消息传递接口MPI

陈俊清

jqchen@math.tsinghua.edu.cn

清华大学数学科学系

March 22, 2013

Part I

数据类型

数据类型

- 为什么要自定义数据类型?

MPI的消息收发函数只能处理连续存储的同一类型的数据。当需要在一个消息中发送或接收具有复杂结构的数据时， 可以通过定义数据类型(datatype)来实现。数据类型是MPI的一个重要特征，它的使用可有效减少消息传递的次数， 增大通信粒度，并且在收发消息时避免或减少数据在内存中的拷贝、复制。

Outline

1. 数据类型的定义
2. 数据类型创建函数
3. 数据类型的使用
4. 数据类型的打包与拆包

Outline

1. 数据类型的定义
2. 数据类型创建函数
3. 数据类型的使用
4. 数据类型的打包与拆包

数据类型的定义

一个MPI数据类型由两个n元序列构成，n为正整数。

- 类型序列（type signature）：

$$\text{Typesig}=\{type_0, type_1, \ldots, type_{n-1}\}.$$

- 位移序列（type displacements）：

$$\text{Typedis}=\{disp_0, disp_1, \ldots, disp_{n-1}\}.$$

位移序列中位移总是以字节为单位计算的。

数据类型的定义

- 基本数据类型：构成序列的数据类型。可以是原始数据类型，也可以是任何已经定义的数据类型。
- 复合数据类型：除原始数据类型以外的数据类型。数据类型是嵌套定义的。

数据类型的定义

- 类型图:

$$\text{Typemap:}\{(type_0, disp_0),(type_1, disp_1),\ldots,(type_{n-1}, disp_{n-1})$$

类型序列刻划了数据的类型特征，位移序列刻划了数据的位置特征。假设数据缓冲区的起始地址为$buff$,则由上述类型图定义的数据类型包含n块数据， 第i块数据的地址为： $buff + disp_i$,类型为$type_i$,i=0,1,. . . ,n-1.

数据类型的定义

原始数据类型的类型图：{(类型，0)}，例如：
MPI_INT的类型图为：{(int,0)}.
位移序列中的位移不必是单调上升的。从而说明，数据类型中的数据不要求按顺序存放。位移也可以是负的。 即数据类型中的数据可以位于缓冲区起始地址之前。

例子

假设数据类型TYPE的类型图为:

$$\{(\text{int},4),(\text{int},12),(\text{int},0)\}$$

则语句:

```
int A[100];
  ... ...
MPI_Send(A,1,TYPE,...)
```

将发送A[1],A[3],A[0].

例子

假设数据类型TYPE类型图为

$$\{(\text{int}, -4),(\text{int}, 0),(\text{int}, 4)\}$$

则语句:

```
int A[3];
... ...
MPI_Send(A+1,1,TYPE, ...);
```

将发送A[0],A[1],A[2].
参看例子:ex4-0.c

数据类型的大小

数据类型的大小（size）指该数据类型中包含的数据长度（字节数），它等于类型序列中所有基本数据类型的大小之和。 数据类型的大小就是消息传递时需要发送或接收的数据长度。假设数据类型type的类型图为：

$$\{(type_0, disp_0), (type_1, disp_1), ...(type_{n-1}, disp_{n-1})$$

则该数据类型的大小为：

$$sizeof(type) = \sum_{i=0}^{n-1} sizeof(type_i)$$

数据类型的上下界与域

- 数据类型的下界（lower bound）:数据的最小位移

$$lb(type) = \min_i\{disp_i\}$$

- 数据类型的上界（upper bound）:

