A new DAG based Dynamic Task Scheduling Algorithm (DYTAS) for Multiprocessor Systems

Dr. D.I. George Amalarethinam
Director - M.C.A., Dept. of Computer Science
Jamal Mohamed College, Trichy, S.India.

G.J. Joyce Mary
Research Scholar, PRIST University,
Thanjavur , S.India.

ABSTRACT
The dynamic tasks scheduling of parallel tasks in multiprocessor systems is still a demanding problem that is being investigated by the researchers. However, the Directed Acyclic Graph (DAG) - based dynamic tasks scheduling is not yet paid enough attention. In this paper a DAG based dynamic tasks scheduling model and a scheduling algorithm DYTAS (DYnamic TAsk Scheduling algorithm) has been proposed with a lower time complexity. Furthermore, the simulation experiments show that, the scheduling model and scheduling algorithm are feasible, a higher scheduling successful ratio may be obtained by this algorithm for parallel jobs with large number of tasks.

General Terms
Parallel program, Directed Acyclic Graph.

Key words : DAG, Dynamic Scheduling, Task, Multiprocessor, Schedule length, Homogeneous.

1. INTRODUCTION
The dynamic tasks scheduling of parallel tasks in multiprocessor system is a challenging problem. Achieving high performance in a multiprocessor system is a key factor of scheduling parallel tasks. The objective of dynamic task scheduling is to map tasks in parallel on the multiprocessors and order their execution so that a minimum schedule length is given under the limit of task precedence requirements. Many scheduling schemes are adopted in parallel computing environment. Most of these schemes consider the precedence constrains among tasks[1]. Up to now, Direct Acyclic Graph (DAG) is a most popular way that is used as modeling the precedence constraints among tasks.

1.1 DAG Model
A parallel program can be represented by a weighed Directed Acyclic Graph(DAG), in which the vertex/node weights represent task processing time and the edge weights represent data dependencies as well as the communication time between tasks. The communication time is also referred as communication cost. Directed Acyclic Graph (DAG) is a directed graph that contains no cycles. A rooted tree is a special kind of DAG and a DAG is a special kind of directed graph.
Directed Acyclic Graph(DAG) $G = (V, E)$, where V is a set of v nodes/vertices and E is a set of e directed edges. The source node of an edge is called the parent node while the sink node is called the child node. A node with no parent is called an entry node and a node with no child is called an exit node.

1.2 Application of DAG
DAGs may be used to model different kinds of structure in mathematics and computer science, to model processes in which information flows in a consistent direction through a network of processors, as a space-efficient representation of a collection of sequences with overlapping subsequences, to represent a network of processing elements and etc. Examples of this include the following:

- In electronic circuit design, a combinational logic circuit is an acyclic system of logic gates that computes a function of an input, where the input and output of the function are represented as individual bits.
- Dataflow programming languages describe systems of values that are related to each other by a directed acyclic graph. When one value changes, its successors are recalculated; each value is evaluated as a function of its predecessors in the DAG.
- In compilers, straight line code (that is, sequences of statements without loops or conditional branches) may be represented by a DAG describing the inputs and outputs of each of the arithmetic operations performed within the code; this representation allows the compiler to perform common sub expression elimination efficiently.

This paper aims at building a dynamic scheduling model with DAGs. In this model, an assigned processor which is called center scheduler, responsible for dynamically schedules the tasks. Based on the proposed dynamic scheduling model we present a new dynamic scheduling algorithm. This algorithm has been tested by conducting experiments in a simulated environment and the results of these experiments show that, the proposed scheduling algorithm is a valid dynamic scheduling algorithm with better performance.

2. RELATED WORK
The dynamic scheduling algorithms presented in many literatures are designed to supporting the real- time system. The real time system is a level of computer responsiveness that a user senses as sufficiently immediate or that enables the computer to keep up with some external process. Real time systems are defined as those system in which the

correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.

