- 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

$$\begin{bmatrix} a00 & a01 & a02 \\ a10 & a11 & a12 \end{bmatrix} * \begin{bmatrix} b00 & b01 & b02 & b03 \\ b10 & b11 & b12 & b13 \\ b20 & b21 & b22 & b23 \end{bmatrix} =$$

$$\begin{bmatrix} a00b00 + a01b10 + a02b20 & a00b01 + a01b11 + a02b21 & \dots & \dots \\ a10b00 + a11b10 + a12b20 & a10b01 + a11b11 + a12b21 & \dots & \dots \end{bmatrix}$$

- 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();

}
```

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)

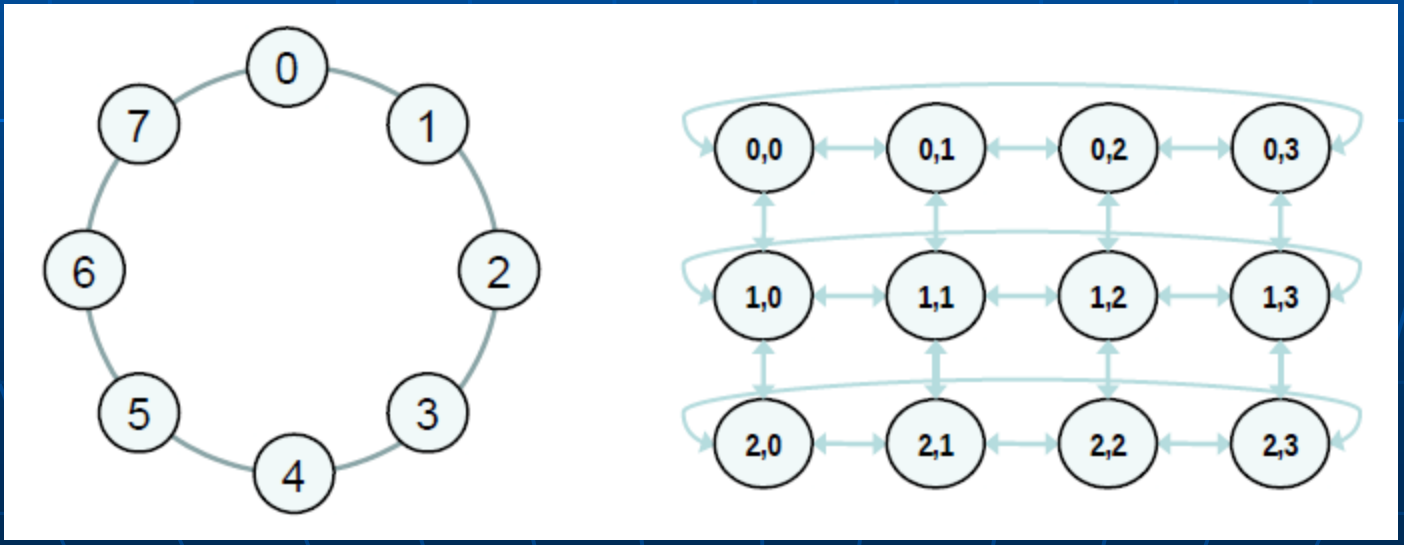
- Virtuelna topologija predstavlja logičko uređenje procesa u topologije tipa 1D, 2D ili 3D grid; još generalnije može biti opisano grafom

- Virtuelna topologija može biti iskorišćena u dodeljivanju procesa procesorima, ako to pomaže poboljšanju performansi prilikom komunikacije

Identifikator procesa

- U MPI postoje dve različite vrste virtuelnih topologija. To su:
  1. Cartesian topologija, i
  2. graf topologija.

- Ovde ćemo se baviti Cartesian topologijama.



Konstruktor Cartesian topologije

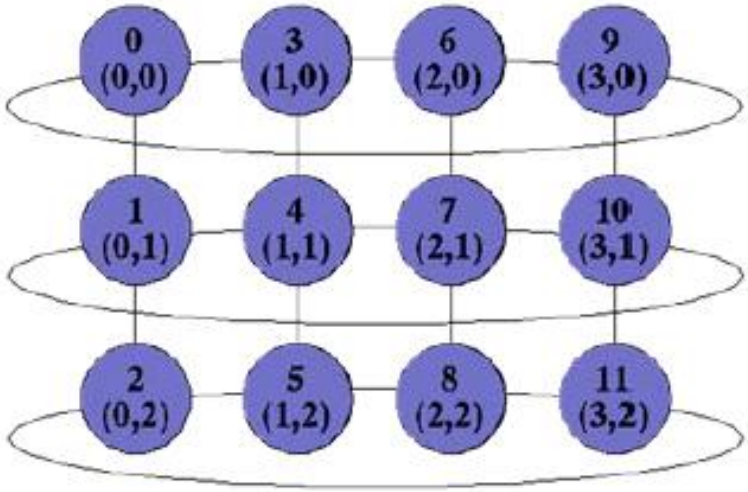
int **MPI_Cart_create**(*MPI_Comm old_comm, int ndims, int *dim_size, int *periods, int reorder, MPI_Comm *comm_cart*)

Argument	Tip argumenta	Značenje
comm_old	IN	Ulazni 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 linearna 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)

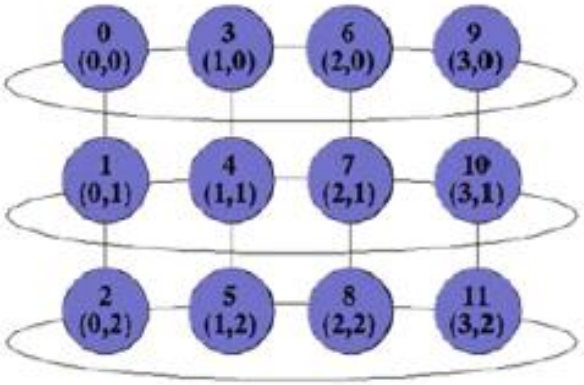
Ova funkcija vrši preslikavanje koordinate procesa u topologiji u rank procesa

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
```

