Curs 4

Programare Paralela si Distribuita

Message Passing Interface - MPI

MPI: Message Passing Interface

- MPI -documentation
  - http://mpi-forum.org

- Tutoriale:
  - https://computing.llnl.gov/tutorials/mpi/
  - …

MPI

- **specificatie de biblioteca(API) pentru programare paralela bazata pe transmitere de mesaje;**
- **propusa ca standard de producatori si utilizatori;**
- **gandita sa ofere performanta mare pe masini paralele dar si pe clustere;**

Istoric

- Apr 1992: Workshop on Standards for Message Passing in a Distributed Memory Environment, sponsored by the Center for Research on Parallel Computing, Williamsburg, Virginia=> Preliminary draft proposal

- Nov 1992: Minneapolis. MPI draft proposal (MPI1) from ORNL presented.
- Nov 1993: Supercomputing 93 conference - draft MPI standard presented.
- May 1994: Final version of MPI-1.0 released
- MPI-1.1 (Jun 1995)
- MPI-1.2 (Jul 1997)
- MPI-1.3 (May 2008).
- 1998: MPI-2 picked up where the first MPI specification left off, and addressed topics which went far beyond the MPI-1 specification.
- MPI-2.1 (Sep 2008)
- MPI-2.2 (Sep 2009)
- Sep 2012: The MPI-3.0 standard approved.
- MPI-3.1 (Jun 2015)
- MPI-4

Implementari

Exemple:

- **MPICH –**
- **Open MPI –**
- **IBM MPI –**

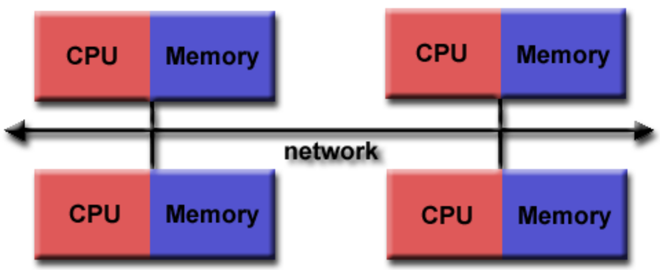
- **IntelMPI (not free)**

- **Links:**

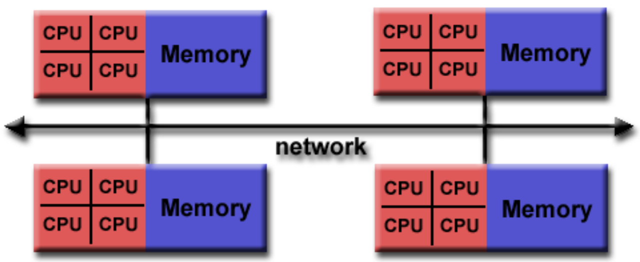
http://www.dcs.ed.ac.uk/home/trollius/www.osc.edu/mpi/

