Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CI-Sched 2007)

Static Task Scheduling Using Genetic Algorithm and Reinforcement Learning

Mohammad Moghimi Najafabadi
Mustafa Zali
Shamim Taheri
Fattaneh Taghiyareh
ECE Department
University of Tehran
Tehran, Iran
Email: {m.moghimi,zali,s.taheri}@ece.ut.ac.ir, ftaghyiar@ut.ac.ir

***Abstract*— Task scheduling in a multi processor system is defined as assigning a set of tasks to a set of processors. The goal is to minimize the execution time while meeting a set of constraints. A wide variety set of deterministic and heuristic methods are proposed to solve the problem. The main problem is that the proposed methods cannot deal with big search spaces and cannot guarantee to find the optimal solution. In this research a novel approach based on reinforcement learning and genetic algorithm is proposed. Being divided using genetic algorithm, the smaller problems can be solved with reinforcement learner scheduler. The result of the method is a set of task processor pairs. Simulation results in standard problem set show that the method outperforms some studied GA based scheduling methods.**

I. Introduction

With increasing use of parallel processing in different fields of applications, the optimum task scheduling such that the time and cost limitations would be met is valuable. In this case, the tasks scheduling is defined as the assignment of some tasks to a limited number of processors [1]. The final goal is to perform this scheduling in a way that the final time and cost of the execution would be minimized and all the constraints, such as prerequisites, are satisfied [2]. In this problem, the scheduling is static, in such a way that the information on the tasks, the way they are related to one another, the execution time of each of them and the number of processors are known before execution. However, there are a variety of problems in the field of dynamic scheduling in which some information is obtained in the tasks execution time[3].

The nature of the scheduling problem, even in the limited cases, is NP-complete [4]. Nevertheless, many heuristic, efficient algorithms with acceptable results are proposed as its solution. Some simplified cases of the problem can be solved in polynomial time [4], [5], [6].

The solutions include various domains of algorithms such as classical AI methods(A^*), clustering methods, genetic algorithms and machine learning techniques(RL).

Since the scheduling problem is NP-complete and the use of reinforcement learning method to find the optimum solution for large state space problem is time-consuming, a combination of genetic algorithm and reinforcement learning has been used to solve the problem.

The reinforcement learning method learns the search policy on the state space. The learning policy determines what should be done in each case. In the process of learning, for each action a reward signal is achieved. The aim of the agent is to find a policy, in the execution time, to maximize the reward in return for the actions [13]. In the scheduling problem, the learned policy determines what the next action should be to satisfy the timing measurement constraints in the best case.

To schedule a number of tasks in a multi processor system using the reinforcement learning method, very large state space is required. Searching the large state spaces is not possible, due to the amount of memory occupied and the time needed to visit all the states; therefore genetic algorithm is used to find the optimum subproblems. Genetic algorithm may satisfy the timing constraints in finding such subproblems.

This paper is organized as follows: A review of the previous related work is presented in section II. Sections III and IV take a glance at the reinforcement learning methods and genetic algorithm. In section V the design method of the problem solving algorithm for the combination of the two methods has been discussed, and in the last two sections the implementation results and the final conclusion have been presented.

II. Related work

The task scheduling problem in multi processor systems is NP-complete and many semi-optimum solutions have been offered for it. The correctness of the optimum solution cannot be verified since time-consuming searching methods are required to fulfill the task. Wide studies have been performed to solve this problem using heuristic solutions. Heuristic methods are open fields to propose complementary discussions. One of the problems of the heuristic solutions is that they cannot be applied to the solution of different problems. These solutions are appropriate only for solving the problem for which they have been designed. On the other hand, many efforts have been made to solve this problem using reinforcement learning method. Solutions based on the reinforcement learning method

1-4244-0704-4/07/$20.00 ©2007 IEEE

that have been already offered cannot be applied to the problems with large state space.

The offered solutions include a wide range of algorithms, as Ahmad and Kwok have made use of the modified A^* algorithm to solve the task scheduling problem[7]. Simulated Annealing is a gradient-based method that can be used to optimize the scheduling problem[14]. Genetic algorithm is also widely used to search in nearly optimum solutions[9], [11]. Correa et al. have used a direct acyclic graph representation to present their crossover[9]. They used an integer string representation to encode their chromosomes such that the number of the processors in each chromosome is encoded and the tasks assigned to each processor vary from one chromosome to the other. They also proved that this representation cannot lead to a complete set of problem solutions. They offered a full search genetic algorithm(FSG) which consists of a string representation capable of searching in all possible assignments. Liu et al. have shown the two phase scheduling effect and claimed that it would work better than the single phase scheduling algorithm [15]. Generally, they consider that at first there are infinite number of identical processors connected completely to each other. All the tasks in a cluster are performed on the same processor. During the scheduling, if the number of the clusters is less than or equal to the number of the processors, all clusters are executed. In the single phase method, the number of processors is an initial parameter given to the algorithm. They claimed that the two phase method is faster and more efficient than the other one. Their two phase method consists of three parts: Integrating the clusters, assignment of the processors and local scheduling.

