文章编号:1001 - 2486(2006)04 - 0068 - 05

利用混合编程改善 SMP 机群上并行矩阵乘法的性能*

吴建平[1],王正华[1],李晓梅[2]

(1. 国防科技大学 计算机学院,湖南 长沙 410073; 2. 装备指挥技术学院,北京 101416)

摘 要:针对 SMP 机群,探讨了分别利用单机优化、OpenMP 与 MPI 从指令级、共享存储级与分布主存级三个层次上改善矩阵并行乘 Fox 算法性能的方法。并通过调用数学函数库与混合编程的方式,在深腾 6800 上进行了实验,取得了相当满意的数值效果。

关键词:SMP 机群;OpenMP;MPI;混合编程;Fox 算法

中图分类号:TP301 文献标识码:A

Improve the Performance of Parallel Matrix Multiplication on Clustered SMP Systems Through Hybrid Programming

WU Jian-ping[1], WANG Zheng-hua[1], LI Xiao-mei[2]

(1. College of Computer, National Univ. of Defense Technology, Changsha 410073, China;
2. Academy of Command and Technology of Equipment, Beijing 101416, China)

Abstract: In order to give a full play of SMP clusters, this research studied the method to improve the performance of Fox algorithm for parallel matrix multiplication exploiting optimization on single processor, OpenMP and MPI, which involves the levels of instruction, shared memory and distributed memory respectively. Through invoking mathematic library and hybrid programming, the numerical results derived on DeepComp 6800 are satisfactory.

Key words: SMP cluster; OpenMP; MPI; hybrid programming; Fox algorithm

高性能计算机的运算速度正从多个层面得到提高。其一,由于制造工艺与存储结构的改善,以及多指令流与流水线等技术的引入,使得单处理机的运算速度越来越快。其二,将多个处理器单元进行集成,共用主存储器,形成性能很高的 SMP 超结点。其三,为形成具有超大规模计算能力的超级计算机,同时克服共享存储机器建造代价高的缺点,将许多个超节点用价格相对低廉的网络相连接,形成 SMP 机群系统。

要想充分发掘 SMP 机群的性能,需要从各个层面上改善算法的性能。在单机层面,需要采用优化技术,对许多数学问题如线性代数,一般已有专门的优化过的函数库,这种情况下应该尽量调用这些库。

在共享存储级,采用 OpenMP 是最好的并行化方法。其一,OpenMP 在相对更细粒度的并行性开发中具有明显优势,从而可扩展性将会比较好。其二,OpenMP 以指导语句的形式指导循环的并行计算,相对而言只需要在原串行程序的适当的循环外加上指导语句即可,易于使用。其三,OpenMP 中不需要显式通信,没有 MPI 程序中那样的消息传递开销。但采用 OpenMP 时,也需要注意几个问题。一是 OpenMP 只能用于共享主存机器上,虽然现在很多分布主存的机器上也提供统一的虚拟内存,同时可以采用 OpenMP 进行并行化,但可以预计其效率相对会比较低。二是在程序中不能有太多的指导语句,否则与指导语句相关的开销将比较大。三是在有大量访存冲突,特别是写冲突的情形下,效率必然低下。

在分布主存级,MPI 是较好的选择。MPI 特别适合于粗粒度并行计算,当粒度较小时,将由于通信开销相对较大而影响整体并行计算效果。在应用 MPI 时,应该尽量采取计算通信重叠、非阻塞通信等减少通信开销的措施。由于在 MPI 模式下,每个任务具有自己独立的存储空间,所以不存在任务间的

* 收稿日期:2006 - 03 - 17
基金项目:国家自然科学重点基金资助项目(69933030)
作者简介:吴建平(1974—),男,副研究员,博士。

访存冲突。

由以上分析可见,在 SMP 机群上,采用 MPI 与 OpenMP 混合编程,是在这类两级结构计算机上开发其并行计算潜力的一种手段,受到国内外的广泛关注[1-4]。本文针对矩阵乘的 Fox 算法,探讨提高整体性能的方法。

