Tasks Scheduling on Parallel Heterogeneous Multi-Processor Systems using Genetic Algorithm

Mohammad Sadeq Garshasbi
Department of Computer Engineering, Germi branch, Islamic Azad University, Germi, Iran.

Mehdi Effatparvar
ECE Department, Ardabil Branch, Islamic Azad University, Ardabil, Iran

ABSTRACT
With the increasing use of computers in research contributions, added need for faster processing has become an essential necessity. Parallel Processing refers to the concept of running tasks that can be run simultaneously on several processors. There are conditions that tasks have deadlines for scheduling. Therefore, the tasks should be scheduled before deadlines. May number of tasks before scheduling reached their deadline, Therefore, these tasks lost. These conditions are unavoidable. Thus, parallel multi-processor system tasks should be scheduled in a way, minimizing lost tasks. On the other hand, achieving good response times is necessary. this is an NP-Complete problem. In this article, we introduce a method based on genetic algorithms for scheduling tasks on parallel heterogeneous multi-processor systems for tasks with deadlines. The results of the simulations indicate reduced number of lost tasks in comparison with the LPT and SPT algorithms. Moreover, the response time of the proposed method due to its number of processing tasks, is appropriate, in comparison with the algorithm LPT and SPT.

General Terms
Scheduling, Parallel systems

Keywords
Parallel Heterogeneous Multi-processor Systems, Genetic Algorithm, Deadline, Lost Tasks.

1. INTRODUCTION
In multiple processing, multiple processors work together to implement a program. The major application of these systems is for problem solving in modeling and engineering sciences (e.g. Applied Physics, Nuclear Physics, Geology and Seismology, Mechanical Engineering, Electrical Engineering, Mathematics etc.). Today, not only scientific problems solving requires parallel processing, but also some commercial applications require fast computers. Many of these applications require the processing of large volumes of complex information. Some of these programs include huge databases, data mining operations, oil exploration, medical imaging and diagnosis etc [2,3,7].

In parallel heterogeneous multi-processor systems, there are conditions that tasks have deadlines for scheduling. Therefore, the tasks should be scheduled before the deadlines. May number of tasks before scheduling reached their deadline, Therefore, these tasks lost. These conditions are unavoidable. Therefore, there should be policies to reduce the number of lost tasks. In addition, obtaining an appropriate response time is a necessity. The allocation sequence of tasks in a heterogeneous multi-processor system has direct impact on the number of lost tasks and response times. Therefore, we use genetic algorithms to determine the task's optimal sequence to allocate to processors.

A Genetic Algorithm (GA) approach is proposed to handle the problem of parallel system task scheduling. A GA starts with generation of an individual, which is encoded as strings known as chromosomes. A chromosome corresponds to a solution to the problem. A fitness function is used to evaluate the fitness of each individual. In general, GAs consist of selection, crossover and mutation operations based on some key parameters such as fitness function, crossover probability, and mutation probability [1].

Results of the simulations indicate reduction in the number of lost tasks in comparison with LPT and SPT the algorithms. In addition, the response time of the proposed method is appropriate due to the number of processing tasks in comparison with the LPT and SPT algorithms.

This study is divided into the following sections: In section 2 an overview of the problem is given along with brief description of the solution methodology. Section 3 presents an improved genetic algorithm in detailed. The proposed method is described in Section 4. Results of the study are analyzed in Section 5. Finally, Section 6 presents the conclusions.

2. Problem Definition
Scheduling can be deterministic and non-deterministic: deterministic means that all information about the tasks are known. This information includes execution time and deadline for each task. Non-deterministic means that only the probability information of tasks are known. Here, we focus on a deterministic schedule [4,8,9].

Parallel multi-processor systems can be homogeneous or heterogeneous: Heterogeneous means that the processors have different computing speeds and capacities. Homogeneous means that all processors have equal computing speeds and capacity [4,5,6,13].

Tasks can be independent or dependent. An independent task means that the tasks, for running, do not need to run other tasks and these tasks can be executed at any time without knowledge of the information of other tasks. On the other hand, a dependent task means that each task may require the information of other tasks for execution. for run a task should be executed other tasks. In addition to dependent and independent tasks, each task may constitute a deadline. If the task is scheduled before the deadline, as a result the task will be executed; otherwise the task lost[12,13].

In this study, a study has been done regarding the task scheduling problem as a deterministic on the heterogeneous multiprocessor environment and independent tasks.

