P-t-P operacije-podsetnik

```
int MPI_Send(void *buf, int count, MPI_Datatype
dtype, int dest, int tag, MPI_Comm comm);

int MPI_Recv(void *buf, int count, MPI_Datatype
dtype, int source, int tag, MPI_Comm comm,
MPI_Status *status);

int MPI_Isend(void *buf, int count, MPI_Datatype
dtype, int dest, int tag, MPI_Comm comm,
MPI_Request *request);

int MPI_Irecv(void *buf, int count, MPI_Datatype
dtype, int source, int tag, MPI_Comm comm,
MPI_Request *request);
```

Grupne operacije-podsetnik

int MPI_Reduce (void* send_buffer,void* recv_buffer, int count, MPI_Datatype datatype, MPI_Op operation, int rank, MPI_Comm comm)

int MPI_Bcast (void* buffer, int count, MPI_Datatype datatype, int rank, MPI_Comm comm)

int MPI_Scatter (void* send_buffer, int send_count, MPI_datatype send_type, void* recv_buffer, int recv_count, MPI_Datatype recv_type, int rank, MPI_Comm comm)

int MPI_Gather (void* send_buffer, int send_count, MPI_datatype send_type, void* recv_buffer, int recv_count, MPI_Datatype recv_type, int rank, MPI_Comm comm)

Izvedeni tipovi podataka

U najprostijem slučaju tip podataka koji se javlja kao argument u nekoj MPI funkciji će biti jedan od osnovnih. MPI, međutim, podržava i koncept izvedenih tipova podataka, čime je podržana manipulacija složenim strukturama kao što su delovi polja(koji s e ne nalaze na sukcesivnim memorijskim lokacijama) ili strukture koje se sastoje od kombinacija osnovnih tipova podataka.

Funkcija koja služi za formiranje tipa podataka koji se sastoji od kombinacije osnovnih tipova podataka je sledeća

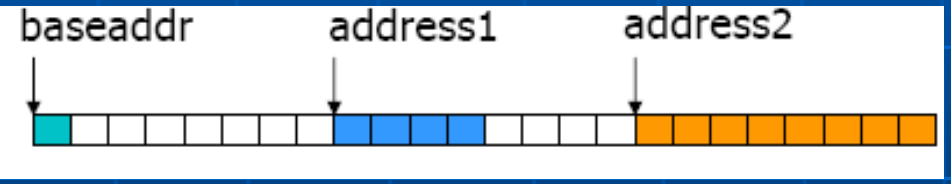
int MPI_Type_struct(int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype)

gde je count broj blokova u novom tipu
array_of_blocklengths-svaki član ovog niza je broj elemenata starog tipa u odgovarajućem bloku,tj. i-ti član je broj elemenata tipa **array_of_types[i]** u i-tom bloku

Izvedeni tipovi podataka

array_of_types- niz koji se sastoji od elemanata starog tipa koji učestvuju u kreiranju novog tipa
newtype-novi tip
array_of_diplacements -niz pomeraja svakog bloka u odnosu na početnu adresu strukture izražen u **bajtovima** koji se dobija uspomoć funkcije:

MPI_Address(void *location, MPI_Aint * address)-vraća adresu u memoriji podatka location



struct { char a; int b;double c;} value;

Za ovaj primer elementi niza array_of_diplacements su:

```
array_of_diplacements[0]=0;
array_of_diplacements[1]=address1-baseaddr;
array_of_diplacements[2]=address2-baseaddr;
```

Gde se odgovarajuće adrese dobijaju funkcijom **MPI_Address**

Izvedeni tipovi podataka

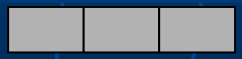
Funkcija kojom se formira novi tip podataka kopiranjem na uzastopne mem. lokacije je

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

Pr.

```
MPI_Type_contiguous(2,MPI_DOUBLE,&MPI_2D_POINT);
MPI_Type_contiguous(3,MPI_DOUBLE,&MPI_3D_POINT);
```

