International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 5, July - 2012

Task Graph Scheduling on Multiprocessor System using Genetic Algorithm

Amit Bansal
M.Tech student
DCSE, G.N.D.U.
Amritsar, India

Ravreet Kaur
Asst. Professor
DCSE, G.N.D.U.
Amritsar, India

Abstract

Task Graph Scheduling is an important issue in the distribution of programs on the processors of a parallel system. Because task graph scheduling is an NP-Complete problem, methods of random search are utilized for finding the nearly optimal schedule. Recently, Genetic algorithms have received much awareness as they are robust and guarantee for a good solution. In this paper a new genetic algorithm is proposed based on Object Migration Automaton and is evaluated in comparison with FCFS and MET scheduling. The proposed algorithm begins with an initial population of randomly generated chromosomes and after some stages, each chromosome maps to an automaton.

Keywords: Genetic algorithms, FCFS (first come- first serve), MET (minimum execution time)

1. Introduction

Multiprocessing scheduling has been a source of challenging problems for researchers in the area of computer engineering[2]. The general problem of multiprocessor scheduling can be stated as scheduling a set of partially ordered computational tasks onto a multiprocessor system so that a set of performance criteria will be optimized. The task scheduling problem with precedence relations and data communications among tasks is known as an *NP-complete* problem for general cases. This paper presents a GA which is robust for the various models. The GA presented in this paper is implemented and optimal schedule is found using the then proposed GA.
The paper has been organized as following. The next section describes problem under consideration. Section 3 and 4 define genetic algorithms and Object Migration Automaton respectively. Section 5 is devoted to the proposed genetic algorithm. Experimental results and conclusions are included in section 6 and 7. References are in section 8.

2. Definition of the problem

Let a (homogeneous) multiprocessor system be a set of m identical processors, $m > 1$. Additionally, let a parallel program be a set of communicating tasks to be executed under a number of precedence constraints. To each task is associated a cost, representing its execution time. A weighted acyclic task digraph[5] can be used to represent the tasks (vertices of the task digraph) and the precedence constraints (arcs of the task digraph). In order to be executed, each task of a given parallel program must be scheduled to some processor of a given multiprocessor system. Consequently, tasks that communicate in the parallel program may be scheduled to different processors, so as to minimize the execution time of the program.
Considering these communications and the precedence constraints between tasks, it follows that different schedule of each task satisfying the precedence constraints lead to different execution times of the parallel program. The purpose is finding permutation of tasks, so that the considered permutation cost is the least.
In the proposed algorithm each chromosome is equal to an automaton and each gene equal to an action of an Object Migration Automaton[4]. The assumed parallel computational model in the paper is described below:

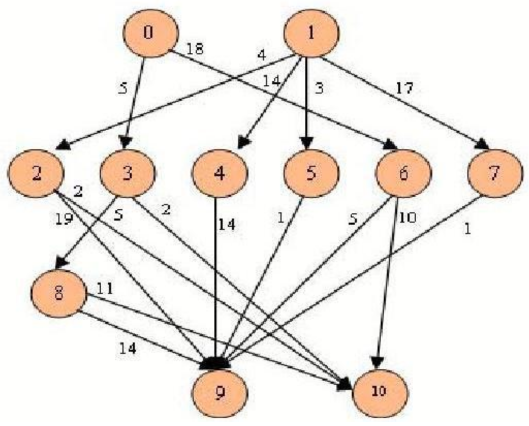

Figure 1: Directed Acyclic Graph

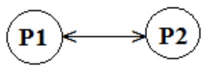

Figure 2: A fully connected Multiprocessor system

Table 1: Computation time for each task

Task	Time
T0	4
T1	15
T2	4
T3	13
T4	10
T5	7
T6	8
T7	4
T8	12
T9	6
T10	9

3. Genetic Algorithm

A genetic Algorithm is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges[1]. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search[7]. In every generation the fittest of that generation selected and after reproduction produce a new set of children. In this process the fittest individuals will survive more probably to the next generations.
The most advantages of this algorithm compared with common methods are: parallel search instead of serial search, not requiring any additional information such as problem solving method, in-deterministic of algorithm, easy implementation and reaching to several choices.

4. Object Migration Automata

In [4,9], B.J.Oommen gave us a detailed description about OMA. The object migrating automaton (OMA) is defined as a quintuple in which there are only R actions, each action representing a certain class. Furthermore, for each action, there are a fixed number of states N[8]. In this paper, the OMA is used to secure the position the position of genes obtained after applying top and bottom level precedence relations. Figure 3 shows an example of the Object Migration Automaton (OMA) with three arms, nine objects and four states in each arm.