The scheduling algorithms are classified into two kinds, namely, static (or off-line) scheduling [1][2][3][4] and dynamic (or on-line) scheduling algorithm[5][6]. Most of these algorithms say that the real-time tasks are independent. While some scheduling algorithms that adopt the DAG model representing the precedence relationship among the tasks only suit to the non real time system. In the recent days, the real-time DAG is used to study the scheduling problem of dependent tasks in a parallel job [3][4]. Xiao Qin et al[3] propose that, the scheduling algorithm doesn't ignore the precedence relationship among the tasks. However they are all static scheduling algorithm. In a word, DAG-based scheduling algorithm that suits to the homogeneous environment with dynamic scheduling in non real time scheduling is rarely seen in the literature. In this paper we present a new dynamic scheduling algorithm - Dynamic Task Scheduling (DYTAS) for non real time system in homogeneous environment. It can deal with multiple parallel tasks that are modeled by DAG. And it also considers the special processing skill of processors.

3. THE SYSTEM MODEL

3.1. The workload model

The arrival of the parallel tasks is assumed dynamic to the homogeneous system. The parallel tasks are modeled by DAG. A non real-time DAG[7] is defined as: $G = (V, E)$, where V is a set of v nodes and E is a set of e directed edges. A node in the DAG represents a task which in turn is a set of instructions which must be executed sequentially without preemption in the same processor. The weight of a node n_i is called the computation cost and is denoted by $w(v_i)$. The edges in the DAG, each of which is denoted by (v_i, v_j), correspond to the communication messages and precedence constraints among the nodes. The weight of an edge is called the communication cost of the edge and is denoted by $c(v_i, v_j)$. The source node of an edge is called the parent node while the sink node is called the child node. A node with no parent is called an entry node and a node with no child is called an exit node.

The homogeneous system comprises a group of processors, $P=\{P_1,P_2,P_3,\ldots P_m\}$, where P_i denotes a processor with local memory. These processors are tightly coupled[8], but, the communication cost between the processors is common.

3.2. The scheduler model

Figure 1 describes a new non real-time scheduler model in homogeneous environment. When all parallel tasks arrive at a assigned processor which is called central scheduler, that will enter into a queue called *Initial Task Queue (ITQ)* , to wait for being scheduled ; In addition to ITQ,

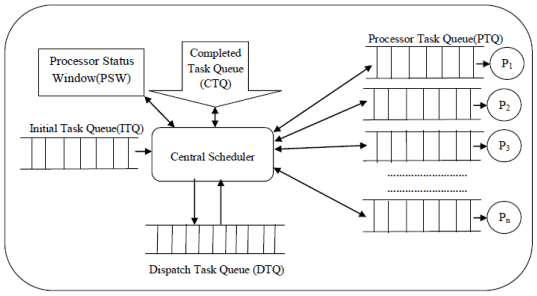

Figure 1. Scheduling model for homogeneous environment

The central scheduler also manages two more queues, Dispatch Task Queue (DTQ) and Completed Task Queue(CTQ). The scheduling algorithm encapsulated in the scheduler is started to work with ITQ. The central scheduler is responsible for scheduling each ready task in the DTQ. Once the scheduling algorithm is started, all the tasks are arranged according to its dependent tasks. After arranging the tasks, the scheduler, schedule the tasks to the individual Processor Task Queue(PTQ). Processors will complete the tasks in their own PTQ by simultaneously checking with its dependent task result in CTQ. If the CTQ is not availing the result of its dependent task , PTQ_i shall point to the next $PTQ_{i+1,}PTQ_{i+2,}\ldots PTQ_{n,}PTQ_{1,}\ldots PTQ_{i-1}$ to fix the suitable task, and migrate that task to the PTQ_i. Until the PTQ_i's become empty, then the scheduling algorithm shall stop working. Then Processor Status Window(PSW) displays the status of each processor that, the list of processors is in running state and in idle state.

4. PROPOSED DYNAMIC SCHEDULING ALGORITHM

We propose a new dynamic scheduling algorithm based on above scheduler model, DYnamic TAsk Schedulling (DYTAS) algorithm. This strategy makes the whole parallel job finished at the possible earliest time viz. the response time of this parallel task is shortest. According to DYTAS, the tasks in ITQ are scheduled by its dependency. The front task in ITQ is always first scheduled and mapped to a processor by the algorithm. While in the static scheduling algorithm, the tasks are sorted by a certain priority rank, because the data of DAG is known in advance[9]. But, the proposed dynamic scheduling algorithm is different from the static scheduling algorithms, by migrating the task during the runtime.

The algorithm DYTAS is focusing on the processor selection strategy. It mainly lies on how to select a processor on which the tasks are mapped even though, the tasks are scheduled earlier.

