解决矩阵链相乘处理器调度问题的一种新算法*)

徐卫志 王洪国 于 惠 杨 海
(山东师范大学信息科学与工程学院 济南 250014)

摘 要 本文介绍了矩阵链相乘处理器分配问题和离散处理器分配算法,描述了 Lee Heejo 等人提出的解决 MCSP 的处理器分配算法,提出了一种解决 MCSP 的时间复杂度更低的算法,使处理器能尽量被充分利用,并对三种分配算法进行了比较分析。

关键词 矩阵链相乘,处理器分配,贪心算法

A New Algorithm to Solve the MCSP

XU Wei-Zhi WANG Hong-Guo YU Hui YANG Hai
(School of Information Science and Engineering, Shandong Normal University, Jinan 250014)

Abstract In this paper, the matrix chain scheduling problem and two discrete processor allocation algorithms are introduced. Then the algorithm which was proposed by Lee Heejo is described. At last, a new greedy algorithm is proposed, which costs less time and make good use of the processors, and three allocation algorithms are compared and analysed.

Keywords Chain of matrix products, Processor allocation, Greedy algorithm

1 引言

矩阵是数值代数中的一个基本概念,许多应用中都用到较长的矩阵链相乘,如机器人、机器控制和计算机动画[1]。矩阵链相乘顺序不同,数量乘法次数相差很大,对计算效率影响很大。用蛮力方法得到 n 个矩阵乘积最优顺序所需时间是 $\Omega(4^n/\sqrt{n})$[2]。用动态规划解决这一问题的时间复杂度为 $\theta(n^3)$[3]。T. C. Hu 和 M. T. Shing[4,5] 提出时间复杂度为 $O(n\log n)$ 的最优的串行算法。此外,还有学者提出时间复杂度为 $O(n)$ 的近似算法。

许多学者对矩阵链相乘最优顺序问题(MCOP, the matrix chain ordering problem)提出了很多并行算法[6~13],这些算法主要基于 PRAM 模型。Lee Heejo 等[1] 指出,针对 MCOP 提出的算法,解决单个处理器上矩阵相乘的顺序问题,使数量乘法次数最少,在单个处理机上的计算时间也最少;Lee Heejo 首次提出矩阵链相乘处理器调度问题(MCSP, the matrix chain scheduling problem),用并行机做矩阵乘运算,所有的处理器按照解决 MCOP 得到的最优顺序逐个矩阵乘积来计算,并不一定使矩阵乘运算的计算时间最少,需要考虑如何调度多个处理器并行地进行矩阵乘运算。在并行系统中,计算时间受各种因素影响,像任务之间的依赖性、通信延迟和处理器的利用率等;处理器增多并不一定会提高性能,如果处理器分配得不合理,反而会使处理器利用率下降。

矩阵链相乘处理器调度问题是如何找到矩阵链相乘的顺序以及处理器如何调度才能使在一个并行系统上矩阵乘运算时间最少的问题。如果处理器足够多,这个问题可以在多项式时间内解决;当处理器个数是有限的 p 个时,MCSP 是一个 NP 难问题[1]。

本文第 2 部分介绍离散处理器分配算法和 Lee Heejo 提出的算法,第 3 部分提出一种新的解决 MCSP 的贪心算法,第 4 部分对三种算法进行比较分析。

2 离散处理器分配算法和两种解决 MCSP 的算法

2.1 符号

$L_{i,j}$:表示 $M_i\cdots M_j$ 的乘积顺序的顺序树;

$p_{i,j}$:用来计算 $M_i\cdots M_j$ 的处理器数量;

(m_i,m_j,m_k):一个 $m_i\times m_j$ 的矩阵和一个 $m_j\times m_k$ 的矩阵相乘;

$\phi(m_i,m_j,m_k,p)$:p 个处理器计算 $m_i\times m_j$ 的矩阵与 $m_j\times m_k$ 的矩阵相乘需要的时间;

