

# **HPC101\_2025**

## **Lab1**

2025/5/20--2025/7/2

3240104875

王耀

## 任务 1:

### OpenMPI:

第一个命令: `../configure --prefix=<path> 2>&1 | tee config.out`

在从源代码构建 **opemMPI** 的过程中，安装配置时一直在循环安装过程，不停止。后来发现安装依赖的有几个文件可能由于是 **WSL** 所以没有，去补充下载了。

但是在安装过程中遇到了一直循环反复开始编译的问题，后来发现是分成好几个片段编译，并没有在循环，只是很像。

第二个命令: `make`

执行时间有点长

```
MCA pml obl: parameter "pml_obl_free_list_inc" (current value:
int)
"60", data source: default, level: 9 dev/all, type:
int)
MCA pml obl: parameter "pml_obl_priority" (current value: "20",
data source: default, level: 9 dev/all, type: int)
MCA pml obl: parameter "pml_obl_send_pipeline_depth" (current
value: "3", data source: default, level: 9 dev/all,
type: int)
MCA pml obl: parameter "pml_obl_recv_pipeline_depth" (current
value: "4", data source: default, level: 9 dev/all,
type: int)
MCA pml obl: parameter "pml_obl_max_rdma_per_request" (current
value: "4", data source: default, level: 9 dev/all,
type: int)
MCA pml obl: parameter "pml_obl_max_send_per_range" (current
value: "4", data source: default, level: 9 dev/all,
type: int)
MCA pml obl: parameter "pml_obl_unexpected_limit" (current
value: "128", data source: default, level: 9
dev/all, type: unsigned_int)
MCA pml obl: parameter "pml_obl_use_all_rdma" (current value:
false, data source: default, level: 5
tuner/detail, type: bool)
Use all available RDMA btls for the RDMA and RDMA
pipeline protocols (default: false)
Valid values: 0|false|disabled|no|n,
1|true|enabled|yes|y
MCA pml obl: parameter "pml_obl_allocator" (current value:
"bucket", data source: default, level: 9 dev/all,
```

### BLAS:

Make

### CBLAS:

Make，但是出现了问题。

```
c_sblat1.f:214:48:
214 |          CALL STEST1(SNRM2TEST(N,SX,INCX),STEMP,STEMP,SFAC)
    |                                     1
Error: Rank mismatch in argument 'strue1' at (1) (scalar and rank-1)
c_sblat1.f:218:48:
218 |          CALL STEST1(SASUMTEST(N,SX,INCX),STEMP,STEMP,SFAC)
    |                                     1
Error: Rank mismatch in argument 'strue1' at (1) (scalar and rank-1)
make[1]: *** [Makefile:132: c_sblat1.o] Error 1
make[1]: Leaving directory '/mnt/d/HPC101_2025/lab1/cblas/CBLAS/testing'
make: *** [Makefile:180: alltst] Error 2
```

**fortran** 编译器的变量类型不匹配

在重新阅读 **README**，发现我没配置好路径依赖和选 **LINUX** 的文件。

在配置好之后，变量类型不匹配的错误再次出现。我去源代码里看了看，似乎 **STEMP** 是个只有一个元素的数组，但是定义的函数要求是一个标量，但是这是第三方的源代码，我不应该擅自修改。询问 **AI**，得知是新版 **gfortran** 不许可这样的传递参数方式，需要增加编译

条件：宽松的编译。

## HPL:

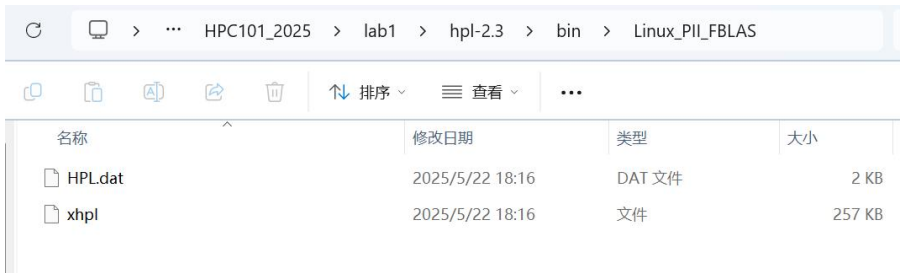
配置路径发现 `openmpi` 安在了 D 盘，不好调用，需要重新安装在 WSL 的 `/usr/local` 下面。

