一个调度 Fork-Join 任务图的新算法*

刘振英[1], 方滨兴[1], 姜 誉[1], 张 毅[2], 赵 宏[1]

[1](哈尔滨工业大学 计算机科学与工程系,黑龙江 哈尔滨 150001);
[2](哈尔滨理工大学 电气与电子工程系,黑龙江 哈尔滨 150040)
E-mail: jy@pact518.hit.edu.cn
http://www.hit.edu.cn

摘要: 任务调度是影响工作站网络效率的关键因素之一.Fork-Join 任务图可以代表很多并行结构,但其他已有调度 Fork-Join 任务图算法忽略了在非全互连工作站网络环境中通信之间不能并行执行的问题,有些效率高的算法又没有考虑节省处理器个数的问题.因此,专门针对该任务图,综合考虑调度长度、非并行通信和节省处理器个数问题,提出了一个基于任务复制的静态调度算法 TSA_FJ.通过随机产生任务的执行时间和通信时间,生成了多个 Fork-Join 任务图,并且采用 TSA_FJ 算法和其他调度算法对生成的任务图进行调度.结果表明,TSA_FJ 算法的调度长度最短、使用的处理器个数最少,它更适合于非全互连的 NOW 环境.

关 键 词: 任务调度;关键路径;调度长度;DAG
中图法分类号: TP316 文献标识码: A

在并行语言的发展过程中,出现了多种多样的并行表达方式.比如:HPF 中的 doall 结构、CC++中的 par 和 parfor 结构,Multi-PASCAL 中的 cobegin 和 coend 结构.我们发现,它们都有一个共同的特征,就是它们都可以被抽象成 Fork-Join 任务图.因此,解决好此类任务图的调度问题,对于实现并行语言的编译器、提高并行计算的性能具有很重要的作用.

近年来,工作站网络(NOW)是一个研究热点.对于 NOW 来说,任务调度问题是一个关键问题,同时它也是一个 NP 完全问题.对此,研究者们做了大量的工作.1996 年,Kwok 和 Ahmad 提出了一个动态关键路径(DCP)调度算法[1].1999 年,张宏莉等人针对单并发任务簇提出了一个最优分配算法 OPTA[2],而单并发任务簇就等同于 Fork-Join 任务图.DCP 和 OPTA 是两个较好的调度算法.1998 年,Darbha 和 Agrawal 提出了一个基于任务复制(TDS)的调度算法[3].TDS 算法在调度长度(schedule length)方面,基本上比在它以前出现的所有算法都优越,因为它通过任务复制减少了通信时间.但是,TDS 算法没有考虑节省处理器个数的问题,这会降低算法的加速比.

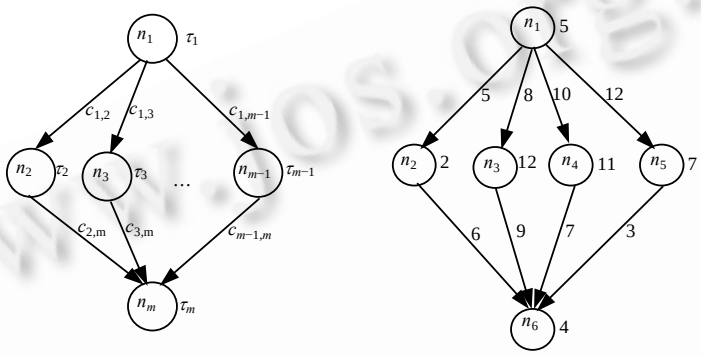
上述调度算法中都存在这样一个问题:允许通信重叠,即可以并行执行通信.或者说,在安排任务的起始执行时间时,只考虑了该任务接收最长消息所需的时间,而此时任务所需的消息还没有全部接收到就开始执行.但是,在一个 NOW 环境中,如果采用的不是全互联的网络,如总线网,每个处理器在发送和接收多个消息时,实质上都是串行进行的.因此,以上的调度算法与实际情况不符.为了更好地适应 NOW,我们提出了一个新的静态调度算法 TSA_FJ.该算法的目标是调度长度最短,采用的处理器个数最少.该算法假定:同一处理器的通信和通信之间必须串行执行.

