超级计算机上矩阵乘的并行计算与实现*

伍湘君　黄丽萍
(中国气象科学研究院,北京 100081)

摘　要

数值预报系统中经常要用到矩阵乘运算。在分布式超级计算机(如 IBM-SP)上,矩阵乘的并行计算需要较多的数据移动,有效的数据传输对矩阵乘的实现至关重要。该文讨论了两种矩阵乘的并行算法,一种是基于矩阵的列-行划分方式,一种是基于矩阵的网格划分方式。在 IBM-SP 计算机上的实验结果表明,网格划分的矩阵乘并行算法通讯开销更小,并行效率更高,其并行加速比较列-行并行算法改善约 10%。

关键词:数值预报　矩阵乘　并行计算　分布式并行计算机　数据通讯

引　言

矩阵是数值代数中的一个基本概念,许多科学计算问题往往归结为对矩阵的操作。如在数值预报系统中,就要经常用到矩阵乘计算。因此,如何在超级计算机上高效地完成矩阵乘的计算,对提高许多数值预报模式的运行效率至关重要。

众所周知,在分布式主存并行计算机系统上,数据划分对数据并行程序运行效率影响重大。在进行大型矩阵计算时,由于受到机器内存容量的限制,必需进行数据传输,这些数据的移动要花费相当的通讯时间。因此,在分布式环境上设计矩阵运算的并行算法,不但要挖掘和开发其内在并行性,同时也要选择最优的数据分布形式,以减少数据通讯开销。

如果没有任何优化策略,两个 $N \times N$ 阶矩阵相乘,标量运算复杂性为 $O(N^3)$。20 世纪 60 年代末,Strassen[1]提出了一个新的快速算法,其标量运算只需 $O(N^{\log_2 7})$。在分布式内存系统上,利用张量代数实现的公式化矩阵乘,使矩阵乘的计算效率大大提高了一步。在 Nagesh Anupindi 等人①②提出的算法中,矩阵采用列-行划分方式,从而省去了计算中乘数与被乘数的数据移动,只在求累加和时进行一次中间结果的置换。由于在许多大型科学计算中,数据更多地采用网格划分的方式,文献[2]中介绍了一种网格并行的 Cannon 算法,但其程序实现比较复杂。因此,我们在本文中提出一种新的矩阵乘网格划

* "十五"国家重点科技攻关计划项目(2001BA607B)"中国气象数值预报技术创新研究"资助。
2003-09-08 收到,2004-02-11 收到修改稿。

① Nagesh Anupindi, Qing Yang, Myoung An. Formulating Data-Partition and Migration in Distributed Memory Multiprocessors. (private communication)

② Nagesh Anupindi, Myoung An, Qing Ying. Parallel Matrix Multiplication Algorithm For Rectangular Arrays. (private communication)

分算法。在 IBM-SP 计算机系统上的运行结果表明,新的网格划分算法容易实现,其通讯开销比列-行算法有所降低,其并行效率也有一定的提高,是一种有效的矩阵乘并行计算算法。

1　矩阵乘列-行划分算法

我们先讨论矩阵乘的列-行划分算法[②]。

设有 $N_1 \times N_2$ 阶矩阵 $\boldsymbol{A}$ 及 $N_2 \times N_3$ 阶矩阵 $\boldsymbol{B}$,计算 $N_1 \times N_3$ 阶矩阵 $\boldsymbol{C}$:

$$\boldsymbol{C} = \boldsymbol{AB} \tag{1}$$

设在分布式主存多机系统中共有 k 台处理机,这里总假定 k 能整除 N_1, N_2, N_3。将 $\boldsymbol{A}$ 按列、$\boldsymbol{B}$ 按行均分于 k 台处理机上,结果,处理机 i 包含了 $\boldsymbol{A}$ 的第 i 个子列阵 $\boldsymbol{A}^{(i)}_{N_1 \times N_2/k}$、$\boldsymbol{B}$ 的第 i 个子行阵 $\boldsymbol{B}^{(i)}_{N_2/k \times N_3}$ $(0 \leqslant i \leqslant k-1)$。计算之前,我们将 $\boldsymbol{A}^{(i)}$ 均分成 k 块: $\boldsymbol{A}(j,i)$, $0 \leqslant j \leqslant k-1$。若结果 $\boldsymbol{C}$ 亦按行分布于各处理机上,即第 t 个处理机上 $\boldsymbol{C}$ 子行阵为 $\boldsymbol{C}^{(t)}_{N_1/k \times N_3}$,则有

$$\boldsymbol{C}(t) = \sum_{l=0}^{k-1} \boldsymbol{A}(t,l)\,\boldsymbol{B}(l), \quad 0 \leqslant t \leqslant k-1 \tag{2}$$

