Groups and Communicators

SoHPC, 2021

Acknowledgments

- This course is based on the MPI course developed by Rolf Rabenseifner at the High-Performance Computing-Center Stuttgart (HLRS), University of Stuttgart in collaboration with the EPCC Training and Education Centre, Edinburgh Parallel Computing Centre, University of Edinburgh.

Outline

- MPI Groups
- Communicators from Groups
- Virtual Topologies
- Create Cartesian Topology
- Neighbours
- Sub-communicators

Motivations

- Need to create sets of processes
  - For multiple programs
  - Make use of collectives routines
- Need to map the abstract topology onto the natural topology of the problem domain
  - For programming convenience
  - For performance

Groups

- We can create sub-groups of the MPI processes.
- A process has an unique ID/rank for each group.
- MPI_Group_rank, MPI_Group_size
- A new group must be created from another group.
- Initially there are no predefined groups.

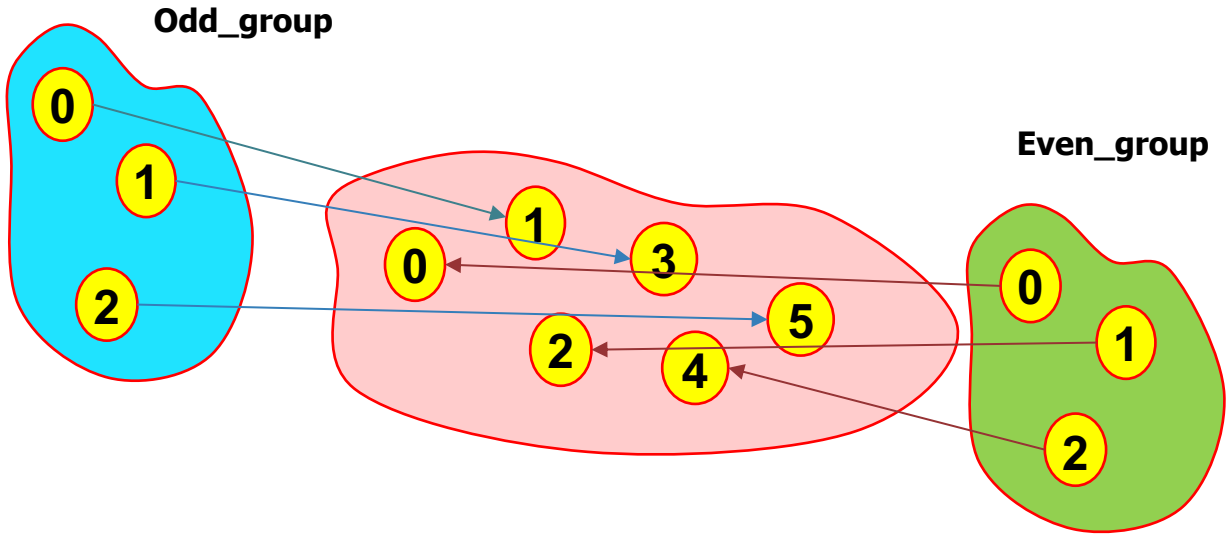
Split into Odd and Even

- Odd_ranks={1, 3, 5}, Even_ranks={0, 2, 4}
- Create groups
  - MPI_comm_group(MPI_COMM_WORLD,Group_all)
  - MPI_Group_incl(Group_all, 3, Odd_ranks, &Odd_group)
  - MPI_Group_incl(Group_all, 3, Even_ranks, &Even_group)
- Groups do not allow communication between its members.

Communicator from a Group

- Create communicator
  - int MPI_Comm_create(MPI_COMM_WORLD, Odd_group, Odd_Comm)
  - int MPI_Comm_create(MPI_COMM_WORLD, Even_group, Even_Comm)
- Alternatively…
  - color = modulo(myrank, 2)
  - MPI_Comm_split(MPI_COMM_WORLD, color, key, &newcomm)

Working with groups



Group Management

- Group Accessors
  - MPI_Group_size(…)
  - MPI_Group_rank(…)
  - …
- Group Constructors
  - MPI_Comm_group(…)
  - MPI_Group_incl(…)
  - MPI_GROUP_EXCL(…)
  - …
- Group Destructors
  - MPI_Group_free(group)

Communicator Management

- Communicator Accessors
  - MPI_Comm_size(…)
  - MPI_Comm_rank(…)
  - …
- Communicator Constructors
  - MPI_Comm_create(…)
  - MPI_Comm_split(…)
- Communicator Destructors
  - MPI_Comm_free(comm)

Virtual Topologies

- Matches arrangement of MPI processes with data.
- Convenient process naming.
- Simplifies writing of code.
- Can allow MPI to optimize communications.

