

# Paralelni sistemi: MPI-Komunikatori i virtuelne topologije



Prof. dr Natalija Stojanović

- Napisati MPI program koji realizuje množenje matrica A i B reda n, čime se dobija rezultujuća matrica C=A\*B. Množenje se obavlja tako što master proces (sa identifikatorom 0) šalje svakom procesu radniku jednu kolonu matrice A i jednu vrstu matrice B. Master proces ne učestvuje u izračunavanju. Štampati dobijenu matricu.

$$m=2 \ n=3 \ k=4$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} * \begin{bmatrix} b_{00} & b_{01} & b_{02} & b_{03} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ b_{20} & b_{21} & b_{22} & b_{23} \end{bmatrix} =$$

$$\begin{bmatrix} a_{00}b_{00} + a_{01}b_{10} + a_{02}b_{20} & a_{00}b_{01} + a_{01}b_{11} + a_{02}b_{21} & \dots & \dots \\ a_{10}b_{00} + a_{11}b_{10} + a_{12}b_{20} & a_{10}b_{01} + a_{11}b_{11} + a_{12}b_{21} & \dots & \dots \end{bmatrix}$$

- Napisati MPI program koji realizuje množenje matrica A i B reda n, čime se dobija rezultujuća matrica C=A\*B. Množenje se obavlja tako što master proces (sa identifikatorom 0) šalje svakom procesu radniku jednu kolonu matrice A i jednu vrstu matrice B. Master proces ne učestvuje u izračunavanju. Štampati dobijenu matricu.

```
#include <stdio.h>
#include <mpi.h>
#define n 3
void main(int argc, char* argv[])
{
    int rank, size, m_rank;
    int a[n][n], b[n][n], c[n][n], vrsta[n], kolona[n], tmp[n][n], rez[n][n];
    int root=0;
    int no[1]={0};
    int br=1;
    int i, j;
    MPI_Status status;
    MPI_Datatype vector;
    MPI_Group mat, world;
    MPI_Comm com;
```

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

MPI_Type_vector(n,1,n,MPI_INT,&vector);
MPI_Type_commit(&vector);

MPI_Comm_group(MPI_COMM_WORLD,&world);

MPI_Group_excl(world,br,no,&mat);
MPI_Comm_create(MPI_COMM_WORLD,mat,&com);
MPI_Group_rank(mat,&m_rank);
```

```
if (rank == root)
{
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            a[i][j] = 3*i+j+1;
            b[i][j] = i+1;
        }
    }
}
```

```

for (i= 0;i < n ; i++)
{
    MPI_Send(&a[0][i],1,vector,i+1,33,MPI_COMM_WORLD);//svakom procesu
odgovarajucu kolonu

    MPI_Send(&b[i][0],n,MPI_INT,i+1,32,MPI_COMM_WORLD);//svakom procesu
odgovarajucu vrstu
}
else
{
    MPI_Recv(&kolona[0],n,MPI_INT,root,32,MPI_COMM_WORLD,&status);
    MPI_Recv(&vrsta[0],n,MPI_INT,root,33,MPI_COMM_WORLD,&status);

    for (int i= 0;i<n; i++)
    {
        for (int j= 0;j<n; j++)

            tmp[i][j] = kolona[i]*vrsta[j];
    }

    MPI_Reduce(&tmp,&rez,n*n,MPI_INT,MPI_SUM,0,com);
}

```

```
if (m_rank == 0)
{
    for (i = 0; i< n; i++)
    {
        for (j = 0; j< n; j++)
        {
            printf("%d ",rez[i][j]);
        }
        printf("\n");
    }
}

MPI_Finalize();

}
```

Napisati MPI program koji realizuje množenje matrice  $A_{nxn}$  i vektora  $b_n$ , čime se dobija rezultujući vektor  $c_n$ . Matrica A i vektor b se inicijalizuju u master procesu. Broj procesa je p i uređeni su kao matrica qxq ( $q^2=p$ ). Matrica A je podeljena u podmatrice dimenzika kxk ( $k=n/q$ ) i master proces distribuira odgovarajuće blokove matrica A po procesima kao što je prikazano na Slici 1. za  $n=8$  i  $p=16$ . Vektor b se distribuira u delovima od po  $n/q$  elemenata, tako da nakon slanja procesi u prvoj koloni matrice procesa sadrže prvih  $n/q$  elemenata, u 2. koloni matrice procesa sledećih  $n/q$  elemenata itd. osnovu primljenih podataka svaki proces obavlja odgovarajuća izračunavanja i učestvuje u generisanju rezultata koji se prikazuje u master procesu. Predvideti da se slanje podmatrica matrice A svakom procesu obavlja sa po jednom naredbom MPI\_Send kojom se šalje samo 1 izvedeni tip podatka. Slanje blokova vektora b i generisanje rezultata implementirati korišćenjem grupnih operacija i funkcija za kreiranje novih komunikatora.

(0,0) (0,1)	(0,2) (0,3)	(0,4) (0,5)	(0,6) (0,7)
<b>P<sub>0</sub></b>	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>
(1,0) (1,1)	(1,2) (1,3)	(1,4) (1,5)	(1,6) (1,7)
<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>	<b>P<sub>7</sub></b>
<b>P<sub>8</sub></b>	<b>P<sub>9</sub></b>	<b>P<sub>10</sub></b>	<b>P<sub>11</sub></b>
(6,0) (6,1)	(6,2) (6,3)	(6,4) (6,5)	(6,6) (6,7)
<b>P<sub>12</sub></b>	<b>P<sub>13</sub></b>	<b>P<sub>14</sub></b>	<b>P<sub>15</sub></b>
(7,0) (7,1)	(7,2) (7,3)	(7,4) (7,5)	(7,6) (7,7)

$a_{00}$	$a_{01}$	$a_{02}$	$a_{03}$	$a_{04}$	$a_{05}$	$a_{06}$	$a_{07}$	$b_0$
$a_{10}$	$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$	$a_{15}$	$a_{16}$	$a_{17}$	$b_1$
$a_{20}$	$a_{21}$	...	...	...	...	$a_{26}$	$a_{27}$	$b_2$
$a_{30}$	$a_{31}$	...	...	...	...	$a_{36}$	$a_{37}$	$b_3$
:	:					:	:	$b_4$
:	:					:	:	$b_5$
$a_{60}$	$a_{61}$	...	...	...	...	$a_{66}$	$a_{67}$	$b_6$
$a_{70}$	$a_{71}$	...	...	...	...	$a_{76}$	$a_{77}$	$b_7$



-P0, P4,P8,P12



-P1, P5,P9,P13



-P2, P6,P10,P14



-P3, P7,P11,P115