Modelul de programare

Initial doar pt DM



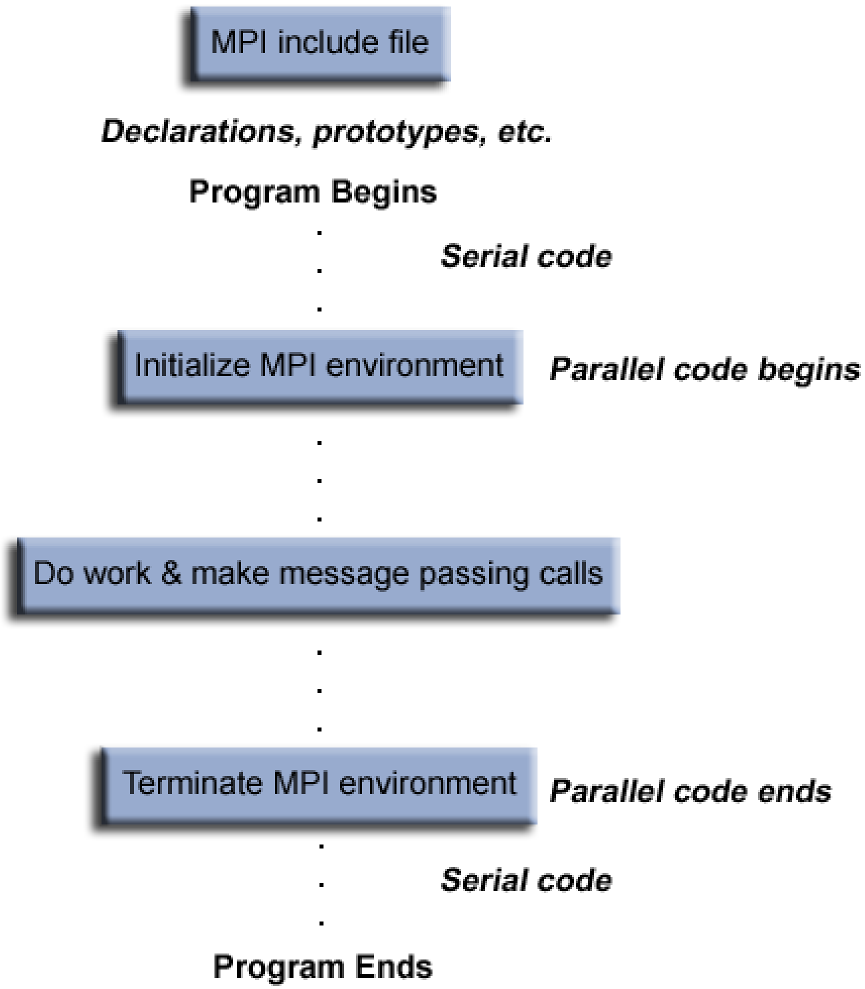
Ulterior si pt SM



Platforme suportate

- Distributed Memory
- Shared Memory
- Hybrid

Structura program MPI



Hello World in MPI

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

int main(int argc, char **argv)
{
    int  namelen, myid, numprocs;
    MPI_Init( &argc, &argv );
        MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);
        printf( "Process %d / %d  : Hello world\n", myid, numprocs);

    MPI_Finalize();
    return 0;
}
```

compilare
$ mpicc hello.c -o hello

executie
$ mpirun -np 4 hello

