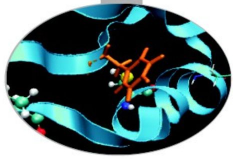
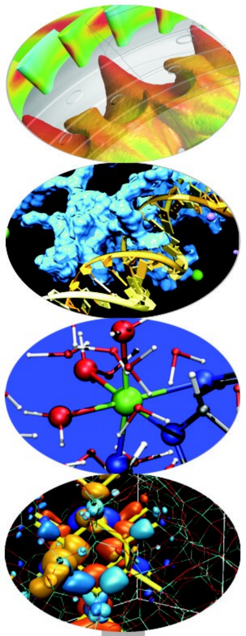
SuperComputing Applications and Innovation



INTRODUCTION TO MPI – COMMUNICATORS AND VIRTUAL TOPOLOGIES

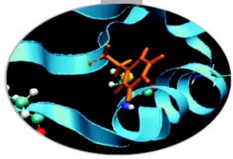


Introduction to Parallel Computing with MPI and OpenMP

24 november 2017

a.marani@cineca.it

WHAT ARE COMMUNICATORS?



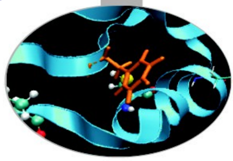
Many users are familiar with the mostly used communicator:
MPI_COMM_WORLD

A **communicator** can be thought as a handle to a **group**.

- a group is a ordered set of processes
- each process is associated with a rank
- ranks are contiguous and start from zero

Groups allow collective operations to be operated on a subset of processes

DEFINITIONS & PROPERTIES



Intracommunicators are used for communications within a single group
Intercommunicators are used for communications between two disjoint groups

Group management:

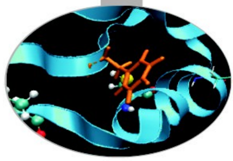
- All group operations are local
- Groups are not initially associated with communicators
- Groups can only be used for message passing within a communicator
- We can access groups, construct groups, destroy groups

Group accessors:

MPI_GROUP_SIZE
This routine returns the number of processes in the group

MPI_GROUP_RANK
This routine returns the rank of the calling process inside a given group

GROUP CONSTRUCTORS

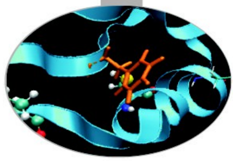


Group constructors are used to create new groups from existing ones (initially from the group associated with MPI_COMM_WORLD; you can use mpi_comm_group to get this).

Group creation is a local operation: no communication is needed

After the creation of a group, no communicator has been associated to this group, and hence no communication is possible within the new group

GROUP CONSTRUCTORS



- **MPI_COMM_GROUP(**comm,group,ierr)

  This routine returns the group associated with the communicator comm

- **MPI_GROUP_UNION**(group_a, group_b, newgroup, ierr)

  This returns the ensemble union of group_a and group_b

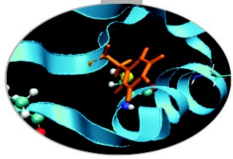
- **MPI_GROUP_INTERSECTION**(group_a, group_b, newgroup, ierr)

  This returns the ensemble intersection of group_a and group_b

- **MPI_GROUP_DIFFERENCE**(group_a, group_b, newgroup, ierr)

  This returns in newgroup all processes in group_a that rare not in group_b, ordered as in group_a

GROUP CONSTRUCTORS



- **MPI_GROUP_INCL**(group, n, ranks, newgroup, ierr)

  This routine creates a new group that consists of all the n processes with ranks ranks[0]... ranks[n-1]

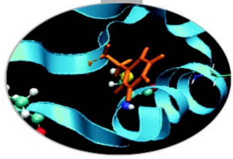
  Example:
  group = {a,b,c,d,e,f,g,h,i,j}
  n = 5
  ranks = {0,3,8,6,2}
  newgroup = {a,d,i,g,c}

- **MPI_GROUP_EXCL**(group,n,ranks,newgroup,ierr)

  This routine returns a newgroup that consists of all the processes in the group after removing processes with ranks: ranks[0]..ranks[n-1]

  Example:
  group = {a,b,c,d,e,f,g,h,i,j}
  n = 5
  ranks = {0,3,8,6,2}
  newgroup = {b,e,f,h,j}

COMMUNICATOR MANAGEMENT

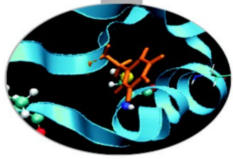


Communicator access operations are local, not requiring interprocess communication

Communicator constructors are collective and may require interprocess communications

We will cover in depth only intracommunicators, giving only some notions about intercommunicators.

