MPI Virtual Topologies

Dr. Süha Tuna

FSMVU – Department of Computer Engineering

Content

1. Definition of Virtual Topology
2. Cartesian Topologies in MPI
3. Graph Topologies in MPI
4. Discussions

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ 2010

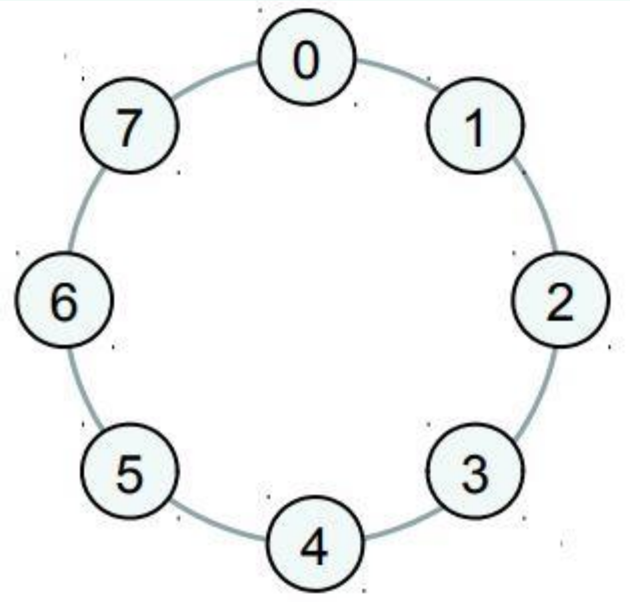
Virtual Topology

- **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).

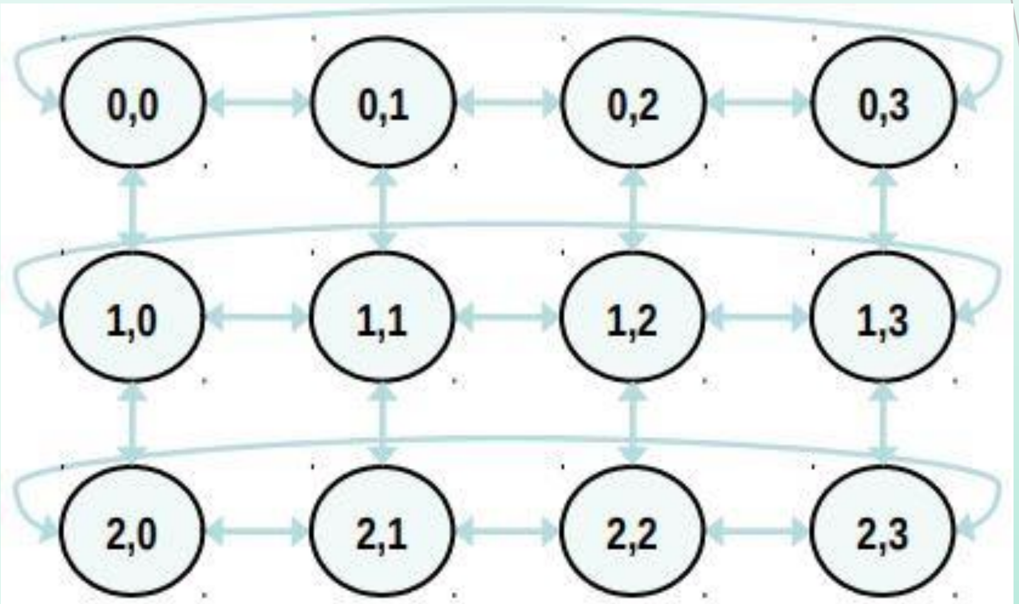
Virtual Topology (cont.)

- 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.

Virtual Topology: Examples



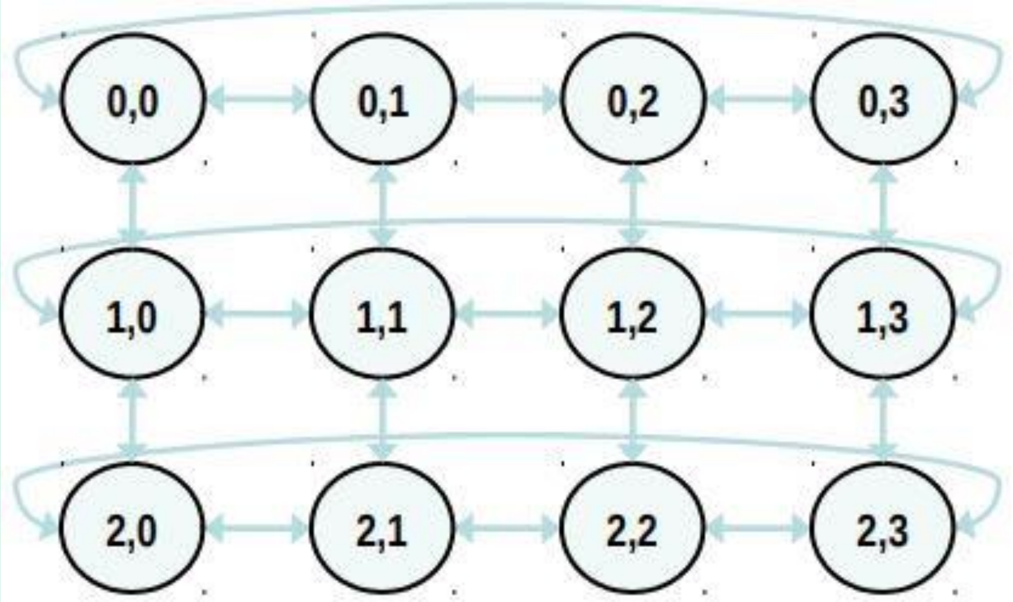
1D GRID, PERIODIC (RING)



