Topologies

Outline

Introduction

Cartesian topology

Some Cartesian topology functions

Some graph topology functions

Example

Introduction

Additional information can be associated, or cached, with a communicator (i.e. not just group and context)

Topology is a mechanism for associating different addressing schemes with processes

A topology can be added to an intra-communicator, but not to inter-communicator

A topology

- can provide a convenient naming mechanism for processes

- may assist the runtime system in mapping processes onto hardware

There are virtual process topology and topology of the underlying hardware

The virtual topology can be exploited by the system in assigning of processes to processors

Two types:

- Cartesian topology
- graph topology

Cartesian topology

Process coordinates begin with 0

Row-major numbering

Example: 12 processes arranged on a 3 x 4 grid

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)

Some Cartesian topology functions

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

Creates a new communicator with Cartesian topology of arbitrary dimension

comm - old input communicator

ndims - number of dimensions of Cartesian grid

dims - array of size ndims specifying the number of processes in each dimension

periods - logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension

reorder - ranking of initial processes may be reordered (true) or not (false)

comm_cart - communicator with new Cartesian topology

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

Rank-to-coordinates translator

comm - communicator with Cartesian structure

rank - rank of a process within group of comm

maxdims - length of vector coords in the calling program

coords -array containing the Cartesian coordinates of specified process

---

```
int MPI_Cart_rank(MPI_Comm comm, int *coords , int *rank)
```

Coordinates-to-rank translator

```
int MPI_Cart_sub(MPI_Comm comm, int *free_coords , MPI Comm *newcomm)
```

Partitions a communicator into subgroups which form lower-dimensional Cartesian subgrids

comm communicator with Cartesian structure

free_coords - an array which specifies which dimensions are free (true) and which are not free (false)

Free dimensions are allowed to vary, i.e. we sum over that index to create a new communicator

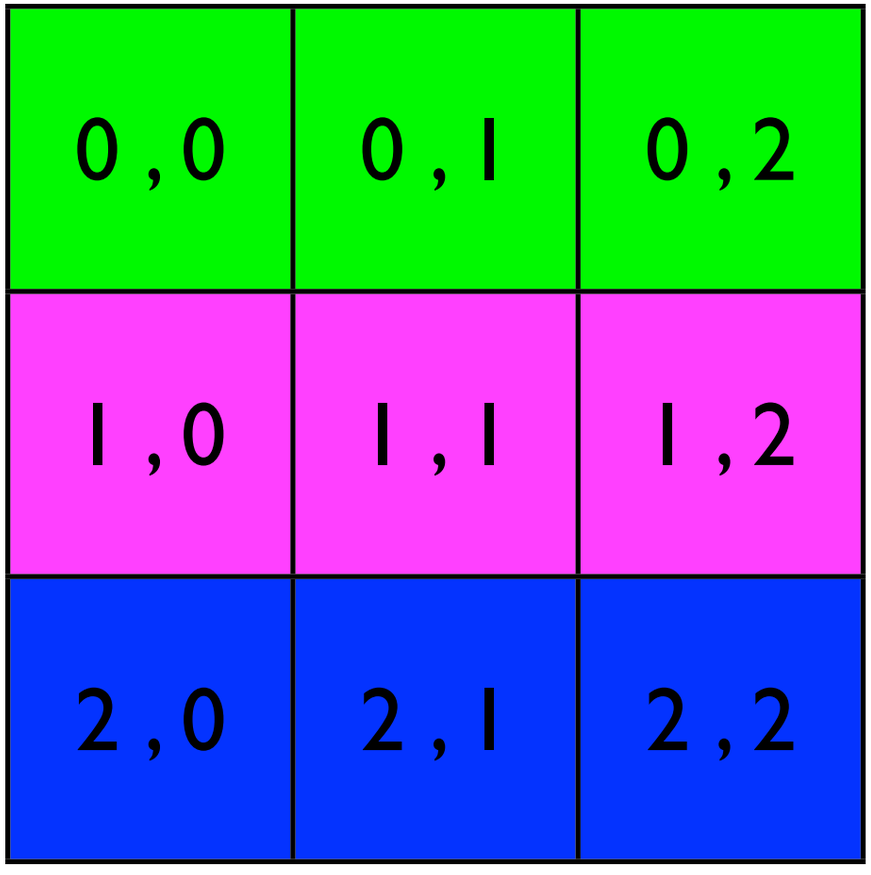
newcomm - communicator containing the subgrid that includes the calling process

