表调度算法的并行化研究*)

李庆华 马 丹 张 薇

(华中科技大学计算机科学与技术学院 武汉430074) (国家高性能计算中心 武汉430074)

摘 要 当目标处理器个数大于2时,调度任意结构并行任务图并获取最优解的问题是NP完全难题。表调度算法作为一类代表性的启发式任务调度算法具有调度性能较好而时间复杂度较低的优点。但当任务图的规模较大时表调度算法的耗时也很可观,无疑并行表调度算法是一种好的解决方法。本文在串行算法LBP的基础上提出了一个新的表调度并行算法PLBP,该算法在保证与串行算法同样调度性能的前提下,时间复杂度有较大的改善。同时,与已有的表调度并行算法相比较,PLBP算法有更小的时间复杂度。

关键词 表调度,并行算法,任务调度

On Parallelizing the List Scheduling Algorithm

LI Qing-Hua MA Dan ZHANG Wei

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

(National High Performance Computing Center, Wuhan 430074)

Abstract When the number of processing units is more than 2, scheduling arbitrary task graph in order to obtain an optimizing solution is a NP-complete problem. As a heuristic scheduling algorithm, list scheduling algorithm has better performance and lower time complexity. But the run time of list scheduling algorithm becomes too long when the size of task graph become huge. Undoubtedly, parallelizing the list scheduling algorithm may reduce the time complexity. Based on serial list scheduling algorithm LBP, this paper presents a new parallel list scheduling algorithm PLBP. It can obtain the same performance as LBP and effectively improve the time complexity of LBP. At the same time, PLBP has lower time complexity comparing to the other parallel list scheduling algorithm in the literature.

Keywords List scheduling, Parallel algorithm, Task scheduling

1 引言

在设定的多目标处理器拓扑结构上对一个复杂并行程序抽象成的有向无环图(DAG)—任务图进行合理调度并追求程序整个执行时间最短的问题是众所周知的NP完全难题。当然,在某些严格的假设条件下调度并行任务问题是可以在多项式时间复杂度算法中找到最优解的,它们一般可以归纳成三种情形:1)任务图限定为in-forest或out-forest图;2)任务图限定为一类特殊的所谓Interval order图;3)当目标处理器个数等于2时。除此之外,在任何情况下想在多项式时间复杂度的前提下寻找最优解是不可能的。所以目前研究者对任务调度问题的研究主要集中在如何设计出多项式时间复杂度的启发式静态(或编译时)任务调度算法,这些算法可以得到近似最优解。这类启发式调度算法绝大部分是基于经典的表调度算法的。如HLFET算法,时间复杂度为$O(v^2)$[1];ISH算法,时间复杂度为$O(v^2)$[2];ETF算法,时间复杂度为$O(pv^2)$[3];MCP算法,时间复杂度为$O(v^2\log v)$[4]等等(其中v表示任务节点的个数,p表示目标处理器的个数)。

从上述算法的时间复杂度来看,尽管是多项式时间复杂度,但是当任务图的规模变大时算法的运行时间也是可观的。同时这些调度算法本身都是串行的,它们都只是运行在某个单一的处理器上,当任务图规模变大时存储整个任务图需要占用较大的空间。因此串行的调度算法在时间和空间上都受到很大的制约。如何改善这种情况我们自然地想到将串行调度算法进行并行化,即在多个处理器上来运行调度算法。并行的调度算法主要在以下两个方面来提高算法的性能:1)减少任务图的调度长度;2)减少调度任务图的耗费时间。这两个目标是相互制约的,所以在不失去平衡的前提下主要研究如何降低算法的时间复杂度。

