Journal of Computers Vol. 30 No. 6, 2019, pp. 242-251
doi:10.3966/199115992019123006020

A Heterogeneous Multiprocessor Independent Task Scheduling Algorithm Based on Improved PSO

Xiaohui Cheng[1*], Fei Dai[1]

[1] Guangxi Key Laboratory of Embedded Technology and Intelligent System & College of Information Science and Engineering, Guilin University of Technology, Jiangan Road No. 12, Guilin 541000, China
{cxiaohui, 102016453}@glut.edu.cn

Received 10 May 2019; Revised 10 June 2019; Accepted 2 July 2019

Abstract. The independent task scheduling problem of heterogeneous multi-processors belongs to the NP-hard problem. The emergence of evolutionary algorithms provides a new idea for solving this problem. Particle swarm optimization (PSO) is a kind of intelligent evolutionary algorithm and it could be used to solve scheduling problem. We firstly discretized the representation of particle swarm optimization algorithm and made it suitable for the scheduling problem of heterogeneous multiprocessors. Then, the PSO algorithm was introduced into heterogeneous multiprocessors independent task scheduling problem by modeling method. In order to overcome particle swarm optimization algorithm's problem that is easy to fall into local optimum and premature convergence. We proposed a heterogeneous multiprocessor independent task scheduling algorithm based on improved PSO by improving the update operation of particle swarm optimization algorithm and transformed it into crossover and mutation operation of genetic algorithm. The experimental results show that the improved PSO scheduling algorithm can overcome the premature defects of PSO algorithm and the makespan of proposed IPSO is smaller than PSO.

Keywords: heterogeneous multiprocessors, independent tasks, particle swarm optimization, task scheduling

1 Introduction

Today, multiprocessor task scheduling challenges scholars due to the problem of efficiently assigning a great deal of tasks in very short execution time, usually limited to the order of few minutes, or even seconds [1]. Indeed, a good task scheduling algorithm will benefit a lot for improving the performance of multiprocessors system. Therefore, efficiently assigning and mapping tasks to various processors become a critical issue. Traditional homogeneous task scheduling problem is one of the NP-hard problems [2], the heterogeneity makes it even harder. As a result, it is necessary for researchers to solve this problem.

The problem of task scheduling could be transformed into the problem of mapping tasks to different processors. The problem of mapping the tasks to multiple processors can be differentiated in terms of static mapping, dynamic mapping, independent task set, dependent task set, flow-shop task set, homogeneous processors, heterogeneous processors, or various qualitative parameters used for performance measures [3]. Static mapping, independent task set, and heterogeneous processor are the main research aspects of this paper. On the basis of it, we built a heterogeneous multiprocessor independent task scheduling model and implemented our algorithm on this scheduling model.

In this paper, we focus on the independent task scheduling of the heterogeneous multiprocessors and improved a particle swarm optimization (PSO) algorithm by using genetic algorithm (GA) ideas. The experiment results show our improved algorithm outperform PSO scheduling algorithm.

The rest of the manuscript is organized as follows. The next section presents the definitions of independent task scheduling model, the assumptions and objectives. The main related work of

* Corresponding Author

independent task scheduling for heterogeneous multiprocessors are reviewed in Section 3. The details about our improved PSO algorithm are described in Section 4. The experiments and analysis are presented in Section 5. Finally, section 6 introduces the main conclusions of our research and discusses the future work.

2 Heterogeneous Multiprocessor Independent Task Scheduling Model

This section introduces system model for heterogeneous multiprocessor independent task scheduling and its objectives.

2.1 System Model

Heterogeneous multiprocessor system consists of a set of m heterogeneous processors having different processing elements. A task will get the resources only from the processor allocated to it. There is a set $T(T_1, T_2, ..., T_n)$ of n independent and simultaneously available tasks and a set $P(P_1, P_2, ..., P_m)$ of m various processors. $C(C_1, C_2, \ldots, C_m)$ represents processing time of task $T_i(i = 1, 2, \ldots, n)$ (Fig. 1).

