Paralelni sistemi: MPI-Point-to-Point i grupne operacije

Prof. dr Natalija Stojanović

Point-to-point komunikacija-bez blokiranja

- Podsetnik!
- Komunikacija sa blokiranjem
- Nakon MPI poziva funkcija
  - Izvorni proces može bezbedno da modifikuje *send_buffer*
    - Potencijalna implementacije za Send operaciju:sinhronizovana ili sa baferovanjem
  - *Recv_buffer* u procesu primaocu sadrži poruku od izvora, *recv_buffer* može da se bezbedno modifikuje
- Može zahtevati sinhronizaciju između procesa, što se može odraziti na performanse

Point-to-point komunikacija-bez blokiranja

- Motivacija za komunikaciju bez blokiranja:
  - Izbegavanja deadlock-a
  - Izbegavanje nezaposlenih procesa
  - Izbegavanje bespotrebne sinhronizacije
  - Preklapanje komunikacije i izračunavanja (korisnog posla), tj. skrivanje ”troškova komunikacije”

Point-to-point komunikacija-bez blokiranja

MPI podržava komunikaciju bez blokiranja, u kome jedan proces može započeti operaciju slanja ili prijema poruke a nakon čega može nastaviti sa obavljanjem drugog posla a zatim se vraća da proveri završetak tj. status operacije. Ovde se slanje i prijem odvija u tri koraka:

1. Iniciranje send/recv operacije pozivom funkcije MPI**_Isend()/MPI_Irecv(I-immidiately)**

2. Obavljanje nekog drugog posla tokom vremena komuniciranja

3. Čekanje na kompletiranje ili testiranje kompletiranja komunikacije korišćenjem funkcija MPI_Wait() ili MPI_Test().

P-t-p komunikacija bez blokiranja- nast.

Iz funkcije MPI_**Isend()** se vraća odmah, pre nego što poruka bude iskopirana u bafer.
Sintaksa funkcije za slanje bez blokiranja je:

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

Promenljiva request je identifikator komunikacionog događaja.
Na osnovu request se proverava (testira) status inicirane operacije ili kompletira njeno izvršenje.
Program ne sme da modifikuje promenljivu buf nakon iniciranja operacije, sve dok MPI_Wait ili MPI_Test funkcija ne daju pozitivnu informaciju o kompletiranju operacije identifikovane sa request.

P-t-p komunikacija bez blokiranja- nast.

Proces vrši prijem bez blokiranja, tj. inicira prijem uspomoć funkcije:

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

Iz ove funkcije se proces vraća odmah, bez potrebe za čekanjem da poruka bude smeštena u prijemni bafer.
Za razliku od **MPI_Recv** ova funkcija ne vraća informaciju o statusu (promenljiva tipa MPI_Status).
Ove informacije se mogu dobiti pri pozivu funkcije za kompletiranje poruke (MPI_Wait, MPI_Test).

P-t-p komunikacija bez blokiranja- nast.

Funkcije koje se koriste za proveru kompletiranja operacija bez blokiranja su:

int MPI_Wait(MPI_Request *request, MPI_Status *status);

iz koje se proces vraća onda kada se operacija identifikovana sa request završi.
Ako je inicirana operacija MPI_Irecv onda promenljiva tipa MPI_ status čuva informaciju o izvoru poruke, oznaci poruke kao i broju primljenih podataka.
U slučaju MPI_ISend čuva informaciju o grešci.
Ova operacija je blokirajuća.

P-t-p komunikacija bez blokiranja- nast.

i funkcija

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);

vraća informaciju o trenutnom stanju operacije koja je identifikovana argumentom request.
Argument flag se postavlja na "true" ukoliko je operacija završena, u suprotnom na "false". Argument status sadrži dodatne statusne informacije.
Ova operacija nije blokirajuća.

P-t-p komunikacija bez blokiranja- izbegnut deadlock

```
#include <stdio.h>
#include <mpi.h>
void main (int argc, char **argv) {
        int myrank,x,y;
        MPI_Request req;
        MPI_Status status;
        MPI_Init(&argc, &argv);  /* Initialize MPI */
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* Get rank */
        if( myrank == 0 ) {
                x=3;
                MPI_Irecv( &y, 1, MPI_INT, 1, 19, MPI_COMM_WORLD, &req);
                MPI_Send( &x, 1, MPI_INT, 1, 17, MPI_COMM_WORLD );
                MPI_Wait( &req, &status );
        }
         else if( myrank == 1 ) {
                x=5;
                MPI_Irecv(&y, 1, MPI_INT, 0, 17, MPI_COMM_WORLD, & req);
                MPI_Send(&x, 1, MPI_INT, 0, 19, MPI_COMM_WORLD );
                MPI_Wait( &req, &status );
        }
        printf("Proc %d y= %d", myrank, y);
        MPI_Finalize();
}
```

Grupne (collective) operacije

Grupne operacije su operacije koje se primenjuju nad svim članovima jedne grupe.

