CCTD:一种通信限制下的Fork-Join任务调度算法

梁珊珊 吴佳骏 张军超
(中国科学院计算技术研究所 北京 100080)

摘 要 现代并行系统的复杂调度问题可以转化为Fork-join图的任务调度问题。然而在实际计算环境中,两个处理节点之间的通信大多以独占方式进行,现有的大多数任务调度算法往往忽略了对通信信道独占性的考虑。提出了一种带通信限制的Fork-join图调度算法CCTD。该算法引入了实际环境中的通信独占性限制,同时保证了Fork-join图的基于复制的优化调度,而且尽可能地减少了对处理器占用。实验结果表明,CCTD算法是一种适应性强的、高效的Fork-join图调度算法。

关键词 任务调度,任务复制,fork-join,通信限制
中图法分类号 TP316 **文献标识码** A

CCTD:A Scheduling Algorithm under Communication Constraints for Fork-Join Task Graphs

LIANG Shan-shan WU Jia-jun ZHANG Jun-chao
(Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China)

Abstract All the complicated scheduling problems in modern parallel systems can be converted to the basic scheduling problem for Fork-Join task graphs. However, the communication between two processing nodes is carried on exclusively in real computation environment, which are normally ignored by most task scheduling algorithms. This paper presentsed a scheduling algorithm under communication constraints for Fork-Join task graphs. The algorithm provided a duplication based approach to schedule Fork-Join task graph while introducing the constraints of exclusive communications in reality. The number of processing nodes was also minimized compared to other scheduling algorithms. The experimental results show that CCTD is an efficient Fork-Join task graph scheduling algorithm, and fit for a broad range of scheduling problems.

Keywords Task scheduling, Task duplication, Fork-join task graph, Communication constraint

1 引言

并行程序的任务调度分为两种,一种是独立的任务调度,任务之间没有依赖关系;另外一种是任务之间存在依赖关系和任务间通信。对后一种任务类型,一般需要进行任务或线程划分,即将原先完整的程序划分成多个可以并行执行的任务。

任务调度的好坏直接影响系统的性能,如果调度不当,则很可能将并行的优点完全给抹煞掉,甚至比串行的效果还差。从调度的时机来看,可以分为静态和动态的。静态调度指的是完全由编译器在编译时决定,程序行为(包括各个任务运行时间、通信、数据依赖以及同步等)必须是编译时就已知的;动态调度则指由调度程序根据运行时情况动态地分配不同的任务到各个处理器上,以求尽量降低总运行时间,同时减少调度程序本身带来的负担[7]。在我们研究的对象中,由于各个任务是在编译阶段执行调度的,所以就不存在动态调度的问题,故本文将主要关注静态调度问题。

进行任务调度算法研究时,通常都是使用有向无环图(Directed Acyclic Graph)或者任务图(Task Graph)来表示被调度的并行程序。DAG一般可以表示为一个四元组$\langle V,E,w,c\rangle$,记为图$G=(V,E,w,c)$。其中V称为任务节点集合,$V=\{n_1,n_2,\cdots,n_v\}$,$v=|V|$,v为G中任务节点个数。E表示图中各个任务节点的通信边集合。$w(n_i)$表示任务节点n_i在处理时所需要的运行时间,$c(n_i,n_j)$为任务节点n_i和n_j之间通信时间。

在任务调度算法研究的有向无环图中,大量的研究工作集中在fork图和join图两种基本图结构上。这是因为,大多数的任务图都可以被分解为这两种基本图结构,所以通过对这两种图结构的相关任务图研究可以进一步地演化推导出各种对任意结构任务图的调度算法。本文也着重于fork和join图的任务调度算法研究。本文提出了一种带通信限制的Fork-join图调度算法CCTD(Communication Constrained Tasks Duplication)。该算法引入了实际并行系统中的通信独占性问题,使用基于复制的方法优化调度任务图。本文的实验结果表明,该算法在引入通信限制后,然后能够获得很好的调度性能,减少了处理器的消耗量,对不同测试用例具有较好的适应性。