并行调度算法的基本思想是变串行算法中每个时刻只调度一个任务节点为每个时刻并行调度多个任务节点。但是,表调度算法一般被认为本质上是串行的,因为除了DAG中任务节点之间本身的数据相关性在调度中需要被保护外,每个任务节点的调度都与其之前调度的节点由于空间的限制可能在时间上也存在先后关系。因此到目前为止,研究并行程序任务图(DAG)的并行调度算法并不多见,见诸文献的主要有Min-You Wu等人提出的HPMCP算法[5]以及Ishfaq Ahmad等人提出的PBSA算法[6]。这些并行算法主要都是基于某种已有的串行算法,它们一般可以概述为以下几个步骤:1)把任务图DAG划分成与参与并行调度处理器个数相等的部分,划分的方法有水平、垂直以及综合三种。HPMCP算法采用水平划分,而PBSA算法采用综合划分。2)所有处理器同时独自采用某种串行调度算法对每个划分的子部分进行调度,调度完成得到多个子调度。HPMCP采用了MCP算法,而PBSA

*)本文研究得到国家自然科学基金资助(No. 60273075)。李庆华 教授,博士导师,主要研究方向是并行分布式计算、智能软件。马 丹 博士生,主要研究方向为并行计算、网格计算。张 薇 博士生,主要研究方向为网格计算、移动代理。

采用了 BSA 算法。3)对所有的子调度进行连接操作最后形成全局的调度。分析上述并行算法的逻辑步骤，发现它们的并行都是建立在割裂任务图以及调度步骤中本身固有的串行性而机械并行的前提下，实际调度时绝大多数并行处理器必须估计其调度任务节点的大致启动时间，因为 DAG 中靠后的任务与靠前的任务存在时序关系，即使任务间不存在先后关系但当它们在同一时刻被调度到同一个目标处理器时也必须先后排列，但由于所有处理器同时动作所以这种关系根本不可能得到保护只能靠估计来维护，因此确定估计值的方法对整个并行算法的性能至关重要。另外，每个处理器得到的调度都是局部调度，必须对局部子调度进行连接操作才能得到全局调度，这个连接过程一般比较复杂即比较耗时。所以上述并行算法即使能够降低其相应串行算法的时间复杂度也很难达到其串行算法的调度性能(即相同的调度长度)。

本文在研究上述调度算法的基础上提出了一种新的并行调度算法，该算法只并行串行算法中可并行的部分而不割裂 DAG 中固有的串行性，每个并行处理器并不调度相同数目的任务节点而采取灵活配置的方法，每个调度步骤可同时得到多个任务节点的最终调度而不必最后进行连接调整。重要的是，该算法能完全保证与其串行算法相同的调度性能而降低了串行算法的时间复杂度。

2 基本模型与定义

一般，DAG 可由图 $G=(V,E)$ 来表示，其中 $V=\{t_i \mid i=1,2,\cdots,v\}$，表示 v 个有权任务节点的集合，$E=\{(t_i,t_j) \mid t_i,t_j \in V\}$，表示 e 个任务间带权有向边的集合。用 $w(t_i)$ 表示任务节点的权重，即任务 t_i 的计算成本。c_{ij} 是有向边 (t_i,t_j) 的权重，即任务 t_i 与任务 t_j 之间的通信成本。有向边的源节点称为父节点，有向边的终节点称为子节点。典型的 DAG 如图1所示。

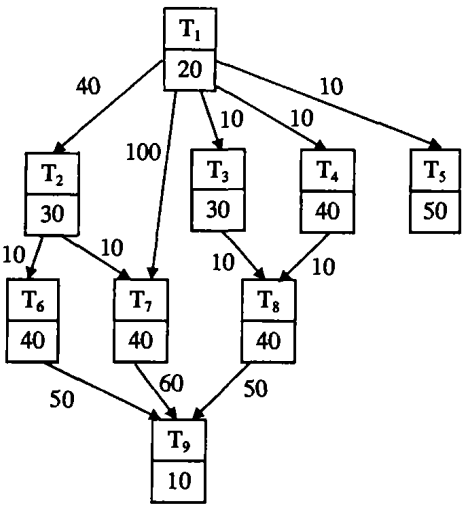

图1 一个典型的 DAG

假设完成并行程序的目标处理器集合是一个分布式内存的多处理器系统，每个处理器与它的本地存储器组成一个处理单元，记为 tp。tp 之间的通信采用消息传递的方式，调度到同一个处理单元的两个任务节点之间的通信成本为零。

