实验一：MPI Programming

施宇 191250119

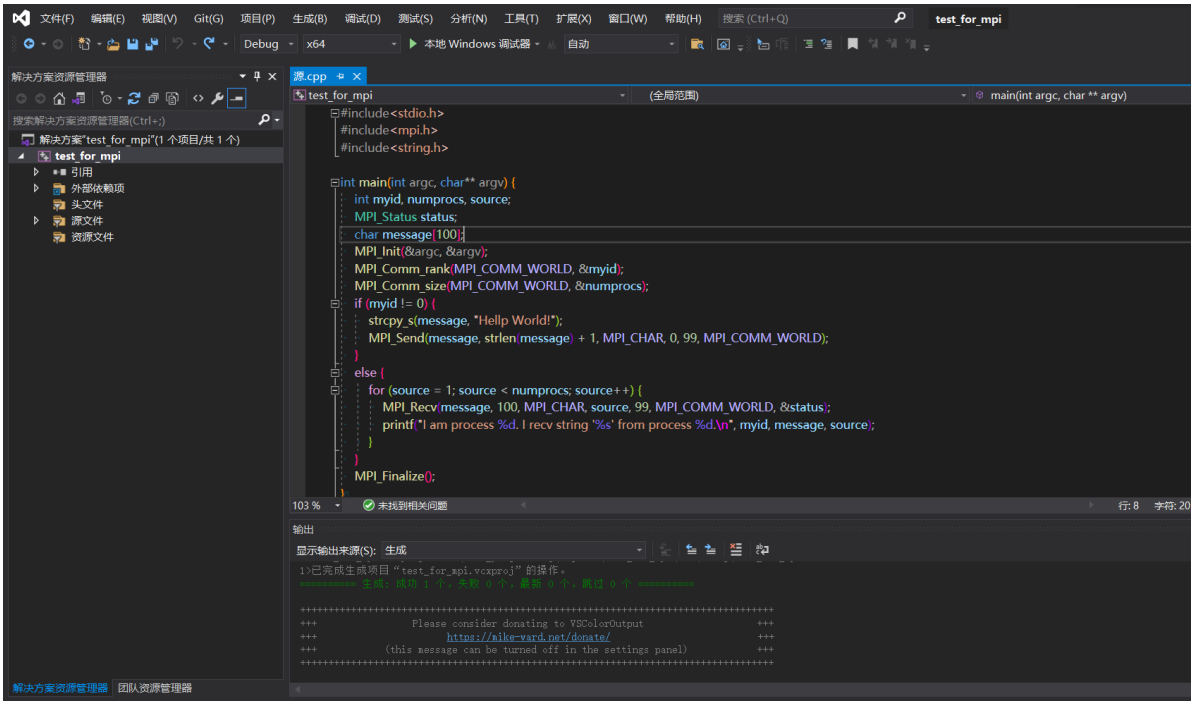
Github-->shiyu-coder/Financial-big-data-programming-course: 金融大数据课程相关内容 (github.com)