$$ub(type) = \max_i\{disp_i + sizeof(type_i)\} + \epsilon$$

- 数据类型的域(跨度)（extent）：上下界之差

$$extent(type) = ub(type) - lb(type)$$

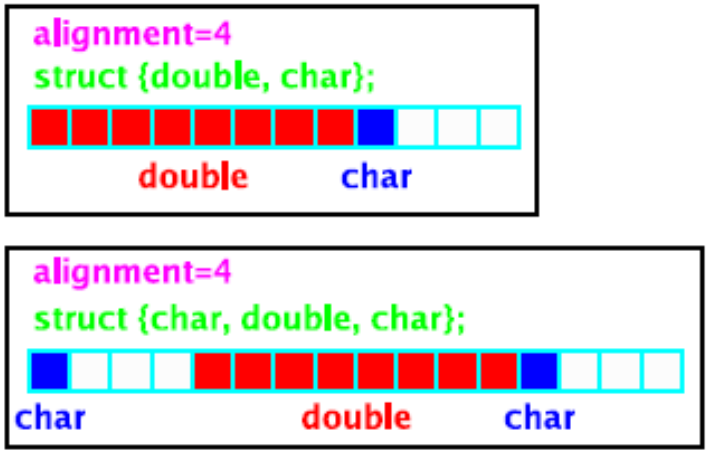
其中ϵ是地址对界修正量，是使得extent能被该数据类型的对界量整除的最小非负整数。

C语言的地址对界

数据类型的对界量:

- 原始数据类型的对界量由编译系统决定;
- 复合数据类型的对界量定义为他的所有基本数据类型的对界量的最大值。

地址对界要求指一个数据类型在内存中的（字节）地址必须是它的对界量的整数倍。 例子程序:ex4-1.c

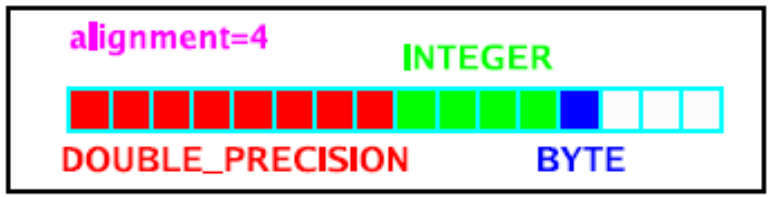


例子程序

假设MPI_DOUBLE与MPI_INT的对界量为4，MPI_BYTE的对界量为1。则类型图

$$\{(\text{MPI\_DOUBLE},0),(\text{MPI\_INT},8),(\text{MPI\_BYTE},12)\}$$

的对界量为4，下界为0，域为16,$\epsilon = 3$. 例子程序：参看ex4-2.c



MPI_LB与MPI_UB

MPI系统提供了两个特殊的数据类型MPI_LB和MPI_UB，称为伪数据类型（pseudo-datatype），他们的大小是0，并且当他们出现在数据类型的类型图中的时候，对该数据类型的实际数据内容不起任何作用。 它们的作用是让用户可以人工指定一个数据类型的上下界。 MPI规定：如果一个数据类型的基本类型中含有MPI_LB,则它的下界定义为：

$$lb(type) = \min_i\{disp_i | type_i = MPI\_LB\}.$$

类似地，如果一个数据类型的基本类型中含有MPI_UB，则它的上界定义为：

$$ub(type) = \max_i\{disp_i | type_i = MPI\_UB\}.$$

MPI_LB 与 MPI_UB的例子

类型图

```
{(MPI_LB,-4),
 (MPI_UB,20),
 (MPI_DOUBLE,0),
 (MPI_INT, 8)
 (MPI_BYTE,12)}
```

的下界为−4，上界为20，域为24。
参看例子程序：ex4-3.c

数据类型查询函数

下面四个函数分别返回指定数据类型的大小、域和上下界。

```
int MPI_Type_size(MPI_Datatype datatype, int *size);
int MPI_Type_extent(MPI_Datatype datatype,
        MPI_Aint *extent);
int MPI_Type_lb(MPI_Datatype datatype, MPI_Aint *lb);
int MPI_Type_ub(MPI_Datatype datatype, MPI_Aint *ub);
```

关于位移和数据大小的限制

在与数据类型相关的许多函数中，一些参数用到了一个数据类型：MPI_Aint（Fortan 77中对应的类型为MPI_INTEGER）。
MPI_Aint是MPI定义的一个用于与位移及数据大小有关的操作的C变量类型， 因为在64位系统中用int存储地址或数据大小可能不够，有些系统上MPI_Aint 通常被定义成long，32位系统中通常为int。用MPI_Aint来说明存储地址的 整型变量便于保证MPI代码中的地址操作在32与64位系统中的可移植性。

Outline

1. 数据类型的定义
2. 数据类型创建函数
3. 数据类型的使用
4. 数据类型的打包与拆包

数据类型创建函数

MPI_Type_contiguous:

```
int MPI_Type_contiguous(int count,
        MPI_Datatype oldtype, MPI_Datatype *newtype);
```

