**Virtual Topologies**

**What Are They?**

In terms of MPI, a virtual topology describes a mapping/ordering of MPI processes into a geometric “shape”.

The two main types of topologies supported by MPI are Cartesian (grid) and Graph.

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.

They must be “programmed” by the application developer.

**Why Use Them?**

**Convenience**

Virtual topologies may be useful for applications with specific communication patterns - patterns that match an MPI topology structure. For example, a Cartesian topology might prove convenient for an application that requires 4-way nearest neighbor communications for grid based data.

**Communication Efficiency**

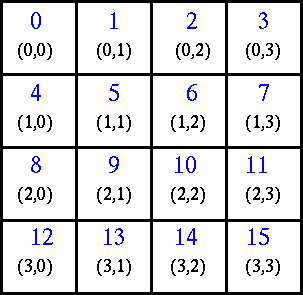
Some hardware architectures may impose penalties for communications between successively distant “nodes”.

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 totally ignored.

**Example:**

A simplified mapping of processes into a Cartesian virtual topology appears below:



**Code examples**

**Create a 4 x 4 Cartesian topology from 16 processors and have each process exchange its rank with four neighbors.**

//

#include <stdlib.h>

#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; // required variable

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);

if (numtasks == SIZE) {

// create cartesian virtual topology, get rank, coordinates, neighbor ranks

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("rank= %d coords= %d %d neighbors(u,d,l,r)= %d %d %d %d\n",

rank, coords[0], coords[1], nbrs[UP], nbrs[DOWN], nbrs[LEFT],

nbrs[RIGHT]);

outbuf = rank;

// exchange data (rank) with 4 neighbors

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();

}

Execution:

C:\Users\krish\source\repos\mproc2022>mpiexec -n 16 .\Debug\mproc2022.exe

Output::

C:\Users\krish\source\repos\mproc2022>mpiexec -n 16 .\Debug\mproc2022.exe

rank= 6 coords= 1 2 neighbors(u,d,l,r)= 2 10 5 7

rank= 6 inbuf(u,d,l,r)= 2 10 5 7

rank= 7 coords= 1 3 neighbors(u,d,l,r)= 3 11 6 -1

rank= 7 inbuf(u,d,l,r)= 3 11 6 -1

rank= 8 coords= 2 0 neighbors(u,d,l,r)= 4 12 -1 9

rank= 8 inbuf(u,d,l,r)= 4 12 -1 9

rank= 10 coords= 2 2 neighbors(u,d,l,r)= 6 14 9 11

rank= 10 inbuf(u,d,l,r)= 6 14 9 11

rank= 12 coords= 3 0 neighbors(u,d,l,r)= 8 -1 -1 13

rank= 12 inbuf(u,d,l,r)= 8 -1 -1 13

rank= 11 coords= 2 3 neighbors(u,d,l,r)= 7 15 10 -1

rank= 11 inbuf(u,d,l,r)= 7 15 10 -1

rank= 0 coords= 0 0 neighbors(u,d,l,r)= -1 4 -1 1

rank= 0 inbuf(u,d,l,r)= -1 4 -1 1

rank= 9 coords= 2 1 neighbors(u,d,l,r)= 5 13 8 10

rank= 9 inbuf(u,d,l,r)= 5 13 8 10

rank= 13 coords= 3 1 neighbors(u,d,l,r)= 9 -1 12 14

rank= 13 inbuf(u,d,l,r)= 9 -1 12 14

rank= 15 coords= 3 3 neighbors(u,d,l,r)= 11 -1 14 -1

rank= 15 inbuf(u,d,l,r)= 11 -1 14 -1

rank= 5 coords= 1 1 neighbors(u,d,l,r)= 1 9 4 6

rank= 5 inbuf(u,d,l,r)= 1 9 4 6

rank= 2 coords= 0 2 neighbors(u,d,l,r)= -1 6 1 3

rank= 2 inbuf(u,d,l,r)= -1 6 1 3

rank= 4 coords= 1 0 neighbors(u,d,l,r)= 0 8 -1 5

rank= 4 inbuf(u,d,l,r)= 0 8 -1 5

rank= 3 coords= 0 3 neighbors(u,d,l,r)= -1 7 2 -1

rank= 3 inbuf(u,d,l,r)= -1 7 2 -1

rank= 14 coords= 3 2 neighbors(u,d,l,r)= 10 -1 13 15

rank= 14 inbuf(u,d,l,r)= 10 -1 13 15

rank= 1 coords= 0 1 neighbors(u,d,l,r)= -1 5 0 2

rank= 1 inbuf(u,d,l,r)= -1 5 0 2

MPI\_Cart\_create: Explanation of “periods” parameter in Cart\_create

I kind of understand what this routine does, in the sense that it creates a Cartesian coordinate system for processors over the actual scheme, but I don't understand what the "periods" argument does.