Topology Types

- Cartesian Topologies
  - each process is connected to its neighbor in a virtual grid,
  - boundaries can be cyclic, or not,
  - processes are identified by Cartesian coordinates,
  - of course, communication between any two processes is still allowed.

- Graph Topologies
  - general graphs,
  - not covered here.

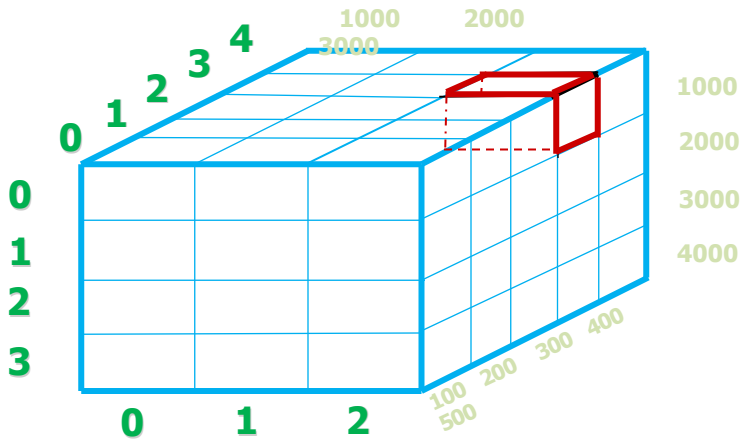
Example

- Global array A(1:3000, 1:4000, 1:500) = $6 \bullet 10^9$ elements
  on **3 x 4 x 5** **= 60 processors**
  process coordinates **0..2, 0..3, 0..4**

- example:
  on process **ic_0=2, ic_1=0, ic_2=3 (rank=43)**
  decomposition, A(2001:3000, 1:1000, 301:400) = $0.1 \bullet 10^9$ elements

- **process coordinates:** handled with **virtual Cartesian topologies**
- Array decomposition: handled by the application program directly

Graphical representation



- Distribution of processes over the grid
- Distribution of the Global Array
- Coordinate (2, 0, 3) represents process number 43
- It is being assigned the cube A(2001:3000, 1:1000, 301:400)

New Communicator

- Creating a topology produces a new communicator.
- MPI provides mapping functions:
  - between ranks based on MPI_Comm_rank
  - And those based on the topology.

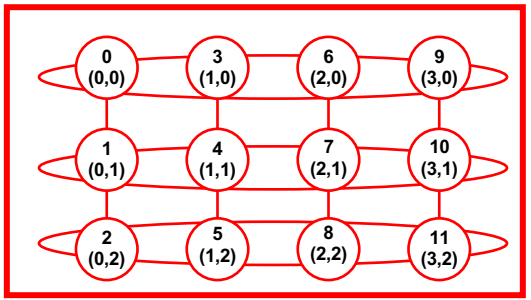
2D Example

- C: int MPI_Cart_create(MPI_Comm comm_old, int ndims,
  int *dims, int *periods, int reorder,
  MPI_Comm *comm_cart)

- Fortran: MPI_Cart_create(comm_old, ndims, dims, periods,
  reorder, comm_cart, ierror)

