**实验报告**

课程名称：

学生姓名：

学号：

指导老师：

实验4：MPI进阶实验

1. **实验目的**

掌握MPI集合通信基础方法使用以及MPI派生数据基础方法使用，可独立实现矩阵乘法的MPI并行计算编程并可完成派生数据类型的聚集和广播。

1. **实验要求**
2. 集合通信

在上次实验中，所有操作都是通过编程者指定发送和接收者来实现的，可以称之为点对点的通信，这种做法十分复杂，而且也不能保证编程者能以此写出最佳性能的程序。因此 MPI 提供了一种包含进程组中所有进程的全局通信操作，我们称之为集合通信。集合通信可以和点对点通信共用一个通信域而不产生消息混淆。集合通信一般实现了三个功能：通信、聚集和同步。通信功能主要完成进程组内进程间数据的传输；聚集功能在通信功能的基础上对给定的数据完成一定的操作，有点类似 OpenMP 中的 reduction；同步功能使进程组内所有进程等待在特定的地点上，从而达到进程执行进度上的一致，有点类似 OpenMP 中的隐式或显示栅栏。

1. 广播

在一个集合通信中，如果属于一个进程的数据被发送到通信域中的所有进程，这样的集合通信就叫做广播（broadcast）。

参考实验指导书4-1节

1. 规约

广播是将消息发散出去，那么归约就是将消息集中起来处理的方法。归约操作是指对分布在不同进程中的数据间进行交互运算，常用的运算有求和、求 最大或最小值等。

程序代码见程序4-2

1. 全局规约

使用 MPI\_Reduce 语句需要指定一个目标进程，也就是说只有一个进程能得到全局的归约结果。但是在某些应用之中，所有的进程都需要得到最后的结果，以便可以完成更灵活的计算。

程序代码见程序4-3

1. 散射

散射（MPI\_Scatter）与广播非常相似，都是一对多的通信方式，不同的是散射是将一段数据的不同部分发送给所有的进程，而广播则是将相同的数据发送给所有的进程。

程序代码见程序4-4

1. 聚集

与散射相对的就是聚集，其作用是从所有的进程中将每个进程的数据集中到目标进程中。

程序代码见程序4-5

1. 全局聚集

与归约类似，聚合也有扩展，叫全局聚集MPI\_Allgather，MPI\_Allgather将分布在所有进程中的数据聚合到每个进程中。

程序代码见程序4-6

1. **实验程序及结果截图**

4-1

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

int n=0;

if(0==rank)

{

cout<<"Enter n:\n";

cin>>n;

