Curs 5

MPI

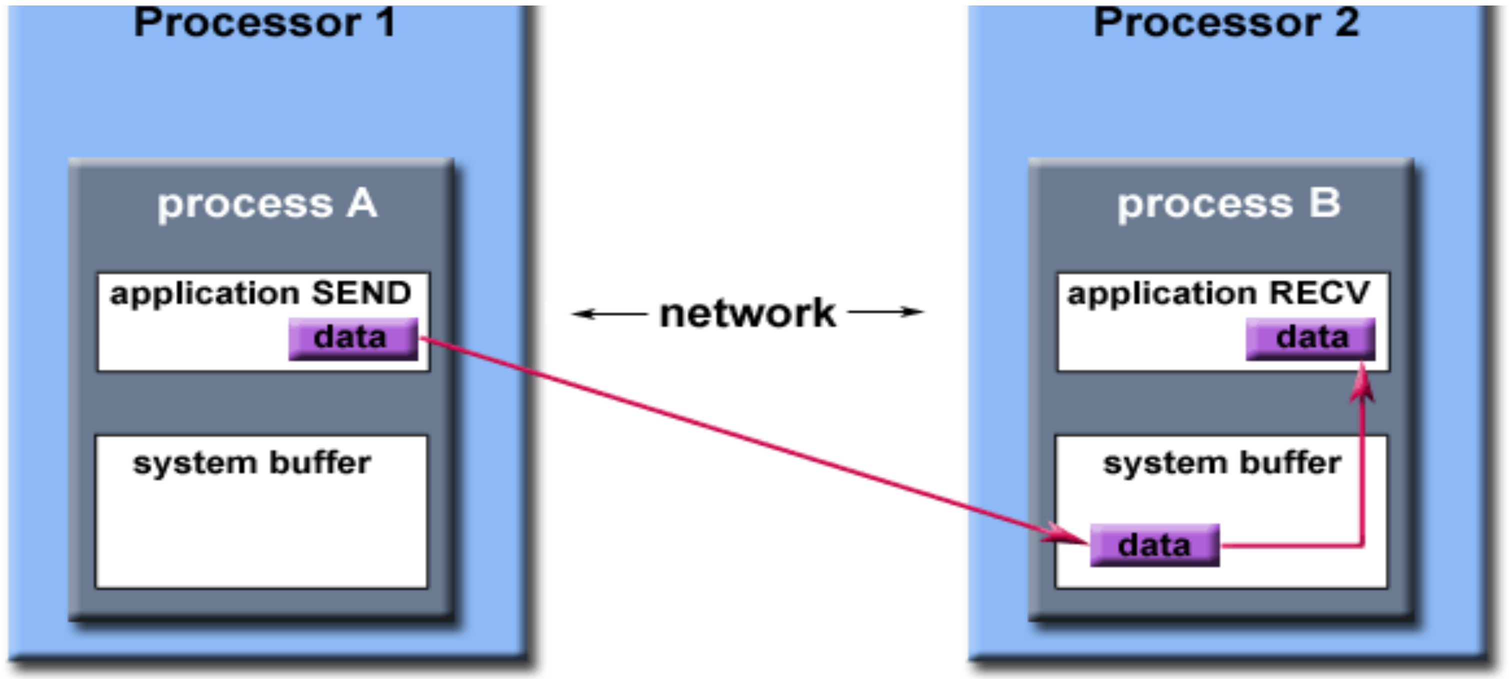
- *Operatii blocate vs ne-blocante*
- operatii de comunicare colectiva

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)

Folosire buffere
(decizie a implementarii MPI)



Path of a message buffered at the receiving process

MPI_Irecv

- Starts a standard-mode, nonblocking receive.

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

- Nonblocking calls allocate a communication request object and associate it with the request handle (the argument request).
- The request can be used later to query the status of the communication or wait for its completion.
- ***A nonblocking receive call indicates that the system may start writing data into the receive buffer.***
- ***The receiver should not access any part of the receive buffer after a nonblocking receive operation is called, until the receive completes***.
- A receive request can be determined being completed by calling the MPI_Wait, MPI_Waitany, MPI_Test, or MPI_Testany with request returned by this function.

