湖南大學

HUNAN UNIVERSITY

计算机设计

学生姓名	周珍冉
学生学号	201708010610
专业班级	智能 1702
指导老师	吴强
完成日期	2019.12.1

Linpack 标准测试程序及分析

一、 Linpack 介绍

LINPACK 是线性系统软件包(Linear system package) 的缩写， 主要开始于 1974 年 4 月， 美国 Argonne 国家实验室应用数学所主任 Jim Pool， 在一系列非正式的讨论会中评估，建立一套专门解线性系统问题之数学软件的可能性。LINPACK 主要的特色是:

- 率先开创了力学 (Mechanics) 分析软件的制作。
- 建立了将来数学软件比较的标准。
- 提供软件链接库， 允许使用者加以修正以便处理特殊问题， (当然程序名称必须改写，并应注明修改之处，以尊重原作者，并避免他人误用。)
- 兼顾了对各计算机系统的通用性， 并提供高效率的运算。

至目前为止， LINPACK 还是广泛地应用于解各种数学和工程问题。 也由于它高效率的运算， 使得其它几种数学软件例如 IMSL、 MATLAB 纷纷加以引用来处理矩阵问题， 所以足见其在科学计算上有举足轻重的地位。

二、 Linpack 性能测试

Linpack 现在在国际上已经成为最流行的用于测试高性能计算机系统浮点性能的 benchmark。通过利用高性能计算机，用高斯消元法求解 N 元一次稠密线性代数方程组的测试，评价高性能计算机的浮点性能。

Linpack 测试包括三类——Linpack100、Linpack1000 和 HPL。

Linpack100 求解规模为 100 阶的稠密线性代数方程组，它只允许采用编译

优化选项进行优化，不得更改代码，甚至代码中的注释也不得修改。

Linpack1000 要求求解规模为 1000 阶的线性代数方程组，达到指定的精度要求，可以在不改变计算量的前提下做算法和代码上做优化。

HPL 即 High Performance Linpack，也叫高度并行计算基准测试，它对数组大小 N 没有限制，求解问题的规模可以改变，除基本算法（计算量）不可改变外，可以采用其它任何优化方法。前两种测试运行规模较小，已不是很适合现代计算机的发展，因此**现在使用较多的测试标准为 HPL，而且阶次 N 也是 linpack 测试必须指明的参数。**

HPL 是针对现代并行计算机提出的测试方式。用户在不修改任意测试程序的基础上，可以调节问题规模大小 N(矩阵大小)、使用到的 CPU 数目、使用各种优化方法等来执行该测试程序，以获取最佳的性能。**HPL 采用高斯消元法求解线性方程组。当求解问题规模为 N 时，浮点运算次数为$(2/3 * N^3 - 2*N^2)$。**因此，只要给出问题规模 N，测得系统计算时间 T，峰值=计算量$(2/3 * N^3 - 2*N^2)$/计算时间 T，测试结果以浮点运算每秒（Flops）给出。

计算峰值

随着产品硬件的不断的升级，整个的计算能力也以数量级的速度提升。衡量计算机性能的一个重要指标就是计算峰值，例如浮点计算峰值，它是指计算机每秒钟能完成的浮点计算最大次数。包括理论浮点峰值和实测浮点峰值：

理论浮点峰值是该计算机理论上能达到的每秒钟能完成浮点计算最大次数，它主要是由 CPU 的主频决定的，

理论浮点峰值=CPU 主频×CPU 每个时钟周期执行浮点运算的次数×系统中 CPU 核心数目。

实测浮点峰值是指 Linpack 测试值，也就是说在这台机器上运行 Linpack 测试程序，通过各种调优方法得到的最优的测试结果。实际上在实际程序运行过程中，几乎不可能达到实测浮点峰值，更不用说达到理论浮点峰值了。这两个值只是作为衡量机器性能的一个指标，用来表明机器处理能力的一个标尺和潜能的度量。

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 Soft — ware(ATLAS)、GotoBLAS、Engineering and Scientific Subroutine Library(ESSL)、Intel Math Kernel Library(MKL)和 AMDCore 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。