}

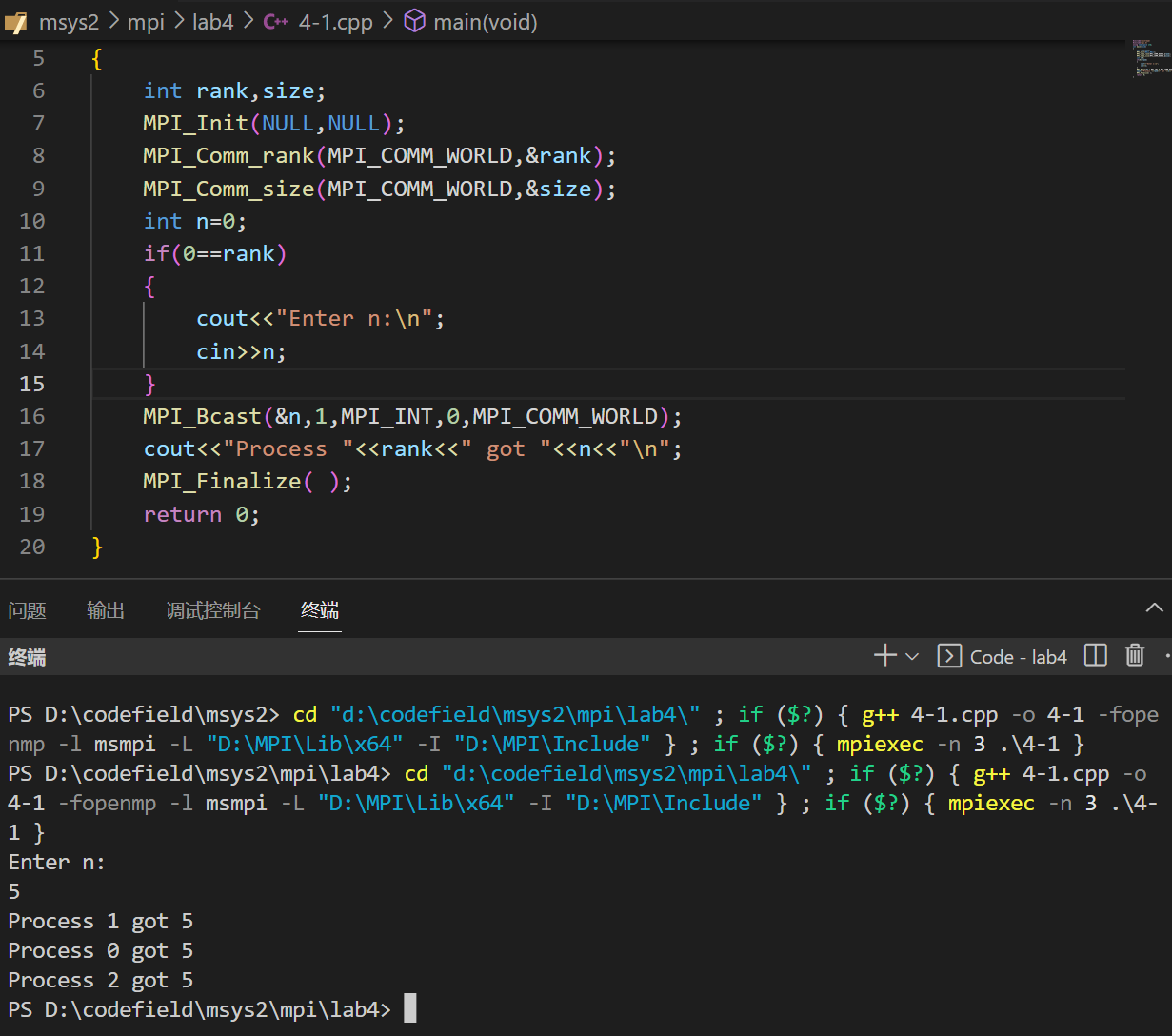
MPI\_Bcast(&n,1,MPI\_INT,0,MPI\_COMM\_WORLD);

cout<<"Process "<<rank<<" got "<<n<<"\n";

MPI\_Finalize( );

return 0;

}



4-2

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size,send\_buff,recv\_buff;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

recv\_buff=0;

send\_buff=rank+10;

MPI\_Reduce(&send\_buff,&recv\_buff,1,MPI\_INT,MPI\_SUM,0,MPI\_COMM\_WORLD);

