2007 IEEE International Conference on Control and Automation
Guangzhou, CHINA - May 30 to June 1, 2007

A Novel Task Scheduling Algorithm for Real-Time Multiprocessor Systems

Yang-ping Chen , Lai-xiong Wang, and Shi-tan Huang
Xi'an Microelectronic Technology Institute
Xi'an, China
chenyangping2005@yahoo.com.cn

***Abstract*—The task scheduling in real-time multiprocessor systems is to map tasks onto processors and order their execution so that the precedence relationships between tasks are maintained and the minimum schedule length is obtained. This is a well-known NP-completed problem. And many heuristic methods have existed, but their performance still needs to be improved. Recently, particle swarm optimization has received much attention as a class of robust stochastic search algorithm for various optimization problems. This paper presents a novel task scheduling algorithm for real-time multiprocessor systems, which takes task's height and particle's position as the task's priority values, and applies the list scheduling strategy to generate the feasible solutions. Simulation results demonstrate that the proposed algorithm, compared with genetic algorithm, produces encouraging results in terms of quality of solution and time complexity.**

Keywords-particle swarm optimization; real-time multiprocessor systems; task scheduling; list scheduling technology

I. INTRODUCTION

The task scheduling in real-time multiprocessor systems is a key factor for a parallel computing system to gain better performance. This is a well-known NP-complete combinatorial problem in general, except in a few simplified situations[1]. Consequently, amounts of heuristic-based algorithms have been presented in the literatures to obtain optimal or near-optimal solutions thus far. The most studied heuristics are based on list scheduling technique or genetic algorithm. The list scheduling technique has a lower complexity with respect to both the time and the space, but it is difficult to guarantee the quality of the solution. The genetic algorithm outperform the list scheduling technique in that it has a wide search space and can obtain a good quality solution. However, its convergence dependences mainly upon both the representation of the chromosome and the parameters selected. In this paper, we present a particle swarm optimization approach to the multiprocessor task scheduling problem.

Particle Swarm Optimization(PSO) [2]is a novel Swarm Intelligence method that models social behavior to guide swarm of particles towards the most promising regions of the search space. PSO has proved to be efficient at solving Unconstrained Global Optimization and engineering problems[3-9]. It is easily implemented, using either binary or floating point encoding, and it usually results in faster convergence rates than the genetic algorithm[10]. Unlike the methods adopted to solve the combinatorial optimization problems [6-7], the presented algorithm ,which is called LPSO(List-based PSO), does not redefine the iterative formulations in PSO. Instead, it takes task's height and particle's position as the task's priority values, and directly applies the original operates in PSO to generate the diverse priority value ,and utilizes the list scheduling strategy to obtain the feasible solutions. LPSO not only maintains the operateability of PSO, but also guarantees the diversity of the schedule. Simulation results demonstrate that LPSO, compared with the genetic algorithms, especially fits to solve the multiprocessor task scheduling problem with a number of tasks and processors.

The remainder of the paper is organized as follows. In section II, the multiprocessor task scheduling problem to be solved is described. The related works is given in Section III. And in section IV, the proposed algorithm is described in detail followed by the experimental results in Section V. Finally, we conclude the whole paper in Section VI.

II. PROBLEM DESCRIPTION

A parallel program is presented by a directed acyclic graph(DAG) $TG = <V_t, E_t, w, c>$, called *a task graph* or *a program graph.* The vertices set $V_t = \{t_1, t_2, \ldots, t_m\}$represents the collection of m tasks, and the edges set E_t represents the precedence relationships among tasks. Two tasks t_i and t_j connected by an edge mean that t_i must be completed before t_j can be initiated. In this case, t_i is said to be an immediate predecessor of t_j, and t_j an immediate successor of t_i. For each task $t_i \in V_t$, a weight $w(t_i)$ is associated with it to represent the execution time of the task t_i. For each edge$(t_i,t_j) \in E_t$, a weight $c(t_i,t_j)$ is given to represent the amount of data transferred between task t_i and t_j. Fig.1(a) shows a DAG with 10 tasks.

