International Journal of Modern Engineering Research (IJMER)

www.ijmer.com Vol.2, Issue.3, May-June 2012 pp-1290-1293 ISSN: 2249-6645

Flexible Tasks Scheduling Problem using A Genetic Algorithm-based Approach

D. R. Raut*, S. M. Patil**, T. M. T. Khan***

* *WIEECT/Computer Engineering, Mumbai, India*

** *BVCOE/Information Technology, Navi Mumbai, India*

*** *AIKTC/Computer Engineering, New Panvel, India*

***Abstract*:** Flexible Task Scheduling in a multiprocessor environment is NP complete problem. In literature, several heuristic methods have been developed that obtain suboptimal solutions in less than the polynomial time. Recently, Genetic algorithms have received much awareness as they are robust and guarantee for an effective solution. Genetic algorithm is wildly used to solve Flexible Task Scheduling Problem. The genetic algorithm we proposed uses many different strategies to get a better result. During the phase of create initial population, the genetic algorithm takes into account the number of operations in each job. And the intelligent mutation strategy is used which makes every individual and gene have different probability to mutate. In this paper, the object of scheduling algorithm is to get a sequence of the operations on machines to minimize the make span. During the experiments the performance of the genetic algorithm is compared with other heuristic algorithm. In our project we comprises of three parts: Quality of solutions, robustness of genetic algorithm, and effect of mutation probability on performance of genetic algorithm.

I. INTRODUCTION

The general problem of multiprocessor scheduling can be stated as scheduling a task graph onto a multiprocessor system so that schedule length can be optimized. Task scheduling in multiprocessor system is a NP-complete problem. Task scheduling algorithms can be broadly classified into two main groups: heuristic based [5] and guided random search based algorithm [5]. Heuristic based algorithm searches a path in the solution space based on the heuristic used while ignoring other possible paths. Due to this reason, they give good results for some inputs while bad for others. List scheduling algorithms [5], clustering [5] and duplication based algorithms [2] fall under this category. Guided random search techniques use random choices to guide them selves through the problem space. Genetic algorithms [1, 3 and 8] are the most popular random search techniques for different kind of task scheduling problems.

In the Multitasking environment considered here, an application task can be decomposed into subtasks, where each subtask is computationally homogeneous well suited to a single machine and different subtasks may have different machine architectural requirements. These subtasks can have data dependences among them. Once the application task is decomposed into subtasks, the following decisions have to be made: matching, i.e., assigning subtasks to machines, and scheduling, i.e., ordering subtask execution for each machine and ordering inter machine data transfers. In this context, the goal of Heterogeneous Computing is to achieve the minimal com-pletion time, i.e., the minimal overall execution time of the application task in the machine suite.

In general genetic algorithm, works on three operators natural selection, crossover and mutation [3, 6]. A genetic algorithm continuously tries to improve the average fitness of a population by construction of new populations. Quality of solution depends heavily on the selection of some key parameters like fitness function, population size, crossover probability and mutation probability. In this paper, we first introduce task scheduling problem having some specified characteristics, after that genetic approach is discussed in detail and the last section presents experiments and results.

Many parallel applications consist of multiple functional units. While the execution of some of the tasks depends on the output of the other tasks, others can be executed independently at the same time, which increases parallelism of the problem. The task scheduling problem is the problem of assigning the tasks in the multiprocessor system in a manner that will optimize the overall performance of the application, while guarantee the correctness of the result. Multiprocessor scheduling problems can be classified into many different categories based on characteristics of the program and tasks to be scheduled, the multiprocessor system, and the availability of information Multiprocessor scheduling problems may be divided in two categories: Static and dynamic task scheduling. A static or deterministic task scheduling is one in which precedence constraints and the relationships among the task are known well in advance while non-deterministic or dynamic scheduling is one in which these information is not known in advance or not known till run time.

