高性能计算系统考试

张祎维 202118013229021

2022 年 5 月 27 日

目录

1

1.1

子函数生成行和列通讯子。

程序如下:

```
1  void mesh(int iam, int np, MPI_Comm comm, int p, int q, int *
       myrow, int *mycol, MPI_Comm *rowcom, MPI_Comm *colcom)
2  {
3      int color, key;
4      if( np < p*q ) return;
5      if( iam < p*q ) color = iam / q;
6      else color = MPI_UNDEFINED;
7      key = iam;
8      MPI_Comm_split( comm, color, key, rowcom );
9
10     /*column communicator*/
11     if( iam < p*q ) color = iam % q;
12     else color = MPI_UNDEFINED;
13     key = iam;
14     MPI_Comm_split( comm, color, key, colcom );
15     if( iam <p*q ) {
16         MPI_Comm_rank( *colcom, myrow );
17         MPI_Comm_rank( *rowcom, mycol );
18     }
19     return;
20
21 }
```

1.2

由于

$$\begin{aligned} s &= i/p \\ t &= j/q \\ k &= i\%p \\ l &= j\%q \end{aligned} \tag{1}$$

因此

$$\begin{aligned} i &= p*s+k \\ j &= q*t+l \end{aligned} \tag{2}$$

即有进程 P_{st} 上 a_{kl} 之值为

$$a_{kl} = (p * s + k) + (q * t + l) \tag{3}$$

1.3

程序片段实现将 P_{00} 的整形数据 A 广播给 $p \times q$ 网格的所有进程。
程序如下，其中 mesh 函数定义见第 1 问。

```
int p, q; // 自定义
int myrank, numprocs;
MPI_Comm global_comm,row_comm,col_comm;

MPI_Comm_dup(MPI_COMM_WORLD,&global_comm);
MPI_Comm_rank(global_comm,&myrank);
MPI_Comm_size(global_comm,&numprocs);

int row,col;

mesh(myrank,numprocs,global_comm,p,q,&row,&col,&row_comm,&
    col_comm);

int a; // 自定义, p00, a为p00的数据
if(row == 0 && col == 0)
    MPI_Bcast(&a,1,MPI_INT,0,row_comm);
MPI_Barrier(row_comm);
MPI_Bcast(&a,1,MPI_INT,col,col_comm);
```

2

2.1

可以发送和接收小块矩阵 A_{00}，也可以一次性发送和接收 A_{00} 和 A_{20}
程序如下：

```
// 这里m=n=m
void make_newtype(int m, int n, int lda,MPI_Datatype * newtype){
    MPI_Datatype vec_type;
    const int count = m;
    const int length = n;
    const int stride = lda;

    MPI_Type_vector(count, length, stride, MPI_INT, &vec_type);
    MPI_Type_create_resized(vec_type,0,sizeof(int)*(m*lda*2),
       newtype);
    MPI_Type_commit(&newtype);
}
```

2.2

如果只是发送 A_{00},A_{20}，是否还有其他方法？
答：有，构造方法如下，使用 MPI_Type_indexed 来构造。

```
// 这里m=n=m
void make_newtype1(int m, int n, int lda, MPI_Datatype *newtype){
    int count = 2 * m;
    int blocklengths[2*m] = {n, n, n, n,...,n},
    displacements[2*m] = {0,lda,2*lda,..., (m-1)*lda, 2*m*lda,
       ..., (3*m-1)*lda};
    MPI_Datatype newtype;
    MPI_Type_indexed(count, blocklengts, displacements, MPI_INT,
       &newtype);
    MPI_Type_commit(&newtype);
}
```

3

如果将进程 0 的数据 x 之前 5 个元素发送给进程 1，请写出相应的程序片断。

程序如下：

```
typedef struct
{
    int m[3];
    float a[2];
    char c[5];
}mixtype;

void mpistruct(newtp)
MPI_Datatype *newtype;
{
    mixtype s;
    MPI_Datatype oldtype[3];
    int blocklen[3];
    MPI_Aint displaces[3];
    MPI_Get_address(&s.m[0], &displaces[0]);
    MPI_Get_address(&s.a[0], &displaces[1]);
    MPI_Get_address(&s.c[0], &displaces[2]);
    displaces[1] -= displaces[0];
    displaces[2] -= displaces[0];
    displaces[0] = 0;
    blocklen[0] = 3;
    blocklen[1] = 2;
    blocklen[2] = 5;
    oldtype[0] = MPI_INT;
    oldtype[1] = MPI_FLOAT;
    oldtype[2] = MPI_CHAR;

    MPI_Type_create_struct(3, blocklen, displaces, oldtype,
       newtype);
}

int myrank, numprocs;
MPI_Comm global_comm;
MPI_Status st;
MPI_Comm_dup(MPI_COMM_WORLD,&global_comm);
MPI_Comm_rank(global_comm,&myrank);
MPI_Comm_size(global_comm,&numprocs);
if (myrank == 0) MPI_Send(x, 5, newtp, 1, 99, comm);
if (myrank == 1) MPI_Recv(x, 5, newtp, 0, 99, comm, &st);
```

4

4.1

构造自己的 MPI_Allgather 函数 mpi_allgatherSelfImpl。

使用 MPI_Send 函数和 MPI_Recv 函数实现。

在每个进程中，都顺序的向所有发送本进程的实型数据，之后顺序的接受来自所有进程的实型数据，构成一个 $n \times n$ 的阵列（n 为进程数，指每个进程中都有一个长度为 n 的实型数组，存储来自这 n 个进程的数据）。

4.2

程序如下，在 c 语言中的实型按 MPI_FLOAT 来处理：

```
1  void mpi_allgatherSelfImpl(const void *sendbuf, int sendcount,
       MPI_Datatype sendtype, void *recvbuf, int recvcount,
       MPI_Datatype recvtype, MPI_Comm comm)
2  {
3      int rank,nproc;
4      MPI_Comm_size( MPI_COMM_WORLD, &nproc );
5      MPI_Comm_rank( MPI_COMM_WORLD, &rank );
6
7      for (int i = 0; i < nproc; ++i){
8          MPI_send(sendbuf, sendcount, sendtype, i, i, comm);
9      }
10
11     for (int i = 0; i < nproc; ++i){
12         MPI_Status status;
13         MPI_Recv(recvbuf+i*4,recvcount,recvtype,i,rank,comm,&
               status);
14     }
15 }
16 int main(int argc, char  *argv[])
17 {
18     int rank, nproc;
19     MPI_Init(&argc, &argv );
20     MPI_Comm_size( MPI_COMM_WORLD, &nproc );
21     MPI_Comm_rank( MPI_COMM_WORLD, &rank );
22     float sendData=rank+1;
23     float receiveData[dataLength];
24     mpi_allgatherSelfImpl(&sendData,1,MPI_FLOAT,receiveData,1,
           MPI_FLOAT,MPI_COMM_WORLD);
25     MPI_Finalize();
26     return 0;
27 }
```