Parallel Programming

Brief overview

MPI(Message Passing Interface)

- MPI is a specification for the developers and users of message passing libraries.
- Portability - There is no need to modify your source code when you port your application to a different platform that supports (and is compliant with) the MPI standard
- In practice, MPI is a set of functions or subroutines used for exchanging data between processes.
- MPI has around 6 basic functions, and around 440+ [in MPI-3] functions/routines in total.
- Interface specifications have been defined for C/C++ and Fortran programs.

MPI Function/Subroutine

- MPI_Init (): is the 1st function to call, initialize MPI variables. Creates MPI_COMM_WORLD communicator, which is list of all the connections among nodes.
- MPI_Finilize(): This is the last function to call which cleans up any job left by MPI.

Example:

```
#include "mpi.h“                    // INCLUDE "mpif.h“ for fortran
#include <stdio.h>                  // part of regular c program
void main(argc, argv)
{
  MPI_Init( &argc, &argv );         // Starts MPI
      ……..
      ……..
  MPI_Finalize();                   // Ends MPI
}
```

MPI Function/Subroutine (Contd.)

- MPI_Comm_size(MPI_COMM_WORLD, ***nprocs***, ierror) //ierror is not in C, but in fortran
  - returns the number of nodes that the code is running on, as specified in the mpirun command (eg **mpirun** -np 8 myprog | tee output.txt)

- MPI_Comm_rank (MPI_COMM_WORLD, myproc, ierror): returns node number, range 0 to (np-1).

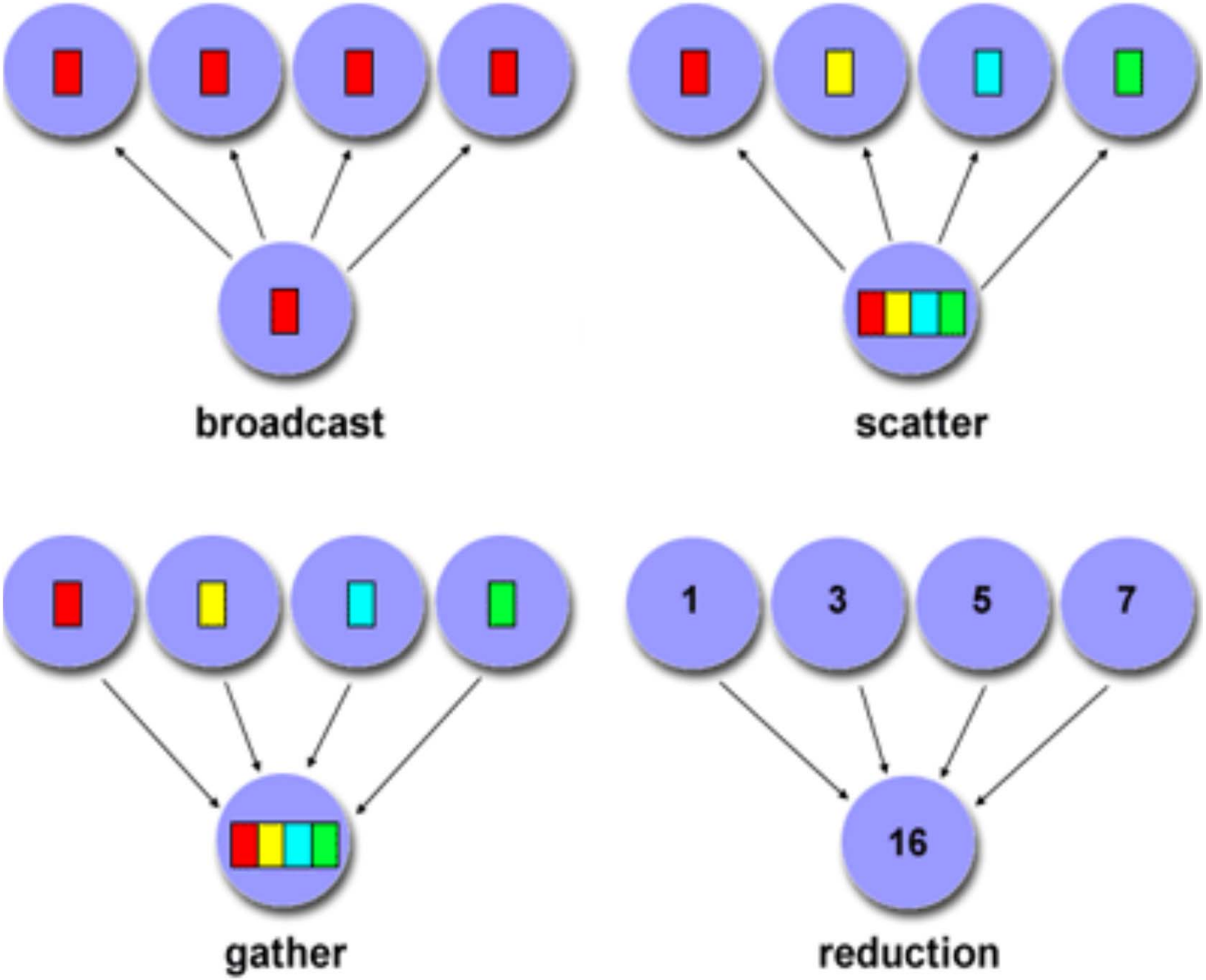
- MPI_Send (...) and MPI_Recv(...): For sending and receiving messages.