尝试在单机上安装并运行MPI环境。（MPICH或者OpenMPI等）

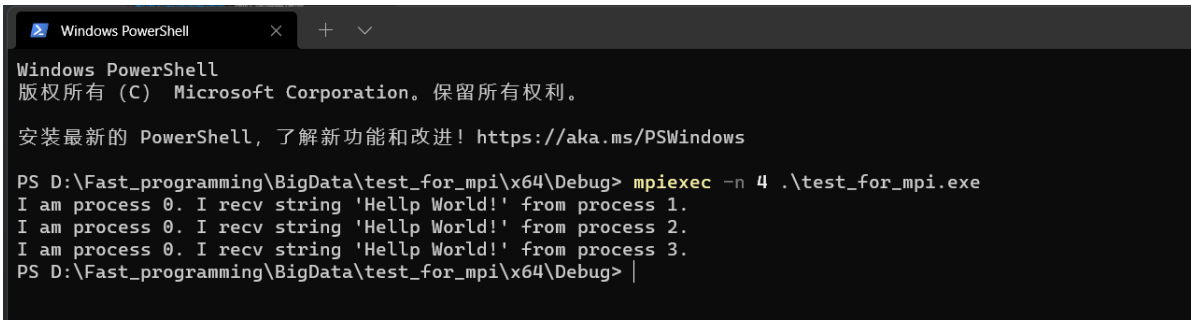
共安装了两套单机环境和一套多机环境：

单机环境：Visual Studio 2019 + MSMPI

编译环境：

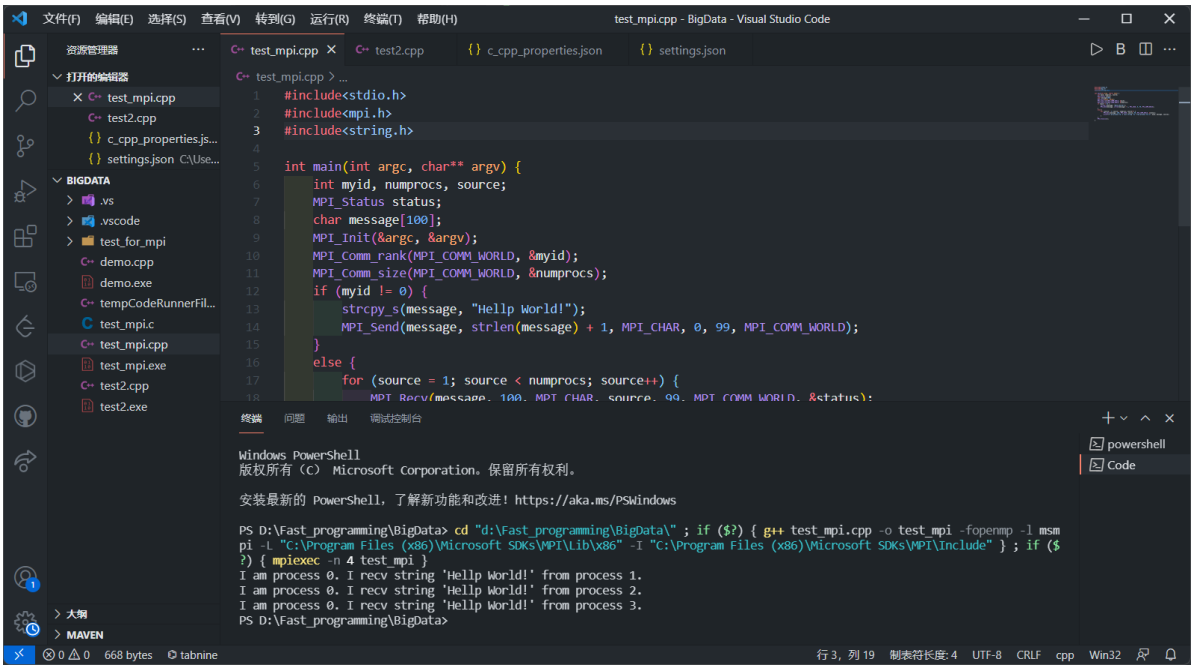


运行环境：



单机环境：Vscode + MSMPI

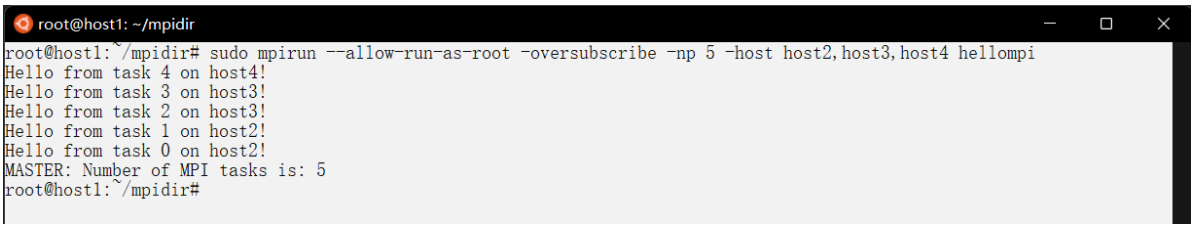
编译运行：



多机环境：Windows11家庭版 + WSL2 + Docker

详细环境配置过程见**附录A**

在三个节点上运行hellompi程序：



当并行数为5时，三个节点都参与工作，且任务数量按顺序平均分配。

编程1

用MPI_Reduce接口改写大数组各元素开平方求和(data[N], data[i]=i*(i+1))的代码（可通过命令行传入N的值，比如1000，10000，100000）

```
#include<iostream>
#include<mpi.h>
#include<cmath>
using namespace std;

int main(int argc, char** argv){
    int id, source, N, numprocs;
    double sqrtSum = 0.0, res = 0.0;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    --numprocs;
    if(id == 0){
        cout <<"Input N: ";
        cin >>N;
    }
    // 将N的值广播给其他进程
    MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
    MPI_Barrier(MPI_COMM_WORLD);
    if(id != 0){
        for(int i=id-1; i<N; i+=numprocs){
            sqrtSum += sqrt(i*(i+1));
        }
    }
    // 接收其他进行的运算结果并求和
    MPI_Reduce(&sqrtSum, &res, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if(id == 0){
        cout <<"I am process "<<id<<". The final value is "<<res<<endl;
    }else{
        cout <<"I am process "<<id<<". My SqrtSum is "<<sqrtSum<<endl;
    }

    MPI_Finalize();
}
```

设置线程数为4时的运行结果：

N=100：

```
PS D:\Fast_programming\BigData> cd "d:\Fast_programming\BigData\" ; if ($?) {
SDKs\MPI\Lib\x86" -I "C:\Program Files (x86)\Microsoft SDKs\MPI\Include" } ;
Input N: 100

I am process 1. My SqrtSum is 1699.34
I am process 3. My SqrtSum is 1666.32
I am process 0. The final value is 4998.93
I am process 2. My SqrtSum is 1633.27
PS D:\Fast_programming\BigData>
```

N=1000：

```
PS D:\Fast_programming\BigData> cd "d:\Fast_programming\BigData\" ; if ($?) {
SDKs\MPI\Lib\x86" -I "C:\Program Files (x86)\Microsoft SDKs\MPI\Include" } ;
Input N: 1000

I am process 1. My SqrtSum is 166999
I am process 3. My SqrtSum is 166666
I am process 2. My SqrtSum is 166333
I am process 0. The final value is 499999
PS D:\Fast_programming\BigData>
```

N=10000：

```
PS D:\Fast_programming\BigData> cd "d:\Fast_programming\BigData\" ; if ($?) { g++ exp1-1.cpp -o
SDKs\MPI\Lib\x86" -I "C:\Program Files (x86)\Microsoft SDKs\MPI\Include" } ; if ($?) { mpiexec -
Input N: 10000

I am process 1. My SqrtSum is 1.667e+007
I am process 2. My SqrtSum is 1.66633e+007
I am process 0. The final value is 5e+007
I am process 3. My SqrtSum is 1.66667e+007
PS D:\Fast_programming\BigData>
```

编程2

用MPI_Send和MPI_Receive接口计算积分：y=x3，求其在[10,100]区间的积分：

```
#include<iostream>
#include<mpi.h>
#include<stdlib.h>
using namespace std;

int main(int argc, char **argv){
    int id, numprocs;
    int a = 10, b = 100;
    int N = 100000000;
    double part_sum = 0.0, dx = (double)(b-a)/N, x;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    --numprocs;
    if(id == 0){
        for(int i = 1; i <= numprocs; i++){
            MPI_Recv(&x, 1, MPI_DOUBLE, i, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
            part_sum += x;
        }
    }else{
        for(int i=id-1; i<N; i+=numprocs) {
```

```
            x = a + i*dx + dx/2;
            part_sum += x*x*x*dx;
        }
        MPI_Send(&part_sum, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
    if(id == 0){
        cout <<"I am process "<<id<<". The final value is "<<part_sum<<endl;
    }else{
        cout <<"I am process "<<id<<". My partSum is "<<part_sum<<endl;
    }

    MPI_Finalize();
}
```

