实验四 HPL 安装和性能测试

一、 HPL 与 High Performance Linpack （准备知识，自学）

目的与要求：使学生掌握 Linpack 和 HPL 的背景知识

主要内容：

1、Linpack 背景及内容

（1）背景介绍

LINPACK 全名 Linear Equations Package，是近年来较为常用的一种计算机系统性能测试的线性方程程序包，内容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。它最早由来自 Tennessee 大学的超级计算专家 Jack Dongarra 提出。程序用 FORTRAN 编写，在此基础上还有 C，JAVA 等版本。Linpack 使用线性代数方程组，利用选主元高斯消去法在分布式内存计算机上按双精度(64 bits)算法，测量求解稠密线性方程组所需的时间。Linpack 的结果按每秒浮点运算次数（flops）表示。第一个 Linpack 测试报告出现在 1979 年的 Linpack 用户手册上，最初 LINPACK 包并不是要制订一个测试计算机性能的统一标准，而是提供了一些很常用的计算方法的实现程序，但是由于这一程序包被广泛使用，就为通过 Linpack 例程来比较不同计算机的性能提供了可能，从而发展出一套完整的 Linpack 测试标准。

（2）测试标准的内容

LINPACK 标准可以解决的问题有：

1) 各种矩阵分解(Matrix factorization)，如 LU 分解，Cholesky 分解， Schur，Gauss 分解，SVD 分解，QR 分解，generalized Schur 分解等

2) 矢量运算（Vector operation），如 Copy，Add，scalar multiple，Interchange

3) 存储模式（Storage Modes），如 full，banded，symmetric

Linpack 原始版本的问题规模为 100×100 的矩阵，目前的 Linpack 测试分成三个层次的问题规模和优化选择：

---- 100×100 的矩阵

在该测试中，不允许对 Linpack 测试程序进行任何修改，哪怕是注释行。 所有的优化工作只能在编译器里完成。

---- 1000×1000 的矩阵

该测试也叫“面向峰值性能的测试”，在该测试中，要求有所放宽，允许对算法和软件进行修改或替换，并尽量利用系统的硬件特点，以达到尽可能高的性能。但是所有的优化都必须保持和标准算法如高斯消去法相同的相对精度，而且必须使用 Linpack 的主程序进行调用。测试者可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。

---- 针对大规模并行计算系统的测试，即 Highly Parallel Computing。

在这一测试中，问题规模限制被取消，针对现代的并行计算机，要求最宽松，即用户可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（必须基于高斯消去法）来行该测试程序，寻求最佳的测试结果。

其实 HPL(主页：http://www.netlib.org/benchmark/hpl/algorithm.html)就是这样的一个大规模高度并行的测试版本，也是当今用来进行世界前 500 超级计算机排名的公用测试标准（网址：http://www.top500.org/）。

目前，用 Linpack 基准测试出的最高性能指标已经成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计算速度值，我们不仅可以得到达到最佳性能的问题规模和性能测试数据。

2、 HPL 基准测试

（1）概述

HPL，即 High Performance Linpack，目前已经成为国际标准的 Linpack 基准测试程序，其 1.0 版于 2000 年 9 月发布，是第一个标准的公开版本并行 Linpack 测试软件包，一般用于全世界 TOP500 超级计算机上的并行超级计算机排名。HPL 测试标准的用户自由度要大很多，使用者可以选择矩阵的规模，分块大小，分解方法等等一系列的各种参数，都是按需要更改的。

HPL 软件包需要在配备了 MPI 环境下的系统中才能运行，还需要底层有线性代数子程序包 BLAS 的支持（或者有另一种向量信号图像处理库 VSIPL 也可）。

HPL 软件包不仅提供了完整的 Linpack 测试程序，还进行了全面细致的计时工作，最后可以得到求解的精确性和计算所花费的总时间。该软件在系统上所能达到的最佳性能值适合很多因素有关的。