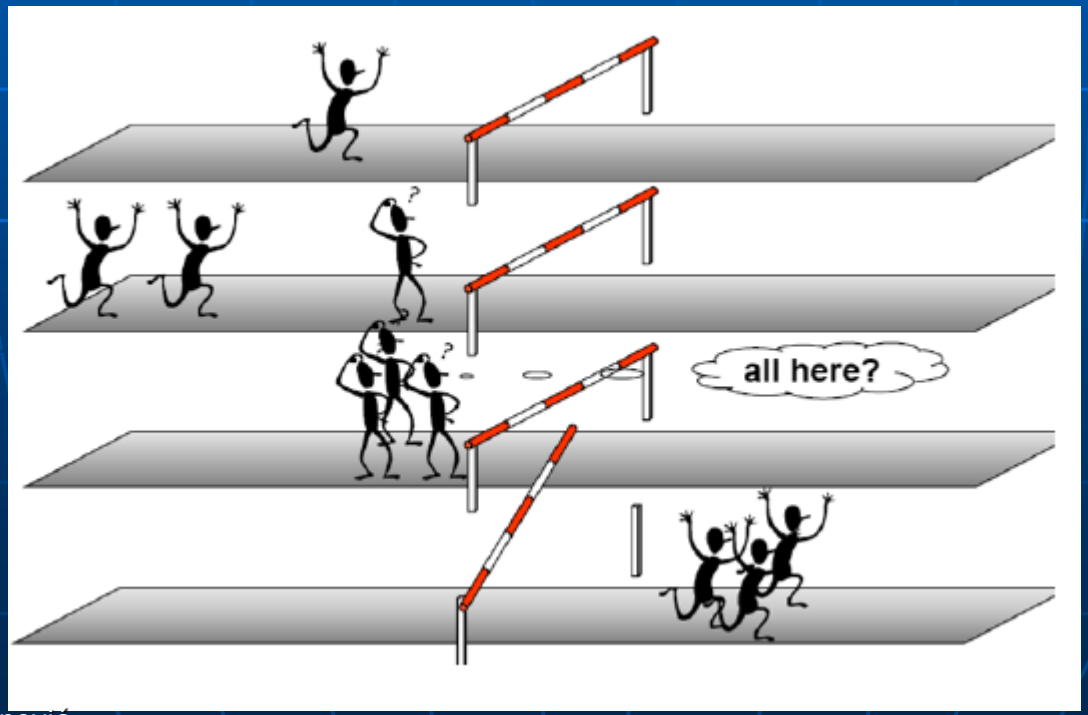
Operacija se izvršava kada svi procesi pozovu odgovarajuću operaciju sa svojim parametrima.

Svaki proces mora da pozove grupnu operaciju da bi se ona obavila!!!!

Dele se na operacije za kontrolu procesa, operacije za globalna izračunavanja i operacije za prenos podataka.

Operacija za kontrolu procesa

int MPI_Barrier (MPI_Comm comm)- implementira sinhronizacioni mehanizam poznat kao barijera. Proces se blokira na toj naredbi dok svi ostali procesi iz grupe ne dođu do te naredbe. Tada se svi procesi vraćaju daljem izvršenju.



Operacije za globalna izračunavanja

Ovde spadaju operacije za redukciju podataka i operacija scan. Operacija redukcije uzima podatke u ulaznim baferima svih procesa, primenjuje nad njima datu operaciju redukcije i smešta rezultat u promenljivu root procesa.

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

send_buffer-adresa send bafera svih procesa gde se nalaze podaci nad kojima se obavlja operacija redukcije
recv_buffer-adresa receive bafera root procesa
count-broj podataka u send i receive baferu
datatype-tip podataka u send i receive baferu
comm-komunikator
rank-identifikator root procesa

Operacije za globalna izračunavanja

operation-operacija redukcije koja može biti

MPI_MAX	maximum
MPI_MIN	minimum
MPI_SUM	sum
MPI_PROD	product
MPI_LAND	logical and
MPI_BAND	bit-wise and
MPI_LOR	logical or
MPI_BOR	bit-wise or
MPI_LXOR	logical xor
MPI_BXOR	bitwise xor
MPI_MINLOC	computes a global minimum and an index attached to the minimum value -- can be used to determine the rank of the process containing the minimum value
MPI_MAXLOC	computes a global maximum and an index attached to the rank of the process containing the minimum value