- **MPI_Bcast** (...) : broadcasts data from one node to all others. (Similar, **MPI_Scatter**)

- **MPI_Reduce** (...): Reverse of broadcast, all processors send to a single processor (similar, **MPI_Gather**).

- MPI_Barrier (MPI_COMM_WORLD, ierror): force synchronization.

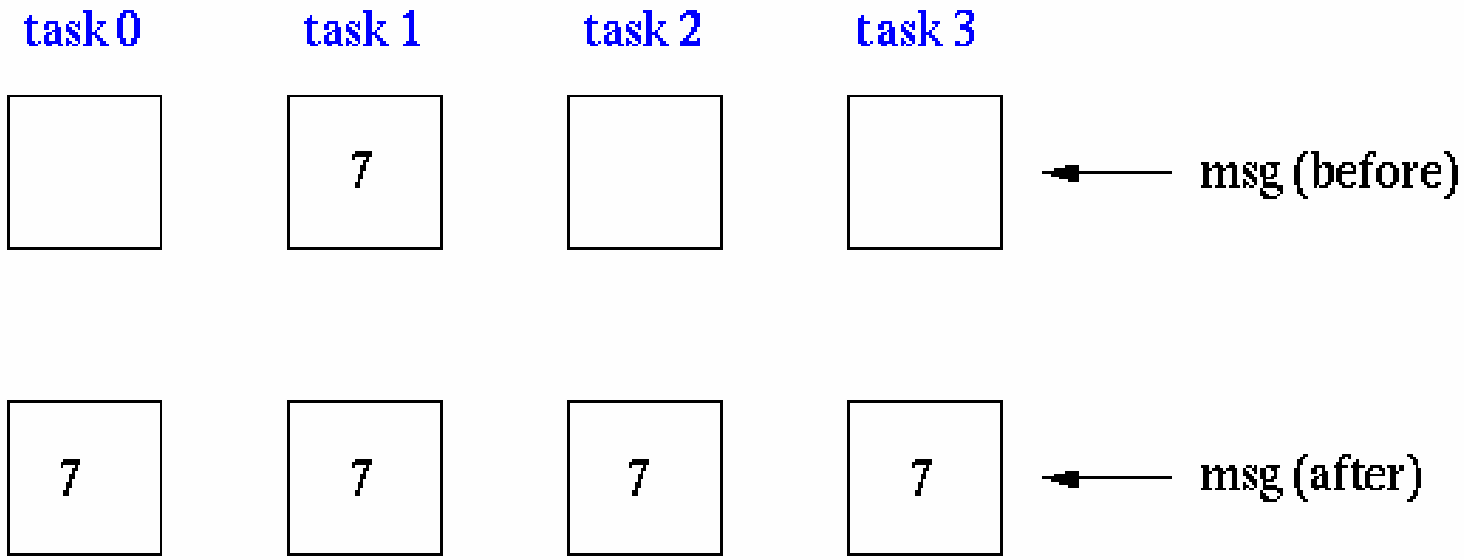
