Multiprocesorski sistemi

MPI (Message Passing Interface)

Marko Mišić, Andrija Bošnjaković

13S114MUPS, 13E114MUPS,

MS1MPS, RI5MS, IR4MPS, SI4MPS

2016/2017.

Zasnovano na:
Blaise Barney, Lawrence Livermore National Laboratory
Message Passing Interface
https://computing.llnl.gov/tutorials/mpi/

Model prosleđivanja poruka (message passing)

- Procesi komuniciraju tako što međusobno razmenjuju poruke sa podacima i zahtevima
  - Eksplicitnom komunikacijom se vrši i sinhronizacija
- Najčešći je kod paralelnih računara sa distribuiranom memorijom
  - Procesne jedinice rade u odvojenim adresnim prostorima
  - Koristan za coarse-grain probleme
- Puno nekompatibilnih standarda u prošlosti

Uvod u MPI

- MPI (Message Passing interface) je standardizovana i portabilna specifikacija biblioteke za razmenu poruka
- MPI definiše rutine za pisanje paralelnih programa na programskim jezicima C i Fortran
- Podržan od neformalnog tela - MPI foruma
  - MPI-1 specifikacija (1994.)
  - MPI-2 specifikacija (1996.)
  - MPI-3 specifikacija (2012.)
- Većina implementacija podržava mešavinu funkcionalnosti iz navedenih specifikacija

Prednosti MPI

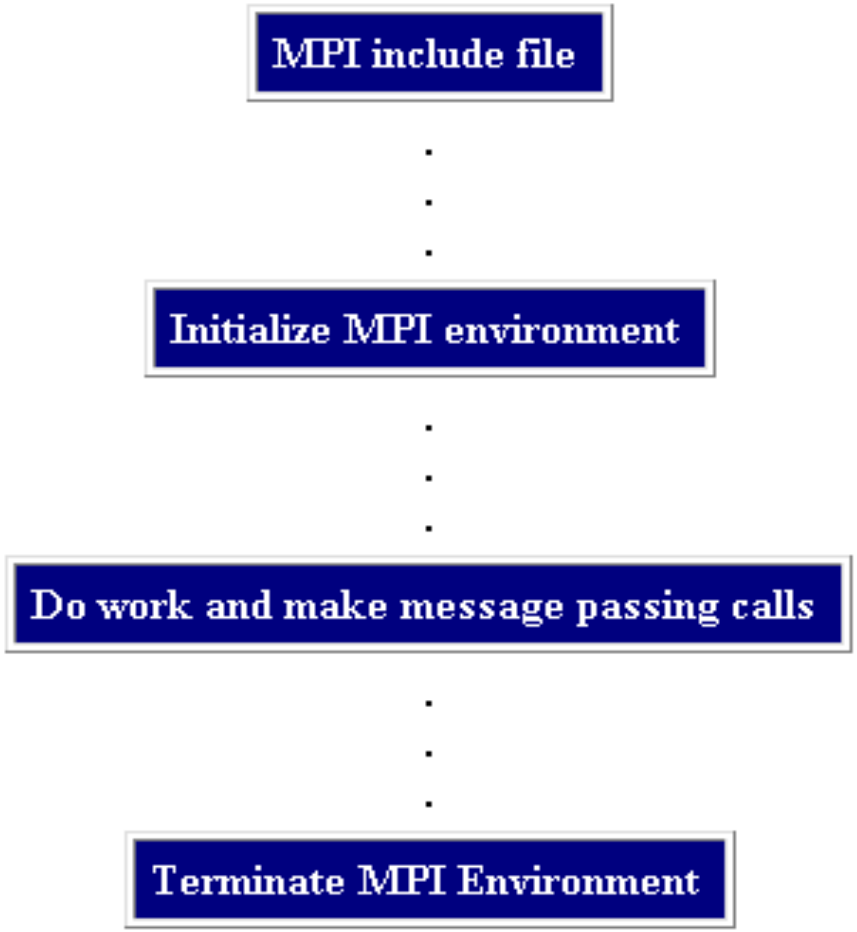
- Standardizacija
  - Jedini može biti smatran za standard i podržan je na skoro svim HPC platformama
- Prenosivost programskog koda
  - Nema potrebe za promenom izvornog koda pri prenosu aplikacija sa platforme na platformu
- Efikasnost
  - Pojedinačnim implementacijama se ostavlja mogućnost da koriste prednosti sistema domaćina i vrše optimizacije
- Funkcionalnost
  - MPI-3 podržava preko 430 različitih rutina
    - Većina programa se može napisati samo sa nekoliko
- Implementacije na raznim platformama i jezicima
  - Unix, Linux, Windows
  - Fortran, C i podrška za druge jezike

Osnovne osobine MPI

- Ciljne arhitekture za implementaciju
  - Distriburiana memorija
  - Deljena memorija
  - Hibridni pristupi
- Programski model prilagođen za distribuiranu memoriju
- Paralelizam je eksplicitan
  - Programer je odgovoran za njegovu identifikaciju i ostvarenje korišćenjem MPI rutina
- Broj procesa se tipično definiše prilikom pokretanja programa
- Može biti upotrebljen i za implementaciju drugih modela paralelnog procesiranja na sistemima sa distribuiranom memorijom
  - Data Parallel

Upotreba MPI u najkraćim crtama (1)

- `#include "mpi.h“`
- Format MPI poziva:
  - C
    `rc = MPI_Xxxxx(…)`
    - Kod greške će biti smešten u rc
  - C++
    `MPI::Xxxxx(…)`
    - Rukovanje greškama putem izuzetaka
- C++ biblioteka je praktično omotač oko C biblioteke



Upotreba MPI u najkraćim crtama (2)

- Uobičajeno se `mpi.h` uključuje pre drugih zaglavlja
- Skoro svi MPI pozivi vraćaju vrednost koja daje informaciju o uspešnosti poziva
  - Na osnovu te vrednosti može biti određen uzrok greške
  - Podrazumevano ponašanje ako dođe do greške je prekid rada
    - Programer ovo može urediti drugačije
- Nadalje je izostavljen `int` sa početka svih C prototipa
- Svi MPI identifikatori u C počinju sa `MPI_`

Organizacija MPI/C

```
int MPI_Init(...);
int MPI_Finalize(...);
int MPI_Send (...);
int MPI_Recv (...);
int MPI_Bcast(...);
int MPI_Reduce(...);
int MPI_Group_size(...);
int MPI_Group_rank(...);
int MPI_Comm_rank (...);
int MPI_Comm_size (...);
int MPI_Type_commit(...);
int MPI_Type_contiguous(...);
int MPI_Type_free(...);
```

Podrška za C++

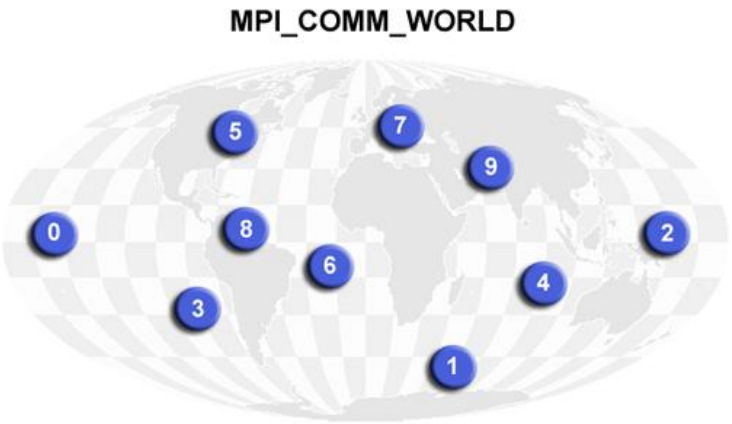
- Postojala do MPI-3 standarda kroz omotačke funkcije
- U jeziku C++, povratne vrednosti MPI poziva nemaju informaciju o uspešnosti poziva
  - Greške bivaju obrađene korišćenjem mehanizma izuzetaka
- Tamo gde je u C prenos preko pokazivača, u C++ je prenos preko reference (INOUT parametri)
- Tamo gde je samo jedan OUT parametar, u C++ je taj parametar povratna vrednost
- Svi MPI identifikatori u u C++ svi su smešteni u `MPI namespace`

Organizacija MPI/C++

```
namespace MPI {
  class Comm                                {...};
  class Intracomm : public Comm             {...};
  class Graphcomm : public Intracomm        {...};
  class Cartcomm  : public Intracomm        {...};
  class Intercomm : public Comm             {...};
  class Datatype                            {...};
  class Errhandler                          {...};
  class Exception                           {...};
  class Group                               {...};
  class Op                                  {...};
  class Request                             {...};
  class Prequest  : public Request          {...};
  class Status                              {...};
};
```

Komunikatori i grupe

- MPI koristi posebne objekte (grupe i komunikatore) da definiše koji procesi mogu međusobnog da komuniciraju
  - Sva razmena poruka ide preko njih
- Za većinu primena je dovoljan osnovni komunikator



- Proces može istovremeno biti član više grupa i više komunikatora
- Sve metode koje imaju veze sa komunikacijom zahtevaju komunikator kao jedan od svojih argumenata

Rang MPI procesa

- Unutar komunikatora,
  svaki proces ima jedinstveni rang (engl. rank),
  - Rang uzima vrednosti od 0 do brProc-1
- Rang procesa je vezan za komunikator
  - Proces ima onoliko rangova
    koliko je komunikatora kojima pripada
  - Određuje ih biblioteka
- Rang određuje izvorište i/ili odredište poruke
- Rang olakšava posvećivanje procesa
  - ako je rang jednak X, radi ovo,
    ako je rang jednak Y, radi ono…
  - ako je proces gospodar, kontroliši sluge,
    ako je proces sluga, radi i ništa ne pitaj ☺

Početak rada sa MPI svetom

- Početak (inicijalizacija MPI sveta)
  - `MPI_Init(int* argc, int*** argv)`
    `void MPI::Init(int& argc, char**& argv)`
  - Ponekad (nije po standardu)
    - `MPI_Init()`
    - `void MPI::Init()`
- Šalje argumente komandne linije do svih procesa u MPI svetu
- Prvi MPI poziv u svakom MPI programu
  - Sme biti pozvana samo jednom
- Utvrđivanje da li je obavljena inicijalizacija MPI sveta
  - `MPI_Initialized(&flag)`
    `bool MPI::Is_initialized()`

Kraj rada sa MPI svetom

- Kraj sveta
  - `MPI_Finalize()`
    `MPI::Finalize()`
  - Kraj celog MPI sveta
  - Poslednji MPI poziv u svakom MPI programu
- Kraj (dela) sveta
  - `MPI_Abort(comm,errorcode)`
    `MPI::Abort(comm,errorcode)`
  - Kraj dela MPI sveta koji je obuhvaćen datim komunikatorom
  - Većina implementacija ovo obradi
    kao kraj celog MPI sveta

Stanje MPI sveta

- Broj procesa u komunikatoru
  - `MPI_Comm_size(comm,&size)`
    `int Comm::Get_size() const`
  - Vraća broj procesa obuhvaćenih komunikatorom
  - Najčešće se koristi sa `MPI_COMM_WORLD` radi utvrđivanja broja procesa u MPI svetu
- Raspored procesa u komunikatoru
  - `MPI_Comm_rank(comm,&rank)`
    `int Comm::Get_rank() const`
  - Vraća jedinstveni redni broj procesa u komunikatoru
  - Isti proces može imati više rangova
    - Po jedan za svaki komunikator kome pripada

Podrška za višenitne programe

- MPI definiše četiri nivoa podrška za programe sa više konkurentnih grana izvršavanja:
  - `MPI_THREAD_SINGLE`
    - Samo jedna nit će se izvršavati u okviru programa
  - `MPI_THREAD_FUNNELED`
    - Proces može sadržati više niti, ali će samo glavna nit izvršavati MPI pozive
  - `MPI_THREAD_SERIALIZED`
    - Proces može sadržati više niti, ali će se MPI pozivi serijalizovati
  - `MPI_THREAD_MULTIPLE`
    - Više niti istovremeno može obavljati MPI pozive
- Funkcija koja se koristi za postavljanje nivoa podrške:

```
MPI_Init_thread
(int *argc, char ***argv,
 int required, int *provided)
```

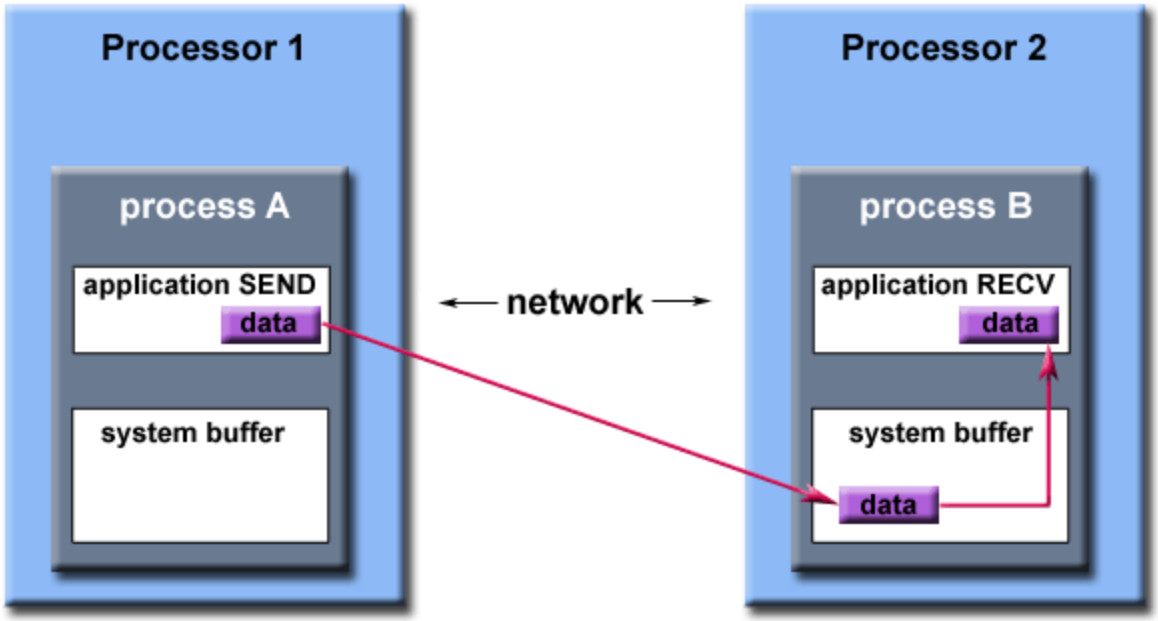
Komunikacija između dva procesa (point-to-point communication)