MPI_Isend

- -Starts a standard-mode, nonblocking send.

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

- MPI_Isend starts a standard-mode, nonblocking send.
- Nonblocking calls allocate a communication request object and associate it with the request handle (the argument request).
- The request can be used later to query the status of the communication or wait for its completion.
- ***A nonblocking send call indicates that the system may start copying data out of the send buffer.***
- ***The sender should not modify any part of the send buffer after a nonblocking send operation is called, until the send completes.***
- A send request can be determined being completed by calling the MPI_Wait, MPI_Waitany, MPI_Test, or MPI_Testany with request returned by this function.

MPI_Wait

- Waits for an MPI send or receive to complete.

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

- A call to MPI_Wait returns when the operation identified by request is complete.
- If the communication object associated with this request was created by a nonblocking send or receive call, then the object is deallocated by the call to MPI_Wait and the request handle is set to MPI_REQUEST_NULL.
- The call returns, in status, information on the completed operation.

- If your application does not need to examine the *status* field, you can save resources by using the predefined constant MPI_STATUS_IGNORE as a special value for the *status* argument.

MPI_Test

- Tests for the completion of a specific send or receive.

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

- A call to MPI_Test returns flag = true if the operation identified by request is complete.
  - In this case, the status object is set to contain information on the completed operation;
  - if the communication object was created by a nonblocking send or receive, then it is deallocated and the request handle is set to MPI_REQUEST_NULL.
- The call returns flag = false, otherwise.
  - In this case, the value of the status object is undefined.
- MPI_Test is a local operation.
- If your application does not need to examine the *status* field, you can save resources by using the predefined constant MPI_STATUS_IGNORE as a special value for the *status* argument.

- The functions MPI_Wait and MPI_Test can be used to complete both sends and receives.

Exemple

- Hello world – async messages

- p procese
- procesele cu id > 0 trimit mesaje catre procesul 0
- procesul 0 le preia in ordinea venirii – le adauga intr-un string
- dupa ce a preluat toate mesajele afiseaza stringul in care aceastea au fost concatenate

MPI_Sendrecv

- **trimite si receptioneaza un mesaj**

MPI_Sendrecv (&sendbuf,sendcount,sendtype, dest, sendtag, &recvbuf, recvcount, recvtype, source, recvtag, comm, &status)

- *Send a message and post a receive before blocking!!!*
- *Will block until the sending application buffer is free for reuse and until the receiving application buffer contains the received message.*
- *useful for executing a shift operation across a chain of processes*

A quick overview of other MPI's send modes

MPI has a number of different "send modes." These represent different choices of buffering (where is the data kept until it is received) and synchronization (when does a send complete). (*In the following, we use "send buffer" for the user-provided buffer to send.*)

- **MPI_Send**
  - **MPI_Send will not return until you can use the send buffer. It may or may not block (it is allowed to buffer, either on the sender or receiver side, or to wait for the matching receive).**
- **MPI_Bsend**
  - **May buffer; returns immediately and you can use the send buffer. A late add-on to the MPI specification. Should be used only when absolutely necessary.**
- **MPI_Ssend**
  - **will not return until matching receive posted**
- **MPI_Rsend**
  - **May be used ONLY if matching receive already posted. User responsible for writing a correct program.**
- **MPI_Isend**
  - **Nonblocking send. But not necessarily asynchronous. You can NOT reuse the send buffer until either a successful, wait/test or you KNOW that the message has been received (see MPI_Request_free). Note also that while the I refers to immediate, there is no performance requirement on MPI_Isend. An immediate send must return to the user without requiring a matching receive at the destination. An implementation is free to send the data to the destination before returning, as long as the send call does not block waiting for a matching receive. Different strategies of when to send the data offer different performance advantages and disadvantages that will depend on the application.**
- **MPI_Ibsend**
  - **buffered nonblocking**
- **MPI_Issend**
- **Synchronous nonblocking. Note that a Wait/Test will complete only when the matching receive is posted.**
- **MPI_Irsend**
  - **As with MPI_Rsend, but nonblocking.**

Note that "nonblocking" refers ONLY to whether the data buffer is available for reuse after the call. No part of the MPI specification, for example, mandates concurent operation of data transfers and computation.

