高性能计算复习

张祎维 202118013229021

2022 年 5 月 27 日

目录

1 第一题 行、列阵子

二维进程网格为 pxq，生成行列通讯子

使用程序片段实现将 P_{00} 的数据 A 广播给 $p \times q$ 网格的所有进程。

```c
#include <stdio.h>
#include "mpi.h"
void mesh( iam,  np, comm, p, q, myrow, mycol, \
            rowcom, colcom )
int iam, np, p, q, *myrow, *mycol; MPI_Comm comm, *rowcom, *colcom;
{
    int color, key;
    if( np < p*q ) return;
    if( iam < p*q ) color = iam / q;
    else color = MPI_UNDEFINED;
    key = iam;
    MPI_Comm_split( comm, color, key, rowcom );

    /*column communicator*/
    if( iam < p*q ) color = iam % q;
    else color = MPI_UNDEFINED;
    key = iam;
    MPI_Comm_split( comm, color, key, colcom );
    if( iam <p*q ) {
        MPI_Comm_rank( *colcom, myrow );
        MPI_Comm_rank( *rowcom, mycol );
    }
    return;

}

int main(int args, char ** argv){
    MPI_Init(&args,&argv);

    const int m = 100;
    const int n = 100;
    const int p = 5;
    const int q = 5;

    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);

    float a;
    if(row == 0 && col == 0){
        a = 100.1;
    }
    else a = -1;
    //p00, a为p00的数据
    if(row == 0 && col == 0)
        MPI_Bcast(&a,1,MPI_FLOAT,0,row_comm);
    MPI_Barrier(row_comm);
    MPI_Bcast(&a,1,MPI_FLOAT,col,col_comm);

    MPI_Finalize();
    return 0;
}
```

k = i mod q; l = j mod q; s = i / p; t = j / q

2 第二题 MPI_Vector

分块矩阵 A = A00 A01 ..... A10 A11 .... 其中 Aij 是 mxn 阶矩阵,构造新数据类型，可以发送和接收 A00，也可以一次性发送和接收 A00,A11

```c
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+n),
       newtype);
    MPI_Type_commit(&newtype);
}
```

如果只是发送 A00,A11，是否还有其他方法

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

3 第三题 MPI_Struct

设结构 int m[3]; float a[2]; char c[5]; 定义的数组为 x[10]，如果将进程 0 的数据 x 发送给进程 1，请写出相应的程序片断。

```
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, 10, newtp, 1, 5, comm);
if (myrank == 1) MPI_Recv(x, 10, newtp, 0, 5, comm, &st);
```

4 第四题 ALL_GATHER

ALL_GATHER

```c
void mpi_allgatherSelfImpl(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)
{
    int rank,nproc;
    MPI_Comm_size( MPI_COMM_WORLD, &nproc );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    for (int i = 0; i < nproc; ++i)
    {
        MPI_send( sendbuf, sendcount, sendtype, i, i, comm);
    }

    for (int i = 0; i < nproc; ++i)
    {
        MPI_Status status;
        MPI_Recv(recvbuf+i*4,recvcount,recvtype,i,rank,comm,&status);
    }
}
int main(int argc, char  *argv[])
{
    int rank,nproc;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &nproc );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    int sendData=rank+1;
    int receiveData[dataLength];
    mpi_allgatherSelfImpl(&sendData,1,MPI_INT,receiveData,1,MPI_INT,MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
}
```