- Obuhvata razmenu poruka između **tačno dva** procesa:
  - Prvi proces obavlja slanje
  - Drugi proces obavlja **odgovarajući** prijem
- Postoji nekoliko tipova ovakve komunikacije
  - Sinhrono slanje
  - Blokirajuće slanje/prijem
  - Neblokirajuće slanje/prijem
  - Baferisano slanje
  - Kombinovano slanje i prijem
  - Slanje spremnom primaocu
- Moguća su uparivanja različitih slanja i prijema
- Postoje i funkcije za čekanje na prijem poruke ili ispitivanje da li je poruka stigla

Baferisanje (1)

- U savršenom slučaju,
  slanje je savršeno uklopljeno sa prijemom
- Šta ako:
  - dođe do slanja pre nego je primalac spreman?
  - istom procesu stigne više poruka odjednom?
- Implementacija MPI rešava ovakve i slične situacije
  - Rešenje nije propisano standardom
- Najčešća je upotreba sistemskog bafera
  - Njime upravlja MPI biblioteka
  - Ograničene je veličine
- Treba razlikovati aplikativni i sistemski bafer
  - Aplikativni bafer čine korisnički podaci
  - Sistemski bafer je transparentan za korisnika

Baferisanje (2)



Path of a message buffered at the receiving process

- Asinhrone operacije mogu poboljšati performanse programa
- Većina point-to-point funkcija može biti i blokirajuća i neblokirajuća
  - Neblokirajuće operacije nameću postojanje sistemskog bafera

Blokirajuća komunikacija

- Blokirajuće slanje se završava tek kada je bezbedno modifikovati poslate korisničke podatke
  - Podaci koje primalac dobija su istovetni podacima na početku slanja
- Ovo ne znači da je primalac primio podatke:
  - Blokirajuće slanje može biti sinhrono (kada primalac poruke vrati potvrdu uspešnog prijema)
  - Blokirajuće slanje može biti asinhrono (ako su podaci negde u sistemskom baferu)
- Blokirajući prijem se završava tek po kopiranju svih poslatih podataka u korisničke bafere primaoca

Neblokirajuća komunikacija

- I slanje i prijem se vraćaju odmah posle poziva, bez čekanja na prenos poruke i potvrdu o prijemu
- Ovakve operacije samo traže da MPI obavi to što je traženo jednom kad se steknu uslovi
- Postoje MPI funkcije koje funkcije koje proveravaju da li su slanje ili prijem stvarno završeni
  - Nije bezbedno da menjati korisničke podatke pre nego se komunikacija stvarno završi
- Neblokirajuće metode se najčešće koriste za preklapanje obrade i komunikacije
  - Ako je paket stigao, obradi taj paket, u suprotnom, radi nešto drugo umesto da čekaš

Redosled poruka i prvenstvo prijema

- MPI garantuje da će poruke stizati redosledom kojim su poslate
  - Pod uslovom da unutar procesa koji komuniciraju nema više od jedne niti zadužene za komunikaciju
  - Ako pošiljalac pošalje poruke P1 i P2 tim redom istom primaocu, primalac će primiti P1 pre nego primi P2
  - Ako primalac očekuje dve poruke (izda dva poziva funkciji za prijem), i oba poziva očekuju istu poruku, prvi poziv će primiti poruku pre drugog poziva
- Ako dva procesa šalju jednu istu poruku trećem procesu, samo jedno slanje će se završiti
  - MPI ne garantuje pravičnost (eng. *fairness*)
  - Programer treba da spreči izgladnjivanje

MPI pozivi za komunikaciju između dva procesa

- Blokirajući
  `MPI_SEND(buffer,count,type,dest,tag,comm)`
  `MPI_RECV(buffer,count,type,source,tag,comm,status)`
- Blokirajući sinhroni
  `MPI_SSEND(buffer,count,datatype,dest,tag,comm,ierr)`
- Blokirajući baferisani
  `MPI_BSEND(buffer,count,datatype,dest,tag,comm,ierr)`
- Neblokirajući
  `MPI_ISEND(buffer,count,type,dest,tag,comm,request)`
  `MPI_IRECV(buffer,count,type,source,tag,comm,request)`
- Istovremeno slanje i prijem
  `MPI_SENDRECV(sendbuf,sendcount,sendtype,dest,sendtag,`
  `recvbuf,recvcount,recvtype,source,recvtag,comm,status)`

Argumetni MPI poziva za komunikaciju između dva procesa (1)

- Buffer
  - Korisnički prostor koji se koristi za slanje/prijem podataka
- Data count
  - Broj elemenata odgovarajućeg tipa podataka za slanje/prijem
- Data type
  - Prost ili izveden MPI tip podatka
- Destination
  - Navodi se kao rang procesa koji prima podatke
- Source
  - Navodi se kao rang procesa koji šalje podatke
  - `MPI_ANY_SOURCE` se može koristiti za prijem poruke od bilo kojeg pošiljaoca

Argumetni MPI poziva za komunikaciju između dva procesa (2)

- Tag
  - Ceo broj koji jedinstveno identifikuje poruku
  - Poruka neće biti primljena ukoliko se ovo polje ne poklapa
  - `MPI_ANY_TAG` se može koristiti za prijem poruke sa bilo kakvim tagom
- Communicator
  - Objekat preko koga se vrši komunikacija
  - Uobičajeno `MPI_COMM_WORLD`
- Status
  - Objekat poziva za prijem tipa `MPI_Status`
  - Omogućava dohvatanje podataka o primljenoj poruci
    - source, tag i sl.
- Request
  - Stvara se kod neblokirajućih operacija slanja i prijema
  - Koristi se kod rutina koje vrše proveru da li su slanje ili prijem završeni

MPI pozivi za ispitivanje stanja komunikacije između dva procesa

- Čekanje na kompletiranje neblokirajućeg poziva

```
MPI_WAIT(request,status)
MPI_WAITANY(count,array_of_requests,index,status)
MPI_WAITALL(count,array_of_requests,array_of_statuses)
MPI_WAITSOME(incount,array_of_requests,outcount,
             array_of_offsets,array_of_statuses)
```

- Ispitivanje da li je poruka primljena

```
MPI_PROBE(source,tag,comm,status)
MPI_IPROBE(source,tag,comm,flag,status)
```

- Ispitivanje da li je prijem završen

```
MPI_TEST(request,flag,status)
MPI_TESTALL(count,array_of_requests,flag,
            array_of_statuses)
```

Stalna komunikacija (1)

- Pojedine primene zahtevaju da dva procesa često komuniciraju koristeći iste argumente
  - Na primer, kod obrade slike, da bi razmenili granične elemente
- U takvim slučajevima, pogodno je otvoriti stalni komunikacioni kanal
  - Na taj način se smanjuje režijsko vreme komunikacije
    - Kroz smanjivanje vremena potrebnog za pripremu poruke
    - Poruka se pripremi prvi put, a zatim se menja telo

Stalna komunikacija (2)

- Iniciranje stalne komunikacije

  `MPI_Send_init(buf,count,datatype,dst,tag,comm,req)`

  `MPI_Recv_init(buf,count,datatype,src,tag,comm,req)`

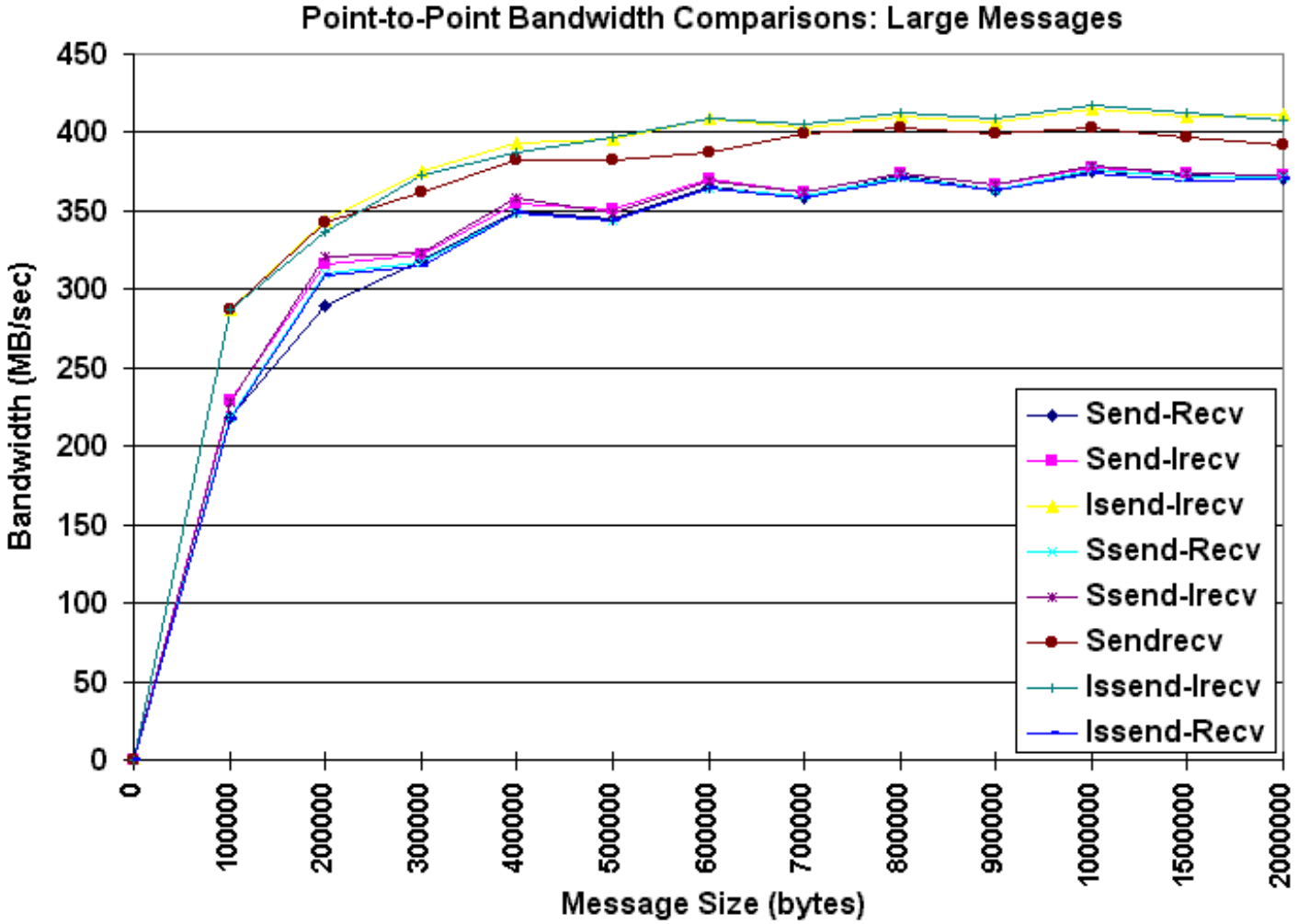
- Vršenje komunikacije

  `MPI_Start(req)`

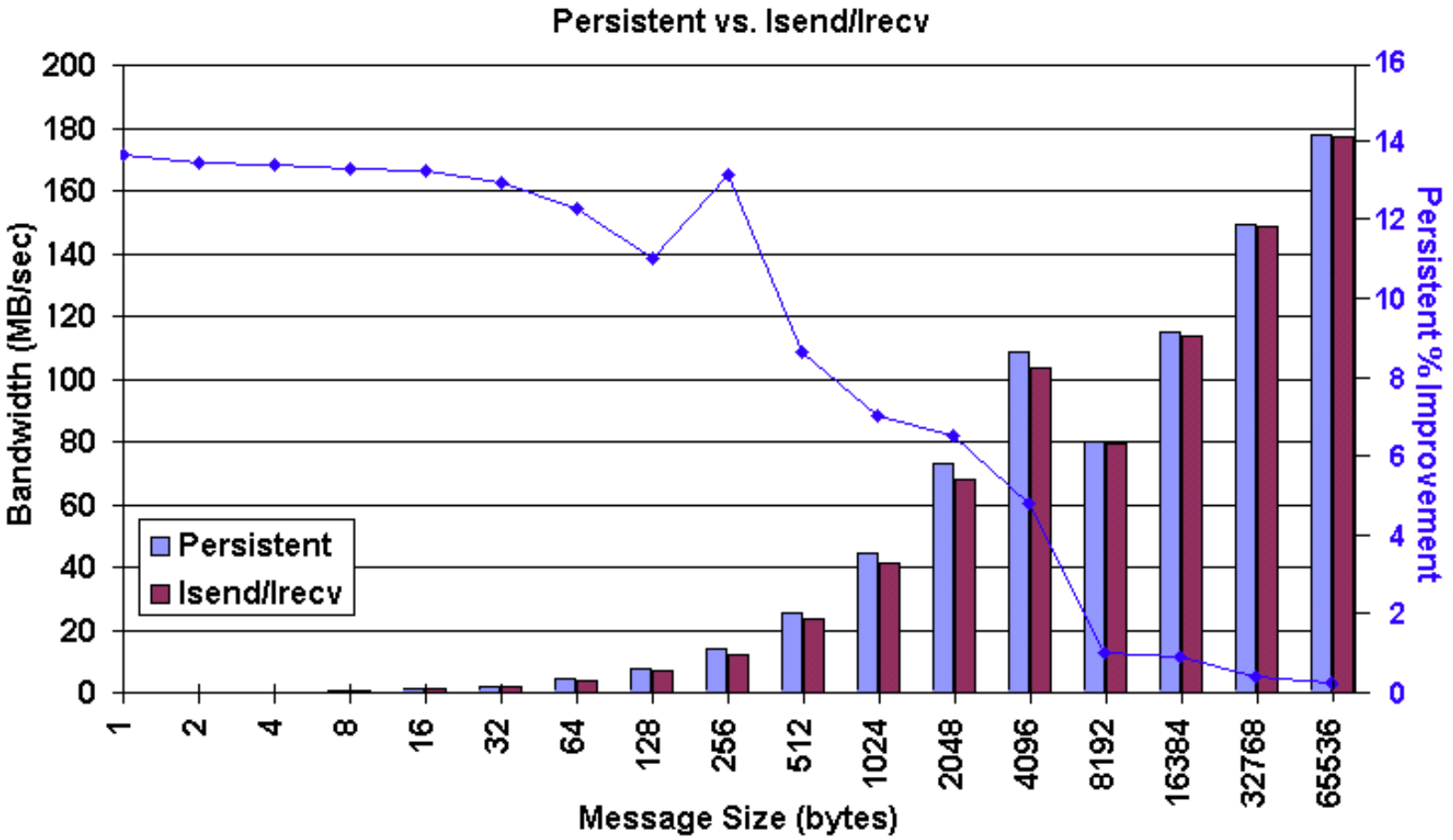
  `MPI_Startall(count,req[])`

- Čekanje da se komunikacija završi
  - Standardne `MPI_Wait()` funkcije
- Oslobađanje objekta zahteva
  - Poziv `MPI_Request_free()`

