# LINPACK 与机群系统的 LINPACK 测试

LINPACK and LINPACK Benchmark

都志辉 吴 博 刘 鹏 陈 渝 王小鸽 李三立
(清华大学计算机系 北京100084)

**Abstract** LINPACK Benchmark is an important benchmark for high performance computing. This paper first briefly introduces LINPACK and LINPACK Benchmark, then gives a detail description of LU decompression which is the key part of LINPACK Benchmark. Some LINPACK test results are based on cluster system. The conclusion is given in the end.

**Keywords** LINPACK, LINPACK Benchmark, Cluster System

## 一、引言

LINPACK[1,2,7]是一个用 Fortran 语言编写的线性代数软件包,主要用于求解线性方程和线性最小平方问题。该软件包提供了各种线性系统中的求解方法,比如各种各样的矩阵运算,可以从文[2]得到完整的 LINPACK 软件包。LINPACK 的初衷并不是制订一个测试计算机性能的统一标准,而只是提供一些常用的计算方法的实现,但是由于该软件包的广泛使用,这样就为通过 LINPACK 例程来比较不同计算机的性能提供了可能。

"LINPACK User's Guide"是一本最早对 LINPACK 进行系统介绍的书,在该书的附录中,给出了当时23台不同类型的机器用高斯消去法求解稠密线性方程组 AX=B 所需要的时间,这就是后来的 LINPACK 测试标准产生的直接原因,也可以把它看作是最早的 LINPACK 测试报告。

LINPACK 测试标准具体包括三个不同内容的测试:1)n=100测试;2)面向峰值性能的 TTP 测试,又称 n=1000测试;3)高度并行化计算的测试,这种测试对数组大小 n 没有限制。

1)n=100测试 这是要求最严格的一种测试,问题的规模是100,所使用的测试程序可以从 http://www.netlib.org/benchmark/LINPACKd 下载,编译运行该程序后,该程序会给出相应机器的性能。这一测试不允许对测试程序进行任何修改,哪怕是注释行也不许改动,所有的优化只能在编译器里完成。可以从 http://www.netlib.org/benchmark/LINPACKc 得到相应的 C 程序,也可以从 http://www.netlib.org/benchmark/LINPACKjava/得到 java 程序。

2)n=1000,面向峰值性能的 TTP 测试 这种测试比第一种要求有所放宽,测试问题的规模是1000,相应的测试例子程序可以从 http://www.netlib.org/benchmark/1000d 得到,但是测试者可以根据自己的需要,修改或替换其中的过程调用例程 DGEFA 和 DGESL。其中 DGEFA 是 LINPACK 软件包中标准的高斯消去 LU 分解过程,而 DGESL 是根据分解后得到的结果回代求解过程。

3)高度并行化计算的测试 这是针对现代的并行计算机提出的测试方式,也是要求最松的一种测试,相对于第二种测试,它进一步取消了对问题规模的限制。即用户可以对任意大小的问题规模,使用任意个数的 CPU,使用各种优化方法(但必须是基于高斯消去法的)来执行该测试程序,寻求最佳的测试结果。为什么必须基于高斯消去法?这是因为,为了测试不同机器的性能,一种最简单的办法就是让它们解决同样计算规模的问题,看各自所需要的时间。对于高斯消去法,求解规模为 n 的问题,其浮点运算次数为:$2/3\ n^3+2n^2$,因此只要给出问题规模 n,测试高斯消去法实现得到的时间,就可以计算给定机器的性能—浮点运算/秒:

$(2/3n^3+2n^2)$/求解时间

全世界最快的前500台计算机排名 TOP500[3,8]的测试结果就是这种类型的测试,其最新的测试报告可以从 http://www.netlib.org/benchmark/performance.ps 得到。TOP500的性能数据从1993年开始收集,每半年更新一次。

对于这三种类型的测试,有一些共同的要求:

①所使用的矩阵不能是自己任意给出的,必须和从

规范中提出了更好的解决方案,即将内存中的数据结构直接映射到 IDL 接口,通过 CORBA 直接传递 XML DOM 树,而不必进行任何的转换,从而大大提高了系统效率。

## 参 考 文 献

1 Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/REC-xml, Oct. 6,2000

2 Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation. World Wide Web Consortium. http://www.w3.org/TR/DOM-Level-2-Core/, Nov. 13, 2000