在处理机 i 上,分别计算 $\boldsymbol{A}(j,i)\,\boldsymbol{B}(i)$, $0 \leqslant j \leqslant k-1$,则 k 步计算之后,处理机 i 上为 k 个 $N_1/k \times N_3$ 的子积矩阵块:

$$\begin{vmatrix} \boldsymbol{A}(0,i)\,\boldsymbol{B}(i) \\ \boldsymbol{A}(1,i)\,\boldsymbol{B}(i) \\ \vdots \\ \boldsymbol{A}(k-1,i)\,\boldsymbol{B}(i) \end{vmatrix}$$

由于

$$\begin{vmatrix} \boldsymbol{A}(0,0)\,\boldsymbol{B}(0) \\ \boldsymbol{A}(0,1)\,\boldsymbol{B}(1) \\ \vdots \\ \boldsymbol{A}(0,k-1)\,\boldsymbol{B}(k-1) \\ \boldsymbol{A}(1,0)\,\boldsymbol{B}(0) \\ \boldsymbol{A}(1,1)\,\boldsymbol{B}(1) \\ \vdots \\ \boldsymbol{A}(1,k-1)\,\boldsymbol{B}(k-1) \\ \vdots \\ \boldsymbol{A}(k-1,k-1)\,\boldsymbol{B}(k-1) \end{vmatrix} = \left| \boldsymbol{P}(k^2,k) \otimes \boldsymbol{I}_{N_1 N_3/k} \right| \begin{vmatrix} \boldsymbol{A}(0,0)\,\boldsymbol{B}(0) \\ \boldsymbol{A}(1,0)\,\boldsymbol{B}(0) \\ \vdots \\ \boldsymbol{A}(k-1,0)\,\boldsymbol{B}(0) \\ \boldsymbol{A}(0,1)\,\boldsymbol{B}(1) \\ \boldsymbol{A}(1,1)\,\boldsymbol{B}(1) \\ \vdots \\ \boldsymbol{A}(k-1,1)\,\boldsymbol{B}(1) \\ \vdots \\ \boldsymbol{A}(k-1,k-1)\,\boldsymbol{B}(k-1) \end{vmatrix} \tag{3}$$

其中 $\otimes$ 为张量乘, $\boldsymbol{I}$ 为恒等矩阵, $\boldsymbol{P}$ 为跨步置换矩阵,其定义如下:

$$P_{ij}(LS,S) = \begin{cases} 1, & i = (t-1)L + u,\ j = (u-1)S + t \\ & 1 \leqslant t \leqslant S,\ 1 \leqslant u \leqslant L \\ 0, & \text{否则} \end{cases}$$

显然可见,处理机 i 经过 $k-1$ 次与其它处理机的数据交换,即可得到 $\boldsymbol{C}(i)$ 所有和数子积矩阵块。这相当于进行了一次矩阵置换,此刻的矩阵元素就是子积矩阵块。

最后,各处理机做子积矩阵累加,从而完成 $\boldsymbol{C}$ 各子行阵的计算:

$$\begin{vmatrix} \boldsymbol{C}(0) \\ \boldsymbol{C}(1) \\ \vdots \\ \boldsymbol{C}(i) \\ \vdots \\ \boldsymbol{C}(k-1) \end{vmatrix} = \begin{vmatrix} \sum_{j=0}^{k-1} \boldsymbol{A}(0,j)\ \boldsymbol{B}(j) \\ \sum_{j=0}^{k-1} \boldsymbol{A}(1,j)\ \boldsymbol{B}(j) \\ \vdots \\ \sum_{j=0}^{k-1} \boldsymbol{A}(i,j)\ \boldsymbol{B}(j) \\ \vdots \\ \sum_{j=0}^{k-1} \boldsymbol{A}(k-1,j)\ \boldsymbol{B}(j) \end{vmatrix} \tag{4}$$

从算法的描述中可见,其主要计算为 k 次子矩阵乘法,子矩阵规模分别为 $\boldsymbol{A}:(N_1/k, N_2/k)$ 和 $\boldsymbol{B}:(N_2/k, N_3)$。数据通讯主要存在于最后的矩阵置换中,每个处理机均要与其它 $k-1$ 个处理机进行数据交换,这可以用 MPI 的一个全互换函数(MPI-ALLTOALL)完成,其通讯开销为:

$$t_{c\text{-}r} = t_s + (k-1)\ N_1\ N_3\ t_e/k \tag{5}$$

这里,t_s 为通讯起步时间,t_e 为单数据通讯时间。

在矩阵乘的列-行算法中,$\boldsymbol{A}$,$\boldsymbol{B}$ 两矩阵的划分方式是不一致的。下面我们来讨论一种网格划分算法,它比较易于与其它计算接口,且在一定程度上可减少通讯开销。

