Back (briefly) to MPI: more on communicators / topology

- Quick overview of communicators in MPI
- Groups
- Virtual topologies
- Cartesian Mappings
- Example: back to the 2-D matrix product

MPI communication groups

- MPI supports grouping of processes
- Can be used to organize application around tasks
- Collective communications operations can be performed within a group.
- Can create virtual communication topologies: Each group has its own communicator
- Groups and communicators are MPI objects

Main MPI Group functions

- *MPI_Group_size:* returns number of processes in group
- *MPI_Group_rank:* returns rank of calling process in group
- *MPI_Group_compare:* compares group members and group order
- *MPI_Group_translate ranks:* translates ranks of processes in one group to those in another group
- *MPI_Comm_ group:* returns the group associated with a communicator
- *MPI_Group_union:* creates a group by combining two groups
- *MPI_Group_intersection:* creates a group from the intersection of two groups

- *MPI_Group_difference:* creates a group from the difference between two groups
- *MPI_Group_incl:* creates a group from listed members of an existing group
- *MPI_Group_excl:* creates a group excluding listed members of an existing group
- *MPI_Group_range incl:* creates a group according to first rank, stride, last rank
- *MPI_Group_range excl:* creates a group by deleting according to first rank, stride, last rank
- *MPI_Group_free:* marks a group for deallocation

Creating a group: Example from P. Patcheco's book

- Declarations etc.

```
  main(int argc, char* argv[]) {
    int          p;
    int          q; /* = sqrt(p) */
    int          my_rank;
    MPI_Group    group_world;
    MPI_Group    row_group;
    MPI_Comm     row_comm;
    int*         process_ranks;
    int          proc;
    int          test = 0;
    int          sum;
    int          my_rank_in_row;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
```

➤ Setting up communicator

```
    ...
    q = (int) sqrt((double) p);
/*--- List of processes in new communicator */
    process_ranks = (int*) malloc(q*sizeof(int));
    for (proc = 0; proc < q; proc++)
        process_ranks[proc] = proc;
/*--- Get group underlying MPI_COMM_WORLD */
    MPI_Comm_group(MPI_COMM_WORLD, &group_world);
/*--- Create the new group */
    MPI_Group_incl(group_world, q, process_ranks,
        &row_group);
/*--- Create the new communicator */
    MPI_Comm_create(MPI_COMM_WORLD,row_group,
         &row_comm);
```

➤ Testing

```
/*--- Test collective ops in row_comm */
    if (my_rank < q) {
      MPI_Comm_rank(row_comm, &my_rank_in_row);
      if (my_rank_in_row == 0) test = 1;
      MPI_Bcast(&test, 1, MPI_INT, 0, row_comm);
    }
    MPI_Reduce(&test, &sum, 1, MPI_INT, MPI_SUM, 0,
        MPI_COMM_WORLD);
    if (my_rank == 0) {
      printf("q = %d, sum = %d\n", q, sum);
    }
    MPI_Finalize();
  }
```

Explore: MPI_Comm_split

```
#include <mpi.h>
int MPI_Comm_split(MPI_Comm comm, int color,
                   int key, MPI_Comm *newcomm)
```

- This function allows to partition a group associated with `comm` into disjoint subgroups, one for each value of color.
- Each subgroup contains processes of the same color.
- A new communicator is created for each subgroup and returned in newcomm.
- We will see something similar when considering cartesian grids

✍1 Explore this [MPI site is enough]

✍2 See how you can use it to redo previous example (2 colors)

MPI Virtual topologies

- Relevant to Lab3.
- MPI allows to create 'virtual' topologies [e.g. a 3-D grid of processors]
- These topologies are virtual: no relation between architectire of the parallel machine and the process topology.
- A virtual topology is built from MPI communicators and groups
- Two types of topologies: *Cartesian (1-D, 2-D, 3-D), Graph*
- Uses: applications with specific communication patterns [Lab 3]
- Also: MPI may reduce communication costs by optimizing process mapping

MPI Cartesian topologies

➤ A few of the commands used for working with MPI Cartesian topologies:

- *MPI_Dims_create:* Create N-Dimensional arrangement of PEs in cartesian grid.
- *MPI_Cart_create:* Create N-Dimensional virtual topology/cartesian grid.
- *MPI_Cart_coords:* Get local PE coordinates in new cartesian grid
- *MPI_Cart_sub:* Partitions a communicator into subgroups which form lower-dimensional cartesian subgrids.
- *MPI_Cart_shift:* Used to find processor neighbors. Returns the shifted source and destination ranks, given a shift direction and amount.