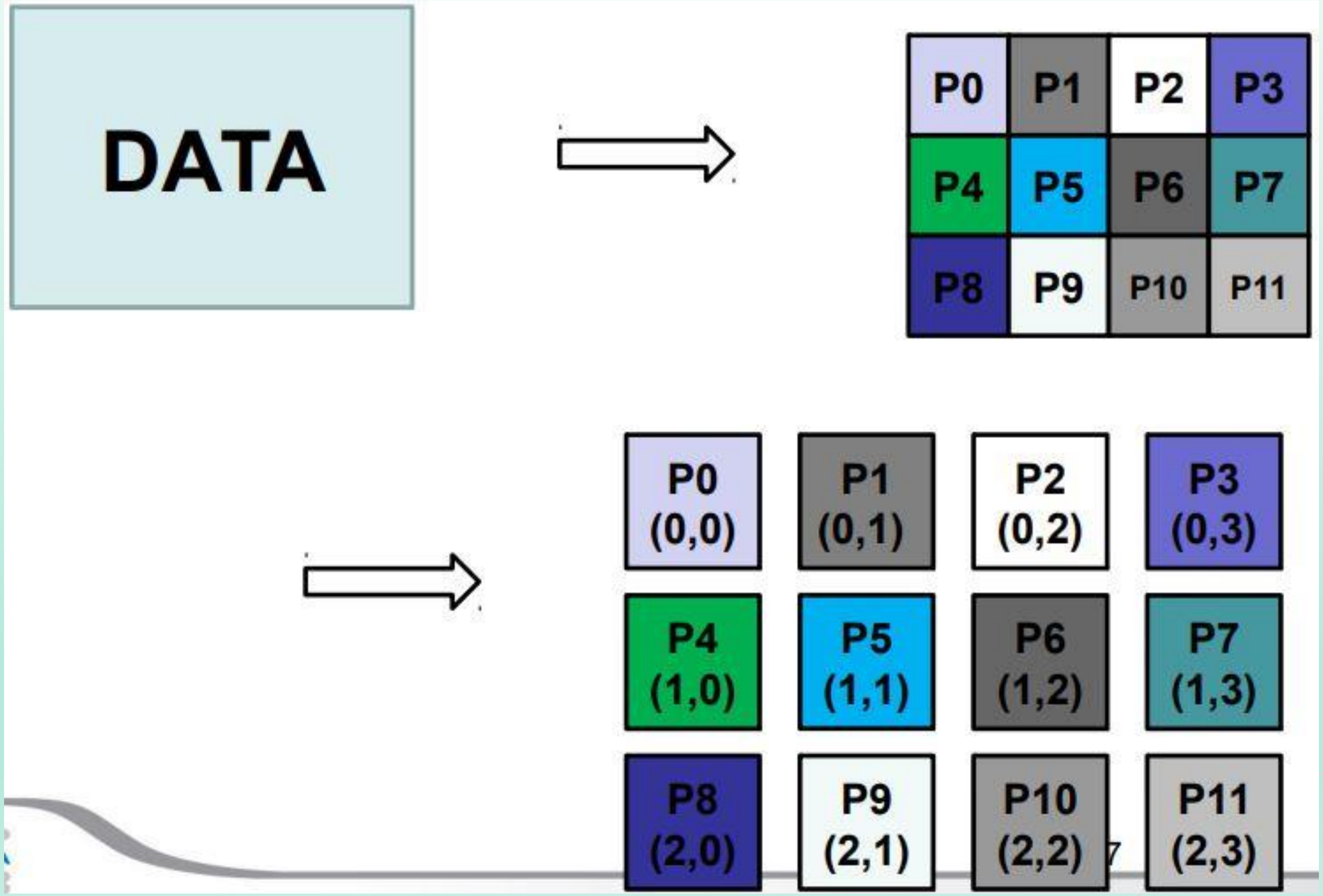
2D GRID, PERIODIC

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 Topology Constructor

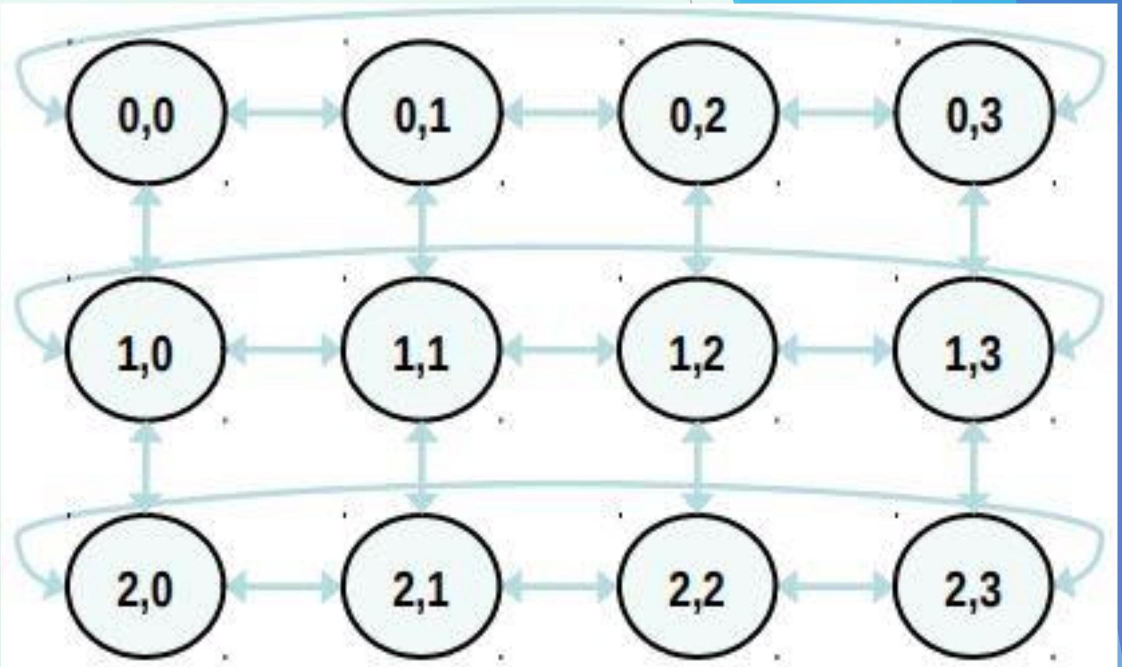
- `int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[], const int periods[], int reorder, MPI_Comm *comm_cart)`
  - `comm_old`: input communicator
  - `ndims`: number of dimensions of Cartesian grid
  - `dims`: integer array of size `ndims` specifying the number of processes in each dimension
  - `periods`: logical array of size `ndims` specifying whether the grid is periodic (true) or not (false) in each dimension
  - `reorder`: ranking may be reordered (true) or not (false)
  - `comm_cart`: communicator with new Cartesian topology

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ 2010

`MPI_Cart_create` properties

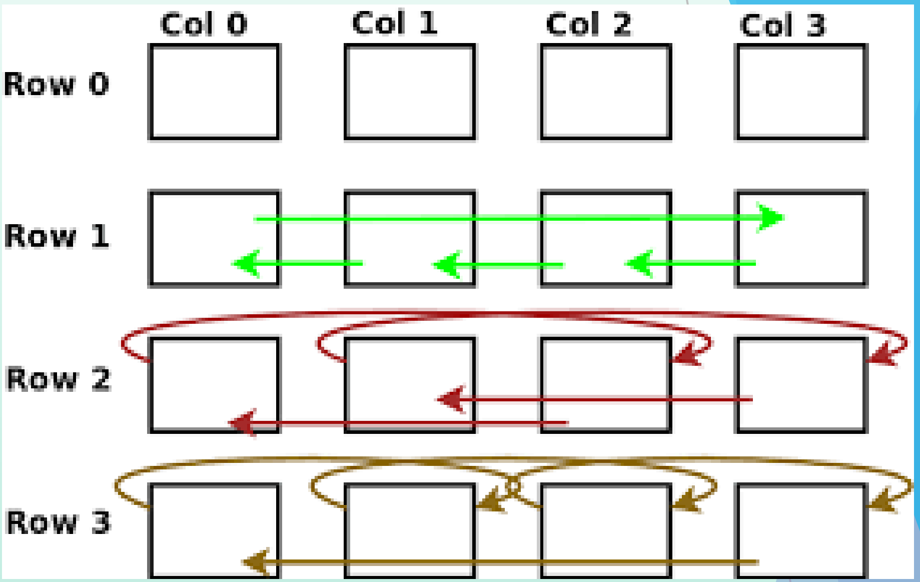
- Returns a handle to a new communicator to which the Cartesian topology information is attached.
- reorder:
  - false: the rank of each process in the new group is identical to its rank in the old group.
  - true: the processes may be reordered, possibly so as to choose a good embedding of the virtual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