Operacije za globalna izračunavanja

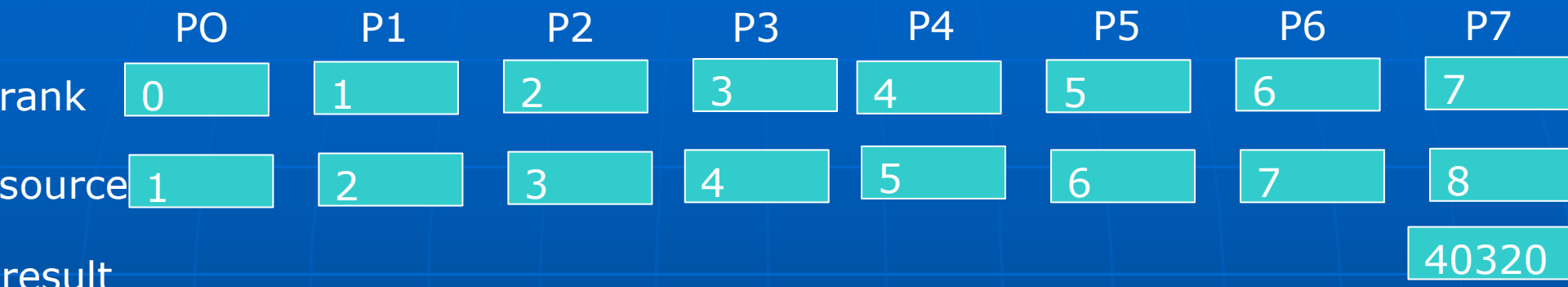
Pr.

```
#include   <stdio.h>
#include   <mpi.h>
void main(int argc, char *argv[])
{
        int rank;
        int source,result,root;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        root=7;
        source=rank+1;
        MPI_Reduce(&source,&result,1,MPI_INT,MPI_PROD,root,MPI_COMM_
        WORLD);
        if(rank==root) printf("PE:%d MPI_PROD result is %d
        \n",rank,result);
        MPI_Finalize();
}
```

Root može biti bilo koji proces!

Operacije za globalna izračunavanja

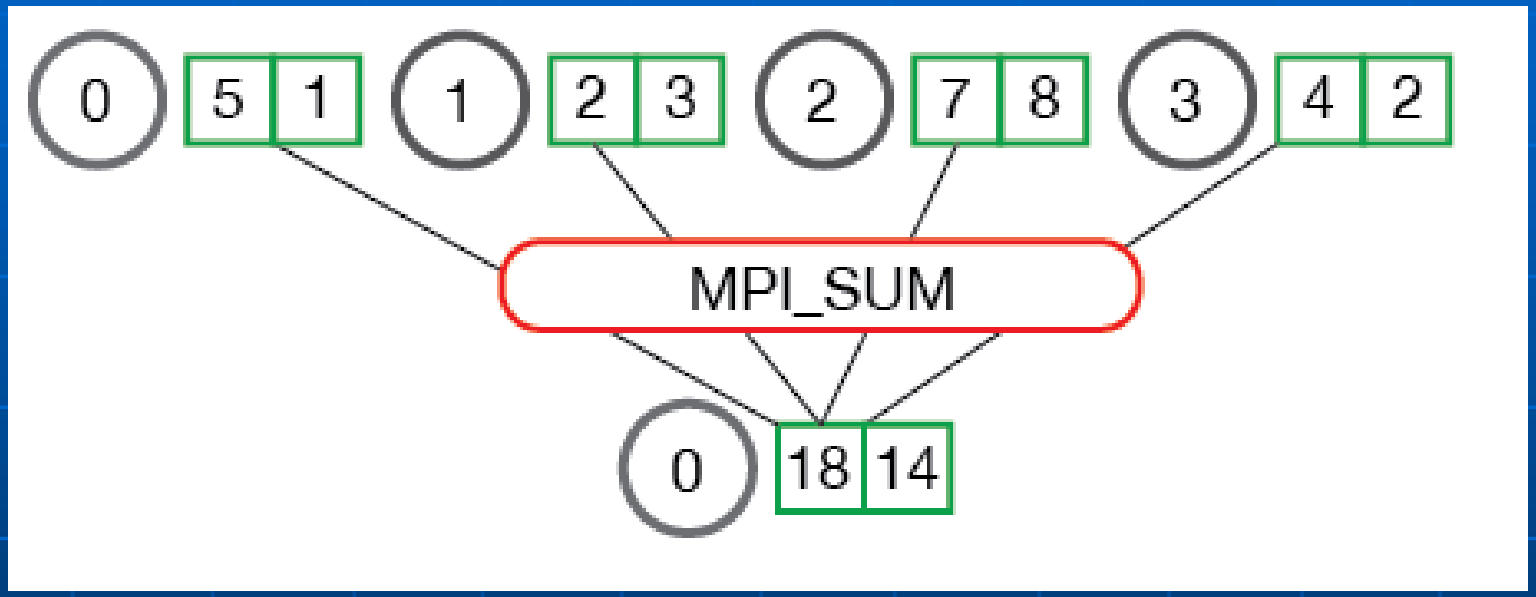
Pr.