编译器的选择也有很大关系。

ⅢHPL 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(PxQ 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。

三、 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，如(1)式所示：Ax=b (1)

其中，A=$(a_{ij})_{N*N}$ 且为非奇异矩阵，b=$(b1，b2，\ldots，bN)^T$，x=$(x1，x2，\ldots，xN)^T$，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：

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

式(2)中 $2N^3/3 + 3N^2/2$ 是浮点运算规模，T_{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 Faetoriza—tion、Left-looking LU Faetorization 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 NBX NB，同一列上的块组成一个 panel。HPL 实现的时候先对 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，PFACT 和 RP—FACT 均有 Right—looking LU Factorization、Left-looking LU Factorization 和 Crout-looking LU Factorization 三种实现形式。这些参数对 LIN—PACK 性能的影响不大。

HPL 的运行还需要 Message Passing Interface(MPI) 和 Basic Linear Algebra Subroutines(131, AS)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。

四、 Linpack 安装

一、 在安装之前需要准备一些软件。

(1) Linux 平台，最新稳定内核的 Linux 发行版最佳，可以选择 Red hat, Centos 等。

(2) MPICH2(并行计算的软件)，在 http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads 下载最新的源码包。

(3) Gotoblas，BLAS 库（Basic LinearAlgebra Subprograms）：执行向量和矩

阵运算的子程序集合，选择公认性能最好的 Gotoblas，最新版到 http://www.tacc.utexas.edu/tacc-projects/下载，需要注册。

(4) HPL，linpack 测试的软件，可在 http://www.netlib.org/benchmark/hpl/ 下载最新版本。

二、 安装方法

1. 安装 MPICH2

1) 解压软件包

tar zxvf mpich2-1.1.1p1.tar.gz cd mpich2-1.1.1p1

指定目录编译 ./configure--prefix=/root/linpack/mpi --with-pm=smpd --enable-f77

make

make install

2) 配置环境变量

vim~/.bashrc

PATH="$PATH:/usr/local/mpi/bin"

source .bashrc 3、

测试环境变量

which smpd

whichmpiexec

3) 修改/root/.mpd.conf

secretword=myword

chmod 600 /root/.mpd.conf

4) 测试 mpich2 的进程 smpd 是否启动

[root@LG01 linpack]# which smpd

/root/linpack/mpi/bin/smpd

[root@LG01 linpack]# smpd –s

[root@LG01 linpack]# ps -ef | grep smpd

测试 mpi 是否启动

[root@LG01 linpack]#mpiexec -n 1 hostname

2. 进入 Linux 系统，使用 root 用户，在/root 下建立 Linpack 文件夹，解压下载的 Gotoblas 和 HPL 文件到 Linpack 文件夹下，改名为 Gotoblas 和 hpl。

#tar -xzvf GotoBLAS2-1.13_bsd.tar.gz

#mv GotoBLAS2-1.13 ~/linpack/Gotoblas

#tar xzvf hpl-2.0.tar.gz

#mv hpl-2.0 ~/linpack/hpl

3. 安装 Gotoblas

进入 Gotoblas 文件夹，执行./ quickbuild.64bit 进入快速安装。安装完成后在该目录下会生成 libgoto2.a 和 libgoto2.so 两个文件。

4. 安装 HPL

进入 hpl 文件夹从 setu 文件夹下提取与平台相近的 Make.Linux_PII_FBLAS 文件，该文件代表 Linux 操作系统，PII 平台，采用 FBLAS 库。

编辑刚刚复制的文件，根据说明修改各个选项，使之符合自己的系统，Intel xeon 平台，mpich2 安装目录为/usr/local/mipch2，hpl 和 gotoblas 安装目录为/root/LinPac 的配置文件 Make.Linux_xeon 需要修改.

