An Aggressive Parallel Tasks Scheduling Algorithm: Relative Mobility Scheduling Algorithm (RMS)

Wai-Yip Chan and Chi-Kwong Li
Department of Electronic Engineering,
The Hong Kong Polytechnic University, Hong Kong.

Abstract

In this paper, an aggressive scheduling algorithm called Relative Mobility Scheduling algorithm (RMS) which is designed based on heuristic of Relative Mobility (RM) is developed. The proposed Aggressive RMS algorithm has the advantage of scheduling tasks into either bounded or unbounded number of processors within a minimize parallel time. The goals of the algorithm are to maximize the utilization of the available processors as well as the parallelism of the scheduling task graph, such that the parallel time of the produced schedule is a minimum. Experimental studies are carried out to evaluate and demonstrate the significant improvement of the proposed algorithm.

1 Introduction

A better scheduling of tasks in parallel and distributed environment is no doubt can lead to overall performance improvement. However, task scheduling is proved to be a NP-complete problem[1]; it is very hard to obtain an optimum scheduling solution in polynomial time complexity. Thus, heuristic approaches of task scheduling algorithms are used to approximate optimal solution within polynomial time complexity. The Mobility Direct algorithm (MD), as proposed by Wu and Gajski[4], is designed with Relative Mobility (M_r) being the heuristic. With this heuristic, it is possible to determine the minimum number of processors required to execute a program with near optimal performance.

In this research, the MD algorithm is studied and it is found that this algorithm cannot produce a shorter parallel time schedule when scheduling task into either bounded or unbounded number of processors. Therefore, an improved scheduling algorithm called Relative Mobility Scheduling algorithm (RMS) is proposed to improve the Wu's MD algorithm. With experiments studies significant improvement is achieved with the proposed algorithm.

0-7803-3905-3/97/$10.00©1997 IEEE

2 The Wu's Mobility Direct algorithm (MD)

The Wu's Mobility Direct algorithm was suggested by Wu and Gajski[4]. The main philosophy of the MD algorithm is based on the list scheduling heuristic[5]. Firstly, the Relative Mobility(M_r) heuristic is used as a priority value to generate a priority list of free tasks for the scheduling the task graph. In fact the M_r is the relative moving range of tasks in which is defined as:

$$M_r(n_i) = \frac{T_L(n_i) - T_S(n_i)}{W_i(n_i)},$$

where $T_L(n_i)$ is the latest start time of a node n_i (ALAP[3]), $T_S(n_i)$ is the earliest start time of a node n_i (ASAP[3]) and $W_i(n_i)$ is the cost of a node n_i.

Secondly, the highest priority task is selected from the list and scheduled into the first available processor that satisfies Fact 1 as described in [4]. Thirdly, dependence constrain of the scheduled task is reorganized and the task is removed from the list. Finally, the M_r of all remaining free tasks are computed. The above steps are then repeated until all tasks in the task graph are scheduled.

The MD algorithm uses the relative mobility and the Fact 1 as a heuristic of scheduling. The application of relative mobility heuristic guarantees moving range of starting a task in execution which does not delay the execution of any other tasks on the critical path. Therefore, data dependence between tasks is satisfied and the length of the critical path will not increase in each scheduling step. As a result, the algorithm is possible to generate a near optimal schedule in the sense of guaranteeing the parallel time of the produced schedule which is not longer than the critical path of the original task graph.

Nevertheless, the task to processors assignment strategy of the MD algorithm does not guarantee the algorithm to produce a shorter parallel time scheduling solution when the algorithm is used to schedule tasks into both bounded and unbounded number of processors. It is because the strategy assigns the selected task into a processor in which the pro-

cessor is selected from scanning through the available processor set of the first one that satisfies the Fact 1. Although this strategy can determine the minimum number of processors required to execute the parallel program. The selected processors may not be the ones that can finish the task in minimum time. This result in accumulate of unnecessary increased parallel time in each scheduling step. As a result, usage of processors in the available processors set cannot be fully utilized. Parallelism of the scheduling task graph cannot be maximized and parallel time of the produced schedule cannot be minimized in the situation of scheduling tasks into both bounded and unbounded of processors.

