

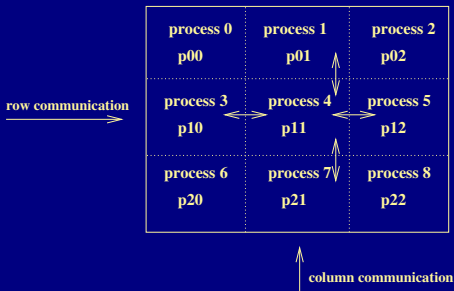
HIGH-PERFORMANCE COMPUTING IN APPLIED MATHEMATICS

MTH 410/510

Communicators and Topologies

So far we have used `MPI_COMM_WORLD` as a communicator among all processes. In many practical situations it is of interest to associate a grid structure with the processes and to create communicators consisting on subgroups of processes.

For example, when performing matrix operations or solving partial differential equations on a two-dimensional spatial domain it may be convenient to view the processes as a grid



with communication required along the rows and the columns.

Communicators and Topologies

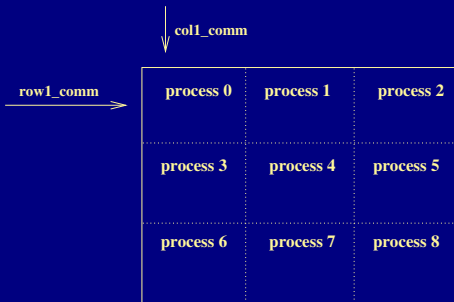
A communicator is a collection of processes that can send messages to each other.

Intra-communicators are a collection of processes that can send messages to each other *and* engage in collective communication operations.

An intra-communicator is composed of

- a group: an ordered collection of processes.
 - each process in the group is assigned an unique rank
 - a process may belong to several communicators and it will have a specific rank in each of these communicators
- a context: a system-defined object that uniquely identifies a communicator (a tag associated to the communicator).
 - two processes may exchange data only if the communicator used to send data is identical with the communicator used to receive data.

For example, in the configuration shown below we may consider a communicator *row1_comm* with members {p0,p1,p2} and a communicator *col1_comm* with members {p0,p3,p6}. The rank of the processes *p0*, *p3* and *p6* in the communicator *col1_comm* will be respectively, 0,1, and 2 !



In this configuration, process *p2* will have rank 2 in *row1_comm* and rank 0 in *col3_comm*.

The MPI functions used to create communicators are

- `MPI_Comm_group(comm, group)`
 - returns the group associated to the communicator *comm*
- `MPI_Group_incl(old_group, new_group_size, ranks_old_group[], new_group)`
 - creates a new group of processes *new_group* from an old group of processes *old_group*
 - the processes to be included are listed in the array *ranks_old_group*
 - Process 0 in *new_group* has rank *ranks_old_group*[0] in *old_group*, process 1 in *new_group* has rank *ranks_old_group*[1], so on.
- `MPI_Comm_create(old_comm, new_group, new_comm)`
 - associates a context with the group *new_group* and creates the communicator *new_comm*.
 - all the processes in the *new_group* belong to the *old_comm*
 - All processes in *old_comm*, including those not joining *new_comm* must call *MPI_Comm_create* with the same arguments.
- If several communicators are created they must be created in the same order on all processes.

Example: assume that `MPI_COMM_WORLD` consists of 9 processes and we want to create a communicator among processes 0, 3, and 6. This may be achieved with the instructions:

```
group_rank(0) = 0  
group_rank(1) = 3  
group_rank(2) = 6
```

```
CALL MPI_COMM_GROUP(MPI_COMM_WORLD, group_world, ierr)  
CALL MPI_GROUP_INCL(group_world, 3, group_rank, col1_group, ierr)  
CALL MPI_COMM_CREATE(MPI_COMM_WORLD, col1_group,  
                      col1_comm, ierr)
```

The new communicator is *col1_comm* and may be used to exchange data among the processes of rank 0,3, and 6 in the `MPI_COMM_WORLD`.

See [comm_create1.f](#)

MPI_Comm_split

In a practical application we may need to create several communicators, for example one for each row and one for each column of processes. This may be achieved using the MPI function

`MPI_Comm_split(old_comm, split_key, rank_key, new_comm)`

- creates a new communicator for each value of *split_key*
- processes with the same value of *split_key* form a new group
- the rank in the new group is determined by the value of *rank_key*
 - if process A and process B call `MPI_Comm_split` with the same value of *split_key*, their rank in *new_comm* is assigned according to the value of *rank_key*: if *rank_key* passed by process A is less than the *rank_key* passed by process B, then in *new_comm* the rank of A will be lower than the rank of B.
 - if the *rank_key* passed by A has the same value as the *rank_key* passed by B, then the system will arbitrarily assign one of the processes a lower rank.
- `MPI_Comm_split` must be called by all processes in *old_comm*

Example: create \sqrt{p} row and \sqrt{p} column communicators

C my_rank is rank in MPI_COMM_WORLD.

C $q * q = p$

C row communicators

```
my_row = my_rank/q
```

```
call MPI_COMM_SPLIT(MPI_COMM_WORLD, my_row, my_rank,  
& my_row_comm, ierr)
```

C column communicators

```
my_col = mod(my_rank,q)
```

```
call MPI_COMM_SPLIT(MPI_COMM_WORLD, my_col, my_rank,  
& my_col_comm, ierr)
```

See [comm_split1.f](#)

Topologies

A topology is a mechanism for associating different addressing schemes with the processes belonging to a group.

It may be used to associate a "virtual" square grid structure with `MPI_COMM_WORLD` by specifying the following information:

- the number of dimensions in the grid (2 for a square)
- the size of each dimension (\sqrt{p} for a squared grid)
- periodicity of each dimension: whether the first entry in each row or column is adjacent to the last entry in that row or column
 - may be useful for periodic boundary conditions !
- option to let the system optimize the mapping of the grid of processes to the physical processors by possibly reordering the processes in the communicator group.

MPI functions to create topologies

`MPI_Cart_create(old_comm, number_dims, dim_sizes[],
wrap_around[], reorder, cart_comm)`

- creates a new communicator *cart_comm*
- *dim_sizes[]* and *wrap_around[]* are arrays of dimension equal to *number_dims* (e.g., 2)
- a dimension *i* is circular if *wrap_around[i] = 1*; if *wrap_around[i] = 0* the dimension is linear
- processes in *comm_cart* are ranked in row-major order:
 - first row consists of processes 0, 1, ..., *dim_sizes[0] - 1*
 - second row consists of processes *dim_sizes[0]*, *dim_sizes[0] + 1*, ..., *2*dim_sizes[0]-1*; etc.
- `MPI_Cart_create` constructs a new communicator, it is a collective operation and must be called by all processes in the *old_comm*

MPI functions to create topologies

Address information functions:

- `MPI_Cart_rank(comm, coordinates[], rank)`
 - returns the rank in the communicator *comm* of the process with Cartesian coordinates *coordinates*
 - *coordinates* is an array with order equal to the order of the dimensions in the Cartesian topology associated with *comm* (e.g., order 2)
- `MPI_Cart_coords(comm, rank, number_dims, coordinates[])`
 - is the inverse to `MPI_Cart_rank`: it returns the coordinates of the process with rank *rank* in the Cartesian communicator *comm*.

See `topology.f`