Zomaya et al. have used an integer string matrix [11]. They gathered the ideas in the initial population production of genetic algorithm and studied on how the efficiency of the genetic algorithm varies by changing the parameters. They used a reordering crossover, which they called RRANDS(Replace, Release and Save). A set of algorithms such as the height algorithm, the priority algorithm and the heuristic algorithm have been also used.

Ahmad et al. have made use of direct acyclic graphs in homogeneous multi processor systems and a problem space genetic algorithm, which combines a list of heuristic scheduling and a genetic algorithm for static scheduling [16].

Wu et al. offered a representation in which each chromosome consists of a set of cells [11]. Each cell is a processor-task pair, (t, p), in which task t is assigned to processor p. In their representation, the number of cells may vary from one chromosome to the other, and hence the length of chromosomes are different. Chromosome is read from left to right, therefore the order in these cells is the same as the order in which tasks are performed on each processor. While the ordered sets are read from left to right, when two tasks are performed on the same processor, the first task should be the prerequisite of the second one. Otherwise, that chromosome would be punished.

Zhang et al. used reinforcement learning method to solve the task scheduling problem in which a critical path is considered first and the number of scheduling conflicts decrease continuously [12].

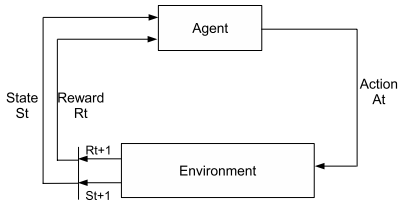

Fig. 1. Reinforcement Learning Framework

III. Reinforcement Learning

Reinforcement learning framework is shown in Figure 2. In the learning cycle, the agent receives information from environment, chooses an action corresponding to it and performs the action. As a result of it, a reward signal is given to the agent. The final goal of the agent is to maximize the sum of the rewards in a period of time. The reinforcement learning, unlike the supervised learning methods in which the agent is told what to do, finds the best action in each situation just by receiving rewards. In all different methods of reinforcement learning, a mapping from the current state of the agent to an action exists. This mapping is called policy and determines what to perform in each state. The agent should optimize his policy based on the received reward. There is always a trade-off between exploration and exploitation in reinforcement learning. The agent should make use of available information to maximize its reward and search for better actions in the current situation. A difficult problem in reinforcement learning is that exploitation and exploration should be performed successfully in balance.

Current reinforcement learning methods act in accordance with discrete Markov chain model. In these methods, the agent and the environment send and receive information in discrete periods of time. In each period, the agent observes the environment in state s_t and performs the action a_t. To reply to this action, the environment goes to state s_{t+1} and gives the real number r_{t+1} to the agent as a reward [13].

One of the reinforcement learning implementations, called Q-learning, is used here. In this method, a function maps the value of an ordered pair of state-action to a real value. This function is called Q^*. If $Q^*(s, a)$ is the expected value for state s, with action a, its approximate value can be obtained by the following relation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (1)$$

in which γ is the discount factor, α is the learning rate and r_{t+1} is the reward in time $t+1$. If each action in each state is performed in infinite steps, the Q values will converge to the Q^* values.

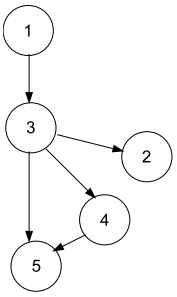

Fig. 2. Tasks Dependencies Graph

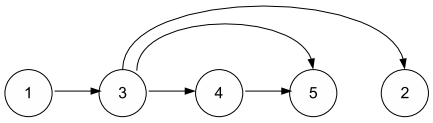

Fig. 3. An Example of Task Topological sort

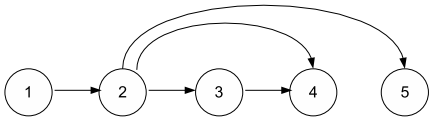

Fig. 4. An Example of Tasks Relabeling

IV. Genetic Algorithm

Genetic Algorithm has come into existence with the inspiration of biological comparative systems and the selection concept in genetic evolutions. Hypothetical responses with constant lengths are encoded as a vector. These vectors are called chromosome. The initial population of these chromosomes are determined randomly. They evolve through consecutive generations. In each generation, the fitness of each chromosome is evaluated which is a factor for improvement rate of the target function. The next generation will be produced through the processes of selection, crossover and mutation in the previous generation chromosomes.