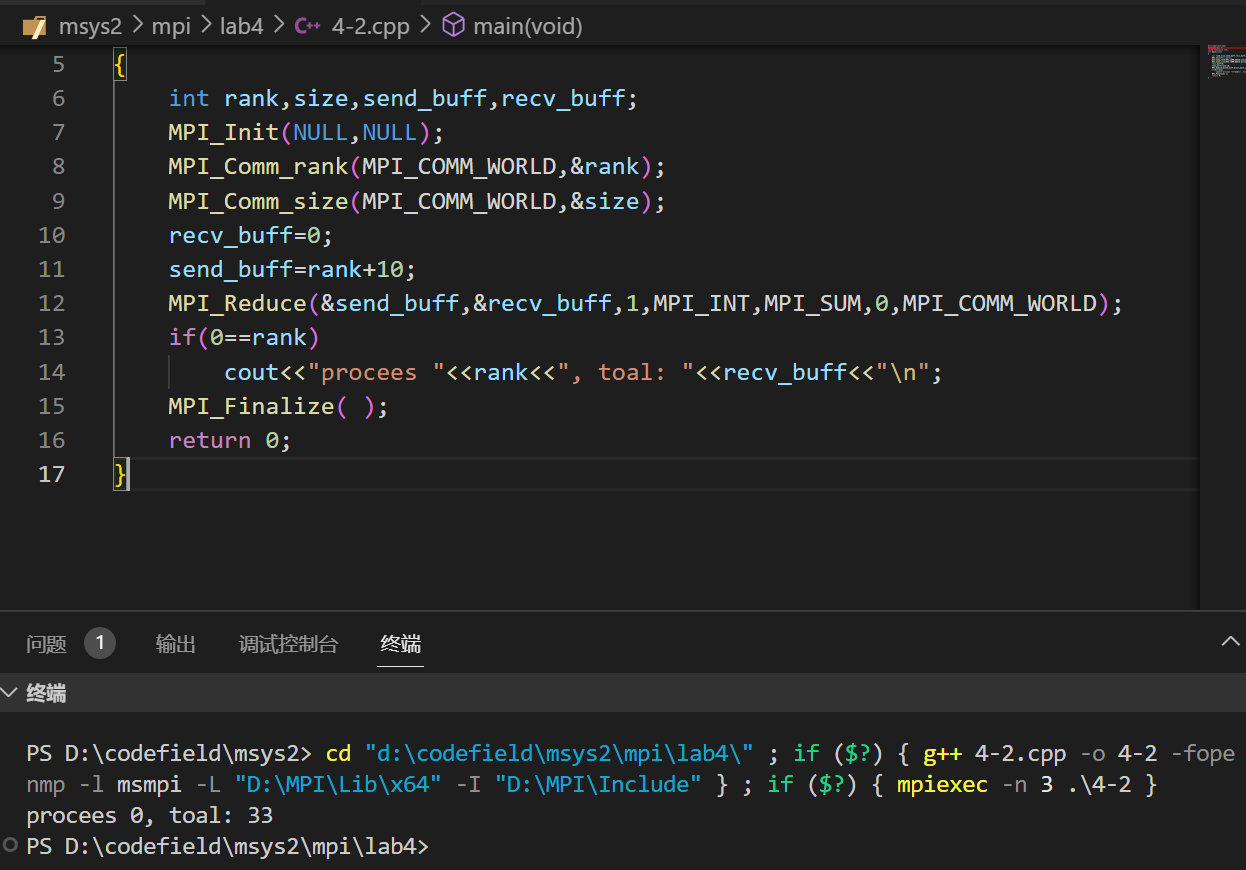
if(0==rank)

cout<<"procees "<<rank<<", toal: "<<recv\_buff<<"\n";

MPI\_Finalize( );

return 0;

}



4-3

#include<iostream>

#include<mpi.h>

using namespace std;

int main(void)

{

int rank,size,send\_buff,recv\_buff;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

recv\_buff=0;

send\_buff=rank;