本文第2节介绍任务图调度的相关研究工作;第3节描述我们的CCTD算法;第4节设计了对比性能实验,并分析了

到稿日期:2008-11-18 返修日期:2009-01-06
梁珊珊(1981-),女,硕士研究生,主要研究方向为任务调度、profiling、机器模型、软件测试,E-mail:liangss316@gmail.com;吴佳骏(1978-),男,博士,主要研究方向为编译优化、计算机体系结构等;张军超(1976-),男,博士,助理研究员,主要研究方向为编译优化、计算机体系结构等。

实验结果；最后给出了结论和以后的研究方向。

2 相关研究工作介绍

在任务调度算法中基于任务复制(TDB)的算法是调度性能较好的一种算法，它充分利用处理器的计算能力来减少通信开销，对于通信开销敏感任务调度，能获得显著的性能提升。许多已有的研究工作都是在基于复制的任务调度算法上进行进一步的研究。

Kwok 和 Ahmad 提出了一种 CPFD(Critical Path Fast Duplication)[1]算法。该算法定义了 3 种不同类型的节点：CPN(关键路径上的节点)、IBN(存在一条到达 CPN 的路径的节点)以及 OBN(其它节点)。CPFD 算法为这 3 种节点依次设定优先级顺序。CPN 节点直接影响任务的最终完成时间，具有最高优先级。IBN 节点有助于提前 CPN 节点开始的时间，具有次高的优先级。OBN 节点的优先级最低。CPFD 算法首先构建称为 CPN-Dominant 的序列，然后按照序列顺序逐步调度任务节点。该算法具有很好的调度性能，但其时间复杂度较高，为 $O(n^4)$。

Darbaha 和 Agrawal 提出了 TDS(Task Duplication based Scheduling)[2]算法。TDS 算法依赖任务的最早开始时间和最迟完成时间之间的差值来找出任务的关键节点，然后复制任务的父节点以提早任务的开始时间，从而完成任务分配。TDS 算法为每个任务节点设定了最早开始时间、最早完成时间等属性，通过计算节点的这些属性来作为调度的依据。

以上的算法没有考虑现实并行计算环境中处理器间的通信信道独占性。它们允许通信是可以重叠的，然而实际中的任务大多无法并行处理数据通信，而是必须串行执行。刘振英等提出了一种 TSA_FJ(Task Schedule Algorithm Fork-Join)算法[3]。该算法着眼于解决任务调度中被忽略的通信信道使用冲突的问题。在大多数任务调度算法中都允许并行执行通信，在安排任务的起始执行时间时，仅考虑该任务接受最长消息需要的时间，实际任务需要的消息并没有全部获取就开始执行。但是现实中的并行计算系统处理器在接受发送消息时是串行进行的。然而，TSA_FJ 算法并不能保证获得 Fork-join 图的较优调度性能。

3 CCTD 任务调度算法

图 1 给出了一个基本的 fork 图和 join 图，分别如图 1(a)和图 1(b)所示。图 1(c)是一个具体的 Fork-join 图实例。在文献[4]中提到了针对 fork 图的研究，文献[5]对 fork 图和 join 图分别给出了非复制模式下处理器数目无限的全相连条件下，它们的最优化调度结果。

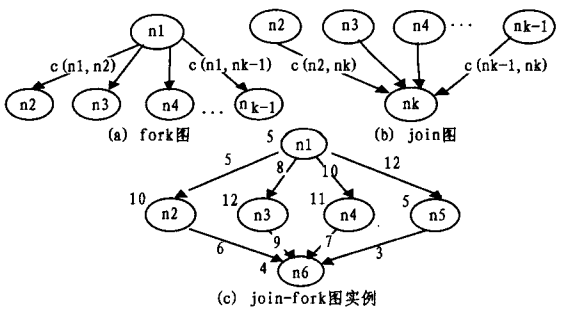

图 1 基本 Fork-join 图

由于 fork 图每个任务节点至多有一个前驱，那么在基于任务复制的调度下直接复制任务节点的前驱就可以消除通信开销保证任务节点得到最早开始时间，从而保证调度的性能最优。我们以调度长度(schedule length，S)来表示调度算法的性能。那么，fork 图的最优化调度性能：

$$S_{fork}=\max_{i=2,\cdots,k-1}\{w(n_1)+w(n_i)\} \tag{1}$$

而对于 join 图，其多父节点单一子节点特性使得任务复制无助于提高子节点最早完成时间，所以使用任务复制算法的 join 图最优调度性能和非复制方式一致。Join 图的最优化调度长度如下：

$$S_{join}=\max\{w(n_k)+\sum_{i=2}^{j}w(n_i),w(n_k)+c(n_{j+1},n_k)+w(n_{j+1})\} \tag{2}$$