3 The proposed Relative Mobility Scheduling algorithm (RMS)

In this section, a more aggressive scheduling algorithm called the Relative Mobility Scheduling algorithm (RMS) is proposed to schedule tasks into both bounded and unbounded number of processors. The goals of the algorithm are to maximize utilization of the available processors and to maximize parallelism of the scheduling task graph such that the parallel time of the produced schedule can be minimized.

An algorithmic description of the RMS algorithm is shown in algorithm 1a and 1b. In step 1 shown in algorithm 1a, the Relative Mobility (M_r) of all tasks in the scheduling task graph is determined. In step 2, a priority list L' is generated using the heuristic of M_r and then the highest priority free task n_i of the list is selected to be scheduled. In step 3, the given processors set P is scanned through to identify a processor P_m such that n_i executed in that processor would satisfy Condition 1 shown in algorithm 1b and finish in minimum time. The Condition 1 is modified from Fact 1 of the Wu's algorithm in which supports the scanning process. After that, dependence constrain of the scheduled task is reorganized and the task is removed from the list. In step 4, the M_r of all remaining free tasks are recalculated and the steps 2-3 are repeated until all tasks in the task graph are scheduled.

1: Calculate relative mobility for all tasks.
2: Let L include all tasks initially. Let L' be the group of unexamined tasks in L with minimum relative mobility. let n_i be a task in L' that does not have any predecessors in L'.
3: Find a processors P_m in a given processors set P in which n_i would satisfy Fact1 and finish in minimum finish time (or the earliest finish time is reached). When n_i is scheduled on P_m, all edges connecting n_i and other tasks already scheduled to P_m are changed to zero.

   If n_i is scheduled before task n_j on P_m, add an edge with cost zero from n_j to n_i in the graph. If n_i is scheduled after task n_j on the processor, add an edge with cost zero from n_j to n_i in the graph. Remove n_i from L.
4: Recalculate relative mobility for the modified graph. Repeat steps 2 to 4 until L is empty.

Algorithm 1a, The RMS algorithm.

Necessary and sufficient condition for scheduling a task to a processor P_m(where $P_m \in P$):
Assume a task n_p is considering to be scheduled to P_m , in which l tasks, n_{m_1}, n_{m_2}, ..., n_{m_l}, have been scheduled into P_m. If the moving intervals of these tasks do not intersect with the moving interval of n_p, then n_p can be scheduled to P_m. Otherwise, assume the moving intervals of tasks n_{m_i}, $n_{m_{i+1}}$, ..., n_{m_j} ($1 \le i \le j \le l$) intersect the moving interval of n_p. n_p can be scheduled to P_m, if there exists k ($i \le k \le j+1$),

$$W(n_p) \le min(T_F(n_p), T_L(n_{m_k})) - max(T_S(n_p), T_S(n_{m_{k-1}}) + W(n_{m_{k-1}})).$$

Where $W(n_i)$ is the cost of n_i; and
The $T_F(n_p)$ and $T_S(n_p)$ computation should assume that task n_p is scheduled to processor P_m; and
When n_p is scheduled to processor P_m, n_p is inserted before the first task in the task sequence of processor P_m that satisfies the inequality listed above; and
if $n_{m_{i-1}}$ does not exist, $T_S(n_{m_{i-1}}) = 0$, $W(n_{m_{i-1}}) = 0$;
if $n_{m_{j+1}}$ does not exist, $T_L(n_{mj+1}) = \infty$. Otherwise, the task cannot be scheduled to P_m.

Algorithm 1b, (Condition 1 of the RMS algorithm): Necessary and sufficient condition for scheduling a task.

The major improvement of the RMS algorithm over the MD algorithm is in the task to processor assignment strategy. The RMS algorithm uses a strategy of scheduling a task into a processor in which it can finish the task as soon as possible. This is shown by the experiments in next section that such change implies an aggressive search for a better allocation of tasks into processors with smaller finish time. As a result, the unnecessary increase in parallel time in each scheduling steps can be reduced and a near to optimum scheduling solution can be produced in most cases of difference grain size of scheduling task graphs.