5 实现 ALLTOALL

实现 ALLTOALL

```c
//用send,recv 等实现MPI_Alltoall
void all2all(sendbuf, sendcount, sendtype, recvbuf,recvcount, recvtype, comm, iam, np)
    MPI_Comm comm;
    int iam, np, sendcount, recvcount;
    float* sendbuf, * recvbuf;
    MPI_Datatype sendtype, recvtype;
{
    MPI_Status st;
    //int front, next;
    for (int i = 0; i < np; i++)
    {
        if (iam == i)
        {
            MPI_Sendrecv(&sendbuf[sendcount * i], sendcount, sendtype, i, 100,
            &recvbuf[recvcount * i], recvcount, recvtype, i, 100, comm, &st);
        }
        if (iam != i)
        {
            MPI_Send(&sendbuf[sendcount * i], sendcount, sendtype, i, i, comm);
            MPI_Recv(&recvbuf[recvcount * i], recvcount, recvtype, i, iam, comm, &st);
        }
    }
    return;
}

for (int i = 0; i < 31; i++)
    for (int j = 0; j < 57; j++)
        a[i][j] = i + j;
all2all(a, 1, MPI_FLOAT, b,1, MPI_FLOAT, comm, iam, np);
printf("\n a = %f,%f,%f on process %d", b[0][0], b[0][1], b[0][2], iam);
```

6 MPI_Bcast 广播函数

MPI_Bcast 广播函数

若节点为 root 则接收来自所有其他除 root 以外的节点的消息，否则向 root 节点发送一条消息。

```c
void My_Bcast(void* sendAddress, int count, MPI_Datatype datatype, int root, MPI_Comm comm) {
    int rank, size, i;
    MPI_Status status;
    int tag = 100;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (rank == root) {
        // If we are the root process, send our data to everyone
        for (i = 0; i < size; i++) {
            if (i != root) {
                MPI_Send(sendAddress, count, datatype, i, tag, comm);
            }
        }
    } else {
        // If we are a receiver process, receive the data from the root
        MPI_Recv(sendAddress, count, datatype, root, tag, comm, &status);
    }
}
```

7 MPI_Gather 收集函数

MPI_Gather 收集函数

当前节点向 root 节点发送一条消息，如果当前节点是 root 节点则枚举接收来自所有节点的消息。

```c
void My_Gather(void* sendAddress, int sendCount, MPI_Datatype sendDatatype, void* recvAddress, int recvCount, MPI_Datatype recvDatatype, int root, MPI_Comm comm)
{
    int rank, size, i;
    int tag = 101;
    MPI_Status status;
    MPI_Request request;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Isend(sendAddress, sendCount, sendDatatype, root, tag, comm, &request);
    if (rank == root) {
        int tsize;
        MPI_Type_size(recvDatatype, &tsize);
        for (i = 0; i < size; i++) {
            MPI_Recv(recvAddress + i * recvCount * tsize, recvCount, recvDatatype, i, tag, comm, &status);
        }
    }
}
```

8 MPI_Scatter 散播函数

判断该节点是否是 root 节点，若是则向所有节点发送一条消息（非阻塞 MPI_Isend）。然后该节点接收一条来自于 root 的消息。

```c
void My_Scatter(void* sendAddress, int sendCount, MPI_Datatype sendDatatype, void* recvAddress, int recvCount, MPI_Datatype recvDatatype, int root, MPI_Comm comm)
{
    int rank, size, i;
    int tag = 102;
    MPI_Request request;
    MPI_Status status;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (rank == root) {
        int tsize;
        MPI_Type_size(sendDatatype, &tsize);
        for (i = 0; i < size; i++) {
            MPI_Isend(sendAddress + i * sendCount * tsize, sendCount, sendDatatype, i, tag, comm, &request);
        }
    }

    MPI_Recv(recvAddress, recvCount, recvDatatype, root, tag, comm, &status);
}
```

