

An Improved Genetic Algorithm for Load Balance in Multiprocessor Systems

Bin JIANG*, Rui LI**, Renfa LI**, Demin HAN*

* Embedded Systems & Networking Laboratory, HUNAN UNIVERSITY, China

** Graduate Training Innovation Base, HUNAN UNIVERSITY, China

binjiang518@yeah.net, rui@hnu.edu.cn, binjiang518@hnu.edu.cn

Abstract— The allocating and scheduling of tasks in parallel and distributed systems has been considered to be an NP-Complete problem, which has received much attention. Although plentiful algorithms have been developed to obtain suboptimal solutions, many of them didn't consider the total execution time and load balancing among processors simultaneously. To solve this problem efficiently, this paper presents an improved genetic algorithm based on the Critical Path Genetic Algorithm (CPGA) with some heuristic principles added to improve the performance. According to the evaluation results, our proposed algorithm could ensure the quality and efficiency of obtained solutions while avoiding the issues of CPGA algorithm, and always outperforms the CPGA algorithm in the respect of load balancing.

Keywords— Task Scheduling, Load Balance, Critical Path, Genetic Algorithm, Multiprocessor System

I. INTRODUCTION

Nowadays, the parallel and distributed systems play an important role in high-performance computing fields, e.g., massive data processing, fluid flow computing and weather modeling. However, the allocating and scheduling of these precedence-constrained tasks to processing elements (PEs e.g., processors) in these systems is known as an NP-complete problem [5, 9, 15].

One of the crucial aspects of scheduling problem in parallel computing system is load unbalancing where some PEs may be overloaded heavily while the others are idle. Thus, the scheduling problem involves how to allocate precedence-constrained tasks to PEs optimally such that minimizing overall execution time (i.e., schedule length), maximizing utilization of PEs and load balancing simultaneously under the precedence-constraints between tasks are satisfied [1]. Certainly, a large amount of algorithms have been proposed to solve this problem such as heuristic methods, among which the Genetic Algorithm (GA) is generally considered an efficient way to gain high quality solutions for a broad range of combinatorial optimization problems, multi-objective optimization problems and function optimization, e.g., the task

scheduling problem in parallel systems. Furthermore, genetic search could take advantage of its inherent parallelism to further reduce its execution time while searching for optimal solutions [2, 9, 14].

Therefore, we proposed an improved genetic algorithm named Improved Critical Path Genetic Algorithm (ICPGA), based on the Critical Path Genetic Algorithm (CPGA). Furthermore, our algorithm could ensure the quality and efficiency of obtained solutions while avoiding the issues of CPGA algorithm.

The remainder of this paper is organized as follows: Section 2 describes some related researches about our proposed algorithm. The scheduling model is demonstrated in section 3. Section 4 illustrates the details of our proposed CPGA-based algorithm. Section 5 gives the performance comparison between our proposed algorithm and the CPGA algorithm. Conclusions are given in section 6.

II. RELATED WORKS

The Standard GA (SGA) [12, 13, 14] first developed by Holland in the mid-1960s, is a stochastic search method based on the principles of biological evolution and natural genetics. The SGA search begins by initializing a population of feasible solutions at first, where individuals are encoded as strings called chromosomes. A chromosome corresponds to a solution to a given problem and some fitness functions are used to evaluate the fitness of individual. Then, good individuals survive after selection according to the fitness of individuals. After that, the survived individuals reproduce offspring through crossover and mutation operators. The above processes are repeated until a termination condition is met. Because of its simple mathematical calculations and efficient search for the approximate optimal solution, many variants of SGA have been developed and used widely in machine learning, pattern recognition, neural networks, artificial intelligence and other industrial applications [11].

