# Point-to-point

**int MPI\_Send(void \*buf, int count, MPI\_Datatype dtype, int dest, int tag, MPI\_Comm comm);**

gde buf ukazuje na mesto u memoriji odakle počinje slanje count podataka tipa dtype. Broj podataka u receive pozivu (MPI\_Recv) treba da bude veći ili jednak broju count u MPI\_Send. dest je rang procesa kome se šalje poruka, tag je proizvoljan broj koji služi za prepoznavanje odgovarajuče poruke na prijemu(taj broj mora da bude isti u MPI\_Send i MPI\_Recv koji razmenjuju istu poruku), comm-komunikator u okviru koga se odvija komunikacija.

**int MPI\_Recv(void \*buf, int count, MPI\_Datatype dtype, int source, int tag, MPI\_Comm comm, MPI\_Status \*status);**

Funkcija koja implementira prijem sa blokiranjem. buf ukazuje na mesto u memoriji od koje počinje poruka koja je primljena. Maksimalan broj podataka tipa dtype koji se prima određen je drugim i trećim argumentom count i dtype. Argument source je rang izvora poruke,tag oznaka poruke a comm komunikator u kojem moraju da budu oba procesa.

**int MPI\_Get\_count(MPI\_Status \*status, MPI\_Datatype datatype, int \*count )**

• status - status of receive operation

• datatype - datatype of each receive buffer element

• count - number of received elements

Tag of received message – status.MPI\_TAG

Rank of the sender – status.MPI\_SOURCE

# Point-to-point bez blokiranja

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()**.

**int MPI\_Isend (void \*buf, int count, MPI\_Datatype dtype, int dest, int tag, MPI\_Comm comm, MPI\_Request \*request);**

Iz funkcije MPI\_Isend() se vraća odmah, pre nego što poruka bude iskopirana u bafer. 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.

**int MPI\_Irecv (void \*buf, int count, MPI\_Datatype dtype, int source, int tag, MPI\_Comm comm, MPI\_Request \*request);**

Proces vrši prijem bez blokiranja, tj. inicira prijem. 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).

Funkcije koje se koriste za proveru kompletiranja operacija bez blokiranja su MPI\_Wait() i MPI\_Test():

**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**.

**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**.

# Grupne (collective) operacije

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: 1) operacije za kontrolu procesa, 2) operacije za globalna izračunavanja i 3) operacije za prenos podataka.

**1) 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.

**2) Operacije za globalna izračunavanja**

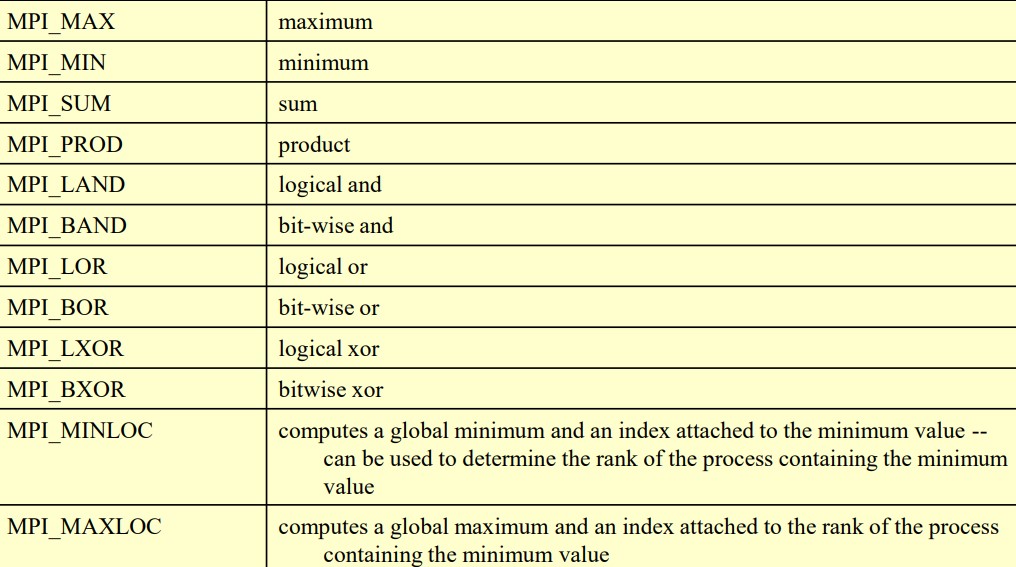
**int MPI\_Reduce ( void\* send\_buffer, void\* recv\_buffer, int count, MPI\_Datatype datatype, MPI\_Op operation, int rank, MPI\_Comm comm );**

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

Operacija redukcije uzima podatke u ulaznim baferima svih procesa, primenjuje nad njima datu operaciju redukcije i smešta rezultat u promenljivu root procesa.

operation-operacija redukcije koja može biti



**int MPI\_Scan ( void\* send\_buffer, void\* recv\_buffer, int count, MPI\_Datatype datatype, MPI\_Op operation, MPI\_Comm comm );**

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.

**3) Operacije za prenos podataka**

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

**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. MPI\_Bcast omogućava kopiranje podataka iz memorije root (rank) 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\_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.

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\_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.

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.

# Izvedeni tipovi podataka

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

**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

array\_of\_types - niz koji se sastoji od elemenata 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 uz pomoć 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

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)**

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

**int MPI\_Type\_commit (MPI\_datatype \*datatype)**

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

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

Tip se kreira opisom N-dimenzionalnog podpolja unutar Ndimenzionalnog 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

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.

