Linpack 标准测试程序和分析

一、Linpack 概述

Linpack 是国际上使用最广泛的测试高性能计算机系统浮点性能的基准测试。通过对高性能计算机采用高斯消元法求解一元 N 次稠密线性代数方程组的测试，评价高性能计算机的浮点计算性能。Linpack 的结果按每秒浮点运算次数（flops）表示。

很多人把用 Linpack 基准测试出的最高性能指标作为衡量机器性能的标准之一。这个数字可以作为对系统峰值性能的一个修正。通过测试求解不同问题规模的实际得分，我们可以得到达到最佳性能的问题规模，而这些数字与理论峰值性能一起列在 TOP500 列表中。

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 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。

注：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。

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(矩阵大小)、使用 CPU 数目、使用各种优化方法等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。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。

计算机计算峰值简介：衡量计算机性能的一个重要指标就是计算峰值或者浮点计算峰值，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。

计算公式如下：**理论浮点峰值＝CPU 主频×CPU 每个时钟周期执行浮点运算次数×CPU 数量**。

二、Linpack 相关组件安装

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，

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 Linear Algebra Subprograms）是执行向量和矩阵运算的子程序集合，这里我们选择公认性能最好的 Gotoblas

最新版可到 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。

安装步骤：

(1)安装 MPICH2，并配置好环境变量

(2)进入 Linux 系统，建议使用 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 xvf  GotoBLAS-*.tar.gz
#mv GotoBLAS-* ~/linpack/Gotoblas
#tar xvf hpl-*.tar.gz
#mv  hpl-* ~/linpack/hpl
```

(3)安装 Gotoblas

进入 Gotoblas 文件夹，在终端下执行./ quickbuild.64bit（如果你是 32 位系统，则执行./ quickbuild.31bit）进行快速安装，当然，你也可以依据 README 里的介绍自定义安装。

如果安装正常，在本目录下就会生成 libgoto2.a 和 libgoto2.so 两个文件。

(4)安装 HPL

进入 hpl 文件夹从 setup 文件夹下提取与自己平台相近的 Make.<arch>文件，复制到 hpl 文件夹内，比如我们的平台为 Intel xeon，所以就选择了 Make.Linux_PII_FBLAS，它代表 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，比如我们系统的详细情况为，Intel xeon 平台，mpich2 安装目录为 /usr/local/mipch2，hpl 和 gotoblas 安装目录为/root/linpack

注：Make 文件中需要修改的变量有：

ARCH： 必须与文件名 Make.<arch>中的<arch>一致

TOPdir：指明 hpl 程序所在的目录

MPdir： MPI 所在的目录

MPlib： MPI 库文件

LAdir： BLAS 库或 VSIPL 库所在的目录

LAinc、LAlib：BLAS 库或 VSIPL 库头文件、库文件

HPL_OPTS：包含采用什么库、是否打印详细的时间、是否在 L 广播之前拷贝 L。若采用 FLBAS 库则置为空，采用 CBLAS 库为“-DHPL_CALL_CBLAS”，采用 VSIPL 为“-DHPL_CALL_VSIPL”

“-DHPL_DETAILED_TIMING”为打印每一步所需的时间，缺省不打印

“-DHPL_COPY_L”为在 L 广播之前拷贝 L，缺省不拷贝（这一选项对性能影响不是很大）

CC： C 语言编译器

CCFLAGS：C 编译选项

LINKER：Fortran 77 编译器

LINKFLAGS：Fortran 77 编译选项(Fortran 77 语言只有在采用 Fortran 库是才需要)

三、运行

运行 hpl 之前，需要修改配置文件 hpl.dat(在 hpl/<arch>/bin 目录下)，次配置文件每一项代表的意思在后续说明。

HPL 的运行方式和 MPI 密切相关，不同的 MPI 在运行方面有一定的差别。对于 MPICH 来说主要有两种运行方法。

(1) 在 hpl/<arch>/bin 目录下执行：mpirun －np <N> xhpl。这种运行方式读取$(MPICH 安装目录)/share/machines.LINUX 配置文件

(2)在 hpl/<arch>/bin 目录下执行：mpirun －p4pg <p4file> xhpl。这种运行方式需要自己编写配置文件<p4file>，以指定每个进程在哪个节点上运行

MPICH 要求至少有一个 MPI 进程在递交任务的节点上运行，但 GM(MPI for Myrinet)、Infi－MPI(MPI for Infiniband)、ScaMPI(MPI for SCI)、BCL 等 MPI 来说，没有这个要求。