The two height functions $hp(t_i)$ and $hs(t_i)$ of task t_i in the task graph ,which represent the precedence relationship between tasks, are defined as follws [11-12]:

$$hp(t_i) = \begin{cases} 0 & \text{if } pred(t_i) = \phi \\ 1 + \max\limits_{t_j \in pred(t_i)} \{hp(t_j)\} & \text{otherwise} \end{cases} \quad (1)$$

$$hs(t_i) = \begin{cases} \max\limits_{t_j \in V_t} \{hp(t_j)\} & \text{if } succ(t_i) = \phi \\ -1 + \min\limits_{t_j \in succ(t_i)} \{hs(t_j)\} & \text{otherwise} \end{cases} \quad (2)$$

where $pred(t_i)$ is a set of the immediate predecessors of t_i, $succ(t_i)$ is a set of immediate successors of t_i.

A real-time multiprocessor systems is represented by an undirected graph $PG=(V_p,E_p, d)$ called *a processor graph* or *a system graph.* The vertices set $V_p = \{p_1,p_2,\ldots,p_n\}$ represents the collection of n parallel identical processors , and the edge set E_p represents channels among processors. For each pair of processors $p_k,p_l \in V_p$, $k \neq l$, a distance $d(p_k,p_l)$ is associated to present the length of the shortest path between p_k and p_l . If two

1-4244-0818-0/07/$20.00 © 2007 IEEE

tasks t_i and t_j are assigned to different processors p_k and p_l, respectively, the time required for task ti to communicate with tj is estimated to be $c(t_i,t_j)*d(p_k,p_l)$. The communication time between two tasks within the same processor is assumed to be zero. We assume that each processor can execute at most one task at a time and task preemptive and duplication are not allowed. Fig.1(b) shows a multiprocessor systems with 3 fully connected processors.

In essence, a schedule is a mapping that maps each task in the task graph to a appropriate processor in the processor graph and to determine starting time .Let F denotes the scheduling function, i.e., $F=\{ V_t\text{->}V_p \times [0,\infty]\}$, where V_t,V_p are stated as before and $[0,\infty]$ is the start time when the task executes. The objective of the task scheduling in real-time multiprocessor systems is to find a schedule f in F in such a way that task-dependence constraints are satisfied and the schedule length (SL) , i.e., the maximum finish time of all tasks in the task graph, is minimized. It can be formulated as follows:

$$\begin{aligned} &\min \quad SL(f) \\ &s.t. \quad SL(f) = \max\{ length\ (p_k)_f \},\ k = 1,2,\ldots,n \\ &\qquad f \in F \end{aligned} \tag{3}$$

where n represents the number of processors in the system graph; $length(p_k)_f$ represents the finishing time for the last task in processor p_k under schedule f.

III. Related work

Scheduling of precedence constrained task graphs in the homogenous parallel real-time multiprocessor systems is a typical NP-complete problem[1]. A number of heuristic methods have been proposed to sacrifice optimality for the sake of efficiency. The most studied heuristics are based on list scheduling technique or genetic algorithm.

The basic idea in the *list scheduling technique* is first to arrange tasks in a sequence, satisfying precedence constraints, generally based on some priority. The highest priority ready task (a task with satisfied precedence constraints) is then chosen for scheduling followed by the selection of most suitable processor, to accommodate it.

There are , however, numerous variations in the methods of assigning the priority value and maintaining the ready list, and criteria for selecting a processor to accommodate a node . Two major attributes for assigning priorities are the t-level(top level) and b-level(bottom level) [13]. The t-level of a task ti is the length of the longest path from an entry task to ti in the DAG(excluding t_i)and the b-level from the task t_i to an exit task. In assigning a task to a processor, most scheduling algorithms attempt to minimize the start-time of a task[14]. However, some algorithms do not necessarily minimize the start-time of a node but consider other factors as well[15].

The genetic *algorithm* (GA)is a search algorithm which is inspired by the principle of evolution an natural genetic. It begins with an initial population (a set of chromosomes) and then operates through a simple cycle of stages: evaluation of the chromosomes, and reproduction to create a new population, using crossover and mutation operators. This process is repeated and terminates after a special number of generations or when a fair time is executed.

