A New Method for Independent Task Scheduling in Nonlinearly DAG Clustering

Qiang-Sheng Hua[1], Zhi-Gang Chen[1], and Francis C.M. Lau[2]
[1]*School of Information Science & Engineering,*
Central South University, Changsha, P.R. China, 410083
[2]*Department of Computer Science & Information Systems,*
The University of Hong Kong, Pokfulam Road, Hong Kong
fcmlau@csis.hku.hk

Abstract

For the parallel tasks represented by the Directed Acyclic Graph (DAG), if it is linearly clustered, the ordering of the execution time of the tasks in each cluster is based on their arrows in the DAG. But for nonlinearly clustering, the ordering of the independent tasks in each cluster is not easily decided. Improper ordering of these independent tasks will greatly increase the scheduling length of the DAG. We discuss the shortcomings of current scheduling algorithms and the reason behind poor performance, and then propose some new node information to be extracted which is used by a new independent tasks scheduling algorithm based on the Maximized Parallelism Degree (MPD). Experimental results show that the MPD algorithm can yield better performance than the previous algorithms.

1. Introduction

There are two different methods in DAG clustering[1,2,4,12]: linearly clustering and nonlinearly clustering. If there are no independent tasks in each cluster (two nodes independent if there are no dependence paths between them[2]), then it is linearly clustering; otherwise nonlinearly clustering. Generally speaking, for coarse grain tasks, linearly clustering is better than the nonlinearly clustering[5,6,8,9]. For fine grain tasks, nonlinearly clustering is better[11].

It is well known that, if the tasks have been linearly clustered, the order of the time to execute is from the end task of the arrows to the front task of the arrows because there are dependence paths between the tasks of each cluster[3,7]. Figure 1(d) shows a linearly clustered task graph, where node n_1,n_2 and n_4 have been mapped into one cluster, and node n_3 is in another cluster. But for nonlinearly clustering, there exist at least two independent tasks in some cluster. Different orderings of these independent tasks always greatly influence the task scheduling length of the DAG. So how to order these tasks in the nonlinearly clustered DAG is crucial. Although some algorithms have been proposed to address the problem using different strategies[2], we found that they cannot satisfy all kinds of DAGs. In this paper, we first discuss the shortcomings of current task scheduling algorithms, point out the reasons why they could lead to poor performance, and then propose some new node information to be extracted which is based a modification of the tlevel and blevel values; finally, the paper presents a new independent task scheduling algorithm for nonlinearly DAG clustering based on the concept of Maximized Parallelism Degree (MPD). Experimental results show that our algorithm yields better performance than previous algorithms.

2. Definitions and Assumptions

Examples of DAG are shown in Figure 1.We suppose that the DAG has been nonlinearly clustered, and there is one processor per cluster. Now we need to order the execution times of the independent tasks in each cluster.

DEFINITION 1: A weighted DAG is a tuple G=(V,E,W,C), where $V=(V_1,V_2,\ldots,V_{|V|})$ is the set of nodes, |V| is the number of the nodes, $E=\{e_{ij}|v_i,v_j \in V\} \subseteq V \times V$, is the set of communication edges, and |E| is the number of edges. The set C is the set of edge communication costs and W the set of node computation costs. The value $C_{ij} \in C$ is the communication cost incurred along the edge $e_{ij} \in E$, which is zero if both nodes are mapped in the same processor. The value $W_i \in W$ is the computation cost for node $V_i \in V$. We use $PRED(V_i)$ for the set of immediate predecessors of V_i and $SUCC(V_i)$ the set of immediate successors of V_i. If $PRED(V_i)= \phi$, then node V_i is an entry node, and symmetrically if $SUCC(V_i)= \phi$, then node V_i is an exit node. Two nodes are called independent if there are no dependence paths between them.

DEFINITION 2: $tlevel(V_i)$ is the length of the longest path from an entry node to V_i excluding the weight of V_i

Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'04)
1087-4089/04 $ 20.00 © 2004 IEEE

in a DAG. blevel(V_i) is the length of the longest path from V_i to an exit node. Because the communication edge on the path may be zeroed during the clustering, both tlevel(V_i) and blevel(V_i) could be dynamically changed. If the calculation of blevel(V_i) does not take the edge weights into account, then it is a static blevel which is abbreviated as sbl. The tlevel value of an entry node is zero, and the blevel value of an exit node is its computation cost.