（2）主算法

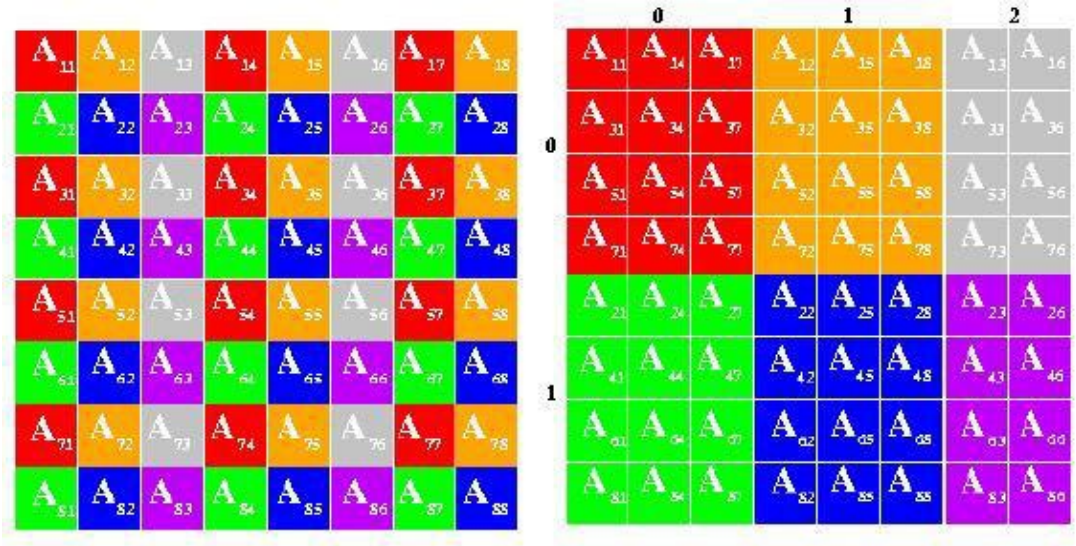
该软件包是用来求一个 N 维的线性方程组 A x = b 的解，首先通过选局部列主元的方法对 N x (N+1)的[A b]系数矩阵进行 LU 分解成如下形式：

[A b]=[[L,U] y]

由于下三角矩阵 L 因子所作的变换在分解的过程中也逐步应用到 b 上，所以最后方程组的解 x 就可以由转化为求解上三角矩阵 U 作为系数矩阵的线性方程组 U x = y 从而得到。

为了保证良好的负载平衡和算法的可扩展性，数据是以循环块的方式分布到一个 P x Q 的由所有进程组成的 2 维网格中。N x (N+1)的系数矩阵首先在逻辑上被分成一个个 Nb x Nb 大小的数据块，然后循环的分配到 P x Q 进程网格上去处理。这个分配的工作在矩阵的行、列两个方向同时进行。如下图所示：

（左图为整个矩阵被分成小块以后，右图为各小块分配到各个进程后的情况）



在 LU 分解的主循环中使用的是向右看（right-looking）的分解法，也就是说，在每次迭代过程对一个 nb 列的板块分解，然后剩余的子矩阵被更新。因此，

我们也可以注意到这里的板块在逻辑上被分为 nb 大小的子块，而这个 Nb 恰好就是前面数据分布中使用的 Nb。

在前面所提到的 N，Nb，P，Q 都是可以根据集群的具体配置和用户需要而随时修改的，也是 HPL 测试中十分关键和重要的几个参数。详细内容请看后面。

二、HPL 的安装及使用

以下实验步骤已在 Windows 11+VMware Workstation17+CentOS 7 下验证通过

目的与要求：使学生掌握 HPL 的安装及运行。

主要软件包：

hpl-2.1.tar.gz

GotoBLAS2-1.13.tar.gz

openmpi-1.6.5.tar.gz

主要内容：

1. HPL 的安装

（1）安装 GotoBLAS

出于提高性能的因数，选择 GotoBLAS，作为 HPL 调用的底层线性代数子程序包。目前最新版本为 GotoBLAS2，下载 GotoBLAS2-1.13.tar.gz

（网址 https://www.tacc.utexas.edu/tacc-projects/GotoBLAS2）

执行步骤如下：

在 usr/local/mathlib/goto 下解压：

$ tar -zxvf GotoBLAS2-1.13.tar.gz

$ cd GotoBLAS2

$ make （TARGET=NEHALEM）

注：可直接输入#make 既可 也可自己选择参数 类似

$ make CC=gcc BINARY=64 TARGET=NEHALEM

依次是编译器 库的位数和 cpu 的类型（architecture），具体可选择的参数可从 GotoBLAS2 的目录下 02QuickInstall.txt 的查找

cpu 的类型（architecture）需要根据 cpu 的型号自行网上查找，可使用命令$ cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq –c 查看 cpu 的型号 可从目录下的 getarch.c 查看目前支持的 architecture，新一点的 intel cpu 需要指定为 TARGET=NEHALEM