COMMUNICATOR ACCESSORS

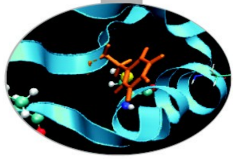


- **MPI_COMM_SIZE**(comm,size,ierr)
  Returns the number of processes in the group associated with the comm

- **MPI_COMM_RANK**(comm,rank,ierr)
  Returns the rank of the calling process within the group associated with the comm

- **MPI_COMM_COMPARE**(comm1,comm2,result,ierr)
  Returns:
  - MPI_IDENT if comm1 and comm2 are the same handle
  - MPI_CONGRUENT if comm1 and comm2 have the same group attribute
  - MPI_SIMILAR if the groups associated with comm1 and comm2 have the same members but in different rank order
  - MPI_UNEQUAL otherwise

COMMUNICATOR CONSTRUCTORS



- **MPI_COMM_DUP**(comm, newcomm,ierr)

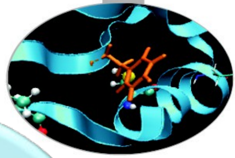
This returns a communicator newcomm identical to the communicator comm

- **MPI_COMM_CREATE**(comm, group, newcomm,ierr)

This collective routine must be called by all the process involved in the group associated with comm. It returns a new communicator that is associated with the group. MPI_COMM_NULL is returned to processes not in the group.

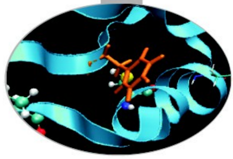
Note that the new group must be a subset of the group associated with comm!

EXAMPLE (C)



```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char **argv) {
    int rank, new_rank, nprocs, sendbuf, recvbuf, ranks1[4]={0,1,2,3},
ranks2[4]={4,5,6,7};
    MPI_Group orig_group, new_group;
    MPI_Comm new_comm;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    sendbuf = rank;
    MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
    if (rank < nprocs/2)
      MPI_Group_incl(orig_group, nprocs/2, ranks1, &new_group);
    else MPI_Group_incl(orig_group, nprocs/2, ranks2, &new_group);
    MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
    MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM, new_comm);
    MPI_Group_rank (new_group, &new_rank);
    printf("rank= %d newrank= %d recvbuf= %d\n",rank,new_rank,recvbuf);
    MPI_Finalize();
    return 0;
}
```

Hypothesis: nprocs=8 credits: http://static.msi.umn.edu



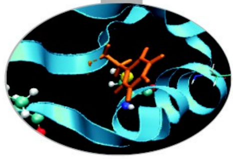
EXAMPLE (C)

RESULTS:

```
rank= 0 newrank= 0 recvbuf= 6

rank= 1 newrank= 1 recvbuf= 6

rank= 2 newrank= 2 recvbuf= 6

rank= 3 newrank= 3 recvbuf= 6

rank= 4 newrank= 0 recvbuf= 22

rank= 5 newrank= 1 recvbuf= 22

rank= 6 newrank= 2 recvbuf= 22

rank= 7 newrank= 3 recvbuf= 22
```

Hypothesis: nprocs=8 credits: http://static.msi.umn.edu

MPI COMM SPLIT



MPI_COMM_SPLIT(comm, color, key, newcomm, ierr)

This routine creates as many new groups and communicators as there are distinct values of color.

- *comm* is the old communicator
- *color* is an array of integers specifying on which group should a process belong to in the new communicator
- *key* is an array of integer that defines the rank that the process will get in the new communicator, that will be assigned in increasing order depending on the associated key value
- *newcomm* is the new communicator

The rankings in the new groups are determined by the value of the key.