When selecting a processor to do the particular task from the PTQ_i , two time-indexes must be considered:

- ✓ The earliest free time of the processor P_i
- ✓ The earliest start time of the task v_i on the processor P_i.

In the proposed scheduler model, the parallel tasks in ITQ and the ready tasks are in processor's PTQ. Even though ITQ and DTQ locate at the central scheduler, the processors on which the mapped tasks are actually executed are separated from the scheduler and placed at PTQ_i's. At the same time, the scheduling process and the executing process are parallel works. So that, the scheduler and the working processors synchronous with each other.

1. Procedure *DYTAS*
2. dtq[ ] = SORT[T_i , T_j]
3. l = 0;
4. while (dtq[ ] is not empty) do
5. for i = 1 to n
6. ptq_i = dtq [l]
7. l = l + 1
8. end for;
9. end while;
10. for each processor P_k in processor group do
11. while (P_k is in running state)
12. skip and select the next ptq_{k+1}
13. end while
14. P_k = ptq_k[j]
15. if (dependent task of ptq_k[j] is in ctq)
16. TASK(ptq_k[ ],P_k , j, ctq, cp_k, CT_j)
17. else
18. do
19. move the pointer to the next ptq
20. if (dependent task of ptq_k[j] is in ctq)
21. TASK(ptq_k[ ],P_k , j, ctq, cp_k, CT_j)
22. exit do
23. endif
24. while(checking with all ptq's once)
25. endif
26. end for
27. end *DYTAS*

Procedure for TASK

28. procedure *TASK*(ptq_k[ ],P_k,j,ctq,cp_k,CT_j)
29. do T_j with P_k
30. remove T_j from ptq_k
31. insert T_j in ctq
32. cp_k = cp_k + Ct_j
33. end *TASK*

Figure 2 : Algorithm DYTAS

Figure 2 listing the algorithm DYTAS. The worst case time complexity for DYTAS is $O(t^3)$, t is the total number of the tasks. The loop 5 – 8 distributes all the tasks in DTQ to PTQ_i 's. The middle loop 11 – 13 tours on every ptq to fetch the task to the processor. The loop 15-25 checks the availability of T_j in CTQ. If T_j is available, proceed with the same processor and if not, proceed selecting the task from the next PTQ's till fetches the suitable task.

After fetched or assigned the task to the processor, post in the PSW and remove from the PTQ_i. This process will repeat all the PTQ_i's becomes empty. If any PTQ completes its Task set at the earliest, migrate the suitable task from the available PTQ's. If suppose, there is no task is suitable to fetch at that cycle, processor has to wait till any one of the dependent task becomes available at CTQ. The waiting time is said to be a processors idle state. The time complexity of the whole DYTAS is $O(t^3)$. It is closely related with the length of DTQ and the total number of processors, also related with the precedence relationship among tasks in DAG.

5. SIMULATION EXPERIMENTS AND ANALYSIS

In order to analyzing the feasibility of the proposed scheduling model and the performance of DYTAS, a series of simulation experiments is designed. The DAG of parallel tasks are generated randomly for simulation. The total number of tasks T_i is assumed as 25 for the view and the number of processor is assumed to be 4. Table 1

Table 1 : Input Task List generated using DAGEN

Task_id	CC	CP	Parent Task_id
T0	0	10	-
T1	8	2	T0
T2	8	10	T0
T3	5	17	T0
T4	1	4	T3
T5	9	17	T1
T6	1	2	T0
T7	9	15	T0
T8	5	16	T4
T9	1	15	T2
T10	4	1	T8
T11	3	18	T4
T12	1	4	T8
T13	1	16	T1
T14	9	1	T6
T15	5	15	T3
T16	6	14	T4
T17	1	14	T13
T18	7	12	T0
T19	1	6	T7
T20	1	1	T13
T21	5	1	T19
T22	6	8	T10
T23	2	11	T19
T24	1	10	T5
T24	8	10	T9
T24	8	10	T11
T24	8	10	T12
T24	4	10	T14
T24	4	10	T15
T24	3	10	T16
T24	1	10	T17
T24	3	10	T18
T24	7	10	T20
T24	1	10	T21
T24	3	10	T22
T24	1	10	T23

represents the input data which randomly generated through a tool named *DAGEN.*

Table 2 : PTQ_i's at the Initial Stage

PTQ1	T0	T3	T5	T16	T19	T20	T24
PTQ2	T18	T2	T9	T11	T12	T17	
PTQ3	T7	T1	T15	T8	T10	T23	
PTQ4	T6	T13	T4	T14	T22	T21	

Tasks are evenly distributed to PTQ's after arranging according to its dependency. The PTQ_i status at the initial stage is described in Table 2. The dynamic scheduling is implemented the task migration during the run time. Table 3 shows the task migration. Table 4 shows the runtime anomaly of parallelized tasks.

