191840265 吴偲羿 实验一实验报告

第一部分 代码展示：

Q1:

```
#include<iostream>
#include<mpi.h>
#include<math.h>
#include<time.h>
using namespace std;
clock_t start,finish;//start与finish 记录进程0开始与结束的时间
int main(){
  int numprocs,myid;
  double allSum=0.0,tempSum=0.0;
  int k;
  MPI_Init(NULL,NULL);
  MPI_Status status;
  MPI_Comm_rank(MPI_COMM_WORLD, &myid);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  numprocs--;
  if(myid==0){
    start=clock();
    cin>>k;
    for(int i=1;i<=numprocs;i++)
    MPI_Send(&k,1,MPI_INT,i,2,MPI_COMM_WORLD);//0进程接收k的输入，并广播至各个进程
    int *p=new int[k+10];//由于本题数据范围较大，开动态数组进行内存管理
    for(int i=1;i<=k;i++){
      p[i]=i*(i+1);
    }
    for(int i=1;i<=k;i++){
      MPI_Send(&p[i],1,MPI_INT,i%numprocs+1,1,MPI_COMM_WORLD);//数据的分发
    }
  }
  if(myid!=0){
    tempSum=0.0;
    MPI_Recv(&k,1,MPI_INT,0,2,MPI_COMM_WORLD,&status);//各进程接受数据
    for(int i=1;i<=k;i++){
      if(i%numprocs+1==myid){
        int temp;
        MPI_Recv(&temp,1,MPI_INT,0,1,MPI_COMM_WORLD,&status);
        tempSum+=sqrt(double(temp));
      }
    }
  }
  MPI_Reduce(&tempSum,&allSum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  if(myid==0){
```

```
    cout<<allSum<<endl;
    finish=clock();//记录0进程结束的时间
    printf("RunningTime:%lfs\n",double(finish-start)/CLOCKS_PER_SEC);//运行时间
  }
  MPI_Finalize();
  return 0;
}
```

Q2

```
#include<iostream>
#include<time.h>
#include<mpi.h>
#define N 100000000
using namespace std;
clock_t start,finish;
int main(){
        int myid,numprocs;
        double local=0.0, dx=(90.0)/N;
        double inte,x,sum=0.0;
        double maxRunningTime;
        MPI_Status status;
        int temp;
        MPI_Init(NULL,NULL);
        start=clock();//Start和finish记录每一个进程的运行时间
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);
        MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
        numprocs--;
        if(myid!=0){
                for(int i=myid;i<N;i+=numprocs){//分配数据
                        x=10+i*dx+dx/2;
                        local+=x*x*x*dx;
                }
                MPI_Send(&local,1,MPI_DOUBLE,0,1,MPI_COMM_WORLD);
        }
        if(myid==0){
                for(int i=1;i<=numprocs;i++){
 MPI_Recv(&inte,1,MPI_DOUBLE,i,1,MPI_COMM_WORLD,&status);
                        sum+=inte;
                }
                printf("The integration of x^3 between [10,100] is: %lf\n",sum);
        }
        finish=clock();
        double RunningTime=double(finish-start)/CLOCKS_PER_SEC;

MPI_Reduce(&RunningTIme,&maxRunningTime,1,MPI_DOUBLE,MPI_Max,0,MPI_COMM_WORLD);//将所有进程的运行时间最大值，按mpireduce方法规约到0进程，作为运行时间。
```

```
        MPI_Finalize();
        if(myid==0) printf("RunningTime: %lfs\n",maxRunningTime);
        return 0;
}
~
```

2.运行截图

Q1:节点数固定为4 不同数据量

```
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
 10
 59.7014
 RunningTime:0.000277s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
 100
 5099.42
 RunningTime:0.000359s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
 99
 4998.93
 RunningTime:0.000404s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
 1000
 500999
 RunningTime:0.000506s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
 10000
 5.001e+07
 RunningTime:0.004551s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
```

