并行 linpack 分析与优化探讨

张文力 陈明宇 冯圣中 樊建平

（中国科学院计算技术研究所 北京 100080）

(zhangwl@ncic.ac.cn)

摘要：HPL 是大规模集群系统广泛采用的 linpack 测试软件包，本文在深入分析线性代数方程组分块并行求解算法和 HPL 实现技巧的基础上，探讨了 HPL 峰值性能的制约因素。重点讨论了 LU 分解过程中主要参数 P×Q 和 NB 对计算性能的影响。理论分析与实验结果表明，定义为矩阵运算时间与其运算量之比的效率因子很大程度上相关于矩阵的分块大小，而与矩阵规模本身关系微乎其微。根据这一规律，本文作者提出通过扫描小规模矩阵运算效率来确定大规模并行测试中分块大小 NB，改善长期以来只是通过反复实验试探获取 NB 的现状，大大缩短了 NB 的确定过程，为其最终定位提供了相对精确的理论化依据。目前实际测试结果基本验证了本文作者的想法，这一方法同样适用于其他过程中的矩阵并行运算。

关键词：HPL(high performance linpack)，线性代数方程组，LU 分解，MPI

Analysis and Optimization Discussion on Parallel Linpack

Zhang Wenli Chen Mingyu Feng Shengzhong Fan Jianping

（Institute of Computing Technology, CAS.，Beijing，100080）

Abstract: HPL is a linpack benchmark package widely adopted on massive scale cluster system. Based on further analysis of the blocked parallel solution algorithm of linear algebra equations and HPL implementation skills, the restriction factors of the HPL peak performance are probed. The influence of main parameter P * Q and NB on computing performance in LU factorization process is discussed especially. Theory analysis and experiment results indicate that, the efficient factor, which is defined as the quotient of the matrix operation time and the total number of operations, is relevant to the matrix block size to a great extent, yet little correlate to the matrix dimensions. According to this law, the author proposes that to determine the block size NB of massively parallel test through scanning the operation efficiency of small-scale matrix. Thus it has improved the status getting NB through try-and-error experiment test without definite aim, shorten the determination course greatly, and offered a relatively accurate theoretical basis for making a reservation finally. The author's idea has been verified by the actual test results at present, as well as been applicable to the matrix parallel operations of other procedure.

Key words: high performance linpack, linear algebra equations，LU factorization，MPI

1．引言

Linpack 是当前国际上流行的性能测试基准，通过对高性能计算机求解稠密线性代数方程组能力的测试，评价高性能计算机系统的浮点性能，由 Jack Dongarra 在 1979 年首次提出，多为 Fortran 版本。它提供多种程序并在其它函数库的支持下解决线性方程问题，包括求解稠密矩阵运算，带状的线性方程，求解最小平方问题以及其它各种矩阵运算，但它们都是基于高斯消去法的原理。

Linpack 根据问题规模与优化选择的不同分为 100× 100，1000× 1000，n×n 三种测试[1]。HPL[2] (High performance linpack) 是第一个标准的公开版本并行 Linpack 测试软件包，是 n×n 测试的 MPI 实现，可适应多体系移植，目前广泛用于 top500 测试[3]。这一测试主要针对分布式存储大规模并行计算系统而设计，它的要求也是 Linpack 标准中最为宽松的，用户可以对任意大小的问题规模，使用任意个数的 CPU，使用基于高斯消去法的各种优化方法来执行该测试程序，寻求最佳的测试结果。性能测试实际就是要计算浮点运算率。由于高斯分解法求解规模为 n 的线性代数方程问题的浮点运算次数($2n^3/3 + 3n^2/2$)是一定的，而这一浮点运算次数的大小只与问题规模有关，也即与系数矩阵的维数有关，这样只要给出问题规模 n，根据线性方程组求解过程中消元和回代部分的计时 t，就可以定量计算出机器的性能参数——浮点运算次数/秒：

$$(2n^3/3 + 3n^2/2)/t \qquad (1)$$

一般来说，要获得 HPL 实测峰值，需要使用与内存匹配的最大问题规模，也就是大约接近内存总容量的 80%。

本文作者在 HPL 源码详细分析的基础上，权衡重点可调参数，开展相关实验并分析结果，得出一些有用的结论，希望对后续 Linpack 测试优化具有一定的指导意义。本文第 2 部分简要给出高斯消去法原理及 LU 算法并行思想，第 3 部分围绕分析及测试过程中相关问题的思考与尝试展开讨论，尤其对确定 NB 依据的探讨上，并给出测试结果及相应分析。最后归纳总结，展望以后的工作。