```
Process 0 / 4 : Hello world
Process 2 / 4 : Hello world
Process 1 / 4 : Hello world
Process 3 / 4 : Hello world
```

Formatul functiilor MPI

rc = MPI_Xxxxx(parameter, ...)

Exemplu:

rc=MPI_Bsend(&buf, count, type, dest, tag, comm)

Cod de eroare: Intors ca "rc". MPI_SUCCESS pentru succes

Comunicatori si grupuri

- **MPI foloseste obiecte numite comunicatori si grupuri pentru a defini ce colectii de procese pot comunica intre ele. Cele mai multe functii MPI necesita specificarea unui comunicator ca argument.**

- **Pentru simplitate exista comunicatorul predefinit care include toate procesele MPI numit MPI_COMM_WORLD.**

Rangul unui proces

- **Intr-un comunicator, fiecare proces are un identificator unic, rang. El are o valoare intreaga, unica in sistem, atribuita la initializarea mediului.**

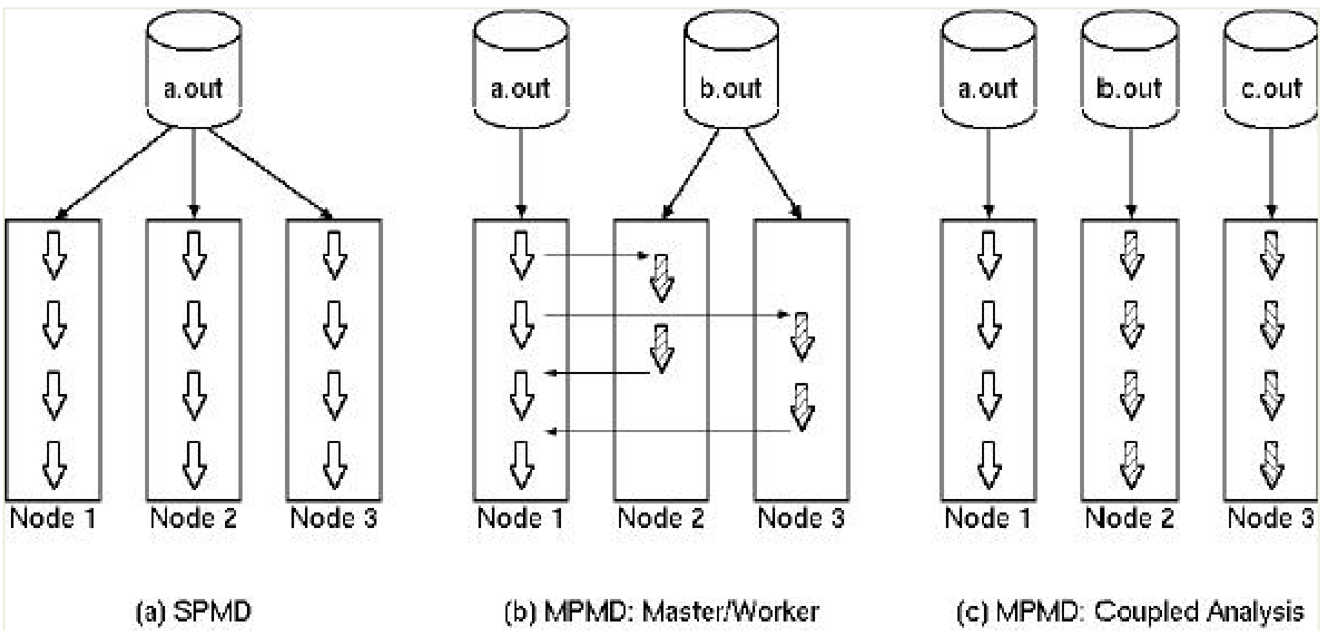
- **Utilizat pentru a specifica sursa si destinatia mesajelor.**

- **De asemenea se foloseste pentru a controla executia programului:**
  e.g: daca rank=0 fa ceva / daca rank=1 fa altceva , etc.

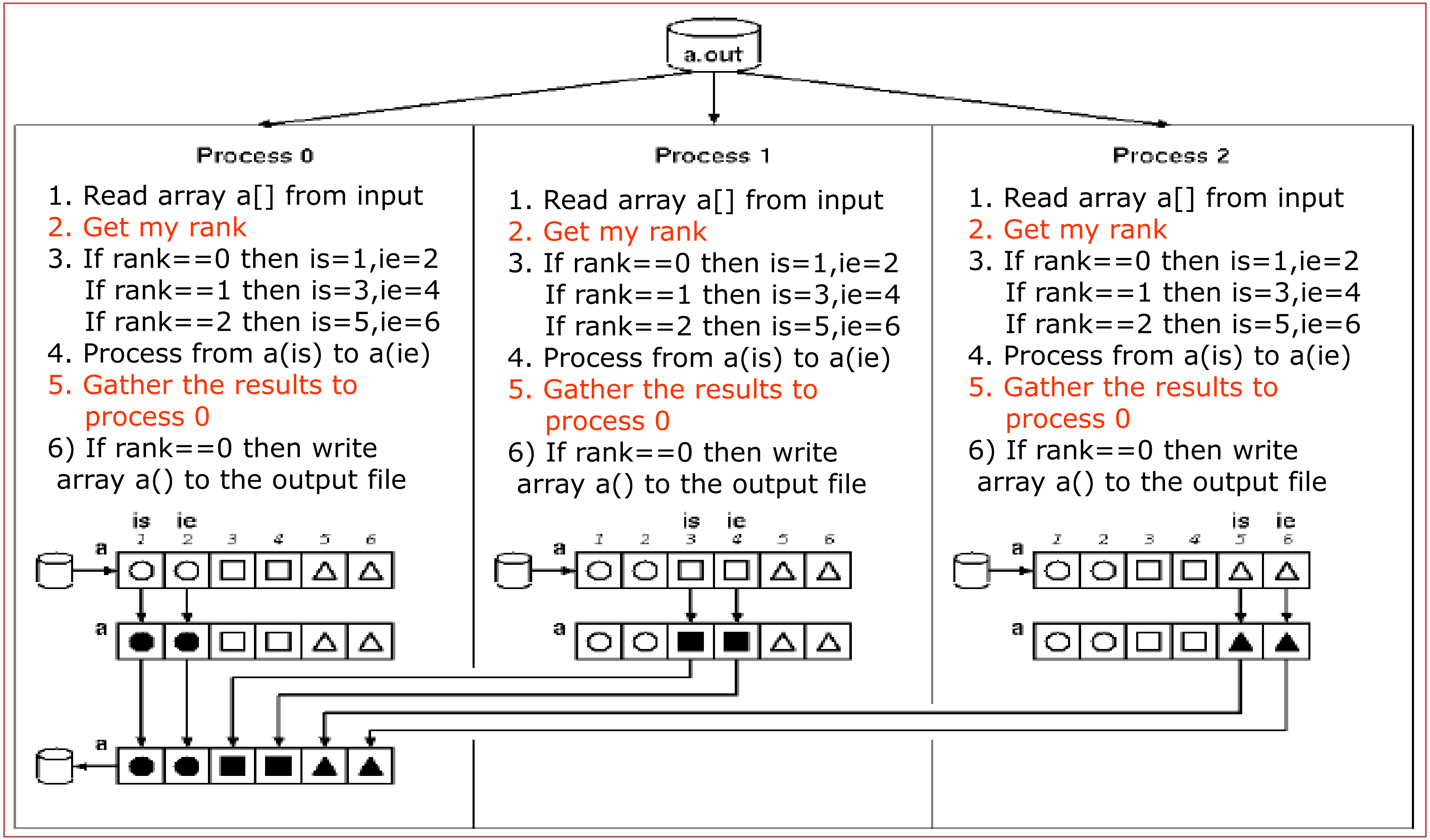
SPMD/MPMD

Modele de calcul paralel in sisteme cu memorie distribuita

SPMD (Single Program Multiple Data) *(Fig a)*
MPMD (Multiple Program Multiple Data) (Fig *b*,*c*)



Modelul SPMD – Single Program Multiple Data



MPI. Clase de functii

- *Functii de management mediu*
- *Functii de comunicatie punct-la-punct*
- *Operatii colective*
- *Operatii de lucru cu fisiere*
- *Grupuri de procese/Comunicatori*
- *Topologii (virtuale) de procese*

Functii de management mediu

- ***initializare, terminare, interogare mediu***

- ***MPI_Init*** ***– initializare mediu***

  MPI_Init (&argc,&argv)
  MPI_INIT (ierr)

- ***MPI_Comm_size*** ***– determina numarul de procese din grupul asociat unui com.***

  MPI_Comm_size (comm,&size)
  MPI_COMM_SIZE (comm,size,ierr)

- ***MPI_Comm_rank*** ***– determina rangul procesului apelant in cadrul unui com.***

  MPI_Comm_rank (comm,&rank)
  MPI_COMM_RANK (comm,rank,ierr)

- ***MPI_Abort*** ***–opreste toate procesele asociate unui comunicator***

  MPI_Abort (comm,errorcode)
  MPI_ABORT (comm,errorcode,ierr)

- ***MPI_Finalize*** ***-finalizare mediu MPI***

  MPI_Finalize ()
  MPI_FINALIZE (ierr)

Exemplu

- ***initializare, terminare, interogare mediu***