新数据类型newtype由count个oldtype数据类型按域连续存放构成。
例:

```
MPI_Send(buff,count,type,...)
```
与
```
MPI_Type_contiguous(count,type,&newtype);
MPI_Type_commit(newtype);
MPI_send(buff,1,newtype,...);
```
是等效的。

Contiguous的说明

令oldtype具有类型图{(double,0),(char,8)},域为16，令count＝3，则新数据类型的类型图为：

{(double,0),(char, 8),(double,16),(char,24),(double 32),(char,40)}

也就是double和char型的交替出现，位移分别是0,8,16,24,32,40。

Contiguous的说明

一般来说，如果oldtype的域为ex，类型图为

$$\{(type_0, disp_0), \ldots, (type_{n-1}, dips_{n-1})\}$$

则新数据类型的类型图为

$$\begin{aligned}
&\{(type_0, disp_0), \ldots, (type_{n-1}, disp_{n-1}), (type_0, disp_0 + ex),\\
&\ldots, (type_{n-1}, disp_{n-1} + ex), \ldots,\\
&(type_0, disp_0 + (count - 1) * ex), \ldots,\\
&(type_{n-1}, disp_{n-1} + (count - 1) * ex)
\end{aligned}$$

数据类型创建函数

MPI_Type_vector:

```
int MPI_Type_vector(int count, int blocklength,
        int stride, MPI_Datatype oldtype,
        MPI_Datatype *newtype)
```

新数据类型newtype由count个数据块构成，每个数据块由blocklength个连续存放的oldtype构成， 相邻两个数据块的位移相差stride×extent(oldtype)个字节。

MPI_Type_vector的说明

例： 假定oldtype具有类型图{(double,0),(char,8)},域为16，则MPI_Type_vector(2,3,4,oldtype,newtype)产生新数据类型newtype的类型图：

{(double,0),(char,8),(double, 16),(char,24),(double,32),(char,40)
(double,64),(char,72),(double,80),(char,88),(double,96),(char, 104)}

即：两块数据，每一块相当于oldtype被contiguous了3次，而两个数据块之间的位移相差4×16＝64个字节。

MPI_Type_vector的说明

例：假定oldtype具有类型图{(double,0),(char,8)},域为16，
则MPI_Type_vector(3,1,-2,oldtype,newtype)产生新数据类型newtype的类型图：

{(double,0),(char,8),(double, -32),(char,-24),(double,-64),(char,-56)}

数据类型创建函数

根据以上说明，可以知道

```
MPI_Type_contiguous(count,oldtype, newtype);
```

与如下函数等价:

```
MPI_Type_vector(count, 1, 1,oldtype,newtype);
```

或

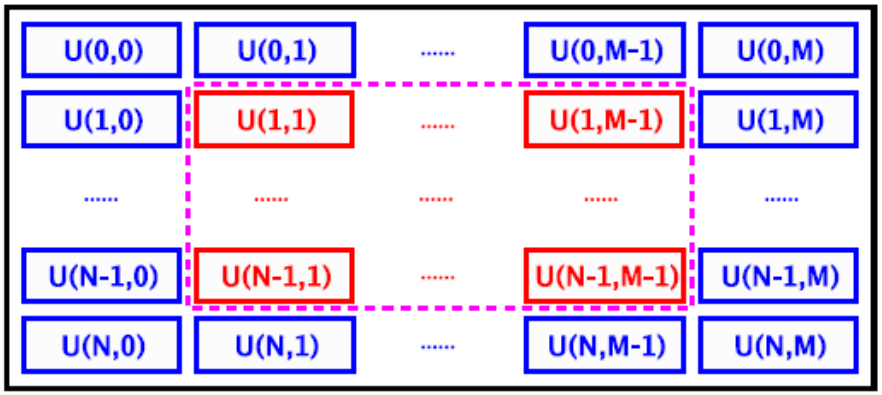
```
MPI_Type_vector(1, count, n, oldtype, newtype);
```

其中，n可为任意整数。

MPI_Type_vector的例子

```
double U[N+1][M+1];
... ...
MPI_Type_vector(N-1,M-1,M+1,
MPI_DOUBLE, &newtype);
MPI_Type_commit(&newtype);
MPI_Send(&U[1][1],1,newtype,...);
```

将发送U(1:N-1,1:M-1),参看例子程序: ex4-4.c.



