# COMP/CS 605: Introduction to Parallel Computing
# Topic : MPI Cartesian Communicators & Topologies

Mary Thomas

Department of Computer Science
Computational Science Research Center (CSRC)
San Diego State University (SDSU)

Posted: 03/07/17
Updated: 03/09/17

## Table of Contents

# MPI Communicators & Topologies

# MPI Communication Groups,

- MPI supports process PE grouping capability
- Group Tasks: application can be organized
- Collective communications operations can be performed on just a group.
- Can use to create virtual communication topologies.
- Communication safety
- Each group has its own communicator: can be used in point-to-point or collective communication routines.
- Groups and communicators are MPI objects:
  - stored in system space
  - accessed by handles

Ref: Pacheco, PPMPI
`http://mpi.deino.net/mpi_functions/index.htm`
`http://www.mpich.org/static/docs/v3.1/www3/`

## Commonly used MPI Group Routines

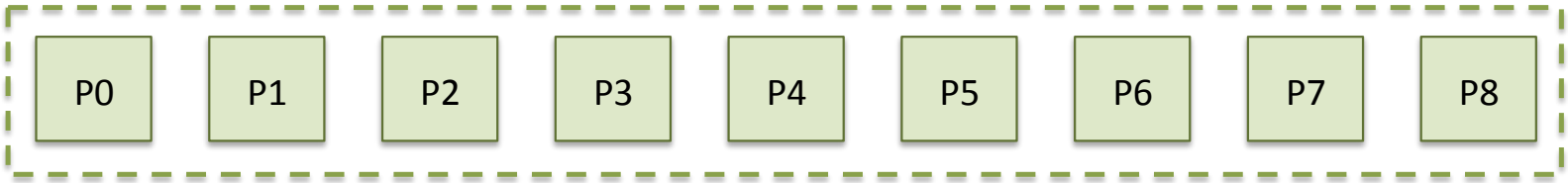
- **MPI_Group_size:** returns number of processes in group
- **MPI_Group_rank:** returns rank of calling process in group
- **MPI_Group_compare:** compares group members and group order
- **MPI_Group_translate_ranks:** translates ranks of processes in one group to those in another group
- **MPI_Comm_group:** 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

See `http://static.msi.umn.edu/tutorial/scicomp/general/MPI/communicator.html`

Example: *mpi-send-recv.c (1/1)*

```
/* mpi-send-recv.c --
 *    basic MPI environment
 *
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 */

#include <stdio.h>
#include "mpi.h"
#include <math.h>

main(int argc, char *argv[]) {
    int         comm_sz;
    int         my_rank;
    int         src, dest, tag=1;
    int         sval, rval;
    int         ierr;
    MPI_Status  status;


    /* start up initial MPI environment */
    /* create MPI_COMM_WORLD parent group */
    MPI_Init(&argc, &argv);

    /* get  number of PE's in MPI_COMM_WORLD group */
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

    /* get  rank of this processor in MPI_COMM_WORLD group */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* this processor is the first processor in the group */
    if (my_rank == 0) {
       rval = my_rank;
       src = my_rank;
       /* send a message to all the other processes */
       for (dest = 1; dest < comm_sz; dest++) {
          sval = 100+10*dest;
          MPI_Send(&sval, 1, MPI_INT, dest, tag,
                           MPI_COMM_WORLD);
       }
    } else { /* my_rank != 0 */
        MPI_Recv(&rval, 1, MPI_INT, src, tag,
                          MPI_COMM_WORLD, &status);
    }
    /* each processor prints it's rank and message */
    printf("My MPI_COMM_WORLD = PW[%d] / TotalPEs=[%d%]: My RVAL=[%d] \n
               my_rank, comm_sz, rval);

    MPI_Finalize();   /* shut down MPI env */
}  /* main */
```

## Output: *mpi-send-recv.c*




```
[comms]%  mpirun -np 5 ./mpi-send-recv
My MPI_COMM_WORLD = PW[0] / TotalPEs=[5]: My RVAL=[0]
My MPI_COMM_WORLD = PW[1] / TotalPEs=[5]: My RVAL=[110]
My MPI_COMM_WORLD = PW[2] / TotalPEs=[5]: My RVAL=[120]
My MPI_COMM_WORLD = PW[3] / TotalPEs=[5]: My RVAL=[130]
My MPI_COMM_WORLD = PW[4] / TotalPEs=[5]: My RVAL=[140]
```

# Example: *mpi-create-comm-group.c* (1/3)

```
/* mpi-create-comm-group.c --
 *     builds a communicator from the first q processes
 *     in a communicator containing arbitrary number of processes.
 *
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 *
 * Based loosely on code from Pacheco'97,
 *     See Chap 7,PPMPI
 *
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>
#include <stdlib.h>

main(int argc, char* argv[]) {
    int        p;
    int        q; /* = sqrt(p) */
    int        my_rank;
    int        proc,proc2;
    int        test = 0;
    int        sum;
    MPI_Group  group_world1,group_world2;
    MPI_Group  comm1_group, comm2_group;
    MPI_Comm   comm1, comm2;
    int*       comm1_procs;
    int*       comm2_procs;
    int        my_comm1_rank=-1, my_comm2_rank=-1;
    int i=0, err;
```

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

/* enable default MPI error handler */
MPI_Errhandler_set(MPI_COMM_WORLD, MPI_ERRORS_RETURN);

q = (int) sqrt((double) p);
if( (q*q) != p)  {
   if( my_rank ==0)
          printf("ERROR: p=%d must be a square number)\n",p)
   MPI_Finalize();
   exit(0);
  }
```

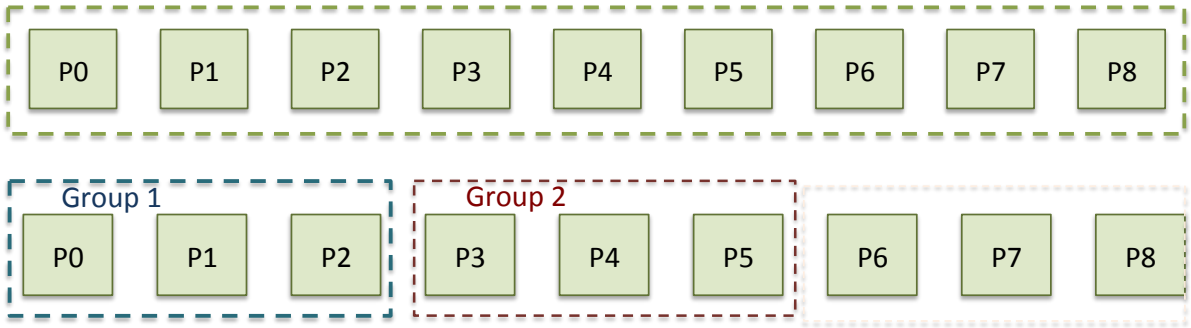
Example: *mpi-create-comm-group.c* (2/3)

```
/*****************************************************************/
/* Make a new communicator group: comm1                          */
/*****************************************************************/
/* Create a list of the comm1 processors */
comm1_procs = (int*) malloc(q*sizeof(int));
for (proc = 0; proc < q; proc++)
    comm1_procs[proc] = proc;

/* Get the group underlying MPI_COMM_WORLD */
MPI_Comm_group(MPI_COMM_WORLD, &group_world1);
/* Create the new group */
MPI_Group_incl(group_world1, q, comm1_procs, &comm1_group);

/* Create the new communicator: comm1 */
err=MPI_Comm_create(MPI_COMM_WORLD, comm1_group, &comm1);
if (err != MPI_SUCCESS) {
    printf("ERROR creating comm1\n");
    MPI_Finalize();    exit(0);
}
/* get my rank in this group */
err=MPI_Comm_rank(comm1, &my_comm1_rank);
if (err != MPI_SUCCESS) {
    printf("P[%d] is not a member of comm1\n",my_rank );
} else {
   printf("P[%d] is a member of comm1 group:  PCM1[%d]\n",my_rank, my_comm1_rank);
}

/* test collective ops in comm1 */
if (my_rank < q) {
    MPI_Comm_rank(comm1, &my_comm1_rank);
    test = 1;
    MPI_Bcast(&test, 1, MPI_INT, 0, comm1);
}
MPI_Reduce(&test, &sum, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
if (my_rank < q)   MPI_Bcast(&sum, 1, MPI_INT, 0, comm1);

printf("PW1[%d]/%d:  PCM1[%d], sum=[%d], sum+my_comm1_rank = %d\n",
    my_rank, q, my_comm1_rank, sum, sum+my_comm1_rank);
```

Example: *mpi-create-comm-group.c* (3/3)

```
    /*****************************************************************/
    /* Make a second communicator group: comm2                       */
    /*****************************************************************/
    /* Create a list of the comm2 processors */
    comm2_procs = (int*) malloc(q*sizeof(int));
    for (proc = 0; proc < q; proc++)
        comm2_procs[proc] = proc+q;

    /* Get the group underlying MPI_COMM_WORLD */
    MPI_Comm_group(MPI_COMM_WORLD, &group_world2);

    /* Create the new group */
    MPI_Group_incl(group_world2, q, comm2_procs, &comm2_group);

    /* Create the new communicator */
    err = MPI_Comm_create(MPI_COMM_WORLD, comm2_group, &comm2);
    if (err != MPI_SUCCESS) {
        printf("ERROR creating comm2\n");
        MPI_Finalize();
        exit(0);
    }
    /* get my rank in this group */
    err=MPI_Comm_rank(comm2, &my_comm2_rank);
    if (err != MPI_SUCCESS) {
        printf("P[%d] is not a member of comm2\n",my_rank );
    } else {
       /* do some collective ops in comm2 */
       printf("P[%d] is a member of comm2 group:  PCM2[%d]\n",my_rank, my_comm2_rank);
        if (my_comm2_rank == 0) test = 2;
        MPI_Bcast(&test, 1, MPI_INT, 0, comm2);
        MPI_Reduce(&test, &sum, 1, MPI_INT, MPI_SUM, 0, comm2);
        MPI_Bcast(&sum, 2, MPI_INT, 0, comm2);
        printf("PW2[%d]/%d:  PCM2[%d], sum=[%d], sum+my_comm2_rank = %d \n",
                my_rank, q, my_comm2_rank, sum, sum+my_comm2_rank);
        fflush(stdout);
    }
    MPI_Finalize();
| /* main */
```

# Output: *mpi-create-comm-group*




Figures showing 1D processor arrangement in MPI_COMM_WORLD (top) and the Comm2 groups (bottom).