固定数据量为10000 不同节点

```
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 2 ./Q1
10000
5.001e+07
RunningTime:0.002782s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 3 ./Q1
10000
5.001e+07
RunningTime:0.001263s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
10000
5.001e+07
RunningTime:0.003024s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 5 ./Q1
10000
5.001e+07
RunningTime:0.004667s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 6 ./Q1
10000
5.001e+07
RunningTime:0.004941s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 7 ./Q1
10000
5.001e+07
RunningTime:0.006908s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 8 ./Q1
10000
5.001e+07
RunningTime:0.008923s
```

Q2:不同进程数量

```
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 2 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999095
 RunningTime: 0.357427s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 3 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.184515s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.128169s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 5 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.106866s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 6 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
```

```
RunningTime: 0.098143s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 7 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999099
 RunningTime: 0.083209s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 8 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.080608s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 9 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.075742s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 10 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.068102s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 11 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.071253s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 12 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.057876s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 13 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999101
 RunningTime: 0.052621s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 14 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999101
 RunningTime: 0.055478s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 15 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999101
 RunningTime: 0.054817s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 16 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999101
 RunningTime: 0.055797s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 17 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999101
 RunningTime: 0.032411s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 18 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.052378s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 19 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.045499s
[quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 20 ./Q2
 The integration of x^3 between [10,100] is: 24997499.999100
```

Docker内两个容器两个节点运行：

```
[mpirun -np 4 -host 172.17.0.2,172.17.0.3 /home/Q1
[1000
 500999
 RunningTime:0.003712s
 # sh: turning off NDELAY mode
[mpirun -np 4 /home/Q1
[1000
 500999
 RunningTime:0.000395s
 # sh: turning off NDELAY mode
[mpirun -np 4 -host 172.17.0.2,172.17.0.3 /home/Q2
 The integration of x^3 between [10,100] is: 24997499.999100
 RunningTime: 0.131948s
```

3.问题总结及解决方案

1.ssh连接问题

尽管按照老师pdf上的教程，在两个容器的root/.ssh文件夹下配置好了 authorized_keys id_rsa以及id_rsa.pub文件，

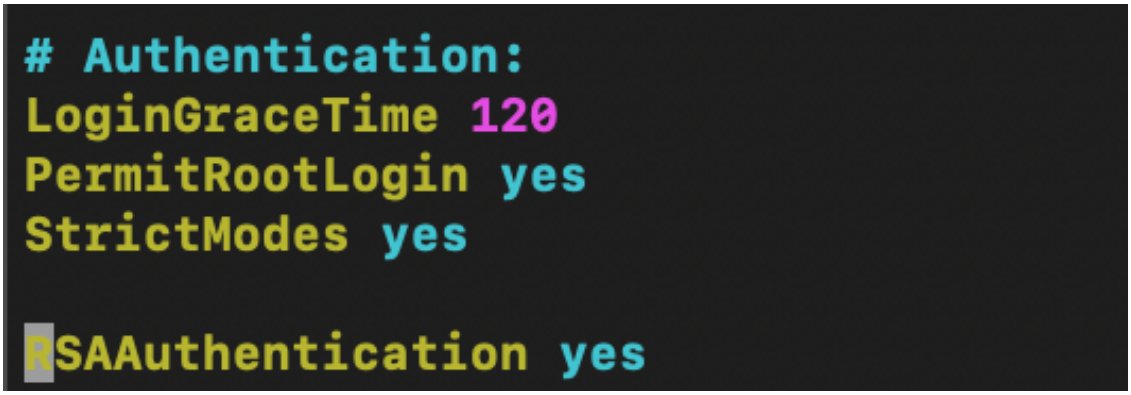
```
 # cd ~/.ssh
[# ls
 authorized_keys  id_rsa  id_rsa.pub
```

但是在请求ssh root@另一个容器时候，仍然出现了需要密码的情况

```
[root@eevee's password:
 Permission denied, please try again.
[root@eevee's password:
 Permission denied, please try again.
[root@eevee's password:
 Permission denied (publickey,password).
[# ssh root@eevee
[root@eevee's password:
 Permission denied, please try again.
```