The multiprocessor computing environment consists of a set of m heterogeneous processor:

$$P = \{p_i : i = 1, 2, 3 \ldots m\} \quad (1)$$

They are fully connected with each other via identical links. Fig 1 shows a fully connected eight parallel system with identical link[4,11].

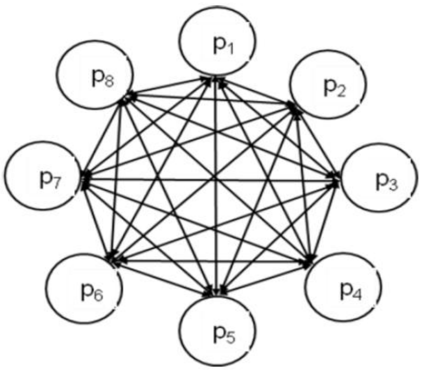

Fig 1: A fully connected parallel processor[4]

P indicates the numbers of heterogeneous processors that can be to *m* of exist heterogeneous processors. Processors are heterogeneous, therefore in heterogeneous environments, every processor works in different speeds and processing capabilities. Assume processor p1 is faster than p2, p3 and so on. Likewise, processor p2 is faster than p3, p4 and so on. (i.e., the order of speed and processing capabilities can be expressed as p1>p2> p3 > p4 > p5 > p6 > p7>p8) [4,10].

According to the above description, table 1 shows an example of this problem. There are five tasks and two heterogeneous processors and each of the tasks have deadlines. Because processors are heterogeneous, each task has different run times on different processors. Task 2 takes 2 time units to complete execution on processor 1, and 4 time units to complete execution on processor 2. This run time has been calculated based on the size of the tasks by processing on different processors. Each of the tasks has a deadline, and must be executed before the deadline.

Table 1: Shows a tasks execution matrix and Deadline on different processors

Tasks	Task1	Task2	Task3	Task4	Task5
P1 (RunTime)	4	1	3	2	3
P2 (RunTime)	6	2	5	4	4
Deadline	4	2	1	3	1

According to the example, assume tasks T3, T5, T4, T2, T1 be scheduled, respectively. Hence, scheduling is according to Fig 2. T3 first enters processor 1. According to table 1, T1 run time in processor 1 is equal to 3. Then, T5 enters processor 2. According to table 1, T5 run time in processor 2 is equal to 4, and on. The obtained result in this schedule is one lost task, and response time is 10.

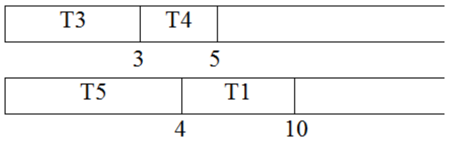

Fig 2

Assume T1, T3, T5, T4, T2 tasks be scheduled, respectively. Therefore scheduling is according to Fig 3. At first, T1 enters processor 1. According to table 1, T1 run time in processor 1 is equal to 4. Then, T3 enters processor 2, according to table 1, T3 run time in processor 2 is equal to 5. Therefore, the deadlines of tasks T5, T4 and T2 have passed. The obtained result in this schedule is three lost tasks, and response time is 5.

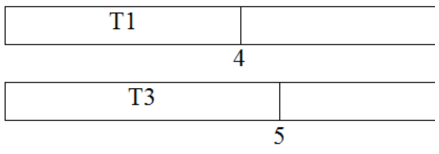

Fig 3

The first scheduling is optimized, compared to the second scheduling the reason is that the number of lost tasks in the first scheduling is low and the response time due to number of processing tasks is appropriate.

Some tasks are without deadlines; therefore, we divide the tasks into two groups: tasks with deadlines, and tasks without deadlines. First, we scheduled the tasks with deadlines using Genetic Algorithm, and then scheduled tasks without deadlines using Genetic Algorithm.

How to respectively Entry (scheduling) of tasks to processors that minimizes the total execution time and also minimize number of lost tasks is NP-Complete problem. Therefore, in this article, we use GA for minimizing total execution time and the number of lost tasks in parallel multi-processor systems.

3. Genetic Algorithms

In the computer science field of artificial intelligence, a genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover[14].

In a genetic algorithm, a population of strings (called chromosomes or the genotype of the genome), which encode candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem, evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Genetic algorithms find application in bioinformatics, phylogenetics, computational science, engineering, economics, chemistry, manufacturing, mathematics, physics and other fields.