思考: 如何发送U的一行、一列或对角线?

数据类型创建函数

MPI_Type_hvector:

```
int MPI_Type_hvector(int count, int blocklength,
MPI_Aint stride, MPI_Datatype oldtype,
MPI_Datatype *newtype)
```

新数据类型newtype由count个数据块构成，每个数据块由blocklength个连续存放的oldtype构成，相邻两个数据块的位移相差stride个字节。MPI_Type_hvector与MPI_Type_vector的唯一区别在于，stride在MPI_Type_vector中以oldtype的域为单位，而在MPI_Type_hvector中以字节为单位。

注: MPICH2中，MPI_Type_hvector被 MPI_Type_create_hvector所取代，该函数与MPI_Type_hvector有完全相同的参数。

MPI_Type_hvector的例：

```
double A[N][M],B[M][N];
MPI_Datatype type1, type2;
MPI_Status status;
int ex;

MPI_Type_vector(M,1,N,MPI_DOUBLE,&type1);
MPI_Type_extent(MPI_DOUBLE,ex);
MPI_Type_hvector(N,1,ex,type1,&type2);
MPI_Type_free(&type1);
MPI_Type_commit(&type2);
MPI_Sendrecv(A,N*M,MPI_DOUBLE,0,111,
        B,1,TYPE2,0,111,
        MPI_COMM_SELF,&status);
MPI_Type_free(&type2);
```

将A的转置拷贝到B中，参看程序：ex4-5.c

数据类型创建函数

MPI_Type_indexed:

```
int MPI_Type_indexed(int count,
int *array_of_blocklengths,
int *array_of_displacements,
MPI_Datatype oldtype, MPI_Datatype *newtype)
```

新数据类型newtype由count个数据块构成。第i个数据块包含array_of_blocklengths(i)个连续存放的oldtype，字节位移为array_of_displacements(i)×extent(oldtype).
MPI_Type_indexed与MPI_Type_vector的区别在于每个数据块的长度可以不同，数据块间也可以不等距。

MPI_Type_indexed的例子

令oldtype的类型图为{(double,0),(char,8)},域为16。
令B＝[3，1]，D＝[4，0]， 则调
用MPI_Type_indexed(2,B,D,oldtype,newtype)后得到新数据类型的类型图为

{(double,64),(char,72),(double,80),(char,88),(double,96),(char,104)
(double,0),(char, 8)}

MPI_Type_indexed的例子

```
double A[N][N];
int i,len[N],disp[N];
MPI_Datatype type;
... ...
len[0]=1;
disp[0]=0;
for(i=1;i<N;i++){
    len[i]=len[i-1]+1;
    disp[i]=(i-1)*N;
}
MPI_Type_indexed(N,len,disp,MPI_DOUBLE,&type);
MPI_Type_commit(&type);
MPI_Send(A,1,type,...);
```

发送A的下三角部分,参看例子程序：ex4-6.c。

数据类型创建函数

MPI_Type_hindexed:

```
int MPI_Type_hindexed(int count,
        int *array_of_blocklengths,
        MPI_Aint *array_of_displacements,
        MPI_Datatype oldtype, MPI_Datatype *newtype)
```

新数据类型newtype由count个数据块构成。第i个数据块包含array_of_blocklengths(i)个连续存放的oldtype，字节位移为array_of_displacements(i).
函数MPI_Type_hindexed与 MPI_Type_indexed的唯一区别是MPI_Type_hindexed中array_of_displacements以字节为单位。

数据类型创建函数

在MPICH2中，MPI_Type_hindexed被MPI_Type_create_hindexed所取代.

```
int MPI_Type_create_hindexed(int count,
        int array_of_blocklengths[],
        MPI_Aint array_of_displacements[],
        MPI_Datatype oldtype, MPI_Datatype *newtype)
```

数据类型创建函数

MPI_Type_create_indexed_block:

```
int MPI_Type_create_indexed_block(int count, int blocklength,
        int array_of_displacements[], MPI_Datatype oldtype,
        MPI_Datatype *newtype)
```

除了数据块的长度都一样长外，该函数与MPI_Type_indexed一样。（该函数常用于无结构网格的编程中）

数据类型创建函数

MPI_Type_struct:

```
int MPI_Type_struct(int count,
        int *array_of_blocklengths,
        MPI_Aint *array_of_displacements,
        MPI_Datatype *array_of_types,
        MPI_Datatype *newtype)
```