Bcast, Scatter, Gather, Reduction



MPI_Bcast

Broadcasts a message to all other processes of that group

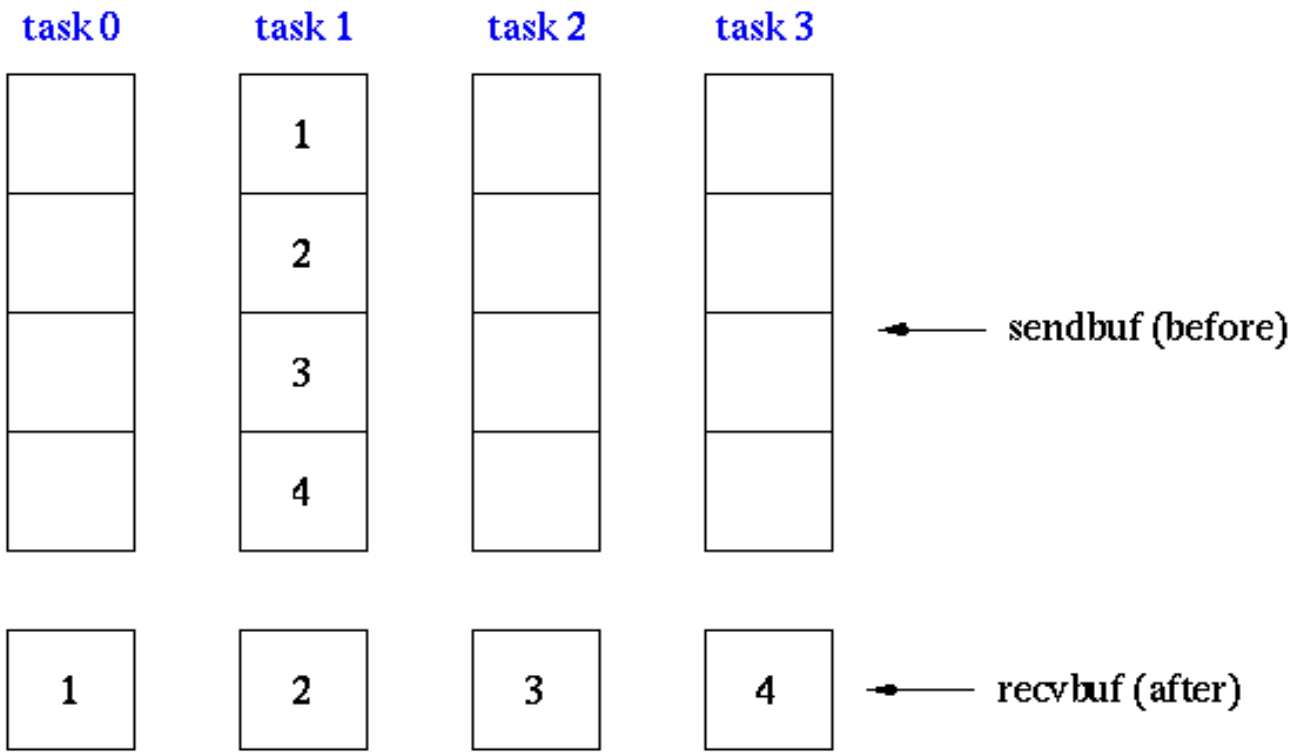
```
count = 1;
source = 1;                    broadcast originates in task 1
MPI_Bcast(&msg, count, MPI_INT, source, MPI_COMM_WORLD);
```



