Software Engineering and Applications 软件工程与应用, 2013, 2, 15-19
http://dx.doi.org/10.12677/sea.2013.21003 Published Online February 2013 (http://www.hanspub.org/journal/sea.html)

Performance Analysis of Large Scale Parallel Matrix Multiplication

Zhi Shang[1], Shuo Chen[2]

[1]Institute of High Performance Computing, Agency for Science, Technology and Research, Singapore
[2]School of Aerospace Engineering and Applied Mechanics, Tongji University, Shanghai
Email: shangz@ihpc.a-star.edu.sg, shangzhi@tsinghua.org.cn, schen_tju@mail.tongji.edu.cn

Received: Nov. 17th, 2012; revised: Dec. 3rd, 2012; accepted: Dec. 9th, 2012

Abstract: The large scale computing is difficult to be avoided following the requirement of modern scientific researches and practical engineering applications. The computing and processing of massive data have to be involved in these large scale computing. The parallel computing therefore is employed to solve these issues of large scale computing not only on fast computing but also on data processing. MPI-based parallel computing can easily realize distributed computing and massive data scattered in the cluster supercomputer, making each a single processor to handle a small portion of data in order to achieve fast computing and large scale computing. Based on MPI parallel programming a large-scale matrix multiplication operation was developed. Through the testes on the parallel performance of the point to point communications with the blocking communication, non-blocking communication and mixed communication, a complete set of rapid communication to prevent the occurrence of deadlock was established. The results were significant for the future practical applications.

Keywords: Large Scale Computing; Massive Data; Parallel Computing; Distributed Computing; Matrix Multiplication

大型矩阵相乘并行计算的特性分析

尚　智[1], 陈　硕[2]

[1]新加坡科技研究局高性能计算研究院，新加坡
[2]同济大学航空航天与力学学院，上海
Email: shangz@ihpc.a-star.edu.sg, shangzhi@tsinghua.org.cn, schen_tju@mail.tongji.edu.cn

收稿日期：2012 年 11 月 17 日；修回日期：2012 年 12 月 3 日；录用日期：2012 年 12 月 9 日

摘　要：随着科学研究和工程计算的发展，大规模计算和模拟已经无法避免。这些大规模计算往往涉及海量数据的运算和处理，因此并行计算被用来一方面解决大规模的快速计算，另一方面解决海量数据的处理。基于 MPI 的并行计算可以方便地进行分布式运算，把海量数据分散在集群超级计算机上，使得每单个处理器(CPU)处理一小部分数据，从而实现快速运算和大规模计算。本文基于 MPI 的并行编程，实现了大规模矩阵的相乘运算，并且测试了点对点通信下的不通信机制(阻塞通信、非阻塞通信及其混合通信)的标准通信的并行性能。针对大型矩阵相乘计算，组建了完整的快速标准通信方法，并且防止死锁的发生。为今后的进一步实际应用奠定基础，提供有用的参考。

关键词：大规模计算；海量数据；并行计算；分布式运算；矩阵相乘

1. 引言

矩阵在科学计算中有着广泛地应用，许多科学计算问题最后实际上都归结为对矩阵的操作和运算上。比如在多重信号分类(multiple signal classification,

MUSIC)算法中，王飞[1]等用一种适于并行化的特征分解方法——Jacobi 迭代法，实现了 MUSIC 算法的高效化。位寅生[2]等也提出一种并行化处理方案来高效化 MUSIC 算法，该方案实现了特征分解和谱峰搜索阶段的并行化处理。这些并行化处理过程中都要涉及到矩阵的并行计算。

在大量的实际应用中都会遇到各种各样要求高效矩阵相乘计算的实例。比如在数值预报系统中，就会经常遇到矩阵相乘的计算。提高矩阵乘法的运算性能对于科学计算有着重大的意义[3]。在影响矩阵相乘的诸多因素中，伍湘君和黄丽萍[4]研究发现，矩阵相乘的网格划分算法的并行效率要高于列－行划分算法。但是伍湘君和黄丽萍[4]没有研究，矩阵相乘的网格划分算法的影响因素。