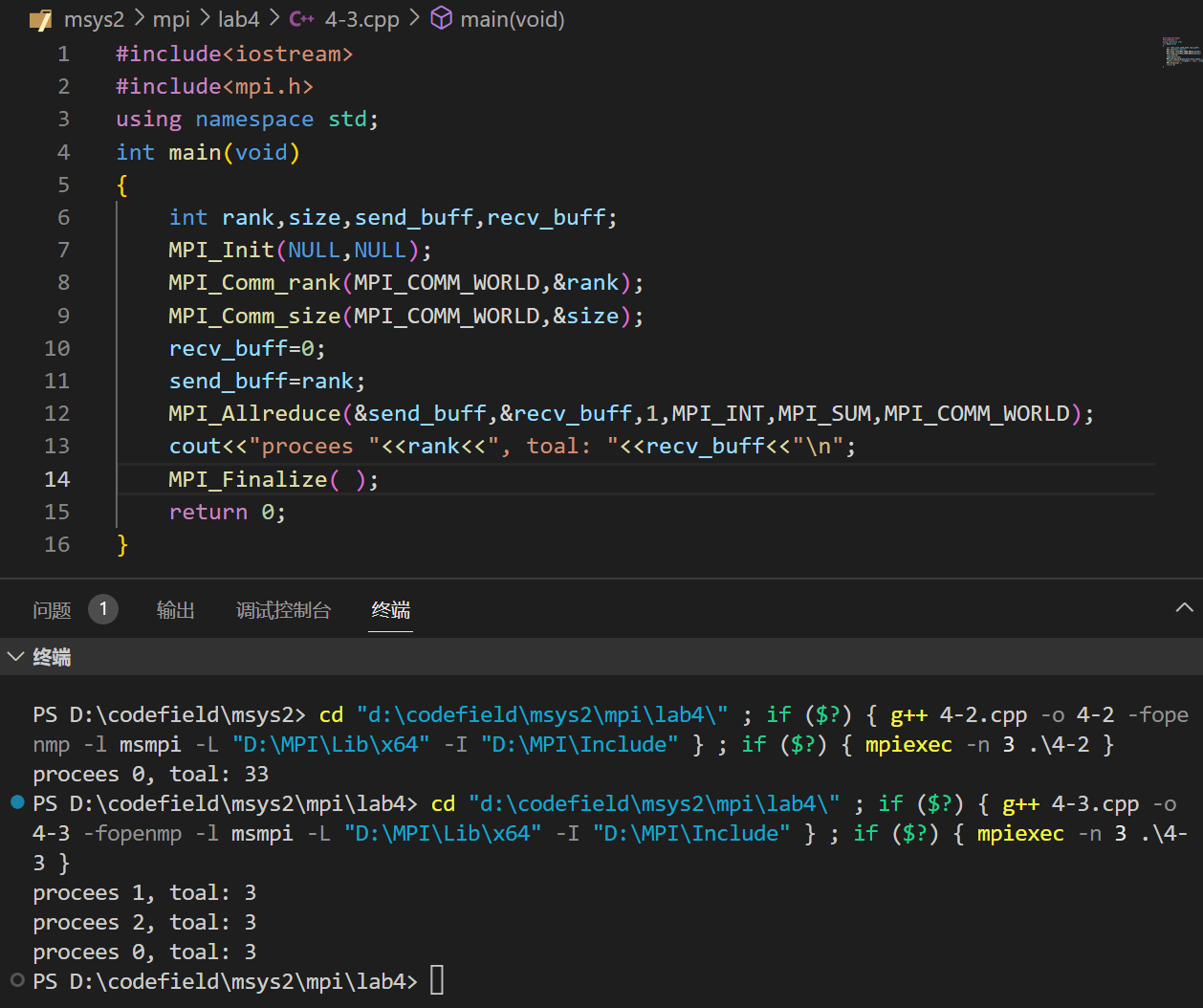
MPI\_Allreduce(&send\_buff,&recv\_buff,1,MPI\_INT,MPI\_SUM,MPI\_COMM\_WORLD);

cout<<"procees "<<rank<<", toal: "<<recv\_buff<<"\n";

MPI\_Finalize( );

return 0;

}



4-4

#include<iostream>

#include<cstdlib>

#include<mpi.h>

using namespace std;

const int N=2;

int main(void)

{

int rank,size,\*send\_buff,\*recv\_buff;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

recv\_buff=(int\*)malloc(N\*sizeof(int));

send\_buff=(int\*)malloc(size\*N\*sizeof(int));

if(0==rank)

{

for(int i=0;i<size\*N;++i)

{

send\_buff[i]=i+10;

cout<<"rank "<<rank<<", send\_buff["<<i<<"] = "<<send\_buff[i]<<"\n";

}

cout<<"\n";

}

MPI\_Scatter(send\_buff,N,MPI\_INT,recv\_buff,N,MPI\_INT,0,MPI\_COMM\_WORLD);

for(int i=0;i<N;++i)

cout<<"rank "<<rank<<", send\_buff["<<i<<"] = "<<recv\_buff[i]<<"\n";