4 Experimental Studies

In this section, an experiment is set up to study performance of the RMS algorithm and the MD algorithm in which are used to scheduling tasks into different sets of processors configurations. Owing to the scheduling problem is NP-complete in nature, heuristic ideas used in both MD and RMS algorithm cannot always lead to an optimal solution. Thus, it is necessary to compare the average performance of these algorithms by using randomly generated graphs. In this experiment, more than 7000 random generated graphs are used. They are divided into three groups of different granularity (ratio of task cost over communication cost: R/C):

Group 1 (G1):	Fine-grain, R/C range = 0.1-0.3.
Group 2 (G2):	Median-grain, R/C range = 0.8-1.2, the average cost of computation and communication are close.
Group 3 (G3):	Coarse-grain, R/C range = 3-10.

Sub-group	Layer	Tasks	Edges
A	9-11	50	170
B	9-11	60	200
C	18-21	100	340
D	18-20	180	620
E	18-20	350	1200
F	36-40	540	1800

Table 1, Sub-grouping of random generated task graphs.

Each group is further classified into six sub-groups summarized in Table 1 that more than 400 graphs each in which they are in different number of layer and tasks configuration. These task graphs are scheduled by both MD and RMS algorithm into three different sets of processors configurations:

Set 1: Unbounded number of processors ($|P| \geq |N|$); and
Set 2: Minimum number of processors which is determined by the MD algorithm required to execute the task graph ($|P| = P_{min}$); and
Set 3: Bounded number of processors ($|N| > |P| > P_{min}$).

,where $|P|$ is the number of processor in a processor set P, $|N|$ is the total number of tasks in the scheduling task graph and P_{min} is the minimum number of processors which is determined by the MD algorithm required to execute the task graph.

Two arguments are used to measure the figure of merit of RMS over MD algorithm. They are the probability of parallel time produced by the RMS algorithm shorter than the MD algorithm (Pr) and parallel time improvement ratio of RMS over MD algorithm (R/M). The R/M ratio is defined as follow:

$$R/M = (1 - \frac{PT_{rms}}{PT_{md}})$$

Sub-group	Group 1		Group 2		Group 3	
	R/M (%)	Pr	R/M (%)	Pr	R/M (%)	Pr
A	52.32	0.94	26.88	0.93	25.21	0.92
B	56.68	0.96	30.24	0.93	26.68	0.94
C	63.51	0.95	29.81	0.95	22.57	0.95
D	74.22	0.97	47.63	0.97	29.06	0.94
E	81.56	0.97	71.96	0.98	40.86	0.96
F	83.35	0.96	75.61	0.98	38.16	0.97
Average	68.61	0.96	47.02	0.96	30.42	0.95

Table 2, To schedule graphs into Set 1($|P| \geq |N|$).

Sub-group	Group 1		Group 2		Group 3	
	R/M (%)	Pr	R/M (%)	Pr	R/M (%)	Pr
A	28.61	0.91	19.49	0.84	17.65	0.83
B	31.99	0.90	17.22	0.84	15.41	0.82
C	37.28	0.94	18.79	0.88	16.56	0.87
D	37.56	0.95	30.21	0.97	15.09	0.86
E	35.85	0.95	51.45	0.97	21.18	0.91
F	38.21	0.91	57.62	0.98	22.35	0.95
Average	34.92	0.93	32.46	0.91	18.04	0.87

Table 3, To schedule graphs into Set 2 ($|P| = P_{min}$).

Sub-group	Group 1		Group 2		Group 3	
	R/M (%)	Pr	R/M (%)	Pr	R/M (%)	Pr
A	27.46	0.92	24.38	0.87	25.86	0.91
B	33.16	0.92	23.43	0.91	25.8	0.91
C	34.72	0.94	22.21	0.93	21.56	0.93
D	38.72	0.96	33.85	0.98	26.73	0.93
E	37.65	0.97	49.95	0.97	34.52	0.95
F	39.77	0.96	57.79	0.98	37.53	0.95
Average	35.25	0.95	35.27	0.94	28.67	0.93

Table 4, To schedule graphs into Set 3 ($|N| > |P| > P_{min}$).