Scheduling length in a single processor of our example is 240 time units. Time taken to complete tasks in the first processor (P1) is 80 time units, P2 is 65 time units, P3 is 67 time units and P4 is 66 time units. We assume that, the number of processors(np) in the architecture is 4. The minimum schedule length can be obtained from

Table 3 : PTQi 's during run time

PTQ1	T0	T3	**T5** migrate to P3 at runtime	**T16** migrate to P4 at runtime	T19	T20	T24
PTQ2	T18	**T2** migrate to P4 at runtime	T9	T11	T12	**T17** migrate to P1 at runtime	
PTQ3	T7	**T1** migrate to P4 at runtime	**T15** migrate to P1 at runtime	T8	T10	**T23** migrate to P4 at runtime	
PTQ4	T6	T13	**T4** migrate to P2 at runtime	T14	**T22** migrate to P3 at runtime	**T21** migrate to P2 at runtime	

Minimum Schedule Length =

$$\frac{\text{Schedule length in a Single Processor}}{\text{Number of Processors in the Architecture}}$$

The minimum schedule length of the above illustration is 60 . Hence the speedup shall be attained from

$$\text{Speedup} = \frac{Minimum\ Schedule\ length}{Schedule\ Length}$$

PRO-CESSORS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
P1	T0(10)										T3(17)																	T15(15)													
P2											T18(12)																	T9(15)												T4(4)	
P3											T7(15)																													T5(17)		
P4											T6(2)					T2(10)								T1(2)																	T13(16)		

PRO-CESSORS	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
P1	T15(15)			T19(6)					T20 (1)		T17(14)																				T24(10)									
P2	T4(4)				T11(18)															T21 (1)			T12(4)																	
P3	T5(17)					T8(16)													T10 (1)		T22(8)																			
P4	T14 (1)				T16(14)																T23(11)																			

Computation cost

Table 4 : Run time anomaly of parallelized tasks

The Speedup percentage of this prototype is 75 %. But at the same time, with the help of this research, we may conclude that, speedup of the parallel system is depending upon not only the proper scheduling, but also the number of processors in the architecture and the number of tasks scheduled to the architecture.

The number of tasks and the number of processors are directly proportional to each other. Though, the architecture used in our model is tightly coupled, the communication cost is not to be entertained, because it is negligible.

The results of simulation experiment shows that, the proposed dynamic scheduling model and scheduling algorithm DYTAS of DAG-based parallel tasks gives the better scheduling with high speed up percentage. The

Speedup curve in Figure 3a, 3b, 3c shows that, the number of tasks and the number of processors is related with each other and affects the speed of the system. We can powerfully use our DYTAS in the multiprocessor system by providing suitable set of tasks.

