CN43-1258/TP
ISSN 1007-130X

计算机工程与科学
COMPUTER ENGINEERING & SCIENCE

2008 年第 30 卷第 A1 期
Vol. 30, No. A1, 2008

文章编号:1007-130X(2008)A1-0032-04

LINPACK 算法及其测试方法改进*
LINPACK and the Improvement of Its Test Method

司照凯,濮 晨
SI Zhao-kai, PU Chen
(江南计算技术研究所,江苏 无锡 214083)
(Jiangnan Institute of Computing Technology, Wuxi 214083, China)

摘 要:HPL (High Performance LINPACK)是一种用来测试计算机浮点性能的基准测试程序,通过求解稠密线性方程组来评估计算机的浮点性能。本文分析了 HPL 的核心算法,并对 HPL 的计时系统进行改进,提出了一种新的基于计时系统的测试方法,以达到快速完成 LINPACK 测试的目的,实验证明这种新的测试方法很有效。

Abstract: HPL (High Performance LINPACK) is a benchmark used to measure the computer's float point performance, it reaches this by solving a random linear system in double precision. In this paper, we introduce the HPL's algorithm, and improve its timing system, then put forward a LINPACK test method: Timing System based LINPACK Test Method, which can help to accelerate the LINPACK test. The experiment shows that this new way is helpful.

关键词:高性能;LINPACK;BLAS;MPI;LU factorization

Key words: high performance LINPACK; BLAS; MPI; LU factorization

中图分类号:TP309 **文献标识码**:A

1 引言

LINPACK 是当前评测计算机浮点性能的基准测试程序,TOP500[也] 是根据计算机的 LINPACK 性能来进行排名。LINPACK 根据矩阵规模可以分为 100×100,1000×1000 和 N×N 三种[1],本文分析的 High Performance LINPACK(HPL)属于 N×N 这一类。

HPL 由 A. Petitet, R. C. Whaley, J. Dongarra 和 A. Cleary 开发,它通过求解一个稠密线性方程组(Random Dense Linear System of Equations)来测试计算机的 LINPACK 性能。本文使用的是 HPL 1.0a,2004 年发布的一个版本。

2 HPL 理论基础

HPL 通过求解一个稠密线性方程组来测试计算机的 LINPACK 性能[2],如(1)式所示:

$$Ax = b \tag{1}$$

其中,$A=(a_{ij})_{N\times N}$且为非奇异矩阵,$b=(b_1, b_2, \cdots, b_N)^T$,$x=(x_1, x_2, \cdots x_N)^T$,A 与 b 均为已知,而 x 是待求的 N 维列向量。

统计求解(1)式的时间,并且利用(2)式来计算浮点速率:

$$R = \frac{2N^3/3 + 3N^2/2}{T_{HPL}} \times 10^{-9} GFLOPS \tag{2}$$

式(2)中 $2N^3/3 + 3N^2/2$ 是浮点运算规模,T_{HPL} 是 HPL 执行时间。得到浮点速率 R 后,和峰值 R_{PEAK} 相除,就是这台计算机的 LINPACK 执行效率 η。

HPL 在求解(1)式的时候,先对矩阵 A 进行 LU 分解(LU Factorization),得到一个上三角矩阵 U 和一个下三角矩阵 L,并且 A 等于这两个矩阵的乘积,以方便方程的求解,这个过程就是 LU 分解。常用的因式分解方法还有 QR 分解和 Cholesky 分解,由于 HPL 采用的是 LU 分解,所以我们重点分析一下 HPL 中 LU 分解的实现方式。

LU 分解的形式有三种:Right-looking LU Factorization、Left-looking LU Factorization 和 Crout-looking LU Factorization,它们之间的区别主要体现在 panel 内 LU 分解以及尾矩阵更新的执行顺序不同·

HPL 中的 LU 分解采用分块的形式实现,将数据分块映射到处理器网格中,以达到均衡负载的目的。分块的大小为 NB×NB,同一列上的块组成一个 panel。HPL 实现