2 网格划分算法

下面我们给出矩阵乘的网格划分算法。

同样针对式(1),假定 $k = k_s^2$,N_1,N_2,N_3 能被 k_s 整除。现将 $\boldsymbol{A}$ 做网格划分并分配到各处理机上,记 (i,j) 处理机上的子矩阵块为 $\boldsymbol{A}(i,j)$,将 $\boldsymbol{B}$ 按 $\boldsymbol{A}$ 同样的方式做网格划分,但在 (i,j) 处理机上分配 $\boldsymbol{B}$ 的 (j,i) 子矩阵块,即 $\boldsymbol{B}(j,i)$,则有

$$\boldsymbol{C}(i,j) = \sum_{l=0}^{k_s-1} \boldsymbol{A}(i,l)\ \boldsymbol{B}(l,j) \tag{6}$$

如 $k_s = 3$ 时,

$$\begin{vmatrix} \boldsymbol{C}(0,0) & \boldsymbol{C}(0,1) & \boldsymbol{C}(0,2) \\ \boldsymbol{C}(1,0) & \boldsymbol{C}(1,1) & \boldsymbol{C}(1,2) \\ \boldsymbol{C}(2,0) & \boldsymbol{C}(2,1) & \boldsymbol{C}(2,2) \end{vmatrix} =$$

$$\begin{vmatrix} \boldsymbol{A}(0,0) & \boldsymbol{A}(0,1) & \boldsymbol{A}(0,2) \\ \boldsymbol{A}(1,0) & \boldsymbol{A}(1,1) & \boldsymbol{A}(1,2) \\ \boldsymbol{A}(2,0) & \boldsymbol{A}(2,1) & \boldsymbol{A}(2,2) \end{vmatrix} \begin{vmatrix} \boldsymbol{B}(0,0) & \boldsymbol{B}(0,1) & \boldsymbol{B}(0,2) \\ \boldsymbol{B}(1,0) & \boldsymbol{B}(1,1) & \boldsymbol{B}(1,2) \\ \boldsymbol{B}(2,0) & \boldsymbol{B}(2,1) & \boldsymbol{B}(2,2) \end{vmatrix}$$

由于 $\boldsymbol{A}(i,l)$ 与 $\boldsymbol{B}(l,j)$ 同处于第 l 列处理机,因此总可以通过 $\boldsymbol{B}$ 在列上的移动而使它们处于同一处理机(见图 1)。

[初始]

$A(0,0)\ B(0,0)$	$A(0,1)\ B(1,0)$	$A(0,2)\ B(2,0)$
$A(1,0)\ B(0,1)$	$A(1,1)\ B(1,1)$	$A(1,2)\ B(2,1)$
$A(2,0)\ B(0,2)$	$A(2,1)\ B(1,2)$	$A(2,2)\ B(2,2)$

[B 上旋一行]

$A(0,0)\ B(0,1)$	$A(0,1)\ B(1,1)$	$A(0,2)\ B(2,1)$
$A(1,0)\ B(0,2)$	$A(1,1)\ B(1,2)$	$A(1,2)\ B(2,2)$
$A(2,0)\ B(0,0)$	$A(2,1)\ B(1,0)$	$A(2,2)\ B(2,0)$

[B 再上旋一行]

$A(0,0)\ B(0,2)$	$A(0,1)\ B(1,2)$	$A(0,2)\ B(2,2)$
$A(1,0)\ B(0,0)$	$A(1,1)\ B(1,0)$	$A(1,2)\ B(2,0)$
$A(2,0)\ B(0,1)$	$A(2,1)\ B(1,1)$	$A(2,2)\ B(2,1)$

图 1　3×3 处理机上的网格矩阵乘

这样,对于一固定的 $C(i,j)$,其各加数 $A(i,l)\ B(l,j)$ 分处于第 i 行的各处理机上。而在某一处理机 (i,j) 上,$k_s - 1$ 步之后其所有子积矩阵块为:

$$\begin{vmatrix} A(i,j)\ B(j,0) \\ A(i,j)\ B(j,1) \\ A(i,j)\ B(j,2) \\ \vdots \\ A(i,j)\ B(j,k_s-1) \end{vmatrix}$$

