MPI Tutorial (part 2)

Patrizio Dazzi
ISTI - CNR
Pisa Research Campus
mail: patrizio.dazzi@isti.cnr.it

Master Degree (Laurea Magistrale) in
Computer Science and Networking
Academic Year 2009-2010

Collective Communication Routines(1)

- **All or None**
  - *Collective communication must involve all processes in the scope of a communicator.*
  - *All processes are members of MPI_COMM_WORLD.*
  - *Programmer's responsibility to insure that all processes within a communicator participate to collective operations.*
- **Types of Collective Operations**
  - *Synchronization - processes wait until all members of the group reach the synchronization point.*
  - *Data Movement - broadcast, scatter/gather, all to all.*
  - *Collective Computation - one member of the group collects data from the other members and performs an operation (min, max, add, multiply, etc.) on data.*

Collective Communication Routines(2)

- **Programming Considerations and Restrictions**
  - *Collective operations are blocking.*
  - *Collective communication routines do not take message tag arguments.*
  - *Collective operations within subsets of processes are accomplished:*
    - by first partitioning the subsets into new groups and
    - then attaching the new groups to new communicators
  - *Can only be used with MPI predefined datatypes - not with MPI Derived Data Types.*

Collective Communication Routines(3)

- **MPI_Barrier (MPI_Barrier (comm))**
  - *Creates a barrier synchronization in a group. Each task, when reaching the MPI_Barrier call, blocks until all tasks in the group reach the same MPI_Barrier call.*

- **MPI_Bcast (MPI_Bcast (&buffer, count, datatype, root, comm))**
  - *Broadcasts (sends) a message from the process with rank "root" to all other processes in the group.*

- **MPI_Scatter (MPI_Scatter (&sendbuf, sendcnt, sendtype, &recvbuf, recvcnt, recvtype, root, comm))**
  - *Distributes distinct messages from a single source task to each task in the group.*

Collective Communication Routines(4)

MPI_Bcast (MPI_Bcast (&buffer, count, datatype, root, comm))

before

after

Collective Communication Routines(5)

MPI_Scatter (MPI_Scatter (&sendbuf, sendcnt, sendtype, &recvbuf, recvcnt, recvtype, root, comm))

before

after

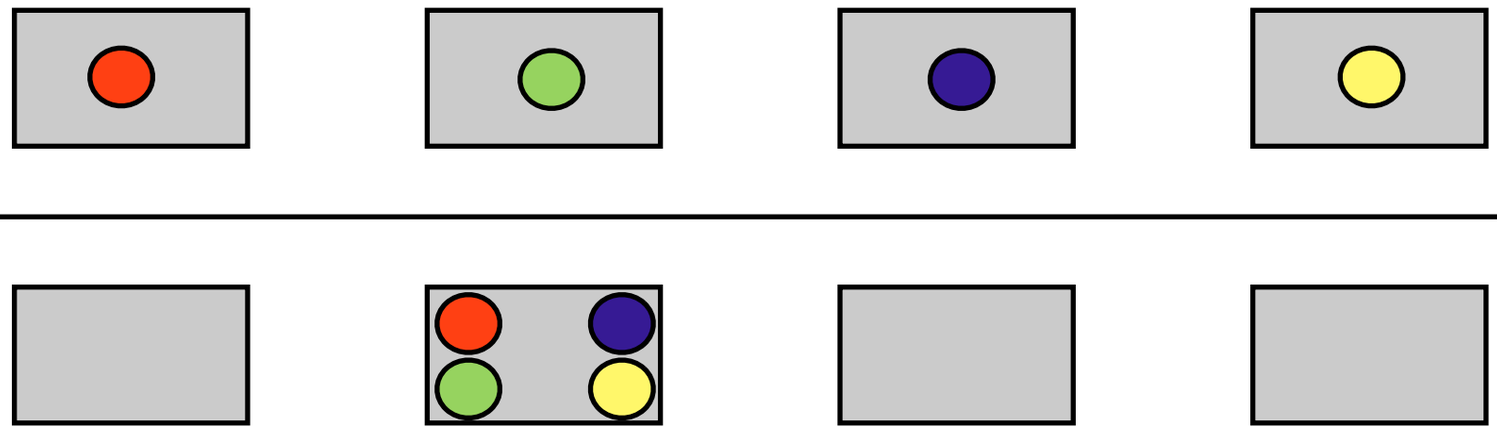
Collective Communication Routines(6)