```
#define n 6
void main(int argc, char *argv[])
{
    int irow, jcol, p,i,j,k,q,l,y=0;
    MPI_Comm row_comm, col_comm, com;
    int rank, row_id, col_id;
    static int a[n][n],b[n],c[n];
    MPI_Datatype vrblok;
    MPI_Status status;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    q=(int)sqrt((double)p);
    k=n/q;
    int* local_a=(int*)calloc(k*k,sizeof(int));
    int* local_b=(int*)calloc(k,sizeof(int));
    MPI_Type_vector(k,k,n,MPI_INT,&vrblok);
    MPI_Type_commit(&vrblok);
```

```
if (rank == 0)
{
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            a[i][j] = i+j;
        }
    }
}
```

```
for(i = 0; i < n; i++)
```

```
    b[i] = 1;
```

```
}
```

```
if (rank==0)
{
    for (i = 0; i< k; i++)
        for (j = 0; j< k; j++)
            local_a[y++] = a[i][j];
    l=1;
    for (j=0;j<q;j++)
        for (i=0;i<q;i++)
            if ((i+j)!=0)
                MPI_Send(&a[j*k][k*i],1,vrblok,l,2,MPI_COMM_WORLD);
            l++;
}
else
    MPI_Recv(local_a,k*k,MPI_INT,0,2,MPI_COMM_WORLD, &status);
```

<b>P<sub>0</sub></b>	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>	row_comm
<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>	<b>P<sub>7</sub></b>	row_comm
<b>P<sub>8</sub></b>	<b>P<sub>9</sub></b>	<b>P<sub>10</sub></b>	<b>P<sub>11</sub></b>	row_comm
<b>P<sub>12</sub></b>	<b>P<sub>13</sub></b>	<b>P<sub>14</sub></b>	<b>P<sub>15</sub></b>	row_comm

(0) <b>P<sub>0</sub></b>	(0) <b>P<sub>1</sub></b>	(0) <b>P<sub>2</sub></b>	(0) <b>P<sub>3</sub></b>	(0)-col_id
(1) <b>P<sub>4</sub></b>	(1) <b>P<sub>5</sub></b>	(1) <b>P<sub>6</sub></b>	(1) <b>P<sub>7</sub></b>	
(2) <b>P<sub>8</sub></b>	(2) <b>P<sub>9</sub></b>	(2) <b>P<sub>10</sub></b>	(2) <b>P<sub>11</sub></b>	
(3) <b>P<sub>12</sub></b>	(3) <b>P<sub>13</sub></b>	(3) <b>P<sub>14</sub></b>	(3) <b>P<sub>15</sub></b>	

col\_comm      col\_comm      col\_comm      col\_comm

```
irow = rank/q;  
jcol = rank%q;  
com =MPI_COMM_WORLD;
```

```
MPI_Comm_split(com, irow, jcol, &row_comm);  
MPI_Comm_split(com, jcol, irow, &col_comm);  
MPI_Comm_rank(row_comm, &row_id);  
MPI_Comm_rank(col_comm, &col_id);
```

```
if (col_id==0)  
    MPI_Scatter(b, k, MPI_INT, local_b,k, MPI_INT,  
0,row_comm);
```

```
MPI_Bcast(local_b,k, MPI_INT, 0, col_comm);  
int* MyResult = (int*) malloc(k * sizeof(int));  
int* Result = (int*) malloc(n * sizeof(int));  
int index = 0;
```

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} & a_{06} & a_{07} \\ a_{10} & a_{11} & a_{12} & a_{13} & \cdots & \cdots & \cdots & \cdots \\ a_{20} & a_{21} & \cdots & \cdots & \cdots & \cdots & a_{26} & a_{27} \\ a_{30} & a_{31} & \cdots & \cdots & \cdots & \cdots & a_{36} & a_{37} \\ \vdots & \vdots & & & & & \vdots & \vdots \\ \vdots & \vdots & & & & & \vdots & \vdots \\ a_{60} & a_{61} & \cdots & \cdots & \cdots & \cdots & a_{66} & a_{67} \\ a_{70} & a_{71} & \cdots & \cdots & \cdots & \cdots & a_{76} & a_{77} \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix}$$

$$\begin{array}{c}
P0 \quad P1 \quad P2 \quad P3 \\
\boxed{\phantom{0}} + \boxed{\phantom{0}} + \boxed{\phantom{0}} + \boxed{\phantom{0}} \\
P4 \quad P5 \quad P6 \quad P7 \\
\boxed{\phantom{0}} + \boxed{\phantom{0}} + \boxed{\phantom{0}} + \boxed{\phantom{0}} \\
P8 \quad P9 \quad P10 \quad P11 \\
\boxed{\phantom{0}} + \boxed{\phantom{0}} + \boxed{\phantom{0}} + \boxed{\phantom{0}} \\
P12 \quad P13 \quad P14 \quad P15
\end{array}$$

```
for(i=0; i < k; i++){
    MyResult[i]=0;
    for(j=0;j<k; j++)
        MyResult[i] += local_a[index++] * local_b[j];
}
MPI_Gather (MyResult,k, MPI_INT, Result, k, MPI_INT, 0, col_comm);
if (col_id==0)
    MPI_Reduce(Result,c,n,MPI_INT, MPI_SUM,0, row_comm);

