实验二

消息传递接口MPI通讯特征

实验指导书

卢本捷

1. 实验目的
   1. 学习MPI的点到点通信的基本方法
   2. 学习阻塞通信和非阻塞通信的不同特征
   3. 学习各种不同的通信模式。
2. 实验内容
3. 测试一共16种通信接收方式。每种方式又分为先发送和后发送。共32种方式。
4. 实验环境
5. 两台或以上的windows 10 等。
6. 节点的网络互联。
7. VC.net 2017,vs 2019 等
8. 或linux 系统及其开发环境。
9. 实验要求
10. 记录实验过程，分析试验现象。
11. 记录主要源代码
12. 撰写实验报告.
13. MPICH2 实验步骤

测试使用代码：

#include "stdafx.h"

#include "mpi.h"

#include <iostream>

#include "windows.h"

using namespace std;

int \_tmain(int *argc*, char\* *argv*[])

{

int rank, size, source =0,dest=1,count=1,tag=1;

int senddata=888,recvdata=777;

MPI\_Status status;

MPI\_Request request;

MPI\_Init(&*argc*, &*argv*); /\*MPI的初始化函数\*/

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); /\*该进程的编号\*/

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); /\*总的进程数目\*/

if (rank==source)

{

//Sleep(1000);

double starttime = MPI\_Wtime();

//阻塞的发送

//int \*buff=new int[100];

//MPI\_Buffer\_attach( buff, 59); //必须大于59.

// MPI\_Bsend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD);

//int ss;

//MPI\_Buffer\_detach(buff,&ss); // 缓冲的发送方式。

//MPI\_Ssend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD);// 同步的发送方式。

//MPI\_Rsend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD);// 就绪的发送方式。

//MPI\_Send(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD); // 标准的发送方式。

//\* 非阻塞的发送

//MPI\_Isend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD,&request); // 标准的发送方式。 非阻塞

//MPI\_Ibsend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD,&request); // 缓冲的发送方式 非阻塞

MPI\_Issend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD,&request);// 同步的发送方式。非阻塞

// MPI\_Irsend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD,&request);// 就绪的发送方式。

MPI\_Wait(&request, &status);

// 非阻塞的发送\*/

double endtime=MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

if (rank==dest)

{

Sleep(1000);

double starttime = MPI\_Wtime();

MPI\_Recv(&recvdata, count, MPI\_INT, source, tag, MPI\_COMM\_WORLD,&status);// 阻塞接收

//MPI\_Irecv(&recvdata, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &request); // 非阻塞接收

//cout<<" 1 recv data : "<<recvdata<<endl;

//MPI\_Wait(&request, &status);

//int flag;

//MPI\_Test(&request,&flag, &status);

//cout<<flag<<endl;

cout<<" recv data : "<<recvdata<<endl;

double endtime= MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

MPI\_Finalize();

return 0;

}

**实验步骤：**

1. **代码框架：**
   1. **所有进程都要做的工作：**

int rank, size, tag=1,source =0,dest=1;

int senddata=888,recvdata=777;

MPI\_Status status;

MPI\_Request request;

MPI\_Init(&*argc*, &*argv*); /\*MPI的初始化函数\*/

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank); /\*该进程的编号\*/

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size); /\*总的进程数目\*/

* 1. **发送进程的工作：**

if (rank==source)

{

//Sleep(1000); // 如果需要的话，休眠一段。

double starttime = MPI\_Wtime();

MPI\_Send(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD);

double endtime=MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;}

* 1. **接收进程的工作：**

if (rank==dest)

{

Sleep(1000); // 如果需要的话，休眠一段。

double starttime = MPI\_Wtime();

MPI\_Recv(&recvdata, count, MPI\_INT, source, tag, MPI\_COMM\_WORLD,&status);

cout<<"recv data : "<<recvdata<<endl;

double endtime= MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

* 1. **所有进程都终止。**

MPI\_Finalize();

return 0;