```
[comms]% % mpirun -np 9 ./mpi-create-comm-group | sort | grep comm1
PW1[0]/3:  PCM1[0], sum=[3], sum+my_comm1_rank = 3
PW1[1]/3:  PCM1[1], sum=[3], sum+my_comm1_rank = 4
PW1[2]/3:  PCM1[2], sum=[3], sum+my_comm1_rank = 5
PW1[3]/3:  PCM1[-1], sum=[0], sum+my_comm1_rank = -1
PW1[4]/3:  PCM1[-1], sum=[0], sum+my_comm1_rank = -1
PW1[5]/3:  PCM1[-1], sum=[0], sum+my_comm1_rank = -1
PW1[6]/3:  PCM1[-1], sum=[0], sum+my_comm1_rank = -1
PW1[7]/3:  PCM1[-1], sum=[0], sum+my_comm1_rank = -1
PW1[8]/3:  PCM1[-1], sum=[0], sum+my_comm1_rank = -1

P[0] is a member of comm1 group:  PCM1[0]
P[1] is a member of comm1 group:  PCM1[1]
P[2] is a member of comm1 group:  PCM1[2]
P[3] is not a member of comm1
P[4] is not a member of comm1
P[5] is not a member of comm1
P[6] is not a member of comm1
P[7] is not a member of comm1
P[8] is not a member of comm1
```

```
PW2[3]/3:  PCM2[0], sum=[6], sum+my_comm2_rank = 6
PW2[4]/3:  PCM2[1], sum=[6], sum+my_comm2_rank = 7
PW2[5]/3:  PCM2[2], sum=[6], sum+my_comm2_rank = 8
P[0] is not a member of comm2
P[1] is not a member of comm2
P[2] is not a member of comm2
P[3] is a member of comm2 group:  PCM2[0]
P[4] is a member of comm2 group:  PCM2[1]
P[5] is a member of comm2 group:  PCM2[2]
P[6] is not a member of comm2
P[7] is not a member of comm2
P[8] is not a member of comm2
```

# MPI Virtual Topologies

- MPI topologies are virtual - there may be no relation between the physical structure of the parallel machine and the process topology.
- Virtual topologies are built upon MPI communicators and groups.
- Must be "programmed" by the application developer.
- Two Types: Cartesian, Graphs
- Cartesian: 1D, 2D, 3D arrangements
- Convenient:
  - Useful for applications with specific communication patterns - patterns that match an MPI topology structure.
- Improved communication efficiency:
  - hardware architectures may impose penalties for communications between successively distant "nodes".
  - A particular implementation may optimize process mapping based upon the physical characteristics of a given parallel machine.
  - The mapping of processes into an MPI virtual topology is dependent upon the MPI implementation, and may be ignored.

# Recall: MPI 3D Cartesian Mapping

- We have looked at MPI collective communication routines that optimize data distribution.
- Next, we need to look at ways to configure the processors to better match the geometry/approach needed to solve the scientific problem.
- Examples below use the following MPI Cartesian topologies:
  - MPI_Dims_create: Create N-Dimensional arrangement of PEs in the cartesian grid.
  - MPI_Cart_create:: Create N-Dimensional virtual topology/cartesian grid.
  - MPI_Cart_coords: Get local PE coordinates in the 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.
- Today we will look more closely at how this works.

# MPI_Dims_Create

**Creates array of PE dimensions in the cartesian grid.**
**If dims=0, then MPI can optimize to machine topology.**
**If dims has values, MPI will use what you have defined.**

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

| Parameter | i/o | Definition |
|---|---|---|
| nnodes: | in | number of nodes in a grid (integer) |
| ndims: | in | number of cartesian dimensions (integer) |
| dims: | in/out | integer array of size ndims specifying the number of nodes in each dimension. A value of 0 indicates that MPI_Dims_create should fill in a suitable value. |

# MPI_Cart_create

**Creates 1D,2D, or 3D virtual topology in a cartesian grid.**
**PE's are ordered from (0:NPEs-1); numbered/ranked in row-major (C lang) order.**

```
int MPI_Cart_create(
  MPI_Comm comm_old,
  int ndims,
  int *dims,
  int *periods,
  int reorder,
  MPI_Comm *comm_cart
);
```

| Parameter | i/o | Definition |
|---|---|---|
| comm_old | in | input communicator (handle) |
| ndims: | in | number of dimensions of cartesian grid (integer) |
| dims: | in | integer array of size ndims specifying the number of processes in each dimension |
| periods: | in | logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension |
| reorder: | in | ranking may be reordered (true) or not (false) (logical) |
| comm_cart: | out | communicator with new cartesian topology (handle) |

# MPI_Cart_coords

## Get local PE coordinates in the new cartesian grid

```
int MPI_Cart_coords(
  MPI_Comm comm,
  int rank,
  int maxdims,
  int *coords
);
```

| Parameter | i/o | Definition |
|---|---|---|
| comm: | in | communicator with cartesian structure (handle) |
| rank: | in | rank of a process within group of comm (integer) |
| maxdims: | in | length of vector coords in the calling program (integer) |
| coords: | out | integer array (of size ndims) containing the Cartesian coordinates of specified process (integer) |

# MPI_Cart_sub

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

```
int MPI_Cart_sub(
  MPI_Comm comm,
  int *free_coords,
  MPI_Comm *newcomm
);
```

| Parameter | i/o | Definition |
|---|---|---|
| comm: | in | communicator with cartesian structure (handle) |
| free_coords: | in | the ith entry of free_coords specifies whether the ith dimension is kept in the subgrid (true) or is dropped (false) (logical vector) |
| newcomm: | out | communicator containing the subgrid that includes the calling process (handle) |

# MPI_Cart_shift

**Returns the shifted source and destination ranks, given a shift direction and amount.**

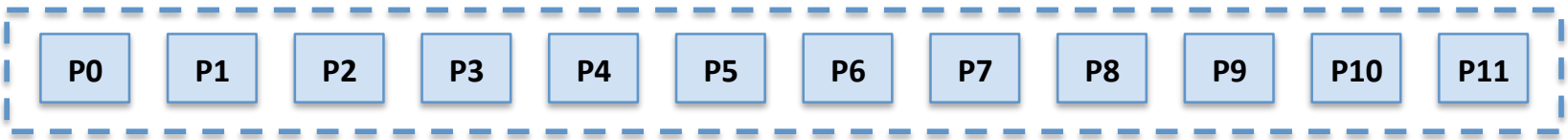
```
int MPI_Cart_shift(
  MPI_Comm comm,
  int direction,
  int displ,
  int *source,
  int *dest
);
```

| Parameter | i/o | Definition |
|---|---|---|
| comm: | in | communicator with cartesian structure (handle) |
| direction: | in | coordinate dimension of shift (integer) |
| displ: | in | displacement (¿ 0: upwards shift, ¡ 0: downwards shift) (integer) |
| source: | out | rank of source process (integer) |
| dest: | out | rank of destination process (integer) |

# Example: *mpi-cart-1D.c*

```
/* mpi-cart-1d.c --
 *     creates 1D cartesian topology
 *
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>

main(int argc, char *argv[]) {
    int       ndims=1;
    int       p;
    int       my_rank;
    MPI_Comm  comm1D;
    int       dims[ndims],coord[ndims];
    int       wrap_around[ndims];
    int       reorder;
    int       my_cart_rank;
    int       ierr;
    int       nrows, ncols;

    /* start up initial MPI environment */
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    dims[0]=0;

    /* create cartesian topology for processes */
    MPI_Dims_create(p, ndims, dims);
    if( my_rank == 0 )
        printf("PW[%d]/[%d%]: PEdims = [%d] \n",
                    my_rank,p,dims[0]);

    /* create cartesian mapping */
    wrap_around[0] = 0;  // set periodicity .false.
    reorder = 1;
    ierr =0;
    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                           wrap_around, reorder, &comm1D);
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    /* find my coordinates in the cartesian communicator group */
    MPI_Cart_coords(comm1D, my_rank, ndims, coord);

    /* use my coordinates to find my rank in cartesian group*/
    MPI_Cart_rank(comm1D, coord, &my_cart_rank);

    printf("PW[%d]: my_cart_rank PCM[%d], my coords = (%d)\n",
        my_rank, my_cart_rank, coord[0]);

    MPI_Comm_free(&comm1D );

    MPI_Finalize();
}  /* main */
```

# OUTPUT: *mpi-cart-1D.c*




```
[comm] %mpirun -np 12 ./mpi-cart-1d

PW[0]/[12]: PEdims = [12]
PW[0]: my_cart_rank PCM[0], my coords = (0)
PW[1]: my_cart_rank PCM[1], my coords = (1)
PW[2]: my_cart_rank PCM[2], my coords = (2)
PW[3]: my_cart_rank PCM[3], my coords = (3)
PW[4]: my_cart_rank PCM[4], my coords = (4)
PW[6]: my_cart_rank PCM[6], my coords = (6)
PW[7]: my_cart_rank PCM[7], my coords = (7)
PW[5]: my_cart_rank PCM[5], my coords = (5)
PW[8]: my_cart_rank PCM[8], my coords = (8)
PW[9]: my_cart_rank PCM[9], my coords = (9)
PW[10]: my_cart_rank PCM[10], my coords = (10)
PW[11]: my_cart_rank PCM[11], my coords = (11)
```

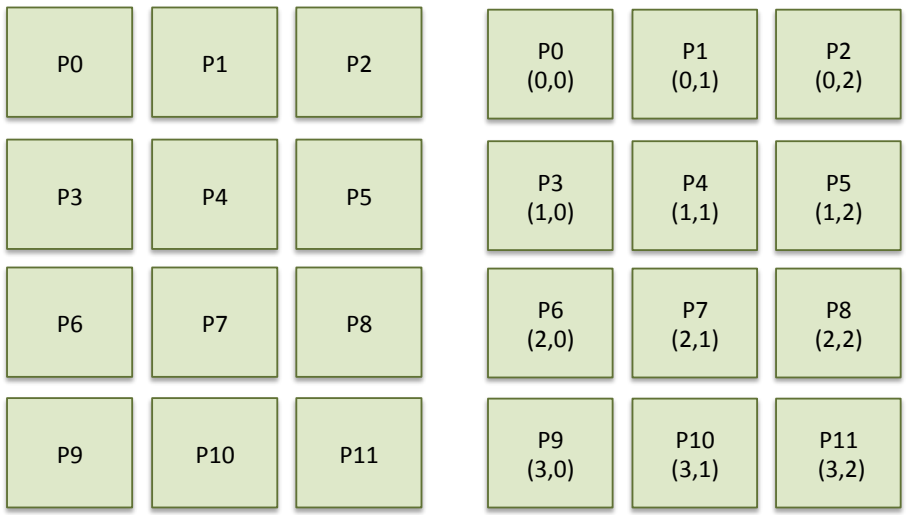
# Example: *mpi-cart-2D.c*