解决方案

采纳了CSDN有关主节点容器链接其他容器免密的相关资料 修改了etc/ssh/sshd_config的相关配置如下：



2.第二题如何输入n的问题

本以为只是修改一个输入的问题，后来发现没有那么简单,轻敌子

在mpi运行过程中，由于每一个进程都需要执行一遍编译好后的可执行文件，导致程序中简单的cin>>n, 从stdin接受输入需要在每个进程都执行一遍。事实上，这也引起了我对于mpi本质的一些思考。

本以为MPI的并行执行过程只是执行MPI_Init与MPI_Finalize之间的代码，但事实发现，mpi的每个进程都会将整个代码执行一遍：实验如下

```
...
  cout<<myid<<":"<<2333<<endl;
  MPI_Init(NULL,NULL);
  ...
  MPI_Finalize();
  return 0;
}
```

执行结果：

0:2333

1:2333

4:2333

2:2333

3:2333

这些2333仿佛是对我的嘲讽，嗯。

因此询问了老师之后，得到了如下的回答：

“编译后程序分发到各个节点上，每个节点运行的是同样的程序，但是只有MPI_Init和MPI_Finalize之间的代码是可以调用MPI接口执行的。否则会报“Attempting to use an MPI routine before initializing MPICH”错误。进程行为的区分用进程标号来区别，比如约定0号进程接受输入给变量赋值，然后把该变量值广播给其它进程。可以参考mpich examples目录下icpi程序的实现。”

茅塞顿开！

那么 回到原问题上，如何解决这个输入的问题呢?

解决方案

事实上，这也是一个在某个进程接受输入，然后分发到其他进程的过程。我查阅相关资料得到了一个叫做MPI_Bcast（）的函数。这个函数允许从某一个进程接受一个输入，并广播到其他所有进程中，是一个一对多的过程。

然而这又存在一个问题了，Bcast并不是一个同步通信的过程，在所有进程同时运行的过程中，其他进程并没有接收到一个“需要等待Bcast广播输入”这样一个信号，而进程的运行先后又没有顺序，往往0进程广播输入k的值后，其他进程已经使用过一个尚未赋值的k值了。

如果非要用BCast，解决方法是有的，也就是需要让其他进程明白，在使用k之前需要接受来自0进程的广播。那么怎么办呢？那就是用MPI_Send和MPI_Recv，Bcast广播完k值后，从0进程发送一个信号，让其他进程用MPI_Recv接受，在这之后再使用k值。这是一个同步通信的过程，可以保证k的使用在接收到BCast广播之后。

我意识到这是很蠢的。。为什么不直接用Send和Recv接受k值呢。。

所以解决代码如下：

```
if(myid==0){
  cin>>k;
  for(int i=1;i<=numprocs;i++){
    MPI_Send(&k,1,MPI_INT,i,1,MPI_COMM_WORLD);
  }
}else{
  MPI_Recv(&k,1,MPI_INT,0,1,MPI_COMM_WORLD,status);
}
```

3.有关运行时间的统计

同样的，如上述所说，如果不对执行统计时间的相关代码的进程加以控制，所有的进程都会统计一遍其运行时间，输出的将是每个进程运行的时间。

由于所有进程是同步进行的，程序实际运行时间其实是耗时最长的进程运行的时间。对于第一题来说，由于在其他进程执行实际运算过程之前，需要等待0进程分配数据，在完成计算之后也需要在0进程汇总并输出，因此可以用0进程的执行时间作为整个程序的执行时间。

代码如下：

```
...
  if(myid==0) start=clock();
  ...
  if(myid==0){
    finish=clock();
    cout<<"RunningTime:"<<double(finish-clock)/CLOCKS_PER_SEC;//转化成秒输出
  }
...
```

然而，对于第二题，由于其他进程是单独判断计算范围，执行计算，最后通过mpireduce规约到0进程后输出，无法用0进程的运行时间作为总运行时间。

考虑到 “程序实际运行时间其实是耗时最长的进程运行的时间”，我们可以利用MPI_Reduce中op=MPI_MAX，计算所有进程运行最长的时间，规约到0进程，作为程序运行时间进行输出