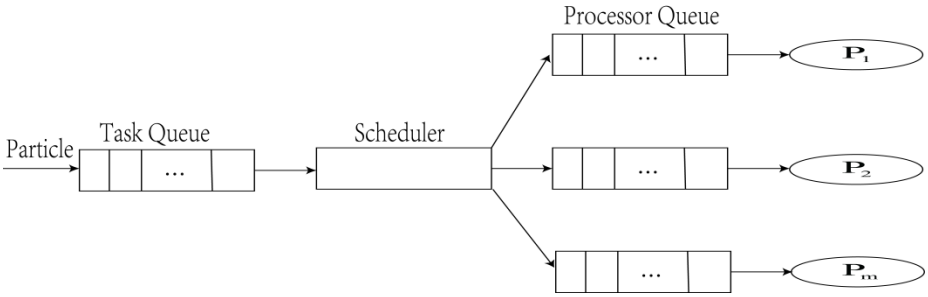

Fig. 1. Particle swarm optimization scheduling model under heterogeneous multiprocessor

2.2 Assumptions

Following assumptions are made while modeling the problem:

Processors are heterogeneous (each with different computing speed) and endlessly available from start-to-end time.

(1) Each processor can only process one task at a time.

(2) All tasks are mutually independent and they do not have executed sequence and cannot be preempted.

(3) All set-up times are included in the execution time and are independent from the logical order of tasks.

2.3 Objectives

There are various qualitative parameters available in order to judge the performance of scheduling algorithms. Here, maximum span among all processors is taken to evaluate the performance of scheduling algorithm. Span for processor i named as Spani could be defined as:

$$Span_{i\in\{1, 2, 3, ..., m\}} = \sum_{j=1}^{n} C_i M_{ij} \quad \textbf{(1)}$$

where $M_{ij} = 1$ if task T_j is assigned to processor P_i, otherwise $M_{ij} = 0$.

The makespan is stated as the maximum completion time for all processors. Maximu $span$ m span can be defined as:

$$MakeSpan = Maximum[Span_{i\in\{1, 2, 3, ..., n\}}] \quad \textbf{(2)}$$

The objective of our problem is that it maps the task set T to the set of processors P while minimizing the makespan having considered the constraints.

3 Related Works

In heterogeneous multi-core scheduling problems, it can be divided into two types: independent tasks and dependent task scheduling. At present, there are many research results for task-dependent scheduling [4-13], but the number of research results in independent task scheduling is relatively small. As a result, this domain requires further research and relevant research results should be combined with practice. In the field of independent task scheduling, some of the research results in recent years are as follows: Gogos et al. [14] proposed a heterogeneous multiprocessor independent task algorithm that uses heuristic and column pricing to achieve shorter scheduling lengths than other scheduling algorithms. Braun et al. [15] compared the effects of seven static heuristics in heterogeneous environment independent task scheduling. The experimental results showed that the genetic algorithm performs optimally. Santiago Iturriaga et al. [1] implemented a parallel random search scheduling method for independent task scheduling in CPU/GPU heterogeneous computing systems. Experimental results showed that the method has short execution time and high solution quality. Dorronsoro and Pineled [16] innovatively combined genetic algorithms with machine learning algorithms and introduced them into independent task scheduling problems to solve this problem. Compared with the other two heuristic algorithms, introducing machine learning into genetic algorithm could improve accuracy of scheduling greatly. Zhou et al. [17] proposed a minimum and earliest completion time algorithm for the independent task of heterogeneous environment. The algorithm introduced Min-min algorithm to solve k tasks with earliest completion time, and first dealt with the high cost scheduling. Compared with the Min-min algorithm, the proposed algorithm is better. At present, some scholars have applied the PSO algorithm to the independent task scheduling problem in multi-core processors and proposed corresponding methods from various aspects [3, 18-22].

In summary, the research on independent task scheduling has yet to be deepened, especially the independent task scheduling problem of heterogeneous multi-cores processor. According to the independent task scheduling problem of heterogeneous multiprocessors, firstly, the heterogeneous multiprocessor independent task scheduling model is constructed. Secondly, referring to the characteristics of the particle swarm optimization algorithm, we introduce the crossover and variation of genetic algorithms in the update operation of PSO algorithm. Then, an improved particle swarm heterogeneous multiprocessor independent task scheduling algorithm based on this model is proposed. We hope that it will bring new inspiration and help to industry and scientific research by solving such problems.