The fitness function is defined over the genetic representation and measures the quality of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The fitness of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Once the genetic representation and the fitness function are defined, a GA proceeds to initialize a population of solutions (usually randomly) and then to improve it through repetitive application of the mutation, crossover, inversion and selection operators.

In genetic algorithms, crossover is a genetic operator used to vary the programming of a chromosome or chromosomes from one generation to the next. It is analogous to reproduction and biological crossover, upon which genetic algorithms are based. Cross over is a process of taking more than one parent solutions and producing a child solution from them. There are methods for selection of the chromosomes.

In genetic algorithms, mutation is a genetic operator used to maintain genetic diversity from one generation of a population of algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search[14]. GA using operator selection, combination and mutation provide the optimal solution that is not possible with other methods.

4. Tasks scheduling using genetic algorithms in parallel multi-processor systems

4.1 Chromosomes and Encoding

The purpose of this study is to find a sequence of tasks so that the number of lost tasks is minimum, and the response time compared to number of processing tasks is appropriate. Thus, each chromosome is sequence variety of tasks. Each task is considered as a gene. Therefore, the best way to encode chromosomes is permutations encoding. To explain how chromosomes are encoded, consider that there are 8 tasks, Ti represents the tasks. Fig 4 shows two encoded chromosomes.

Ch 1	T1	T4	T8	T2	T7	T3	T6	T5
Ch 2	T6	T1	T7	T3	T5	T8	T2	T4

Fig 4: two of chromosome encoded

Therefore, size of chromosomes in the first step is equal to the number of tasks with deadlines, and size of chromosomes in the second step is equal to the number of tasks without deadlines. The first step means task scheduling with deadlines, and second step means task scheduling without deadlines.

4.2 Generate the Initial Population

To start, GA should generate an initial random population for entry into the first generation. For this, a random generator function of chromosomes must be employed[4]. In order to create an initial population, we need Information on the number of processors, number of tasks, task deadlines, and the size of the population. Random chromosomes generate the initial population.

4.3 Fitness Function

The important part of GA is the fitness function. The fitness function is defined over the genetic representation, and measures quality of the chromosomes. The fitness function is always dependent on the problem. In this article, the fitness function separates evaluation into two parts: lost tasks, and total response time. The fitness function in the first step is based on the number of lost tasks and total response time. the fitness function in the second step is only based on total response time.

The fitness function is calculated according to the (2) equation:

$$F = (\alpha \times M) + ((1 - \alpha) \times TFT) \qquad (2)$$

Where M is the number of lost tasks, TFT is response time obtained from the chromosome, and α is a value between zero and one ($\alpha = [0,1]$). If $\alpha = 1$, then the evaluation is based on the number of lost tasks. If $\alpha = 0$, then the evaluation is based on total response time. α can have a value between 1 and 0, this value determines importance of the total response time, and importance of the number of lost tasks. Lesser value of the above equation corresponds to a better fitness value for the chromosome.

4.4 Selection Operator

The design of the fitness function is the basic of selection operation, so how to design the fitness function will directly affect the performance of genetic algorithm. GAs uses selection operator to select the superior and eliminate the inferior. The individual are selected according to their fitness value. Once fitness values have been evaluated for all chromosomes, we can select good chromosomes through rotating roulette wheel strategy. This operator generate next generationby selecting best chromosomes from parents and offspring[4].

4.5 Crossover Operator

Crossover operator randomly selects two parent chromosomes (chromosomes with higher values have more chance to be selected) and randomly chooses their crossover points, and mates them to produce two child (offspring) chromosomes[4]. We consider one point crossover in here, In one point crossover, the segments to the right of the crossover points are exchanged to form two offspring as shown in Fig. 5.

parent 1	T1	T3	T4	T2	T8	T5	T6	T7
parent 2	T5	T3	T1	T4	T7	T8	T6	T2
Child	T1	T3	T4	T5	T7	T8	T6	T2

Fig 5: One point crossover

4.6 Mutation Operator
A mutation operation works by randomly selecting two tasks and swapping them. Firstly, it randomly selects a processor, and then randomly selects a task on that processor[4]. This task is the first task of the pair to be swapped. Secondly, it randomly selects a second processor (it may be the same as the first), and randomly selects a task. If the two selected tasks are the same task the search continues on. If the two tasks are different then they are swapped over (provided that the precedence relations must satisfy). Fig 6.

befor	T1	**T4**	T8	T2	**T7**	T3	T6	T5
ofter	T1	**T7**	T8	T2	**T4**	T3	T6	T5

Fig 6: Mutation operator

5. Evaluation of Simulation Results
In this section, we present and discuss the experimental results of the proposed scheme. All simulations were performed using MATLAB software. We evaluated the performance of our proposed scheme in comparison with LPT (Largest Processing Time) and SPT (Shortest Processing Time) algorithms in a Parallel multi-processor system. The simulation results showed two instances: when the numbers of tasks are more, and when the numbers of tasks are less.