MPI_UNDEFINED is used as a color when the process shouldn't be included in any of the new groups

MPI COMM SPLIT

Rank	0	1	2	3	4	5	6	7	8	9	10
Process	a	b	c	d	e	f	g	h	i	j	k
Color	U	3	1	1	3	7	3	3	1	U	3
Key	0	1	2	3	1	9	3	8	1	0	0

Both process a and j are returned MPI_COMM_NULL

3 new groups are created

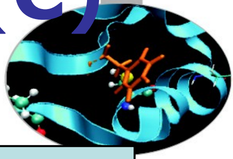
{i, c, d}

{k, b, e, g, h}

{f}

SuperComputing Applications and Innovation

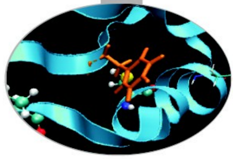
MPI COMM SPLIT – EXAMPLE (C)



```
if(myid%2==0){
        color=1;
}else{
        color=2;
}
MPI_COMM_SPLIT(MPI_COMM_WORLD,color,myid,&subcomm);
MPI_COMM_RANK(subcomm,mynewid);
printf(“rank in MPICOMM_WORLD %d”,myid,”rank in Subcomm %d”,
mynewid);
```

```
I am rank 2 in MPI_COMM_WORLD, but 1 in Comm 1.
I am rank 7 in MPI_COMM_WORLD, but 3 in Comm 2.
I am rank 0 in MPI_COMM_WORLD, but 0 in Comm 1.
I am rank 4 in MPI_COMM_WORLD, but 2 in Comm 1.
I am rank 6 in MPI_COMM_WORLD, but 3 in Comm 1.
I am rank 3 in MPI_COMM_WORLD, but 1 in Comm 2.
I am rank 5 in MPI_COMM_WORLD, but 2 in Comm 2.
I am rank 1 in MPI_COMM_WORLD, but 0 in Comm 2.
```

DESTRUCTORS

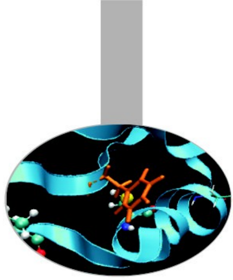


The communicators and groups from a process' viewpoint are just handles.
Like all handles, there is a limited number available: you could (in principle) run out!

MPI_GROUP_FREE(group, ierr)
MPI_COMM_FREE(comm,ierr)

Remember to free your handles after they are no longer needed, it is always a good practice (like with allocatable arrays)

INTERCOMMUNICATORS



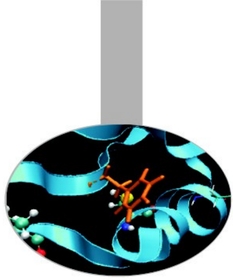
Intercommunicators are associated with 2 groups of disjoint processes.

Intercommunicators are associated with a remote group and a local group

The target process (destination for send, source for receive) is its rank in the remote group

A communicator is either intra or inter, never both

VIRTUAL TOPOLOGY



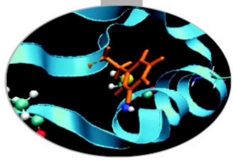
Topology:

- extra, optional attribute that can be given to an intra-communicator; topologies cannot be added to inter-communicators.
- can provide a convenient naming mechanism for the processes of a group (within a communicator), and additionally, may assist the runtime system in mapping the processes onto hardware.

A process group in MPI is a collection of n processes:

- each process in the group is assigned a rank between 0 and n-1.
- in many parallel applications a linear ranking of processes does not adequately reflect the logical communication pattern of the processes (which is usually determined by the underlying problem geometry and the numerical algorithm used).

