Australian Journal of Basic and Applied Sciences, 5(12): 947-951, 2011
ISSN 1991-8178

An Improved Genetic Algorithm For Solving The Multiprocessor Scheduling Problem

[1]Imad Fakhri Al Shaikhli, [2]Ismail Khalil

[1]Dept. of Computer Science Kict- Iium Kuala Lumpur-Malaysia.
[2]Institute of Telecooperation Johannes Kepler University Linz.

Abstract: Multiprocessor Scheduling Problem (MSP) is an NP-complete optimization problem. The applications of this problem are numerous, but are, as suggested by the name of the problem, most strongly associated with the scheduling of computational tasks in a multiprocessor environment. Many methods and algorithms were suggested to solve this problem due to its importance. Genetic algorithms were among the suggested methods. In this research, sound improvements were done on one of the known papers (Hou, E.S.H., N. Ansari and H. Ren, 1994). Results show very good improvements in increasing the percentage of getting the exact solution as well as decreasing the number of generations needed to converge.

Key words: multiprocessor scheduling problem, genetic algorithms, list scheduler, optimization.

INTRODUCTION

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 difficulty of the problem is characterized by:

a. The topology of the task graph representing the precedence relations among the computational tasks.
b. The topology of the multiprocessor system.
c. The number of parallel processors.
d. The uniformity of the task processing time.
e. The performance criteria chosen.

In general, the MSP is computationally intractable even under simplified assumptions (Gorey, M.R. and D.S. Johnson, 1979; Stone, H.S., 1977). Because of its computational complexity, heuristic algorithms have been proposed to obtain suboptimal solutions to various scheduling problems (Kasahara, P. and S. Narita, 1985; Chen, C.L., C.S.G. Lee and E.H. Hou, 1988; Hellstrom, B. and L. Kanal, 1992). Most of the existing techniques are based on list scheduling (Wang, Q. and K.H. Cheng, 1991; Tao Yang and Apostalos Gerasoulis, 1993). In list scheduling, each task is assigned a priority and whenever a processor becomes available, a task with the highest priority is selected from the list and assigned to the processor. The objective of the present work was to do analyze one the known genetic algorithms (Hou, E.S.H., N. Ansari and H. Ren, 1994) that was proposed for solving the multiprocessor scheduling problem by applying it on a common test bed so as to be able to study its performance, in order to improve as a result of the insights gained. In order to formalize the MSP, first we define a homogeneous multiprocessor system and a parallel program. A homogeneous multiprocessor system composed of a set P
$P = p_1\ p_2\ p_m$ of m identical processors, each processor can execute at most one task at a time and task preemption is not allowed.

The parallel program is described by an acyclic graph D = (T, A). The vertices represent the set $T= (t_1\ t_{2,...,}\ t_n)$ of tasks and each arc represents the precedence relation between two tasks. An arc $(t_{i1}\ t_{i2}) \in A$ represents the fact that t_{i2} can start execution only after the completion of the execution of t_{i1}. A path is a sequence of nodes, $(t_{i1}\ t_{i2}\ \ t_{ik)}$, $1 < k < n$, such that t_{il} is an immediate predecessor of t_{il+1}, $1 < l < k$.
To every task t_i, there is an associated value representing its duration and we assume that these durations are known before the execution of the program.
Hence a schedule S is a vector:

$S= \{S_1, S_2,\dots,S_n\}$ where $S_j= \{t_{i1},t_{i2},\dots,t_{inj}\}$, i.e S_j is the set of the n_j tasks scheduled to P_j.

For each task t_{il} , l represents its execution rank in P_j under the schedule S. Further, for each task t_i we denote $P(t_i,S)$ and $r(t_i,S)$ respectively, the processor and the rank in this processor of t_i under the schedule S. The execution time yielded by a schedule is called make span.

