金融大数据实验1：MPI集群搭建

姓名：盛祺晨　学号：191220093

目录

成果展示

代码

1.使用了MPI_Reduce改写了大数组各元素开平方求和

```
#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
int main(int argc, char** argv)
{
    MPI_Init(&argc, &argv);
    clock_t start, finish;
    double  duration;
    MPI_Status status;
    char message[100];
    int myid, P, source, C=0,numprocs;
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

```
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    int N;
    double SqrtSum=0.0,d,local=0;
    double* data;
    if(myid==0){ // 发送数据N
        printf("请输入N\n");
        scanf("%d",&N);
        start = clock();
        for(int otherprocess=1;otherprocess<numprocs;++otherprocess){
            MPI_Send(&N, 1, MPI_DOUBLE, otherprocess, 50, MPI_COMM_WORLD);
        }
    }
    else{ //接受数据N
        MPI_Recv(&N, 1, MPI_DOUBLE, 0, 50, MPI_COMM_WORLD, &status);
    }
    // 开N个double大小的数组
    data = (double*) malloc(N*sizeof(double)+5);
//-------------------------------------------------------------------
    for(int i =0;i<N;++i){
        data[i]=i*(i+1);
    }
    for(int i = myid; i<N;i+=numprocs)
    {
        data[i] = sqrt(data[i]);
        local += data[i];
    }
    free(data);data=NULL;  //释放空间
    /* MPI_Reduce 改写！！！*/
    MPI_Reduce(&local,&SqrtSum,1,MPI_DOUBLE,MPI_SUM, 0, MPI_COMM_WORLD);
    /* MPI_Reduce 改写！！！*/
    if(myid == 0){
        printf("SqrtSum: %lf\n",SqrtSum);
        finish = clock();
        duration = (double)(finish - start)/CLOCKS_PER_SEC;
        printf("花费时间 %f seconds\n", duration );
    }
    MPI_Finalize();
}
```

2.用MPI_Send和MPI_Recv改写了计算积分

```
#define N 100000000
#define a 10
#define b 100
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "mpi.h"
int main(int argc, char** argv)
```

```
{
    MPI_Init(&argc, &argv);
    clock_t start, finish;
    MPI_Status status;
    double  duration;
    int myid,numprocs;
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    --numprocs; /*数据分配时除去0号主节点*/
    int i;
    double local=0.0, dx=(double)(b-a)/N; /* 小矩形宽度 */
    double inte, x;
    if(myid==0) /* 规约所有节点上的累加和并送到主节点0 */
    {   /* 主节点打印累加和*/
        start = clock();
        double sum =0 ;
        for(int source = 1;source<=numprocs;++source){
            MPI_Recv(&inte, 1,MPI_DOUBLE, source,99,MPI_COMM_WORLD,&status);
            printf("从%d 进程获取返回值 %lf\n",source,inte);
            sum += inte;
        }
        printf("The integal of x*x*x in region [%d,%d] =%16.15f\n", a, b, sum);
        finish = clock();
        duration = (double)(finish - start)/CLOCKS_PER_SEC;
        printf( "花费时间 %f seconds\n", duration );
    }
    else
    {
        for(i=myid;i<N;i=i+numprocs) /*根据节点数目将N个矩形分为图示的多个颜色组*/
        { /*每个节点计算一个颜色组的矩形面积并累加*/
            x = a + i*dx +dx/2; /* 以每个矩形的中心点x值计算矩形高度 */
            local +=x*x*x*dx; /* 矩形面积 = 高度x宽度=y*dx */
        }
        MPI_Send(&local, 1, MPI_DOUBLE, 0, 99,MPI_COMM_WORLD);
    }
    MPI_Finalize();
    return 0;
}
```

结果-计算平方根和

```
(base) chengqichendeMacBook-Pro:MPI_Exp1 shengqichen$ mpirun -n 3 ./a.out
请输入N
10000
SqrtSum: 49999998.349880
花费时间 0.000333 seconds
(base) chengqichendeMacBook-Pro:MPI_Exp1 shengqichen$ mpirun -n 10 ./a.out
请输入N
10000
SqrtSum: 49999998.349880
花费时间 0.006780 seconds
```

可以看出，10进程明显慢于3，可能原因在后面问题分析第8个。

结果-计算积分