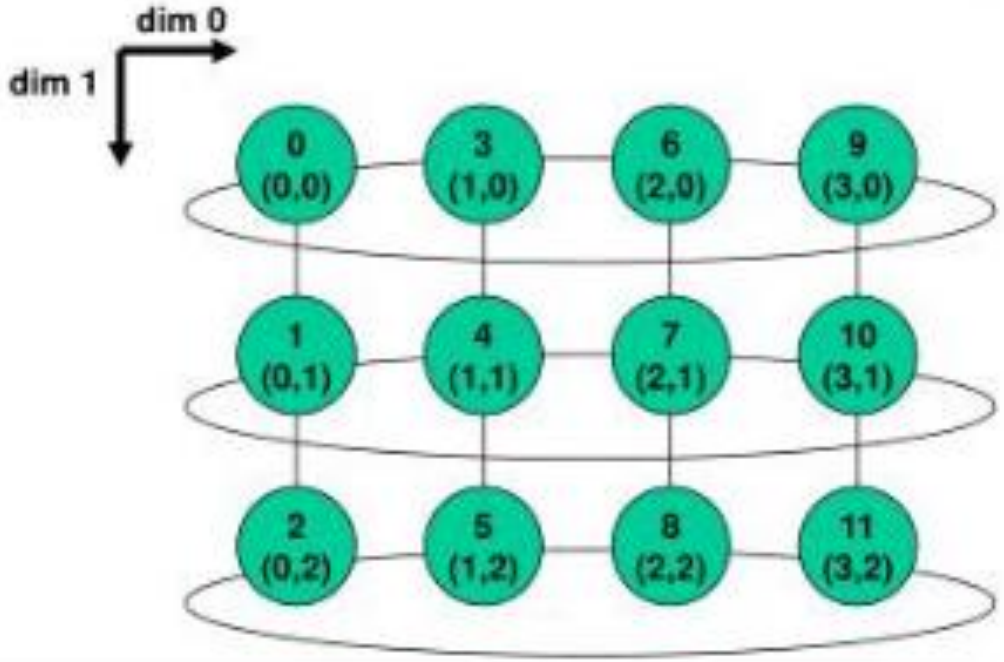


Cartesian topologija

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();
}
```



```
#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();
}
```

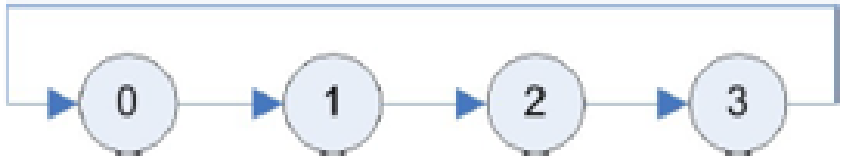
Program Output

```
P:9 my neighbors are r:0 d:10 1:6 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

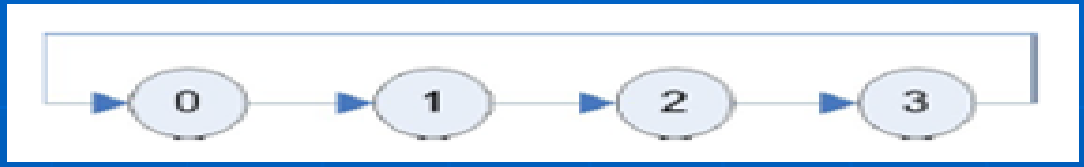