Ako se program startuje za 8 procesa izlaz je:

PE:7 MPI_PROD result is 40320

Reduce operacija nad više elemenata



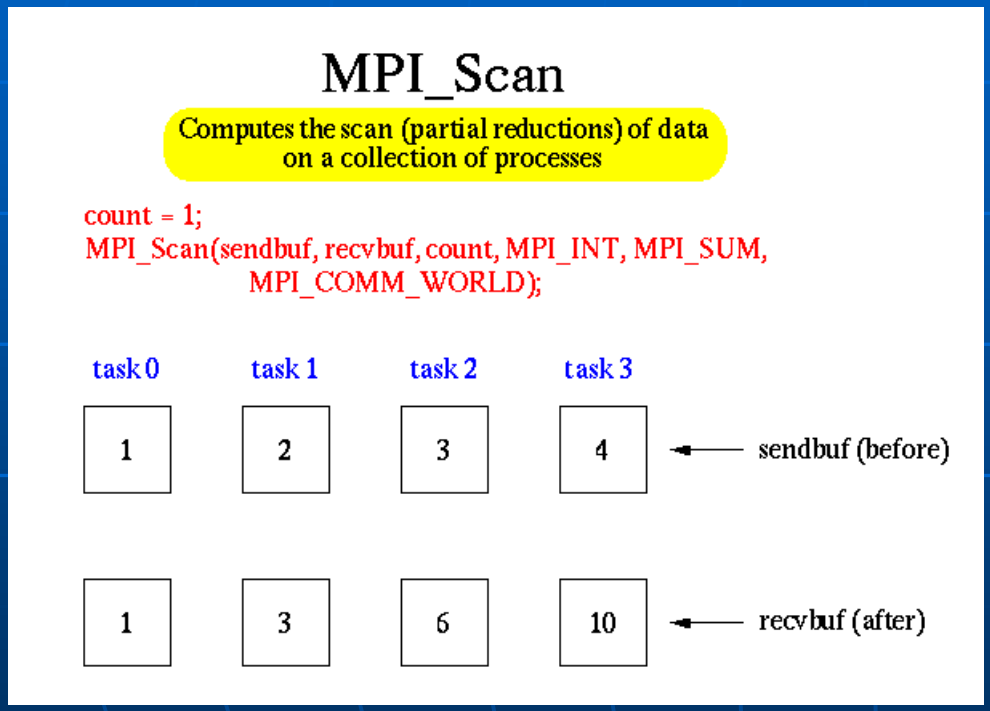
Operacije za globalna izračunavanja

Operacija Scan - Ova operacija se još zove prefix reduce operacija. Operacija vraća u receive bafer procesa sa rangom i, redukciju vrednosti u send baferima sa rangovima 0..i.

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

```
#include   <stdio.h>
#include   <mpi.h>
void main(int argc, char *argv[])
{
        int rank;
        int source,result;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        source=rank+1;
        MPI_Scan(&source,&result,1,MPI_INT,MPI_SUM,MPI_COMM_
        WORLD);
        printf("PE:%d SUM %d \n",rank,result);
        MPI_Finalize();
}
```

Operacije za globalna izračunavanja



Izlaz za prethodni program za 4 procesa:
PE:3 SUM 10
PE:1 SUM 3
PE:0 SUM 1
PE:2 SUM 6