本文的研究正是基于这个考虑，预着重研究影响矩阵相乘的网格划分算法的因素。通过 MPI 编程，作者发现多 CPU 之间 MPI 数据通信的组合繁多，包括有阻塞的接受与传送以及非阻塞的接受与传送。那么这些 MPI 数据通信会不会影响矩阵相乘的网格划分算法呢？如果有影响，那么影响的程度有多大？

根据唐俊奇[5]的研究，随着矩阵规模的增大和 CPU 数量的增多，数据通信所花费的时间比例越来越大，以至于大部分 CPU 时间都要花在数据通信上，甚至于超过计算(加法和乘法)所花费的时间。可是唐俊奇[5]的研究没有给出不通的通信机制对于大型矩阵相乘运算的具体影响。

本文将基于 Cannon 算法[6]，研究不同的 MPI 数据通信组合对于大型矩阵相乘并行计算的影响。Cannon 算法是一种高效的网格划分并行算法。虽然程序实现起来比较复杂，但是算法中多 CPU 之间 MPI 数据通信又是该算法的关键所在。所以比较适合用来研究 MPI 数据通信的组合对并行矩阵相乘的影响。

在本文的研究中，根据目前 MPI 数据通信机制(纯阻塞通信、纯非阻塞通信、以及混合通信)，编制具体的并行程序。程序中调用这些通信机制，通过对一系列的规模矩阵相乘的耗时统计，来比较这些通信机制的表现，从而推荐出比较合适的通信机制、或者配对调用(混合通信)来适应大规模矩阵相乘运算。

2. Cannon 矩阵乘法

矩阵乘积($\boldsymbol{A} \times \boldsymbol{B} = \boldsymbol{C}$)在实际工程领域中是经常要用到的。虽然许多高效的串行程序库可以进行矩阵乘积运算[7-9]。但是单机串行运算总会遇到内存和速度的瓶颈，有必要在并行计算环境上实现矩阵乘积。因此研究并行算法是非常必要的[10]。高效的并行算法不但可以解决运算速度问题，而且可以处理大规模的数据处理。作者尝试了用串行矩阵相乘的程序在内存 2 GB 的单机上运行两个 2048 × 2048 的随机方阵相乘，运行可以进行。但是当矩阵容量增加到 4096 × 4096 时，由于内存容量问题，程序无法运行。

在串行程序中，要同时开出三个矩阵(***A***[N][N]、***B***[N][N]和 ***C***[N][N])，那么所需要的存储量就是 $3 \times N^2$ 个双精度浮点数。随着 N 值的增很大，所需内存量将随着指数增加。如果 N 值增加一倍，所需的内存量将增加 12 倍。这样的要求是很难通过增加单机的内存容量来满足的。基于集群的超级计算机可以把单机的管理内存通过并行计算程序综合起来加以应用，这样就扩展了内存的容量，随着集群单机的数量增加，内存的供应也可以实现指数增加。另外，对于单机运算很吃力的运算，也可以运用并行计算实现加速计，快速完成计算。不过这要求有高效的并行算法和程序的支持才能得以实现[10]。

因此，具有很好的负载平衡的高效 Cannon 算法将被选用来进行本文的并行计算特性分析[10]。

Cannon 算法在计算前，先将二维矩阵 ***A***、***B*** 和 ***C*** 分别分块存储在二维笛卡儿并行处理器上，每个处理器存储 $3\frac{N \times N}{P}$ 个数据，如果需要缓存区来处理数据的交换，还要多 $\frac{N \times N}{P}$ 个数据。公式中的 P 是处理器(CPU)的个数，等于 X 和 Y 方向处理器(CPU)分配个数的乘积，即：$P = P_x \times P_y$。图 1 显示了矩阵 ***A*** 在 4 × 4 二维笛卡儿并行处理器上分布。