其中，j 满足 $\sum_{i=2}^{j}w(n_i)\leqslant w(n_j)+c(n_j,n_{k-1})$ 且

$$\sum_{i=2}^{j+1}w(n_i)>w(n_{j+1})+c(n_{j+1},n_{k-1}) \tag{3}$$

不失一般性，假设父节点按照执行时间与通信开销之和从大到小排列：

$$w(n_2)+c(n_2,n_k)\geqslant w(n_3)+c(n_3,n_k)\geqslant\cdots\geqslant w(n_{k-1})+c(n_{k-1},n_k) \tag{4}$$

那么，基于 join 和 fork 图的最优调度结果为：

$$S_{FJ}=w(n_1)+\max\{w(n_k)+\sum_{i=2}^{j}w(n_i),w(n_k)+c(n_{j+1},n_k)+w(n_{j+1})\} \tag{5}$$

CCTD 算法利用基于复制的 Fork-join 最优化调度结果来对 Fork-join 任务图进行调度，并且在调度过程中加入处理器之间的通信限制，也就是一条通信信道不能同时被两个通信事件占用，处理器必须串行处理两个通信事件。CCTD 算法在调度时力求减少处理器需要的数目，尽量调度任务到已有处理器上，在保证性能的同时最小化调度所需要的处理器数。CCTD 算法的描述如下：

算法 1 对于 Fork-join 图的 CCTD 调度算法

1. 假定节点的执行时间和相互通信开销满足不等式(4)；
2. 找出满足不等式(3)的 j，将 2 至 j 的任务分配到处理器 P_0；
3. 任务 $n_i(1<i<k)$最大完成时间 $F_i=\max\{\sum_{i=2}^{j}w(n_i),c(n_{j+1},n_k)+w(n_{j+1})\}$；
4. 设 $T[i]$为处理器 P_i 上的任务完成时间，$C[i]$为处理器 P_i 上的通信完成时间；
5. 初始处理器个数 $s=1$
6. 循环分配 $m+1$ 到 $n-1$ 的任务 j
   - 6.1 如果调度 j 到处理器 P_0 并且任务完成时间小于 F_i，那么就调度 j 到处理器 P_0
   - 6.2 循环处理器 P_1 到 P_s
     - 6.2.1 如果处理器 P_i 调度上任务 j 后任务完成时间小于 F_i 那么就调度 j 到处理器 P_i 上，并且终止循环
     - 6.2.2 否则切换下一个处理器
   - 6.3 如果任务 j 没有被调度到任何处理器，那么 $s++$，调度 j 到处理器 P_s
7. 插入任务 n_0 到 P_0 至 P_s 处理器的起点，插入任务 n_k 到 P_0 处理器任务列表的尾部。

算法首先对输入的 Fork-join 中间任务节点按照它们执行开销与出口节点通信开销之和从大到小排序。这一步需要进行排序，其算法的时间复杂度为 $O(n^2)$。然后找出满足不

等式(3)的拐点，其时间复杂度为 $O(n)$。为了表示通信信道是具有独占性的，算法使用数组 $C[i]$来记录处理器上通信信道的占用时间，因为任务图的出口节点固定调度到处理器 P_0，所以通信信道都被认为是当前处理器与处理器 P_0 之间的连接。$C[i]$表示的是处理器 P_i 与处理器 P_0 之间的通信信道。另一个数组 $T[i]$保存处理器上任务完成的最晚时间。算法把满足不等式(4)的 $2\cdots j$ 号节点调度到处理器 P_0。如果存在 $j+1(j+1<>n)$那么就调度 $j+1$ 到处理器 P_1，设定 $C[1]$和 $T[1]$的值。然后循环分配未被调度的节点到处理器。为了减少调度所占用的处理器数目，算法每次遍历已占用的处理器，在保证不增加任务最后完成时间的前提下尽量插入任务到已有处理器。该遍历过程的时间复杂度是 $O(n\log n)$。插入任务节点要满足条件：

$$\begin{cases} T[i]+w(n_i)<F_t, i=0 \\ \max(T[i]+w(n_i)+c(n_i,n_k), C[i]+c(n_i,n_k))<F_t, i>0 \end{cases} \quad (6)$$

