MIMD 并行机上解决矩阵链乘序问题的算法研究*)

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 介绍了并行机向 MIMD 的发展趋势,描述了 MIMD 并行机上解决矩阵链乘序问题的算法,并对其复杂度进行了分析。针对处理器之间任务分配的问题,提出了一种合理分配任务的算法,并对这种算法的复杂度进行了分析。

关键词 多指令流多数据流,矩阵链乘序问题,任务分配

Research on Algorithms Solving the Matrix China Ordering Problem on MIMD Parallel Computer

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

Abstract The developing trend of the MIMD computer is introduced first. Then an algorithm solving the matrix chain ordering problem on MIMD computer is described and analysed. At last, in order to solve the task allocation problem between the computers, a new algorithm is proposed and analysed.

Keywords MIMD, MCOP, Task allocation

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。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 $\Omega(4^n/\sqrt{n})$[1]。用动态规划解决这一问题的时间复杂度为 $O(n^3)$[2]。T. C. Hu 和 M. T. Shing 提出时间复杂度为 $O(n\log n)$的最优的串行算法[3,4]。此外,还有学者提出时间复杂度为 $O(n)$的近似算法[5]。

许多学者提出了很多解决矩阵链相乘最优顺序问题(MCOP)的并行算法[6-13],这些算法主要基于 SIMD-PRAM 模型。Heejo lee 等[14]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;Heejo lee 首次提出矩阵链相乘处理器调度问题(MCSP),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。

根据指令流和数据流的不同,通常可将计算机系统分为:单指令流单数据流(SISD)、单指令流多数据流(SIMD)、多指令流单数据流(MISD)和多指令流多数据流(MIMD)。绝大多数串行计算机属于 SISD;直到目前,国内外还没有见到多指令流单数据流方面的计算机产品;在 SIMD 并行机中,各处理器同一时刻执行相同的指令,处理不同的数据;在 MIMD 并行机中,各处理器执行不同的指令,且处理不同的数据。

SIMD 类型并行机对并行计算机的发展起到了重要推动作用,但由于微处理芯片技术的发展,20 世纪 90 年代以后,并行机均朝 MIMD 方向发展,单处理机性能都非常强大,用于科学与工程计算的 SIMD 类型并行机已基本退出了历史舞台。促使 MIMD 结构发展的因素有:(1)它能支持更高的并行度(子程序、任务级或更高)。(2)多台处理机的性能价格比远比一台处理机高。

已提出的解决矩阵链相乘最优顺序问题(MCOP)的并行算法主要基于 SIMD-PRAM 模型,随着并行机的不断发展,研究 MIMD 并行机上的并行算法具有重要意义。

本文第 2 部分介绍 MIMD 分布存储并行机上解决矩阵链乘序问题的算法,第 3 部分提出一种多处理机任务分配算法,最后是结论。

2 MIMD 并行机上解决矩阵链乘序问题的算法及复杂度分析

Steve A. Strate 提出了超立方结构上解决矩阵链乘序问题动态规划算法,实际上,在线性阵列上也可以实现这个算法,并且 Steve A. Strate 的文章中,并没提出计算任务分配算法,只是人工分配。本节描述在 MIMD 分布存储机器上解决矩阵链乘序问题的动态规划算法,而 MIMD 共享存储机器上的算法与这个算法基本相同,不同之处在于处理器之间的通信是通过共享内存。

2.1 算法描述

(1)任务分配,为每个处理器分配若干行数据项。

(2)每个处理器将已分配给它的计算任务自左向右,逐列地进行计算,在每一列都是自底向上地进行计算。

(3)第 i 个处理器计算完一列数据项后,将它所存储的该列的数据项传给它前面的处理器(第 $i-1$ 个处理器)。

(4)第 i 处理器,如果它需要的第 $i+1$ 个处理器的数据还没传送到,那么第 i 个处理器将处于等待状态。

2.2 算法举例

在下面的例子中,两个处理器组成线性阵列,按照基本运算的次数来分配任务,每一个处理器负责计算若干行的数据,

*)基金项目:山东省自然科学基金(Q2006G03)。徐卫志 硕士研究生,研究方向为并行算法、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。