3 Henning M, Vinoski Steve. Advanced CORBA Programming with C++(Addison Wesley, 1999)

4 SAX 2.0: The Simple API for XML. http://www.megginson.com/SAX/

5 Schmidt D C, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture: Patterns for Concurrency and Distributed Objects, Volume 2 (Wiley & Sons, 2000). http://www.posa.uci.edu/

6 Object Management Group. XMLDOM: DOM/Value Mapping Specification. http://www.omg.org/cgi-bin/doc? ptc/2001-04-04, 2001

7 Vinoski S. New Features for CORBA 3.0. Communications of the ACM, 1998, 41(10)

netlib 下载得到的程序所使用的相同。用给定的伪随机数产生器得到的矩阵可以保证在消去过程中必然有主元的选取过程。

②求解误差规范化后应该和O(1)是一个数量级，若误差大于O(100)，则结果一般认为是不正确的，即必须满足：

$\|Ax-b\|/\|A\|\|x\|n\varepsilon \leqslant O(1)$，其中 $\varepsilon$=机器精度

③求解精度为64位或64位以上。

LINPACK 测试是关于稠密矩阵的测试，从2000年5月开始收集稀疏矩阵测试标准，相关信息可以从 http://www.netlib.org/benchmark/sparsebench/得到。

## 二、与 LINPACK 相关的软件包

### 1. BLAS

BLAS(Basic Linear Algebra Subroutines)[4]是一些关于矩阵的基本操作，BLAS 用于简单的向量操作，比如给指定的向量增加多少倍后再加到另一个向量上 (SAXPY)，向量内积(SDOT)，LINPACK 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低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。BLAS 包分三层，第一层(最底层)实现向量与向量的运算，第二层(中间层)实现向量与矩阵的运算，第三层(最高层)实现矩阵与矩阵的运算。上一层是建立在下一层的基础之上的。

LINPACK 软件包建立在 BLAS 之上，准确地说主要是建立在 BLAS 的第一层之上的。

### 2. LAPACK

现在 LINPACK 软件包的各种功能已经被新的软件包 LAPACK[5]所取代，可以从 http://www.netlib.org/lapack/得到该软件包。因为 LINPACK 在70年代末产生，LINPACK 设计目标是用于70年代末和80年代初的超级计算机，LINPACK 在新型体系结构的计算机上是无效的，究其原因是算法的内存访问模式没有考虑 RISC 结构和向量计算机的多层内存结构，因此，大量的时间用在数据的移动而不是进行有效的浮点操作上。LAPACK 解决此问题的方法就是将算法进行重新组织，使用块矩阵操作，比如内层循环的矩阵乘。对每一计算机结构，通过块优化可以解决内存层次的问题，以一种可以传输的方式在现代各种不同的计算机上赢得高性能，由于为了取得最快的性能，LAPACK 需要最高级别的优化，对每一机器上的最佳矩阵块大小进行调整，这些操作通常通过3级的 BLAS 调用实现。它可以有效解决具有层次化内存结构的计算机上的大规模问题。

### 3. ScaLAPACK

ScaLAPACK[6,9]是 LAPACK 并行化版本，其基本思路是通过分块的矩阵运算来提高效率。它包括 PBLAS，BLAS，BLACS，LAPACK 等几个部分，它们之间的关系如图1所示。

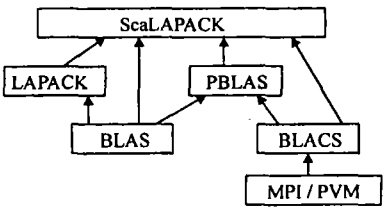


图1

## 三、高斯消去法与 LU 分解

对于线性方程组 AX=B，高斯消去法通过消元首先将 A 化为一个三角阵，然后回代过程可以求得 X。用矩阵运算表示就是将 A 分解为一个单位下三角矩阵 L 和一个上三角矩阵 U，即 A=LU，则有 LUX=B，令 Y=UX 得 LY=B，可以求得 Y，然后由 UX=Y 求得 X。求解过程中 LU 分解占去的时间是主要部分，为 $O(n^3)$，而回代求解所需要的时间仅为 $O(n^2)$，下面介绍 LU 分解的递推公式。令

$$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \quad L=\begin{pmatrix} l_{11} & \cdots & l_{1n} \\ \vdots & & \vdots \\ l_{n1} & \cdots & l_{nn} \end{pmatrix} \quad U=\begin{pmatrix} u_{11} & \cdots & u_{1n} \\ \vdots & & \vdots \\ u_{n1} & \cdots & u_{nn} \end{pmatrix}$$

