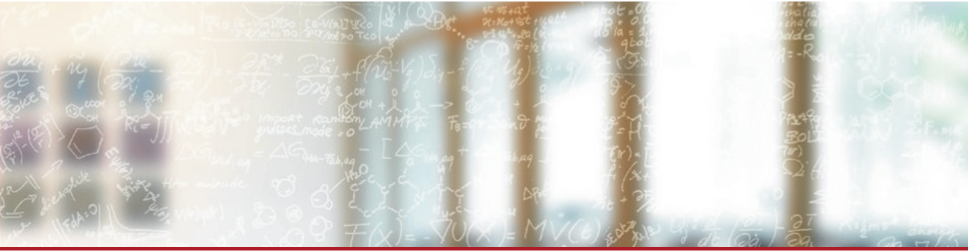
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Message Passing Interface (MPI)

Summer School 2017 – Effective High Performance Computing
Tim Robinson, CSCS
July 19–20, 2017

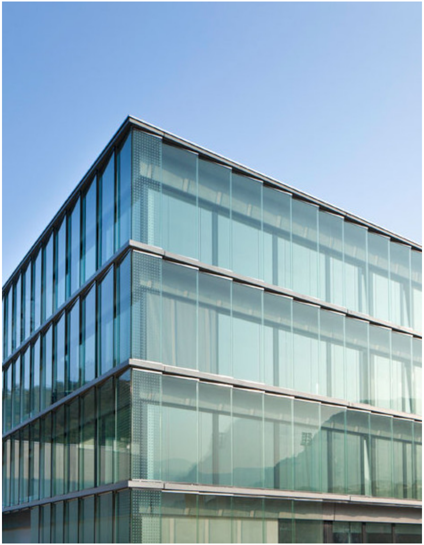
Previous course summary

- Point-to-point communication, Blocking and non-blocking
- Collective operations
- Derived datatypes

Course Objectives

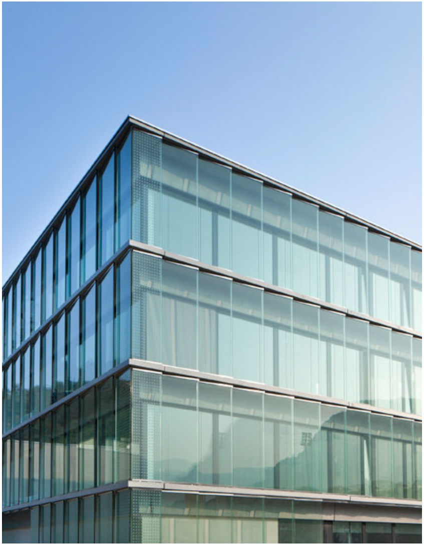
- The understanding of a topology and communicators
- How to build and use a topology

General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Datatypes
- Topology

General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Datatypes
- Topology
  - Groups and communicator
  - Topology with MPI
  - Domain decomposition
  - Cartesian topology
  - Graph topology

Topology

Groups and communicator

- A group is an ordered set of processes, each with a unique integer rank. In MPI, a group is represented within system memory as an object. It is accessible to the programmer only by a ”handle”. A group is always associated with a communicator object.
- A communicator encompasses a group of processes that may communicate with each other. All MPI messages must specify a communicator. Like groups, communicators are accessible to the programmer only by ”handles”. The handle for the communicator that comprises all processes is `MPI_COMM_WORLD`.

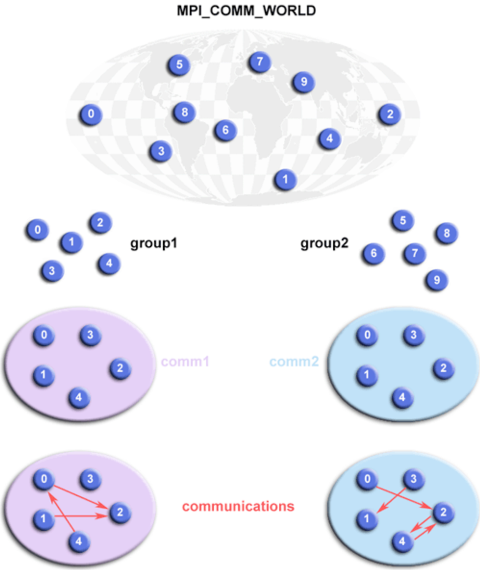
From the programmer’s perspective, a group and a communicator are one. The group routines are primarily used to specify which processes should be used to construct a communicator.

Groups and communicator

Goals:

- Allow you to organize tasks, based upon function, into task groups.
- Enable Collective Communications operations across a subset of related tasks.
- Provide basis for implementing user defined virtual topology.

Remarks: Groups/communicators can be created and destroyed during program execution. Processes may be in more than one group/communicator having a unique rank within each group/communicator.