* 收稿日期: 2000-04-04; 修改日期: 2001-08-21
基金项目: 国家"九五"国防预研基金资助项目(16.6.2.5)

作者简介: 刘振英(1972－),女,黑龙江阿城人,博士,讲师,主要研究领域为并行计算,并行编译;方滨兴(1960－),男,江西万年人,博士,教授,博士生导师,主要研究领域为计算机网络,计算机系统结构,并行计算;姜誉(1968－),男,山东乳山人,高级工程师,主要研究领域为并行与分布计算,计算机网络;张毅(1971－),男,贵州水城人,工程师,主要研究领域为网络计算,数据库技术;赵宏(1968－),男,黑龙江哈尔滨人,工程师,主要研究领域为并行计算.

1 问题描述

当一个任务群在一个NOW上运行时,我们假定,该NOW中可用处理器数目是无限的.不同的处理器利用初始数据执行同一任务的不同拷贝,同一任务的不同副本所生成的数据是相互一致的.我们用一个无环有向图(DAG)来抽象描述任务图.这个 DAG 可以用一个四元组(V,E,τ,c)来表示.其中 $V=\{n_1,n_2,\ldots,n_m\}$是任务集,$v=|V|$表示任务的个数;E 是边的集合,$e=|E|$表示边的个数;τ是计算开销集合,τ_i 是任务 $n_i(n_i\in V)$的计算开销;c 是通信开销集合,对于每条边,如果从任务 n_i 到任务 n_j 的边 $e(n_i,n_j)\in E$,则它的通信开销为 $c_{i,j}$.任务的执行时间τ_i可以由任务中的程序代码的计算量来衡量,它是一个静态的参量,可以通过编译器在用户代码的编译过程中得到.在应用程序中,任务间的通信量是由应用程序问题本身决定的,任务间通信以及对执行性能的影响主要分为如下几类:同步等待、数据依赖、执行完成等待.Fork-Join 任务图是一类特殊的任务图,它的特点是,有一个根节点,它与多个子任务之间一一进行通信,而这些子任务又要与一个惟一的叶结点进行通信.Fork-Join 任务图如图 1 所示.



(a) A common DAG of Fork-Join (b) An example of a Fork-Join DAG

(a) 一个通用的 Fork-Join 的 DAG (b) 一个 Fork-Join 的 DAG 的例

Fig.1 The DAG of Fork-Join

图 1 Fork-Join 任务图

2 一个新的调度算法 TSA_FJ

在 NOW 中,如果采用的互连网不是全互连的,那么同一个处理器在接收和发送消息时,通信都是不能并行执行的,而在已有的算法中,都没有考虑这个问题,也很少考虑节省处理器个数的问题.从图 1 可以看出,一个 Fork-Join 任务图是由一个 Fork 和一个 Join 任务图组合而成的.在减少总的处理器个数方面,Fork 任务图的调度主要是通过复制公共的父任务结点,并且把子任务结点适当地组合起来.而 Join 任务图的调度则是让公共的子任务结点与具有最大计算和通信时间和的父结点处于同一处理器,并且在该处理器中适当地吸纳其他父结点.我们知道,采用任务复制的方法可以通过任务的冗余减少通信时间,这时,一个任务可能被同时分配到多个处理器上.基于上述分析,针对图 1(a)中的任务图,我们提出了一种新的基于任务复制的静态调度算法 TSA_FJ,综合 Fork 和 Join 调度算法的优点,综合考虑调度长度和使用的处理器个数问题,专门调度 Fork-Join 任务图.其中 TSA 是 Task Schedule Algorithm 的缩写,FJ 是 Fork-Join 的缩写.

算法 **1**. Fork-Join 任务图调度算法 TSA_FJ.

```
1.  k=1;
2.  x=y=j=z=0;
3.  将 n1 分配至处理器 P0;
4.  for (i=2;i<m;i++)
5.  {
6.      j=(τi>z)?(ci,m+y+τi−z):(ci,m+y);
7.      if (x+τi<j){
8.          将 ni 分配至处理器 P0;
9.          x=x+τi;/*x 存放 P0 上的累计任务执行时间*/
```

```
10.       }
11.       else{
12.             将 n1 和 ni 分配至处理器 Pk;
13.           if (k==1)   {y=ci,m;z=τi;}
14.             else{/*y 存放不在 P0 上的任务与 nm 的通信完成时间*/
15.                   if (τi≥z)   y+=ci,m+(τi−z);
16.                   else y+=ci,m;}
17.             k++;
18.       }
19. }
20. 将 nm 分配至处理器 P0.
```

TSA_FJ 算法主要由一个 for 循环构成,循环体的时间复杂度为 $O(1)$,因此,该算法的时间复杂度为 $O(m)$.由于 m 代表的是任务集中任务的个数,因此 TSA_FJ 算法的时间复杂度为 $O(v)$.