2．高斯消去法的原理

线性代数方程组的数值解法分为直接法和迭代法两种。对于系数矩阵稠密的线性代数方程组

$$\mathbf{Ax} = \mathbf{b} \qquad (2)$$

其中，$\mathbf{A}=(a_{ij})_{n\times n}$ 且为非奇异矩阵；$\mathbf{b}=(b_1, b_2, \cdots, b_n)^T$, $\mathbf{x}=(x_1, x_2, \cdots, x_n)^T$, A 与 b 均为已知，而 x 是待求的 n 维向量，其经典的直接解法是 LU 分解法[4, 5, 6, 7]。而高斯消去法是实现 LU 分解的经典算法，按消元过程和回代过程两步建立具体的求解。

也就是，LU 分解将问题(1)化为两个三角形方程组的求解

$$LUx = Pb$$

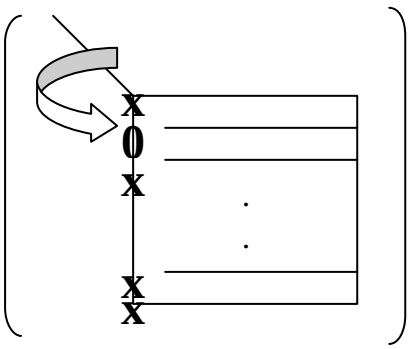
等价于

$$Ly = Pb \qquad (3)$$

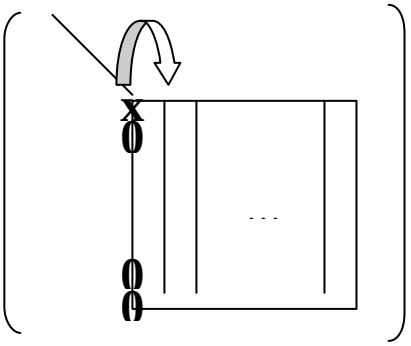
$$Ux = y \qquad (4)$$

其中，P 表示行交换的初等变换矩阵，式（3）在 HPL 中与 LU 分解过程同时操作，$\mathbf{y}=(y_1, y_2, \cdots, y_n)^T$ 是增广矩阵经过消元处理后 b 相应的部分，式（4）则表示回代求解过程。

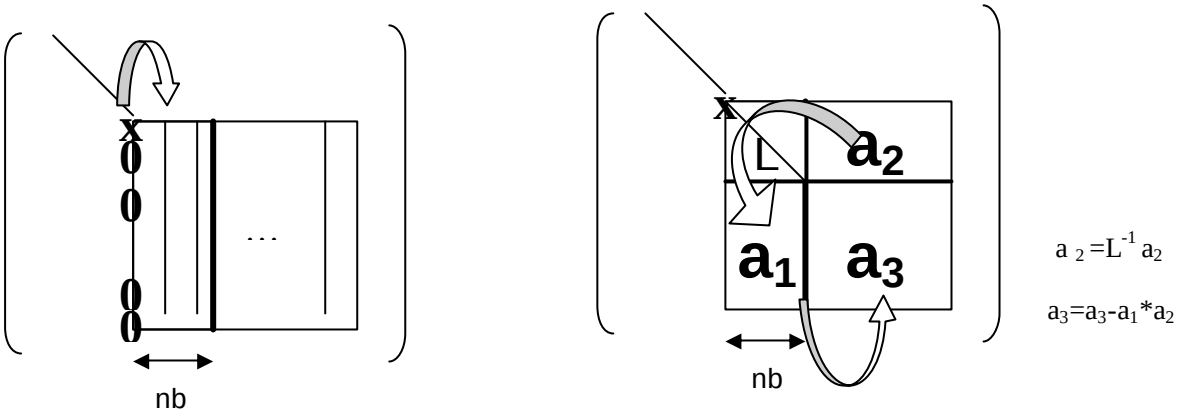
2.1 消元过程演进[10]