4 PSO Algorithm

Particle swarm optimization is inspired by interactions involved in the collective social behavior of animals. For example, a group of migratory birds maintain a certain formation flying between each other. The PSO algorithm focuses on the motion of a group of examples to cover the solution space to find different possible solutions for approximate optimal solutions.

Assuming that search space of the problem is an n-dimensional space, the position and velocity vectors of the first particle can be expressed as follows:

$$X_i = [x_{i1}, x_{i2}, ..., x_{in}]$$
$$V_i = [v_{i1}, v_{i2}, ..., v_{in}]$$

During each iteration, the particle continuously adjusts its position and updates it by tracking the position and velocity extremum. The optimal solution found by the first particle itself is marked as the individual optimal solution $pbest_i = [pbest_{i1}, pbest_{i2},\cdots, pbest_{in}]$, and the other is the optimal solution found by the whole population at present, which is called the global optimal solution, denoted as $gbest_i = [gbest_{i1}, gbest_{i2}, \ldots, gbest_{in}]$. In addition, the optimal solution of all the neighbors of the particle is the local optimal solution.

In the original particle swarm optimization algorithm, the particle position and velocity variation formula [5] are as follows:

$$V_{is}(t+1) = V_{is}(t) + c_1 r_1(t)[p_{is}(t) - X_{is}(t) + c_2 r_2(t)[p_{gs}(t) - x_{is}(t)] \quad \textbf{(3)}$$

$$X_{is}(t+1) = X_{is}(t) + V_{is}(t+1) \quad \textbf{(4)}$$

In above equations, i = [1, m], s = [1, S]. c1, c2 are a non-negative integer and are called learning factors. r1, r2 are independent random number between [0, 1]. The V_{max} is the maximum speed of particle, and it is constant and set by users. The m represents the population size and the t represents iteration number.

5 Based on Improved PSO Scheduling Algorithm

The improved particle swarm optimization algorithm proposed in this paper defines each particle as the potential solution of the problem, that is, after initializing the generated particle, each particle obtains a scheduling length and a scheduling sequence.

The innovation of the improved particle swarm optimization scheduling algorithm proposed in this paper is that it mainly transforms the update operation of the particle swarm optimization algorithm and introduces the crossover and mutation operations of the genetic algorithm into the particle swarm optimization algorithm. In the update operation of the algorithm, the main function of the crossover operation is that it exchanges the same type but different sizes tasks of the two processors, and then calculates their scheduling sequence and the scheduling length. The function of the mutation operation is that it chooses two processors in the processor list randomly, one processor reduces a task and the other processor adds the task that former processor's lost, and then calculates the scheduling length and running time of the two processors.

Pseudo code of update algorithm (the core of this algorithm) can be seen as follows:

Table 1. Update algorithm

Input: parameters for scheduling algorithm Output: Makespan, LocalBestParticle
1. for i=1 to a1
2. Crossoveroperator (Schedule,LocalBestSchedule);
3. end for
4. for j=1 to a2
5. CrossOverOperator (Schedule,GlobalBestSchedule);
6. end for
7. for k=1 to b
8. MutationOperator();
9. end for
10. CalculateMakespan
11. SetLocalBestParticle

Table 2. CrossOverOperator algorithm

Input: Schedule, LocalBestSchedule/GlobeBestSchedule Output. New Schedule
1. Choose a randomly a processor ID random_processor_id;
2. Get the processor to move from the chosen processor of best particle;
3. Get a random task from the shortlisted tasks
4. Look for this task in processor_schedule. That is, check which processor currently has the task in processor_schedule
5. Get the task to lose from processor_map
6. Get a random task from the shortlisted tasks
7. Swap the tasks between the processors in processor_schedule

Table 3. MutationOperater algorithm

Input: Output:
1. Get two random processors, random_p1_id, random_p2_id 2. Compare run time to select which processor will gain/lose a task 3. Select which process will be moved from losing p to gaining p 4. One processor loses a task from its task_list and the other get this task and add it in its task_list

The flow chart of the improved algorithm is as follows:

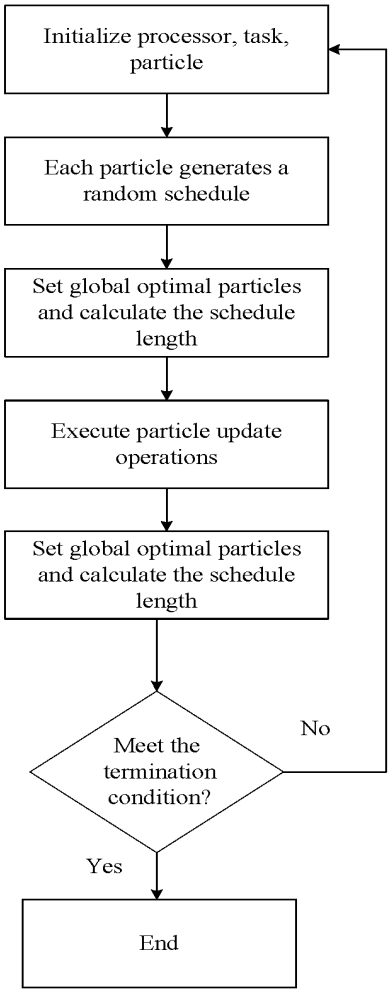

Fig. 1. Algorithm flowchart

Algorithm detailed calculation process is as follows:

Step 1. m specified different processing frequencies heterogeneous processor and n tasks to be processed are Initialized (n>m). Also, p particles are randomly generated.

Step 2. After generating p particles, each particle will generate a random scheduling. Firstly, the task is assigned to the processor randomly and equally, the random scheduling sequence is obtained, the scheduling length of the particle is calculated, and the local optimal scheduling is set. When each particle calculates its scheduling length and obtains their scheduling sequence, the global optimal scheduling length and its scheduling sequence are obtained.

Step 3. Particle update operation is performed. Here is the core of the algorithm. Firstly, the scheduling sequence and the local scheduling sequence are run a1 times crossover operation. After the crossover operation, the optimal local scheduling sequence can be obtained between the particles. Then a2 times cross operation is performed on the scheduling sequence and the globally optimal scheduling sequence. After the cross operation is completed, the scheduling sequence obtains the global optimal scheduling

sequence. Finally, b times mutation operation is carried out.
Step 4. After the update operation, the scheduling length is calculated, the global optimal particle is set, and the scheduling length is outputted.
Step 5. Judge whether the stop condition is satisfied or not, otherwise repeat the first step.

6 Experiments and Analysis

In this section, we will demonstrate the experiment in detail. The experiment environment and algorithm parameters will be shown on Section 6.1, the experiment result and analysis of the experiment result will be depicted on Section 6.2.

6.1 Environment of the Experiments

The experimental environment in this paper is shown in Table 4.

Table 4. Experimental environment setup

CPU	AMD Athlon (tm) II X4 645 Processor 2.3GHz
Memory	4G
Operating system	Windows 7
Development platform	Visual Studio 2013
Program language	C++

In this experimental environment, the algorithm parameters are set as follows: cognitive factor a1 = 5, social factor a2 = 5, inertial factor b = 10, particle number =20, iteration number = 1000. The processing speed of processor is randomly generated from 1000 to 300000 instructions per millisecond. Communication rate between processors is randomly set from 10 to 1000 bytes per millisecond. Average instruction of task is set from 100 to 100000000 instructions randomly. Data amount of task is generated from 0 to 100000 bytes randomly.

6.2 Experimental Result

In order to verify the effectiveness of the algorithm, this paper uses two groups of experiments to test the performance of the algorithm: The first group is set to execute a fixed number of tasks but under the system environment of different number of cores. The second group, as control group, is set to execute different number of tasks and under the system environment of a fixed number of cores. The first group of experiments includes experiments 1 to 3, and the experiment setup is as follows: the number of fixed tasks is 1000, and the experiments are carried out under the environment of 10, 15 and 20 heterogeneous processor cores respectively. In contrast, the second group of experiments includes experiments 4 to 6, the experimental setup is as follows: the fixed number of heterogeneous processor cores is 10, and the performance of the algorithm is tested under the task number of 350, 600, 850. The following experimental results are the average of the results obtained after the algorithm is run 20 times.

In above table, the experiment 1 shows every 50 iterations' makespan of PSO and the improved PSO scheduling algorithm in the system environment of processor core number of 10, task number of 1000. The experiment 2 displays every 50 iterations' makespan of PSO and the improved PSO scheduling algorithm in the system environment of processor core number of 15, task number of 1000. The experiment 3 demonstrates every 50 iterations' makespan of PSO and the improved PSO scheduling algorithm in the system environment of processor core number of 20, task number of 1000.
The makespan of the first group of experiments after 1000 iterations is shown in Table 5.