Poređenje različitih kombinacija za point-to-point komunikaciju (1)



Poređenje različitih kombinacija za point-to-point komunikaciju (2)



Komunikacija svih procesa komunikatora (collective communication)

- Pozivi kolektivnim operacijama se odnose na sve procese u zadatom komunikatoru
  - Programerova je odgovornost da svi procesi učestvuju u ovim operacijama
- Vrste kolektivne komunikacije:
  - Sinhronizacija
    - Proces čeka dok svi u datoj grupi ne postignu odgovarajuću tačku u programu `MPI_BARRIER(comm)`
  - Razmena podataka
    - broadcast, scatter/gather, all to all
  - Kolektivna obrada (reductions)
    - Jedan proces skuplja potrebne podatke od ostalih i obavlja određenu operaciju nad prikupljenim podacima
    - Računanje minimuma, maksimuma, sumiranje, logičke operacije

Osobine kolektivne komunikacije

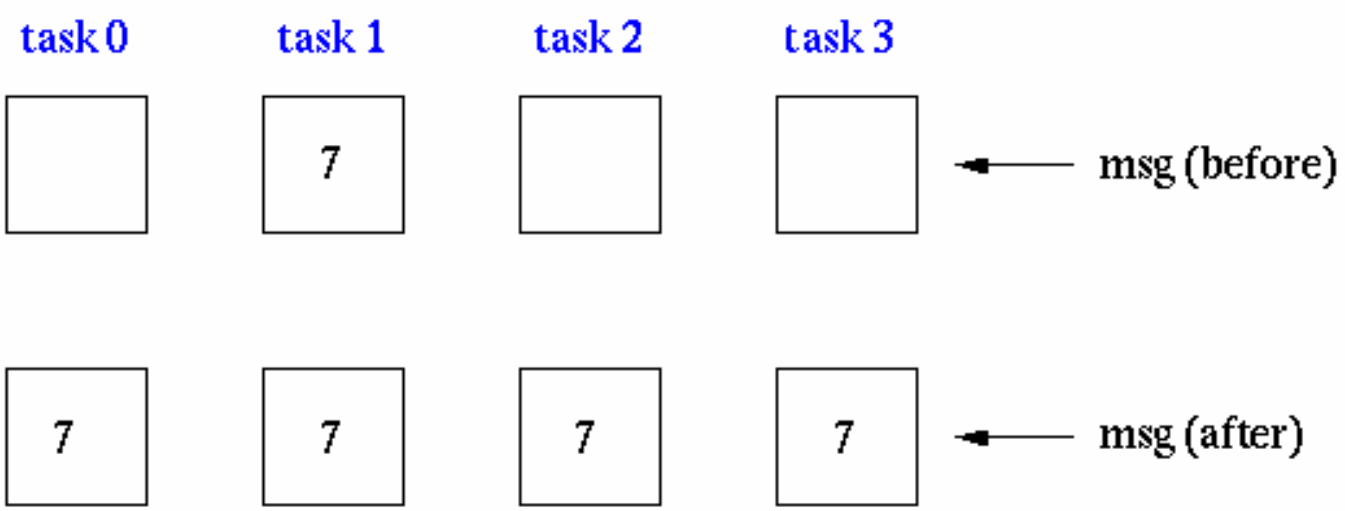
- Komunikacija obavezno uključuje sve procese
  - Kolektivna komunikacija između određenog broja procesa je moguća samo stvaranjem nove grupe i novog komunikatora
- Pozivi kolektivnim operacijama su obično blokirajući
  - MPI-3 standardi uvode i neblokirajuće pozive
- Pojedinačne poruke se ne mogu obeležavati tagovima
- U kolektivnim operacijama se mogu koristi samo prosti MPI tipovi
- Kolektivne operacije nisu uvek pogodne za vremenski kritične aplikacije
  - MPI standard ne propisuje način njihove implementacije

Slanje istog podatka svima u grupi

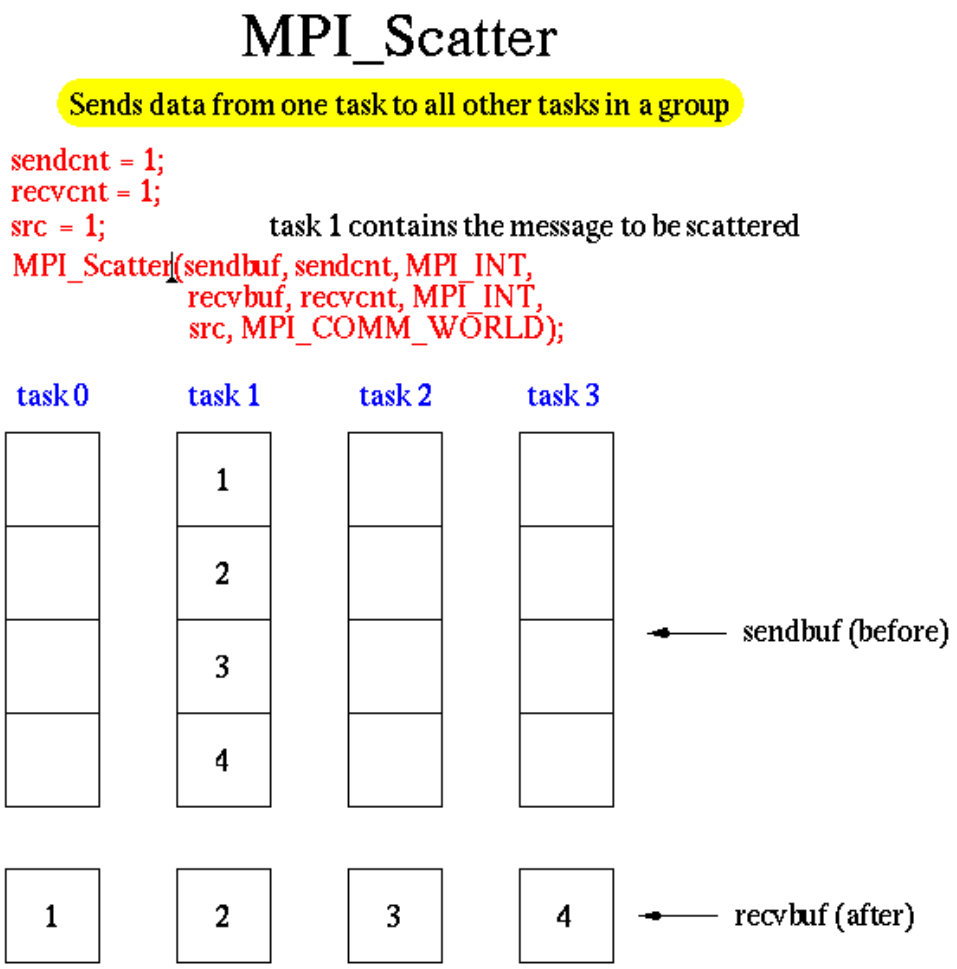
MPI_Bcast

Broadcasts a message to all other processes of that group

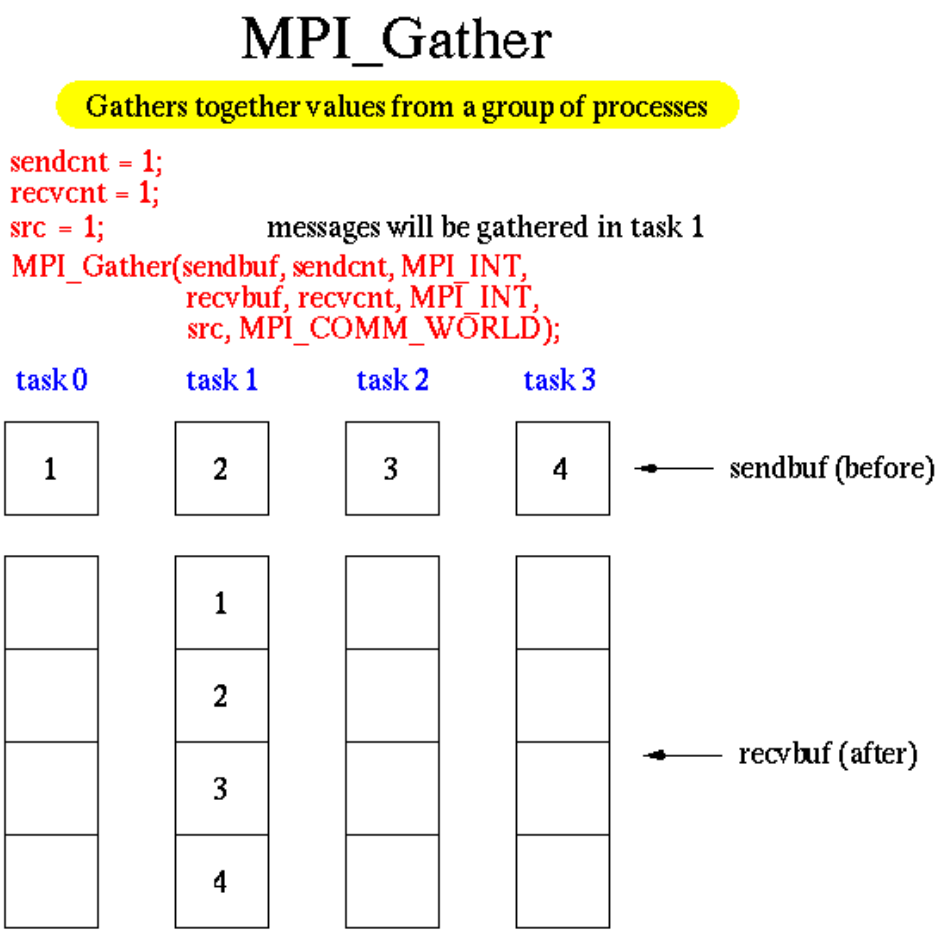
```
count = 1;
source = 1;                    broadcast originates in task 1
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```



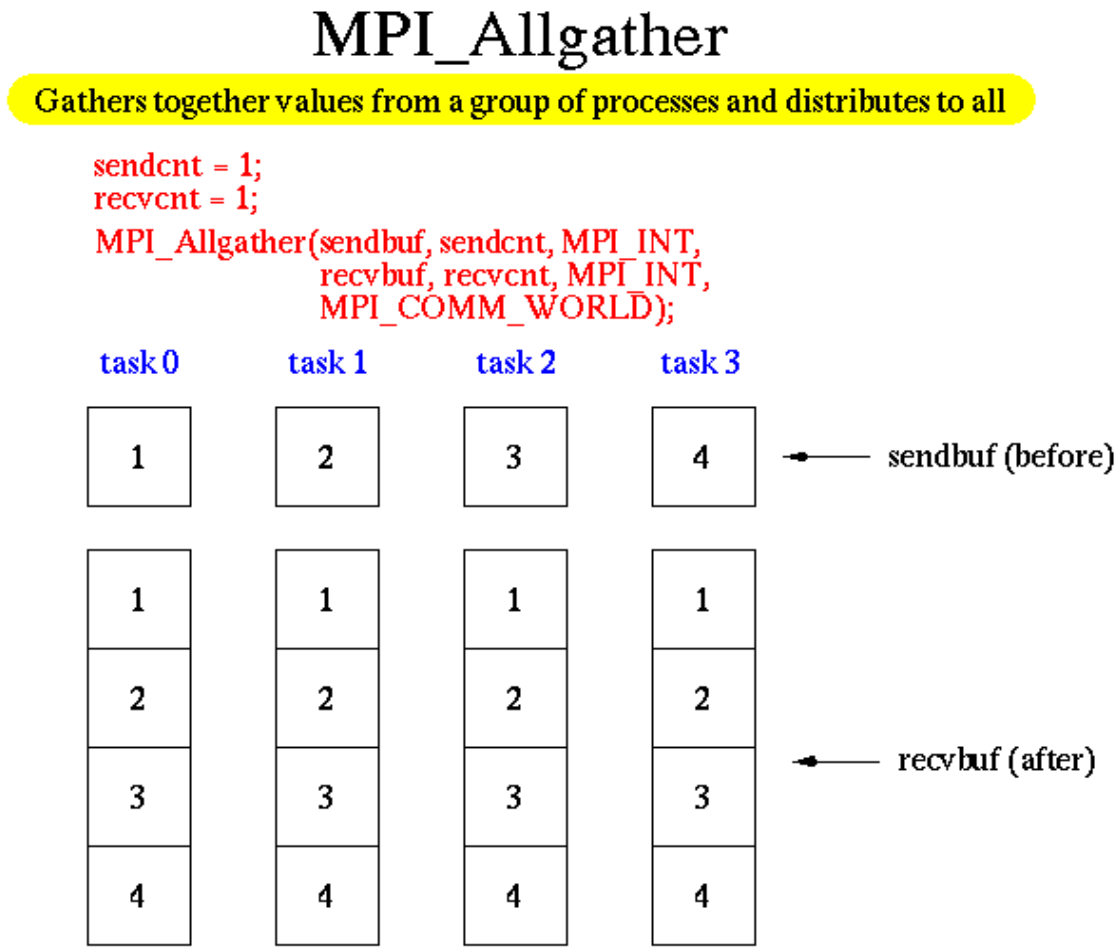
Podela paketa podataka unutar grupe



Objedinjavanje podataka od svih u grupi



Objedinjavanje podataka i slanje svima

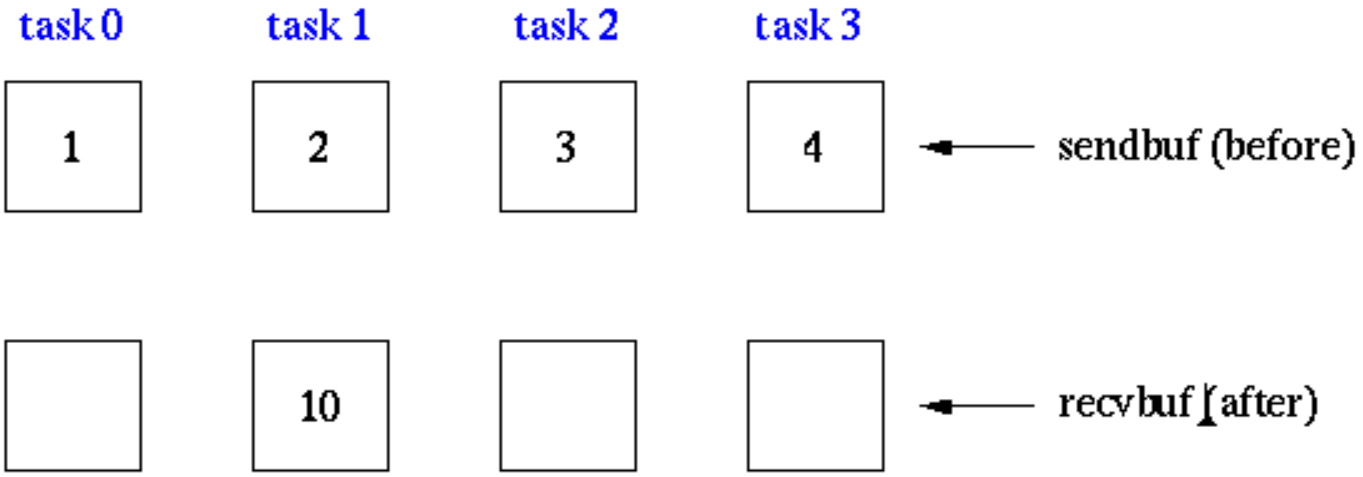


Svodna obrada podataka iz grupe

MPI_Reduce

Perform and associate reduction operation across all tasks in the group and place the result in one task