9 CANNON

```
//Cannon Algorithm implemetation,the each A as m*k,and B is k*n,
//so the total matrix size is np*m*nq*k for matrix A,and forth.
//a[i][j] = a[i * lda + j]
void cannon(rowcom, colcom, p, myrow, mycol, m, k, n, a, lda, b, ldb, c, ldc, at, ldaw, bt, ldbw)
MPI_Comm rowcom, colcom;
int p, myrow, mycol, m, k, n, lda, ldb, ldc, ldaw, ldbw;
float* a, * b, * c, * at, * bt;//矩阵用一维定义
{
    int i, j, l, front, next;
    MPI_Status st;
    MPI_Datatype btp, attp, bttp;//定义新的数据类型
    // typemat(m,k,lda,&atp);
    // MPI_Type_commit(&atp);
    typemat(k, n, ldb, &btp);
    MPI_Type_commit(&btp);
    typemat(m, k, ldaw, &attp);
    MPI_Type_commit(&attp);
    typemat(k, n, ldbw, &bttp);
    MPI_Type_commit(&bttp);

    l = myrow;
    front = (myrow - 1 + p) % p;
    next = (myrow + 1) % p;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            c[i * ldc + j] = 0.0;
        }
    }

    for (i = 0; i < p; i++)
    {
        if (mycol == l) //对角线上的元素
        {
            scopy(m, k, a, lda, at, ldaw);//把a复制给at
        }
        MPI_Bcast(at, 1, attp, l, rowcom);//对角线上的元素广播到行的每个元素
        gemmm(m, k, n, at, ldaw, b, ldb, c, ldc);
```

```c
        if (i == p - 1) continue;
        //把b的列移动
        MPI_Sendrecv(b, 1, btp, front, 1, bt, 1, bttp, next, 1, colcom, &st);
        scopy(k, n, bt, ldbw, b, ldb);
        l = (l + 1) % p;
    }
    return;
}


p = 3;
if (np < 9) return;
proc2d(comm, np, iam, p, p, &rowcom, &colcom, &rowid, &colid);
/*
if (iam == 0)
{
    fp = fopen("inputmkn.txt", "r");
    i = fscanf(fp, "%*[^\n%*c");
    i = fscanf(fp, "%*[^\n%*c %d,%d,%d", &bnp[0], &bnp[1], &bnp[2]);
    fclose(fp);
    printf("nm = %d, k = %d, n = %d\n", bnp[0], bnp[1], bnp[2]);
}
MPI_Bcast(bnp, 3, MPI_INT, 0, comm);*/
m = 11;
k = 10;
n = 12;
if (iam < 9)
{
    setinittab(p, rowid, colid, m, k, n, &a[0][0], 57, &b[0][0], 59);
    cannon(rowcom, colcom, p, rowid, colid, m, k, n, &a[0][0], 57, & b[0][0], 59, &c[0][0], 61, &w[0][0], 53, &u[0][0], 41);
    printf("\n c = %f,%f,%f,%f on process %d\n", c[1][1], c[1][2], c[1][3], c[1][4], iam);
}
```

10 $y = Ax + b$

假设 n = m x p, A 是 nxn 的矩阵，并行计算 y = Ax + b。其中 A 按列分块存放在处理机中，即处理器 Pi 中存放 A 的第 i 个列块，仍记为 A，这是 A 是 nxm 矩阵。在每个处理机中的 x 是能够与 A 相乘的部分，也就是一个 m 为向量。b 存放在 P0 中。写出计算 Y 的子程序，并把最终结果放入 x 中。

```c
void gmv(m, a, x, y)
int m;
int* a, * x, *y;
{
    int i, j;
    for (i = 0; i < m; i++)
        for (j = 0; j < m; j++)
            y[i] += a[i * m + j] * x[j];
    return;
}

void cpy(m, x, y)
int m;
int* x, * y;
{
    for (int i = 0; i < m; i++)
        x[i] = y[i];
}

void mv(comm, a, x, b, y, w, m, n, np, iam)
MPI_Comm comm;
int m, n, np, iam;
int* a, * b, * x, *y,*w;
{
    int i, j, l;
    int front, next;
    MPI_Status st;
    front = (np + iam - 1) % np;
    next = (iam + 1) % np;
    l = 0;
    for (i = 0; i < m; i++)
    {
        y[i] = b[i];
        w[i] = x[i];

    }
    for (i = 0; i < np - 1; i++)
    {
        if (i % 2 == 0)
        {
            gmv(m, &a[l], x, y);
            MPI_Sendrecv(x, m, MPI_INT, front, 1, w, m, MPI_INT,
               next, 1, comm, &st);
        }
        else
        {
            gmv(m, &a[l], w, y);
            MPI_Sendrecv(w, m, MPI_INT, front, 1, x, m, MPI_INT,
               next, 1, comm, &st);
        }
        l += m * m;
        if (l == n * m) l == 0;
    }
    if ((np - 1) % 2 == 0)
    {
        gmv(m, &a[l], x, y);
    }
    else
    {
        gmv(m, &a[l], w, y);
    }
    cpy(m, x, y);
}
```