How to create a Cartesian Topology




```
#include <mpi.h>
int main(int argc, char *argv[]) {
 MPI_Comm cart_comm;
 int dims[] = {3, 4};
 int period[] = {0, 1};
 int reorder = 1;
 MPI_Init(&argc, &argv);
 MPI_Cart_create(MPI_COMM_WORLD, 2, dims, period, reorder,
&cart_comm);
 ...
}
```

Periodicity




```
#include <mpi.h>
int main(int argc, char *argv[]) {
 MPI_Comm cart_comm;
 int dims[] = {4, 4};
 int period[] = {0, 1};
 int reorder = 1;
 MPI_Init(&argc, &argv);
 MPI_Cart_create(MPI_COMM_WORLD, 2, dims, period, reorder,
&cart_comm);
 ...
}
```

Cartesian Topology routines in MPI

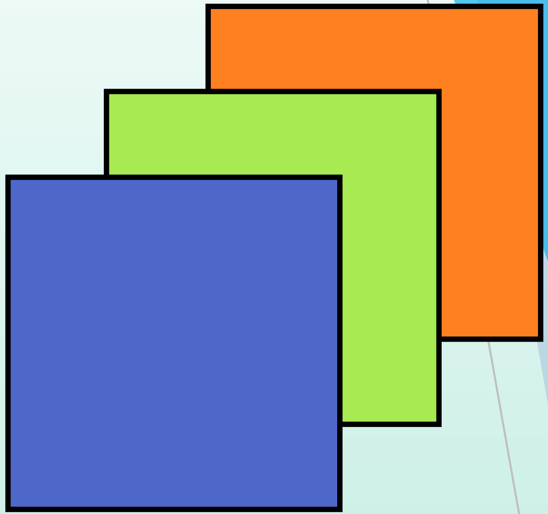
- `MPI_Dims_create`:
  - compute optimal balanced distribution of processes per coordinate direction with respect to:
    - a given dimensionality
    - the number of processes in a group
    - optional constraints
- `MPI_Cart_coords`:
  - given a rank, returns process's coordinates
- `MPI_Cart_rank`:
  - given process's coordinates, returns the rank
- `MPI_Cart_shift`:
  - get source and destination rank id’s in *Sendrecv* operations