II. PROBLEM DESCRIPTION

A static scheduling problem consists of three main components: A multiprocessor system, an application and an objective for scheduling. The multiprocessor system consists of a limited number of fully connected processors (P1, P2... Pm).All the processors are heterogeneous meaning thereby a task may take different execution time on each processor. An application comprises tasks and their dependencies on each other. It can be represented as a directed acyclic graph (DAG) [7, 9], G= (V, E, W), where the vertices set V consists of v non preemptive tasks, and vi denotes the ith task. The edge set E represents the precedence relationships among tasks. A directed edge eij in E indicated that vj can not begin its execution before receiving data from vi. W is a matrix of vxm, and wij in W represents the estimated execution time of vi on jth processor. Here we assume that communication costs do not exist. The main objective of the task scheduling is to determine the assignment of tasks of a given application to a given parallel system such that the execution time (or schedule length) of this application is minimized satisfying all precedence constraints.

III. GENETIC BASED APPROACH

Genetic Algorithms or evolutionary algorithms are developed by John Holland in 60s. They are random search based algorithm premised on the evolutionary ideas of natural selection and genetic. The basic concept of GA is designed to simulate processes in natural system necessary for evolution. They use three operators known as natural selection, crossover and mutation. Natural Selection [3] process forms a new population of strings by selecting strings in the old population based on their fitness values. Crossover [3] produces new chromosomes that have some parts of both parent's genetic material. Mutation [3] is a genetic operator that alters one ore more gene values in a chromosome from its initial state to produce new chromosomes.

A. Structure of Genetic Algorithm

Typically, a genetic algorithm consists of the following steps:
GA1: Initialization – initialize the population.
GA2: Evaluation – evaluate each chromosome using fitness function.
GA3: Genetic operations –Select the parent and apply genetic operators on them to produce new chromosomes.
GA4: Repeat steps GA2 and GA3 until termination condition reached.
From the above steps, we can see that genetic algorithms utilize the concept of survival of the fittest; passing “good” chromosomes to the next generation, and combining different strings to explore new search points.

B. Initial population (Structure of the chromosome)

Designing of chromosome structure is crucial for devising GA. We define our chromosome structure as a combination of two strings SQ and SP, whose length equal to the number of tasks. SQ (scheduling queue) maintains precedence constraints between tasks, and an entry in SQ represents a task to be scheduled. An entry in SP (scheduling processor) represents the processor the corresponding task is scheduled onto.

The details to generate a chromosome can be seen in following steps:
IP1: Select randomly a task from the entire entry tasks. Set this task as the first task in SQ.
IP2: Repeat step IP3 for (v-1) times.
IP3: Randomly select a task who is not in SQ and whose predecessors all have been in SQ, and add this task to SQ.
IP4: For SP part, randomly generate an integer number between 1 and m for each task in SQ and add it to SP.

C. Evaluation and Selection:

Roulette Wheel Mechanism In order to select good chromosomes, we define the fitness function as: F (i) = (maxFT-FT (i) +1/ (maxFT-minFT+1) (1) Where: maxFT and minFT is the maximum and minimum finishing time of chromosomes in current generation, respectively. FT (i) is the finishing time of the ith chromosome. Once the fitness values of all the chromosomes have been evaluated, we can select the higher fitness value chromosomes using the roulette wheel mechanism.

D. *Reproduction:* *Crossover and Mutation* Crossover

As our chromosome comprises two separate parts SP and SQ having different characteristics, for each part we employ different crossover policies. We randomly select one or the second part and apply two different crossover operators for these two parts. Details about crossover are given in following steps: CR1: Input the Crossover probability Pc.
CR2: Randomly select pairs of chromosomes and generate a float number (FLC) between 0 and 1 for each pair.
CR3: If FLC <= Pc, then repeat step CR4 to step CR5 Else directly reproduce those two chromosomes to the next generation.
CR4: Randomly generate two crossover points, p and q, between 1 and v and crossover flag CF between 0 and 1.
CR5: If CF=0 then rearrange the order of tasks in SQ between p and q of one chromosome according to the order of tasks of another chromosome, the rest of the two chromosomes are remained. Else exchange the part in SP between p and q of two chromosomes and the rest of the two chromosomes are remained.