if (rank==0)
    for (i = 0; i< n; i++)
    {
        printf("c[%d]=%d ",i,c[i]);
    }
MPI_Finalize();
}
```

# Komunikatori i topologije

▪ Garantuju sigurnu komunikaciju – definišu skup procesa kojima je dozvoljeno da međusobno komuniciraju

▪ Podela:

- *Intrakomunikatori* – za operacije između procesa unutar jedne grupe

atributi:

- \* grupa procesa
- \* topologija

- *Interkomunikatori* – za operacije između različitih grupa procesa

atributi:

- \* dve disjunktne grupe procesa

# Virtuelne topologije (1)

- Opcioni atribut koji se može pridružiti intrakomunikatoru
  - Može obezbediti zgodan mehanizam imenovanja procesa u grupi (komunikatoru) i dodatno može pomoći runtime sistemu u mapiranju procesa na hardver (procesore)

# Virtuelne topologije (2)

- **Grupa procesa u MPI je skup od p procesa**

- Svakom procesu je dodeljen rank od 0 do  $p-1$ .
- U velikom broju paralelnih aplikacija ovakva dodela rankova realno ne oslikava logički obrazac komunikacije koji je obično određen geometrijom problema koji rešavamo

# Virtuelne topologije (3)

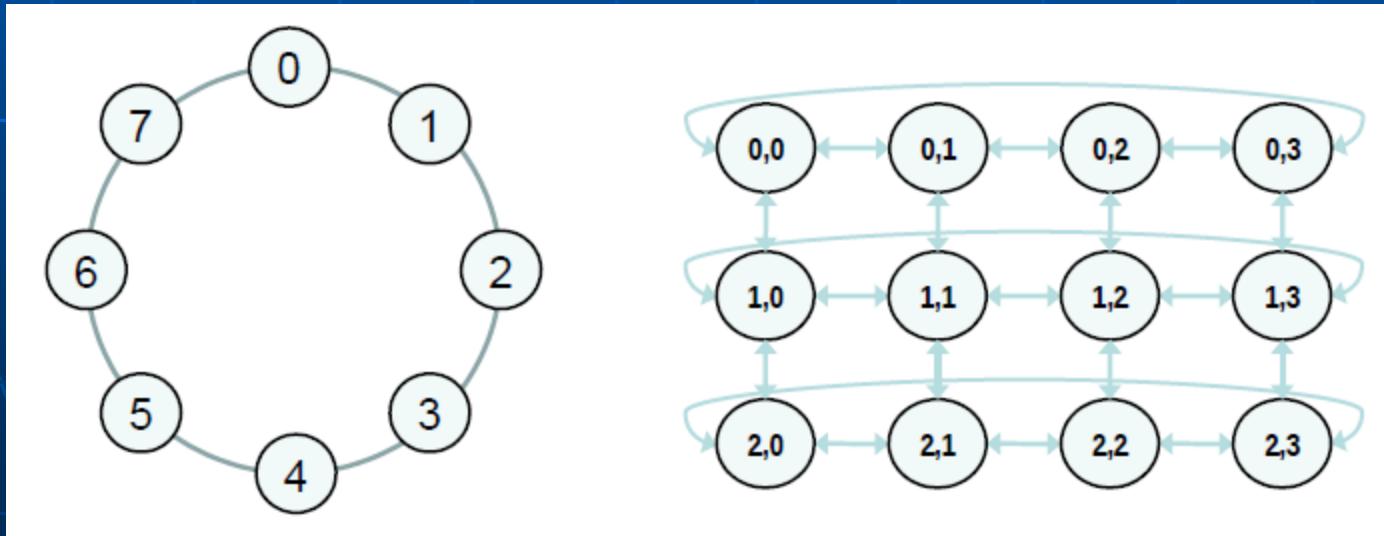
- Virtuelna topologija predstavlja logičko uređenje procesa u topologije tipa 1D, 2D ili 3D grid; još generalnije može biti opisano grafom
- Virtuelna topologija može biti iskorišćena u dodeljivanju procesa procesorima, ako to pomaže poboljšanju performansi prilikom komunikacije

# Identifikator procesa

- U MPI postoji dve različite vrste virtuelnih topologija. To su:

1. Cartesian topologija, i
2. graf topologija.

- Ovde ćemo se baviti Cartesian topologijama.



# Konstruktor Cartesian topologije

```
int MPI_Cart_create(MPI_Comm old_comm, int ndims, int  
*dim_size, int *periods, int reorder, MPI_Comm *comm_cart)
```

Argument	Tip argumenta	Značenje
comm_old	IN	Ulagni komunikator
ndims	IN	Broj dimenzija u rešetki
dims	IN	Integer polje veličine ndims koje određuje broj procesa u svakoj dimenziji
periods	IN	Lokalno polje veličine ndims koje određuje da li je rešetka periodična (true) ili ne (false) u svakoj dimenziji
reorder	IN	Vrednost koja određuje hoće li identifikatori procesa u rešetki biti preuređeni (true) ili ne (false) u cilju poboljšanja performansi
comm_cart	OUT	Novi komunikator sa Cartesian topologijom

# Konstruktor Cartesian topologije

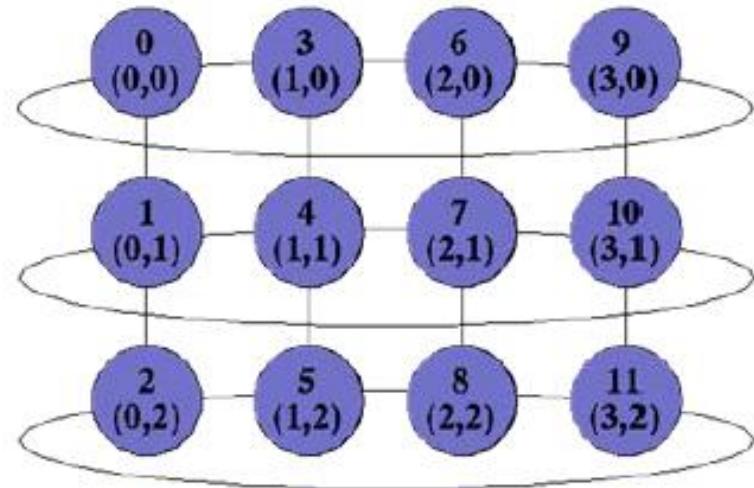
- Funkcija MPI\_Cart\_create() se može koristiti za kreiranje Cartesian struktura proizvoljnog broja dimenzija. Za svaki koordinatni pravac se navodi da li je struktura periodična ili ne.
- Za jednodimenzionalnu topologiju, ukoliko ona nije periodična, to je linearne struktura, a ukoliko je periodična-radi se o strukturi prstena.
- Dvodimenzionalna topologija može biti pravougaona-neperiodična, cilindrična-periodična po jednoj dimenziji, ili torus-potpuno periodična.

# Konstruktor Cartesian topologije

```
MPI_Comm vu;  
int dim[2], period[2], reorder;
```

```
dim[0]=4; dim[1]=3;  
period[0]=TRUE; period[1]=FALSE;  
reorder=TRUE;
```

```
MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &vu);
```



# Cartesian topologija