```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char *argv[] )
{
int numtasks, rank, rc;
rc = MPI_Init(&argc,&argv);
if (rc != MPI_SUCCESS) {
      printf ("Error starting MPI program. Terminating.\n");
      MPI_Abort(MPI_COMM_WORLD, rc);
}
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf ("Number of tasks= %d My rank= %d\n", numtasks,rank);
      /******* do some work *******/
 MPI_Finalize();
}
```

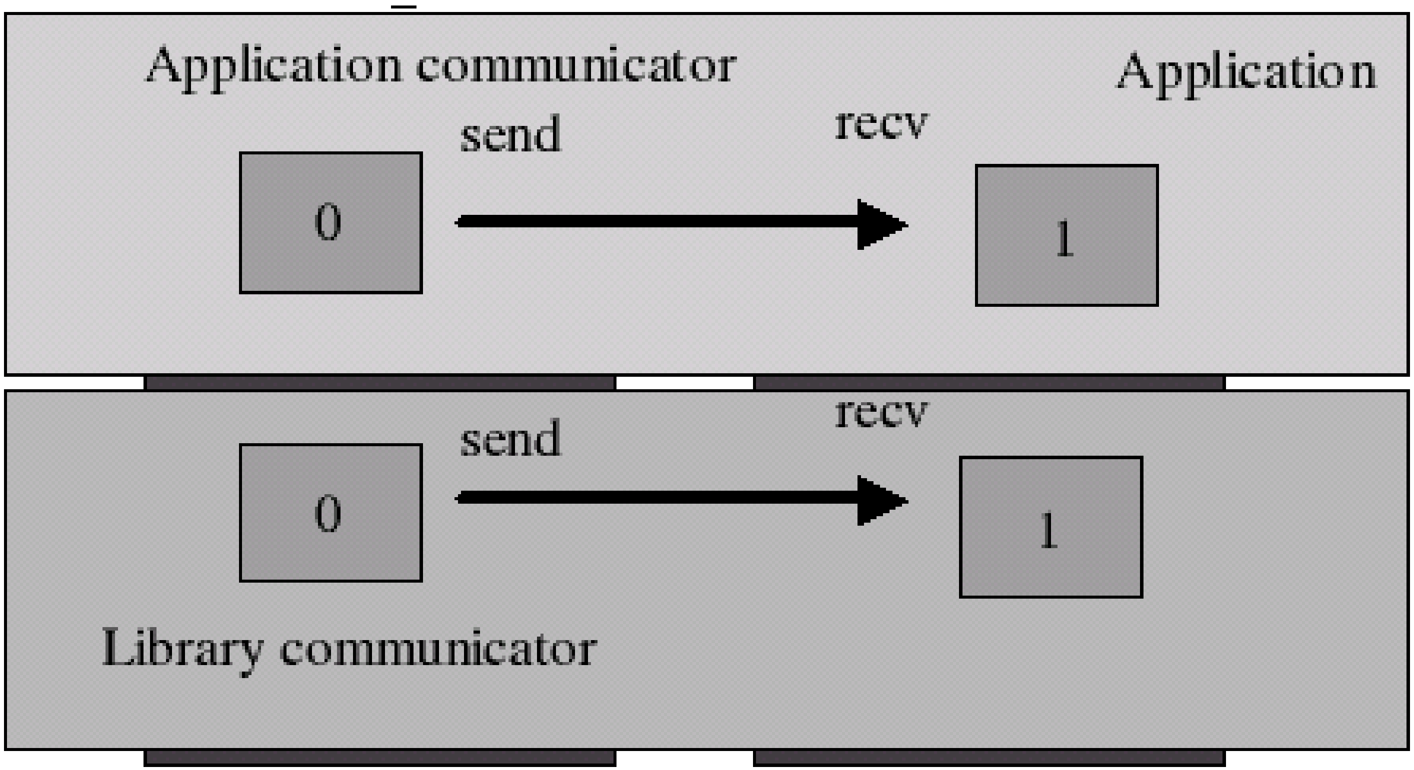
Executie

$ mpirun -np 4 test

>mpiexec –n 4 test

- se creeaza 4 instante de process si fiecare executa acelasi program executabil
- Intre *init* si *finalize* se pot apela functii MPI
  - context in care interactiune dinte procese este posibila prin functiile MPI

Comunicatii send-recv



Determinism si nedeterminism

- Modele de programare paralela bazate pe transmitere de mesaje sunt implicit nedeterministe: ordinea in care mesajele transmise de la doua procese A si B la al treilea C, nu este definita.
  - Este responsabilitatea programatorul de a asigura o executie determinista, daca aceasta se cere.

- In modelul bazat pe transmitere pe canale de comunicatie, determinismul este garantat prin definirea de canale separate pentru comunicatii diferite, si prin asigurarea faptului ca fiecare canal are doar un singur „scriitor” si un singur „cititor”.

Comunicatie punct-la-punct

Transferul de mesaje intre 2 taskuri MPI distincte intr-un anumit sens.

• *Tipuri de operatii punct-la-punct*

Exista diferite semantici pentru operatiile de *send/receive* :

- Synchronous send
- Blocking send / blocking receive
- Non-blocking send / non-blocking receive
- Buffered send
- Combined send/receive
- "Ready" send

•o rutina *send* poate fi utilizata cu orice alt tip de rutina *receive*

•rutine MPI asociate (*wait,probe*)

Comunicatie punct-la-punct- *Operatii blocate vs ne-blocante*

send are 4 moduri de comunicare:
- **Standard**
- **Buffered**
- **Synchronous**
- **Ready**

Fiecare poate fi ***blocking*** or ***non-blocking***

receive are 2 moduri de comunicare
- **blocking**
- **non-blocking**

Operatii blocante

O operatie de *send blocanta* va “returna”(se va finaliza) doar atunci cand zona de date folosita pentru trimitere poate fi reutilizata, fara sa afecteze datele primite de destinatar.

Operatii ne-blocante

Returneaza controlul imediat, notifica libraria care se va ocupa de transfer. Exista functii speciale de asteptare/interogare a statusului transferului.

Comunicatie punct-la-punct

Operatii blocate vs ne-blocante

Blocking send	MPI_Send(buffer,count,type,dest,tag,comm)
Blocking receive	MPI_Recv(buffer,count,type,source,tag,comm, status)
Blocking Probe	MPI_Probe (source,tag,comm,&status)
Non-blocking send	MPI_Isend(buffer,count,type,dest,tag,comm, request)
Non-blocking receive	MPI_Irecv(buffer,count,type,source,tag,comm, request)
Wait	MPI_Wait (&request,&status)
Test	MPI_Test (&request,&flag,&status)