则

$$a_{ij}=\sum_{k=1}^{n} l_{ik}u_{kj}=\begin{cases} \sum_{k=1}^{i-1} l_{ik}u_{kj}+u_{ij}, i\leqslant j \\ \sum_{k=1}^{j-1} l_{ik}u_{kj}+l_{ij}u_{jj}, i>j \end{cases} \Rightarrow$$

$$\begin{cases} u_{1j}=a_{1j}, j=1,\cdots,n, u_{ij}=a_{ij}-\sum_{k=1}^{i-1} l_{ik}u_{kj}, 2\leqslant i\leqslant j \\ l_{i1}=a_{i1}/u_{11}, i=2,\cdots,n, l_{ij}=(a_{ij}-\sum_{k=1}^{j-1} l_{ik}u_{kj})/u_{jj}, i>j>2 \end{cases}$$

对于 LU 分解过程的第 k 步，假设经过 k−1步分解后，矩阵 A 的分解过程为如下形式(图2)。

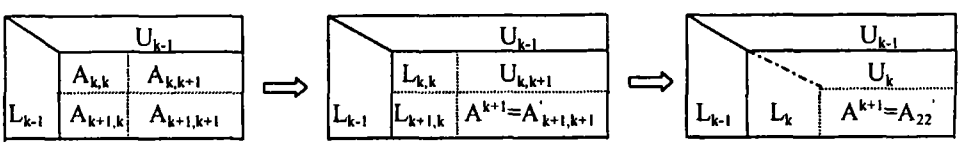


图2

其中 $L_{k-1}$ 与 $U_{k-1}$ 是前面 k−1次分解后得到的部分，将其它的部分记为 $A^k$：

$$A^k=\begin{pmatrix} A_{k,k} & A_{k,k+1} \\ A_{k+1,k} & A_{k+1,k+1} \end{pmatrix}$$

$$=P^k\begin{pmatrix} L_{k,k} & 0 \\ L_{k+1,k} & L_{k+1,k+1} \end{pmatrix}\begin{pmatrix} U_{k,k} & U_{k,k+1} \\ 0 & U_{k+1,k+1} \end{pmatrix}$$

$$=P^k\begin{pmatrix} L_{k,k}U_{k,k} & L_{k,k}U_{k,k+1} \\ L_{k+1,k}U_{k,k} & L_{k+1,k}U_{k,k+1}+L_{k+1,k+1}U_{k+1,k+1} \end{pmatrix}$$

由于在分解过程中需要选主元，这样就需要在不同行之间进行交换，因此引入变换矩阵 $P^k$。对 $A^k$ 进一步分解得到 $L_k$ 和 $U_k$。递归分解下去可得完整的 L 和 U，从而有 A=PLU。

对于上面 LU 分解过程的每一步，可以一次分解得到 U 的一行和 L 的一列，也可以一次分解得到多行多列。

## 四、数组的循环块分解与并行求解的实现

### 1. 数组在处理器阵列上的循环块分布

对于给定个数的 CPU 共有 E=P×Q 个，它们形成一个二维网格。给定 M×N 的矩阵 $A_{MN}$，将它的元素以循环块分布的形式，分别存放到 P×Q 的处理器阵列 E 上，分块大小为 $m_b\times n_b$，假设 P×Q=2×2=4个处理器，M×N=16×16，$m_b$