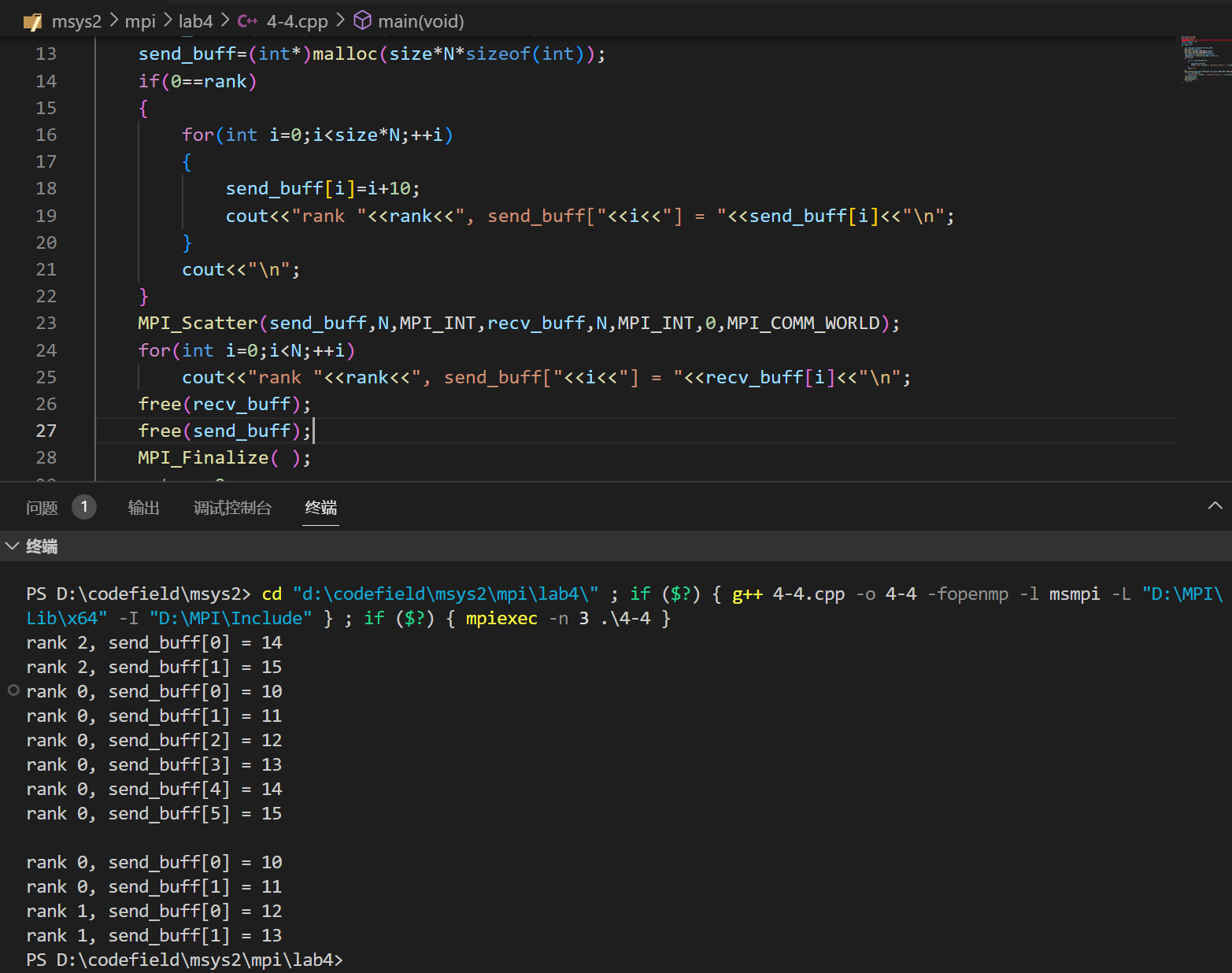
free(recv\_buff);

free(send\_buff);

MPI\_Finalize( );

return 0;

}



4-5

#include<iostream>

#include<cstdlib>

#include<mpi.h>

using namespace std;

const int N=1;

int main(void)

{

int rank,size,send\_buff,\*recv\_buff;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

recv\_buff=(int\*)malloc(size\*N\*sizeof(int));

send\_buff=rank+10;

MPI\_Gather(&send\_buff,N,MPI\_INT,recv\_buff,N,MPI\_INT,0,MPI\_COMM\_WORLD);

if(0==rank)

{

for(int i=0;i<size\*N;++i)

cout<<"rank "<<rank<<", recv\_buff["<<i<<"] = "<<recv\_buff[i]<<"\n";