Defining a new communicator: MPI_COMM_SPLIT

Pseudo-code

```
MPI_Comm_split(comm, color, key, comm_out)
```

color identifies the group

key specifies a member of the group (rank)

6 ranks

P0	P1
P2	P3
P4	P5

row_comm

P0	P1
P2	P3
P4	P5

col_comm

P0	P1
P2	P3
P4	P5

myRank	0	1	2	3	4	5
iRow	0	0	1	1	2	2
jCol	0	1	0	1	0	1

Fortran

```fortran
! logical 2D topology with nrow=3 rows and mcol=2 columns
! 6 ranks, it is a collective operation
iRow = myRank/mcol         !! logical row number
jCol = mod(myRank, mcol) !! logical column number

Call MPI_COMM_SPLIT(MPI_COMM_WORLD, iRow, jCol, row_comm,
    ierr)
Call MPI_COMM_SPLIT(MPI_COMM_WORLD, jCol, iRow, col_comm,
    ierr)
```

Topology with MPI

- A virtual topology describes the ”connectivity” of MPI processes in a communicator.
- The two main types of topology are Cartesian and Graph.
- MPI topology are virtual - there may be no relation between the physical structure of the parallel machine and the process topology.
- Virtual topology are built upon MPI communicators and groups.

Cartesian topology

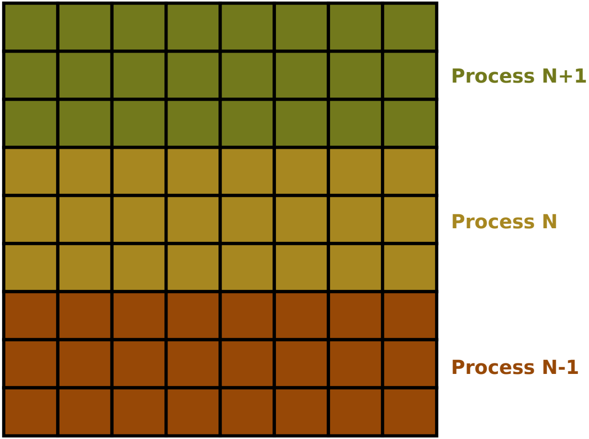
- Each process is “connected” to its neighbours in a virtual grid
- Boundaries can be cyclic
- Identified by Cartesian coordinates (i, j, k)

Graph topology

- Graphs are used to describe communication patterns
- The most general description of communication patterns

Domain decomposition

Planar distribution: data are distributed linearly between processors. Default mapping when using `MPI_COMM_WORLD`.

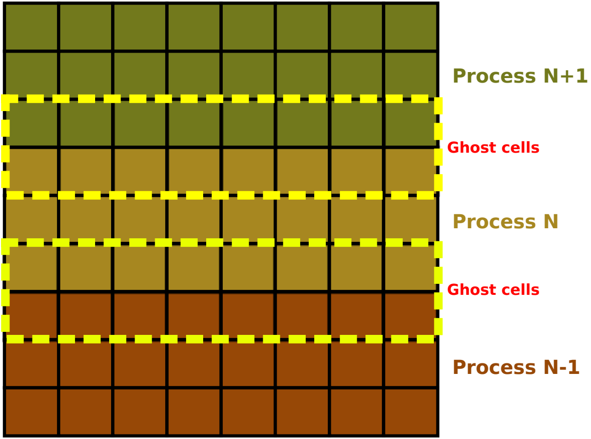


Domain decomposition

Planar distribution: data are distributed linearly between processors.

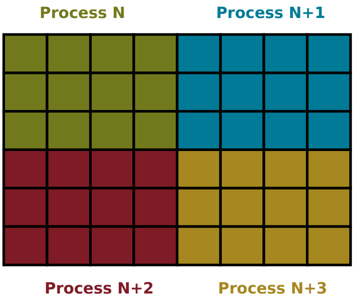
Default mapping when using `MPI_COMM_WORLD`.

Ghost cells are exchanged: processor N communicates with N-1 and N+1



Domain decomposition

Cartesian distribution: data are distributed linearly between processors.



This is in general a more effective way of distribute the domain, since:

- It is much more scalable
- Communicated data volume can be smaller (especially when a large number of processors is used)
- It can better map the geometry of the problem and of the algorithm

However, it is more difficult to handle: who are my neighbours?

Cartesian topology

Pseudo-code

```
MPI_Cart_create(comm_old, ndims, dims, periods, reorder,
                comm_cart)
```

comm_old input communicator

ndims number of dimensions of Cartesian grid

dims specifies the number of processes in each dimension

periods specifies 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

Row-major numbering:

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

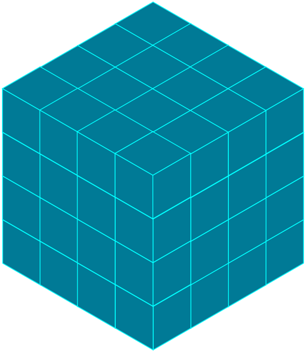
Cartesian topology example

Fortran

```fortran
INTEGER :: comm_cart
INTEGER :: ierr
INTEGER :: dims(3)
LOGICAL :: periods(3)

dims(1) = NprocX
dims(2) = NprocY
dims(3) = NprocZ
periods = .TRUE.

Call MPI_CART_CREATE(MPI_COMM_WORLD, 3,
     dims, periods, .TRUE., comm_cart,
    ierr)
```



Periods and reorder

Periods: Define if the boundary of the grid are periodic or not.

periods = False					
-1	←0	1	2	3→	-1
periods = True					
3	←0	1	2	3→	0

Note: `MPI_PROC_NULL=-1`

Reorder: allows MPI processes reordered for efficiency, possibly so as to choose a good embedding of the virtual topology onto the physical machine.

