

doi: 10.3969/j.issn.1006-1576.2010.03.006

一个新的 Fork 任务图的调度算法

杨峰¹, 张建军²

(1. 海军工程大学 理学院, 湖北 武汉 430033; 2. 华中科技大学 计算机学院, 湖北 武汉 430074)

摘要: 针对 Fork 任务图的结构特点, 提出了基于任务复制的调度算法。在对算法基础、术语、新的 Fork 任务图的调度算法-NSF 进行了介绍的基础上, 运用实例对调度进行了分析比较。该算法在保证得到最优调度长度的前提下, 减少了使用处理机的个数。实验结果表明, 该算法综合性能优于其它算法。

关键词: Fork 任务图; 任务调度; 任务复制; 加速比

中图分类号: TP301.6; TP274⁺.2 **文献标识码:** A

A New Scheduling Algorithm for Fork Task Graphs

YANG Feng¹, ZHANG Jian-jun²

(1. College of Science, Naval University of Engineering, Wuhan 430033, China;

2. Dept. of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan 430074, China)

Abstract: Aiming at the features of Fork task graphs, proposes a scheduling algorithm based on task duplication. On the basis of introducing algorithm base, nomenclature, the new scheduling algorithm of Fork task graph -NSF, the examples are used to analyze and compare to the scheduling. The algorithm ensures the optimal scheduling length and reduces the number of used processors. Experiment results show that the proposed algorithm has better comprehensive performance than other algorithms.

Keywords: Fork task graphs; Task scheduling; Task duplication; Speedup

0 引言

并行任务常用有向无环图 (Directed Acyclic Graph, DAG) 表示。任务调度是指将 DAG 调度到处理机上, 使任务满足约束关系, 并使得调度长度短, 使用的处理机数少。Fork 任务图是 DAG 最简单的一种类型, 寻找出针对 Fork 任务图的有效调度算法对进一步研究各种类型 DAG 会有很大的帮助。一般来说, 基于任务复制的调度优于不是基于任务复制的调度^[1-2]。当前, 基于任务复制的调度已成为研究热点^[3-5]。故针对 Fork 任务图, 在使调度长度最短的情况下, 利用任务复制的方法, 在同构的环境下, 提出了一个节省处理机数目的调度算法。

1 算法基础与术语

当一个任务群在并行处理机上运行时, 设一个任务是不可再分的工作单元, 并行处理机中处理机数目无限。不同处理机利用初始数据执行同任务的不同拷贝, 同任务的不同副本生成的数据相互一致。用 DAG 来描述任务图。DAG 用一个四元组 (V, E, w, c) 来表示。其中 $V = \{n_1, n_2, \dots, n_m\}$ 表示任务集, $v = |V|$ 表示任务个数; E 是边的集合, $e = |E|$ 表示边的条数; w_i 是每个任务 n_i ($n_i \in V$) 的计算开销。对每条边,

如从任务 n_i 到任务 n_j 的边 $e(n_i, n_j) \in E$, 其通信开销为 $c_{i,j}$ 。如图 1, 处理机的集合为 $P = \{P_0, P_1, \dots, P_n, \dots\}$ 。

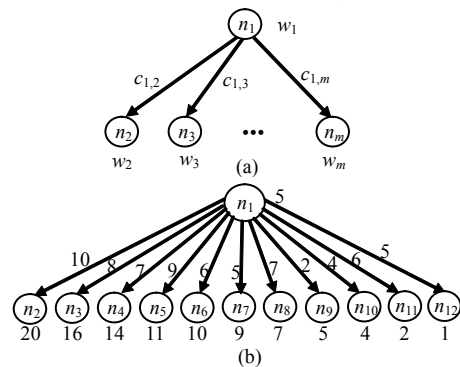


图 1 Fork 任务图及其实例

采用任务复制的方法来分配处理机, 一个任务可能被分配到多个处理机上。其中, 任务的执行时间 w_i 可以由任务中程序代码的计算量来衡量, 这可以通过编译器在用户代码的编译过程中得到, 是一个静态的参量。在应用程序中, 任务间的通讯量是由应用程序问题本身决定的。在应用程序的执行中, 任务间通信以及对执行性能的影响主要分为如下几类: 1) 同步等待: 应用程序的同步点设置, 等待各任务都执行到相应的程序点; 2) 数据依赖: 某处理机上的任务执行需要另一处理机上任务计算所产生

