

MPI – Communicators and Topologies

Based on notes by
Science & Technology Support
High Performance Computing

Ohio Supercomputer Center

Communicators

- A communicator is a parameter in all MPI message passing routines
- A communicator is a collection of processors that can engage in communication
- A communicator consists of a *group* of processes and a *context*
- MPI_COMM_WORLD is the default communicator that consists of all processors
- MPI allows you to create subsets of communicators

Why Communicators?

- Isolate communication to a small number of processors
- Useful for creating libraries
- Different processors can work on different parts of the problem
- Useful for communicating with "nearest neighbors"

MPI_Comm_split

- Provides a short cut method to create a collection of communicators
- All processors with the "same color" will be in the same communicator
- Key is used in determining the rank in new communicator (It is ordered by key/index pair)
- Fortran
 - call `MPI_COMM_SPLIT(OLD_COMM, color, key, NEW_COMM, mpi_err)`
- C
 - `MPI_Comm_split(OLD_COMM, color, key, &NEW_COMM)`

MPI_Comm_split

- Split odd and even processors into 2 communicators

```
Program comm_split
include "mpif.h"
Integer color,zero_one
call MPI_INIT( mpi_err )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numnodes, mpi_err )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, mpi_err )
color=mod(myid,2) !color is either 1 or 0
call MPI_COMM_SPLIT(MPI_COMM_WORLD,color,myid,NEW_COMM,mpi_err)
call MPI_COMM_RANK( NEW_COMM, new_id, mpi_err )
call MPI_COMM_SIZE( NEW_COMM, new_nodes, mpi_err )
Zero_one = -1
If(new_id==0)Zero_one = color
Call MPI_Bcast(Zero_one,1,MPI_INTEGER,0, NEW_COMM,mpi_err)
If(zero_one==0)write(*,*)"part of even processor communicator"
If(zero_one==1)write(*,*)"part of odd processor communicator"
Write(*,*)"old_id=", myid, "new_id=", new_id
Call MPI_FINALIZE(mpi_err)
End program
```

DiSCoV



22 March 2007

Paul A. Farrell
Cluster Computing 5

```
#include "mpi.h"
#include <math.h>
int main(argc,argv)
int argc;
char *argv[];
{
  int myid, numprocs;
  int color,Zero_one,new_id,new_nodes;
  MPI_Comm NEW_COMM;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  color=myid % 2;
  MPI_Comm_split(MPI_COMM_WORLD,color,myid,&NEW_COMM);
  MPI_Comm_rank( NEW_COMM, &new_id);
  MPI_Comm_size( NEW_COMM, &new_nodes);
  Zero_one = -1;
  if(new_id==0)Zero_one = color;
  MPI_Bcast(&Zero_one,1,MPI_INT,0, NEW_COMM);
  if(Zero_one==0)printf("part of even processor communicator \n");
  if(Zero_one==1)printf("part of odd processor communicator \n");
  printf("old_id= %d new_id= %d\n", myid, new_id);
  MPI_Finalize();
}
```

DiSCoV



22 March 2007

Paul A. Farrell
Cluster Computing 6

MPI_Comm_split

- Split odd and even processors into 2 communicators

0: part of even processor communicator

0: old_id= 0 new_id= 0

2: part of even processor communicator

2: old_id= 2 new_id= 1

1: part of odd processor communicator

1: old_id= 1 new_id= 0

3: part of odd processor communicator

3: old_id= 3 new_id= 1

Other Communicator Functions

- MPI_Comm_group gives the group of a communicator
 - MPI_COMM_GROUP(comm, group)
 - [IN comm] communicator (handle)
 - [OUT group] group corresponding to comm (handle)
- MPI_Group_incl defines a new group consisting of processes of specified rank from old group
 - MPI_GROUP_INCL(group, n, ranks, newgroup)
 - [IN group] group (handle)
 - [IN n] number of elements in array ranks (and size of newgroup) (integer)
 - [IN ranks] ranks of processes in group to appear in newgroup (array of integers)
 - [OUT newgroup] new group derived from above, in the order defined by ranks (handle)
- MPI_Comm_create creates a sub-communicator specified by a given group from a communicator
 - MPI_COMM_CREATE(comm, group, newcomm)
 - [IN comm] communicator (handle)
 - [IN group] Group, which is a subset of the group of comm (handle)
 - [OUT newcomm] new communicator (handle)

Examples of Communicator Functions

- `Comm_split`
- [Other communicator operations](#)

Virtual Topologies

- [Virtual Topologies](#)
- [Topology Types](#)
- [Creating a Cartesian Virtual Topology](#)
- [Cartesian Example](#)
- [Cartesian Mapping Functions](#)
 - `MPI_CART_RANK*`
 - `MPI_CART_COORDS*`
 - `MPI_CART_SHIFT*`
- [Cartesian Partitioning](#)