In general this call creates multiple new communicators, though only one on each process

Illustration with code

```
int free_coords[2];
MPI_Comm row_comm;

free_coords[0] = 0;
free_coords[1] = 1;
MPI_Cart_sub(grid_comm, free_coords, &row_comm);
```



New communicator row_comm on processes 0.0 0.1 0.2

New communicator row_comm on processes 1.0 1.1 1.2

New communicator row_comm on processes 2.0 2.1 2.2

Example

Code adapted from P. Pacheco, PP with MPI

```
/* top_fcns.c -- test basic topology functions
 *
 * 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
 */
```

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

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

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

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);

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();
  return 0;
}
```

Sending and receiving in Cartesian topology

There is no MPI_Cart_send or MPI_Cart_recv which would allow you to send a message to process (1,0) in your Cartesian topology, for example

You must use standard communication functions

There is a convenient way to obtain the rank of the desired destination/source process from your Cartesian coordinate grid

Usually one needs to determine which are the adjacent processes in the grid and obtain their ranks in order to communicate

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

---

comm - communicator with cartesian structure

direction - coordinate dimension of shift, in range [0,n-1] for an n-dimensional Cartesian grid

displ - displacement (> 0: upwards shift, < 0: downwards shift), with periodic wraparound possible if communicator created with periodic boundary conditions turned on

Outputs are possible inputs to MPI_Sendrecv

source - rank of process to receive data from, obtained by subtracting displ from coordinate determined by direction

dest - rank of process to send data to, obtained by adding displ to coordinate determined by direction

These may be undefined (i.e. = MPI_PROC_NULL) if shift points outside grid structure and the periodic boundary conditions off

```
MPI_Cart_shift(comm, 1, 1, &source, &dest);
```

0 (0,0) 2,1	1 (0,1) 0,2	2 (0,2) 1,0
3 (1,0) 5,4	4 (1,1) 3,5	5 (1,2) 4,3
6 (2,0) 8,7	7 (2,1) 6,8	8 (2,2) 7,6

Communicator defined with periodic boundary conditions

```
MPI_Cart_shift(comm, 1, 1, &source, &dest);
```

0 (0,0) -1,1	1 (0,1) 0,2	2 (0,2) 1,-1
3 (1,0) -1,4	4 (1,1) 3,5	5 (1,2) 4,-1
6 (2,0) -1,7	7 (2,1) 6,8	8 (2,2) 7,-1

Communicator NOT defined with periodic boundary conditions