VIRTUAL TOPOLOGY



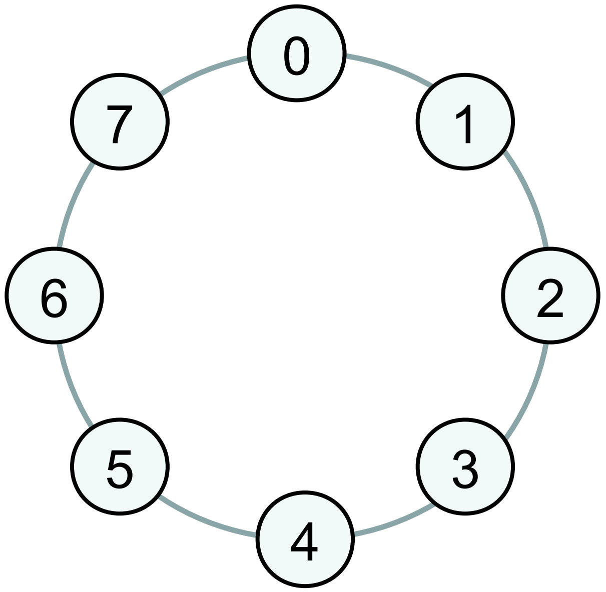
Virtual topology:

- logical process arrangement in topological patterns such as 2D or 3D grid; more generally, the logical process arrangement is described by a graph.

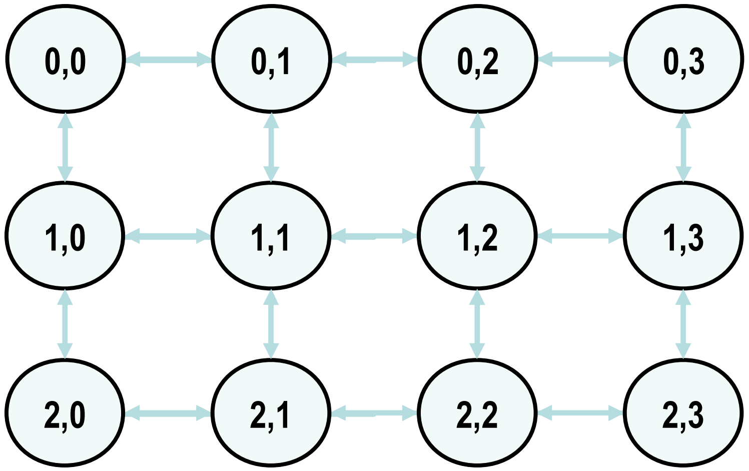
Virtual process topology .vs. topology of the underlying, physical hardware:

- virtual topology can be exploited by the system in the assignment of processes to physical processors, if this helps to improve the communication performance on a given machine.
- the description of the virtual topology depends only on the application, and is machine-independent.