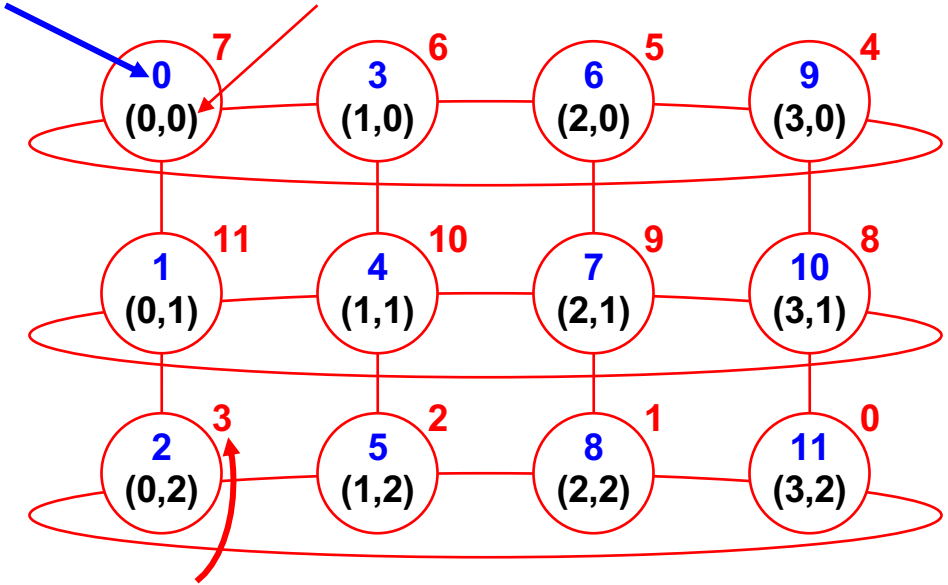
  integer :: comm_old, ndims, dims(:)
  logical :: periods(:), reorder
  integer :: comm_cart, ierror

```
comm_old  =  MPI_COMM_WORLD
   ndims  =  2
    dims  = ( 4,          3          )
 periods  = ( 1/.true.,   0/.false.  )
 reorder  =  see next slide
```



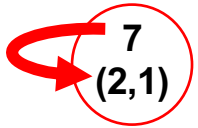
Using Reorder

- Ranks in comm and comm_cart may differ, if reorder = 1 or .TRUE.
- This reordering can allow MPI to optimize communications



Cartesian Mapping Functions

- Mapping ranks to process grid coordinates



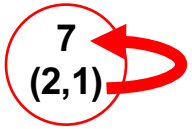
- C: int MPI_Cart_coords(MPI_Comm comm_cart,
  int rank, int maxdims, int **coords*)

- Fortran: MPI_Cart_coords(comm_cart, rank,
  maxdims, *coords*, *ierror*)

```
integer ::      comm_cart, rank
integer ::      maxdims, coords(*), ierror
```

Cartesian Mapping Functions

- Mapping process grid coordinates to ranks



- C: int MPI_Cart_rank(MPI_Comm comm_cart,
  int *coords, int **rank*)

- Fortran: MPI_Cart_rank(comm_cart,
  coords, *rank*, *ierror*)

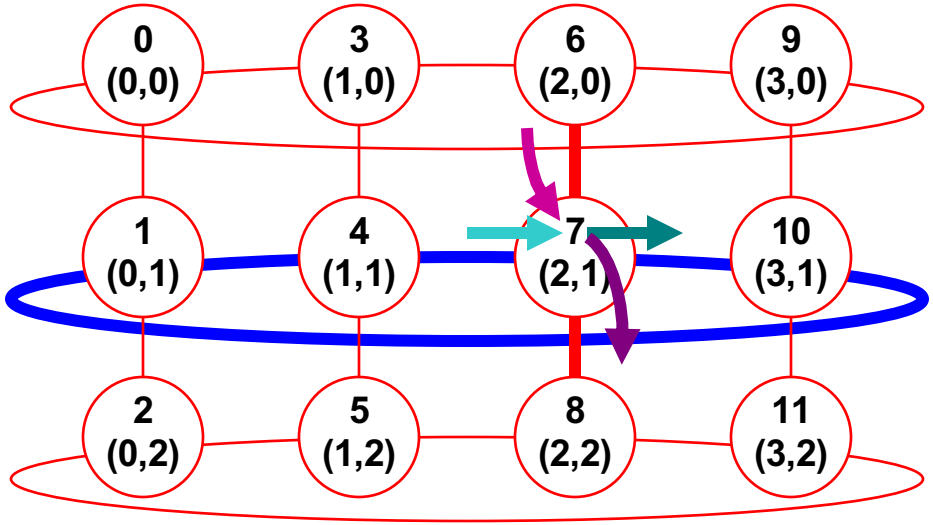
  integer :: comm_cart, coords(:)
  integer :: rank, ierror

Get Neighbours

- Computing ranks of neighboring processes
- C: int MPI_Cart_shift(MPI_Comm comm_cart, int direction, int disp,
  int *_rank_prev_, int *_rank_next_)
- Fortran: MPI_Cart_shift(comm_cart, direction, disp,
  _rank_prev_, _rank_next_, _ierror_)

  integer :: comm_cart, direction
  integer :: disp, rank_source
  integer :: rank_dest, ierror
- Returns MPI_PROC_NULL if there is no neighbour.
- MPI_PROC_NULL can be used as source or destination rank in each communication then, this communication will be a noop!