$D(x)$:x 的约数的集合。

2.2 基本概念

在 PRAM-CREW 模型上计算两个 $n\times n$ 的矩阵相乘。当分配的处理器个数超过 n^2 时,处理器的利用率开始下降。为了提高处理器的利用率,最多分配 n^2 个处理器。同样地,$m_i\times m_j$ 的矩阵和 $m_j\times m_k$ 的矩阵相乘,最多分配 m_im_k 个处理器;为使所有处理器负载均衡,限定分配的处理器个数为 $D(m_im_k)$ 中的一个值。例如,用 20 个处理器计算乘积(2,3,8)和(4,4,3),按数量乘法次数的比例分配,各分配 10 个处理器;要使各个处理器的负载均衡,为它们分配的处理器个数分别是 16 和 12 的约数,所以分别分配 8 个和 6 个处理器。同时,计算两个乘积需要 $\max(2\times3\times8/8,4\times4\times3/6)=8$ 单位时间。把剩余的处理器分配给计算时间较长的乘积,分别用 8 个和 12 个处理器来计算,则计算时间为 $\max(2\times3\times8/8,4\times4\times3/12)=6$。按照这种思想分配处理器,得到下面的 DPA 算法。

2.3 两对矩阵乘积的离散处理器分配算法(DPA 算法)

输入:两对矩阵乘积,$X=(m_x,m_{x+1},m_{x+2})$ 和 $Y=(m_y,m_{y+1},m_{y+2})$;p 个处理器

输出:分配给乘积 X 和 Y 的处理器个数 p_x 和 p_y,$1\leqslant$

*)山东省自然科学基金(Q2006G03)。徐卫志 硕士研究生,研究方向:并行算法、矩阵链相乘问题;王洪国 博士后,教授,硕士生导师;于 惠 硕士研究生;杨 海 硕士研究生。

$p_x, p_y \leqslant p$ 且 $p_x + p_y \leqslant p$

(1) $p_{prop} = (m_x m_{x+1} m_{x+2})\, p/(m_x m_{x+1} m_{x+2} + m_y m_{y+1} m_{y+2})$

(2) 从 $D(m_x m_{x+2})$中找到最大的 $d_{x,i}$，并且 $d_{x,i} \geqslant p_{prop}$

(3) 从 $D(m_y m_{y+2})$中找到最大的 $d_{y,j}$，并且 $d_{y,j} \leqslant p - p_{prop}$

(4) If $\phi(m_x, m_{x+1}, m_{x+2}, d_{x,i}) < \phi(m_y, m_{y+1}, m_{y+2}, d_{y,j})$

(5) Then $p_x = d_{x,i}$, $p_y = p - d_{x,i}$

(6) Else $p_x = p - d_{y,j}$, $p_y = d_{y,j}$

DPA 算法使两对矩阵乘积在最短时间内完成[1]。算法第(2)步和第(3)步需要 $O(p)$的时间，其余都是常数，算法的时间复杂度为 $O(p)$。

2.4 DPA-k 算法

将 DPA 算法推广，就得到 k 对矩阵乘积的离散处理器分配算法 DPA-k 算法。

输入：k 对矩阵相乘 $X_i = (m_{i,1}, m_{i,2}, m_{i,3})$，$1 \leqslant i \leqslant k$；$p$ 个处理器