The Critical Path Genetic Algorithm (CPGA) based on the Modified Critical Path (MCP) [2, 3] algorithm, is considered to be one of the best genetic algorithms for task scheduling in multiprocessor system. It reuses these idle times of processors

efficiently to reduce the schedule length by assigning some tasks to idle time slots, and gives high priorities to Critical Path Nodes (CPNs) such that these nodes can start as early as possible. On the other hand, the fitness function used to evaluate the performance of load balance among processors, is calculated by the ratio of the maximal execution time to the average execution time over all processors. However, there are some deficiencies for the CPGA algorithm: First, different selection pressures should be applied at different stages of population evolution to promote the diversity of population, and to avoid falling into local optimal solution. Second, the fitness function for load balance can be changed to improve the quality of solutions in some degree. Third, the chromosome representation can be modified to be more suitable for the case of Partitioned Scheduling [6].

III. THE DESCRIPTION OF PROBLEM

In general, the task scheduling problem in multiprocessor system could be divided into the following two main categories: deterministic (i.e., static) and non-deterministic (i.e., dynamic) [7, 8, 10]. In the former case, the attributions of a parallel program such as task dependencies, communication costs, task processing time and synchronization constraints are all known before execution. The work in this paper is only concerned with deterministic scheduling problem in uniform and non-preemptive system, where tasks are allocated onto homogeneous multiprocessors and each processor must completes the current task before executes the next one. Therefore, the scheduling model is constructed as follows [2, 4, 15]:

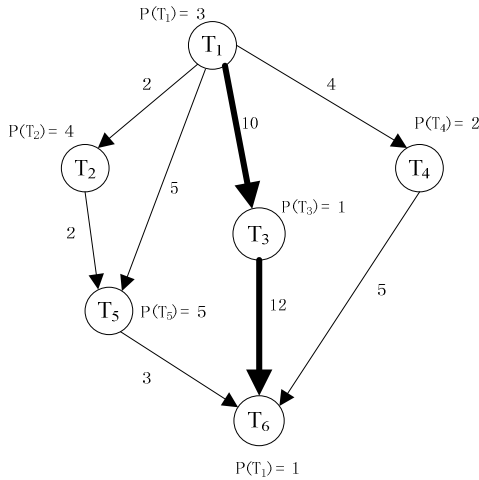


Figure 1. Example of a DAG

- Let a task graph $G = (V, E)$ be a Directed Acyclic Graph (DAG), where V and E is a set of nodes and directed edges respectively.
- Each task represented by a node, is a set of instructions that must be executed sequentially with no preemption on the same processor. The weight of each node is the processing time needed by the corresponding task, which is denoted by $P(n_i)$.

- The directed edge represents a partial order among the tasks. For example, if $n_i \rightarrow n_j$, then n_j cannot start until n_i finishes. The weight on a directed edge represents the communication cost between tasks, which is denoted by $C(n_i, n_j)$. Besides, the $C(n_i, n_j)$ would be equal to zero if n_i and n_j are scheduled on the same processor.
- If a node n_i is scheduled on processor P , the start time and finish time of n_i are denoted by $ST(n_i, P)$ and $FT(n_i, P)$ respectively.

IV. THE PROPOSED CPGA-BASED ALGORITHM

As mentioned above, the main objective of our proposed algorithm is to start the tasks as early as possible. Before presenting our algorithm in details, some definitions should be given first.

Definition 1: The total execution time (Schedule_length) of a parallel program is the maximum finishing time of all tasks, and can be defined as follows [2]:

$$\text{Schedule_length} = \max\{FT(n_i, P_j)\}$$

$$i = 1, 2, \dots, \text{NumOfNodes};$$

$$j = 1, 2, \dots, \text{NumOfProcessors}.$$

Definition 2: Because each task shouldn't execute until its parents have finished, the data arrival time (DAT) for each task is the maximum sum of its parent's finish time and communication cost, which can be defined as follows [2]:

$$DAT(n_i) = \max\{FT(n_k, P_j) + C(n_k, n_i)\}$$

$$\forall (n_k, n_i) \in E;$$

$$i = 1, 2, \dots, \text{NumOfNodes};$$

$$j = 1, 2, \dots, \text{NumOfProcessors}.$$

A. Encoding of Chromosome

The representation of chromosome is based on the schedule of each processor, which is more suitable for the case of Partitioned Scheduling than the CPGA algorithm. In other words, each processor has a scheduling list containing these tasks that have assigned to it, and preserving the precedence constraints of DAG as well. Figure 2 shows an example of such a representation of this chromosome, where Tasks T_1, T_3, T_6 would be scheduled on processor P_1 sequentially, tasks T_2, T_5 would be scheduled on processor P_2 , and task T_4 would be scheduled on processor P_3 .

