ISSN No. 0976-5697

Volume 3, No. 1, Jan-Feb 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A New Scheduler for Relative Threads on Multi Processors by Randomize Algorithm

Morteza Babazadeh*
Department of computer,Babol-branch,Islamic Azad University, Babol, Iran
Morteza_babazade@yahoo.com

Shirin Hatami
Young research club, Islamic Azad University, Babol-branch, Babol, Iran
Shirin_hatami@yahoo.com

Abstract: In this paper, we first study the existent method of scheduling multi processors, then we try to propose a new scheduler for multi processors with randomize algorithm, in the situation that the processes are related with each other. Randomize algorithm is a method like genetic algorithm but without any cross over function. The proposed method considers all precedence limitations and then tries to observe priority of the processes. The basic ability of this algorithm is in considering priority for tasks. Proposed algorithm can be run in an acceptable time for huge amount of tasks**.**

Key Words: Multiprocessor, scheduling, Randomize Algorithm, Genetic Algorithm

I. INTRODUCTION

Now day's multiprocessors have a several usage in hard problems, so scheduling is one of the most important concepts for them. In multi processors the huge amount of process and threads is existence for execution that can be relative or irrelative. For scheduling irrelative processes we have many classic methods like LPT, RLPT, SPT, LSPT [1], DFS [3], SMP [4], that we do not survey them in this paper. However several methods are existence for scheduling relative processes too, such as space sharing.

In line-base architecture this fact that two relative processes are exist on the same processor or not is effecting on the execution time because we have a global memory and each processor has own cache memory. for execute two processes on two suppurate processor, the first processor have to rewrite the results on global memory , then the second copy them on the cache [2] generally, the relation of the processes can be shown by a DAG (Directed Acyclic Graph) like G=< V,E,T,C > that :V is the set of processes E is the set of Relations in graph , Value of $t \in T$ represent the execution time of $n_i \in$ V and $c_{ij} \in$ C is determine the cost of e_{ij} . The cost of e_{ij} is zero in situation that v_i and v_j are executing on the same processor. If we have a relation line e_{ij} then n_i is a predecessor for n_j and n_j is an immediate successor for n_i . the process that has not any successor named an output process. An example of process and thread and the relation between them is shown is table 1.

II. CONCEPT OF PRECEDENCE AND PRIORITY

In the graph that we created for represent relations between process and threads if e_{ij} be a relation from n_i to n_j then n_i has precedence to n_j. So in figure 1, process A has precedence to B, in the other word a scheduling that lets B to execute before A is completely invalid. Each process can be run after running whole of its precedence.

Tab.1 sample

n_i	t_i	e_{ij}	c_{ij}	e_{ij}	c_{ij}
T_0	15	$E_{0\,3}$	3	$E_{4\,9}$	2
T_1	6	$E_{0\,4}$	1	$E_{5\,6}$	1
T_2	2	$E_{0\,5}$	1	$E_{7\,8}$	3
T_3	3	$E_{1\,3}$	2	$E_{9\,10}$	2
T_4	9	$E_{1\,5}$	2		
T_5	4	$E_{2\,3}$	2		
T_6	1	$E_{2\,4}$	2		
T_7	9	$E_{2\,7}$	3		
T_8	4	$E_{3\,5}$	2		
T_9	9	$E_{3\,7}$	1		
T_{10}	2	$E_{4\,5}$	1		
T_{11}	10	$E_{4\,6}$	1		