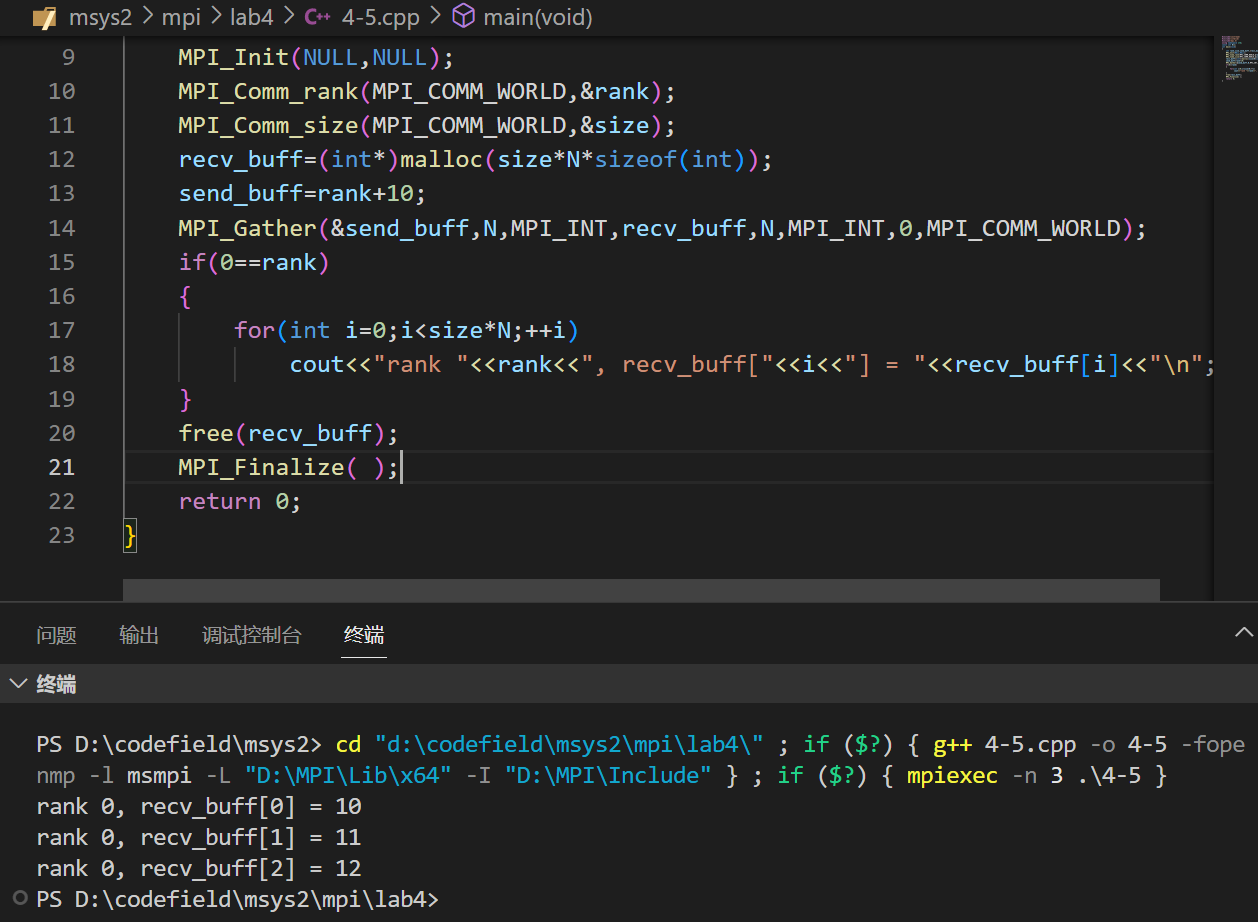
}

free(recv\_buff);

MPI\_Finalize( );

return 0;

}



4-6

#include<iostream>

#include<cstdlib>

#include<mpi.h>

using namespace std;

const int N=1;

int main(void)

{

int rank,size,send\_buff,\*recv\_buff;

MPI\_Init(NULL,NULL);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

recv\_buff=(int\*)malloc(size\*N\*sizeof(int));

send\_buff=rank+10;

MPI\_Allgather(&send\_buff,N,MPI\_INT,recv\_buff,N,MPI\_INT,MPI\_COMM\_WORLD);

for(int i=0;i<size\*N;++i)