GA has been applied to the task scheduling problem in a number of ways. The two main approaches are methods that

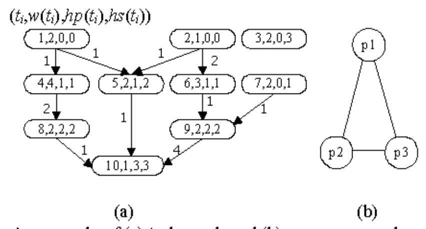


Fig. 1. An example of (a) task graph and (b) processor graph

use a GA to evolve the actual assignment and order of tasks into processor [11-12] [16-17] and methods that use a GA in combination with other list scheduling techniques[18-20].

With respect to PSO for the scheduling problem , it is mainly used to solve the job shop scheduling problem[21-22]and the blending scheduling[23]. In this paper, we attempt to apply PSO to the task scheduling problem in real-time multiprocessor systems.

IV. LPSO Algorithm

A. Particle

PSO was originally introduced by Kennedy and Eberhart in 1995 as an alternative to the standard genetic algorithm[2].It was developed based on the hypothesis that members of a population called *swarm* can profit from their past experiences and the experiences of other individuals called *particles*. Each particle represents a candidate solution to a optimization problem at hand and is composed of three vectors: X, P and V. The X-vector represents the current position and contains the current potential solution. The P-vector contains the location of the best potential solution discovered by a particle. The V-vector, known as the velocity vector, represents the distance to be traveled by particle from its current position and is used to determine the next potential solution to be evaluated.

In order to apply PSO to the problem of multiprocessor task scheduling，we add two vectors, T and H ,to each particle. The T-vector represents all task at least once but once only. The H-vector is the height of each task t_i in the task graph, whose value is a random integer between $hp(t_i)$and $hs(t_i)$.Therefore, a particle in LPSO is composed of five vectors, X, P, V, T and H. The first three vectors are used to generate new priority value, and the last two and X are used to schedule the tasks onto the processors.

B. Fitness Function

The fitness function is essentially the objective function for the problem. It provides means of evaluation of particles in the swarm and it also controls the iteration process. Mapping an original objective function value to a fitness value is a feature of the evolution function. In the multiprocessor task scheduling problem, the objective function is to minimize the maximum of complete-time on all processors. Therefore, in LPSO, we use the schedule length as the fitness function of a candidate. A particle with the lowest fitness will be superior to other particles and should be reserved in the search process. Let $est(t_i)$ represents the earliest start time of task t_i, i.e., the maximum of time at which processors become available and the time at which the last message arrives from predecessor of task t_i. $eft(t_i)$ represents the earliest finish time, i.e. $w(t_i)+est(t_i)$. The schedule length(SL) is calculated by

```
Step1(sequencing)
●   Sort T in incremental order of H;
●   Sort T in incremental order of X when tasks
    in T have the same height;
Step2(scheduling)
●   Schedule each task t_i in T onto the
    processor which gives the lowest eft(t_i)
    when the precedence relationships and the
    communication delay between tasks are
    considered;
Step3(result)
●   Return the maximum finish time of the last
    task.
```

Fig. 2 Algorithm 1—Fitness Function

Algorithm 1 shown in Fig.2.

C. Overall procedure

In the past years, researchers have been explored several models of PSO algorithm. In this paper , we use the global model equations, which are described as follows[2] [24]:

$$V_{i,d}(t) = \omega \times V_{i,d}(t-1) + c_1 \times rand_1() \times (P_{i,d} - X_{i,d}(t-1)) + c_2 \times rand_2() \times (P_{g,d} - X_{i,d}(t-1)) \quad (4)$$

$$V_{i,d}(t) = rand_3() \qquad if \quad |V_{i,d}(t)| \geq V_{\max} \quad (5)$$

$$X_{i,d}(t) = X_{i,d}(t-1) + V_{i,d}(t) \quad (6)$$

$$\omega = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{genMax} t \quad (7)$$

$$P_i(t) = \begin{cases} P_i(t-1) & if \quad SL(X_i) \geq SL(pBest_i) \\ X_i(t) & otherwise \end{cases} \quad (8)$$

$$P_g(t) = \arg \min SL(pBest_i), \quad 1 \leq i \leq sizeP \quad (9)$$

where $V_{i,d}$ is the number d component of the velocity of particle i, $X_{i,d}$ is the number d component of the position of particle i, $P_{i,d}$ is the position of the local best particle called *pBest*$_i$, $P_{g,d}$ is the position of the global best particle called *gBest* in the swarm. ω is the inertia weight of velocity and it is used to regulates the trade-off between the global exploration and local exploitation ability of the swarm .V_{max} is the maximum velocity, which can avoid the absolute value of $V_{i,d}$ may be great in which the particle may overshoot the problem space. $\omega_{\max}$ and $\omega_{\min}$ are the initial and the final value of ω, respectively. *genMax* is the maximum number of iteration. c_1 and c_2 denote the acceleration coefficients that pull each particle toward the position of *pBest*$_i$ and *gBest*. *rand*$_1()$、*rand*$_2()$ and *rand*$_3()$ are three random functions with rang[0,1]. t is the current number of iteration and sizeP is the swarm size.