标准方法
实施行初等变换



LINPACK
实施列初等变换



HPL

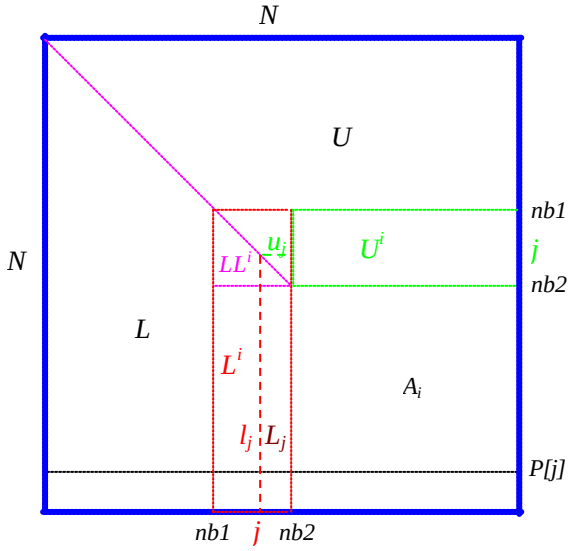
一次性处理nb列　　然后利用处理的nb列更新矩阵剩余部分

图 1. 消元过程演进三阶段示意图

作为解线性方程组的一类重要而有效的方法，高斯列主元消去法的算法是这样的，为增加算法的数值稳定性，首先寻找增广矩阵(A|b)第一列中绝对值最大的数，即主元，选取该行作为参照行，将其它行的第一个元素消为零，然后对剩余行依次重复该操作，直到将矩阵 A 分解为下三角阵 L 与上三角阵 U 的乘积，完成消元过程。之后通过上三角方程回代求得线性方程组(2)的解。

只是传统的高斯消去法对增广矩阵(A|b)施行行初等变换，而 linpack 中出于 Fortran 语言存储、列向找主元等运算方便的考虑以列为操作单位。随着现代并行计算机的出现，为了提高并行度，HPL 则继承 Lapack[8]采用矩阵分块的思想，利用 BLAS 库[9]对向量、矩阵进行操作，按图 2 示意的过程完成消元。演进的三阶段示意如图 1。

BLAS（Basic Linear Algebra Subroutines）包是一些关于矩阵的基本操作。 共分三层，第一层（最底层）实现向量与向量的运算，比如向量内积(DDOT)，DAXPY，即 z = ax + y，其中 x，y 为向量，a 为标量)；第二层（中间层）实现向量与矩阵的运算，比如 DGEMV, 即 z = aAx + by，其中 x, y 为向量，a 为标量，A 为矩阵；第三层（最高层）实现矩阵与矩阵的运算，如 DGEMM, 即 D = aAB + bC，其中 a, b 为标量，A, B 和 C 为矩阵。上一层是建立在下一层的基础之上的，运行效率也有相当提高。Linpack 软件包建立在 BLAS 之上，HPL 算法中大量的浮点运算是通过调用 BLAS 实现的，这就为使用特定的计算机硬件但不修改底层的算法提供了条件，不需牺牲可靠性，就可以实现移植和软件透明。



[1] nb1=i×NB, nb2=nb1+NB

[2] for(j=nb1; j<nb2; j++)

[2.1] 找主元 P[j] s.t. $|L^i_{P[j],j}|>=|L^i_{j:N,j}|$

[2.2] $L^i_{j,:} \longleftrightarrow L^i_{P[j],:}$

[2.3] $l_j \leftarrow l_j/L^i_{j,j}$

[2.4] $L_j \leftarrow L_j - l_j u_j$

[3] 行向广播 LL^i, L^i 及行交换信息

[4] for (j=nb1; j<nb2; j++)

[4.1] $A^i_{j,:} \longleftrightarrow A^i_{P[j],(Li\ 之后的列)}$

[5] $U^i \leftarrow (LL^i)^{-1}U^i$

[6] $A^i \leftarrow A^i - L^i U^i$

图 2. 分块矩阵 LU 分解过程示意

2.2 LU 分解算法并行思想

根据上述算法过程分析，容易获知算法中 LU 分解的运算量涉及 A(i+1:n,i)/A(i,i)和 A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n)两部分，即

