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
- Exercise

* 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

Going “up” from here gives MPI_PROC_NULL (-1) and hence batch output is returned as error output.
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comm_old</td>
<td>existing communicator</td>
</tr>
<tr>
<td>ndims</td>
<td>number of dimensions</td>
</tr>
<tr>
<td>periods</td>
<td>logical array indicating whether a dimension is cyclic</td>
</tr>
<tr>
<td></td>
<td>(TRUE=&gt;cyclic boundary conditions)</td>
</tr>
<tr>
<td>reorder</td>
<td>logical</td>
</tr>
<tr>
<td></td>
<td>(FALSE=&gt;rank preserved)</td>
</tr>
<tr>
<td></td>
<td>(TRUE=&gt;possible rank reordering)</td>
</tr>
<tr>
<td>comm_cart</td>
<td>new cartesian communicator</td>
</tr>
</tbody>
</table>
Cartesian Example

```c
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:

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

Fortran:

```fortran
INTEGER COMM,COORDS(*),RANK,IERROR
CALL MPI_CART_RANK(COMM,COORDS,RANK,IERROR)
```
Cartesian Mapping Functions

Mapping ranks to process grid coordinates

C:

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

Fortran:

```fortran
INTEGER COMM, RANK, MAXDIMS, COORDS(*), IERROR
CALL MPI_CART_COORDS(COMM, RANK, MAXDIMS, COORDS, IERROR)
```
Sample Program #9 - C

```c
#include<mpi.h>
/* Run with 12 processes */
int 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();
}
```

Program output
- The processor at position (3,1) has rank 10
- P:5 My coordinates are 1 2
Sample Program #9 - Fortran

PROGRAM Cartesian
C
C Run with 12 processes
C
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
CALL MPI_FINALIZE(err)
END

Program Output
P:5 my coordinates are 1, 2
P:0 processor at position 3, 1 is 10
Cartesian mapping functions

Computing ranks of neighboring processes

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

Fortran:
```fortran
INTEGER COMM, DIRECTION, DISP
INTEGER RANK_SOURCE, RANK_DEST, IERROR
CALL MPI_CART_SHIFT(COMM, DIRECTION, DISP,
                     RANK_SOURCE, RANK_DEST, IERROR)
```
MPI_Cart_shift

• Does not actually shift data: returns the correct ranks for a shift that 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 we shift off of the topology, MPI_Proc_null (-1) is returned
Sample Program #10 - C

#include<mpi.h>
define TRUE 1
define FALSE 0
int 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 1:%d u:%d\n", rank, right, down, left, up);
    }
    MPI_Finalize();
}
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:

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

Fortran:

```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 its new communicator only
Class Exercise: Ring Topology

• Rewrite the “Calculating Ring” exercise using a Cartesian Topology

• Extra credit: Extend the problem to two dimensions. Each row of the grid should compute its own separate sum