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ 2010

`MPI_Dims_create` Binding

- `int MPI_Dims_create(int nnodes, int ndims, int dims[])`
  - `nnodes`: number of nodes in a grid
  - `ndims`: number of Cartesian dimensions
  - `dims`: integer array of size `ndims` specifying the number of nodes in each dimension
- helps 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

Using `MPI_Dims_create`



```
…
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
int dims[3];
dims[0] = 0; // let MPI arrange
dims[1] = 0; // let MPI arrange
dims[2] = 3; // I want exactly 3 planes
MPI_Dims_create(nprocs, 3, dims);
if (dims[0]*dims[1]*dims[2] < nprocs) {
 fprintf(stderr, "WARNING: some processes are not in use!\n"
}
int period[] = {1, 1, 0};
int reorder = 0;
MPI_Cart_create(MPI_COMM_WORLD, 3, dims, period, reorder,
&cube_comm);
…
```

`MPI_Cart_rank`: coordinates → rank

- `MPI_Cart_rank(MPI_Comm comm, const int coords[], int *rank)`
  - `comm:` communicator with Cartesian structure
  - `coords:` integer array (of size `ndims`) specifying the Cartesian coordinates of a process
  - `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

Example: mapping, old and new ranks

```
// buffer to collect MPI_COMM_WORLD rank ids in new cartesian rank sorting
int *world_ranks = (int *) malloc (nprocs, sizeof(int));
int oldrank;
MPI_Comm_rank(MPI_COMM_WORLD, &oldrank);
MPI_Cart_create(MPI_COMM_WORLD, 2, dim, period, 1, &comm_cart);
// indexing sorting is now performed on rank id of comm_cart communicator
MPI_Gather(&oldrank, 1, MPI_INT, world_ranks, 1, MPI_INT, 0, comm_cart);

if (oldrank == 0) {
  for (int i=0; i<dim[0]; i++) {
    for (int j=0; j<dim[1]; j++) {
      int new_rank;
      int coords[2]; coords[0]=i; coords[1]=j;
      MPI_Cart_rank(cart_comm, coords, &new_rank);
      printf("([%d, %d]) ", new_rank, world_ranks[new_rank]);
     }; printf("\n");
  }
}
```

`MPI_Cart_coords`: rank → coordinate

- `int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int coords[])`
  - `comm:` communicator with Cartesian structure
  - `rank:` rank of a process within group of comm
  - `maxdims:` length of vector `coords` in the calling program
  - `coords:` integer array (of size `ndims`) containing the Cartesian coordinates of specified process

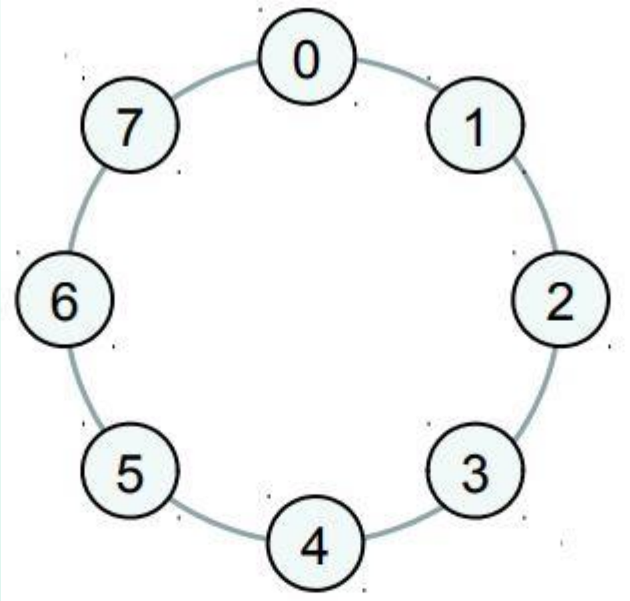
- For each MPI process in Cartesian communicator, the coordinate within the Cartesian topology are returned

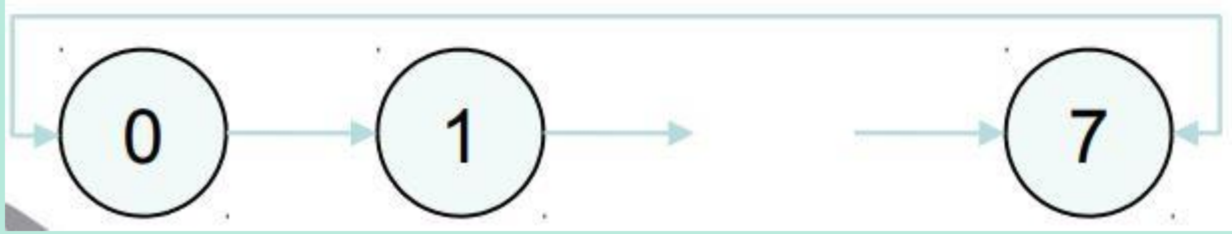
Usage of MPI_Cart_coords

```
. . .
ndim = (int*)calloc(dim,sizeof(int));
ndim[0] = row; ndim[1] = col;
period = (int*)calloc(dim,sizeof(int));
period[0] = period[1] = 0;
reorder = 0;
// 2D grid creation
MPI_Cart_create(MPI_COMM_WORLD,dim,ndim,period,reorder,
&comm_grid);
MPI_Comm_rank(comm_grid,&menum_grid);
// Coordinate of each mpi rank within the cartesian communicator
MPI_Cart_coords(comm_grid,menum,dim,coordinate);
printf("Procs %d coordinates in 2D grid (%d,%d) \n", menum,
*coordinate, *(coordinate+1));
. . .
```