```
count = 1;
dest = 1;                    result will be placed in task 1
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
           dest, MPI_COMM_WORLD);
```

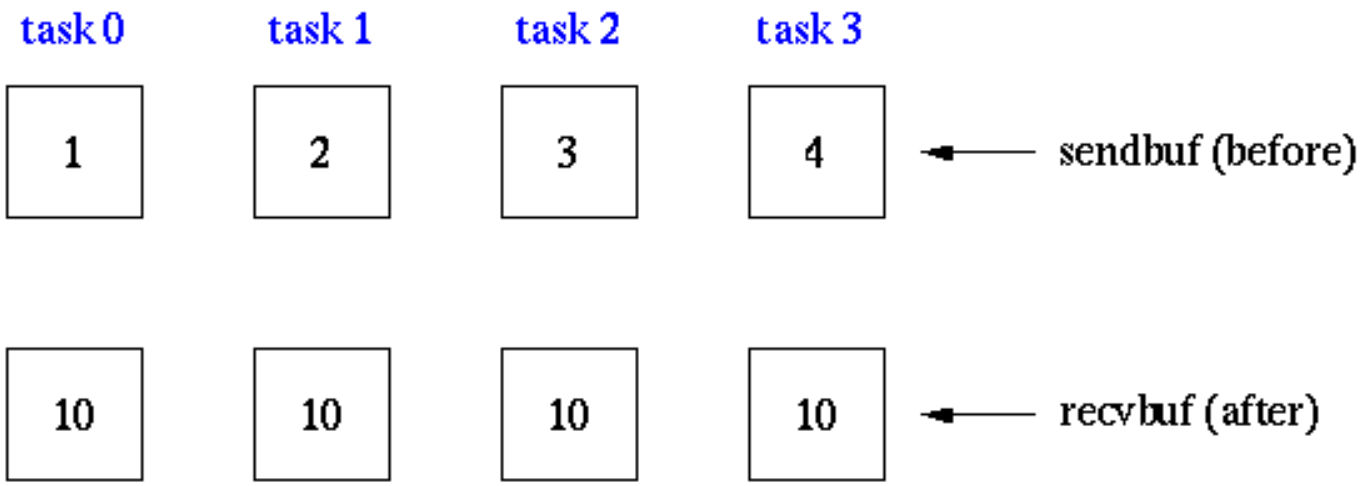


Svodna obrada podataka iz grupe i slanje rezultata svim procesima

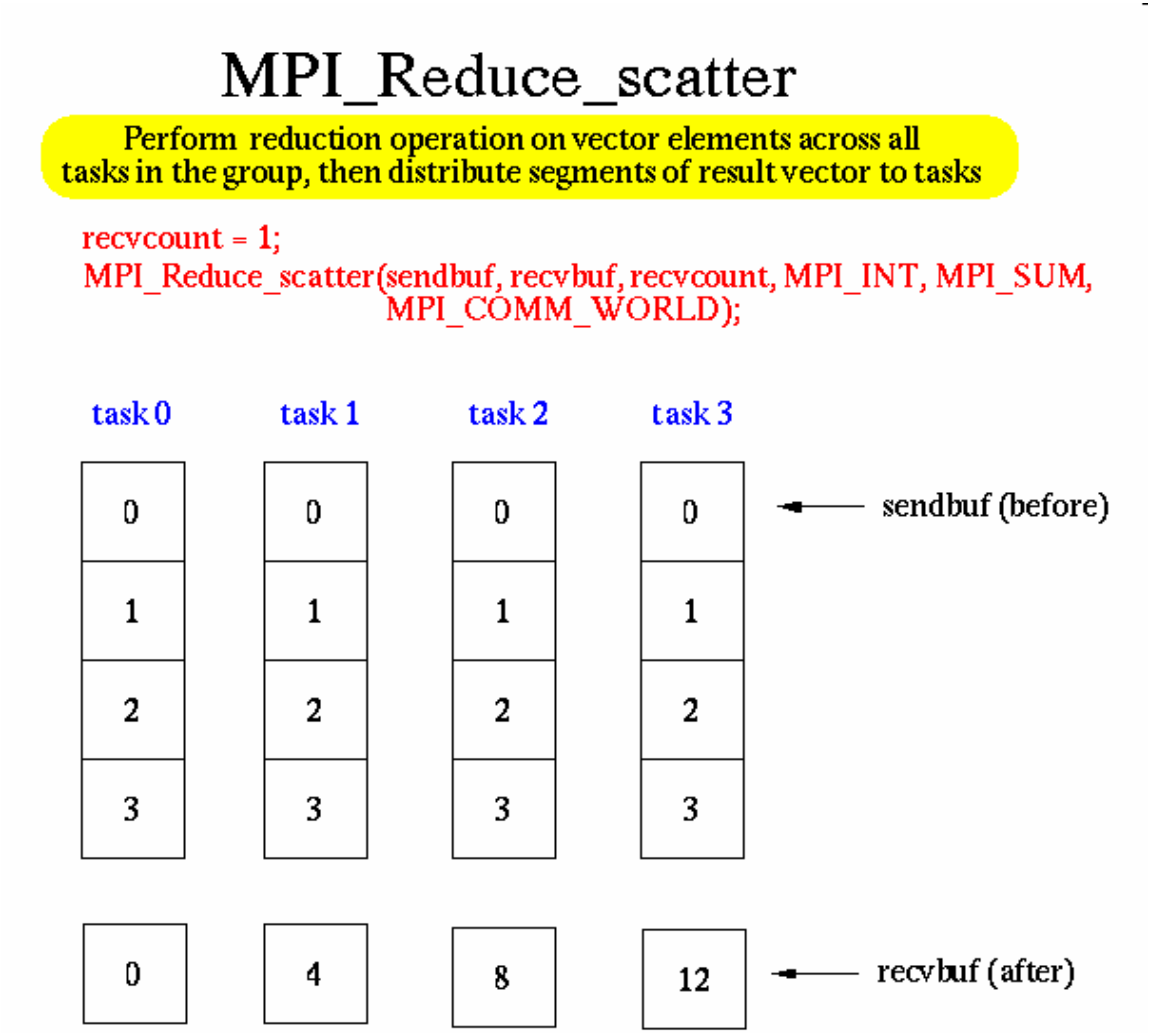
MPI_Allreduce

Perform and associate reduction operation across all tasks in the group and place the result in all tasks

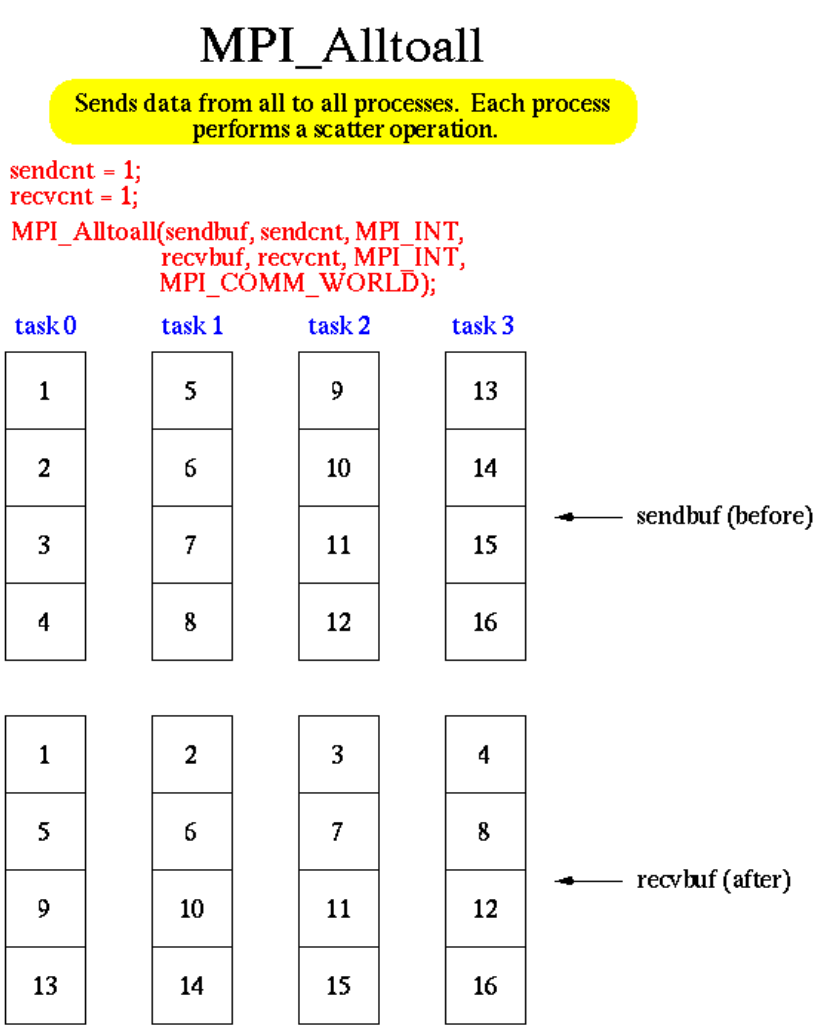
```
count = 1;
MPI_Allreduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
              MPI_COMM_WORLD);
```



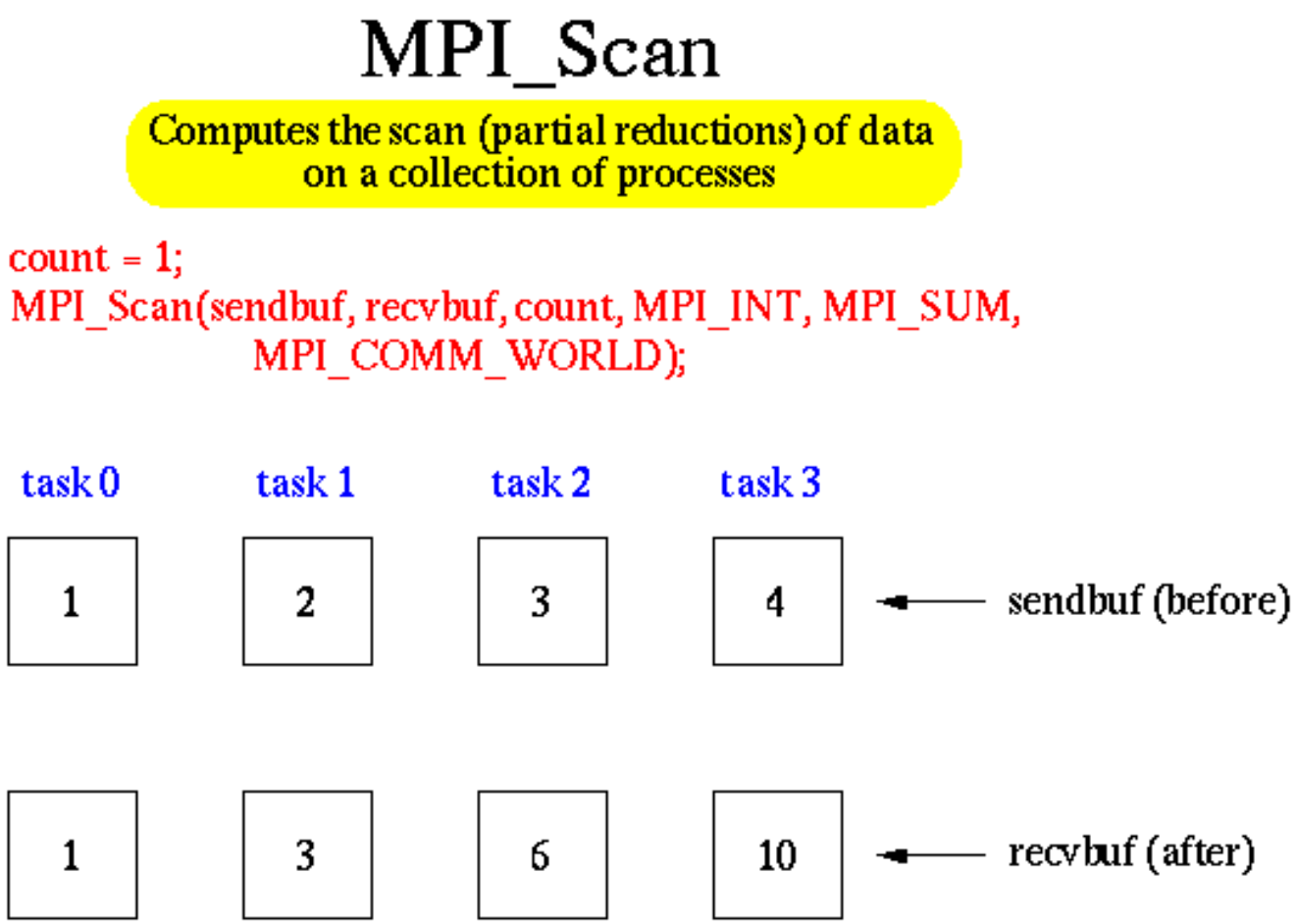
Svodna obrada podataka iz grupe nakon podele podataka po procesima



Grupna podela paketa po grupi (svaki proces šalje paket svakom procesu)



Delimična svodna obrada podataka iz grupe



Osnovni MPI tipovi podataka za jezike C i C++

MPI_CHAR	signed char
MPI_DOUBLE	double
MPI_FLOAT	float
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_DOUBLE	long double
MPI_LONG_LONG_INT	signed long long int
MPI_SHORT	signed short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_SHORT	unsigned short int
MPI_BYTE	8-bitni podaci
MPI_PACKED	Podaci spakovani sa MPI_Pack()/ MPI_Unpack

MPI izvedeni tipovi podataka (1)

- MPI dozvoljava konstruisanje korisničkih struktura podataka baziranih na prostim MPI tipovima
  - Prosti tipovi su uvek kontinualni (neprekidni) u memoriji
  - Izvedeni tipovi omogućavaju nekontinualnim i raznorodnim podacima da budu tretirani kao neprekidni
- Korisni uvek kad osnovni tipovi nisu dovoljni
  - Kada je potrebno opisati entitet složenih osobina
  - Kada je potrebno imati
    - Podatke istog tipa koji nisu neprekidni u memoriji
    - Neprekidne podatke različitih tipova
    - Podatke različitog tipa koji nisu neprekidni

MPI izvedeni tipovi podataka (2)

- Izvedeni tipovi povećavaju čitljivost i prenosivost programskog koda
- Omogućavaju da se izbegne režijsko vreme potrebno za slanje i prijem više kratkih poruka
  - Umesto toga, podaci se kombinuju u veću poruku i šalju/primaju kao jedinstvena poruka jednom Send/Recieve operacijom
- Režijsko vreme se može smanjiti i upotrebom MPI_PACKED ili MPI_BYTE tipova podataka
  - Treba upotrebljavati oprezno, pošto vodi mogućem padu performansi i smanjenju prenosivosti programskog koda
  - Razmotriti tek nakon testiranja konkretne MPI implementacije

Konstrukcija MPI izvedenih tipova

- Tipovi se realizuju na osnovu postojećih tipova, navođenjem strukture izvedenog tipa i dodavanjem izvedenog tipa u MPI okruženje
  - Deklariše se objekat tipa
  - Zadaje se opis odgovarajućim pozivom
  - Tip se postavlja u okruženje
  - Na kraju je potrebno uništiti objekat tipa
- Postoje četiri vrste izvedenih tipova:
  - Neprekidni (kontinualni)
  - Vektorski
  - Indeksirani
  - Strukturirani

Konstrukcija MPI izvedenih tipova (C)

- Deklaracija objekta za definiciju tipa
  ```
  MPI_Datatype ime_tipa;
  ```
- Priprema za konstrukciju objekta (po potrebi)
- Konstrukcija objekta (nakon pripreme)
  ```
  MPI_Type_xxx(Opis_tipa, &ime_tipa);
  ```
- Postavljanje tipa
  ```
  MPI_Commit(&ime_tipa);
  ```
- Upotreba
- Uništavanje tipa
  ```
  MPI_Type_free(&ime_tipa);
  ```

Konstrukcija MPI izvedenih tipova (C++)

- Deklaracija objekta za definiciju tipa
  ```
  MPI::Datatype imeTipa;
  ```
- Priprema za konstrukciju objekta (po potrebi)
- Konstrukcija objekta (nakon pripreme)
  ```
  imeTipa = MPI::OSNOVNI_TIP.Create_xxx(...);
  ```
- Postavljanje tipa
  ```
  imeTipa.Commit();
  ```
- Upotreba
- Uništavanje tipa
  ```
  imeTipa.Free();
  ```

Neprekidni tip

- Najjednostavniji tip
- Stvara novi tip tako što jednostavno nadoveže dati broj primeraka starog tipa