List Scheduling:

In priority list scheduling, initially a list of tasks is ordered according to some given priority. At each scheduling step, the list scheduling heuristic schedules the highest priority task on the first idle processor. A generic procedure of list scheduling is described below:

a. priority -list = $(n_1;n_2,\dots,n_v)$ sorted in a descending order of task priority;

Corresponding Author: Imad Fakhri Al Shaikhli, Dept. of Computer Science Kict- Iium Kuala Lumpur-Malaysia.
E-mail: imadf@iium.edu.my

b. clock = 0;
c. while(priority-list is not empty) do remove the leftmost free task (t j);
d. schedule t j to an available idle processor;
e. end{while} clock = the next earliest time when a processor becomes available;

What is interesting about list scheduling is that the performance is within 50% of the optimum independent of the priority list as shown by Graham *et al*., An important question is what choices of priority-lists will consistently give schedules that are ’close’ to the optimum schedule?. Experimentally, Adam, Chanly and Dickson (1974) answered the above question by conducting an extensive empirical performance study of the Critical Path (CP) algorithm with other four priority list-scheduling algorithms. Their conclusion is that the CP heuristic is superior to other algorithms since it is near-optimal (i.e. within 5% of the optimal in 90% of random cases).

Genetic Algorithm for solving the MSP problem (Hou, E.S.H., N. Ansari and H. Ren, 1994):

There are many genetic algorithms for solving the MSP (Ahmad, I. and M.K. Dhodhi, 1996; Bauer, T., *et al*., 1995; Annie, S., *et al*., 2004; Shuang Zhou, E., Yong Liu, Di Jiang, 2006), one of them was (Hou, E.S.H., N. Ansari and H. Ren, 1994), which will be denoted in the rest of this article as (HAR). This algorithm will be analysed and discussed through the five main components of GA.

a. Representation: They represent the schedule as a list of computational tasks executed on number of processors, the order of the tasks in each list indicates the order of execution.

b. Initial Population: for generating the initial population the following steps were taken:

Compute the height for each task in the task graph, where : height

$$Ti = \begin{cases} 0 & \text{if } Pred(Ti)\ ; \\ 1 + maxTj \in Pred(Ti) height(Tj) & \text{otherwise} \end{cases}$$

Partition the tasks in the task graph into different sets G (h), (G (h) is defined as the set of tasks with height h), according to the value of the height.

Each of the first p-1 processors, do the next step.

Each set G(h):

- Set NG(h) to the number of tasks in G(h).
- Generate a number r (randomly) between 0 and NG(h).
- Pick r tasks from G(h) and assign them to the current processor.

Assign the remaining tasks in the sets to the last processor.

c. Fitness Function (FF), FF(S) = C_{max} - FT(S), where C_{max} is the maximum finishing time observed so far(up to the previous generation),

FT(S) = $\max_{Pj}$ ftp(Pj), where ftp(Pj) is the finishing time for the last task in processor Pj.

d. Genetic Operators

Crossover: They used the standard one point crossover but the important thing here is the selection of the crossover site(the position where we can do the crossover), which have been chosen so that it satisfies the following conditions :

- The height of the tasks next to the crossover site is different.
- The height of the tasks immediately in front of the crossover site is the same.

Mutation for a schedule S does the following: pick (randomly) a task Ti. _ search S for a task Tj with the same height. Form a new schedule by exchanging the two tasks Ti, Tj in S .

e. Parameters; they have used two sets of parameters : POPSIZE = 10 , 20, Crossover probability = 1.0 , 0.5, Mutation probability: = 0.05 , 0.005, Number o f generations = 1500 , 2000

The Test Bed:

We have used DAGs (Directed Acyclic Graphs) of sizes (number of tasks) =12, 16, 20, 24, 28, 32, 40 and 48, with number of processors = 2,3,4,5,6,8, 10 and 12 (higher number of processors for DAGs with bigger sizes), with number of edges in each DAG = same, double, triple and 4 * (the number of tasks) .In all, we have 92 DAG’s.

We followed the method of (Yu-Kwong, Kwok and I. Ahmad) for generating these DAG’s: Suppose that the optimal schedule length of a graph and the number of processors used are specified as SL_{opt} and P respectively. Then for each processing element i, we randomly generate a number x_i from a uniform distribution with mean v/p (v the number of tasks). The time interval between 0 and SL_{opt} of the processing element i is then randomly partitioned into xi sections.

Each section represents the execution time for one task. Thus, x_i tasks are scheduled to processor element i with no idle time slot. In this manner, v tasks are generated so that every processor has the same schedule length.

To generate an edge, two tasks n_a and n_b are randomly chosen such that
$FT(n_a) < ST(n_b)$ (FT stand for the finishing time and ST for the starting time).
The edge is made to emerge from n_a to n_b.