MPI_2D_POINT



MPI_3D_POINT

Izvedeni tipovi podataka

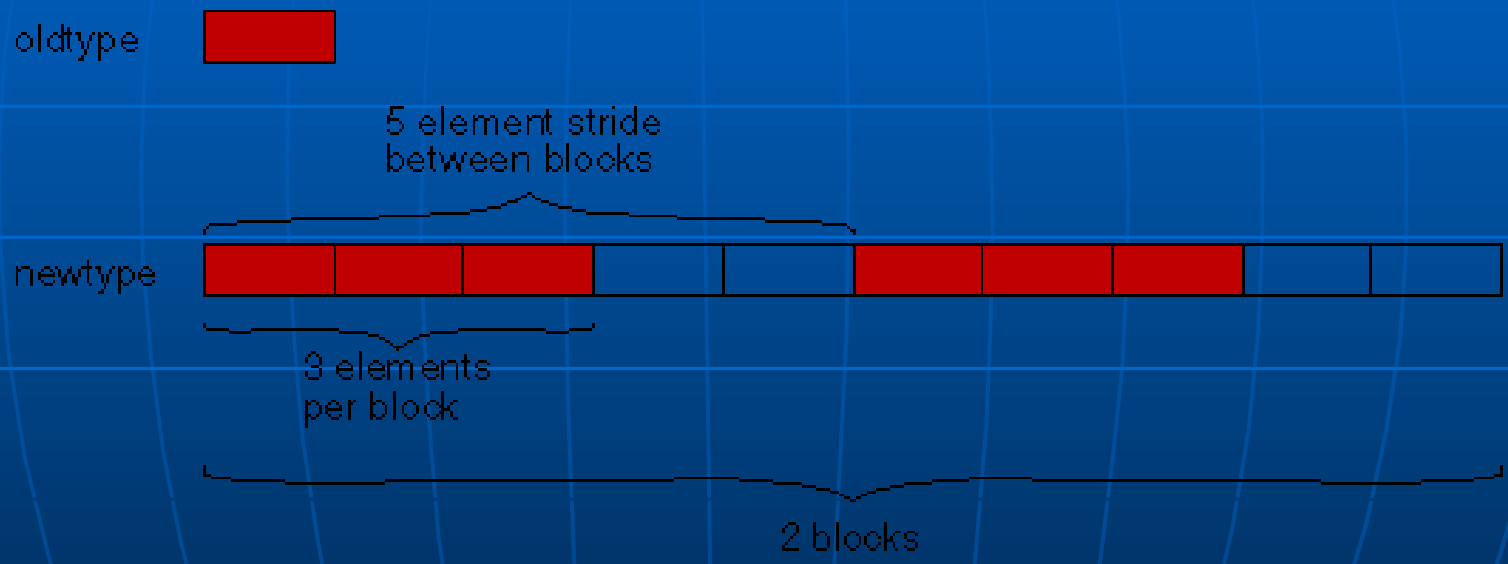
Funkcija koja omogućava kreiranje novog tipa podatka na osnovu starog tipa podatka, gde se blokovi nalaze na uniformno odvojenim memorijskim lokacijama i gde se svaki blok sastoji od istog broja podataka starog tipa:

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

gde je count-broj blokova
blocklength-broj elemenata starog tipa u svakom bloku
stride -broj elemenata starog tipa između početaka dva bloka
oldtype-stari tip
newtype-novi tip

Izvedeni tipovi podataka

Npr.za count=2,blocklength=3 i stride=5



Kviz

```
int count = 4, blocklen = 2, stride = 4;
MPI_Datatype newtype;
MPI_Type_vector(count, blocklen, stride, MPI_FLOAT, &newtype);
```

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

a[4][4]

```
MPI_Send(&a[0][1], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

Šta će biti poslato?

Kviz

```
int count = 4, blocklen = 2, stride = 4;
MPI_Datatype newtype;
MPI_Type_vector(count, blocklen, stride, MPI_FLOAT, &newtype);
```

blocklen = 2

stride = 4

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

a[4][4]

```
MPI_Send(&a[0][1], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

2.0	3.0	6.0	7.0	10.0	11.0	14.0	15.0

88

Šta će biti poslato?

Kviz

```
int count = 3, blocklen = 1, stride = 5;
MPI_Datatype newtype;
MPI_Type_vector(count, blocklen, stride, MPI_FLOAT, &newtype);
```

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

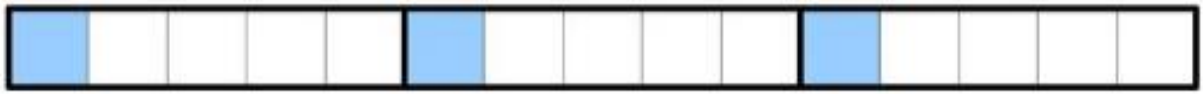
a[4][4]

```
MPI_Send(&a[0][1], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

Šta će biti poslato?

Kviz

```
int count = 3, blocklen = 1, stride = 5;
MPI_Datatype newtype;
MPI_Type_vector(count, blocklen, stride, MPI_FLOAT, &newtype);
```



blocklen = 1

stride = 5

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

a[4][4]

```
MPI_Send(&a[0][1], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

2.0	7.0	12.0

Kviz

```
int count = ???, blocklen = ???, stride = ???;
MPI_Datatype newtype;
MPI_Type_vector(count, blocklen, stride, MPI_FLOAT, &newtype);
```

a[4][4]

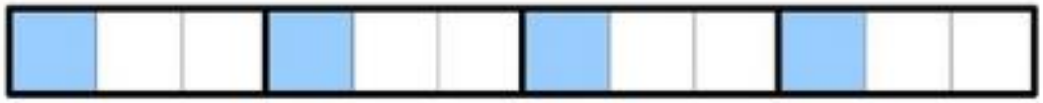
```
MPI_Send(???????, 1, newtype, dest, tag, MPI_COMM_WORLD);
```

91

Da bi se poslali podaci sa slike šta treba da stoji na mestu ???