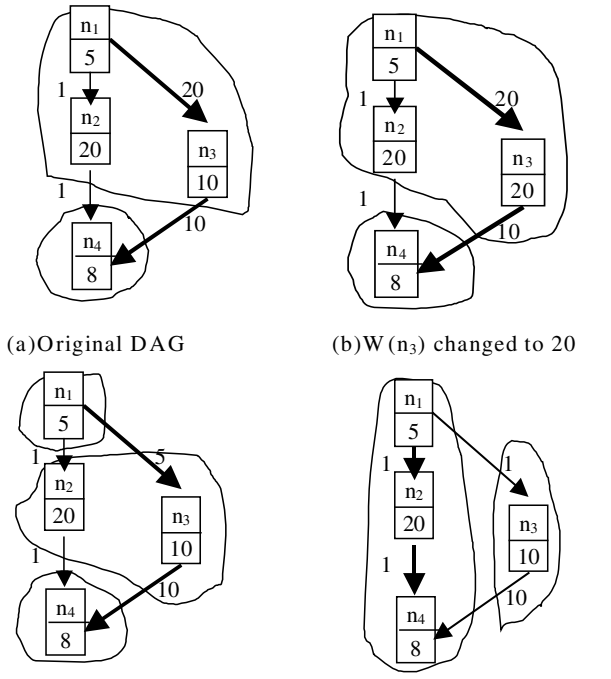


Figure1 Four clustered DAG (Fig a,b,c nonlinearly clustered, Fig d linearly clustered)

3. Current Task Scheduling Strategies

The node information defined in Section 2 such as tlevel, blevel and sbl comes from previous task scheduling algorithms[2]. Since then, nearly all subsequent scheduling algorithms made use of these node values. These values have played a great role in both the clustering step and the scheduling step. Because the node value "tlevel" decides the earliest start time of a task, many algorithms such as MCP[2] give higher priority to the node which has a lower "tlevel" value, to make sure that the node can be executed earlier than other tasks, which also means to execute the tasks in a DAG with a topological order. Because the node value "blevel" is closely related to the critical path of a DAG, in order to schedule the tasks on the critical path (CP), some algorithms such as Sarkar[3] give higher priority to the node which has a higher "blevel" value; this is also to make sure that this task can be executed earlier than other tasks. And because of the clustering step, the node values have to be changed during the process. Some algorithm such as EZ (Edge Zeroing) and ETF (Earliest Time First)[2] give higher priority to the node which has a higher "sbl" value. In addition, in order to comprehensively consider the earliest start time of a node and the critical path node, other algorithms[2] give higher priority to the node which has a higher "btlevel" value (btlevel is represented by blevel minus tlevel). In order to give a clearer picture in which to present our MPD algorithm, some detailed applications of these current scheduling strategies are given below. Note that the DAG in Figure 1(a), (b) and (c) has been nonlinearly clustered.

For Figure 1(a), tasks n_1,n_2 and n_3 have been merged into a cluster, and the remaining task n_4 becomes another cluster. It is obvious that node n_2 and n_3 are independent tasks. Their node information is shown in Table 1. Because tlevel(n_2)=5 and tlevel(n_3)=5, so it is impossible to order their execution times based on their tlevel values. Because blevel(n_2)=29 and blevel(n_3)=28, if we give higher priority to the node which has a higher blevel value, then node n_2 will be executed earlier than node n_3. Then the scheduling length of this DAG would be **5+20+10+10+8=53** (note that the communication value in a cluster has become zero). If we give higher priority to the node which has a higher sbl value or higher btlevel (blevel-tleve) value, then because sbl(n_2)=28, sbl(n_3)=18，btlevel(n_2)=24 and btlevel(n_3)=23, node n_2 would still be executed earlier than node n_3, and the scheduling length would still be 53. But if we tried to schedule n_3 earlier, then the scheduling length becomes **5+10+20+1+8=44,** this scheduling strategy greatly reduces the scheduling length. But all of the current scheduling strategies would fail to make sure that n_3 should be executed earlier than n_2.

