

# HPC101 实验报告 1

## 实验一：简单集群搭建

姓名: 洪奕迅

学号: 3230102930

班号: 工信 2319

课程综合实践 I

(短学期, 2024)

浙江大学

计算机学院

2024 年 5 月 22 日

# 目录

<b>1 集群搭建流程</b>	<b>2</b>
1.1 虚拟机配置 . . . . .	2
1.2 BLAS 和 CBLAS 配置 . . . . .	2
1.3 OpenMPI 配置 . . . . .	4
1.4 HPL 配置 . . . . .	6
1.5 集群网络和 ssh 免密登陆配置 . . . . .	7
<b>2 集群测试</b>	<b>10</b>
<b>A 附录</b>	<b>14</b>
A.1 自我介绍 . . . . .	14
A.2 代码 . . . . .	14

# 集群搭建流程

## 1.1 虚拟机配置

在开始进行环境配置之前，我们需要搭建一个环境合适的系统，以下是系统的硬件配置：

系统：Debian12 - 64bit

处理器：7800X3D - 4 slots

内存：4GB

硬盘：20GB

网络模式：NAT 网络

此处应当注意网络，在参照文档选择 NAT 网络之前，首先需要在全局设置里配置 NAT 网络，以下是操作截图，在这样配置完成后，复制的不同 mac 的虚拟机将获得不同的 IP：

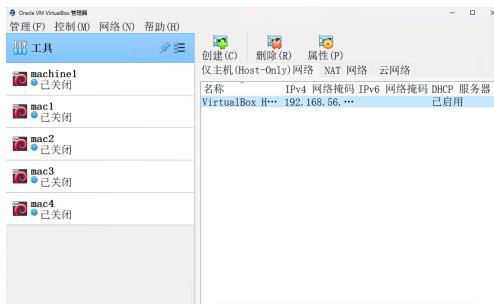


图 1.1: 全局网络设定



图 1.2: 虚拟机网络设置

这样我们获得了一个能连接外网且可以从 DHCP 服务器获得 IP 的 Linux 虚拟机，接下来开始进行集群计算环境的配置。

## 1.2 BLAS 和 CBLAS 配置

在第一次使用 root 用户完成配置后，发现 HPL 希望用户在普通用户权限下使用，因此后续所有的操作将在普通环境下进行，但是为了执行部分操作，需要将该用户加入 Sudoer 组。执行以下命令：

**Code Snippet 1.2.1 ▶ 切换用户**

```
1 vim /etc/sudoers
2 # 找到用户组
3 i
4 username = (ALL)ALL
5 esc wq
6 su username
7 cd /home/username #root 用户的主目录在 /root, 进入当前主目录进行接下来的操作
```

现在你的普通用户获得了通过输入密码和 sudo 指令来执行某些指令的权限，接下来进行 BLAS 和 CBLAS 的下载和部署。首先进行环境安装并下载包：

**Code Snippet 1.2.2 ▶ 环境配置和包下载**

```
1 gcc --version
2 # 如果提示找不到执行 sudo apt-get install gcc
3 gfortran --version
4 # 如果提示找不到执行 sudo apt-get install gfortran
5 # 下载 BLAS 和 CBLAS 并进行解压
6 wget http://www.netlib.org/blas/blas-3.8.0.tgz
7 wget http://www.netlib.org/blas/blast-forum/cblas.tgz
8 tar -zxvf blas-3.8.0.tgz
9 tar -zxvf cblas.tgz
```

完成以上准备工作后我们先对 BLAS 进行配置：

**Code Snippet 1.2.3 ▶ BLAS 环境配置**

```
1 cd BLAS-3.8.0
2 make
3 # 将所有 .o 文件与库文件进行链接
4 ar rv libblas.a *.o
5 # 将库文件复制到系统库文件
6 sudo cp blas_LINUX.a /usr/local/lib
```

