum computing power limit, there is an edge between the two nodes. This assignment consumption directly relates to the energy consumption of the task on the processor.

Therefore, in general, we construct an $n \times m$ energy matrix: n represents the n tasks, m represents the m processors, and $u_{i,j}$ is represented by the energy utilization of the task i in the jth processor. Each value of the matrix is set as $(0, 1) \cup +\infty$; if no tasks are assigned on the particular processors, we set the corresponding value of the element in the matrix $+\infty$. Now, we define the constraints: in each row there can only be an element to be visited; accumulated value of the energy of each column cannot exceed 1.

The same as energy consumption matrix, we can build a matrix recording the running time of a task in the corresponding processor. Each element in the matrix $dExeTime[i][j]$ of the execution time is $dExeTime[i][j] = nCycles/nSpeed$.

4.2.2. The Update of the Velocity, Position, and Inertia Weight of the Particle. The velocity of the particles is the critical factor for the positions of the particles. The velocity of the particles will affect the overall convergence of the PSO algorithm and will affect the efficiency of the algorithm's global searching. We consider (6) as a speed profile. The particle's position updates present the next position of the task. As the particle position updates, we have mentioned formula (7) in the third section, $\mathrm{CP}'_i = \mathrm{CP}_i + V'_i$. When $V'_i > 0$, it indicates that it needs to adjust the number of the processor, and then, $\mathrm{CP}'_i = V'_i$; otherwise, the position of the particle remains unchanged; that is, $\mathrm{CP}'_i = \mathrm{CP}_i$.

The parameter ω in the PSO algorithm plays a balanced role in global searching and local searching. And over time, the number of iterations increases gradually while ω linearly reduces. The formula of updating ω is

$$\omega = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{t_{\max}} \times t, \qquad (8)$$

where t is the number of iterations and $t_{\max}$ is the total number of iterations.

4.2.3. Optimization of the Energy Consumption. When we find a feasible solution by the PSO algorithm, we often need to optimize the feasible solution to achieve the second objective: energy consumption target, that is, forthcoming a feasible solution with high energy consumption through a task assigned to other processor or exchanging their corresponding processor running two tasks to reduce the overall energy consumption.

In the initial state, for a processor if its utilization is greater than 1, we extract the task with maximal energy consumption in the processor, run this task in the processor with the lowest utilization, and compare the utilization of the processor to see whether it is greater than 1. If it is not greater than 1, then the corresponding coordinate of this task is updated.

Thereafter, in accordance with the calculated corresponding local and global optimum position of each task and from formula (2), the speed of the particles has to be updated. Subsequently, we check the speed of particles, in case utilization is less than the upper limit of the maximum utilization of each processor; if the speed V is greater than $V_{\max}$, V will be assigned to $V_{\max}$; if the speed is less than 0, then the speed is set as 0.

In the optimization, first we backup and then analyze the following three cases.

(1) Particle.v > 0 and Particle.v ≤ $V_{\max}$. Let *Particle.x* equal *Particle.v* and calculate the corresponding utilization of the processor, in case guaranteed utilization is less than 1, recalculating fitness value. If the energy consumption ratio has been decreased, we modify the original solution and update the value of *Particle.x*. If there is no reduction of the energy consumption ratio, we do not change the original solution.

(2) Particle.x > x_{up}. We will let the value of *Particle.x* be x_{up} and recalculate the corresponding processor utilization of x_{up}, in case guaranteed utilization is less than 1; we recalculate the fitness value and observe whether the energy consumption ratio has been decreased. If declined, we will alter the original plan; if not, we will not change the original plan for the assignment.

(3) Particle.x < 0. The general idea is the same with the second case; we will assign *Particle.x* to 0 and recalculate the utilization of their corresponding processor, in case of utilization is less than 1; we recalculate the fitness value and observe the ratio of the energy consumption to see whether it has been declined; we will change the original plan for the scheduling if so; if not reduced, we will not change the original plan for the assignment.

We assume that if the fitness value does not decrease or remain the same; we quit the iteration and return after iteration 1000 in the PSO algorithm.

4.2.4. PSO Algorithm for eRTSP. See Algorithm 1.

5. Experiment and Result Analysis

In this section, at first, as for the PSO algorithm, we want to determine its parameters in resolving eRTSP. After that, we solve the eRTSP problem with the PSO algorithm and analyze the comparison of the performance of the PSO algorithm, GA, and SFLA in eRTSP with the solution quality and energy consumption.