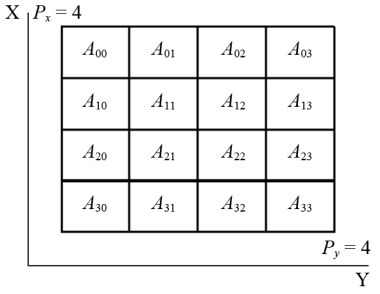

Figure 1. 4 × 4 element distributions of matrix *A*
图 1. 矩阵 *A* 的 4 × 4 分布

计算步骤如下：

1) 串行的方法计算各块(P_x，P_y)上的分块矩阵的乘法，即：$C_{PxPy} = A_{PxPy} \times B_{PxPy}$；

2) 对矩阵 $\boldsymbol{A}$ 进行各块的左右转移，即把 $\boldsymbol{A}_{00}$ 转移给 $\boldsymbol{A}_{01}$，把 $\boldsymbol{A}_{01}$ 转移给 $\boldsymbol{A}_{02}$，把 $\boldsymbol{A}_{02}$ 转移给 $\boldsymbol{A}_{03}$，$\boldsymbol{A}_{03}$ 转移给 $\boldsymbol{A}_{00}$。以此类推，用并行通信完成 $\boldsymbol{A}_{10}$ 行，$\boldsymbol{A}_{20}$ 行和 $\boldsymbol{A}_{30}$ 行的转移；

3) 用同样轮换方法，完成对矩阵 $\boldsymbol{B}$ 进行各块的上下转移；

4) 重复方法 1~3，直到 P_x 或者 P_y 等于 4。

由于计算量对每个处理机来说是相同的，因此在选择算法时只需考虑通信量。迟学斌[10]给出了详细的 Cannon 算法说明和计算量的统计，有兴趣的读者可以参阅这篇文献。

3. MPI 并行编程

MPI 是通过并行应用程序在各并行任务之间通过相互通信来实现并行处理的协调和同步的，它的发展很快、使用面也广。当前很多大型工程应用的软件包都是基于 MPI 的编程实现并行计算的。比如大型综合工程计算软件包 Code_Saturne[11]，它集燃烧(天然气，煤，重油等)，磁流体动力学，不可压缩流，可压缩流等物理计算于一身，可以在支持 MPI 的并行分布式内存机器上运行，被法国 EDF 集团用来研发核反应堆。

MPI 的可移植性，使得 MPI 在几乎所有的集群超级计算机都可以实现。因此，本文的并行编程也是基于 MPI 的编程原理。在上述的 Cannon 计算原理上，一个利用 MPI 的并行矩阵相乘的程序得以编制，这个并行程序的计算机语言采用 C 语言。例如下面是程序的片段，主要是 Cannon 算法的实现部分。

```
for(shift=0;shift<Px;shift++)
{
// Matrix multiplication
  for(i=0;i<Nl;i++)
    for(k=0;k<Nl;k++)
      for(j=0;j<Nl;j++)
        C[i*Nl][j]+=A[i*Nl][k]*B[k*Nl][j];
  if(shift==Px-1) break;
  // Communication for A matrix
  MPI_Irecv(buf, Nl*Nl, MPI_DOUBLE, left, 1, cannon_comm, &req1);
  MPI_Isend(A, Nl*Nl, MPI_DOUBLE, right, 1, cannon_comm, &req2);
  MPI_Wait(&req1, &status);
  MPI_Wait(&req2, &status);
  tmp=buf; buf=A; A=tmp;
  // Communication for B matrix
  MPI_Irecv(buf, Nl*Nl, MPI_DOUBLE, up, 2, cannon_comm, &req3);
  MPI_Isend(B, Nl*Nl, MPI_DOUBLE, down, 2, cannon_comm, &req4);
  MPI_Wait(&req3, &status);
  MPI_Wait(&req4, &status);
  tmp=buf; buf=B; B=tmp;
}
```