The fitness of each chromosome determines the probability of the combination of it. The crossover operator mixes the information of a selected pair of chromosome by selecting random parts of each, and makes them appear in the next generation. Because of the presence of random parameters, the produced children from the crossover operator may have a fitness less than or greater than their parent chromosomes. The mutation is used to help keep the diversity in population. Mutation of random changes is in some parts of the chromosomes [18].

V. Methodology

As indicated in the introduction, the scheduling problem is NP-complete and hence the appropriate solution is applying to intelligent methods. To solve the problem, the reinforcement learning method is used. It can be proved that the state space of the problem is very large and therefore cannot be solved by reinforcement learning method. So the idea of dividing the problem into smaller subproblem is proposed.

The tasks in the scheduling problem are only dependent on their few previous tasks and dependencies are not too wide. Thus, the problem can be divided into smaller subproblems and each subproblem can be solved individually. To divide problem into independent subproblems, a topological sort of the tasks is required. For example, consider the graph in Figure 2. Figure 3 shows a topological sort of this graph.

Since there are no cycles in the tasks dependency graph, it is always possible to find such an order. After constructing the dependency graph, the number of each task can be changed in a way that corresponds to the order presented in dependency graph as indicated in Figure 4.

Now a subproblem cab be defined as a continuous set of tasks in the form of t_i to t_j. Figure 5 shows how to devide problem into subproblems.

Then smaller subproblems can be solved. It is evident that dividing is not appropriate for the small problems such as figure 3, but for some problems, e.g. Figure 7, suboptimum solutions can be gained by dividing approach.

To find the best way of dividing, genetic algorithm is used. The problem representation as a single chromosome is as follows: A chromosome, whose length is equal to the number of tasks, is composed of some zeros and ones, in which the ones represent the places where the problem has been divided. For instance, the chromosome of Figure. 2 can be represented in the form of Figure 6.

To divide the problem into subproblems the single point crossover has been used. This kind of crossover is meaningful since if two children also benefits from high fitness, it is most probable that their children also benefit from high fitness. This is true based on the fact that if a suboptimum solution can be achieved for the left and right sides of the dividing point, the whole problem also has a suboptimum solution most probably. The mutation point is selected by a uniformly distributed probability and the mutation changes the bit located there.

To solve the optimum subproblem, as stated before, making use of intelligent methods seems suitable. Here, to solve the subproblems scheduling, the reinforcement learning method has been used. To define a reinforcement learning problem, it is necessary to determine the two main elements; the agent

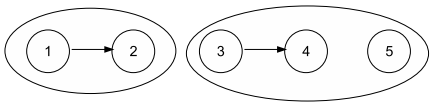

Fig. 5. Graphical Representation of Subproblems

00100

Fig. 6. The Representation of the Tasks in a Chromosome

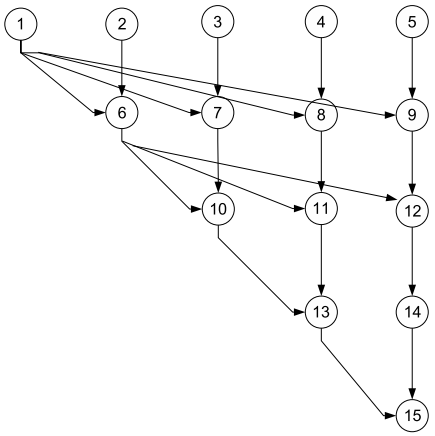

Fig. 7. An Example of Suitable Problem

and the environment. In this problem, the agent performs as scheduler and gets rewards from the environment as time passes. In addition to these two elements, the scheduling policy, reward and the value function should be defined.

Policy is the way of scheduling and here, by a constant probability, is the best possible action in the current state or selecting an action randomly. In this problem, the reward, which determines the final goal, is considered as the negative of the execution time.

The value function (Q^* function) represents the efficiency of each action in the current state and its approximate value is obtained by Eq. 1. The definition of the reinforcement learning problem would be completed when the space state of the agent and the reward function are determined. Here, the state space is the set of all the possible states of tasks related to the processors.