5.1. Environment of the Experiments

CPU: Intel Core 2 CPU 1.67 GHz.

Cache: 512 KB.

Memory: 2074492 KB.

Operating system: Windows 7.

Development platform: Visual Studio 2003.NET.

We will get the results from a large number of randomly generated problem sets with the PSO algorithm. There are a lot of different situations in problem sets and each issue is initialized as m processors and n tasks.

5.2. The Parameter of the PSO Algorithm. According to the PSO algorithm, there are three parameters ω, c_1, and c_2, which impact the performance of the PSO algorithm (see Table 1). ω denotes the inertia weight heavy; c_1 and c_2 denote the acceleration. The following experiments are set to determine the best combination of the three parameters. The results are shown in Table 4.

As seen from the results in Table 4, the results of the different parameters of the PSO algorithm running the same eRTSP problem are not the same; the parameters for $\omega = 1$, $c_1 = 2$, and $c_2 = 2$ in this group when solving eRTSP get the largest number of feasible solution and its running time is the shortest. Therefore, in the subsequent experiments, we will select the parameter in this group in solving the eRTSP and compare the performance with the GA and SFLA.

5.3. The Comparison of the Results among the PSO Algorithm, GA, and SFLA in eRTSP. To show a wider range of heterogeneous environments, the use of matrix values is varied. For a periodic task T_i, the definition of the task frequency is the average speed of execution of the tasks before deadline and is defined as c_i/p_i. In the PTS, the variance of the frequency of the task is defined as task heterogeneity. In the HMP, for

```
Input: set of HTM and PTS, $\omega$, $c_1$, $c_2$
Output: feasible solution
(1) Initialize the inertia factor $\omega$, learning factor $c_1$, $c_2$
(2) Calculate the eRTSP instances theoretical maximum energy consumption
(3) Randomly generated the corresponding number of particle represents the corresponding number of tasks
(4) While the current number of iterations is less than the set max number of iterations
(5) Do
(6)      Calculated for all processors, fitness fitness, that is, the ratio of the current energy consumption and
         maximum energy consumption;
(7)      Calculated for each processor load does not reach 100%;
(8)      If it reaches 100%
(9)          The task is to re-arrange the largest processor utilization to its share of the minimum utilization
             will not exceed the maximum processor utilization processor;
(10)     Else
(11)         For each task
(12)             calculate the local best position pbest according to a task;
(13)             calculated the best position gbest according to a task;
(14)         End each
(15)     update the speed of the particle;
(16)     Check whether the velocity is negative or exceeds the maximum;
(17)     If the velocity of the particle is greater than $V_{max}$
(18)         Change speed of the particle to $V_{max}$;
(19)     If the velocity of the particle is less than 0
(20)         Change the speed to 0;
(21)     Energy Optimization;
(22)     Best fitness of all particles is is gbest calculated as a processor;
(23)     Update the inertia factor $\omega$;
(24)     If the degree of fitness does not change more than 10 times,
(25)         Then exit the iteration loop to obtain the final solution;
(26) EndWhile
```

Algorithm 1: PSO algorithm for eRTSP.

Table 3: The parameter of GA and SFLA.

No.	GA		SFLA	
	Parameter	Value	Parameter	Value
1	Crossover rate	60%	Population size	200.00
2	Mutation rate	40%	Subpopulation size	20.00
3	Population size	200.00	Subpopulation iterations	10.00
4			Total iterations	100.00

Table 4: The value of the parameter of the PSO algorithm.

ω	c_1	c_2	Fes	Time (ms)
0.8	2	2	11	4.30
1.0	2	2	15	3.94
0.4	0	0	12	5.65
0.4	4	0	12	5.16
0.4	0	4	11	4.98
0.4	4	4	12	9.30
0.9	0	0	12	17.75
0.9	0	4	10	7.37
0.9	4	0	10	7.12
0.9	4	4	7	6.21

a given task, the variance of the processing time of each processor is defined as the processor heterogeneity.

The method of generation of the consumption utilization matrix of the task is as follows.

(1) Generate a vector C with n random elements in the range of $[100, 1000]$; its element C_i represents the implementation of clock cycles for each of the tasks T_i.

(2) Construct a vector containing n floating-point type elements T_B; the size of its elements is in the range $[1, \varphi_T]$. Said that the task of the heterogeneity. This vector can also reflect the frequency of the task T_i.

