Paralelné programovanie
Message Passing Interface

Bc. št. prog. Informatika - 2010/2011

Ing. Michal Čerňanský, PhD.

Fakulta informatiky a
informačných technológií,
STU Bratislava

Prehľad tém

- Základné princípy programátorských modelov zasielania správ
- Základné stavebné kamene: operácie „send“ a „receive“
- Topológie a mapovania
- Prelínanie komunikácie a výpočtob
- Kolektívne komunikačné a výpočtové operácie
- Skupiny a komunikátory

Základné princípy programátorských modelov zasielania správ

- Logický pohľad na paralelný systém:
  - P procesov - každý s vlastným priestorom adries
- Každý údaj patrí do práve jedného priestoru adries – explicitné rozdelenie do priestorov a umiestnenie údajov do priestorov
- Všetky interakcie vyžadujú kooperáciu dvoch procesov
  - Proces ktorý vlastní údaje
  - Proces ktorý chce pristúpiť k údajom

Základné princípy programátorských modelov zasielania správ

- MPI programy sú často riešené pomocou asynchrónnych a voľne synchrónnych paradigiem.
  - Asynchrónne paradigmy – súbežné úlohy sú vykonávané asynchrónne
  - Voľne synchrónne paradigmy – úlohy alebo podmnožiny úloh sa synchronizujú v priebehu vykonávania výpočtu, medzi synchronizáciami úlohy sú vykonávané asynchrónne
- Väčšina programov využíva SPMD model

Základné stavebné kamene: operácie „send“ a „receive“

- Prototypy operácií:

```
send(void *sendbuf, int nelems, int dest)
receive(void *recvbuf, int nelems, int source)
```

- Napríklad nasledujúce fragmenty zdr. kódu:

```
P0                          P1
a = 100;                    receive(&a, 1, 0)
send(&a, 1, 1);             printf("%d\n", a);
a = 0;
```

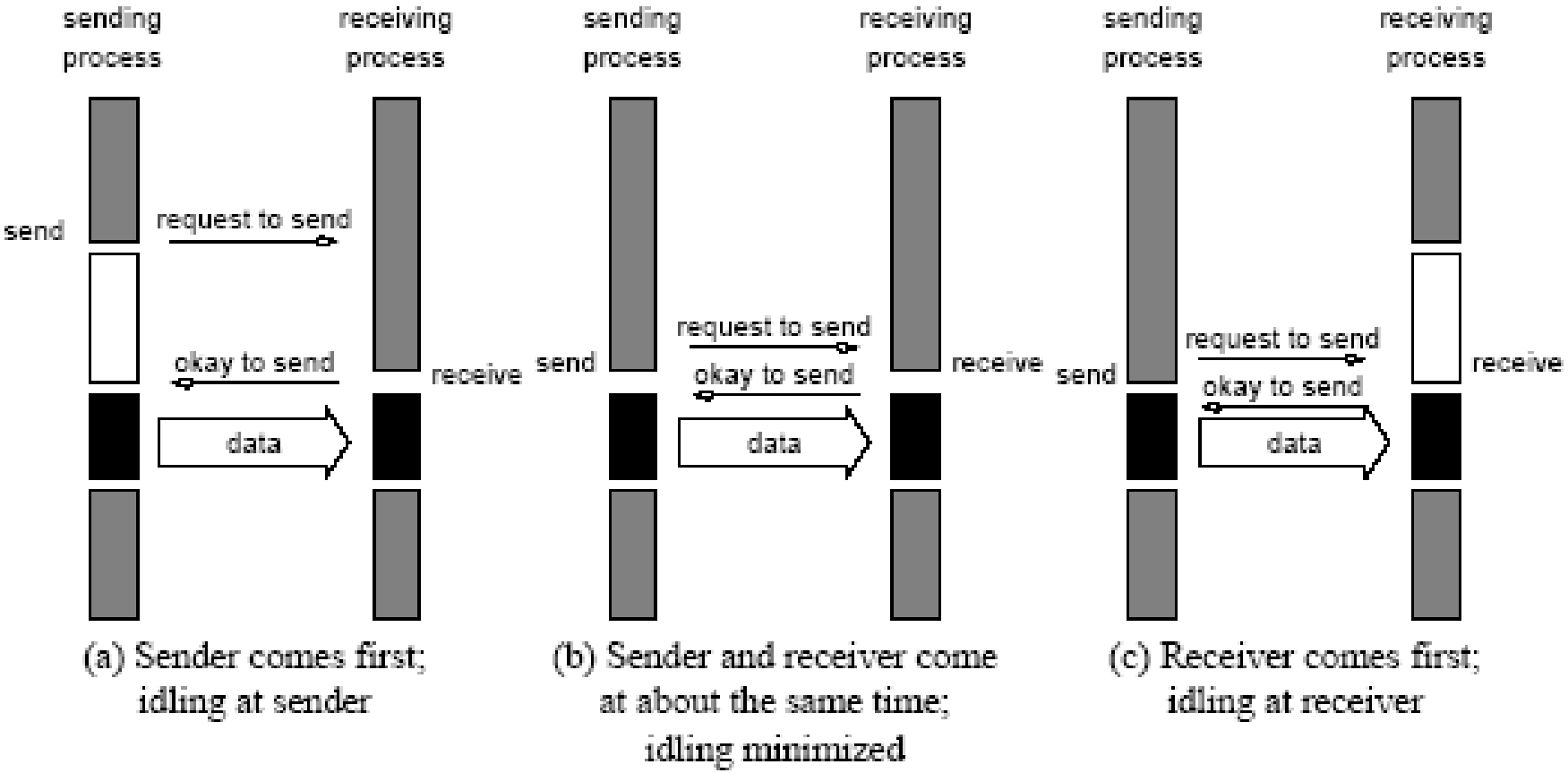
- Sémantika send operácie: po vykonaní hodnota prijatá P1 procesom musí byť 100