Table 5. First group of experiments

Iteration	Experiment 1		Experiment 2		Experiment 3	
	PSO Makespan	IPSO Makespan	PSO Makespan	IPSO Makespan	PSO Makespan	IPSO Makespan
0	1599.269	1601.006	1768.758	1767.658	1367.156	1368.866
1	1518.013	1593.036	1645.268	1611.598	1246.121	1272.121
50	308.924	284.924	225.033	210.033	190.003	167.003
100	299.409	282.409	223.476	208.446	188.852	165.852
150	293.463	280.463	217.459	207.459	188.344	165.344
200	292.646	279.083	216.198	206.691	172.760	164.660
250	292.036	278.636	216.836	205.836	172.722	164.022
300	292.036	278.094	216.386	205.386	171.693	163.693
350	291.831	277.831	216.002	205.002	171.612	163.512
400	291.790	277.790	215.774	204.774	171.612	163.404
450	291.790	277.790	215.588	204.588	171.612	163.155
500	291.790	277.790	215.411	204.411	170.227	163.138
550	291.747	277.747	215.353	204.353	170.227	163.102
600	291.663	277.663	215.310	204.310	170.227	163.027
650	291.663	277.661	214.308	204.308	170.217	163.027
700	290.663	277.634	214.292	204.292	170.185	163.008
750	290.663	277.633	214.260	204.258	170.166	163.007
800	290.663	277.609	214.260	204.258	170.125	163.006
850	290.646	277.567	214.260	204.197	170.106	163.001
900	290.646	277.466	214.260	204.190	170.106	162.864
950	290.646	277.466	214.260	204.160	170.106	162.853
1000	290.646	277.466	214.198	204.099	170.106	162.423

In contrast, we set up a second group to test the performance of our proposed algorithm. Table 6 depicts the experiment results of second group.

Table 6. Second group of experiments

Iteration	Experiment 4		Experiment 5		Experiment 6	
	PSO Makespan	IPSO Makespan	PSO Makespan	IPSO Makespan	PSO Makespan	IPSO Makespan
0	529.489	528.723	985.158	987.559	1426.258	1423.381
1	418.692	414.790	863.159	869.120	1335.365	1310.822
50	138.576	101.024	200.568	173.732	302.924	242.301
100	112.684	99.959	183.526	171.944	259.409	239.831
150	112.684	99.794	183.326	171.331	250.463	238.073
200	110.583	99.721	185.326	170.902	257.083	237.369
250	111.583	99.681	183.232	170.774	257.036	236.953
300	111.583	99.608	183.232	170.701	257.036	236.705
350	111.583	99.467	183.232	170.628	256.831	236.691
400	110.883	99.374	182.126	170.588	256.790	236.675
450	110.883	99.374	182.126	170.520	256.790	236.597
500	110.883	99.311	181.626	170.446	256.790	236.532
550	110.883	99.299	181.626	170.421	255.747	236.408
600	110.883	99.274	181.428	170.406	255.663	236.339
650	110.883	99.263	181.426	170.366	255.663	236.334
700	110.883	99.263	181.126	170.223	255.663	236.294
750	110.879	99.258	180.126	170.212	255.649	236.279
800	110.879	99.258	180.126	170.210	255.649	236.237
850	110.579	99.258	180.126	170.179	255.646	236.237
900	110.579	99.248	180.126	170.122	255.646	236.237
950	110.179	99.248	180.326	170.110	255.646	236.191
1000	110.179	99.238	182.326	170.104	255.646	236.107

From Table 6, Experiment 4 reveals comparison of experimental results between PSO and improved PSO scheduling algorithm in the system environment of heterogeneous processor core number of 10 and a task number of 350. Experiment 5 exhibits comparison of experimental results between PSO and improved PSO scheduling algorithm in the system environment of heterogeneous processor core number of 10 and a task number of 600. Experiment 6 displays comparison of experimental results between PSO and improved PSO scheduling algorithm in the system environment of heterogeneous processor core number of 10 and a task number of 850.