对于 GM 来说，可以采用 mpirun －machinefile <machinefile> -np <N> xhpl。这也是很多 MPI 所支持的一种运行方式，这种运行方式也需要自己编写<machinefile>以指定每个进程在哪个节点上运行

测试结果输出到指定文件中(在配置文件 hpl.dat 中定义)，缺省文件名为 HPL.out。

----------HPL.dat----------

(1)第 1、2 行为注释说明行，不需要作修改

(2)第 3、4 行说明输出结果文件的形式

“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

(3)第 5、6 行说明求解矩阵的大小 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。

考虑到操作系统本身需要占用一定的内存，除了矩阵 A（N×N）之外，HPL 还有其它的内存开销，另外通信也需要占用一些缓存（具体占用的大小视不同的 MPI 而定）。一般来说，矩阵 A 占用系统总内存的 80％左右为最佳，即 N×N×8＝系统总内存×80％。

这只是一个参考值，具体 N 最优选择还跟实际的软硬件环境密切相关。当整个系统规模较小、节点数较少、每个节点的内存较大时，N 可以选择大一点。当整个系统规模较大、节点数较多、每个节点的内存较小时是，N 可以选择大一点。

(4)第 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数密切相关。

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 的倍数等等。

(5)第 9 行是 HPL 1.0a 的新增项，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。在 HPL 1.0 中，其缺省方式就是按列的排列方式。

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 数较少的瘦系统；而按行的排列方式适用于节点数较少、每个节点内 CPU 数较多的胖系统。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。

(6)10～12 行说明二维处理器网格（P×Q）

二维处理器网格（P×Q）的要遵循以下几个要求：P×Q＝进程数。这是 HPL 的硬性规定；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，这是一个测试经验值，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.等等这些，需要测试者多摸索。

四、查看运算结果

HPL 允许一次顺序做多个不同配置测试，所以结果输出文件(缺省文件名为 HPL.out)可能同时有多项测试结果。

在文件的第一部分为配置文件 hpl.dat 的配置。在下面的部分

使用基准测试一般需要和收集的信息包括：

R: 它是系统的最大的理论峰值性能，按 GFLOPS 表示。如 10 个 Pentium III CPU 的 Rpeak 值。

N: 给出有最高 GFLOPS 值的矩阵规模或问题规模。正如拇指规则，对于最好的性能，此数一般不高于总内存的 80%。

Rmax: 在 Nmax 规定的问题规模下，达到的最大 GFLOPS。

NB: 对于数据分配和计算粒度，HPL 使用的块尺度 NB。小心选择 NB 尺度。从数据分配的角度看，最小的 NB 应是理想的；但太小的 NB 值也可以限制计算性能。虽然最好值取决于系统的计算/通信性能比，但有代表性的良好块规模是 32 到 256 个间隔。

类似运算结果输出如下：

```
BCAST  :   1ring
DEPTH  :      0
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual checks will be computed:
   1) ||Ax-b||_oo / ( eps * ||A||_1  * N        )
   2) ||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  )
   3) ||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo )
- The relative machine precision (eps) is taken to be           1.11022
- Computational tests pass if scaled residuals are less than

================================================================================
T/V                N    NB     P     Q               Time
--------------------------------------------------------------------------------
WR00L2L2       109000   168    16    16             438.02             1.9
--------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        0.0028926 ......
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0024547 ......
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0004369 ......
================================================================================
```