Blokujúce operácie zasielania správ bez vyrovnávacej pamäte

- Jednoduchý spôsob realizovania send/receive sémantiky: send je ukončené len ak je to bezpečné
- Blokujúci „send“ bez vyrovnávacej pamäte – operácia neskončí, kým nie je vyvolaný prislúchajúci „receive“
- Čakanie a uviaznutie – najčastejšie problémy s blokujúcimi „send“ operáciami bez vyrovnávacej pamäte

Blokujúce operácie zasielania správ bez vyrovnávacej pamäte

- „send“ a „receive“ v rôznych časoch - réžia



(a) Sender comes first; idling at sender

(b) Sender and receiver come at about the same time; idling minimized

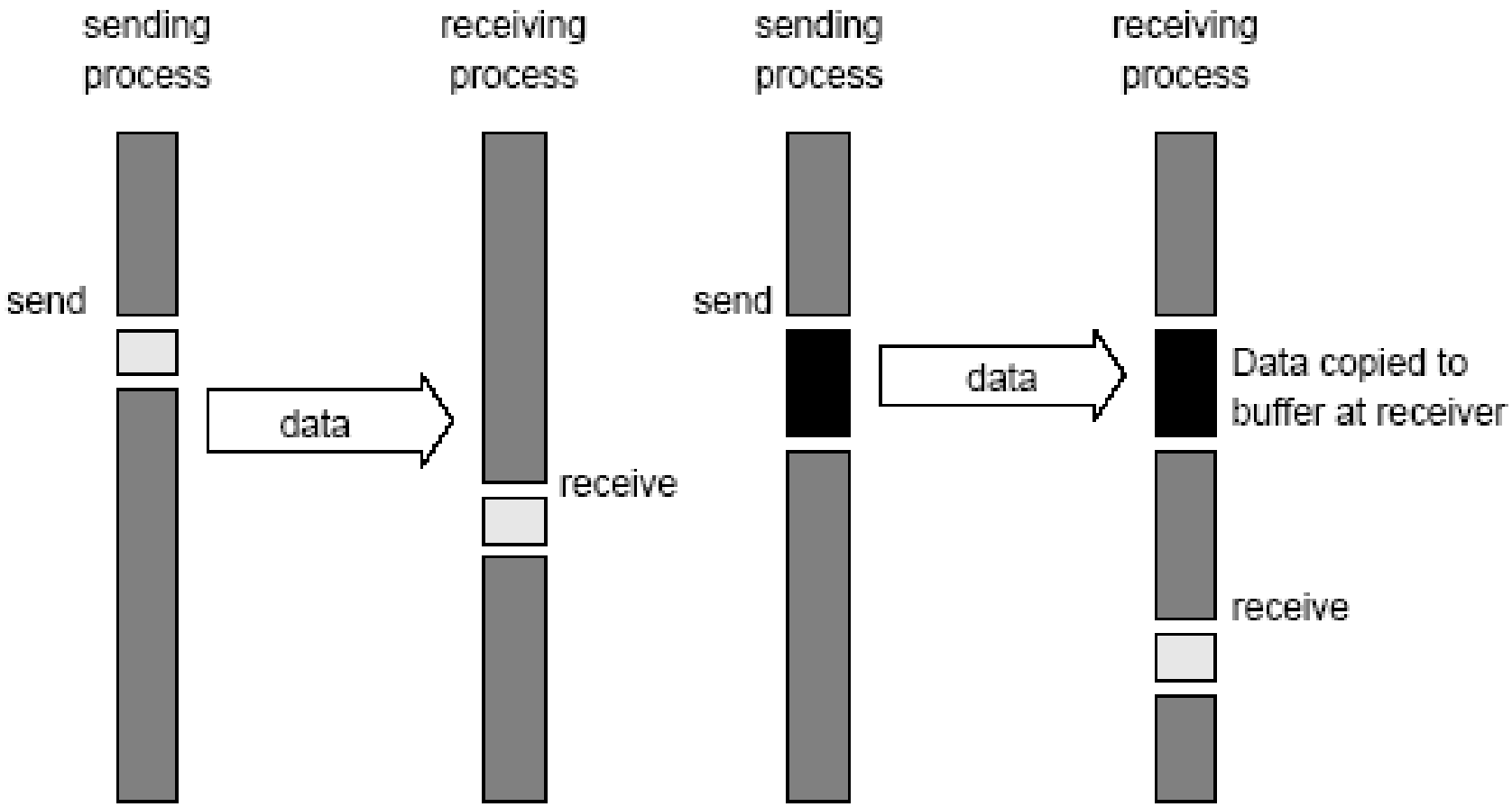
(c) Receiver comes first; idling at receiver