Kviz

```
int count = 4; blocklen = 1, stride = 3;
MPI_Datatype newtype;
MPI_Type_vector(count, blocklen, stride, MPI_FLOAT, &newtype);
```



blocklen = 1

stride = 3

1.0	2.0	3.0	4.0
5.0	6.0	7.0	8.0
9.0	10.0	11.0	12.0
13.0	14.0	15.0	16.0

a[4][4]

```
MPI_Send(&a[0][3], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

Izvedeni tipovi podataka

Funkcija koja omogućava da blokovi budu različitih dužina kao i da razmaci između početaka blokova budu različitih dužina je:

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

gde je
count-broj blokova
array_of_blocklengths –niz čiji su elementi brojevi elemenata starog tipa u svakom bloku
array_of_displacements–niz koji sadrži pomeraje za svaki blok izražen u elementima starog tipa

Izvedeni tipovi podataka

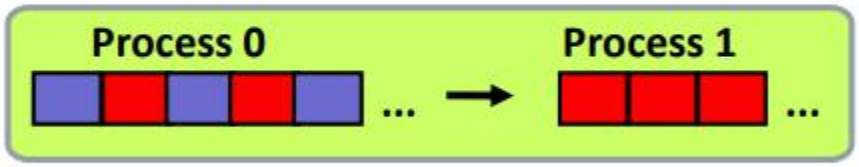
Primer za **MPI_Type_indexed**

```
count = 3      blocklengths[0] = 2 displacements[0] = 0
               blocklengths[1] = 1 displacements[1] = 3
               blocklengths[2] = 4 displacements[2] = 5
```

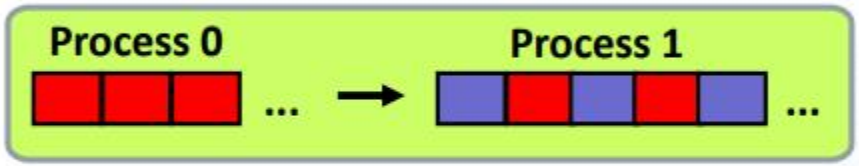
Pre nego što se koristi u funkcijama MPIa tip mora biti potvrđen i to se obavlja funkcijom:

int MPI_Type_commit (MPI_datatype *datatype)

Izvedeni tipovi podataka




```
if (myid == 0)
  MPI_Type_vector(n, 1, 2,
                  MPI_FLOAT, &newtype)
  ...
  MPI_Send(A, 1, newtype, 1, ...)
else
  MPI_Recv(B, n, MPI_FLOAT, 0, ...)
```



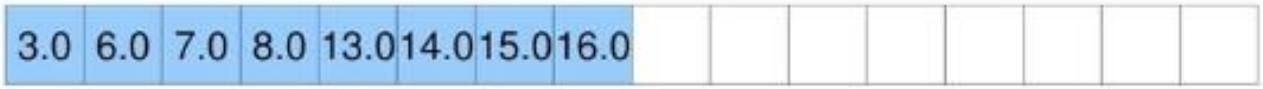

```
if (myid == 0)
  MPI_Send(A, n, MPI_FLOAT, 1, ...)
else
  MPI_Type_vector(n, 1, 2, MPI_FLOAT,
                  &newtype)
  ...
  MPI_Recv(B, 1, newtype, 0, ...)
```

Izvedeni tipovi podataka

```
int count = 3; int blklens[] = {1, 3, 4}; int displs[] = {2, 5, 12};
MPI_Datatype newtype;
MPI_Type_indexed(count, blklens, displs, MPI_FLOAT, &newtype);
```

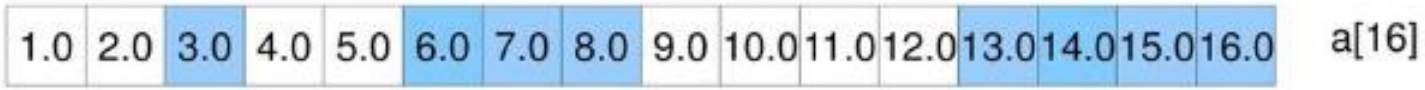

```
MPI_Send(&a[0], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

```
MPI_Recv(&b[0], 8, MPI_FLOAT, src, tag, MPI_COMM_WORLD);
```



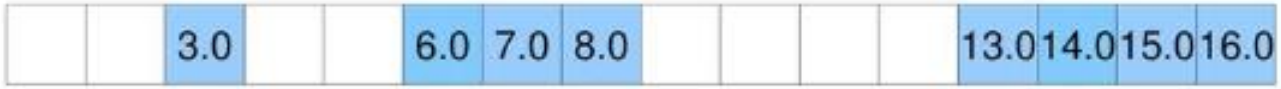
Izvedeni tipovi podataka

```
int count = 3; int blklens[] = {1, 3, 4}; int displs[] = {2, 5, 12};
MPI_Datatype newtype;
MPI_Type_indexed(count, blklens, displs, MPI_FLOAT, &newtype);
```




```
MPI_Send(&a[0], 1, newtype, dest, tag, MPI_COMM_WORLD);
```

```
MPI_Recv(&b[0], 1, newtype, src, tag, MPI_COMM_WORLD);
```