收稿日期: 2009-10-21; 修回日期: 2009-11-30

基金项目: 海军工程大学自然科学基金项目, 现代模糊信息优化处理技术及其应用研究(HGDJJ05005)

作者简介: 杨峰 (1979-), 男, 湖北人, 讲师, 硕士, 从事数据挖掘、算法设计研究。

的结果以及中间数据; 3) 执行完成等待: 当处理机上的任务完成计算后, 等待其他处理机上任务的计算结束, 以结束整个并行应用程序。故需要在处理机分配过程中, 既考虑系统开销最小, 又考虑处理机的负载平衡。因此, 求解任务调度算法, 就是对该 DAG 找到一个算法, 使并行执行时间 (即“调度长度”) 最短, 采用的处理机个数最少。

图 1(a)中给出一个 Fork 任务图, 图 1(b)是具体的 Fork 任务图的例子。从图 1 中可见 Fork 任务图的特点。其中, 任务集 $V=\{n_1, n_2, \dots, n_m\}$, 边集 $E=\{e(n_1, n_2), \dots, e(n_1, n_m)\}$ 表明: 除了根结点外的每个任务结点, 都依赖根结点的执行。对于 Fork 任务图求解调度算法, 只要把根结点复制到每个叶结点的处理机, 就解决了调度长度最短的问题。故该问题就转化为求解: 除了执行时间最长 $w_{\max}=\{w_2, w_3, \dots, w_m\}$ 的叶结点外, 其他叶结点如何组成集合, 使得总的集合数最少, 且每个集合中叶结点执行时间的和小于 w_{\max} 。该问题是装箱问题—典型的 NP 难题。

2 新的 Fork 任务图的调度算法—NSF

设 Fork 任务图中共有 m 个结点, 其中, n_1 是根结点, n_2, \dots, n_m 是叶结点。根据任务复制的思想, 除了具有调度长度的关键路径外, 为节省处理机的个数, 将 Fork 任务图中若干个叶结点组合, 只要叶结点和根结点的执行时间之和不大于调度长度, 就可把叶结点连同根结点合并到一个处理机上执行。

此提出的算法称为 NSF。该算法的第 1 步采用快速排序, 复杂度为 $o(v \log v)$, 在其余步骤中复杂

度为 $o(v^2)$, 所以总复杂度为 $o(v^2)$ 。

算法: 新的 Fork 任务图的调度算法—NSF。

Begin

- 1) 用快速排序将 n_2 至 n_m 按照任务的计算时间降序排列, 排列后的顺序记为 n_2, n_3, \dots, n_m
- 2) $i=2, k=1$;
- 3) 将 n_1 和 n_2 分配到处理机 P_0 ;
- 4) 当 $i \leq m$;
- 5) {清空 *candidate task set*, 并且把 n_1 和 n_i 插入到 *candidate task set* 中;
- 6) $i++$;
- 7) 令 $h_q = w_2 - w_i$;
- 8) {令 $j=i+1$;
- 9) 如果 $h_q - w_j = 0$, 则将 w_j 插入 *candidate task set* 中;
- 跳出循环;
- 如果 $h_q - w_j < 0$, 则直接跳出循环;
- 如果 $h_q - w_j > 0$, 则将 w_j 插入 *candidate task set* 中, 再考虑剩下的空闲时间是否还有可以插入的任务, 直到没有任务可以插入为止;
- $j++$;
- }while ($j \leq m$)
- } while ($i \leq m-1$)
- 10) 把 *candidate task set* 中的任务分配到处理机 P_k ;
- 11) $k++$;

