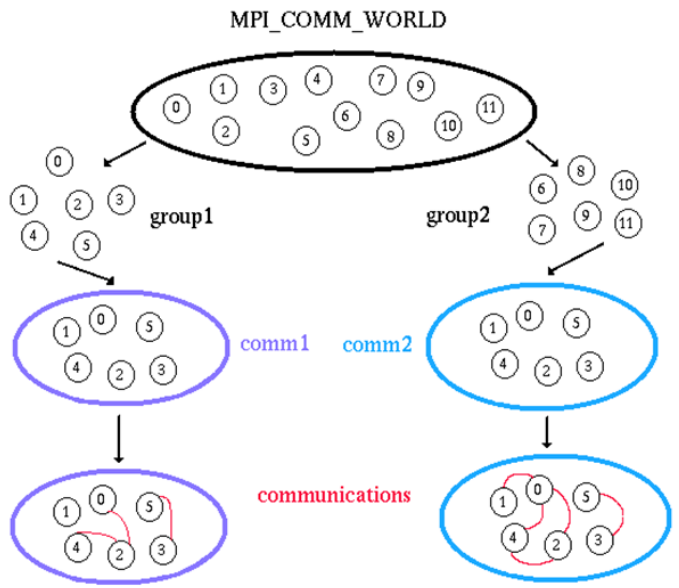
Lecture 7: Advanced MPI – Communicators and Process Topologies

MPI Communicators

- Communicators provides a separate communication space. It is possible to treat a subset of processes as a communication universe.
- MPI_COMM_WORD



- Can create sub-groups of processes, or sub-communicators

- Two types of communicators:
  - **Intra-communicator** : a collection of processes that can send messages to each other and engage in collective communication operations.
  - **Inter-communicator**: are used for sending messages between processes belonging to disjoint intra-communicators.

Intra-Communicator

- A intra-communicator is composed of:
  - A **group**: is an ordered collection of processes. If a group consists of *p* processes, each process in the group is assigned a unique **rank**, which is a non-negative integer in the range 0, 1, …, *p*-1.
  - A **context**: a system-defined object that uniquely identifies a communicator. Two distinct communicators have different contexts, even if they have identical underlying groups.
  - **Attributes:** topology

Remark: A minimal intra-communicator has at least group and context.

Working with Groups, Contexts and Communicators

Assume that there are p processes under MPI_COMM_WORLD, where $q^2 = p$.

- Create new communicator with a group of processes with ranks 0, 1, …, $q - 1$.

Code 1

```
MPI_Group     group_world;
MPI_Group     first_row_group;
MPI_Comm      first_row_comm;
int           *process_ranks;

// make a list of processes in the new communicator
process_ranks = (int*) malloc(q*sizeof(int));
for(int I = 0; I < q; I++)
   process_ranks[I] = I;

//get the group under MPI_COMM_WORLD
MPI_Comm_group(MPI_COMM_WORLD, &group_world);

// create the new group
MPI_Group_incl(group_world, q, process_ranks, &first_row_group);

// create the new communicator
MPI_Comm_create(MPI_COMM_WORLD, first_row_group, &first_row_comm);
```

int MPI_Comm_group(MPI_Comm *comm*, **MPI_Group** **group* **);**

Obtain the group associated with the *comm.*

- *comm*: communicator
- *group*: group in communicator (handle)

int MPI_Group_incl(MPI_Group *group*, **int** *n*, **int** **ranks*, **MPI_Group** **newgroup* **);**

Produces a group by reordering an existing group and taking only listed members

- *n*: number of elements in array *ranks*
- *ranks:* ranks of processes in *group* to appear in *newgroup*
- *newgroup:* new group constructed

int MPI_Comm_create(MPI_Comm *comm*, **MPI_Group** *group*, **MPI_Comm** **newcomm* **);**

Creates a new communicator, which implicitly associate context and group.

```
// continue on Code 1
double x = 10.0;

// broadcast x to processes in first_row_comm
MPI_Bast(x, 1, MPI_DOUBLE, 0,
first_row_comm);
```

Local and Collective Operation

- **MPI_Comm_create()**
  - *This is a collective operation.* All the processes in *comm* **must** call this function, regardless whether processes join new communicator or not.
- **MPI_Group_incl() & MPI_Comm_group()**
  - These are local operations. No communication among processes are involved.

