Wednesday, 02/01/12:

Ch. 7 Communicators and Topologies:

Communicator:

MPI_COMM_WORLD is the default communicator. It is the universe of all MPI processes and predefined in any MPI job.

You can subdivide communicators into groups of processes. These are called intra-communicators, since they are within one original communicator and they can communicate with each other. Contrast: Inter-communicators refers to several communicators that exist simultaneously and may not be be part of one original communicator → MPI 2 = newer version of MPI that is designed to do things that MPI 1 could not, e.g., add a new process or combine jobs that each started with their own communicator.

Now: – example of Fox's algorithm → Sec. 7.2, 7.8
– Learn how to subdivide communicators
– how to subdivide and arrange in a topology

Pacheco example in Ch. 7 considers, say, 16 MPI processes that you think of as arranged in a Cartesian grid.

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

Notice that we really just have 16 processes that are running on one or more nodes in the cluster. We merely think of them as arranged in a grid. Want for example: one communicator for each row ⇒ So, need to create 4 communicators.

From here on, let's think more generally of q^2 processes, that is, assume that p = number of MPI processes, is a square and that $p = q^2$.

Option! = most general MPI command:

Idea: Specify a list (in an integer array) of id numbers of those processes to be included in the new communicator.

First, we need a way to interact with the existing communicator.

MPI_Comm variable like MPI_COMM_WORLD does not let you do that, so need to get a MPI_Group variable for it:

```
MPI_Group   group_world
MPI_Comm_group (MPI_COMM_WORLD, &group_world
```

Create one communicator for second row only.

-> Need to list id numbers $q, q+1, \ldots, 2q-1$

(see picture for 16 processes above)

```
int count = q
int process_ranks[q]
for (k=0; k< q ; k++)
        process_ranks[k] = q + k
MPI_Comm_incl ( group_world, count,
                process_ranks, &row_comm2)
MPI_Comm_create (&row_comm2 )
```

Notice: There are collective calls = all processes need to execute them!

Clearly: For purposes of creating one communicator for <u>each</u> row, this is not the best approach.

Option 2 = split communicator by some criterion based on id number.

```
MPI_Comm_split (
    MPI_Comm    old_comm,
    int         splitkey,
    int         rank_key,
    MPI_Comm    *new_comm)
```

All process in old_comm that have the same value of int split_key are put in one communicator. The process in the new_comm will be ordered as indicated by rank_key. Notice that the number of communicators created is not explicitly controlled.

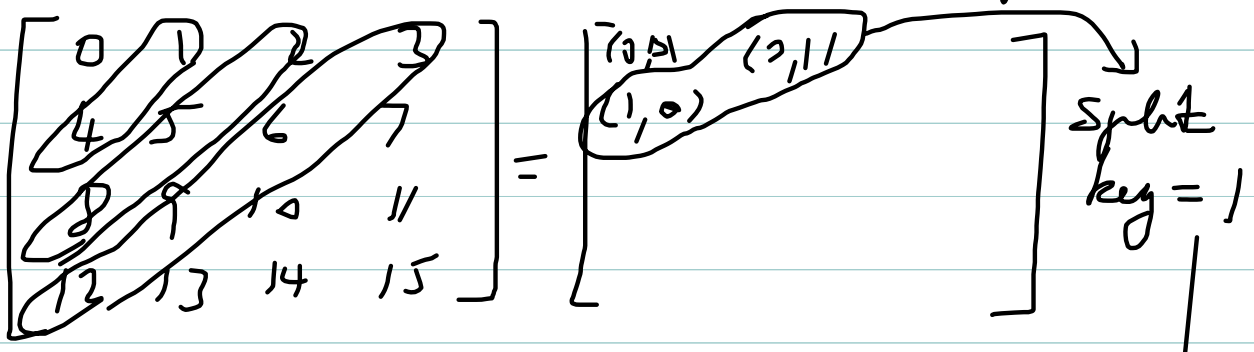
Given: own id number id, total number np

```
q = (int) sqrt ((double) np)
if (np != (q*q)) {
    if (id == 0)
        fprintf (stderr, "np should be square!")
    MPI_Abort (MPI_COMM_WORLD, 1)
}
```

```
my_row = id / q      /* integer division */
my_col = id % q      /* integer remainder */
MPI_Comm_split (MPI_COMM_WORLD,
                my_row, my_col, &row_comm)
MPI_Comm_split ( MPI_COMM_WORLD,
                my_col, my_row, &col_comm)
```

As before, all MPI processes in the old_comm need to run this. Here, q new communicators are created and each process is automatically in one row_comm and one col_comm.

Ex.: You could create other communicators like for each diagonal. Then the communicators created would not have same number of processes each.



Use split_key = my_row + my_col

Option 3 = Special case of Cartesian grids = Cartesian topology commands.

```
MPI_Comm        grid_comm
int             ndims = 2
int             dimsizes[2]
int             wrap_around[2]
int             reorder    = 1   /* yes */
dimsize[0] = dimsize[1] = q
wrap_around[0] = 0 /* no */
wrap_around[1] = 1 /* yes */
MPI_Cart_create (MPI_COMM_WORLD,
ndims, dimsize, wrap_around, reorder,
&grid_comm )
```

reorder = 1 lets MPI just process into rows and columns in a way that allows for most efficient communications. This is a key advantage over option 2.

Now that we have grid_comm, which might have reordered processes, we can obtain 2-D process ids for (row, col) and their conversion to the original process ids.

```
MPI_Cart_coords (
    MPI_Comm        grid_comm,
    int             my_grid_rank,
    int             ndims,
    int             *coords )
```

Here, ndims is an input, since MPI cannot itself determine the dimensions of the grid comm.
coords is an output that is an array of length ndims.

Ex.: $q = 4$, $np = 16$, ndims = 2 then

my_grid_rank = 5 => coords = (1, 1)

```
MPI_Cart_rank (
    MPI_Comm        grid_comm,
    int             coords,
    int             *grid_rank )
```

Here, grid_rank is a scalar that is output.

Ex.: coords = (1,1) => grid_rank = 5

We created the entire Cartesian communicator in one call.
So, if we want row and column communicators,
we still have to create them.

```
MPI_Cart_sub (
        MPI_Comm        grid_comm,
        int             *free_coords,
        MPI_Comm        *new_comm )
```

Here, free_coords is a vector of length ndims.

free_coords[i] = 0 = false = no ⟺ the i-th coordinate is not free

Man page for MPI_Cart_sub uses remain_coords instead

remain_coords[i] = 1 = true = yes ⟺ the i-th coordinate remains and is not eliminated in new communicator

Ex.: row_comm ⟺ { remain_dims[0] = 0
                  " [1] = 1

col_comm ⟺ { remain_dims[0] = 1
             " [1] = 0

next time: Fox's algorithm for C = AB in block form