接下来进行 CBLAS 配置：

**Code Snippet 1.2.4 ▶ CBLAS 配置**

```
1 cd ../../CBLAS
2 # 将此前编译的 BLAS 库文件复制到此处
3 cp /home/username/BLAS-3.8.0/libblas.a ./
4 # 修改 Makefile.in 的配置
5 vim Makefile.in
6 # 找到语句做如下修改
7 BLLIB = /home/username/BLAS-3.8.0/blas-LINUX.a
8 esc wq
9 # 准备部署
10 cd ..
11 cp /home/username/BLAS-3.8.0/libblas.a testing
12 make
13 # 将生成的库文件复制到系统库路径
14 sudo cp lib/cbla_LINUX.a /usr/local/lib
```

注意在这里 make 过程中可能由于 Fortran 版本问题会遇到 *Rank mismatch... (scalar and rank - 1)* 报错，为了解决这个问题，我们向 *Makefile* 中加入 *-fallow-argument-mismatch* 选择忽略这个错误即可（按照官方文档中所述，这个问题发生的原因是在数组参数传递时传递了标量），在后续过程中还会碰到这个问题，执行以下命令：

**Code Snippet 1.2.5 ▶ 解决报错**

```
1 cd testing
2 vim Makefile
3 # 找到最后几行在编译指令加入参数
4 .f.o:
5     $(FC) $(FFLAGS) -c -fallow-argument-mismatch $.f
```

## 1.3 OpenMPI 配置

在完成了向量矩阵计算库 BLAS 和 CBLAS 的配置后，接下来我们进行并行运算工具 MPI 的配置，这里选择的是 OpenMPI。依照以下的指令进行配置：

**Code Snippet 1.3.1 ▶ OpenMPI 配置**

```

1 cd /home/forever # 回到主目录方便日后管理
2 wget https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-4.0.4.tar.gz #
   ↳ 下载压缩包
3 tar -zxvf openmpi-4.0.4.tar.gz # 解压压缩包
4 cd openmpi-4.0.4 # 进入目标路径
5 # 开始配置
6 ./configure --prefix=/usr/local/openmpi 自定义路径安装
7 make
8 make install
9 # 配置环境变量
10 vim /etc/profile
11 # 在末尾加入以下环境变量路径声明
12 MPI_HOME=/usr/local/openmpi
13 export PATH=${MPI_HOME}/bin:$PATH
14 export LD_LIBRARY_PATH=${MPI_HOME}/lib:$LD_LIBRARY_PATH
15 export MANPATH=${MPI_HOME}/share/man:$MANPATH
16 esc wq

```

现在 OpenMPI 的配置也完成了，接下来进行测试确保配置完成，执行以下命令：

**Code Snippet 1.3.2 ▶ MPI 测试**

```

1 # 进入测试文件夹
2 cd examples
3 make
4 mpirun/mpiexec -np 4 hello_c

```

如果配置正确编译不会报错且应该会看到以下输出：

```

Hello, world, I am 0 of 4, (Open MPI v4.0.4, package: Open MPI forever@forever Distribution, ident: 4.0.4, repo rev: v4.0.4, Jun 10, 2020, 110)
Hello, world, I am 1 of 4, (Open MPI v4.0.4, package: Open MPI forever@forever Distribution, ident: 4.0.4, repo rev: v4.0.4, Jun 10, 2020, 110)
Hello, world, I am 2 of 4, (Open MPI v4.0.4, package: Open MPI forever@forever Distribution, ident: 4.0.4, repo rev: v4.0.4, Jun 10, 2020, 110)
Hello, world, I am 3 of 4, (Open MPI v4.0.4, package: Open MPI forever@forever Distribution, ident: 4.0.4, repo rev: v4.0.4, Jun 10, 2020, 110)

```

图 1.3: 期望输出

## 1.4 HPL 配置

基本同此前的操作相同，我们先进行下载和解压然后开始配置：

### Code Snippet 1.4.1 ▶ HPL 下载配置

```

1 wget http://www.netlib.org/benchmark/hpl/hpl-2.3.tar.gz
2 tar -zxvf hpl-2.3.tar.gz
3 cd hpl-2.3/
4 # 根据自己的平台选择正确后缀 make 文件并复制回上一级目录
5 cp setup/Make.Linux_PII_CBLAS ./
6 # 开始进行最重要的配置文件修改
7 vim Make.Linux_PII_CBLAS