```
MPI_Type_contiguous (count,oldtype,&newtype)
MPI::Datatype noviTip =
    MPI::OSNOVNI_TIP.Create_contiguous(count)
```

- Primena u programima radi pojednostavljenja
  - Slanje/prijem celog niza jednom naredbom
  - Particionisanje većeg niza na manje delove i slanje/prijem tih delova jednom naredbom

MPI_ Type_contiguous

```
count = 4;
MPI_Type_contiguous(count, MPI_FLOAT, &rowtype);
```

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[2][0], 1, rowtype, dest, tag, comm);
```

9.0	10.0	11.0	12.0

1 element of rowtype

Vektorski tip

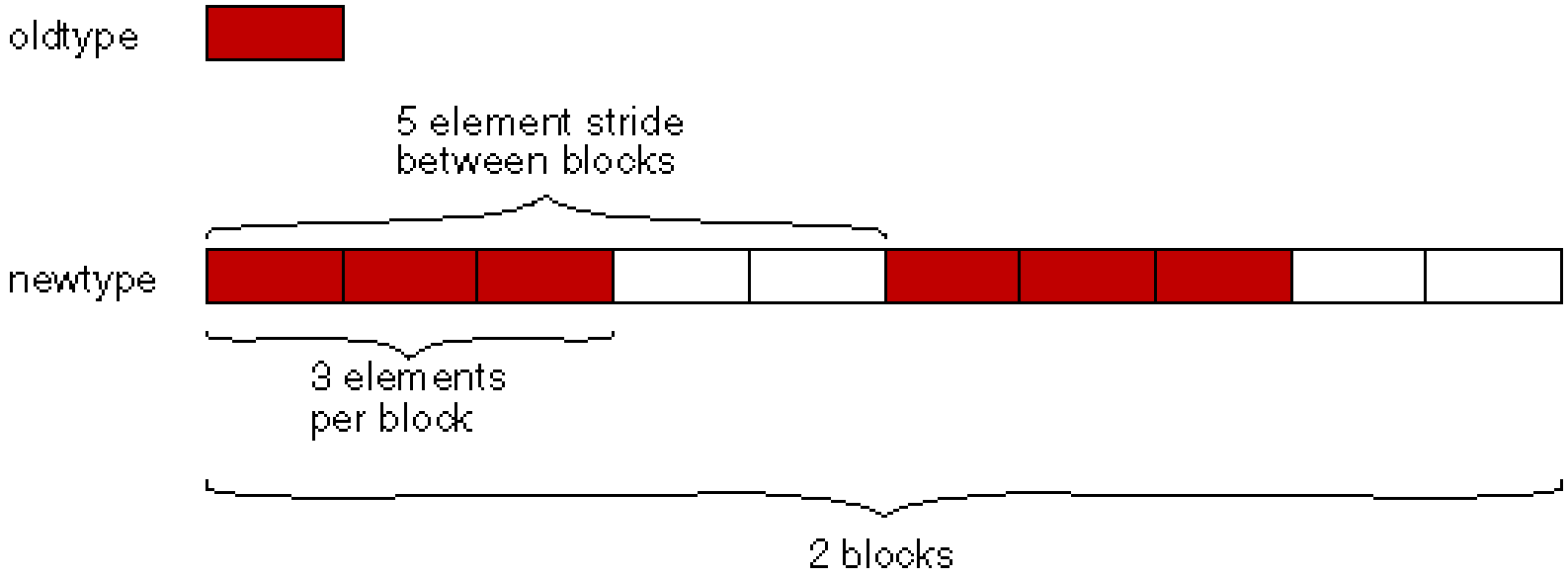
- Sličan je neprekidnom tipu
  (jedna naredba primenjena nad ovim tipom će poslati/primiti više podataka)
- Razlika je u proredu između podataka

```
MPI_Type_vector / MPI_Type_hvector
(count,blocklength,stride,oldtype,&newtype)
MPI::Datatype noviTip =
    MPI::OSNOVNI_TIP.
    Create_vector(count,blocklength,stride)
```

- `h` varijanta obezbeđuje poravnavanje adresa elemenata prema karakteristikama računara

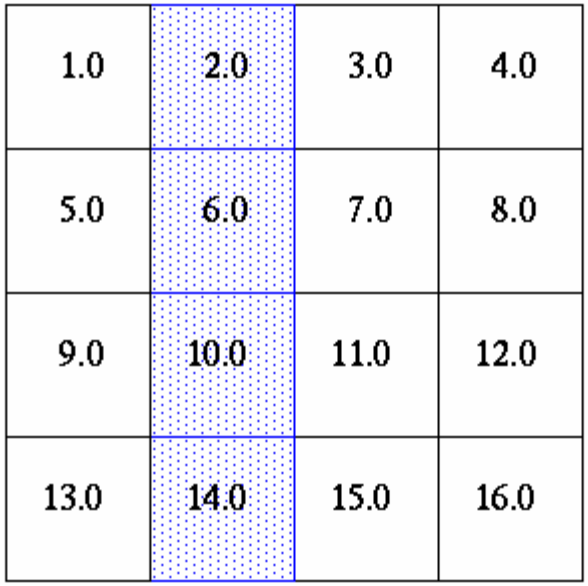
Primer deklaracije vektorskog tipa

- count = 2
- blocklength = 3
- stride = 5



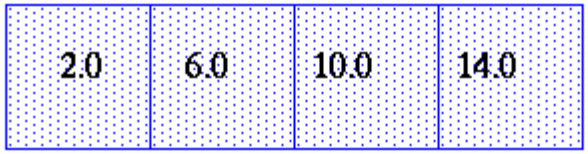
MPI_ Type_vector

count = 4; blocklength = 1; stride = 4;
MPI_Type_vector(count, blocklength, stride, MPI_FLOAT, &columntype);



a[4][4]

MPI_Send(&a[0][1], 1, columntype, dest, tag, comm);



1 element of columntype

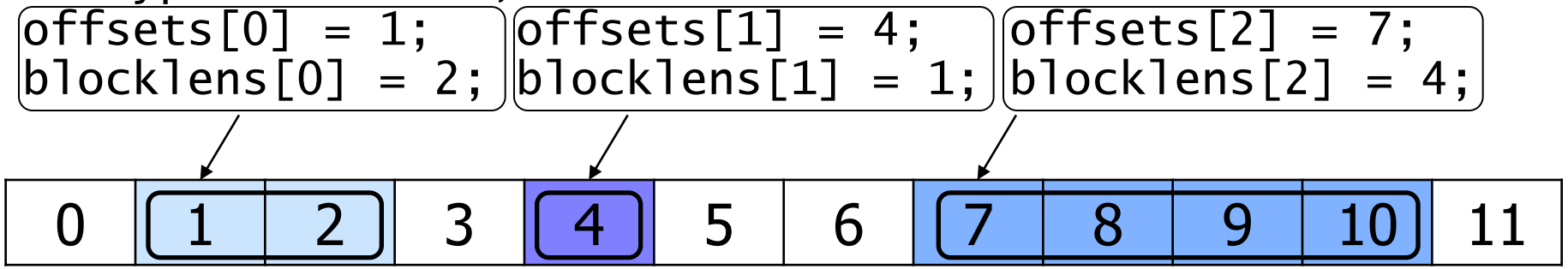
Indeksirani tip

- Koristi niz pomeraja osnovnog podatka kao mapu za upotrebu novog izvedenog tipa

```
MPI_Type_indexed / MPI_Type_hindexed
    (count,blocklens[],offsets[],old_type,&newtype)
MPI::Datatype noviTip =
    MPI::OSNOVNI_TIP.
    Create_indexed(count,blocklens,offsets)
```

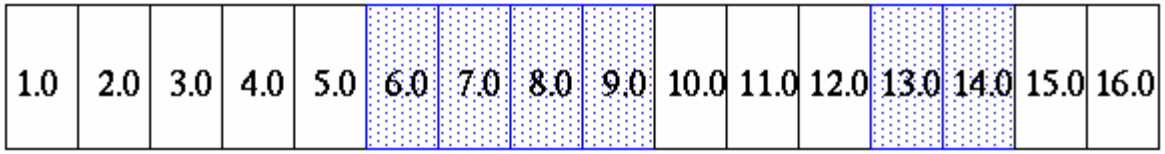
- Primer deklaracije indeksiranog tipa:

```
count = 3;
oldtype = MPI_INT;
offsets[0] = 1;
blocklens[0] = 2;
offsets[1] = 4;
blocklens[1] = 1;
offsets[2] = 7;
blocklens[2] = 4;
```



MPI_ Type_indexed

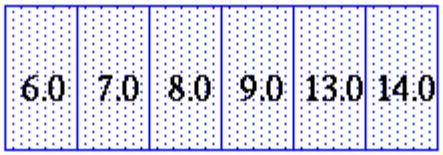
count = 2;
blocklengths[0] = 4;
displacements[0] = 5;
blocklengths[1] = 2;
displacements[1] = 12;



a[16]

MPI_Type_indexed(count, blocklengths, displacements, MPI_FLOAT, &indextype);

MPI_Send(&a, 1, indextype, dest, tag, comm);