* 1. **程序执行方法：**

**mpiexec –n 2 G:\MPICH2\examples\x64\Release\cpi.exe**

1. **阻塞发送，阻塞接收 一直等着**
2. **标准发送** 
   1. **先发送**

if (rank==source)

{

double starttime = MPI\_Wtime();

MPI\_Send(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD);

double endtime=MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

if (rank==dest)

{

Sleep(1000);

double starttime = MPI\_Wtime();

MPI\_Recv(&recvdata, count, MPI\_INT, source, tag, MPI\_COMM\_WORLD,&status);

cout<<"recv data : "<<recvdata<<endl;

double endtime= MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

**实验结果：**

发送方立刻返回。接收方随后启动，并接受到数据。

双方时间显示均为0.

* 1. **后发送**

Sleep(1000);放在发送方。则：

实验结果：

接收方会阻塞直至数据到达。

1. **缓冲发送**
   1. **先发送**

发送方：

if (rank==source)

{

double starttime = MPI\_Wtime();

//Sleep(1000);

int \*buff=new int[100];

MPI\_Buffer\_attach( buff, 59); //必须大于59.

int res= MPI\_Bsend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD);

int ss;

MPI\_Buffer\_detach(buff,&ss); // 释放缓冲区 什么意思??

double endtime=MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

接收方不变。

结果：发送方立刻返回。接收方随后启动,并接受到数据。

(与标准发送相同,只不过发送方多了一个缓冲区)

* 1. **后发送**

Sleep(1000) 放到发送方。

结果：

接收方阻塞直至数据到达。

1. **同步发送**
   1. **先发送**

if (rank==source)

{

double starttime = MPI\_Wtime();

MPI\_Ssend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD);// 同步的发送方式。

double endtime=MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

接收方不变。

结果：发送方等待接收方启动，双方同时结束。

* 1. **后发送**

Sleep(1000) 放到发送方。

结果：

接收方阻塞直至数据到达。

**：发送过程自动启动, 只有相应的接收过程已经启动，发送过程才正确返回。**

1. **就绪发送**
   1. **先发送**

if (rank==source)

{

double starttime = MPI\_Wtime();

MPI\_Rsend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD);// 就绪的发送方式。

double endtime=MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

接收方不变

接收方先发送，安照标准会出错。MPICH的实现不出错，等价于缓冲方式。

* 1. **后发送**

Sleep(1000) 放到发送方。

结果：

接收方阻塞直至数据到达。

**发送操作只有在接收进程相应的接收操作已经开始才进行发送。**

1. **阻塞发送，非阻塞接收** 
   1. **标准发送**
      1. **先发送**

**发送方代码同前二.1 ，接收方：**

if (rank==dest)

{

double starttime = MPI\_Wtime();

Sleep(1000);

MPI\_Irecv(&recvdata, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &request); // 非阻塞接收

cout<<" 1 recv data : "<<recvdata<<endl;

MPI\_Wait(&request, &status);

cout<<" 2 recv data : "<<recvdata<<endl;

double endtime= MPI\_Wtime();

cout<<"process "<<rank<<", time is : "<<endtime-starttime<<endl;

}

结果：

发送方立刻返回。

接收方等待1秒启动后，也能立刻返回。但是结果未必已经收到。

第一次打印，没有得到结果。（尽管，已经等待1秒后启动。）

MPI\_Wait 或MPI\_Test后，才得到结果。

MPI\_Test() 检测是否收到数据,如果没有立刻返回.如果有则那数据后再返回。

MPI\_Wait()检测是否收到数据,如果没有阻塞等待.