```

以下为按照本文档配置下需要进行的修改：

```

# ARCH      = Linux_PII_CBLAS <- 修改为 Make 文件后缀
#
# -----
# - HPL Directory Structure / HPL library
#
# TOPdir    = $(HOME)/hpl-2.3 <- HPL 的安装路径
INCdir    = $(TOPdir)/include
BINdir    = $(TOPdir)/bin/$(ARCH)
LIBdir    = $(TOPdir)/lib/$(ARCH)
#
HPLlib     = $(LIBdir)/libhpl.a
#
# -----
# - Message Passing library (MPI) -----
#
# MPinc tells the C compiler where to find the Message Passing library
# header files. MPlib is defined to be the name of the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
MPdir      = /usr/local/openmpi <- openmpi 的安装路径
MPinc      = -I$(MPdir)/include
MPlib      = -L$(MPdir)/lib <- 修改为 Make 文件后缀
#
LAdir      = /home/forever/BLAS-3.8.0 <- BLAS 的安装路径
LAinc      =
LAlib      = /usr/local/lib/cblas_LINUX.a /usr/local/libblas_LINUX.a <- 库文件位置
#
#
HPL_OPTS    = -DHPL_CALL_CBLAS
#
#
CC          = /usr/local/openmpi/bin/mpicc <- 使用 which mpicc 找到该位置
CCNOOPT   = $(HPL_DEFS)
CCFLAGS    = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops <- 编译参数
#
# On some platforms, it is necessary to use the Fortran linker to
# the Fortran internals used in the BLAS library.
#
LINKER     = /usr/local/openmpi/bin/mpif77 <- 使用 which mpif77 找到该位置
LINKFLAGS  = $(CCFLAGS)

```

图 1.4: Make 文件修改

在保存后执行 `makearch = xxx` 进行安装即可，如过果无误应该会在 `bin/Linux - PII_CBLAS` 中找到一个 `xhpl` 文件，留待集群配置完成后再使用。至此环境配置完成。

## 1.5 集群网络和 ssh 免密登陆配置

将此前的虚拟机复制四份，注意重新生成 mac 地址，并且在网络部分选择 NAT 网络，如果设置正确的话，使用 `ipaddr` 指令应该能看到各虚拟机获得了不同的 IP，接下来检验一下网络的连通性：