设置4个节点时的运行结果：

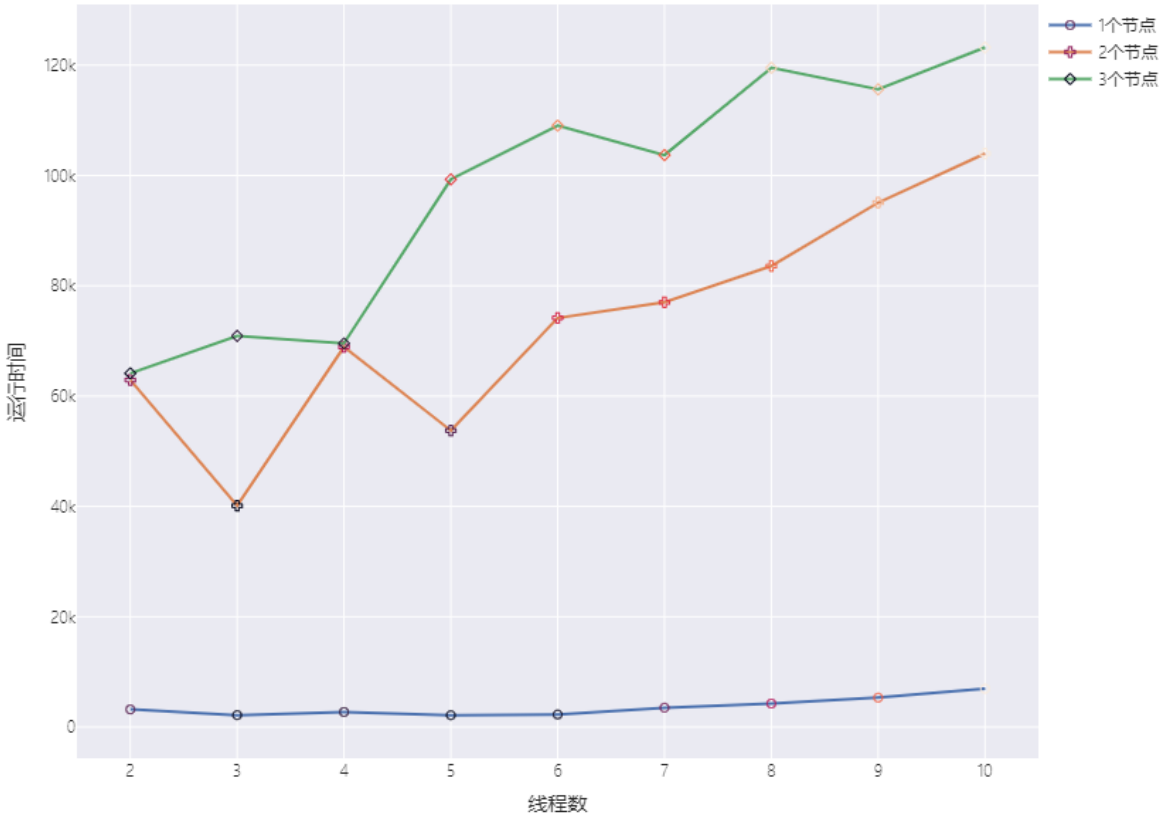
```
PS D:\Fast_programming\BigData> cd "d:\Fast_programming\BigData\" ; if ($?) { g++ ex
SDKs\MPI\Lib\x86" -I "C:\Program Files (x86)\Microsoft SDKs\MPI\Include" } ; if ($?)
I am process 1. My partSum is 8.3325e+006
I am process 2. My partSum is 8.3325e+006
I am process 3. My partSum is 8.3325e+006
I am process 0. The final value is 2.49975e+007
PS D:\Fast_programming\BigData>
```