(3) For each column vector S_i, it contains m elements; the size of its elements $[\varphi_T, \varphi_T \times \varphi_B]$ indicates the degree of processor heterogeneity.

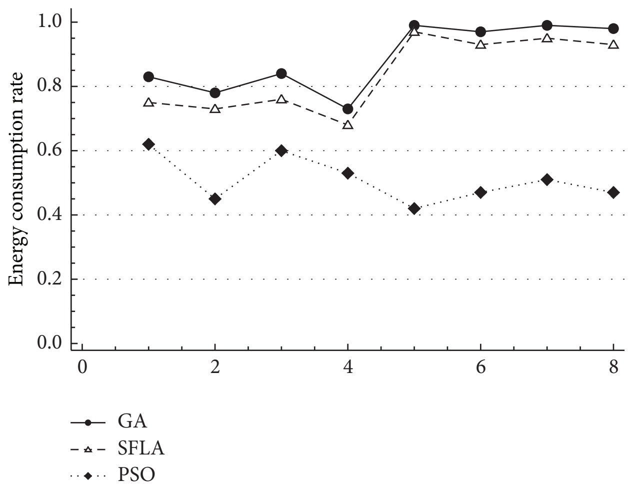

Figure 1: Energy consumption comparison among GA, SFLA, and PSO.

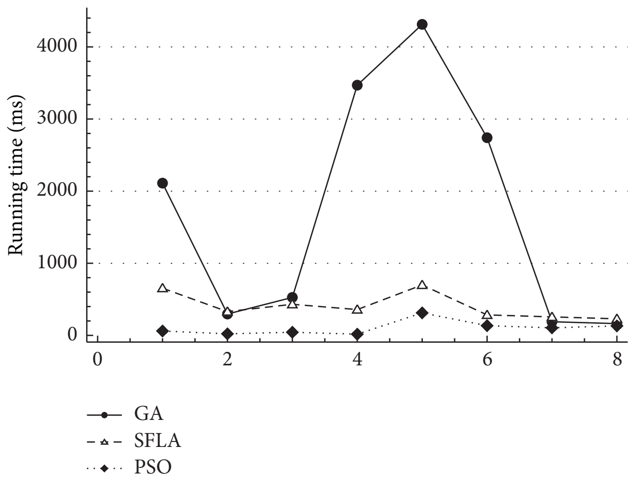

Figure 2: Running time comparison among GA, SFLA, and PSO.

(4) Configure utilization of an $n \times m$ matrix, whose element is $\mathrm{TB}(i)/\mathrm{Si}(j)$. Accordingly, the size of the elements is affected by the degree of task and processor heterogeneity. The element size range is $[0, 1/\varphi_T \times \varphi_B]$.

In order to obtain the true and objective evaluation of the performance of each algorithm, the characteristics of the utilization matrix are task heterogeneity, processor heterogeneity, and consistency. Therefore, we generated a combination of eight kinds of experiments according to the above features: using the matrix of the high and low task heterogeneity, high and low processor heterogeneity, and being consistent or nonconsistent. High task heterogeneity is represented as 100; low task heterogeneity is expressed as 5. Highly heterogeneous processor is represented as 20; low heterogeneous processor is represented as 5. When the processor P_j performs any task shorter than the processor P_k, we use of matrix consistency. A consistent utility matrix is generated by sorting each vector, and P_0 has the fastest processing speed of all the processors and processor $P_{(m-1)}$ is the slowest. In contrast, nonconsistent utility matrix is that processor P_j processes fast on certain tasks than the processor P_k, but the processing speed is slow in the other tasks. It is an unsorted matrix randomly generated. Above 8 experimental category combinations are shown in Table 5.

From Figure 1, on the energy consumption aspect, the energy consumption of the GA and SFLA is higher than the PSO algorithm, wherein the PSO energy consumption using the different test dataset is about 40–50% of the GA and SLFA in energy consumption.

Figure 2 and Table 3 show that the three algorithms running time is different in the same environment of eRTSP; the running time of the GA is the longest, followed by the SFLA. The PSO algorithm is the fastest in 8 groups for the test in general. In particular, we focus on comparing the PSO algorithm and SLFA in the first problem set, finding that the PSO algorithm running time is 10% of the running time of the SFLA. Under consistency utility matrix conditions, the running time of SFLA and the PSO algorithm is essentially the same.

