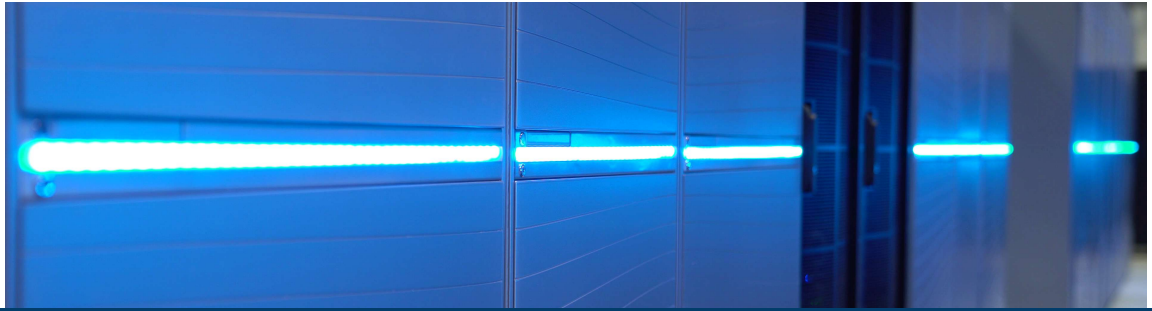


Part XI: Communicators

MOTIVATION

Communicators are a scope for communication within (*intra-communicators*) or between groups (*inter-communicators*) of processes. New communicators with different scope or topological properties can be used to accommodate certain needs.

- Separation of communication spaces: A software library that uses MPI underneath is used in an application that directly uses MPI itself. Communication due to the library should not conflict with communication due to the application.
- Partitioning of process groups: Parts of your software exhibit a collective communication pattern, but only across a subset of processes.
- Exploiting inherent topology: Your application uses a regular cartesian grid to discretize the problem and this translates into certain nearest neighbor communication patterns.

DUPLICATE [MPI-4.0, 7.4.2]

C

```
int MPI_Comm_dup(MPI_Comm comm, MPI_Comm *newcomm)
```

F08

```
MPI_Comm_dup(comm, newcomm, ierror)
type(MPI_Comm), intent(in) :: comm
type(MPI_Comm), intent(out) :: newcomm
integer, optional, intent(out) :: ierror
```

- Duplicates an existing communicator comm
- New communicator has the same properties but a new context

SPLIT [MPI-4.0, 7.4.2]

C

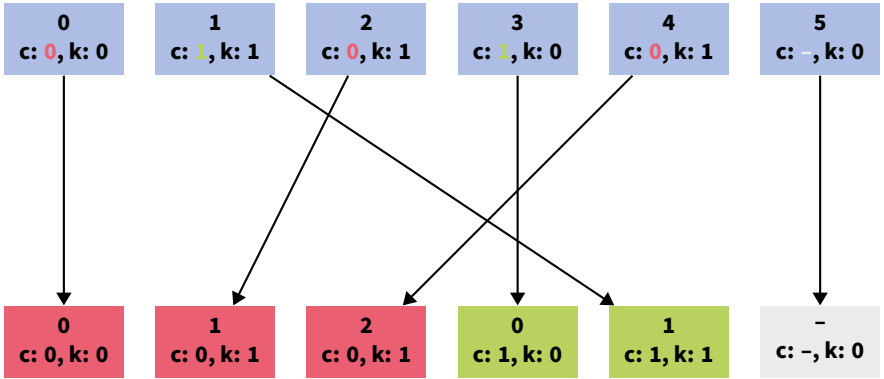
```c
int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)
```

F08

```f08
MPI_Comm_split(comm, color, key, newcomm, ierror)
type(MPI_Comm), intent(in) :: comm
integer, intent(in) :: color, key
type(MPI_Comm), intent(out) :: newcomm
integer, optional, intent(out) :: ierror
```

- Collective call, needs to be called by all processes in communicator comm
- Splits the processes in a communicator into disjoint subgroups
- Processes are grouped by `color`, one new communicator per distinct value
- Special color value `MPI_UNDEFINED` does not create a new communicator (`MPI_COMM_NULL` is returned in `newcomm`)
- Processes in new communicator are ordered by ascending value of `key`, for equal key values according to their rank in the old group

SPLIT [MPI-4.0, 7.4.2]



CREATE [MPI-4.0, 7.4.2]

C

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

F08

```
MPI_Comm_create(comm, group, newcomm, ierror)
type(MPI_Comm), intent(out) :: comm
type(MPI_Group), intent(out) :: group
type(MPI_Comm), intent(out) :: newcomm
integer, optional, intent(out) :: ierror
```