```
/* mpi-cart-2D.c -- test basic -cartesian functions
 * Written by Mary Thomas- Updated Mar, 2015
 * Based loosely on code from Pacheco'97,
 *       Chap 7, pp. 121 & ff in PPMPI  */
#include <stdio.h>
#include "mpi.h"
#include <math.h>

main(int argc, char *argv[]) {
    int       ndims=2,ierr;
    int       p,my_rank,my_cart_rank;
    MPI_Comm  comm2D;
    int       dims[ndims],coord[ndims];
    int       wrap_around[ndims];
    int       reorder,nrows, ncols;

    /* start up initial MPI environment */
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* process command line arguments*/
   if (argc == 3) {
      nrows = atoi (argv[1]); ncols = atoi (argv[2]);
      dims[0] = nrows; /* number of rows */
      dims[1] = ncols; /* number of columns */
      if( (nrows*ncols) != p)  {
         if( my_rank ==0)
              printf("ERROR: nrows*ncols)=%d * %d = %d != p\n",
                           nrows, ncols, nrows*ncols,p);
         MPI_Finalize();
         exit(0);
      }
   } else {
      nrows=ncols=(int)sqrt(p);
      dims[0]=dims[1]=0;
   }
```

```
  /* create cartesian topology for processes */
    MPI_Dims_create(p, ndims, dims);
    if( my_rank == 0 )
        printf("PW[%d]/[%d%]: PEdims = [%d x %d] \n",
                        my_rank,p,dims[0],dims[1]);

    /* create cartesian mapping */
    wrap_around[0] = wrap_around[1] = 0;  // set periodicity
    reorder = 1;
    ierr =0;
    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                           wrap_around, reorder, &comm2D);
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    /* find my coordinates in the cartesian communicator group */
    MPI_Cart_coords(comm2D, my_rank, ndims, coord);

    /* use my  coords to find my rank in cartesian group*/
    MPI_Cart_rank(comm2D, coord, &my_cart_rank);

    printf("PW[%d]: my_cart_rank PCM[%d], my coords = (%d,%d)\n",
        my_rank, my_cart_rank, coord[0], coord[1]);

    MPI_Comm_free(comm2D);

    MPI_Finalize();
}  /* main */
```

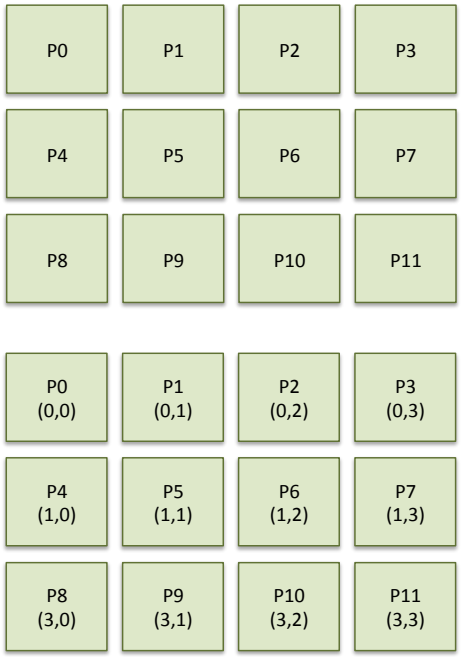
# OUTPUT: *mpi-cart-2D.c*: PE[4x3] topology (using default MPI PE arrangement)

```
[comms]%  mpirun -np 12 ./mpi-cart-2d

PW[0]/[12]: PEdims = [4 x 3]

PW[0]:  my_cart_rank PCM[0],  my coords = (0,0)
PW[1]:  my_cart_rank PCM[1],  my coords = (0,1)
PW[2]:  my_cart_rank PCM[2],  my coords = (0,2)
-----------------------------------------------
PW[3]:  my_cart_rank PCM[3],  my coords = (1,0)
PW[4]:  my_cart_rank PCM[4],  my coords = (1,1)
PW[5]:  my_cart_rank PCM[5],  my coords = (1,2)
-----------------------------------------------
PW[6]:  my_cart_rank PCM[6],  my coords = (2,0)
PW[7]:  my_cart_rank PCM[7],  my coords = (2,1)
PW[8]:  my_cart_rank PCM[8],  my coords = (2,2)
-----------------------------------------------
PW[9]:  my_cart_rank PCM[9],  my coords = (3,0)
PW[10]: my_cart_rank PCM[10], my coords = (3,1)
PW[11]: my_cart_rank PCM[11], my coords = (3,2)
```



# OUTPUT: *mpi-cart-2D.c*
# PE[3x4] topology (using command line arguments for nrows, ncols)

```
[comms]%  mpirun -np 12 ./mpi-cart-2d  3  4

PW[0]/[12]: PEdims = [3 x 4]

PW[0]:  my_cart_rank PCM[0],  my coords = (0,0)
PW[1]:  my_cart_rank PCM[1],  my coords = (0,1)
PW[2]:  my_cart_rank PCM[2],  my coords = (0,2)
PW[3]:  my_cart_rank PCM[3],  my coords = (0,3)
-----------------------------------------------
PW[4]:  my_cart_rank PCM[4],  my coords = (1,0)
PW[5]:  my_cart_rank PCM[5],  my coords = (1,1)
PW[6]:  my_cart_rank PCM[6],  my coords = (1,2)
PW[7]:  my_cart_rank PCM[7],  my coords = (1,3)
-----------------------------------------------
PW[8]:  my_cart_rank PCM[8],  my coords = (2,0)
PW[9]:  my_cart_rank PCM[9],  my coords = (2,1)
PW[10]: my_cart_rank PCM[10], my coords = (2,2)
PW[11]: my_cart_rank PCM[11], my coords = (2,3)
```



# Example: *mpi-cart-1D-get-nbrs.c* (1/2)

```
/* mpi-cart-1D-get-nbrs.c -- finds the neighbors in a cartesian communicator
 *
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 */
#include "mpi.h"
#include <stdio.h>
int main( int argc, char *argv[] )   {
    int errs = 0, ndims=1, size, rank,  source, dest;
    int dims[ndims], periods[ndims];
    MPI_Comm comm1D;

    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );

    /*************************************************************/
    /* Create periodic shift                                     */
    /*************************************************************/
    dims[0] = size;      /* processor dimensions */
    periods[0] = 1;  /* periodic shift is  .true. */

    /* create cartesian topology for processes */
    MPI_Dims_create(size, ndims, dims);
    if( rank == 0 )
        printf("PW[%d]/[%d%]: NDims=%d, PEdims = [%d] \n",rank,size,ndims,dims[0]);

    /* create cartesian mapping */
    MPI_Cart_create( MPI_COMM_WORLD, ndims, dims, periods, 0, &comm1D );

    MPI_Cart_shift( comm1D, 0, 1, &source, &dest );
    printf( "P[%d]: peri: shift  1:  src[%d] P[%d] dest[%d] \n", rank,source,rank,dest );fflush(stdout);

    MPI_Cart_shift( comm1D, 0, 0, &source, &dest );
    printf( "P[%d]: peri: shift  0:  src[%d] P[%d] dest[%d] \n", rank,source,rank,dest );fflush(stdout);

    MPI_Cart_shift( comm1D, 0, -1, &source, &dest );
    printf( "P[%d]: peri: shift -1:  src[%d] P[%d] dest[%d] \n", rank,source,rank,dest );fflush(stdout);
```

Example: *mpi-cart-1D-get-nbrs.c* (2/2)

```
    /*************************************************************/
    /* Create non-periodic shift                                 */
    /*************************************************************/
    if(rank == 0 ) printf("non-periodic next \n");
    MPI_Comm_free( &comm1D );

    periods[0] = 1;  /* periodic shift is  .false. */

    MPI_Cart_create( MPI_COMM_WORLD, 1, dims, periods, 0, &comm1D );
    MPI_Cart_shift( comm1D, 0, 1, &source, &dest );
    printf( "P[%d]: nonp: shift  1:  src[%d] P[%d] dest[%d] \n", rank,source,rank,dest );fflush(stdout);

    MPI_Cart_shift( comm1D, 0, 0, &source, &dest );
    printf( "P[%d]: nonp: shift  0:  src[%d] P[%d] dest[%d] \n", rank,source,rank,dest );fflush(stdout);

    MPI_Cart_shift( comm1D, 0, -1, &source, &dest );
    printf( "P[%d]: nonp: shift -1:  src[%d] P[%d] dest[%d] \n", rank,source,rank,dest );fflush(stdout);

    MPI_Comm_free( &comm1D );

    MPI_Finalize();
    return 0;
}
```

## OUTPUT: *mpi-cart-1D-get-nbrs*

```
[comms]%  mpirun -np 4 ./mpi-cart-1D-get-nbrs | sort
-------------------------------------------------
PW[0]/[4]: NDims=1, PEdims = [4]
-------------------------------------------------
P[0]: peri: shift -1:  src[1] P[0] dest[7]
P[0]: peri: shift  0:  src[0] P[0] dest[0]
P[0]: peri: shift  1:  src[7] P[0] dest[1]
-------------------------------------------------
P[1]: peri: shift -1:  src[2] P[1] dest[0]
P[1]: peri: shift  0:  src[1] P[1] dest[1]
P[1]: peri: shift  1:  src[0] P[1] dest[2]
-------------------------------------------------
P[2]: peri: shift -1:  src[3] P[2] dest[1]
P[2]: peri: shift  0:  src[2] P[2] dest[2]
P[2]: peri: shift  1:  src[1] P[2] dest[3]
-------------------------------------------------
P[3]: peri: shift -1:  src[4] P[3] dest[2]
P[3]: peri: shift  0:  src[3] P[3] dest[3]
P[3]: peri: shift  1:  src[2] P[3] dest[4]
-------------------------------------------------
P[4]: peri: shift -1:  src[5] P[4] dest[3]
P[4]: peri: shift  0:  src[4] P[4] dest[4]
P[4]: peri: shift  1:  src[3] P[4] dest[5]
-------------------------------------------------
P[5]: peri: shift -1:  src[6] P[5] dest[4]
P[5]: peri: shift  0:  src[5] P[5] dest[5]
P[5]: peri: shift  1:  src[4] P[5] dest[6]
-------------------------------------------------
P[6]: peri: shift -1:  src[7] P[6] dest[5]
P[6]: peri: shift  0:  src[6] P[6] dest[6]
P[6]: peri: shift  1:  src[5] P[6] dest[7]
-------------------------------------------------
P[7]: peri: shift -1:  src[0] P[7] dest[6]
P[7]: peri: shift  0:  src[7] P[7] dest[7]
P[7]: peri: shift  1:  src[6] P[7] dest[0]
```


