

实验 CPU 浮点计算性能测试

实验目的与要求：理解 CPU 性能测试的背景知识，学会 hpl 的使用。

1 准备知识

1.1 计算性能

衡量计算机性能的一个重要指标是浮点计算性能峰值 **FLOPS**，它是指计算机每秒钟能完成的浮点运算最大次数，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运算最大次数，它主要是由 CPU 的主频决定的。计算公式：**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。

1.2 Linpack 测试基准

Linpack 全名 Linear Equations Package，是计算机上的线性方程程序包，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浮点计算性能提供了可能，从而发展出一套完整的 Linpack 测试标准。

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，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(及 HPL)采用高斯消元法求解线性方程组。求解的问题规模为 N 时，浮点运算次数为 $(\frac{2}{3} * N^3 - 2 * N^2)$ 。因此，只要给出问题规模 N，测得系统计算时间 T，则：

$$\text{峰值} = \text{计算量}(\frac{2}{3} * N^3 - 2 * N^2) / \text{计算时间 } T \quad (\text{Flops})$$

1.3 三类 Linpack 测试

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。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。

HPL-----即 High Performance Linpack，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。

国际超算 TOP500 榜单就是根据计算机的 HPL 值来进行排名的。

1.4 HPL

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。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）来执行该测试程序。

1.5 CPU 的其它性能测试基准

● CPU 的整数运算性能-----Dhrystone

Dhrystone 是测量处理器运算能力的最常见基准程序之一，常用于处理器的整型运算性能的测量。程序是用 C 语言编写的，因此 C 编译器的编译效率对测试结果也有很大影响。

Dhrystone 是由 Reinhold P. Weicker 在 1984 年提出来的一个基准测试程序，其主要目的是测试处理器的整数运算和逻辑运算的性能。Dhrystone 首先用 Ada 语言发布，后来 Rick Richardson 为 Unix 开发了用 C 语言编写的 Version 1.1，这个版本也成功的推动了 Dhrystone 的广泛应用。

Dhrystone 标准的测试方法很简单，就是单位时间内跑了多少次 Dhrystone 程序，其指标单位为 DMIPS/MHz。MIPS 是 Million Instructions Per Second 的缩写，每秒处理的百万级的机器语言指令数。DMIPS 中的 D 是 Dhrystone 的缩写，它表示了在 Dhrystone 标准的测试方法下的 MIPS。

关于 DMIPS 有一个不得不注意的点，因为历史原因我们把在 VAX-11/780 机器上的测试结果 1757 Dhrystones/s 定义为 1 DMIPS，因此在其他平台测试到的每秒 Dhrystones 数应除以 1757，才是真正的 DMIPS 数值，故 DMIPS 其实表示的是一个相对值。

核心程序下载 http://www.roylongbottom.org.uk/classic_benchmarks.tar.gz

● T/CESA 1213,1214,1214-2022 通用计算 CPU 性能测试基准

中国团体标准，有中国电子工业标准化技术协会发布。

配套的“CPUBench”性能测试基准工具，由计算产品性能基准工作组设计开发，并在 2021 年世界计算机大会上发布，参与单位包括 CPU 厂商、整机厂商、用户单位、研究机构等 60 余家单位。

CPUBench 可以兼容国内外如龙芯、申威、飞腾、鲲鹏、海光、兆芯、合芯、AMD、intel 等 CPU 产品的指令集架构。工具下载请前往 www.cppb-wg.com 申请。

CPUBench 共包含四个测试套件：IntSingle、IntConcurrent、FloatSingle 和 FloatConcurrent，分别用于评估计算机系统的单核整型运算能力、多核整型运算能力、单核浮点运算能力和多核浮点运算能力。每个测试套件均可指定 typical 或 extreme 模式来运行，其中 typical 模式是基本优化下的性能测试，任何优化措施统一应用到所有的测试负载；extreme 模式可针对不同的测试负载采用不同的优化措施。

单核测试套件里每个测试负载的得分是一个比值，即该负载在参考机器上运行得到的参考时间除以被测系统运行该负载的时长，而后将测试套件所有负载的得分取几何平均，便是该套件的最终得分。

● Whetstone

Whetstone 是一套基准程序，用来测试 CPU 的处理性能，它的基本原则是：在程序编译后生成的机器指令中，各种指令出现的频率与统计数据中指令出现的频率相符合。

Whetstone 的程序比较简单，由 8 个模块组件，每个模块循环一定次数、执行一种基本运算。以下是每个模块的简单描述：

- 数组元素(浮点)，循环 12 次
- 数组参数(浮点)，循环 14 次
- 条件判断(if then else)，循环 345 次
- 整型计算，循环 210 次
- 三角函数(sin, cos, etc.)，循环 32 次

- 程序调用(浮点), 循环 899 次
- 数组操作(赋值), 循环 616 次
- 标准函数(exp, sqrt, etc.)循环 93 次

整个测试执行若干次, 第一次执行 100 次循环, 之后每次递增 10 次循环, 即第二次执行 110 次循环。测试结果是每组循环的平均时间和所有循环的平均 KWIPS 值(Thousands of Whetstone Instruction mixes Per Second)

● Dhrystone

Dhrystone 是测量处理器运算能力的最常见基准程序之一, 常用于处理器的整型运算性能的测量。程序是用 C 语言编写的, 因此 C 编译器的编译效率对测试结果也有很大影响。

Dhrystone 是由 Reinhold P. Weicker 在 1984 年提出来的一个基准测试程序, 其主要目的是测试处理器的 **整数运算和逻辑运算** 的性能。Dhrystone 首先用 Ada 语言发布, 后来 Rick Richardson 为 Unix 开发了用 C 语言编写的 Version 1.1, 这个版本也成功的推动了 Dhrystone 的广泛应用。

Dhrystone 标准的测试方法很简单, 就是单位时间内跑了多少次 Dhrystone 程序, 其指标单位为 DMIPS/MHz。MIPS 是 Million Instructions Per Second 的缩写, 每秒处理的百万级的机器语言指令数。DMIPS 中的 D 是 Dhrystone 的缩写, 它表示了 Dhrystone 标准的测试方法下的 MIPS。

关于 DMIPS 有一个不得不注意的点, 因为历史原因我们把在 VAX-11/780 机器上的测试结果 1757 Dhrystones/s 定义为 1 DMIPS, 因此在其他平台测试到的每秒 Dhrystones 数应除以 1757, 才是真正的 DMIPS 数值, 故 DMIPS 其实表示的是一个相对值。

源码获取: 核心程序下载 http://www.roylongbottom.org.uk/classic_benchmarks.tar.gz

缺陷:

- 它的代码与具有代表性的实际程序代码并不相同。
- Dhrystone 代码量过小, 在现代 CPU 中, 它能够被放进指令缓存中, 所以它并不能严格的测量取指性能。
- 它易受编译器影响。

● SPEC

标准性能评估公司 (SPEC, Standard Performance Evaluation Corporation) 成立于 1988 年, SPEC 从各种不同的应用场景中选出一些比较有代表性的程序, 称为基准套件 (BenchMark Suit)。SPEC 基准中最出名的是它的 CPU 套件, 用于测试 CPU 的吞吐量、Cache 和存储器的访问速度。

SPEC CPU2017 分别包含: 10 个 Integer rate, 10 个 Integer speed, 13 个 Floating Point rate, 10 个 Floating Point speed

- 在嵌入式领域, EEMBC (Embedded Microprocessor Benchmark Consortium) 基准常被使用, 其应用涵盖汽车、消费电子、通信等领域。