配置 HPL 的 `makefile`。Make 说找不到 `make.inc` 文件。但是在解压的时候没有这个文件。我跟着报错信息进入到了报错时所在的文件夹，发现了 `Make.Inc`。

我尝试了直接改那个路径下的 `makefile`，但是这个 `makefile` 是生成的，改了也会被新生成的替换掉。

再详细地读报错信息。报错信息说是在 `include make.inc` 时没找到，也没找到生成的规则，所以就创建了一个 `0kb` 的 `make.inc`。

最终发现原因竟然是因为我的 `make.Linux_PII_FBLAS` 里的一个路径配置不规范。



## 任务二

### 搭建 docker

我想利用 `docker`，因为虚拟机实在太大了。

安装这个过程实在是很长，所以我决定先生成一个安装好的镜像，导出之后作为镜像基础，在构建网络。

不过 `openmpi` 在容器里配置的好快，是因为任务一我是在 WSL 写入 windows 的原因吗？因为之前用过 `docker`，这一部分很快。

现在我把容器提取成镜像，作为 `dockerfile` 的载入。

### 测试 HPL

提示我建议不要以 `root` 身份运行，所以我在主节点增加了一个叫 `fiona` 的用户，再测试。

```
PMIx was unable to find a usable compression library
on the system. We will therefore be unable to compress
large data streams. This may result in longer-than-normal
startup times and larger memory footprints. We will
continue, but strongly recommend installing zlib or
a comparable compression library for better user experience.
```

You can suppress this warning by adding "pcompress\_base\_silence\_warning=1"

to your PMIx MCA default parameter file, or by adding

"PMIX\_MCA\_pcompress\_base\_silence\_warning=1" to your environment.

说没有压缩库，会很慢。但是这个警告也能无视。

这也算一个优化途径，先测测没有 **zlib** 的。

**hpl** 要求能够免密 **SSH**，而我在 **ssh** 的时候发现还是需要密码的。好像是 **ubuntu** 镜像默认不允许 **ssh** 访问 **root**？可是我已经配置了 **fiona** 用户可以相互访问

难道是 **Hostfile** 设置有问题吗？

```
fiona@node01:/$ mpirun -n 6 mpi-hello-world
-----
PMIx was unable to find a usable compression library
on the system. We will therefore be unable to compress
large data streams. This may result in longer-than-normal
startup times and larger memory footprints. We will
continue, but strongly recommend installing zlib or
a comparable compression library for better user experience.

You can suppress this warning by adding "pcompress_base_silence_warning=1"
to your PMIx MCA default parameter file, or by adding
"PMIX_MCA_pcompress_base_silence_warning=1" to your environment.
-----
Hello from rank 2 of 6
Hello from rank 3 of 6
Hello from rank 5 of 6
Hello from rank 4 of 6
Hello from rank 1 of 6
Hello from rank 0 of 6
fiona@node01:/$
```

可以发现，按照官网的指引写的一个文件是能够运行的，单机安装的 **mpi** 是没有问题的。

本来想写个脚本配置 **ssh**，但脚本里有两个交互式命令不会处理，但是反正容器数量少，最终手动。

但是即使都配置了 **ssh**，还是很长时间没有反应，**HPL.Dat** 的问题规模也不是很大。

```
=====
|Ax-b|_oo/(eps*(|A|_oo*|x|_oo+|b|_oo)*N)= 2.29176713e-02 ..... PASSED
=====
T/V          N    NB    P    Q          Time          Gflops
-----
MR00R2R2      35     4     4     1          0.00          7.3776e-01
HPL_pdgesv() start time Fri May 23 17:06:29 2025
HPL_pdgesv() end time   Fri May 23 17:06:29 2025

=====
|Ax-b|_oo/(eps*(|A|_oo*|x|_oo+|b|_oo)*N)= 2.29176713e-02 ..... PASSED
=====
T/V          N    NB    P    Q          Time          Gflops
-----
MR00R2R4      35     4     4     1          0.00          7.7240e-01
HPL_pdgesv() start time Fri May 23 17:06:29 2025
HPL_pdgesv() end time   Fri May 23 17:06:29 2025

=====
|Ax-b|_oo/(eps*(|A|_oo*|x|_oo+|b|_oo)*N)= 2.29176713e-02 ..... PASSED
=====
Finished      864 tests with the following results:
              864 tests completed and passed residual checks,
               0 tests completed and failed residual checks,
```