Now one may think that if we give lower priority instead of higher priority to these nodes which have a higher blevel, sbl or btlevel value, we can make sure that n_3 would be executed earlier than n_2. It is true for this example, but what about if we changed some costs of the nodes in Figure 1(a)?

For Figure 1(b), we changed the value of n_3 from 10 to 20, and Table 2 shows their new node information. Because now sbl(n_2) equals sbl(n_3), it is impossible to order their execution times based on the sbl values. But because blevel(n_2)<blevel(n_3) and btlevel(n_2)<btlevel(n_3), if we choose to first execute the nodes with a lower blevel or btlevel value, node n_2 would still be executed earlier than node n_3, and the scheduling length would be **5+20+20+10+8=63**. But if we try to first execute node n_3, then the scheduling length becomes **5+20+20+1+8=54.**

So this strategy cannot work either. And scheduling task n_3 first is still better than scheduling task n_2 first.

For Figure 1(c), let us try to schedule the task which has a lower tlevel value to see if it can give a better result. We now change $C(n_1,n_3)$ from 20 to 5, and now there exist three clusters, tasks n_2 and n_3, task n_1 and task n_4. Task n_2 and n_3 are still independent tasks. The new node information is shown in Table 3. Because $tlevel(n_2)<tlevel(n_3)$, $blevel(n_2)>blevel(n_3)$, $sbl(n_2)>sbl(n_3)$ and $btlevel(n_2)>btlevel(n_3)$, it is easy to see that task n_2 should be executed earlier than task n_3. And if we try to schedule the task which has a higher blevel, sbl or btlevel value, n_2 would still be executed earlier than n_3. The scheduling length would be **5+1+20+10+10+8=54**. But if we tried to execute task n_3 first, then the scheduling length would become **5+5+10+20+1+8=49.** So scheduling the task which has a lower tlevel value cannot make sure that the scheduling length of the parallel tasks would be the shortest either.

In short, node information such as blevel and tlevel extracted in the DAG is not sufficient for ensuring the shortest scheduling length. We need to extract new node information and to design a new scheduling algorithm based on the new node values.

4. A New Scheduling Algorithm based on Maximized Parallelism Degree (MPD)

From the examples shown in Section 3, we can see that previous algorithms cannot ensure the shortest scheduling lengths. And scheduling task n_3 first is always better than scheduling task n_2. The reason is that the current scheduling strategies fail to consider the maximized parallelism degree in the scheduling process, or we can say that they fail to maximize the parallelism among the computations of the nodes and the communications of the nodes.

Because the tasks have been clustered, the computation time of each node and the communication time between nodes have been decided, and they cannot be changed during the scheduling process; the only way then to reduce the scheduling length of the DAG is to maximize the parallelism degree. In order to maximize the parallelism degree, we need to extract new node information from the DAG.

4.1 New node information extracted from the DAG

In order to maximize the parallelism degree of the nodes in the DAG and to measure the values of the parallelism degree, we introduce the following definition.

DEFINITION 3: $televel(V_i)$ is the length of the longest path from an entry node to V_i including the weight of V_i in a DAG. Symmetrically $belevel(V_i)$ is the length of the longest path from V_i to an exit node excluding the weight of V_i in a DAG. The televel value of an entry node is its computation cost, and the belevel value of an exit node is zero.

Node value \ Nodes	n_1	n_2	n_3	n_4
tlevel	0	5	5	26
blevel	34	29	28	8
(btlevel) blevel-tlevel	34	24	23	-18
Static_blevel	33	28	18	8
televel	5	25	15	34
belevel	29	9	18	0

Table1 Node information from Fig1(a)

Node value \ Nodes	n_1	n_2	n_3	n_4
tlevel	0	5	5	35
blevel	43	29	38	8
(btlevel) blevel-tlevel	43	24	33	-27
Static_blevel	33	28	28	8
televel	5	25	25	43
belevel	38	9	18	0

Table2 Node information from Fig1(b)

Node value \ Nodes	n_1	n_2	n_3	n_4
tlevel	0	6	10	30
blevel	38	29	28	8
(btlevel) blevel-tlevel	38	23	18	-22
Static_blevel	33	28	18	8
televel	5	26	20	38
belevel	33	9	18	0

Table3 Node information from Fig1(c)

4.2 The MPD algorithm

Suppose node V_i and node V_j are the independent tasks in a cluster, the ordering of their execution times is decided by the following algorithm (where function min(a, b) returns the smaller value between a and b).