*includes sample C and Fortran programs

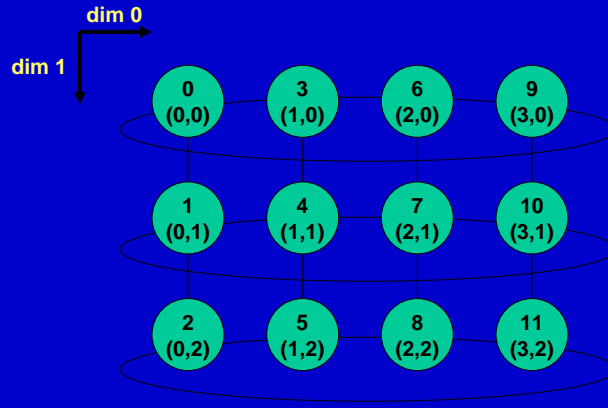
Virtual Topologies

- Convenient process naming
- Naming scheme to fit the communication pattern
- Simplifies writing of code
- Can allow MPI to optimize communications
- Rationale: access to useful topology routines

How to use a Virtual Topology

- Creating a topology produces a new communicator
- MPI provides “mapping functions”
- Mapping functions compute processor ranks, based on the topology naming scheme

Example - 2D Torus



Topology types

- Cartesian topologies
 - Each process is connected to its neighbors in a virtual grid
 - Boundaries can be cyclic
 - Processes can be identified by Cartesian coordinates
- Graph topologies
 - General graphs
 - Will not be covered here

Creating a Cartesian Virtual Topology

C:

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

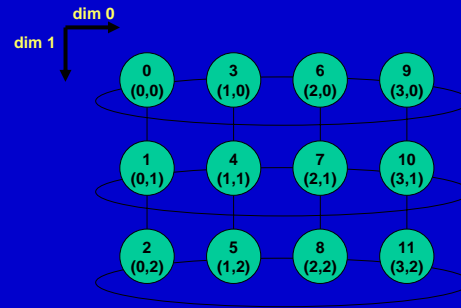
Fortran:

```
INTEGER    COMM_OLD, NDIMS, DIMS (*), COMM_CART, IERROR  
LOGICAL    PERIODS (*), REORDER  
  
CALL MPI_CART_CREATE (COMM_OLD, NDIMS, DIMS, PERIODS, REORDER,  
                      COMM_CART, IERROR)
```

Arguments

<code>comm_old</code>	existing communicator
<code>ndims</code>	number of dimensions
<code>periods</code>	logical array indicating whether a dimension is cyclic (If TRUE, cyclic boundary conditions)
<code>reorder</code>	logical (If FALSE, rank preserved) (If TRUE, possible rank reordering)
<code>comm_cart</code>	new cartesian communicator

Cartesian Example



```
MPI_Comm vu;
int dim[2], period[2], reorder;

dim[0]=4; dim[1]=3;
period[0]=TRUE; period[1]=FALSE;
reorder=TRUE;

MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &vu);
```

Cartesian Mapping Functions

Mapping process grid coordinates to ranks

C:

```
int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)
```

Fortran:

```
INTEGER COMM, COORDS (*), RANK, IERROR
```

```
CALL MPI_CART_RANK (COMM, COORDS, RANK, IERROR)
```

Cartesian Mapping Functions

Mapping ranks to process grid coordinates

C:

```
int MPI_Cart_coords (MPI_Comm comm, int rank, int
                    maxdims, int *coords)
```

Fortran:

```
INTEGER    COMM,RANK,MAXDIMS,COORDS(*),IERROR
```

```
CALL MPI_CART_COORDS (COMM,RANK,MAXDIMS,COORDS,IERROR)
```

DISCoV



22 March 2007

Paul A. Farrell
Cluster Computing 19

Sample Program #9 - C

```
#include<mpi.h>
/* Run with 12 processes */
void main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2],period[2],reorder;
    int coord[2],id;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD,2,dim,period,reorder,&vu);
    if(rank==5){
        MPI_Cart_coords(vu,rank,2,coord);
        printf("P:%d My coordinates are %d %d\n",rank,coord[0],coord[1]);
    }
    if(rank==0) {
        coord[0]=3; coord[1]=1;
        MPI_Cart_rank(vu,coord,&id);
        printf("The processor at position (%d, %d) has rank
%d\n",coord[0],coord[1],id);
    }
    MPI_Finalize();
}
```

```
The processor at position (3,1) has rank 10
P:5 My coordinates are 1 2
```