$$\Sigma_{i=1\sim(n-1)} [\ (n-i) + 2*(n-i)^2\ ] = (2/3)*\ n^3 + O(n^2)$$

求解 L*y=b 和 U*x=y 各自的运算量是 $O(n^2)$，所以总的运算量仍然是 $(2/3) * n^3 + O(n^2)$。

可以看出，LU 分解是线性方程组求解的主要部分，它占去的运行时间是 O（n^3）。而串行分解中，计算工作量又主要在更新矩阵 A，即 a[i][k]-a[i][j]*a[j][k]，所以并行的主要任务就是在多处理机上同时对矩阵 A 的不同部分作更新。系列的分析与比较[10]最终确定了稠密矩阵在处理器上的块循环分布格局。一方面数据按块分布可以考虑每个处理器的内存层次，在单处理器计算部分更好的利用 BLAS3 矩阵运算统一作延迟更新，提高运算效率，减少通信，一方面循环分布可以尽可能合理的在各处理器间分配任务，均衡负载。这样一来 NB 的确定就变得至关重要了。

随机生成的稠密矩阵 $A_{N\times(N+1)}$被分成 NB×NB 的块，循环分布在 p=P×Q 的二维处理器阵列上，每个进程消元一个局部存储的约 N/P× N/Q 个子块。方便起见，从 0 到 p-1 编号处理器，0 到 N+1（或者 N）标识矩阵各列（或者行），则矩阵元素的分布和图 3 处理器阵列的对应关系如图 4 所示。对于矩阵 A 的元素 A(m,n)，它分配在 p(i,j)上，用公式表示则为，