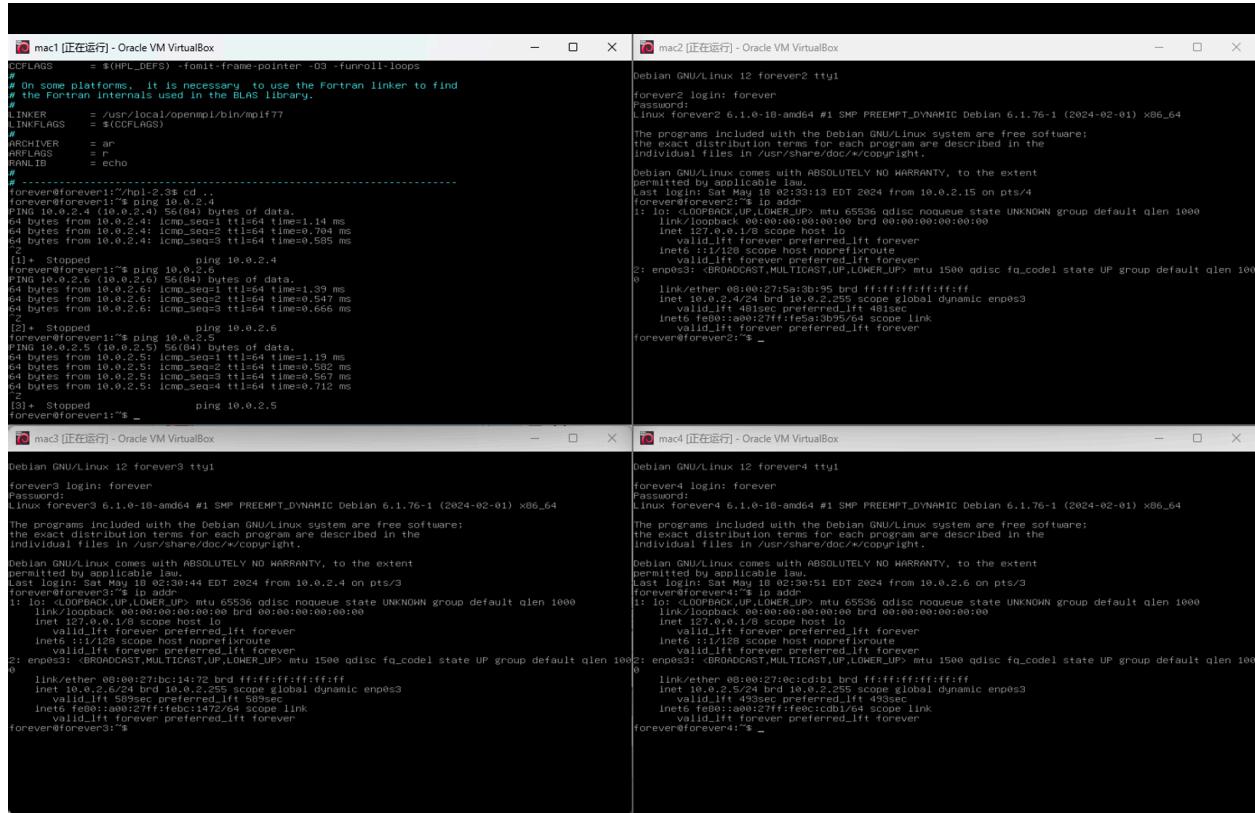


图 1.5: 连通性测试

在正式开始配置 ssh 密钥和免密登陆之前，让我们先来做一点准备工作避免在登录时出现 `PermissionDenied` 问题，在所有虚拟机下输入以下指令来开启允许 ssh 远程登录：

### Code Snippet 1.5.1 ▶ 准备工作

```

1 # 进入配置文件
2 vim /etc/ssh/sshd_config
3 # 编辑 PermitRootLogin 和 PasswordAuthentication 两项为 yes
4 PermitRootLogin yes
5 PasswordAuthentication yes
6 esc :wq
7 /etc/ssh/init.d restart

```

```

8 # 修改 hostname 避免 hosts 中出现重复，在所有机器上分别执行
9 hostnamectl set-host-name name1/2/3/4
10 systemctl restart network
11 # 修改 hosts 方便后续使用缩写
12 vim /etc/hosts
13 # 添加所有机器 ip
14 xxx.xxx.xxx.xxx name1
15 xxx.xxx.xxx.xxx name2
16 xxx.xxx.xxx.xxx name3
17 xxx.xxx.xxx.xxx name4

```

在确保完成准备工作后，开始配置 ssh 密钥，首先对实现免密登陆需要的文件结构做简单介绍：

### Technique 1.5.2 ▶ ssh 密钥配置要求

在一台机器向目标机器发起 ssh 请求时，如果 `/.ssh/authorized_keys` 中存在目标密钥 `id_rsa.pub` 则可以直接通过密钥认证实现免密登陆，因此我们接下来的目标就是让每台机器的 `authorized_keys` 文件夹中都保存有包括自己在内的所有 rsa 密钥。

让我们从首先在每台机器上执行以下命令：

### Code Snippet 1.5.3 ▶ 生成密钥

```

1 # 生成本机密钥并移入本机授权密钥文件夹
2 ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
3 cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys
4 # 连接本机进行检验
5 ssh localhost # 首次登陆需要输入 yes

```

接下来我们利用 ssh 自带的 `ssh-copy-id` 和 `scp` 指令分别将 2, 3, 4 虚拟机密钥移入虚拟机 1，再将拥有所有密钥的虚拟机 1 的授权文件夹传输给剩下的机器：

### Code Snippet 1.5.4 ▶ 跨虚拟机免密登陆

```

1 # 在虚拟机 2, 3, 4 上各执行一次，需要输入用户名和密码
2 ssh-copy-id -i ~/.ssh/id_rsa.pub name1
3 # 在虚拟机 1 上执行，需要输入用户名和密码
4 scp ~/.ssh/authorized_keys name2:~/.ssh/
5 scp ~/.ssh/authorized_keys name3:~/.ssh/
6 scp ~/.ssh/authorized_keys name4:~/.ssh/