新数据类型newtype由count个数据块构成。第i块数据包含array_of_blocklength(i)个连续存放、类型为array_of_types(i)的数据，字节位移为array_of_displacements(i). 函数MPI_Type_struct与MPI_Type_hindexed的区别在于各数据块可由不同的数据类型构成。

数据类型创建函数

MPICH2中, MPI_Type_struct被MPI_Type_create_struct所取代:

```
int MPI_Type_create_struct(int count,
        int array_of_blocklengths[],
        MPI_Aint array_of_displacements[],
        MPI_Datatype array_of_types[],
        MPI_Datatype *newtype)
```

数据类型创建函数

例：令type1具有类型图{(double, 0),(char, 8)},域为16，
令B=(2,1,3),D=(0,16,26),T=(MPI_FLOAT, type1,MPI_CHAR).
则MPI_Type_struct(3,B,D,T,newtype)得到数据类型的类型图为：

{(float, 0), (float,4), (double, 16),(char, 24),(char, 26),(char, 27),(char, 28)}

即：从位移0开始连着2个float; 从16开始一个type1，从26开始3个char。

地址函数MPI_Address

函数MPI_Address返回指定变量的绝对地址。可以用来计算变量的位移量（主要用于Fortran语言中， C语言中可以通过指针实现对地址的任何操作）。
例:

```
double A[N],B[N];
MPI_Datatype type;
MPI_Aint a,b;
... ...
MPI_Address(A,&a);
MPI_Address(B,&b);
MPI_Type_hvector(2,N,b-a,MPI_DOUBLE,&type);
... ...
MPI_Send(A,1,type,...)
... ...
```

同时发送两个同类型的数组A和B。

Outline

1. 数据类型的定义
2. 数据类型创建函数
3. 数据类型的使用
4. 数据类型的打包与拆包

数据类型的提交

除原始数据类型外，其它数据类型在首次用于消息传递之前必须通过MPI_Type_commit函数进行提交。

```
int MPI_Type_commit(MPI_Datatype *datatype);
```

一个数据类型在被提交后就可以和MPI的原始数据类型完全一样地在消息传递中使用。
如果一个数据类型仅仅用于创建其它数据类型的中间步骤而不直接在消息传递中使用，则不必将它提交， 一旦基于它的其它数据类型创建完毕即可立即释放。（参看例子程序:ex4-5.c）

数据类型的释放

一个非原始数据类型在不再需要时应该调用MPI_Type_free函数进行释放，以便释放它所占用的系统资源。

```
int MPI_Type_free(MPI_Datatype *datatype);
```

MPI_Type_free释放指定的数据类型。函数返回后， datatype将被置成MPI_DATATYPE_NULL。正在进行的使用该数据类型的通信将会正常完成。一个数据类型的释放对在它基础上创建的其它数据类型不产生影响。

数据类型提交和释放的例子

```
double U[N][M],V[N][M];
MPI_Aint iu,iv;
MPI_Datatype ctype,ltype;
... ...
MPI_Address(U,&iu);
MPI_Address(V,&iv);
MPI_Type_vector(N,1,M,MPI_DOUBLE,&ctype);
MPI_Type_hvector(2,1,iv-iu,ctype,&ltype);
MPI_Type_free(&ctype);
MPI_Type_hvector(2,M,iv-iu,MPI_DOUBLE,&ctype);
  ... ...
```

数据类型提交和释放的例子（续）

```
//发送U（1，1:M），V（1，1:M）
MPI_Send(U,1,ctype,...)
//发送U（1:N，1），V（1:N，1）
MPI_Send(U,1,ltype,...)
//发送U（N，1:M），V（N，1:M）
MPI_Send(&U[N][1],1,ctype,...)
//发送U（1:N，M），V（1:N，M）
MPI_Send(&U[1][M],1,ltype,...)
```

MPI_Get_element

函数MPI_Get_element与MPI_Get_count类似，但它返回的是消息中所包含的MPI原始数据类型的个数。 MPI_Get_element返回的count值如果不等于MPI_UNDEFINED的话，则必然是MPI_Get_count返回的count值的倍数。

```
int MPI_Get_elements(MPI_Status *status,
        MPI_Datatype datatype, int *count)
```