```
-------------------------------------------------
P[0]: nonp: shift  1:  src[-1] P[0] dest[1]
P[0]: nonp: shift  0:  src[0] P[0] dest[0]
P[0]: nonp: shift -1:  src[1] P[0] dest[-1]
-------------------------------------------------
P[1]: nonp: shift  1:  src[0] P[1] dest[2]
P[1]: nonp: shift  0:  src[1] P[1] dest[1]
P[1]: nonp: shift -1:  src[2] P[1] dest[0]
-------------------------------------------------
P[2]: nonp: shift  1:  src[1] P[2] dest[3]
P[2]: nonp: shift  0:  src[2] P[2] dest[2]
P[2]: nonp: shift -1:  src[3] P[2] dest[1]
-------------------------------------------------
P[3]: nonp: shift  1:  src[2] P[3] dest[4]
P[3]: nonp: shift  0:  src[3] P[3] dest[3]
P[3]: nonp: shift -1:  src[4] P[3] dest[2]
-------------------------------------------------
P[4]: nonp: shift  1:  src[3] P[4] dest[5]
P[4]: nonp: shift  0:  src[4] P[4] dest[4]
P[4]: nonp: shift -1:  src[5] P[4] dest[3]
-------------------------------------------------
P[5]: nonp: shift  1:  src[4] P[5] dest[6]
P[5]: nonp: shift  0:  src[5] P[5] dest[5]
P[5]: nonp: shift -1:  src[6] P[5] dest[4]
-------------------------------------------------
P[6]: nonp: shift  1:  src[5] P[6] dest[7]
P[6]: nonp: shift  0:  src[6] P[6] dest[6]
P[6]: nonp: shift -1:  src[7] P[6] dest[5]
-------------------------------------------------
P[7]: nonp: shift  1:  src[6] P[7] dest[-1]
P[7]: nonp: shift  0:  src[7] P[7] dest[7]
P[7]: nonp: shift -1:  src[-1] P[7] dest[6]
```

# Example: *mpi-cart-2D-get-nbrs.c* (1/2)

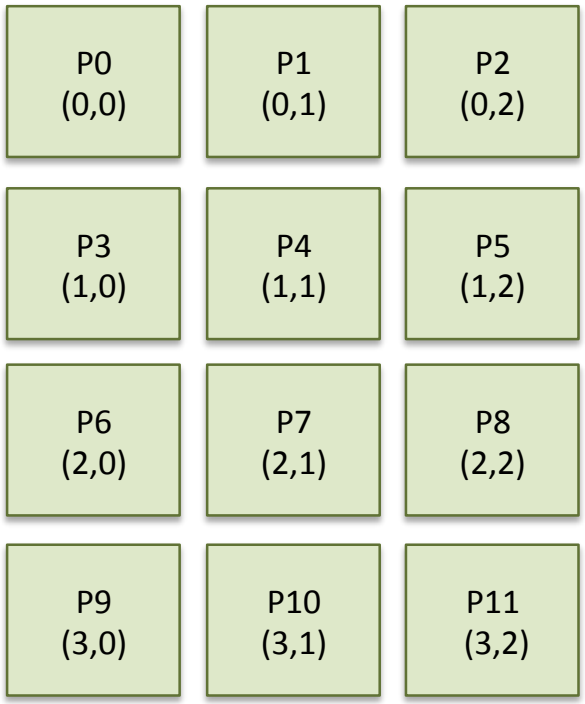
```
/* mpi-cart-get-nbrs.c -- gets the neighbors in a cartesian communicator
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>
#define NBRILO  10
#define NBRIHI  11
#define NBRJLO  12
#define NBRJHI  13
#define SHIFT_ROW 0
#define SHIFT_COL 1
#define DISP 1
int main(int argc, char *argv[]) {
    int       ndims=2,  size, my_rank, reorder, my_cart_rank, ierr, errs, nrows, ncols, source, dest, nbr_i_lo, nbr_i_hi;
    MPI_Comm  comm2D;
    int       dims[ndims],coord[ndims];
    int       wrap_around[ndims];

    /* start up initial MPI environment */
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    /* process command line arguments*/
   if (argc == 3) {
      nrows = atoi (argv[1]);
      ncols = atoi (argv[2]);
      dims[0] = nrows; /* number of rows */
      dims[1] = ncols; /* number of columns */
      if( (nrows*ncols) != size)  {
         if( my_rank ==0) printf("ERROR: nrows*ncols)=%d * %d = %d != size\n", nrows, ncols, nrows*ncols,size);
         MPI_Finalize();  exit(0);
      }
   } else {
      nrows=ncols=(int)sqrt(size);
      dims[0]=dims[1]=0;
   }
```

Example: *mpi-cart-2D-get-nbrs.c* (2/2)

```
    /*************************************************************/
    /* create cartesian topology for processes */
    /*************************************************************/
    MPI_Dims_create(size, ndims, dims);
    if(my_rank==0)
        printf("PW[%d], CommSz[%d%]: PEdims = [%d x %d] \n",my_rank,size,dims[0],dims[1]);

    /* create cartesian mapping */
    wrap_around[0] = wrap_around[1] = 0;  /* periodic shift is  .false. */
    reorder = 1;
    ierr =0;
    ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                           wrap_around, reorder, &comm2D);
    if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

    /* find my coordinates in the cartesian communicator group */
    MPI_Cart_coords(comm2D, my_rank, ndims, coord);

    /* use my cartesian coordinates to find my rank in cartesian group*/
    MPI_Cart_rank(comm2D, coord, &my_cart_rank);

    /* get my neighbors; axis is coordinate dimension of shift */
    /*  axis=0 ==> shift along the rows:   P[my_row-1]: P[me] : P[my_row+1] */
    /*  axis=1 ==> shift along the columns P[my_col-1]: P[me] : P[my_col+1] */
    MPI_Cart_shift( comm2D, SHIFT_ROW, DISP, &nbr_i_lo, &nbr_i_hi );
    MPI_Cart_shift( comm2D, SHIFT_COL, DISP, &nbr_j_lo, &nbr_j_hi );
    printf( "PW[%2d] Coord(%d,%d): SHIFT_DIM[%d], Shift=%d:  nbr_lo[%2d]  P[%2d]  nbr_hi[%2d] \n",
              my_rank, coord[0], coord[1],SHIFT_ROW, DISP,nbr_i_lo ,my_rank,nbr_i_hi );
    printf( "PW[%2d] Coord(%d,%d): SHIFT_DIM[%d], Shift=%d:  nbr_lo[%2d]  P[%2d]  nbr_hi[%2d] \n",
              my_rank, coord[0], coord[1],SHIFT_COL, DISP,nbr_j_lo ,my_rank,nbr_j_hi );
    fflush(stdout);

    MPI_Comm_free( &comm2D );

    MPI_Finalize();
}
```

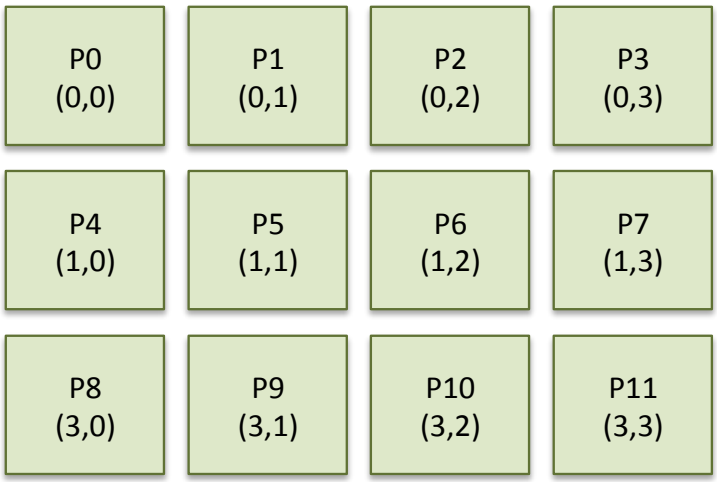
# OUTPUT:: *mpi-cart-2D-get-nbrs*: PE[4 x 3] topology using default PE dimensions.

```
[comms]%  mpirun -np 12 ./mpi-cart-2D-get-nbrs | sort
PW[0], CommSz[12]: PEdims = [4 x 3]
------------------------------------------------------------------
PW[ 0] C(0,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 0]  nbr_hi[ 1]
PW[ 0] C(0,0): SHIFT_DIM[0], Shift=1:  nbr_lo[-1]  P[ 0]  nbr_hi[ 3]
PW[ 1] C(0,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 0]  P[ 1]  nbr_hi[ 2]
PW[ 1] C(0,1): SHIFT_DIM[0], Shift=1:  nbr_lo[-1]  P[ 1]  nbr_hi[ 4]
PW[ 2] C(0,2): SHIFT_DIM[1], Shift=1:  nbr_lo[ 1]  P[ 2]  nbr_hi[-1]
PW[ 2] C(0,2): SHIFT_DIM[0], Shift=1:  nbr_lo[-1]  P[ 2]  nbr_hi[ 5]
------------------------------------------------------------------
PW[ 3] C(1,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 3]  nbr_hi[ 4]
PW[ 3] C(1,0): SHIFT_DIM[0], Shift=1:  nbr_lo[ 0]  P[ 3]  nbr_hi[ 6]
PW[ 4] C(1,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 3]  P[ 4]  nbr_hi[ 5]
PW[ 4] C(1,1): SHIFT_DIM[0], Shift=1:  nbr_lo[ 1]  P[ 4]  nbr_hi[ 7]
PW[ 5] C(1,2): SHIFT_DIM[1], Shift=1:  nbr_lo[ 4]  P[ 5]  nbr_hi[-1]
PW[ 5] C(1,2): SHIFT_DIM[0], Shift=1:  nbr_lo[ 2]  P[ 5]  nbr_hi[ 8]
------------------------------------------------------------------
PW[ 6] C(2,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 6]  nbr_hi[ 7]
PW[ 6] C(2,0): SHIFT_DIM[0], Shift=1:  nbr_lo[ 3]  P[ 6]  nbr_hi[ 9]
PW[ 7] C(2,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 6]  P[ 7]  nbr_hi[ 8]
PW[ 7] C(2,1): SHIFT_DIM[0], Shift=1:  nbr_lo[ 4]  P[ 7]  nbr_hi[10]
PW[ 8] C(2,2): SHIFT_DIM[1], Shift=1:  nbr_lo[ 7]  P[ 8]  nbr_hi[-1]
PW[ 8] C(2,2): SHIFT_DIM[0], Shift=1:  nbr_lo[ 5]  P[ 8]  nbr_hi[11]
------------------------------------------------------------------
PW[ 9] C(3,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 9]  nbr_hi[10]
PW[ 9] C(3,0): SHIFT_DIM[0], Shift=1:  nbr_lo[ 6]  P[ 9]  nbr_hi[-1]
PW[10] C(3,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 9]  P[10]  nbr_hi[11]
PW[10] C(3,1): SHIFT_DIM[0], Shift=1:  nbr_lo[ 7]  P[10]  nbr_hi[-1]
PW[11] C(3,2): SHIFT_DIM[1], Shift=1:  nbr_lo[10]  P[11]  nbr_hi[-1]
PW[11] C(3,2): SHIFT_DIM[0], Shift=1:  nbr_lo[ 8]  P[11]  nbr_hi[-1]
```



# OUTPUT:: *mpi-cart-2D-get-nbrs*: PE[3 x 4] topology using default PE dimensions.