* 收稿日期:2008-04-13;修订日期:2008-07-10
作者简介:司照凯(1984-),男,硕士生,助理工程师,研究方向为 VISI 验证;濮晨,硕士生,助理工程师,研究方向为 VISI 设计。
通讯地址:214083 江苏省无锡市江南计算技术研究所;Tel:(021)50807696-8531;E-mail:szk1128@163.com
Address: Jiangnan Institute of Computing Technology, Wuxi, Jiangsu 214083, P. R. China

的时候先对 panel 内的数据进行 LU 分解，然后对尾矩阵进行更新，也就是 update 操作。得到 L 矩阵和 U 矩阵之后，再求出方程的解 x，并且计算误差。对 panel 内的数据进行 LU 分解是通过 Panel Factorization(PFACT)和 Recursive Panel Factorization(RPFACT)协作完成[4]，PFACT 和 RPFACT 均有 Right-looking LU Factorization、Left-looking LU Factorization 和 Crout-looking LU Factorization 三种实现形式。在后面的实验中我们发现，这些参数对 LINPACK 性能的影响不大。

HPL 的运行还需要 Message Passing Interface (MPI)和 Basic Linear Algebra Subroutines (BLAS)或者 Vector Signal Image Processing Library (VSIPL)的配合。MPI 主要用来进行各个处理器之间的通信，BLAS 和 VSIPL 为 LU 分解提供各种矩阵或者向量运算函数。

3 LINPACK 测试方法

HPL 执行的时间比较长，特别是当矩阵规模比较大的时候，而且 LINPACK 性能受各种软硬件因素以及 HPL 执行参数的影响很大，所以想取得一个较佳的结果，需要有一套合理的测试方法，不能盲目进行。

3.1 影响 LINPACK 性能的因素

LINPACK 性能主要受三个因素的影响，分别是硬件因素、软件因素和 HPL 执行参数。

Ⅰ 硬件因素

硬件因素主要包括 cache 大小和存储系统结构、访存速度、处理器性能、计算机系统的结构以及互连网络的性能等，这些因素都会影响机器的 LINPACK 性能。

Ⅱ 软件因素

软件因素主要指的是 MPI 和 BLAS 对 HPL 性能的影响。MPI 常用的有 LAMMPI、MPICH 和 OpenMPI，这三种 MPI 的性能不一样，有些针对一些特殊的结构(如 SMP)会进行优化。

BLAS 也有 Automatically Tuned Linear Algebra Software(ATLAS)、GotoBLAS、Engineering and Scientific Subroutine Library(ESSL)、Intel Math Kernel Library(MKL)和 AMD Core Math Library(ACML)，其中 ESSL、MKL 和 ACML 分别由 IBM、Intel 和 AMD 开发，并且对各自的处理器支持比较好。选择哪种 BLAS，不仅要参考计算机硬件类型，还要通过实验分析[3]。

编译器的选择也有很大关系，文献分析了两个版本的 gcc 编译器对 LINPACK 性能的影响。

Ⅲ HPL 执行参数

HPL 执行参数很多，在文件“HPL. dat”中进行设置，其中对 LINPACK 性能影响比较大的是 P×Q、N 和 NB[5]（P×Q 是处理器网格的排列形式，N 是矩阵规模，NB 是矩阵分块的大小），测试时需要不断的进行调整。

由于硬件一般比较固定，所以我们在做 LINPACK 测试时主要调整的是软件以及 HPL 执行参数。

3.2 LINPACK 测试方法

为了提高测试速度，有几种比较有效的测试方法，文献[5]和文献[6]分析了几种加快测试的方法。

文献[5]根据各参数对 LINPACK 性能影响的大小，将其分为 A 类参数和 B 类参数。A 类参数包括 P×Q、N 和 NB，其他参数都是 B 类参数。A 类参数对 LINPACK 性能影响很大，B 类参数影响较小。所以测试的时候先找到最佳的 P×Q，再确定最佳的 N，然后是 NB，当三个 A 类参数定下来之后，再确定 B 类参数。

文献[6]根据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分块大小，而与矩阵规模本身关系微乎其微这一规律，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，以达到缩短测试时间的目的。