2 HPL 安装和测试

2.1 软件主页

<http://www.netlib.org/benchmark/hpl/algorithm.html>

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 VSIPPL 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是和很多因素有关的。

2.2 主算法

该软件包是用高斯消元法来求一个 N 维的线性方程组 $Ax = b$ 的解，使用 row partial pivoting（只在当前进行变换的列中选择主元，只需要进行行交换），并先求 A 的 LU 分解。

高斯消元法其实是让 $Ax=b$ 变成 $Ux=c$ ，例如：

$$Ax = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 7 \end{bmatrix}$$

$\mathbf{A} \quad \mathbf{x} \quad \mathbf{b}$

$$Ux = \begin{bmatrix} 2 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 1 \\ -4 \\ -4 \end{bmatrix}$$

$\mathbf{U} \quad \mathbf{x} \quad \mathbf{c}$

高斯消元法的时间复杂度： $O(N^3)$ 。如果系数矩阵为三角阵，则为 $O(N^2)$ 。

当待解决的问题中系数矩阵不变而右端项变化时，应首先对 A 进行 LU 分解，把原问题化为两步：

$$Ax = b \Leftrightarrow L(Ux) = b \Leftrightarrow \begin{cases} Ly = b \\ Ux = y \end{cases}$$

LU 分解的复杂度为 3 阶，但只需在预处理阶段进行分解一次；对于变化的 b，只要求两次三角形线性方程组，总体复杂度为 2 阶，求解效率显著提高。

高斯消元可以看做用一连串的消元矩阵 E 乘以 A，达到消元的目的，累积这些消元矩阵的逆矩阵可以得到 L：

$$E_{32}E_{31}E_{21}A = (E_{32}E_{31}E_{21})A = ZA = U$$

$$A = E_{21}^{-1}E_{31}^{-1}E_{32}^{-1}U$$

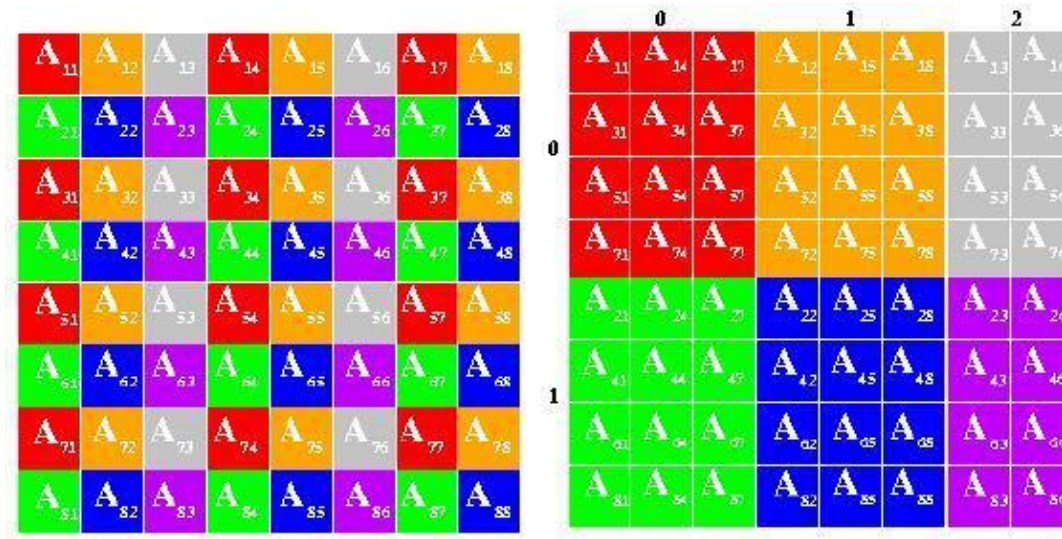
$$L = E_{21}^{-1}E_{31}^{-1}E_{32}^{-1}, \quad \text{so that} \quad A = LU$$

LU 分解过程总结：在消元的过程中，如果不存在主元为 0 的情况，我们可以把对矩阵 A 的高斯消元过程用矩阵的形式表示成 $A=LU$ ，其中 L 是一个下三角矩阵，L 的主对角线上的元素全是 1，主对角线下面(i,j)处的元素是消元过程中每一步所乘的倍数。U 是一个上三角矩阵，是高斯消元的结果，其主对角线上的元素是主元。对应上面的例子：

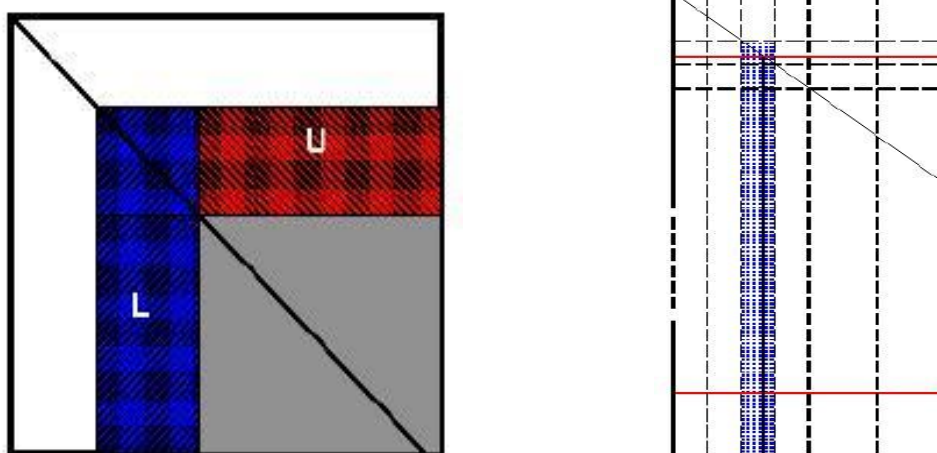
$$A = \begin{bmatrix} 2 & 1 & 1 \\ 4 & 1 & 0 \\ -2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -3 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 1 \\ 0 & -1 & -2 \\ 0 & 0 & -4 \end{bmatrix} = LU$$

LU 分解是唯一的！

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由所有进程组成的 $P * Q$ 的二维网格中。 $N * N$ 的系数矩阵首先在逻辑上被分成一个个 $Nb * Nb$ 大小的数据块，然后循环地分配到 $P * Q$ 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，如下图所示，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，可以注意到计算也在逻辑上被分为 Nb 大小的子块，与前面数据分块一致。在消元过程中，采用每次完成 NB 列的消元，然后更新后面的矩阵（其它列要用刚才的 NB 列的对角元素进行更新）。这 NB 的消元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。



$P \times Q = \text{系统 CPU 数} = \text{进程数}$ 。一般来说一个进程对于一个 CPU 可以得到最佳性能。取 $P \leq Q$ ，P 的值尽量小一点，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。 $P = 2^n$ 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， $P \times Q$ 选择为 1×4 的效果要比选择 2×2 好一些。

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。

2.3 HPL 的安装

官网 <https://netlib.org/benchmark/hpl/>，有更新的版本，可以自己选择。学习通上提供软件包版本是：

hpl-2.3.tar.gz

GotoBLAS2-1.13.tar.gz

主要内容：

(1) 安装 gotoblas

注：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

下载 GotoBLAS2-1.13.tar.gz

执行步骤：

在/usr/local/mathlib/goto 下解压：

```
$ tar -zxvf GotoBLAS2-1.13.tar.gz
```

```
$ cd GotoBLAS2
```

```
$ 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。

如果出现报错：试试将 `f_check #298` 行改为如下：