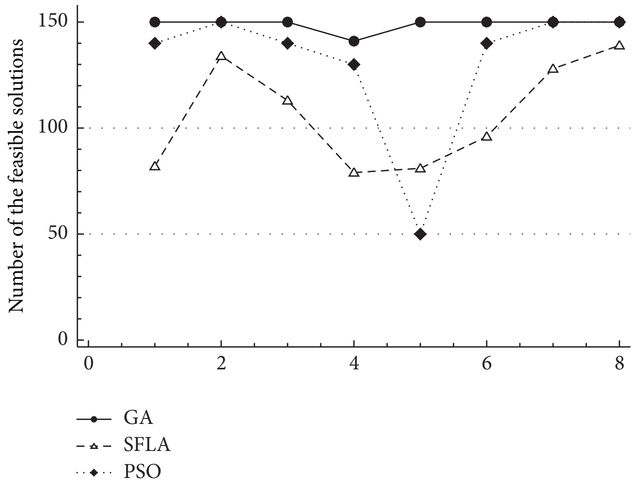

Figure 3: Comparison on searching the number of feasible solutions among GA, SFLA, and PSO algorithms.

Table 5: Eight utilization matrix scales of PSO's parameter test.

No.	Config.	Size
T1	C_HT_HP	U_{90*4}
T2	C_HT_LP	U_{50*8}
T3	C_LT_HP	U_{70*4}
T4	C_LT_LP	U_{40*8}
T5	IC_HT_HP	U_{140*5}
T6	IC_HT_HP	U_{50*8}
T7	IC_LT_HP	U_{90*4}
T8	IC_LT_LP	U_{50*8}

In Figure 3, we compare the three algorithms to find feasible solution volume. The number of feasible solutions found by SFLA and PSO algorithm is less than the GA (see Table 6). The PSO algorithm in the ability to find feasible

Table 6: Average runtime, number of feasible solutions, and energy consumption with GA, SFLA, and PSO algorithm.

No.	GA			SFLA			PSO		
	Avg.	Feas.	Energy	Avg.	Feas.	Energy	Avg.	Feas.	Energy
T1	2110.95	150	0.83	654.33	82	0.75	60.83	140	0.62
T2	296.07	150	0.78	329.54	134	0.73	21.62	150	0.45
T3	526.51	150	0.84	431.03	113	0.76	43.47	140	0.60
T4	3469.31	141	0.73	357.84	79	0.68	16.55	130	0.53
T5	4312.55	150	0.99	699.16	81	0.97	312.04	50	0.42
T6	2741.01	150	0.97	283.86	96	0.93	134.51	140	0.47
T7	189.20	150	0.99	256.37	128	0.95	104.81	150	0.51
T8	163.21	150	0.98	229.73	139	0.93	131.36	150	0.47

solution is slightly worse than the GA. Not all of feasible solutions in the GA in the fourth set of experiments are found; however, in other conditions all are found, but the PSO algorithm has a stronger ability to find the feasible solutions than the SFLA. The feasible solution number of the PSO algorithm is 50% more than that of the SFLA algorithm.

Therefore, from the above results we can get that energy consumption and running time of the PSO algorithm on the eRTSP are relatively small, compared with SFLA; GA and has certain advantages. The GA mainly uses the crossover and mutation method and the running time of looking for a feasible solution is much slower and the energy consumption is larger. Running time with SFLA is slower and the number of feasible solutions is less than the PSO algorithm, and the PSO algorithm can find a feasible solution in most cases. When operating time between SFLA and PSO algorithm is similar, energy consumption with the PSO algorithm uses lower energy consumption than SFLA. Therefore, considering the above several function tests, it can be said that the PSO algorithm outperforms GA and SFLA.

6. Conclusions

This paper has a formal description of the real-time task scheduling problem in a heterogeneous environment based on energy consumption and puts forward a new heuristic algorithm based on the particle swarm optimization algorithm to solve the problem. The proposed algorithm not only finds much more feasible solutions within the specified time but also optimizes the energy consumption. According to the results of extensive experiments, the PSO algorithm has a better performance in reducing energy consumption and running time and increasing the number of feasible solutions. The energy consumption is only 40–50% of GA and SLFA. In addition, in finding the feasible solution volume, the PSO algorithm finds a total of 19% more feasible solutions than SFLA in 7 out of 8 sets of test data and finds about 8% less feasible solutions than GA. At running time, the PSO algorithm is faster than GA and SFLA and about 10% faster than SFLA.