解决方案

```
MPI_Reduce(&RunningTIme,&maxRunningTime,1,MPI_DOUBLE,MPI_Max,0,MPI_COMM_WORLD);//将所有进程的运行时间最大值，按mpireduce方法规约到0进程，作为运行时间。
```

这样，统计出运行时间最长的进程就是总运行时间了。

4.其他思考

有关进程数目与程序运行速度的思考

这也是一个老生常谈的问题了，对于一个需要进行固定规模运算的程序，是否mpi中同步运行的进程越多，程序运行速度就越快？对此问题我做了一点非常简单的思考。

假设在理想情况中，一个问题规模为n个单位，分配给k个进程，假设每个进程运算的速度于数据规模N的函数关系为f(N)，给一个进程分配、收集数据的时间与进程数k的关系为g(k)，在这个理想模型中，一个程序运行时间T则为问题规模n与进程数k的函数：

$$T(n,k)=f(\frac{n}{k})+g(k)$$

我们发现，当进程数k不变时，T与n的函数关系

$$T(n)=f(\frac{n}{k})+C$$

实际上就是这个程序的复杂度。例如第一题，明显是一个线性的算法，那么我们固定进程数为4，可以得到:

```
1000
500999
RunningTime:0.000376s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
2000
2.002e+06
RunningTime:0.000433s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
3000
4.503e+06
RunningTime:0.000598s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
4000
8.004e+06
RunningTime:0.000500s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
10000
5.001e+07
RunningTime:0.001823s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
20000
2.0002e+08
RunningTime:0.004942s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
30000
4.5003e+08
RunningTime:0.005451s
quinton_541@MacBook-Air-de-541 MPI_Test % mpirun -np 4 ./Q1
40000
8.0004e+08
RunningTime:0.007258s
```

我们基本可以发现，随着数据规模的增大，运行时间是逐渐增加的，但并不一定按照我们的假设，线性增加。事实上，回到我们的假设，我们认为，分配进程所占用的时间仅仅随着进程的变化而变化，查阅相关资料得知，其需要的时间与数据量，代码逻辑与电脑性能都有一定的关系。因此，我们在这里仅仅只能得到**运行时间与数据量正相关**的结论，尚且有待定量研究。

