一个调度 Fork-Join 任务图的最优算法*

李庆华[1], 阮幼林[1+], 刘 干[2], 蒋盛益[1], 杨世达[1]

[1](华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

[2](华中科技大学 电子与信息工程系,湖北 武汉 430074)

An Optimal Scheduling Algorithm for Fork-Join Task Graphs

LI Qing-Hua[1], RUAN You-Lin[1+], LIU Gan[2], JIANG Sheng-Yi[1], YANG Shi-Da[1]

[1](Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

[2](Department of Electronics and Information, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: Phn: +86-27-62711038, Fax: +86-27-87544471, E-mail: ruanyl@263.net, http://www.hust.edu.cn

Received 2003-07-07; Accepted 2004-07-21

Li QH, Ruan YL, Liu G, Jiang SY, Yang SD. An optimal scheduling algorithm for fork-join task graphs. ***Journal of Software*****, 2005,16(5):684−690.** DOI: 10.1360/jos160684

Abstract: The Fork-Join structure is one of the basic modeling structures for parallel processing. Although some algorithms are able to find an optimal schedule under certain conditions, they ignore to economize processors and minimize the total completion time. This paper presents a Task Duplication based Balance Scheduling(TDBS) algorithm which can generate an optimal schedule for fork-join task graph with a complexity of $O(vq+v\log v)$, where v and q are the number of tasks and processors respectively. By considering workload and idle time slots of the used processors, TDBS algorithm tries to assign tasks to scheduled processors and maximize their utilization. Simulation results show that TDBS algorithm has better speedup and efficiency than other compared algorithms. Therefore, TDBS algorithm is a viable option for practical high performance applications.

Key words: task scheduling; task duplication; fork-join task graph; speedup

摘 要: Fork-Join 任务图是一种并行处理的基本结构.虽然许多算法在任务满足某些条件时能产生最优调度,但往往没有考虑节省处理器个数和减少任务集的总完成时间,从而降低算法的加速比和效率.因此,提出一种基于任务复制的平衡调度算法,其时间复杂度为 $O(vq+v\log v)$,v 和 q 分别表示任务集中任务的个数和使用的处理器个数.通过分析已用处理器的负载和空闲时间段,把任务尽量分配到已用的处理器上以均衡负载,从而提高其利用率.实验结果表明,该算法的加速比和总体效率优于其他算法.因此,该算法对于高性能应用程序的调度是一个较好的选择.

* Supported by the National Natural Science Foundation of China under Grant No.60273075 (国家自然科学基金)

LI Qing-Hua was born in 1940. He is a professor and doctoral supervisor at the Huazhong University of Science and Technology. His research areas are parallel and grid computing. **RUAN You-Lin** was born in 1972. He is a Ph.D. candidate at the Huazhong University of Science and Technology. His current research areas are data mining and parallel computing. **LIU Gan** was born in 1969. He is a Ph.D. candidate at the Huazhong University of Science and Technology. His current research areas are quality of service (QoS) sensitive computing in broadband wireless networks, multimedia communications systems. **JIANG Sheng-Yi** was born in 1963. He is an associate professor and Ph.D. candidate at the Huazhong University of Science and Technology. His current research areas are data mining and algorithm design. **YANG Shi-Da** was born in 1969. He is a Ph.D. candidate at the Huazhong University of Science and Technology. His current research areas are genetic algorithm and parallel computing.

关键词: 任务调度;任务复制;Fork-Join 任务图;加速比
中图法分类号: TP301 文献标识码: A

1 Introduction

Efficient scheduling of parallel programs onto processors is extremely difficult and important. The goals of a scheduling algorithm are to efficiently utilize resources and to achieve performance objectives of the application. Since it has been shown that the multiprocessor scheduling problem is NP-complete, many researchers have proposed scheduling algorithms based on heuristics[1−9]. It is well known that the complexity and quality of a scheduling algorithm largely depend on task graph structure[3,4]. In this paper, we consider the scheduling of a class of fork-join task graphs[4]. The fork-join structure is one of the basic modeling structures for parallel processing. Many parallel programs exhibit in the fork-join structure, and this type of parallel program paradigm arises in many application areas.

The task duplication based scheduling is a new approach to the scheduling problems. Since the communication time between tasks assigned to the same processor is considered to be negligible, task duplication is one way of reducing the interprocessor communication overhead. By using this approach, some of the more critical tasks of a parallel program are duplicated on more than one processor. This can potentially reduce the start time of the waiting tasks and eventually improve the overall completion time of the entire program. It has been shown that task duplication based scheduling algorithms always perform better than any other scheduling algorithms with no task duplication[4,6].

