23rd Summer School on PARALLEL COMPUTING

MPI topologies

Maurizio Cremonesi, m.cremonesi@cineca.it

SuperComputing Applications and Innovation Department

CINECA

Virtual topologies

- Virtual topologies
- MPI supported topologies
- How to create a cartesian topology
- Cartesian mapping functions
- Cartesian partitioning

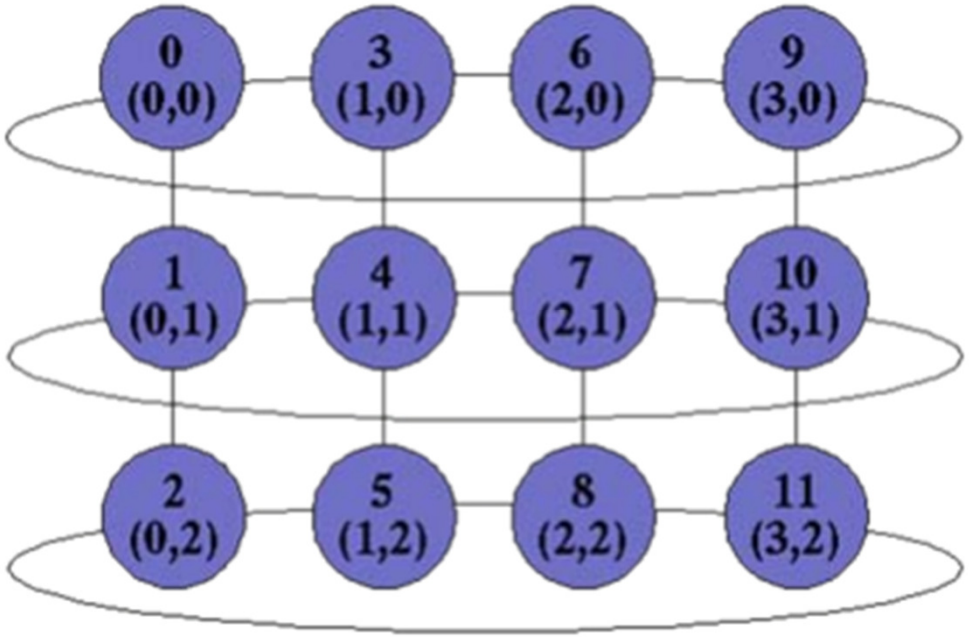
Why a virtual topology can be useful?

- Convenient process naming
- Naming scheme to fit the communication pattern
- Simplifies the writing of the code
- Can allow MPI to optimize communications

How to use a virtual topology?

- A new topology = a new communicator
- MPI provides some “mapping functions” to manage virtual topologies
- Mapping functions compute processor ranks, based on the topology name scheme

Cartesian topology on a 2D torus



MPI supports...

- Cartesian topologies
  - each process is connected to its neighbours in a virtual grid
  - boundaries can be cyclic
  - processes can be identified by cartesian coords
- Graph topologies

MPI_Cart_Create

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

```
interface
    subroutine mpi_cart_create(comm_old, ndims, ldims, &
&       periods, reorder, comm_cart, ierr)
            integer, intent(in) :: comm_old, ndims
            integer, dimension(:), intent(in) :: ldims
            logical, dimension(:), intent(in) :: periods
            logical, intent(in) :: reorder
            integer, intent(out) :: comm_cart, ierr
    end subroutine mpi_cart_create
end interface
```

MPI_Cart_Create

```
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)
```

Useful functions

Grid coords → ranks

MPI_Cart_rank

ranks → Grid coords

MPI_Cart_coords

Moving upwards, downwords, leftside, rightside...

MPI_Cart _shift

MPI_Cart_rank

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

```
interface
    subroutine mpi_cart_rank(comm, coords, rank, ierr)
        integer, intent(in) :: comm
        integer, dimension(:), intent(in) :: coords
        integer, intent(out) :: rank, ierr
    end subroutine mpi_cart_rank
end interface
```

MPI_Cart_coords

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

```
interface
    subroutine mpi_cart_coords(comm, rank, maxdims, &
&       coords, ierr)
          integer, intent(in) :: comm, rank, maxdims
          integer, dimension(:), intent(out) :: coords
          integer, intent(out) :: ierr
    end subroutine mpi_cart_coords
end interface
```

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

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_Cart_shift

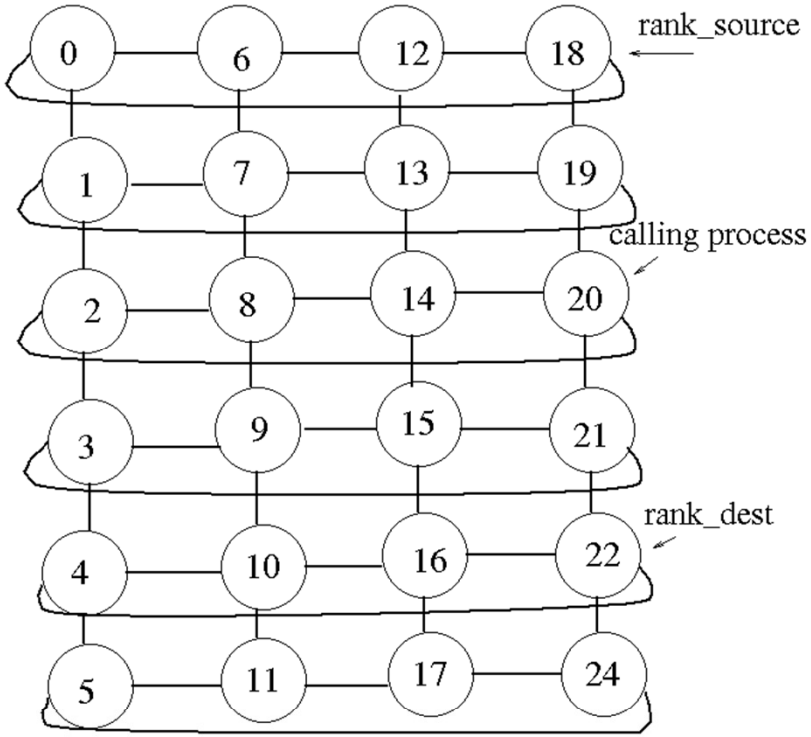
• It doesn't shift data actually: it returns the correct ranks for a shift that can be used in the subsequent communication call