Parametric Study:

As part of the analysis of this algorithm, we will follow the method proposed in (Taha, Imad Fakhri, 2011). To study the performance of the algorithm, a parametric study was done to know the suitable set of parameters by running the algorithm on the same DAG and changing the probabilities of crossover and mutation as follows: prob. of crossover from the set 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9 prob. of mutation from the set 0.00121, 0.00156, 0.00175, 0.0026, 0.0034, 0.006, 0.01, 0.16, 0.2, 0.25. The result is shown in table 1:

Table 1: results of the parametric study of HAR.

Prob. of crossover	Prob. of mutation
0.73	0.3

Experimental Results Before Improvements:

After developing the algorithm and generating the DAGs with known optimal schedule length, experiment was done by executing the algorithm three times on each DAG and then by taking the best out of the three, the following parameters were fixed throughout the experiment: number of generation (NOG) = 100
population size (POPSIZE) = number of tasks in the DAG
probability of crossover for HAR = 0.73
probability of mutation for HAR = 0.2

The results are shown in table 2.

Table 2: experimental results on HAR.

No. of tasks	% of exact solution	No. of generations	% of Deviation	Execution time
12	11	29.7	17	6.1
16	27	28	19	20
20	9	45.18	22	44.5
24	27	47.27	25	69.6
28	9	53.81	16	135.5
32	18	59.54	14	163
40	30	55.5	4	307
48	30	70.6	3	361.8
Avg.	20	48.7	15	138.4

From the results in table 2 we can notice the following:

a. HAR failed to find the exact solution in 80% of the cases.

b. The deviation from the exact solution was 0.15. The most important question now is: why this unsatisfactory performance of HAR though it is not a blind search?

To answer this question we used the analysis method proposed in (Taha, Imad Fakhri, 2011) and found the following points:

- It cannot capture the whole search space (Bauer, T., *et al*., 1995) because it is possible to find legal schedules that violates the height condition but still maintain the interprocessor precedence relations and it is therefore, unfortunately, possible that a schedule that violates the height condition could be an optimal schedule.
- The initialization procedure is biased because the probability for a processor with higher index to have tasks assigned to it is lesser than the probability for a processor with lower index. This means that the distribution of the tasks among the processors is not uniform and that's why when we have higher number of processors, the performance will decays as some of the processors might not get any task assigned to it. We have seen this when we tried to execute HAR on DAG's of 32,40 and 48 tasks with number of processors 8,10 and 12. There were some problems because many processors will be empty because of the initialization.
- We have noticed that when the differences between the execution time of some tasks in a DAG is high, then HAR fails to get even a good solution because the difference between the execution time of the processors will be high while in the real case the execution time of the processors should be (usually) comparable.

The Proposed Algorithm:

After analyzing HAR using the method of (Taha, Imad Fakhri, 2011) and studying the drawbacks, the following improvements will be suggested:

a. New calculation of the height (Bauer, T., *et al*., 1995): To overcome the problem of not capturing some of the potential solutions which violate the height condition but still maintains the precedence relation, we used the following equation for calculating the height :

height0(Tj)=random integer$\begin{cases} \text{maxTi2PRED(Tj)height(Ti)} + 1; \\ \text{minTm2SUCC(Tj)height(Tm)} - 1 \end{cases}$

b. Using initialization of (Ahmad, I. and M.K. Dhodhi, 1996): Here they use a heuristic algorithm for the initialization which depends on the level of the tasks and that it gives a good initial population. So we tried to use this initialization after changing it to suit HAR as follows :

Call the same initialize from (Ahmad, I. and M.K. Dhodhi, 1996) to generate the initial population.

Call the list scheduler used in (Ahmad, I. and M.K. Dhodhi, 1996) for calculating the execution time for each schedule.

Use the output (schedules) of the list scheduler above as an initial population for HAR.

c. Self-fixer operator: To overcome the problem mentioned above which happens when there were big gaps between the execution time of the processors, we have introduced a new operator which can adjust the schedule length by doing the following: Firstly the operator will work according to a probability Ps. after choosing a schedule(chromosome) we will do the following :

choose the processor pi with the highest execution time and then
choose the task Ti with the highest execution time within this processor.
choose a task Tj randomly from the processor pi (other than Ti).
Choose a processor pj randomly (other than pi).
Assign Tj to pj (put it in a proper sequence).
Remove Tj from pi

Experimental Results After Improvement:

After doing the improvements mentioned above, we did the same experiments again on the same DAG's and results are shown in the table below:

Table 2: experimental results of improved HAR.