输出：为每对矩阵 X_i 分配的处理器个数 p_i，且满足 $\sum_{i=1}^{k} p_i \leqslant p$

Stage-1 按数量乘法次数的比例分配处理器

Stage-2 把多余的处理器分配给计算时间最长的一对矩阵

DPA-k 算法使独立的 k 对矩阵相乘时间最短，算法的时间复杂性是 $O(p+k)$。

2.5 两种解决 MCSP 的算法

算法 1：所有的处理器同时计算一对矩阵乘积，按最优顺序逐个乘积进行计算。

算法 2：Lee Heejo 等提出的矩阵链相乘多处理器调度算法。

输入：处理器的个数 p；n 个矩阵的维数 $m_1, m_2, \cdots, m_{n+1}$

输出：p 个处理器的分配结果

Stage-1 用一个并行算法解决 MCOP，按照矩阵链相乘最优顺序生成顺序树 L。

Stage-2 根据顺序树 L，按数量乘法次数的比例自顶向下分配处理器。

Stage-3 对于顺序树 L 的所有的叶子，执行以下步骤，直到所有的乘积都被分配处理器：

(1) $M_k M_{k+1}$是顺序树中的两个叶子乘积，$p_{k,k+1}$是分配给它的处理器个数，如果 $p_{k,k+1} < m_k m_{k+2}$，直接转到第 5 步。

(2) 通过搜索两个叶子结点的祖先找到一个候选乘积。如果没有这样的乘积，转到第 5 步。

(3) $M_l M_{l+1}$是通过搜索两个叶子结点的祖先而找到的候选乘积。检验这个候选乘积是否满足与 $M_k M_{k+1}$并行执行的条件，如果不满足，转到第 2 步。

(4) 修改顺序树的结构，使 $M_l M_{l+1}$能与 $M_k M_{k+1}$并行执行。使用 DPA 算法对 $p_{k,k+1}$个处理器重新分配。对新产生的两个叶子乘积 $M_l M_{l+1}$和原来的 $M_k M_{k+1}$，都转到第 1 步。

(5) 为这两个叶子乘积分配 $\min(p_{i,j}, m_k m_{k+1} m_{k+2})$个处理器。把这两个叶子结点的父亲节点作为一个新的叶子结点。

算法 2 的时间复杂度是 $O(n^2 + np)$。

3 解决 MCSP 的一种新算法

3.1 算法思想

用 p 个处理器计算 n 个矩阵相乘，以一种自底向上的过程来分配处理器，与分配完成以后，计算 n 个矩阵乘积的过程是一致的，如图 1。第一步，对 $n/2$ 个矩阵乘法用 DPA-k 算法分配 p 个处理器，第二步对 $n/4$ 个矩阵乘法用 DPA-k 算法分配 p 个处理器，最后一步对两个矩阵乘法用 DPA-k 算法分配 p 个处理器。如果要计算的矩阵个数是奇数，则最后一个矩阵不参与分配，进入下一轮分配，每一次循环的分配结果都存储到二维数组 a 中。