Non-blocking probe	MPI_Iprobe (source,tag,comm,&flag,&status)

MPI_Probe allows checking of incoming messages, without actual receipt of them.
The user can then decide how to receive them, based on the information returned by the probe in the status variable.
For example, the user may allocate memory for the receive buffer, according to the length of the probed message.

Determinism in MPI

- Pentru obtinerea determinismului in MPI, sistemul trebuie sa adauge anumite informatii datelor pe care programul trebuie sa le trimita. Aceste informatii aditionale formeaza un asa numit "plic" al mesajului.

- In MPI acesta contine urmatoarele informatii:
  - un comunicator.
  - rangul procesului transmitator
  - rangul procesului receptor
  - **un tag (marcaj)-**– este un intreg specificat de catre programator, pentru a se putea face distinctie intre mesaje receptionate de la acelasi proces transmitator.

- Comunicatorul stabileste grupul de procese in care se face transmiterea.

MPI Basic (Blocking) Send

MPI_SEND (start, count, datatype, dest, tag, comm)

- mesajul descris de (`start, count, datatype`)
- `dest – id process destinatie`

MPI Basic (Blocking) Recv

MPI_Recv (&buf,count,datatype,source,tag,comm,&status)

- MPI permite omiterea specificarii procesului de la care trebuie sa se primeasca mesajul, caz in care se va folosi constanta predefinita: MPI_ANY_SOURCE. (pt send- procesul destinatie trebuie precizat intotdeauna exact.)
  - Marcajul – tagul – mesajului poate fi inlocuit de MPI_ANY_TAG, daca se considera ca lipsa lui nu poate duce la ambiguitate.

- Ultimul parametru al functiei MPI_Recv, **status,** returneaza informatii despre datele care au fost receptionate in fapt. Reprezinta o referinta la o inregistrare cu doua campuri: unul pentru sursa si unul pentru tag. Astfel daca sursa a fost MPI_ANY_SOURCE, in status se poate gasi rangul procesului care a trimis de fapt mesajul respective.

MPI Data Types

- **MPI_CHAR** signed char
- **MPI_SHORT** signed short int
- **MPI_INT** signed int
- **MPI_LONG** signed long int
- **MPI_LONG_LONG_INT**
- **MPI_LONG_LONG** signed long long int
- **MPI_SIGNED_CHAR** signed char
- **MPI_UNSIGNED_CHAR** unsigned char
- **MPI_UNSIGNED_SHORT** unsigned short int
- **MPI_UNSIGNED** unsigned int
- **MPI_UNSIGNED_LONG** unsigned long int
- **MPI_UNSIGNED_LONG_LONG** unsigned long long int
- **MPI_FLOAT** float
- **MPI_DOUBLE** double
- **MPI_LONG_DOUBLE** long double
- …

Exemplu operatii blocante

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[]) {
int numtasks, rank, dest, source, rc, count, tag=1;
char inmsg, outmsg='x';
MPI_Status Stat;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank == 0) {
      dest = source = 1;
      rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag,
      MPI_COMM_WORLD);
      rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag,
      MPI_COMM_WORLD, &Stat);
}
```