显然,在 i 行处理机上,有

$$\begin{vmatrix} A(i,0)\ B(0,0) & A(i,0)\ B(0,1) & & A(i,0)\ B(0,k_s-1) \\ A(i,1)\ B(1,0) & A(i,1)\ B(1,1) & & A(i,1)\ B(1,k_s-1) \\ A(i,2)\ B(2,1) & A(i,2)\ B(2,1) & \cdots & A(i,2)\ B(2,k_s-1) \\ \vdots & \vdots & & \vdots \\ A(i,k_s-1)\ B(k_s-1,0) & A(i,k_s-1)\ B(k_s-1,1) & \cdots & A(i,k_s-1)\ B(k_s-1,k_s-1) \end{vmatrix}^T$$

$$= \begin{vmatrix} A(i,0)\ B(0,0) & A(i,1)\ B(1,0) & & A(i,k_s-1)\ B(k_s-1,0) \\ A(i,0)\ B(0,1) & A(i,1)\ B(1,1) & & A(i,k_s-1)\ B(k_s-1,1) \\ A(i,0)\ B(0,2) & A(i,1)\ B(1,2) & \cdots & A(i,k_s-1)\ B(k_s-1,2) \\ \vdots & \vdots & & \vdots \\ A(i,0)\ B(0,k_s-1) & A(i,1)\ B(1,k_s-1) & \cdots & A(i,k_s-1)\ B(k_s-1,k_s-1) \end{vmatrix} \tag{7}$$

于是,通过行处理机间以子积为元素矩阵的置换(8),并在各处理机上做累加,即可得到 C 在各处理机上的子矩阵 $C(i,j)$,从而最终得到 C。

$$\hat{G} = \left| I_{k_s} \otimes P(k_s^2, k_s) \otimes I_{N_1 N_3 / k} \right| \left| P(k, k_s) \otimes I_{N_1 N_3 / k_s} \right| \tag{8}$$

式(8)中,$\hat{G}$ 表示矩阵置换操作。

下面讨论一下此算法的计算开销和通讯开销。

从上面描述可以看到,计算主要为 k_s 次子矩阵乘法,子矩阵规模为 $\boldsymbol{A}:(N_1/k_s, N_2/k_s)$ 和 $\boldsymbol{B}:(N_2/k_s, N_3/k_s)$,通讯主要存在于计算子积时矩阵 $\boldsymbol{B}$ 的上旋及子积矩阵的置换。$\boldsymbol{B}$ 的上旋共需进行 (k_s-1) 次,每一次通讯块长度为 $N_2 N_3/k$,通讯开销为:

$$(k_s-1)\ t_s + (k_s-1)\ N_2 N_3 t_e/k$$

在做行间矩阵置换时,每个处理机须与同行 k_s-1 个处理机各做一次数据交换,这同样可用一个 MPI 的全互换实现,其通讯开销为

$$t_s + (k_s-1)\ N_1 N_3 t_e/k$$

所以总的通讯开销

$$t_{mesh} = k_s t_s + (k_s-1)(N_1+N_2) N_3 t_e/k \tag{9}$$

将 t_{mesh} 与式(4)的 t_{c-r} 相减,有

$$\begin{aligned} t_{c-r} - t_{mesh} &= (1-k_s) t_s + [(k-1) N_1 - (k_s-1)(N_1+N_2)] N_3 t_e/k \\ &= (1-k_s) t_s + [(k_s^2-1) N_1 - (k_s-1)(N_1+N_2)] N_3 t_e/k_s^2 \end{aligned} \tag{10}$$

当 $N_1 = N_2 = N_3 = N$,即 $\boldsymbol{A}$、$\boldsymbol{B}$ 为方阵时,

$$t_{c-r} - t_{mesh} = (1-k_s) t_s + (k_s-1)^2 N^2 t_e/k_s^2 \tag{11}$$

这时若

$$N^2 \geqslant k_s^2/(k_s-1) \times t_s/t_e \tag{12}$$

则

$$t_{c-r} - t_{mesh} \geqslant 0$$

式(12)说明,对于方阵而言,当计算机条件一定时(这时 t_s/t_e 为常数),网格并行计算在矩阵规模 N 较大时比列-行并行计算有效。将式(12)改写为如下形式:

$$k_s - 1 \geqslant k_s^2/N^2 \times t_s/t_e \tag{13}$$

式(13)说明,在计算机条件一定且单处理机上矩阵元数个数一定时,矩阵规模越大,则网格并行计算的优势越大。

事实上,当矩阵不为方阵时,上述结论仍然成立,这里不再做一般性的推导。应该说明的是,上面所有的讨论都是以矩阵阶数可被处理机数整除为前提的,当矩阵不能被处理机数整除时,只要矩阵在数据分配时保证最优负载平衡,结论仍然可成立。