No. of tasks	% of exact solution	No. of generations	% of Deviation	Execution time
12	77	1.6	0.5	5.3
16	54	23.45	1	16.8
20	45	19.18	0.6	39
24	36	12.18	1	53.3
28	36	27.72	1	97.6
32	36	30.38	1	128
40	53	25	1	284.6
48	38	13.3	1	417.4
avrg	45	19.1	1	150.25

From the above table we can see that the performance of the improved HAR is now much better than the old HAR.

Conclusions:

A careful analysis were done on the genetic algorithm of (Hou, E.S.H., N. Ansari and H. Ren, 1994) using the method of (Taha, Imad Fakhri, 2011) to have insight of the algorithm and to find the cases and circumstances where the algorithm fail. Then, major improvements were suggested base on the analysis done. Experimental results show that the improved algorithm outperforms the original one where the percentage of getting the exact solution increased from 20% to 45% and the number of generation needed to find the exact solution decreased by 40%.

REFERENCES

Adam, T., K.M. Chandy and J.R. Dickson, 1974. A comparison study of list schedules for parallel processing systems. CACM, 17(12): 685-690.

Ahmad, I. and M.K. Dhodhi, 1996. Multiprocessor scheduling in a genetic paradigm. Parallel Computing, pp: 395-406.

Amir, M.R. and Mojtaba Rezvani, 2009. "A Novel Genetic Algorithm for Static Task Scheduling in Distributed Systems", International Journal of Computer Theory and Engineering, 1(1): 1793-8201.

Annie, S., Wu, Han Yu, Shiyuan Jin, Kuo-Chi Lin and Guy Schiavone, 2004. "An Incremental Genetic Algorithm Approach to Multiprocessor Scheduling", IEEE Transactions on Parallel and Distributed Systems, Vol.15, No. 9, On page(s): 824 – 834, ISSN: 1045-9219, INSPECAccession Number:8094176, Digital Object Identifier: 10.1109/TPDS. 38, 13.

Bauer, T., O. Ejlerse and N. Holm, Genetica, 1995. A genetic algorithm for multiprocessor scheduling. Master's thesis, Institute of Electronic systems, AALBORG University, Denmark.

Chen, C.L., C.S.G. Lee and E.H. Hou, 1988. Efficient scheduling algorithms for robot inverse dynamics computation on a multiprocessor system. IEEE Trans. System,Man,Cybernetics, pp: 729-743, Dec.

Gorey, M.R. and D.S. Johnson, 1979. Computers and Intractability. freeman, New York.

Graham, R.L. Bounds for certain multiprocessing anomalies. Bell System Tech. J., pp: 1563-1581.

Hellstrom, B. and L. Kanal, 1992. Asymetric meanfield neural networks for multiprocessor scheduling. Neural Networks, pp: 671-686.

Hou, E.S.H., N. Ansari and H. Ren, 1994. A genetic algorithm for multiprocessor scheduling. IEEE, Trans. Parallel and Distributed Systems, 5(2): FEb.

Kasahara, P. and S. Narita, 1985. Parallel processing of robot arm control computation on a multimicro processor system. IEEE Trans. J. Robotics Automatic, RA., 1(2): 104-113.

Shuang Zhou, E., Yong Liu, Di Jiang, 2006. "A Genetic-Annealing Algorithm for Task Scheduling Based on Precedence Task Duplication", CIT, Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, Page: 117, ISBN: 0-7695-2687-X, IEEE Computer Society Washington, DC, USA

Stone, H.S., 1977. Multiprocessor scheduling with the aid of network flow algoriths. IEEE Trans. Software Eng., SE., 3(1): 85-93.

Taha, Imad Fakhri, 2011. A practical method for the analysis of genetic algorithms. Journal of advanced computer science and technology research (JACSTR), pp: 1-9.

Tao Yang and Apostalos Gerasoulis, 1993. List scheduling with and without communication delays. Parallel Processing, pp: 1321-1344.

Wang, Q. and K.H. Cheng, 1991. List scheduling of parallel tasks. Information Processing Letters, pp: 291-297.

Yu-Kwong, Kwok and I. Ahmad, Eficient scheduling of arbitrary task graphs to multiprocessors using a parallel ga. J. of Parallel and Distributed Computing, pp: 58-77.