Topologies

Ned Nedialkov

McMaster University
Canada

SE 4F03
March 2016

Outline

Introduction

- Additional information can be associated, or *cached*, with a communicator
- *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

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

- `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 *remain_dims,
    MPI_Comm *newcomm)
```

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

- `comm` communicator with Cartesian structure
- `remain_dims` the `i`th entry of `remain_dims` specifies whether the `i`th dimension is kept in the subgrid (true) or is dropped (false) (logical vector)
- `newcomm` communicator containing the subgrid that includes the calling process

```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], coordinates[2], free_coords[2];
  int reorder = 1;
  int my_grid_rank, grid_rank;
  int       row_test, col_test;
  MPI_Comm  row_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;
}
```

Some graph topology functions

```
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 `i`th entry of `index` stores the total number of neighbors of the first `i` graph nodes
- The list of neighbors of nodes 0, 1, . . . , `nnodes`-1 are stored in consecutive locations in array `edges`

Example

Assume 4 processes such that

process	neighbors
0	1, 3
1	0
2	3
3	0, 2

The input should be

`nnodes` = 4
`index` = (2, 3, 4, 6)
`edges` = (1, 3, 0, 3, 0, 2)

- `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