在程序中 Nl 是分配在每个处理器上的子矩阵的维数大小，即 N/Px(或者 N/Py)。程序中 MPI 的数据传递使用的是点对点通信的非阻塞标准通信模式，通信顺序是先进行矩阵 $\boldsymbol{A}$ 的转移，然后进行矩阵 $\boldsymbol{B}$ 的转移，转移是通过数据缓存 buf 实现的。

4. 并行高效能性能分析

MPI 在进行点对点通信时，可以通过多个函数调用来实现。这些函数根据实施机制可以分为纯阻塞通信、纯非阻塞通信，以及他们的混合调用下的混合通信。纯阻塞通信的调用函数有三个：MPI_Send 和 MPI_Recv、MPI_Sendrecv 以及 MPI_Sendrecv_replace。其中 MPI_Send 和 MPI_Recv 是单个发送和单个接收，使用时要把他们配对调用。也就是一个处理器调用一个发送，就必须在另一个处理器调用一个相应的接收，来完成完整的发送调用过程。在使用配对的发送调用通信时，还必须注意通信死锁的发生。本文使用奇数处理器发送、偶数处理器接收；以及奇数处理器接收、偶数处理器发送的双向配对方法来避免通信死锁的发生。通过实际运算，证实这个方法是可靠的。MPI_Sendrecv 调用，直接由函数的调用来自身实现发送和接收的配对，这个函数的调用可以自动防止通信死锁的发生。MPI_Sendrecv_replace 的调用跟 MPI_Sendrecv 的调用一样，一个函数自动完成发送和接收的配对，同时完成传送数组的自动更新，不需要开出

额外的 buf 数组来完成数据更新。

MPI_Sendrecv 和 MPI_Sendrecv_replace 的调用虽然简单，它们在运行时需要 MPI 额外多做许多工作，这些额外的工作是否会影响程序运行的速度，则需要通过测试才能确定。一般测试运算速度的快慢可以用每秒中完成浮点数运算的次数(FLOP/s)的性能来衡量，这也是一般通行的方法[7]。在本文中为了方便，定义下面的公式来计算浮点数运算的次数。

$$Perf = \frac{Nl \times Nl \times Nl \times P}{t} \quad (1)$$

式中：*Perf* 是并行计算性能；*Nl* 是数组在单个处理器中的维数大小；*P* 是 MPI 调用处理器的总数；*t* 是并行程序运行的时间。从这个公式可以看到，这里的浮点数运算一次实际上包含一个加法和一个乘法。

表 1 展示了这三种纯阻塞通信的并行性能比较。从表 1 的比较，可以看出当数据量小的时候(4096 × 4096)，MPI_Send 和 MPI_Recv 同 MPI_Sendrecv 具有相同的并行性能，MPI_Sendrecv_replace 的并行性能最差，几乎是 MPI_Send 和 MPI_Recv 或 MPI_Sendrecv 的一半。随着数据量的增加，可以看到当数据量中等(16,384 × 16,384)时 MPI_Sendrecv_replace 的并行性能有所增加，但是仍然比 MPI_Send 和 MPI_Recv 或 MPI_Sendrecv 低，这种趋势一直保持到大数据量(32,768 × 32,768)。在大数据量情况下，MPI_Send 和 MPI_Recv 的并行性能最好。MPI_Sendrecv 和 MPI_Sendrecv_replace 的并行性能比 MPI_Send 和 MPI_Recv 差，主要是因为在 MPI 调用 MPI_Sendrecv 和 MPI_Sendrecv_replace 时，MPI 需要做一些额外的工作，比如判断和避免通信死锁的发生，以及额外的动态缓存实现数据转移等。所有这些额外的工作都是需要消耗并行计算时间的。