Operacije za prenos podataka

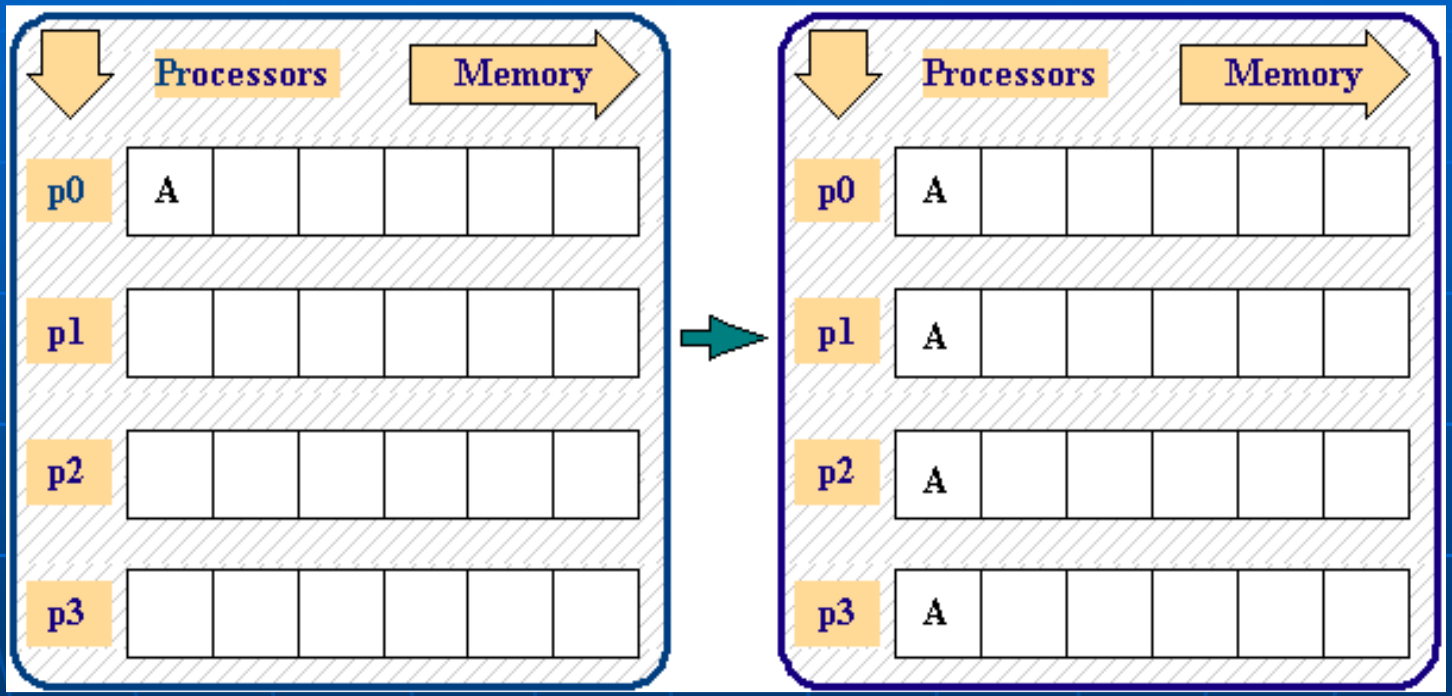
Osnovne operacije za prenos podataka su brodcast, scatter i gather.

MPI_Bcast omogućava kopiranje podataka iz memorije root procesa u mesta u memoriji ostalih procesa. Ovu funkciju u slučaju ovakve razmene podataka moraju pozvati svi procesi koji učestvuju u razmeni, tj. koji pripadaju datom komunikatoru.

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

gde je buffer adresa bafera sa koje se šalje count podataka tipa datatype svim procesima.

Operacije za prenos podataka



Operacije za prenos podataka

```
Pr.# include <stdio.h>
   #include <mpi.h>
   void main(int argc, char *argv[]) {
        int rank;
        double param;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        if(rank==5)
                param=23.0;
        MPI_Bcast(&param,1,MPI_DOUBLE,5,MPI_COMM_WORLD);
        printf("P:%d after broadcast parameter is %f \n",rank,param);
        MPI_Finalize();
}
```

Operacije za prenos podataka

Izlaz za 7 procesa:

```
P: 0  after broadcast param is  23.
P: 5  after broadcast param is  23.
P: 2  after broadcast param is  23.
P: 3  after broadcast param is  23.
P: 4  after broadcast param is  23.
P: 1  after broadcast param is  23.
P: 6  after broadcast param is  23.
```