Equations (4) ~(6) are used to update the velocity and position of the particle. For (4), the first part represents the inertia of previous velocity; the second part is the "cognition" part, which represents individuals thinking independently, and the third part is the "social" part, which represents cooperation among the particles [25] . (5) means that $V_{i,d}$ will be reinitilized a random real value between 0.0 and 1.0, when its absolute value is greater than V_{max} . (7) implies that the inertia weight linearly decreases from a relative large value to a relatively small value over the course of operation. Therefore, LPSO tends to have more global search-ability at the beginning of the run, while having more local search-ability near the end of the run. (8) is used to update the local best position of each particle and (9) is used to find the global best position in the swarm.

```
Step1(initialization)
●   Read the task graph and the system graph;
●   Calculate the height functions and the length
    of the shortest path among processors;
●   Initialize swarm, including sizeP and
    particle's five vectors;
●   Calculate each paritcle's schedule length by
    Algorithm 1;
●   Initialize pBest as a copy of particle
    itself;
●   Initialize gBest as the particle with the
    lowest schedule length;
●   Give initial value: $\omega_{\max}$, $\omega_{\min}$, $V_{max}$, $C_1$, $C_2$ and
    genMax ;
Step2( iterative search )
FOR ( t =1 to genMax) DO
BEGIN
●   Update $\omega$ by (7);
●   Generate next swarm by (4)~(6)
●   Calculate each paritcle's scheduling length
    by Algorithm 1;
●   Update pBest by (8);
●   Update gBest by (9);
ENDFOR
Step3(result)
●   Output gBest particle and its schedule
    length.
```

Fig.3 Algorithm 2—LPSO

LPSO begins by reading the task graph and the system graph. After calculating the height functions of each task and the length of the shortest path among processors, LPSO randomly initializes the element of both the *X*-vector and *V*-vector to be random real number between 0.0 and 1.0. in the *m*-dimensional space, where *m* is the total number of task in the task graph. Initially, the *P*-vector is set equal to the *X*-vector, *T*-vector is a permutation of all tasks in the task graph and each component of the H-vector is the random integer between $hp(t_i)$ and $hs(t_i)$ where t_i corresponds to the element of the *T*-vector.

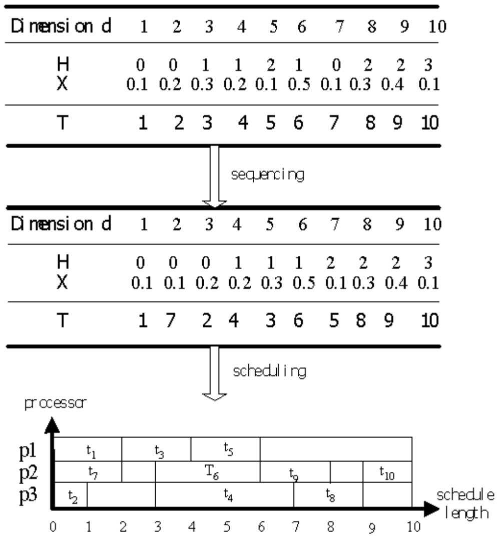
The overall procedure of LPSO for real-time multiprocessor task scheduling in this paper is shown in Fig. 3 .

Taking scheduling 10 tasks in Fig.1(a) onto 3 processors in Fig.1(b) as an example, the optimal schedule length is 10 unit times obtained by LPSO. The process of scheduling according to three vectors *T*、*H* and *X* is shown in Fig.4.