1 element of indextype

Strukturirani tip

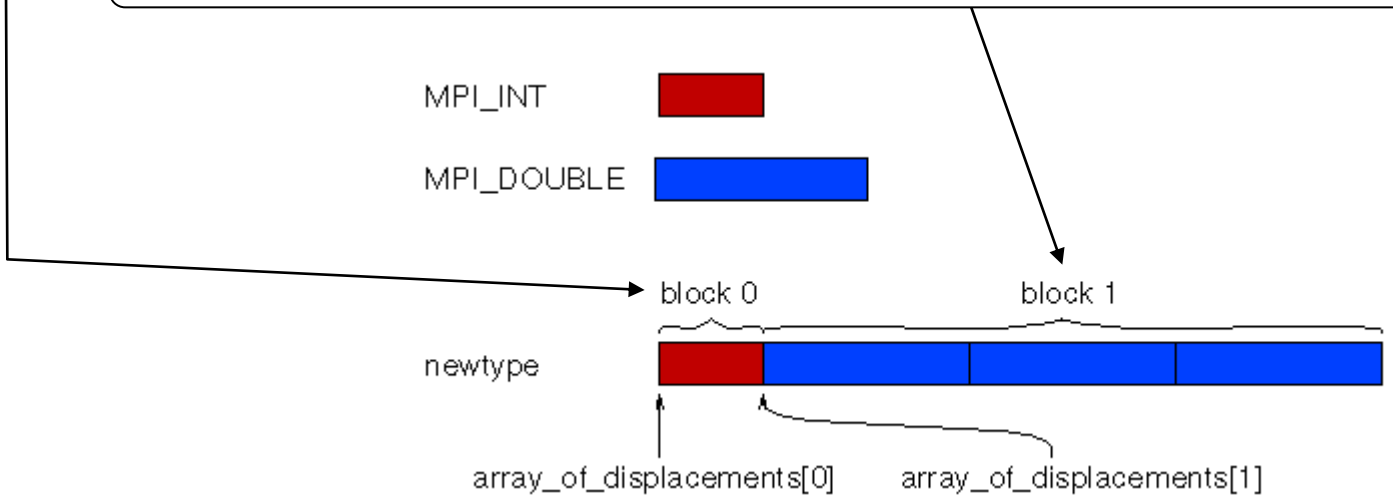
- Nadskup indeksiranog tipa (svaki element nizova sa dužinama blokova i pomerajima ima pridruženu oznaku tipa)

```
MPI_Type_struct
  (count,blocklens[],offsets[],
  oldtypes[],&newtype)
```

```
static MPI::Datatype
  MPI::Datatype::Create_struct
  (int count, const int blocklens,
  const MPI::Aint offsets[],
  const MPI::Datatype oldtypes[])
```

Primer deklaracije strukturiranog tipa

```
count = 2;
blocklens[0] = 1;
offsets[0] = 0;
oldtypes[0] = MPI_INT;
blocklens[1] = 3;
offsets[1] = offset1; /*objašenjenje na sledećem slajdu*/
oldtypes[1] = MPI_DOUBLE;
```



Veličina MPI tipova podataka

- Standardni `sizeof()` operator nije pogodan za dohvatanje informacije o veličini izvedenih MPI tipova
- Zato postoji funkcija koja vraća broj bajtova koje zauzima određeni tip

```
MPI_Type_extent(datatype, &extent)
MPI_Aint extent;
```

- Rezultat ove funkcije je koristan za funkcije koje kao argument zahtevaju veličinu podatka ili pomeraj izveden od te veličine
- Za primer sa prethodnog slajda potrebno je dodati:

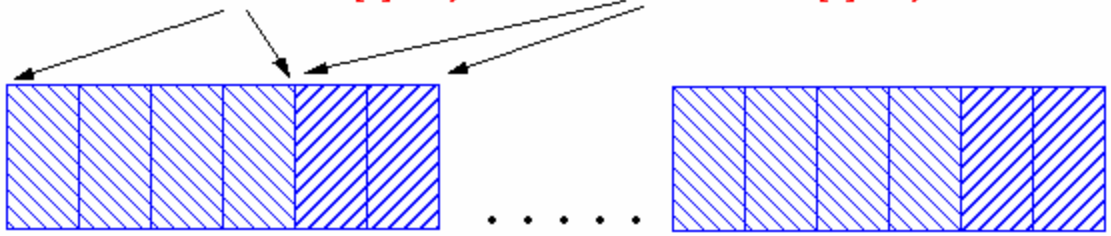
```
MPI_Aint extent, pom1;
MPI_Type_extent(MPI_INT, &extent);
offset1 = 1 * extent;
```

MPI_ Type_struct

```
typedef struct { float x, y, z, velocity; int n, type; } Particle;
Particle particles[NELEM];

MPI_Type_extent(MPI_FLOAT, &extent);

count = 2; oldtypes[0] = MPI_FLOAT;   oldtypes[1] = MPI_INT
           offsets[0] = 0;            offsets[1] = 4 * extent;
           blockcounts[0] = 4;        blockcounts[1] = 2;
```



particles[NELEM]

```
MPI_Type_struct(count, blockcounts, offsets, oldtypes, &particletype);

MPI_Send(particles, NELEM, particletype, dest, tag, comm);
```

Sends entire (NELEM) array of particles, each particle being comprised four floats and two integers.

Strukturirani tip – napomene (1)

- Svi ostali izvedeni tipovi služe da se lakše stigne do određenih podataka **istog** tipa
  - Korišćenjem proreda ili indeksa
- Samo strukturirani tip dozvoljava objedinjavanje više različitih tipova
- To omogućava programeru da objedini više nepovezanih tipova i tretira ih kao jedinstveni tip
  - Prilikom korišćenja Send i Receive funkcija
- U C++ ovo olakšava slanje primeraka klasa

Strukturirani tip – napomene (2)

- Ako su podaci koji sačinjavaju neki izvedeni tip definisani kao struktura, programer može sam da odredi pomeraje za sve podatke, korišćenjem MPI_Extent
  - Tip će se formirati korišćenjem relativnih adresa
- Ako su podaci raštrkani po memoriji, programer mora:
  - Koristiti funkciju MPI_ADDRESS da odredi apsolutne adrese podataka
  - Zbog korišćenja apsolutnih adresa, adresu celog podatka strukturnog tipa navesti kao MPI_BOTTOM

Strukturirani tip – napomene (3)

- Prilikom izvođenja tipova uz pomoć apsolutnih adresa, dobijeni tipovi ne smeju sadržati promenljive čije je vreme života kraće od vremena života izvedenog tipa
  - Greška je analogna korišćenju adresa lokalnih promenljivih za povratne rezultate funkcija (po promeni konteksta, na datoj adresi verovatno više neće biti potrebni podatak)

Grupe i komunikatori (1)

- Grupa je uređeni skup od N procesa
  - Svaki proces u grupi ima svoj jedinstveni rang [0..N-1] gde je N broj procesa u grupi
  - Grupi je uvek pridružen komunikator
- Grupa je dostupna programeru putem odgovarajućeg objekta grupe

```
MPI_Group *handle;
```

- Na početku svi procesi pripadaju globalnoj grupi

Grupe i komunikatori (2)

- Grupa je obavezno povezana sa objektom komunikatora
- Komunikator obuhvata grupu procesa koji mogu međusobno komunicirati (razmenjivati poruke)
  - Svaka MPI poruka mora navesti komunikator preko koga se prosleđuje
- Na početku svi procesi pripadaju globalnom komunikatoru MPI_COMM_WORLD
- Grupa i komunikator **nisu** isto

Osnovna namena grupa i komunikatora

- Ustrojstvo procesa u grupe, prema funkciji koju obavljaju
- Kolektivna komunikacija samo između određenih procesa u MPI svetu
- Osnova za virtuelne topologije procesa (Dekartova, graf)
- Bezbednija komunikacija
  - Izbegavanje mešanja poruka sa logički nepovezanim procesima iz MPI sveta

Osnovne osobine grupa i komunikatora

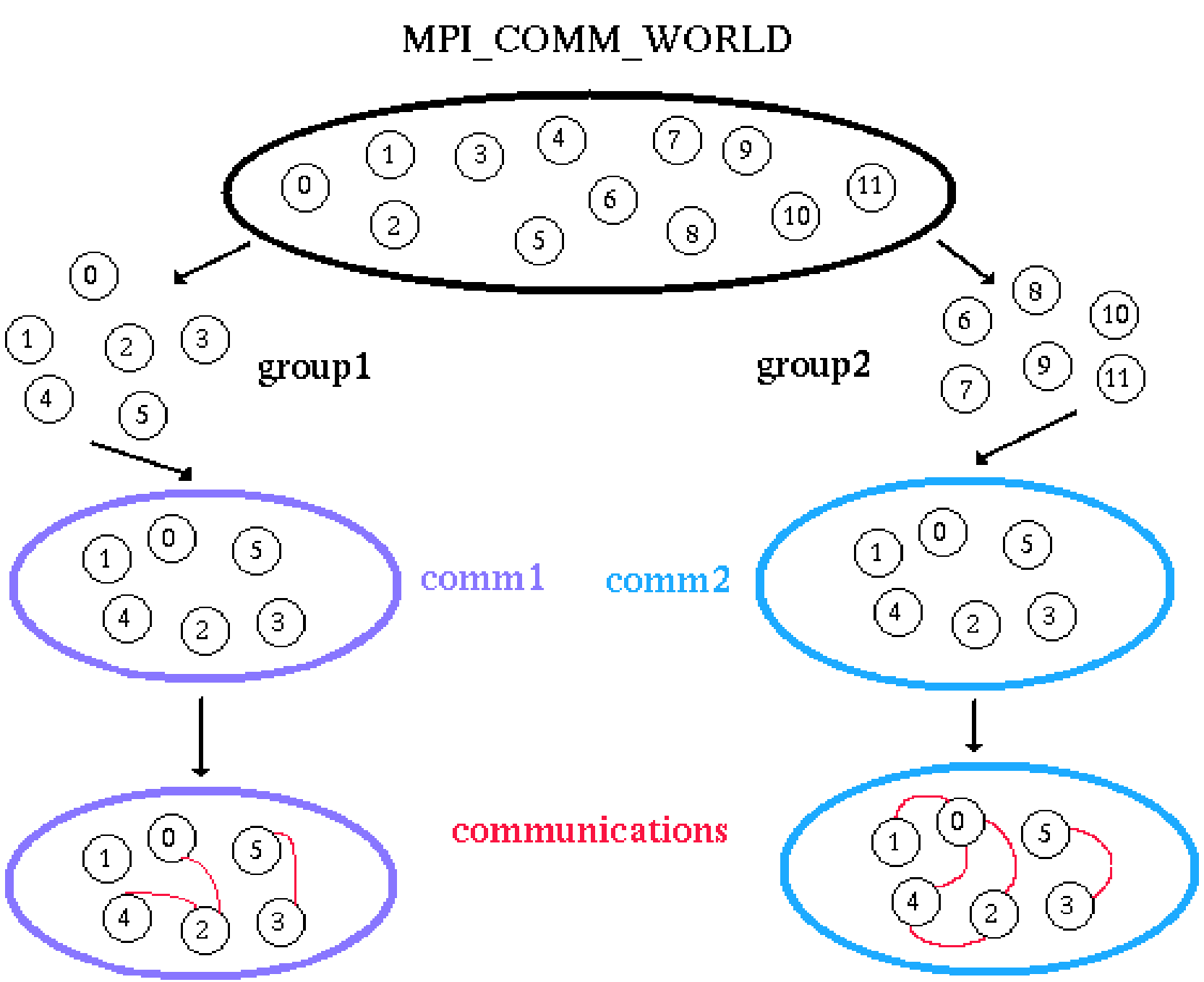
- Svaki proces je u sastavu MPI_COMM_WORLD, koji postoji od početka do kraja MPI sveta
- Ostale grupe i ostali komunikatori su dinamički
  - Nastaju i nestaju u toku izvršavanja programa, prema ukazanoj potrebi programa
- Proces može biti u sastavu više grupa/komunikatora
  - Unutar svakog para grupa/komunikator proces će imati svoj rang

Uobičajeni postupak sa grupama i komunikatorima (C)

- Dohvatanje globalne grupe od `MPI_COMM_WORLD`
  ```
  MPI_Comm_group(MPI_COMM_WORLD,MPI_Group *glob_group_addr)
  ```
- Stvaranje nove grupe koja je podskup globalne
  ```
  MPI_Group_incl(MPI_Group group, int process_count, int
  *ranks, MPI_Group *new_group_addr)
  ```
- Stvaranje komunikatora za upravo stvorenu grupu
  ```
  MPI_Comm_create
  (MPI_Comm comm, MPI_Group group, MPI_Comm *new_comm_addr)
  ```
- Određivanje ranga procesa u stvorenom komunikatoru
  ```
  MPI_Comm_rank(MPI_Comm comm, int *rank_addr)
  ```
- Upotreba MPI poruka kao i sa `MPI_COMM_WORLD`
- Uništavanje stvorenih grupe i komunikatora
  ```
  MPI_Comm_free(MPI_Comm *comm_addr)
  MPI_Group_free(MPI_Group *group_addr)
  ```

Uobičajeni postupak sa grupama i komunikatorima (C++)

- Dohvatanje globalne grupe od MPI_COMM_WORLD
```
MPI::Group globalGroup = MPI::COMM_WORLD.Get_group();
```
- Stvaranje nove grupe koja je podskup globalne
```
MPI::Group newGroup =
    globalGroup::Incl(process_count, ranks);
```
- Stvaranje komunikatora za upravo stvorenu grupu
```
MPI::Intracomm newCommunicator =
    MPI::COMM_WORLD::Create(newGroup);
```
- Određivanje ranga procesa u stvorenom komunikatoru
```
int newRank = newCommunicator.Get_rank();
```
- Upotreba MPI poruka kao i sa MPI::COMM_WORLD
- Uništavanje stvorenih grupe i komunikatora
```
newGroup.Free();
newCommunicator.Free();
```


MPI_COMM_WORLD
group1
group2
comm1
comm2
communications

Funkcije za stvaranje grupe

- Stvaranje grupe od postojeće grupe

```
MPI_Group_incl
(MPI_Group group,int n,int *ranks, MPI_Group *newgroup)
MPI_Group_excl
(MPI_Group group,int n,int *ranks, MPI_Group *newgroup)
MPI_Group_range_incl
(MPI_Group group,int n,int ranges[][3],MPI_Group *newgroup)
MPI_Group_range_excl
(MPI_Group group,int n,int ranges[][3],MPI_Group *newgroup)
```