上面两种测试方法都比较有效，但是它们都只把 LINPACK 的浮点速率作为唯一的评判基准，而忽略了很多细节，比如通信时间的比重，矩阵运算时间的比重等等。为了更好的进行 LINPACK 测试，需要提取更为丰富的信息。

本文提出了一种新的基于计时系统的测试方法，对原有的 HPL 计时系统进行改进，提取更加丰富的关键时间参数。通过计时系统，可以快速定位最佳测试平台软硬件配置和 HPL 执行参数，以达到快速完成 LINPACK 测试的目的。

4 HPL 计时系统

从(2)式中可以看出，HPL 执行时间 T_{HPL} 越短，浮点执行速率就越高。所以我们对 T_{HPL} 进行深入分析，并且从中找出影响 LINPACK 效率的因素。

4.1 HPL 执行时间分析

假设内存空间足够，不发生交换，不考虑 cache 缺失，HPL 的执行时间可以粗略的表示为[7]：

$$T_{HPL}=\frac{2N^3}{3PQ\gamma}+\frac{\beta N^2(3P+Q)}{2PQ}+\frac{\alpha N((NB+1)\log(P)+P)}{NB} \tag{3}$$

其中，α 表示处理器之间进行一次通信的启动时间，β 表示通信速率，γ 表示处理器的矩阵－矩阵浮点运算速率。

式(3)中，T_{HPL} 主要由 T_{GEMM} 和 T_{BCAST_LASWP} 两部分组成，T_{GEMM} 表示 Level 3 BLAS 函数的执行时间，也就是 HPL 源码里的 GEMM 函数。T_{BCAST_LASWP} 指的是广播通信的时间。

$$T_{GEMM}=\frac{2N^3}{3PQ\gamma} \tag{4}$$

$$T_{BCAST\_LASWP}=\frac{\beta N^2(3P+Q)}{2PQ}+\frac{\alpha N((NB+1)\log(P)+P)}{NB} \tag{5}$$

T_{GEMM} 主要受 BLAS 函数库和 cache 缺失率的影响，所以可以根据这个时间参数来选取合适的 BLAS 库和 NB 大小。T_{BCAST_LASWP} 主要受互联网路结构、P×Q、广播算法和 MPI 的影响，所以可以根据这个时间参数来选取较优的互联网络、MPI 库、P×Q 和广播算法等。本文提出的基于计时系统的测试方法正是基于以上理论基础，通过计时系统来尽快确定影响 LINPACK 性能的关键因素。

4.2 原有 HPL 计时系统

HPL 本身已经提供了一套计时系统，对 6 个时间参数

进行了统计，每个处理器都有自己对 6 个时间参数的统计信息，HPL 还会把各个处理器中 6 个参数的最大值提取出来。虽然计时系统最后只得到 6 个最大值，但是由于算法的均衡性较好，所以不同处理器各项时间参数的取值相差不大。这 6 个时间参数分别是：

HPL_TIMING_RPFACT：表示 panel 内 LU 分解的执行时间总和。

HPL_TIMING_PFACT：表示 PFACT 的执行时间，这个时间参数被 HPL_TIMING_RPFACT 所包含，是它的一个子集。

HPL_TIMING_MXSWP：表示 panel 内 LU 分解时，panel 内行交换的执行时间。

HPL_TIMING_UPDATE：表示 update 函数（update 函数用来对尾矩阵进行更新）的执行时间。

HPL_TIMING_LASWP：表示 update 操作时行交换的执行时间。

HPL_TIMING_PTRSV：表示上三角方程求解的执行时间。

这个计时系统存在很多不足之处。首先，它没有反映出广播通信的时间，但是当 P×Q 选取不同，广播算法、互联网络和 MPI 函数选取不一样的时候，广播通信时间所占的比重差异很大；其次，HPL_TIMING_UPDATE 时间参数是 update 的执行时间，但是在 update 函数内也有很多广播通信的发生，所以这个时间不能准确对应 T_{GEMM}。