```
(base) chengqichendeMacBook-Pro:calsum.c shengqichen$ mpirun -n 10 ./a.out
从1 进程获取返回值 2777499.600300
从2 进程获取返回值 2777499.700200
从3 进程获取返回值 2777499.800100
从4 进程获取返回值 2777499.900000
从5 进程获取返回值 2777499.999900
从6 进程获取返回值 2777500.099800
从7 进程获取返回值 2777500.199700
从8 进程获取返回值 2777500.299600
从9 进程获取返回值 2777500.399500
The integal of x*x*x in region [10,100] =24997499.999099746346474
花费时间 0.068037 seconds
(base) chengqichendeMacBook-Pro:calsum.c shengqichen$ mpirun -n 2 ./a.out
从1 进程获取返回值 24997499.999095
The integal of x*x*x in region [10,100] =24997499.999095488339663
花费时间 0.281068 seconds
(base) chengqichendeMacBook-Pro:calsum.c shengqichen$
```

可以看出，计算结果基本没有变化，2进程的速度明显慢于10进程，进程数多对于加速有利。

集群搭建

1.搭建集群并用hello程序验证环境

```
root@host1:/# cd hello
root@host1:/hello# ls
hello  hello.c
root@host1:/hello# mpirun -np 10 -host host1,host2 hello
Unexpected end of /proc/mounts line `overlay / overlay rw,relatime,lowerdir=/v
lay2/l/M6LAWCPBTDHYZLYYGXCHSDORZW:/var/lib/docker/overlay2/l/5IV6MFIL6W5TTTJTS
d8261e0aaa2d4ad8f1d49c37afb5031150ed/diff,workdir=/var/lib/docker/overlay2/e16
root@host2's password:
Permission denied, please try again.
root@host2's password:
Unexpected end of /proc/mounts line `overlay / overlay rw,relatime,lowerdir=/v
lay2/l/M6LAWCPBTDHYZLYYGXCHSDORZW:/var/lib/docker/overlay2/l/5IV6MFIL6W5TTTJTS
b85b2d264f6f594e9c213ff9ed897c4c3e21/diff,workdir=/var/lib/docker/overlay2/c24
I am process 0. I recv string 'Hello World!' from process 1.
I am process 0. I recv string 'Hello World!' from process 2.
I am process 0. I recv string 'Hello World!' from process 3.
I am process 0. I recv string 'Hello World!' from process 4.
I am process 0. I recv string 'Hello World!' from process 5.
I am process 0. I recv string 'Hello World!' from process 6.
I am process 0. I recv string 'Hello World!' from process 7.
I am process 0. I recv string 'Hello World!' from process 8.
I am process 0. I recv string 'Hello World!' from process 9.
root@host1:/hello#
```

在host1中输入“mpirun -np 10 -host host1,host2 hello”，输入host2的密码，则会出现正确程序结果，虽然会有一段乱码，但是这个乱码据我查找的资料不会影响程序运行和输出，可能是版本问题。下同，故不再说明。

2.将单机的程序放到集群中run

1）大数组各元素开平方求和

运行指令 mpirun -np 3 -host host1,host2 sqrtSum 后得到结果