尝试在多个节点上运行上述MPI程序，可设置不同的进程数对结果进行比较，并评估所需时间。

通过`time.h`库中的`clock`函数记录程序运行的时间。

程序1：

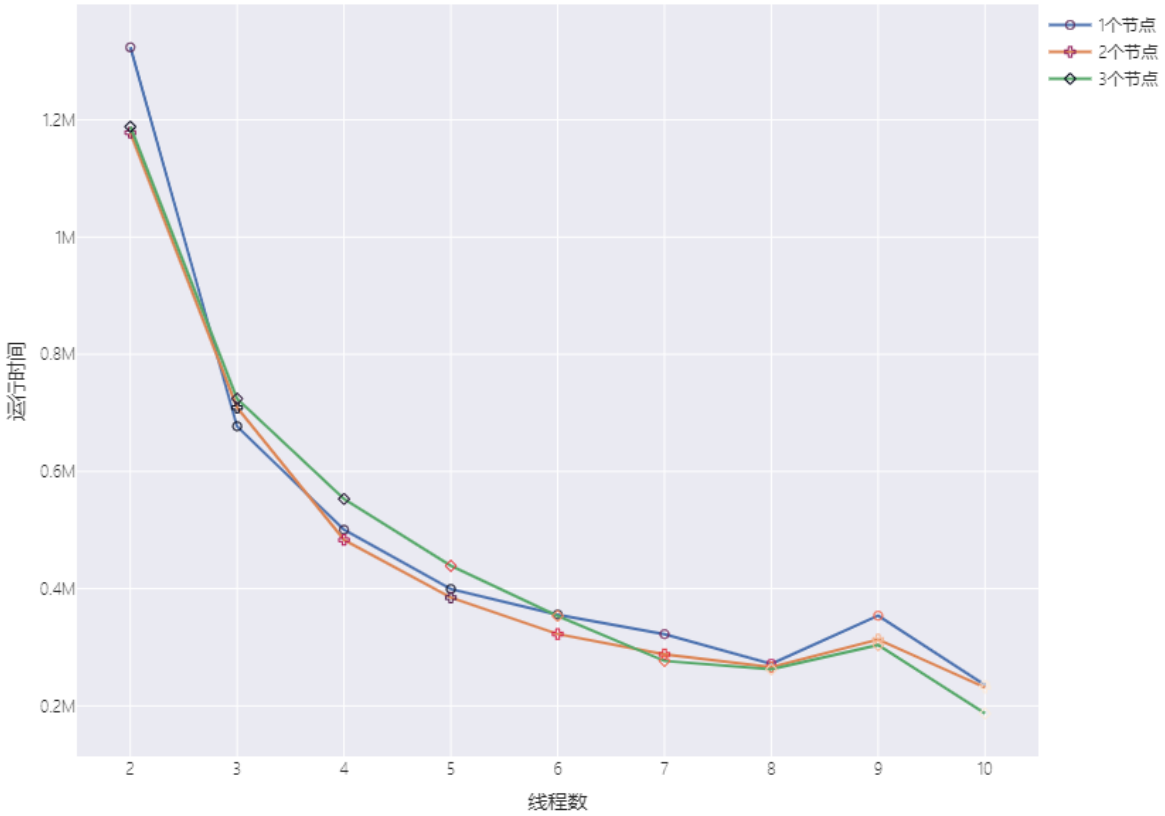
设置N=10000，节点个数分别为1，2和3，进程数从2到10，测试程序1从输入N后开始的运行时间，结果如下图：



从图中可以看出，当线程数一定时，节点从一个增加到两个时，运行时间大幅增加，节点从二个增加到三个也导致了运行时间的小幅增加；当节点数量一定时，运行时间随着线程数的增加而增加。原因可能是线程数和节点数的增加带来的对计算时间的节省小于节点间通信所需时间的增加。

程序2：

设置节点个数分别为1，2和3，进程数从2到10，测试程序1从输入N后开始的运行时间，结果如下图：



从图中可以看出，当节点数一定时，运行时间随着线程数的增加而减少，且减少速度逐渐放缓。考虑到该程序的计算量较大，增加线程数量能够加快程序运行速度，减少运行时间。当线程数量一定时，节点数量对运行时间的影响较小，随着线程数逐渐增加，节点数量多时程序运行时间的减少要快于节点数量少时。说明节点数量的增加能够在一定程度上加快程序运行速度，但是对程序计算速度的提升略高于增加节点数量带来的节点间通信所需时间的增加，因此增加节点数量对程序运行速度的增加程度并不明显。

综上，程序的运行速度与线程数和节点数密切相关。MPI是分布式处理系统下的一种并行化方法，属于进程级并行范畴，粗粒度并行。因此为了加快MPI程序的运行速度，需要根据程序的具体内容选择合适的线程数和进程数。考虑到操作系统内核调度执行的基本单位是线程，对于单线程进程，如果相同数量的进程和相同数量的线程都能同时得到内核调度，计算能力应该相同，在不考虑其他影响因素的情况下。但是由于CPU核的数量一般情况下会小于进程核线程数量，所以不能同时得到调度，这时需要考虑进程间通讯和线程间通讯及切换对程序运行速度的影响。

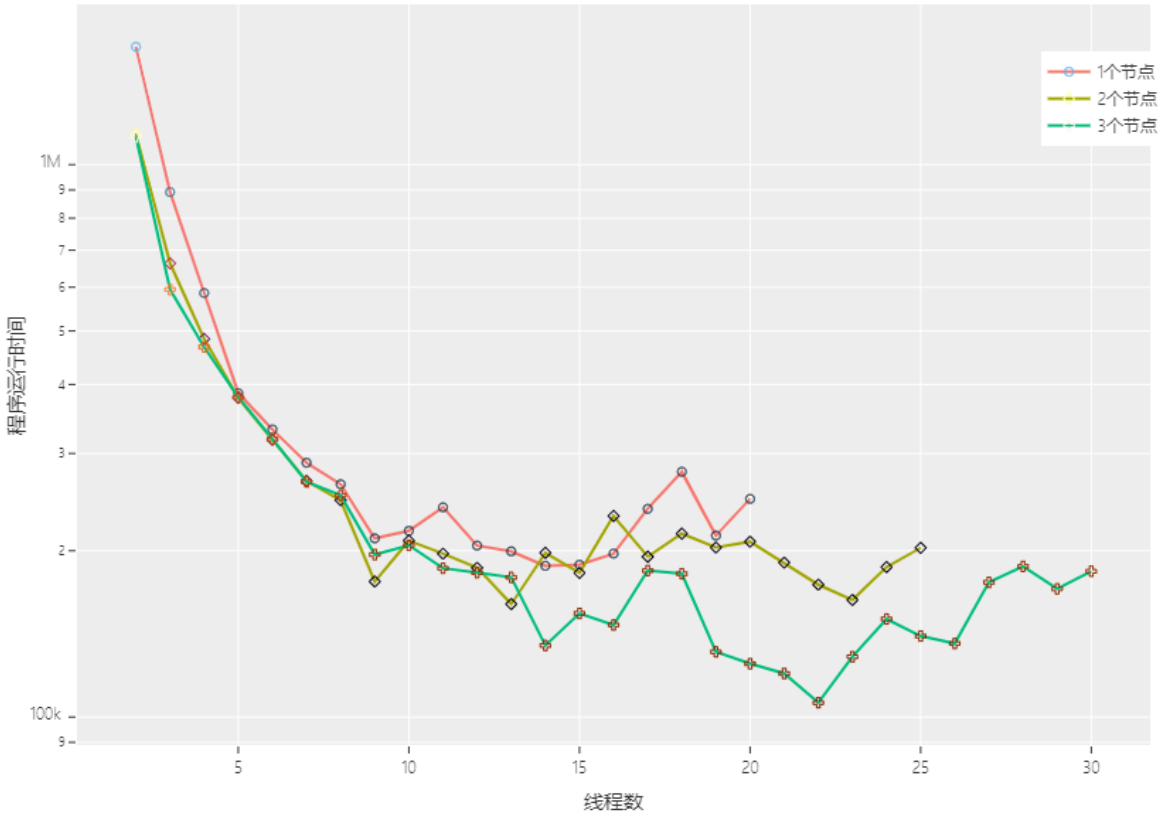
就本次实验中涉及到的两个程序而言，对于程序一，每个线程执行的运算的时间成本略小于线程间的切换和通讯成本，远小于进程间的通讯成本。因此当进程数固定时，随着线程数的增加，线程间切换的时间成本成为影响程序运行速度的主导因素，线程数越高，程序运行速度越慢；当线程数固定时，由于进程间通讯的时间成本远大于每个线程的运算成本，因此当节点数增加到2时，程序的运行时间具有较大程度的增加。因此，对于程序一，单节点少量线程能够使程序达到较高的运行速度。