由于第 i 行的计算量大于第 $i+1$ 行，因此两个处理器分的行数不同，参见图 1。用两个处理器来计算，处理器 1 计算前两行，处理器 2 计算后四行，计算方法是逐列进行计算。

处理器 1 自左向右地逐列计算，每一列都是自底向上地计算，首先计算它的第一列 $C[1,1]$，然后依次是第二列 $C[2,2]$ 和 $C[1,2]$；第三列 $C[2,3]$ 和 $C[1,3]$ 等等；计算 $C[1,3]$ 时，如果处理器 2 还没有把 $C[3,3]$ 传过来，那么处理器 1 等待。

处理器 2 也是自左向右地逐列计算，每一列都是自底向上地计算，每计算完成一列就把这一列的计算结果传给处理器 1。

表 1　用两个处理器计算的例子

	d=0	d=1	d=2	d=3	d=4	d=5	
处理器 1	C[1,1]	C[1,2]	C[1,3]	C[1,4]	C[1,5]	C[1,6]	i=1
		C[2,2]	C[2,3]	C[2,4]	C[2,5]	C[2,6]	i=2
处理器 2			C[3,3]	C[3,4]	C[3,5]	C[3,6]	i=3
				C[4,4]	C[4,5]	C[4,6]	i=4
					C[5,5]	C[5,6]	i=5
						C[6,6]	i=6

2.3　算法复杂分析

(1)时间及空间复杂性分析

由于把计算任务大致平均地分给 p 个处理器，而串行算法的时间复杂度是 $O(n^3)$，因此该并行算法的时间复杂度是 $O(n^3/p)$。

第 1 个处理器最坏情况下需要存储所有的数据项，第 i 个处理器最坏情况下需要存储它分得的几行数据项以及这几行以下的所有数据项。所以算法的空间复杂度为 $O(n^2)$。

(2)通信复杂度分析

$O(n^2)$ 个数据项分配给 p 个处理器。每个处理器(除了第一个)每次传送一列数据，第二个处理器传送 $n-l_1$ 列(l_1 是第一个处理器分得的行数)，第三个处理器传送 $n-l_1-l_2$ 列(l_2 是第二个处理器分得的行数)等等。因此算法的通信总次数是 $O(n^2)$，平均通信复杂度是 $O(n^2/p)$。

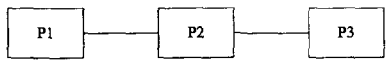

图 1　三个处理器的线性阵列

3　处理器之间的任务分配算法

3.1　算法思想

在动态规划算法中，主要计算时间用于计算式子 $C[i,j]=\{C[i,k-1]+C[k,j]+r[i]r[k]r[j+1]\}$，其中包括三种基本运算，乘法、加法和取最小值。在两个数中取最小值需要进行一次比较，其运算代价看成 x，则在四个值中取最小值代价为 $3x$；整数乘法的代价看成 y，整数加法的代价看成 z。假设三种运算代价之间是这样一种关系，乘法代价是加法的 a 倍，比较运算是加法的 b 倍，就可以把乘法和比较运算的代价转化为加法的代价。

例如计算 $C[1,5]$ 时，参见表 1，需要在四个值中取最小值，计算这四个值都需要两次整数乘法和两次整数加法。计算 $C[1,5]$ 的代价是 $3x+8y+8z=3bz+8az+8z=(3b+8a+8)z$，由于在同一对角线上的数据项的计算代价是相同的，因此 $C[2,6]$ 的计算代价也是 $(3b+8a+8)z$。

处理器为 p 个，矩阵个数为 n，计算出每一行的代价，加起来得到总的代价 C，则 C/p 就是每个处理器平均的计算量，然后自上而下逐行地分配计算任务，使每个处理器的计算量接近于 C/p。按照平均值逐行分配计算任务的过程中，采用这样的策略，对于其中某一行，如果它的一半以上都分给了一个处理器，那么这一行都分给这个处理器，否则，这一行就不分给这个处理器。

由于同一对角线上的数据项的计算量是相同的，第 $i-1$ 行比第 i 行多一个数据项，因此如果从第 $n-1$ 行开始向上计算总的计算任务，则第 i 行比第 $i+1$ 行多 $n-i$ 次比较，$2(n-i)$ 次乘法和 $2(n-i)$ 次加法。