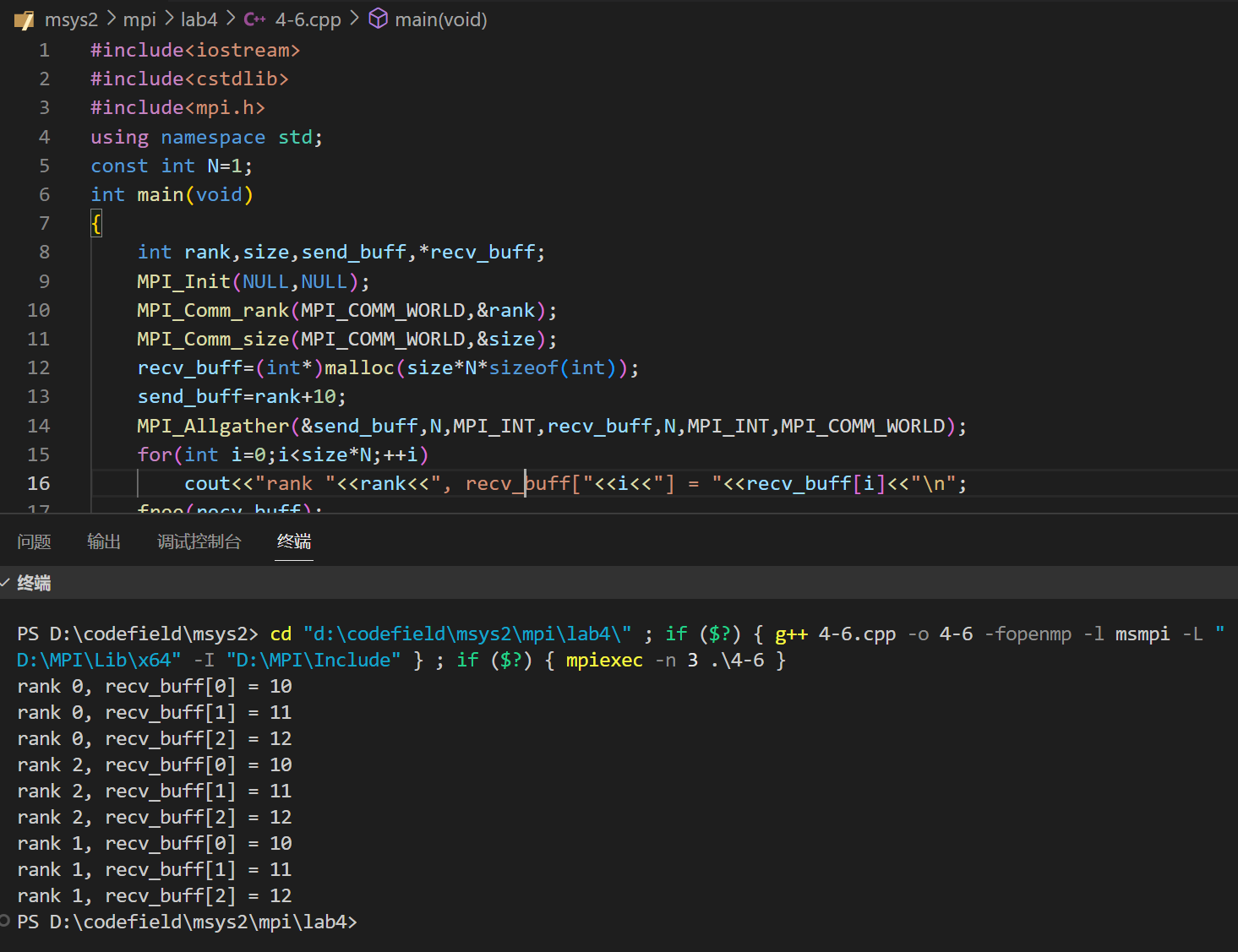
cout<<"rank "<<rank<<", recv\_buff["<<i<<"] = "<<recv\_buff[i]<<"\n";

free(recv\_buff);

MPI\_Finalize( );

return 0;

}



1. **总结**

谈谈实验中遇到的问题及如何解决的，在实验中的一些个人体会。

4-4开始出现的malloc的返回值是void\*类型，需要强制类型转换为int\*类型才能编译通过。广播是同样信息一发多，规约是一受多，全局规约是多受多，散射是分块信息一发多，聚集是多发一，全局聚集是多发多。