Topologies and Embedding

Instructor

Dr B Krishna Priya

Topologies and Embedding

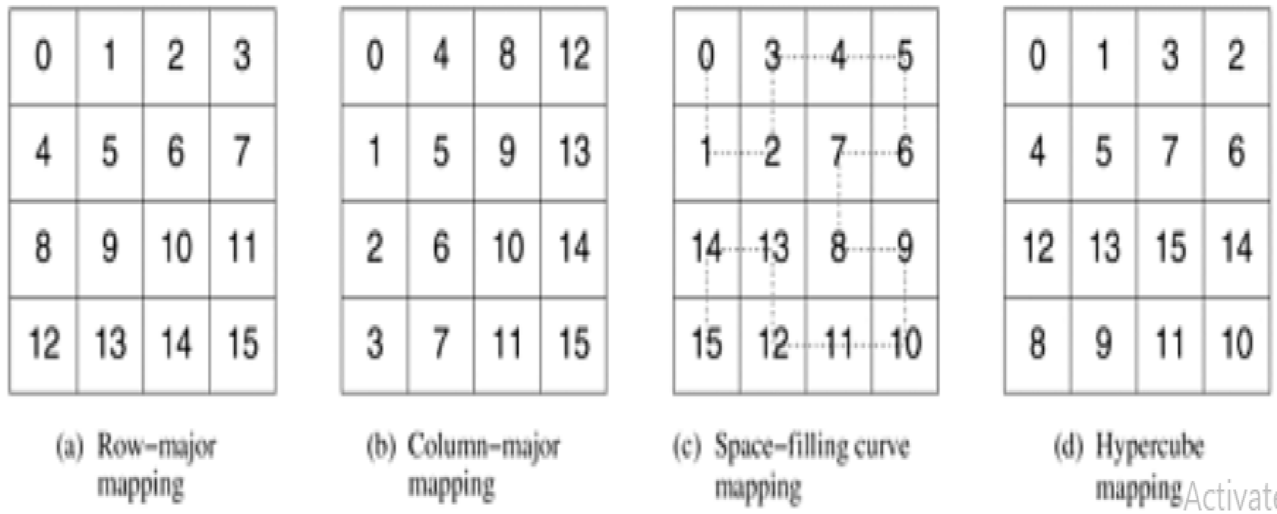
- Processes as being arranged in a one-dimensional topology and uses a linear ordering to number the processes.
- In parallel programs:
- processes are naturally arranged in higher-dimensional topologies (e.g., two- or three-dimensional)
- computation and the set of interacting processes are naturally identified by their coordinates in that topology.

Contd..

- In a parallel program in which the processes are arranged in a two-dimensional topology, process (i , j) may need to send message to (or receive message from) process (k , l).
- An MPI process with rank rank corresponds to process (row , col) in the grid such that row = rank/4 and col = rank%4

Contd..

Figure 6.5. Different ways to map a set of processes to a two-dimensional grid. (a) and (b) show a row- and column-wise mapping of these processes, (c) shows a mapping that follows a space-filling curve (dotted line), and (d) shows a mapping in which neighboring processes are directly connected in a hypercube.



Contd..

- MPI provides a set of routines that allows the programmer to arrange the processes in different topologies without having to explicitly specify how these processes are mapped onto the processors.
- It is up to the MPI library to find the most appropriate mapping that reduces the cost of sending and receiving messages.

Creating and Using Cartesian Topologies

- Graphs of processes can be used to specify any desired topology. However, most commonly used topologies in message-passing programs are one-, two-, or higher-dimensional grids, that are also referred to as Cartesian topologies

Contd..

- MPI's function for describing Cartesian topologies is called MPI_Cart_create .
- Its calling sequence is as follows.

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

- This function takes the group of processes that belong to the communicator comm_old and creates a virtual process topology.
- The topology information is attached to a new communicator comm_cart that is created by MPI_Cart_create .

Contd..

- The shape and properties of the topology are specified by the arguments ndims , dims , and periods.
- ndims: specifies the number of dimensions of the topology.
- dims: specify the size along each dimension of the topology.
- periods:specify whether or not the topology has wraparound connections.
- periods[i] is true (non-zero in C), then the topology has wraparound connections along dimension i , otherwise it does not.

Contd.

- reorder:if the processes in the new group (i.e., communicator) are to be reordered or not.
- If reorder is false, then the rank of each process in the new group is identical to its rank in the old group.
- Total number of processes specified in the dims array is smaller than the number of processes in the communicator specified by comm_old , then some processes will not be part of the Cartesian topology. For this set of processes, the value of comm_cart will be set to MPI_COMM_NULL (an MPI defined constant).
- If it is greater, it will leads to error.

Process naming

- MPI provides two functions, MPI_Cart_rank and MPI_Cart_coord , for performing coordinate-to-rank and rank-to-coordinate translations, respectively.
- The calling sequences of these routines are the following:
- int MPI_Cart_rank(MPI_Comm comm_cart, int *coords, int *rank)
- int MPI_Cart_coord(MPI_Comm comm_cart, int rank, int maxdims, int *coords)
- The MPI_Cart_rank takes the coordinates of the process as argument in the coords array and returns its rank in rank .

- The MPI_Cart_coords takes the rank of the process rank and returns its Cartesian coordinates in the array coords , of length maxdims.
- Note:maxdims should be at least as large as the number of dimensions in the Cartesian topology specified by the communicator comm_cart .

- The communication performed among processes in a Cartesian topology is that of shifting data along a dimension of the topology.
- int MPI_Cart_shift(MPI_Comm comm_cart, int dir, int s_step, int *rank_source, int *rank_dest)
- dir: direction of the shift
- s_step: size of the shift step
- The computed ranks are returned in rank_source and rank_dest.