There are several task duplication based scheduling schemes. Although TDS algorithm[5] is able to find an optimal schedule under certain conditions, it ignores to economize the processors so that the speedup of parallel programs is decreased. Task Scheduling Algorithm(TSA-FJ)[7] only tries to utilize the idle time slot of the first processor, but the workloads of other processors are not considered. TSA-OT algorithm[8] assigns the task nodes of the critical paths to processors at first, and allocates the rest nodes to the used processors if possible without changing the scheduling length. However its complexity is very high. A low-complexity algorithm ELS[9] is an extension of the list scheduling, and its performance fluctuates with task graph and cannot be guaranteed.

A Task Duplication based Balance Scheduling (TDBS) algorithm is introduced in this paper. It is proven that the proposed algorithm can generate an optimal schedule for fork-join task graphs with high speedup and efficiency. The rest of the paper is organized as follows. Section 2 describes the system model and the problem definition. Section 3 presents the design principles of our approach, followed by a description of the proposed algorithm and some of its properties. Finally, we present an example to demonstrate the operation of the algorithm. Section 4 includes the performance results and comparisons with other algorithms. We provide concluding remarks in the last section.

2 System Model and Problem Statement

A parallel program usually represented by a Directed Acyclic Graph(DAG), which is also called a task graph. A DAG consists of a tuple (V,E,w,c), where V,E,w,c are the set of tasks nodes, the set of communication edges, the set of computation costs associated with the task nodes, and the set of communication costs associated with the edges respectively. $w(n_i)$ is the computation cost for task n_i and $c(n_i,n_j)$ is the communication cost for edge $e(n_i,n_j)\in E$ that connects task n_i and task n_j. The edge $e(n_i,n_j)$ represents the precedence constraint between task n_i and task n_j. In other words, task n_j can start only after the completion of task n_i. For task n_i, $st(n_i)$ and $ct(n_i)$ denote the start time

and finish time of task node n_i respectively. Therefore, the schedule length is defined as the finish time of exit node n_{exit}. The major objective of scheduling algorithm is to minimize $ct(n_{exit})$.

A few terms are defined in the following for a more clear presentation. For each processor p, $ct(p)$ denotes the finish time of last task node n_{end} assigned to p, that is $ct(p)=ct(n_{end})$. For each task, indegree is the number of input edges and outdegree is the number of output edges. For example, in Fig.1, the indegree of node n_6 is 4 and the outdegree of node n_1 is 4.

Definition 1. A node is called a fork node if its outdegree is greater than 1.

Definition 2. A node is called a join node if its indegree is greater than 1.

Note that a node can be neither a fork nor a join node, i.e., both of the node's indegree and outdegree are one, such as nodes n_2, n_3, n_4, and n_5 shown in Fig.1. The fork-join structure is one of the basic modeling structures for parallel processing. A fork-join task graph is a hybrid of an in-tree task graph and an out-tree task graph, which is shown in Fig.1. It has a fork node that spawns a number of children nodes. The output edges of the children nodes are connected to a join node. An out-tree task graph is a connected graph in which every node has only one parent node. An in-tree task graph is a connected graph in which every node has only one child node. Both of them can represent some divide-and-conquer algorithms.

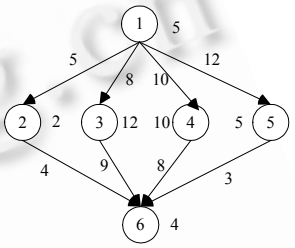

Fig.1 An example of the fork-join task graph

3 The Proposed Algorithm

Although the main performance measure of a scheduling algorithm is the schedule length, speedup and efficiency are also important. The main drawback of TDS algorithm is that it does not allow two or more parent tasks of a join task to be scheduled in the same processor, which only merge one parent task without considering other parent tasks, thus losing more chances of finding shorter schedule and wasting many processors. TSA_FJ algorithm only tries to utilize the idle time slot of the first processor, but the workloads of other processors are not considered. Although TSA-OT algorithm allocates no critical nodes to the used processors if possible without changing the scheduling length, its complexity is very high. On the contrary, ELS algorithm is the extension of the list scheduling, which has low-complexity, but its performance fluctuates with task graph and cannot be guaranteed. Therefore, the purpose of the proposed algorithm is to improve speedup and efficiency with low-complexity by combining advantages of the above algorithms. So, we propose an efficient algorithm whose schedule length and processors are minimized. By considering workloads and idle time slots of each used processor, it tries to assign tasks to the scheduled processors and maximize their utilization while minimizing the number of processors used. The following is the steps of the proposed algorithm.