```
[comms]%  mpirun -np 12 ./mpi-cart-2D-get-nbrs | sort
PW[0], CommSz[12]: PEdims = [3 x 4]
------------------------------------------------------------------
PW[ 0] C(0,0): SHIFT_DIM[0], Shift=1:  nbr_lo[-1]  P[ 0]  nbr_hi[ 3]
PW[ 0] C(0,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 0]  nbr_hi[ 1]
PW[ 1] C(0,1): SHIFT_DIM[0], Shift=1:  nbr_lo[-1]  P[ 1]  nbr_hi[ 4]
PW[ 1] C(0,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 0]  P[ 1]  nbr_hi[ 2]
PW[ 2] C(0,2): SHIFT_DIM[0], Shift=1:  nbr_lo[-1]  P[ 2]  nbr_hi[ 5]
PW[ 2] C(0,2): SHIFT_DIM[1], Shift=1:  nbr_lo[ 1]  P[ 2]  nbr_hi[-1]
------------------------------------------------------------------
PW[ 3] C(1,0): SHIFT_DIM[0], Shift=1:  nbr_lo[ 0]  P[ 3]  nbr_hi[ 6]
PW[ 3] C(1,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 3]  nbr_hi[ 4]
PW[ 4] C(1,1): SHIFT_DIM[0], Shift=1:  nbr_lo[ 1]  P[ 4]  nbr_hi[ 7]
PW[ 4] C(1,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 3]  P[ 4]  nbr_hi[ 5]
PW[ 5] C(1,2): SHIFT_DIM[0], Shift=1:  nbr_lo[ 2]  P[ 5]  nbr_hi[ 8]
PW[ 5] C(1,2): SHIFT_DIM[1], Shift=1:  nbr_lo[ 4]  P[ 5]  nbr_hi[-1]
------------------------------------------------------------------
PW[ 6] C(2,0): SHIFT_DIM[0], Shift=1:  nbr_lo[ 3]  P[ 6]  nbr_hi[-1]
PW[ 6] C(2,0): SHIFT_DIM[1], Shift=1:  nbr_lo[-1]  P[ 6]  nbr_hi[ 7]
PW[ 7] C(2,1): SHIFT_DIM[0], Shift=1:  nbr_lo[ 4]  P[ 7]  nbr_hi[-1]
PW[ 7] C(2,1): SHIFT_DIM[1], Shift=1:  nbr_lo[ 6]  P[ 7]  nbr_hi[ 8]
PW[ 8] C(2,2): SHIFT_DIM[0], Shift=1:  nbr_lo[ 5]  P[ 8]  nbr_hi[-1]
PW[ 8] C(2,2): SHIFT_DIM[1], Shift=1:  nbr_lo[ 7]  P[ 8]  nbr_hi[-1]
------------------------------------------------------------------
```



# Example: *mpi-cart-2D-row.c (1/4)*

```
/* mpi-cart-2D-row.c --
 *     creates a 1D row cartesian communicator
 *
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 *
 * Based loosely on code from Pacheco'97,
 *       Chap 7,  PPMPI
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>


main(int argc, char *argv[]) {
    int       ndims=2;
    int       p;
    int       my_rank;
    int       dims[ndims],coord[ndims];
    int       wrap_around[ndims];
    int       reorder;
    int       my_cart_rank;
    int       ierr;
    int       nrows, ncols;
    char      name[200], nameout[200], rname[100];
    int       cnt, rlen;
    MPI_Comm  comm2D;

    /* for row split  */
    int       my_row_rank;
    int       free_coords[ndims];
    int       row_coord[ndims];
    int       row_val;
    int       row_sum=0, row_root;
    MPI_Comm  comm1D_row;
```

# Example: *mpi-cart-2D-row.c (2/4)*

```
/*************************************************************/
   /* start up initial MPI environment: MPI_COMM_WORLD          */
   /*************************************************************/

   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &p);
   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
   /* turn on error handling */
   MPI_Errhandler_set(MPI_COMM_WORLD, MPI_ERRORS_RETURN);

   /* get the name of communicator */
   nameout[0] = 0;
   MPI_Comm_get_name( MPI_COMM_WORLD, nameout, &rlen );
   if(my_rank == 0) printf( "Name of comm world is: %s\n", nameout );
   fflush(stdout);

   /* process command line arguments*/
  if (argc == 3) {
     nrows = atoi (argv[1]);
     ncols = atoi (argv[2]);
     dims[0] = nrows; /* number of rows */
     dims[1] = ncols; /* number of columns */
     if( (nrows*ncols) != p)  {
        if( my_rank ==0) printf("ERROR: nrows*ncols)=%d * %d = %d != p\n",
                                nrows, ncols, nrows*ncols,p);
        MPI_Finalize();
        exit(0);
     }
  } else {
     nrows=ncols=(int)sqrt(p);
     dims[0]=dims[1]=0;
  }
```

# Example: *mpi-cart-2D-row.c (3/4)*

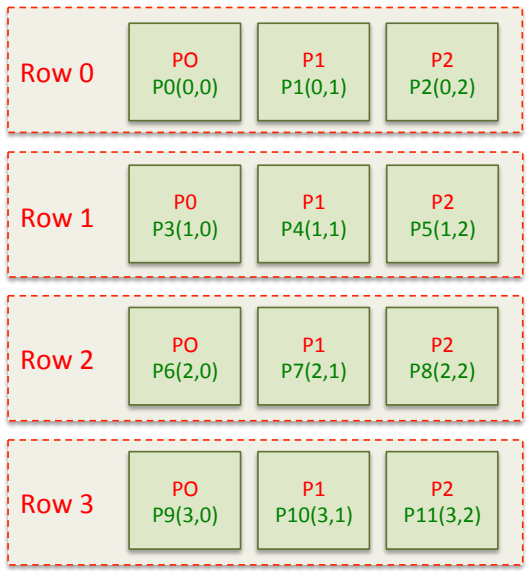
```
/*************************************************************/
/* create a 2D cartesian communicator: comm2D                */
/*************************************************************/

/* set dimensions for cartesian topology for processes */
MPI_Dims_create(p, ndims, dims);
if( my_rank == 0 )
    printf("PW[%d]/[%d%]: PEdims = [%d x %d] \n",my_rank,p,dims[0],dims[1]);

/* create cartesian mapping */
wrap_around[0] = wrap_around[1] = 0;  // set periodicity
reorder = 1;
ierr =0;
ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                       wrap_around, reorder, &comm2D);
if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

/* set the name of cartesian communicator */
strcpy( name, "comm2D" );
MPI_Comm_set_name( comm2D, name );
nameout[0] = 0;

/* get the name of communicator */
MPI_Comm_get_name( comm2D, nameout, &rlen );
if(my_rank == 0) printf( "Name of comm world is: %s\n", nameout );
fflush(stdout);

/* find my coordinates in the cartesian communicator group */
MPI_Cart_coords(comm2D, my_rank, ndims, coord);

/* use my cartesian coordinates to find my rank in cartesian group*/
MPI_Cart_rank(comm2D, coord, &my_cart_rank);
```

# Example: *mpi-cart-2D-row.c (4/4)*

```
    /*************************************************************/
    /* split comm2D into a 1D row-based communicator: comm1D_row */
    /*************************************************************/
    free_coords[0] = 0;  /* rows */;       free_coords[1] = 1;  /* cols */
    MPI_Cart_sub(comm2D, free_coords, &comm1D_row);

    /*  get my_row_rank in my comm1D_row group */
    MPI_Comm_rank(comm1D_row, &my_row_rank);

    /* use my_row_rank to find my coordinates in my comm1D_row group */
    MPI_Cart_coords(comm1D_row, my_row_rank, 1, &row_coord);

    /* set the name of cartesian communicator */
    row_root=-999;
    if(coord[1] == 0) row_root = my_rank;   /* use rows root rank for ID of row */
    MPI_Bcast(&row_root, 1, MPI_INT, 0, comm1D_row);

    /* set the name of cartesian communicator and build a unique name for it */
    sprintf(rname, "%d", row_root);
    strcpy( name, "comm1D_row");  strcat( name, "_"); strcat( name, rname);
    MPI_Comm_set_name( comm1D_row, name );

     /* get the name of communicator */
    MPI_Comm_get_name( comm1D_row, nameout, &rlen );

    /* test row com: row_sum processor ranks across rows */
    if (coord[1] == 0)
        row_val = coord[0];
    else
        row_val = -1;
    MPI_Bcast(&row_val, 1, MPI_INT, 0, comm1D_row);
    MPI_Reduce(&my_rank, &row_sum, 1, MPI_INT, MPI_SUM, 0, comm1D_row);
    MPI_Bcast(&row_sum, 1, MPI_INT, 0, comm1D_row);
     printf("PW[%d]: PCM[%d%]: C(%d,%d), RowComm: Root PW[ %d], Name=%s,RSum=%d \n",
         my_rank, my_row_rank , coord[0], coord[1], row_root, nameout,row_sum);

    MPI_Comm_free(comm2D);(comm1D_row);
    MPI_Finalize();
}  /* main */
```

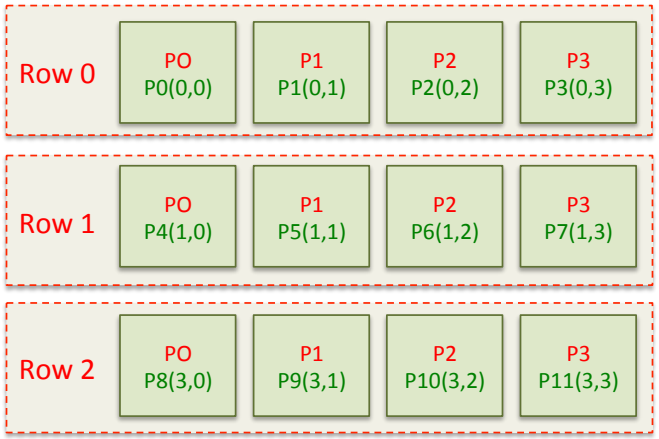
# OUTPUT:: *mpi-cart-2D-row*: PE[4x3] topology, MPI defined PE dims.