可以看到单机运行 **xhpl** 是可以的，说明问题就出在容器的连接问题上。那这个问题到底发生在哪，又应该如何解决呢？难道是每个节点没有 **--allow-root-run** 导致进程不进行吗？

我为每个节点都设置了 **fiona** 用户并建立了互联的 **ssh** 连接，但是还是遇到了一模一样的问题，只输出一个没有 **zlib** 的警告，其他什么也不动。

```
在伟大的 chatGPT 的建议下，用一个 mpirun --hostfile hostfile -np 4 \
--mca plm_base_verbose 10 \
--mca ras_base_verbose 10 \
--mca btl_base_verbose 100 \
--mca plm_rsh_agent "ssh -v -o StrictHostKeyChecking=no" \
-wdir /hpl-2.3/bin/Linux_PII_FBLAS\
./xhpl
```

详细地输出了程序运行过程中的操作。根据大段的错误信息来看，应该是访问不到 HPL.dat 文件。

```
fiona@node01:/$ ssh docker-node02-1 ls -l HPL.dat
ls: cannot access 'HPL.dat': No such file or directory
fiona@node01:/$ ssh docker-node02-1 ls -l ./hpl-2.3/bin/Linux_PII_FBLAS/HPL.dat
ls: cannot access './hpl-2.3/bin/Linux_PII_FBLAS/HPL.dat': No such file or directory
fiona@node01:/$ ssh docker-node02-1 ls -l /hpl-2.3/bin/Linux_PII_FBLAS/HPL.dat
-rw-r--r-- 1 root root 1133 May 23 13:22 /hpl-2.3/bin/Linux_PII_FBLAS/HPL.dat
```

这个输出或许能反应一些问题。。。。

```
fiona@node01:/$ ssh docker-node02-1 'hpl-2.3/bin/Linux_PII_FBLAS/xhpl -f /hpl-2.3/bin/Linux_PII_FBLAS/HPL.dat'
HPL ERROR from process # 0, on line 299 of function HPL_pdninfo:
>>> cannot open file HPL.dat <<<
```

```
[node02:00374] *** Process received signal ***
[node02:00374] Signal: Segmentation fault (11)
[node02:00374] Signal code: Address not mapped (1)
[node02:00374] Failing at address: (nil)
[node02:00374] [ 0] /lib/x86_64-linux-gnu/libc.so.6(+0x45810) [0x7fcc4909b810]
[node02:00374] [ 1] /lib/x86_64-linux-gnu/libc.so.6(_IO_fclose+0x14) [0x7fcc490e1844]
[node02:00374] [ 2] /hpl-2.3/bin/Linux_PII_FBLAS/xhpl(+0x602e) [0x55eb28fe502e]
[node02:00374] [ 3] /hpl-2.3/bin/Linux_PII_FBLAS/xhpl(+0x24e6) [0x55eb28fe14e6]
[node02:00374] [ 4] /lib/x86_64-linux-gnu/libc.so.6(+0x2a338) [0x7fcc49080338]
[node02:00374] [ 5] /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0x8b) [0x7fcc490803fb]
[node02:00374] [ 6] /hpl-2.3/bin/Linux_PII_FBLAS/xhpl(+0x2ca5) [0x55eb28fe1ca5]
[node02:00374] *** End of error message ***
```

GPT 推测说可能 xhpl 不能接受 -f 的输入，所以只能在工作目录下寻找 HPL.dat 文件。那我就把这个文件都复制到 /home/fiona 下。

还是没有用。Ssh docker-node02-1 mpirun -n 6 xhpl 倒是成功运行了。

发现 MPI 在通讯的时候建立了五个 daemon，是因为它认为需要启动五个 orted 来守护进程，而后因为我自作主张，在 hostfile 里写的是 docker-node01-1 这种格式的，本来认为这样能防止 docker 内部解析 node 出错，但是反而让 openmpi 根据 docker 网络中的 node1,docker-node0x-x 这五个不同的名称给我解析了五个节点，但是只有四个节点参与了运算，最后一个安全 orted 迟迟没有建立，就一直不运行。

我把 node 之间的 ssh 重新建立了一遍，终于跑起来了。但是跑的非常慢，可能和压缩有点关系。等会在节点里面搞个 zlib。

## 调整 HPL.dat

电脑配置如下：

设备名称	LAPTOP-RAP32OLU
处理器	Intel(R) Core(TM) i9-14900HX 2.20 GHz
机带 RAM	16.0 GB (15.7 GB 可用)
系统类型	64 位操作系统, 基于 x64 的处理器