11 行列分块矩阵乘

The matrix A is partitioned by row and B by column,$a_{ij} = i + j$ $b_{ij} = 1$ if J is even, else -1

```
// m X k is the block matrix order in iam,the full matrix A is np * m X k mareix
//把A矩阵分成了np个m*k的小矩阵

void matmul(m, k, n, lda, a, ldb, b, ldc, c)
int m, k, n, lda, ldb, ldc;
float *a, *b, *c;
{
    int i, j, l;
    for (i = 0; i < m; i++)
        for (j = 0; j < n; j++)
        {
            c[i*ldc + j] = 0.0;
            for (l = 0; l < k; l++)
                c[i * ldc + j] += a[i * lda + l] * b[l * ldb + j];
        }
    return;
}
```

12 row-column partitioned algorithm

```c
void rcmatmul(comm, np, iam, m, k, n, lda, a, ldb, b, ldc, c, ldw, w)
//w 临时空间， send b,recv w。m是原矩阵的行数除以np,n(原矩阵n * np)列数除以np.k为a列数， b行数.
MPI_Comm comm;
int np, iam, m, k, n, lda, ldb, ldc, ldw;
//float a[][lda], b[][ldb], c[][ldc], w[][ldw];
float *a, *b, *c, *w;
{
    int i, front, next, l;
    MPI_Datatype rectb, rectw;
    MPI_Status st;
    //create a new datatype for matrix b
    MPI_Type_vector(k, n, ldb, MPI_FLOAT, &rectb);
    MPI_Type_vector(k, n, ldw, MPI_FLOAT, &rectw);
    MPI_Type_commit(&rectb);
    MPI_Type_commit(&rectw);
    l = iam * n;
    front = (np + iam - 1) % np;
    next = (iam + 1) % np;
    for (i = 0; i < np - 1; i++)
    {
        if (i % 2 == 0)
        {
            matmul(m, k, n, lda, a, ldb, b, ldc, &c[l]);
            MPI_Sendrecv(b, 1, rectb, front, 1, w, 1, rectw, next, 1, comm, &st);
        }
        else
        {
            matmul(m, k, n, lda, a, ldw, w, ldc, &c[l]);
            MPI_Sendrecv(w, 1, rectw, front, 1, b, 1, rectb, next, 1, comm, &st);
        }
        l += n;
        if (l == np * n) l = 0;
    }
    if ((np - 1) % 2 == 0)
        matmul(m, k, n, lda, a, ldb, b, ldc, &c[l]);
    else
        matmul(m, k, n, lda, a, ldw, w, ldc, &c[l]);
    MPI_Type_free(&rectb);
    MPI_Type_free(&rectw);
    return;
}
```

13 RING

```c
void ring( m, n, comm, np, iam)
int m, *n, np, iam;
MPI_Comm comm;
{
    int next = (iam + 1) % np, front = (np + iam - 1) % np, tag = 1;
    MPI_Status st;
    if( iam % 2 == 0 ) {
        MPI_Send( &m, 1, MPI_INT, next, tag, comm );
        MPI_Recv( n, 1, MPI_INT, front, tag, comm, &st );
    }
    else {
        MPI_Recv( n, 1, MPI_INT, front, tag, comm, &st );
        MPI_Send( &m, 1, MPI_INT, next, tag, comm );
    }
    return;
}
```