对于程序二，每个线程执行的运算的时间成本大于线程间的切换和通讯成本，约等于进程间通讯成本。因此当进程数固定时，随着线程数的增加，每个线程的运行时间相应缩短，程序的运行速度加快；当线程数固定时，由于进行数增加带来的额外开销与每个线程的计算成本几乎相同，因此随着进程数的增加，程序的运行速度没有明显的变化。对于程序二，多线程数能够加快程序的运行速度，而进程数的增加对程序运行速度的提升没有明显效果。

对猜想的验证

按照以上想法，对于程序二，当增加N的大小后，每个线程的计算开销随之增加，当N增加到一定程度后，增加进程数量将会带来明显的程序运行时间的降低。下面验证该想法：

设定`long N=100000000000;`，重新测试不同进程数和线程数情况下程序二的运行时间，如图所示：



这里纵坐标采用对数坐标轴，以反映线程数较大时不同进程下程序运行时间的差异。从图中可以看出，在线程数一定时，随着节点数的增加，程序运行时间有了显著的降低。对于线程数较低时，进程数的增加带来的程序运行时间的下降尤为显著，并且当进程数为3时，程序的最短运行时间（在线程数为22时取到）也要小于进程数为1和2时的最短运行时间。因此证实了原先的猜想。

综上，可以看出，增加线程数的开销<增加进程数的开销，应当先考虑设定合适的线程数，再设定合适的进程数，以达到最佳的程序运行速度。当然，线程数和进程数的设定要结合CPU核心数量和集群的节点数量考虑选择合适的数值。

遇到的问题及解决方案

无法连接到Docker daemon at ……

```
shiyu@DESKTOP-520GOQ9:~$ docker run hello-world
docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?.
See 'docker run --help'.
```

在WSL1中运行docker会遇到以上错误，因为WSL1使用翻译层将Linux系统调用转化成Windows系统调用，并不支持完成的Linux内核和一些系统调用，因此无法运行docker。而WSL2则使用了一个轻量级的、无需维护的虚拟机，并在这个虚拟机中运行了一个完整的Linux内核。因此将WSL1改为WSL2即可支持docker。但是由于WSL2基于Hyper-V，一款Windows推出的虚拟机，而该组件只有在Windows专业版上才有，因此对于Windows家庭版，还需要手动安装Hyper-V后，才能使用WSL2.

mpicc编译报错

```
root@host1:~/mpidir# mpicc exp1-1.c -o exp1-1
/usr/bin/ld: /tmp/ccMoE8Yi.o: undefined reference to symbol 'sqrt@@GLIBC_2.2.5'
//lib/x86_64-linux-gnu/libm.so.6: error adding symbols: DSO missing from command line
collect2: error: ld returned 1 exit status
```

解决方案：数学函数sqrt（）位于libm.so库文件中（这些库文件通常位于/lib目录下），-lm选项使得编译器可以到相应的库文件中寻找用到的函数。-lm用于跟数学库函数的链接， 跟编译无关，undefined reference to `sqrt' 也表示编译通过而链接没有通过。

将编译命令改为：`mpicc -o exp1-1 exp1-1.c -lm`即可。

MSMPI编译报错

```
PS D:\Fast_programming\BigData> cd "d:\Fast_programming\BigData\" ; if ($?) { g++ exp1-2.cpp -o exp1-2 -fopenmp -l msmpi -L "C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x64" -I "C:\Program Files (x86)\Microsoft SDKs\MPI\Include" } ; if ($?) { mpiexec -n 4 exp1-2 }
C:\Users\86137\AppData\Local\Temp\ccEeiAvx.o:exp1-2.cpp:(.text+0x52): undefined reference to `MPI_Init@8'
C:\Users\86137\AppData\Local\Temp\ccEeiAvx.o:exp1-2.cpp:(.text+0x68): undefined reference to `MPI_Comm_rank@8'
C:\Users\86137\AppData\Local\Temp\ccEeiAvx.o:exp1-2.cpp:(.text+0x7e): undefined reference to `MPI_Comm_size@8'
C:\Users\86137\AppData\Local\Temp\ccEeiAvx.o:exp1-2.cpp:(.text+0xde): undefined reference to `MPI_Recv@28'
C:\Users\86137\AppData\Local\Temp\ccEeiAvx.o:exp1-2.cpp:(.text+0x171): undefined reference to `MPI_Send@24'
C:\Users\86137\AppData\Local\Temp\ccEeiAvx.o:exp1-2.cpp:(.text+0x230): undefined reference to `MPI_Finalize@0'
collect2.exe: error: ld returned 1 exit status
PS D:\Fast_programming\BigData>
```