If the number of the processors is $|P|$, then each tasks has $2^{|P|}$ possible states. So when $|T|$ tasks considered, the size of the state space would be $2^{|P|\times|T|}$. Assuming that there are ten tasks and four processors, the state space size will be 2^{40} which cannot be solved by the reinforcement learning method. Therefore, dividing the problem into subproblems is necessary. To make better use of memory, the state space may be stored in a bit string in which there is a substring in correspondence with each task. In each substring, there exists a bit for each processor and the value of that bit shows whether the task has been assigned to that processors. To realize the above statements, an example will be explained.

Assume the processors set to be $P = p_1, p_2$ and the tasks set to be $T = t_1, t_2, t_3, t_4$ and that the tasks dependencies obey the graph. First the ordered pair (t_1, p_1) and then (t_2, p_2) are selected. Therefore, the first state would be $(00000001)_2$ and the second state would be $(0100001)_2$.

The legal actions in each state are to assign one task to one processor. For each action, the following requirements should be checked:

- That task should not has been assigned to that processor
- The previous actions of that action should be performed

Therefore, the maximum legal action in each state are $|T| \times |P|$.

TABLE I
THE SELECTED PROBLEM FOR EXPERIMENTS

p	$no.$	Reference	Name	t_{min},t_{max}
P1	15	Tsuchiya, et al[19]	Gauss-Jordan alg. (CD=25)	40,40
P2	15	Tsuchiya, et al[19]	Gauss-Jordan alg. (CD=25)	40,40
P3	14	Tsuchiya, et al[19]	LU Decomposition(CD=20)	10,50
P4	14	Tsuchiya, et al[19]	LU Decomposition(CD=20)	10,50
P5	15	Kruatrachue et al[20]	(CD=212)	80,101
P6	16	Wu and Gajski[21]	Laplace (CD=40)	80,100
P7	16	Wu and Gajski[21]	Laplace (CD=40)	80,100

VI. RESULTS

The experiments is compared to the results obtained from [11], [17]. The list of selected problems is available in Table I.

The parameters for GA such as mutation rate and way of crossover is set by trial an error. Our experiments show that setting mutation rate equal to 0.1 and using single cross over is the best configuration for the proposed genetic algorithm.

Genetic algorithm has been performed 50 times for two population, one with 100 and one with 400 individuals, and for 100 generations. The mutation probability is considered 0.1 for both instance and the fitness function is the negative of the execution time of each of the subproblems. The selecting method is considered as a uniformly distributed probability. The crossover, as stated above, obeys the single point model. In solving subproblems with reinforcement learning method to set a balance between the exploration and exploitation, the search probability is $\epsilon = 0.1$ and the learning rate and discount factor are $\alpha = 1$ and $\gamma = 0.8$, respectively.

The results of the simulation and the comparison to the available methods are shown in Table II. For each problem the minimum execution time, the time when the last task is done, found by our method and other methods in [1] and [17] indicated in the II. For each problem, the exact information about tasks dependency graph, duration of tasks and communication delay of transmitting the result of executed tasks is available. Note that in the three last problems $P5$ to $P6$, convergence is obtained slower than the four ones because that problems dependency graph have more complex structures.

VII. CONCLUSION

In this problem, a new combinational method for solving the scheduling problem based on genetic algorithm and reinforcement learning has been offered. This method is novel because of the presentation of the idea of dividing the main problem into subproblems and also due to the use of the reinforcement learning method. As the results that the simulation

TABLE II
THE COMPARATIVE EXPERIMENTAL RESULTS OF EXECUTION TIMES

Problem	Best[1]	Best[17]	Best of Proposed Method
P1	300	300	300
P2	420	400	400
P3	260	260	260
P4	360	330	340
P5	438	438	444
P6	760	760	760
P7	1035	1040	1025

demonstrates, the proposed method is more efficient compared to the others.

The idea of solving the problem using genetic algorithm can be applied to different problems due to the existence of many problems that can be solved locally. To solve the subproblems intelligent algorithms are required, since their exact solution has an exponential order. It is important to note that this method, as the others, does not guarantee to find the best solution. The future work is to find a method that is capable of solving the dividing problem faster and discover a better way of defining the problem as a reinforcement learning one.