Illustration (examles from Lab3)

```
#include <mpi.h>
int MPI_Cart_create(MPI_Comm comm_old, int ndims,
    const int dims[], const int periods[],
    int reorder, MPI_Comm *comm_cart)
```

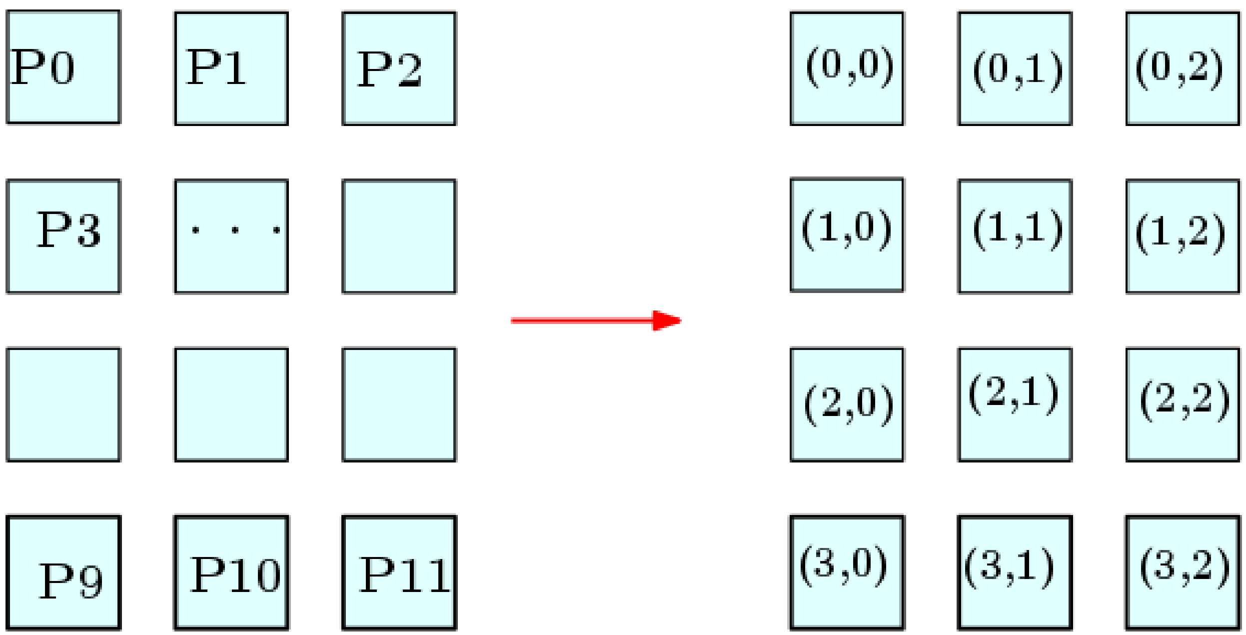
"Makes a new communicator to which Cartesian topology information has been attached."

Parameter	**i/o**	**Definition**
comm_old:	in	input communicator (handle)
ndims:	in	number of dimensions of cartesian grid
dims:	in	integer array of size ndims specifying the number of processes in each dim.
periods:	in	logical array of size ndims specifying for each dimens
reorder:	in	ranking may be reordered (true) or not (false) (logical)
comm_cart:	out	communicator with new cartesian topology (handle)

- One note: *reorder* tells whether or not to allow reordering of processes so to achieve a good embedding with respect to physical architecture.

- PEs are ordered from (0:np-1); numbered/ranked in row-major (C lang) order.

Create a (4,3) grid from a 12 process group:



- In Lab3:

- *MPI_Comm comm2D* declares a communicator named comm2D.

- We then create a cartesian communicator with a $qi \times qj$ grid (2-D) topology:

```
/*---------------------- CREATE 2-D communicator*/
dims[0] = qi;
dims[1] = qj;
periods[0]=periods[1]=1;
/*----------------------2D topoology created*/
MPI_Cart_create(MPI_COMM_WORLD,2,dims,periods,0,&
  comm2D);
/*---------------------- my rank in 2-D form */
MPI_Cart_coords(comm2D,myid,2,coords);
```

```
/*-------------------- rank of West neighbor */
coords[0]=qi;
coords[1]=qj-1;
MPI_Cart_rank(comm2D,coords,&westNB);
```

- Finds the rank of neighbor tp the west. westNB can now be used to commicate with node $(qi, qj - 1)$ in grid.
- This is very convenient as it simplifies communication.

An important function MPI_Cart_sub

```
#include <mpi.h>
int MPI_Cart_sub(MPI_Comm comm,
  const int remain_dims[],
  MPI_Comm *comm_new)
```

➤ Partitions a communicator into subgroups which form lower-dimensional cartesian subgrids

➤ remain_dims: This is a logical vector the ith entry of which specifies whether the ith dimension is kept in the subgrid (true) or is dropped (false)

➤ comm_new: (out) communicator containing the subgrid that includes the calling process (handle)

➤ Recall *MPI_Comm_split* - this is similar.

Show how you can use to implement the 2-D matrix multiply [page 12-17 of Lecture notes set 12]