使用x64的MSMPI编译程序时会出现以上错误，出现上述一系列找不到MPI库函数的编译错误的原因在于提供的MPI链接库为x86版本的，因此应该改用相应的x86版本的MPI编译器进行编译：

```
cd "d:\Fast_programming\BigData\" ; if ($?) { g++ exp1-2.cpp -o exp1-2 -fopenmp
-l msmpi -L "C:\Program Files (x86)\Microsoft SDKs\MPI\Lib\x86" -I "C:\Program
Files (x86)\Microsoft SDKs\MPI\Include" } ; if ($?) { mpiexec -n 4 exp1-2 }
```

安装libopenmpi-dev失败

在使用Ubuntu14.04的镜像时使用命令`sudo apt-get install libopenmpi-dev`安装libopenmpi-dev时会报错，经查阅原因后发现是由于14.04版本的Ubuntu系统的apt-get组件版本较老，导致无法更新安装一些较新的包。于是尝试更新apt-get组件，需要通过`wget`命令手动下载最新的apt-get组件并安装，但是由于从docker hub上pull下来的，该系统为了减小系统自身的大小，删除了很多非必须组件，包括wget。于是尝试通过`apt-get install wget`下载wget组件，但是由于apt-get组件版本较老，无法下载。因此陷入了死循环：apt-get版本较老->通过wget更新apt-get->docker hub上pull下来的Ubuntu14.04没有wget组件->通过apt-get安装wget->由于apt-get版本较老而安装失败。

解决方案：使用docker hub上的Ubuntu18.04系统作为镜像，由于18.04系统的apt-get组件版本较高，因此能够顺利的安装libopenmpi-dev组件。

关于向子节点发送MPI程序的代码文件的简便方法

通常情况下，是通过ssh等方式将要运行的MPI程序的代码文件从主节点发送给子节点，然而通过docker提供的挂载共享文件夹，能够快速的将代码文件共享到所有节点中：

```
docker run -dit --hostname host1 --name mpi-host1 -v ~/mpidir:/root/mpidir mpi-
ubuntu:latest
```

该命令在启动节点host1时，将/root/mpidir文件夹挂在到节点的~/mpidir文件夹位置，这样所有通过这种方式启动的子节点都在相同位置挂载了同样的文件夹，这时将需要运行的代码文件放入该文件夹中，即可直接在多节点环境下运行相应的MPI程序。

实验感想

由于我的实验环境非常特殊：Windows11家庭版+WSL2，因此在环境配置中遇到了许多“冷门”错误，网上提供的解决方案寥寥无几，很多情况下需要自己去阅读报错信息并推断可能的错误原因，然后向可能的解决方向进行尝试。在解决问题的过程中，我也逐渐对相应的环境和程序有所熟悉，遇到棘手的错误时也不会感到慌乱，而是尽可能的静下心来，寻找可能的解决方案。虽然整个环境的配置过程花费了不少的时间，但是回过头来看，也是收获满满。另外，对两个MPI程序的改写也让我更加熟悉了MPI程序的编程规范，对MPI的运行机制，进程和线程等知识有了基本的了解。最后，在尝试网上CSDN和各种博客提供的解决方案时，做好相关的备份工作是非常重要的，一旦发现尝试失败，能够迅速回到原先的状态，继续其他解决方案的尝试。

附录A

多机环境(Windows11家庭版 + WSL2 + Docker)配置：

安装Hyper-V

由于WSL2需要Hyper-V组件提供虚拟机支持，而该组件只有Windows专业版上才有，因此在Windows家庭版上需要手动下载安装该组件。

将下面的内容添加到记事本中：

```
pushd "%~dp0"

dir /b %SystemRoot%\servicing\Packages\*Hyper-V*.mum >hyper-v.txt

for /f %%i in ('findstr /i . hyper-v.txt 2^>nul') do dism /online /norestart
/add-package:"%SystemRoot%\servicing\Packages\%%i"

del hyper-v.txt

Dism /online /enable-feature /featurename:Microsoft-Hyper-V-All /LimitAccess
/ALL
```

然后另存为Hyper-V.cmd文件，右键该文件以管理员身份执行，待下载结束后输入Y重启，完成Hyper-V的安装。

安装WSL2

下载Linux内核更新包：

https://links.jianshu.com/go?to=https%3A%2F%2Fwslstorestorage.blob.core.windows.net%2Fwslblob%2Fwsl_update_x64.msi