同样执行 DAG 并行调度的调度处理器集合也是一个由本地存储器和处理器组成的处理单元系统，为了不与目标处理器发生混淆，该处理单元记为 pp。

另外，本文的算法中所需的一些术语符号及其定义在表1中详细列出。

表1 基本术语符号的定义

符号	定义
t_i	DAG 图中的任一任务节点
v	DAG 图中所有任务节点的总个数
$w(t_i)$	任务节点 t_i 的计算成本
e	DAG 图中通信边的总个数
c_{ij}	通信边 (t_i,t_j) 的通信成本
L_i	任务 t_i 的层次值，层次指相互间无数据相关性的一组任务集合
B_i	任务 t_i 的分支值
tp_i	执行并行程序的任一目标处理器
tp_i	执行并行程序的目标处理器总个数
pp_i	执行调度算法的任一物理处理器
pps	执行调度算法的物理处理器总个数
$trt(t_i,tp_j)$	当任务 t_i 被调度在 tp_j 时，t_i 收到其所有父节点通信消息的最早时刻，称任务就绪时间
$st(t_i,tp_j)$	任务 t_i 能在 tp_j 上启动执行的最早时刻，称任务启动时间
$free(tp_i)$	目标处理器 tp_i 最早的空闲时刻

3 调度算法的并行化

我们先提出一个基于表调度的串行调度算法 LBP，然后对该串行算法进行并行研究并给出最终的并行算法 PLBP，最后简单证明并行算法与串行算法可以获得同样的调度性能。

3.1 串行调度算法 LBP

算法 LBP(Level-Branch Priority)是一个表调度算法，与一般表调度算法不同之处在于任务优先级别的确定，它采用先层次值 L_i 后分支值 B_i 来确定任务 t_i 的优先级。按照任务的优先级别从大到小排列成一个就绪任务表后，每次从表头取出一个任务，把该任务调度到能使它在最早时刻启动的目标处理器上(如果目标处理器异构，则把任务调度到能使其在最早时刻完成的目标处理器上)。

层次值 L_i 与分支值 B_i 的确定：DAG 中没有相互数据依赖关系的任务节点定义成同一层节点。采用自底向上的方法计算 L_i，即按照从出口节点到入口节点的顺序依次计算各个节点的 L_i，设 $\sim L_i$ 是从出口节点到任务节点 t_i 之间最长路径上的通信边数之和，如果从出口节点到 t_i 之间有 j 条路径且相应的路径长度记为 $\sim L_{ij}$，则有 $\sim L_i=\max\{\sim L_{i1},\sim L_{i2},\cdots,\sim L_{ij}\}$。又设：$L_{max}=\max\{\sim L_1,\sim L_2,\cdots,\sim L_i,\cdots,\sim L_v\}$。则：$L_i=L_{max}-\sim L_i$。分支值 B_i 是任务节点 t_i 的所有出边权重之和，即 $B_i=\Sigma(c_{ij})$，t_j 是任务 t_i 的子节点。

任务节点 t_i 的优先级别确定：L_i 越小 t_i 的优先级越高，如果 L_i 和 L_j 相同，则 B_i 值越大的任务优先级越高。对图1来说：$L_1,L_2,\cdots,L_9=\{0,1,1,1,3,2,2,2,3\}$，$B_1,B_2,\cdots,B_9=\{170,20,10,10,0,50,60,50,0\}$，则任务优先级从大到小顺序为：$t_1>t_2>t_3>t_4>t_7>t_6>t_8>t_5>t_9$(如任务优先级相同，则任务号小的排在前面)。