I'm currently setting it all to 0, but I'd like to know what periods are, can't find anything online on the subject.

Here's the signature :

int MPI\_Cart\_create(

MPI\_Comm comm\_old,

int ndims,

int \*dims,

int \*periods,

int reorder,

MPI\_Comm \*comm\_cart

)

The parameter periods[n] specifies whether the nth dimension is periodic, that is whether coordinate 0 in dimension n is a neighbour of coordinate n\_max.

Some examples for visualization: If ndims == 1, then having the single dimension's period being false means the topology looks like a line of workers, whereas if you set it to true, you get a circle (ring topology).

With ndims == 2 and the periods being false, false you get a square, with true, false you get a cylinder (wrap the square sheet around one dimension) - and with true, true you get a torus (doughnut).

# Cart Shift Example

This page imported from: /afs/bu.edu/cwis/webuser/web/s/c/scv/documentation/tutorials/MPI/alliance/virtual\_topology/codes/cart\_shift\_example.html

## **MPI\_Cart\_shift Example**

In this example, we demonstrate the usage of  
MPI\_Cart\_shift. With six (6) active processes, we  
proceed to create a 2D cartesian topology for these 6 processes.  
This results in a 3×2 cartesian topology representation for the  
6 processes. Furthermore, we impose a cyclic boundary condition  
down the rows — but not the columns – of this 2D grid.  
Given the calling process rank number in the cartesian grid  
communicator, upon calling MPI\_Cart\_shift, the source and destination  
ranks of the calling process rank are returned:

1. along the rows and a displacement of +1, the source is the  
   rank above the calling rank and the destination is the rank below it.
2. along the rows and a displacement of -2, the source is two  
   ranks above the calling rank and the destination is two ranks below it.
3. across the columns and a displacement of +1, the source is the  
   rank to the left of the calling rank and the destination is the rank  
   to the right.
4. across the columns and a displacement of -1, the source is the rank  
   to the right of the calling rank and the destination is the rank  
   to the left.

|  |
| --- |
|  |

## **Output**

Note that some returned ranks in the last four columns are negative  
because they are out of bound and are assigned the value MPI\_UNDEFINED.  
The value of MPI\_UNDEFINED is implementation dependent. In this case, it  
is -1.

|  |
| --- |
| tonka:virtual\_topology/codes % mpirun -np 6 cart\_shift\_example  ( along the rows )( across columns )    2D Row Col From To From To From To From To  Rank i j Src Dest Src Dest Src Dest Src Dest  0 0 0 4 2 4 2 -1 1 1 -1  1 0 1 5 3 5 3 0 -1 -1 0  2 1 0 0 4 0 4 -1 3 3 -1  3 1 1 1 5 1 5 2 -1 -1 2  4 2 0 2 0 2 0 -1 5 5 -1  5 2 1 3 1 3 1 4 -1 -1 4 |

For your convenience in seeing what MPI\_Cart\_shift does under the four  
different sets of input parameters, shown in Figure a below is the 3×2  
cartesian topology grid.  
Here, the index pair “i,j” represent row “i” and column “j”. The number  
in parentheses represents the rank  
number associated with the cartesian coordinates.

|  |  |  |  |
| --- | --- | --- | --- |
| Figure a. periods(0)=.true.;periods(1)=.false. | | | |
|  | *-2,0 (2)* | *-1,1 (3)* |  |
|  | *-1,0 (4)* | *-1,1 (5)* |  |
| *0,-1(-1)* | *0,0 (0)* | *0,1 (1)* | *0,2(-1)* |
| *1,-1(-1)* | *1,0 (2)* | *1,1 (3)* | *1,2 (-1)* |
| *2,-1(-1)* | *2,0 (4)* | *2,1 (5)* | *2,2 (-1)* |
|  | *3,0 (0)* | *3,1 (1)* |  |
|  | *4,0 (2)* | *4,1 (3)* |  |

**Note in the program attached we have set period to 0 so there will be no wrap around**