Input: DAG(V,E,w,c)
Output: Schedule
Begin:
1. Sort the tasks from n_2 to n_{m-1}, then find the critical tasks.
2. Compute $st(n_i)$ and $ct(n_i)$ for all tasks ($n_i \in V$)
3. Assign tasks to processors (refer to Fig.3)
End

Fig.2 Pseudocode of scheduling the algorithm

The pseudocode in Fig.2 shows the steps involved in the algorithm. In step one, sort the tasks from n_2 to n_{m-1} in descending order by the sum of computation cost $w(n_i)$ and communication cost $c(n_i,n_m)$. As we show below:

$$w(n_2)+c(n_2,n_m) \geq w(n_3)+c(n_3,n_m) \geq \ldots \geq w(n_{m-1})+c(n_{m-1},n_m)$$

then find out the most critical task s($1<s<m$) for generating the optimal schedule.

where $\sum_{i=2}^{s} w(n_i) \le w(n_s)+c(n_s,n_m)$ and $\sum_{i=2}^{s+1} w(n_i) > w(n_{s+1})+c(n_{s+1},n_m)$.

In Step 2, the task graph is traversed to compute the *st* and *ct* of each task that in performed using (1) and (2). The computation of the start and completion times proceeds in a top-down fashion, starting with the entry task and terminating at the exit task.

$$st(n_i)=\begin{cases}0 & \text{if} \quad i=1\\ ct(n_{i-1}) & \text{if} \quad i\le s\\ ct(n_1) & \text{if} \quad s<i<m\\ \max\{ct(n_s),ct(n_{s+1})+c(n_{s+1},n_m)\} & \text{if} \quad i=m\end{cases} \tag{1}$$

$$ct(n_i)=st(n_i)+w(n_i) \tag{2}$$

According to the *st* and *ct*, each task should be inserted into a used processor if possible in step 3 as shown in Fig.3. If inserting decreases the *st* of the task, it means that the inserting condition is satisfied. If the inserting condition cannot be satisfied, the task must be assigned to a new processor with the entry task n_1. If all tasks are assigned to processors, then the algorithm terminates.

```
Output: Schedule
  Begin
     k=1; ct[k]=0;
     for(i=1; i<=s; i++) assign n_i to P_k;
     ct[k]=ct[n_s];
     for(i=s+1; i<m; i++)
            {success=false;
           for(j=1; j<=k; j++)
                    {if ((j=1) and (w(n_i)<=st[n_m]-ct[p])) or ((j<>1) and (st[n_m]-ct[p]>=w(n_i)+c(n_i,n_m)))
                 {st[n_i]=ct[p]; ct[n_i]=st[n_i]+w(n_i);
                  ct[p]=ct[n_i]; success=true; break;
                  }
                    }
           if (success) {assign n_i to P_j; ct[j]=ct[n_i];}
          else {k++; assign n_1, n_i to P_k; ct[k]=ct[n_i];}
                }
     assign n_m to P_1;
  End
```

Fig.3 Details of step 3 of TDBS algorithm

3.1 Complexity analysis

In the first step, the tasks from n_2 to n_{m-1} can be sorted by the merging algorithm whose complexity would be $O(v\log v)$, where v is the number of tasks. The second step of the proposed algorithm is to traverse each task of the task graph and compute its start and completion times. Thus, the complexity of the step would be $O(v)$. In the last step, for each task, all the used processors may be examined. Thus, the worst case complexity of the step would be $O(qv)$, where q is the number of the used processors. $O(v)$ is negligible compared to $O(qv)$ and $O(v\log v)$. Thus, the overall time complexity of the TDBS algorithm is $O(v\log v+qv)$. If $\log v>q$, the time complexity is $O(v\log v)$, otherwise is $O(qv)$.

3.2 An example

In this section, we use an example of fork-join task graph (see Fig.1) to illustrate the effectiveness of the proposed algorithm.

1) By sorting, results of the first step are shown in Fig.4.

2) For each task n_i, the $st(n_i)$ and $ct(n_i)$ can be computed by using (1) and (2). The *st* and *ct* of all tasks of the DAG are shown in Table 1.