Napomena: `ranges` trojke su oblika `(first_rank, last_rank, stride)`

- Stvaranje grupe od postojećih grupa

```
MPI_Group_intersection
(MPI_Group group1,MPI_Group group2,MPI_Group *newgroup)
MPI_Group_union
(MPI_Group group1,MPI_Group group2,MPI_Group *newgroup)
MPI_Group_difference
(MPI_Group group1,MPI_Group group2,MPI_Group *newgroup)
```

Ostale funkcije za rad sa grupama

- Dohvatanje (početne) grupe od komunikatora

```
MPI_Comm_group(MPI_Comm comm, MPI_Group *group)
```

- Pristup postojećim grupama

```
MPI_Group_rank(MPI_Group group,int *rank)
MPI_Group_size(MPI_Group group,int *size)
MPI_Group_compare
(MPI_Group group1,MPI_Group group2,int *result)
```

Rezultat: `MPI_IDENT,MPI_SIMILAR,MPI_UNEQUAL`

```
MPI_Group_translate_ranks(MPI_Group group1,int n,
int *ranks1,MPI_Group group2,int *ranks2)
```

- Uništavanje grupe

```
MPI_Group_free(MPI_Group *group)
```

Funkcije za rad sa komunikatorima (1)

- Pristup postojećim komunikatorima

```
MPI_Comm_size(MPI_Comm comm,int *size)
MPI_Comm_rank(MPI_Comm comm,int *rank)
MPI_Comm_compare
(MPI_Comm comm1,MPI_Comm comm2,int *result)
```

  Rezultat:

```
MPI_IDENT,      //isti objekat
MPI_CONGRUENT, //isti procesi, isti poredak
MPI_SIMILAR,    //isti procesi,drugi poredak
MPI_UNEQUAL     //svi ostali slučajevi
```

- Uništavanje komunikatora

```
MPI_Comm_free(MPI_Comm *comm)
```

Funkcije za rad sa komunikatorima (2)

- Stvaranje komunikatora
  `MPI_Comm_create(MPI_Comm comm_in, MPI_Group group, MPI_Comm *comm_out)`
- Stvaranje duplikata komunikatora
  `MPI_Comm_dup (MPI_Comm comm, MPI_Comm *newcomm)`
- Podela komunikatora
  `MPI_Comm_split(MPI_Comm comm_in, int color, int key, MPI_Comm *comm_out)`
  Rezultat: **više disjunktnih** komunikatora, za svaku boju (`color`) po jedan
- Ove tri operacije su kolektivne
- Ostale obrađene operacije su lokalne

Vrste komunikatora

- Za komunikaciju unutar grupe
  Intracommunicator (MPI::Intracomm)
- Za komunikaciju između dveju grupa
  Intercommunicator (MPI::Intercomm)
- Za virtuelnu Dekartovu topologiju
  Cartesian communicator (MPI::Cartcomm)
- Za virtuelnu topologiju grafa
  Graph communicator (MPI::Graphcomm)

Virtuelne topologije

- Virtuelna topologija uređuje preslikavanje između rangova procesa i njihovih koordinata u zamišljenom geometrijskom figuri ili telu
- "Virtuelna" zato što ne mora postojati veza između koordinata procesa u zamišljenoj geometriji i fizičke strukture MPI sveta
- Virtuelne topologije se ostvaruju putem određenih vrsta komunikatora
- Programer sam mora da rasporedi procese zarad postizanja željene topologije
- Postoje Dekartova i grafovska virtuelna topologija

Motivacija i implementacioni detalji

- Pogodnosti virtuelnih topologija
  - Paralelne aplikacije često imaju specifičan obrazac komunikacije,
  - Takav obrazac često može biti predstavljen prstenom, cilindrom, torusom ili grafom
  - Uz pomoć odgovarajućih funkcija programer dobija koordinate susednih procesa u virtuelnoj topologiji
- Efikasnost komunikacije kod virtuelnih topologija
  - Pojedine paralelne arhitekture mogu imati visoka kašnjenja poruka između susednih procesa
  - Pojedine MPI implementacije mogu optimizovati mapiranje procesa prema fizičkim karakteristikama paralelne arhitekture

Primeri Cartesian topologije

	-1,0 (4)	-1,1 (5)	
0,-1(-1)	0,0 (0)	0,1 (1)	0,2(-1)
1,-1(-1)	1,0 (2)	1,1 (3)	1,2 (-1)
2,-1(-1)	2,0 (4)	2,1 (5)	2,2 (-1)
	3,0 (0)	3,1 (1)	

```
periods = { TRUE, FALSE }
```

Cartesian topologija preslikava jednodimenzioni niz rangova (npr. od 0 do 5) u Dekartove koordinate (npr. 3X2)

```
periods = { FALSE, TRUE }
```

	-1,0 (-1)	-1,1 (-1)	
0,-1 (1)	0,0 (0)	0,1 (1)	0,2 (0)
1,-1 (3)	1,0 (2)	1,1 (3)	1,2 (2)
2,-1 (5)	2,0 (4)	2,1 (5)	2,2 (4)
	3,0 (-1)	3,1 (-1)	

Funkcije za rad sa Cartesian topologijom (1)

- Stvaranje komunikatora
  `MPI_Cart_create(MPI_Comm comm_old,int ndims,int *dims, int *periods,int reorder,MPI_Comm *comm_cart)`
  - u suvišnim procesima vraća `MPI_COMM_NULL`
  - `reorder` biva ignorisan u nekim implementacijama, vrednost 1 je preporučena zbog mogućeg ubrzanja
  - `comm_old` mora biti intrakomunikator
- Prevođenje ranga u koordinate unutar topologije
  `MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int *coords)`
- Prevođenje koordinata unutar topologije u rang
  `MPI_Cart_rank(MPI_Comm comm,int *coords,int *rank)`

Funkcije za rad sa Cartesian topologijom (2)

- Deljenje komunikatora
  `MPI_Cart_sub(MPI_Comm comm, int *remain_dims, MPI_Comm *newcomm)`
  - Rezultat: **više** disjunktnih komunikatora, **po jedan za svaku kombinaciju** dimenzija_koja_biva_deljena – veličina_deljene_dimenzije
  - Svaki novi komunikator sadrži jednak deo prvobitne strukture
- Primer 1: postoji 2X3X4 topologija u komunikatoru `comm`, `remain_dims = {1,0,1}`.
  Poziv će napraviti 3 komunikatora, svaki sa 2X4 procesa
- Primer 2: ista topologija, `remain_dims = {0,0,1}`.
  Poziv će napraviti 6 komunikatora, svakom niz od 4 procesa

Funkcije za rad sa Cartesian topologijom (3)

- Dohvatanje informacija o topologiji
  `MPI_Cart_get(MPI_Comm comm, int maxdims, int *dims, int *periods, int *coords)`
  - Rezultat je paket podataka vezan za pozivajući proces u datom komunikatoru – dimenzije topologije, cikličnost topologije i koordinate procesa unutar topologije
- Proračun dimenzija topologije za dati broj procesa
  `MPI_Dims_create (int nnodes,int ndims,int *dims)`
- Dohvatanje broja dimenzija u komunikatoru
  `MPI_Cartdim_get (MPI_Comm cart_comm,int *ndims)`

Funkcije za rad sa Cartesian topologijom (4)

- Računanje ciljnih koordinata na obe strane ose
  `MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)`
  - `disp` je pomeraj od ranga trenutnog procesa prema početku (`rank_source == rank - disp`) i od početka (`rank_dest == rank + disp`) koordinatne ose
  - `source` i `dest` zato što često ti podaci budu posle upotrebljeni kao argumenti `MPI_SendRecv` poziva
- Preraspoređivanje procesa (promena rangova) prema karakteristikama paralelne arhitekture
  `MPI_Cart_map(MPI_Comm comm, int ndims, int *dims, int *periods, int *newrank)`

Funkcije za rad sa Graph topologijom (1)

- Topologija grafa obezbeđuje programeru mehanizam za definisanje proizvoljnih veza između procesa
- Graf je usmeren => relacija susedstva je asimetrična (ako je prvi čvor sused drugom, drugi čvor nije automatski sused prvom)
- Stvaranje komunikatora

```
MPI_Graph_create(MPI_Comm comm_old,
int nnodes, int *degree_indices, int *edges,
int reorder, MPI_Comm *comm_graph)
```

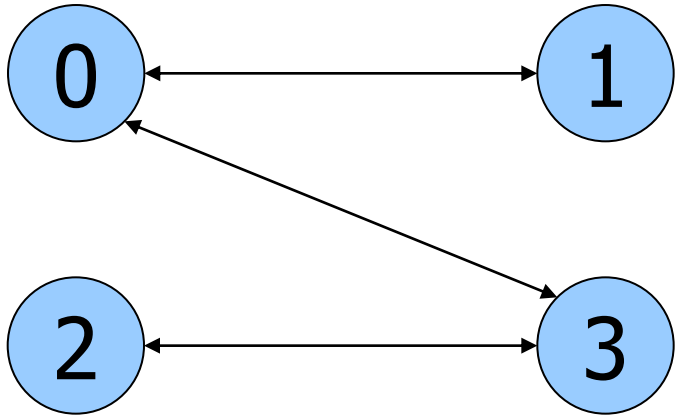
  - Broj čvorova u grafu (`nnodes`) ne sme biti veći od broja procesa u grupi koju komunikator obuhvata
  - U suvišnim procesima vraća `MPI_COMM_NULL`
  - `reorder` biva ignorisan u nekim implementacijama, vrednost 1 je preporučena zbog mogućeg ubrzanja
  - `comm_old` mora biti intrakomunikator

Primer 1

- Linije između procesa označavaju komunikaciju koju je programer definisao
- Strelice označavaju smer (ovde – sve relacije dvosmerne)
- Argumenti za MPI_Graph_create:

```
int nnodes = 4;
int degree_indices[4]
  = {2, 3, 4, 6};
int edges[6]
  = {1, 3, 0, 3, 0, 2};
```

♠ ♦ ♥ ♣

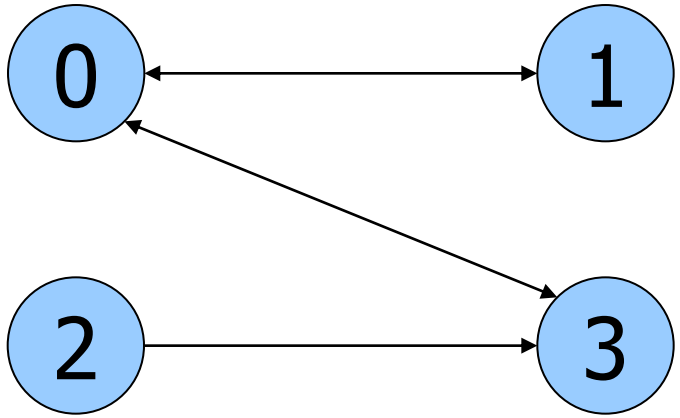


Čvor	Broj suseda	Prvi sledeći indeks	Susedi
0	2	2 (edges[0], edges[1])	1,3 ♠
1	1	3 (edges[2])	0 ♦
2	1	4 (edges[3])	3 ♥
3	2	6 (edges[4], edges[5])	0,2 ♣

Primer 2

- Linije između procesa označavaju komunikaciju koju je programer definisao
- Strelice označavaju smer (relacija 2-3 je jednosmerna)
- Argumenti za MPI_Graph_create:

```
int nnodes = 4;
int degree_indices[4]
  = {2, 3, 4, 5};
int edges[6]
  = {1, 3, 0, 3, 0};
```



Čvor	Broj suseda	Prvi sledeći indeks	Susedi
0	2	2 (edges[0], edges[1])	1,3 ♠
1	1	3 (edges[2])	0 ♦
2	1	4 (edges[3])	3 ♥
3	1	6 (edges[4])	0 ♣

Funkcije za rad sa Graph topologijom (2)

- Dohvatanje broja susednih procesa u grafu
  `MPI_Graph_neighbors_count (MPI_Comm comm, int rank, int *neighbors_count)`
  - Rezultat je broj suseda smešten u `neighbors_count`, koji će biti argument za poziv `MPI_Graph_neighbors`
- Dohvatanje rangova susednih procesa u grafu
  `MPI_Graph_neighbors(MPI_Comm comm, int rank, int neighbors_count, int *neighbors);`
  - Niz `neighbors` mora imati makar `neighbors_count` elemenata
- **Važno:** za obe funkcije argument comm **mora biti** tipa grafovskog komunikatora

Funkcije za rad sa Graph topologijom

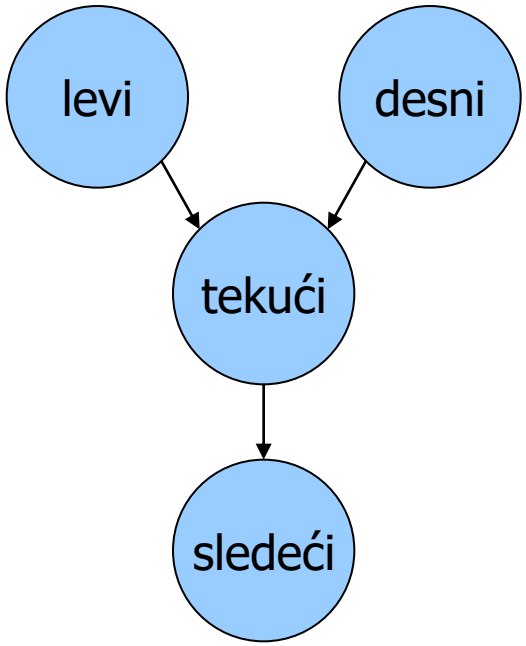
- Dohvatanje dimenzija grafa
  `MPI_Graphdims_get (MPI_Comm comm, int *node_count, int *edge_count);`
  - Rezultat su broj čvorova i broj suseda smešteni u `node_count`, i `edge_count`, respektivno
  - Ova dva podatka će biti argument za poziv `MPI_Graph_get`
- Dohvatanje čvorova i ivica grafa
  `MPI_Graph_get(MPI_Comm comm, int node_count, int edge_count, int *degree_indices,int *edges);`
  - Nizovi degree_indices i `edges` moraju imati makar `node_count` i `edge_count` elemenata, respektivno
  - Rezultat su nizovi prosleđeni prilikom stvaranja komunikatora `comm`
- **Važno:** za obe funkcije argument comm **mora biti** tipa grafovskog komunikatora