```
int MPI_Cart_coords( MPI_Comm comm, int rank, int  
maxdims, int *coords )
```

Ova funkcija vrši preslikavanje ranka procesa u koordinate procesa u Cartesian topologiji

Argument	Tip argumenta	Značenje
comm	IN	Komunikator Cartesian strukture
rank	IN	Identifikator procesa u grupi koja odgovara komunikatoru comm
maxdims	IN	Dužina coords vektora
coords	OUT	Integer polje koje sadrži koordinate datog procesa

# Cartesian topologija

```
int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)
```

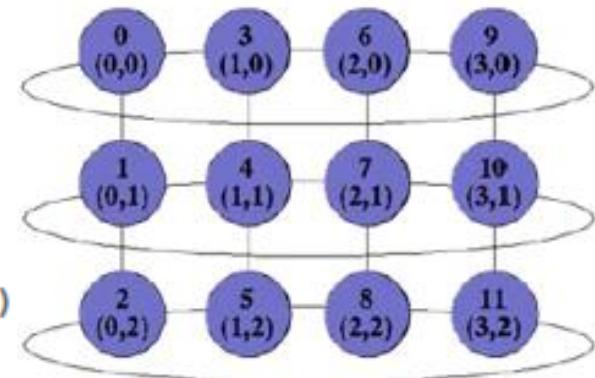
Ova funkcija vrši preslikavanje koordinate procesa u topologiji u rank procesa

Argument	Tip argumenta	Značenje
comm	IN	Komunikator Cartesian strukture
coords	IN	Integer polje veličine ndims koje određuje koordinate procesa u Cartesian strukturi
rank	OUT	Identifikator datog procesa

# Cartesian topologija

```
#include<mpi.h>
/* Run with 12 processes */
int main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2], period[2], reorder;
    int coord[2], id;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &vu)
    if(rank==5) {
        MPI_Cart_coords(vu, rank, 2, coord);
        printf("P:%d My coordinates are %d %d\n", rank, coord[0], coord[1]);
    }
    if(rank==0) {
        coord[0]=3; coord[1]=1;
        MPI_Cart_rank(vu, coord, &id);
        printf("The processor at position (%d, %d) has rank %d\n", coord[0], coord[1], id);
    }
    MPI_Finalize();
}
```

Program output  
The processor at position (3,1) has rank 10  
P:5 My coordinates are 1 2



## Cartesian topologija

```
int MPI_Cart_shift(MPI_Comm comm, int direction, int disp,  
int *rank_source, int *rank_dest).
```

Ova funkcija izračunava rankove susednih procesa u Cart. topologiji za proces koji je poziva. Ovi rankovi će biti iskorišćeni u pomeranju prilikom poziva funkcija za komunikaciju koji se dešavaju nakon poziva ove funkcije. Ako za neki proces sused nije definisan onda je povratna vrednost MPI\_PROC\_NULL. (vrednost zavisna od implementacije, uglavnom -1)

Argument	Tip argumenta	Značenje
comm	IN	Komunikator Cartesian strukture
direction	IN	Dimenzija duž koje se pomeranje obaviti nakon poziva ove funkcije
disp	IN	Korak, može biti + ili -; ako je >0, pomeranje se obavlja unapred, ako je <0, pomeranje se obavlja unazad
rank_source	OUT	Identifikator izvornog procesa(od koga će proces primati podatke)
rank_dest	OUT	Identifikator odredišnog procesa(kome će proces slati podatke)

```

#include<mpi.h>
#define TRUE 1
#define FALSE 0
int main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2], period[2], reorder;
    int up, down, right, left;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank)
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &vu);
    if(rank==9) {
        MPI_Cart_shift(vu,0,1,&left,&right);
        MPI_Cart_shift(vu,1,1,&up,&down);
        printf("P:%d My neighbors are r: %d d:%d 1:%d u:%d\n", rank, right, down, left, up);
    }
    MPI_Finalize();
}

```

The diagram illustrates a 2D grid of 12 MPI processes (ranks 0-11) arranged in 3 rows (dim 1) and 4 columns (dim 0). The ranks are represented as green circles with their corresponding coordinates in parentheses. The grid structure is as follows:

- Row 0 (dim 1 = 0):** Ranks 0 (0,0), 3 (1,0), 6 (2,0), and 9 (3,0).
- Row 1 (dim 1 = 1):** Ranks 1 (0,1), 4 (1,1), 7 (2,1), and 10 (3,1).
- Row 2 (dim 1 = 2):** Ranks 2 (0,2), 5 (1,2), 8 (2,2), and 11 (3,2).

Neighborhood connections are indicated by lines between adjacent ranks. Each rank has specific neighbors based on its position in the grid.

```
#include<mpi.h>
#define TRUE 1
#define FALSE 0
int main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2],period[2],reorder;
    int up,down,right,left;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &vu);
    if(rank==9) {
        MPI_Cart_shift(vu,0,1,&left,&right);
        MPI_Cart_shift(vu,1,1,&up,&down);
        printf("P:%d My neighbors are r: %d d:%d l:%d u:%d\n",rank,right,down,left,up);
    }
    MPI_Finalize();
}
```

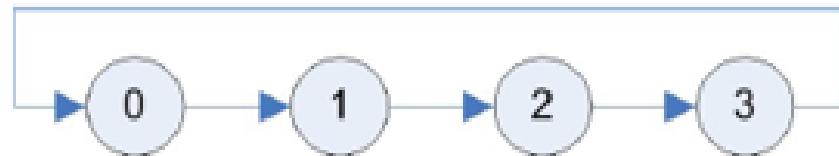
Program Output

P:9 my neighbors are r:0 d:10 l:6 u:-1

# Periodična 1D Cartesian topologija

```
#include "mpi.h"
#include <stdio.h>
#define ndims 1

void main(int argc,char *argv[])
{
int size, rank, source, dest;
int dims[ndims], periods[ndims];
MPI_Comm comm1D;
MPI_Init( &argc, &argv );
MPI_Comm_size( MPI_COMM_WORLD, &size );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
/*****************************************/
/* Create periodic shift */
/*****************************************/
dims[0] = size; /* processor dimensions */
periods[0] = 1; /* periodic shift is .true. */
MPI_Dims_create(size, ndims, dims);
if( rank == 0 )
printf("PW[%d]/[%d%]: NDims=%d, PEdims = [%d] \n",rank,size,ndims,dims[0]);
/* create cartesian mapping */
MPI_Cart_create( MPI_COMM_WORLD, ndims, dims, periods, 0, &comm1D );
MPI_Cart_shift( comm1D, 0, 1, &source, &dest );
```



# Periodična 1D Cartesian topologija

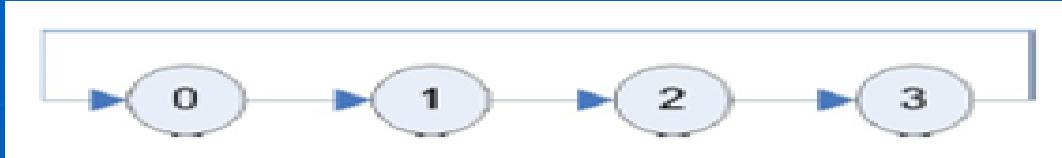
```
printf( "P[%d]: peri: shift 1: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
MPI_Cart_shift( comm1D, 0, 0, &source, &dest );
printf( "P[%d]: peri: shift 0: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
MPI_Cart_shift( comm1D, 0, -1, &source, &dest );
printf( "P[%d]: peri: shift -1: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
```

# Neperiodična 1D Cartesian topologija