• Arguments:
  - direction: in which direction the shift should be made
  - disp: length of the shift
  - rank_source: where the calling process should receive a message from during the shift
  - rank_dest: where the calling process should send a message to during the shift

MPI_Cart_shift

```
int MPI_Cart_shift(MPI_Comm comm,int dim,int delta,
       int *source,int *dest)
```

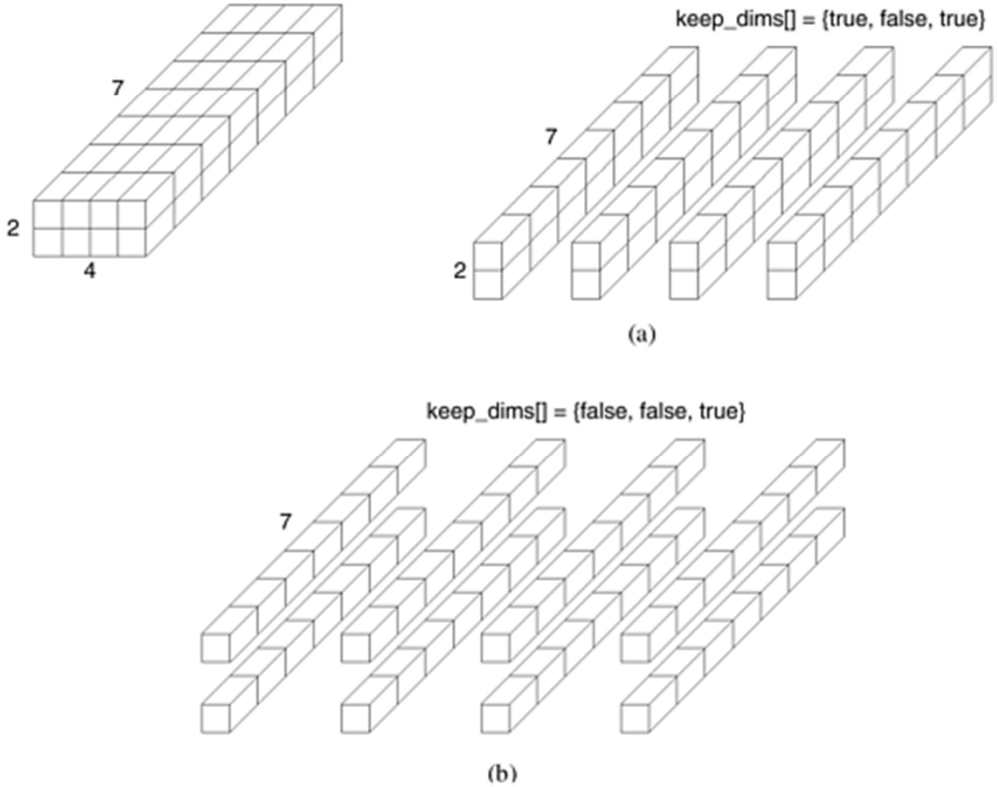
```
interface
    subroutine mpi_cart_shift(comm, dim, delta, source, &
&       dest, ierr)
          integer, intent(in) :: comm, dim, delta
          integer, intent(out) :: source, dest, ierr
    end subroutine mpi_cart_shift
end interface
```


0
6
12
18
rank_source
1
7
13
19
calling process
2
8
14
20
3
9
15
21
rank_dest
4
10
16
22
5
11
17
24

```
C find process rank

   CALL MPI_COMM_RANK(comm, rank, ierr))

C find cartesian coordinates

   CALL MPI_CART_COORDS(comm, rank, maxdims, coords, ierr)

C compute shift source and destination

   dir = 1; disp = 2

   CALL MPI_CART_SHIFT(comm, dir, disp, source, dest, ierr)

C skew array

   CALL MPI_SENDRECV_REPLACE(A, 1, MPI_REAL, dest, 0, &

&       source, 0, comm, status, ierr)
```

Cartesian partitioning

- Often we want to do an operation on only a part of an existing cartesian topology
- Cut a grid up into “slices”
- A new communicator (i.e. a new cart. topology) is produced for each slide
- Each slice can perform its own collective communications




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

MPI_Cart_sub

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

```
interface
    subroutine mpi_cart_sub(comm, remain_dims, newcomm, &
&       ierr)
          integer, intent(in) :: comm
          logical, dimension(:), intent(in) :: remain_dims
          integer, intent(out) :: newcomm, ierr
    end subroutine mpi_cart_sub
end interface
```

```
C create cartesian topology

   dim = [2, 4, 7];  period = [.F., .F., .F.]

   reorder = .T.

   CALL MPI_CART_CREATE(MPI_COMM_WORLD,3,dim,period,

        reorder,cart_comm);


C split topology: 4 new topologies are generated

   remain_dims = [.T.,.F.,.T.]

   CALL MPI_CART_SUB(cart_comm, remain_dims, newcomm, &

&       ierr))
```