1 分布式机器上矩阵乘的 Fox 算法与改进

假设 A 与 B 分别为 $m \times k$ 与 $k \times n$ 矩阵,矩阵乘法是计算 $C = AB$。设将矩阵 A 与 B 分块为 $A = (A_{i,j})_{p \times p}$ 与 $B = (B_{i,j})_{p \times p}$,其中 $A_{i,j}$ 与 $B_{i,j}$ 分别为 $m_i \times k_j$ 与 $k_i \times n_j$ 矩阵。当在分布主存的并行机上进行计算时,为了使得负载平衡,对所有 $i \neq j$,在分块时应尽量使得 $m_i = m_j$、$n_i = n_j$、$k_i = k_j$。在进行分块后,矩阵乘法操作可以分解为

$$C_{i,j} = \sum_{l=0}^{p-1} A_{i,l} B_{l,j} \tag{1}$$

最初的 Fox 算法[5-6]针对结点组织成 $p \times p$ 网格的分布式并行机,1999 年经吴建平与迟学斌推广,得到的并行算法已适用于结点组织成长方网格的分布式并行机[7],并且得到了一个结论,即 Fox 算法的最佳性能在最接近于正方网格的情形下得到。

假设结点组织成 $q \times r$ 网格,取 $p = [q, r]$ 为 q 与 r 的最小公倍数,采用二维块循环方式存放分块矩阵,记 $q' = p/q$,$r' = p/r$,即在结点 $P_{i,j}$($i = 0,1,\cdots,q-1$, $j = 0,1,\cdots,r-1$)上存放

$$\begin{array}{cccc}
A_{i,j} & A_{i,\mathrm{mod}(j+r,p)} & \cdots & A_{i,\mathrm{mod}(j+(r'-1)r,p)} \\
A_{\mathrm{mod}(i+q,p),j} & A_{\mathrm{mod}(i+q,p),\mathrm{mod}(j+r,p)} & \cdots & A_{\mathrm{mod}(i+q,p),\mathrm{mod}(j+(r'-1)r,p)} \\
\vdots & \vdots & \ddots & \vdots \\
A_{\mathrm{mod}(i+(q'-1)q,p),j} & A_{\mathrm{mod}(i+(q'-1)q,p),\mathrm{mod}(j+r,p)} & \cdots & A_{\mathrm{mod}(i+(q'-1)q,p),\mathrm{mod}(j+(r'-1)r,p)}
\end{array}$$

这里仅列出了 A 的块的存储方式,矩阵 B 与 C 与之完全相同,不再赘述。在具体存储时,将所有块进行连续存放,即

$$A'_{u,v} = A_{\mathrm{mod}(i+uq,p),\mathrm{mod}(j+vr,p)} \quad (u = 0,1,\cdots,q'-1, v = 0,1,\cdots,r'-1)$$

对 B 与 C 也类似存放。这样,可以将长方网格上的 Fox 算法具体描述如算法 1 所示。

算法 1 (分布式长方网格上的 Fox 算法[7])

1. 置 $C'_{0:q'-1,0:r'-1} = 0$;
2. For $i = 0$ to $q-1$ do
3. For $j = 0$ to $q'-1$ do
4. 计算 $k = \mathrm{mod}(i + jq + \mathrm{myrow}, p)$;
5. If ($\mathrm{mod}(k,r) = \mathrm{mycol}$) 将 $A'_{0:q'-1,(k-\mathrm{mycol})/r}$ 复制到 W;
6. 以处理机列 $\mathrm{mod}(k,r)$ 为根,在处理机行中广播 W;
7. 更新 $C'_{0:q'-1,0:r'-1} = C'_{0:q'-1,0:r'-1} + WB'_{j,0:r'-1}$;
8. Endfor
9. If ($i \neq q-1$) 在处理机列 mycol 中向上循环移动 B' 一次;
10. Endfor

算法 1 的正确性证明参见文献[7]。仔细考察算法 1,可以发现当 $r = 1$ 时,实际上并不需要广播操作,这样就可以对算法 1 进行改进,如算法 2 所示。