```
printf( "P[%d]: peri: shift 1: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
MPI_Cart_shift( comm1D, 0, 0, &source, &dest );
printf( "P[%d]: peri: shift 0: src[%d] P[%d] dest[%d] \n",
rank,source,rank,dest );fflush(stdout);
MPI_Cart_shift( comm1D, 0, -1, &source, &dest );
printf(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



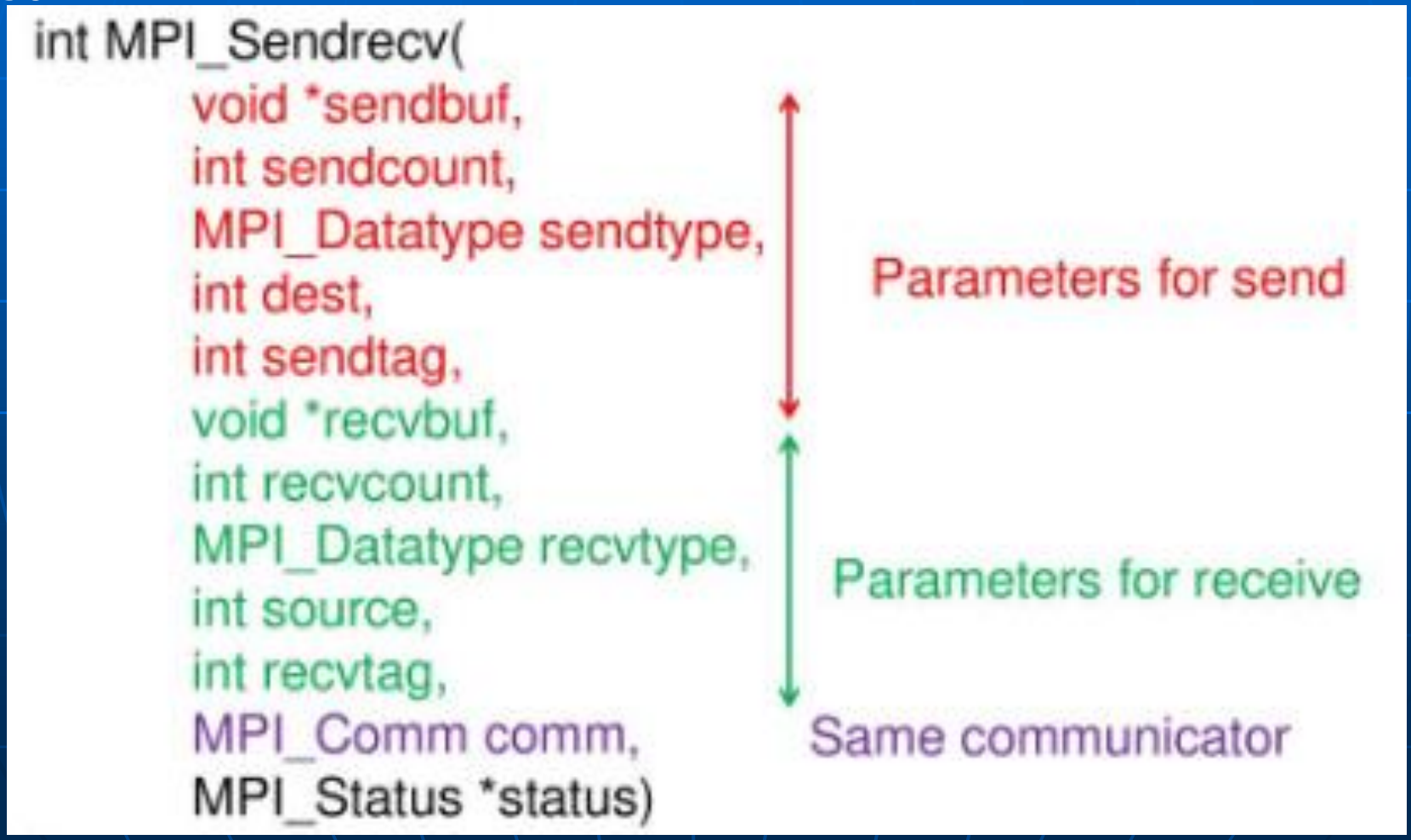