GPU 英伟达 4060  
测试平台 docker, 4 容器

让我学习一下 HPL.dat 文件。

slots 是节点运行进程上限, 这个 .dat 文件里的 NB 应该是分块维度, 也就是分块矩阵是 NB\*NB 的。N 是问题规模, 即矩阵大小是 N\*N 的。P\*Q 指把进程分成 P 行 Q 列, 依次填入 slots 进行运算。 $|P|*|Q|$  是总进程数。每个进程分配到的子块大小即  $N/P*N/Q$ , 而 NB 是每个进程执行面板分解时操作的数据块大小, 应该尽量接近本地缓存容量。

P\*Q 是对所有的进程进行分块, 比如如果 hostfile 设定为四个 node 都有 slots=4, 那么总共有 16 个进程可用, 这是 P\*Q 最好为 16, 充分利用性能。

P\*Q 要尽量接近方形, 这是因为解  $AX=B$  的过程中要对分块的矩阵进行行选取最大元素, 列交换, 都有通信的必要。一般是越接近正方形越好, 或者  $P<Q$  的时候更好。

设最大矩阵为 n, 则有  $n*n*8<16*1024*1024*1024$ , 大约 46340 是我的电脑内存能够承受的最大 N, 但是考虑到还有其他应用占用内存, 而且 N 很大时算的很慢, 所以最后 N 最大取 2500。以下测试如不特殊说明, 均建立在 hostfile 内容为 node0x slots=5 的基础上, x=1,2,3,4

NB	1			8			16		
P*Q	2*2	1*4	4*1	2*2	1*4	4*1	2*2	1*4	4*1
Gflops	2.2e-5	4.3e-5	1.3e-5	3.1e-5	2.8e-4	3.6e-5	4.2e-5	7.6e-4	4.0e-5

N=35 (设置 4 个节点 slots=1)

NB	1			8			16		
P*Q	2*2	1*4	4*1	2*2	1*4	4*1	2*2	1*4	4*1
Gflops	2.1e-1	3.9e-1	3.2e-1	4.0e-1	1.5	5.0e-1	7.4e-1	7.8e-1	4.8e-1
Gflops 平均	1.5e-1	1.2e-2	1.3e-1	3.0e-1	9.1e-1	3.2e-1	5e-1	5.2e-1	3e-1

N=35

NB	25				200				1000			
P*Q	2*10	4*5	5*4	10*2	2*10	4*5	5*4	10*2	2*10	4*5	5*4	10*2
Gflops	1.2	0.2	0.18	0.1	5.7	0.25	0.29	0.1	3.2	0.23	0.8	0.1

N=2500

可见一般情况下, P\*Q 接近于正方形和 P 略小于 Q 时 Gflops 比较高, 推测原因是 hpl 过程中的分块矩阵算法对 P 和 Q 计算次数不同导致的。而且 N 对 Gflops 的影响比较大, 但是好像是有上限的。NB 适中比较好。

## 编译优化

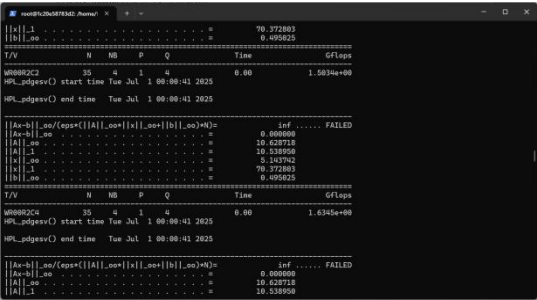
这部分与 bonus 有所重合, 优化 BLAS 的部分放在 bonus 里面。

以下测试都在单节点内部。我开了两个新的容器, 其中一个给 intel, 一个给 aocc。Intel 内部我下载了 icx, ifx 进行 CBLAS 的编译。需要修改 Makefile.in 文件中的编译器选项, 而且要配套的 Intel。过程中还是遇到了许多问题, 软连接不会被 clean, ifx 不认原本 make 文件中的参数, mpirun 冲突等等, 不再赘述。