现在来分析算法 2 的通信需求,为方便起见,不妨设 m、n 与 k 都能被 p 整除。不难发现,当 $r = 1$ 时,不需要进行广播,且只需要向邻近结点发送 $q-1$ 次,每次发送 $kn/(qr)$ 个浮点数;同时只需要从邻

近结点接收 $q-1$ 次,每次接收 $kn/(qr)$ 个浮点数。当 $r \neq 1$ 时,还需要广播 p 次,每次广播 $mk/(qp)$ 个浮点数。

2 矩阵乘法在SMP机群上的性能考虑

在第1节中已经看到,当 $m=n=k$ 时,每个处理器上的计算量约 $2n^3/(qr)$,通信量约为 $2(q-1)n^2/(qr)+n^2/q$ 个浮点数,计算通信比相对比较小,所以当处理器数较多时,整体性能必然不高。为提高整体性能,减少通信量至关重要。

在SMP机群上,每个处理机结点由多个共享主存的处理器组成,在结点内部,可以采用共享方式来减少通信量。但为达到整体性能的目的,在共享计算时,不能有太多的访存写冲突,即处理器不能出现太多更改同一存储单元的操作。

算法2 (分布式长方网格上的改进型Fox算法[7])

1. 置 $C'_{0:q'-1,0:r'-1}=0$;
2. For $i=0$ to $q-1$ do
3. For $j=0$ to $q'-1$ do
4. 计算 $k=\mathrm{mod}(i+jq+\mathrm{myrow},p)$;
5. If ($r\neq 1$) then;
6. If ($\mathrm{mod}(k,r)=\mathrm{mycol}$)将 $A'_{0:q'-1,(k-mycol)/r}$ 复制到 W;
7. 以处理机列 $\mathrm{mod}(k,r)$ 为根,在处理机行中广播 W;
8. 更新 $C'_{0:q'-1,0:r'-1}=C'_{0:q'-1,0:r'-1}+WB'_{j,0:r'-1}$;
9. Else;
10. 更新 $C'_{0:q'-1,0:r'-1}=C'_{0:q'-1,0:r'-1}+A'_{0:q'-1,(k-\mathrm{mycol})/r}B'_{j,0:r'-1}$;
11. Endif
12. Endfor
13. If ($i\neq q-1$) 在处理机列 mycol 中向上循环移动 B' 一次;
14. Endfor

算法3 (共享处理机上的矩阵乘并行算法)

```
c $omp parallel do shared(s,n,m,k,α,β,a,lda,b,ldb,c,ldc), private(i,j,l),
c $omp&           default(none), schedule(static), num_threads(s)
      do j=1,n
        do i=1,m
          c_ij = β * c_ij
          do l=1,k
            c_ij = c_ij + α * a_il * b_lj
          Enddo
        Enddo
      Enddo
c $omp end parallel do
```

算法4 (共享处理机上矩阵乘的改进型并行算法)

```
c $omp parallel private(ib,ni,i), shared(s,n,m,k,α,β,a,lda,b,ldb,c,ldc),
c $omp&           default(none), num_threads(s)
c $omp do schedule(static)
      do i = 0, s-1
        call bschedule(n, s, i, ib, ni)
        call dgemm('n','n', m, ni, k, α, a, lda,
```

```
      &                b(1,ib), ldb, β, c(1,ib), ldc)
        enddo
c $omp enddo nowait
c $omp end parallel
```

以进行矩阵乘法 $C = \beta C + \alpha AB$ 为例,利用 OpenMP 可以将并行矩阵乘的计算过程的 Fortran 程序段描述如算法 3 所示。可以看到,在算法 3 中,各处理器上的计算结果将分别存储在不同的存储单元中,不会出现写冲突,这对提高并行计算效率十分有利。但是在算法 3 中,没有利用高效的数学函数库,而矩阵乘在 BLAS 中存在对应的高效子程序 dgemm。为充分发挥计算效率,应该尽量调用该子程序。