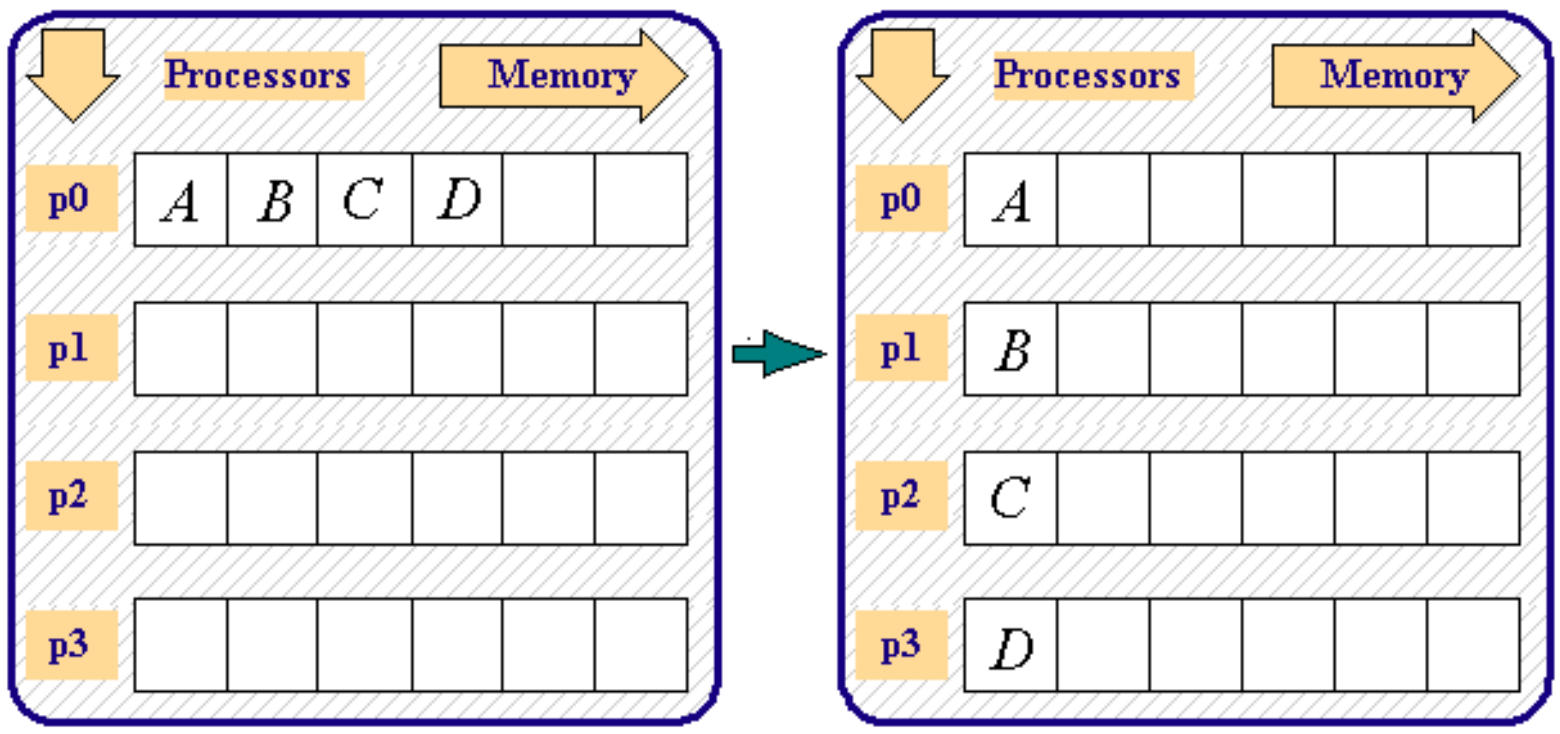
Operacije za prenos podataka

Operacija MPI_Scatter omogućava da i-ti segment bafera root procesa bude poslat i-tom procesu u grupi gde su segmenti iste veličine.

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)

send_buffer -adresa bafera root procesa odakle počinje slanje podataka.
send_count -koliko podataka se šalje svakom procesu (broj podataka u segmentu)
send_type- tip podataka koji se šalje
recv_buffer-adresa prijemnog bafera
recv_count- koliko podataka se prima u prijemni bafer
recv_type – tip podatka u prijemnom baferu
rank-rang root procesa koji šalje podatke

Operacije za prenos podataka



Operacije za prenos podataka

```
void main(int argc, char *argv[]) {
        int rank,size,i;
        double param[8],mine;
        int sndcnt,rcvcnt;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&size);
        rcvcnt=1;
        if(rank==3) {
                for(i=0;i<8;++i) param[i]=23.0+i;
                        sndcnt=1;
        }
        MPI_Scatter(param,sndcnt,MPI_DOUBLE,&mine,rcvcnt,MPI_DOU
        BLE,3,MPI_COMM_WORLD);

        printf("P:%d mine is %f \n",rank,mine);

        MPI_Finalize();
}
```

Operacije za prenos podataka

Izlaz za 8 procesa
P:0 mine is 23.000000
P:1 mine is 24.000000
P:2 mine is 25.000000
P:3 mine is 26.000000
P:4 mine is 27.000000
P:5 mine is 28.000000
P:6 mine is 29.000000
P:7 mine is 30.000000