如下是 `mpirun -np 6 ./xhpl` 的结果，HPL.dat 取初始状态。Gflops 均取最大值。

GCC -O3	ICX -O3	GCC -O2	GCC -Ofast
1.3	1.6	1.1	0.9

可见使用 intel 特化的 `icx` 和 `ifx` 进行编译还是有独特的优势的，就是 intel 内部自带的 `mpirun` 引起的冲突和 `blas`, `cblas`, `openmpi`, `hpl` 在安装时因为没考虑 `icx` 编译参数引起的错误和警告属实是比较烦人。



对于编译选项优化，`hpl` 中的编译选项是 `-O3`，几乎已经是最快的了。但是为什么 `-Ofast` 反而是最慢的一个？测了三次，最大的 Gflops 都在这 0.9 附近，此后即刻再跑 `-O3` 也还是 1.3 左右。

GPT 认为可能是 `-Ofast` 的优化的核心思想是牺牲严格的 IEEE 754 来换取速度提升，导致虽然单个浮点快了但是不稳定性和收敛变慢（`hpl` 测试还会收敛？那还哪来的测试通过一说？）。也可能过于激进的循环展开、向量化和指令调度使得寄存器压力过大，指令缓存压力，指令调度低效等。

但是无穷残差那一行并没有明显的大于 `-O3` 的值，所以我觉得应该不是不收敛重复计算的问题，可能还是激进的优化导致的反作用。

更换 BLAS 库见 Bonus 部分，这部分我会在课程提供的超算节点上完成。（真香）

## 运行环境优化

使用 `zlib` 并没有显著的提升 Gflops 值。

我尝试优化 `openMP` 绑核参数。经过了解，`OpenMP` 是一个线程的管理库，所以 `openMP` 是不会参与进程之间的通信的。而优化绑核参数，就是要防止同一进程中的不同线程在不同的 `cpu` 核之间分散地执行，甚至丢失数据，相互“抢核”，浪费时间。

似乎本身 `openmpi` 的程序中就有 `openmp` 的成分，所以我可能只需要调整环境变量看看 Gflops 有没有什么变化。

依据 `OpenMP 5.0` 的规范，`OMP_PLACES` 的取值有 `thread`（每个超线程当作独立的可分配单元），`core`（如有超线程，会都分配到一个物理核心），`sockets`（每个 `cpu` 插槽作为一个物理核心），`numa`（每个 `NUMA` 作为一个核心，`numa` 也是 `cpu` 插槽级别的东西），`explicit` 显示设置。`OMP_PROC_BIND` 的取值有 `false`（不绑定，任意调度），`true`, `master`（仅绑定主线程），`close`（按 `place` 列表顺序分配线程），`spread`（`place` 列表均匀分布）。

我在 `csdn` 上也了解到可以通过 `--bind-to-core` 和 `--bind-to-x` 来临时实现 `core` 和 `sockets/numa` 的效果。去 `openMPI` 的官网阅读了使用手册，找到了正确的“绑核参数”的测试方式。

下面测试的 `hostfile` 为 `node0x slots=1,x=1,2,3,4`，HPL.dat 中 `N=36`，`NB=9`，`P*Q=2*2`

参数--bind-to	core	none	numa	package	默认
-------------	------	------	------	---------	----

Gflops	2.1e-4	1.2e-1	1.4e-1	1.3e-1	2.1e-4
--------	--------	--------	--------	--------	--------

可见 `core` 是默认设置，绑定到 `cpu` 的逻辑核心。`None` 是任意调度，`package` 是绑定到一颗物理 `cpu`，`numa` 是一个或多个 `package` 和相关的内存系统组成的“非一致性内存访问框架”。

根据测试的结果，我怀疑初始绑定到 `core` 的设置导致某些核心任务堆积而其他核心早早完成，导致了算力空白。当我们绑定 `none` 任意分配，速度就快了很多，绑定到 `package` 和 `numa` 有所提升的原因可能是因为数据访问本地性越来越好，所以访问和计算就更快。

而另一个参数`--map-by`，负责的是 `mpi` 的进程绑定，也就是 `rank` 拓扑。

参数 <code>--map-by</code>	<code>core</code>	<code>package</code>	<code>numa</code>	<code>slot</code>	<code>node</code>	默认
Gflops(slots==1)	2.1e-4	1.2e-1	1.4e-1	1.3e-1	1.4e-1	2.1e-4
Slots==5	8.6e-1	8.0e-1	7.5e-1	8.8e-1	1.3e-1	8.6e-1