```
root@host2:/Exp1/sqrtSum# mpicc sqrtSum.c -std=c99 -o sqrtSum -lm
root@host2:/Exp1/sqrtSum# mpirun -np 3 -host host1,host2 sqrtSum
Unexpected end of /proc/mounts line `overlay / overlay rw,relatime,lowerdir=/va
/lib/docker/overlay2/l/YU5O7Y4C2CETAILLBIJAEOGNUL:/var/lib/docker/overlay2/l/LF
NKBT75NQCAC2L5I3EVGZND4:/var/lib/docker/overlay2/l/M6LAWCPBTDHYZLYYGXCHSDORZW:/
ar/lib/docker/overlay2/l/5IV6MFIL6W5TTTJTST3RPJ5MEH:/var/lib/docker/overlay2/l/
ZCTHKOVQXYT35NSUFWRCDZFML,upperdir=/var/lib/docker/overlay2/c24ca9c62ed5d5aba12
d28f932cb85b2d264f6f594e9c213ff9ed897c4c3e21/diff,workdir=/var/lib/docker/overl
y2/c24ca9c62ed5d5aba12cd28f932cb85b2d264f6f594e9c213ff9ed897c4c3e21/'
Unexpected end of /proc/mounts line `overlay / overlay rw,relatime,lowerdir=/va
/lib/docker/overlay2/l/OCMVWL3JRHCLRAVTLTPTDGVCYX:/var/lib/docker/overlay2/l/LF
NKBT75NQCAC2L5I3EVGZND4:/var/lib/docker/overlay2/l/M6LAWCPBTDHYZLYYGXCHSDORZW:/
ar/lib/docker/overlay2/l/5IV6MFIL6W5TTTJTST3RPJ5MEH:/var/lib/docker/overlay2/l/
ZCTHKOVQXYT35NSUFWRCDZFML,upperdir=/var/lib/docker/overlay2/e165f2efc91b7b49838
93b50a79d8261e0aaa2d4ad8f1d49c37afb5031150ed/diff,workdir=/var/lib/docker/overl
y2/e165f2efc91b7b49838293b50a79d8261e0aaa2d4ad8f1d49c37afb5031150ed/'
请输入N
1000
1进程花费时间 0.000006 seconds
2进程花费时间 0.000014 seconds
0进程花费时间 0.000367 seconds
SqrtSum: 499998.637703
total花费时间 0.000454 seconds
root@host2:/Exp1/sqrtSum#
```

2）计算积分

运行指令 mpirun -np 10 -host host1,host2 calsum 后得到结果

```
从1 进程获取返回值 2777499.600300
从2 进程获取返回值 2777499.700200
从3 进程获取返回值 2777499.800100
从4 进程获取返回值 2777499.900000
从5 进程获取返回值 2777499.999900
从6 进程获取返回值 2777500.099800
从7 进程获取返回值 2777500.199700
从8 进程获取返回值 2777500.299600
从9 进程获取返回值 2777500.399500
The integal of x*x*x in region [10,100] =24997499.999099746346474
花费时间 0.100437 seconds
root@host2:/Exp1/calsum# mpirun -np 3 -host host1,host2 calsum
```

运行指令 mpirun -np 3 -host host1,host2 calsum

```
从1 进程获取返回值 12498750.224775
从2 进程获取返回值 12498749.774325
The integal of x*x*x in region [10,100] =24997499.999099630862474
花费时间 0.508545 seconds
```

问题和解决思路

1.不会使用vi，换源多次粘贴失败。

解决思路：安装vim。

```
apt-get install vim
```

2.#include<stdio.h>失败。

解决方法：安装gcc

```
apk add build-base
```

3.scanf读取错误，不能运行或运行卡住

问题描述：把scanf放在了MPI_Init之前，发现print了多次（进程数量次），scanf读取错误，不能运行或运行卡住。

解决方法：

将scanf装在0进程中，只读一次，后通过send或bcast传入其他进程。

```
if(myid==0){ // 发送数据N
  printf("请输入N\n");
  scanf("%d",&N);
  start = clock();
}
```

4.ssh连接时输入正确的密码却连不了

解决方式：

1）重置密码，且网上说密码要8位以上含字母和数字（原先设置的密码过于简单，是aaaaa，不能操作），按照如下操作修改了root的密码，可以正常运行。

1、修改所有容器中root账户密码

ssh到远程主机时，首次需要密码访问，因此需要修改root账号密码。 密码必须要8位以上字母数字混合。

```
$>passwd
```

2、在spark30容器生成公私秘钥对

```
$>ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
```

3、在spark30使用ssh-copy-id复制公钥到远程主机