```
[comms]% mpirun -np 12 ./mpi-cart-2D-row | sort
Name of comm world is: MPI_COMM_WORLD; Name of comm world is: comm2D
PW[0]/[12]: PEdims = [4 x 3]
------------------------------------------------------------------
PW[0],PWC(0,0)::RowComm::PC[0],CC(0),RootPW[0],Name=comm1D_row_0,RSum=3
PW[1],PWC(0,1)::RowComm::PC[1],CC(1),RootPW[0],Name=comm1D_row_0,RSum=3
PW[2],PWC(0,2)::RowComm::PC[2],CC(2),RootPW[0],Name=comm1D_row_0,RSum=3
------------------------------------------------------------------
PW[3],PWC(1,0)::RowComm::PC[0],CC(0),RootPW[3],Name=comm1D_row_3,RSum=12
PW[4],PWC(1,1)::RowComm::PC[1],CC(1),RootPW[3],Name=comm1D_row_3,RSum=12
PW[5],PWC(1,2)::RowComm::PC[2],CC(2),RootPW[3],Name=comm1D_row_3,RSum=12
------------------------------------------------------------------
PW[6],PWC(2,0)::RowComm::PC[0],CC(0),RootPW[6],Name=comm1D_row_6,RSum=21
PW[7],PWC(2,1)::RowComm::PC[1],CC(1),RootPW[6],Name=comm1D_row_6,RSum=21
PW[8],PWC(2,2)::RowComm::PC[2],CC(2),RootPW[6],Name=comm1D_row_6,RSum=21
------------------------------------------------------------------
PW[9], PWC(3,0)::RowComm::PC[0],CC(0),RootPW[9],Name=comm1D_row_9,RSum=30
PW[10],PWC(3,1)::RowComm::PC[1],CC(1),RootPW[9],Name=comm1D_row_9,RSum=30
PW[11],PWC(3,2)::RowComm::PC[2],CC(2),RootPW[9],Name=comm1D_row_9,RSum=30
------------------------------------------------------------------
```



# OUTPUT:: *mpi-cart-2D-row*: PE[3x4] topology, User defined PE dims.

```
[comms]% % mpirun -np 12 ./mpi-cart-2D-row 3 4
Name of comm world is: MPI_COMM_WORLD; Name of comm1D world is: comm2D
PW[0]/[12]: PEdims = [3 x 4]
--------------------------------------------------------------------------
PW[0],PWC(0,0)::RowComm::PC[0],CC(0),RootPW[0],Name=comm1D_row_0,RSum=6
PW[1],PWC(0,1)::RowComm::PC[1],CC(1),RootPW[0],Name=comm1D_row_0,RSum=6
PW[2],PWC(0,2)::RowComm::PC[2],CC(2),RootPW[0],Name=comm1D_row_0,RSum=6
PW[3],PWC(0,3)::RowComm::PC[3],CC(3),RootPW[0],Name=comm1D_row_0,RSum=6
--------------------------------------------------------------------------
PW[4],PWC(1,0)::RowComm::PC[0],CC(0),RootPW[4],Name=comm1D_row_4,RSum=22
PW[5],PWC(1,1)::RowComm::PC[1],CC(1),RootPW[4],Name=comm1D_row_4,RSum=22
PW[6],PWC(1,2)::RowComm::PC[2],CC(2),RootPW[4],Name=comm1D_row_4,RSum=22
PW[7],PWC(1,3)::RowComm::PC[3],CC(3),RootPW[4],Name=comm1D_row_4,RSum=22
--------------------------------------------------------------------------
PW[8], PWC(2,0)::RowComm::PC[0],CC(0),RootPW[8],Name=comm1D_row_8,RSum=38
PW[9], PWC(2,1)::RowComm::PC[1],CC(1),RootPW[8],Name=comm1D_row_8,RSum=38
PW[10],PWC(2,2)::RowComm::PC[2],CC(2),RootPW[8],Name=comm1D_row_8,RSum=38
PW[11],PWC(2,3)::RowComm::PC[3],CC(3),RootPW[8],Name=comm1D_row_8,RSum=38
--------------------------------------------------------------------------
```



# Example: *mpi-cart-2D-col.c* (1/4)

```
/* mpi-cart-2D-col.c --
 *     creates a 1D column cartesian communicator
 *
 * Written by Mary Thomas
 *       - Updated Mar, 2015
 *
 * Based loosely on code from Pacheco'97,
 *       Chap 7,  PPMPI
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>

main(int argc, char *argv[]) {
    int       ndims=2;
    int       p;
    int       my_rank;
    int       dims[ndims],coord[ndims];
    int       wrap_around[ndims];
    int       reorder;
    int       my_cart_rank;
    int       ierr;
    int       nrows, ncols;
    char      name[200], nameout[200], rname[100];
    int       cnt, rlen;
    MPI_Comm  comm2D;

     /* for col split  */
    int       my_col_rank;
    int       free_coords[ndims];
    int       col_coord;
    int       col_val;
    int       col_sum=0, col_root;
    MPI_Comm  comm1D_col;
```

Example: *mpi-cart-2D-col.c* (2/4)

```
 /*************************************************************/
 /* start up initial MPI environment: MPI_COMM_WORLD          */
 /*************************************************************/

 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &p);
 MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
 /* turn on error handling */
 MPI_Errhandler_set(MPI_COMM_WORLD, MPI_ERRORS_RETURN);

 /* get the name of communicator */
 nameout[0] = 0;
 MPI_Comm_get_name( MPI_COMM_WORLD, nameout, &rlen );
 if(my_rank == 0) printf( "Name of comm world is: %s\n", nameout );
 fflush(stdout);

 /* process command line arguments*/
if (argc == 3) {
   nrows = atoi (argv[1]);
   ncols = atoi (argv[2]);
   dims[0] = nrows; /* number of rows */
   dims[1] = ncols; /* number of columns */
   if( (nrows*ncols) != p)  {
      if( my_rank ==0) printf("ERROR: nrows*ncols)=%d * %d = %d != p\n",
                              nrows, ncols, nrows*ncols,p);
      MPI_Finalize();
      exit(0);
   }
} else {
   nrows=ncols=(int)sqrt(p);
   dims[0]=dims[1]=0;
}
```

Example: *mpi-cart-2D-col.c* (3/4)

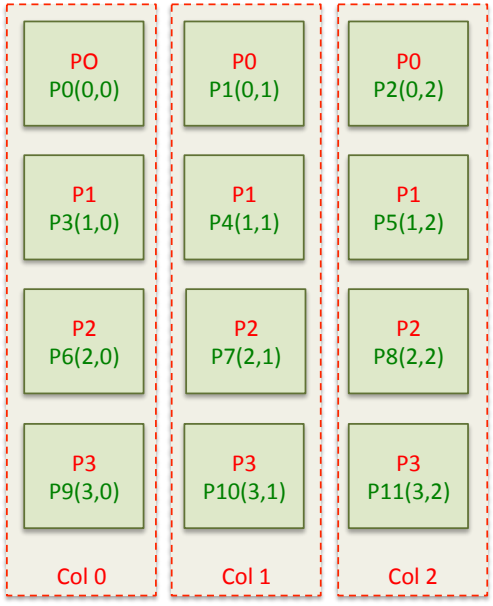
```
/*************************************************************/
/* create a 2D cartesian communicator: comm2D                */
/*************************************************************/

/* set dimensions for cartesian topology for processes */
MPI_Dims_create(p, ndims, dims);
if( my_rank == 0 )
    printf("PW[%d]/[%d%]: PEdims = [%d x %d] \n",my_rank,p,dims[0],dims[1]);

/* create cartesian mapping */
ierr =0;
wrap_around[0] = wrap_around[1] = 0;  // set periodicity
reorder = 1;
ierr = MPI_Cart_create(MPI_COMM_WORLD, ndims, dims,
                       wrap_around, reorder, &comm2D);
if(ierr != 0) printf("ERROR[%d] creating CART\n",ierr);

/* set the name of cartesian communicator */
strcpy( name, "comm2D" );
MPI_Comm_set_name( comm2D, name );
nameout[0] = 0;

/* get the name of communicator */
MPI_Comm_get_name( comm2D, nameout, &rlen );
if(my_rank == 0) printf( "Name of comm world is: %s\n", nameout );
fflush(stdout);

/* find my coordinates in the cartesian communicator group */
MPI_Cart_coords(comm2D, my_rank, ndims, coord);

/* use my cartesian coordinates to find my rank in cartesian group*/
MPI_Cart_rank(comm2D, coord, &my_cart_rank);
```

Example: *mpi-cart-2D-col.c* (4/4)

```
 /*************************************************************/
/* split comm2D into a 1D row-based communicator: comm1D_col */
/*************************************************************/
free_coords[0] = 1;  /* rows: this *dimension* belongs to subgrid  */
free_coords[1] = 0;  /* cols */
MPI_Cart_sub(comm2D, free_coords, &comm1D_col);

/* use my PW cartesian coordinates to get my_col_rank in my row comm group */
MPI_Cart_rank(comm1D_col, coord, &my_col_rank);

/* use my_col_rank to find my coordinates in the COL cart. comm group */
MPI_Cart_coords(comm1D_col, my_col_rank, 1, &col_coord);

/* set the name of cartesian communicator */
col_root=-999;
if(coord[0] == 0) col_root = my_rank;   /* use rows root rank for ID of row */
MPI_Bcast(&col_root, 1, MPI_INT, 0, comm1D_col);

/* set the name of cartesian communicator and build a unique name for it */
sprintf(rname, "%d", col_root);
strcpy( name, "comm1D_col");  strcat( name, "_"); strcat( name, rname);
MPI_Comm_set_name( comm1D_col, name );
nameout[0] = 0;