14 列迭代

```
void iteration(comm, np, iam, n, en, a, lda, b, x, num)
//每个进程上矩阵的列数，n/np.en 代表每个进程中的列数,num 代表迭代次数
MPI_Comm comm;
int np, iam, n, en, lda, num;
float* a, * b, * x;
{
    int i, j, *rc;
    float *y;
    rc = (int* )malloc(np * sizeof(int));
    for (i = 0; i < np; i++) rc[i] = en;//接收块的大小
    y = (float* )malloc(n * sizeof(float));//y = ax + b;
    for (i = 0; i < num; i++)
    {
        if (iam == 0) //假设iam = 0的时候有b,其他情况没有b
            for (j = 0; j < n; j++) y[j] = b[j];
        else
            for (j = 0; j < n; j++) y[j] = 0.0;
        gemmv(n, en, a, lda, x, y);
        MPI_Reduce_scatter(y, x, rc, MPI_FLOAT, MPI_SUM, comm);
    }
    free(y);
    free(rc);
    return;
}
```

15 题目 1 MPI_Alltoall 2

使用 MPI_Sendrecv(),MPI_Send(),MPI_Recv() 实现 MPI_Alltoall

```
//*使用MPI_Sendrecv(),MPI_Send(),MPI_Recv()实现MPI_Alltoall*//
#include "mpi.h"
#include "stdio.h"

#define  maxlen 10
//ALLtoAll 函数
int  My_Alltoall(int  *sendBuffer,int sendcnt,MPI_Datatype sendtype, int *receiveBuffer,int recvcnt,MPI_Datatype recvtype,MPI_Comm comm,int rank,int size)
{
    int i;
    int j;
    MPI_Status status;

    if(size!=sendcnt||sendtype!=recvtype)
        return 0;
    for(i=0;i<size;i++)
    {
        if(rank==i)
        {
        MPI_Sendrecv(&sendBuffer[i],1,sendtype,i,99,&receiveBuffer[i],1,recvtype,i,99,comm,&status);
        }
        else
        {
        MPI_Send(&sendBuffer[i],1,sendtype,i,i,comm);
        MPI_Recv(&receiveBuffer[i],1,recvtype,i,rank,comm,&status);
        }

    }
    return 1;
}

int main(int argc,char *argv[])
{
    int rank,size;
    MPI_Status status;


```

```c
    int sendBuffer[maxlen],receiveBuffer[maxlen];
    int i,j;
    int count;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    //判断进程数是否合法
    if( size < 1 || size > 10 )
        { if( rank == 0 ) printf("Please input a  number between 1-10\n");
        MPI_Finalize();
        return 0;
        }
    count=size;

    for(i=0;i<maxlen;i++)
    {
        sendBuffer[i]=(rank+1)*(i+1); //初始化发送缓冲区
        receiveBuffer[i]=0;   //初始化接收缓冲区
    }


    My_Alltoall(sendBuffer,count,MPI_INT,receiveBuffer,count,MPI_INT,MPI_COMM_WORLD,rank,size);

    if(rank==0)
    {
        for(i=0;i<count;i++)
        {
        printf("%d\t",receiveBuffer[i]);
        }
    }

    MPI_Finalize();
    return (0);
}
```

16 题目 2 下三角矩阵取出

给你一个方阵，将它的下三角矩阵取出来传给另一个进程

```
//*给你一个方阵，将它的下三角矩阵取出来传给另一个进程
#include "mpi.h"
#include "stdio.h"

#define  maxlen 20  //矩阵维度
//初始化矩阵
void init_mat(int (*a)[maxlen],int count)
{
    int i,j;
    for(i=0;i<count;i++)
        for(j=0;j<=i;j++)
            a[i][j]=i+j;
}

int main(int argc,char *argv[])
{
    int rank,size; //进程号，进程数
    int tag=1;
    MPI_Status status; //进程状态
    int blockLen[maxlen],indices[maxlen]; //数据块长度数组，数据块相对于(0,0)位置的位移 数组
    int a[maxlen][maxlen];//声明矩阵
    int count=15;  //count 表示数据块数
    MPI_Datatype newtype;
    int i;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    MPI_Comm_size(MPI_COMM_WORLD,&size);

    for(i=0;i<count;i++)
    {
    blockLen[i]=i+1;    //数据块的长度1,2,3......count
    indices[i]=lda*i;   //数据块相对于起始位置位移为 矩阵的维度x当前行数
    }
    MPI_Type_indexed(count,blockLen,indices,MPI_INT,&newtype);
    MPI_Type_commit(&newtype);

    if(rank==0) //发送进程
    {
        init_mat(a,count); //初始化矩阵
        MPI_Send(a,1,newtype,1,tag,MPI_COMM_WORLD);
    }
    else if(rank==1)//接收进程
    {
        MPI_Recv(a,1,newtype,0,tag,MPI_COMM_WORLD,&status);
        for(i=0;i<count;i++)
            printf("a[%d][0]:%d\n",i,a[i][0]);
    }

    MPI_Type_free(&newtype);
    MPI_Finalize();
}
```