Izvedeni tipovi podataka

Tip se kreira opisom N-dimenzionalnog podpolja unutar N-dimenzionalnog polja

int MPI_Type_create_subarray(int ndims, int *sizes, int *subsizes, int *offsets, int order, MPI_Datatype oldtype, MPI_Datatype *newtype)

ndims -broj dimenzija polja (N)(pozitivan broj)
sizes - broj elemenata starog tipa (**oldtype**) u svakoj dimenziji polja (niz pozitivnih celih brojeva)
subsizes - broj elemenata starog tipa (**oldtype**) u svakoj dimenziji podpolja(niz pozitivnih celih brojeva)
offsets - početne koordinate podpolja u svakoj dimenziji(niz nenegativhih brojeva)
order - način predstavljanja polja u memoriji. Ili MPI_ORDER_C ili MPI_ORDER_FORTRAN

Example: subarray

```
int array_size[2]     = {5,5};
int subarray_size[2]  = {2,2};
int subarray_start[2] = {1,1};
MPI_Datatype subtype;
double **array

for (i=0; i<array_size[0]; i++)
  for (j=0; j<array_size[1]; j++)
    array[i][j] = rank;
MPI_Type_create_subarray(2, array_size, subarray_size, subarray_start,
                         MPI_ORDER_C, MPI_DOUBLE, &subtype);
MPI_Type_commit(&subtype);
if (rank==0)
  MPI_Recv(array[0], 1, subtype, 1, 123, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
if (rank==1)
  MPI_Send(array[0], 1, subtype, 0, 123, MPI_COMM_WORLD);
```

```
Rank 0: original array
  0.0   0.0   0.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0
Rank 0: array after receive
  0.0   0.0   0.0   0.0   0.0
  0.0   1.0   1.0   0.0   0.0
  0.0   1.0   1.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0
  0.0   0.0   0.0   0.0   0.0
```

Izvedeni tipovi podataka-zadaci

zad. Napisati program kojim se korišćenjem izvedenih tipova podataka vrši slanje kolone 0 procesa 0 u vrstu 0 procesa 1.

```
#include <stdio.h>
#include "mpi.h„
#define n 10
main(int argc, char* argv[]) {

    int my_rank;
    float A[n][n];
    MPI_Status status;
    MPI_Datatype column_mpi_t;
    int i, j;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    MPI_Type_vector(n, 1, n, MPI_FLOAT, &column_mpi_t);
    MPI_Type_commit(&column_mpi_t);
```

Izvedeni tipovi podataka-zadaci

```
if (my_rank == 0) {
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                A[i][j] = (float) i;
        MPI_Send(&(A[0][0]), 1, column_mpi_t, 1, 0,
                 MPI_COMM_WORLD);
    } else { /* my_rank = 1 */
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                A[i][j] = 0.0;
        MPI_Recv(&(A[0][0]), n, MPI_FLOAT, 0, 0,
                 MPI_COMM_WORLD, &status);
        for (j = 0; j < n; j++)
            printf("%3.1f ", A[0][j]);
        printf("\n");
    }
    MPI_Finalize();
}
```

Izvedeni tipovi podataka-zadaci

P0

A

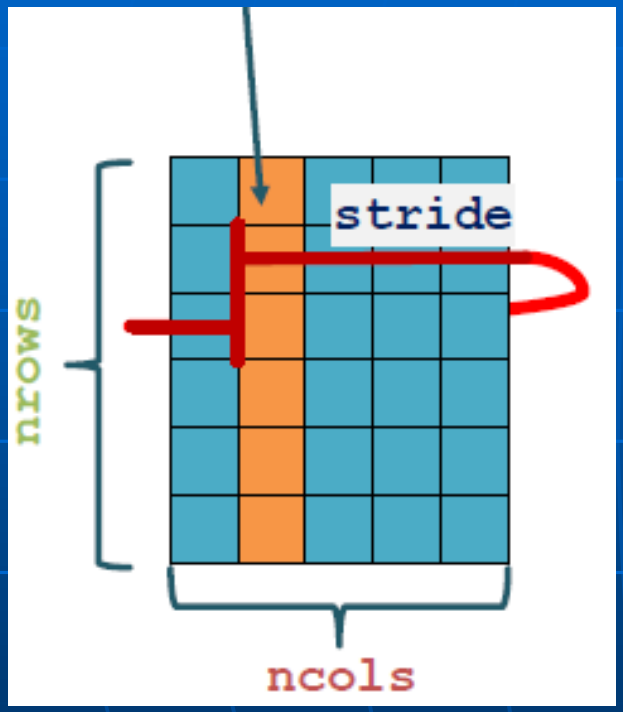
0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4

P1

A

0	1	2	3	4
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Izvedeni tipovi podataka