编译出现错误请先输入$ gmake clean 再重新开始

如果成功则显示类似如下图:

```
GotoBLAS build complete.

 OS               ... Linux
 Architecture     ... x86_64
 BINARY           ... 64bit
 C compiler       ... GCC  (command line : gcc)
 Fortran compiler ... INTEL  (command line : ifort)
 Library Name     ... libgoto2_nehalemp-r1.13.a (Multi threaded; Max num-thread
```

（3）安装 openmpi

（4）安装 HPL

下载 hpl-2.1.tar.gz (网址：http://www.netlib.org/benchmark/hpl/hpl-2.1.tar.gz)

在用户目录下解压：

$ tar –zxvf hpl-2.1.tar.gz

$ cd hpl-2.1

根据机器的情况复制 Makefile 模板：

```
[asc14@cu02 setup]$ ls
Make.FreeBSD_PIV_CBLAS   Make.Linux_PII_CBLAS     Make.PWRPC_FBLAS
make_generic             Make.Linux_PII_CBLAS_gm  Make.SUN4SOL2_FBLAS
Make.HPUX_FBLAS          Make.Linux_PII_FBLAS     Make.SUN4SOL2-g_FBLAS
Make.I860_FBLAS          Make.Linux_PII_FBLAS_gm  Make.SUN4SOL2-g_VSIPL
Make.IRIX_FBLAS          Make.Linux_PII_VSIPL     Make.T3E_FBLAS
Make.Linux_ATHLON_CBLAS  Make.Linux_PII_VSIPL_gm  Make.Tru64_FBLAS
Make.Linux_ATHLON_FBLAS  Make.PWR2_FBLAS          Make.Tru64_FBLAS_elan
Make.Linux_ATHLON_VSIPL  Make.PWR3_FBLAS          Make.UNKNOWN.in
```

$ cp setup/Make.Linux_PII_CBLAS Make.Linux

$ vi Make.Linux

如下根据具体情况修改 Make.Linux（附件中有修改好的，但是需要修改 MPI 和 GotoBLAS2 的安装路径）：

ARCH = Linux

TOPdir = /home/yanxinyu/ hpl-2.1

MPdir = /opt/mpich-3.4　　实验 3 安装 MPI 时选择的位置

MPinc = $(MPdir)/include

MPlib = -L$(MPdir)/lib

LAdir = /home/yanxinyu/GotoBLAS2　　GotoBLAS2 的安装目录

LAlib = $(LAdir)/ libgoto2_nehalemp-r1.13.a

HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) -I$(LAinc) -I$(MPinc)

CC = /opt/mpich-3.4/bin/mpicc　**可用#which mpicc 指令查看 也可直接填 mpicc**

CCNOOPT = $(HPL_DEFS)

CCFLAGS = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops

LINKER = 同 CC

LINKFLAGS =同 CCFLAGS

（5）编译

在 HPL 安装目录下运行

$ make arch=Linux

注：编译成功会在 bin/linux 目录下有 HPL.dat 和 xhpl 文件，如果失败,重新编译前加入#make arch=Linux clean_arch_all。如果使用 icc ，使用命令#which mpiicc 查看后 修改 Make.Linux

其他机器可以 scp 发送编译好的 GotoBLAS 和 HPL，免去安装，命令：scp -r GotoBLAS2 hpl-2.1 slave1:/home/yanxinyu/

2．HPL 的运行

（1）进入 ～/安装 hpl 的目录/bin/Linux 目录

$ cd /home/用户目录/hpl 安装目录/bin/Linux

(2)准备节点文件

$ **vi nodes**

内容类似：

master

slave1

slave2

……

(3)修改 HPL.dat，设置运算规模和进程数等

$ vi HPL.dat（见附件）

(4)运行

$ mpirun -np 最大进程数 (多节点还需要增加： -machinefile nodes) ./xhpl > HPLResult.txt (结果写到文件中方便查看)