V. SIMULATION RESULTS

LPSO algorithm was implemented and tested on deterministic and random task graphs.The experiments were performed on PC(Pentium IV 2GHz CPU, 256M RAM, Windows 2000 OS, VC++6.0). LPSO used the following parameters throughout the simulation experiment: c_1=c_2=2 , $\omega_{\max}$=1.0 , $\omega_{\min}$=0.4, V_{max} =2, *genMax*=2000,and sizeP = 40. GA used the following parameters: crossover probability=0.9, mutation probability =0.05, maximum numbers of iterations=2000,populations size=40, roulette wheel selection. The performance metric is the schedule length and the execution time. The iteration terminates when the number of the same schedule length value is fifty .

In the first experiment, the deterministic task graph was taken from literature[12] which has 88 tasks and its optimal schedule is known . When the processors are fully connected, the results that LPSO compared with GA [11] is shown in Table I. Each result is an average of run 20 times. The value in



Dimension d	1	2	3	4	5	6	7	8	9	10
H	0	0	1	1	2	1	0	2	2	3
X	0.1	0.2	0.3	0.2	0.1	0.5	0.1	0.3	0.4	0.1
T	1	2	3	4	5	6	7	8	9	10

Dimension d	1	2	3	4	5	6	7	8	9	10
H	0	0	0	1	1	1	2	2	2	3
X	0.1	0.1	0.2	0.2	0.3	0.5	0.1	0.3	0.4	0.1
T	1	7	2	4	3	6	5	8	9	10

Fig.4 The Process of scheduling according to particle. This is one of optimal solution scheduling Fig.1(a) onto Fig.1(b)

OPT is taken from [11] as a reference. From the Table1, one can see that both LPSO and GA do not obtain the optimal schedule, and there is no significant different between the results obtained by LPSO and GA , but the solution obtained by LPSO is consistently better than GA except for 2 processors, which indicates that LPSO fits the multiprocessor task scheduling problem with a large number of processors. The last two rows indicate that the maximum of the parallel processors running the task graph is 8.

In the second experiment, the random task graphs are generated by the random graph generator in[14]in the following manner. Given m, the number of tasks in the DAG, we first randomly generated the height of the DAG from a uniform distribution with the mean roughly equal to $\sqrt{m}$. For each level, we generated a random number of tasks which were also selected from a uniform distribution with a mean roughly equal to $\sqrt{m}$. Then, we randomly connected the tasks from the higher level to the lower level. The sizes of the random DAGs is 1000 and 2000. Five value of the communication-computation-ratio(CCR)were selected to be 0.1,1,2,5,and 10. The computation time for each task was a random number between 10 and 50.The performance data are the average two hundred graphs.

We compared LPSO with GA on these random graph proceed in different topologies of multiprocessor systemss such as full20 ,full40 and ring20. The results are shown in Table II and Fig.5. Because of the random task graph, we show the critical path length (CPL) as a reference in Table II. These results show CCR have no significant effect on the execution time but on the schedule length. With the increase of CCR, the execution time of GA and LPSO changes slowly, but the execution time of LPSO is always smaller than that of GA. Except for the case that the number of processor is more than the width of the task graph ,for example, full40 and 1000 tasks, which results that the schedule lengths obtained by GA and LPSO are equal to the CPL, LPSO can obtain the smaller schedule length than GA. On the average , schedule length

TABLE I. RESULTS ON THE DETERMINISTIC TASK GRAPH

Number of processor	OPT [11]	Schedule Length GA	Schedule Length LPSO	Execution Time(s) GA	Execution Time(s) LPSO
2	1242	1245	1253	9.890	5.359
3	879	938	889	10.921	6.516
4	659	774	732	8.953	7.575
5	586	679	652	13.171	8.891
6	573	627	614	10.218	9.578
7	570	609	587	15.453	10.515
8	570	570	570	10.968	0.094
9	570	570	570	12.124	0.094

obtained by LPSO improved that of by GA about 14.7% and GA is 1.97 times slower than LPSO .Therefore the proposed LPSO algorithm possesses the strength to explore good solutions for scheduling different types of task graphs with different CCR onto different system graphs. LPSO is a viable alternative for the multiprocessor task scheduling problem in real-time system.

VI. Conclusions

In this paper, a novel task scheduling algorithm for real-time multiprocessor systems is presented. Simulation results demonstrate the effectiveness and efficiency of the presented algorithm. Furthermore, it also provides a novel way to apply PSO to the combinatorial optimization problem, i.e., PSO can be directly utilized without redefining the operators, if the special encoding and decoding methods are adopted.