```
print MAKEFILE "FEXTRALIB=$linker_L -lgfortran -lm -lquadmath -lm $linker_a\n";
```

然后 `$make clean` 清除之前编译好的文件，重新编译。

具体操作流程如下：

先创建文件夹/usr/local/mathlib/goto: `mkdir -p /usr/local/mathlib/goto`

```
[root@slave1 ~]# mkdir -p /usr/local/mathlib/goto
[root@slave1 ~]# ls /usr/local/mathlib/goto
[root@slave1 ~]#
```

将文件 GotoBLAS2-1.13.tar.gz 传输到刚才创建的文件夹下，也可以用其他方法或直接下载：

```
scp -r D:\Users\FL\Desktop\实验三\GotoBLAS2-1.13.tar.gz root@192.168.86.222:/usr/local/mathlib/goto
```

注：scp 具体使用方法可参考实验一流程文档

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.22631.4460]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\FL>D:

D:>scp -r D:\Users\FL\Desktop\实验三\GotoBLAS2-1.13.tar.gz root@192.168.86.222:/usr/local/mathlib/goto
root@192.168.86.222's password:
GotoBLAS2-1.13.tar.gz                               100% 3105KB  72.2MB/s   00:00

D:\>
```

查看是否传输成功：

```
[root@slave1 ~]# ls /usr/local/mathlib/goto
GotoBLAS2-1.13.tar.gz
[root@slave1 ~]#
```

进入/usr/local/mathlib/goto 目录再进行解压： `tar -zxvf GotoBLAS2-1.13.tar.gz`

```
[root@slave1 ~]# cd /usr/local/mathlib/goto
[root@slave1 goto]# tar -zxvf GotoBLAS2-1.13.tar.gz
```

解压完成后进入 GotoBLAS2 目录，再修改 `f_check` 文件的 298 行处代码：

```
[root@slave1 goto]# ls
GotoBLAS2  GotoBLAS2-1.13.tar.gz
[root@slave1 goto]# cd GotoBLAS2/
[root@slave1 GotoBLAS2]# vi f_check
```

直接跳转至相应行号，方法如下：未进入编辑状态时输入:行号，然后回车即可定位到所给定的行数，例如定位到 298 行则输入:298 然后回车即可：

```
#!/usr/bin/perl

#
# 1. Not specified
# 1.1 Automatically detect, then check compiler
# 1.2 If no fortran compiler is detected, g77 is default with NOFORTRAN definition
# 2. Specified
# 2.1 If path is correct, check compiler
# 2.2 If path is not correct, but still valid compiler name, force setting
# 2.2.2 Path is not correct, invalid compiler name, then g77 is default with NOFORTRAN definition
#
#
$makefile = shift(@ARGV);
$config = shift(@ARGV);

$nofortran = 0;

$compiler = join(" ", @ARGV);

# f77 is too ambiguous
$compiler = "" if $compiler eq "f77";

@path = split(/:/, $ENV{"PATH"});

if ($compiler eq "") {

    @lists = ("f77", "g77", "g95", "gfortran", "frt", "fort", "openf90", "openf95",
              "sunf77", "sunf90", "sunf95",
              "xlf95", "xlf90", "xlf",
              "ppuf77", "ppuf95", "ppuf90", "ppuxlf",
              "pathf90", "pathf95",
              "pgf95", "pgf90", "pgf77",
              "ifort");

    foreach $lists (@lists) {
:298
```

将原有内容修改如下：

print MAKEFILE "FEXTRALIB=\$linker_L -lgfortran -lm -lquadmath -lm \$linker_a\n";

```
    && ($flags =~ /gfortranbegin/)
    && ($flags =~ /frtbegin/)
    && ($flags =~ /pathfstart/)
    && ($flags =~ /numa/)
    && ($flags =~ /crt10-91/)
    && ($flags =~ /gcc/)
    && ($flags =~ /user32/)
    && ($flags =~ /kernel32/)
    && ($flags =~ /advapi32/)
    && ($flags =~ /shell32/)
    ) {
        $linker_l .= $flags . " ";
    }

    $linker_a .= $flags . " " if $flags =~ /\.a$/;
}

}

open(MAKEFILE, ">> $makefile") || die "Can't append $makefile";
open(CONFFILE, ">> $config" ) || die "Can't append $config";

print MAKEFILE "F_COMPILER=$vendor\n";
print MAKEFILE "FC=$compiler\n";
print MAKEFILE "BU=$bu\n" if $bu ne "";
print MAKEFILE "NOFORTRAN=1\n" if $nofortran == 1;

print CONFFILE "#define BUNDERSCORE\t$bu\n" if $bu ne "";
print CONFFILE "#define NEEDBUNDERSCORE\t1\n" if $bu ne "";

if (($linker_l ne "") || ($linker_a ne "")) {
    #print MAKEFILE "FEXTRALIB=$linker_L $linker_l $linker_a\n";
    print MAKEFILE "FEXTRALIB=$linker_L -lgfortran -lm -lquadmath -lm $linker_a\n";
}

close(MAKEFILE);
-- INSERT --
```

将其改为如下内容：

接着进行编译: make CC=gcc BINARY=64 TARGET=NEHALEM

```
"f_check" 303L, 6039C written
[root@slave1 GotoBLAS2]# make CC=gcc BINARY=64 TARGET=NEHALEM_
```

结果报错如下:

```
=1 -DASMFNAME=xgetrs_T_single -DASMFNAME=xgetrs_T_single_ -DNAME=xgetrs_T_single_ -DCNAME=xgetrs_T_si
ngle -DCHAR_NAME=\"xgetrs_T_single_\" -DCHAR_CNAME=\"xgetrs_T_single_\" -I../.. -DXDOUBLE -DCOMPLEX -
DCOMPLEX -DXDOUBLE -DTRANS=2 zgetrs_single.c -o xgetrs_T_single.o
gcc -c -O2 -DEXPRECISION -m128bit-long-double -Wall -m64 -DF_INTERFACE_GFORT -fPIC -D_MAX_CPU_NUMBER
=1 -DASMFNAME=xgetrs_R_single -DASMFNAME=xgetrs_R_single_ -DNAME=xgetrs_R_single_ -DCNAME=xgetrs_R_si
ngle -DCHAR_NAME=\"xgetrs_R_single_\" -DCHAR_CNAME=\"xgetrs_R_single_\" -I../.. -DXDOUBLE -DCOMPLEX -
DCOMPLEX -DXDOUBLE -DTRANS=3 zgetrs_single.c -o xgetrs_R_single.o
gcc -c -O2 -DEXPRECISION -m128bit-long-double -Wall -m64 -DF_INTERFACE_GFORT -fPIC -D_MAX_CPU_NUMBER
=1 -DASMFNAME=xgetrs_C_single -DASMFNAME=xgetrs_C_single_ -DNAME=xgetrs_C_single_ -DCNAME=xgetrs_C_si
ngle -DCHAR_NAME=\"xgetrs_C_single_\" -DCHAR_CNAME=\"xgetrs_C_single_\" -I../.. -DXDOUBLE -DCOMPLEX -
DCOMPLEX -DXDOUBLE -DTRANS=4 zgetrs_single.c -o xgetrs_C_single.o
ar -ru ../libgoto2_nehalem-r1.13.a sgetrs_N_single.o sgetrs_T_single.o dgetrs_N_single.o dgetrs_
T_single.o cgetrs_N_single.o cgetrs_T_single.o cgetrs_R_single.o cgetrs_C_single.o zgetrs_N_single.o
zgetrs_T_single.o zgetrs_R_single.o zgetrs_C_single.o qgetrs_N_single.o qgetrs_T_single.o xgetrs_N_
single.o xgetrs_T_single.o xgetrs_R_single.o xgetrs_C_single.o
make[2]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/lapack/getrs'
make[1]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/lapack'
wget http://www.netlib.org/lapack/lapack-3.1.1.tgz
--2025-04-06 14:05:45-- http://www.netlib.org/lapack/lapack-3.1.1.tgz
Resolving www.netlib.org (www.netlib.org)... 160.36.239.231
Connecting to www.netlib.org (www.netlib.org):160.36.239.231:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.netlib.org/lapack/lapack-3.1.1.tgz [following]
--2025-04-06 14:05:48-- https://www.netlib.org/lapack/lapack-3.1.1.tgz
Connecting to www.netlib.org (www.netlib.org):160.36.239.231:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10407595 (9.9M) [application/x-gzip]
Saving to: 'lapack-3.1.1.tgz'

100%[=====>] 10,407,595 1.96MB/s in 12s

2025-04-06 14:06:02 (843 KB/s) - 'lapack-3.1.1.tgz' saved [10407595/10407595]

tar xzf lapack-3.1.1.tgz
/bin/sh: line 2: patch: command not found
make: *** [lapack-3.1.1] Error 127
[root@slave1 GotoBLAS2]#
```