那么如果数据量一定呢

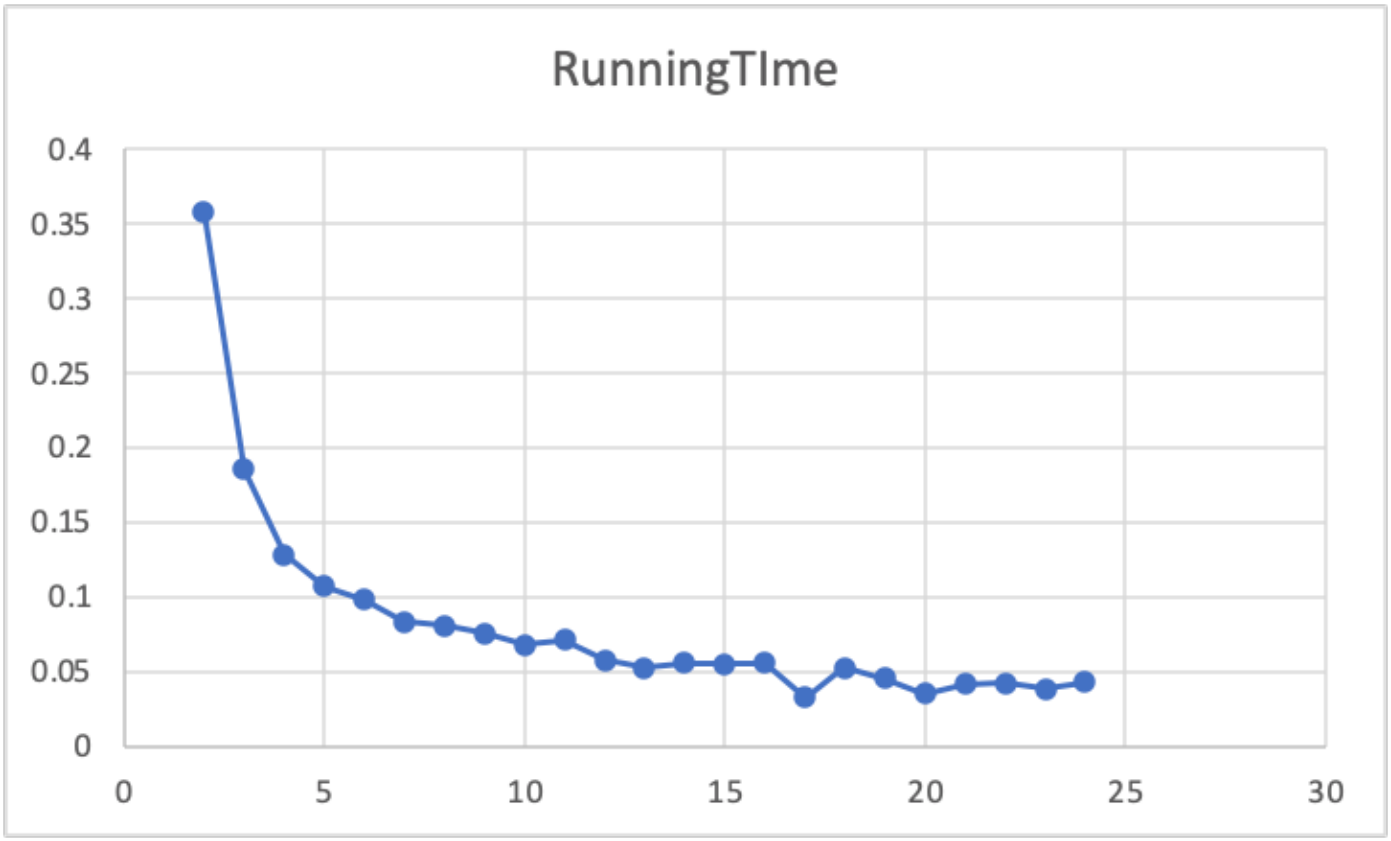
第二题就满足这个模型

$$T(k) = f(\frac{n}{k}) + g(k)$$

可以发现，我们无法直观地从这个式子中得到一个定性的结论，随着进程的增多，每个进程分配到的数据量减小，运行时间也减少，但是分配进程所占用的时间也越多。因此 我们对第二题不同的进程量进行了研究 得到下表

Numprocs	RunningTIme（s）
2	0.357918
3	0.185275
4	0.128169
5	0.106866
6	0.098143
7	0.083209
8	0.080608
9	0.075742
10	0.068102
11	0.071253
12	0.057876
13	0.052621
14	0.055478
15	0.054817
16	0.055797
17	0.032411
18	0.052378
19	0.045499
20	0.035201
21	0.041327
22	0.041961
23	0.03862
24	0.035913
100	0.030794
110	0.042258
120	0.027978

对2-20进程数作图：



我们发现，在进程数较少的时候随着进程数的增加，程序运行时间有明显的降低，但随着进程数的增长，程序运行时间渐渐保持平稳态势，并且增加到100、110、120进程数时，运行仍保持同样水平，并无明显减少或增长。

这是一个有趣的发现，这说明，当进程数较少时，程序的运行速度f(n/k)占据主导，T(k)保持单调递减的态势，而随着进程数的增多，程序的运行速度与分配进程耗费的时间趋于平衡，程序的运行速度也保持相同的水平。如上所言，程序运行需要的时间与数据量，代码逻辑与电脑性能都有一定的关系，仍需要深入的定量研究。

另外，由于本题数据规模仍然非常小，导致分配进程的时间依然是一个重要的因素。因此，在本次实验中，多容器多节点的运算速度反而不及单机多进程的运算速度，这是可以理解的，不同容器之间通信的时间要大于单机进程间通信的时间，在数据规模比较小的时候，就体现在程序的运行时间上了。