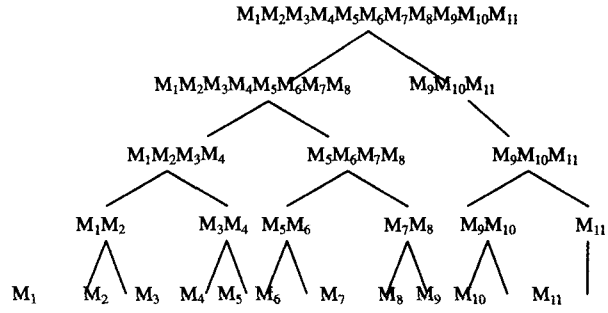

图 1

算法 3：解决 MCSP 的一种新算法

输入：要分配的处理器个数 p；n 个矩阵的维数 $m_0, m_1, \cdots, m_n$

输出：计算 n 个矩阵相乘每一步的处理器分配，保存在二维数组 a 中

算法过程：

Step1 $t=1; x=0$

Step2 while $(t<n)$

Step3 $x=x+1; s=t; t=2s; i=0; c=0;$

Step4 计算数量乘法总的次数 c

(1) while $(i+t \leqslant n)$

(2) $c=c+m_i m_{i+s} m_{i+t}; i=i+t$

(3) end while

(4) if $(i+s<n)$ then

(5) $c=c+m_i m_{i+s} m_n$

(6) end if

Step5 按比例分配处理器

(1) $i=0$

(2) while $(i+t \leqslant n)$

(3) $a[x][i] = (m_i m_{i+s} m_{i+t}/c)\, p$

(4) 找到 $D(m_i m_{i+t})$中满足 $d_{i,j} \leqslant a[x][i]$的最大 $d_{i,j}$

(5) $a[x][i] = d_{i,j}; i=i+t$

(6) end while

(7) if $i+s<n$ then

(8) $a[x][i] = (m_i m_{i+s} m_n/c)\, p$

(9) 找到 $D(m_i m_n)$中满足 $d_{i,j} \leqslant a[x][i]$的最大 $d_{i,j}$

(10) $a[x][i] = d_{i,j}$

(11) end if

Step6 把多余的处理器分配给计算时间最长的一对矩阵

(1) while $(p - \sum_{i=0} a[x][i] > 0)$

(2) 找到 $\phi(m_i, m_{i+s}, m_{i+t}, a[x][i])$的值最大的一对矩阵 X_i

(3) if $(a[x][i] < m_i m_{i+t})$ then

(4) 找到 $D(m_i m_{i+t})$中满足 $d_{i,j} > a[x][i]$的最小 $d_{i,j}$

(5) if $(p - \sum_{k=0} a[x][k] - (d_{i,j} - a[x][i])) > 0)$ then

(6) $a[x][i] = d_{i,j}$

(7) else break

(8) end if

(9) else break

(10) end if

(11) end while

Step7 end while

3.3 算法分析

最外层 step2 至 step7 循环一次，矩阵乘积的个数减半，Step4 的运行时间可以表示为 $n/2+n/4+n/8+\cdots+1=O(n)$；Step5 循环一次时间复杂度是 $O(p)$，外层循环大约要运行 $\log n$ 次，所以时间复杂度为 $O(p\log n)$；最外层每循环一次，Step6 的循环执行次数设为常数 k，则 Step6 第(2)步的时间复杂度可表示为 $kn/2+kn/4+kn/8+\cdots+1=O(n)$，而 Step6 第(4)步的时间复杂度不会超过 $O(p)$，所以整个算法的时间复杂度为 $O(n+p\log n)$。

算法 3 中，采用贪心策略，外层每循环一次，用 DPA-k 算法分配一次处理器，大约 $\log n$ 次处理器分配，每一次处理器分配都是最优的。

4 算法比较

4.1 时间复杂度比较

算法 2 需要 $O(n^2+np)$ 的时间，算法 3 改进了时间复杂度，需要 $O(n+p\log n)$ 的时间。当 n 和 p 在一个数量级上时，两个算法的复杂度简化为 $O(n^2)$ 和 $O(n\log n)$。

4.2 举例分析

设 5 个矩阵相乘 M_1：5×10，M_2：10×4，M_3：4×6，M_4：6×10，M_5：10×2 用 60 个处理器计算，解决 MCOP 得到的最优计算顺序是 $(M_1(M_2(M_3(M_4M_5))))$，如图 2。

根据算法 1，用 60 个处理器按最优顺序逐个乘积进行计算，所用时间为

(6×10×2)/min(60,6×2)+(4×6×2)/min(60,4×2)+(10×4×2)/min(60,10×2)+(5×10×2)/min(60,5×2)=30。

根据算法 2，将调整树的结构，如图 3，其中 M_2M_3 和 M_4M_5 并行执行，用 DPA 算法给这两个乘积分配处理器，分别为 30 个和 12 个处理器，计算时间是

max(10×4×6/min(30,10×6),6×10×2/min(12,6×2))+10×6×2/min(60,10×2)+5×10×2/min(60,5×2)=26。

根据算法 3，如图 4，其中 M_1M_2 和 M_3M_4 并行执行，用 DPA 算法给这两个乘积分配处理器，分别为 20 个和 40 个处理器，计算时间是

max(5×10×4/min(20,5×4),4×6×10/min(40,4×10))+5×4×10/min(60,5×10)+5×10×2/min(60,5×2)=24