- Collective call, needs to be called by all processes in communicator comm
- Takes as argument handle to a subgroup of the group associated with communicator comm

GROUPS [MPI-4.0, 7.3.2]

C
```
int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)
```

C
```
int MPI_Group_incl(MPI_Group group, int n, const int ranks[], MPI_Group
 ↪  *newgroup)
```

- process with rank i in newgroup is the process with rank ranks[i] in group
- elements of ranks must be a valid rank in group and all elements must be distinct

C
```
int MPI_Group_free(MPI_Group *group)
```

CARTESIAN TOPOLOGY [MPI-4.0, 8.5.1]

C

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, const int dims[], const
 ↪  int periods[], int reorder, MPI_Comm *comm_cart)
```

F08

```
MPI_Cart_create(comm_old, ndims, dims, periods, reorder, comm_cart, ierror)
type(MPI_Comm), intent(in) :: comm_old
integer, intent(in) :: ndims, dims(ndims)
logical, intent(in) :: periods(ndims), reorder
type(MPI_Comm), intent(out) :: comm_cart
integer, optional, intent(out) :: ierror
```

- Creates a new communicator with processes arranged on a (possibly periodic) Cartesian grid
- The grid has `ndims` dimensions and `dims[i]` points in dimension `i`
- If `reorder` is true, MPI is free to assign new ranks to processes

CARTESIAN TOPOLOGY [MPI-4.0, 8.5.1]

Input:

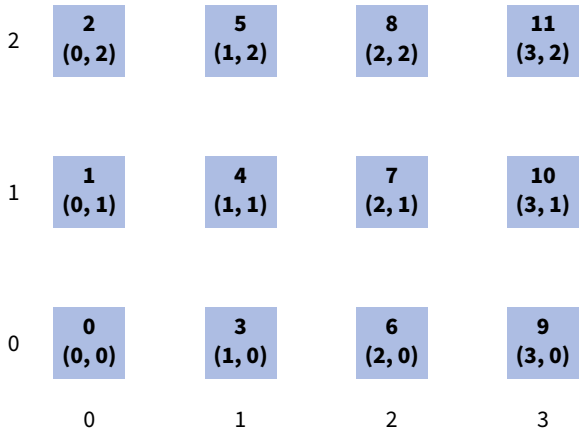
`comm_old` contains 12 processes (or more)

```
ndims = 2, dims = [ 4, 3 ],
periods = [ .false., .false. ]
reorder = .false.
```

Output:

process 0–11: new communicator with topology as shown

process 12–: `MPI_COMM_NULL`



JÜLICH Forschungszentrum

RANK TO COORDINATE [MPI-4.0, 8.5.5]

C

```c
int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int coords[])
```

F08

```f08
MPI_Cart_coords(comm, rank, maxdims, coords, ierror)
type(MPI_Comm), intent(in) :: comm
integer, intent(in) :: rank, maxdims
integer, intent(out) :: coords(maxdims)
integer, optional, intent(out) :: ierror
```

Translates the rank of a process into its coordinate on the Cartesian grid.

COORDINATE TO RANK [MPI-4.0, 8.5.5]

C

```c
int MPI_Cart_rank(MPI_Comm comm, const int coords[], int *rank)
```

F08

```f08
MPI_Cart_rank(comm, coords, rank, ierror)
type(MPI_Comm), intent(in) :: comm
integer, intent(in) :: coords(*)
integer, intent(out) :: rank
integer, optional, intent(out) :: ierror
```

Translates the coordinate on the Cartesian grid of a process into its rank.

CARTESIAN SHIFT [MPI-4.0, 8.5.6]

C

```
int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int
 ↪  *rank_source, int *rank_dest)
```

F08

```
MPI_Cart_shift(comm, direction, disp, rank_source, rank_dest, ierror)
type(MPI_Comm), intent(in) :: comm
integer, intent(in) :: direction, disp
integer, intent(out) :: rank_source, rank_dest
integer, optional, intent(out) :: ierror
```

- Calculates the ranks of source and destination processes in a shift operation on a Cartesian grid
- `direction` gives the number of the axis (starting at 0)
- `disp` gives the displacement

CARTESIAN SHIFT [MPI-4.0, 8.5.6]

Input:

`direction = 0`, `disp = 1`, not periodic

Output:

process 0:

```
rank_source = MPI_PROC_NULL,
rank_dest = 3
```

…

process 3:

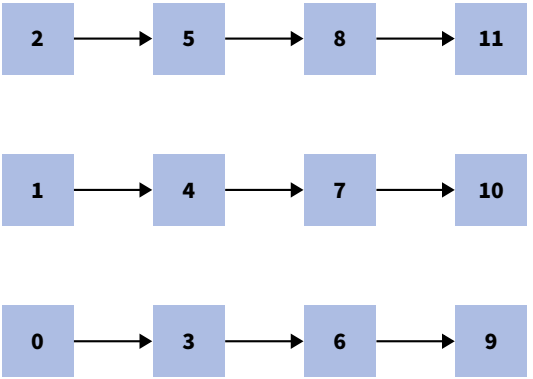
```
rank_source = 0,
rank_dest = 6
```

…

process 9:

```
rank_source = 6,
rank_dest = MPI_PROC_NULL
```

…



CARTESIAN SHIFT [MPI-4.0, 8.5.6]

Input:

`direction = 0, disp = 1, periodic`

Output:

process 0:

```
rank_source = 9,
rank_dest = 3
```

…

process 3:

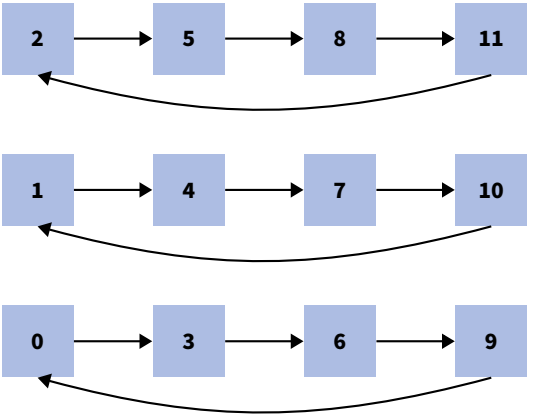
```
rank_source = 0,
rank_dest = 6
```

…

process 9:

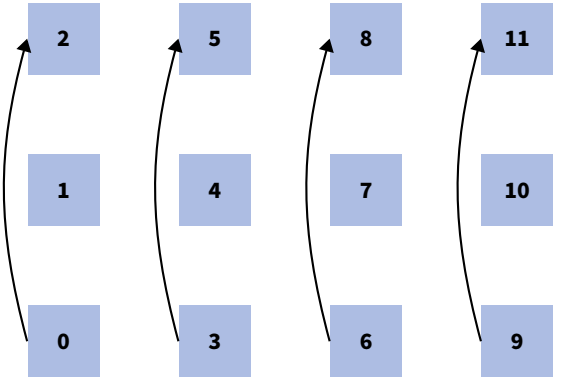
```
rank_source = 6,
rank_dest = 0
```

…



CARTESIAN SHIFT [MPI-4.0, 8.5.6]

Input:
`direction = 1, disp = 2`, not periodic
Output:
process 0:
`rank_source = MPI_PROC_NULL,`
`rank_dest = 2`
process 1:
`rank_source = MPI_PROC_NULL,`
`rank_dest = MPI_PROC_NULL`
process 2:
`rank_source = 0,`
`rank_dest = MPI_PROC_NULL`
…



NULL PROCESSES [MPI-4.0, 3.10]

C
```c
int MPI_PROC_NULL = /* implementation defined */
```

F08
```f
integer, parameter :: MPI_PROC_NULL = ! implementation defined
```

- Can be used as source or destination for point-to-point communication
- Communication with `MPI_PROC_NULL` has no effect
- May simplify code structure (communication with special source/destination instead of branch)
- `MPI_Cart_shift` returns `MPI_PROC_NULL` for out of range shifts

COMPARISON [MPI-4.0, 7.4.1]

C
```c
int MPI_Comm_compare(MPI_Comm comm1, MPI_Comm comm2, int *result)
```

F08
```f08
MPI_Comm_compare(comm1, comm2, result, ierror)
type(MPI_Comm), intent(in) :: comm1, comm2
integer, intent(out) :: result
integer, optional, intent(out) :: ierror
```

Compares two communicators. The result is one of:

MPI_IDENT	The two communicators are the same.
MPI_CONGRUENT	The two communicators consist of the same processes in the same order but communicate in different contexts.
MPI_SIMILAR	The two communicators consist of the same processes in a different order.
MPI_UNEQUAL	Otherwise.

FREE [MPI-4.0, 7.4.3]

C

```c
int MPI_Comm_free(MPI_Comm *comm)
```

F08

```f08
MPI_Comm_free(comm, ierror)
type(MPI_Comm), intent(inout) :: comm
integer, optional, intent(out) :: ierror
```

Marks a communicator for deallocation.