TSA_FJ 算法的原则是让 Fork-Join 任务图中的最后一个任务结点 n_m 尽早开始执行.有两个因素决定 n_m 任务的开始执行时间:一是与 n_m 处于同一处理器的其他任务的结束时间,在程序中用 x 来表示;另一个是与 n_m 不在同一处理器的任务向 n_m 通信的完成时间,用 y 表示.算法 TSA_FJ 在第 6、7、15、16 步同时考虑了这两个因素,以尽量缩短并行时间.对于如图 1(b)所示的 Fork-Join 任务图的例,算法 TSA_FJ 的调度结果如图 2 所示.任务被分配到 3 个处理器中,处理器 P_2 的计算最先完成,其完成时间是 12,通信时间为 3(此时该处理器已无计算任务),因此总的完成时间是 15;在处理器 P_0 中,n_6 前面的计算任务的完成时间是 19,处理器 P_2 与 P_0 通信过程中 P_0 的计算尚未完成,对 P_0 来说计算与通信重叠;处理器 P_1 的计算完成时间是 16,开始与 P_0 通信,通信时间为 7,总的完成时间是 23,此时对 P_0 来说,计算与通信重叠的时间是 3(19−16),无计算的通信时间为 4,所以 n_6 的起始执行时间是 max{15,23,23},而不是 max{15,23,29}.图 2 中 P_0 的阴影部分表示计算与通信重叠.

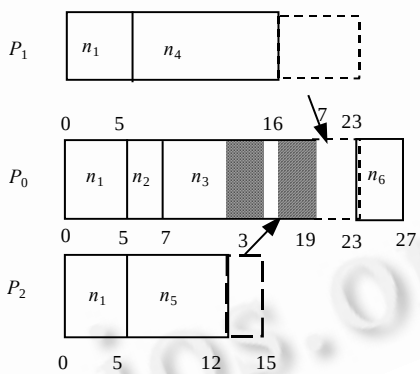


Fig.2 The scheduling results of algorithm TSA_FJ for a Fork-Join task graph in Fig.1(b)

图 2 算法 TSA_FJ 对图 1(b)的 Fork-Join 任务图的调度结果

3 相关工作的对比

利用前面提到的 DCP,OPTA,TDS 算法调度任务图 1(b),如果按照原来的调度算法进行分配,根据在非全互连网络中的通信必须串行执行的要求,我们只是改正一下每个任务的正确起始、终止执行时间,那么就得到表 1.从表 1 中调度算法的对比可以看出,TSA_FJ 算法虽然在调度长度上与 TDS 相当,但是 TSA_FJ 算法采用的处理器个数最少.另外,与其他算法相比,算法 TSA_FJ 和 TDS 在调度长度上占据很大优势的原因是采用了任务复制,即复制了任务 n_1,因此避免了其他结点与该复制任务结点的通信开销.总的来说,算法 TSA_FJ 要优于其他调度算法.

Table 1 Comparison between schedule results of TSA_FJ and other scheduling algorithms for a task graph in Fig.1(b)

表 1 TSA_FJ 算法与其他调度算法对图 1(b)任务图调度结果的比较

	TSA_FJ algorithm①	OPTA algorithm②	DCP algorithm③	TDS algorithm④
Schedule length⑤	27	45	47	27
Number of processors⑥	3	4	4	4
Complexity⑦	$O(v)$	$O(v)$	$O(v^3)$	$O(v^2)$

①TSA_FJ 算法,②OPTA 算法,③DCP 算法,④TDS 算法,⑤调度长度,⑥处理器个数,⑦复杂度.

另外,我们还采用了文献[4]中生成随机数的方法和程序,产生了 100 个 10~50 之间的随机数,并用这些随机数构造了 5 个分别由 5、6、7、8、9 个任务结点组成的 Fork-Join 任务图.这些随机数分别用来表示任务图中的任务执行时间和通信时间.针对这 5 个任务图,采用 TSA_FJ,OPTA,DCP,TDS 算法进行调度的结果见表 2.