```
# 复制公钥到spark容器
$>ssh-copy-id root@192.168.231.31
$>ssh-copy-id root@192.168.231.40
$>ssh-copy-id root@192.168.231.41
```

2）可能是ssh服务问题，22端口没有开启

#sudo vim /etc/ssh/sshd_config

找到并用#注释掉这行：PermitRootLogin prohibit-password

新建一行 添加：PermitRootLogin yes

重启服务

#sudo service ssh restart

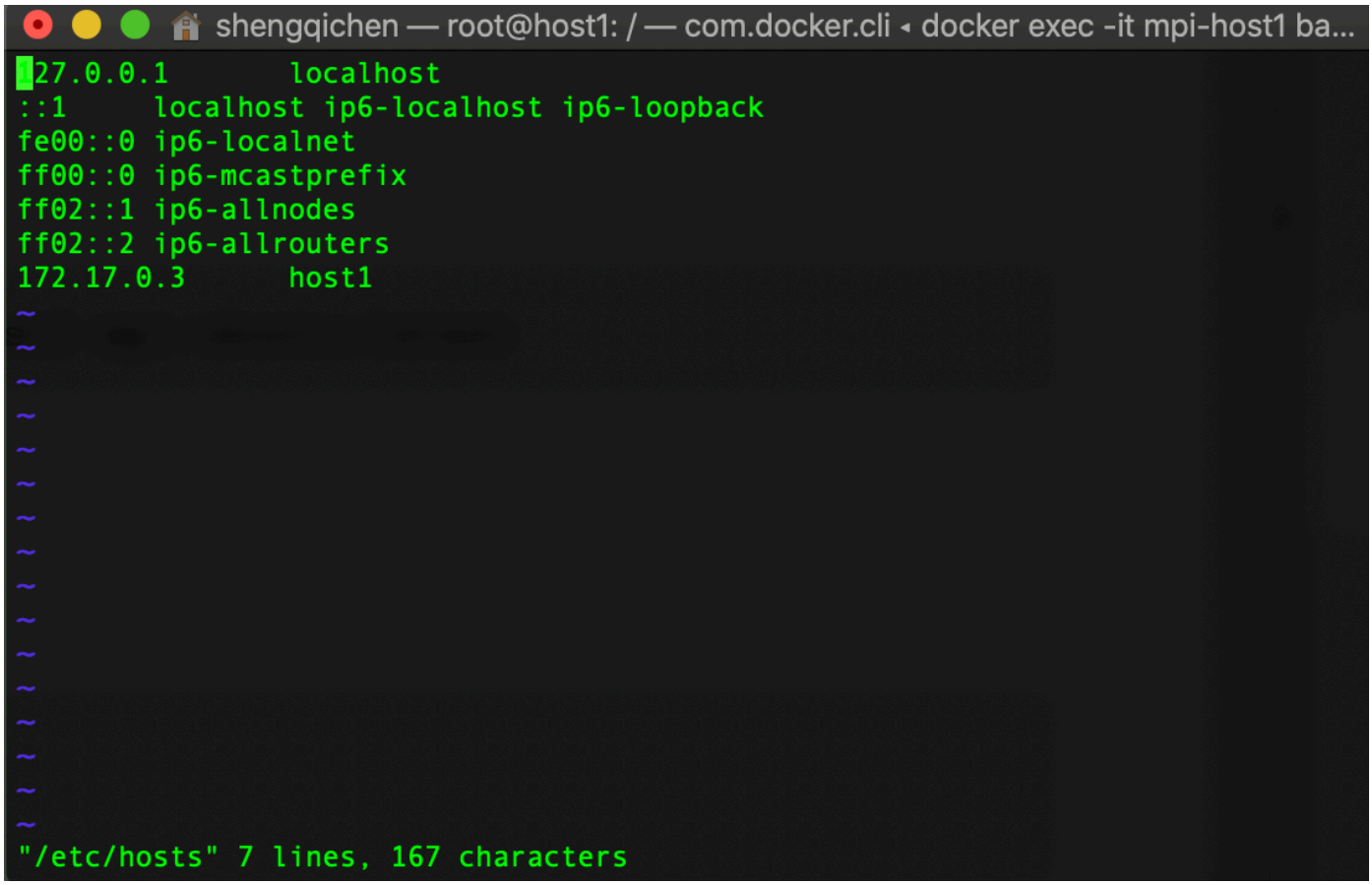
意思是：

PermitRootLogin yes　　允许root登录，设为yes。