/* get the name of communicator */
MPI_Comm_get_name( comm1D_col, nameout, &rlen );

/* test row com: col_sum processor ranks across rows */
if (coord[0] == 0)
    col_val = coord[1];
else
    col_val = -1;
MPI_Bcast(&col_val, 1, MPI_INT, 0, comm1D_col);
MPI_Reduce(&my_rank, &col_sum, 1, MPI_INT, MPI_SUM, 0, comm1D_col);
MPI_Bcast(&col_sum, 1, MPI_INT, 0, comm1D_col);
printf("PW[%d],PWC(%d,%d)::ColComm::PC[%d],CC(%d),Root=%d,Name=%s,CSum=%d\n",
     my_rank,coord[0],coord[1],my_col_rank,col_coord,col_root, nameout,col_sum);
MPI_Comm_free(&comm1D_col);     MPI_Comm_free(&comm2D);
MPI_Finalize();
}  /* main */
```

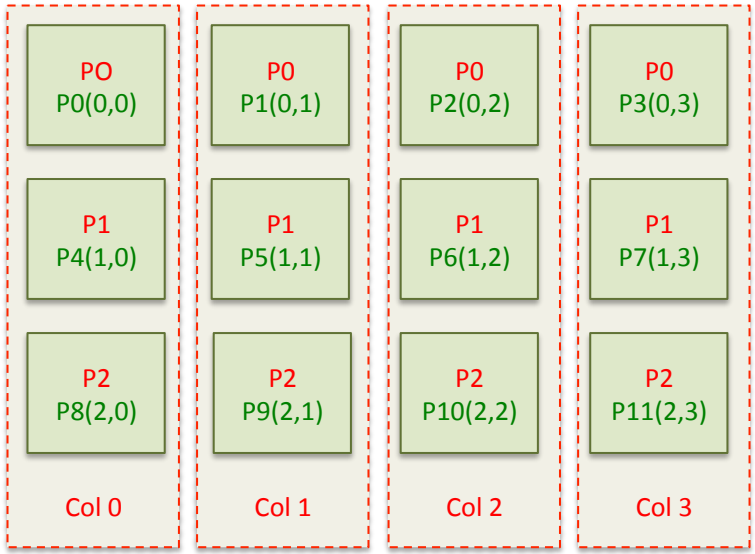
# OUTPUT:: *mpi-cart-2D-col*: PE[4x3] topology, MPI defined PE dims..

```
[comms]% % mpirun -np 12 ./mpi-cart-2D-col
Name of comm world is: MPI_COMM_WORLD
PW[0]/[12]: PEdims = [4 x 3]
Name of comm world is: comm2D
------------------------------------------------------------------------
PW[0], PWC(0,0)::ColComm::PC[0],CC(0),Root=0,Name=comm1D_col_0,CSum=18
PW[3], PWC(1,0)::ColComm::PC[1],CC(1),Root=0,Name=comm1D_col_0,CSum=18
PW[6], PWC(2,0)::ColComm::PC[2],CC(2),Root=0,Name=comm1D_col_0,CSum=18
PW[9], PWC(3,0)::ColComm::PC[3],CC(3),Root=0,Name=comm1D_col_0,CSum=18
------------------------------------------------------------------------
PW[1], PWC(0,1)::ColComm::PC[0],CC(0),Root=1,Name=comm1D_col_1,CSum=22
PW[4], PWC(1,1)::ColComm::PC[1],CC(1),Root=1,Name=comm1D_col_1,CSum=22
PW[7], PWC(2,1)::ColComm::PC[2],CC(2),Root=1,Name=comm1D_col_1,CSum=22
PW[10],PWC(3,1)::ColComm::PC[3],CC(3),Root=1,Name=comm1D_col_1,CSum=22
------------------------------------------------------------------------
PW[2], PWC(0,2)::ColComm::PC[0],CC(0),Root=2,Name=comm1D_col_2,CSum=26
PW[5], PWC(1,2)::ColComm::PC[1],CC(1),Root=2,Name=comm1D_col_2,CSum=26
PW[8], PWC(2,2)::ColComm::PC[2],CC(2),Root=2,Name=comm1D_col_2,CSum=26
PW[11],PWC(3,2)::ColComm::PC[3],CC(3),Root=2,Name=comm1D_col_2,CSum=26
------------------------------------------------------------------------
```



# OUTPUT:: *mpi-cart-2D-col*: PE[3x4] topology; User defined PE dims..

```
[comms]%   mpirun -np 12 ./mpi-cart-2D-col 3 4
Name of comm world is: MPI_COMM_WORLD
PW[0]/[12]: PEdims = [3 x 4]
Name of comm world is: comm2D
-------------------------------------------------------------------------
PW[0],PWC(0,0)::ColComm::PC[0],CC(0),Root=0,Name=comm1D_col_0,CSum=12
PW[4],PWC(1,0)::ColComm::PC[1],CC(1),Root=0,Name=comm1D_col_0,CSum=12
PW[8],PWC(2,0)::ColComm::PC[2],CC(2),Root=0,Name=comm1D_col_0,CSum=12
-------------------------------------------------------------------------
PW[1],PWC(0,1)::ColComm::PC[0],CC(0),Root=1,Name=comm1D_col_1,CSum=15
PW[5],PWC(1,1)::ColComm::PC[1],CC(1),Root=1,Name=comm1D_col_1,CSum=15
PW[9],PWC(2,1)::ColComm::PC[2],CC(2),Root=1,Name=comm1D_col_1,CSum=15
-------------------------------------------------------------------------
PW[2], PWC(0,2)::ColComm::PC[0],CC(0),Root=2,Name=comm1D_col_2,CSum=18
PW[6], PWC(1,2)::ColComm::PC[1],CC(1),Root=2,Name=comm1D_col_2,CSum=18
PW[10],PWC(2,2)::ColComm::PC[2],CC(2),Root=2,Name=comm1D_col_2,CSum=18
-------------------------------------------------------------------------
PW[3], PWC(0,3)::ColComm::PC[0],CC(0),Root=3,Name=comm1D_col_3,CSum=21
PW[7], PWC(1,3)::ColComm::PC[1],CC(1),Root=3,Name=comm1D_col_3,CSum=21
PW[11],PWC(2,3)::ColComm::PC[2],CC(2),Root=3,Name=comm1D_col_3,CSum=21
-------------------------------------------------------------------------
```



# Example: Pacheco top_fcns.c

```
/* top_fcns.c -- test basic topology functions
 *
 * Input: none
 * Output: results of calls to various functions
 *     testing topology creation
 *
 * Algorithm:
 *     1.  Build a 2-dimensional Cartesian communicator from
 *         MPI_Comm_world
 *     2.  Print topology information for each process
 *     3.  Use MPI_Cart_sub to build a communicator for each
 *         row of the Cartesian communicator
 *     4.  Carry out a broadcast across each row communicator
 *     5.  Print results of broadcast
 *     6.  Use MPI_Cart_sub to build a communicator for each
 *         column of the Cartesian communicator
 *     7.  Carry out a broadcast across each column communicator
 *     8.  Print results of broadcast
 *
 * Note: Assumes the number of processes, p, is a perfect square
 *
 * See Chap 7, pp. 121 & ff in PPMPI
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>
```

```
main(int argc, char* argv[]) {
    int       p;
    int       my_rank;
    int       q;
    MPI_Comm  grid_comm;
    int       dim_sizes[2];
    int       wrap_around[2];
    int       reorder = 1;
    int       coordinates[2];
    int       my_grid_rank;
    int       grid_rank;
    int       free_coords[2];
    MPI_Comm  row_comm;
    MPI_Comm  col_comm;
    int       row_test;
    int       col_test;

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

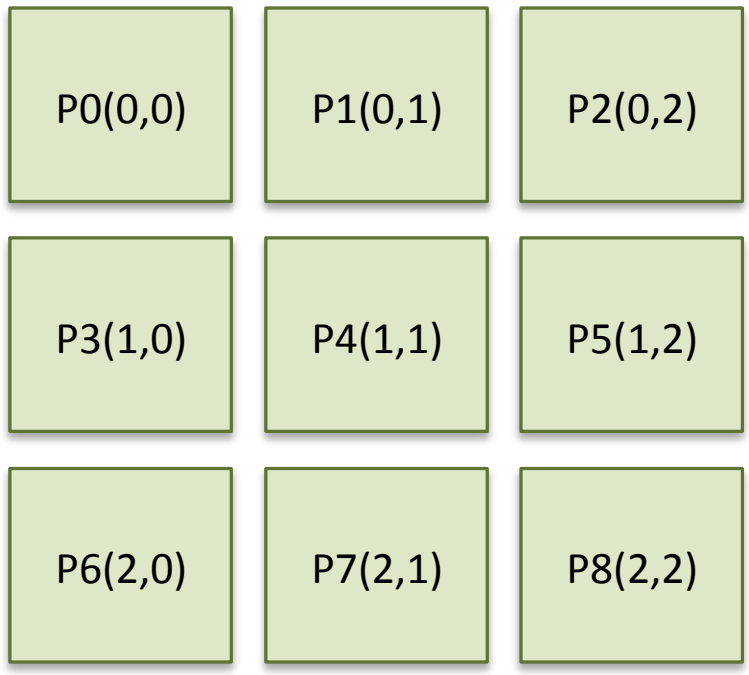
# MPI Topology Functions - Pacheco' top_fcns.c

```
   q = (int) sqrt((double) p);

   dim_sizes[0] = dim_sizes[1] = q;
   wrap_around[0] = wrap_around[1] = 1;
   MPI_Cart_create(MPI_COMM_WORLD, 2, dim_sizes,
       wrap_around, reorder, &grid_comm);

   MPI_Comm_rank(grid_comm, &my_grid_rank);
   MPI_Cart_coords(grid_comm, my_grid_rank, 2,
       coordinates);

   MPI_Cart_rank(grid_comm, coordinates, &grid_rank);

   printf("Process %d > my_grid_rank = %d,
                  coords = (%d,%d), grid_rank = %d\n",
       my_rank, my_grid_rank, coordinates[0],
                coordinates[1], grid_rank);

  free_coords[0] = 0;
   free_coords[1] = 1;
   MPI_Cart_sub(grid_comm, free_coords, &row_comm);
   if (coordinates[1] == 0)
       row_test = coordinates[0];
   else
       row_test = -1;
   MPI_Bcast(&row_test, 1, MPI_INT, 0, row_comm);
   printf("Process %d > coords = (%d,%d), row_test = %d\n",
        my_rank, coordinates[0], coordinates[1], row_test);


}  /* main */
```

```
free_coords[0] = 1;
free_coords[1] = 0;
MPI_Cart_sub(grid_comm, free_coords, &col_comm);
if (coordinates[0] == 0)
    col_test = coordinates[1];
else
    col_test = -1;
MPI_Bcast(&col_test, 1, MPI_INT, 0, col_comm);
printf("Process %d > coords = (%d,%d), col_test = %d\n",
     my_rank, coordinates[0], coordinates[1], col_test);

MPI_Finalize();
```