MPI_Cart_shift – Example



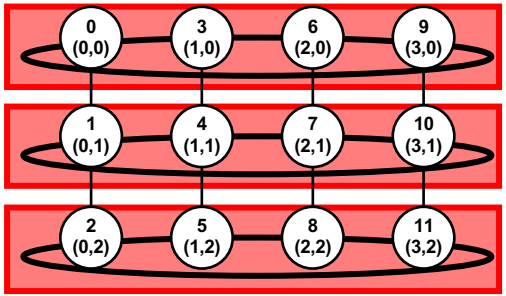
invisible input argument: **my_rank** in cart

MPI_Cart_shift(cart, direction, displace, &*rank_prev*, &*rank_next)*
MPI_Cart_shift(cart, direction, displace, *rank_prev*, *rank_next*, *ierror*)

example on	**0**	**+1**	**4**	**10**
process rank=**7**	**1**	**+1**	**6**	**8**

Cartesian Partitioning

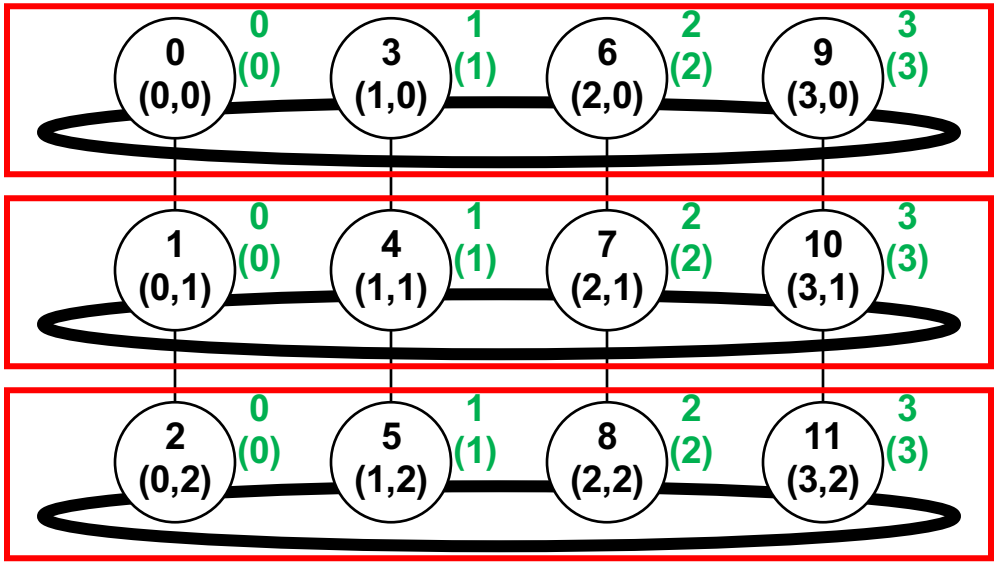
- Cut a grid up into *slices*.
- A new communicator is produced for each slice.
- Each slice can then perform its own collective communications.
- C: int MPI_Cart_sub(MPI_Comm comm_cart, int *remain_dims, MPI_Comm *comm_slice)
- Fortran: MPI_Cart_sub(comm_cart, remain_dims, comm_slice, ierror)




```
integer             :: comm_cart
logical :: remain_dims(*)
integer             :: comm_slice, ierror
```

MPI_Cart_sub – Example

- Ranks and Cartesian process coordinates in comm_sub



- MPI_Cart_sub(comm_cart, remain_dims, ***comm_sub***)
- MPI_Cart_sub(comm_cart, remain_dims, ***comm_sub***, *ierror*)

(true, false)

Summary

- We can create new communicators from old ones.
- We can create groups of processes or use colour.
- Virtual topologies create a imaginary grid of ranks in any dimension order (2D, 3D etc).
- This mainly offers a convenient way to code, based on the data structure.