Circular Shift: a 1D Cartesian Topology

- Circular shift is another typical MPI communication pattern:
  - each process communicates only with its neighbours along one direction
  - periodic boundary conditions should be set for letting first and last processes participate in the communication



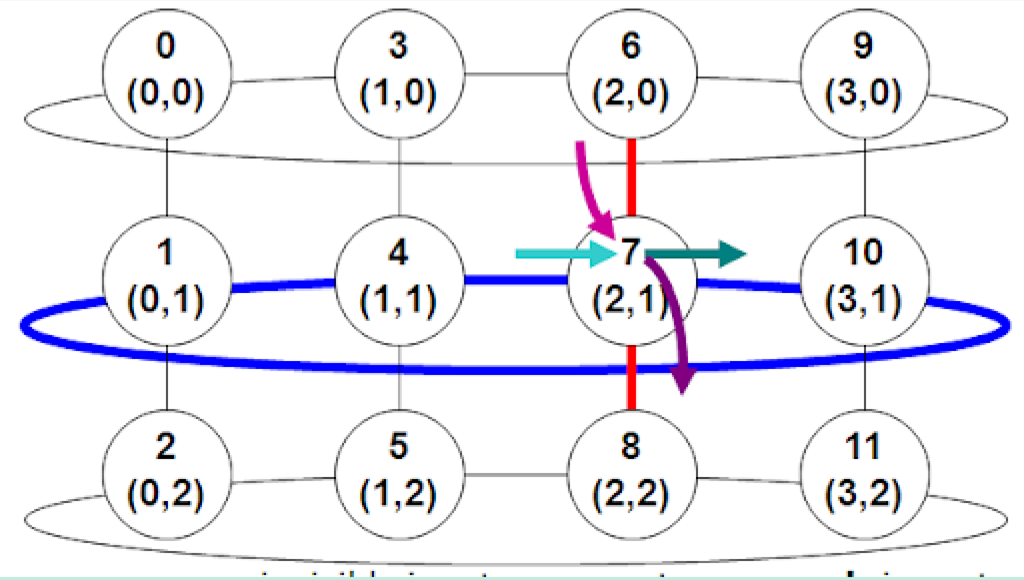


`MPI_Cart_shift` routine

- `int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)`
  - `comm`: communicator with Cartesian structure
  - `direction`: coordinate dimension of shift
  - `disp`: displacement (>0: upwards shift; <0: downwards shift)
  - `rank_source`: rank of source process
  - `rank_dest`: rank of destination process

- Returns the shifted source and destination ranks, given a shift direction and amount

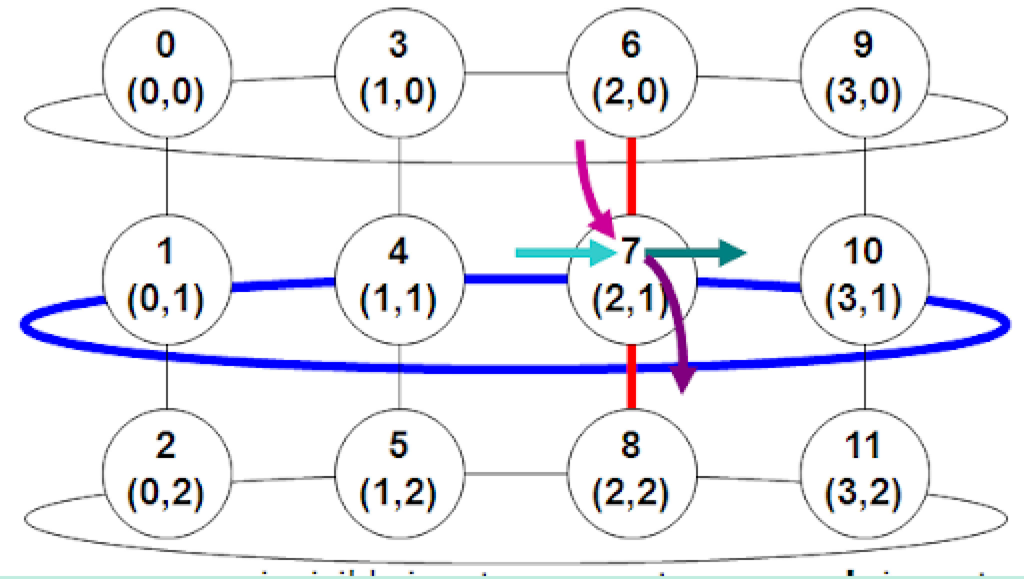
MPI_Cart_shift (vertical displacement)