纯非阻塞点对点通信调用的函数只有 MPI_Isend 和 MPI_Irecv 单个发送和单个接收的配对，使用时跟单个阻塞发送和单个阻塞接收一样要配对调用。不同的是他们不需要考虑通信死锁的问题，只是在调用后要再调用各自的请求等待(MPI_Wait)，等待处理器来处理这个发送或者接收。非阻塞调用的最大优点就是当计算工作量很大时，各处理器可以不用等待运算的同步去处理发送和接收，而是先各自处理自己的运算，等运算结束再处理数据传送。在大型机算时，这种处理方式能保证程序的运算高效地执行。单个的非阻塞通信还可以跟单个的阻塞通信组合出混合通信。

为了比较各种非阻塞通信性能，表 2 进行了纯非阻塞和混合非阻塞通信，在不同的矩阵大小下的并行性能的比较。从比较中可以看到，MPI_Isend 和 MPI_Recv 式中都保持着较高的并行性能。甚至在相同的数据量下，其并行性能高于纯阻塞通信 MPI_Send 和 MPI_Recv 的组合。在具体编程时，跟 MPI_Send 和 MPI_Recv 的组合编程类似要采用奇数处理器发送、偶数处理器接收；以及奇数处理器接收、偶数处理器发送的双向配对方法来避免通信死锁的发生。

从阻塞和非阻塞通信比较，可以看出纯阻塞通信 MPI_Send 和 MPI_Recv 的组合和混合非阻塞通信 MPI_Isend 和 MPI_Recv 的组合具备高并行性能，尤其是在大数据量的情况下。为了进一步测试他们的性能，图 2 示出了它们的在大数据量(32,768 × 32,768)情况下的加速性能比。这个加速比是以串行计算下的 CPU 时间做为参考时间，除以其它在并行多 CPU 下并行计算时间所得的结果。由于多 CPU 下并行计算会比串行计算快，所以这个比值往往大于 1。比值越大表明并行计算速度越快。理想情况下，用多少个 CPU，速度就会加快多少倍。实际上由于计算机的内存分配、数据总线的传输延迟等原因，使得实际的加速比要比理想值低。但是越接近理想值，说明算法的加速效果越好。

Table 1. Performance comparison of parallel computing under different blocking communication mechanism
表 1. 不同阻塞通信机制下并行计算性能的比较

矩阵大小	通信机制		
	MPI_Send 和 MPI_Recv	MPI_Sendrecv	MPI_Sendrecv_replace
4096 × 4096	0.5497 GFlop/s	0.5468 GFlop/s	0.3480 GFlop/s
16,384 × 16,384	0.6078 GFlop/s	0.6651 GFlop/s	0.5902 GFlop/s
32,768 × 32,768	1.1681 GFlop/s	1.1642 GFlop/s	1.1191 GFlop/s

Table 2. Performance comparison of parallel computing under different non-blocking communication mechanism
表 2. 不同非阻塞通信机制下并行计算性能的比较

矩阵大小	通信机制		
	MPI_Isend 和 MPI_Irecv	MPI_Send 和 MPI_Irecv	MPI_Isend 和 MPI_Recv
4096 × 4096	0.3496 GFlop/s	0.3483 GFlop/s	0.5506 GFlop/s
16,384 × 16,384	0.6212 GFlop/s	0.6211 GFlop/s	0.6306 GFlop/s
32,768 × 32,768	1.1606 GFlop/s	1.1525 GFlop/s	1.1876 GFlop/s