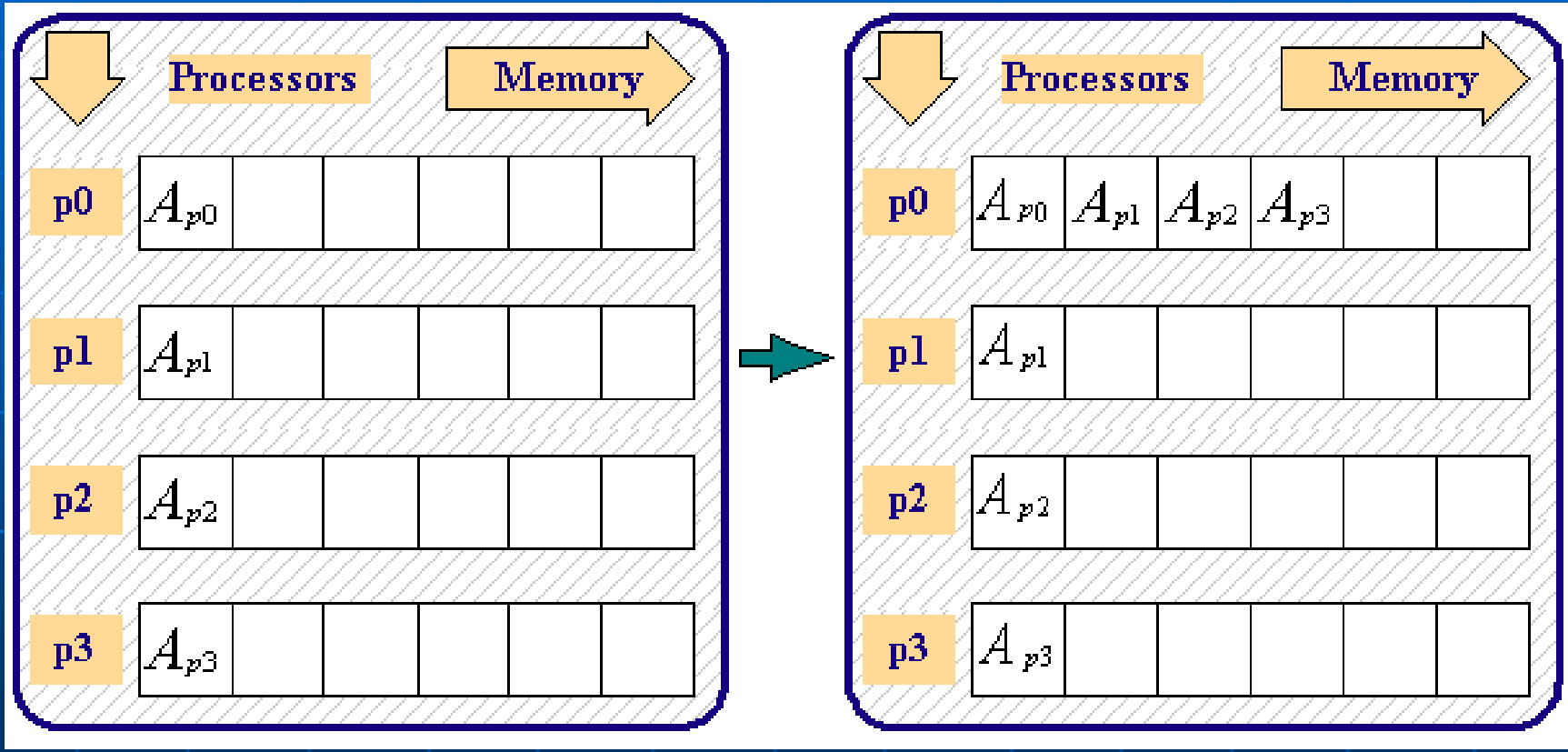
Operacije za prenos podataka

Operacija MPI_Gather omogućava da jedan proces sadržaj svog bafera formira kao skup podataka prikupljenih od ostalih procesa u datoj grupi. Tako je i-ti segment tog bafera preuzet od i-tog procesa.

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)

Proces prima podatke i skladišti ih na osnovu identifikatora procesa u toj grupi. Podaci iz send _bafer-a prvog člana grupe biće iskopiran u prvih recv_count lokacija bafera recv_buffer, podaci iz send_buffer-a drugog procesa u grupi biće iskopiran u sledećih recv_count lokacija i tako redom.

Operacije za prenos podataka



Operacije za prenos podataka

Pr.

```
#include   <stdio.h>
#include   <mpi.h>
void main(int argc, char *argv[])
{
        int rank,size;
        double param[16],mine;
        int sndcnt,rcvcnt;
        int i;
        MPI_Init(&argc, &argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        MPI_Comm_size(MPI_COMM_WORLD,&size);
        sndcnt=1;
        mine=23.0+rank;
        if(rank==7) rcvcnt=1;
        MPI_Gather(&mine,sndcnt,MPI_DOUBLE,param,rcvcnt,MPI_DOUBL
        E,7,MPI_COMM_WORLD);
        if(rank==7)
                for(i=0;i<size;++i) printf("PE:%d param[%d] is %f
        \n",rank,i,param[i]);
        MPI_Finalize();
}
```

Operacije za prenos podataka

Izlaz za 10 procesa:

```
PE:7 param[0] is 23.000000
PE:7 param[1] is 24.000000
PE:7 param[2] is 25.000000
PE:7 param[3] is 26.000000
PE:7 param[4] is 27.000000
PE:7 param[5] is 28.000000
PE:7 param[6] is 29.000000
PE:7 param[7] is 30.000000
PE:7 param[8] is 31.000000
PE:7 param[9] is 32.000000
```