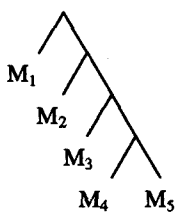

图 2

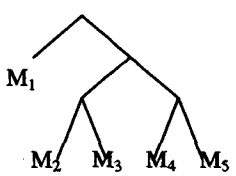

图 3

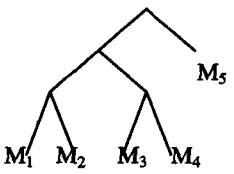

图 4

因此，当处理器个数很少时，处理器能充分利用，所有处理器按最优顺序逐个计算矩阵的乘积，由于数量乘次数最少，计算时间也最少；当处理器个数增多时，按算法 1 计算，有一部分处理器不能充分利用，而另外两种算法按一定的规则分配处理器，处理器能尽量的充分利用，计算时间少于算法 1。如表 1，当处理器个数超过 40 时，按算法 2 和算法 3 分配的结果计算，计算时间少于算法 1。

当处理器个数增加到一定数量时，按算法 2 和算法 3 分配的结果计算，计算时间都不再减少，按这两种算法分配的结果计算的计算时间是非常接近的。如表 1，当处理器的个数超过 60 时，两个算法的计算时间分别为 26 和 24，计算时间不再减少，算法 3 使用的时间要比算法 2 少。

表 1

处理器个数	10	20	30	40	50	60	70	80
算法 1 时间	44	30	30	30	30	30	30	30
算法 2 时间	44	30	30	26	26	26	26	26
算法 3 时间	78	54	38	30	30	24	24	24

结束语 本文提出了一个解决 MCSP 的新算法，它的时间复杂度优于 Lee Heejo 等提出的算法，当处理器个数较多时，按该算法分配处理器，使处理器尽量被充分利用，减少了矩阵链相乘的计算时间。今后研究的重点是找到在多项式时间内解决 MCSP 的最优算法，提出时间复杂度更低和效果更好的近似算法，探索解决这一问题的其他方法。

参 考 文 献

1 Lee Heejo, Kim Jong, Hong Sungje, et al. Parallelizing Matrix Chain Products:[Tech Rep]. CS-HPC-97003. Pohang University of Science and Technology, 1997

2 Gould H. Bell and Catalan numbers. Combinatorial Research Institute, Morgantown, WV. June 1977

3 Godbole S. An Efficient Computation of Matrix Chain Products. IEEE Trans on Computers, Sept. 1973. 864～866

4 Hu T C, Shing M T. Computation of Matrix Chain Products. Part Ⅰ. SIAM J Compute, May 1982, 11:362～373

5 Hu T C, Shing M T. Computation of Matrix Chain Products. Part Ⅱ. SIAM J Compute, May 1984, 13:228～251

6 Valiant L, Skyum S, Berkowitz S, et al. Fast Parallel Computation of Polynomials Using Few Processors. SIAM Journal on Computing, 1983, 12: 641～644

7 Rytter W. Note on Efficient Parallel Computations for Some Dynamic Programming Problems. Theoret Comp Sci. 1988, 59:297～307

8 Huang S H S, Liu H, Viswanathan V. Parallel Dynamic Programming. IEEE Trans. on Parallel and Distributed Systems, Mar. 1994, 5:326～328

9 Bradford P G, Rawlins G J, Shannon G E. Efficient Matrix Chain Ordering in Polylogtime. In: Proc. of Int'l Parallel Processing Symp, 1994. 234～241

10 Czumaj A. Parallel Algorithm for the Matrix Chain Product and the Optimal Triangulation Problems. In: Proc. of Symp on Theoret Aspects of Computer Science, Springer Verlag, 1993. 294～305

11 Czumaj A. Very Fast Approximation of the Matrix Chain Product Problem. J Algorithms, 1996, 21(1): 71～79

12 Ramanan P. An Efficient Parallel Algorithm for the Matrix Chain Product Problem. SIAM J Comput, Aug. 1996, 25:874～893

13 Strate S A, Wainwright R L. Parallelization of the Dynamic Programming Algorithm for the Matrix Chain Product on a Hypercube. IEEE, 1990