* + 1. **后发送**

接收方立刻返回，使用MPI\_Test也会立刻返回，但是数据没有接收到，使用MPI\_Wait 会阻塞，直到数据到达。

* 1. **缓冲发送**
     1. **先发送**
     2. **后发送**

特征和标准的一样，差异在于要准备缓冲区。

* 1. **同步发送**
     1. **先发送**

发送方会等待接收方启动，接收方仍然需要Test或者Wait。才能得到数据。

* + 1. **后发送**

接收方立刻返回，但数据没有到达，使用Test不能保证数据，必须Wait。

* 1. **就绪发送**
     1. **先发送**
     2. **后发送**

特征同缓冲的一样。 接受操作已经开始, 发送方才发送数据, 这不已经晚了吗?

1. **非阻塞发送，阻塞接收**
   1. **标准发送**
      1. **先发送**

MPI\_Isend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD,&request);

发送方立即返回。

接收方随后会受到数据

因为发送是主动进行的，所以往往不需要使用MPI\_Wait 等待。

即使使用，也不会阻塞直到接收方受到数据。因为“标准”是基于“缓冲”的

* + 1. **后发送**

接收方会阻塞，等待

发送方调用后立即返回。

* 1. **缓冲发送**
     1. **先发送**
     2. **后发送**

MPI\_Ibsend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD,&request);

和标准的区别只在缓冲区

* 1. **同步发送(也是阻塞??)**
     1. **先发送**

MPI\_Issend(&senddata,count,MPI\_INT,dest,tag,MPI\_COMM\_WORLD,&request);

发送方立即返回。

接收方随后会受到数据

因为发送是主动进行的，所以往往不需要使用MPI\_Wait 等待。?? 难道不等待接收方启动后,发送方才返回吗?

如果使用MPI\_Wait，**会**阻塞直到接收方受到数据。因为是同步的。

* + 1. **后发送**

接收方会阻塞，等待

发送方调用后立即返回。??