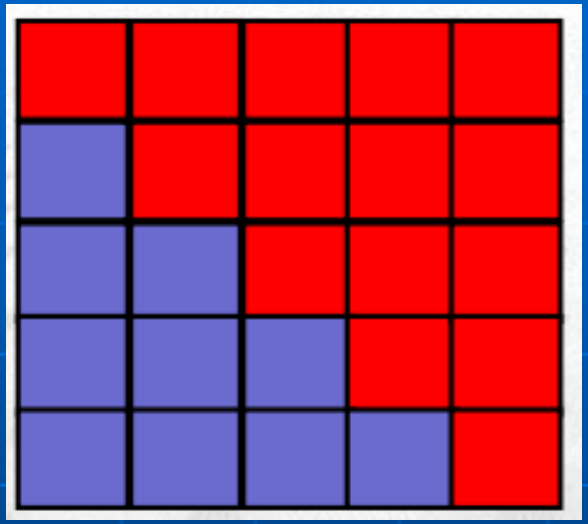


nrows-broj vrsta
ncols-broj kolons

```
MPI_Type_vector(nrows, 1, ncols,
MPI_DOUBLE, &nt);
MPI_Type_commit(&nt);

MPI_Send(&matrix[0][1], 1, nt, …);
```

Izvedeni tipovi podataka-zadaci

zad. Napisati program kojim se elementi gornje trougaone matrice Anxn procesa 0, šalju u gornju trougaonu matricu Tnxn procesa 1.



```
#include <stdio.h>
#include "mpi.h"
#define n 10
main(int argc, char* argv[]) {
    int p;
    int my_rank;
    float        A[n][n];
    float        T[n][n];
    int          displacements[n];
    int          block_lengths[n];
    MPI_Datatype  index_mpi_t;
    int          i, j;
    MPI_Status    status;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

displ

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

Izvedeni tipovi podataka-zadaci

```
for (i = 0; i < n; i++) {
    block_lengths[i] = n-i;
    displacements[i] = (n+1)*i;
}
MPI_Type_indexed(n, block_lengths, displacements,
    MPI_FLOAT, &index_mpi_t);
MPI_Type_commit(&index_mpi_t);
if (my_rank == 0) {
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            A[i][j] = (float) i + j;
    MPI_Send(A, 1, index_mpi_t, 1, 0, MPI_COMM_WORLD);
} else
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            T[i][j] = 0.0;
    MPI_Recv(T, 1, index_mpi_t, 0, 0, MPI_COMM_WORLD, &status);
```

Izvedeni tipovi podataka-zadaci

```
for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++)
            printf("%4.1f ", T[i][j]);
        printf("\n");
    }
  }
  MPI_Finalize();
}
```

P0

A

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

P1

T

0	1	2	3	4
0	2	3	4	5
0	0	4	5	6
0	0	0	6	7
0	0	0	0	8

Donja u gornju trougaonu matricu?

```
for (i = 0; i < n; i++) {
        block_lengths[i] = n-i;
        displacements[i] = (n+1)*i;
    }
    MPI_Type_indexed(n, block_lengths, displacements,
        MPI_FLOAT, &gornja);
    MPI_Type_commit(&gornja);
```

displ

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

```
 for (i = 0; i < n; i++) {
        block_lengths[i] = ?;
        displacements[i] = ?;
    }
    MPI_Type_indexed(n, block_lengths, displacements,
        MPI_FLOAT, &donja);
    MPI_Type_commit(&donja);
```

Donja u gornju trougaonu?

```
if (my_rank == 0) {
      for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
            A[i][j] = (float) i + j;
      MPI_Send(A, 1, donja, 1, 0, MPI_COMM_WORLD);
   } else
      for (i = 0; i < n; i++)
         for (j = 0; j < n; j++)
            T[i][j] = 0.0;
      MPI_Recv(T, 1, gornja, 0, 0, MPI_COMM_WORLD, &status);
```

P0

P0

A

0	1	2	3	4
1	2	3	4	5
2	3	4	5	6
3	4	5	6	7
4	5	6	7	8

P1

T

0	1	2	2	3
0	4	3	4	5
0	0	6	4	5
0	0	0	6	7
0	0	0	0	8

Izvedeni tipovi podataka-zadaci

zad. Napisati MPI program koji čita celobrojne (integer) i vrednosti duple tačnosti (double-precision) sa standardnog ulaza i saopštava to svim ostalim procesima

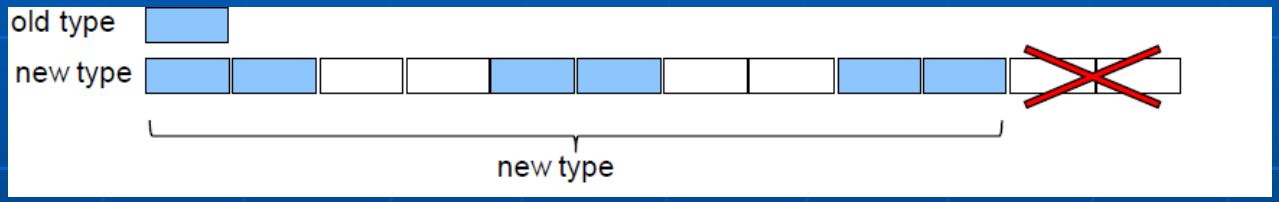
```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char **argv )
{
    int          rang;
    struct { int a; double b; } value;
    MPI_Datatype mystruct;
    int          blocklens[2];
    MPI_Aint     indices[2];
    MPI_Datatype old_types[2];
    MPI_Init( &argc, &argv );
    MPI_Comm_rang( MPI_COMM_WORLD, &rang );
    blocklens[0] = 1;
    blocklens[1] = 1;
    old_types[0] = MPI_INT;
    old_types[1] = MPI_DOUBLE;
```

Izvedeni tipovi podataka-zadaci

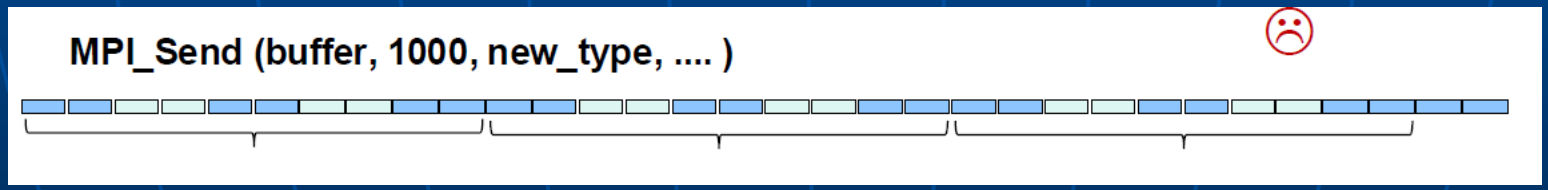
```
/* Lokacije svakog elementa */
    MPI_Address( &value.a, &indices[0] );
    MPI_Address( &value.b, &indices[1] );
    indices[1] = indices[1] - indices[0];
    indices[0] = 0;
    MPI_Type_struct( 2, blocklens, indices, old_types, &mystruct );
    MPI_Type_commit( &mystruct );
    if (rang == 0)
            scanf( "%d %lf", &value.a, &value.b );
    MPI_Bcast( &value, 1, mystruct, 0, MPI_COMM_WORLD );
    printf( "Process %d got %d and %lf\n", rang, value.a, value.b );