3.2　算法描述

用数组 $a[n]$，$b[n]$ 和 $c[n]$ 分别存储各行比较运算、乘法运算和加法运算的次数。$a[i]$，$b[i]$ 和 $c[i]$ 分别表示第 i 行的比较、乘法和加法的运算次数。$L[i]$ 表示第 i 行的三种运算总计算量，C 表示 n 行三种运算的总计算量。数组 $d[n]$ 存储最终分配结果，$d[i]$ 表示第 i 行分配给第 $d[i]$ 个处理器。

(1)计算总任务量 C

各个变量和数组初始值均为 0；

```
for(i=1;i<n;i++)
  {
     a[n-i]=a[n-i+1]+i-1;
     b[n-i]=b[n-i+1]+2i;
     c[n-i]=c[n-i+1]+2i;
  }
for(i=1;i<=n;i++)
{
   L[i]=a[i]*b+b[i]*a+c[i];// 乘法代价是加法的 a 倍，比较
   运算是加法的 b 倍
   C=C+L[i];

    }
```

(2)分配计算任务

```
average=C/p; //求 p 个处理器的平均计算量
t=average;j=1; //j 表示第 j 个处理器
for(i=1;i<=n;i++)
{
   if(t>L[i]/2)
     {d[i]=j;t=t-L[i];}
   else
     {t=average;j=j+1;}
}
```

3.3　算法复杂度分析

由于算法中最多只有一层循环，算法的时间复杂度是 $O(n)$；算法中只用到一维数组和一些变量，算法的空间复杂度也是 $O(n)$。

结束语　本文指出了并行机向 MIMD 的发展趋势，介绍了 MIMD 并行机上解决矩阵链乘序问题的算法，并提出了一种合理分配处理器之间任务的算法。

本文提出 MIMD 并行机上解决矩阵链乘序问题的算法，前面的处理器需要后面的处理器传送同一列的数据，如果后面处理器没有计算完成这一列，那么前面的处理器将处于等待，怎样减少等待时间是下一步需要解决的问题，这个问题也涉及到计算任务的分配。

本文提出的任务分配算法，采用逐行分配的策略，按照这种方式分配，可能会导致分配不均。考虑 n 和 p 的关系，如果 n 远大于 p，那么分给每个处理器的行数就很多，一行的误差对分配结果的影响不大；如果 n 与 p 相差不大，那么一行的误差对分配结果的影响很大，甚至会导致算法无法进行。可以在分界的这一行逐个数据项分配，这样使分配结果更均匀，但是增加了算法的时间复杂度。

参 考 文 献

[1]　Bell H G, Numbers C. Combinatorial Research Institute, Mor-

gantown, WV. June 1977

[2] Godbole S. An Efficient Computation of Matrix China Products. IEEE Trans. on Computers, Sept. 1973: 864-866

[3] Hu T C, Shing M T. Computation of Matrix China Products. part Ⅰ. SIAM J. Compute, May 1982, 11: 362-373

[4] Hu T C, Shing M T. Computation of Matrix China Products. part Ⅱ. SIAM J. Compute, May 1984, 13: 228-251

[5] Chin F. An O(n) Algorithm for Determining a Near-Optimal Computation Order of Matrix China Product. Comm. ACM, 1978: 544-549

[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 China O-rdering in Polylogtime // Proc. of Int'l Parallel Processing Symp. 1994: 234-241

[10] Czumaj A. Parallel Algorithm for the Matrix China Product and t-he Optimal Triangulation Problems // Proc. of Symp. on Theoret. Aspects of Computer Science. Springer Verlag, 1993: 294-305

[11] Czumaj A. Very Fast Approximation of the Matrix China Product Problem. J. Algorithms, 1996, 21(1): 71-79

[12] Ramanan P. An Efficient Parallel Algorithm for the Matrix C-hain 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 China Product on a Hypercube. IEEE, 1990

[14] Lee H, Kim J, Hong S, et al. Parallelizing Matrix China Products. Tech. Rep. CS-HPC-97003. Pohang University of Science and Technology, 1997

(上接第 171 页)

理解为正确实体链的比率。因此 Luo 将由式(30)产生的式(27)称为基于提及(mention-based)的 CEAF 方法；由式(31)得到的式(27)称为基于实体(entity-based)的 CEAF 方法。不同的层次上，可对 $\phi_3(\cdot,\cdot)$和 $\phi_4(\cdot,\cdot)$定义相应的权重值，如根据不同实体类(ACE 中为 NAME, NOMINAL, PRONOUN)定义不同权重值。