- **MPI_Gather (MPI_Gather (&sendbuf, sendcnt, sendtype, &recvbuf, recvcount, recvtype, root, comm))**
  - *Gathers distinct messages from each task in the group to a single destination task.*
  - *This routine is the reverse operation of MPI_Scatter.*

- **MPI_Allgather (MPI_Allgather (&sendbuf, sendcount, sendtype, &recvbuf, recvcount, recvtype, comm))**
  - *Concatenation of data to all tasks in a group.*
  - *Each task in the group, in effect, performs a one-to-all broadcasting operation within the group.*

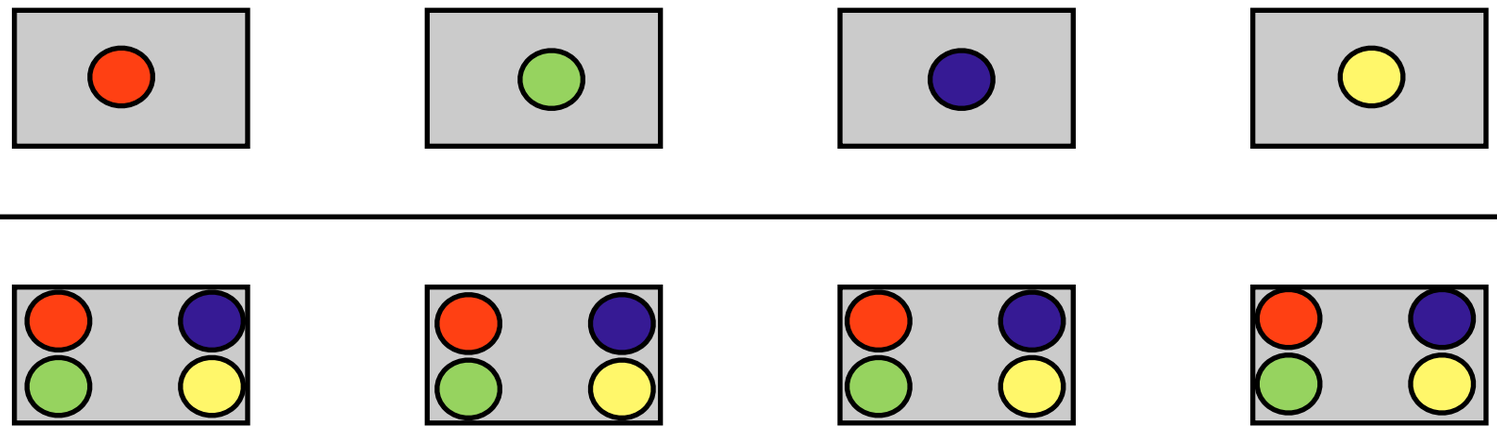
Collective Communication Routines(7)

MPI_Gather (MPI_Gather (&sendbuf, sendcnt, sendtype,
&recvbuf, recvcount, recvtype, root, comm))