条件不等式(6)是为了保证在处理器 P_i 上的任务完成时间不会推迟 join 节点的开始时间。由于处理器 P_0 上的任务与 join 节点在同一个处理器上，因此就不需要考虑它们之间的通信时间。而对于其它处理器上的任务，必须同时满足通信时间 $C[i]$的限制。

下面我们以图 1(c)中的 Fork-join 图为实例，给出应用 CCTD 调度算法后的任务执行情况示意图，如图 2 所示。其中，灰色部分表示对应任务在处理器上占用的时间片长度，斜线阴影部分表示任务之间的数据通信对通信信道的占用情况。

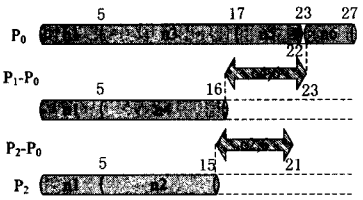

图 2　对图 1(c)中例子使用 CCTD 调度结果

在 CCTD 算法中，对中间任务节点按照执行时间与开始时间之和从大到小排序后，其序列是$\{n_3, n_4, n_2, n_5\}$。计算得到拐点 j 为 2，$F_t=18$。首先，将分配 n_3 到 P_0，设定 $T[0]$为 12。n_4 分配到 P_1，$T[1]=11$，$C[1]=18$。然后分配 n_2：在 P_0 处理器上 $w(n_2)+T[0]=22>F_t$，因此选取下一个处理器 P_1。但 $w(n_2)+T[1]+c(n_2,n_6)=27>F_t$，所以分配一个新处理器 P_2，并设定 $T[2]=10$ 以及 $C[2]=21$。选取 n_5，$T[0]+w(n_5)=17<F_t$，满足限制条件，分配 n_5 到 P_0，完成中间节点调度。插入 n_0 到每个处理器上的任务列表的头部，n_k 加入到 P_0 任务列表的尾部，这样就完成了所有任务调度。最终，该任务调度长度为 27。

4　实验结果

本文实现了 CCTD 算法，并将它与其它几种基于任务复制的调度算法 TSA_FJ，CPFD，TDS 进行了对比测试。首先对图 1(c)中的例子进行调度性能测试。其测试结果如表 1 所列，CCTD 算法得到了最短的调度长度。而与 CPFD 和 TDS 算法相比，虽然最终调度长度相同，但是 CCTD 算法所需要的处理器个数最少并且算法时间复杂度也较低。相对于 TSA_FJ 算法而言，CCTD 虽然时间复杂度较高，但是它具有比 TSA_FJ 更优的调度结果。因此，CCTD 在算法复杂度与调度性能间找到了一个较好的平衡点。

表 1　对图 1(c)中例子的调度结果对比

调度算法	CCTD	CPFD	TDS	TSA-FJ
任务调度长度	27	27	27	30
所需处理器个数	3	4	4	3
算法时间复杂度	$O(v^2)$	$O(v^4)$	$O(v^2)$	$O(v)$

为了更广范围地对比这几种算法的性能，本文使用程序随机生成不同大小的 Fork-join 图，通过这些随机测试图评测这 4 种任务调度算法在不同任务规模下的调度性能。我们分别按节点数目为 4，16，32 和 64 这 4 类随机生成测试图。每一类需要生成 100 个随机 Fork-join 图进行调度，最后记录平均调度长度和所需处理器个数，并进行性能比较，如图 3 所示。