Mutation

Mutation can be considered as a random alternation of the individual. We employ two policies to mute the chromosome as given in following steps:

MT1: Input the Mutation probability Pm.

MT2: For each chromosome, generate a float number (FLM) between 0 and 1.

MT3: If FLM <= Pm, then repeat step MT4 to step MT5 Else directly reproduce this chromosome to the next generation.

MT4: Randomly generate a mutation point p between 1 and v and mutation flag MF between 0 and 1.

MT5: If MF=0 then select randomly a location between location of the nearest immediate predecessor and that of successor of sqp. Then move sqp to this location. Else change randomly the processor of sqp between 1 and m as spp.

IV. EXPERIMENTS AND RESULTS

In our work, we implemented Genetic algorithm for solution of multiprocessor flexible task scheduling problem. Detail block diagram(Fig.1) represents the sequence of operations to get the desired results.We have compared our results with heuristic algorithm. For performance evaluation of our algorithm we generated some problems of varying sizes and solved them by both the algorithms. We assume that size of problem ranges from 10 to 50 with an interval of 5, there is no limit on the number of successors of each task except the exit task which does not have any successor, the execution time for each task is a random number between 5 and 25 and number of processors varies from 4 to 8 according to the size of problems. As we did not put any restriction over the number of successor a task may have, task graph may be much complicated. So, the problems we have chosen may be considered difficult in comparison to the kind of problems we normally see in literature, where a restriction on maximum number of successor tasks has been put.

The proposed genetic algorithm discussed in previous sections was implemented and evaluated on an application of college campus . In college campus application we considered various tasks relates to entities like Student , Teacher , Employee, Books etc. Results obtained are re shown in Table I. We set parameters for our Genetic Algorithm as: Population Size=25, Maximum Generations= 5000, Crossover Probability= .6 and Mutation Probability =.2.

Comparison of GA and Heuristic Algorithm.

Results obtained from our experiments are analyzed for following factors:

1) Quality of solution: Results of average schedule length of the GA is given in Table I . Results demonstrate that our proposed Genetic Algorithm is able to compete with heuristic based algorithms as far as quality of solution is concerned. As heuristics are biased towards certain characteristics of solution so they tend to search solution only in a small part of whole search space. It is also possible that they never explore a particular region of search space. Thus for some problems heuristics may give bad results also if they are not chosen carefully. On the other hand GA is a more powerful method as it searches simultaneously in many parts of search space. Because of mutation operator, change in region being searched, gives potential to GA to search in any part of the search space. Thus it is more likely to find a better or best solution.
2) Robustness and guarantee for good solution: During our experiments on GA we noted Average schedule lengths of populations emerging generations after generation Though we have shown results for problem size 10 to 50 in Table. 1, for each problem irrespective of its size we observe that average schedule length is continuously decreasing as more and more generations are evolving. This shows that Genetic Algorithm is robust and ultimately it will give us a good quality solution as quality of solution set is continuously improved. It also reveals that more generations we evolve; it is likely to have better quality in solution.
3) Effect of mutation probability on the performance of GA: As mutation is the key to change the region of search space, mutation probability may have dominating role in finding solutions of good quality. Thus, we repeated our experiments by fixing crossover probability and changing mutation probabilities from 0.05 to .40 and noted average schedule lengths. We done our experiments on the problem having size 25. we can observe the similar trend in the problems of all sizes. Table. 1 shows the further average of results, mixing the effect of all crossover probabilities which clearly shows that up till mutation probability is .20, increase in mutation probability leading to better results. After .20 results are fluctuating in a small range but normally are not better than that we obtained for .20. So, we have found best mutation probability for our set of problems as .20.So, we have found best mutation probability for our set of problems as .20. During the experiments, we have seen that for 28.8% problems GA gives lower schedule length, for 4.44% problems GA gives slightly higher schedule length while for 66.67% problems GA gives equal schedule length in comparison with HEFT. On an average, we analyzed that GA gives better results than the heuristic based algorithm and it is robust also as the average schedule length is continuously decreases as more and more generations evolve.