References

[1] M.R.Garey, D.S.Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, New York:W.H.Freeman ,1979.

[2] .J.Kennedy, R.C,Eberhart, “Particle swarm optimization,” In Proceedings of the Fourth IEEE International Conference on Neural Networks, Perth, Australia , 1995, pp.1942-1948.

[3] R.C,Eberhart,Y.H,Shi, “Evolving artificial neural networks,” In Proceedings of 1998 International Conference on Neural Networks and Brain, Beijing, China, 1998, pp.5-13.

[4] M.Jacqueline, C.Richard, D.Gerry, “Multiobjective particle swarm optimization,”In Proceedings of the 38th annual on southeast regional conference, ACM Press, 2000, pp.56-57.

[5] Y.H.Shi,R.C.Eberhart,Y.Chen, “Implementation of evolutionary fuzzy systems,”IEEE Transaction on Fuzzy Systems, vol.7,no.2, 1999, pp.109-119.

[6] P.Wei, K.P.Wang, C.G.Zhou, L.J.Dong, “Fuzzy discrete particle swarm optimization for solving traveling salesman,”IEEE 2004 the Fourth International Conference on Computer and Information, 2004, pp.796-800.

[7] J.X.Du, D.S.Huang, J.Zhang, X.F. Wang, “Shape matching using fuzzy discrete particle swarm optimization,”In Proceedings of IEEE Swarm IntelligenceSymposium, 2005, pp.405-408.

[8] T.Y.Sun, S.T.Hsieh, H.M.Wang, “Floorplanning based on particle swarm optimization,”In Proceedings of the 2006 Emerging VLSI Technologies and Architectures, IEEE Computer Society,2006.

[9] A.El-Dib Amgad, K.M.Y.Hosam, M.M.EL-Metwally, Z.Osman, “Optimum varSizing & allocation using particle swarm optimization,”IEEE PES TD 2005/2006, 2006, pp.1108-1115.

[10] J.Kennedy, R.C.Eberhart, Y.Shi, Swarm Intelligence, San Francisco:Morgan KaufmannPublishers, 2001.

[11] E.S.Hou,N.Ansari, H.Ren, “A genetic algorithm for multiprocessor scheduling,” IEEETransaction on Parallel and Distributed Systems, vol. 5, no. 2, 1994, pp.113-120.

[12] Y.Tsujimura, M.Gen, “Genetic algorithms for solving multiprocessor scheduling problems,”Lecture Notes in Computer Science, Springer Berlin / Heidelberg, vol.1285, 1997, pp.106 – 115.

[13] M-y Wu, W Shu, J Gu, “Efficient local search for DAG scheduling,”IEEE Transaction on Parallel and Distributed Systems, vol.12,no.6, 2001, pp.617-627.

TABLE II. COMPARISON OF SCHEDULE LENGTH ON THE RANDOM TASK GRAPH

Number of task	CCR	CPL	full20		full40		ring20	
			GA	LPSO	GA	LPSO	GA	LPSO
1000	0.1	340	589	500	340	340	601	530
	1	610	690	620	610	610	1166	910
	2	910	1068	920	910	910	1530	1280
	5	1810	2131	1822	1810	1810	2561	2270
	10	3310	3692	3320	3310	3310	4155	3630
2000	0.1	483	1145	981	602	531	1189	1001
	1	870	1284	1010	1021	880	1751	1470
	2	1300	1382	1350	1555	1310	2300	2010
	5	2590	3171	2620	3008	2600	4199	3560
	10	4740	5392	4770	5068	4750	7016	6070