17 题目 3 处理器阵列

给你一个矩阵 8*6，把它放到 4*3 的处理器阵列上

```c
// 将8*6的矩阵放到4*3的进程拓扑上
#include <stdio.h>
#include "mpi.h"
#define row_P 4  //row of Processors
#define col_P 3  //column of Processors
#define row_A 8  // row of Matrix A
#define col_A 6  // column of Matrix A

void rowcolcomm( int myid , MPI_Comm comm )
{
    int rowid, colid;
    int ma, ka, rowcolor, colcolor;
    int i,j;
    int A[ row_A ][ col_A ];

    MPI_Comm rowcomm, colcomm;

    rowcolor = myid / col_P;
    MPI_Comm_split(comm, rowcolor, myid, &rowcomm); // 分割行
    MPI_Comm_rank( rowcomm, &colid);

    colcolor = myid % col_P;
    MPI_Comm_split(comm, colcolor, myid, &colcomm); // 分割列
    MPI_Comm_rank( colcomm, &rowid);

    if( rowid < row_A % row_P)
    {
        ma = row_A / row_P +1 ;
    }
    else
    {
        ma = row_A / row_P;
    }

    if( colid < col_A % col_P)
    {
        ka = col_A / col_P +1;
    }
    else
    {
        ka = col_A / col_P;
    }

    printf("Process %3d  ma=%d ka=%d  Aij is ",myid,ma,ka);
    for( i = 0 ; i < ma; i++ )
        for( j = 0 ; j < ka ; j++ ){
        A[i][j] =  i*row_P + j*col_P +rowid+colid ; // 卷帘方式存储的 a(i)(j) = i + j
        printf("%3d ",A[i][j]);
        }
    printf("\n");

}

int main( int argc, char *argv[] )
{
    int rank ;
    int size ;
    MPI_Comm  mycomm ;

    MPI_Init(&argc,&argv);
    MPI_Comm_dup(MPI_COMM_WORLD, &mycomm);
    MPI_Comm_rank(mycomm, &rank);
    MPI_Comm_size(mycomm, &size);

    rowcolcomm( rank , mycomm) ;

    MPI_Finalize();
    return 0;
}
```

18 题目 4 行列编号 2

将 4*3 个处理器按行和按列划分，列出每个处理器在自己的通信组里的编号

```c
#include <stdio.h>
#include "mpi.h"

#define rowcount 4
#define colcount 3

void comm_matrix(MPI_Comm comm)
{
    int rank,size;
    int rowid, colid;
    int color;
    MPI_Comm *rowcomm, *colcomm;

    MPI_Comm_rank(comm,&rank);
    MPI_Comm_size(comm,&size);

    color=rank/colcount;
    MPI_Comm_split(comm,color,rank,rowcomm);
    MPI_Comm_rank( rowcomm, &colid);

    color=rank%colcount;
    MPI_Comm_split(comm,color,rank,colcomm);
    MPI_Comm_rank( colcomm, &rowid);

    printf("%d,%d \n",rowid,colid);
}

int main( int argc, char *argv[] )
{
    MPI_Comm  mycomm ;

    MPI_Init(&argc,&argv);
    MPI_Comm_dup(MPI_COMM_WORLD, &mycomm);

    comm_matrix(mycomm) ;

    MPI_Finalize();
    return 0;
}
```