根据我在容器内和宿主机进行 `nproc` 命令的查看，发现他们能使用的 `cpu` 逻辑核数都是 32，这或许可以解释 `slots` 设定不同时 `rank` 拓扑参数所导致的不同结果。`Slots` 都为 1 时，`--map-by core` 成果最差，但是 `slots` 都为 5 时，反而表现得很好，这让我非常迷惑。现推测原因如下：

`Slots` 为 1 时，受限每个节点的进程限制，在按照 `core` 分配进程时依据 `round robin`，为四个节点轮流分配，导致最终 `mpi` 通信成本很高，这部分广播和通信会影响 `Gflops` 值。`Slots` 为 5 时，因为每个节点并不受到 `cpu` 使用上的限制，所以四个进程很可能被放在了同一个节点内，至少很小概率分散在四个节点当中，所以 `mpi` 通信成本比较低。

至于 `slot` 表现最好，也是同理。`Slot` 为 1，和 `node` 时间接近，因为此时每个 `node` 只允许一个 `slot`，二者近似等价。`Slot` 为 5，二者差距很大，`node` 和 `slot=1` 时相近，再次印证了我们的猜想。

## Bonus · 集群搭建

对于正确为每台机器设置主机名，并能从主机名解析出 `ip` 地址，这个在 `docker` 网络中很容易实现，上面测试 `hpl` 也依赖于这一点，不再赘述。如果实在真正的集群中，似乎哪位学长上课提到过，维护一个专门的内部 `DNS` 主机即可，所有的访问通过这台机器匹配。

要实现集群间文件共享，对于 `docker` 而言最好的方法是直接挂载 `volume`。但是在一个真正地集群中显然没有 `docker` 这种方便的解决方案，所以我还是决定尝试 `NFS`。我以 `node01` 作为服务端，其他三台机器作为客户端，进行 `nfs` 文件共享的尝试。

我的 `node02~04` 的 `ip` 地址为 `172.20.0.3~5`，所以我把 `exports` 中设定一个 `/home/enovo/fiona 172.20.0.0/16`，我尝试将 `/home/fiona/nfs` 这个文件夹进行共享。

但是系统拒绝了我的设置。好像 `docker` 的 `overlay` 的文件系统，`overlay` 本身即是一种联合挂载机制，所以我做的事情类似于挂载一个挂载文件，当然是不可能成功的。那我就只能在宿主机上安装 `nfs` 服务端了。但是在 `node01` 上挂载的过程中疑似被防火墙挡了。

`WSL` 系统没法作为 `nfs` 服务端。我的 `windows` 系统又偏偏是家庭版，不支持 `nfs`，但是支持 `SMB 1.0/CIFS`，但是这个好像不是很稳定。那么按理来说我升级成专业版然后再使用 `nfs` 就可以了。但是学校的正版平台不提供 `win11` 的专业版转化，重装系统的话这个电脑上文件不易备份。

按理说是 `rpcbind` 启用了之后按照 `lab1` 里的指导就可以完成。我尝试开一个 `linux` 的虚拟机作为服务端试试。

在测试中，我发现想要让虚拟机的文件夹类型不为 `overlay`，需要让它放到实际的磁盘

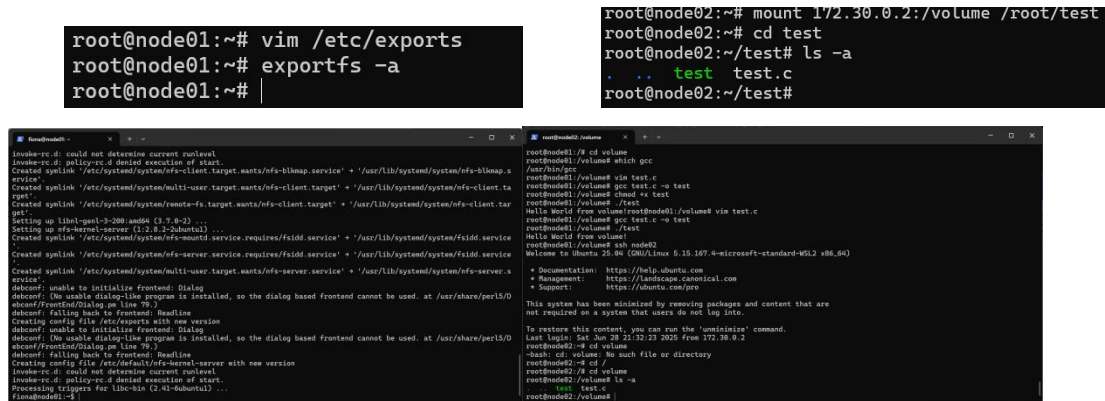


上。在 VMware 创建新虚拟机的设置中我确实找到了这个选项，但是由于我对磁盘的构造是在不是很了解，不是很敢轻举妄动，所以最后也没能找到能作为 nfs 服务端的设备。