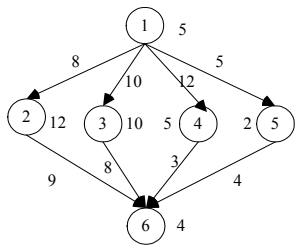

Fig.4 Fork-join task graph after sorting

3) At first, assign task n_i ($1 \le i \le s=2$) to processor 1. For the following each task n_i($2<i<m$), the algorithm searches the scheduled processors to select which processor suits for inserting the task. If the inserting condition is satisfied, the algorithm would modify $st(n_i)$ and $ct(n_i)$ of the task and assign the task to the suited processor, such as tasks n_4 and n_5. Otherwise, it assigns both entry task n_1 and the task n_i to a new processor, such as task n_3.

4) When all tasks are assigned to processors, the final start and completion times of each task obtained by the proposed algorithm are also shown in Table 1.

Table 1 The first and final start and completion times for the tasks

Task	*st*(first)	*ct*(first)	*st*(final)	*ct*(final)
1	0	5	0	5
2	5	17	5	17
3	5	15	5	15
4	5	10	17	22
5	5	7	15	17
6	23	27	23	27

In the end, the processor allocation and the schedule times obtained by TDBS algorithm are shown in Fig.5. For this example, it can be obtained that the schedule length is equal to c*t* of the exit task n_6. (shadow denotes the idle time slot)

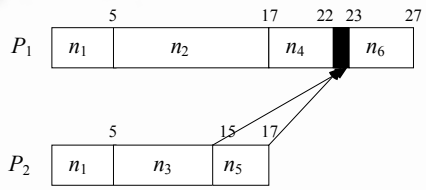

Fig.5 Results of scheduling

4 Performance and Comparison

The proposed algorithm can be applied to any fork-join task graph and generate an optimal schedule. We can prove that the TDBS algorithm generates an optimal schedule whose length is equal to

$$w(n_1)+\max\left\{\sum_{i=2}^{s} w(n_i), w(n_{s+1},n_m)\right\}+w(n_m)$$

where $\sum_{i=2}^{s} w(n_i) \le w(n_s)+c(n_s,n_m)$ and $\sum_{i=2}^{s+1} w(n_i) > w(n_{s+1})+c(n_{s+1},n_m)$

Proof. When n_m is considered for scheduling, task n_2 is the critical task that will be considered for duplication. After duplication, the start time of n_m will be reduced because $c(n_2,n_m)$ is zeroed. The start time of n_m is now constrained by data from n_3, i.e. $st(n_m)=w(n_1)+\max\left\{\sum_{i=2}^{2} w(n_i), w(n_3)+c(n_3,n_m)\right\}=w(n_1)+\{w(n_3)+c(n_3,n_m)$.

$st(n_m)=w(n_1)+\max\left\{\sum_{i=2}^{2} w(n_i), w(n_3)+c(n_3,n_m)\right\}=w(n_1)+w(n_3)+c(n_3,n_m)$ because of $\sum_{i=2}^{2} w(n_i) \le w(n_3)+c(n_3,n_m)$.

Thus, n_3 becomes the new critical task for considering and is also duplicated. This process is repeated until n_{s+1} is considered. At that point, the start time of n_m will increase if n_{s+1} is duplicated. So, $st(n_m)=w(n_1)+\max\left\{\sum_{i=2}^{s} w(n_i), w(n_{s+1})+c(n_{s+1},n_m)\right\}$. Thus, the schedule length is optimal because n_m cannot start any earlier.

In the following, the fork-join task graph shown in Fig.1 is used to compare the TDBS algorithm with other

algorithms. The schedule length generated by each of the algorithms and their complexities and number of required processors are shown in Table 2.

Table 2 Comparison with other algorithms for DAG in Fig.1

Parameter	TDBS	TDS	TSA_FJ	TSA-OT	ELS
Schedule Length	27	27	27	27	27
Number of processors	2	4	3	3	3
Complexity	$O(qv+v\log v)$	$O(v^2)$	$O(v)$	$O(ev^2)$	$O(v\log(p)+e)$

Next, we present the comparative evaluation of the proposed algorithm, TDS algorithm, TSA_FJ algorithm, TSA-OT algorithm and ELS algorithm. We consider the randomly generated application graphs as the workload for testing the algorithms. In each graph, the computation costs of the individual nodes are randomly selected from a uniform distribution with mean equal to the chosen average computation cost. Similarly, the communication costs of the edges are randomly selected from a uniform distribution with mean equal to the average communication cost.