```

```
7 # 现在在任意虚拟机上执行 ssh 指令来进行检验  
8 ssh name1/2/3/4 # 第一次登陆需要输入 yes
```

需要注意以上过程是以用户为单位的，因为我们希望在普通用户下进行后续测试，因此务必在 `/home/username` 进行操作，避免在管理员用户操作完后切换用户发现不可用的问题。

# 集群测试

在进行性能测试之前，让我们先确认一下 OpenMPI 能够在我们的集群下正确执行：

## Code Snippet 2.0.1 ▶ 可用性检验

```

1 # 随便选个路径（只要你记得住，此处以主目录为例）新建 hostfile
2 vim /home/forever/hosts
3 # 输入以下内容
4 name1 slots=x #x 为你分配的核数
5 name2 slots=x
6 name3 slots=x
7 name4 slots=x
8 esc :wq
9 # 测试查看上线时间
10 mpirun --hostfile hosts uptime --prefix /usr/local/openmpi #openmpi 的安装目录

```

关于为什么需要加入 `--prefix`, 尽管在每个节点的环境变量都加入了 `PATH` 和 `LD_LIBRARY_PATH`, 但仍然出现了找不到文件夹的目录，在查询文档后发现可以指定绝对路径来运行（当然，确保你的每台虚拟机在 OpenMPI 这个问题上有相同的文件结构），在完成该指令后，如果没有意外，你会得到这样的输出：

```

forever@forever1: ~$ ls /usr/local
bin etc games include lib man openmpi sbin share src
forever@forever1: ~$ mpirun --hostfile hosts --prefix /usr/local/openmpi uptime
07:23:44 up 2:42, 1 user, load average: 0.00, 0.00, 0.00
07:23:44 up 2:42, 1 user, load average: 0.00, 0.00, 0.00
07:23:44 up 2:42, 1 user, load average: 0.00, 0.00, 0.00
07:23:44 up 2:42, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 38 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 38 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 38 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 38 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00
07:23:43 up 40 min, 1 user, load average: 0.00, 0.00, 0.00

```

图 2.1: 集群上线时间

还记得之前提到的 *xhpl* 文件吗（在配置 HPL 后生成的），让我们开始运行他来进行测试：

### Code Snippet 2.0.2 ▶ xhpl 测试

```
1 cd /home/username/hpl-2.3/bin/Linux_PII_CBLAS
2 mpirun ./xhpl --hostfile /home/username/hosts --prefix /usr/local/openmpi >
   result.txt # 运行且输出到 result.txt 中
```

结束以上操作后不出意外你会得到一个巨长无比的输出文件，大概长这样：

```
mac1 正在运行 - Oracle VM VirtualBox

||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 1.72533487e-02 ..... PASSED
=====
T/V      N    NB    P    Q        Time       Gflops
=====
WR00R2D4  29     1    2    2        0.00      4.5329e-01
HPL_pdgesv() start time Wed May 22 07:32:38 2024
HPL_pdgesv() end time   Wed May 22 07:32:38 2024

||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 1.72533487e-02 ..... PASSED
=====
T/V      N    NB    P    Q        Time       Gflops
=====
WR00R2R2  29     1    2    2        0.00      4.2663e-01
HPL_pdgesv() start time Wed May 22 07:32:38 2024
HPL_pdgesv() end time   Wed May 22 07:32:38 2024

||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 1.72533487e-02 ..... PASSED
=====
T/V      N    NB    P    Q        Time       Gflops
=====
WR00R2R4  29     1    2    2        0.00      4.5840e-01
HPL_pdgesv() start time Wed May 22 07:32:38 2024
HPL_pdgesv() end time   Wed May 22 07:32:38 2024

||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 1.72533487e-02 ..... PASSED
=====
T/V      N    NB    P    Q        Time       Gflops
=====
WR00R2R8  29     1    2    2        0.00      4.5840e-01
HPL_pdgesv() start time Wed May 22 07:32:38 2024
HPL_pdgesv() end time   Wed May 22 07:32:38 2024

'result.txt' 8693L, 508564B                                208.1          2%
```

图 2.2: 运行 xhpl 的输出

大概是包括了一些数学表达式和他们的运算结果、时间之类的。例如第一个测试块大致包含了一下信息：

- PASSED: 代表该测试块通过
- $N = 29$ ,  $NB = 1$  代表共 29, 分块 1
- $P = 1$ ,  $Q = 1$  代表  $P$ ,  $Q$  的取值情况
- $Time = 0.00Gflops = \dots$  代表时间和运算速度