Funkcije za rad sa Graph topologijom

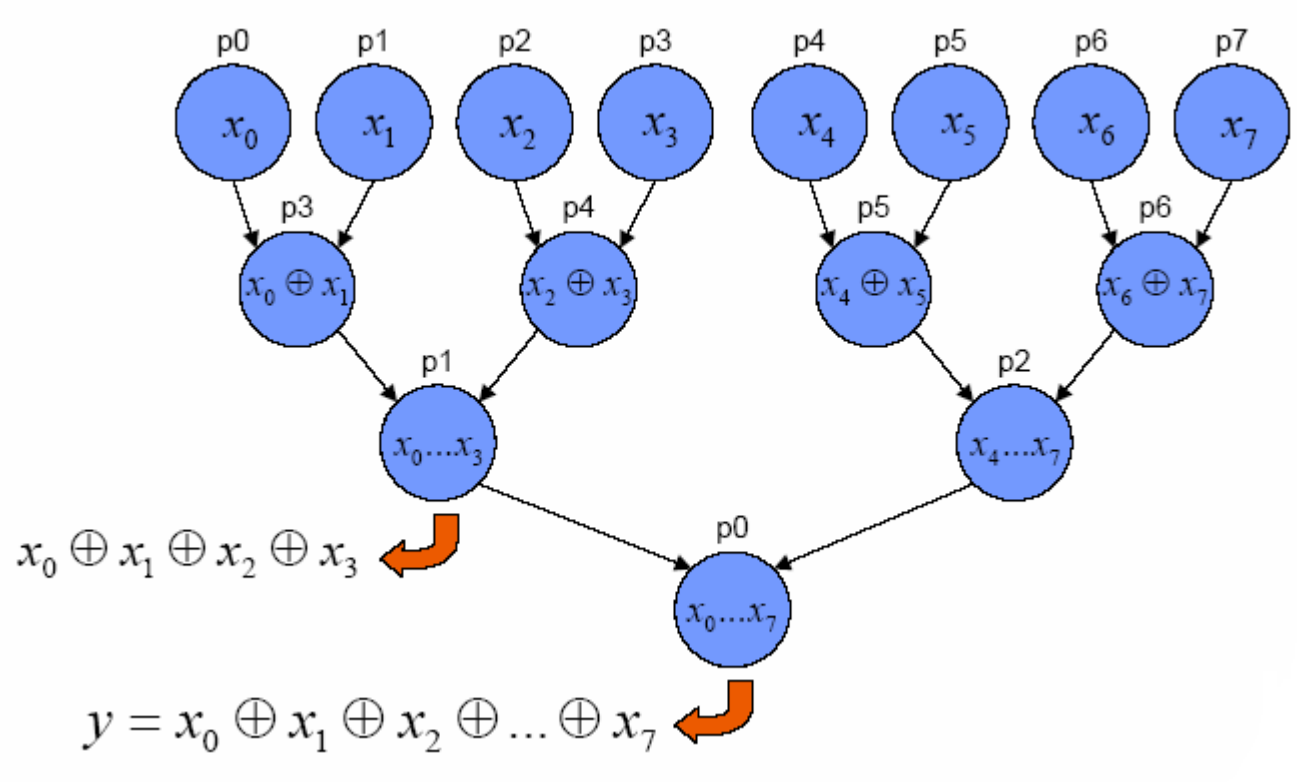
- Ispitivanje komunikatoru pridružene topologije
  `MPI_Topo_test(MPI_Comm comm,int *status)`
  rezultat će biti smešten u `status`
  - `MPI_GRAPH`, ako je graf topologija
  - `MPI_CART`, ako je Dekartova topologija
  - `MPI_UNDEFINED`,
    ako komunikatoru nije pridružena nikakva topologija
- Preraspoređivanje procesa (promena rangova) prema karakteristikama paralelne arhitekture
  `MPI_Graph_map(MPI_Comm comm, int nnodes, int *index, int *edges, int *newrank);`

Kraći opis procesa hijerarhijske obrade

- Ulazni podaci će biti podeljeni na n nezavisnih delova (n je broj procesa koji obavljaju računanje)
- Učitaj podatke u svaki list i pošalji sledećim u stablu
- Za svaki tekući čvor, izvrši proračun nad primljenim podacima i pošalji rezultat sledećem u stablu
- Kasnije, tekući čvor prima rezultate levog i desnog i ponavlja proračun
- Poslednji čvor u stablu određuje konačni rezultat



Primer hijerarhijske obrade



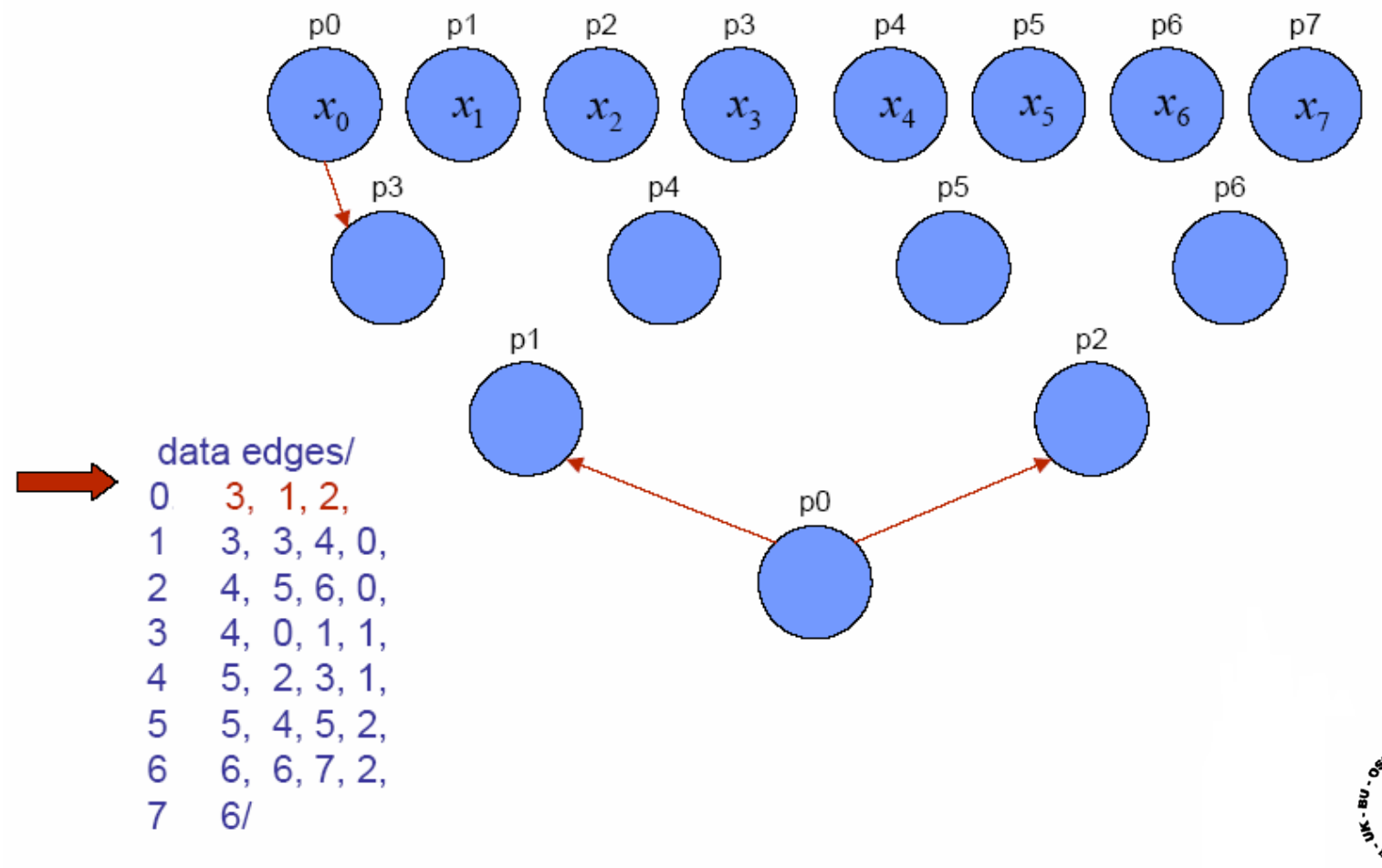
Primer struktura podataka za stablo

- Ivice povezuju čvor sa sledećim za prvi rezultat, levim, desnim, sledećim za međurezultat

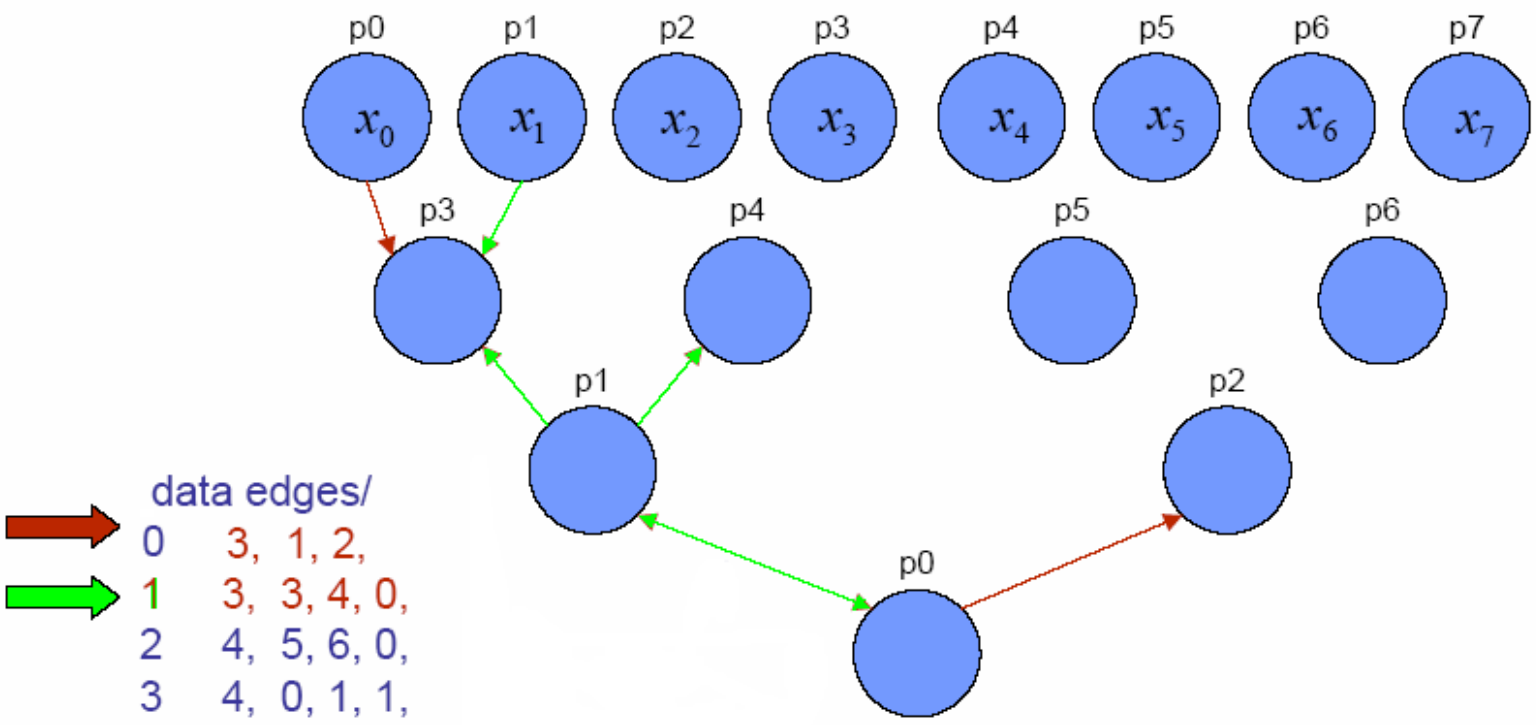
```
int edges [28] = {
   3,1,2,
   3,3,4,0,
   4,5,6,0,
   4,0,1,1,
   5,2,3,1,
   5,4,5,2,
   6,6,7,2,
   6 };
```

Čvor	Broj suseda	Prvi sledeći indeks	Susedi
0	3	3	3,1,2
1	4	7	3,3,4,0
2	4	11	4,5,6,0
3	4	15	4,0,1,1
4	4	19	5,2,3,1
5	4	23	5,4,5,2
6	4	27	6,6,7,2
7	1	28	6

Primer povezivanja čvorova u stablu



Primer povezivanja čvorova u stablu



Reference

- MPI forum
  http://www.mpi-forum.org
- MPI web pages at Argonne National Laboratory
  http://www-unix.mcs.anl.gov/mpi
- "Using MPI", Gropp, Lusk and Skjellum. MIT Press, 1994.
- Livermore Computing specific information:
  - MPI at LLNL (odavde je preuzet primetan deo ove prezentacije)
    http://www.llnl.gov/computing/mpi
  - IBM SP Systems Overview tutorial
    http://www.llnl.gov/computing/tutorials/ibm_sp
  - Compaq Clusters Overview tutorial
    http://www.llnl.gov/computing/tutorials/compaq_clusters
  - IA 32 Linux Clusters Overview tutorial
    http://www.llnl.gov/computing/tutorials/linux_clusters
- "MPI and Elan Library Environment Variables" document
  www.llnl.gov/computing/mpi/elan.html

Reference

- IBM Parallel Environment Manuals
  http://www-1.ibm.com/servers/eserver/pseries/library/sp_books
- IBM Compiler Documentation:
  Fortran: www-4.ibm.com/software/ad/fortran
  C/C++: www-4.ibm.com/software/ad/caix
- Intel Compilers
  intel.com/software/products/compilers
- "RS/6000 SP: Practical MPI Programming", Yukiya Aoyama and Jun Nakano, RS/6000 Technical Support Center, IBM Japan. Available from IBM's Redbooks server at http://www.redbooks.ibm.com.
- "A User's Guide to MPI", Peter S. Pacheco. Department of Mathematics, University of San Francisco.