为了使提取的时间参数更好的和 T_{GEMM}、T_{BCAST_LASWP} 对应，我们对 HPL 进行了修改，改进了它的计时系统，以便于 LINPACK 测试。

4.3 改进后的 HPL 计时系统

为了更好统计 T_{GEMM} 和 T_{BCAST_LASWP}，我们对 HPL 计时系统进行了改进。T_{GEMM} 主要是由 update 操作中的 GEMM 函数执行时间组成，PFACT 和 RPFACT 中也有 GEMM 函数，但是比重很小，可以忽略。T_{BCAST_LASWP} 由两部分组成，一部分是 panel 的 LU 分解之后，update 之前的广播通信时间，另一部分是 update 时行交换时间。

为此，我们增加了 2 个时间参数：

HPL_TIMING_BCAST：表示广播通信的耗时。

HPL_TIMING_GEMM：表示 update 函数内 GEMM 函数的执行时间。

在 HPL 源码中找到时间参数对应的具体函数，在函数执行的开始和结束都进行计时，这样就可以统计相应操作的执行时间。新的计时系统一共提取了 8 个时间参数。T_{GEMM} 对应着 HPL_TIMING_GEMM，T_{BCAST_LASWP} 由 HPL_TIMING_BCAST 和 HPL_TIMING_LASWP 组成。基于计时系统的 LINPACK 测试方法主要依据的就是这 3 个时间参数。

为了验证这套计时系统是否会占用过多的时间，从而影响 HPL 的执行效率，我们做了一个实验。

如表 1 所示，分别取 N＝20000、N＝16000 和 N＝10000，其它参数和配置不变，统计 HPL 执行的总时间，单位为"秒"。最后得出结论，加上计时系统对 HPL 的执行效率影响甚微。

表 1 计时系统对 HPL 的影响

N		有计时系统（秒）	无计时系统（秒）	平均偏差（%）
N=20 000	i	726.94	726.35	0.119 6
	ii	728.70	727.82	
	iii	726.03	724.87	
N=16 000	i	374.56	375.25	0.045 2
	ii	378.30	377.20	
	iii	374.41	374.31	
N=10 000	i	96.86	96.85	0.412 1
	ii	97.68	96.70	
	iii	96.66	96.46	

5 基于计时系统的 LINPACK 测试

通过计时系统可以深入的分析 LINPACK 测试的特性，并且提供一个很好的指导。本部分讲述了基于计时系统的 LINPACK 测试的一般方法，并且在一个两处理器 SMP 系统的 x86 服务器中运用这种方法进行测试。

5.1 测试一般步骤

第三部分总结了两种比较常用的 LINPACK 测试方法，接下来将在这些方法的基础上引入计时系统，讲述一种新的测试方法——基于计时系统的 LINPACK 测试方法。这种方法重点关注三个时间参数，涵盖 LINPACK 的矩阵运算和广播通信，通过这些时间参数来引导测试者调整软硬件配置和 HPL 执行参数。

基于计时系统的 LINPACK 测试方法的一般步骤如下：

第一步：确定互联网络。在不同的互联网络中运行同样的 HPL 程序，通过比较时间参数 HPL_TIMING_BCAST 和 HPL_TIMING_LASWP 确定选择哪种互联网络。

第二步：确定 BLAS，并且提取各个 BLAS 对应的性能较好的 NB 值。在单处理器、N 值较小、NB 取值范围较大的情况下进行测试和比较，确定最佳的 BLAS 函数库，时间参数 HPL_TIMING_GEMM 反应出 BLAS 的优劣。