EXAMPLES



RING

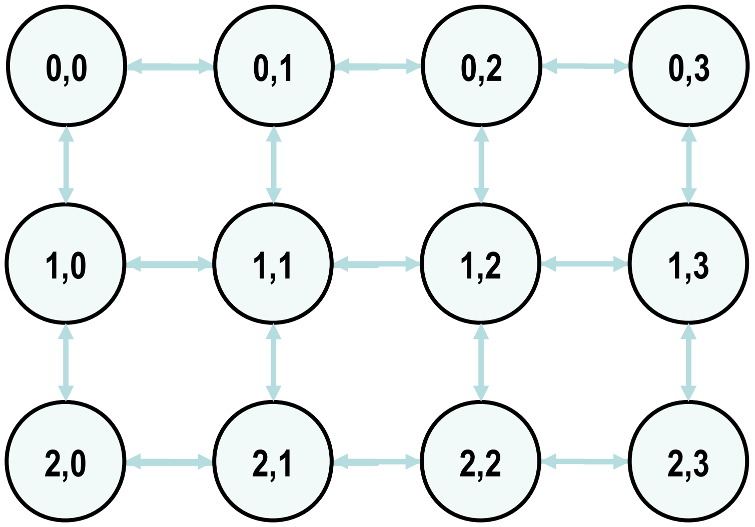


2D-GRID

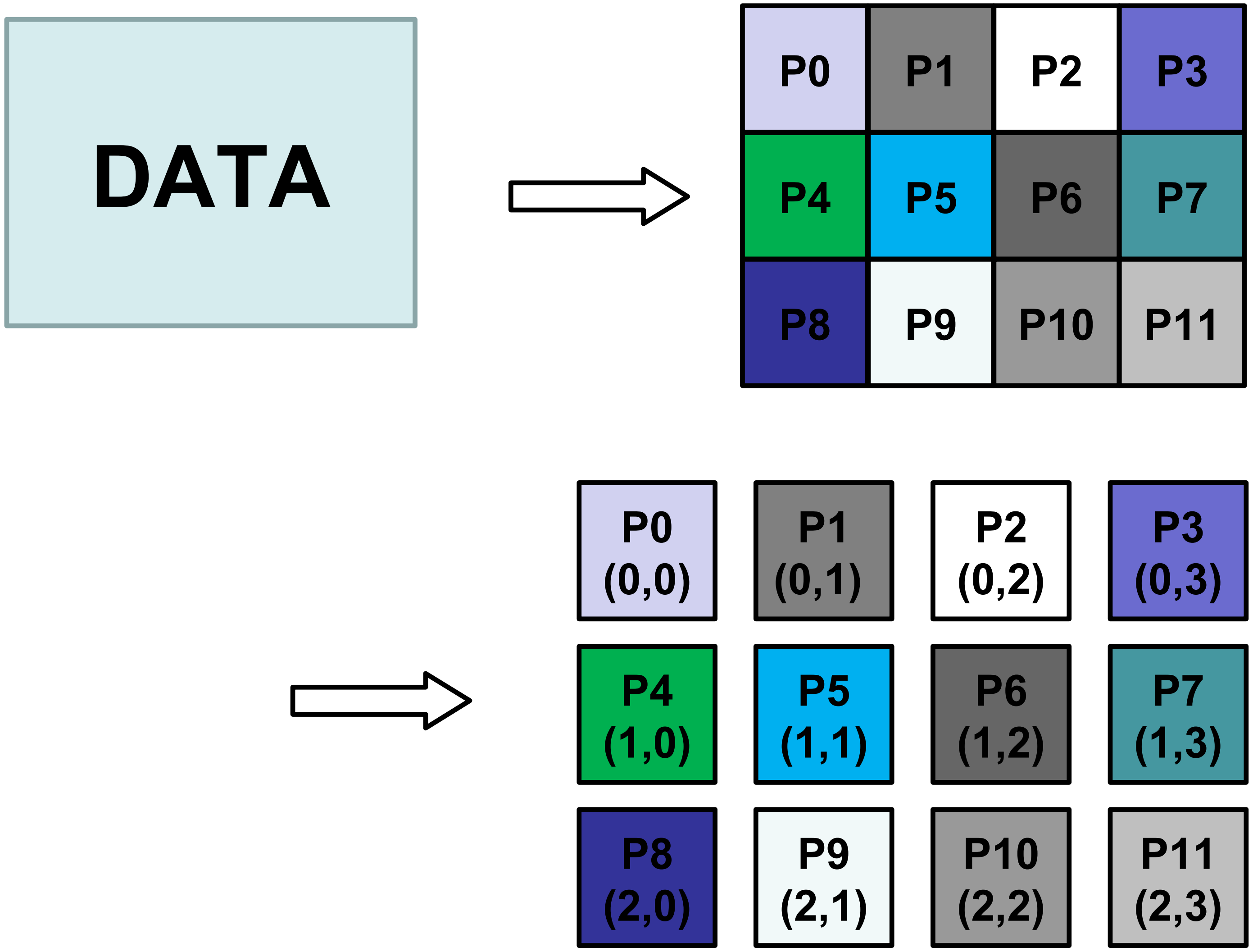
CARTESIAN TOPOLOGY

A grid of processes is easily described with a cartesian topology:

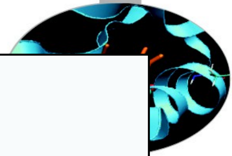
- each process can be identified by cartesian coordinates
- periodicity can be selected for each direction
- communications are performed along grid dimensions only