# OUTPUT: Pacheco' top_fcns.c

```
[gidget:comms] mthomas% mpirun -np 9 ./top_fcns
Process 0 > my_grid_rank = 0, coords = (0,0), grid_rank = 0
Process 2 > my_grid_rank = 2, coords = (0,2), grid_rank = 2
Process 3 > my_grid_rank = 3, coords = (1,0), grid_rank = 3
Process 4 > my_grid_rank = 4, coords = (1,1), grid_rank = 4
Process 5 > my_grid_rank = 5, coords = (1,2), grid_rank = 5
Process 6 > my_grid_rank = 6, coords = (2,0), grid_rank = 6
Process 7 > my_grid_rank = 7, coords = (2,1), grid_rank = 7
Process 8 > my_grid_rank = 8, coords = (2,2), grid_rank = 8
Process 1 > my_grid_rank = 1, coords = (0,1), grid_rank = 1
------------------------------------------------------------
Process 3 > coords = (1,0), row_test = 1
Process 4 > coords = (1,1), row_test = 1
Process 5 > coords = (1,2), row_test = 1
Process 6 > coords = (2,0), row_test = 2
Process 8 > coords = (2,2), row_test = 2
Process 7 > coords = (2,1), row_test = 2
Process 0 > coords = (0,0), row_test = 0
Process 1 > coords = (0,1), row_test = 0
Process 2 > coords = (0---,2), row_test = 0
------------------------------------
Process 0 > coords = (0,0), col_test = 0
Process 1 > coords = (0,1), col_test = 1
Process 2 > coords = (0,2), col_test = 2
Process 4 > coords = (1,1), col_test = 1
Process 5 > coords = (1,2), col_test = 2
Process 6 > coords = (2,0), col_test = 0
Process 7 > coords = (2,1), col_test = 1
Process 8 > coords = (2,2), col_test = 2
Process 3 > coords = (1,0), col_test = 0
```



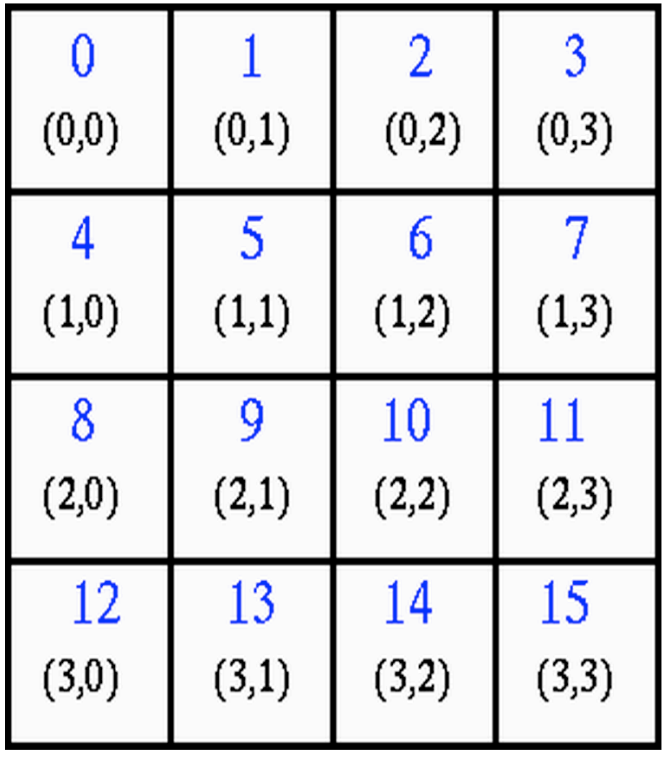
# Example: LLNL MPI_Comm_shift.c

```
/*  virt_topo.c
 *  https://computing.llnl.gov/tutorials/mpi/#Virtual_Topologies
*/
#include "mpi.h"
#include <stdio.h>
#define SIZE 16
#define UP    0
#define DOWN  1
#define LEFT  2
#define RIGHT 3

main(int argc, char *argv[])  {
int numtasks, rank, source, dest, outbuf, i, tag=1,
   inbuf[4]={MPI_PROC_NULL,MPI_PROC_NULL,
                 MPI_PROC_NULL,MPI_PROC_NULL,},
   nbrs[4], dims[2]={4,4},
   periods[2]={0,0}, reorder=0, coords[2];

MPI_Request reqs[8];
MPI_Status stats[8];
MPI_Comm cartcomm;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

if (numtasks == SIZE) {
  MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods,
                  reorder, &cartcomm);
  MPI_Comm_rank(cartcomm, &rank);
  MPI_Cart_coords(cartcomm, rank, 2, coords);
  MPI_Cart_shift(cartcomm, 0, 1, &nbrs[UP], &nbrs[DOWN]);
  MPI_Cart_shift(cartcomm, 1, 1, &nbrs[LEFT], &nbrs[RIGHT]);

  printf("P[%d]: coords= %d %d \n",
         rank,coords[0],coords[1]);
  printf("P[%d]:   neighbors(u,d,l,r)=%d %d %d %d\n",
         rank,nbrs[UP],nbrs[DOWN],nbrs[LEFT], nbrs[RIGHT]);

  outbuf = rank;

  for (i=0; i<4; i++) {
     dest = nbrs[i];
     source = nbrs[i];
     MPI_Isend(&outbuf, 1, MPI_INT, dest, tag,
               MPI_COMM_WORLD, &reqs[i]);
     MPI_Irecv(&inbuf[i], 1, MPI_INT, source, tag,
               MPI_COMM_WORLD, &reqs[i+4]);
     }

  MPI_Waitall(8, reqs, stats);

  printf("rank= %d                  inbuf(u,d,l,r)= %d %d %d %d\n",
         rank,inbuf[UP],inbuf[DOWN],inbuf[LEFT],inbuf[RIGHT]);
 }
else
  printf("Must specify %d processors. Terminating.\n",SIZE);

MPI_Finalize();
}
```

## Example: LLNL MPI_Comm_shift.c

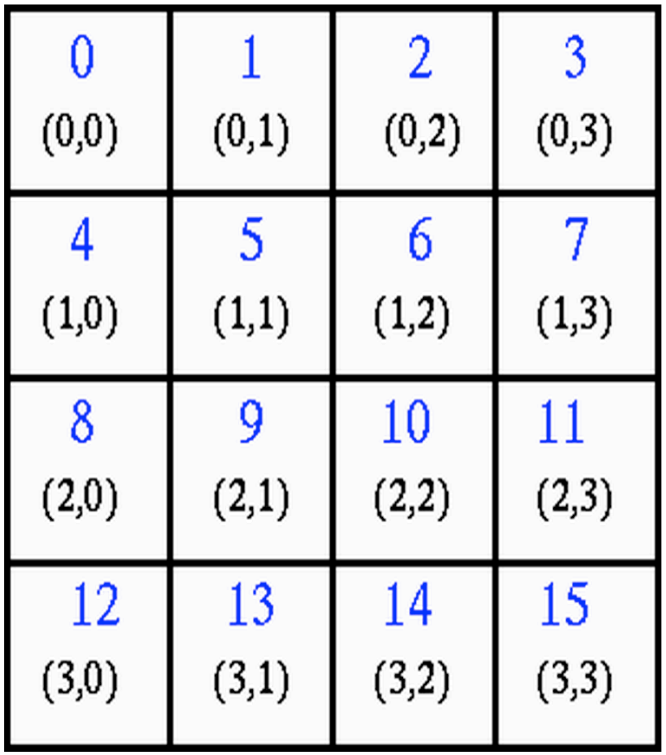
MPI will assign PE's in row-major $PE(I, J)$. If 2D, starts with J-plane, then I-plane. For indexing and (i,j) coordinates.

```
[gidget] mthomas% mpirun -np 16 ./virt_topo | sort | grep
P[0]: coords= 0 0
P[1]: coords= 0 1
P[2]: coords= 0 2
P[3]: coords= 0 3
P[4]: coords= 1 0
P[5]: coords= 1 1
P[6]: coords= 1 2
P[7]: coords= 1 3
P[8]: coords= 2 0
P[9]: coords= 2 1
P[10]: coords= 2 2
P[11]: coords= 2 3
P[12]: coords= 3 0
P[13]: coords= 3 1
P[14]: coords= 3 2
P[15]: coords= 3 3
```



# Example: LLNL MPI_Comm_shift.c

```
[gidget:comms] mthomas%mpirun -np 16 ./virt_topo

P[0]:   neighbors(u,d,l,r)=-1 4 -1 1
P[1]:   neighbors(u,d,l,r)=-1 5 0 2
P[2]:   neighbors(u,d,l,r)=-1 6 1 3
P[3]:   neighbors(u,d,l,r)=-1 7 2 -1
-----------------------------------------------
P[4]:   neighbors(u,d,l,r)=0 8 -1 5
P[5]:   neighbors(u,d,l,r)=1 9 4 6
P[6]:   neighbors(u,d,l,r)=2 10 5 7
P[7]:   neighbors(u,d,l,r)=3 11 6 -1
-----------------------------------------------
P[8]:   neighbors(u,d,l,r)=4 12 -1 9
P[9]:   neighbors(u,d,l,r)=5 13 8 10
P[10]:   neighbors(u,d,l,r)=6 14 9 11
P[11]:   neighbors(u,d,l,r)=7 15 10 -1
-----------------------------------------------
P[12]:   neighbors(u,d,l,r)=8 -1 -1 13
P[13]:   neighbors(u,d,l,r)=9 -1 12 14
P[14]:   neighbors(u,d,l,r)=10 -1 13 15
P[15]:   neighbors(u,d,l,r)=11 -1 14 -1
```



# 3D Neighbor Mapping

3D cartesian mapping for the General Curvilinear Environmental Model (GCEM).

- Map shows neighbor processors located around a processor at the center (0,0,0).
- Indexing follows C logic (row,col,depth)
- Mapping uses $(\pm i, \pm j, \pm k)$ indexing