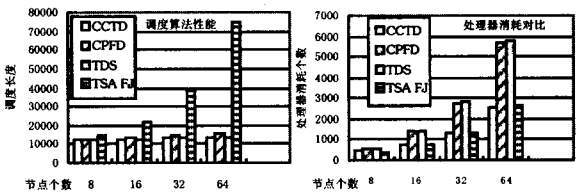

图 3　随机图测试结果

从图 3 中我们可以看出，CCTD 算法与其他算法相比有着最好的调度性能。同时，该算法具有良好的可扩展性，随着节点数目的增加，其调度效果并没有受到影响。CCTD 在保持高性能的同时，所占用处理器数目也很小，处理器的消耗量仅占 TDS 与 CPFD 的 50%。TSA_FJ 算法虽然占用的处理器数目少，但是其性能可扩展性不好。对于节点数目较多的大规模调度，该算法并不适用。CPFD 与 TDS 都有着不错的调度性能，但是随之带来的是大量处理器的消耗。测试表明，对于 Fork-join 任务图，CCTD 算法的适应性强，具有良好的性能与高效的处理器利用率。

结束语　本文提出了基于任务复制的 CCTD 调度算法，利用已有的对 fork 图和 join 图的研究成果，改进了对 Fork-join 任务图的调度算法，并且在算法中考虑到了实际并行系统中的通信信道独占性。CCTD 可以对具有无限处理器个数的全相连并行系统进行任务调度。CCTD 算法在保证调度长度最短的同时，最小化了所占用的处理器数目，提高了处理器利用率。在与其他算法进行的对比实验中，我们可以看到 CCTD 算法取得了最短的调度长度和较低的时间复杂度。在同等调度性能下，其所需要的处理器数目最少。由于在算法中引入了通信独占性限制，CCTD 相比大多数算法在实际系统中能得到更广泛的应用。

本文中，CCTD 仅研究了 Fork-join 这种特殊结构的任务图调度。但是通用形式的复杂任务图都可以分解和转换为基本 Fork-join 图的组合。在未来的研究工作中，我们将进一步改进 CCTD 算法，使之能适用于通用任务图的调度，从而使得基于复制的任务调度能获得更好的性能。

参 考 文 献

[1] Ahmad I，Kwok Y. On exploiting task duplication in parallel program scheduling[J]. IEEE Trans. on Parallel and Distributed Systems，1998，9(9)：872-892

[2] Darbha S，Agrawal D P. Optimal scheduling algorithm for dis-

tributed-memory machines[J]. IEEE Transactions on Parallel and Distributed Systems,1998,9(1):87-95
[3] 刘振英,方滨兴,姜誉,等. 一个调度 Fork-Join 任务图的新算法[J]. 软件学报,2002,13(4):693-697
[4] Gerasoulis A, Yang T. A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors[J]. Journal or Parallel and Distributed Computing, Dec. 1992,16:276-291
[5] Kwok Y-K, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors[J]. ACM Comput. Surv.,1999,31(4):406-471

(上接第 275 页)

库包括从 1992 年 4 月到 1994 年 4 月剑桥大学实验室拍摄的一系列人脸图像,具体为 40 个人,每个人由不同表情或不同视点的 10 幅图像所构成,倾斜角度不超过 20 度。人脸库中的部分人脸图像如图 1 所示。

图 1 ORL 人脸数据库部分人脸图像

实验中的训练数据集和测试数据集均随机生成,分别从每类中取 $\vartheta=4,5,6$ 构成训练样本集。每次实验中的训练样本集均随机产生,数据库中训练样本集之外的数据构成测试样本集,最终抽取得到 39 维特征,其中的核函数采用 2 种典型的核函数形式:

1)多项式核函数

$$k(x,y)=(x\cdot y)^d \tag{35}$$

2)高斯核函数

$$k(x,y)=\exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right) \tag{36}$$

最后采用最近邻分类器进行分类。我们的实验中,在每个不同训练样本数目下均做 10 次不同的实验。表 1 显示了在不同的实验方法中 10 次不同结果的平均识别率(%)和方差比较,括号中的数据为特征抽取时间(s)。

从表 1 的实验结果可以得出,在不同的核函数下,本文提出的基于正交变换的快速核 Foley-Sammon 鉴别分析方法在识别性能上明显优于传统的核线性鉴别分析方法,并且稍优于核 Foley-Sammon 鉴别分析方法。但是在特征提取的速度上,本文提出的新方法明显优于传统的核 Foley-Sammon 鉴别分析方法,并且基本达到了核线性鉴别分析方法的特征提取速度。