修改好后编译:

#make arch=Linux_xeon

编译正常，在 hpl/bin/Linux_xeon 目录下就会生成两个文件 HPL.dat 和 xhpl。HPL.dat 文件是 LinPack 测试的优化配置文件，xhpl 为可执行程序。

五、 简单测试

进入 hpl/bin/Linux_xeon 目录，启动 mpd 服务，在终端运行命令#mpirun -np 4 xhpl 启动测试。

六、 Linpack 优化方法

在 LinPack 测试的理论分析工作中，需要解决矩阵的规模、矩阵分块、进程的映射、L 通信、U 通信、负载均衡等问题。而 LinPack 的测试结果受多方面因素的影响，包括与算法 相关的参数设置、CPU 的架构数量和效率、内存容量、互联网络的通信性能、系统规模等。 这些因素共同决定了系统 LinPack 的测试的最终结果。

通过分析系统默认的 HPL.dat 的配置情况来解析其测试原理，修改该文件的参数达到优化的目的。

1：HPLinpack benchmark input file

2：Innovative Computing Laboratory, University of Tennessee

注释说明行，不做修改

3：HPL.out output file name (if any)

4：6 device out (6=stdout,7=stderr,file)

3、4 行说明输出结果的形式，通过设置此处来保存测试结果：

"device out"为"6"时，测试结果输出至标准输出（stdout）

"device out"为"7"时，测试结果输出至标准错误输出（stderr）

"device out"为其它值时，测试结果输出至第三行所指定的文件中

```
5：4      # of problems sizes (N)
```

选择矩阵数量，这里选择为 4，则为第 4 个矩阵

```
6：29 30 34 35    Ns
```

5、6 行说明求解矩阵的次数和大小 N：

矩阵的规模 N 越大，有效计算所占的比例也越大，系统浮点处理性能也就越高；但与此同时，矩阵规模 N 的增加会导致内存消耗量的增加，一旦系统实际内存空间不足，使用缓存、性能会大幅度降低。因此，对于一般系统而言，要尽量增大矩阵规模 N 的同时，又 要保证不使用系统缓存。因为操作系统本身需要占用一定的内存，除了矩阵（N×N）之外，HPL 还有其他的内存开销，另外通信也需要占用一些缓存。矩阵占用系统总内存的 80% 左右为最佳，即 N×N×8=系统总内存×80%。如 4GB 内存的计算过程 （内存请换算为 byte 计算）。

```
7：4      # of NBs
```

```
8：1 2 3 4          NBs
```

7、8 行说明求解矩阵分块的大小 NB：

为提高数据的局部性，从而提高整体性能，HPL 采用分块矩阵的算法。分块的大小对性 能有很大的影响，NB 的选择和软硬件许多因素密切相关。 NB 值的选择主要是通过实际测试得到最优值。但 NB 的选择上还是有一些规律可寻，如：NB 不可能太大或太小，一般在 256 以下；NB × 8 一定是 Cache line 的

倍数等。例如， 我们的 L2 cache 为 1024K, NB 就设置为 192 另外，NB 大小的选择还跟通信方式、矩阵规模、网络、处理器速度等有关系。一般通 过单节点或单 CPU 测试可以得到几个较好的 NB 值，但当系统规模增加、问题规模变大，有 些 NB 取值所得性能会下降。所以最好在小规模测试时选择 3 个左右性能不错的 NB，再通过大规模测试检验这些选择。

9：1 PMAP process mapping (0=Row-,1=Column-major)

9 行是选择处理器阵列是按列的排列方式还是按行的排列方式：

按 HPL 文档中介绍，按列的排列方式适用于节点数较多、每个节点内 CPU 数较少的系统；而按行的排列方式适用于节点数较少、每个节点内 CPU 数较多的大规模系统。在机群系统上，按列的排列方式的性能远好于按行的排列方式。

10：3 # of process grids (P x Q)

11：2 1 4 Ps

12：2 4 1 Qs

10 ~ 12 行说明二维处理器网格（P×Q），二维处理器网格（P×Q）的有以下几个要求：

P×Q =系统 CPU 数=进程数。一般来说一个进程对于一个 CPU 可以得到最佳性能。对于 Intel Xeon 来说，关闭超线程可以提高 HPL 性能。

P≤Q；一般来说，P 的值尽量取得小一点，因为列向通信量（通信次数和通信数据量）要远大于横向通信。

P = 2n，即 P 最好选择 2 的幂。HPL 中，L 分解的列向通信采用二元交换法（ Binary Exchange），当列向处理器个数 P 为 2 的幂时，性能最优。例如，当系统进程数为 4 的时候，P×Q 选择为 1×4 的效果要比选择 2×2 好一些。