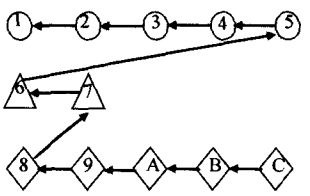

图 4 系统输出情况图

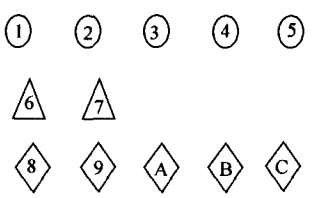

图 5 系统输出情况图

表 1 各类评测算法准确率、召回率及调和 F-值情况比较

图例	MUC			B^3			CEAF					
	P	R	F	P	R	F	ϕ_3-P	ϕ_3-R	ϕ_3-F	ϕ_4-P	ϕ_4-R	ϕ_4-F
2	0.9	1	0.947	0.76	1	0.864	0.833	0.833	0.833	0.917	0.611	0.733
3	0.9	1	0.947	0.58	1	0.734	0.583	0.583	0.583	0.833	0.555	0.667
4	0.918	1	0.9	0.375	1	0.54	0.417	0.417	0.417	0.588	0.196	0.294
5	—	—	—	1	0.25	0.4	0.25	0.25	0.25	0.111	0.444	0.178

最后依然以图 2,3 中数据为基准，同时引入图 4,5 两种极端输出，针对不同 CEAF 方法进行计算，结果见表 1，表中第三行至第六行分别对应图 2—5 情况。为了方便比较分析，将 B^3 与 MUC-6 两种方法针对图 4,5 的情况一同给出。分析可以看到，尽管 B^3 最终的 F 值比 MUC-6 理想一些，但它在极端条件下，准确率和召回率上均出现了不合理值。而 CEAF 方法的三类指标的结果更具说服力，也更为合理直观。最后需要指出的是 Luo 中提出的最佳对齐方法是一个经典的最大双向匹配问题，其算法思想得益于 Kuhn(1955)和 Munkres(1957)提出的 Kuhn-Munkres 算法[5,6]。

结束语 MUC-6 算法的缺点主要有两个：一个是算法没有区分单集，因此无法正确处理类似图 5 中的输出情况；一个是所有的错误都被等同看待，算法仅是平等的对各种类型错误进行惩罚。B^3 算法采用基于实体的策略，但它与 MUC-6 一样存在同一实体被多次匹配重复计算问题，仍存在 F-值不合理现象。ACE-Value 是近年来共指消解任务的官方评测算法，但是在分值解释方面不直观，除此之外，由于该算法严格依赖其 ACE 评测标注语料，该语料无法公开共享使用，评测中的诸多细节也少见诸公开文献。CEAF 方法含义清晰，理解直观，评分解释合乎实际。目前，汉语共指消解处理尚无大规模共享标注语料，依靠实体与提及的绝对和相对数目的统计进行客观比较，综上所述，CEAF 不失为一种最佳选择，特别是它能够根据实际需要决定不同权重参与的灵活性，更加适合于当前共指消解问题仅能在限定领域开展较为深入的工作的实际情况。

参 考 文 献

[1] Luo Xiaoqiang. On Coreference Resolution Performance Metrics // Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP). Vancouver, October 2005: 25-32

[2] Vilain, Mare, et al. A Model-Theoretic Coreference Scoring Scheme // Proceedings of the 6th Message Understanding Conference (MUC-6). Nov. 1995: 45-52

[3] Bagga A, Baldwin B. Algorithms for scoring coreference chains // Proceedings of Linguistic Coreference Workshop at the 1st International Conference on Language Resources and Evaluation. 1998: 563-566

[4] ACE 07. The ACE evaluation plan. www.nist.gov/speech/tests/ace/index.htm.-ace07-evalplan.v1.3a.pdf.

[5] Kuhn H W. The hungarian method for the assignment problem. Naval Research Logistics Quarterly, 1955 (2): 83

[6] Munkres J. Algorithms for the assignment and transportation problems. Journal of SIAM, 1957 (5): 32-38