MPI_Comm_split() to form communicators

int MPI_Comm_split(MPI_Comm *comm*, **int** *color*, **int** *key*, **MPI_Comm** **newcomm* **);**

- Creates new communicators based on colors and keys. This function partitions the group associated with comm into disjoint subgroups, one for each value of color. Each subgroup contains all processes of the same color. Within each subgroup, the processes are ranked in the order defined by the value of the argument key, with ties broken according to their rank in the old group.
- *comm*: old communicator
- *color*: control of subset assignment (nonnegative integer). Processes with the ***same color*** are in the same new communicator
- *key*: control of rank assignment
- *newcomm*: new communicator
- This is a collective call. Each process can provide its own *color* and *key*.

Example. Split processes with odd and even ranks into 2 communicators

```
#include "mpi.h"
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
   int myid,   numprocs;
   int color,  broad_val, 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);

    if(new_id == 0) broad_val = color;
    MPI_Bcast(&broad_val, 1, MPI_INT, 0, New_Comm);
    cout<<"Old_proc["<< myid <<"] has new rank " << new_id <<"received value "<< broad_val <<endl;
    MPI_Finalize();
}
```

See another sample "sample_grid_split_comm.cpp" for splitting process grid.

Virtual Topologies

- It is possible to associate additional information (beyond group and context) with a communicator.
- **Topology** is one of the attributes for communicator.
- In MPI, a **topology** is a mechanism for associating different addressing schemes with the processes belonging to a group.
- MPI topology is a *virtual* topology: there is no simple relation between the process structure and actual underlying physical structure of the parallel system.
- Two main topology types: Cartesian (or grid) and graphs. Graphs are the more general case.

Associating Square Grid Structure with MPI_COMM_WORLD

Specify:

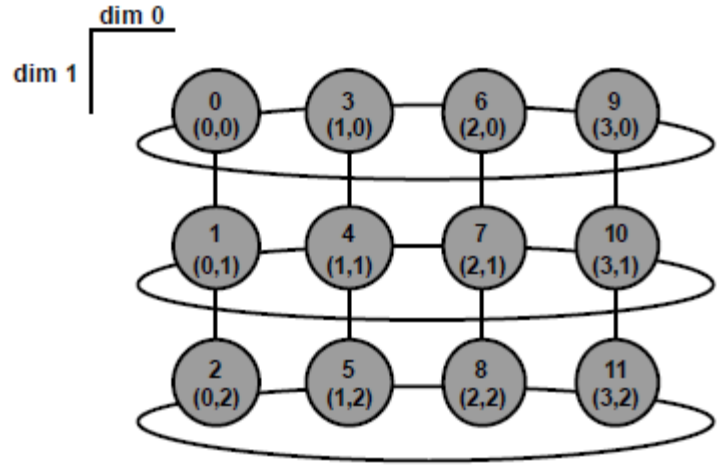
- Number of grid dimensions
- The size of each dimension: number of rows q and number of columns q
- Periodicity of each dimension (whether wraparound). This specifies whether the first entry in each row or columns is “adjacent: to the last entry in that row or column, respectively.
- MPI gives the user the option of allowing the system to optimize the mapping of the grid of processes to the underlying physical processors by possibly reordering the processes in the group underlying the communicator.