End

3 实例分析及比较

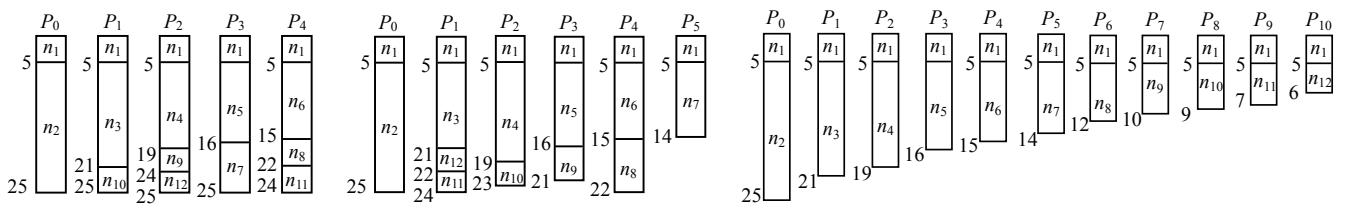


图 2 Fork 任务图的调度结果

对图 1(b)中的 Fork 任务图的例子 (为了便于大家理解, 图 1(b)中的任务已经按执行时间的长短按顺序排列好), 算法 NSF 的调度过程是: 首先, 将任务按照执行时间降序排列, 得到序列如图 1(b), 将 n_1 和 n_2 分配到处理机 P_0 , 接着把 n_1 和 n_3 分配到处理机 P_1 , 由于 $20 - 16 = 4 \geq w_{10} = 4$, 故将 n_1 、 n_3 和 n_{10} 分配到同一处理机 P_1 。再将 n_1 和 n_4 分配到处理机 P_2 , 因为 $20 - 14 = 6 > 5 = w_9$, 所以将 n_9 分配到处理机 P_2 , 又因为 $6 - 5 = 1 \geq w_{12} = 1$, 所以将 n_{12} 分

配到处理机 P_2 , 因此, 最后 n_1 、 n_4 、 n_9 、 n_{12} 分配到同一处理机 P_2 。接着, 将 n_1 和 n_5 分配到处理机 P_3 , 又由于 $20 - 11 = 9 \geq w_7 = 9$, 因此, 将 n_7 添加到处理机 P_3 中, 所以, n_1 、 n_5 、 n_7 分配到同一处理机 P_3 。最后, 将 n_1 和 n_6 分配到处理机 P_4 , 又由于 $20 - 10 = 10 > w_8 = 7$, $10 - 7 = 3 > w_{11} = 2$, 所以 n_1 、 n_6 、 n_8 、 n_{11} 分配到同一处理机 P_4 。下面, 给出 NSF 算法、TSA_FHM 算法和 TDS 算法对图 1(b)中的 Fork 任务图的调度结果, 如图 2。

(下转第 20 页)

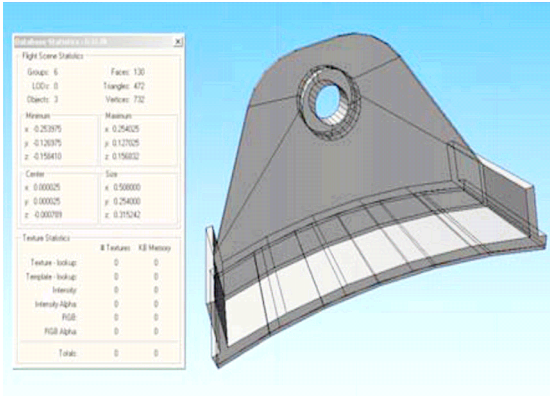


图 4 简化模型图

3 结束语

应用建模方法建立大型复杂装备的虚拟维修仿真样机几何模型可满足仿真要求,克服了建立 VR 几何模型过程中信息的丢失与仿真过程模型的复杂

(上接第 15 页)

另外, 将它们的调度结果进行了比较, 如表 1。

表 1 不同算法对图 1(b)调度性能比较

算法	NSF 算法	TSA_FHM 算法	TDS 算法
调度长度	25	25	25
处理机个数	5	6	11
算法复杂度	$O(v^2)$	$O(v \log v)$	$O(v^2)$
效率	80%	67%	36%
加速比	4.0	4.0	4.0