大概如此, 此外还可以修改 *HPL.dat* 来调整测试的规模和细节从而得到不同情况下的性能表现, 首先做一个结果显而易见的调整——调整数据规模  $N$ :

```

||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 8.97025337e-03 ..... PASSED
=====
T/V          N      NB      P      Q           Time           Gflops
-----
WR00R2R4      290      1      2      2           0.00          9.3495e+00
HPL_pdgesv() start time Wed May 22 08:03:44 2024
HPL_pdgesv() end time   Wed May 22 08:03:44 2024

=====
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)= 8.97025337e-03 ..... PASSED
=====
T/V          N      NB      P      Q           Time           Gflops
-----
WR00L2L2      290      2      2      2           0.00          1.0491e+01
HPL_pdgesv() start time Wed May 22 08:03:44 2024
HPL_pdgesv() end time   Wed May 22 08:03:44 2024

```

图 2.4: 调整数据后的测试结果

```

HPLinpack benchmark input file
Innovative Computing Laboratory, University of T
HPL.out        output file name (if any)
6              device out (6=stdout,7=stderr,file)
4              # of problems sizes (N)
2900 3000 3400 3500  Ns

```

图 2.3: 调整数据规模  $N$  为原来 100 倍

事实上我选择了调整成 10 倍, 100 倍跑不出来, 太慢了, 然后我们可以(武断地)得出结论, 在其他参数不做任何调整的情况下,  $N$  越大, 花费的时间也越多, 下面是调整后新的测试结果:

此外还测试了调整分块数量  $NB$  造成的效果, 一个大致的结论是: 分块越多并不一定性能越高, 这与你 CPU 的数目和分配的资源都有关系, 但由于我没有尝试特别特别多的分块, 导致看起来似乎变成了多多益善:

T/V	N	NB	P	Q	Time	Gflops
WR00L2L2	290	20	2	2	0.00	2.4440e+01
HPL_pdgesv() start time Wed May 22 08:10:26 2024						
HPL_pdgesv() end time Wed May 22 08:10:26 2024						
$\ Ax-b\ _oo / (\text{eps} * (\ A\ _oo * \ x\ _oo + \ b\ _oo) * N) = 9.28610736e-03 \dots \text{PASSED}$						
T/V	N	NB	P	Q	Time	Gflops
WR00L2L4	290	20	2	2	0.00	2.4601e+01
HPL_pdgesv() start time Wed May 22 08:10:26 2024						
HPL_pdgesv() end time Wed May 22 08:10:26 2024						

图 2.5:  $NB = 20$ 

T/V	N	NB	P	Q	Time	Gflops
WR00L2L2	290	50	2	2	0.00	1.3751e+01
HPL_pdgesv() start time Wed May 22 08:13:11 2024						
HPL_pdgesv() end time Wed May 22 08:13:11 2024						
$\ Ax-b\ _oo / (\text{eps} * (\ A\ _oo * \ x\ _oo + \ b\ _oo) * N) = 1.31632151e-02 \dots \text{PASSED}$						
T/V	N	NB	P	Q	Time	Gflops
WR00L2L4	290	50	2	2	0.00	1.7467e+01
HPL_pdgesv() start time Wed May 22 08:13:11 2024						
HPL_pdgesv() end time Wed May 22 08:13:11 2024						

图 2.6:  $NB = 50$ 

在测试之外，查阅了一些资料，大概知道  $p = 1, q = n$  是相对来说最快的设置，可以基于这个参数设置来调整分块的数量。

# 附录

## A.1 自我介绍

- 姓名: 洪奕迅
- 班级: 工信 2319
- 专业: 信息安全
- 相关爱好经历: 比较爱折腾 (?) 比如自己装了好多台电脑, 还喜欢拆东西。然后高中有一段短暂的 oi 经历。对超算感兴趣 (主要是好奇)。成绩还可以, 起码数学相关课程学的都不差 ()。

## A.2 代码

所有代码及配置文件已上传至本地克隆仓库: ForeverHYX