参看例子程序:ex4-7.c

非原始数据类型在通信中的应用

```
MPI_Type_contiguous(count,type,newtype);
MPI_Type_commit(newtype);
MPI_Send(buf,1,newtype,dest,tag,comm);
```

假定一个发送操作MPI_Send(buf,count,datatype,dest,tag,comm)被执行，datatype具有类型图:

$$\{(type_0, disp_0), (type_1, disp_1), \ldots, (type_{n-1}, disp_{n-1})\}$$

和域extent，则发送操作发送$n \cdot count$个数据，其中 第$i \cdot n + j$个数据的地址是

$$addr_{i,j} = buf + extent \cdot i + disp_j$$

类型为$type_j$,$i = 0, \ldots, count - 1$,$j = 0, \ldots, n - 1$.这些数据的顺序可以是任意的。

接收操作MPI_Recv(buf,count,datatype,src,tag,status)被执行后，会收到$n \cdot count$个数据，第$i \cdot n + j$个数据的地址为$buf + extent * i + disp_j$,类型为$type_j$

数据类型匹配

类型匹配：根据数据类型的类型序列，即基本数据类型来定义，而不是依据数据类型定义的其它方面， 比如位移序列或是中间数据类型。例如：

```
... ...
MPI_Type_contiguous(2,MPI_DOUBLE, type2)
MPI_Type_contiguous(4,MPI_DOUBLE, type4)
MPI_Type_contiguous(2,type2,type22);
... ...
MPI_Send(a,4,MPI_DOUBLE,...)
MPI_Send(a,2,type2,...)
MPI_Send(a,1,type22,...)
MPI_Send(a,1,type4,...)
```

数据类型匹配（续）

```
...
MPI_Recv(a,4,MPI_DOUBLE,...)
MPI_Recv(a,2,type2,...)
MPI_Recv(a,1,type22,...)
MPI_Recv(a,1,type4,...)
```

其中任意一个发送都与所有的接收匹配。

Outline

1. 数据类型的定义
2. 数据类型创建函数
3. 数据类型的使用
4. 数据类型的打包与拆包

数据的打包

在MPI中，通过使用特殊数据类型MPI_PACKED，用户可以将不同的数据进行打包后再一次发送出去， 接收方在收到消息后载进行拆包。
数据打包:

```
int MPI_Pack(void *inbuf, int incount,
        MPI_Datatype datatype, void *outbuf,
        int outsize, int *position, MPI_Comm comm)
```

该函数将缓冲区inbuf中的incount个类型为datatype的数据进行打包。outsize给出的是outbuf的总长度 （字节数，供函数检查打包缓冲区是否越界用）。comm是发送打包数据将使用的通信器。
position是打包缓冲区中的位移，第一次调用MPI_Pack前用户程序应将position设为0， 随后MPI_Pack将自动修改它，使得它总是指向打包缓冲区中尚未使用部分的起始位置。 每次调用MPI_Pack后的position实际上就是已打包数据的总长度。

数据类型的拆包

```
int MPI_Unpack(void *inbuf, int insize, int *position,
        void *outbuf, int outcount,
        MPI_Datatype datatype, MPI_Comm comm)
```

MPI_Unpack进行数据拆包操作，它正好是MPI_Pack的逆操作：它从inbuf中拆包outcount个类型为datatype的数据到outbuf中。 函数中各参数的含义语MPI_Pack类似，只是这里的inbuf和insize对应于MPI_Pack的outbuf和outsize，而outbuf和outcount则对应于MPI_Pack的inbuf与incount。

得到打包后的数据大小

由于MPI打包时会在用户数据中加入一些附加信息，因此打包后的数据大小不等于用户数据的大小。 函数MPI_Pack_size返回一个数据打包后的大小，可以用来计算所需的打包缓冲区长度。

```
int MPI_Pack_size(int incount, MPI_Datatype datatype,
MPI_Comm comm, int *size)
```

size返回incount个类型为datatype的数据为了在通信器comm中进行发送或接收而打包后的数据长度.

数据打包发送的例子

参看例子程序:ex4-8.c

作业

- 修改例子程序ex4-5.c，在发送方利用创建数据类型，使得实现与ex4-5.c同样的转置效果。
- 根据本次课中的内容，写一段程序发送一个矩阵的一行（一列）或对角线。