安装 patch: yum install patch

```
=1 -DASMFNAME=xgetrs_T_single -DASMFNAME=xgetrs_T_single_ -DNAME=xgetrs_T_single_ -DCNAME=xgetrs_T_si
ngle -DCHAR_NAME=\"xgetrs_T_single_\" -DCHAR_CNAME=\"xgetrs_T_single_\" -I../.. -DXDOUBLE -DCOMPLEX -
DCOMPLEX -DXDOUBLE -DTRANS=2 zgetrs_single.c -o xgetrs_T_single.o
gcc -c -O2 -DEXPRECISION -m128bit-long-double -Wall -m64 -DF_INTERFACE_GFORT -fPIC -D_MAX_CPU_NUMBER
=1 -DASMFNAME=xgetrs_R_single -DASMFNAME=xgetrs_R_single_ -DNAME=xgetrs_R_single_ -DCNAME=xgetrs_R_si
ngle -DCHAR_NAME=\"xgetrs_R_single_\" -DCHAR_CNAME=\"xgetrs_R_single_\" -I../.. -DXDOUBLE -DCOMPLEX -
DCOMPLEX -DXDOUBLE -DTRANS=3 zgetrs_single.c -o xgetrs_R_single.o
gcc -c -O2 -DEXPRECISION -m128bit-long-double -Wall -m64 -DF_INTERFACE_GFORT -fPIC -D_MAX_CPU_NUMBER
=1 -DASMFNAME=xgetrs_C_single -DASMFNAME=xgetrs_C_single_ -DNAME=xgetrs_C_single_ -DCNAME=xgetrs_C_si
ngle -DCHAR_NAME=\"xgetrs_C_single_\" -DCHAR_CNAME=\"xgetrs_C_single_\" -I../.. -DXDOUBLE -DCOMPLEX -
DCOMPLEX -DXDOUBLE -DTRANS=4 zgetrs_single.c -o xgetrs_C_single.o
ar -ru ../libgoto2_nehalem-r1.13.a sgetrs_N_single.o sgetrs_T_single.o dgetrs_N_single.o dgetrs_
T_single.o cgetrs_N_single.o cgetrs_T_single.o cgetrs_R_single.o cgetrs_C_single.o zgetrs_N_single.o
zgetrs_T_single.o zgetrs_R_single.o zgetrs_C_single.o qgetrs_N_single.o qgetrs_T_single.o xgetrs_N_
single.o xgetrs_T_single.o xgetrs_R_single.o xgetrs_C_single.o
make[2]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/lapack/getrs'
make[1]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/lapack'
wget http://www.netlib.org/lapack/lapack-3.1.1.tgz
--2025-04-06 14:05:45-- http://www.netlib.org/lapack/lapack-3.1.1.tgz
Resolving www.netlib.org (www.netlib.org)... 160.36.239.231
Connecting to www.netlib.org (www.netlib.org):160.36.239.231:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.netlib.org/lapack/lapack-3.1.1.tgz [following]
--2025-04-06 14:05:48-- https://www.netlib.org/lapack/lapack-3.1.1.tgz
Connecting to www.netlib.org (www.netlib.org):160.36.239.231:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10407595 (9.9M) [application/x-gzip]
Saving to: 'lapack-3.1.1.tgz'

100%[=====>] 10,407,595 1.96MB/s in 12s

2025-04-06 14:06:02 (843 KB/s) - 'lapack-3.1.1.tgz' saved [10407595/10407595]

tar xzf lapack-3.1.1.tgz
/bin/sh: line 2: patch: command not found
make: *** [lapack-3.1.1] Error 127
[root@slave1 GotoBLAS2]# yum install patch
```


安装过程中需要确认，安装完成后的结果如下：

```
docker-ce-stable           | 3.5 kB  00:00:00
extras                     | 2.9 kB  00:00:00
updates                    | 2.9 kB  00:00:00
Resolving Dependencies
--> Running transaction check
--> Package patch.x86_64 0:2.7.1-12.el7_7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package Arch Version Repository Size
=====
Installing:
patch x86_64 2.7.1-12.el7_7 base 111 k

Transaction Summary
=====
Install 1 Package

Total download size: 111 k
Installed size: 210 k
Is this ok [y/d/N]: y
Downloading packages:
patch-2.7.1-12.el7_7.x86_64.rpm | 111 kB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : patch-2.7.1-12.el7_7.x86_64 1/1
Verifying : patch-2.7.1-12.el7_7.x86_64 1/1

Installed:
patch.x86_64 0:2.7.1-12.el7_7

Complete!
[root@slave1 GotoBLAS2]#
```

清理之前的编译结果：make clean

```
Installed:
patch.x86_64 0:2.7.1-12.el7_7

Complete!
[root@slave1 GotoBLAS2]# make clean
```

再次进行编译：make CC=gcc BINARY=64 TARGET=NEHALEM

```
[root@slave1 GotoBLAS2]# make CC=gcc BINARY=64 TARGET=NEHALEM
```

编译成功的截图如下：

```
lartg.o dlaruv.o dlas2.o dlascl.o dlasd0.o dlasd1.o dlasd2.o dlasd3.o dlasd4.o dlasd5.o dlasd6.o dlasd7.o dlasd8.o dlasda.o dlasdq.o dlasdt.o dlaset.o dlasq1.o dlasq2.o dlasq3.o dlasq4.o dlasq5.o dlasq6.o dlasr.o dlasrt.o dlassq.o dlasv2.o dpttrf.o dstebz.o dstedc.o dsteqr.o dsterrf.o dlaisnan.o disnan.o ../INSTALL/dsecnd_NONE.o ilaenv.o ieeeck.o lsamen.o iparmq.o ../INSTALL/ilaenv.o
ranlib ../libgoto2_nehalem-r1.13.a
make[2]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/lapack-3.1.1/SRC'
make[1]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/lapack-3.1.1'
make -j 1 -C exports so
make[1]: Entering directory '/usr/local/mathlib/goto/GotoBLAS2/exports'
perl ./gensymbol linux x86_64 _1 0 > linux.def
perl ./gensymbol linktest x86_64 _1 0 > linktest.c
gcc -O2 -DEXPRECISION -m128bit-long-double -Wall -m64 -DF_INTERFACE_GFORTRAN -fPIC -D_MAX_CPU_NUMBER=1 -D_ASMNAME= -D_ASMNAME= -D_NAME= -D_NAME= -D_CHAR_NAME="\\" -D_CHAR_NAME="\\" -I.. -shared -o ../libgoto2_nehalem-r1.13.so \
-Wl,--whole-archive ../libgoto2_nehalem-r1.13.a -Wl,--no-whole-archive \
-Wl,--retain-symbols-file=linux.def -lm -lm
gcc -O2 -DEXPRECISION -m128bit-long-double -Wall -m64 -DF_INTERFACE_GFORTRAN -fPIC -D_MAX_CPU_NUMBER=1 -D_ASMNAME= -D_ASMNAME= -D_NAME= -D_NAME= -D_CHAR_NAME="\\" -D_CHAR_NAME="\\" -I.. -w -o linktest linktest.c ../libgoto2_nehalem-r1.13.so -L/usr/lib/gcc/x86_64-redhat-linux/4.8.5 -L/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64 -L/lib/../../lib64 -L/usr/lib/../../lib64 -L/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64 -lgfortran -lm -lquadmath -lm && echo OK.
OK.
rm -f linktest
make[1]: Leaving directory '/usr/local/mathlib/goto/GotoBLAS2/exports'
ln -fs libgoto2_nehalem-r1.13.so libgoto2.so

GotoBLAS build complete.

OS ... Linux
Architecture ... x86_64
BINARY ... 64bit
C compiler ... GCC (command line : gcc)
Fortran compiler ... GFORTRAN (command line : gfortran)
Library Name ... libgoto2_nehalem-r1.13.a (Single threaded)

[root@slave1 GotoBLAS2]#
```