EXAMPLE: 2D DOMAIN DECOMPOSITION



CARTESIAN CONSTRUCTOR

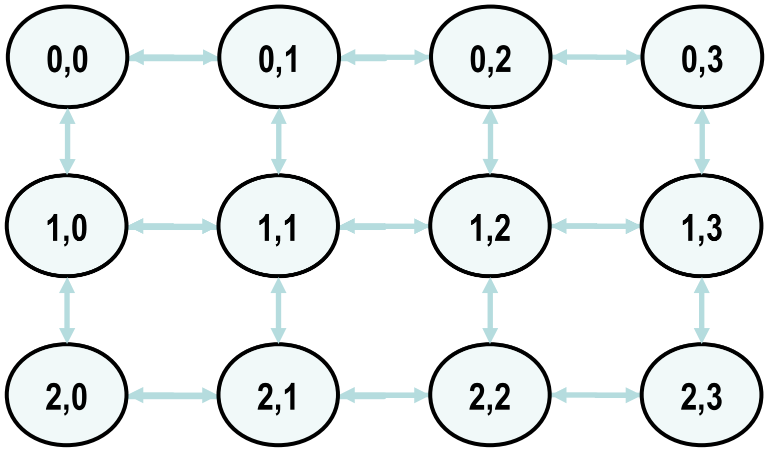


```
MPI_CART_CREATE(comm_old, ndims, dims, periods, reorder,
comm_cart)
IN comm_old:   input communicator (handle)
IN ndims: number of dimensions of Cartesian grid (integer)
IN dims: integer array of size ndims specifying the number of
processes in each dimension
IN periods: logical array of size ndims specifying whether the grid is
periodic (true) or not (false) in each dimension
IN reorder: ranking may be reordered (true) or not (false)
OUT comm_cart: communicator with new Cartesian topology (handle)
```

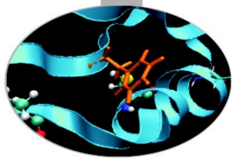
- Returns a handle to a new communicator to which the Cartesian topology information is attached.
- Reorder:
  - false: the rank of each process in the new group is identical to its rank in the old group.
  - True: the processes may be reordered, possibly so as to choose a good embedding of the virtual topology onto physical machine.
- If cart has less processes than the starting communicator, leftover processes have MPI_COMM_NULL as return value

EXAMPLE (C)

```
#include <mpi.h>

int main(int argc, char *argv[])
{

  MPI_Comm cart_comm;
  int dim[] = {4, 3};
  int period[] = {1, 0};
  int reorder = 0;

  MPI_Init(&argc, &argv);

  MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, reorder, &cart_comm);
  ...
}
```



CARTESIAN TOPOLOGY UTILITIES



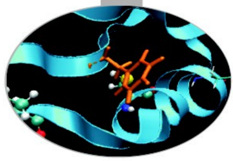
- **MPI_Dims_Create:**
  - compute optimal balanced distribution of processes per coordinate direction with respect to:
    - a given dimensionality
    - the number of processes in a group
    - optional constraints

- **MPI_Cart_coords:**
  - given a rank, returns process coordinates

- **MPI_Cart_rank:**
  - given process coordinates, returns the rank

- **MPI_Cart_shift:**
  - get source and destination rank ids (neighbours) for SendRecv operations

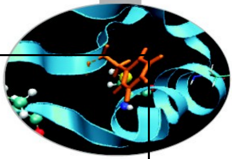
MPI DIMS CREATE



```
MPI_DIMS_CREATE(nnodes, ndims, dims)
IN nnodes: number of nodes in a grid (integer)
IN ndims: number of Cartesian dimensions (integer)
IN/OUT dims: integer array of size ndims specifying the number of
nodes in each dimension
```

- Help user to select a balanced distribution of processes per coordinate direction, depending on the number of processes in the group to be balanced and optional constraints that can be specified by the user
- if `dims[i]` is set to a positive number, the routine will not modify the number of nodes in that i dimension
- negative value of `dims[i]` are erroneous

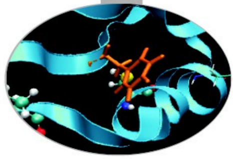
IN/OUT OF "DIMS"



```
MPI_DIMS_CREATE(nnodes, ndims, dims)
IN nnodes: number of nodes in a grid (integer)
IN ndims: number of Cartesian dimensions (integer)
IN/OUT dims: integer array of size ndims specifying the number of
nodes in each dimension
```