Problem je taj što slanje sledećeg podatka tipa newtype kreće sa adrese koja je odmah iza poslednjeg poslatog bajta podatka (ne šalje i ostale elemente poslednjeg bloka koji nisu u blocklegth). Najčešće, ovo nije ono što mi želimo da bude poslato. Problem možemo rešiti korišćenjem funkcije

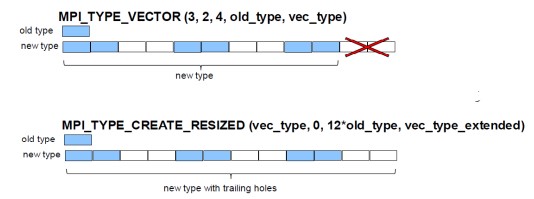
**int MPI\_Type\_create\_resized (MPI\_Datatype oldtype, MPI\_Aint lb, MPI\_Aint extent, MPI\_Datatype \*newtype)**

lb - nova donja granica tipa (najmanji pomeraj u novom tipu, uglavnom je jednak onom u starom tipu, tj. 0)

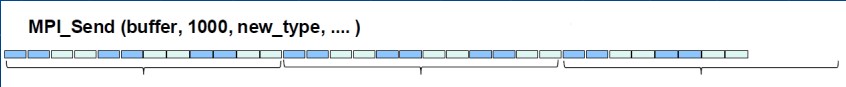
extent – nova veličina (u Bajtovima) tipa koja utiče na to odakle će krenuti slanje sledeće jedinice novog tipa

oldtype - stari tip

newtype - novi tip



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, ...).

# Grupe i komunikatori

**int MPI\_Comm\_group ( MPI\_Comm comm, MPI\_Group \*group )** -kao argument uzima komunikator i vraća odgovarajuću grupu procesa. Ova funkcija je važna za kasnije formiranje novih grupa od osnovne grupe

**int MPI\_Group\_rank ( MPI\_Group group, int \*rank )** –vraća identifikator procesa unutar grupe

**int MPI\_Group\_size (MPI\_Group group, int \*size)** –vraća broj procesa unutar grupe

**int MPI\_Group\_excl ( MPI\_Group group, int count, int \*nonmembers, MPI\_Group \*new\_group )**

- vraća novu grupu tako što se iz stare grupe isključe procesi sa identifikatorima koji su definisani sa nonmembers, i kojih ima count. Redosled procesa u novoj grupi prati redosled procesa u staroj grupi.

**int MPI\_Group\_incl ( MPI\_Group old\_group, int count, int \*members, MPI\_Group \*new\_group )**

-vraća novu grupu tako što procesi sa identifikatorima iz stare grupe koji su definisani sa members čine novu grupu. Procesa u novoj grupi ima count. Proces members[i] u novoj grupi ima rang i.

**int MPI\_Group\_intersection (MPI\_Group group1, MPI\_Group group2, MPI\_Group \*newgroup)**

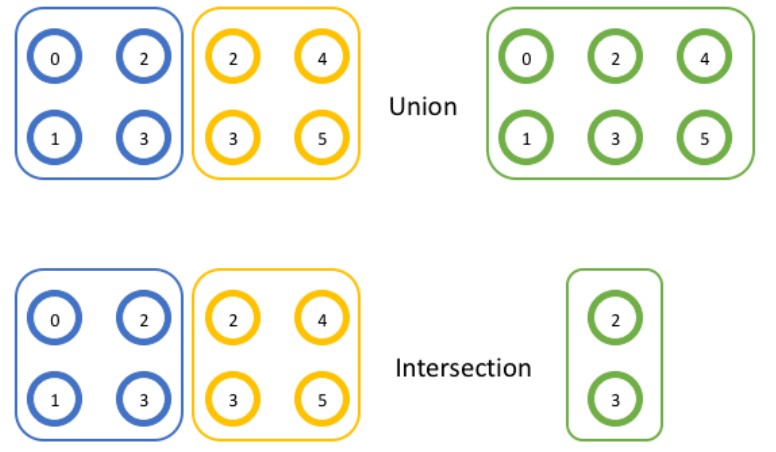
- vraća novu grupu koja se sastoji od procesa preseka grupa group1 i group2, s tim što je redosled procesa u novoj grupi kao u prvoj grupi

**int MPI\_Group\_union (MPI\_Group group1, MPI\_Group group2, MPI\_Group \*newgroup)**

-vraća novu grupu koja se sastoji od procesa grupe group1 na koju se nadovezuju elementi group2 koji nisu već u group1

**int MPI\_Group\_difference (MPI\_Group group1, MPI\_Group group2, MPI\_Group \*newgroup)**

-vraća novu grupu koja se sastoji od procesa grupe group1 koji nisu u group2, uređene kao u group1



**int MPI\_Comm\_create ( MPI\_Comm old\_comm, MPI\_Group, MPI\_Comm \* new\_comm)**

-kreira novi komunikator new\_comm od procesa iz grupe group koja je kreirana iz old\_comm

**int MPI\_Comm\_split ( MPI\_Comm old\_comm, int color, int key, MPI\_Comm \*new\_comm )**

-razbija komunikator old\_comm na više delova tako što svi procesi koji imaju istu vrednost za color pripadaju istom podkomunikatoru. Key određuje redosled procesa u podkomunikatorima. Ako je key isti za sve procese onda se redosled procesa preuzima iz starog komunikatora