(2) 安装 OpenMPI 或 MPICH

详细步骤请参见实验二中的 MPICH 流程文档

(3) 安装 HPL

下载 hpl-2.3.tar.gz (网址: <http://www.netlib.org/benchmark/hpl/hpl-2.3.tar.gz>)

在用户目录下解压:

```
$ tar -zxvf hpl-2.3.tar.gz
```

```
$ cd hpl-2.3
```

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

ARCH = Linux

TOPdir = /home/用户目录/hpl-2.3

MPdir = MPI 安装目录

MPinc = \$(MPdir)/include

MPlib = -L\$(MPdir)/lib

LAdir = /usr/local/mathlib/goto/GotoBLAS2 #GotoBLAS2 的安装目录

LAlib = \$(LAdir)/libgoto2_nehalem-r1.13.a

HPL_INCLUDES = -I\$(INCdir) -I\$(INCdir)/\$(ARCH) -I\$(Lainc) -I\$(MPinc)

CC = MPI 安装目录/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 = \$(CC) #同 CC

LINKFLAGS = \$(CCFLAGS) #同 CCFLAGS

具体操作流程如下:

将文件 hpl-2.3.tar.gz 传输到/root 文件夹下 (文件夹可以自己定义), 也可以用其他方法或直接下载:

```
scp -r D:\Users\FL\Desktop\实验三\hpl-2.3.tar.gz root@192.168.86.222:/root
```

```
D:\> scp -r D:\Users\FL\Desktop\实验三\hpl-2.3.tar.gz root@192.168.86.222:/root
root@192.168.86.222's password:
hpl-2.3.tar.gz 100% 645KB 70.0MB/s 00:00
D:\>
```

查看是否传输成功：

```
[root@slave1 ~]# ls
anaconda-ks.cfg          hellomp.c               mpich-3.4               ■ ■ ■ ■
cmake-3.25.1-linux-x86_64 hellomp.out             my-code-server.tar     ■ ■ ■ ■
cmake-3.25.1-linux-x86_64.tar.gz matplotlib-cpp         ■ ■ ■ ■
[root@slave1 ~]# ls
anaconda-ks.cfg          hellomp.c               matplotlib-cpp          ■ ■ ■ ■
cmake-3.25.1-linux-x86_64 hellomp.out             mpich-3.4              ■ ■ ■ ■
cmake-3.25.1-linux-x86_64.tar.gz hpl-2.3.tar.gz         my-code-server.tar     ■ ■ ■ ■
[root@slave1 ~]#
```

接着进行解压：tar zxvf hpl-2.3.tar.gz

```
[root@slave1 ~]# tar zxvf hpl-2.3.tar.gz
```

查看 Makefile 模板：

```
CentOS Linux 7 (Core)
Kernel 3.10.0-1160.el7.x86_64 on an x86_64

slave1 login: root
Password:
Last login: Sun Apr  6 13:46:23 on tty1
[root@slave1 ~]# ls hpl-2.3/setup
Make.FreeBSD_PII_CBLAS  Make.Linux_ATHLON_USIPL  Make.Linux_PII_USIPL_gm  Make.SUN4SOL2-g_USIPL
make_generic            Make.Linux_Intel64      Make.MacOSX_Accelerate  Make.T3E_FBLAS
Make.HPUX_FBLAS         Make.Linux_PII_CBLAS    Make.PWR2_FBLAS         Make.Tru64_FBLAS
Make.I860_FBLAS         Make.Linux_PII_CBLAS_gm  Make.PWR3_FBLAS         Make.Tru64_FBLAS_elan
Make.IRIX_FBLAS         Make.Linux_PII_FBLAS    Make.PWRPC_FBLAS        Make.UNKNOWN.in
Make.Linux_ATHLON_CBLAS  Make.Linux_PII_FBLAS_gm  Make.SUN4SOL2_FBLAS
Make.Linux_ATHLON_FBLAS  Make.Linux_PII_USIPL    Make.SUN4SOL2-g_FBLAS
[root@slave1 ~]#
```

将 setup 文件夹下的 Make.Linux_PII_CBLAS 模板文件复制到 hpl-2.3 文件夹下并命名为 Make.Linux：

cp setup/Make.Linux_PII_CBLAS Make.Linux

```
[root@slave1 ~]# cd hpl-2.3/
[root@slave1 hpl-2.3]# ls
acinclude.m4  ChangeLog  configure  depcomp  install-sh  makes  NEWS  testing  www
aclocal.m4    compile   configure.ac  HISTORY  Makefile    Make.top  README  THANKS
AUTHORS       config.guess  COPYING      include  Makefile.am  man      setup  TODO
BUGS          config.sub  COPYRIGHT    INSTALL  Makefile.in  missing  src    TUNING
[root@slave1 hpl-2.3]# cp setup/Make.Linux_PII_CBLAS Make.Linux
[root@slave1 hpl-2.3]# ls
acinclude.m4  compile   COPYING      INSTALL  Make.Linux  NEWS  THANKS
aclocal.m4    config.guess  COPYRIGHT    install-sh  makes      README  TODO
AUTHORS       config.sub  depcomp      Makefile    Make.top    setup  TUNING
BUGS          configure  HISTORY      Makefile.am  man        src    www
ChangeLog     configure.ac  include      Makefile.in  missing    testing
[root@slave1 hpl-2.3]#
```