3 数值结果

数值实验是在 IBM-SP 计算机上用 Fortran90 + MPI 实现的[3],考虑到实际应用中矩阵计算多为方阵,因此程序设计上仅针对方阵进行了测试。实验用 IBM-SP 计算机共含有 10 个计算节点,每个节点含 8 个 CPU,节点内为共享内存,节点间为分布式内存。计算机上提供并行计算环境 poe 和并行软件工具 MPI。

表 1 给出了矩阵乘运算在 IBM-SP 计算机上的几个测试结果。这里应该说明的是,测试结果与所用矩阵乘的串行算法关系密切。经过大量试算,我们选取了 IBM 公司工程与科学计算库提供的矩阵乘库函数作为基本的矩阵乘串行计算程序。

分析表 1,可以发现:矩阵乘的网格划分算法比列-行划分算法时间上有所改善,与列-

行算法相比 ,网格划分算法平均改善了 6 %左右 。

表 1　不同大小矩阵的测试

矩阵规模	列-行(C-R)(s)	网格(MESH)(s)
10000 × 10000	175 .7	174 .9
11264 × 11264	263 .5	245 .7
12000 × 12000	303 .9	297 .6
13312 × 13312	459 .2	390 .4
14000 × 14000	520 .6	480 .3

采用 16 个 CPU ,分布于 8 个节点 ,每个节点 2 个 CPU 。

表 2 是为了将矩阵列-行算法和网格算法与串行计算比较而做的测试 。由于串行计算规模受到机器内存的限制 ,我们选取矩阵大小为 9216 × 9216 。从表中可见 ,矩阵网格并行的加速比较矩阵列-行并行改善约 10 % 。

表 2　9216 × 9216 在不同个数 CPU 运行时的时间比较

CPU 数	C-R		MESH		串行(s)
	时间(s)	加速比	时间(s)	加速比	
4	551 .2	3 .82	511 .7	4 .11	2104 .9
16	159 .5	13 .20	130 .1	16 .20	

4　结　论

矩阵计算是数值预报系统经常遇到的一个计算问题[4,5] ,数值预报系统计算中许多问题的求解最终归结为对矩阵的计算 。随着计算技术的不断发展[6] ,开展矩阵计算的研究对数值预报计算程序的高效实现是有意义的 。

通过上面的讨论 ,我们认为 :对于矩阵乘并行计算而言 ,在计算机条件一定时 ,随着矩阵规模的增大 ,采用网格并行计算不但能更好地与实际应用中的网格划分接口 ,而且能够进一步减少通讯开销 ,因而更具优势 ,是一种有效的矩阵乘计算方法 。

由于目前所具有计算机条件的限制 ,没有能够对矩阵乘计算的一般性情况进行数值试验 。今后一旦条件许可 ,我们将对矩阵计算做进一步深入的研究 。

致　谢 : 感谢国防科技大学李晓梅教授 、宋君强教授和中国气象科学研究院金之雁研究员对论文作者的指导 ,他们与作者的讨论使作者受益匪浅 。

参 考 文 献

1　Strassen V . Gaussian Elimination is Not Optinal . *Numerical Mathematics* ,1969 ,**13** :354 ~ 356 .

2　Barry Wilkinson , Michael Allen 著 . 陆鑫达译 . 并行程序设计 . 北京 :机械工业出版社 ,2002 .

3　都志辉著 . 高性能计算并行编程技术 —— MPI 并行程序设计 . 北京 :清华大学出版社 ,2001 .

4　李晓梅 ,蒋增荣著 . 并行算法 . 长沙 :湖南科学技术出版社 ,1992 .

5　施妙根 ,顾丽珍编著 . 科学和工程计算基础 . 北京 :清华大学出版社 ,1999 .

6　金之雁 ,王鼎兴 . 大规模数据并行问题的可扩展性分析 . 应用气象学报 , 2003 ,**14**(3) :369 ~ 374 .

IMPLEMENTATION OF MATRICES-MULTIPLICATION ON SUPERCOMPUTER

Wu Xiangjun Huang Liping
(*Chinese Academy of Meteorological Sciences, Beijing* 100081)

Abstract

The matrices multiplication is often used in NWP. On distributed systems, such as IBM-SP, the multiplication of two matrices requires data transpose and the efficient data communication are crucial to its performance. Two parallel algorithms are presented, one is based on column-row decomposition and another is based on mesh partition, and the implementation and communication-time of this two different methods are discussed. Results on IBM-SP show that the communication in mesh algorithm are less and the improvement on speedup is up to 10%.

Key words: NWP Matrices-multiplication Parallel-computing Distributed system Data communication