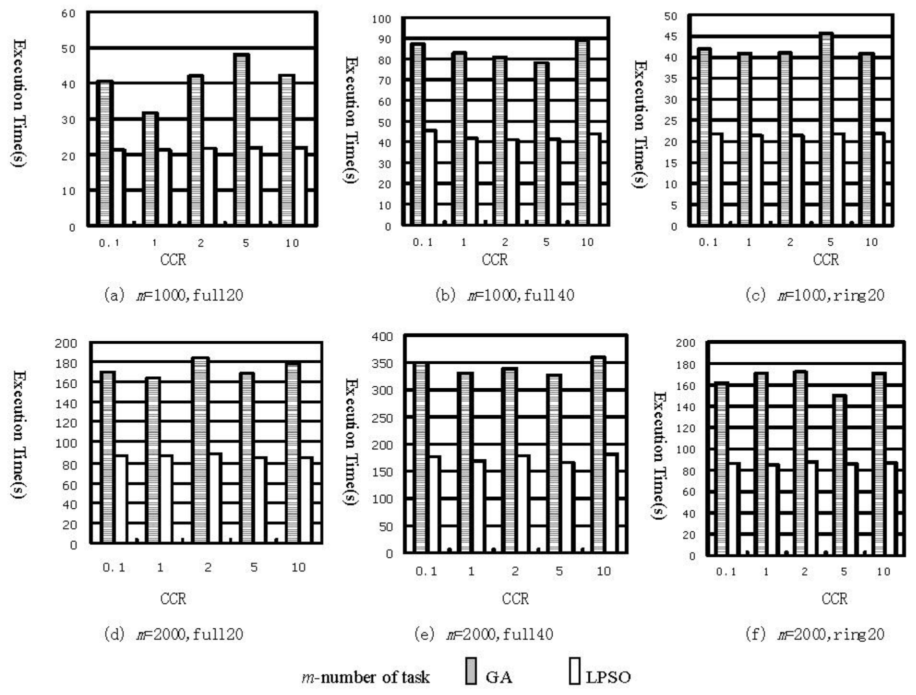


Fig.5 Comparison of runtime on the random task graph

[14] Y.K.Kwok , I. Ahmad, "Benchmarking and comparison of the task graph scheduling algorithms,"Journal of Parallel and Distributed Computing, vol.59,no.3, 1999, pp.381-422.

[15] H.P.Chen, L.S.Huang, "Processor selection policy in heuristic task scheduling," Journal of Software, vol.10, no.11, 1999, pp.1194-1198.

[16] T.Tsuchiya, T.Osada, T.Kikuno, "Genetic-based multiprocessor scheduling using task duplication,"Microprocessors and Microsystems, vol. 22,no.3/4, 1998, pp.197-207.

[17] A.Y.Zomaya, C.Ward, B.Macey, "Genetic scheduling for parallel processor systems comparative studies and performance issues," IEEE Transaction on Parallel and Distributed Systems, vol. 10, no. 8, 1999, pp.795-812.

[18] M.K.Dhodhi, I.Ahmad, "A multiprocessor scheduling scheme using problem-space genetic algorithms," Proc. IEEE Int'l Conf. Evolutionary Computation , 1995, pp.214-219.

[19] I.Ahmad, M.K.Dhodhi, "Multiprocessor scheduling in a genetic paradigm," Parallel Computing, vol. 22, 1996, pp.395-406.

[20] Y.Kwok, I.Ahmad, "Efficient scheduling of arbitrary task graphs to multiprocessors using a parallel genetic algorithm," Journal of Parallel and Distributed Computing, vol.47, no.1, 1997, pp.58-77.

[21] X.Q.Zhao, G.Rong, "Blending scheduling based on particle swarm pptimization algorithm," In Proceedings of the IEEE International Conference on Information Reuse and Intergration, 2004, pp.618-622.

[22] W.J.Xia, Z.M.Wu, "A hybrid particle swarm optimization approach for the job-shop scheduling problem," Springer London, 2006, pp.360-366.

[23] B.Liu, L.Wang, Y.H.Jin, "Hybrid particle swarm optimization for flow shop scheduling with stochastic processor time," Springer-Verlag Berlin/Heidelberg. CIS2005, part I, LNAI3801, 2005, pp.630-637.

[24] Y.Shi, R.C.Eberhart, "A modified particle swarm optimize,"In Proceedings of the IEEEInternational Conference on Evolutionary Computation, 1998, pp.69-73.

[25] J.Kennedy, "The particle swarm: social adaptation of knowledge," In Proceedings of IEEE International Conference on Evolutionary Computation, 1997,pp.303–308.

[26] J.P.Nowacki, G. Pycka , F.Seredynski, "Multiprocessor scheduling with support by genetic algorithms-based learning classifier system," J.Rolim et al. (Eds) IPDPS 2000 Workshops, Springer-Verlag Berlin/Heidelberg, vol.1800, 2000, pp.604-611.