整个 LBP 算法简单描述如下所示。

```
1. 输入 DAG，根据 Li、Bi 确定 ti 的优先级；
2. 把任务 ti 按照优先级递减的顺序依放入任务就绪表；
3. While 任务就绪表非空 Do
4.       从任务就绪表中取出表头元素 ti 开始调度；
5.       For 所有的目标处理器 tpj Do
6.             计算 ti 在目标处理器 tpj 上的最早启动时间，不考虑插
               入操作；
7.       Endfor
8.       把任务 ti 调度到能使其最早启动的目标处理器上；
```

9. Endwhile
10. 输出调度结果甘特图

3.2 并行调度算法 PLBP

根据上述串行算法 LBP 各步骤中数据和操作的相关性，采用 CRCW PRAM 计算模型，设计出一种基于串行表调度算法 LBP 的并行调度算法 PLBP，假设可用于并行调度的物理处理器个数 pps 为 DAG 的宽度，即 DAG 中包含任务节点最多的层次的任务个数。对 PLBP 并行算法进行非形式化描述如下：

1. 输入 DAG，计算 L_i，B_i 及 DAG 的总层数 L_{max}，每层的任务节点数 TT_i，初始化 $trt(t_i,tp_j)$、$st(t_i,tp_j)$、$free(tp_i)$。// 所有数据存于全局存储器 //
2. For $i=1$ to L_{max} Do
3. 激活 TT_i 个物理处理器 $pp_j(1\leqslant j\leqslant TT_i)$，按照分配的任务节点对激活的 pp_j 重新编号，层次 i 中任务节点的 B 值越大，则分配给它的 pp 编号越小。// 每个 pp 负责层次 i 中一个任务节点的调度 //
4. For 所有激活的处理器 $pp_j(1\leqslant j\leqslant TT_i)$ Do in parallel
5. For $k=1$ to tps Do
6. if $k=m$ then $trt(t_i,tp_k)=\max\{st(t_j,tp_m)+w(t_j)\}$（$t_j\in t_i$ 的父节点集，且 t_j 被调度到 tp_m）
7. else
8. $trt(t_i,tp_k)=\max\{st(t_j,tp_m)+w(t_j)+c_{ji}\}$（$t_j\in t_i$ 的父节点集，且 t_j 被调度到 tp_m）
9. end if
10. $st(t_i,tp_k)=\max\{trt(t_i,tp_k),free(tp_k)\}$ //计算层次中各任务在所有 tp 上的最早启动时间 //
11. Next k
12. For $j=1$ to TT_i Do
处理器 pp_j 计算 $st(t_i,tp_x)=\min\{st(t_i,tp_k)\}$，$free(tp_x)=st(t_i,tp_x)+w(t_i)(1\leqslant k\leqslant tps,tp_x$ 为 tp_k 中使 t_i 最早启动的处理器，t_i 是 pp_j 负责的任务节点），并把 t_i，调度到 tp_x（若能最早启动的处理器不只一个则取编号最小的 tp）
13. pp_j 把 $free(tp_x)$ 广播给 pp_{i+1} 至 pp_{TT_i} 处理器，使相应物理处理器更新相应的 $st(t_i,tp_k)$；同时把 $st(t_i,tp_x)$ 和 $free(tp_x)$ 值传递给全局存储器
14. Next j
15. Next i
16. 输出任务调度甘特图

4 算法的性能及时间复杂度、平均加速比分析

4.1 算法的性能分析

定理1 对相同的 DAG 及同样的目标处理器集合，并行调度算法 PLBP 能够获得与其串行算法 LBP 相同的调度性能（调度长度）。

证明：设 $start(t_i,tp_j,f_{lbp})$，$start(t_i,tp_j,f_{plbp})$ 分别表示在调度算法 LBP 和 PLBP 中，任务 t_i 被确实映射到目标处理器 tp_j 上的启动时间。如果对于所有的 $t_i\in V$，有 $start(t_i,tp_j,f_{lbp})=start(t_i,tp_j,f_{plbp})$ 则定理被证明。