$\times n_b = 4 \times 4$，则处理器阵列为：

| E(0,0) | E(0,1) |
|---|---|
| E(1,0) | E(1,1) |

矩阵按 $m_b \times n_b = 4 \times 4$ 循环分块后，数组元素的分布和处理器阵列的对应关系如图3所示。

| | | | |
|---|---|---|---|
| A(0,0)~A(0,3)<br>\|E(0,0)\|<br>A(3,0)~A(3,3) | A(0,4)~A(0,7)<br>\|E(0,1)\|<br>A(3,4)~A(3,7) | A(0,8)~A(0,11)<br>\|E(0,0)\|<br>A(3,8)~A(3,11) | A(0,12)~A(0,16)<br>\|E(0,1)\|<br>A(3,12)~A(3,16) |
| A(4,0)~A(4,3)<br>\|E(1,0)\|<br>A(7,0)~A(7,3) | A(4,4)~A(4,7)<br>\|E(1,1)\|<br>A(7,4)~A(7,7) | A(4,8)~A(4,11)<br>\|E(1,0)\|<br>A(7,8)~A(7,11) | A(4,12)~A(4,16)<br>\|E(1,1)\|<br>A(7,12)~A(7,16) |
| A(8,0)~A(8,3)<br>\|E(0,0)\|<br>A(11,0)~A(11,3) | A(8,4)~A(8,7)<br>\|E(0,1)\|<br>A(11,4)~A(11,7) | A(8,8)~A(8,11)<br>\|E(0,0)\|<br>A(11,8)~A(11,11) | A(8,12)~A(8,16)<br>\|E(0,1)\|<br>A(11,12)~A(11,16) |
| A(12,0)~A(12,3)<br>\|E(1,0)\|<br>A(15,0)~A(15,3) | A(11,4)~A(11,7)<br>\|E(1,1)\|<br>A(15,4)~A(15,7) | A(11,8)~A(11,11)<br>\|E(1,0)\|<br>A(15,8)~A(15,11) | A(11,12)~A(11,16)<br>\|E(1,1)\|<br>A(15,12)~A(15,15) |

图3 数组元素的分布与处理器阵列的对应关系

用公式表示为，对于矩阵 A 的元素 A(m,n)，它分配在 E(i,j)上，则 $i=(m/m_b) \bmod P$，$j=(n/n_b) \bmod Q$。

## 五、测试与分析

### 1. 测试环境与测试结果

我们在一个机群系统上进行了大量的测试。该机群系统的配置是这样的。每个结点是双 CPU 的 SMP 结构，CPU 是 PIII 700，共有8个结点，共16个 CPU。通信网络包括100M 的 Ethernet 和高速的 Myrinet 两套通信系统。通信环境是 MPI，具体的实现版本是 MPICH-1.1.1，使用的操作系统是目前最常使用的自由软件 Linux。具体的实现版本是 Redhat 6.2。图4－7是部分测试结果，其中 MR 是指在 Myrinet 上的测试结果，ETH 是指在100M 以太网上的测试结果，NB 表示计算时循环块分布的大小，N 表示计算的矩阵数组大小为 N * N，P * Q 表示 CPU 的排列方法。

