MPI – Message Passing Interface
Communicator groups and Process Topologies

Source: http://www.netlib.org/utk/papers/mpi-book/mpi-book.html

Communicators and Groups

Communicators

- For logical division of processes
- For forming communication contexts and avoiding message conflicts
- Communicator specifies a communication domain for communications
- Can be
  - Intra – used for communicating within a single group of processes
  - Inter – used for communication between two disjoint group of processes
- Default communicators – MPI_COMM_WORLD, MPI_COMM_SELF

Groups

- An ordered set of processes.
- New group derived from base groups.
- Group represented by a communicator
- Group associated with MPI_COMM_WORLD is the first base group
- New groups can be created with Unions, intersections, difference of existing groups
- Functions provided for obtaining sizes, ranks

Communicator functions

- MPI_COMM_DUP(comm, newcomm)
- MPI_COMM_CREATE(comm, group, newcomm)

  MPI_GROUP_INCL(group, n, ranks, newgroup)

  MPI_COMM_GROUP(comm, group)
- MPI_COMM_SPLIT(comm, color, key, newcomm)

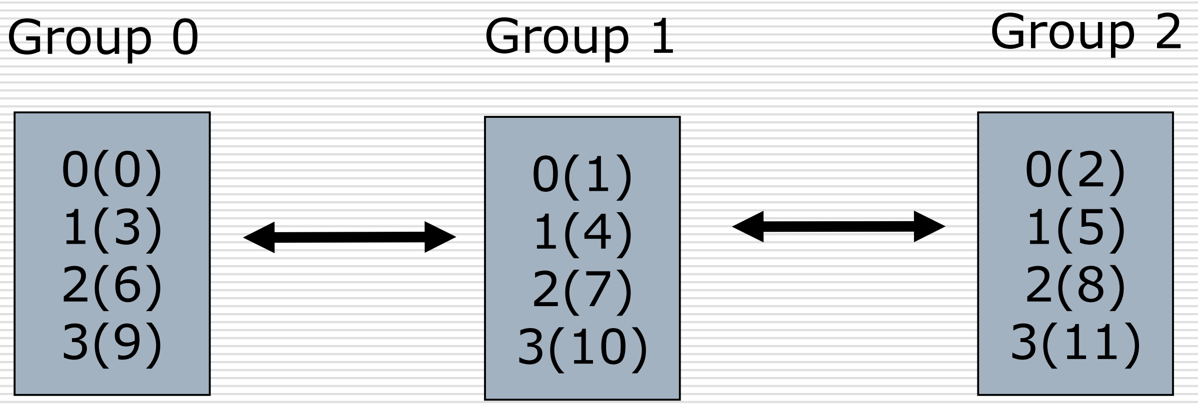
Rank	0	1	2	3	4	5	6	7	8	9
Process	A	B	C	D	E	F	G	H	I	J
Color	0	N	3	0	3	0	0	5	3	N
Key	3	1	2	5	1	1	1	2	1	0

{F,G,A,D}, {E,I,C}, {h}

Intercommunicators

- For multi-disciplinary applications, pipeline applications, easy readability of program
- Inter-communicator can be used for point-point communication (send and recv) between processes of disjoint groups
- Does not support collectives in 1.1
- MPI_INTERCOMM_CREATE(local_comm, local_leader, bridge_comm, remote_leader, tag, comm)
- MPI_INTERCOMM_MERGE(intercomm, high, newintracomm)

Communicator and Groups example




```
main(){
  membership = rank % 3;
  MPI_Comm_Split(MPI_COMM_WORLD, membership, rank, &mycomm);
```

Communicator and Groups example

```
if(membership == 0){
  MPI_Intercomm_Create(mycomm, 0, MPI_COMM_WORLD, 1, 01, my1stcomm);
}
else if(membership == 1){
  MPI_Intercomm_Create(mycomm, 0, MPI_COMM_WORLD, 0, 01, my1stcomm);
  MPI_Intercomm_Create(mycomm, 0, MPI_COMM_WORLD, 2, 12, my2ndcomm);
}
else{
  MPI_Intercomm_Create(mycomm, 0, MPI_COMM_WORLD, 1, 12, my1stcomm);
}
```