Table 2 Comparison of schedule results of different algorithms for five Fork-Join task graphs generated from random numbers

表 2 不同算法对随机产生的 5 个 Fork-Join 任务图调度结果对比

m		TSA_FJ algorithm①	OPTA algorithm②	DCP algorithm③	TDS algorithm④
5	SL⑤	114	139	147	114
	P⑥	2	3	3	3
6	SL	163	189	180	163
	P	4	4	3	4
7	SL	166	230	196	181
	P	3	5	4	5
8	SL	167	251	233	218
	P	4	6	6	6
9	SL	218	325	297	331
	P	5	7	6	7

①TSA_FJ 算法,②OPTA 算法,③DCP 算法,④TDS 算法,⑤调度长度,⑥处理器个数.

在表 2 中,m 代表任务图中任务结点的个数.与 OPTA,DCP,TDS 调度算法相比,在以上 5 个任务图的调度结果中,TSA_FJ 的调度结果都是最优的,即调度长度最短、所用处理器个数最少,而且总结点数越多,TSA_FJ 算法就越优于其他算法.上述实验可以说明,TSA_FJ 算法是调度 Fork_Join 任务图的最优算法.

4 结 论

任务调度问题是一个 NP 完全问题,要想得到多项式时间的最优调度算法是极为困难的.因此,针对不同的任务图进行调度,是一种行之有效的方法.在已有的所提出的调度算法里,基于任务复制的方法可以通过任务的冗余减少通信时间,因此这类独特的调度方法是最佳的选择.由于在 NOW 中,如果采用的互连网不是全互连的,那么同一个处理器在接收和发送消息时,通信都是不能并行执行的,而在已有的算法中,都没有考虑这个问题.因此,本文针对这一问题,专门设计了一个静态调度算法 TSA_FJ,并且通过实验说明,在非全互连的 NOW 环境下,与其他的调度算法相比,该算法调度长度最短、所用处理器个数最少,并且任务图中的总任务数越多,TSA_FJ 就越优于其他算法.它更适合于 NOW,应用性更强.但是,本文提出的调度算法,仅适应于调度 Fork-Join 任务图.对于其他的任务图,仍需根据任务图的特性,研制出适合的调度算法.

References:

[1] Kwok, Y.K., Ahmad, I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1996,7(5):506~521.

[2] Zhang, Hong-li, Hu, Ming-zeng, Fang, Bin-xing, *et al*. A sub-optimal algorithm on allocating a single task cluster on NOWs. Journal of Computer Research and Development, 1999,36(9):1076~1079 (in Chinese).

[3] Darbha, S., Agrawal, D.P. Optimal scheduling algorithm for distributed-memory machines. IEEE Transactions on Parallel and Distributed Systems, 1998,9(1):87~95.

[4] Xu, Shi-liang. C Programs for Common Algorithms. 2ed, Beijing: Tsinghua University Press, 1999. 317~323 (in Chinese).

附中文参考文献:

[2] 张宏莉,胡铭曾,方滨兴,等.群机系统上单并发任务簇的近优分配算法.计算机研究与发展,1999,36(9):1076~1079.

[4] 徐士良.C 常用算法程序集.第 2 版,北京:清华大学出版社,1999.317~323.

A New Algorithm for Scheduling Fork-Join Task Graph*

LIU Zhen-ying[1], FANG Bin-xing[1], JIANG Yu[1], ZHANG Yi[2], ZHAO Hong[1]

[1](*Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin* 150001, *China*);

[2](*Department of Electrical and Electronic Engineering, Harbin University of Science and Technology, Harbin* 150040, *China*)

E-mail: jy@pact518.hit.edu.cn

http://www.hit.edu.cn

Abstract: Task scheduling is one of the crucial factors influencing the efficiency of a network of workstations. Fork-Join task graphs can represent many parallel structures. All of the existing algorithms which schedule Fork-Join task graphs have ignored the problem that communications cannot be executed in parallel in non-fully-connected NOW, and some of algorithms with high efficiency even did not take the problem of how to save the processors into account. In this paper, a new static task scheduling algorithm called TSA_FJ is proposed, which is based on task duplication and trying to synthetically take the problems of schedule length shortening, unparallel communications and processor saving into account. By randomly generating the task execution time and communication time, several Fork-Join task graphs are got and the scheduling results of TSA_FJ are compared with that of other algorithms for the generated task graphs. It shows that TSA_FJ algorithm has the shortest scheduling length and uses much less processors. It is much suitable to non-fully-connected NOW.

Key words: task scheduling; critical path; scheduling length; DAG

* Received April 4, 2000; accepted August 21, 2001

Supported by the Defence Pre-Research Project of 'Ninth Five-Year-Plan' of China under Grant No.16.6.2.5