MPI_Scatter

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

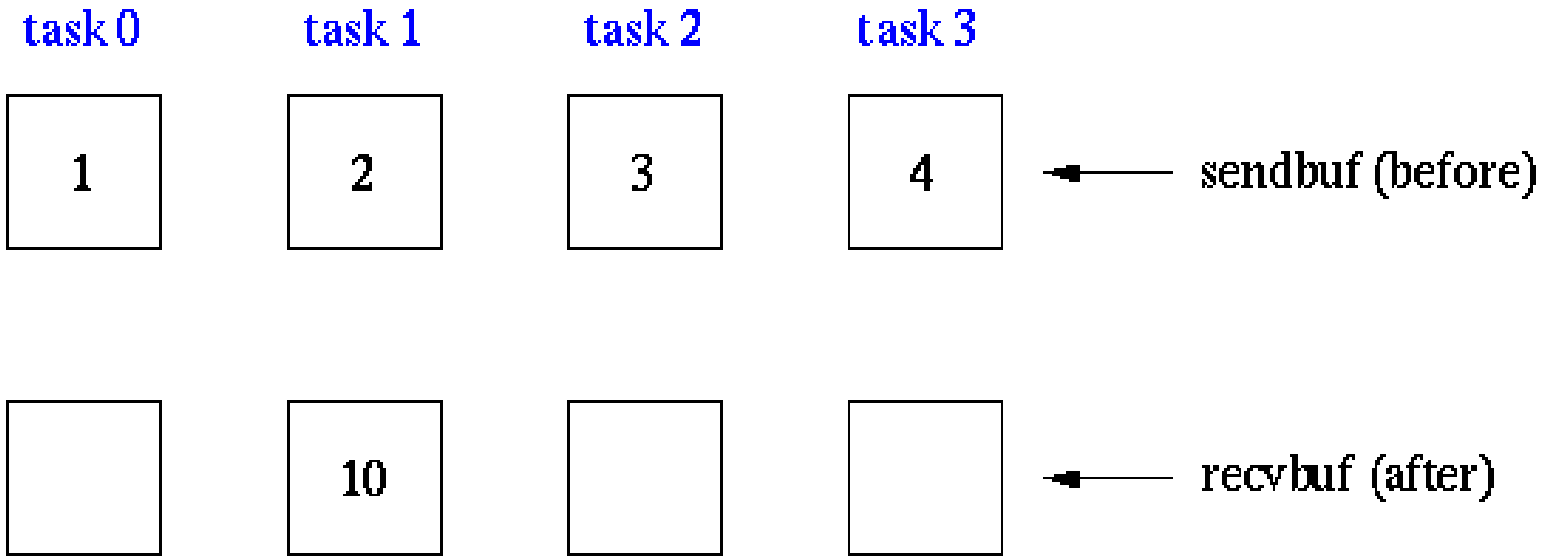
```
sendcnt = 1;
recvcnt = 1;
src = 1;                    task 1 contains the message to be scattered
MPI_Scatter(sendbuf, sendcnt, MPI_INT,
            recvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```



MPI_Reduce

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

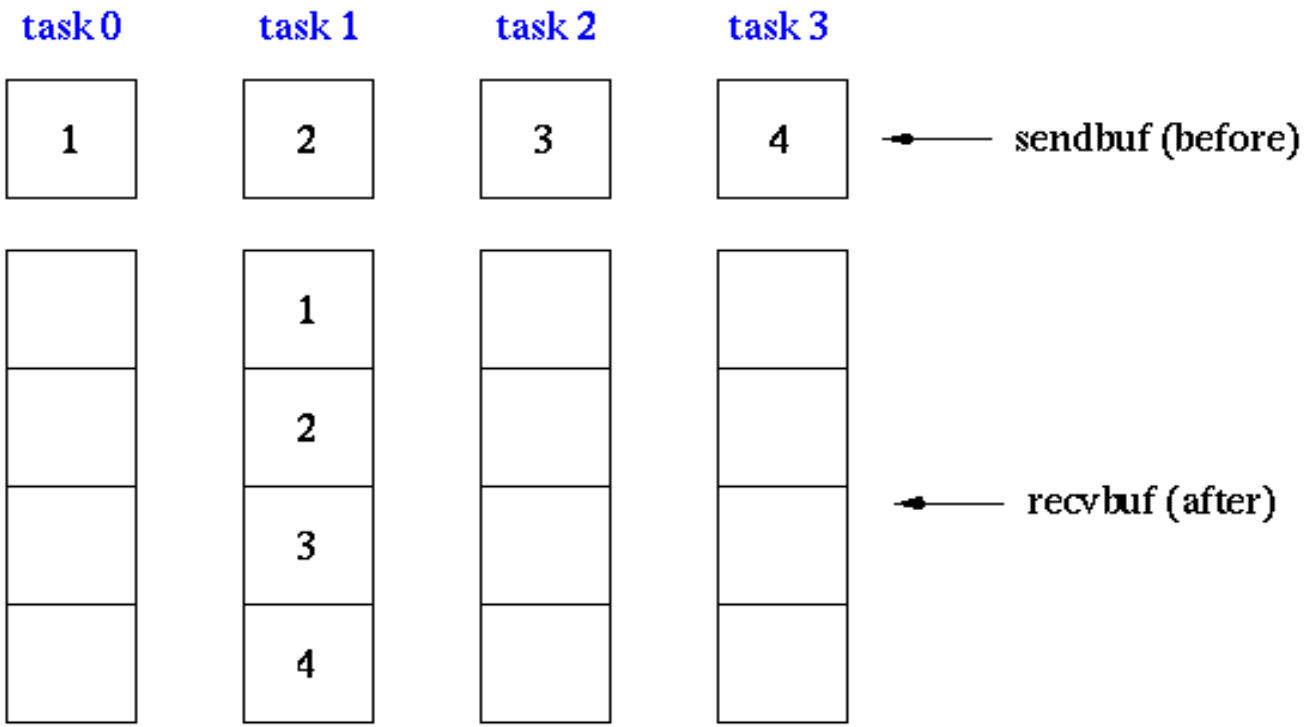
```
count = 1;
dest  = 1;                          result will be placed in task 1
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
           dest, MPI_COMM_WORLD);
```



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);
```