The parameters of the considered GA are as follows:

Number of generations = 40
Crossover probability= 50%
Mutation probability = 20%
Chromosomes that enter the next generation unchanged = 30%
Number of GA iterations= 200
$\alpha = 0$

When the number of tasks is 100 (60 tasks have deadlines and 40 tasks are without deadline). Fig 7 shows the number of lost tasks by applying LPT, SPT and genetic algorithms for task scheduling on parallel multi-processor systems in these conditions(tasks=100). The vertical axis represents the number of lost tasks, and the horizontal axis represents the number of processors. Fig 8 shows the number of lost tasks when the number of tasks is 1000 (600 tasks have deadlines and 400 tasks are without deadline). Fig 9 and 10 show the total response time when the number of tasks is 100 and 1000, respectively.

Our approach in addition to reducing the number of lost tasks provides good total response time compared to the number of processed tasks. Since the number of lost tasks is fewer in GA, therefore, the number of tasks to be processed is more, and the total response time increases. However, the increase in total response time is not significant.

Obtained results in large and small scales indicate that our proposed method can provide similar results in different scales, and proves the robustness of the proposed method in different scales.

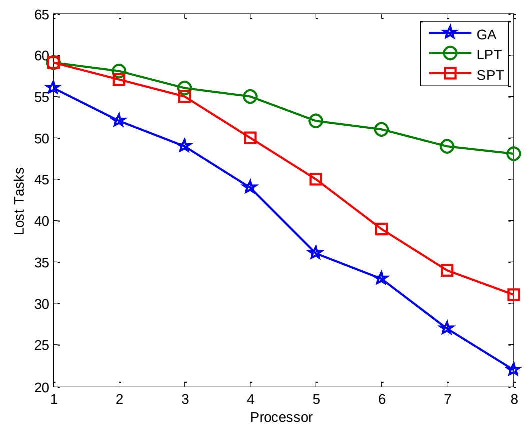

Fig 7: lost tasks when the number of tasks is 100.

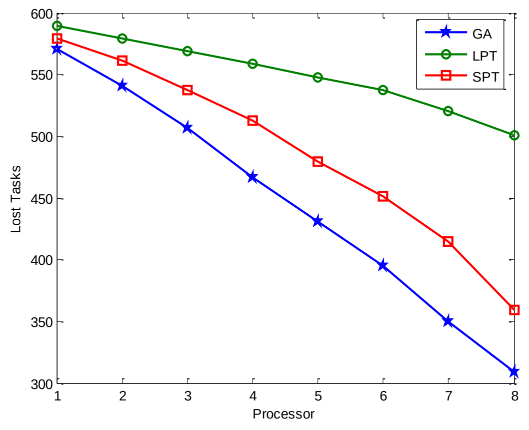

Fig 8: lost tasks when the number of tasks is 1000.