接着编辑 Make.Linux 文件：vi Make.Linux

```
[root@slave1 hpl-2.3]# vi Make.Linux
```

需要修改的内容如下图红框圈出的内容所示（共 8 处）：

注意：如果上一步没有将 Makefile 模板重新命名则第一处 ARCH 的值无需修改

```
##
## - Platform identifier -
##
ARCH = Linux_PII_CBLAS
##
## - HPL Directory Structure / HPL library -
##
TOPdir = $(HOME)/hpl
INCdir = $(TOPdir)/include
BINdir = $(TOPdir)/bin/$(ARCH)
LIBdir = $(TOPdir)/lib/$(ARCH)
HPLlib = $(LIBdir)/libhpl.a
##
## - Message Passing library (MPI) -
##
## MPinc tells the C compiler where to find the Message Passing library
## header files, MPlib is defined to be the name of the library to be
## used. The variable MPdir is only used for defining MPinc and MPlib.
##
MPdir = /usr/local/mpi
MPinc = -I$(MPdir)/include
MPlib = $(MPdir)/lib/libmpich.a
##
## - Linear Algebra library (BLAS or USIPL) -
##
## LAlinc tells the C compiler where to find the Linear Algebra library
## header files, LAlib is defined to be the name of the library to be
## used. The variable LAdir is only used for defining LAlinc and LAlib.
##
```

```

# -----
# - Linear Algebra library (BLAS or USIPL) -----
# -----
# LAinc tells the C compiler where to find the Linear Algebra library
# header files, LAlib is defined to be the name of the library to be
# used. The variable LAdir is only used for defining LAinc and LAlib.
#
LAdir      = $(HOME)/netlib/ARCHIVES/Linux_PII
LAinc      =
LAlib      = $(LAdir)/libcblas.a $(LAdir)/libatlas.a
#
# -----
# - F77 / C interface -----
# -----

# -----
# - Compilers / linkers - Optimization flags -----
# -----
#
CC          = /usr/bin/gcc
CCNOOPT     = $(HPL_DEFS)
CCFLAGS     = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops
#
# On some platforms, it is necessary to use the Fortran linker to find
# the Fortran internals used in the BLAS library.
#
LINKER      = /usr/bin/g77
LINKFLAGS   = $(CCFLAGS)
#
ARCHIVER     = ar
ARFLAGS     = r
RANLIB      = echo
#
# -----

```

具体修改如下：

注意：有关路径设置要与自己的安装路径相一致

```

# - Platform identifier -----
# -----
#
#ARCH       = Linux_PII_CBLAS
ARCH        = Linux
#
# -----
# - HPL Directory Structure / HPL library -----
# -----
#
#TOPdir     = $(HOME)/hpl
TOPdir      = /root/hpl-2.3
INCdir      = $(TOPdir)/include
BINdir      = $(TOPdir)/bin/$(ARCH)
LIBdir      = $(TOPdir)/lib/$(ARCH)
#
HPLlib      = $(LIBdir)/libhpl.a
#
# -----
# - Message Passing library (MPI) -----
# -----
#
# MPinc tells the C compiler where to find the Message Passing library
# header files, MPlib is defined to be the name of the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
#MPdir      = /usr/local/mpi
MPdir       = /opt/mpich-3.4
MPinc       = -I$(MPdir)/include
#MPlib      = $(MPdir)/lib/libmpich.a
MPlib       = -L$(MPdir)/lib
#
# -----

```



```

# -----
# - Linear Algebra library (BLAS or USIPL) -----
# -----
# LAlinc tells the C compiler where to find the Linear Algebra library
# header files, LAlib is defined to be the name of the library to be
# used. The variable LAdir is only used for defining LAlinc and LAlib.
#
#LAdir      = $(HOME)/netlib/ARCHIVES/Linux_PII
LAdir      = /usr/local/mathlib/goto/GotoBLAS2
LAlinc      =
#LAlib      = $(LAdir)/libcbblas.a $(LAdir)/libatlas.a
LAlib      = $(LAdir)/libgoto2.a $(LAdir)/libgoto2.so
#
# -----
# - F77 / C interface -----
# -----

```

```

# -----
# - Compilers / linkers - Optimization flags -----
# -----
#
#CC          = /usr/bin/gcc
CC          = /opt/mpich-3.4/bin/mpicc
CCNOOPT     = $(HPL_DEFS)
CCFLAGS     = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops
#
# On some platforms, it is necessary to use the Fortran linker to find
# the Fortran internals used in the BLAS library.
#
#LINKER      = /usr/bin/g77
LINKER      = /opt/mpich-3.4/bin/mpif77
LINKFLAGS   = $(CCFLAGS)
#
ARCHIVER     = ar
ARFLAGS     = r
RANLIB      = echo
#
# -----
# -----

```

(4) 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**

错误信息是：undefined reference to symbol 'pthread_create@@GLIBC_

则试试在 Make.Linux 中的 CC 定义里加参数：

CC = mpicc -lpthread -L/usr/lib64 #/usr/lib64 目录是 libpthread 库所在目录，核查一下

如果使用 icc ，使用命令 **#which mpiicc** 查看后 修改 Make.Linux

具体操作流程如下：

进行编译：make arch=Linux

注意：此处 arch 的值需要和上述操作中的 Makefile 模板名称相一致

```
[root@slave1 hpl-2.3]# make arch=Linux
```

编译完成后的截图如下：

```
( cd testing/ptimer/Linux; make )
make[2]: Entering directory `/root/hpl-2.3/testing/ptimer/Linux'
/opt/mpich-3.4/bin/mpicc -o HPL_ptimer.o -c -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops ../HPL_ptimer.c
/opt/mpich-3.4/bin/mpicc -o HPL_ptimer_cputime.o -c -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops ../HPL_ptimer_cputime.c
/opt/mpich-3.4/bin/mpicc -o HPL_ptimer_walltime.o -c -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops ../HPL_ptimer_walltime.c
ar r /root/hpl-2.3/lib/Linux/libhpl.a HPL_ptimer.o HPL_ptimer_cputime.o HPL_ptimer_walltime.o
echo /root/hpl-2.3/lib/Linux/libhpl.a
/root/hpl-2.3/lib/Linux/libhpl.a
touch lib.grd
make[2]: Leaving directory `/root/hpl-2.3/testing/ptimer/Linux'
( cd testing/ptest/Linux; make )
make[2]: Entering directory `/root/hpl-2.3/testing/ptest/Linux'
/opt/mpich-3.4/bin/mpicc -o HPL_pddriver.o -c -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops ../HPL_pddriver.c
/opt/mpich-3.4/bin/mpicc -o HPL_pinfo.o -c -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops ../HPL_pinfo.c
/opt/mpich-3.4/bin/mpicc -o HPL_pptest.o -c -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops ../HPL_pptest.c
/opt/mpich-3.4/bin/mpif77 -DHPL_CALL_CBLAS -I/root/hpl-2.3/include -I/root/hpl-2.3/include/Linux -I/opt/mpich-3.4/include -fomit-frame-pointer -O3 -funroll-loops -o /root/hpl-2.3/bin/Linux/xhpl HPL_pddriver.o HPL_pinfo.o HPL_pptest.o /root/hpl-2.3/lib/Linux/libhpl.a /usr/local/mathlib/goto/GotoBLAS2/libgoto2.a /usr/local/mathlib/goto/GotoBLAS2/libgoto2.so -L/opt/mpich-3.4/lib
make /root/hpl-2.3/bin/Linux/HPL.dat
make[3]: Entering directory `/root/hpl-2.3/testing/ptest/Linux'
( cp ../HPL.dat /root/hpl-2.3/bin/Linux )
make[3]: Leaving directory `/root/hpl-2.3/testing/ptest/Linux'
touch dexe.grd
make[2]: Leaving directory `/root/hpl-2.3/testing/ptest/Linux'
make[1]: Leaving directory `/root/hpl-2.3'
[root@slave1 hpl-2.3]# _
```

编译成功后在 bin/Linux 目录下会出现 xhpl 文件：

```
[root@slave1 hpl-2.3]# cd bin/Linux/
[root@slave1 Linux]# ls
HPL.dat xhpl
[root@slave1 Linux]#
```