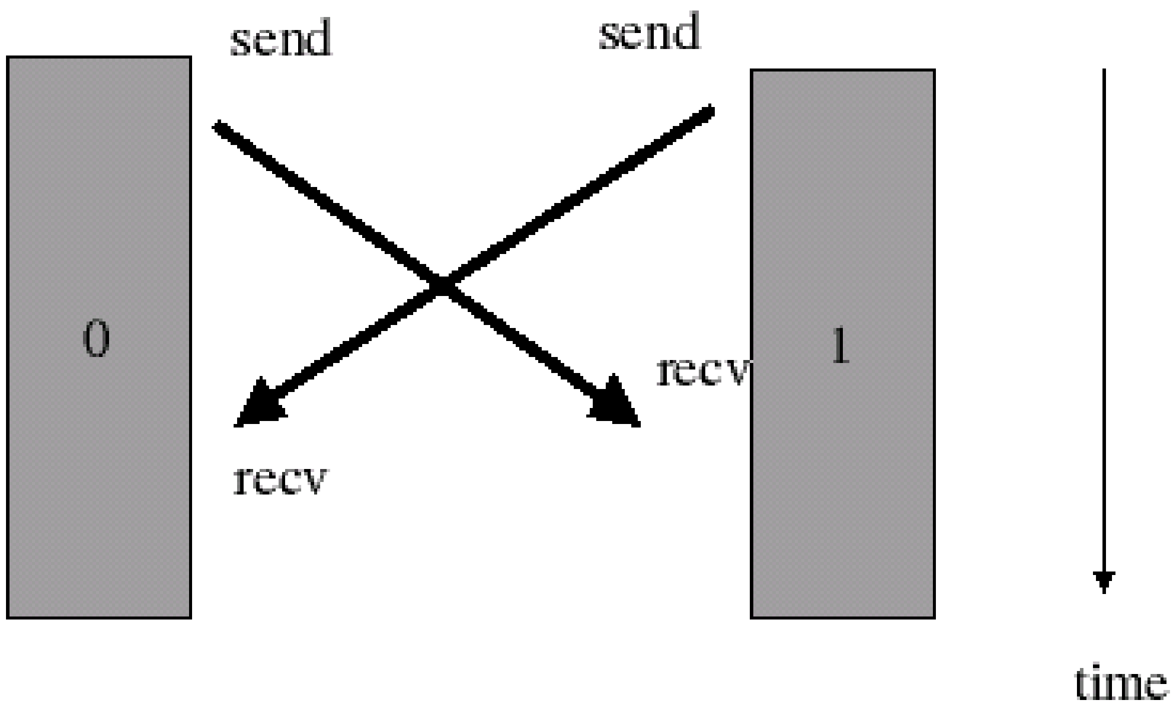
Exemplu operatii blocante (cont)

```
else if (rank == 1){
       dest = source = 0;
       rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag,
                    MPI_COMM_WORLD, &Stat);
       rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag,
                     MPI_COMM_WORLD);
}
rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
printf("Task %d: Received %d char(s) from task %d with tag %d
       \n", rank, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
MPI_Finalize();
}
```