- `MPI_Cart_shift(cartCOMM, 0, 1, *up, *down);`

0th direction: vertical

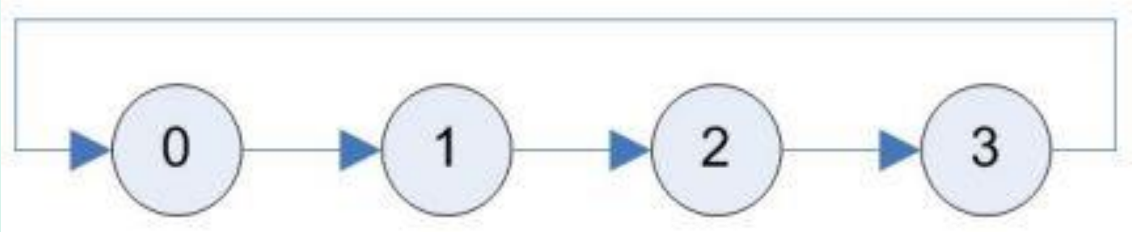
MPI_Cart_shift (horizontal displacement)



- `MPI_Cart_shift(cartCOMM, 1, 1, *left, *right);`

1st direction: horizontal

Sendrecv with 1D Cartesian Topologies



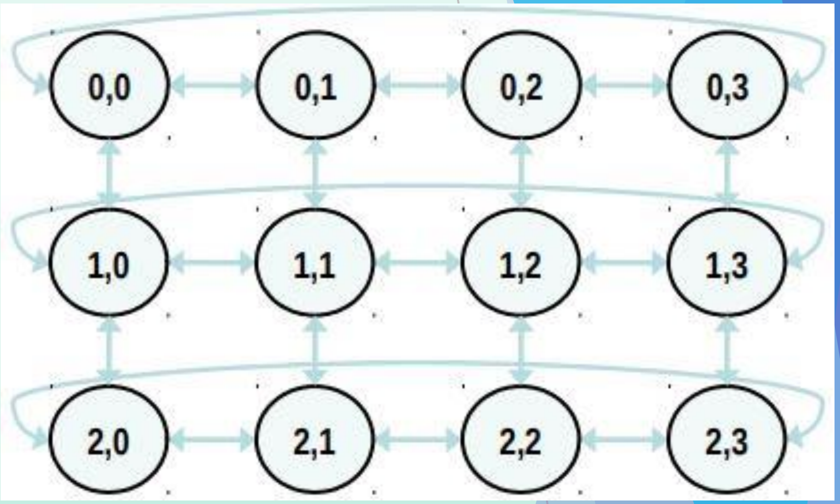

```
...
int dim[1], period[1];
dim[0] = nprocs;
period[0] = 1;
MPI_Comm ring_comm;

MPI_Cart_create(MPI_COMM_WORLD, 1, dim, period, 0, &ring_comm);
int source, dest;
MPI_Cart_shift(ring_comm, 0, 1, &source, &dest);
MPI_Sendrecv(&toRight, n, MPI_INT, dest, rtag,
 &fromLeft, n, MPI_INT, source, ltag, ring_comm, &status);
...
```

sendrecv with 2D Cartesian Topologies

```
...
int dim[] = {3, 4};
int period[] = {0, 1};
MPI_Comm grid_comm;

MPI_Cart_create(MPI_COMM_WORLD, 2, dim,
    period, 0, &grid_comm);
int source, dest;
for (int dimension = 0; dimension < 2; dimension++) {
  for (int versus = -1; versus < 2; versus+=2;) {
    MPI_Cart_shift(grid_comm, dimension, versus, &source, &dest);
    MPI_Sendrecv(buffer, n, MPI_INT, source, stag, buffer, n, MPI_INT, dest,
dtag, grid_comm, &status);
  }
}
```



Partitioning of Cartesian Structures

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

MPI_Cart_sub routine

- `int MPI_Cart_sub(MPI_Comm comm, const int remain_dims[], MPI_Comm *newcomm)`
  - `comm`: communicator with Cartesian structure
  - `remain_dims`: the i-th entry of remain_dims specifies whether the i-th dimension is kept in the sub-grid (true) or is dropped (false) (logical vector)
  - `newcomm`: communicator containing the sub-grid that includes the calling process
- Example:

```
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
```

Example:

```
/* First, create a 1-dim cartesian communicator */
    periods[0] = 0;
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    dims[0] = size;
    MPI_Cart_create( MPI_COMM_WORLD, 1, dims, periods, 0, &comm );

/* Now, extract a communicator with no dimensions */
    remain[0] = 0;
    MPI_Cart_sub( comm, remain, &newcomm );

/* Free the new communicator */
    MPI_Comm_free( &newcomm );
    MPI_Comm_free( &comm );
```