$$i = (m/\ NB)\ \mathrm{mod}\ P，\ j = (n/\ NB)\ \mathrm{mod}\ Q$$

即 $p = p(i, j) = i+jP$

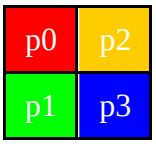


图 3. 列向优先处理器阵列分布示意（P×Q=2×2）

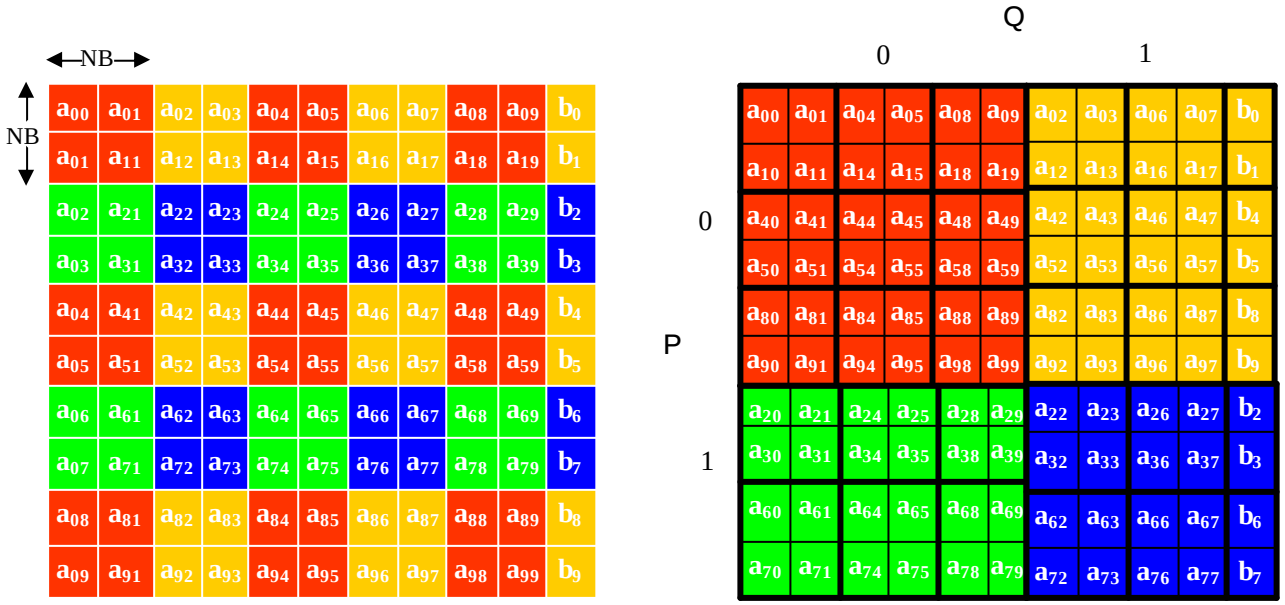


图 4. 分块矩阵对应于处理器阵列的数据分布情况（P×Q=2×2，N =10，NB× NB =2×2）

对于 HPL 来说，即使假定每个处理器的负载可以保证处理能力的完全使用，而多处理器并行操作时

不可避免的通信开销以及策略性的冗余计算，一定程度上制约了并行处理的峰值性能。为此按图 2 伪码所示步骤对分解过程中的开销做些定量的分析。

设α=通信准备延迟，β= 1/带宽，L= 通信包大小，以公式形式定义通信时间为：

$$T = \alpha + \beta L$$

同时设矩阵运算的效率因子为γ，即实测矩阵运算时间与相应运算量的比值，其值越小，运算效率越高。

首先，以 NB 列的一个 Panel 为研究对象，给出并行操作中的处理器开销，设 jj =j % NB 为局部于 Panel 的列号。

消元过程中，每次[2.1]找主元，同时完成[2.2] Panel 内 NB 列的行交换，将有（LogP)(α+β(2×NB+4))的通信量，将主元交换到矩阵对角线位置后，利用冗余存储各处理器同时[2.3]求 l_j 需要（N -j）的计算量，[2.4]更新 Panel 内后续列需要 2(N-j)(NB-jj）的计算量。像这样完成第 i 个 Panel 消元后，按 6 种通信方式[2]之一进行[3]行向通信，这里主要涉及克服网络带宽限制及自身负载问题的考虑。以 increase ring modified 方式为例，每次需要α+β((N-i×NB)×NB)/P 的通信量，然后矩阵剩余部分根据获得信息进行[4]行交换，也就是列向广播 U^i 给同一列其余的 P-1 个处理器，此处以 long 方式为例，需要α(LogP+P-1) + 3β(N-(i+1)NB)×NB/Q 的通信量。各处理器再利用冗余存储通过[5]同时算得更新需要的 U^i，耗费 (N-(i+1)NB)×NB²/Q，利用冗余计算屏蔽了单个处理器计算再广播的通信开销，做[6]完成矩阵剩余部分的更新，需要 2(N-(i+1)NB)²×NB/(PQ)的计算量。

粗略地汇总可知，Panel 处理的消耗为

$T_{消元} = T_{算L} + T_{找主元} + T_{广播L}$

$= [\ \gamma((N-i\times NB)/P - NB/3)\times NB^2 + NB(LogP)(\alpha+\beta(2\times NB+4)) + \alpha+\beta(N-i\times NB)NB/P\ ]$

$T_{更新} = T_{算U} + T_{余部更新} + T_{换行}$

$= [\ \gamma((N-(i+1)NB)\times NB^2/Q + 2(N-(i+1)NB)^2\times NB/(PQ)) + \alpha(LogP+P-1) + 3\beta(N-(i+1)NB)\times NB/Q\ ]$

回代过程基本属串行操作，而且所占比例一般不超过 5%，并行优化通常无需对此部分做过多考虑。

综上，我们很容易看出，并行算法中增加了 P-1 次算 U 的冗余计算量，及 $T_{找主元} + T_{广播L} + T_{换行}$三部分通信相关开销，这些都是提高多处理器效率不可逾越的。系统实际的性能应该是（1）的修正：

$$(2n^3/3 + 3n^2/2) + \text{冗余计算}) / (t\text{-通信时间})$$

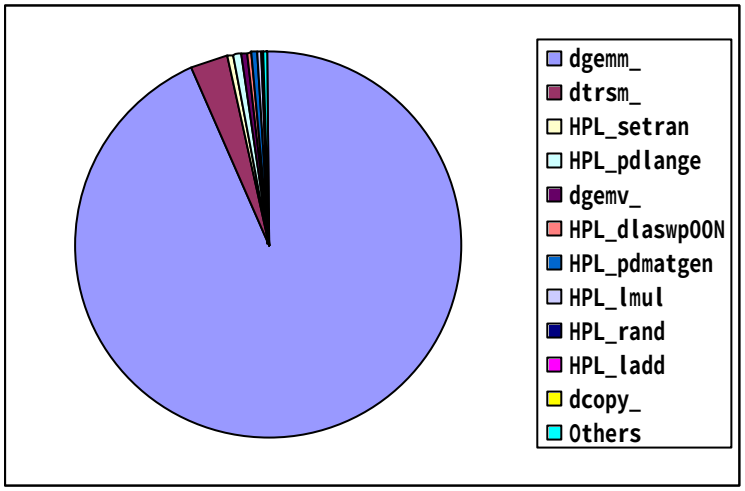
由于通信性能相关于通信延迟和带宽，受限于互连情况，且随着互连网络的飞速发展，不成为实质性的瓶颈，而计算性能仅与系统主频和问题规模等相关，利于量化。经验表明对于既定的系统，矩阵规模一般参照内存的 80%，所以将分析锁定在主要影响因素：分块大小 NB，处理器阵列 P×Q，矩阵规模 N 和通信算法[11]的前两者。