$$\text{Chromosome} = \begin{cases} P_1 (T_1 | T_3 | T_6) \\ P_2 (T_2 | T_5) \\ P_3 (T_4) \end{cases}$$

Figure 2. Representation of a chromosome

B. Fitness function

The fitness function is used to evaluate the quality of candidate solutions to the scheduling problem. In our proposed algorithm, the main aim of scheduling is to minimize schedule length and maximize the performance of load balance among processors simultaneously. Therefore, the

fitness functions of our proposed algorithm are defined as follows, where $E_time[P_j]$ represents the execution time of processor P_j used for tasks:

$$\begin{aligned} fitness_function1 &= \alpha / (schedule_length)^\beta \\ fitness_function2 &= \lambda / \sum_{i=1}^N (E_time[P_i] - Avg)^2 \\ Avg &= \sum_{i=1}^N E_time[P_i] / N; \\ N &= NumOfProcessors; \\ \alpha, \beta, \lambda & \text{ is a parameter.} \end{aligned} \quad (3)$$

The $fitness_function1$ used to evaluate the performance of schedule length, could help expanding the differences between individuals greater than the CPGA algorithm's. Whereas, the $fitness_function2$ is only applied when there are two or more chromosomes with almost the same schedule length whose difference is in the range of acceptable. Here, in order to evaluate the performance of load distribution among processors more accurately than CPGA algorithm according to the chromosome, our proposed algorithm uses the method based on variance to evaluate the performance of these potential solutions.

Generally, the two fitness functions could distinguish the differences between individuals better than the CPGA algorithm. Besides, the computation of the Schedule Length for each chromosome in our proposed algorithm is implemented as follows [2]:

Begin

1. Assign TL = the topological order of DAG;
2. Adjust TL to preserve the intra-processor precedence constraints according to the chromosome;
3. Initiate $PT[1 \dots NumOfProcessors] = start_time$;
4. **While** TL is not empty **do**
5. Remove a task T_i from TL;
6. Get the processor number P_j where T_i is scheduled according to the chromosome;
7. $ST[T_i] = \max\{DAT(T_i), PT[P_j]\}$;
8. $FT[T_i] = ST[T_i] + P(T_i)$;
9. $PT[P_j] = FT[T_i]$;
10. **End While**
11. Schedule length = $\max\{FT(T, P)\}$;

End

Figure 3. The pseudo code of Schedule length

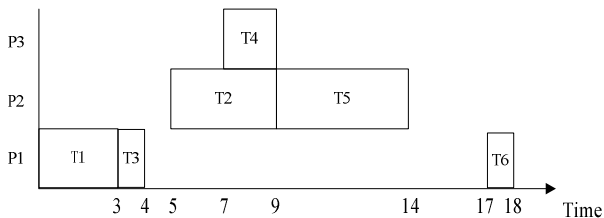


Figure 4. Gantt chart

For example 1: The schedule length of the chromosome in Figure 2 is 18, whose Gantt chart is shown in Figure 4. Note

that the $C(n_i, n_j)$ would be equal to zero if n_i and n_j are scheduled on the same processor.

C. Genetic operators

Genetic operators are used to reproduce a new population through selection, crossover and mutation. On one hand, we should promote the diversity of population such that our algorithm could scan the entire space of feasible solutions. On the other hand, the convergence rate of optimal solution shouldn't be ignored.

1) Initial population

All tasks are assigned to processors randomly, but the intra-processor precedence constraints must be preserved according to the topological order of the DAG.