* 1. **就绪发送**
     1. **先发送**
     2. **后发送**

特征同缓冲

1. **非阻塞发送，非阻塞接收**

发送方的特征与四完全相同，接收方特征与三完全相同。

MPI 点对点通信特征

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 种类 | 发送 | 接收 | 先发送 后接受 | 后发送，先接受 | 备注 |
| 阻塞发送  阻塞接收 | 标准MPI\_Send | 阻塞  MPI\_Recv  MPI\_Status status;  MPI\_Recv(&recvdata, count, MPI\_INT, source, tag, MPI\_COMM\_WORLD, &status) | MPICH按照缓冲的方式实现，但提供了足够的缓冲区。 见下。 | | MPI\_Send(&senddata, count, MPI\_INT,dest, tag, MPI\_COMM\_WORLD) |
| 同步  MPI\_ Ssend | 后启动的等待对方，双方同时结束。 | | MPI\_Ssend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD) |
| 缓冲  MPI\_ Bsend | * 发送方立即返回 * 接收方启动后立即可接收到数据 * 发送方返回后要显示调用MPI\_Buffer\_detach(buff,&size);来释放缓冲区，否则如果直接退出，接收方会收不到信息。（注意进程并不会在MPI\_Buffer\_detach上阻塞。 | * 接收方阻塞，等待发送方 * 发送方发送立即成功。 | 需要事先准备缓冲区  int \*buff=new int[100]; MPI\_Buffer\_attach( buff, 400); //必须大于59.  int data=333;  MPI\_Bsend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD);  MPI\_Buffer\_detach(buff,&size);  注： 缓冲区和发送的内容不同。 |
| 就绪  MPI\_ Rsend | 接收方必须先启动，否则出错，但是MPICH的实现不出错，等同于缓冲形式。 | | MPI\_Rsend(&senddata, count, MPI\_INT, dest, tag, MPI\_COMM\_WORLD); |
|  | | | | | |
| 阻塞发送  非阻塞接收 | 标准MPI\_Send | 非阻塞  MPI\_Irecv  MPI\_Status status;  MPI\_Request request; MPI\_Irecv(&recvdata, count, MPI\_INT, source, tag, MPI\_COMM\_WORLD, &request); MPI\_Wait(&request, &status); | * 发送方立刻返回，数据已经在缓冲区。 * 接收方也立刻返回，但不一定已经成功接收。需要使用MPI\_Wait(&request, &status) 会阻塞直至成功返回； | * 接收方立刻返回，但是显然还没有结果。 * 发送方立刻返回。 * 接收方阻塞在MPI\_Wait中，直至得到结果 * 或使用MPI\_TEST检测（但是非缓冲的，立刻返回，有可能仍然未成功）MPI\_Test(&request,&flag, &status); | 实际上  MPI\_Irecv + MPI\_Wait  等价于MPI\_Recv |
| 同步  MPI\_ Ssend | * 发送方会等待对方。 * 接收方会立刻返回，结果仍然要依靠wait | * 接收方会立刻返回，结果仍然要依靠wait * 发送方也立刻返回。 |  |
| 缓冲  MPI\_ Bsend | 当使用MPI\_Irecv + MPI\_Wait 特征同阻塞发送+  阻塞接收 | |  |
| 就绪  MPI\_ Rsend | 同上 | |  |
|  | | | | | |
| 非阻塞发送  阻塞接收 | 标准MPI\_ISend | 阻塞  MPI\_Recv | * 发送方立即返回。 * 接收方随后会受到数据 * 因为发送是主动进行的，所以往往不需要使用MPI\_Wait 等待。 * 即使使用，也不会阻塞直到接收方受到数据。因为“标准”是基于“缓冲”的 | * 接收方会阻塞，等待 * 发送方调用后立即返回。 | MPI\_Isend(&senddata, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD,&request);  MPI\_Isend+ MPI\_Wait等价于 MPI\_Send |
| 同步  MPI\_I Ssend | * 发送方立即返回。 * 接收方随后会受到数据 * 因为发送是主动进行的，所以往往不需要使用MPI\_Wait 等待。 * 如果使用，**会**阻塞直到接收方受到数据。因为是同步的。 | * 接收方会阻塞，等待 * 发送方调用后立即返回。 | MPI\_Issend(&senddata, 1, MPI\_INT, 1, tag, MPI\_COMM\_WORLD,&request);  MPI\_ISsend+ MPI\_Wait等价于 MPI\_Ssend |
| 缓冲  MPI\_ IBsend | 因为是“缓冲”，跟阻塞的情形很类似。 | |  |
| 就绪  MPI\_I Rsend | 同上 | |  |
|  | | | | | |
| 非阻塞发送  非阻塞接收 | 标准MPI\_ISend | 非阻塞  MPI\_Irecv | 与非阻塞发送+阻塞接收的不同，只表现在  MPI\_Irecv + MPI\_Wait=MPI\_Recv 上。 | | |
| 同步  MPI\_I Ssend |
| 缓冲  MPI\_I Bsend |
| 就绪  MPI\_I Rsend |
|  |  |  |  |  |  |

**简化记忆：**

1. 缓冲，就绪，标准 这三种发送模式等价，都是基于缓冲的，但缓冲模式需要自己准备缓冲区。同步是另外的模式。所以一般只考虑标准和同步模式。
2. 发送时，只有阻塞的同步模式可能阻塞，其他模式都会立刻返回。
3. 非阻塞的操作+Wait等价于阻塞的对应操作。（包括发送和接收）
4. 发送时，即便是非阻塞模式，一般也无需Wait或Test。
5. 接收时，非阻塞模式，必须使用Wait或Test。
6. 典型的使用方法：(推荐的使用方法)
   1. 发送
      1. 阻塞：标准MPI\_Send 和同步MPI\_Ssend
      2. 不阻塞：一般不用，因为实际上标准的，即缓冲的，就会立刻返回。
   2. 接收
      1. 阻塞：MPI\_Recv 适用于多数情形。
      2. 不阻塞的：MPI\_Irecv+Wait 等价于阻塞情形。

若要提高效率使用MPI\_Irecv+Test