先进行单节点测试：mpirun -np 4 ./xhpl

```
[root@slave1 Linux]# mpirun -np 4 ./xhpl
```

测试刚开始时的截图：

```
An explanation of the input/output parameters follows:
T/U    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:
N      :      29      30      34      35
NB     :      1      2      3      4
PMAP   : Row-major process mapping
P      :      2      1      4
Q      :      2      4      1
PFACT  : Left      Crout      Right
NBMIN  :      2      4
NDIV   :      2
RFACT  : Left      Crout      Right
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 1.118223e-16
- Computational tests pass if scaled residuals are less than 16.0
```

测试结束后的截图：

```
HPL_pdgesv() start time Sun Apr 6 17:19:23 2025
HPL_pdgesv() end time Sun Apr 6 17:19:25 2025

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 2.41305194e-02 ..... PASSED
=====
T/U          N    NB    P    Q          Time          Gflops
-----
WR00R2R2      35     4     4     1          1.99          1.5302e-05
HPL_pdgesv() start time Sun Apr 6 17:19:26 2025
HPL_pdgesv() end time Sun Apr 6 17:19:28 2025

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 1.88391027e-02 ..... PASSED
=====
T/U          N    NB    P    Q          Time          Gflops
-----
WR00R2R4      35     4     4     1          2.06          1.4796e-05
HPL_pdgesv() start time Sun Apr 6 17:19:28 2025
HPL_pdgesv() end time Sun Apr 6 17:19:30 2025

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 2.80968060e-02 ..... PASSED
=====

Finished      864 tests with the following results:
              864 tests completed and passed residual checks,
               0 tests completed and failed residual checks,
               0 tests skipped because of illegal input values.
-----

End of Tests.
=====
[root@slave1 Linux]#
```

2.4 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：

cu01

cu01

cu01

cu01

cu02

.....

(3) 修改 HPL.dat，设置运算规模和进程数等

\$ vi HPL.dat

(4) 运行

\$ mpirun -np 最大进程数 -machinefile nodes xhpl

具体操作流程如下：在上一步操作过程中已经进行了单节点测试，下面进行集群（多节点）性能测试：

创建 nodes 文件，并指定 linpack 运行的节点名称：vi nodes

注意：所有节点要能够 ssh 免密登录，同时关闭系统的防火墙 firewalld

```
[root@master Linux]# ls
HPL.dat  xhpl
[root@master Linux]# vi nodes
```

nodes 的内容如下:

master

slave1

这样默认两个节点各分配一半的进程, 也可以指定分配给各节点的进程数, 如

master:3

slave1:1

```
master
slave1
```

进行多节点测试, 并将测试结果写入到 HPL-Benchmark.txt 文件中 (这样原本输出到终端的内容将全部写入到指定的 HPL-Benchmark.txt 文件中):

mpirun -machinefile nodes -np 4 ./xhpl > HPL-Benchmark.txt

```
"nodes" 2L, 14C written
[root@master Linux]# mpirun -machinefile nodes -np 4 ./xhpl > HPL-Benchmark.txt

[root@master Linux]# mpirun -machinefile nodes -np 4 ./xhpl > HPL-Benchmark.txt
[root@master Linux]# ls
HPL-Benchmark.txt  HPL.dat  nodes  xhpl
[root@master Linux]#
```

其中 HPL-Benchmark.txt 内容如下:

```
=====
HPLinpack 2.3 -- High-Performance Linpack benchmark -- December 2, 2018
Written by A. Petitet and R. Clint Whaley, Innovative Computing Laboratory, UTK
Modified by Piotr Luszczyk, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
=====

An explanation of the input/output parameters follows:
T/U      : Wall time / encoded variant.
N        : The order of the coefficient matrix A.
NB       : The partitioning blocking factor.
P        : The number of process rows.
Q        : The number of process columns.
Time     : Time in seconds to solve the linear system.
Gflops   : Rate of execution for solving the linear system.

The following parameter values will be used:

N       :      1960      2048
NB      :       60       80
PMAP    : Row-major process mapping
P       :        2        1
Q       :        2        4
PFACT   :   Left   Crout   Right
NBMIN   :        2        4
NDIV    :        2
RFACT   :   Left   Crout   Right
BCAST   :   1ring
DEPTH   :         0
SWAP    : Mix (threshold = 64)
L1      : transposed form
U       : transposed form
EQUIL   : yes
ALIGN   : 8 double precision words

-----
"HPL-Benchmark.txt" 1493L, 86590C
```

注意: 多节点运行需要在所有节点上安装 mpich、GotoBLAS2 和 hpl, 也可以只一个节点安装, 其余节点采用 NFS 挂载的方式; 但剩余节点都需要将主节点 mpich、GotoBLAS2、hpl 的安装目录分别挂载一遍, 具体挂载操作流程可查看实验二的 NFS 操作文档, 例如:

mount master:/root/mpich-3.4 /root/mpich-3.4

mount master:/usr/local/mathlib/goto/GotoBLAS2 /usr/local/mathlib/goto/GotoBLAS2

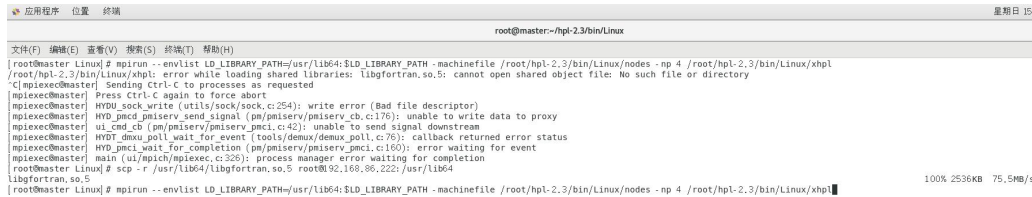
mount master:/root/hpl-2.3 /root/hpl-2.3

```
[root@slave1 ~]# mount master:/root/mpich-3.4 /root/mpich-3.4
[root@slave1 ~]# mount master:/usr/local/mathlib/goto/GotoBLAS2 /usr/local/mathlib/goto/GotoBLAS2
```

尝试进行多节点测试: `mpirun -machinefile /root/hpl-2.3/bin/Linux/nodes -np 4 /root/hpl-2.3/bin/Linux/xhpl`

注意: nodes 和 xhpl 文件的路径要写全即绝对路径

可能会遇到如下提示缺少 libgfortran.so.5 文件的错误:



解决方法: 先输出存放 libgfortran.so.5 文件的路径: `find / -name libgfortran.so.5 2>/dev/null`

例如本机的 libgfortran.so.5 文件存放在 /usr/lib64/ 路径下, 然后将 libgfortran.so.5 文件发送给其余节点的相同目录下, 这里将发送给节点 slave1:

`scp -r /usr/lib64/libgfortran.so.5 root@192.168.86.222:/usr/lib64`

完成上述步骤后则能够进行多节点测试。

修改 HPL 配置文件: `vi HPL.dat`

注: HPL.dat 文件中各项内容的具体含义可参考章节 2.5 Performance Tuning 的《(2) 修改 HPL.dat 设置运行参数》部分的内容

```
[root@slave1 Linux]# ls
HPL.dat  xhpl
[root@slave1 Linux]# vi HPL.dat
```

HPL.dat 文件的内容如下:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
4            # of problems sizes (N)
29 30 34 35  Ns
4            # of NBs
1 2 3 4      NBs
0            PMAP process mapping (0=Row-,1=Column-major)
3            # of process grids (P x Q)
2 1 4        Ps
2 4 1        Qs
16.0         threshold
9            # of panel fact
0 1 2        PFACTs (0=left, 1=Crout, 2=Right)
2            # of recursive stopping criterium
2 4          NBMINs (>= 1)
1            # of panels in recursion
2            NDIUs
3            # of recursive panel fact.
0 1 2        RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
0            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
0            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)