成功运行后出现类似如下图:

```
The following parameter values will be used:

N      :    1200
NB     :     192
PMAP   : Row-major process mapping
P      :       1
Q      :       4
PFACT  :    Left
NBMIN  :       2
NDIV   :       2
RFACT  :    Left
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 check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be             1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00L2L2        1200   192     1     4               0.81             1.431e+00
HPL_pdgesv() start time Fri Jan 24 03:11:45 2014

HPL_pdgesv() end time   Fri Jan 24 03:11:46 2014

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0074933 ...... PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
[asc14@cu02 intel64]$
```

三、Performance Tuning

目的与要求：使学生了解 Performance Tuning 各个环节，掌握 Performance Tuning 技术。

重点：修改 HPL.dat 设置运行参数

1．选择优化的 BLAS 库

HPL 软件包需要在配备了 MPI 环境下的系统中才能运行，还需要底层有线性代数子程序包 BLAS 的支持（或者有另一种向量信号图像处理库 VSIPL 也可）。

虽然同样是对 BLAS 运算功能的实现，但具体不同的实现方法所生成的库，对 HPL 测试结果也有着很大的影响。

采用 GOTO 开发的 BLAS 库要效果要好得多，尤其是在问题规模增大，计算量增多以后，效果就更明显。另外，它还有一个用 pthread 库编译的版本，可以用来在 SMP 机群上，使节点内部的通信方式不用 MPI 而采用 Pthread，效率会更高一些。对于 GOTO 在提高节点间的通信性能的基础上，提高每个节点自身的效率也是优化集群性能的一种手段。处理器有自己的峰值速度，由它的主频和每周期的浮点运算次数所决定，对处理器效率的优化就是要使实际运算速度能够尽可能的接近峰值速度。

针对处理器的硬件特点而进行相应的优化会取得很显著的效果。GotoBLAS 库的设计思想，主要是基于如下几个基础方面的观察和考虑：

由浮点运算单元进行的浮点运算速度与从第二级 cache 经过的浮点数据流的速度相比，相对而言这个比例是很小的。因此，启用第二级 cache 处理这些数据流是一笔不必要的开销

很大一部分数据流的开销来自于阻塞 CPU 的 TLB (Translation Look-aside Buffers 转换表缓冲区) 不命中率，这种不命中很多是可以避免的，不能避免的也可以分摊到大量的计算当中去。

可见 GotoBLAS 之所以大大优化了性能，是由于它有效的针对 Pentium4 处理器的 SSE2 技术，提高了二级缓存的利用率，大大降低了 TLB 的不命中率，减少了不必要的开销。作为 HPL 需要频繁调用的底层库，BLAS 自身的高效率对 Linpack 测试的性能指标产生了显著的影响。对一台计算机来讲，内存的性能对整个机器的影响是仅次于 CPU 的。所以要提高节点的运算效率，只一味的专注于 CPU 速度是不够的。对于一个 SMP 节点，两个处理器是共享一块物理内存的，因此处理好节点内部的并行才能够提高节点自身的性能。在节点内部采用 pthread 库的进程通信方式代替 MPI 消息传递方式，能够使 SMP 节点的实际运算速度更接近于峰值速度。这虽然是对 SMP 节点性能的软件上的优化，但归根结底是从其硬件特性出发，采用合适的进程通信方式，更合理的利用内存空间，减少访问冲突，使内存的使用性能提高，从而也提高了整个节点的性能。