2) Selection operator

The selection operator should be applied first to generate a mating pool for crossover and mutation operators, which could help guiding genetic search to a promising area of feasible solutions. To use a smaller selection pressure early in the genetic search for broad searching, and a bigger one in the late to limit search space, our proposed algorithm applied the Boltzmann annealing selection method in the selection phase, where the selection probability of individual a_j is as follows:

$$P_s(a_j) = e^{fitness_function(a_j)/T} / \sum_{i=1}^n e^{fitness_function(a_i)/T},$$

$$j = 1, 2, \dots, n;$$

$$T > 0.$$
(4)

where the annealing temperature T is the key control parameter for selection pressure and is related with the maximal evolution generation.

3) Crossover operator

Each chromosome in the population will be subjected to crossover with probability P_c . The specific steps of crossover operator could be described sequentially as follows:

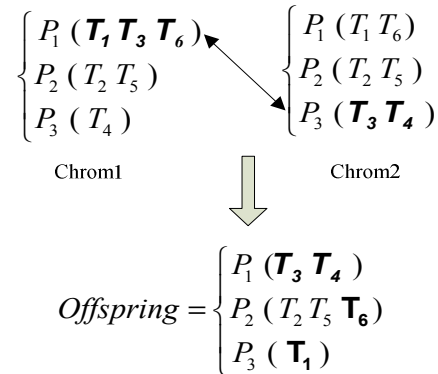


Figure 5. Crossover operator

- Choose two chromosomes Chrom1 and Chrom2 from current population, and let LT be a list of tasks according to the topological order of DAT and Chrom1.
- Generate two random numbers M, N, and exchange the P_M of Chrom1 with the P_N of Chrom2. Remove those tasks that have included in the Chrom1 from LT. Then,

leave those positions T_j to be free slots if T_j has been assigned to P_M .

- Fill in the above free slots using the tasks of LT in an ascending order according to processor number.
- If LT is not empty, the remaining tasks would be assigned to processors randomly except P_M . Adjust the task lists according to the topological order of the DAG.

For example 2: As shown in Figure 5, $LT = \{T_1, T_2, T_3, T_4, T_5, T_6\}$ and $M = 1, N = 3$. First, exchange $Chrom1.P_1$ with $Chrom2.P_3$. After removing and changing, $Chrom1 = \{P_1(T_3, T_4), P_2(T_2, T_5), P_3(_)\}$ and $LT = \{T_1, T_6\}$. Then, using T_1 to fill the free slots denoted by character ' $_$ ', and assigning T_6 to P_R , where $R \in \{\{1 \dots NumOfProcessors\} - \{M\}\}$ is a random number. Finally, adjust the task lists according to the topological order of the DAG.

D. Mutation operator

Each chromosome in the population will be subjected to mutation with probability P_m . At first, a chromosome is selected from population. Then, two task lists (e.g., P_n and P_m) are selected randomly from the chromosome, and a random number is generated as a time point to split each of the task lists into two parts: $ST(T_i, P) < R, \forall T_i \in$ the left parts and $ST(T_j, P) \geq R, \forall T_j \in$ the right parts. Finally, the parts of task lists lying to the right of the time point are exchanged to produce a new offspring (see Figure 6).

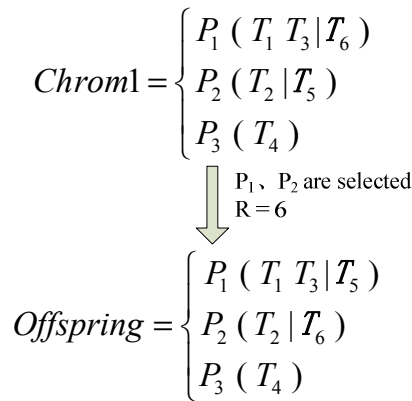


Figure 6. Mutation operator

E. Local Search

To take full advantage of each processor, our algorithm used the improved Test_slots function based on the CPGA algorithm to adjust the intra-processor scheduling order for each processor. There exist a situation in the CPGA algorithm that a task T may be assigned to the idle time slot whose S_idle_time is greater than $ST(T, P)$, which will be contrary to the principle of start task as early as possible. In addition, the data arrival time of a task may be greater than the S_idle_time of an idle time slot and the idle time slot could accommodate the task indeed, but the idle time slot will be nothing in CPGA algorithm. Therefore, the pseudo code of our proposed algorithm for local search is described as follows:

Begin

```

1. Assign  $LT$  = the topological order of the chromosome;
2. Let  $LP[1 \dots NumOfProcessors]$  be a list of idle time slots of each processor;
3. While  $LT$  is not empty do
4.   Remove a task  $T_i$  from  $LT$ ;
5.   For  $j = 1$  to  $NumOfProcessors$  do
6.     IF  $T_i$  is scheduled on  $P_j$  &&  $LP[j] \neq \text{NULL}$  Then
7.       For each time_slot of the  $LP[j]$  do
8.         IF  $E\_idle\_time - S\_idle\_time \geq P[T_i]$ 
9.           &&  $ST(T_i, P_j) > S\_idle\_time$ 
10.          &&  $(DAT(T_i, P_j) \leq S\_idle\_time$ 
11.            ||  $DAT(T_i, P_j) + P[T_i] \leq E\_idle\_time$ ) Then
12.              Schedule the  $T_i$  on the idle_time_slot;
13.              Update the idle time slots list of  $P_j$ ;
14.          End IF
15.        End While
16.      End For
17.   End While

```

Figure 7. The pseudo code of Test_Slots

V. PERFORMANCE EVALUATION

We have implemented our proposed algorithm and the CPGA algorithm on Pentium(R) Dual-Core CPU (3.19GHz) using C++ language to make a performance comparison between them. Our implemented programs would be terminated if the upper bound of generations is reached, or the best individual fitness exceeds a pre-set value, or the average fitness exceeds a pre-set value in respective iterations. Besides, the population size is set to 100, and the task graph composed of 400 tasks is generated randomly. These communication costs are distributed uniformly between 1 and a specified maximum (MCC). There are some control parameters for programs Pg1, Pg2 and Pg3 of our proposed algorithm respectively (see TABLE 1).

TABLE 1. CONTROL PARAMETERS

Programs	α	β	λ	P_m	P_c	Fitness Tolerance	MCC
Pg1	1	1	1	0.99	0.01	1e-2	15
Pg2	1	2	1	0.99	0.01	1e-4	15
Pg3	1	4	1	0.99	0.01	1e-5	15

First, our proposed algorithm and the CPGA algorithm are compared with respect to the overall execution time (i.e., schedule length) where the number of processors is 20. As shown in Figure 8, the schedule length of both algorithms decreases with the number of generations increases, and eventually converges to a constant. However, the control parameter β could help distinguishing the differences between individuals better than CPGA algorithm.

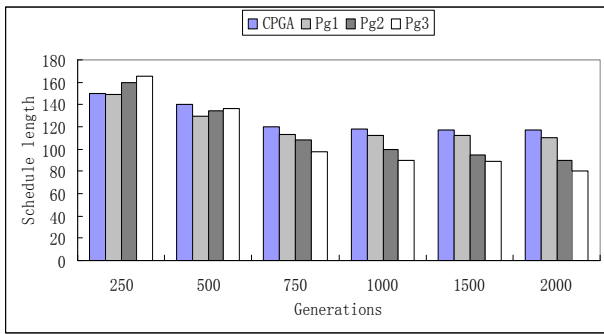


Figure 8. The comparison of schedule length

Second, our proposed algorithm and the CPGA algorithm are compared with respect to the Load Balance (LB) with almost the same schedule length. The LB is defined as follows:

$$LB = \sqrt{\sum_{i=1}^N (E_time[P_i] - Avg)^2}$$

where the $E_time[P_i]$, N and Avg are defined the same as equation (3). Figure 9 shows that our proposed algorithm could obtain a better approximate solution than the CPGA's in this respect, thus always outperform the CPGA algorithm in general. Here, the LB value is smaller, the better performance of the load balancing.