要证明 $start(t_i,tp_j,f_{lbp})=start(t_i,tp_j,f_{plbp})$，$t_i\in V$。只需证明 t_i 在串行算法及并行算法中的最终调度次序完全一致即可，因为无论是串行还是并行算法对任务 t_i 的调度都是采取同样的贪心策略，即调度 t_i 到使其启动时间最早的目标处理器上。

在串行算法 LBP 中，任务 t_i 的调度次序是按任务优先级确定的，而任务优先级是按先层次后分支计算的，即层次越低的任务优先级越高，同一层的任务节点分支值越高则优先级越高。

再看并行算法 PLBP 的情形，算法中步骤2的最外层循环保证了任务是按层次进行调度的，即层次低的任务先调度。对于同一层次任务的调度，尽管有些计算过程并行执行，如步骤6～9计算任务初略的最早启动时间，它们并不决定任务的最终调度，而必须经过步骤12～14的修正来确定最终调度。在步骤12～14的循环中，每次循环决定同层任务中具有最大优先级的任务被最终调度，而其它任务的最早启动时间被修正一次。因此，并行算法 PLBP 中同层任务也是按与串行算法同样的优先级高低依次完成调度的。

综上所述，定理1证毕。

4.2 算法的时间复杂度及平均加速比分析

· 时间复杂度：在串行算法 LBP 中，计算任务的优先级别及对优先级别的排序需花费时间 $O(e+v\log v)$，而完成任务调度需要时间 $O(tps\times e)$，所以整个串行算法 LBP 的时间复杂度为 $O(tps\times e+v\log v)$。并行算法 PLBP 花费与 LBP 相同的时间来计算任务优先级别和排序，即 $O(e+v\log v)$，而完成任务调度的时间为 $O(v\log^{tps}+tps\times\sum_{i=1}^{L_{max}}e_{i\max})$，其中 $e_{i\max}$ 为 DAG 第 i 层所有任务节点中单个节点拥有的最大入边数，故并行算法 PLBP 的时间复杂度为 $O(tps\times\sum_{i=1}^{L_{max}}e_{i\max}+e+v\log v)$。可以看出，并行算法时间很大部分花费在计算任务的优先级别上，因为 LBP 与 PLBP 在计算任务优先级别时都采用同样的串行方法来遍历 DAG 的每条边。而在任务调度阶段，并行算法的时间复杂度比串行算法降低了许多。如果能对计算优先级别的串行方法也进行并行，则整个并行算法的时间复杂度无疑会下降更多。

· 平均加速比：由于并行算法 PLBP 是建立在 PRAM 并行计算模型上，而 PRAM 模型是不真实地简单的，它并未考虑调度处理器 pp 之间的通讯复杂性，只主要研究算法的理论并行效率，因此本文只对 PLBP 算法进行了理论数值分析。分析过程如下：

平均加速比 S_m=LBP 的时间复杂度/PLBP 的时间复杂度

即：$S_m=(tps\times e+v\log v)/(v\log^{tps}+tps\times\sum_{i=1}^{L_{max}}e_{i\max})$

从上式表面来看，平均加速比 S_m 与参与并行计算的处理器个数 pps 无关，其实不然，因为 S_m 与任务节点个数 v 是密切相关的，而 PLBP 中 pps 是等于任务图的宽度，不失一般性，我们取 $pps=|\sqrt{v}|$，假设任务图是平均密集程度，取 $e\approx(v*\sqrt{v}-1)/2$，$e_{i\max}$ 在 $\sqrt{v}/2$ 与 $\sqrt{v}$ 之间取值，表3和图2分别表示当任务图规模为 $v=100,200,500,1000,2000,4000$ 及 $tps=4,8,16$ 时，平均加速比 S_m 的理论分析数值和拟合曲线。