我在挂完 volume 之后发现 volume 中的共享文件夹可以作为 nfs 的导出文件夹，虽然这么做比较幽默，但是也算是找到了一个解决方法。只有 volume 这种直接放在切实的磁盘的文件夹才可以被 nfs 选中，而其他的 overlay 的文件是不可以的。第四张图是 volume 共享成功的记录。第二张图是以 volume 共享到 node02 的 /root/test 的记录。

```
root@node01:~# vim /etc/exports
root@node01:~# exportfs -a
root@node01:~# |

root@node02:~# mount 172.30.0.2:/volume /root/test
root@node02:~# cd test
root@node02:~/test# ls -la
.  ..  test  test.c
root@node02:~/test#
```



我在部署 NIS 时又遇到了类似的问题，禁止我修改 domainname。但是这个可以在创建集群的时候在 .yaml 文件里增加一个 privileged: true 的设定解决。但是 docker 容器没有 systemctl 命令实在是非常难受的一件事。

我按照官网和 github 的指导，大致上修改了我能找到的配置文件，和我知道怎么执行的命令。配置了一下发现 node02 进行 ypbind 不能 bind 到 node01 这个服务端上去。执行输出 debug 细节的命令，发现说 node01 服务的域不是 cluster.local。我根据 gpt 的提示去看了 makefile，确实它设置的 DOMAIN 是 database \pwd\，应该是 yp。

但是我在把各处的 domain 都修改为 yp 之后还是出现了同样的问题。而且我发现之前我设置为 cluster.local 时也同样产生了一个文件夹，所以问题可能不在 makefile 上。

```
root@node02:~# ypbind -debug -verbose -foreground
83: parsing config file
83: Trying entry: ypserver 172.30.0.2
83: parsed ypserver 172.30.0.2
83: add_server() domain: yp, host: 172.30.0.2, slot: 0
83: [Welcome to ypbind-mt, version 2.7.2]

83: ping interval is 300 seconds

83: Register ypbind for inet,udp
83: Register ypbind for inet,tcp
83: Register ypbind for inet6,udp
83: Register ypbind for inet6,tcp
85: ping host '172.30.0.2', domain 'yp'
85: domain 'yp' not served by '172.30.0.2'
```

在检查之后，我确认了 node01 中的 domainname 确实是 yp，而且 ypserv 确实已经启用。在 GPT 的指导下，我发现从 node02 执行 rpcinfo -t node01 100004 2 连接超时，-u 连接也超时。但是在 node01 上能明显地看到，有这个服务。说明 node02 被拦截了。GPT 说可能被防火墙拦了，也可能 docker 网络对 NIS 不友好。

我现在尝试暂时关掉防火墙，node02 果然过去了。那我现在尝试在 node01 管理 node02 的用户信息。在 /var/yp/yp 下 yppush -d yp \*.by\*，我又遇到了新的问题。说不能连接到服务此域的服务器。这似乎是把更新的 .by\* 文件推送给从服务器，那我如何让 node02 接收到更新？