MPI deadlocks

- Scenariu:
  - Presupunem ca avem doua procese in cadrul carora comunicatia se face dupa urmatorul protocol
    - Primul proces trimite date catre cel de-al doilea si asteapta raspunsuri de la acesta.
    - Cel de-al doilea proces trimite date catre primul si apoi asteapta raspunsul de la acesta.
  - Daca bufferele sistem nu sunt suficiente se poate ajunge la deadlock. Orice comunicatie care se bazeaza pe bufferele sistem este nesigura din punct de vedere al deadlock-ului.
  - In orice tip de comunicatie care include cicluri pot apare deadlock-uri.

Deadlock



OPERATII COLECTIVE

Operatii colective

• *Operatiile colective implica toate procesele din cadrul unui comunicator. Toate procesele sunt membre ale comunicatorului initial, predefinit MPI_COMM_WORLD.*

Tipuri de operatii colective:

•Sincronizare: procesele asteapta toti membrii grupului sa ajunga in punctul de jonctiune.

•Transfer de date - broadcast, scatter/gather, all to all.

•Calcule colective (reductions) – un membru al grupului colecteaza datele de la toti ceilalti membrii si realizeaza o operatie asupra acestora (min, max, adunare, inmultire, etc.)

Observatie:
Toate operatiile colective sunt blocante!

Operatii colective

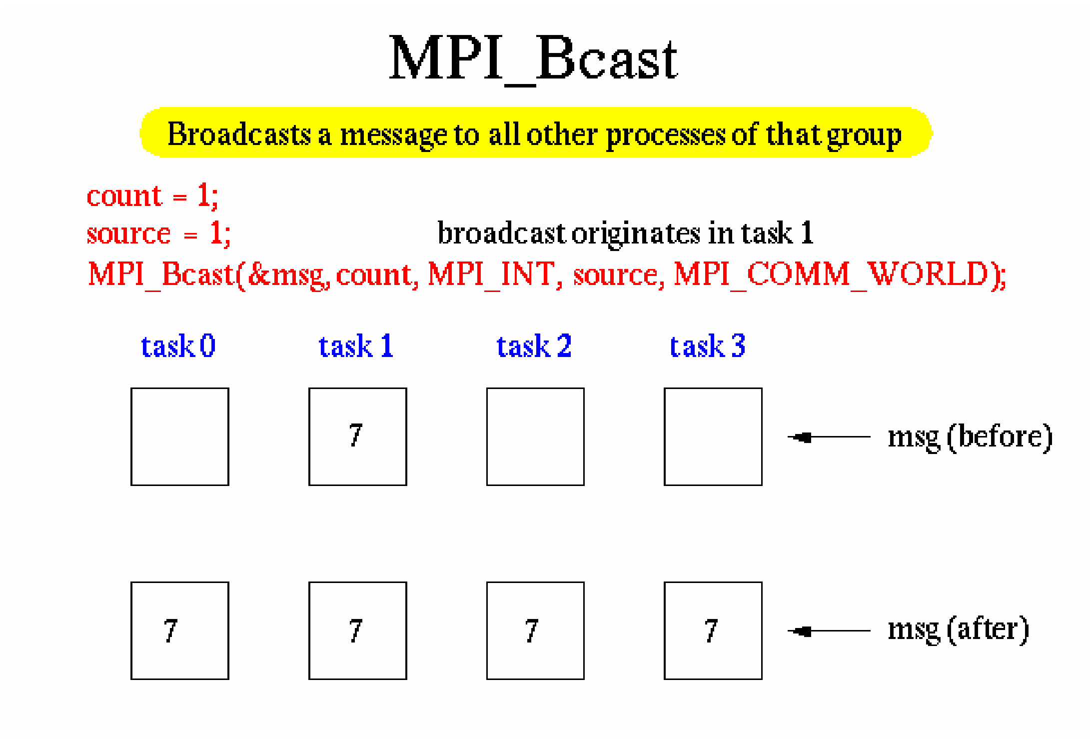
MPI_Barrier

MPI_Barrier (comm)