The makespan of the second group of experiments after 1000 iterations is shown in the following figure. From the experimental results of Fig. 3 and Fig. 4, it can be seen that the proposed IPSO heterogeneous multiprocessor independent task scheduling algorithm can obtain shorter scheduling length than the PSO scheduling algorithm after 1000 iterations, regardless of whether in the system condition of various number of processors and fixed number of tasks or different number of tasks and fixed number of processors.

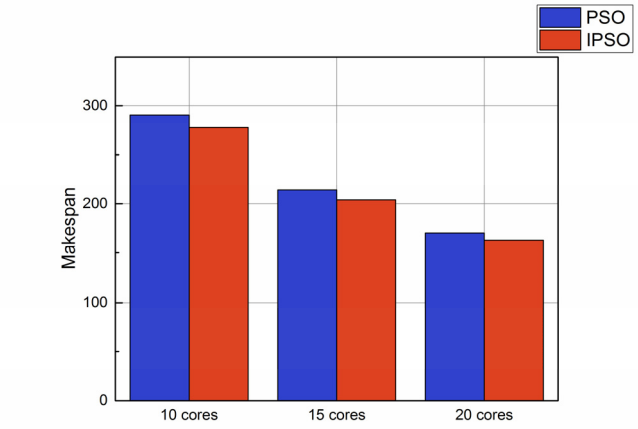

Fig. 2. Two algorithms' makespan Diagram in 10, 15, 20 processors after 1000 iterations

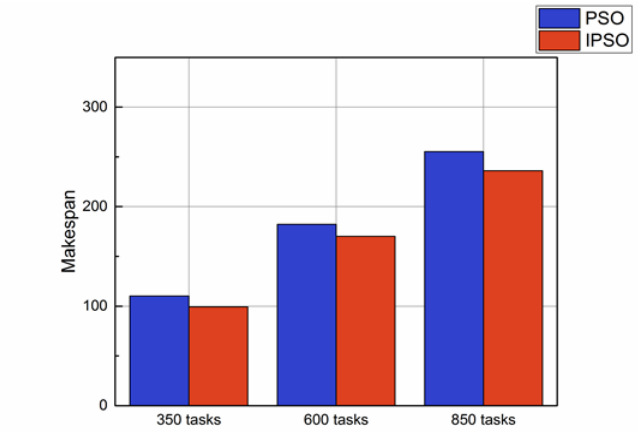

Fig. 3. Two algorithms' makespan diagram in 350, 600, 850 tasks after 1000 iterations

It can be seen from Table 5 that both IPSO and PSO algorithms' makespan are getting smaller and smaller after each iteration under the conditions of different numbers of processors and 1000 tasks, but IPSO on average obtains lower scheduling length. From Table 6, we can see that both IPSO and PSO algorithm's makespan decrease as the iteration goes, but overall IPSO can obtain lower makespan than PSO algorithm under the condition of various tasks' number and 10 processors. Considering the experimental results of above experimental tables and images, we can draw the conclusion that IPSO algorithm has lower scheduling length and faster convergence speed.

7 Conclusions

In this paper, a heterogeneous multi-processors independent scheduling model is established, and the assumptions and algorithm objectives are given. On above basis, a heterogeneous multi-processors independent task scheduling algorithm based on improved PSO is proposed. The experimental results show that the proposed improved PSO heterogeneous multi-processors independent task scheduling algorithm is better than the PSO algorithm under the scheduling model, which can jump out of the local optimal solution faster and avoid premature occurrence. In the next work, we plan to introduce methods such as machine learning to optimize the algorithm further.

Acknowledgments

As the research of the thesis is sponsored by National Natural Science Foundation of China (No: 61662017, No: 61262075), Key R & D projects of Guangxi Science and Technology Program (AB17195042), Guangxi Science and Technology Development Special Science and Technology Major Project (No: AA18118009), Guangxi Key Laboratory Fund of Embedded Technology and Intelligent System, we would like to extend our sincere gratitude to them.

References

[1] S. Iturriaga, S. Nesmachnow, F. Luna, E. Alba, A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems, The Journal of Supercomputing, 71(2)(2015) 648-672.

[2] S.K. Sahni, Algorithms for scheduling independent tasks, Journal of the ACM 23(1)(1976) 116-127.

[3] S. Shriya, R.S. Sharma, S. Sumit, S. Choudhary, Directed search-based PSO algorithm and its application to scheduling independent task in multiprocessor environment, in: Proc. the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA), 2016.