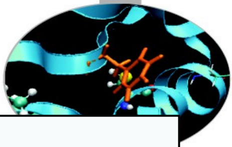
dims before call	Function call	dims on return
(0, 0)	MPI_DIMS_CREATE(6, 2, dims)	(3, 2)
(0, 0)	MPI_DIMS_CREATE(7, 2, dims)	(7, 1)
(0, 3, 0)	MPI_DIMS_CREATE(6, 3, dims)	(2, 3, 1)
(0, 3, 0)	MPI_DIMS_CREATE(7, 2, dims)	erroneous call

USING MPI_DIMS_CREATE (FORTRAN)



```
integer :: dim(3), cube_comm, ierr
logical :: period(3),reorder

CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs,ierr)

dim(1) = 0 ! let MPI arrange
dim(2) = 0 ! let MPI arrange
dim(3) = 3 ! I want exactly 3 planes

CALL MPI_DIMS_CREATE(nprocs, 3, dim, ierr)

if (dim(1)*dim(2)*dim(3) .LE. nprocs) then
  print *,"WARNING: some processes are not in use!"
endif

period = (.true., .true., .false.)
reorder = .false.

CALL MPI_CART_CREATE(MPI_COMM_WORLD, 3, dim, period, reorder, &
cube_comm,ierr)
```

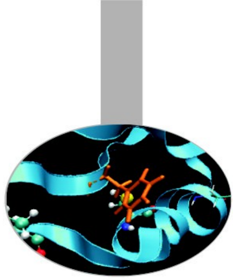
FROM COORDINATE TO RANK



```
MPI_CART_RANK(comm, coords, rank)

IN comm: communicator with Cartesian structure

IN coords: integer array (of size ndims) specifying the Cartesian

coordinates of a process

OUT rank: rank of specified process
```

- translation of the logical process coordinates to process ranks as they are used by the point-to-point routines
- if `dimension i` is periodic, when i-th coordinate is out of range, it is shifted back to the interval `0<coords(i)<dims(i)` automatically
- out-of-range coordinates are erroneous for non-periodic dimensions

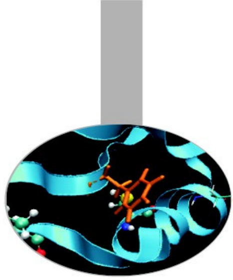
FROM RANK TO COORDINATE



```
MPI_CART_COORDS(comm, rank, maxdim, coords)
IN comm: communicator with Cartesian structure
IN rank: rank of a process within group of comm
IN maxdims: length of vector coords in the calling program
OUT coords: integer array (of size ndims) containing the Cartesain
coordinates of specified process
```

- For each MPI process in Cartesian communicator, the coordinate whitin the cartesian topology are returned

MAPPING OF COORDINATES (C)

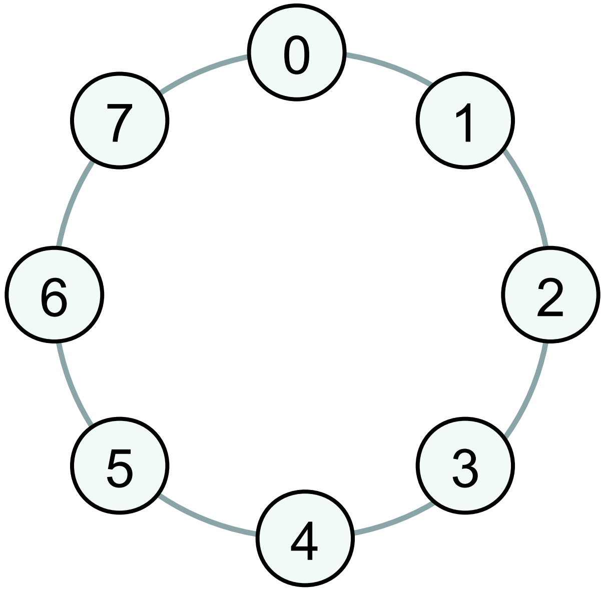


```
int cart_rank;
MPI_Comm_rank(cart_comm, &cart_rank);

int coords[2];
MPI_Cart_coords(cart_comm, cart_rank, 2, coords);

// set linear boundary values on top/left-hand domain
if (coords[0] == 0 || coords[1] == 0) {
  SetBoundary( linear(min, max), domain);
}

// set sinusoidal boundary values along lower domain
if (coords[0] == dim[0]) {
  SetBoundary( sinusoid(), domain);
}

// set polynomial boundary values along right-hand of domain
if (coords[1] == dim[1]) {
  SetBoundary( polynomial(order, params), domain);
}
```