In the algorithm, $min[televel(V_i),tlevel(V_j)]$ stands for the first part of the parallelism degree where node V_i has

been executed but node V_j has not been executed; min[belevel(V_i) + blevel(V_j)] stands for the remaining parallelism degree where node V_j has also been executed; and the sum of the two functions represents the whole parallelism degree where node V_i is executed earlier than node V_j. Let us use sum1 to denote this sum. And symmetrically, min[televel(V_j), tlevel(V_i)] stands for the first part of the parallelism degree where node V_j has been executed but node V_i has not been executed; min[belevel(V_j) + blevel(V_i)] stands for the remaining parallelism degree where node V_i has also been executed; and the sum of the two functions stands for the whole parallelism degree where node V_j is executed earlier than node V_i. Let us use sum2 to denote this sum. Now we can compare the sum1 with the sum2 to see which is larger, and the larger sum certainly stands for the maximized parallelism degree. And the absolute value of the difference between sum1 and sum2 stands for the difference between the different scheduling lengths.

MPD Algorithm:
Step 1: If there exist independent tasks V_i and V_j in a cluster, then compute the televel and belevel values;
If
min[televel(V_i),tlevel(V_j)]+min[belevel(V_i)+blevel(V_j)]>=
min[televel(V_j),tlevel(V_i)]+min[belevel(V_j)+blevel(V_i)]
then schedule task V_i first
else schedule task V_j first;
Step 2: Add a dashed line from the first scheduled node to the second scheduled node;
Step 3: Recompute the node information, and goto step1.

The MPD algorithm can ensure that the node which has the larger parallelism degree is executed earlier than other independent nodes in the same cluster. Thus the scheduling length of the parallel tasks will definitely be the shortest. The new node information, "televel" and "belevel" has been calculated in Table 1, Table 2 and Table 3 for the three DAGs in Figure 1(a), (b) and (c) respectively.

For Figure 1(a),
Sum1=min[televel(n_3),tlevel(n_2)]+min[belevel(n_3)+blevel(n_2)]= min[15,5]+ min[18,29]=5+18=23
Sum2=min[televel(n_2),tlevel(n_3)]+min[belevel(n_2)+blevel(n_3)]= min[25,5]+ min[9,28]=5+9=14
So, node n_3 should be executed earlier than node n_2
The absolute value of the difference between sum1 and sum2 is 9, which is equal to the difference between the two different scheduling lengths (53-44=9).

For Figure 1(b),
Sum1=min[televel(n_3),tlevel(n_2)]+min[belevel(n_3)+blevel(n_2)]= min[25,5]+ min[18,29]=5+18=23
Sum2=min[televel(n_2),tlevel(n_3)]+min[belevel(n_2)+blevel(n_3)]= min[25,5]+ min[9,38]=5+9=14
So, node n_3 should be executed earlier than node n_2.
The absolute value of the difference between sum1 and sum2 is 9, which is equal to the difference between the two different scheduling lengths (63-54=9).

For Figure 1(c),
Sum1=min[televel(n_3),tlevel(n_2)]+min[belevel(n_3)+blevel(n_2)]= min[20,6]+ min[18,29]=6+18=24
Sum2=min[televel(n_2),tlevel(n_3)]+min[belevel(n_2)+blevel(n_3)]= min[26,10]+ min[9,28]=10+9=19
So, node n_3 should be executed earlier than node n_2
The absolute value of the difference between sum1 and sum2 is 5, which is equal to the difference between the two different scheduling lengths (54-49=5).