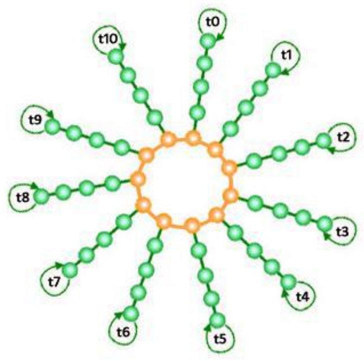

Figure 3: Example of an Object Migration Automaton

5. The Proposed Algorithm

The algorithm is described in four parts as follows and then pseudo-code for proposed genetic algorithm is presented:

I. Initial Population IP ()

The first step in the genetic algorithm is the creation of the initial population. Number of processors, number of tasks and population size are needed to generate initial population. Each individual of the initial population is generated through a min-min heuristic[6] along with bottom-level or top-level precedence resolution to avoid the problem of same execution time or completion time and same precedence.
The task to be scheduled for each iteration is determined by the following rules:
1. Calculate the bottom-level precedence relation of each task.
2. Sort the tasks with the precedence relation according to their bottom-level in descending order.
3. Assign the tasks to the processors in the order of their bottom-level.
OR
1. Calculate the top-level precedence relation of each task.
2. Sort the tasks with the precedence relation according to their top-level in descending order.
3. Assign the tasks to the processors in the order of their top-level.
The bottom-level and top-level precedence relations for DAG in Figure 1 are shown as follows:

Table 2: Bottom Level and Top Level Precedence Relations

Task Number	Bottom Level	Top Level
0	49	0
1	59	0
2	29	19
3	50	9
4	30	29
5	14	18
6	27	22
7	11	32
8	32	27
9	6	53
10	9	50

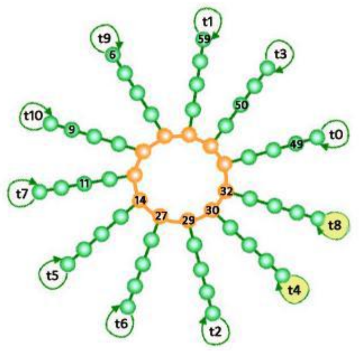

Figure 4: Chromosome created by Bottom-Level Precedence Relations

II. Fitness Function for ith chromosome.

```
Procedure FitnessFunc ()
  Begin
      for i = 1 to n do // OMA = Object Migration Automata
    f ($OMA_i$) = 1 / (Length of Permutation of $OMA_i$)
    End Procedure
```

Figure 5: Fitness function

III. Crossover Operator

Procedure Crossover (OMA_1,OMA_2)

To do this operator one of methods which is appropriate to work with permutations can be used: Partially Mapped Crossover, Ordered Crossover, Cycle Crossover, and New Crossover. In this paper only the proposed method namely New Crossover[3] is explained.

The two parent chromosome are chosen first and genes i, j are selected randomly in one of these chromosomes. Then these two genes are selected in another parent chromosome. The set of genes with number between i and j is called the crossover set. Then the genes with the same number are replaced with one another in two crossover set. (for example gene number i from the first crossover set with gene number i from the second crossover set, gene number i+1 from the first crossover set with gene number i+1 from the second crossover set, …). The result of this process is two chromosomes which are called the two parent automata's children.

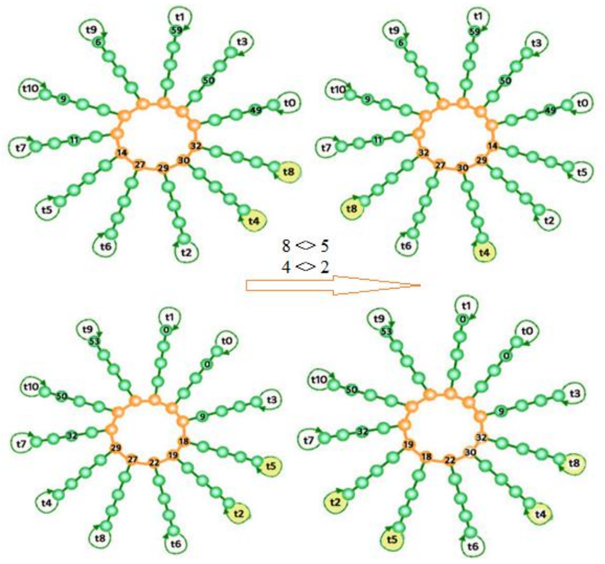

Figure 6: New Crossover Operator

IV: Mutation

```
Procedure Mutation (OMA)
  Begin
    i = Random *n; // n = total no. of chromosomes
    j = Random *n;
  Swap   OMA.Object(OMA.Action(i)),
         OMA.Object(OMA.Action(j));
  End Procedure
```