呃，防火墙设置回来了，node02 仍然访问不到了。

并不是回来了，而是能访问端口但是访问不了服务。现在问题有两种可能性：一是服务端的服务层有问题，一种是服务端认为自己服务的域名不是 yp。但是这怎么可能呢？

我在服务端的 /etc/yp.conf 文件里也配置了 domain yp server 172.30.0.2，这次有了

不一样的报错信息。我还发现/etc/hosts 这个文件在每次 docker 重新启动后都会重置为只有本节点信息的文件，这会导致 node02 无法访问 node01，这二者可能是我 nis 始终失败的原因。

```
root@node02:~# ypwhich -d yp
node01
root@node02:~# id fiona
uid=1002(fiona) gid=1002(fiona) groups=1002(fiona)
root@node02:~#

root@node02:~# ypbind session optional pam_mkhomeedit
root@node02:~# ypbind -debug -verbose
38: parsing config file
38: Trying entry: domain yp server node01
38: parsed domain 'yp' server 'node01'
38: add_server() domain: yp, host: node01, slot: 0
38: [Welcome to ypbind-mt, version 2.7.2]

38: ping interval is 300 seconds

38: Register ypbind for inet,udp
38: Register ypbind for inet,tcp
38: Register ypbind for inet6,udp
38: Register ypbind for inet6,tcp
40: ping host 'node01', domain 'yp'
40: NIS server for domain 'yp' set to 'node01'
40: Update binding file for 'yp' with 'node01'
40: NIS server for domain 'yp' is 'node01'
38: ypbindproc_domain_3_svc (yp) from 127.0.0.1 port 641
38: Ping active server for 'yp'
38: YPBINDPROC_DOMAIN: server 'node01', port 814
38: Status: YPBIND_SUCC_VAL
```

fiona 本来我 node01 中的测试节点，可见 NIS 已经安装完成。下面可以看到，已经能够在 node01 上修改 fiona 用户密码并在 node02 接收了。我还在 github 上学到了自动创建家目录的小技巧，Go to nano /etc/pam.d/common-session and add this at the end session optional pam\_mkhome.so skel=/etc/skel umask=077

```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

PS C:\Users\Lenovo\Desktop> docker exec -it nis-node02-1 bash
rpcbind: another rpcbind is already running. Aborting
root@node02:~# ypwhich -d yp
node01
root@node02:~# id fiona
uid=1002(fiona) gid=1002(fiona) groups=1002(fiona)
root@node02:~# ypcat passwd
ubuntu:!:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
cheer:!:1001:1001:./home/cheer:/bin/sh
fiona:!:1002:1002:./home/fiona:/bin/sh
nobody:!:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
root@node02:~# ypcat passwd | grep fiona
fiona:$y$j9T$iiU2zjER7zA5kP5/a0$1RPTle6.4E0cQLs1aR3L32yIleCtAQIbaS1gXh8x.:1002:1002:./home/fiona:/bin/sh
root@node02:~# ypcat passwd | grep fiona
fiona:$y$j9T$fd.L38tbZsslwmfQEtA85Q1$m829jxvW78/L1lb8rxqDy57ZNBNTCc2e6AnUjy8FF51:1002:1002:./home/fiona:/bin/sh
root@node02:~#

安装最新的 PowerShell，了解新功能和改进！https://aka.ms/PSWindows

PS C:\Users\Lenovo\Desktop> docker exec -it nis-node01-1 bash
rpcbind: another rpcbind is already running. Aborting
root@node01:~# passwd fiona
New password:
Retype new password:
passwd: password updated successfully
root@node01:~# cd /var/yp
root@node01:/var/yp# make
gmake[1]: Entering directory '/var/yp/yp'
Updating passwd.byname...
Updating passwd.byuid...
Updating hosts.byname...
Updating hosts.byaddr...
Updating netid.byname...
Updating shadow.byname... Ignored -> merged with passwd
gmake[1]: Leaving directory '/var/yp/yp'
root@node01:/var/yp# passwd fiona
New password:
Retype new password:
passwd: password updated successfully
root@node01:/var/yp# make
gmake[1]: Entering directory '/var/yp/yp'
Updating passwd.byname...
Updating passwd.byuid...
Updating shadow.byname... Ignored -> merged with passwd
gmake[1]: Leaving directory '/var/yp/yp'
root@node01:/var/yp#
```

## Bonus · 软件安装

不得不说，spack 管理环境是真香，但是 spack 安装也是真慢。一个 openmpi 安了好长时间，这就是 NP 问题+python 的恐怖之处吗。

利用 spack env 类管理 spack 环境，我用 srun 申请了计算节点的权限，然后进行了测试，结果如下，所用节点为 m600 四个进程。我把 openmpi 放到了所有环境都能用的层次，然后进行 hpl 和 blas 的构造。

我遇到了 mpirun 在 m600 上单节点可运行但是跨节点无法运行的问题。WARNING: Open MPI failed to TCP connect to a peer MPI process. This should not happen. 然后就 connection refused 了。

所以我决定直接单节点上测试，保持控制变量。

OpemBLAS	LAPACK	MKL	AOCL
2. 2e-1	2. 3e-1	2. 5e-1	Spack providers blas 中无 aocl

我的 spack 环境保留在了超算集群中。Spack 来管理环境，确实比 docker 创建好多容器

自己慢慢安装方便多了，也快多了。