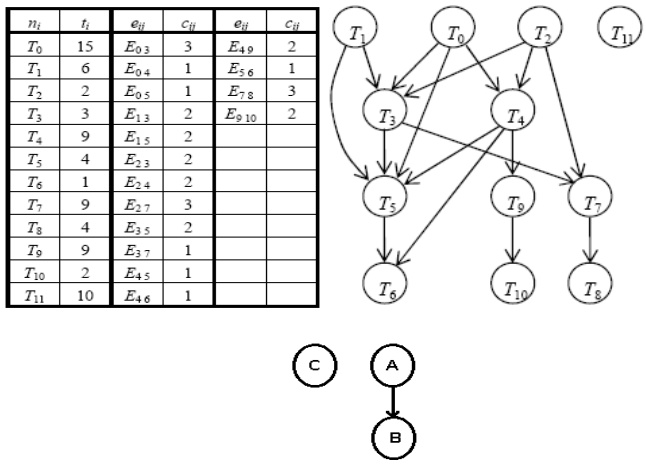

Figure 1. Sample graph

But another concept is priority. If we represent priority with a positive number (for example A=2, B=3, C=1) then A has priority into C, but A and C have not precedence into each other. In this situation it's better that A run before C, however if in a scheduling C run before A it's a valid scheduling too. In the proposed method we conserve all of precedence as a limitation and then we will try to consider priorities. There are many methods for specify priority that can be calculate in other parts of operating system but it is important that scheduling algorithm able to consider priority number in scheduling. For example one of the methods for specifying priority is based on number of children's. In this way whatever the children and grandchild's of a process is more the priority will be greater.

III. STRUCTURE OF PROPOSED CHROMOSOME

In the chromosome that we have designed each gene represents a thread. In fact we are scheduling base on thread. Each gene has three fields: C, T and L. and each chromosome has three fitness: F_r , F_t and F_p (figure 2)

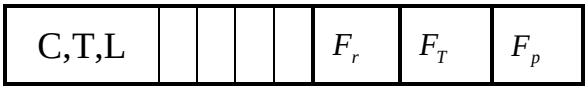

Figure 2. Chromosome structure

The first field of the gene (C) represent processor number that this thread will be execute on it. This field will be specifying randomly while algorithm is running. The second field (T) is the beginning time of thread execution. And L is level number of gene in chromosome.

IV. VALUATION OF L IN GENE

Level number represents the execution sequence of threads in chromosome. Level number of each gene cannot be less than it predecessor's level number. Threads that there is no any precedence between them will give random level number. we can set L easily by, topological sort algorithm.

V. VALUATION OF T IN GENE

T is the starting time of each thread. After than one processor selected for a thread we have to calculate start time of it. We must obtain three conditions to start a thread:

a. Selected CPU has to free (T_{fc}).
b. All of predecessors have to execute (T_p).
c. If one of the predecessors executed on another CPU then the transmission time must be pass (T_i^d).By (1) we have:

$$T_{pi} = T_i^f + T_i^d \quad (1)$$

That T_{pi} is finishing time of predecessor number i , T_i^f is finishing time of execution predecessor i and T_i^d is data transmission time. T_i^d is zero if the current thread and its predecessor be executed on the same CPU. Now by consider that probably we have k predecessors for each thread, last T_p will be the biggest existence T_{pi}. (2)

$$T_p = \max\{T_{pi}\}, i = 1 \ldots N_K \quad (2)$$

After calculation T_p and T_{fc}, main value of T is maximum of T_{fc} and T_p (3), whereas in this time CPU is free, predecessors are executed and data transmission time is passed.

$$T = \max\{T_p, T_{fc}\} \quad (3)$$

VI. IMPLEMENTATION OF MUTATION OPERATOR

We implemented mutation in two ways. After selection one gene for mutation base on P_m , we will produce a number between zero and one that specified kind of mutation. Now introduction two kind of mutation: 1-Mutation in C: in this situation the CPU number that will execute this thread will be changed. Values of each gene is calculating by consider to values of genes in lower level, so value of T have to update from mutated gene to greatest level number.2-Mutation in L: in this situation level number will be change. In other word the sequence of execution threads will be change. Any gene cannot mute before its predecessor or after its successor.

VII. FITNESS OF CHROMOSOME

In this section we will explain method of creation chromosome fitness. Chromosome has three different fitness named F_r , F_t and F_p . F_r is average of the response time of threads in a chromosome and we will try reduce its value. F_t or Total time is greater service time in all of the CPU's. We will try to reduce its value. The optimum total time is calculating by (4) where n is number of threads and m is number of CPU's.

$$Totaltime = \sum_{i=1}^{n} (S_i)/m \quad (4)$$

F_p has been created for incrementing priority observance. We will try to reduce its value. In (5) n is number of threads, P_i is priority of thread i, W_i is waiting time of it. As you see below we have to decrement waiting time to grow this fitness. Between two chromosomes that have equal F_r and F_t , the chromosome is better that has a smaller F_p .

$$F_p = \sum_{i-1}^{n} (p_i * w_i)/n \quad (5)$$

If we have many fitness in a chromosome, we have to consider one importance coefficient between 0 to 1 for each fitness as sum of all coefficients is equal one.