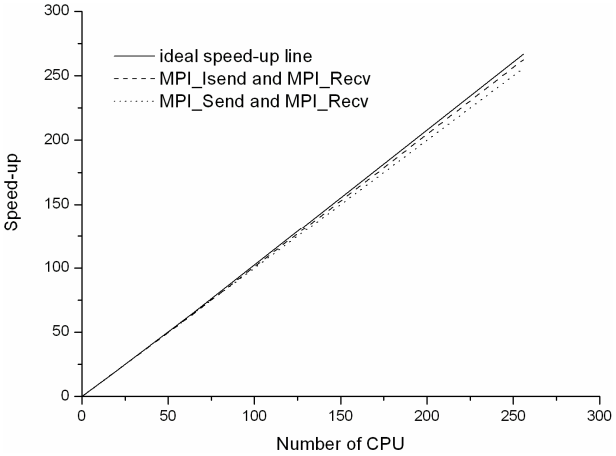

Figure 2. Curve of speed-up
图 2. 加速比曲线对比

另外，需要加以说明的是，本文的研究只限于不同通信机制对于矩阵相乘计算的研究。这个研究是基于所有的 CPU 都工作良好的情况下，不涉及当某个 CPU 失效时的应急处理情况，所以在编程中不涉及这种应急处理的编制。

5. 结论

从并行性能的测试可以看出，总体上阻塞通信和非阻塞通信的并行性能都随着数据量的加大，并行性能也增加，这点很适合于海量数据的运算和处理。从加速比可以看出，两种通信机制都能具备良好的并行加速性能，均能以线性加速比上升。

单纯的阻塞通信(MPI_Send 和 MPI_Recv 的组合)和混合非阻塞通信(MPI_Isend 和 MPI_Recv 的组合)均可以实现较高的并行性能，混合非阻塞通信的并行性能要高于单纯的阻塞通信。尤其是在大数据量的情况下，混合非阻塞通信的并行性能高于单纯的阻塞通信。这些测试结果对于今后的海量数据矩阵相乘情况下，MPI 通信的函数的合理选择具有重要意义。

从测试结果可以得出在大规模矩阵相乘并行运算时，混合非阻塞通信 MPI_Isend 和 MPI_Recv 的组合是值得首选的，其次是单纯的阻塞通信 MPI_Send 和 MPI_Recv 的组合。继而根据此项研究结果，可以推论在海量数据运算和处理时，也可以参考使用这个结果。

6. 致谢

感谢国家自然科学基金(批准号：51276130)和中央高校基本科研业务费专项资金(The Fundamental Research Funds for the Central Universities)的资助。

参考文献 (References)

[1] 王飞, 王建业, 张安堂. 实对称矩阵特征值分解高速并行算法的 FPGA 实现[J]. 空军工程大学(自然科学版), 2008, 9(6): 67-70.

[2] 位寅生, 谭久彬, 郭荣. MUSIC 空间谱估计并行运算算法[J]. 系统工程与电子技术, 2012, 34(1): 12-16.

[3] 程豪, 张云泉, 张先轶, 李玉成. CPU-GPU 并行矩阵乘法的实现与性能分析[J]. 软件技术与数据库, 2010, 36(13): 24-26.

[4] 伍湘君, 黄丽萍. 超级计算机上矩阵乘的并行计算与实现[J]. 应用气象学报, 2005, 16(1): 122-127.

[5] 唐俊奇. 多处理机中矩阵乘法的算法研究[J]. 中国西部科技, 2007, 2: 4-8.

[6] L. E. Cannon. A cellular computer to implement the Kalman filter algorithm. Montana State University, 1969.

[7] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington and R. C. Whaley. An updated set of basic linear algebra subprograms (BLAS). ACM Transactions on Mathematical Software, 2002, 28(2): 135-151.

[8] S. Robinson. Toward an optimal algorithm for matrix multiplication. SIAM News, 2005, http://www.siam.org/pdf/news/174.pdf

[9] H. Cohn, R. Kleinberg, B. Szegedy and C. Umans. Group-theoretic algorithms for matrix multiplication. Proceedings of the 46th Annual Symposium on Foundations of Computer Science, 23-25 October 2005.

[10] 迟学斌. 高性能并行计算[M]. 北京: 中国科学院计算机网络信息中心, 2005: 31-36.

[11] Y. Fournier, J. Bonelle, C. Moulinec, Z. Shang, A. G. Sunderland and J. C. Uribe. Optimizing Code_Saturne computations on Petascale systems. Computers & Fluids, 2011, 45: 103-108.