The current study in this paper focuses on independent, nonpreemptive, and periodic tasks, but many other factors are not taken into account in real-time task scheduling problem in a heterogeneous processor environment. In the future, we will reduce the constraint conditions and study the problem of real-time task scheduling with priority and communication between tasks.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant no. 61173145, the National Basic Research Program of China under Grant no. G2011CB302605, and the National High Technology Research and Development Program of China under Grant no. 2011AA010705. Albert M. K. Cheng is supported in part by the US National Science Foundation under Awards nos. 0720856 and 1219082.

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS '01)*, pp. 95–105, London, UK, December 2001.

[2] V. Tiwari, D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez, "Reducing power in high-performance microprocessors," in *Proceedings of the 35th Design Automation Conference*, pp. 732–737, June 1998.

[3] D. M. Brooks, P. Bose, S. E. Schuster et al., "Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors," *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[4] J. Y. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237–250, 1982.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multi-programming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 47–61, 1973.

[6] A. K. Mok, *Fundamental design problems of distributed systems for hard-real-time environments [Ph.D. thesis]*, Massachusetts Institute of Technology, Cambridge, Mass, USA, 1983.

[7] L. Puente-Maury, P. Mejía-Alvarez, and L. E. Leyva-Del-Foyo, "A binary integer linear programming-based approach for solving the allocation problem in multiprocessor partitioned scheduling," in *Proceedings of the 8th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE '11)*, Merida City, Mexico, October 2011.

[8] H. Chen, A. M. K. Cheng, and Y. Kuo, "Assigning real-time tasks to heterogeneous processors by applying ant colony optimization," *Journal of Parallel and Distributed Computing*, vol. 71, no. 1, pp. 132–142, 2011.

[9] G. Menghani, "A fast genetic algorithm based static heuristic for scheduling independent tasks on heterogeneous systems," in *Proceedings of the 1st International Conference on Parallel, Distributed and Grid Computing (PDGC '10)*, pp. 113–117, October 2010.

[10] W. Sun, "A novel genetic admission control for real-time multiprocessor systems," in *Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '09)*, pp. 130–137, December 2009.

[11] U. Manchon, C. Ho, S. Funk, and K. Rasheed, "GART: A genetic algorithm based real-time system scheduler," in *Proceedings of the IEEE Congress of Evolutionary Computation (CEC '11)*, pp. 886–893, New Orleans, La, USA, June 2011.

[12] M. M. Eusuff and K. E. Lansey, "Optimization of water distribution network design using the shuffled frog leaping algorithm," *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 210–225, 2003.

[13] L. Xue-hui, Y. Ye, and L. Xia, "Solving TSP with Shuffled Frog-Leaping Algorithm," in *Proceedings of the 8th International Conference on Intelligent Systems Design and Applications (ISDA '08)*, vol. 3, pp. 228–232, November 2008.

[14] T. D. Braun, H. J. Siegel, N. Beck et al., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[15] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *Proceedings of the 25th IEEE International Real-Time Systems Symposium (RTSS '04)*, pp. 37–46, December 2004.

[16] S. Baruah, "Task partitioning upon heterogeneous multiprocessor platforms," in *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '04)*, pp. 536–543, Toronto, Canada, May 2004.

[17] S. K. Baruah, "Partitioning real-time tasks among heterogeneous multiprocessors," in *Proceedings International Conference on Parallel Processing (ICPP '04)*, pp. 467–474, August 2004.

[18] H. Aydin and Q. Yang, "Energy-aware partitioning for multiprocessor real-time systems," in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2003.

[19] D. Zhu, N. AbouGhazaleh, D. Mosse, and R. Melhem, "Power aware scheduling for and/or graphs in multi-processor real-time systems," in *Proceedings of the International Conference on Parallel Processing*, 2002.

[20] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, pp. 69–73, IEEE Press, Piscataway, NJ, USA, 1998.

[21] R. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization," in *Proceedings of the 7th Annual Conference on Evolutionary Programming*, pp. 611–618, Springer, Berlin, Geramanhy, 1998.

[22] W. Zhang, H. He, and J. Ye, "A two-level cache for distributed information retrieval in search engines," *The Scientific World Journal*, vol. 2013, Article ID 596724, 6 pages, 2013.

[23] W. Zhang, H. He, G. Chen, and J. Sun, "Multiple virtual machines resource scheduling for cloud computing," *Applied Mathematics and Information Sciences*, vol. 7, no. 5, pp. 2089–2096, 2013.