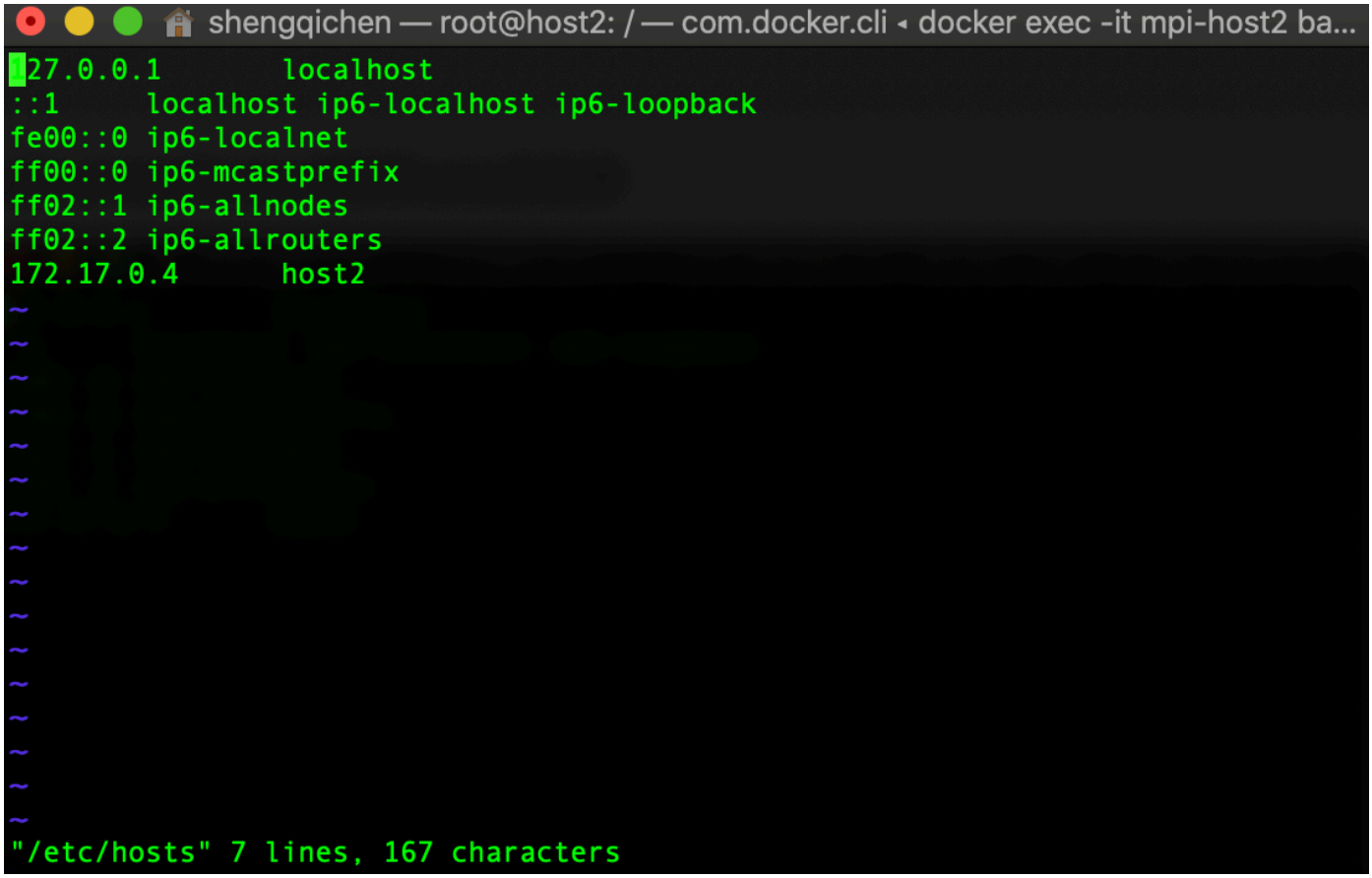
PermitRootLogin prohibit-password　允许root登录，但是禁止root用密码登录

很明显这行是需要被注释掉的！！！

5.sudo vi /etc/hosts #对host1用了ssh连接操作但是忘记在host2中连接。



解决方式：exit host1后，打开host2进行同host1中配置，将host1的地址加入hosts文件



6.gcc版本导致编译出错

问题描述：'for' loop initial declarations are only allowed in C99 mode，for循环中使用了for(int i=0 ;i .....)其中，int初始化需要在C99之后才能用。