表3 PLBP 的平均加速比理论分析值

v	100			200			500			1000			2000			4000		
pps	10			14			22			31			44			63		
e	450			1274			5082			13950			41624			121086		
tps	4	8	16	4	8	16	4	8	16	4	8	16	4	8	16	4	8	16
Sm	5.0	4.9	5.1	6.8	6.7	7.1	10.2	10.4	11.1	13.8	14.3	15.5	19.5	20.5	22.4	27.4	29.2	32.2

（下转第175页）

看各处运行情况。

(11) 防误操作功能，测控装置接到主站命令后，对现场运行情况进行分析，能检验命令的合理性。

(12) 事故记录统计，为计划检修提供原始资料。

(13) 可根据用户需求，增加特殊功能。

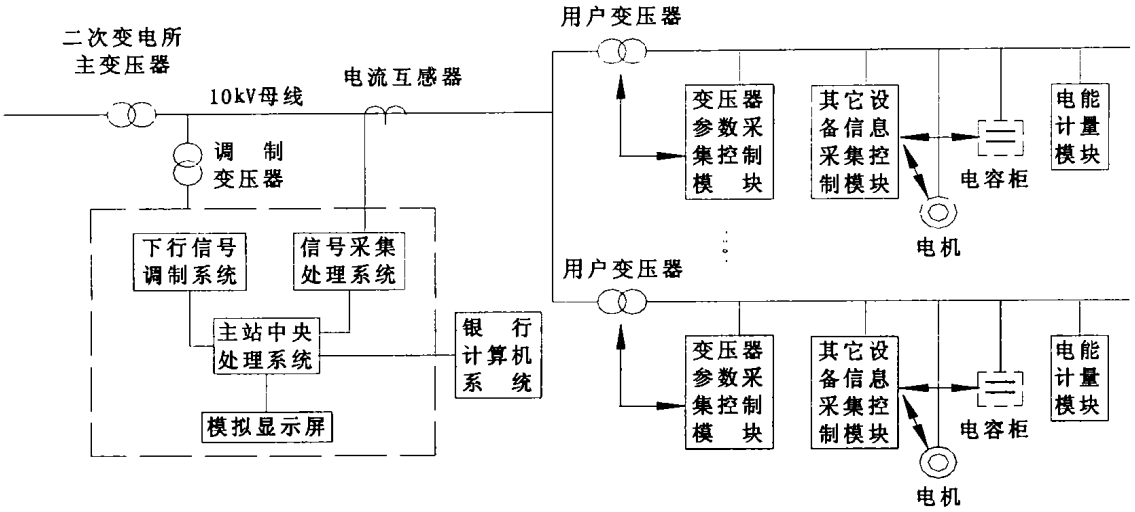

图2 油井采油过程计算机监控系统结构图

该系统主要作用有：(1) 通过配电双向自动通讯可以大大降低实现成本；(2) 可以实现对油井在无人值守的情况下难于及时了解油井工作情况的问题；(3) 使用计算机对获得的信息进行快速分析和解释，进而可以实现对油井的实时控制；(4) 大幅度提高油井采油过程的自动化水平，提高抽油效率和产量，降低采油成本；(5) 通过计算机联网大幅度提高油田管理水平，实现管理自动化。

结束语 该研究成果能解决目前我国油田油井采油过程的自动监测和控制问题。这对改变我国现有油井采油过程自动化水平、采油效率低的状况有着重要的意义，同时它也将会产生很高的经济效益和社会效益。