Creating a Cartesian Virtual Topology

```
MPI_Comm  grid_comm;
int       dim_sizes[2];
int       wrap_around[2];
int       reorder = 1;

dim_sizes[0] = 4;
dim_sizes[1] = 3;
wrap_around[0] =   1; wrap_around[1] = 0;
MPI_Cart_create(MPI_COMM_WORLD, 2, dim_sizes,
      wrap_around, reorder, &grid_comm);
```

"sample_cart.cpp"



- **int MPI_Cart_create(MPI_Comm** *comm_old*, **int** *ndims*, **int** **dims*, **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) (logical)
  - *comm_cart*[out]: communicator with new cartesian topology (handle)

Cartesian Mapping

- The processes in grid_comm are ranked in *row-major* order. Thus it may be advantageous to change the relative ranking of the processes in MPI_COMM_WORLD.

```
int        coordinates[2];
int        my_grid_rank;


MPI_Comm_rank(grid_comm, &my_grid_rank);
MPI_Cart_coords( grid_comm, my_grid_rank, 2, coordinates);
```

- For a process to determine its coordinates:
  - reorder = 1: The original process ranking in MPI_COMM_WORLD may change. It is necessary to call MPI_Comm_rank()
  - The inverse to MPI_Cart_coords() is MPI_Cart_rank(grid_comm, coordinates, &grid_rank);

Finding “Nearby” Neighbors

int MPI_Cart_shift(MPI_Comm *comm*, **int** *direction*, **int** *displ*, **int** **source*, **int** **dest* **);**

- Returns the shifted source and destination ranks, given a shift direction and amount.
- *direction*[in]: coordinate dimension of shift
- *displ*[in]: displacement (> 0: upwards shift, < 0: downwards shift)
  - *source* is obtained by subtracting *disp* from the *n*th coordinate of the local task, where *n* is equal to *direction*. Similarly, *dest* is obtained by adding *disp* to the *n*th coordinate.
- *source*[out]: rank of source process
- *dest*[out]: rank of destination process
- See sample code “sample_cart_shift_2.cpp”

Partitioning a Grid into Grids of Lower Dimension

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

- creates new communicators for subgrids of up to (N-1) dimensions from an N-dimensional Cartesian grid.
- *remain_dims*[in]: the ith entry of remain_dims specifies whether the ith dimension is kept in the subgrid (true) or is dropped (false) (logical vector)
- *newcomm*[out]: communicator containing the subgrid that includes the calling process

```
int             free_coords[2];
MPI_Comm        row_comm;

free_coords[0] = 0;
free_coords[1] = 1;
MPI_Cart_sub(grid_comm, free_coords, &row_comm);
```

- Create communicators for the row of the grid.
- free_coords[] specifies whether each dimension “belongs” to the new communicator.
- Each new communicator consists of the processes obtained by fixing the row coordinates and letting the column coordinates vary.

```
/* Create 2D Cartesian topology for processes */
MPI_Cart_create(MPI_COMM_WORLD, ndim, dims, period, reorder, &comm2D);
MPI_Comm_rank(comm2D, &id2D);
MPI_Cart_coords(comm2D, id2D, ndim, coords2D);
/* Create 1D row subgrids */
belongs[0] = 0;
belongs[1] = 1; // this dimension belongs to subgrid
MPI_Cart_sub(comm2D, belongs, &commrow);
/* Create 1D column subgrids */
belongs[0] = 1; // this dimension belongs to subgrid
belongs[1] = 0;
MPI_Cart_sub(comm2D, belongs, &commcol);
```

0,0 (0)	**0, 1 (1)**
1, 0 (2)	**1, 1 (3)**
2, 0 (4)	**2, 1 (5)**

2D Cartesian Grid

0, 0 (0) **0 (0)**	**0, 1 (1)** **1 (1)**
1, 0 (2) 0 (0)	1, 1, (3) 1 (1)
2, 0 (4) 0 (0)	2, 1 (5) 1 (1)

3 row subgrids

0, 0 (0) **0 (0)**	**0, 1 (1)** **0 (0)**
1, 0 (2) 1 (1)	1, 1 (3) 1 (1)
2, 0 (4) 2 (2)	2, 1 (5) 2 (2)

2 column subgrids

MPI_Cart_Sub and MPI_Comm_split

- Both can divide a logical grid into arbitrary subgrids.
- MPI_Comm_split is more general than MPI_Cart_create.
- MPI_Comm_split creates logical grid and is referred to by its linear rank number; MPI_Cart_sub creates cartesian grid and rank can be referred to by cartesian coordinates.
- MPI_Cart_sub must be used with communicator associated with a grid topology.