表 1 ORL 人脸数据库上的实验结果

# Training sample / class (ϑ)		4	5	6
多项式核函数	KLDA	87.46±2.51 (0.3130)	91.75±1.72 (0.4530)	94.82±1.32 (0.3430)
	KFSLDA	89.75±2.67 (5.2340)	94.90±1.24 (9.2340)	95.81±1.38 (15.8900)
	KFSDA	91.71±2.99 (0.3130)	95.15±1.20 (0.4600)	96.12±1.17 (0.3600)
高斯核函数	KLDA	91.58±1.39 (0.2500)	92.40±2.83 (0.4530)	95.10±2.23 (0.3290)
	KFSLDA	92.67±1.83 (6.4840)	94.60±2.40 (11.4380)	96.13±1.83 (18.7820)
	KFSDA	93.12±1.72 (0.2500)	95.50±1.92 (0.4530)	96.69±1.32 (0.3340)

结束语 抽取复杂样本的非线性特征是模式识别研究的一个关键步骤,但是 KFSDA 在实际应用中存在计算时耗较大的弱点。本文基于传统的核线性鉴别分析,给出了一种快速算法,有效地抽取到了原始样本的非线性正交特征矢量。与传统的核 Foley-Sammon 鉴别分析方法相比较,本文提出的算法抽取最佳鉴别矢量集速度快,并且保持了较高的识别率。在 ORL 人脸数据库上的实验结果验证了该算法的有效性。

参 考 文 献

[1] Fisher R. The use of multiple measures in taxonomic problems [J]. Ann. Eugenics.,1936,7:79-188
[2] Wilks S S. Mathematical Statistics[M]. New York: Wiley,1962
[3] Belhumeur P N, Hespanha J P, Kriengman D J. Eigenfaces vs Fisherfaces: Recognition using class specific linear projection [J]. IEEE Trans. Pattern Anal. Machine Intell,1997,19(7):711-720
[4] Foley D H, Sammon J W Jr. An optimal set of discriminant vectors[J]. IEEE Trans. Computers,1975,24(3):281-289
[5] Lu J, Platanintis K N, Venetsanopoulos A N. Face Recognition Using Kernel Direct Discriminant Analysis Algorithms [J]. IEEE Transactions on Neural Networks,2003,14(1):117-125
[6] Vapnik V N. The Nature of Statistical Learning Theory[M]. 2nd ed. New York: John Wiley and Sons,1998
[7] Baudat G, Anouar F. Generalized discriminant analysis using a kernel approach[J]. Neural Computation,2000,12(10):2385-2404
[8] 甘俊英,张有为. 模式识别中广义核函数 Fisher 最佳鉴别[J]. 模式识别与人工智能,2002,15(4):429-433
[9] Mika S, Rätsch G, Weston J, et al. Fisher discrimi-nant analysis with kernels[A]//Proceedings of IEEE International Workshop on Neural Networks for Singal Processing[C]. Madison, Wisconsin, August 1999:41-48
[10] Müller K-R, Mika S, Rätsch G, et al. An introduction to kernel-based learning algorithms[J]. IEEE Trans. Neural Networks, 2001,12(2):181-201
[11] Yang J, Jin Z, Yang J Y, et al. Essence of Kernel Fisher Discriminant: KPCA plus LDA[J]. Pattern Recognition,2004,37(10):2097-2100
[12] 高秀梅,杨静宇,金忠,等. 基于核的 Foley-Sammon 鉴别分析与人脸识别[J]. 计算机辅助设计与图形学学报,2004,16(7):962-967
[13] Zheng W M, Zhao L, Zou C R. Foley-Sammon Optimal Discriminant Vectors Using Kernel Approach[J]. IEEE Trans. Neural Networks,2005,16(1):1-9
[14] Liu K, Yang J Y, Cheng Y Q, et al. An efficient algorithm for Foley-Sammon optimal set of discriminant vectors by algebraic method[J]. International Journal of Pattern Recognition and Artificial Intelligence,1992,6(5):817-829
[15] Song F X, Liu S H, Yang J Y. Orthogonalized Fisher discriminant[J]. Pattern Recognition,2005,38:311-313