显然,在算法 3 中不能直接调用 dgemm。为了调用这个子程序,可以利用手动方法事先静态地对 $j = 1$ 到 n 进行分段,如果有 s 个共享同一主存的处理器,事先确定 n_0、n_1、…、n_{s-1},使得其尽量相等,而不利用 OpenMP 去调度。这样,可将算法 3 改写为算法 4 的形式。在算法 4 中,子程序 bschedule(n, s, i, ib, n_i)用于确定对应于 s 个处理器的共享主存并行机上第 i 个处理器中的 ib 与 n_i 值,这里 $ib = n_0 + n_1 + \cdots + n_{i-1}$。

此外,由于在计算过程中不存在写冲突,所以在并行计算的结尾处也不需要同步,故而在 OpenMP 的指导语句 c $omp enddo 后显式地加上 nowait 子句,以避免不必要的同步。

当在 SMP 机群上进行计算时,将启动逻辑上组织成长方网格的若干个进程,在该网格上用算法 2 进行分布式计算,而每个进程在算法 2 第 6、8 与 10 步中在某个结点内部再启动若干个线程,利用算法 4 进行计算。

3 实验结果与分析

在深腾 6800 网格超性能并行计算机上,对本文中的并行矩阵乘法进行了数值实验。该机包括 265 个结点,每个结点由 4 个共享主存且主频为 1.3GHz 的安腾 2 处理器芯片构成,其中总共有 1024 个处理器芯片用于计算。整机总内存为 2.6TB,磁盘总容量为 80TB。结点间采用 Quadrics 公司的 QsNet 高速连接网络进行通信,点对点通信带宽为 300MB/s,延迟时间小于 7μs。在结点上采用 Linux 作为操作系统,且自带优化编译技术和数学库函数,包括 BLAS 子程序库,这使得实际计算速度高,整机效率高。整机峰值浮点运算速度为每秒 5.324 万亿次,初步 Linpack 测试结果为每秒 4.148 万亿次,在 2003 年 12 月与 2004 年 6 月全球 Top500 中分别排名第 14 位与第 26 位。

由于深腾 6800 上的应用程序十分多,作为普通用户很难申请到更多的计算资源,所以本文只在其中 4 个结点共 16 个处理器芯片上进行数值实验。虽然如此,但从表 1 的数值结果可以看到,这足以反应本文采用的混合编程算法的优越性。在进行实验时,在每个处理器芯片上,调用 BLAS 库中的 dgemm;在进程内部,采用如算法 4 所示的 OpenMP 并行化;在进程间,采用 MPI 作为通信平台。这样,编写的程序整体采用 mpif77 进行编译,并采用参数-openmp 以激发 OpenMP 指导语句,采用参数-O3 进行优化。

在表 1 中列出了当 $m = n = k = 2000$ 时的部分双精度计算结果,所使用的处理器芯片数量以 $q \times r \times s$ 的形式给出,其中 $q \times r$ 是所用的进程数,逻辑上组织成 $q \times r$ 的长方网格,s 为每个进程启动的线程数量。nrecv、nsend 与 nbcst 分别表示每个进程的接收次数、发送次数与广播次数,lrecv、lsend 与 lbcst 表示每次接收、发送与广播的浮点数个数,单位为百万个浮点数。

由于算法 2 相对算法 1 的改进,使得在 $r = 1$ 时,不再需要广播,所以在进行分布式计算时,Fox 算法的性能在正方网格上不一定比在长方网格上高。相反,最佳性能可能在 $r = 1$ 的情况下获得。

从表 1 中可以看出,在同样个数的超节点上计算时,很多情况下可以通过混合编程来减少通信量与通信次数,从而得到了比单独采用 MPI 编程时更高的性能。当然,$4 \times 4 \times 1$ 与 $4 \times 2 \times 2$ 是例外,这是因为此时通信量不但没有减少,反而有所增加,同时增加了与 OpenMP 指导语句相关的开销。由此可见,为得到最佳的性能,应该事先分析,确定最佳的(q, r, s)组合,以最大程度地减少通信次数与通信量。