2．修改 HPL.dat 设置运行参数

在 HPL 测试中，使用的参数选择与测试的结果有很大的关系。HPL 中参数的设定是通过从一个配置文件 HPL.dat 中读取的，所以在测试前要改写 HPL.dat 文件，设置需要使用的各种参数，然后再开始运行测试程序。配置文件内容的结构如下：

```
HPLinpack benchmark input file                          //文件头，说明
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)                  //如果使用文件保留输出结果，设定文件名
6            device out (6=stdout,7=stderr,file)        //输出方式选择（stdout,stderr 或文件）
2            # of problems sizes (N)                   //指出要计算的矩阵规格有几种
1960  2048Ns                                           //每种规格分别的数值
2            # of NBs                                  //指出使用几种不同的分块大小
60 80        NBs                                       //分别指出每种大小的具体值
2            # of process grids (P x Q-l               //指出用几种进程组合方式
2   4         Ps                                       //每对 PQ 具体的值
```

```
2   1       Qs
16.0         threshold                          //余数的阈值
1            # of panel fact                    //用几种分解方法
1            PFACTs (0=left, 1=Crout, 2=Right)   //具体用哪种,0 left,1 crout,2 right
1            # of recursive stopping criterium   //几种停止递归的判断标准
4            NBMINs (>= 1)                       //具体的标准数值（须不小于 1）
1            # of panels in recursion             //递归中用几种分割法
2            NDIVs                              //这里用一种 NDIV 值为 2，即每次递归分成两块
1            # of recursive panel fact.          //用几种递归分解方法
2            RFACTs (0=left, 1=Crout, 2=Right)     //这里每种都用到（左，右，crout 分解）
1            # of broadcast                        //用几种广播方法
3            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)   //指定具体哪种（有 1-ring,1-ring
Modified,2-ring,2ring Modified,Long 以及 long-Modified）
1            # of lookahead depth     //用几种向前看的步数
1            DEPTHs (>=0)             //具体步数值（须大于等于 0）
2            SWAP (0=bin-exch,1=long,2=mix)  //哪种交换算法（bin-exchange，long 或者二者混合）
64           swapping threshold     //采用混合的交换算法时使用的阈值
0            L1 in (0=transposed,1=no-transposed) form     //L1 是否用转置形式
0            U  in (0=transposed,1=no-transposed) form    //U 是否用转置形式表示
1            Equilibration (0=no,1=yes)                 //是否采用平衡状态
8            memory alignment in double (> 0)     //指出程序运行时内存分配中的采用的对齐方式
```

要得到调试出高的性能，必须考虑内存大小，网络类型以及拓扑结构，调试上面的参数，直到得出最高性能。

本次实验需要对以下三组参数进行设置：

```
2            # of problems sizes (N)              //指出要计算的矩阵规格有几种
1960  2048Ns                                      //每种规格分别的数值
```

指出要计算的矩阵规格有 2 种，规格是 1960，2048

```
2            # of NBs                            //指出使用几种不同的分块大小
60 80        NBs                                  //分别指出每种大小的具体值
```

指出使用 2 种不同的分块大小，大小为 60，80

```
2            # of process grids (P x Q-l         //指出用几种进程组合方式
2   4         Ps                                 //每对 PQ 具体的值
2   1       Qs
```

指出用 2 种进程组合方式，分别为（p＝2，q=2） 和（p＝4，q=1）

注：p＝2，q=2 时需要的进程数是 p×q＝2×2＝4，运行时 mpirun 命令行中指定的进程数必须大于等于 4

以上 3 组每组有两种情况，组合后一共有 8 种情况，将得到 8 个性能测试值，经过不断的调试将会得出一个最大的性能值，这就是得到的最高性能值。

以下是其中一个性能测试值，规格为 2048，分块是 60，p＝2，q=2 时，运行时间为：56.14，运算速度为 0.8165 Gflops。PASSED 代表结果符合要求。

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
W13R2C4         2048    60     2     2              56.14              8.165e-01
```

```
-------------------------------------------------------------------------------
||Ax-b||_oo / ( eps * ||A||_1  * N        ) =        0.0175089 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_1  * ||x||_1  ) =        0.0035454 ..... PASSED
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo ) =        0.0007503 ..... PASSED
===============================================================================
```

3．编译器选择

（1）使用 gcc 编译

（2）使用 icc 编译

实验四 集群系统性能测试

一、实验步骤：

1. 计算计算机峰值速度

CPU 主频：查看/proc/cpuinfo 文件，将看见 cpu 的详细信息，其中 cpu MHz 是主频值

网上查找资料计算峰值速度

2. 性能测试

使用 gcc 编译器的情况下测试，并将最佳测试结果填写下面表格

进程个数	4	6	8	32
HPL 测试峰值速度（Gflops）				
效率				

参与运算主机名	进程数	N	NB	P	Q	Time	HPL 测试的 Gflops	效率
master	4							
master	6							
master	8							
master	32							
master, slave1	4							
master, slave1	6							
master, slave1	8							
master, slave1	32							
master, slave1,slave2	4							
master, slave1,slave2	6							
master, slave1,slave2	8							
master, slave1,slave2	32							

3. 完成上述测试后比较和分析上面的测试结果，特别是如何能够得到高的性能测试值

二、设计思考实验

1、 还有什么技术会影响测试结果，例如 sse2、超线程等，请设计实验。并详细书写实验的采用的库文件、Makefile 文件、测试结果、数据分析等。

2、 如果采用 Intel 编译器进行测试，比较测试结果