Blokujúce operácie zasielania správ s vyrovnávacou pamäťou

- Jednoduché riešenie vzhľadom na čakanie a uviaznutie – použitie vyrovnávacej pamäte na strane odosielateľa aj príjmateľa
- Pri použití vyrovnávacej pamäte – odosielateľ skopíruje údaje do pamäte a skončí po skončení operácie kopírovaia
- Údaje kopírované do vyrovnávacej pamäte aj na strane prijímateľa
- Vyrovnávacia pamäť odľahčuje čakanie za cenu väčšej réžie spojenej s kopírovaním

Blokujúce operácie zasielania správ s vyrovnávacou pamäťou

- (a) Použitý komunikačný HW, (b) bez HW - réžia



Blokujúce operácie zasielania správ s vyrovnávacou pamäťou

- Konečná veľkosť vyrovnávacej pamäte – značný dosah na výkonnosť
- Konzument pomalší ako producent?

```
P0
for (i = 0; i < 1000; i++) {
  produce_data(&a);
  send(&a, 1, 1);
}
```

```
P1
for (i = 0; i < 1000; i++){
  receive(&a, 1, 0);
  consume_data(&a);
}
```

Blokujúce operácie zasielania správ s vyrovnávacou pamäťou

- Uviaznutie je možné aj keď použitie „buffer-ov“

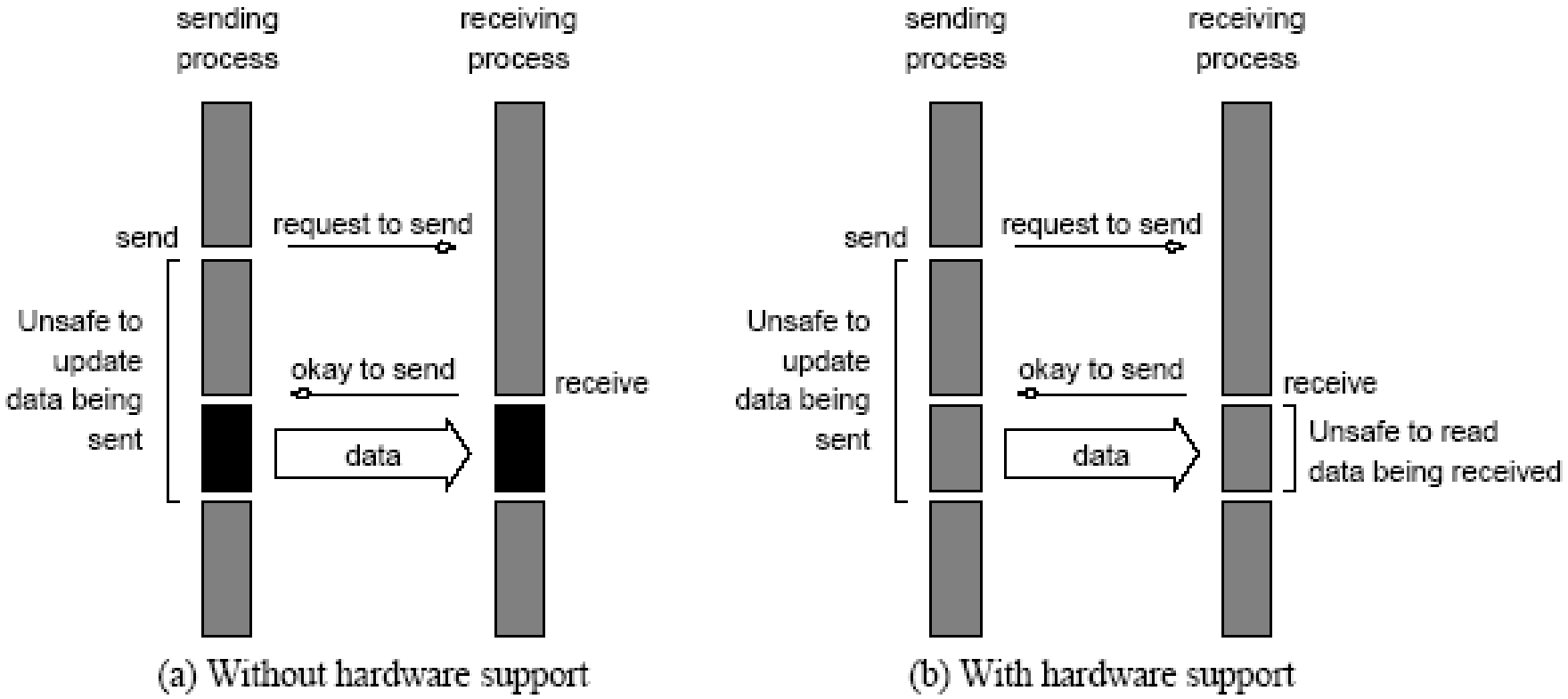
```
P0                          P1
receive(&a, 1, 1);          receive(&a, 1, 0);
send(&b, 1, 1);             send(&b, 1, 0);
```