Figure 7: Mutation Operator

The proposed **Genetic Algorithm** pseudo-code is as follows:

```
Function GA (DAG):
 Begin
  Create the initial population IP ();
   // Select chromosomes based on their fitness
      from the current population.
  FitnessFunc ();
   // Apply crossover and mutation to generate
      offspring to form a new population
      ($OMA_{new}$).
  $OMA_{new}$ = Crossover ($OMA_1$,$OMA_2$)
  // Replace the current population with a new
     population.
  $OMA_{mut}$= Mutation ($OMA_{new}$)
  Select the best final solution.
 End
```

Figure 8: Proposed Genetic Algorithm pseudo-code

6. Experimental Results

The final best schedule of directed acyclic graph of Figure 1 obtained after applying proposed Genetic Algorithm is represented in Figure 11. The completion time obtained by proposed Genetic Algorithm is 49 time units. The result was compared with FCFS (First Come First Serve) scheduling method on multiprocessor system shown in Figure 9 and also MET scheduling method on same multiprocessor system.

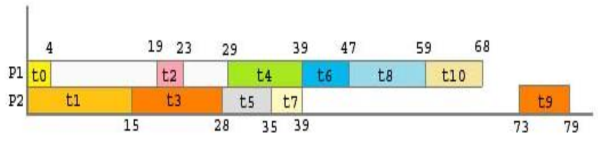

Figure 9: A Gantt chart of FCFS scheduler

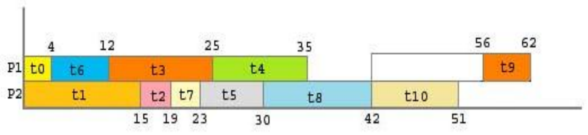

Figure 10: A Gantt chart of MET Scheduler

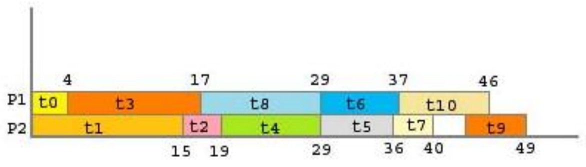

Figure 11: A Gantt chart of Proposed Genetic Algorithm

6.1. Performance Analysis

Speedup: It is defined as the completion time on a uniprocessor divided by completion time on a multiprocessor system.

Proposed Advance GA:
The speedup achieved for the illustrative example is:
Speedup = 92/49 = 1.877
Efficiency = (Speedup*100)/m where m = number of processors.
Efficiency = 1.877*100/2 = 93.87%

FCFS Scheduler:
Speedup = 92/79 = 1.164
Efficiency = 1.164*100/2 = 58.22%

MET Scheduler:
Speedup = 92/62 = 1.483
Efficiency = 1.483*100/2 = 74.19%

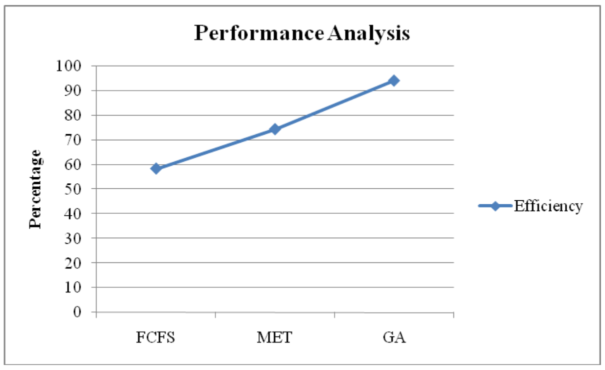

Figure 12: Performance Analysis of schedulers for given DAG

The proposed Genetic Algorithm is implemented on four different graphs with different number of nodes. The numbers of nodes taken in different graphs were 7, 11, 15 and 19. Then the results are compared with FCFS scheduling, MET scheduling and Uniprocessor scheduling.