13：16.0 threshold

13 行说明测试的精度：

这个值就是在做完线性方程组的求解以后，检测求解结果是否正确。若误差在这个值以内就是正确，否则错误。一般而言，若是求解错误，其误差非常大；若正确，则很小。没有必要修改此值

14：3 # of panel fact

15：0 1 2 PFACTs (0=left, 1=Crout, 2=Right)

16：2 # of recursive stopping criterium

17：2 4 NBMINs (>= 1)

18：1 # of panels in recursion

19：2 NDIVs

20：3 # of recursive panel fact.

21：0 1 2 RFACTs (0=left, 1=Crout, 2=Right)

14 ~ 21 行指明 L 分解的方式：

在消元过程中，zHPL 采用每次完成 NB 列的消元，然后更新后面的矩阵。这 NB 的消元就是 L 的分解。每次 L 的分解只在一列处理器中完成。

对每一个小矩阵作消元时，都有 3 种算法：L、R、C，分别代表 Left、Right 和 Crout。在 LU 分解中，具体的算法很多，测试经验，NDIVs 选择 2 比较理想，NBMINs4 或 8 都不错。 而对于 RFACTs 和 PFACTs，对性能的影响不大。

在 HPL 官方文档中，推荐的设置为：

```
1            # of panel fact
1            PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
4 8          NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
2            RFACTs (0=left, 1=Crout, 2=Right)
```

22：1 # of broadcast

23：0 BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)

22、23 行说明 L 的横向广播方式：

HPL 中提供了 6 种广播方式。其中，前 4 种适合于快速网络；后两种采用将数据切割后传送的方式，主要适合于速度较慢的网络。目前，机群系统一般采用千兆以太网甚至光纤等高速网络，所以一般不采用后两种方式。

一般来说，在小规模系统中，选择 0 或 1；对于大规模系统，选择 3。

推荐的配置为：

```
2            # of broadcast
1 3          BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
```

24：1 # of lookahead depth

25：0 DEPTHs (>=0)

24、25 行说明横向通信的通信深度：

这依赖于机器的配置和问题规模的大小。

推荐配置为：

```
2            # of lookahead depth
0 1          DEPTHs (>=0)
```

26：0 SWAP (0=bin-exch,1=long,2=mix)

27：32 swapping threshold

26、27 行说明 U 的广播算法：

U 的广播为列向广播，HPL 提供了 3 种 U 的广播算法：二元交换（Binary Exchange）法、Long 法和二者混合法。SWAP="0"，采用二元交换法；SWAP="1"，采用 Long 法；SWAP="2"，采用混合法。

推荐配置为：

```
2            SWAP (0=bin-exch,1=long,2=mix)
60           swapping threshold
```

28：0 L1 in (0=transposed,1=no-transposed) form

29：0 U in (0=transposed,1=no-transposed) form

28、29 行分别说明 L 和 U 的数据存放格式。若选择"transposed"，则采用按列 存放，否则按行存放。

推荐配置为：

```
0            L1 in (0=transposed,1=no-transposed) form
0            U    in (0=transposed,1=no-transposed) form
```

30：1 Equilibration (0=no,1=yes)

30 行主要在回代中使用，一般使用其默认值。

31：8 memory alignment in double (> 0)

31 行的值主要为内存地址对齐而设置，用于在内存分配中对齐地址。出于安全考虑，可以选择 8。

七、 总结心得

本次报告很可惜，因为安装教程并不是很理解而且 Linpack 介绍是用于对高性能计算机的测试，我觉得我的电脑并不符合他的测试条件，所以我并没有进行实际的测试来进行分析。只能通过网上的资料通过分析其代码以及原理来了解其

运行过程、作用。按照我的理解，Linpack 就是通过对高性能计算机采用高斯消元求解 N 元一次稠密线性代数方程组的测试，来评价高性能计算机的浮点性能。简单地说就是以计算线性方程的时间来评价性能。