第三步：确定 MPI。P×Q 取一个适中的值，通过提取时间参数 HPL_TIMING_BCAST 和 HPL_TIMING_LASWP，确定 MPI 库。

第四步：确定 P×Q。

第五步：确定 N，NB。

第六步：确定广播算法、DEPTH 以及其他 HPL 的执行参数。

这就是计时系统的 LINPACK 测试方法的一般步骤，在实际中可能根据不同的情况会稍有变化。

5.2 测试举例

下面将运用计时系统在一个 x86 服务器的测试平台上做 LINPACK 测试，测试平台包括软件和硬件两个方面，所用平台的软硬件情况如表 2：

表 2 测试平台软硬件配置

Processor	2×Intel(R) Xeon(TM) CPU 2.80GHZ
Level 1 Cache	8KB ICache, 8KB DCache
Level 2 Cache	512KB L2 Cache
Memory	3.8GB
R_{PEAK}	11.2GFLOPS
MPI	LAMMPI-6.5.6, MPICH-1.2.7, OpenMPI-1.2.3
BLAS	GotoBLAS-1.06, ATLAS-3.6.0
Operating System	Linux2.4 SMP
Compiler	gcc 2.96
HPL	HPL 1.0a

测试平台比较简单，P×Q只有1×2和2×1两种选择，接下来将讲述如何在这个平台下进行基于计时系统的LINPACK测试。

在我们的测试平台下，LINPACK的步骤如下所示：

第一步：确定BLAS

由于测试平台是个2处理器的SMP系统，各个处理器之间的通过共享内存进行通信，所以不需要对互联网络进行选择。图1中的数据N=10000，P×Q=1×1，NB的取值从31～310，分别使用ATLAS和GotoBLAS测试。

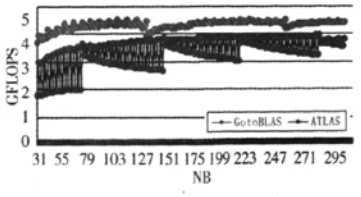

图1　P×Q=1×1时不同BLAS库(ATLAS和GotoBLAS)情况下的LINPACK浮点运算速率

从图1中可以看出，GotoBLAS比ATLAS性能优越，并且GotoBLAS在NB等于128和256左边附近的点浮点速率较快，所以最终确定使用GotoBLAS。

第二步：确定P×Q

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分块大小，而与矩阵规模本身关系微乎其微这一规律在图2中得到验证。图2中的数据N=10000，P×Q=1×2/2×1/1×1，NB的取值从31～310。

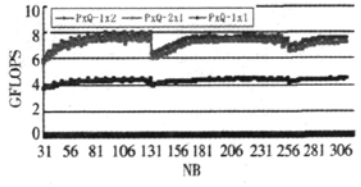

图2　曲线的相似性

从图2可以看出，P×Q在三种配置形式下的曲线具有相似性，并且P×Q=1×2时比P×Q=2×1浮点速率快，所以最终确定设置P×Q=1×2。图2的实验中NB的取值范围很大，这是为了证明曲线的相似性，实际上只要对小范围的NB进行测试就可以确定最佳P×Q值，如图3所示，只对几个NB值进行测试就可以看出P×Q=1×2时性能最佳。

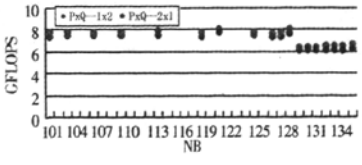

图3　几个NB值情况下的LINPACK性能

图4和图5是三个参数HPL_TIMING_GEMM、HPL_TIMING_BCAST和HPL_TIMING_LASWP的值。在图4中P×Q=2×1时HPL_TIMING_BCAST=0，这是因为panel的LU分解之后不需要广播。从图4中可以看出P×Q=1×2时的HPL_TIMING_BCAST和HPL_TIMING_LASWP之和与P×Q=2×1时的HPL_TIMING_LASWP差异不大。从图5中可以看出P×Q=1×2时的HPL_TIMING_GEMM与P×Q=2×1时的HPL_TIMING_GEMM差异也不大。在N=128的拐点处HPL_TIMING_GEMM跳跃很大，这说明这个差异主要是由BLAS导致的，与广播通信关系不大。