The result of scheduling groups G1, G2, G3 under three different configurations of processors are summarized in Table 2, 3 and 4. It is shown in these tables that the R/M ratio of RMS algorithm is higher than that of MD algorithm when tasks are scheduled into both bounded and unbounded number of processor (Table 2 to 4). It is interesting to find in Table 4 that even the RMS is used to schedule task into processors set Set 3 ($|P| = P_{min}$), the produced parallel time is better than that of the MD algorithm in most situation. These are because the MD algorithm stops to find a more suitable schedule of task after the first available processors satisfied Fact 1 is found. On the other hand, the RMS algorithm is more aggressively finding better schedule of processors that can make the parallel time shorter. This not only can maximize the utilization of available processors, but also increase the parallelism of the final schedule. As a result, the R/M of RMS algorithm is better than that of MD algorithm in both bounded and unbounded set of processors.

It is observed that the improvement ratio R/M in fine-grain groups is the highest and then gradually decreases as the granularity increase under different processor configurations. Although the granularity approaches to 10, there are over 90% of parallel time produced by the RMS algorithm being shorter than that of MD algorithm in 25 %. This finding implies that the parallel time produced by the RMS algorithm is shorter than that of the MD algorithm in most situations of fine-grain task graph. For the granularity is increasing, the parallel time produced by the RMS algorithm becomes close to but probably shorter than that of the MD algorithm produced.

Furthermore, owing to both MD and RMS algorithm are using heuristic information to make a scheduling decision. This algorithm does not guarantee to produce an optimum solution. As a result, parallel time produced by RMS is not always shorter than that of MD algorithm. In the experiment a probability value (Pr) is used to determine the probability of parallel time produced by RMS is better than that of MD algorithm. It is found that 90% of parallel time produced by RMS is shorter than that of MD algorithm in the three different processors configurations. This implies that the RMS algorithm is not better than MD algorithm in all situations, but it is better than MD algorithm in most situations.

The weakness of the RMS algorithm is that the minimum number of processors needed to execute the program cannot be determined as compare with the MD algorithm. However, the parallel time of the scheduled graph produced by the RMS algorithm is better than that of the MD algorithm in most situations, especially in scheduling fine grain task graphs and scheduling tasks into unbounded number of processors.

5 Conclusion

An aggressive task scheduling algorithm called RMS algorithm is proposed in this paper. Using this algorithm improved performance is achieved. This algorithm can also schedule all kinds of task graphs in shorter parallel time with relatively lower time complexity. It is also found that from the experiment studies the proposed RMS algorithm can aggressively schedule tasks with aims at maximizing processors utilization and increasing parallelism of scheduling task graph. As a result, the parallel time of the final schedule time can be minimized. The proposed algorithm is a high performance scheduling algorithm and is very suitable to be deploy in scheduling parallel of tasks with wide range of granularity into homogeneous parallel and distributed systems.

References

[1] P.Chretienne, "Task scheduling over distributed memory machines," in Proc. Int. Workshop Parallel Distrib. Algorithms, 1989.

[2] Heshan El-Rewini, Theodre G. Lewis amd Jesjam H. Ali, "Task Scheduling in Parallel and Distributed System," pp. 56-81, Prentic Hall. 1993.

[3] Landskov, D., Davidson, S., Shriver, B., and Mallett, P. W. "Local microcode computation techniques." Computing Surveys, 12(3): pp.261-294, 1980.

[4] M.Y. Wu and D. Gajski, "Hypertool: A programming aid for message-passing system," IEEE Trans. Parallel Distrib. Syst., vol. 1, pp.330-343, 1990.

[5] J. Baxter and J.H. Patel, "The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm," Proceedings of the 1989 Int. Conference on Parallel Processing, Vol.2, pp. 217-222, 1989.