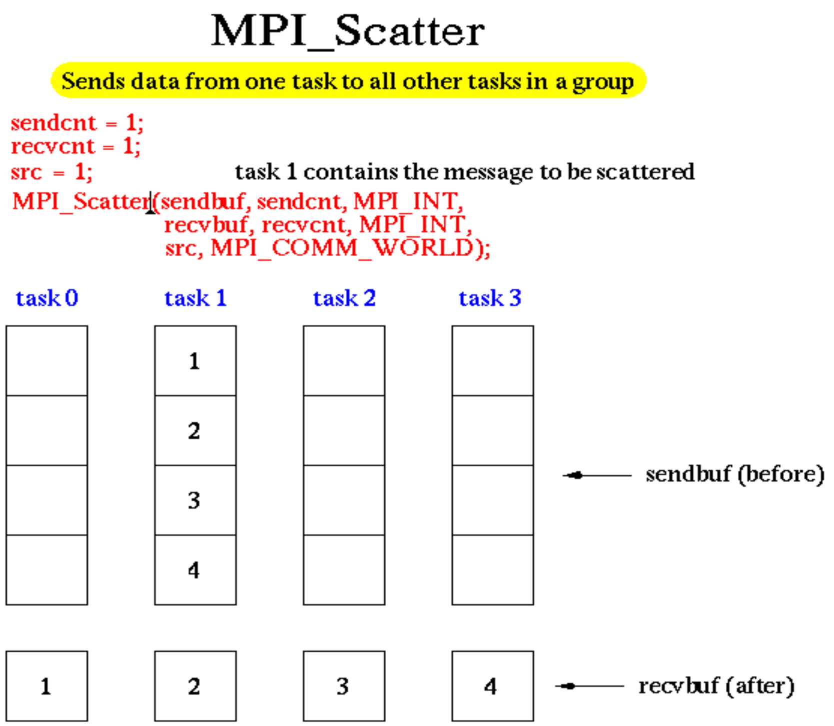
MPI_BARRIER (comm,ierr)

Fiecare task se va bloca in acest apel pana ce toti membri din grup au ajuns in acest punct

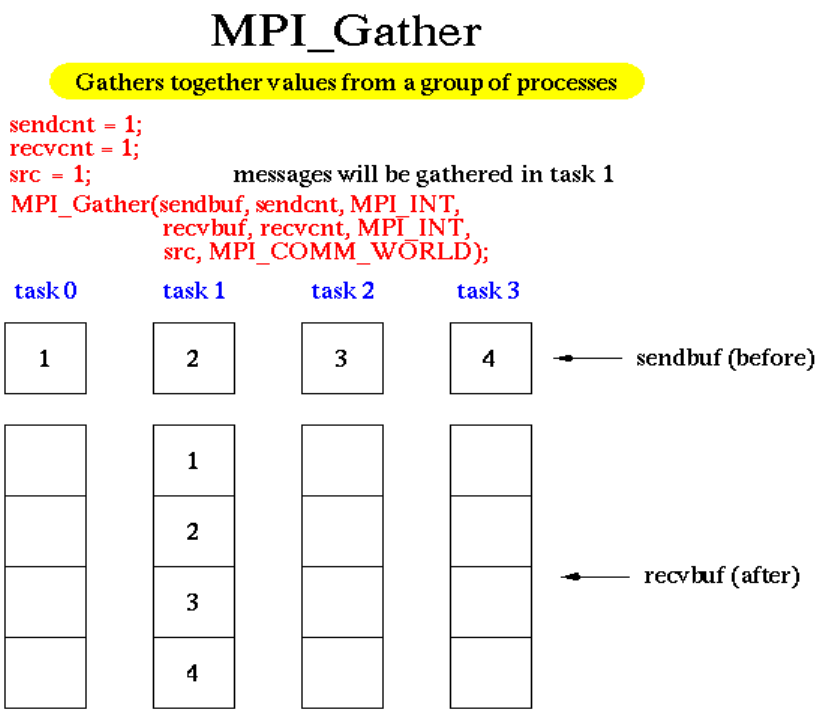
Operatii colective



Operatii colective



Operatii colective



Operatii colective