3．观察与探讨

从 HPL 源程序的分析及前述算法复杂度的计算可以看出，列向开销涉及找主元，虽然每次仅有（2×NB+4）通信量，但矩阵的每列都需要做 LogP 次，相当频繁，以及为 U^i 做的行交换(N-(i+1)NB)×NB/Q，列存储完成行交换，开销较大；行向则主要是通信量(N - i×NB)NB/P 的 L 广播。基于这样的考虑，在 HPL 测试中做出 P 为 2 的幂值且 P、Q 大体相当，但 P 略小于 Q 的决定更近乎合理。

NB 的确定则始终依赖于经验性的试探，没有相对定性的方法。测试经验[2]也不过给出：NB 应尽可能接近 Cache 行大小，既能最大限度发挥 Cache 性能，又减少 Cache 冲突，还能尽可能的利用 BLAS3 进行矩阵运算，一般在 32 到 256 之间。从数据分布的角度看，NB 越小，负载越均衡；从计算角度看，NB 太小在很大程度上限制计算性能，因为这样在内存高层几乎没有数据重用，消息的数量也会增加。

从主频 1.6GHz 双 CPU 的 AMD 64 位机器上的实际测试，见图 5，我们可以明显的看出，更新部分的矩阵加乘操作 DGEMM 是 HPL 过程中关键部分，即耗时最多，所以将确定 NB 的主要目标锁定在单机峰值计算性能的获取上。



Operation	Scale(%)
dgemm_	93.71
dtrsm_	3
HPL_setran	0.6
HPL_pdlange	0.48
dgemv_	0.48
HPL_dlaswp00N	0.42
HPL_pdmatgen	0.42
HPL_lmul	0.3
HPL_rand	0.18
HPL_ladd	0.12
dcopy_	0.12
Others	0.17

图 5. HPL 整个流程各部分耗时比例

算法及程序的运算过程显示，矩阵乘加操作只与 N 和 NB 有关。此时的效率因子相应定义为实测时间 t_{DGEMM} 与相应运算量 $2(N-(i+1)NB)^2 \times NB/(PQ)$ 的比值，即

$$\gamma = t_{DGEMM} / (2(N-(i+1)NB)^2 \times NB/(PQ))$$

图 6 和图 7 基于 Goto 库[12]的实验表明，矩阵运算的效率因子随 N 的倍增只有千分之几的变化，却很大程度上相关于 NB 的变化，随着 NB 一定范围内的增加，其值明显减小。另外，利用图中转平的点可以大致确定矩阵操作发挥效率的最小规模，从而界定 NB 初值。