Some people have expressed concern about not having a single "perfect" send routine. But note that in general you can't write code in Fortran that will run at optimum speed on both Vector and RICS/Cache machines without picking different code for the different architectures. MPI at least lets you express the different algorithms, just like C or Fortran.

Recommendations

The best performance is likely if you can write your program so that you could use just MPI_Ssend; in that case, an MPI implementation can completely avoid buffering data. Use MPI_Send instead; this allows the MPI implementation the maximum flexibility in choosing how to deliver your data. (Unfortunately, one vendor has chosen to have MPI_Send emphasize buffering over performance; on that system, MPI_Ssend may perform better.) If nonblocking routines are necessary, then try to use MPI_Isend or MPI_Irecv. Use MPI_Bsend only when it is too inconvienent to use MPI_Isend. The remaining routines, MPI_Rsend, MPI_Issend, etc., are rarely used but may be of value in writing system-dependent message-passing code entirely within MPI.

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

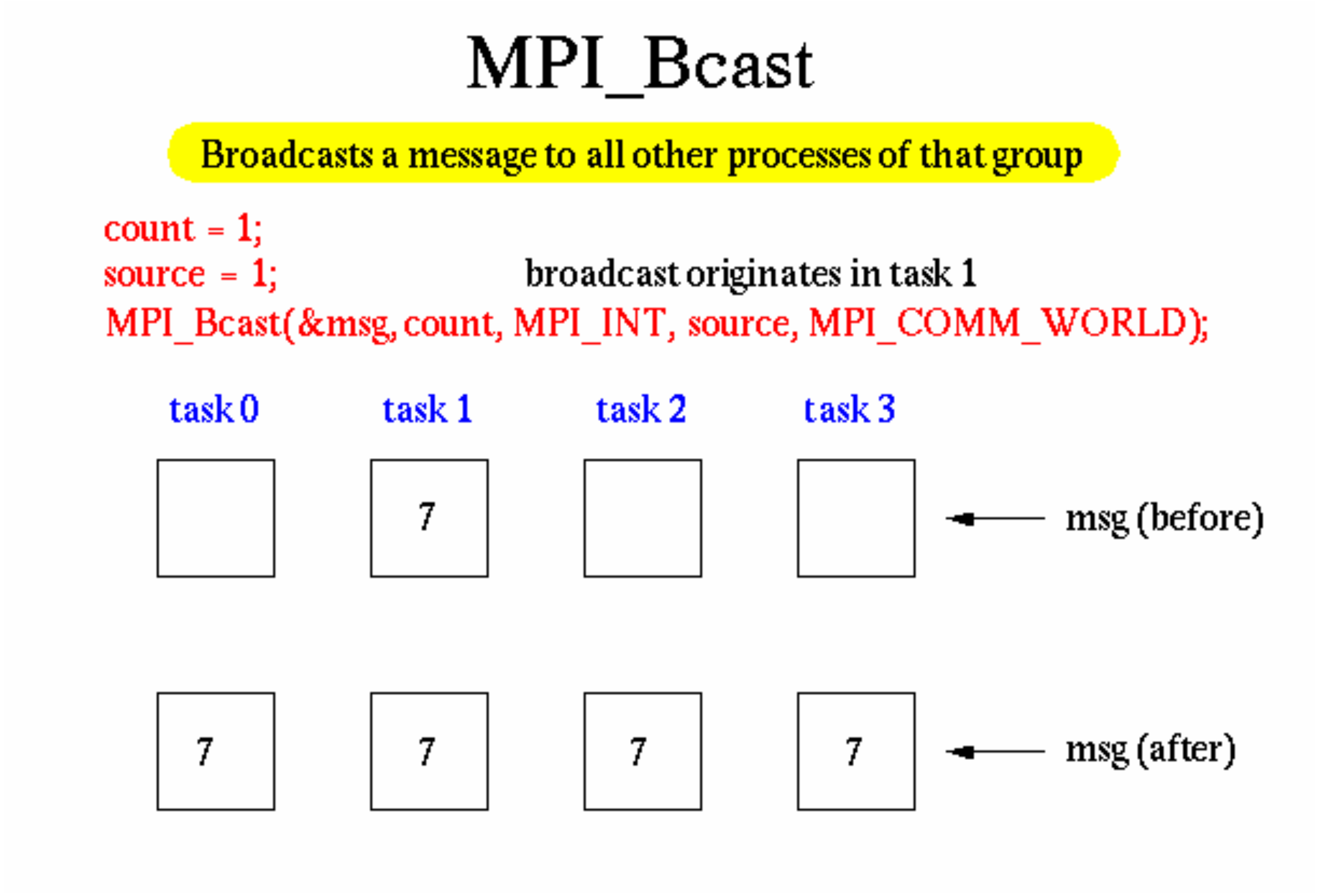
Operatii colective

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



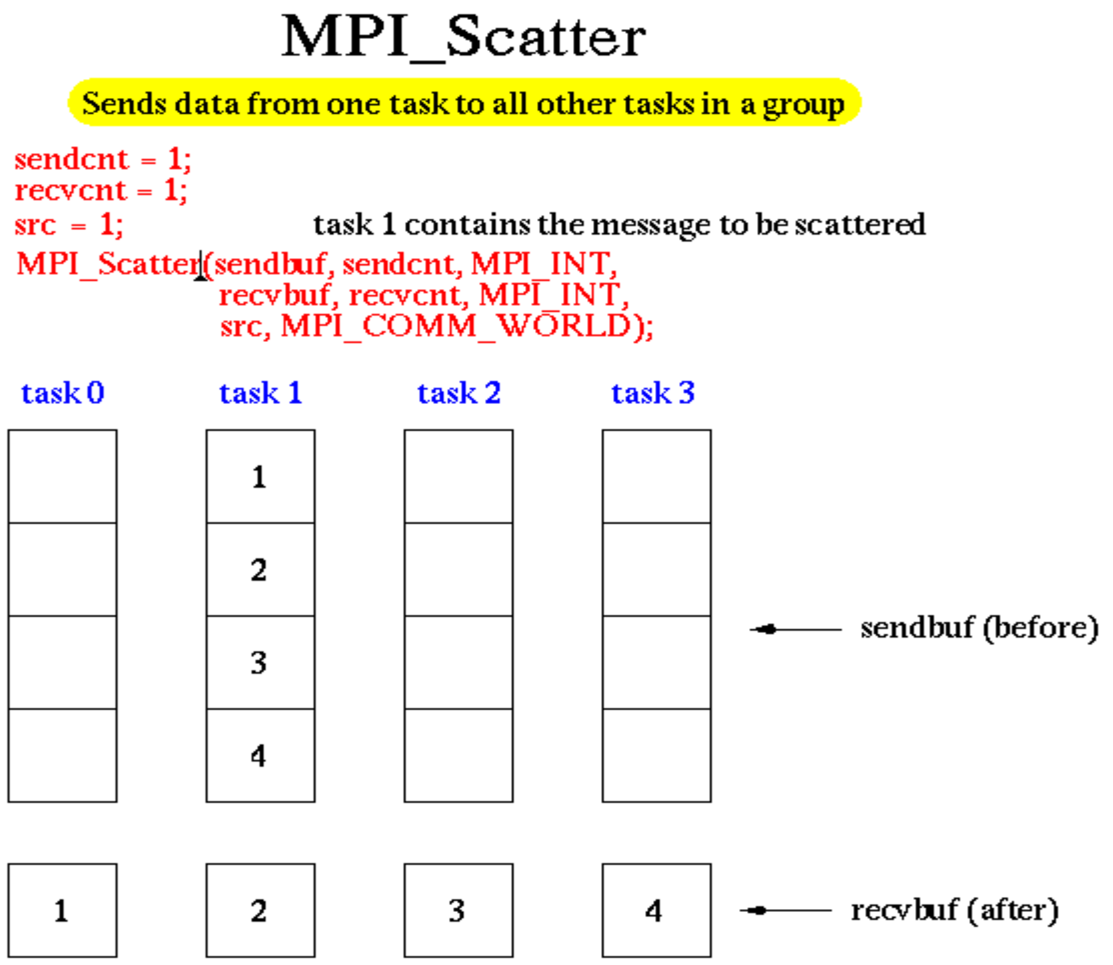
MPI_Bcast

Broadcasts a message to all other processes of that group

```
count = 1;
source = 1;
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```

broadcast originates in task 1

Operatii colective

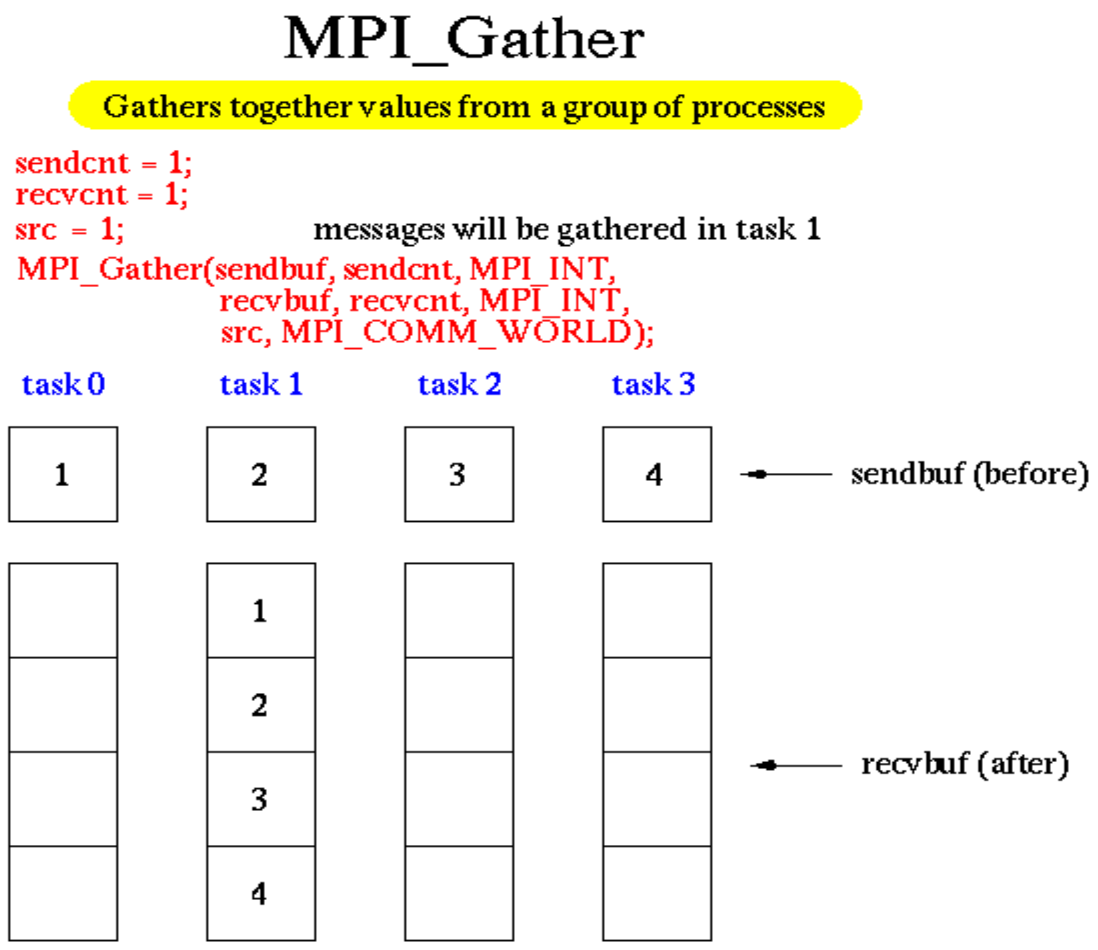


MPI_Scatter

Sends data from one task to all other tasks in a group