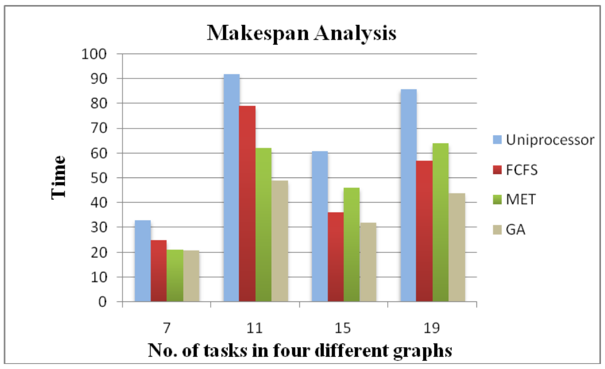

Figure 13: Experimental results for different graphs

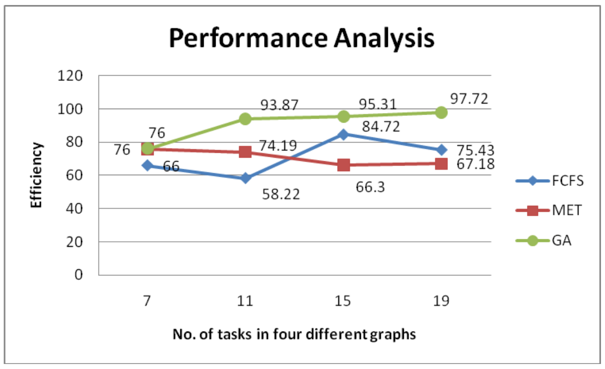

Figure 14: Performace Analysis of different graphs

The performance of proposed algorithm is calculated with different number of processors. Figure 13 shows the makespan of proposed GA of two different graphs having nodes 7 and 10 when executed on n=2 and n=3 processors.

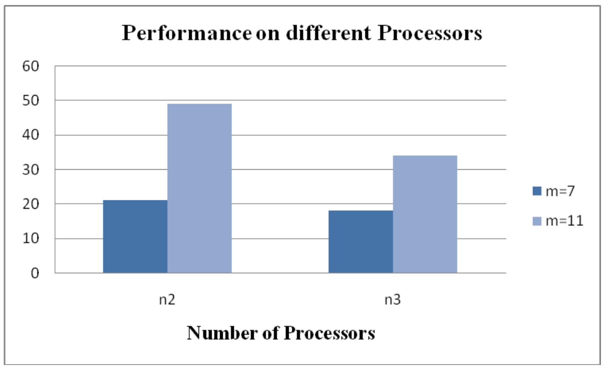

Figure 15: Makespan of Proposed GA on different processors on two graphs

7. Conclusions and further work

In this paper, a new genetic algorithm is proposed for task scheduling in parallel multi-processor system including the communication delays to reduce the completion time and to increase the throughput of the system. By analyzing the performance results, it has been verified that the proposed algorithm works well and reduces the overall completion time in comparison to other algorithms. In proposed technique, the initial population was generated on the basis of bottom-level and top-level heuristics with the help of which it was easy to amalgamate the overall procedure with Object Migration Automata and thus crossover operator was applied easily. The performance study is based on the best randomly generated schedule of the suggested GA. In future, OMA can be used with full functionality including its reward and penalize operations.

8. References

1. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Transactions on System, Man and Cybernetics*, pp. 122-128, 1986.
2. Edwin S. H. Hou and N. Ansari, "A Genetic Algorithm for Multiprocessor Scheduling", *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 2, pp. 113 – 120, 1994.
3. Bager Zarei and M.R. Meybodi, "A Hybrid method for Solving Traveling Salesman Problem", *ICIS 2007*, pp. 394-399, 2007.
4. B. J. Oommen and D. C. Y. Ma, "Deterministic Learning Automata Solution to the Keyboard Optimization Problem", *IEEE Trans. On Computers*, Vol. 37, No. 1, pp. 2-3, 1988.
5. Ricardo C. Correa and Afonso Ferreira , "Scheduling Multiprocessor Tasks with Genetic Algorithms" *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 8, pp. 825 – 837, 1999.
6. Kamaljit Kaur, Amit Chhabra and Gurvinder Singh, "Heuristics Based Genetic Algorithm for Scheduling Static Tasks in Homogeneous Parallel System" *International Journal of Computer Science and Security* 2010.
7. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.
8. K. S. Narendra and M. A. L. Thathachar, "Learning Automata: An Introduction", Prentice-hall, Englewood cliffs, 1989.
9. B. J. Oommen, R. S. Valiveti, and, J. R. Zgierski, "An Adaptive Learning Solution to the Keyboard Optimization Problem" , *IEEE Trans. On Systems. Man. And Cybernetics*, Vol. 21, No. 6, pp. 1608-1618, 1991.