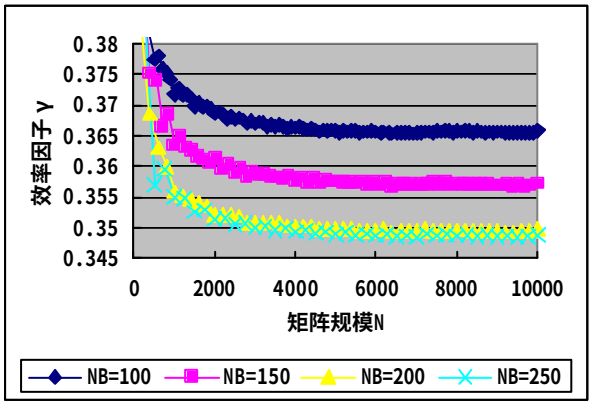


图 6. N=10000，NB=100, 150, 200, 250 四种情况下，实测 γ 结果

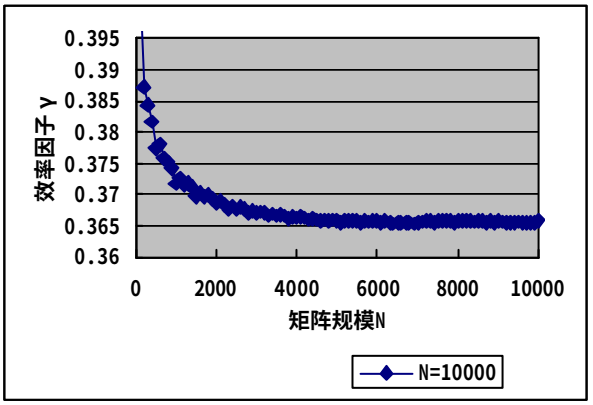


图 7. NB=100, N=0~10000 情况下，实测 γ 结果

根据上述规律，使得利用适当规模的矩阵加乘操作定性的确定 NB 成为可能。这样，只要用很小的规模 N 就可以在较短时间内得到 NB 较大范围内 γ 的变化情况，同时权衡 Cache 等其他因素，从而定性的确定合理的 NB 值。