    MPI_Finalize( );
    return 0;
}
```

Izvedeni tipovi podataka

Ako želimo da u nekoj od funkcija za komunikaciju koristimo izvedeni tip podatka a da je pritom count>1, kao na primer MPI_Send(buff,10,izvedeni_tip,...) moramo voditi računa o tome šta će u tom slučaju biti poslato i da li je to ono što želimo da pošaljemo. Npr. funkcijom **MPI_Type_vector(3,2,4,oldtype,&newtype)** kreiramo tip:



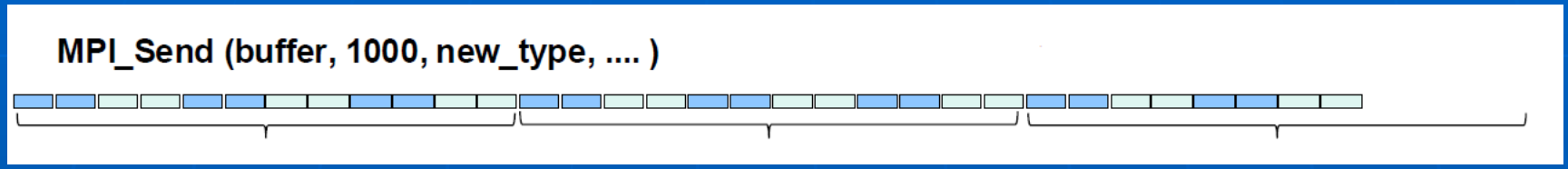
Ako želimo da sa MPI_Send pošaljemo cnt>1 podataka newtype, ono što će biti poslato je:



tj. slanje sledećeg podatka tipa newtype kreće sa adrese koja je odmah iza poslednjeg poslatog bajta podatka. Najčešće, ovo nije ono što mi želimo da bude poslato.

Izvedeni tipovi podataka

Ako je ono što je potrebno da bude poslato sa MPI_Send:



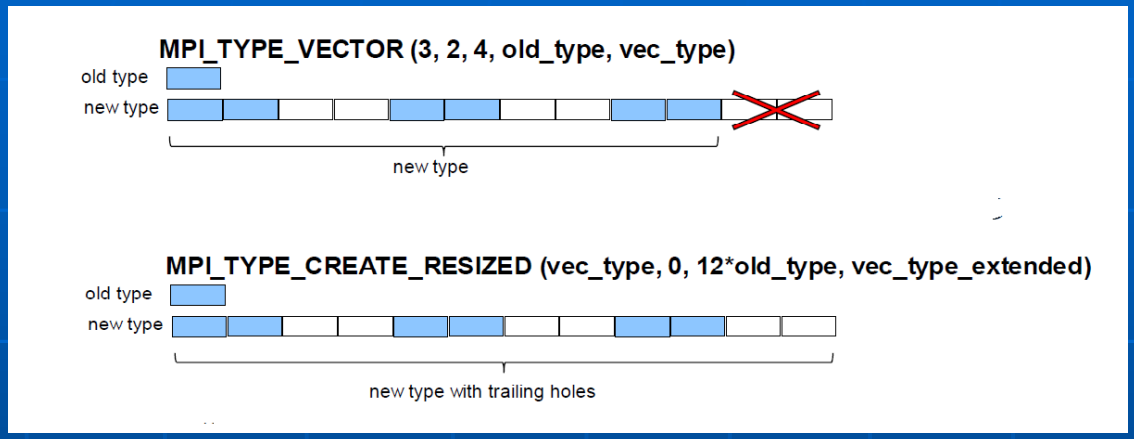
Problem možemo rešiti korišćenjem funkcije

int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint extent, MPI_Datatype *newtype)

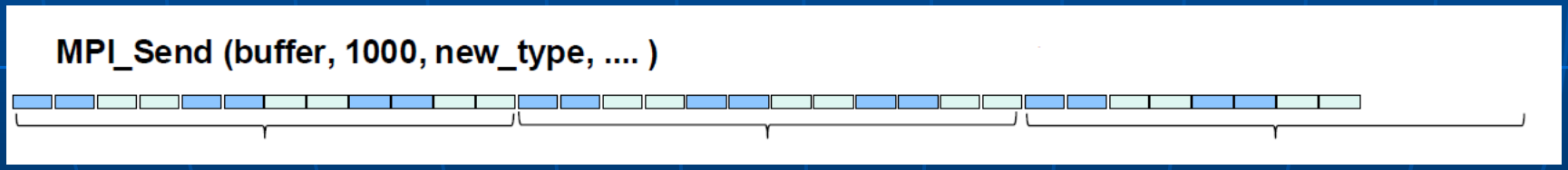
lb -nova donja granica tipa (namanji pomeraj u novom tipu, uglavnom je jednak onom u starom tipu, tj 0)
extent –nova veličina tipa koja utiče na to odakle će krenuti slanje sledeće jedinice novog tipa
oldtype-stari tip
newtype-novi tip

Izvedeni tipovi podataka

Dakle, uradićemo sledeće



Ono što će biti poslato sa MPI_Send je

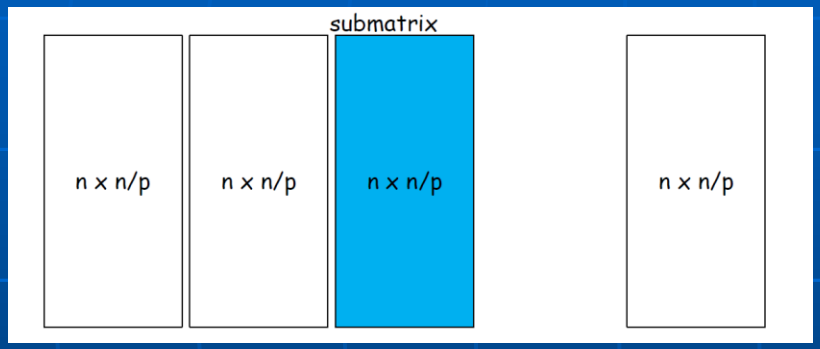


Možemo da zaključimo da MPI_Send(buff, cnt, ndt,...) praktično implementira for (i=0;i<cnt;i++) MPI_Send(buff[i*extent] , 1, ndt,...) .

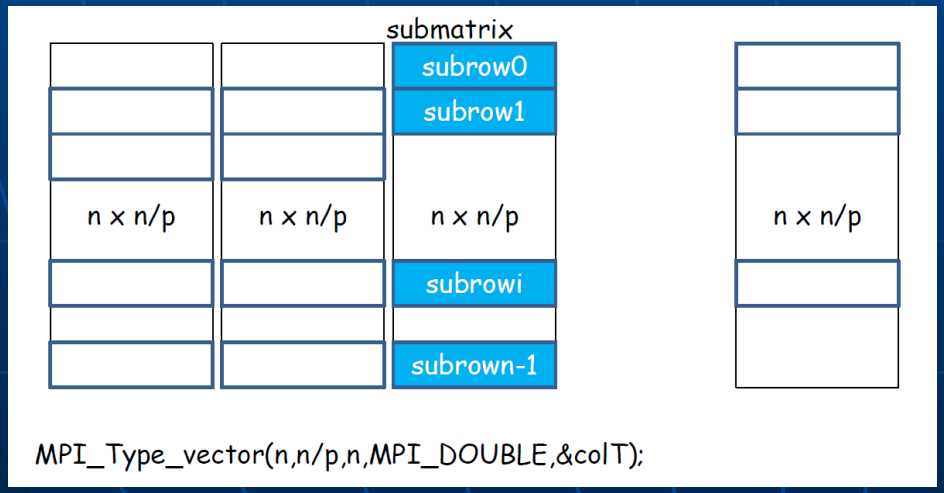
Sva ova zapažanja važe za korišćenje i drugih funkcija za komunikaciju (Scatter, Gather,...)

Izvedeni tipovi podataka

Primer. Želimo da pošaljemo iz master procesa (npr.P0) po n/p (n-dimenizija matrice, p-broj procesa, n deljivo sa p) kolona matrice A svakom procesu i da se u svakom procesu (pa i u P0) te vrednosti nakon slanja nadju u polju niz. U ovu svrhu možemo koristiti funkciju MPI_Scatter.

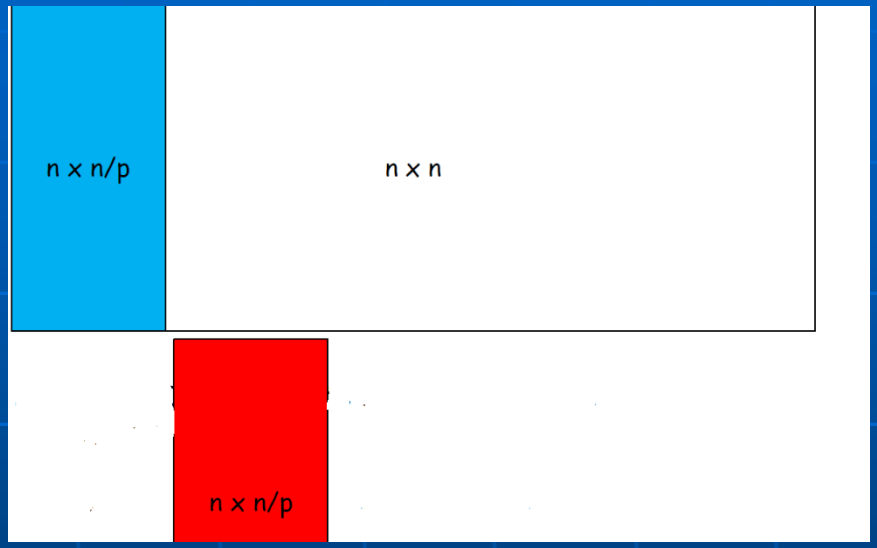


Da bi poslali n/p kolona odjednom kreiraćemo izvedeni tip colT:



Izvedeni tipovi podataka

Ono što bi se redom slalo iz svakog procesa sa MPI_Scatter(a,1,colT,niz,n*n/p,MPI_DOUBLE,0, MPI_COMM_WORLD) je:



Pošto to ne odgovara onom što smo želeli da bude poslato, koristimo:

```
MPI_Type_vector(n,n/p,n,MPI_DOUBLE,&colT);
MPI_Type_create_resized(colT,0,n/p*sizeof(double),&subT);
MPI_Type_commit(&subT);
```

koji nam omogućava da sa MPI_Scatter(a,1,subT,niz,n*n/p,MPI_DOUBLE,0, MPI_COMM_WORLD) pošaljemo ono što smo hteli.

Zad. Napisati MPI program koji realizuje množenje matrice A_{mxn} i matrice B_{nxk}, čime se dobija i prikazuje rezultujuća matrica C_{mxk}. Izračunavanje se obavlja tako što master proces šalje svakom procesu po jednu kolonu matrice A i po jednu vrstu matrice B. Svi elementi kolone matrice A se šalju odjednom. Svi procesi učestvuju u izračunavanjima potrebnim za generisanje rezultata programa. Zadatak rešiti isključivo

primenom grupnih operacija

m=2 n=3 k=4

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

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

Zad. Napisati MPI program koji realizuje množenje matrice A_{mxn} i matrice B_{nxk}, čime se dobija i prikazuje rezultujuća matrica C_{mxk}. Izračunavanje se obavlja tako što master proces šalje svakom procesu po jednu kolonu matrice A i po jednu vrstu matrice B. Svi elementi kolone matrice A se šalju odjednom. Svi procesi učestvuju u izračunavanjima potrebnim za generisanje rezultata programa. Zadatak rešiti isključivo primenom grupnih operacija