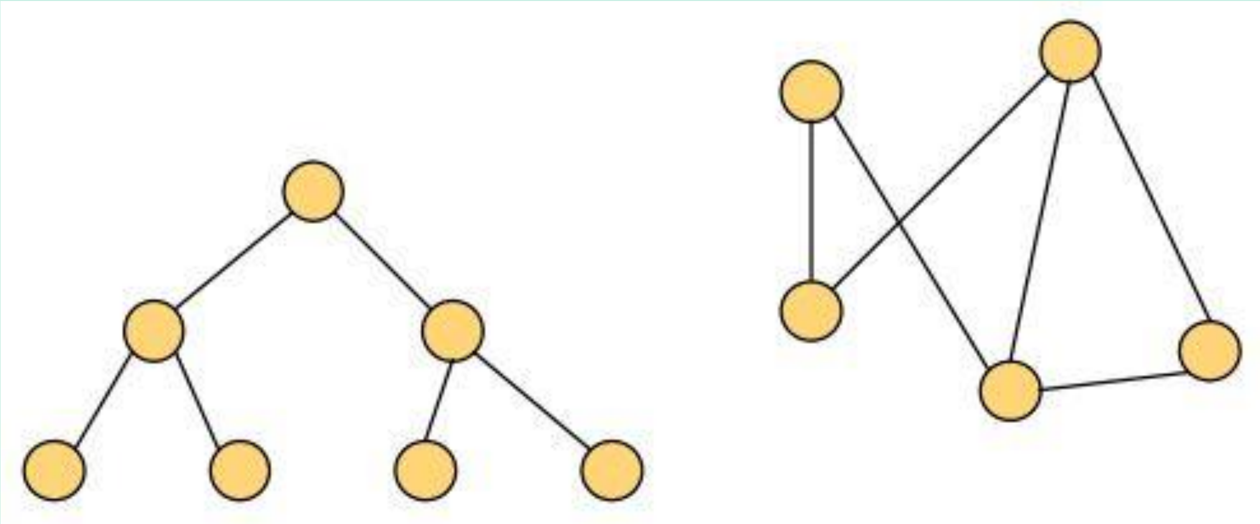
MPI_Cart_get routine

- `int MPI_Cart_get(MPI_Comm comm, int maxdims, int dims[], int periods[], int coords[])`
  - `comm`: communicator with Cartesian structure
  - `maxdims`: length of vectors `dims`, `periods`, and `coords` in the calling program
  - `dims`: number of processes for each cartesian dimension
  - `periods`: periodicity (true/false) for each cartesian dimension
  - `coords`: coordinates of calling process in Cartesian structure

- Retrieves Cartesian topology information associated with a communicator

Graph Topology: Introduction

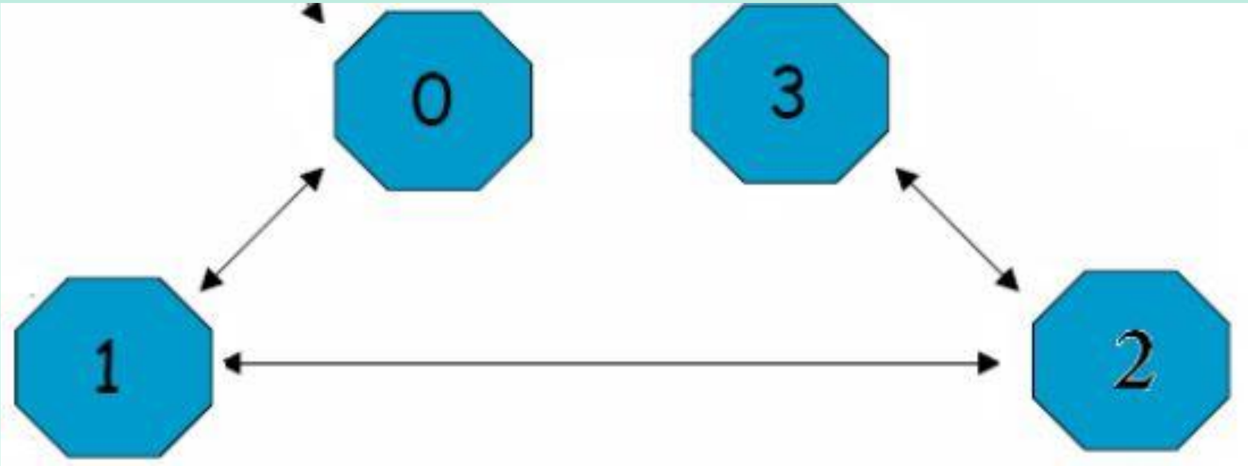
- Graph topology gives opportunity to make optional connections between processes to programmers
- We use hierarchical systems which are given by graph topology for solving weakness problem of MPI topology.
- More generally, the process organization is described by a graph



Elements of graph topology

- Communication link(s)
- Nodes (processes)
- Neighbours (edges)
  - index: array of `int`
- Type of mapping

Node	# neighb.	Index	Edges
0	1	1	1
1	2	3	0, 2
2	2	5	1, 3
3	1	6	2



Properties of graph topology

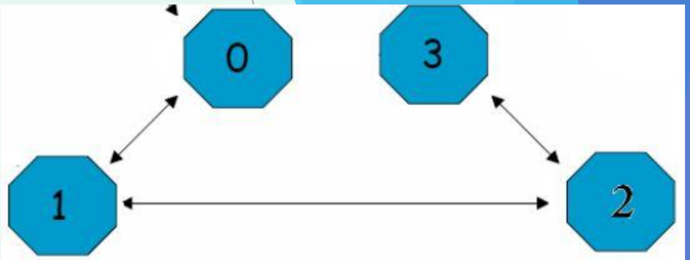
- Graph topology can only be used in intra-communicators.
- Number of graph nodes must not be more than number of processors.
- In a graph, communication speed may increase if process addressing reordered by system.
- One node can be neighbour of another when opposite can not be. This means asymmetric structure can be exploited.
- For only IBM, Graph topologies must be symmetric. If x is a neighbour of y, then y is a neighbour of x.