```
sendcnt = 1;
recvcnt = 1;
src  = 1;                task 1 contains the message to be scattered
MPI_Scatter(sendbuf, sendcnt, MPI_INT,
            recvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```

Operatii colective

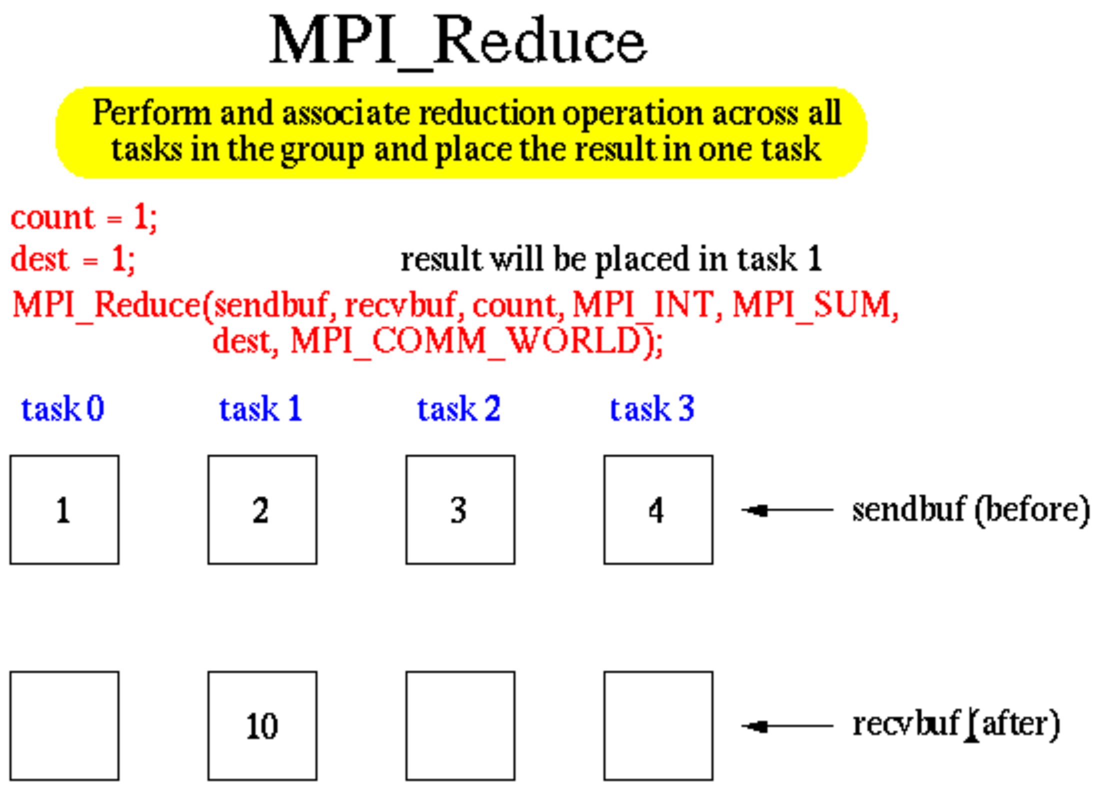


MPI_Gather

Gathers together values from a group of processes

```
sendcnt = 1;
recvcnt = 1;
src = 1;                messages will be gathered in task 1
MPI_Gather(sendbuf, sendcnt, MPI_INT,
           recvbuf, recvcnt, MPI_INT,
           src, MPI_COMM_WORLD);
```

Operatii colective



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);
```