表1 深腾6800上Fox算法(双精度)的浮点性能(MFLOPS)与加速比

Tab.1 Performance(MFLOPS) and speedup of Fox algorithm on deepcomp 6800

$q \times r \times s$	nrecv lrecv	nsend lsend	nbcst lbcst	浮点性能	加速比
1×1×1	0 0.00	0 0.00	0 0.00	4,660.3	1.00
4×1×1	3 1.00	3 1.00	0 0.00	14,480.9	3.11
2×2×1	1 1.00	1 1.00	2 1.00	13,588.6	2.92
2×1×2	1 2.00	1 2.00	0 0.00	14,786.3	3.17
1×1×4	0 0.00	0 0.00	0 0.00	15,810.4	3.39
4×2×1	3 0.50	3 0.50	4 0.25	22,451.8	4.82
4×1×2	3 1.00	3 1.00	0 0.00	23,309.1	5.00
2×2×2	1 1.00	1 1.00	2 1.00	22,342.9	4.79
2×1×4	1 2.00	1 2.00	0 0.00	25,769.2	5.53
4×4×1	3 0.25	3 0.25	4 0.25	26,407.3	6.95
4×2×2	3 0.50	3 0.50	4 0.25	32,225.3	6.91
4×1×4	3 1.00	3 1.00	0 0.00	35,600.8	7.64
2×2×4	1 1.00	1 1.00	2 1.00	33,467.1	7.18

4 结论

本文通过对分布式Fox算法的改进,以及在SMP机群上从多个层面上对性能的考虑与混合编程,在深腾6800高性能计算机上取得了相当满意的计算结果。通过本文的研究,可以得到以下结论或经验:

(1)在SMP机群上,通过将共享编程与分布式编程结合起来,有潜力得到比单纯分布式编程更高的计算性能。特别是在共享计算时,如果没有存储器访问冲突,则更是如此。进而,混合编程也更有潜力获得更好的可扩展性。

(2)为提高SMP机群上的整体性能,首先要考虑如何有效减少进程间的通信开销,同时必须考虑最小化进程启动线程的开销,也必须考虑如何调用现有的优化好的单机上的高效库函数。

(3)并非混合编程一定比单纯分布式编程优越,谁优谁劣的关键是看在那种方式下的总体开销较小,这里所说的开销包括通信开销、与并行计算相关的额外计算,以及与指导语句相关的各种开销。由于存在这种复杂性,所以要事先判定谁优谁劣并非易事。

致谢

感谢中国科学院高性能计算中心为本研究提供的深腾6800超级计算机实验环境,特别感谢该中心迟学斌研究员与曹宗雁同学提供的帮助。

参考文献:

[1] Smith L, Bull M. Development of Mixed Mode MPI/OpenMP Applications[J]. Scientific Programming, 2001, 9: 83 – 98.

[2] Mahinthakumar G, Saied F. A Hybrid MPI-OpenMP Implementation of an Implicit Finite-Element Code on Parallel Architectures[J]. The International Journal of High Performance Computing Applications, 2002, 16(4): 371 – 393.

[3] Nakajima K. OpenMP/MPI Hybrid vs. Flat MPI on the Earth Simulator: Parallel Iterative Solvers for Finite Element Method[A]. The fifth International Symposium on High Performance Computing[C], Tokyo, Japan, October 20 – 22, 2003.

[4] Rabenseifner R, Wellein G. Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures[A]. Proceedings of the Fourth European Workshop on OpenMP, EWOMP 2002[C], Roma, Italy, Sep. 18 – 20, 2002.

[5] Grama A, Gupta A, Karypis G, et al. Introduction to Parallel Computing(second edition)[M]. The Benjamin/Cummings Publishing Company. Inc., 1994.

[6] 孙家昶, 张林波, 迟学斌, 等. 网络并行计算与分布式编程环境[M]. 北京: 科学出版社, 1996.

[7] 吴建平, 迟学斌. 分布式系统上并行矩阵乘法[J]. 计算数学, 1999, 21(3): 99 – 108.