MPI_Graph_create routine

- `int MPI_Graph_create(MPI_Comm comm_old, int nnodes, const int index[], const int edges[], int reorder, MPI_Comm *comm_graph)`
  - `comm_old`: input communicator without topology
  - `nnodes`: number of nodes in graph
  - `index`: array of integers describing node degrees
  - `edges`: array of integers describing graph edges
  - `reorder`: ranking may be reordered (true) or not (false)
  - `comm_graph`: communicator with graph topology added

- Makes a new communicator to which topology information has been attached

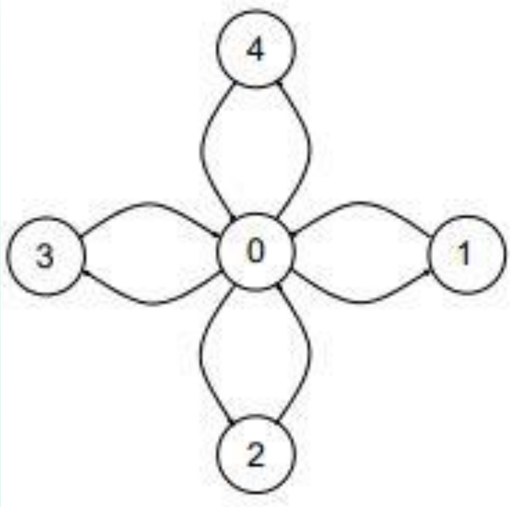
Example1: Creating a graph in MPI




```
MPI_Comm graph_comm;
int nnodes = 4; /* number of nodes */
int index[4] = {1, 3, 5, 6}; /* index definition */
int edges[6] = {1, 0, 2, 1, 3, 2}; /* edges
definition */
int reorder = 1; /* allows processes
reordered for efficiency */
MPI_Graph_create(MPI_COMM_WORLD, nnodes,
   index, edges, reorder, &graph_comm);
```

Node	# neighb.	Index	Edges
0	1	1	1
1	2	3	0, 2
2	2	5	1, 3
3	1	6	2

Example2: Constructing star topology



Node	# of neighb.	Index	Edges
0	4	4	1,2,3,4
1	1	5	0
2	1	6	0
3	1	7	0
4	1	8	0

```
int index[] = { 4, 5, 6, 7, 8 };
int edges[] = { 1, 2, 3, 4, 0, 0, 0, 0 };
MPI_Comm StarComm;
MPI_Graph_create(MPI_COMM_WORLD, 5, index, edges,
   1, &StarComm);
```

MPI_Graph_neighbors_count routine

- `int MPI_Graph_neighbors_count(MPI_Comm comm; int rank; int *nneighbors)`
  - `comm`: communicator structered by a graph topology
  - `rank`: rank of the process in `comm`
  - `nneigbors`: number of neighbours of the process `rank`

- Returns the number of neighbors of a node associated with a graph topology

MPI_Graph_neighbors routine

- `int MPI_Graph_neighbors( MPI_Comm comm, int rank, int maxneighbors, int *neighbors );`
  - `comm`: communicator with graph topology
  - `rank`: rank of process in group of comm
  - `maxneighbors`: size of array neighbors
  - `neighbors`: ranks of processes that are neighbors to specified process

- Returns the neighbors of a node associated with a graph topology

Implementation

Node	# of neighb.	Index	Edges
0	1	1	1
1	2	3	0, 2
2	2	5	1, 3
3	1	6	2

```
int node, nneighbors, my_edges[2];
…
MPI_Comm_rank(graph_comm, &node);
…
MPI_Graph_neighbors_count(graph_comm, node, &nneighbors);
MPI_Graph_neighbors(graph_comm, 2, nneighbors, my_edges);
```

- Input: node=2
- Output: nneighbors=2, my_edges={1, 3}

MPI_Graph_get routine

- `int MPI_Graph_get(MPI_Comm comm, int maxindex, int maxedges, int index[], int edges[])`
  - `comm`: communicator with graph structure
  - `maxindex`: length of vector indx in the calling program
  - `maxedges`: length of vector edges in the calling program
  - `index`: array of integers containing the graph structure
  - `edges`: array of integers containing the graph structure
- Retrieves graph topology information associated with a communicator

MPI_Graphdims_get routine

- `int MPI_Graphdims_get(MPI_Comm comm, int *nnodes, int *nedges)`
  - `comm`: communicator for group with graph structure
  - `nnodes`: number of nodes in graph
  - `nedges`: number of edges in graph

- Retrieves graph topology information associated with a communicator

Example:

```
int nnodes, nedges, index[4], edges[6];
…
MPI_Graphdims_get(graph_comm, &nnodes, &nedges);
MPI_Graph_get(graph_comm, nnodes, nedges, index, edges);
```

- Output:
  - nnodes=4
  - nedges=6
  - index= {1,3,5,6}
  - edges={1,0,2,1,3,2}

Node	# of neighb.	Index	Edges
0	1	1	1
1	2	3	0, 2
2	2	5	1, 3
3	1	6	2

`MPI_Topo_test` routine

- `int MPI_Topo_test(MPI_Comm comm, int *status)`
  - `comm`: communicator (handle)
  - `status`: topology type of communicator comm (integer).

- If the communicator has no associated topology, returns `MPI_UNDEFINED`.

- Determines the type of topology (if any) associated with a communicator

FATİH SULTAN MEHMET VAKIF ÜNİVERSİTESİ 2010

Discussions

- Advantages of the use of the additional communicators
- Benefits of using the Cartesian and Graph virtual topologies