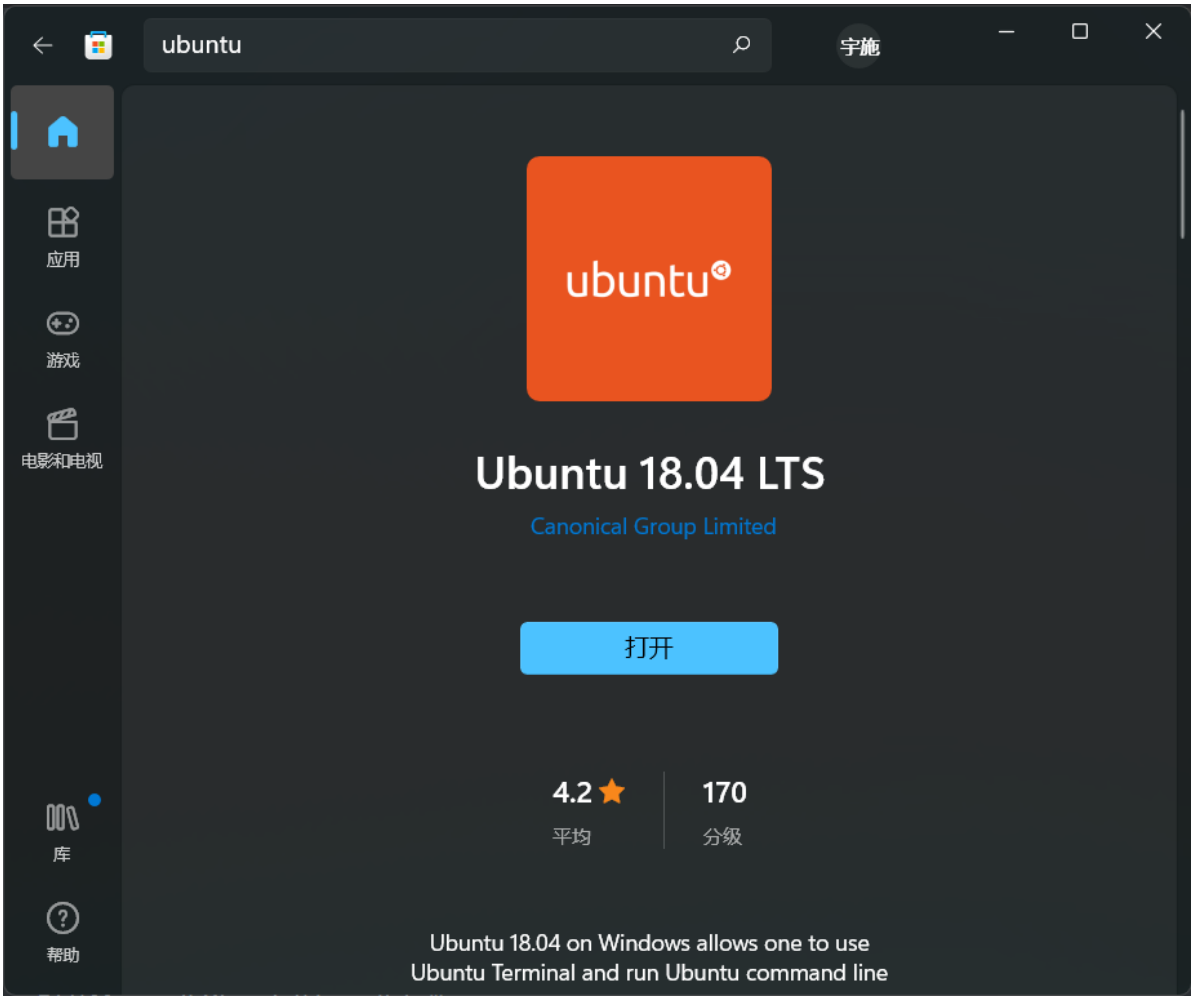
运行下载的更新包。 （双击以运行 - 系统将提示提供提升的权限，选择“yes”以批准此安装。）

打开终端输入下方命令，将WSL默认版本设置为WSL2

```
wsl --set-default-version 2
```

打开Microsoft Store下载Linux发行版：

这里我选择的是Ubuntu18.04：



安装完成后在菜单栏找到点击运行，输入用户名密码后即可进入Ubuntu终端。

在Windows终端中输入`wsl -l -v`检查是否是WSL2版本。

安装Docker

配置阿里云镜像，`/etc/apt/sources.list`文件的内容替换如下：

```
deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe
multiverse
```

```
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted
universe multiverse
```

更新源：

```
sudo apt-get update
```

安装Docker：

```
sudo apt-get remove docker docker-engine docker.io
sudo apt-get update
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
sudo apt-get update
sudo apt-get install docker-ce
```

测试docker是否安装成功：

```
docker version
```

添加用户：

```
sudo adduser $USER docker
```

启动docker服务：

```
sudo service docker restart
```

测试docker能否正常运行：

```
sudo docker run hello-world
```

如果提示：Cannot connect to the Docker daemon at ……则还需要以下操作：

关闭wsl窗口，重新右键以管理员身份运行：

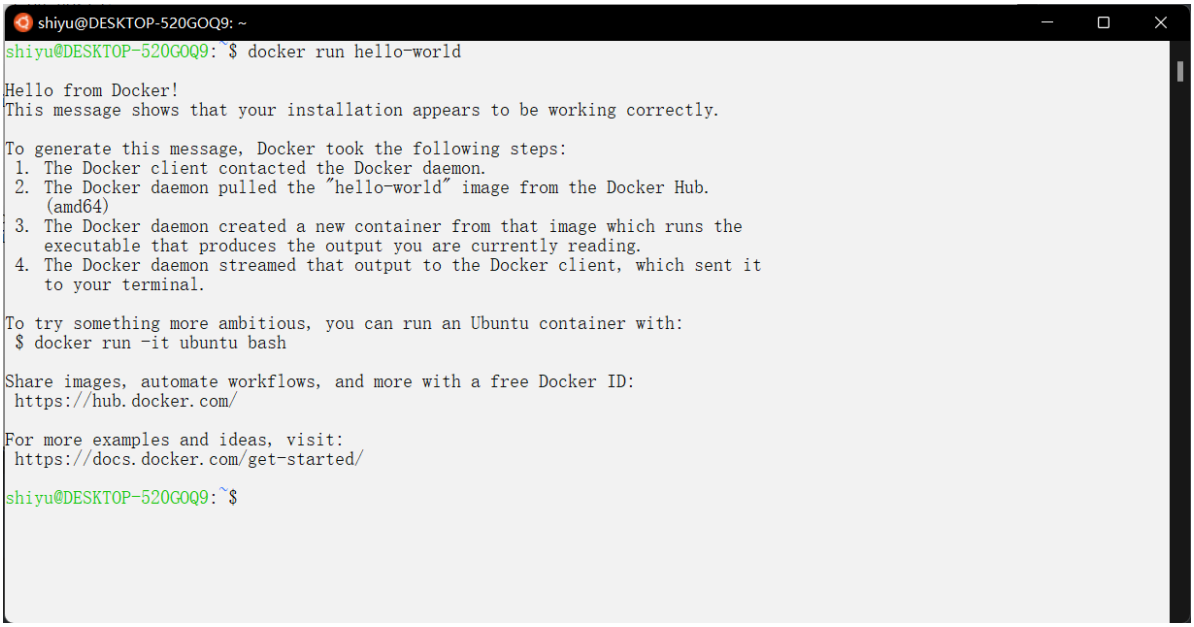
然后执行以下命令：

```
sudo apt-get install cgroupfs-mount
sudo cgroupfs-mount
sudo service docker restart
```