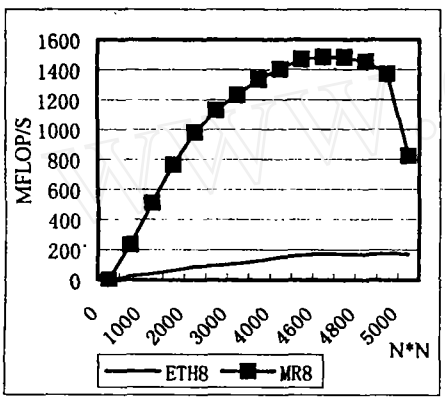


图 4 Myrinet 与 Ethernet 测试结果的对比
16CPU，NB=32，P*Q=2*8

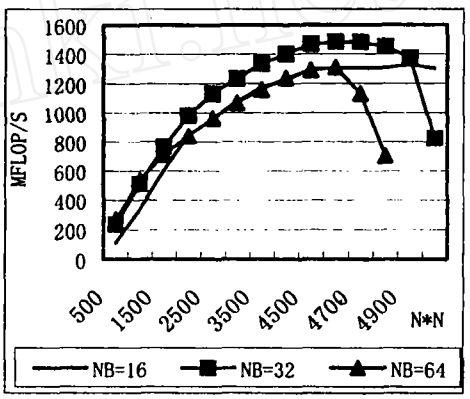


图 5 分块大小对测试结果的影响
8CPU，P*Q=1*8，Myrinet 网络

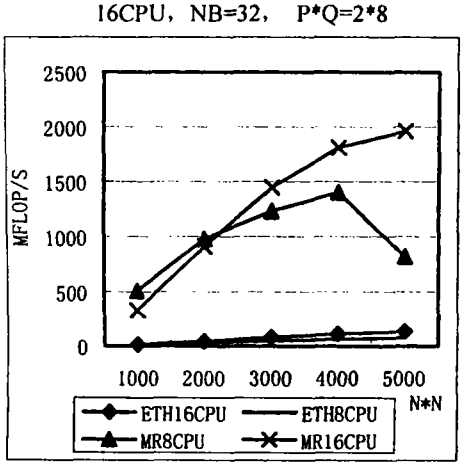


图 6 不同处理器的测试结果
NB=32，Myrinet 与 Ethernet 网络

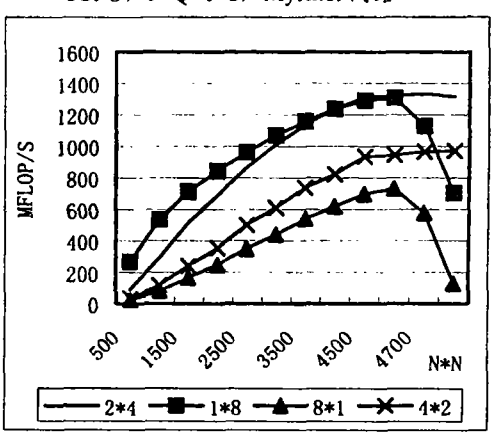


图 7 不同处理器拓扑的测试结果
8CPU，NB=64，Myrinet 网络

### 2. 结果分析

首先从图4可以看出，通信性能对 LINPACK 测试结果的影响是十分显著的，显然 Myrinet 网络上的测试结果要远远优于 Ethernet 上的测试结果。值得一提的是，这里使用的是基于 Myrinet 的 TCP/IP 通信，其通信开销还是很大的，如果使用基于用户层的轻量级通信形式，则测试结果将会更好。这从另一个方面说明了通信性能是机群系统性能的重要瓶颈，提高通信性能对于机群系统整体性能的提高有重要的意义。

数据在不同处理器上的分布方式对并行计算的影响也非常大，数据分块如果过大，则容易造成负载不平衡；但是如果数据分块过小，则通信开销就会很大，同样会影响计算的整体性能。因此，合适的数据分布对提高机群计算的性能也是十分重要的。从图5可以看出，适合与特定系统的数据分块（这里是32）相对于不合适的分块（这里是16或64），其性能是很不相同的。

（下转第59页）

的大规模视频点播系统。

## 参 考 文 献


1 http://www.synlead.com.cn/china/ixjet/vod.htm

2 http://www.ncube.com

3 周笑波，谢立. Reinhard Lueling，视频服务器网络中影像对象映射问题. 软件学报，2000，11(12)：1620～1627

4 周笑波. 分布式即时影像服务器网络中若干问题的研究：[南京大学博士论文]. 2000，4

5 Ahn Y J，Won Y H. A Storage Strategy Considering Characteristics of Video Objects on Video-On-Demand Storage Servers. High Performance Computing in the Asia-Pacific Region，2000. In：Proc. of The Fourth International Conference/Exhibition on Volume：1，2000

6 Sumari P，Merabti M，Pereira R. Video-on-demand server：Strategies for improving performance. IEEE Proc-Softw，1999，146(1)

7 Tsao S L，Huang Y M. An Efficient Storage Server in Near Video-On-Demand Systems. IEEE Trans. on Consumer Electronics，1998，44(1)

8 Jun S B，Lee W suk. Video Allocation Methods In a Multi-level Server For Large-scale VOD Services. IEEE Trans. on Consumer Electronics，1998，44(4)

9 Huang Yueh-Min，Tsao S L. Effient Data Placement and Retrieval Scheme of Zoend-Disk to Support Interactive Playout for Video Servers. IEEE Trans. on Consumer Electronics，1997，43(1)

10 Rangan P V，Vin H M. Efficient Storage Techniques for Digital Continuous Multimedia. IEEE Trans. on Knowledge and Data Engineering，1993，5(4)

11 Hua K A，Sheu S. Skyscraper Broadcasting：A New Broadcasting Scheme for Metropolitan Video-On-Demand Systems. In：Proc. SIGCOMM 97，Canners，France，ACM，1997. 89～100

12 Viswanathan S，Imielinski T. Pyramid broadcasting for video on demand service. In：Proc. of IEEE Conf. on Multimedia Computing and Networking，Vol. 2417，San Jose，CA，1995

13 Viswanathan S，Imielinski T. Metropolitan area video-on-demand service using pyramid broadcasting. Multimedia system，1996，4(4)