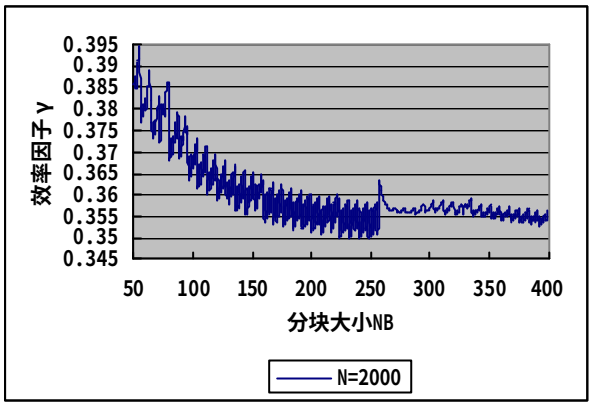


图 8-1．N=2000，NB=50~400 情况下，实测γ结果

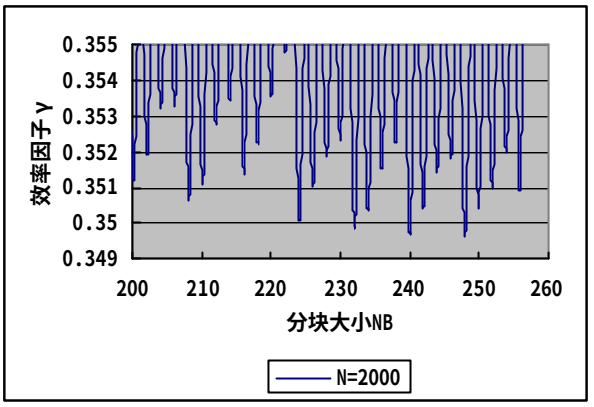


图 8-2．截取图 8-1 中 N=200~260 部分放大

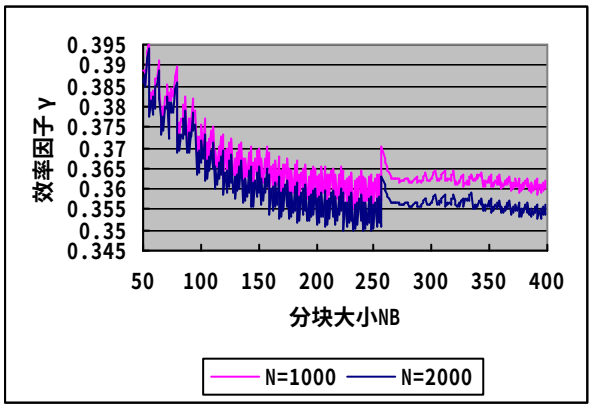


图 9-1．N=1000 和 2000 情况下实测γ结果对比

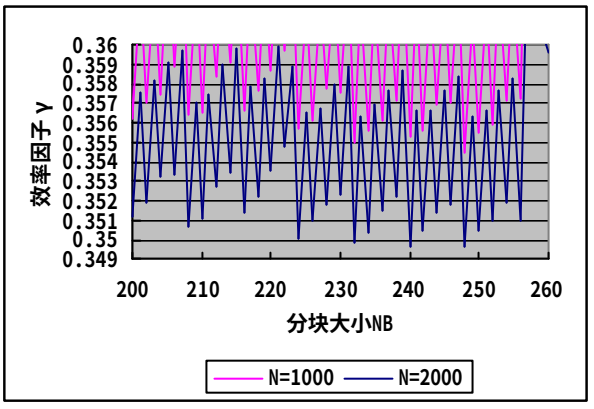


图 9-2．图 9-1 中高运算效率区域放大

为了验证这一想法，我们做了相关实验。从图 8 利用小规模矩阵测得的γ值系列中分别选取有代表性的高、低效率的点，对其 NB 值进行真实系统的实际测试。表 1 的测试结果对我们的推断给出了有利的验证。只是在小规模时，NB 的改变对系统效率的改善并不明显。随着矩阵规模增大，优势才显现出来。不过对于 HPL 来说，NB 的最终确定还要在此基础上根据 Cache 等系统特性进一步筛选。图 9 则更有力的回应了γ与规模 N 几乎无关的论断。

表 1. 代表性 NB 值实际系统效率测试结果

N	NB	γ	时间(秒)	Gflops	效率(%)
1000	240	0.355267	0.30	2.221e+00	69.4
	257	0.370144	0.32	2.112e+00	66.0
2000	240	0.349712	2.16	2.467E+00	77.1
	257	0.363151	2.25	2.373E+00	74.2
10000	240	-	239.34	2.786E+00	87.1
	257	-	247.77	2.691E+00	84.1

结论及进一步的工作

实验表明，矩阵操作的运算效率与分块大小 NB 相关。这样很容易从小规模矩阵的运算效率曲线的变化趋势中初步确定合适的 NB，为进一步根据 Cache 等情况最终确定最佳的 NB 获取峰值性能提供了很好的依据，极大的减小了 NB 选取的盲目性。这一结论同样适用于其他过程中的矩阵并行操作。

但上述测试有待于进行真实的大规模系统检验，并且有必要针对通信做进一步改进完善。本文作者对 HPL 的分析研究尚属初期，有许多验证分析工作、改进的设想需要分析评估并具体实现。比如，计算和通信部分的交迭，改同步通信为异步以隐藏延迟，打包和通信动作的流水化等，尤其是权衡内存大小与列存储行交换的开销，进一步分析利用硬盘预取扩大矩阵规模和大内存中同时做行、列存储缩小矩阵规模的可行性，从而提高测试的峰值性能等问题有待于进一步实验论证。另外，将开展线程和进程对比研究。

参考文献

[1] Jack J. Dongarra and Piotr Luszczek and Antoine Petitet. The LINPACK Benchmark: Past, Present, and Future, Concurrency and Computation: Practice and Experience 15, 2003.

[2] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary, HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, http://www.netlib.org/benchmark/hpl/.

[3] Hans W. Meuer, Erik Strohmaier, Jack J. Dongarra, and H.D. Simon. Top500 Supercomputer Sites, 17th edition, November 2 2001. (The report can be downloaded from http://www.netlib.org/benchmark/top500.html).

[4] 张宝琳等著。数值并行计算原理与方法，北京：国防工业出版社，1999.7。

[5] 林成森，编著。数值计算方法（上册），北京：科学出版社，1998。

[6] 陈国良，编著。并行计算：结构、算法、编程（修订版），北京：高等教育出版社，2003.8。

[7] 孙志忠等编著。数值分析（第 2 版），南京：东南大学出版社，2002.1。

[8] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenny, A., Ostrouchov, S., and Sorensen, D. LAPACK Users' Guide, 2nd edition. SIAM, Philadelphia PA, 1995.

[9] R.P. Brent and P.E. Strazdins. Implementation of BLAS level 3 and LINPACK benchmark on the AP1000. *Fujitsu Scientific and Technical Journal* , 29(1):61-70, 1993.

[10] http://www.cs.utk.edu/~dongarra/WEB-PAGES/SPRING-2000/lect08.pdf

[11] H.-Y. Lin and P. Luszczek, "Tuning LINPACK N*N for PA-RISC Platforms", Presented at High Performance Computing on Hewlett-Packard Systems Conference, Bremen, Germany, October 7-9, 2001.

[12] http://www.cs.utexas.edu/users/kgoto/

作者简介：

张文力，女，中科院计算所在读硕士研究生，主要研究方向为并行操作系统，高性能计算。

电话：62565533-5804、5732

E-mail：zhangwl@ncic.ac.cn