Neblokujúce operácie zasielania správ

- Programátor je zodpovedný za sémanticky správne operácie „send“ a „receive“
- „send“ a „receive“ končia skôr ako je to sémanticky bezpečné
- Neblokujúce operácie sú zvyčajne kombinované operáciami na zistenie stavu
- Možnosť realizovať prelínanie medzi výpočtami a komunikáciou
- Knižnice zasielania správ zvyčajne poskytujú blokujúce aj neblokujúce primitívy

Neblokujúce operácie zasielania správ

- (a) Použitý komunikačný HW, (b) bez HW - réžia



Prehľad protokolov "send/receive"

	Blokujúce operácie	Neblokujúce operácie
Vyrovnávacia pamäť	Operácia v odosielajúcom procese skončí po skopírovaní údajov do odosielajúceho buffera	Operácia v odosielajúcom procese skončí po inicializácii kopírovania do vyrovnávacej pamäti.
Bez vyrovnávacej pamäti	Operácia v odosielajúcom procese je blokovaná (čaká), kým zodpovedajúca „receive" operácia nie je vykonaná	
	send a receive sémanticky zabezpečené	Programátor musí explicitne zabezpečiť sémantickú správnosť

MPI – Message Passing Interface

- MPI – štandardná knižnica pre paralelné programátorské modely zasielania správ – prenositeľné programy
- Štandard MPI - syntax a sémantika množiný základných funkcií
- Väčšina HW výrobcov paralelných platforiem poskytuje implementáciu MPI
- Rozsiahla knižnica, ale malá podmnožina funkcií môže byť postačujúca

MPI – Message Passing Interface

- Základné MPI funkcie:

`MPI_Init`	Inicializácia MPI
`MPI_Finalize`	Ukončenie MPI
`MPI_Comm_size`	Zistenie počtu procesov
`MPI_Comm_rank`	Číslo MPI procesu
`MPI_Send`	Odoslanie správy
`MPI_Recv`	Prijatie správy

Inicializácia a ukončenie používania MPI knižnice

- `MPI_Init` – inicializácia MPI prostredia
- `MPI_Finalize` – ukončenie práce s MPI prostredím, uvoľnenie pamäte a pod.
- Prototypy operácií:

```
int MPI_Init(int *argc, char ***argv)
int MPI_Finalize()
```

- `MPI_Init` – spracovanie argumentov príkazového riadku
- MPI operácie, dátové typy a konštantymajú prefix "`MPI_`"
- Návratový kód v prípade úspechu `MPI_SUCCESS`

Komunikátory

- Komunikátor definuje komunikačnú doménu – množinu operácií, ktorým je povolené spolu komunikovať
- Informácie o komunikačnej doméne sú v premennej typu `MPI_Comm`.
- Komunikátory sú uvádzané ako argumenty
- Proces môže patriť viacerým (aj prekrývajúcim sa) komunikačným doménam
- Predvolený komunikátor `MPI_COMM_WORLD` – všetky procesy

Získavanie informácii od prostredia

- Funkcie `MPI_Comm_size` a `MPI_Comm_rank` - určenie počtu procesov a číslo procesu v skupine procesov.
- Prototypy operácií:

```
int MPI_Comm_size(MPI_Comm comm, int *size)
int MPI_Comm_rank(MPI_Comm comm, int *rank)
```

- „rank“ procesu – číslo od 0 po veľkosť komunikátora mínus 1.

MPI „Hello World“ program

```
#include <mpi.h>

main(int argc, char *argv[])
{
   int npes, myrank;
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &npes);
   MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
   printf("From process %d out of %d, Hello
   World!\n",
       myrank, npes);
   MPI_Finalize();
}
```

Zasielanie a prijímanie správ

- Základné operácie zasielania a príjmania správ: `MPI_Send` a `MPI_Recv`
- Prototypy funkcií:

```
int MPI_Send(void *buf, int count, MPI_Datatype  datatype,
     int dest, int tag, MPI_Comm comm)
int MPI_Recv(void *buf, int count, MPI_Datatype  datatype,
     int source, int tag, MPI_Comm comm, MPI_Status *status)
```

- MPI poskytuje dátové typy ekvivalentné s C - prenositeľnosť
- `MPI_BYTE` - byte (8 bitov)
- `MPI_PACKED` – štruktúra vytvorená z rôznych typov
- Značka správy (tag) – od 0 po `MPI_TAG_UB`.

Dátové typy MPI

MPI Dátové typy	C Dátové typy
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Zasielanie a prijímanie správ

- MPI podpruje „žolíka“ (wildcard) pre značku
- Zdroj `MPI_ANY_SOURCE` – ľubovoľný proces u komunikačnej domény môže byť zdrojom
- Zdroj `MPI_TAG_ANY` – správa s ľubovoľnou značkou je akceptovaná
- Na strane prijímateľa dĺžka správy musí byť rovnaká alebo menšia ako pole dĺžka

Zasielanie a prijímanie správ

- Na strane prijímateľa – premenná definujúca stav vykonanej `MPI_Recv` operácie
- Zodpovedajúca dátová štruktúra:

```
typedef struct MPI_Status {
  int MPI_SOURCE;
  int MPI_TAG;
  int MPI_ERROR; };
```