14 Carter S W，Long D D E. Improving bandwidth efficiency of video-on-demand servers. Computer Networks，1999，31：111～123

15 Eager D L，Vernon M K. Dynamic Skyscraper Broadcast for Video-on-Demand. In：Proc. 4th Intl. Workshop on Multimedia Information System(MIS'98)，Istanbul Turekey，Sept. 1998. 18～32

16 Cleary K. Video on demand-competing technologies and services. Broadcasting Convention，1995. IBC 95.，Intl. 1995. 432～437

17 Eager D L，Ferris M C，Vernon M K. Optimized Regional Caching for On-Demand Data Delivery. In：Proc. IS&T/SPIE Conf. on Multimedia Computing and Networking1999 (MMCN'99)，San Jose，CA，Jan. 1999. 301～316

18 Eager D，Vernon M，Zahorjan J. Minimizing Bandwidth Requirements for On-Demand Data Delivery. In：Proc. 5th Intl Workshop on Multimedia Information Systems (MIS'99)，Indian Wells，CA，Oct. 1999

19 Mielke M，Zhang A. A multi-level buffering and feedback scheme for distributed multimedia presentation systems. Computer Communications and Networks. In：Proc. 7th Intl. Conf. on，1998. 219～226

20 Barnett S A，Anido G J. A cost comparison of distributed and centralized approaches to video-on-demand，Selected Areas in Communications. IEEE Journal，1996，146：1173～1183

21 郭建新. 层次型 VOD 系统的设计：[南京大学硕士论文]. 2001. 6


(上接第10页)

从这些测试结果还可以看出，当数组的规模越大，一般来说测试性能就会越好，整体呈上升趋势，但是达到一定阶段后却转而下降。这是因为当数组规模增大时，计算相对于通信的开销增大，因而有利于提高机群计算的性能。但是它还有一个重要的制约条件就是内存的大小。当数组增大到超过内存的容量时，测试性能反而会下降，这是因为使用磁盘交换区的性能显然要远远低于内存的性能。因此适当增加机群系统的内存对于提高机群系统的整体性能是至关重要的。

通过以上测试可以发现(图6)，当处理器个数增加时，从绝对性能看，计算效率整体呈上升趋势，因此处理器规模的扩大对计算性能是有好处的。但是也必须得看到，这种扩展性是局限于一定的条件和范围的，是受通信、内存容量等因素的影响的。从这里我们也可以得出这样的结论，计算机的整体性能和各个部分的性能密切相关，只有优化配置，协同一致，才可以充分发挥其整体的性能。不然，就会受到不合理配置造成的性能瓶颈的制约。

处理器拓扑的改变会影响数据在处理器上的分布方式，从图7可以看出，处理器拓扑的改变对计算的性能也会产生较大的影响。对于 LINPACK 测试，根据在机群系统上多组数据测试的结果发现，若 P×Q＝E，则分布形式为 P×Q＝1×E 的分布性能一般较好。分析其原因，是因为它很好地保持了数据的局部性，同时又降低了列分块之间的通信开销。

**结论** 本文通过在机群系统上 LINPACK 的测试，发现了一些基于机群系统影响 LINPACK 性能的因素。这从一个方面说明了机群系统提高整体性能的努力方向。正如 LINPACK 测试所看到的那样，通信性能显然对机群计算性能的影响很大，因此高效的机群系统必须有高效的通信系统的支持，同时数据分布和存储对机群也有非常重要的影响。

必须说明的是，LINPACK 测试只是反映计算机性能的一个方面，并不表示计算机系统性能的全部，准确地评价机器和操作系统需要收集更可靠和更有代表性的数据，但是它可以用它来作为一个重要的参考性能指标。

## 参 考 文 献


1 Bunch J，Dongarra J，Moler C，Stewart G W. LINPACK users' guide. SIAM，Philadelphia. PA，1979

2 http://www.netlib.org/LINPACK

3 TOP 500 Home Page. http://www.top500.org/

4 BLAS Home Page. http://www.netlib.org/blas/

5 LAPACK Home Page. http://www.netlib.org/lapack/

6 ScaLAPACK Home Page. http://www.netlib.org/scalapack/

7 Dongarra J J. The LINPACK Benchmark：An Explanation. Lecture Notes in Computer Science，Berlin：Springer，1988，297：456～474

8 Hack J J. Peak vs. Sustained Performance in Highly Concurrent Vector Machines. Computer，1986，19(9)：11～19

9 ScaLAPACK Tutorial. http://www.netlib.org/scalapack/tutorial/