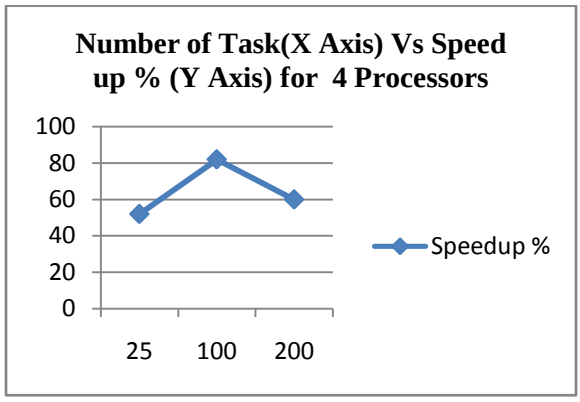

Figure 3.a The Speedup curve when P = 4

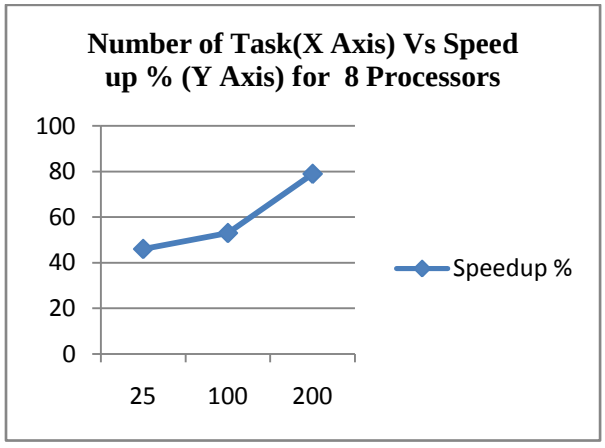

Figure 3.b The Speedup curve when P = 8

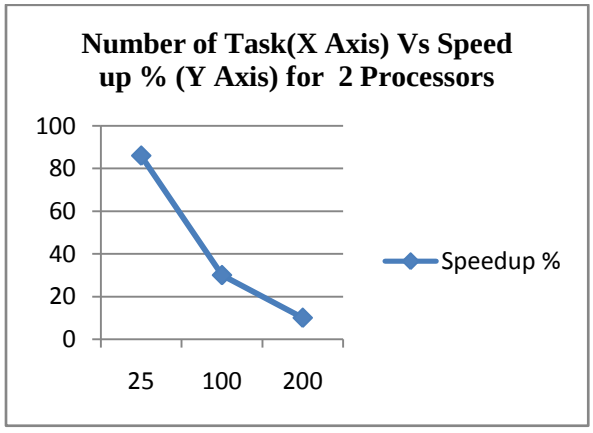

Figure 3.c The Speedup curve when P = 2

6. CONCLUSION AND FUTURE WORK

Studying of task scheduling problem is still a research focus. The scheduling algorithms presented by researcher are often seen in the literature. But the DAG-based parallel tasks in dynamic task scheduling algorithm don't yet magnetize enough attentions. So we build a new dynamic scheduling model and present a scheduling algorithm DYTAS based on the new model. Obviously, the worst case time complexity of DYTAS is low. The results of simulation experiment shows that, DYTAS is feasible with DAG based dynamic scheduling. The dynamic scheduling using DYTAS is based on dependent tasks, and the distribution based on the availability of processors. The speedup is reasonable and it shows that, the algorithm is working properly. The proposed scheduling algorithm should add fault-tolerant function in the further work.

REFERENCE

[1] T.F.Abdulzaher, K.G.Shin, 1999 "Combined task and message scheduling in distributed real-time systems," IEEE Transaction on Parallel and Distributed Systems, Vol.10, No.11

[2] J.C.Palencia, H.M.Gonzalez, 1998 "Schedulability analysis for tasks with static and dynamic offsets," In Proceeding of the 19th IEEE Real-Time Systems Symposium, pp.26-37.

[3] Xiao Qin, Hong Jiang, C.S.Xie, Z.F.Han, 2000 "Reliabilitydriven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems," In Proceeding of 12th International Conference Parallel and Distributed Computing and Systems .

[4] Xiao Qin, Z.F.Han, H.Jin, L.P.Pang,2000 "Real-time faulttolerant scheduling in heterogeneous distributed systems," in proceeding of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications, Vol.I, pp. 421-427.

[5] V.Kalogeraki, P.M.Melliar-Smith, L.E.Moser, 2000, "Dynamic scheduling for soft real-time distributed object systems," In proceeding of Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pp. 114-121.

[6]G.Manimaran and C.S.R Murthy,1998, " An efficient dynamic scheduling algorithm for multiprocessor real-time systems," IEEE Transaction on Parallel and Distributed system, Vol.9, No.3, pp. 312-319.

[7] Dan Ma, Wei Zhang, Qinghua Li, 2004, **"**Dynamic Scheduling Algorithm for Parallel Real-time Jobs in Heterogeneous System" Proceedings of the Fourth International Conference on Computer and Information Technology (CIT'04) IEEE

[8] Kai Hwang and Faye A. Briggs, 1984, "Computer Architecture and Parallel Processing"

[9] G.J. Joyce Mary, D.I. George Amalarethinam, 2010, Dynamic Task Scheduling in Multiprocessor and the Swift Embryonic World of Parallel Computing – A Survey. Published in the "International Journal of Algorithm, Computing and Mathematics" – Vol. III – No. 4 , PP 53