```
#include <stdio.h>
#include <mpi.h>
#define m 2
#define n 3
#define k 4
void main(int argc, char* argv[])
{int a[m][n],b[n][k],c[m][k],local_c[m][k],root=0,niza[m],nizb[k],rez[m],rank,i,j,p;
MPI_Datatype vector,column,dt;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&p);
```

```
MPI_Type_vector(m,1,n,MPI_INT,&vector);
MPI_Type_commit(&vector);
MPI_Type_create_resized( vector, 0, 1*sizeof(int),&column );
MPI_Type_commit(&column);
if (rank == root)
{
    for(i = 0; i < m; i++)

            for(j = 0; j < n; j++)
                        a[i][j] = i+j;

    for(i = 0; i < n; i++)
            for(j = 0; j < k; j++)
                    b[i][j] = 1+j-i;

}
MPI_Scatter(a,1,column,niza,m,MPI_INT,0,MPI_COMM_WORLD);
MPI_Scatter(b,k,MPI_INT,nizb,k,MPI_INT,0,MPI_COMM_WORLD);
for (i=0;i<m;i++)
            for (j=0;j<k;j++)
                        local_c[i][j]=niza[i]*nizb[j];
for (i=0;i<m;i++)
            for (j=0;j<k;j++)
                        c[i][j]=0;
```

```
MPI_Reduce(&local_c[0][0],&c[0][0],m*k,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD;
if (rank==0)
{
   for (i = 0; i< m; i++)
  {
            for (j = 0; j< k; j++)
                  printf("%d ",c[i][j]);

           printf("\n");
   }

     MPI_Finalize();

}
```