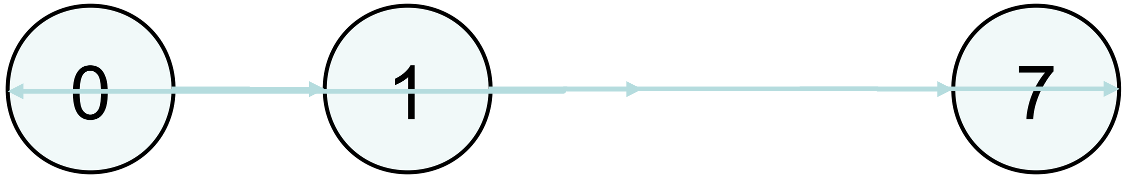
CARTESIAN SHIFT: A 1D CARTESIAN TOPOLOGY

Circular shift is another tipical MPI communication pattern:

- each process communicate only with its neighbors along one direction
- periodic boundary conditions can be set for letting first and last processes partecipate in the communication



such a pattern is nothing more than a 1D cartesian grid topology with optional periodicity

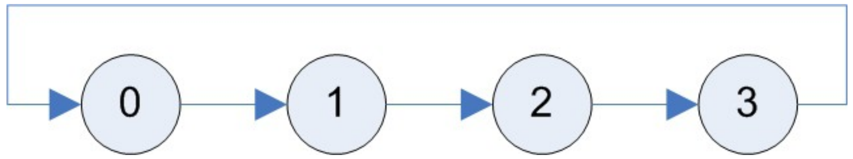


MPI CART SHIFT

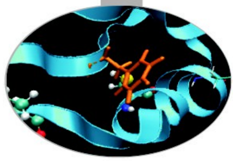
```
MPI_CART_SHIFT(comm, direction, disp, rank_source, rank_dest)
IN comm: communicator with Cartesian structure
IN direction: coordinate dimension of shift
IN disp: displacement (>0: upwards shift; <0: downwards shift
OUT rank_source: rank of source process
OUT rank_dest: rank of destination process
```

- Depending on the periodicity of the Cartesian group in the specied coordinate direction, MPI_CART_SHIFT provides the identifiers for a circular or an end-o shift.
- In the case of an end-o shift, the value **MPI_PROC_NULL** may be returned in rank_source or rank_dest, indicating that the source or the destination for the shift is out of range.
- provides the calling process the ranks of source and destination processes for an MPI_SENDRECV with respect to a specified coordinate direction and step size of the shift

EXAMPLE (FORTRAN)



```
...

integer ::  dim = nprocs
integer ::  period = 1
integer ::  source, dest, ring_comm, status(MPI_STATUS_SIZE),ierr

CALL MPI_CART_CREATE(MPI_COMM_WORLD, 1, dim, period, 0,ring_comm,ierr)

CALL MPI_CART_SHIFT(ring_comm, 0, 1, source, dest, ierr)

CALL MPI_SENDRECV(right_boundary, n, MPI_INT, dest, rtag, left_boundary,
n, MPI_INT, source, ltag, ring_comm, status, ierr)


...
```



PARTITIONING OF CARTESIAN STRUCTURES

- It is often useful to partition a cartesian communicator into subgroups that form lower dimensional cartesian subgrids
  - new communicators are derived
  - lower dimensional communicators cannot communicate among them (unless inter-communicators are used)

MPI CART SUB

MPI_CART_SUB(comm, remain_dims, newcomm)

IN comm: communicator with Cartesian structure

IN remain_dims: the i-th entry of remain_dims specifies whether the i-th dimension is kept in the subgrid (true) or is dropped (false) (logical vector)

OUT newcomm: communicator containing the subgrid that includes the calling process

```
int dim[] = {2, 3, 4};

int remain_dims[] = {1, 0, 1}; // 3 comm with 2x4 processes 2D grid
...
int remain_dims[] = {0, 0, 1}; // 6 comm with 4 processes 1D topology
```