[4] J. Yi, Q. Zhuge, J. Hu, S. Gu, M. Qin, E.H.M. Sha, Reliability-guaranteed task assignment and scheduling for heterogeneous multiprocessors considering timing constraint, Journal of Signal Processing Systems 81(3)(2015) 359-375.

[5] N. Kumar, D.P. Vidyarthi, A novel hybrid PSO–GA meta-heuristic for scheduling of DAG with communication on multiprocessor systems, Engineering with Computers 32(1)(2016) 35-47.

[6] Y.M. Xu, K.L. Li, J.T. Hu, K.Q. Li, A genetic algorithm for task scheduling on heterogeneous computing systems using multiple priority queues, Information Sciences 270(6)(2014) 255-287.

[7] R. Ayari, I. Hafnaoui, G. Beltrame, G. Nicolescu, ImGA: an improved genetic algorithm for partitioned scheduling on heterogeneous multi-core systems, Design Automation for Embedded Systems 22(1-2)(2018) 183-197.

[8] Y. Jiang, Z. Shao, Y. Guo, H. Zhang, K. Niu, DRSCRO: a metaheuristic algorithm for task scheduling on heterogeneous systems, Mathematical Problems in Engineering 2015(2015) Article ID 396582.

[9] K. Prescilla, A.I. Selvakumar, Modified binary particle swarm optimization algorithm application to real-time task assignment in heterogeneous multiprocessor. Microprocessors and Microsystems 37(6-7) (2013) 583-589.

[10] G. Xie, G. Zeng, L. Liu, R. Li, K. Li, Mixed real-time scheduling of multiple DAGs-based applications on heterogeneous multi-core processors, Microprocessors and Microsystems 47(2016) 93-103.

[11] C. Xu, T. Li, Chemical reaction optimization for task mapping in heterogeneous embedded multiprocessor systems, Advanced Materials Research, Trans Tech Publications 712(2013) 2604-2610.

[12] Y. Xu, K. Li, L. He, T.K. Truong, A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization, Journal of Parallel and Distributed Computing 73(9)(2013) 1306-1322.

[13] K. Rzadca, F. Seredynski, Heterogeneous multiprocessor scheduling with differential evolution, in: Proc. 2005 IEEE Congress on Evolutionary Computation, 2005.

[14] C. Gogos, C. Valouxis, P. Alefragis, G. Goulas, N. Voros, E. Housos, Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing, Future Generation Computer Systems 60(2016) 48-66.

[15] T.D. Brauna, H.J. Siegel, N. Beck, L.L. Bölönid, M. Maheswarane, A.I. Reuther, J.P. Robertson, M. D. Theys, B. Yao, D. Hensgen, R.F. Freundk, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed computing 61(6)(2001) 810-837.

[16] B. Dorronsoro, F. Pinel, Combining machine learning and genetic algorithms to solve the independent tasks scheduling problem, in: Proc. 2017 3rd IEEE International Conference on Cybernetics (CYBCONF), 2017.

[17] Y. Zhou, C. Jiang, Y. Fang, Research on independent task scheduling algorithm in heterogeneous environment, Computer Science 35(8)(2008) 90-92+97.

[18] A. Omidi, A. M. Rahmani, Multiprocessor independent tasks scheduling using a novel heuristic PSO algorithm, in: Proc. IEEE International Conference on Computer Science and Information Technology, 2009.

[19] W. Zhang, H. Xie, B. Cao, A.M.K. Cheng, Energy-aware real-time task scheduling for heterogeneous multiprocessors with particle swarm optimization algorithm, Mathematical Problems in Engineering, (2014) 1-9.

[20] S. Sarathambekai, K. Umamaheswari, Intelligent discrete particle swarm optimization for multiprocessor task scheduling problem, Journal of Algorithms & Computational Technology 11(1)(2016) 58-67.

[21] J. Chen, Q. Pan, Improved particle swarm optimization algorithm for solving independent task scheduling problem, Microelectronics & Computers 26(1)(2009) 214-215.

[22] Y. L. Wang, N. Wang, C. H. Yang, W. H. GUI, A discrete particle swarm optimization algorithm for task assignment problem, Journal of Central South University Science and Technology 39(3)(2008) 570-576.