参 考 文 献

1 吴斌，等. 基于配电网的自动抄表[J]. 电测与仪表，2001(4)
2 Mak S T, Reed D L. TWACS, A New Viable Two-Way Automatic Communication System for Distribution Networks, Part I: Outbound Communication. IEEE Trans. on Power Apparatus and Systems [J], 1982, 101(8): 2941～2949
3 赵学增，吴斌，张绍卿. 基于匹配滤波器的跨变压器台区工频电力通信[J]. 电力系统自动化，2002(4)
4 盛寿麟. 电力系统远动原理. 西安交通大学，水利电力出版社
5 刘贯宇. 电力系统远动技术. 华北电力学院，水利电力出版社

(上接第168页)

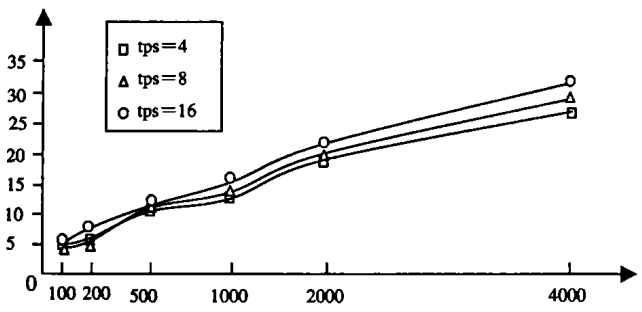

图2 PLBP 的平均加速比曲线

结束语 由于表调度算法本身固有的串行性，基于表调度的并行算法并不多见。已有的并行表调度算法 HPMCP 及 PBSA 都存在一些缺陷。比如，对任务图的划分比较机械，从而导致大部分并行处理器在进行调度前必须估计前面任务节点的完成时间，同时各个并行处理器只能得到子调度结果，而要得到最终的调度结果必须对子调度进行连接处理，这部分工作往往比较复杂。本文在分析串行算法 LBP 的基础上，提出了一种新的基于表调度的并行算法 PLBP，该算法与 HPMCP 及 PBSA 算法相比较，采用了更为灵活的划分方法，克服了上述两种算法必须估计任务节点启动时间的不足之处。在算法的时间复杂度方面，HPMCP 的时间复杂度为 $O(v^2/pps+e+v\log v)$[5]，PBSA 的时间复杂度为 $O(tps^2\times\lceil e/pps\rceil\lceil v/pps\rceil+pps\times\lceil v/pps\rceil^2)$[6]，毫无疑问，PLBP 的时间复杂度比它们都低。另外，PLBP 在调度过程中并未破坏其串行算法的固有串行性，所以其调度性能与其串行算法是完全一致的，这在本文中已经得到证明，而且其时间复杂度与串行算法相比有较大的改善。

参 考 文 献

1 Adam T L, Chandy K M, Dickson J. A Comparison of List Scheduling for Parallel Processing Systems. Communications of the ACM, Dec. 1974, 17: 685～690
2 EL-Rewini H, Lewis T G, Ali H H. Task Scheduling in Parallel and Distributed Systems. Englewood Cliffs, New Jersey: Prentice Hall, 1994
3 Hwang J J, Chow Y C, Anger F D, Lee C Y. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. SIAM Journal on Computing, 1989, 18(2): 244～257
4 Wu M Y, Gajski D D. Hypertool: A Programming Aid for Message-Passing Systems. IEEE Trans. on Parallel and Distributed Systems, 1990, 1(3): 330～343
5 Wu Min-You, Shu Wei. Parallelization of scheduling algorithms. In: Parallel Architecture, Algorithm, and Network, 1996 Proc. Second Intl. Symposium on, June 1996. 357～360
6 Ahmad I, Kwok Yu-Kwong. On parallelizing the multiprocessor scheduling problem. IEEE Trans. on Parallel and Distributed System, 1999, 10(4): 414～431
7 Radulescu A, Van Gemund A J C. Low-cost task scheduling for distributed-memory machine. IEEE Trans. on Parallel and Distributed System, 2002, 13(6): 648～658
8 Buyya R 著. 郑纬民，等译. 高性能集群计算：结构与系统(第一卷). 北京：电子工业出版社，2001