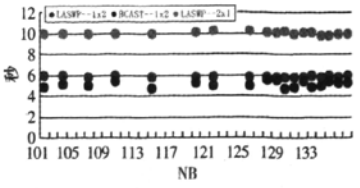

图4　不同P×Q形式下HPL_TIMING_BCAST和HPL_TIMING_LASWP的值

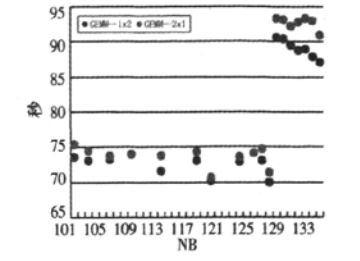

图5　不同P×Q形式下HPL_TIMING_GEMM的值

第三步：确定MPI

表3中的数据N=16000，P×Q=1×2，分别选用OpenMPI、MPICH和LAMMPI进行实验，从表中可以看出OpenMPI的性能最佳，尤其在NB值比较小的时候，所以最终确定使用OpenMPI。

表3　不同MPI库时的HPL_TIMING_BCAST值

		HPL_TIMING_BCAST(秒)		
		OpenMPI	MPICH	LAMMPI
NB	110	8.71	11.93	16.19
	114	8.40	14.45	18.21
	118	9.12	11.99	17.14
	125	8.85	14.12	17.70
	127	10.59	12.60	19.52
	128	10.81	15.84	17.50
	130	10.70	16.31	17.74
	240	13.15	18.47	20.94
	248	15.84	17.15	20.81
	256	14.67	19.51	21.04
	260	15.11	18.03	22.41
	368	17.80	22.52	24.99
	375	19.31	23.17	25.90
	384	18.35	22.02	27.74
	390	22.66	28.31	30.39
平均值(秒)		13.60	17.76	19.76

第四步：确定N、NB以及其他参数

提高N值，并且对第一步得到的几个较优NB进行测试，调节其他参数，得到最终的较佳值。通过多次实验，如表4所示，最佳的浮点速率是8.608 Gflops。

表4　几个较佳的测试结果

T/V	N	NB	P×Q	Gflops
WR00C2R8	19 200	120	1×2	8.601
WR00C2R8	19 200	128	1×2	8.608
WR00C2R8	20 224	128	1×2	8.542
WR00C2R8	19 200	112	1×2	8.178
WR00C2R8	19 200	104	1×2	8.445

(下转第44页)

换。本文借助这些转换函数解决 C 中浮点数向 VHDL 中标准位 std_logic_vector 的转换。在具体实现时，首先分别将有效数字和小数点后的有效数字位数转换成整数，然后调用 conv_std_logic_vector(ARG:integer)转换函数，将整数转换为标准逻辑向量，对应设置 std_logic_vector 中的每一位，注意第七位设置为'1'。当两个浮点数在进行运算时，首先判断两个浮点数向量的后七位是否相同，如果相同，直接对有效数进行预算即可，否则就要调整两个浮点数的后七位，以两者的后七位中最大者为基准进行调整，然后两者再进行运算。

4 结束语

HTTM 虽然能够初步实现软件任务到硬件任务的转换，但目前仍存在一些问题。首先它对输入的 ANSI C 源语言还有所限制，即不支持指针、嵌套调用、case-switch 控制语句。然后就是 HTTM 在生成硬件任务通信接口时效率不是很高。最后就是 HTTM 中用到的并行分析方法目前还只适用于特定的一类问题，即 list homomorphism 问题，目前还不具有通用性。

本文设计了一个软硬件混合任务转换模型，并研究了相应的转换算法。论文研究初步实现了软件任务和硬件任务之间的转换。考虑到该模型的具体应用，项目组正在进行基于可重构计算的实时操作系统平台的研究工作，该模型的具体研究和关键技术将被应用到该平台，HTTM 将对降低混合系统的设计周期和提高系统的运行效率方面会起到积极的促进作用。

参考文献：

[1] G De, Micheli and D. Ku. A System for High-level Synthesis[C]// Proc of 25th ACM/IEEE Design Automation Conf, 1988:483-488.

[2] Hallberg J, Peng Z. Synthesis under Local Timing Constraints in the CAMAD High-Level Synthesis System[C]// Proc of IEEE EUROMICRO'95, 1995:150-166.

[3] G F. Marchioro, J. M. Daveau, T. B. Transformational Partitioning for Codesign. IEE Proceedings on Computers and Digital Techniques, Vol. 145 No. 3, May 1998, 181-195.