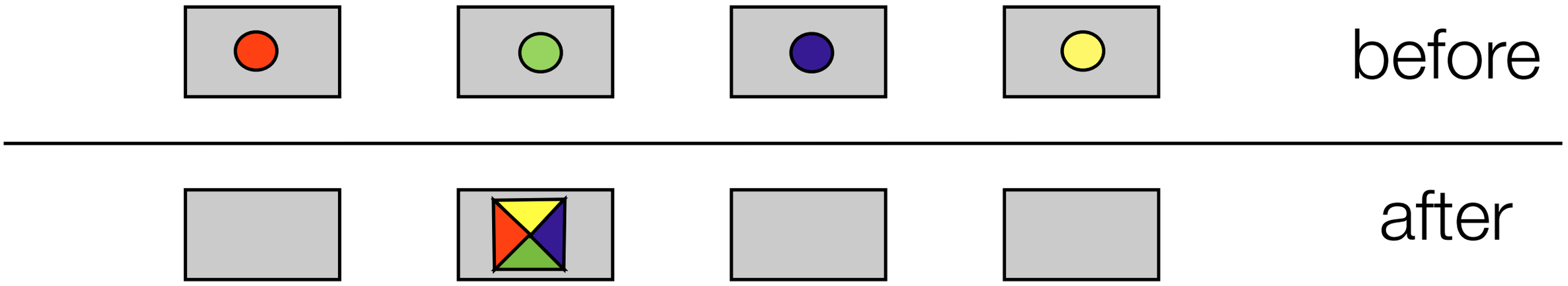
Collective Communication Routines(8)

MPI_Allgather (MPI_Allgather (&sendbuf, sendcount, sendtype, &recvbuf, recvcount, recvtype, comm))



Collective Communication Routines(9)

- **MPI_Reduce (MPI_Reduce (&sendbuf, &recvbuf, count, datatype, op, root, comm))**
  - *Applies a reduction operation on all tasks in the group and places the result in one task.*



Collective Communication Routines (10)

The predefined MPI reduction operations appear below.
Users can also define their own reduction functions by using the MPI_Op_create routine.

MPI Reduction Operation		C Data Types
MPI_MAX	maximum	integer, float
MPI_MIN	minimum	integer, float
MPI_SUM	sum	integer, float
MPI_PROD	product	integer, float
MPI_LAND	logical AND	integer
MPI_BAND	bit-wise AND	integer, MPI_BYTE
MPI_LOR	logical OR	integer
MPI_BOR	bit-wise OR	integer, MPI_BYTE
MPI_LXOR	logical XOR	integer
MPI_BXOR	bit-wise XOR	integer, MPI_BYTE
MPI_MAXLOC	max value and location	float, double and long double
MPI_MINLOC	min value and location	float, double and long double

Collective Communication Routines - Example 1


```
#include "mpi.h"
#include <stdio.h>
#define SIZE 4

int main(argc,argv){
    int numtasks, rank, sendcount, recvcount, source;
    float sendbuf[SIZE][SIZE] = {  {1.0, 2.0, 3.0, 4.0}, {5.0, 6.0, 7.0, 8.0},
                                   {9.0, 1.0, 1.0, 2.0}, {3.0, 4.0, 5.0, 6.0}  };
    float recvbuf[SIZE];

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

    if (numtasks == SIZE) {
      source = 1;
      sendcount = SIZE;
      recvcount = SIZE;
      MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
                  MPI_FLOAT,source,MPI_COMM_WORLD);

      printf("%d Results: %f %f %f %f\n", rank, recvbuf[0], recvbuf[1], recvbuf[2], recvbuf[3]);

    }else
      printf("Must specify %d processors. Terminating.\n",SIZE);

    MPI_Finalize();
}
```

Collective Communication Routines - Example 2


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

int main(int argc,char** argv){
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264 3;
    double mypi, pi, h, sum, x;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   n = 2000000000;

   if (myid == 0) {
       printf("The number of intervals is: %d \n",n);
          scand
          }

       MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

       h   = 1.0 / (double) n;
       sum = 0.0;
       for (i = myid + 1; i <= n; i += numprocs) {
           x = h * ((double)i - 0.5);
           sum += 4.0 / (1.0 + x*x);
       }
       mypi = h * sum;

       MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

       if (myid == 0) printf("pi is approximately %.16f, Error is %.16f\n", pi, fabs(pi - PI25DT));

       MPI_Finalize();
        return 0;
    }
```

Questions ?