```
*****
/* Create non-periodic shift */
*****
if(rank == 0 ) printf("non-periodic next \n");
MPI_Comm_free( &comm1D );
periods[0] = 0;
MPI_Cart_create( MPI_COMM_WORLD, 1, dims, periods, 0, &comm1D );
MPI_Cart_shift( comm1D, 0, 1, &source, &dest );
printf( "P[%d]: nonp: shift 1: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
MPI_Cart_shift( comm1D, 0, 0, &source, &dest );
printf( "P[%d]: nonp: shift 0: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
MPI_Cart_shift( comm1D, 0, -1, &source, &dest );
printf( "P[%d]: nonp: shift -1: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);

MPI_Comm_free( &comm1D );
MPI_Finalize();
}
```

# Periodična 1D Cartesian topologija-izlaz



```
PW[0]/[4]: NDims=1, PEdims = [4]
P[0]: peri: shift 1: src[3] P[0] dest[1]
P[0]: peri: shift 0: src[0] P[0] dest[0]
P[0]: peri: shift -1: src[1] P[0] dest[3]
P[1]: peri: shift 1: src[0] P[1] dest[2]
P[1]: peri: shift 0: src[1] P[1] dest[1]
P[1]: peri: shift -1: src[2] P[1] dest[0]
P[2]: peri: shift 1: src[1] P[2] dest[3]
P[2]: peri: shift 0: src[2] P[2] dest[2]
P[2]: peri: shift -1: src[3] P[2] dest[1]
P[3]: peri: shift 1: src[2] P[3] dest[0]
P[3]: peri: shift 0: src[3] P[3] dest[3]
P[3]: peri: shift -1: src[0] P[3] dest[2]
```

# Neperiodična 1D Cartesian topologija-izlaz

```
PW[0]/[4]: NDims=1, PEdims = [4]
non-periodic next
P[0]: nonp: shift 1: src[-1] P[0] dest[1]
P[0]: nonp: shift 0: src[0] P[0] dest[0]
P[0]: nonp: shift -1: src[1] P[0] dest[-1]
P[2]: nonp: shift 1: src[1] P[2] dest[3]
P[2]: nonp: shift 0: src[2] P[2] dest[2]
P[2]: nonp: shift -1: src[3] P[2] dest[1]
P[1]: nonp: shift 1: src[0] P[1] dest[2]
P[1]: nonp: shift 0: src[1] P[1] dest[1]
P[1]: nonp: shift -1: src[2] P[1] dest[0]
P[3]: nonp: shift 1: src[2] P[3] dest[-1]
P[3]: nonp: shift 0: src[3] P[3] dest[3]
P[3]: nonp: shift -1: src[-1] P[3] dest[2]
```

# MPI\_Sendrecv

Kombinuje blokirajuće Send i Recv u jednu operaciju koja se odvija bez deadloka. Zgodno za rešavanje problema, gde svaki proces i prima i šalje podatke. Proces koji je izvršava šalje max 1 poruku i prima max 1 poruku. Dest i source mogu biti različiti, ali i isti, po potrebi.

```
int MPI_Sendrecv(  
    void *sendbuf,  
    int sendcount,  
    MPI_Datatype sendtype,  
    int dest,  
    int sendtag,  
    void *recvbuf,  
    int recvcount,  
    MPI_Datatype recvtype,  
    int source,  
    int recvtag,  
    MPI_Comm comm,  
    MPI_Status *status)
```

Parameters for send

Parameters for receive

Same communicator

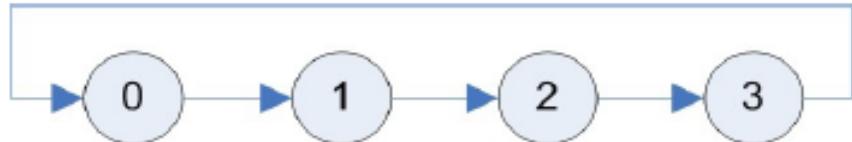
# Sendrecv sa 1D Cartesian topologijom

```
...
int dim[1],period[1];
dim[0] = nprocs;
period[0] = 1;
MPI_Comm ring_comm;

MPI_Cart_create(MPI_COMM_WORLD, 1, dim, period, 0, &ring_comm);

int source, dest;
MPI_Cart_shift(ring_comm, 0, 1, &source, &dest);

MPI_Sendrecv(right_bounday, n, MPI_INT, dest, rtag,
             left_bounday, n, MPI_INT, source, ltag,
             ring_comm, &status);
...
...
```



Ako je  $n=1$ ,  $\text{right\_boundary}[0]=\text{rank}; // 0 1 2 3$

Nakon  $\text{MPI\_Sendrecv}$ , u procesima P0-P3:

$\text{left\_boundary}[0]:3 0 1 2$

# **MPI\_Sendrecv\_replace**

Varijanta MPI\_Sendrecv koja koristi isti bafer za slanje i primanje podataka.

Primljeni podaci u buf se kopiraju na mesta, odakle su poslati podaci iz buf.

```
int MPI_Sendrecv_replace(void *buf, int count,  
                         MPI_Datatype datatype, int dest, int sendtag,  
                         int source, int recvtag, MPI_Comm comm,  
                         MPI_Status *status)
```

# 2D Cartesian topologijom

## Sendrecv\_replace

```
#define UP    0
#define DOWN  1
#define LEFT   2
#define RIGHT  3

.

.

.

MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &cartcomm);
MPI_Comm_rank(cartcomm, &rank);
MPI_Cart_coords(cartcomm, rank, 2, coords);

MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP], &nbrs[DOWN]);
MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT], &nbrs[RIGHT]);

outbuf = rank;
dest = nbrs[1];
source = nbrs[0];
MPI_Sendrecv_replace(&outbuf, 1, MPI_INT, dest, 0, source, 0, cartcomm, &st);

|.
```

# Izlaz za 16 procesa (4 procesa po dimenziji, periodična po obe dimenzije)

```
rank= 8 outbuf = 4
rank= 4 outbuf = 0
rank= 7 outbuf = 3
rank= 5 outbuf = 1
rank= 6 outbuf = 2
rank= 12 outbuf = 8
rank= 2 outbuf = 14
rank= 14 outbuf = 10
rank= 3 outbuf = 15
rank= 13 outbuf = 9
rank= 1 outbuf = 13
rank= 10 outbuf = 6
rank= 11 outbuf = 7
rank= 9 outbuf = 5
rank= 15 outbuf = 11
rank= 0 outbuf = 12
```

coord					rank				
0,0	0,1	0,2	0,3		0	1	2	3	
1,0	1,1	1,2	1,3		4	5	6	7	
2,0	2,1	2,2	2,3		8	9	10	11	
3,0	3,1	3,2	3,3		12	13	14	15	

outbuf=rank				
0	1	2	3	
4	5	6	7	
8	9	10	11	
12	13	14	15	

Outbuf nakon MPI\_Sendrecv\_replace:

12	13	14	15
0	1	2	3
4	5	6	7
8	9	10	11

Napisati deo MPI koda koji formira komunikator Cartesian topologije, koji omogućava da sadržaj promenljive  $a$ , u svakom procesu, pre i nakon poziva funkcije:  
`MPI_Sendrecv_replace(&a, 1, MPI_INT, dest, 0, source, 0, cartcomm, &st);`

izgleda kao na slici na primeru komunikatora  $3 \times 4$ .

Vrednosti  $a$  pre:

0	1	2	3
4	5	6	7
8	9	10	11

coord				rank			
0,0	0,1	0,2	0,3	0	1	2	3
1,0	1,1	1,2	1,3	4	5	6	7
2,0	2,1	2,2	2,3	8	9	10	11

Vrednosti  $a$  posle:

0	9	6	3
4	1	10	7
8	5	2	11

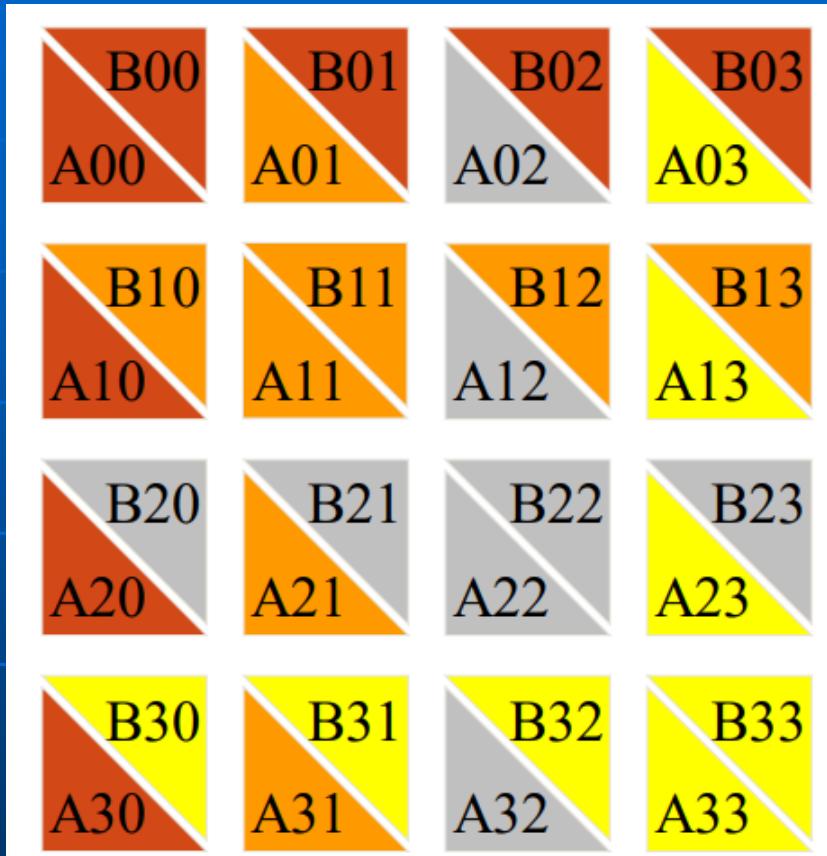
# Rešenje