[4] Saul J M. Programming Research Group. Hardware/Software Codesign for FPGA-Based Systems. IEEE Proceedings of the 32nd Hawaii International Conference on System Sciences, 2000, 3040-3046.

[5] Sobha Sankaran, Dr. Roger L. Haggard. A Convenient Methodology for Efficient Translation of C to VHDL. IEEE, 2001, 203-207.

[6] Jim Stevens, Fabrice Baijot The Hybrid Threads Compiler. Last accessed February 20, http://wiki. ittc. ku. edu/hybridthread/Main Page, 2007.

(上接第 35 页)

这样就完成了 2 处理器 SMP 系统的 LINPACK 测试，浮点效率为：

$$\eta = \frac{8.608}{11.2} = 76.86\%$$

这个实验中处理器只有两个，P×Q 配置方式只有两种，广播方式能够选择的配置方式少，不存在跨越网卡的通信。所有的这些都限制了计时系统作用的发挥，使得某些数据不够直观，甚至一些功能得不到表现。比如计时系统可以用于选择广播算法；当处理器数目较多的时候，计时系统对 P×Q 的确定也会有更好的指导作用；计时系统在确定互连网络结构的时候也很有用途。所以在大机群系统中，计时系统有着很大的潜力。

6 结束语

本文通过对计时系统进行改进，提出了一种基于计时系统的 LINPACK 测试方法。这种测试方法可以提取详细的时间参数，更好地引导 LINPACK 的快速测试。文中分析了这种测试方法的一般步骤，并且在一个测试平台上进行了实验。由于测试平台的局限性，很多计时系统的优势没能完全展现。但是总体来说，这种测试方法还是非常有效的。

参考文献：

[1] Xiao Mingwang, Xu Jian, Che Yonggang, Wang Zhenghua. HPL Benchmark and Analysis of a Real High Performance PC Cluster. Application Research of Computers, 2004, Vol21 (9): P183-187.
(肖明旺，许坚，车永刚，王正华. 一个实用高性能 PC 集群的 Linpack 测试与分析. 计算机应用研究，2004, Vol21(9): P183-187.

[2] Jack J. Dongarra, J. Bunch, Cleve Moler, G. W. Stewart. LINPACK User's Guide. USA:SIAM, 1979.

[3] Luo Shuihua, Yang Guangwen, Zhanglinbo, Shi Wei, Zheng Weimin. Analysis of LinPack Performance Test on Parallel Cluster System. Journal on Numerical Methods and Computer Applications, 2003, Vol(4).
(罗水华，杨广文，张林波，石威，郑纬民. 并行集群系统的 Linpack 性能测试分析. 数值计算与计算机应用,2003, vol(4).

[4] A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary. HPL-A Portable Implementation of the High-Performance Linpack Benchmark for Distribured Memory Computers. Available at http://www. netlib. org/benchmark/hpl ,Jan 20,2004.

[5] Jiang Xiaoling, Ren Guolin. The Fast Linpack Test Technique Based on IBM1350 Cluster System. Computer Technology and Development, 2007, Vol17(3): P65-68.
(姜晓玲，任国林. 基于 IBM1350 机群的 Linpack 快速测试. 计算机技术与发展，2007, Vol17(3): P65-68.)

[6] Zhang Wenli, Chen Mingyu, Fan Jianping. Emulation and Forecast of HPL Test Performance. Journal of Computer Research and Development, 2006, Vol43(3):P557-562.
(张文力，陈明宇，樊建平. HPL 测试性能仿真与预测. 计算机研究与发展，2006, Vol43(3):P557-562.

[7] Peng Wang, George Turner, Daniel A. Lauer, Matthew Allen, Stephen Simms, David Hart, Mary Papakhian, Craig A. Stewart. LINPACK Performance on a Geographically Distributed Linux Cluster . IEEE, 2004, 18th International Volume. SI Zhaokai, born in 1984, Graduate Student, Assistant Engineer. He is interested in the verification of VLSI.