3 种算法比较得出: 决定一个调度算法优劣最关键的性能指标是调度长度, 其次是所用的处理机的个数和算法的复杂度。加速比也是一个被广泛采用衡量算法性能的指标, 效率是用来度量在并行计算中整个系统的资源利用率, 其计算公式为:

$$\text{效率} = \frac{\text{加速比}}{\text{处理器个数}}$$

从表 1 中可以看出: NSF 算法综合性能要优于其他算法。3 种算法调度长度都最短, 并且它们的加速比也相同, 但是 NSF 算法采用的处理机的个数比另两种方法少, 其效率也优于另外 2 种算法。

为更全面地评估和比较 NSF 算法与其他算法, 将随机生成的 Fork 任务图作为测试这些算法的负载。通过随机产生的任务计算时间和通信时间, 选取 6 个 Fork 任务图, 将 NSF 算法、TSA_FHM 算法及 TDS 算法的调度结果进行比较, 如图 3。

从图 3 可以看出: 与 TSA_FHM 算法和 TDS 算法相比, NSF 算法在保证最优调度长度的前提下, 所用的处理机的个数最少。而且, 任务图中的结点数越多, 本算法越显著优于其他算法。可见, 对 Fork

度和仿真实时性之间的矛盾, 使虚拟维修样机几何模型与实际装备在结构上具有相似性, 在功能和行为上具有一致性, 支持不同的应用重构和重组。

参考文献:

[1] 苏群星, 刘鹏远. 大型复杂装备虚拟维修训练系统设计[J]. 兵工学报, 2006, 27(1): 80-83.

[2] 王松山. 虚拟维修样机技术研究[R]. 中国国防技术报告 GF-A0065309G, 石家庄: 军械工程学院, 2003.

[3] 刘鹏远, 张锡恩, 等. 虚拟维修训练中基于知识的操作响应机制研究与实现[J]. 计算机工程, 2003, 29(1): 53-255.

[4] 杜平安, 于德江, 岳萍. 虚拟样机技术的技术与方法体系研究[J]. 系统仿真学报, 2007, 19(15): 3448-3449.

[5] 王志伟, 马明江. 理论力学[M]. 北京: 机械工业出版社, 2006.

[6] Antonino, Gabriel Zachmann. Virtual Reality as a Tool for Verification of Assembly and Maintenance Processes [J]. Computers&Graphics, 1999, 23(3): 389-403.

任务图, 本算法应用性更强。

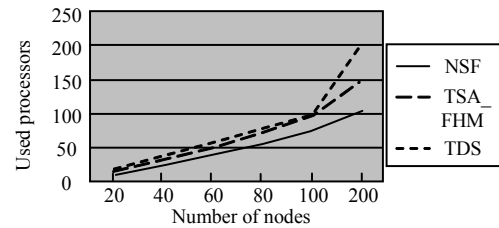


图 3 各种算法所用处理机数目比较

4 总结

该算法在保证得到最优调度长度的前提下, 减少了使用处理机的个数。实验结果表明, 该算法综合性能优于其它算法。下一步将研究特殊任务图的有效调度算法, 例如 in-tree、out-tree、fork-join 任务图的调度问题。

参考文献:

[1] K. Yu-Kwong, I. Ahmad, Benchmarking and Comparison of the Task Graph Scheduling Algorithms[J]. Parallel and Distributed Computing, 1999, 59(2): 381-422.

[2] S. Srinivasan, N. Jha, Safety and Reliability Driven Task Allocation in Distributed Systems[J]. IEEE Transaction on Parallel and Distributed Systems, 1999, 5(10): 56-60.

[3] S. Darbha, D.P. Agrawal, Optimal Scheduling Algorithm for Distributed-Memory Machines[J]. IEEE Trans. Parallel and Distributed Systems, 1998, 9(1): 87-94.

[4] Park C I, Choe T Y, An Optimal Scheduling Algorithm Based on Task Duplication[J]. IEEE Trans. Computers, 2002, 51(4): 444-448.

[5] Zhengying Liu, A New Algorithm for Scheduling Fork-Join Task Graph[J]. Journal of Software, 2002, 13(4): 693-696.