```
MPI_Cart_shift(comm, 0, -1, &source, &dest);
```

0 (0,0) 3,-1	1 (0,1) 4,-1	2 (0,2) 5,-1
3 (1,0) 6,0	4 (1,1) 7,1	5 (1,2) 8,2
6 (2,0) -1,3	7 (2,1) -1,4	8 (2,2) -1,5

Communicator NOT defined with periodic boundary conditions

Graph topology

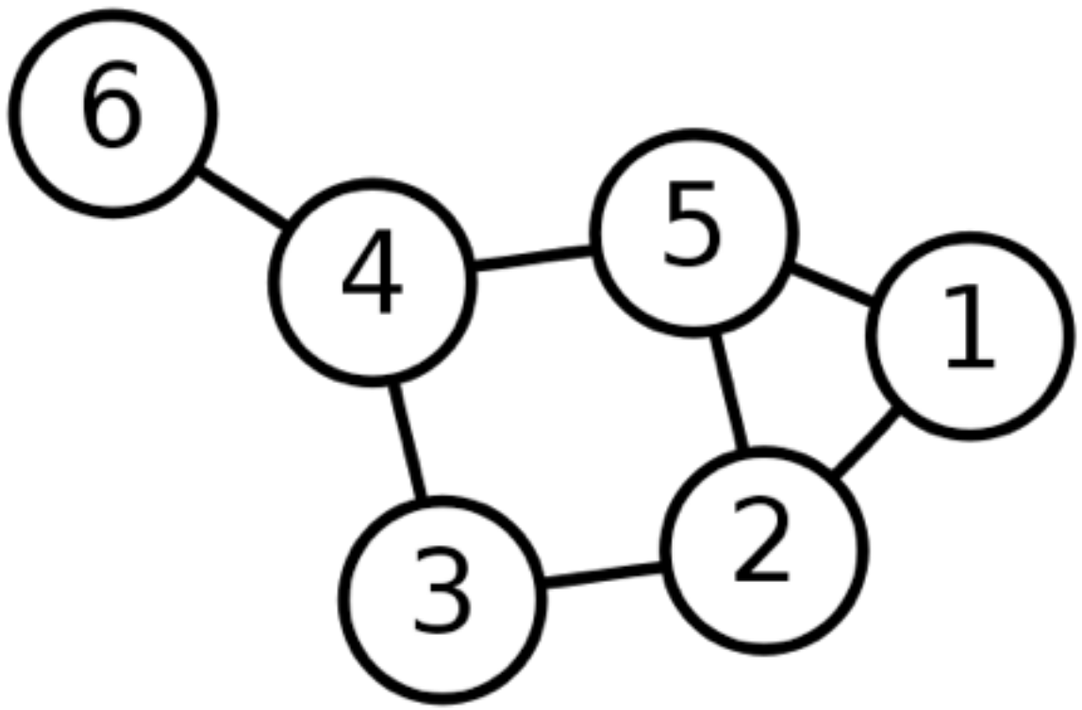
Graph - abstract representation of a set of objects (vertices, nodes, points) where some pairs are connected by links (edges)

The degree of a vertex is the number of edges that connect to it

In MPI topology, vertices usually stand for processes and edges for communications links

However, it is still possible to send a message between any two vertices, they do not have to be connected

Edges may represent optimal (fast) communications links



MPI_Graph_create

```
int MPI_Graph_create(MPI_Comm comm_old, int nnodes, int *index ,
int *edges , int reorder , MPI_Comm *comm graph)
```

Creates a communicator with a graph topology attached

comm_old - input communicator without topology

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) (logical)

comm_graph - communicator with graph topology added

The ith entry of index store the total number of neighbours of the first i graph nodes (i.e. index is cumulative)

The list of neighbours of nodes 0, 1, . . . , nnodes-1 are stored in consecutive locations in array edges (each edge counted twice since bi-directional communication assumed)

Example:

Assume 4 processes such that

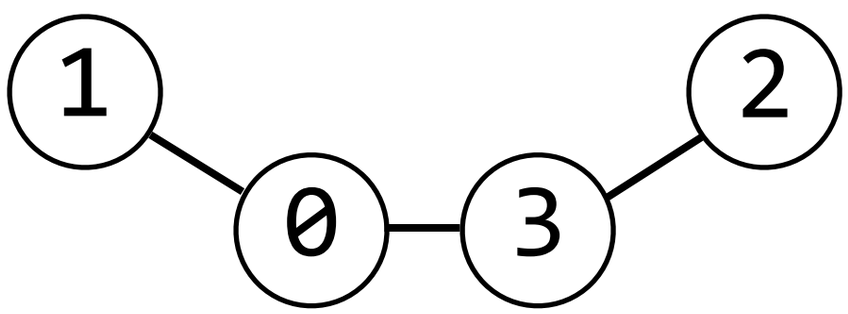
The input should be

nnodes = 4

index = (2,3,4,6)

edges = (1,3,0,3,0,2)

process	neighbours
0	1 , 3
1	0
2	3
3	0 , 2



Some graph topology functions

MPI_Graphdims_get - returns number of nodes and edges in a graph

MPI_Graph_get - returns index and edges as supplied to MPI_Graph_create

MPI_Graph - neighbours count returns the number of neighbours of a given process

MPI_Graph_neighbours - returns the edges associated with given process

MPI_Graph - map returns a graph topology recommended by the MPI system

Overall, graph topology is more general that Cartesian, but not widely used as it is less convenient for many problems

If the computational problem cannot be represented in a Cartesian grid topology, then most likely load balancing becomes a serious issue to be addressed