4.3 Time complexity of MPD Algorithm

Because there is another step to recompute the belevel and televel value of the independents nodes in a cluster, it is obvious that the time complexity of the proposed algorithm is O((|V|+|E|)*|V|), where |V| and |E| stand for the number of the nodes and the number of edges respectively. For the coarse grain DAGs, the complexity of this algorithm is comparable to that of the previous algorithms which need to compute the node information such as blevel and tlevel.

5. Experimental Results

A parallel program represented by a DAG is shown in Figure 2(a)[10], it has been merged into two clusters, PE0 and PE1. (In the PE1 cluster, because nodes n_1, n_2, n_5 and n_6 are in one cluster, their communication time is zero.) And their node information is shown in Table 4. From Figure 2(a), we can see that nodes n_3 and n_4 in the PE0 cluster are independent tasks, while in the PE1 cluster, there are two pairs of independent tasks -- n_1 and n_2, and n_2 and n_5. Based on the Maximized Parallelism Degree algorithm, the first node to be executed in each cluster is decided by the following formula.

For the PE0 Cluster:
Sum1=min[televel(n_3),tlevel(n_4)]+min[belevel(n_3),blevel(n_4)]=min[3,3]+min[2,4]=3+2=5
Sum2=min[televel(n_4)+tlevel(n_3)]+min[belevel(n_4),blevel(n_3)]=min[5,2]+min[2,3]=2+2=4
So, node n_3 should be executed earlier than node n_4
The absolute value of the difference between sum1 and

sum2 is 1, which is equal to the difference between the two different scheduling lengths (8-7=1).

For the PE1 cluster:
Sum1=min[televel(n_1),tlevel(n_2)]+min[belevel(n_1), blevel(n_2)]=min[1,0]+min[6,5]=0+5=5
Sum2=min[televel(n_2),tlevel(n_1)]+min[belevel(n_2), blevel(n_1)]=min[4,0]+min[1,7]=0+1=1
So, node n_1 should be executed earlier than node n_2,

The absolute value of the difference between sum1 and sum2 is 4, which is equal to the difference between the two different scheduling lengths (12-8=4).

Add a dashed line from n_1 to n_2, and then recompute the node information.
Sum1=min[televel(n_2),tlevel(n_5)]+min[belevel(n_2), blevel(n_5)]=min[5,1]+min[1,2]=1+1=2
Sum2=min[televel(n_5),tlevel(n_2)]+min[belevel(n_5), blevel(n_2)]=min[2,1]+min[1,5]=1+1=2
So nodes n_2 and n_5 are randomly selected to schedule.

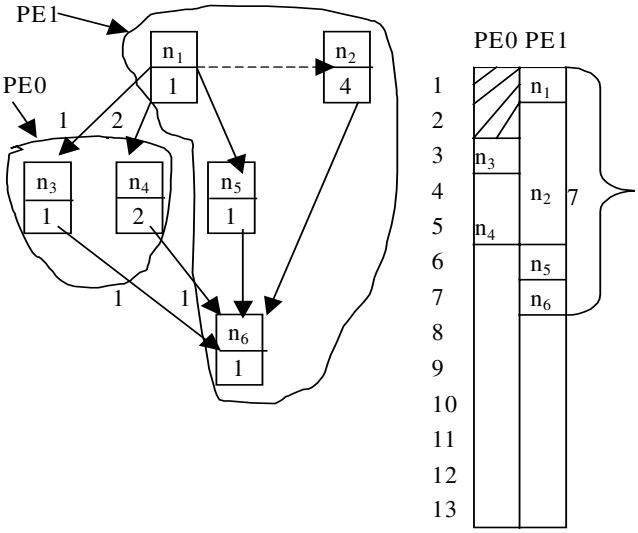
The scheduling Gantt graph based on the presented algorithm is shown as Figure 2(b), where the scheduling length is 7. If we randomly select the node to be executed during the clusters, then its scheduling length could be 12, as shown as Figure 2(c). But if we schedule the tasks in the cluster based on the previous algorithms, then node n_4 should be executed earlier than node n_3, and the scheduling length is 8 which is shown in Figure 2(d).