```
PW[0]/[4]: NDims=1, PEdims = [4]
P[0]: peri: shift 1: src[3] P[0] dest[1]
P[0]: peri: shift 0: src[0] P[0] dest[0]
P[0]: peri: shift -1: src[1] P[0] dest[3]
P[1]: peri: shift 1: src[0] P[1] dest[2]
P[1]: peri: shift 0: src[1] P[1] dest[1]
P[1]: peri: shift -1: src[2] P[1] dest[0]
P[2]: peri: shift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

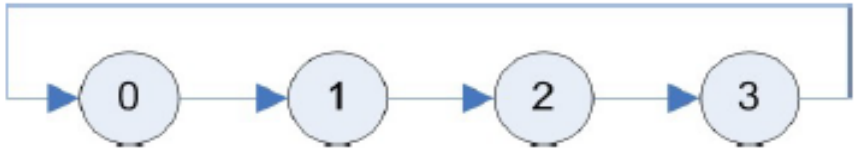
```
PW[0]/[4]: NDims=1, PEdims = [4]
non-periodic next
P[0]: nonp: shift 1: src[-1] P[0] dest[1]
P[0]: nonp: shift 0: src[0] P[0] dest[0]
P[0]: nonp: shift -1: src[1] P[0] dest[-1]
P[2]: nonp: shift 1: src[1] P[2] dest[3]
P[2]: nonp: shift 0: src[2] P[2] dest[2]
P[2]: nonp: shift -1: src[3] P[2] dest[1]
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 process i prima i šalje podatke. Proces koje je izvršava šalje max 1 poruku i prima max 1 poruku. Dest i source mogu biti različiti, ali i isti, po potrebi.



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_boundary, n, MPI_INT, source, ltag,
             ring_comm, &status);
...
```

Ako je n=1, right_boundary[0]=rank;// 0 1 2 3
Nakon MPI_Sendrecv, u procesima P0-P3:

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

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

rank

0	1	2	3
4	5	6	7
8	9	10	11
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 3x4.

Vrednosti *a* pre:

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

coord			
0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3

rank			
0	1	2	3
4	5	6	7
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 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..Na 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.