再次运行测试命令：

```
sudo docker run hello-world
```

运行成功：



多机MPI配置

这里使用ubuntu:18.04作为节点镜像：

```
docker pull ubuntu:18.04
```

拉取镜像之后，需要对基础镜像进行环境的配置，在此之前先进行换源

开启一个容器:

```
docker run -dit --hostname origin --name origin ubuntu:18.04
```

进入容器：

```
docker exec -it origin bash
```

安装一下sudo和vim：

```
apt-get install sudo
sudo apt-get install vim
```

切换阿里云的源：

```
sudo cp /etc/apt/sources.list /etc/apt/sources.list.bak
sudo vim /etc/apt/sources.list
```

替换为以下内容：

```
deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted
universe multiverse
```

更新源：

```
sudo apt-get update
```

安装MPI：

```
sudo apt-get install libopenmpi-dev
sudo apt-get install openmpi-bin
```

检查是否安装成功：

```
mpirun --version
```

安装ssh：

```
sudo apt-get install openssh-server
```

尝试ssh一下自己，应该会要求输密码，三次回车即可（拒绝访问）：

```
ssh localhost
```

到~/目录下，检查是否有.ssh文件夹：

```
ll
```

进入该文件夹，然后应该能找到authorized_keys文件。

修改authorized_keys文件权限为700：

```
sudo chmod 700 authorized_keys
```

修改.ssh目录权限为600：

```
cd ..
sudo chmod 600 .ssh
```

Ctrl+Q+P退出当前镜像。

创建新镜像：

```
docker commit -a <作者名字> origin mpi-ubuntu:latest
```

罗列出本地镜像，看是否创建成功:

```
docker images
```

通过挂在本地文件夹的方式运行容器（这里以四个为例）：

```
docker run -dit --hostname host1 --name mpi-host1 -v ~/mpidir:/root/mpidir mpi-
ubuntu:latest
docker run -dit --hostname host2 --name mpi-host2 -v ~/mpidir:/root/mpidir mpi-
ubuntu:latest
docker run -dit --hostname host3 --name mpi-host3 -v ~/mpidir:/root/mpidir mpi-
ubuntu:latest
docker run -dit --hostname host4 --name mpi-host4 -v ~/mpidir:/root/mpidir mpi-
ubuntu:latest
```

进入host1：

```
docker exec -it mpi-host1 bash
```

启动ssh服务：

```
service ssh start
```

打开hosts文件：

```
sudo vim /etc/hosts
```

可以看到文本中最后一行为：

```
172.17.0.2      host1
```

前面是ip地址（也是本容器的地址），后面是主机名（可以看作是ip的别名，可以用它来代替ip地址的书写）

在后面添加host2，host3和host4的ip地址和主机名：

```
172.17.0.3      host2
172.17.0.4      host3
172.17.0.5      host4
```

到~/.ssh目录中，在当前目录下生成密钥（输完命令后直接三次回车，不要做其他操作）：

```
ssh-keygen -t rsa
```

然后会发现生成了id_rsa.pub文件，将该文件复制到挂载目录~/mpidir中：

```
cp id_rsa.pub ~/mpidir
```

退出当前脚本镜像，分别进入host2， host3， host4，进行如下操作：

启动ssh服务：

```
service ssh start
```

进入~/mpidir目录，执行下方命令，将host1的公钥加入到其他子节点的authorized_keys中，设置免密ssh登录：

```
cat id_rsa.pub >> ~/.ssh/authorized_keys
```

退出当前镜像。

全部执行完成后，进入host1：

依次检查能够免密ssh连接host2，host3和host4（连接后通过Ctrl+D退出）：

```
ssh root@host2
ssh root@host3
ssh root@host4
```

如果都能成功连接，则测试hellompi程序运行：

```
mpicc hellompi.c -o hellompi
mpirun -np 10 -host host2,host3,host4 hellompi
```

如果报错：

```
--------------------------------------------------------------------------
mpirun has detected an attempt to run as root.
Running at root is *strongly* discouraged as any mistake (e.g., in
defining TMPDIR) or bug can result in catastrophic damage to the OS
file system, leaving your system in an unusable state.

You can override this protection by adding the --allow-run-as-root
option to your cmd line. However, we reiterate our strong advice
against doing so - please do so at your own risk.
--------------------------------------------------------------------------
```

加上参数`--allow-run-as-root`。

如果报错：

```
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 5 slots
that were requested by the application:
  hellompi

Either request fewer slots for your application, or make more slots available
for use.
--------------------------------------------------------------------------
```

则说明mpi自动检测判断当前cpu不适合以给定的并行数执行，这里选择添加参数`-oversubscribe`来强制执行。最终的执行命令如下：

```
sudo mpirun --allow-run-as-root -oversubscribe -np 5 -host host2,host3,host4
hellompi
```

执行成功：

```
root@host1:~/mpidir# sudo mpirun --allow-run-as-root -oversubscribe -np 10 -host host2,host3,host4 hellompi
Hello from task 9 on host4!
Hello from task 5 on host3!
Hello from task 4 on host3!
Hello from task 0 on host2!
MASTER: Number of MPI tasks is: 10
Hello from task 1 on host2!
Hello from task 2 on host2!
Hello from task 8 on host4!
Hello from task 6 on host3!
Hello from task 7 on host4!
Hello from task 3 on host2!
root@host1:~/mpidir#
```

当关闭WSL再次进入时，需要切换到root用户，然后重启docker服务才能正常使用：

```
sudo -s
sudo service docker restart
```

启动已经存在的容器：

```
docker ps -a
docker start <Container ID>
```