VIII. EVALUATION OF ALGORITHM MODEL

For evaluation efficiency of the algorithm we have implemented proposed algorithm and a classical algorithm by aid MATLAB. Classical algorithm schedules by verifying graph. The input graph has been created randomly. Just we can determine number of nodes. For less than 30 nodes the classical algorithm is better. For this beyond the proposed algorithm can scheduling properly but classical algorithm drops dramatically, as shown in figure 3

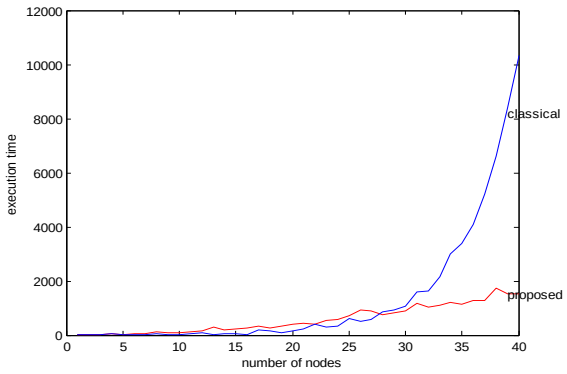

Figure 3. algorithm efficiency

Then we have focused on effect of priority factor. So we generated a random DAG with 25 nodes and ran it several times on proposed algorithm. in each execution we incremented priority of a separate node. In figure 4 we have shown effect of priority factor in response time of separated node and total time. In all executions basic population is 50, number of generation is 10, p_m is 0.2 and number of CPU is 5.

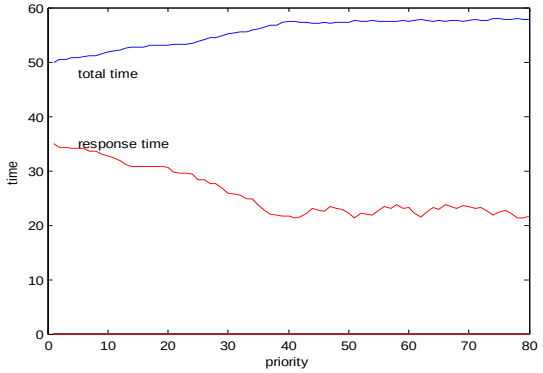

Figure 4. effect of priority

IX. CONCLUSION

In this project we proposed a new scheduler based genetic algorithm that can do scheduling with a new edition of chromosome. One of the positive points in this algorithm is ability to considering priority in scheduling and achieves better response time for group tasks.

We think form of designed chromosome is the basic property of this algorithm. With a little change we can produce new algorithms with new goals. For example by producing a relation graph between CPU's we can scheduling NUMA machines. But proposed algorithm can be used for collective memory structure in multi processors.

X. REFERENCES

[1]. Tanenbaum, A.S. "Modern Operating Systems" , Prentice Hall , 1992

[2]. Tanenbaum, "Distributed Systems Principle and paradigms", Prentice Hall, 2002.

[3]. Chandra, A. , Adler , M. , Shenoy ,P. "Deadline fair scheduling: Bridging the theory and practice of proportionate fair scheduling in multiprocessor systems" , In Proceeding of the 7th IEEE Real-Time Technology and Applications Symposium , May 2001.

[4]. Holman, P., Anderson, J., "Adapting Pfair Scheduling for symmetric Multiprocessors", submitted to journal of Embedded Computing, 2004.

[5]. Auyeung, A., Gondra, I., Dai , H.K., "Intergrating Random Ordering into Multi-Heuristic List Scheduling Generic Algorithm", in proceeding of the third International Conference on Intelligent Systems Design and Applications , Springer-Verlag , pp 447-458 , 2003.

[6]. Auyeung , A., Gondra, I., Dai, H.K., "Multi heuristic List Scheduling Generic Algorithm for Task Scheduling", Proceeding if the 8th Annual ACM Symposium on Applied Computing , ACM Press, pp 721-724 , 2009.

[7]. Lee, Y.H., Chen, C. " A Modified Generic Algorithm for Task Scheduling in Multiprocessor Systems", the 9th workshop on compiler techniques for high-performance computing , 2009.

[8]. Alaoui, S.S., Frieder, O., Elghazwi, T.A , "A Parallel Generic Algoritm for Task Mapping on Parallel Machines", Proceeding of the 11 IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, Springer verlag , London , UK,1999