```
#define SIZE 12
#define UP  0
#define DOWN 1
void main(int argc,char *argv[])
{
int numtasks, rank, source, dest, outbuf, i, tag=1, nbrs[4],
dims[2]={3,4},periods[2]={1,0}, reorder=0, coords[2];
MPI_Comm cartcomm;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, reorder, &cartcomm);
MPI_Comm_rank(cartcomm, &rank);
MPI_Cart_coords(cartcomm, rank, 2,coords);
MPI_Cart_shift(cartcomm, 0, coords[1], &nbrs[UP], &nbrs[DOWN]);
outbuf = rank;dest = nbrs[1];source = nbrs[0];
MPI_Sendrecv_replace(&outbuf, 1, MPI_INT, dest, 0, source,0,cartcomm,&st);
}
```

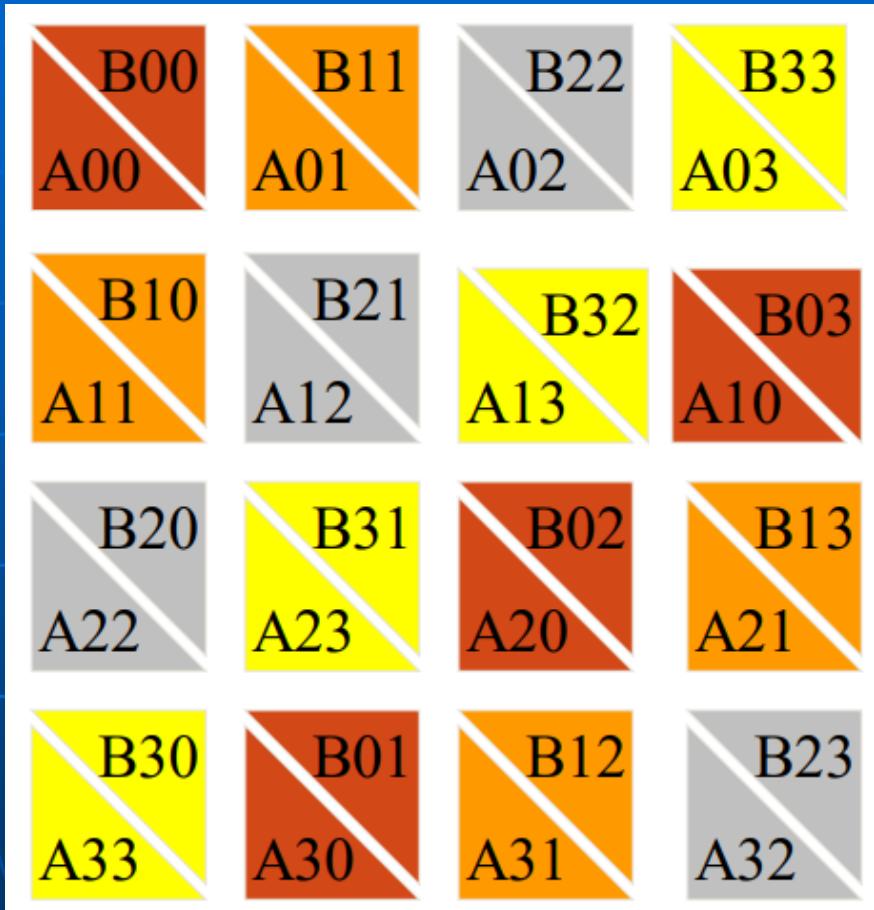
- Napisati MPI program za množenje matrica A i B, reda n. Matrice se inicijalizuju u master procesu. Broj procesa je p i uređeni su kao matrica qxq. Matrice su podeljene u podmatrice dimenzija kxk ( $k=n/q$ ) i master proces distribuiruće blokove matrica po procesima. Svaki proces obavlja odgovarajuća izračunavanja u cilju generisanja rezultata za jedan blok rezultujuće matrice C. Generisanje rezultata implementirati korišćenjem virtuelnih topologija.

<b>(0,0) (0,1)</b>	<b>(0,2) (0,3)</b>	<b>(0,4) (0,5)</b>	<b>(0,6) (0,7)</b>
<b>P<sub>0</sub></b>	<b>P<sub>1</sub></b>	<b>P<sub>2</sub></b>	<b>P<sub>3</sub></b>
<b>(1,0) (1,1)</b>	<b>(1,2) (1,3)</b>	<b>(1,4) (1,5)</b>	<b>(1,6) (1,7)</b>
<b>P<sub>4</sub></b>	<b>P<sub>5</sub></b>	<b>P<sub>6</sub></b>	<b>P<sub>7</sub></b>
<b>P<sub>8</sub></b>	<b>P<sub>9</sub></b>	<b>P<sub>10</sub></b>	<b>P<sub>11</sub></b>
<b>(6,0) (6,1)</b>	<b>(6,2) (6,3)</b>	<b>(6,4) (6,5)</b>	<b>(6,6) (6,7)</b>
<b>P<sub>12</sub></b>	<b>P<sub>13</sub></b>	<b>P<sub>14</sub></b>	<b>P<sub>15</sub></b>
<b>(7,0) (7,1)</b>	<b>(7,2) (7,3)</b>	<b>(7,4) (7,5)</b>	<b>(7,6) (7,7)</b>

Početna raspodela delova matrica A i B procesima u 2d gridu dimenzija 4\*4



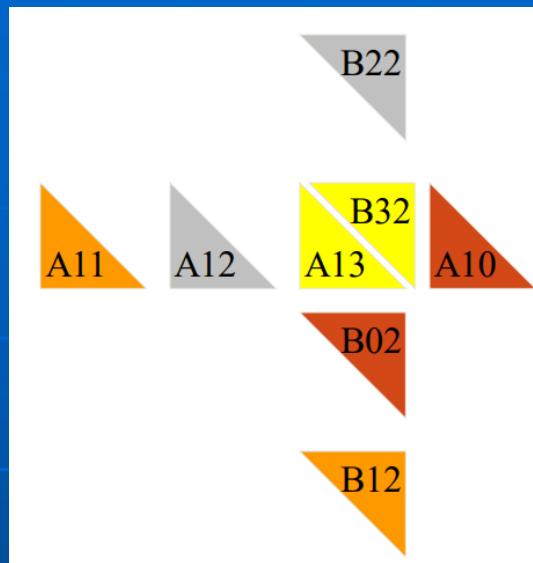
- Svaki „trougao“ je blok matrice
- Samo trouglovi iste boje na istoj poziciji treba množiti
- Koje blokove treba izmnožiti da bi proces  $P_{12}$  mogao da izgeneriše odgovarajući deo matrice C?



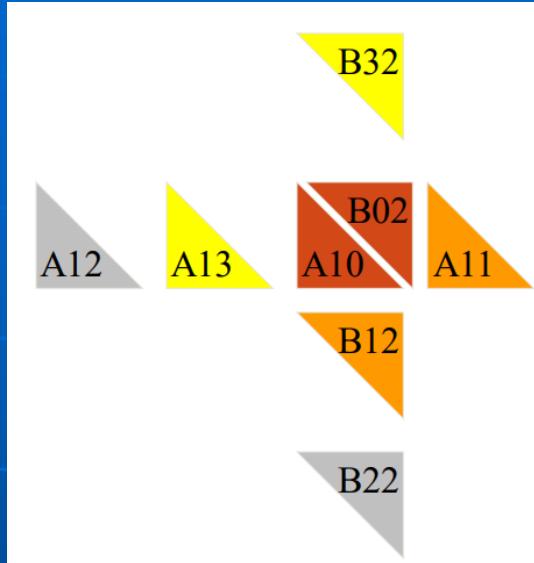
- Ciklično pomeriti blokove matrice A za i pozicija levo, gde je i indeks reda
- Ciklično pomeriti blokove matrice B za j pozicija gore, gde je j indeks kolone
- Ovim pomeranjem, dobija se prikazan raspored i svaki proces množi odgovarajuće blokove. Time proces dobija jedan deo rezultata za matricu C
- Nakon toga, treba pomeriti svaku vrstu/kolonu ciklično za jedno mesto levo/gore

Kako ovo izgleda za proces P<sub>12</sub>?

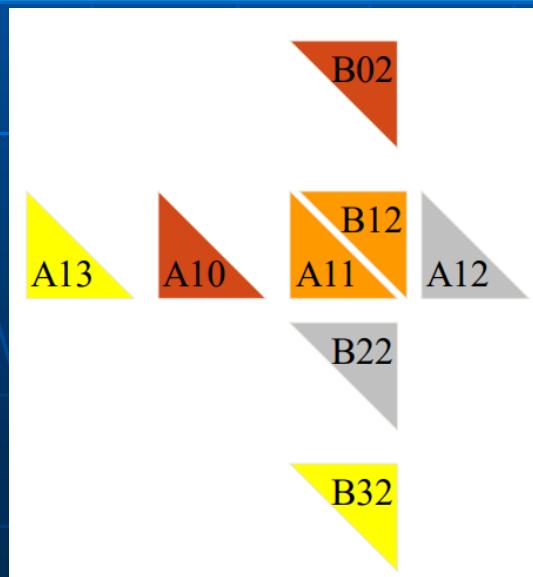
1



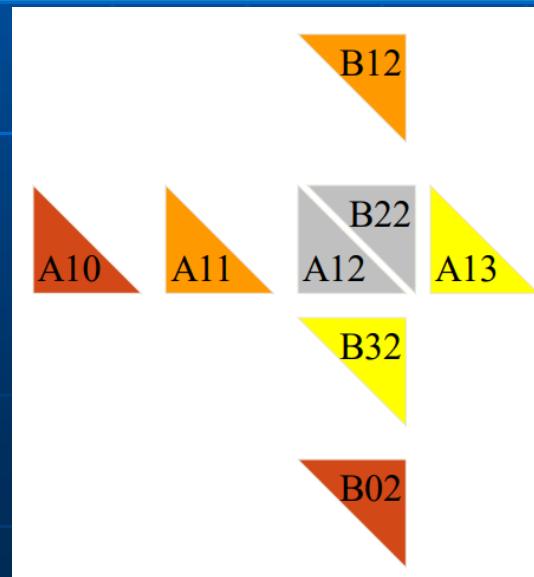
2



3



4