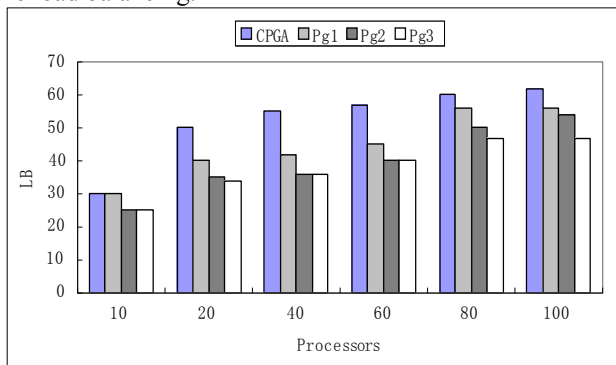


Figure 9. The comparison of load balance

VI. CONCLUSIONS

After using Boltzmann selection method and modified fitness functions, the stability and accuracy of GA computing for optimal solutions are greatly improved, although a little time overhead is increased. According to the obtained results, our proposed algorithm could distinguish the differences between individuals better than the CPGA algorithm in general. Besides, our proposed algorithm could ensure the quality and efficiency of obtained solutions to scheduling problem in multiprocessor systems while avoiding the issues of CPGA algorithms. Furthermore, our next objective is to improve the proposed algorithm so that it could be applied to dynamic scheduling model.

REFERENCES

[1] M.Nikravan and M.H.Kashani, "A genetic algorithm for process scheduling in distributed operating systems considering load

balancing," Proceeding 21st European Conference on Modeling and simulation (ECMS), 2007.

[2] Fatma A. Omara, Mona M.Arafa, "Genetic algorithm for task scheduling problem," Journal of Parallel and Distributed Computing, 2010, 13-22.

[3] MIN-YOU WU, DANIEL D.GAJSKI, "Hypertool: A programming aid for message-passing systems," IEEE Transaction on Parallel and Distributed Systems, 1990, 330-343.

[4] Reakook Hwang, Mitsuo Gen, Hiroshi katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs," Computers & Operations research, 2008, 976-993

[5] Hesham El-Rewini, Hesham Hi.Ali, "Task scheduling in multiprocessing systems," IEEE 1995, 27-36.

[6] Karthik Lakshmanan, Ragunathan (Raj) Rajkumar and P.Lehoczky, "Partitioned Fixed-Priority Preemptive Scheduling for Multi-Core Processors," IEEE Computer Society, 2009: 239-248.

[7] Y.KWOK, I.Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," ACM Comput.Surv.31 (1999) 406-471.

[8] M.A. Palis, J.C. Liou, S.Rajasekaran, S.Shende, S.S.L Wei, "Online scheduling of dynamic trees," Parallel Process. Lett.5 (1995) 635-646.

[9] A.S Wu, H. Yu, S.jin, K.-C.Lin, G.Schiavone, "An incremental genetic algorithm approach to multiprocessor scheduling," IEEE Trans. Parallel Distrib. Syst. 15(2004) 824-834.

[10] W.Yao, J.Yao&B.LI, "Main Sequences Genetic Scheduling for Multiprocessor Systems Using Task Duplication," International Journal of Microprocessors and Microsystems, 2004, 85-94

[11] Wenxiu ZHANG, Leung Yee, "The mathematical basis of the genetic algorithm," Xi'an Jiaotong University Press. 2001.

[12] Xiaoping Wang, Liming CAO, "Genetic algorithmsh: Theory, Application and Software," Xi'an Jiaotong University Press. 2002.

[13] Mitsuo Gen, Runwei Cheng, "Genetic algorithms and Engineering Optimization," Wiley-interscience. 1999.

[14] Ruifeng HAN, "Principles and Applications of genetic algorithms," Ordnance Industry Press. 2010.

[15] H.El-Rewini, T.G.Lewis, H.H. All, "Task scheduling in Parallel and Distributed Systems," Prentice-Hall International Editions, 1994.