- Funkcia `MPI_Get_count` vráti presný počet prijatých údajov

```
int MPI_Get_count(MPI_Status *status, MPI_Datatype
                  datatype, int *count)
```

Možnosť uviaznutia

```
int a[10], b[10], myrank;
MPI_Status status;
...
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
    MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
    MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
}
else if (myrank == 1) {
    MPI_Recv(b, 10, MPI_INT, 0, 2, MPI_COMM_WORLD);
    MPI_Recv(a, 10, MPI_INT, 0, 1, MPI_COMM_WORLD);
}
...
```

Možnosť uviaznutia

- Cyklické zasielanie správ: proces i posiela procesu i+1

```
int a[10], b[10], npes, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
            MPI_COMM_WORLD);
MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
      MPI_COMM_WORLD);
...
```

Možnosť uviaznutia

```
int a[10], b[10], npes, myrank;
MPI_Status status;
...
MPI_Comm_size(MPI_COMM_WORLD, &npes);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank%2 == 1) {
   MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
            MPI_COMM_WORLD);
   MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
            MPI_COMM_WORLD);
}
else {
   MPI_Recv(b, 10, MPI_INT, (myrank-1+npes)%npes, 1,
            MPI_COMM_WORLD);
   MPI_Send(a, 10, MPI_INT, (myrank+1)%npes, 1,
            MPI_COMM_WORLD);
}
...
```

Simultánne zasielanie a prijímanie správ

- Vzájomná výmena správ – MPI poskytuje nasledujúcu funkciu:

```
int MPI_Sendrecv(void *sendbuf, int sendcount,
    MPI_Datatype senddatatype, int dest, int
    sendtag, void *recvbuf, int recvcount,
    MPI_Datatype recvdatatype, int source, int recvtag,
    MPI_Comm comm, MPI_Status *status)
```

- Argumenty zahŕňajú argumenty „send“ a „receive“ funkcií. Možnosť použiť rovnaký „buffer“ pre „send“ a „receive“:

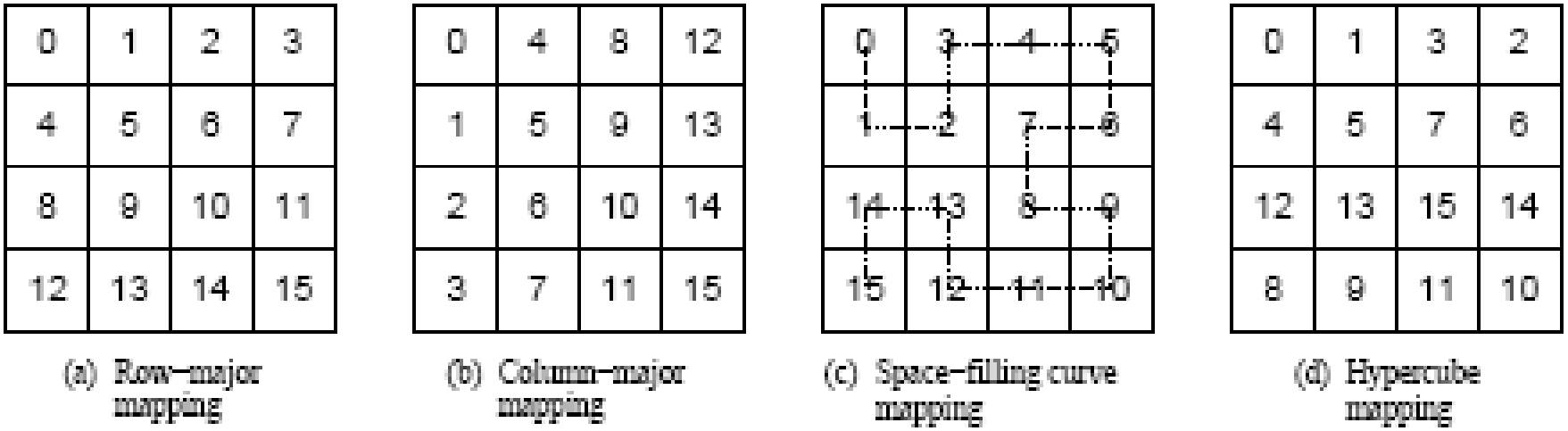
```
int MPI_Sendrecv_replace(void *buf, int count,
    MPI_Datatype datatype, int dest, int sendtag,
    int source, int recvtag, MPI_Comm comm,
    MPI_Status *status)
```

Topológie a mapovania

- MPI umožňuje programátorovi organizovať procesory do logických k-dimenzionálnych mriežok
- ID procesorov z `MPI_COMM_WORLD` môžu byť mapované do iných komunikátorov zodpovedajúcich viac-dimenzionálnych mriežok rôznymi spôsobmi
- Vhodnosť mapovania je určená programom a topológiou paralelného počítačového systému

Topológie a mapovania

- Mapovanie procesorov na 2D mriežku



(a) Row-major mapping

(b) Column-major mapping

(c) Space-filling curve mapping

(d) Hypercube mapping

Vytváranie a používanie kartézskych topológií

- Vytvorenie kartézskych topológií:

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims,
  int *dims, int *periods, int reorder,
  MPI_Comm *comm_cart)
```

- ID zo starého komunikátora – nový komunikátor s dimenziami podľa dims
- Každý procesor môže byť identifikovaný v novej kartézskej topológií vektorom domenzií

Vytváranie a používanie kartézskych topológií

- Zasielanie a prijímanie správ vyžaduje jednodimenzionálne id – rank do kartézskych súradníc a späť:

```
int MPI_Cart_coord(MPI_Comm comm_cart, int rank,
   int maxdims, int *coords)
int MPI_Cart_rank(MPI_Comm comm_cart, int *coords,
   int *rank)
```

- Posunutie – najčastejšia operácia v kartézskych topológiach – zistenie zdrojového a cieľového procesu:

```
int MPI_Cart_shift(MPI_Comm comm_cart, int dir,
   int s_step, int *rank_source, int *rank_dest)
```

Prelínanie komunikácie a výpočtov

- Neblokujúci „send“ a „recieve“:

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

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

- Návrat z funkcií skôr ako operácie zaslanie alebo prijatia skončené - funkcia `MPI_Test` testuje stav operácie:

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

- `MPI_Wait` čaká na dokončenie opeácie.

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

Riešenie uviaznutia

```
int a[10], b[10], myrank;
MPI_Status status;
...
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
if (myrank == 0) {
   MPI_Send(a, 10, MPI_INT, 1, 1, MPI_COMM_WORLD);
   MPI_Send(b, 10, MPI_INT, 1, 2, MPI_COMM_WORLD);
}
else if (myrank == 1) {
   MPI_Recv(b, 10, MPI_INT, 0, 2, &status, MPI_COMM_WORLD);
   MPI_Recv(a, 10, MPI_INT, 0, 1, &status, MPI_COMM_WORLD);
}
...
```

- Nahradenie blokujúcich operácií neblokujúcimi odstráni uviaznutie

Kolektívne komunikačné a výpočtové operácie

- MPI – široká množina funkcií pre bežné kolektívne operácie
- Každá kolektívna operácia definovaná na skupine danej komunikátorom
- Všetky procesy musia zavolať operáciu

Kolektívne komunikačné operácie

- Barierová synchronizácia:

```
int MPI_Barrier(MPI_Comm comm)
```

- Broadcast (jeden pre všetkých):

```
int MPI_Bcast(void *buf, int count, MPI_Datatype
  datatype, int source, MPI_Comm comm)
```

- Redukcia (všetky do jedného):

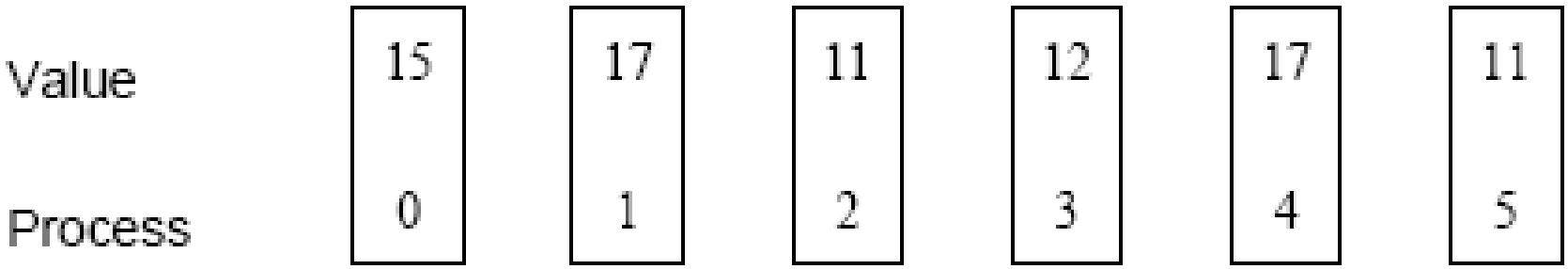
```
int MPI_Reduce(void *sendbuf, void *recvbuf, int
  count, MPI_Datatype datatype, MPI_Op op, int
  target, MPI_Comm comm)
```

Preddefinované operácie redukcie

Operation	Meaning	Datatypes
MPI_MAX	Maximum	C integers and floating point
MPI_MIN	Minimum	C integers and floating point
MPI_SUM	Sum	C integers and floating point
MPI_PROD	Product	C integers and floating point
MPI_LAND	Logical AND	C integers
MPI_BAND	Bit-wise AND	C integers and byte
MPI_LOR	Logical OR	C integers
MPI_BOR	Bit-wise OR	C integers and byte
MPI_LXOR	Logical XOR	C integers
MPI_BXOR	Bit-wise XOR	C integers and byte
MPI_MAXLOC	max-min value-location	Data-pairs
MPI_MINLOC	min-min value-location	Data-pairs

Kolektívne komunikačné operácie

- `MPI_MAXLOC` kombinuje dvojice (v_i, l_i) a vráti dvojicu (v, l) takú, že v je maximum zo všetkých v_i a l je zodpovedajúce l_i (ak viaceré, tak najmenšie l_i)
- `MPI_MINLOC` – minimum z v_i.