```
    for(int i = myid; i<N;i+=numprocs)
    ^
sqrtSum.c:27:5: note: use option -std=c99 or -std=gnu99 to compile your code
```

解决方式：使用 mpicc sqrtSum.c -std=c99 -o sqrtSum 来编译sqrtSum.c文件

使用gcc编译代码是报出

error: 'for' loop initial declarations are only allowed in C99 mode

note: use option -std=c99 or -std=gnu99 to compile your code

错误，这是因为在gcc中直接在for循环中初始化了增量：

```
    for(int i=0; i<len; i++) {
    }
```

这语法在gcc中是错误的，必须先先定义i变量：

```
int i;
for(i=0;i<len;i++){

}
```

这是因为gcc基于c89标准，换成C99标准就可以在for循环内定义i变量了：

gcc src.c -std=c99 -o src

7.math.h中的函数sqrt编译出错

解决方式：-lm用于数学函数增加，更改编译语句为：

```
mpicc sqrtSum.c -std=c99 -o sqrtSum -lm
```

因为math位于libm.so库文件中，-lm选项告诉编译器，我们程序中用到的数学函数要到这个文件里找。

8.开的进程越多，花的时间越长。

问题描述：计算大数组各元素开平方求和时，和预期不一样，发现开的进程数量越多，花的总时间越长。可能是因为进程通信时候花费了时间太多。

解决方案：改写了sqrtSum.c文件，主要有以下改动：

```
if(myid==0){ // 发送数据N
  printf("请输入N\n");
  scanf("%d",&N);
  start = clock();
}
MPI_Bcast(&N,1,MPI_INT,0,MPI_COMM_WORLD);
//---------------------------------------------------------------------
if (myid!=0){
  start = clock();
}
for(int i = myid; i<N;i+=numprocs)
{
  local += sqrt(i*(i+1));
}
```

```
finish = clock();
duration = (double)(finish - start)/CLOCKS_PER_SEC;
printf("%d进程花费时间 %f seconds\n", myid,duration );
```

1）取消了动态数组（节省资源），而是在0进程输入N，用MPI_Bcast取代了MPI_Send和MPI_Recv来接收N的方式，更简洁。每个进程用变量local来记录总和。

2）在输入N后开始计时（没有变化），在把N广播到每个进程后，本进程计算结束后（获得local总值后，用MPI_Reduce把local传入0进程之前）计算时差，整个程序结束后计算时差。

得到结果如下：

```
(base) chengqichendeMacBook-Pro:MPI_Exp1 shengqichen$ mpirun -n 10 ./a.out
请输入N
10000
0进程花费时间 0.000105 seconds
4进程花费时间 0.000010 seconds
6进程花费时间 0.000010 seconds
7进程花费时间 0.000010 seconds
8进程花费时间 0.000016 seconds
9进程花费时间 0.000009 seconds
1进程花费时间 0.000014 seconds
5进程花费时间 0.000010 seconds
2进程花费时间 0.000009 seconds
3进程花费时间 0.000009 seconds
SqrtSum: 49999998.349880
total花费时间 0.007677 seconds
(base) chengqichendeMacBook-Pro:MPI_Exp1 shengqichen$ mpirun -n 3 ./a.out
请输入N
10000
0进程花费时间 0.000103 seconds
2进程花费时间 0.000024 seconds
1进程花费时间 0.000025 seconds
SqrtSum: 49999998.349880
total花费时间 0.000264 seconds
(base) chengqichendeMacBook-Pro:MPI_Exp1 shengqichen$
```

可以看出，10进程花费总时间仍然超过3进程总时间，但是可以发现，在MPI_Reduce把local传入0进程之前，计算时差，10进程的花费时间比3进程少，这是因为10进程分到的data量少。也侧面说明了10进程花费的总时间多，可能是因为通信花费的时间大大超过了分散数据量的加速效应。

思考与收获

对MPI有了更深的理解，例如发现scanf和print对于MPI不太好用，最好放到某个进程里面再分发，对于docker，我也入门了一些基础的指令，学会了连接host，构建集群。踩了一些坑，但是也收获了一些知识。