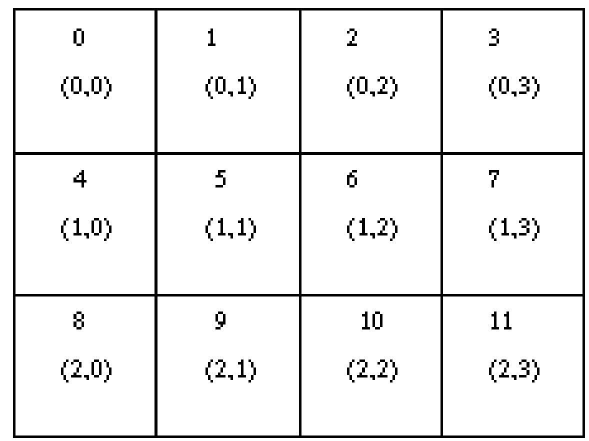
MPI Process Topologies

Motivation

- Logical process arrangement
  - For convenient identification of processes – program readability
  - For assisting runtime system in mapping processes onto hardware – increase in performance
- Default – all-to-all, ranks from 0 – n-1
- Virtual topology can give rise to trees, graphs, meshes etc.

Introduction

- Any process topology can be represented by graphs.
- MPI provides defaults for ring, mesh, torus and other common structures



Cartesian Topology

- Cartesian structures of arbitrary dimensions
- Can be periodic along any number of dimensions
- Popular cartesian structures – linear array, ring, rectangular mesh, cylinder, torus (hypercubes)

Cartesian Topology - constructors

MPI_CART_CREATE(
comm_old - old communicator,
ndims – number of dimensions,
dims - number of processes along each dimension,
periods – periodicity of the dimensions,
reorder – whether ranks may be reordered,
comm_cart – new communicator representing cartesian topology
)

Collective communication call

Cartesian Topology - Constructors

MPI_DIMS_CREATE(
nnodes(in) - number of nodes in a grid,
ndims(in) – number of dimensions,
dims(inout) - number of processes along each dimension
)

Helps to create size of dimensions such that the sizes are as close to each other as possible.

User can specify constraints by specifying +ve integers in certain entries of dims.

Only entries with 0 are modified.

dims before call	(nnodes, ndims)	dims after call
(0, 0)	(6, 2)	(3,2)
(0, 3, 0)	(6,3)	(2,3, 1)
(0, 3, 0)	(7, 3)	error

Cartesian Topology – Inquiry & Translators

- MPI_CARTDIM_GET(comm, ndims)
- MPI_CART_GET(comm, maxdims, dims, periodic, coords)
- MPI_CART_RANK(comm, coords, rank) coordinates -> rank
- MPI_CART_COORDS(comm, rank, maxdims, coords) rank->coordinates

Cartesian topology - Shifting

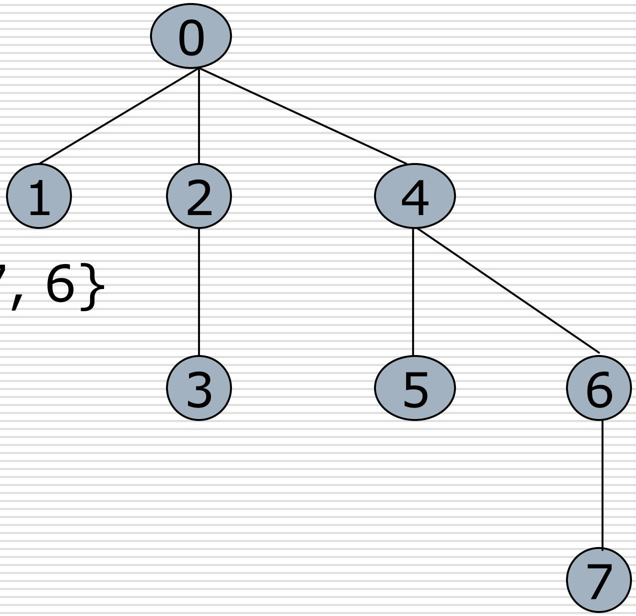
- MPI_CART_SHIFT(
comm,
direction,
displacement,
source,
dest
)
Useful for subsequent SendRecv
MPI_SendRecv(…, dest…., source)
Example:

MPI_CART_SHIFT(comm, 1, 1, &source, &dest)

General Graph Topology

MPI_GRAPH_CREATE(comm_old,
nnodes, index, edges,
reorder, comm_graph)

Example:
nnodes = 8,
index = {3, 4, 6, 7, 10, 11, 13, 14}
edges = {1, 2, 4, 0, 0, 3, 2, 0, 5, 6, 4, 4, 7, 6}



General Graph Topology - Inquiry

- MPI_Graphdims_get(MPI_Comm comm, int *nnodes, int *nedges)
- MPI_Graph_get(MPI_Comm comm, int maxindex, int maxedges, int *index, int *edges)
- MPI_Graph_neighbors_count(MPI_Comm comm, int rank, int *nneighbors)
- MPI_Graph_neighbors(MPI_Comm comm, int rank, int maxneighbors, int *neighbors)
- MPI_TOPO_TEST(comm, status)

status can be MPI_GRAPH, MPI_CART, MPI_UNDEFINED

Example: Cannon's Matrix-Matrix Multiplication

sqrt(P)

sqrt(P)