```
#define n 8

void main(int argc, char* argv[])
{
    int i, j, l, t, k, p, q, y=0;
    int dims[2], periods[2];
    int rank, my2drank, mycoords[2];
    int leftrank, rightrank, uprank, downrank, shiftsource, shiftdest;
    static int a[n][n], b[n][n], c[n][n];
    MPI_Comm comm_2d;
    MPI_Datatype vrblok;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    q = (int)sqrt((double)p);
    k = n / q;
    int* local_a = (int*)malloc(k * k, sizeof(int));
    int* local_b = (int*)calloc(k * k, sizeof(int));
    int* local_c = (int*)calloc(k * k, sizeof(int));
```

```
MPI_Type_vector(k, k, n, MPI_INT, &vrblok);
MPI_Type_commit(&vrblok);
```

```
dims[0] = dims[1] = q;
periods[0] = periods[1] = 1;
```

```
MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, 0, &comm_2d);
MPI_Comm_rank(comm_2d, &my2drank);
MPI_Cart_coords(comm_2d, my2drank, 2, mycoords);
```

```
if (rank == 0)
{
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
        {
            a[i][j] = i + j;
            b[i][j] = i + 2*j;
            c[i][j] = 0;
        }
}
```

```

for (i = 0; i < k; i++)
    for (j = 0; j < k; j++)
        local_a[y++] = a[i][j];

l = 1;
for (j = 0; j < q; j++)
    for (i = 0; i < q; i++)
        if ((i + j) != 0)
    {
        MPI_Send(&a[j * k][k * i], 1, vrblok, l, 2, MPI_COMM_WORLD);
        l++;
    }
for (i = 0; i < k; i++)
    for (j = 0; j < k; j++)
        local_b[y++] = b[i][j];

l = 1;
for (j = 0; j < q; j++)
    for (i = 0; i < q; i++)
        if ((i + j) != 0)
    {
        MPI_Send(&b[j * k][k * i], 1, vrblok, l, 3, MPI_COMM_WORLD);
        l++;
    }

```

```
else
{
    MPI_Recv(local_a, k * k, MPI_INT, 0, 2, MPI_COMM_WORLD, &status);
    MPI_Recv(local_b, k * k, MPI_INT, 0, 3, MPI_COMM_WORLD, &status);
}

for (i = 0; i < k; i++)
    for (j = 0; j < k; j++)
        local_c[i+j] = 0;

MPI_Cart_shift(comm_2d, 0, -1, &rightrank, &leftrank);
MPI_Cart_shift(comm_2d, 1, -1, &downrank, &uprank);

MPI_Cart_shift(comm_2d, 0, -mycoords[0], &shiftsource, &shiftdest);
MPI_Sendrecv_replace(&local_a, k * k, MPI_INT, shiftdest, 1, shiftsource,
1, comm_2d, &status);

MPI_Cart_shift(comm_2d, 1, -mycoords[1], &shiftsource, &shiftdest);
MPI_Sendrecv_replace(&local_b, k * k, MPI_INT, shiftdest, 1, shiftsource,
1, comm_2d, &status);
```

```
for (i = 0; i < dims[0]; i++)
{
    for (t = 0; i < k; i++)
        for (j = 0; j < k; j++)
            for (l = 0; l < k; k++)
                local_c[t * n + j] += local_a[t * n + l] * local_b[l * n + j];

/* Shift matrix a left by one */
MPI_Sendrecv_replace(local_a, k * k, MPI_DOUBLE, leftrank, 1,
rightrank, 1, comm_2d, &status);

/* Shift matrix b up by one */
MPI_Sendrecv_replace(local_b, k * k, MPI_DOUBLE, uprank, 1,
downrank, 1, comm_2d, &status);
}
```

```
if (rank>0)
    MPI_Send(local_c, k * k, MPI_INT, 0, 4, MPI_COMM_WORLD);
else
{
    y = 0;
    for (i = 0; i < k; i++)
        for (j = 0; j < k; j++)
            c[i][j] = local_c[y++];

    l = 1;
    for (j = 0; j < q; j++)
        for (i = 0; i < q; i++)
            if ((i + j) != 0) {
                MPI_Recv(&c[j * k][k * i], 1, vrblok, l, 4, MPI_COMM_WORLD);
                l++;
            }

    for (i = 0; i < n; i++)
        for (j = 0; j < ; j++)
            printf("%d ",c[i][j]);
}

MPI_Finalize();
}
```