```
MinLoc(Value, Process) = (11, 2)
MaxLoc(Value, Process) = (17, 1)
```

Kolektívne komunikačné operácie

- Dátové typy pre `MPI_MAXLOC` a `MPI_MINLOC`

MPI dátový typ	C dátový typ
`MPI_2INT`	dvojica int-ov
`MPI_SHORT_INT`	short a int
`MPI_LONG_INT`	long a int
`MPI_LONG_DOUBLE_INT`	long double a int
`MPI_FLOAT_INT`	float a int
`MPI_DOUBLE_INT`	double a int

Kolektívne komunikačné operácie

- Ak je výsledok redukčnej operácie potrebný vo všetkých procesoch:

```
int MPI_Allreduce(void *sendbuf, void *recvbuf,
  int count, MPI_Datatype datatype, MPI_Op op,
  MPI_Comm comm)
```

- Výpočet čiastočnej redukcie:

```
int MPI_Scan(void *sendbuf, void *recvbuf,
  int count, MPI_Datatype datatype,
  MPI_Op op, MPI_Comm comm)
```

Kolektívne komunikačné operácie

- Operácia „zhromaždenia“ (gather):

```
int MPI_Gather(void *sendbuf, int sendcount,
  MPI_Datatype senddatatype, void *recvbuf,
  int recvcount, MPI_Datatype recvdatatype,
  int target, MPI_Comm comm)
```

- Operácia „zhromaždenia“ pre všetky procesy:

```
int MPI_Allgather(void *sendbuf, int sendcount,
  MPI_Datatype senddatatype, void *recvbuf,
  int recvcount, MPI_Datatype recvdatatype,
  MPI_Comm comm)
```

- Operácia „rozptýlenia“ (scatter):

```
int MPI_Scatter(void *sendbuf, int sendcount,
  MPI_Datatype senddatatype, void *recvbuf,
  int recvcount, MPI_Datatype recvdatatype,
  int source, MPI_Comm comm)
```

Kolektívne komunikačné operácie

- „Osobná“ komunikácia každý s každým:

```
int MPI_Alltoall(void *sendbuf, int sendcount,
        MPI_Datatype senddatatype, void *recvbuf,
        int recvcount, MPI_Datatype recvdatatype,
        MPI_Comm comm)
```

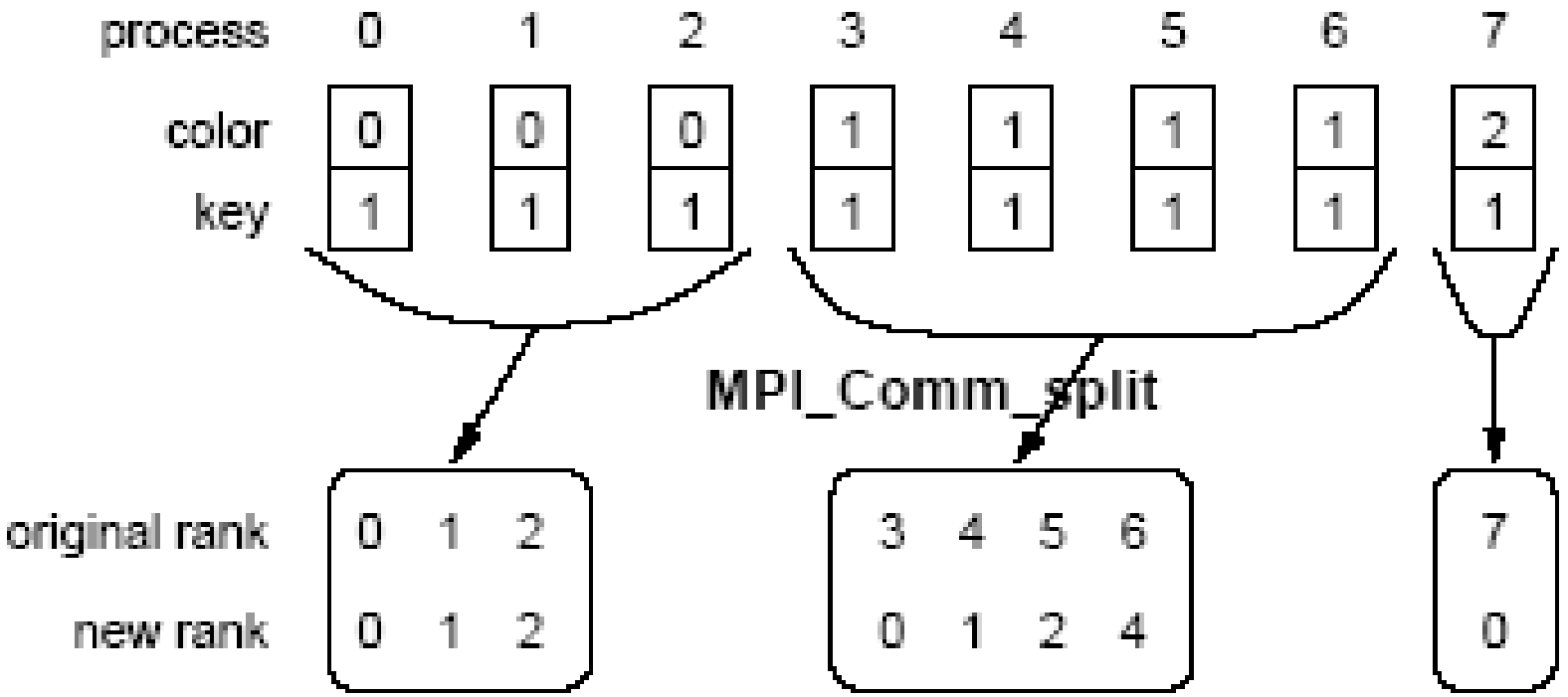
- Množina funkcií pre bežné kolektívne operácie - významné uľahčenie implementácie mnohých problémov

Skupiny a komunikátory

- V mnohých paralelných algoritmoch komunikácia musí byť zúžená na podmnožinu procesov
- MPI – mechanizmy na rozdeľovanie skupiny procesov zodpovedajúcich komunikátoru do podskupín každá zodpovedajúca inému komunikátoru
- Najjednoduchší takýto mechanizmus:

```
int MPI_Comm_split(MPI_Comm comm, int color,
  int key, MPI_Comm *newcomm)
```

Skupiny a komunikátory



Skupiny a komunikátory

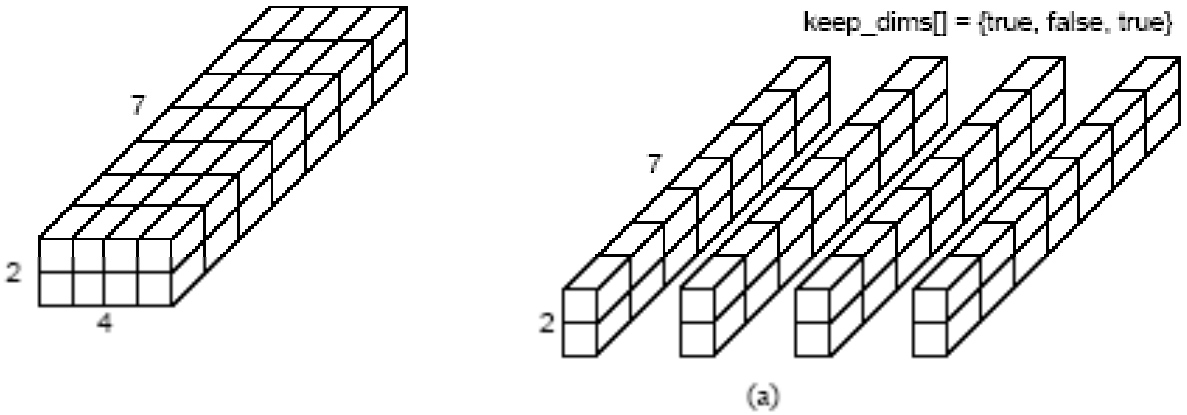
- V mnohých paralelných algoritmoch procesy sú organizované do virtuálnej mriežky, a v rôznych krokoch algoritmu komunikácia musí byť obmedzená na rôzne podmnožiny mriežky
- MPI poskytuje možnosť rozdeliť kartézsku topológiu do mriežok nižších dimenzií:

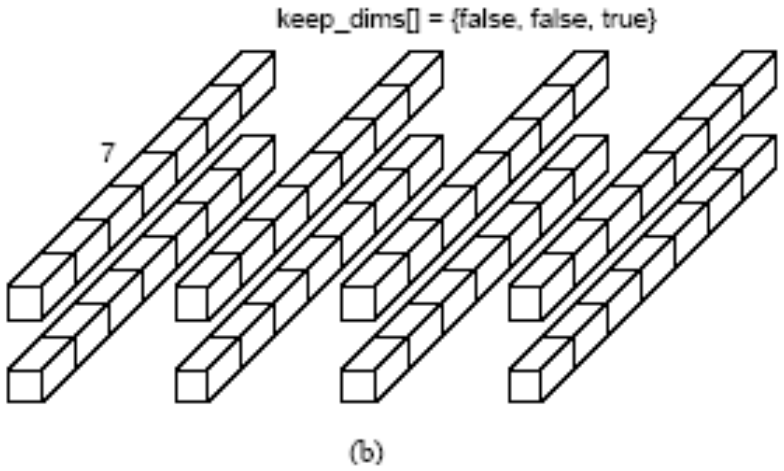
```
int MPI_Cart_sub(MPI_Comm comm_cart, int *keep_dims,
        MPI_Comm *comm_subcart)
```

- Ak `keep_dims[i]` je „true“ tak `i`-ta dimenzia je ponechaná v topológií
- Súradnice procesu v pod-mriežke vytvorenej pomocou `MPI_Cart_sub` môžu byť získané zo súradníc pôvodnej topológie ignorujúc súradnice „maskovaných“ dimenzií

Skupiny a komunikátory

- Rozdelenie kartézskej topológie 2x4x7 ba 4 podskupiny 2x1x7 a 8 podskupín 1x1x7





Zdroje

- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to ParallelComputing, 2nd Edition, Addison-Wesley 2003„Introduction to Parallel Computing“ http://www-users.cs.umn.edu/~karypis/parbook/

- Obrázky prevzaté z:
  - Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar. Introduction to ParallelComputing, 2nd Edition, Addison-Wesley 2003„Introduction to Parallel Computing“ http://www-users.cs.umn.edu/~karypis/parbook/