Utility functions

Create dimensions:

Pseudo-code

```
MPI_Dims_create(nnodes, ndims, dims)
```

Retrieves Cartesian topology information associated with a communicator

Pseudo-code

```
MPI_Cartdim_get(comm, ndims)
MPI_Cart_get(comm, ndims, dims, periods, coords)
```

Coordinates to rank:

Pseudo-code

```
MPI_Cart_rank(comm, coords, rank)
```

Rank to coordinates:

Pseudo-code

```
MPI_Cart_coords(comm, rank, maxdims, coords)
```

Finding neighbours: Shift

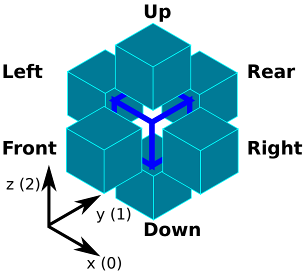
Pseudo-code

```
MPI_Cart_shift(comm, direction, disp,
               rank1, rank2)
```

comm	communicator with Cartesian structure
direction	coordinate dimension of shift
disp	displacement
rank1	rank of nearby process
rank2	rank of nearby process

Fortran

```
Call MPI_CART_SHIFT(comm_cart, 0, 1,
                    left, right, ierr)
Call MPI_CART_SHIFT(comm_cart, 1, 1,
                    front, rear, ierr)
Call MPI_CART_SHIFT(comm_cart, 2, 1,
                    down, up, ierr)
```



Sub-grids in Cartesian topology

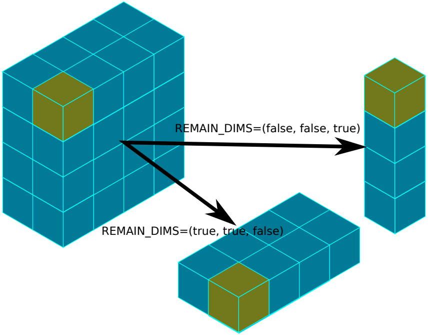
Pseudo-code

```
MPI_Cart_sub(comm, remain_dims,
             newcomm)
```

comm communicator with Cartesian structure

remain_dims the ith entry of remain_dims specifies whether the ith dimension is kept in the subgrid (true) or is dropped (false)

newcom communicator containing the subgrid including the calling process


REMAIN_DIMS=(false, false, true)
REMAIN_DIMS=(true, true, false)

Graph topology

Pseudo-code

```
MPI_Graph_create(comm_old, nnodes, index, edges, reorder,
                 comm_graph)
```

comm_old input communicator

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)

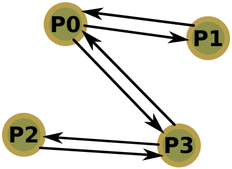
comm_graph communicator with graph topology added

Process	Neighbors
0	1,3
1	0
2	3
3	0,2

nnodes = 4

index = 2, 3, 4, 6

edges = 1, 3, 0, 3, 0, 2



Other functions

- Manage communicators:

  `MPI_Comm_compare, MPI_Comm_dup` . . .

- Manipulate groups:

  `MPI_Group_union, MPI_Group_intersection` . . .

- Cartesian topology, map a process:

  `MPI_Cart_map`

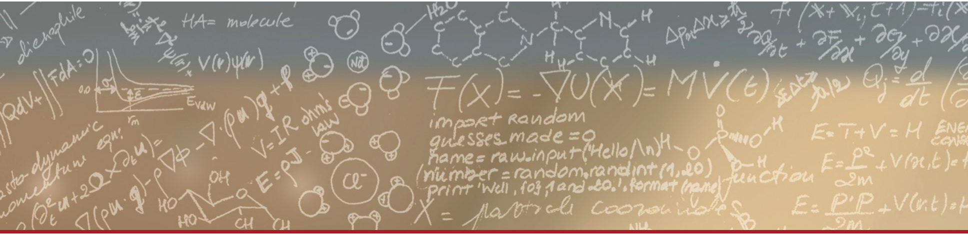
- Graph topology:

  `MPI_Graph_map, MPI_Graph_get` . . .

Practicals

Exercise: 05.MPI_Topo

1. Create a 1-dimension topology - a ring
2. Ghost cell exchanges using a topology



Thank you for your attention.