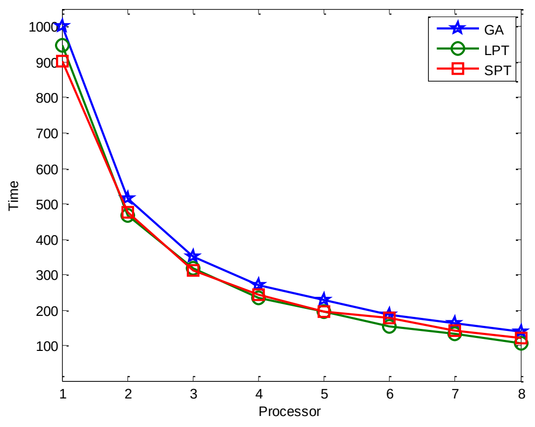

Fig 9: total response time, when the number of tasks is 100

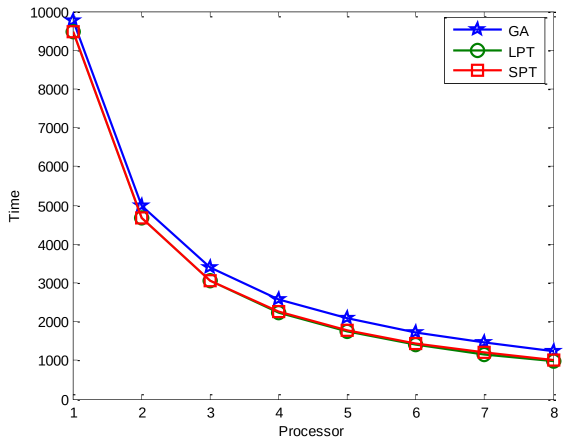

Fig 10: total response time when the number of tasks is 1000.

6. Conclusion

In this study, we proposed the Genetic Algorithm (GA) for tasks scheduling in heterogeneous parallel multiprocessor systems that number of tasks have deadlines, and number of tasks are without deadlines. In our approach, at first we scheduled tasks with deadlines, and then tasks scheduling without deadlines. The proposed method found a better solution for assigning tasks to the heterogeneous parallel multiprocessor system. This method reduces the number of lost tasks and provides appropriate total response time compared to the number of processed tasks. The method proposed in this article was compared with SPT and LPT algorithms. The results of simulations indicate that our method is better compared with the LPT and SPT algorithms. In addition, the obtained results are based on a limited number of reproduction and genetic simple operators. Certainly, gain the better results using of efficiently operators.

7. REFERENCES

[1] Mitchell, Melanie, "An Introduction to Genetic Algorithm", Published Bu MIT Press 1996.

[2] Ananth Grama, Georage Karypis, Anshul Gupta, Vipin Kumar, "Introduction to parallel computing", Published by Pearson Education, 2009.

[3] Tran, Van Hoai, "Task Scheduling for Parallel Systems ", Faculty of Computer Science and Engineering HCMC University of Technology, 2009-2010.

[4] Jasbir, Gurvinder, "Improved Task Scheduling on Parallel System using Genetic Algorithm", International Journal of Computer Applications (0975 – 8887) Volume 39– No.17, February 2012.

[5] Albert Y. Zomaya, Senior, Chris Ward and Ben Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 10, NO. 8, AUGUST 1999.

[6] Kamaljit Kaur, Amit Chhabra and Gurvinder Singh, " Modified Genetic Algorithm for Task Scheduling in Homogeneous Parallel System Using Heuristics", International Journal of Soft Computing 5 (2):42-51, 2010.

[7] Hadi Shahamfar and Sohrab Khanmohamadi, " A new Genetic Algorithm base on Neighborhood Search and Tabu List (GTNS) for Task Scheduling in Multiprocessing", International Journal of Soft Computing 3 (3):254-259, 2008.

[8] Probir Roy, Md. Mejbah Ul Alam and Nishita Das, " HEURISTIC BASED TASK SCHEDULING IN MULTIPROCESSOR SYSTEMS WITH GENETIC ALGORITHM BY CHOOSING THE ELIGIBLE PROCESSOR", International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.4, July 2012.

[9] Amit Bansal and Ravreet Kaur, "Task Graph Scheduling on Multiprocessor System using Genetic Algorithm", International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 1 Issue 5, July – 2012.

[10] Edwin S.H. Hou, Nirwan Ansari and Hong Ren, " A Genetic Algorithm for Multiprocessor Scheduling", IEEE TRANSZCTIONS ON PARALLEL AND DISTRIBUTED SYSTEM, VOL, 5, NO, 2 , FEBRUARY 1994.

[11] Rakesh, Rajiv, Sanjeev, Ashwani, Genetic Algorithm approach to Operating system process scheduling problem, International Journal of Engineering Science and Technology Vol. 2(9), 2010, 4247-4252.

[12] Preeti, Vaishali, Genetic algorithm Approach for Optimal CPU Scheduling, IJCST Vol. 2, Iss ue 2, June 2011.

[13] Yi-Wen Zhong; Jian-Gang Yang, "A genetic algorithm for tasks scheduling in parallel multiprocessor systems", IEEE Machine Learning and Cybernetics, 2003 International Conference on, 2-5 Nov. 2003, 1785 - 1790 Vol.3.

[14] http://en.wikipedia.org/wiki/Genetic_algorithm/