6. Conclusion

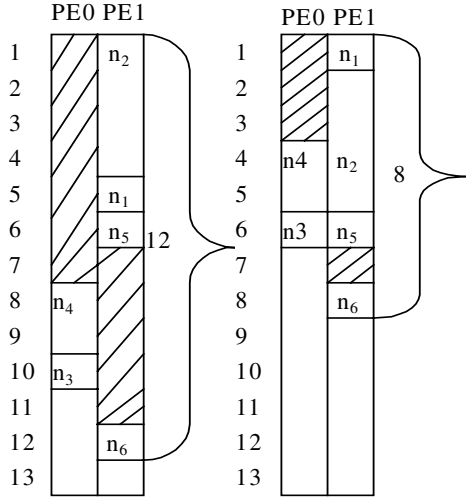
Motivated by the shortcomings of the current independent tasks scheduling strategies under nonlinearly clustering, this paper first points out why the current scheduling algorithms cannot produce the shortest scheduling length. It is because that they cannot maximize the whole parallelism degree of the independent tasks in the clustered DAG. In order to achieve a better result, this paper proposes to extract new node information from the DAG and a new task scheduling algorithm called MPD which is based on the Maximized Parallelism Degree. Experimental results show that the presented algorithm is better than any other scheduling algorithm.

Nodes / Node value	n_1	n_2	n_3	n_4	n_5	n_6
tlevel	0	0	2	3	1	6
blevel	7	5	3	4	2	1
(btlevel) blevel-tlevel	7	5	1	1	1	-5
Static_blevel	4	5	2	3	2	1
televel	1	4	3	5	2	7
belevel	6	1	2	2	1	0

Table 4 Node Information from Figure2(a)



(a) two clusters of the DAG (b) scheduling length (7)



(c) scheduling length (12) (d) scheduling length (8)

Figure2 Three Gantt Graphs based on different independent tasks scheduling algorithms

References

[1] Haluk Topcuoglu, Salim Hariri, Min-You Wu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems, V13(3): 260-274 , 2002.

[2] Yu-Kwong Kwok and Ishfaq Ahmad, Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors, ACM Computing Surveys, 1999, v31(4):406-471.

[3] Tao Yang and Apostolos Gerasoulis, DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors ， IEEE Transactions on Parallel and Distributed Systems, 1994, V5(9):951-967.

[4] Tao Yang and Apostolos Gerasoulis, PYRROS: Static Task Scheduling and Code Generation for Message Passing Multiprocessors, Proceedings of 6th ACM International Conference on Supercomputing (ICS 92):428-437, 1992, Washington D.C.

[5] Apostolos Gerasoulis and Tao Yang, On the Granularity and Clustering of Directed Acyclic Task Graphs, IEEE Transactions on Parallel and Distributed Systems, 1993, V4(6):686-701.

[6] B. Kruatrachue and T. Lewis: Grain Size Determination for Parallel Processing. IEEE Software:23-32, 1988.

[7] Yu-Kwong Kwok, Ishfaq Ahmad: Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems, V7(5): 506-521, 1996.

[8] J. Liou and M.A. Palis. An Efficient Clustering Heuristic for Scheduling DAGs on Multiprocessors. Proc. Symp. Parallel and Distributed Processing, 1996.

[9] R.L. Graham, Bounds on Multiprocessing Timing Anomalies, SIAM J. Appl. Math., V17(2), March 1969:417-429.

[10] Yu-Kwong Kwok and Ishfaq Ahmad, Benchmarking and Comparison of the Task Graph Scheduling Algorithms. Journal of Parallel and Distributed Computing 59(3): 381-422, 1999.

[11] Qiangsheng Hua and Zhigang Chen, Efficient Granularity and Clustering of the Directed Acyclic Graphs, Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'03), IEEE Press, pp. 625-628, 2003, Chengdu, China.

[12] Chen Zhi-gang and Hua Qiang-Sheng，EZDCP:A new static task scheduling algorithm with edge-zeroing based on dynamic critical paths，Journal of Central South University of Technology (English Edition), V10(2):140-144, 2003.