REFERENCES

[1] Mehdi Salmani Jelodar, S. Najmeh Fakhraie, Faezeh Montazeri, Seid Mehdi Fakhraie, Majid Nili Ahmadabadi *A representation for genetic-algorithm-based multiprocessor task scheduling*, IEEE World Congress on Computational Intelligence, WCCI'06, pp. 1044-1051, Vancouver, BC, Canada, Jul. 2006.

[2] Faezeh Montazeri, Mehdi Salmani Jelodar, S. Najmeh Fakhraie and S. Mehdi Fakhraie, *Evolutionary Multiprocessor Task Scheduling*, Intl. Symp. on Parallel Computing in Electrical Engineering, IEEE PARELEC 2006, pp. 68-76, Poland, September, 2006.

[3] B.Hamzezadeh, L. Y. Kit and D. J. Lilja, *dynamic task scheduling using online optimization*, IEEE Trans. Parallel and Distributed Systems, vol. 11, no. 11, pp. 1151-1162, Nov. 2000

[4] P.Chretienne et al.(Eds), *cheduling Theory and its Applications*,New Yourk: Wiley, 1995.

[5] M,R Garey, D. S. Johnson,*Computers and Intractability: A Guide to the Theory of NP Completeness*, New York: W.H.Freeman and Company, 1979.

[6] H.El- Rewini, T. G. Lewis, H. H.Ali,*Task Scheduling in Parallel and Distributed Systems.*, Englewood Cliffs, NJ: Prentice-Hall, 1994.

[7] Y. Kwok, and I. Ahmad,*On multiprocessor task Scheduling Using Efficient State Space Search Approaches*,Parallel and Distributed Computing, vol. 65, pp. 1515-1532, 2005.

[8] K. Ramamritham,*Allocation and scheduling of precedence related periodic tasks*, IEEE Trans. Parallel and Distributed Systems, vol. 6, no. 4, pp. 412-420, April 1995.

[9] R.C. Correa, A. Ferreira and P. Rebreyend,*Scheduling Multiprocessor Tasks with Genetic Algorithms*, IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 8, pp. 825-837, August 1999.

[10] E.S. H.Hou, N. Ansari and H.Ren ,*A genetic Algorithm for multiprocessor scheduling*, IEEE Tran. Parallel and Distributed Systems, vol. 5, no. 2, pp. 113-120, Feb. 1994

[11] A.Y. Zomaya, C. Ward, and B.Macey,*Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues*,IEEE Trans. Parallel and Distributed Systems, vol. 10, no. 8, August 1999.

[12] Wei Zhang, Thomas G. Dietterich,*A Reinforcement Learning Approach to job-shop Scheduling*, Conf. on Artificial Intelligence, vol. 2 , pp. 1114-1120, 1995.

[13] Sutton, R.S., Barto, A.G.*Reinforcement Learning: An Introduction*, MIT Press, 1998.

[14] M.DiNatale and J. A. Stankovic,*Applicability of simulated annealing methods to real-time scheduling and jitter control*,Proc. 1995 IEEE Int. Conf. Real-Time Systems Symposium, pp. 190-199, 1995.

[15] J.Liu, P. H. Chou, N. Bagherzadeh and F. Kurdahi,*for power-aware systems*, In Proceedings of the 9th International Symposium on Hardware/Software Codesign, pp. 153-158, Copenhagen, Denmark, 2001.

[16] I. Ahmad and M. K. Dhodhi,*Short Communication Multiprocessor Scheduling in a Genetic Paradigm*, Parallel Computing, vol. 22, pp. 395-406, 1996.

[17] A. S. Wu, H. Yu, S.Jin, K. Lin and G. Schiavone,*An Incremental Genetic Algorithm Appraoch to Multiprocessor Scheduling*, IEEE Trans. Parallel and Distributed Systems, vol. 15, no. 9, pp. 824-834, September 2004.

[18] A. E. Eiben, J. E. Smith, Agoston E. Eiben, J. D. Smith, Introduction to Evolutionary Computing, Spring 2003.

[19] T.Tsuchiya, T. Osada, and T. Kikuno,*Genetic-based mutiprocessor scheduling using task duplication*, Microprocssors and Microsystems, vol. 22, pp 197-207, 1998.

[20] B. Kruatrachue and T.G Lewis,*Grain size determination for parallel processing*, IEEE Software, vol. 5, no. 1, pp. 23-32, 1998.

[21] M.Y Yu and D.D Gajski,*Hypertool: a programming aid for message passing system*, IEEE trans. Parallel and Distributed Systems, vol. 1, no. 3, pp 330-343, 1990.