Grupne operacije –zadaci

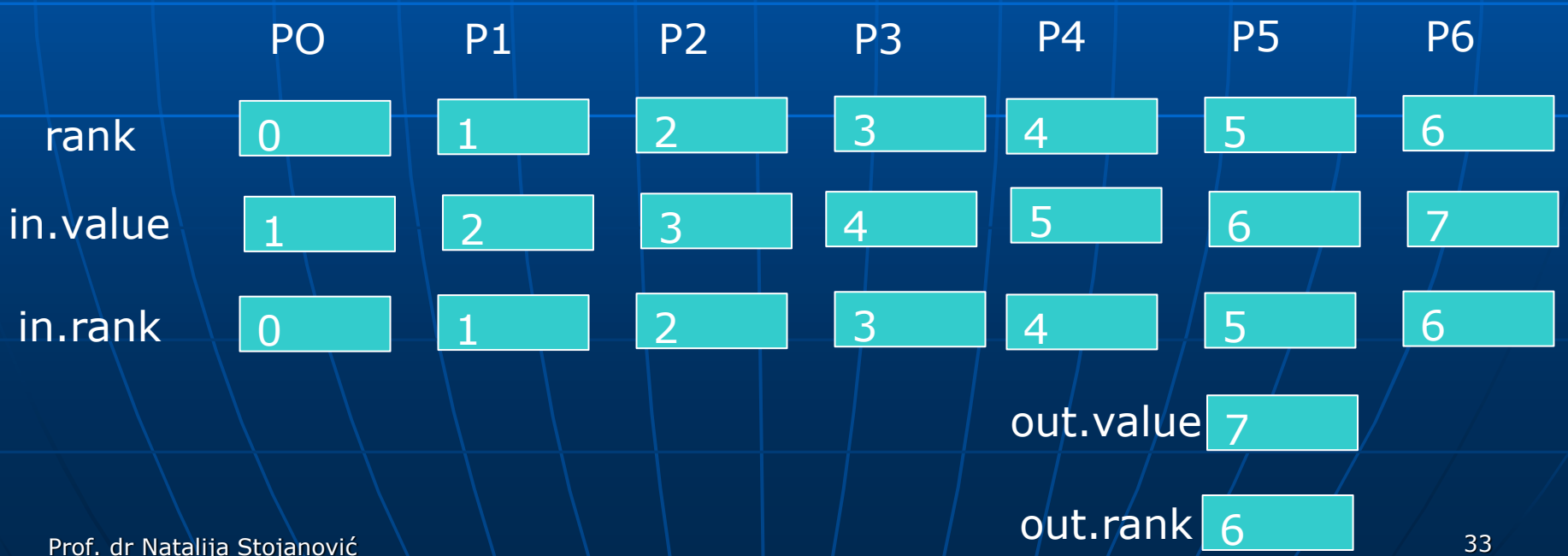
zad. Napisati MPI program koji nalazi minimalnu i maximalnu vrednost zadate promenljive za N procesa kao i identifikatore procesa koji sadrže te vrednosti.

```
#include <mpi.h>
void main (int argc, char *argv[]) {
int rank;
struct {
double value;
int rank;
} in, out;
int root;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
in.value=rank+1;
in.rank=rank;
root=5;
MPI_Reduce(&in,&out,1,MPI_DOUBLE_INT,MPI_MAXLOC,root,MPI_COMM_W
ORLD);
```

MPI_FLOAT_INT
MPI_DOUBLE_INT
MPI_2INT

Grupne operacije –zadaci

```
if(rank==root) printf("PE:%d max=%lf at rank%d\n", rank,
out.value,out.rank);
MPI_Reduce(&in,&out,1,MPI_DOUBLE_INT,MPI_MINLOC,root,MPI_COMM
_WORLD);
if(rank==root) printf("PE:%d min=%lf at rank%d\n", rank,
out.value,out.rank);
MPI_Finalize();
}
```



Grupne operacije –zadaci

zad. Svaki od N procesa sadrži 30 realnih brojeva. Napisati MPI program koji nalazi maksimalnu vrednost na svakoj od 30 lokacija, kao i identifikator procesa koji sadrži tu vrednost.

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

void main (int argc, char *argv[])
{

struct {
double val;
int rank;} in[30], out[30];
int i, myrank,size,sndcnt;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_rank(MPI_COMM_WORLD, &size);
```

Grupne operacije –zadaci

```
for (i=0; i < 30; i++)
{
in[i].val = double(myrank+i)
in[i].rank = myrank;
}
MPI_Reduce(in, out, 30, MPI_DOUBLE_INT, MPI_MAXLOC,0,
MPI_COMM_WORLD);
if (myrank == 0)
for (i=0 ; i < 30 ; i++) {
 printf("outval %f ",out[i].val);
 printf("outrank %d\n",out[i].rank);
}
MPI_Finalize();
}
```

Grupne operacije –zadaci