DISCoV



22 March 2007

Paul A. Farrell
Cluster Computing 20

Sample Program #9 - Fortran

```
PROGRAM Cartesian
C Run with 12 processes
INCLUDE 'mpif.h'
INTEGER err, rank, size
integer vu,dim(2),coord(2),id
logical period(2),reorder
CALL MPI_INIT(err)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
dim(1)=4
dim(2)=3
period(1)=.true.
period(2)=.false.
reorder=.true.
call MPI_CART_CREATE(MPI_COMM_WORLD,2,dim,period,reorder,vu,err)
if(rank.eq.5) then
  call MPI_CART_COORDS(vu,rank,2,coord,err)
  print*,'P:',rank,' my coordinates are',coord
end if
if(rank.eq.0) then
  coord(1)=3
  coord(2)=1
  call MPI_CART_RANK(vu,coord,id,err)
  print*,'P:',rank,' processor at position',coord,' is',id
end if
```

```
P:5 my coordinates are 1, 2
P:0 processor at position 3, 1 is 10
```

DISCoV



22 March 2007

Paul A. Farrell
Cluster Computing 21

Cartesian Mapping Functions

Computing ranks of neighboring processes

C:

```
int MPI_Cart_shift (MPI_Comm comm, int direction,
  int disp, int *rank_source, int *rank_dest)
```

Fortran:

```
INTEGER
  COMM,DIRECTION,DISP,RANK_SOURCE,RANK_DEST,IERROR

CALL MPI_CART_SHIFT(COMM,DIRECTION,DISP,RANK_SOURCE,
  RANK_DEST,IERROR)
```

DISCoV



22 March 2007

Paul A. Farrell
Cluster Computing 22

MPI_Cart_shift

- Does not actually shift data: returns the correct ranks for a shift which can be used in subsequent communication calls
- Arguments:
 - direction** dimension in which the shift should be made
 - disp** length of the shift in processor coordinates (+ or -)
 - rank_source** where calling process should receive a message **from** during the shift
 - rank_dest** where calling process should send a message **to** during the shift
- If shift off of the topology, MPI_PROC_NULL is returned

Sample Program #10 - C

```
#include<mpi.h>
#define TRUE 1
#define FALSE 0
void main(int argc, char *argv[]) {
    int rank;
    MPI_Comm vu;
    int dim[2],period[2],reorder;
    int up,down,right,left;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    dim[0]=4; dim[1]=3;
    period[0]=TRUE; period[1]=FALSE;
    reorder=TRUE;
    MPI_Cart_create(MPI_COMM_WORLD,2,dim,period,reorder,&vu);
    if(rank==9){
        MPI_Cart_shift(vu,0,1,&left,&right);
        MPI_Cart_shift(vu,1,1,&up,&down);
        printf("P:%d My neighbors are r: %d d:%d l:%d
u:%d\n",rank,right,down,left,up);
    }
    MPI_Finalize();
}
```

P:9 my neighbors are r:0 d:10 l:6 u:-1

Sample Program #10- Fortran

```
PROGRAM neighbors
C
C Run with 12 processes
C
  INCLUDE 'mpif.h'
  INTEGER err, rank, size
  integer vu
  integer dim(2)
  logical period(2),reorder
  integer up,down,right,left
  CALL MPI_INIT(err)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD,rank,err)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD,size,err)
  dim(1)=4
  dim(2)=3
  period(1)=.true.
  period(2)=.false.
  reorder=.true.
  call MPI_CART_CREATE(MPI_COMM_WORLD,2,dim,period,reorder,vu,err)
  if(rank.eq.9) then
    call MPI_CART_SHIFT(vu,0,1,left,right,err)
    call MPI_CART_SHIFT(vu,1,1,up,down,err)
    print*,'P:',rank,' neighbors (rdlu)are',right,down,left,up
  end if
  CALL MPI_FINALIZE(err)
END
```

Cartesian Partitioning

- Often we want to do an operation on only part of an existing Cartesian topology
- Cut a grid up into 'slices'
- A new communicator is produced for each slice
- Each slice can then perform its own collective communications
- MPI_Cart_sub and MPI_CART_SUB generate new communicators for the slice

MPI_Cart_sub

C:

```
int MPI_Cart_sub (MPI_Comm comm, int *remain_dims,  
                 MPI_Comm *newcomm)
```

Fortran:

```
INTEGER    COMM,NEWCOMM,IERROR  
LOGICAL    REMAIN_DIMS(*)
```

```
CALL MPI_CART_SUB(COMM,REMAIN_DIMS,NEWCOMM,IERROR)
```

- If `comm` is a 2x3x4 grid and `remain_dims={TRUE,FALSE,TRUE}`, then three new communicators are created each being a 2x4 grid
- Calling processor receives back only the new communicator it is in

Topology Example

- [Topological functions](#)
- [Fox's algorithm](#)

Problem Set

- 1) Write a program that will make a virtual topology using 8 processors. The topology should consist of 4 processor rows and 2 processors columns with no wrap-around (cyclic) in either dimension. Each processor in the topology should have an integer variable with the same values used in Problem 4 (Day 3).

After your program has created the topology, it should use virtual topology utility functions to have each processor calculate the average value of its integer and the integers contained in it's neighbors. Each processor should then output its calculated average. (NOTE: do not use "diagonal" neighbors in the averaging. Only use "up", "down", "left", and "right" neighbors, if they exist).