MPI Data Type

Be familiar with MPI data type. Example:
MPI_CHAR
MPI_BYTE
MPI_SHORT
MPI_INT
….. <more>

Compiling …

mpicc -o hello_world hello-world.c
Or, mpicc -o hello_world hello-world.c –lm // to link math library

–mpicc //c
–mpif90 //fotran 90
–mpiCC // C++
–mpif77 //fortran 77

Running…

mpirun -np 8 hello_world

Internet Resources

There are many sites, few are mentioned here:

For MPI:
https://computing.llnl.gov/tutorials/mpi/

http://www.mcs.anl.gov/research/projects/mpi/tutorial/gropp/talk.html
And http://www.mcs.anl.gov/research/projects/mpi/tutorial/

For Pthreads or, POSIX Thread:
https://computing.llnl.gov/tutorials/pthreads/
http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html
+ example(s) in the text book.

Example: mpi_hello.c

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


int main (int argc, char* argv[])
{
  int rank, size, i, j;
  MPI_Init (&argc, &argv);                /* Initialization step to start MPI */
  MPI_Comm_rank (MPI_COMM_WORLD, &rank); /* get the id of the current process */
  MPI_Comm_size (MPI_COMM_WORLD, &size); /* get the number of processes involved */

  srand((int)time(NULL));                 /* calling random number(RN) */
  int r = rand() % 20000;

  double y = r* sin((size-rank+1)*r);   /* using sin to vary RN arbitrarily */
   r = abs(y);

   for (i=0; i<=r; i++)                /* To make random delay */
    {
        j=i*i*i;
     }

   printf( "Hello world from process %d of %d\n", rank, size);

   MPI_Finalize();                     /* Terminates MPI execution environment */

  return 0;
}
```