A0,0	A0,1	A0,2	A0,3
A1,0	A1,1	A1,2	A1,3
A2,0	A2,1	A2,2	A2,3
A3,0	A3,1	A3,2	A3,3

B0,0	B0,1	B0,2	B0,3
B1,0	B1,1	B1,2	B1,3
B2,0	B2,1	B2,2	B2,3
B3,0	B3,1	B3,2	B3,3

A0,0 B0,0	A0,1 B1,1	A0,2 B2,2	A0,3 B3,3
A1,1 B1,0	A1,2 B2,1	A1,3 B3,2	A1,0 B0,3
A2,2 B2,0	A2,3 B3,1	A2,0 B0,2	A2,1 B1,3
A3,3 B3,0	A3,0 B0,1	A3,1 B1,2	A3,2 B2,3

Initial Realignment

Example: Cannon’s Matrix-Matrix Multiplication

A0,1 B1,0	A0,2 B2,1	A0,3 B3,2	A0,0 B0,3
A1,2 B2,0	A1,3 B3,1	A1,0 B0,2	A1,1 B1,3
A2,3 B3,0	A2,0 B0,1	A2,1 B1,2	A2,2 B2,3
A3,0 B0,0	A3,1 B1,1	A3,2 B2,2	A3,3 B3,3

First shift

A0,2 B2,0	A0,3 B3,1	A0,0 B0,2	A0,1 B1,3
A1,3 B3,0	A1,0 B0,1	A1,1 B1,2	A1,2 B2,3
A2,0 B0,0	A2,1 B1,1	A2,2 B2,2	A2,3 B3,3
A3,1 B1,0	A3,2 B2,1	A3,3 B3,2	A3,0 B0,3

Second shift

A0,3 B3,0	A0,0 B0,1	A0,1 B1,2	A0,2 B2,3
A1,0 B0,0	A1,1 B1,1	A1,2 B2,2	A1,3 B3,3
A2,1 B1,0	A2,2 B2,1	A2,3 B3,2	A2,0 B0,3
A3,2 B2,0	A3,3 B3,1	A3,0 B0,2	A3,1 B1,3

Third shift

Cannon’s Algotihm with MPI Topologies

```
dims[0] = dims[1] = sqrt(P);
periods[0] = periods[1] = 1;

MPI_Cart_Create(comm,2,dims,periods,1,&comm_2d);
MPI_Comm_rank(comm_2d, &my2drank);
MPI_Cart_coords(comm_2d, my2drank, 2, mycoords);

MPI_Cart_shift(comm_2d, 0, -1, &rightrank, &leftrank);
MPI_Cart_shift(comm_2d, 1, -1, &downrank, &uprank);

nlocal = n/dims[0];
```

Cannon's Algotihm with MPI Topologies

```
/* Initial Matrix Alignment */
MPI_Cart_shift(comm_2d, 0, -mycoords[0], &shiftsource,
    &shiftdest);
MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
    shiftdest, 1, shiftsource, 1, comm_2d, &status);

MPI_Cart_shift(comm_2d, 1, -mycoords[1], &shiftsource,
    &shiftdest);
MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
    shiftdest, 1, shiftsource, 1, comm_2d, &status);
```

Cannon's Algotihm with MPI Topologies

```
/* Main Computation Loop */
for(i=0; i<dims[0]; i++){
  MatrixMultiply(nlocal,a,b,c); /* c=c+a*b*/

  /* Shift matrix a left by one */
  MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
     leftrank, 1, rightrank, 1, comm_2d, &status);

  /* Shift matrix b up by one */
  MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
     uprank, 1, downrank, 1, comm_2d, &status);
}
```

Cannon’s Algotihm with MPI Topologies

```
/* Restore original distribution of a and b */
MPI_Cart_shift(comm_2d, 0, +mycoords[0],
    &shiftsource, &shiftdest);
MPI_Sendrecv_replace(a, nlocal*nlocal, MPI_DOUBLE,
    shiftdest, 1, shiftsource, 1, comm_2d, &status);

MPI_Cart_shift(comm_2d, 1, +mycoords[1],
    &shiftsource, &shiftdest);
MPI_Sendrecv_replace(b, nlocal*nlocal, MPI_DOUBLE,
    shiftdest, 1, shiftsource, 1, comm_2d, &status);
```

- END