"HPL.dat" 31L, 1133C
```

本次实验只需要修改下图框线框出部分的内容：

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
4           # of problems sizes (N)
29 30 34 35  Ns
4           # of NBs
1 2 3 4      NBs
0           PMAP process mapping (0=Row-,1=Column-major)
3           # of process grids (P x Q)
2 1 4       Ps
2 4 1       Qs
16.0        threshold
3           # of panel fact
0 1 2       PFACTs (0=left, 1=Crout, 2=Right)
2           # of recursive stopping criterium
2 4         NBMINs (>= 1)
1           # of panels in recursion
2           NDIUs
3           # of recursive panel fact.
0 1 2       RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
0           BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1           # of lookahead depth
0           DEPTHs (>=0)
2           SWAP (0=bin-exch,1=long,2=mix)
64          swapping threshold
0           L1 in (0=transposed,1=no-transposed) form
0           U  in (0=transposed,1=no-transposed) form
1           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
~
~
~
~
"HPL.dat" 31L, 1133C
```

可修改为如下：

```
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
2           # of problems sizes (N)
1960 2048    Ns
2           # of NBs
60 80        NBs
0           PMAP process mapping (0=Row-,1=Column-major)
2           # of process grids (P x Q)
2 1          Ps
2 4          Qs
16.0        threshold
3           # of panel fact
0 1 2       PFACTs (0=left, 1=Crout, 2=Right)
2           # of recursive stopping criterium
2 4         NBMINs (>= 1)
1           # of panels in recursion
2           NDIUs
3           # of recursive panel fact.
0 1 2       RFACTs (0=left, 1=Crout, 2=Right)
1           # of broadcast
0           BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1           # of lookahead depth
0           DEPTHs (>=0)
2           SWAP (0=bin-exch,1=long,2=mix)
64          swapping threshold
0           L1 in (0=transposed,1=no-transposed) form
0           U  in (0=transposed,1=no-transposed) form
1           Equilibration (0=no,1=yes)
8           memory alignment in double (> 0)
~
~
~
~
"HPL.dat" 31L, 1133C written
[root@slave1 Linux]# _
```

再次进行测试：

```
An explanation of the input/output parameters follows:
T/U   : Wall time / encoded variant.
N     : The order of the coefficient matrix A.
NB    : The partitioning blocking factor.
P     : The number of process rows.
Q     : The number of process columns.
Time  : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :      1960      2048
NB     :      60       80
PMAP   : Row-major process mapping
P      :      2        1
Q      :      2        4
PFACT  : Left      Crout      Right
NBMIN  :      2        4
NDIU   :      2
RFACT  : Left      Crout      Right
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      1.118223e-16
- Computational tests pass if scaled residuals are less than 16.0
```

测试结束后的结果如下图所示：

```
HPL_pdgesv() start time Sun Apr  6 18:46:34 2025
HPL_pdgesv() end time   Sun Apr  6 18:46:38 2025

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 5.18889608e-03 ..... PASSED
=====
T/U      N      NB      P      Q      Time      Gflops
-----
WR00R2R2 2048    80      1      4      4.22      1.3579e+00
HPL_pdgesv() start time Sun Apr  6 18:46:39 2025
HPL_pdgesv() end time   Sun Apr  6 18:46:43 2025

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 4.79583163e-03 ..... PASSED
=====
T/U      N      NB      P      Q      Time      Gflops
-----
WR00R2R4 2048    80      1      4      4.21      1.3628e+00
HPL_pdgesv() start time Sun Apr  6 18:46:44 2025
HPL_pdgesv() end time   Sun Apr  6 18:46:48 2025

-----
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 5.45168137e-03 ..... PASSED
=====

Finished      144 tests with the following results:
              144 tests completed and passed residual checks,
               0 tests completed and failed residual checks,
               0 tests skipped because of illegal input values.

-----

End of Tests.
=====
[root@slave1 Linux]#
```

注：根据配置文件，测试组合的数量由以下参数决定：

矩阵规格：2 种（1960 和 2048）。

分块大小：2 种（60 和 80）。

进程网格：2 种（2x2 和 4x1）。

因此，上述例子中总的测试组合数量为：2×2×2=8

2.5 Performance Tuning

(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 库的效果要好得多，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。GOTO BLAS 库的设计思想，主要是基于几个方面的观察和考虑，其中之一是：

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 GOTO BLAS 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 2048            Ns                                        //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、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 2048 Ns                               //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 1 Ps                                    //每对 PQ 具体的值
2 4 Qs

```

指出用 2 种进程组合方式，分别为（p=2，q=2）和（p=1，q=4）

注：p=2，q=2 时需要的进程数是 $p \times q = 2 \times 2 = 4$ ，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
$\ Ax-b\ _{\infty} / (\epsilon \cdot \ A\ _1 \cdot N) = 0.0175089 \dots \text{PASSED}$ $\ Ax-b\ _{\infty} / (\epsilon \cdot \ A\ _1 \cdot \ x\ _1) = 0.0035454 \dots \text{PASSED}$ $\ Ax-b\ _{\infty} / (\epsilon \cdot \ A\ _{\infty} \cdot \ x\ _{\infty}) = 0.0007503 \dots \text{PASSED}$						

2.6 程序运行的注意事项

- (1) mpirun 的 -np 参数值是服务器的 CPU 总核心数。对应着参数文件种 $P * Q$ 的值。要求 -np 参数的值 $\geq P * Q$, 否则程序报错。
- (2) HPL 若使用超线程后总 CPU 线程数来进行计算, 其 FLOPS 值更低, 比用总核心数低约 20%。
- (3) 若可以的话, 设置不同的参数来让 HPL 程序运行多次, 取其最高值(峰值)作为服务的 FLOPS 计算性能。
- (4) 配置以 # 开始的参数用于设置多个值, 程序有 7 个参数可以设置不同的多个值, 于是排列组合后可以运行很多次, 选择最大值作为 FLOPS 计算峰值。
- (5) 当设置多种参数值, 会很消耗计算时间, 且其结果都相差不大, 则可以考虑节约时间, 不用设置太多的参数值。
- (6) 程序第一个运行种 CPU 刚唤醒, 导致其有效计算所占的比例要小些, 其结果会差些, 特别是 N 值设置不够大时更明显, 推荐至少有 2 次运行。
- (7) 设置稍大的 N 值, 让每次 Linpack 程序运行时间长度超过 200 秒, 则获得的结果更准确些。若运行时间太长, 则感觉很费时间。

2.7 HPL 性能测试记录

(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

将 HPL 最佳测试结果填写下面表格

进程个数	1	2	3	4
峰值速度				
HPL Gflops				
效率				
N				
NB				
P				
Q				
Time				
参与运算主机名				

注: 在 HPL 测试中, 效率 (Efficiency) 是一个重要的指标, 用于衡量实际性能与理论峰值性能之间的关系。效率的计算公式如下:

$$\text{Efficiency} = \text{实际性能 (Gflops)} / \text{理论峰值性能 (Gflops)}$$

其中, **实际性能:** 实际性能是指 HPL 测试中测得的浮点运算次数, 单位为 Gflops。从 HPL 测试结果中可以直接读取到。

理论峰值性能: 理论峰值性能是指计算机理论上能达到的每秒钟能完成的浮点运算次数。计算公式如下:

理论峰值性能 = CPU 主频 × CPU 核数 × 每个时钟周期执行的浮点运算次数 (与 CPU 的架构和指令集有关)

(3) 完成上述测试后比较和分析测试结果, 特别是如何能够得到高的性能测试值