As shown in Table 3, for the randomly generated fork-join task graphs, the schedule results of TDBS are compared with those of other four algorithms for the generated task graphs. TDS algorithm assigns parents tasks of a join task to different processors, which not only loses the chance of finding shorter schedules but also ignores to economize the processors. TSA_FJ algorithm only considers the workload of the first processor. Thus, the NoP and TCCT of both algorithms are more than other algorithms, especially the number of task is very large. Because of allocating the rest nodes to the used processors as possible, TSA-OT algorithm has a better performance than other algorithms except TDBS algorithm. ELS algorithm is a kind of list scheduling, its performances vary with task graphs. From the comparison in Table 3, we can see that TDBS algorithm has the shortest schedule length, much less processors and total completion time, and has low complexity. Based on these experiments, the TDBS algorithm outperforms the other four algorithms for any graph size in terms of the schedule length and used processors. Thus, the speedup and efficiency of the TDBS algorithm are better than other algorithms. By considering workload and idle time slots of processors, TDBS algorithm tries to assign tasks to the scheduled processors and maximize their utilization while minimizing the number of processor used. Thus, when the number of tasks is larger, the used processors are fewer, and the advantage of TDBS algorithm is more evident. Therefore, the TDBS algorithm is more practical than other compared algorithms. (SL denotes the schedule length; NoP denotes the number of processor; TCCT denotes the total computation and communication times)

Table 3 Comparison of schedule results with other algorithms

m	Parameter	TDBS	TSA_FJ	TDS	TSA-OT	ELS
20	SL	48	53	58	48	50
	TCCT	253	299	342	268	284
	NoP	12	15	18	13	14
50	SL	54	58	66	56	60
	TCCT	568	778	843	644	678
	NoP	28	44	48	32	34
100	SL	60	62	68	60	62
	TCCT	1206	1628	1724	1328	1420
	NoP	54	94	98	58	62
200	SL	64	68	74	68	72
	TCCT	2248	3446	3620	2542	2746
	NoP	82	194	198	92	102
400	SL	70	76	80	74	76
	TCCT	4284	6928	7468	4926	5334
	NoP	132	392	398	144	158

5 Conclusions

This paper presents an optimal algorithm to schedule tasks of fork-join graph onto processors, which is based

on task duplication and load balancing of processors. TDBS algorithm has a complexity of $O(vq+v\log v)$, where v and q are the number of tasks and processors respectively. The performance of TDBS has been compared with other algorithms in terms of its complexity, the schedule length, total computation and communication times, and the number of used processors. Simulation results showed that TDBS algorithm has better speedup and efficiency than other compared algorithms. Therefore, TDBS algorithm is a viable option for practical high performance applications.

References:

[1] Yang T, Gerasoulis A. DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Trans. on Parallel and Distributed Systems, 1994,5(9):951−967.

[2] Kwok YK, Ahmad I. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. IEEE Trans. on Parallel and Distributed Systems, 1996,7(5):506−521.

[3] Palis MA, Jing-Chiou Liou, Wei DSL. Task clustering and scheduling for distributed memory parallel architecture. IEEE Trans. on Parallel and Distributed Systems, 1996,7(1):46−55.

[4] Ahmad I, Kwok YK. On exploiting task duplication in parallel program scheduling. IEEE Trans. on Parallel and Distributed Systems, 1998,9(9):872−891.

[5] Darbha S, Agrawal DP. Optimal scheduling algorithm for distributed-memory machines. IEEE Trans. on Parallel and Distributed Systems, 1998,9(1):87−94.

[6] Chan-Ik Park, Tae-Young Choe. An optimal scheduling algorithm based on task duplication. IEEE Trans. on Computers, 2002, 51(4):444−448.

[7] Liu ZY, Fang BX, Jiang Y, Zhang Y, Zhao H. A new algorithm for scheduling fork-join task graph. Journal of Software, 2002,13(4):693−696 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/13/693.pdf

[8] Liu ZY, Fang BX, Zhang Y. TSA-OT: An algorithm scheduling an out-tree DAG. Chinese Journal of Computers, 2001,24(4): 390−394 (in Chinese with English abstract).

[9] Andrei Radulescu, Arjan JC, van Gemund. Low-Cost task scheduling for distributed-memory machines. IEEE Trans. on Parallel and Distributed Systems, 2002,13(6):648−658.

附中文参考文献:

[7] 刘振英,方滨兴,姜誉,张毅,赵宏.一个调度 Fork-Join 任务图的新算法.软件学报,2002,13(4):693−697. http://www.jos.org.cn/1000-9825/13/693.pdf

[8] 刘振英,方滨兴,张毅.TSA-OT:一个调度 Out-Tree 任务图的算法.计算机学报,2003,14(2):190−196.