IV. CONCLUSION

A genetic algorithm based on principles of evolution found in nature for finding an optimal solution. Genetic Algorithm use random choices to guide themselves to the problem space and they are used for different kind of task scheduling problems. It continuously tries to improve the average fitness of a population by construction of new population .Quality of solution depends heavily on the selection of some key parameters like fitness function ,population size , crossover probability and mutation probability.

Task scheduling in multiprocessor system using genetic algorithm is an efficient way due to the characteristics of Genetic Algorithm as : quality of solution ,robustness and guarantee for good solution effect of mutation probability on the performance on Genetic Algorithm.

TABLE I
RESULTS OF GENETIC ALGORITHM

Problem size	PIN(0-4)					Average Schedule Length
	0	1	2	3	4	
10	80	90	107	101	93	94.2
15	111	96	98	107	104	103.2
20	130	132	118	115	117	122.4
25	116	118	135	145	167	136.2
30	165	145	187	125	160	156.4
35	172	190	165	169	180	175.2
40	190	187	175	169	180	180.2
45	225	230	190	185	199	205.8
50	222	220	232	218	217	221.8

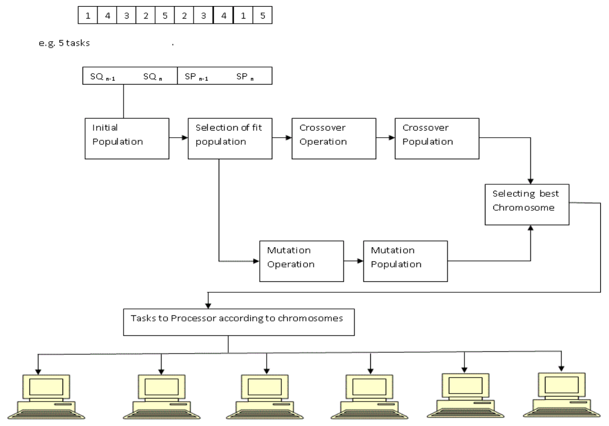

FIGURE I
BLOCK DIAGRAM OF PROPOSED SYSTEM

REFERENCES

[1] A.Chipperfied and P. Flemming. Parallel Genetic Algorithms. Parallel and Distributed Computing Handbook ,first ed.A.Y Zomaya.ed.,1,118-1,143. Mcgraw-Hill,New York,1996.

[2] B. Kruatrachue and T.G. Lewis. Duplication Scheduling Heuristic, a New Precedence Task Scheduler for Parallel Systems. Technical Report 87-60-3, Oregon State Univ., 1987.

[3] D.C. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning. Add.Wesley publishing,1989.

[4] H. EL-Rewini, T.G. Lewis, and H.H. Ali. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall . 1994.

[5] H. Topcuoglu, M.Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing.
IEEE Transactions on parallel and distributed System, Vol. 13, pp.260-274, 3, 2002.

[6] J.H. Holland. Adaptation in Natural and Artificial Systems. MIT Press, 1975.

[7] Jonathan l. Gross, Jay Yellen. *Handbook of Graph Theory*. CRC Press.

[8] M. Srinivas and L.M. Patnaik. Genetic Algorithms: A Survey. Computer vol. 27, pp. 1726, 1994.

[9] Martin Charles Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Second Edition, Elsevier, 2004.\

[10] M. R. Garey and D. S**.** Johnson. *Computers and Intractability:* A *Guide to the Theory of NP Completeness*, San Francisco, CA**,** W. H.Freeman, 1979.