MPI Topologies
Graph Topology

Kübra ADALI    Emrah ERKAAN
As said before;

✓ The first part of MPI: Basic MPI

✓ The second part of MPI: Advanced MPI
Advanced MPI

- Contains;
  - MPI Topologies
  - Analysis of benefits
MPI Topologies

Introduction

- Provide available naming need of a process in a group of processes
- It is an attribute of processes only in the group
- Helps runtime systems in organizing processes onto processors (hardware)
- The term “Virtual Topology” gives this main idea: machine independent
- Benefits of MPI topologies:
  - Applications have specific communication patterns
  - Topologies advice plans to the program when it’s running
MPI Topologies

✓ There are two types of MPI topologies
  ➢ Cartesian Topology
  ➢ Graph Topology
Graph Topology

- What we will see about!!!
  - Introduction
  - Elements of Graph Topology
  - Important tips of Graph Topology
  - Main MPI Graph Functions
  - Example
Graph Topology

● Introduction

- Firstly, graph topology, gives opportunity to make optional connections between processes to programmers.

- We use hierarchical systems which are given by graph topology for solving weakness problem of MPI topology.

- More generally, the process organizing is described by a graph.
Graph Topology

- Elements of Graph Topology
  - Communication link
  - Nodes in the graph
  - Neighbours of per node
  - Type of mapping
Graph Topology

- Elements of Graph Topology
  - Nodes: Processors
  - Lines: Communicators between nodes
  - Arrows: Show origins and destinations of links
  - Index: Array of integers describing node degrees

<table>
<thead>
<tr>
<th>Node</th>
<th>Nneighbors</th>
<th>index</th>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0,2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1,3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>
Graph Topology

Important tips of graph topology

- Graph topology can only be used in intra-communicators.
- Number of graph nodes must not be more than number of processors.
- In a graph, communication speed may increase if process addressing reordered by system.
- One node can be neighbour of another when opposite cannot be. This means asymmetric structure can be used.
- For only IBM, Graph topologies must be symmetric. If x is neighbour of y, then y is neighbour of x.
Graph Topology

- Main MPI Graph Functions

  - **MPI_GRAPH_CREATE**:  
    ✓ creates communicator with user-defined graph topology  
    ✓ Usage:  
    ```c
    int MPI_Graph_create(MPI_Comm comm_old, int nnodes, int *index, int *edges, int reorder, MPI_Comm *comm_graph);
    ```
    ✓ Parameters
    - `comm_old`  
      - [in] input communicator without topology (handle)
    - `nnodes`  
      - [in] number of nodes in graph (integer)
    - `index`  
      - [in] array of integers describing node degrees
    - `edges`  
      - [in] array of integers describing graph edges
    - `reorder`  
      - [in] ranking may be reordered (true) or not (false) (logical)
    - `comm_graph`  
      - [out] communicator with graph topology added (handle)
Graph Topology

- Main MPI Graph Functions

  - **MPI_Graph_create Usage Example**

```
#include "mpi.h"

MPI_Comm graph_comm;

int nnodes = 4; /* number of nodes */
int index[4] = {1, 3, 5, 6}; /* index definition */
int edges[6] = {1, 0, 2, 1, 3, 2}; /* edges definition */
int reorder = 1; /* allows processes reordered for efficiency */

MPI_Graph_create(MPI_COMM_WORLD, nnodes, index, edges, reorder, graph_comm);
```
Graph Topology

- Main MPI Graph Functions

- **MPI_GRAPH_NEIGHBORS_COUNT**

  - Returns the number of neighbors of a node associated with a graph topology
  - **Usage:**

    ```c
    int MPI_Graph_neighbors_count(MPI_Comm comm, int rank, int *nneighbors);
    ```

  - **Parameters:**
    - `comm` [in] communicator with graph topology (handle)
    - `rank` [in] rank of process in group of comm (integer)
    - `nneighbors` [out] number of neighbors of specified process (integer)
Graph Topology

- Main MPI Graph Functions
  - MPI_GRAPH_NEIGHBORS

✓ Returns the neighbors of a node associated with a graph topology
✓ Usage:
  ```c
  int MPI_Graph_neighbors( MPI_Comm comm, int rank, int maxneighbors, int *neighbors );
  ```
✓ Parameters
  - `comm`
    - [in] communicator with graph topology (handle)
  - `rank`
    - [in] rank of process in group of comm (integer)
  - `maxneighbors`
    - [in] size of array neighbors (integer)
  - `neighbors`
    - [out] ranks of processes that are neighbors to specified process (array of integer)
Graph Topology

- Main MPI Graph Functions

- MPI_Graph_neighbors_count, MPI_Graph_neighbors

C:

```c
int node, my_neighbors, my_edges[2];
.. ..
MPI_Comm_rank(graph_comm, &node);
.. ..
MPI_Graph_neighbors_count(graph_comm, node, &my_neighbors);
..
..
MPI_Graph_neighbors(graph_comm, node, Nneighbors, my_edges);
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Nneighbors</th>
<th>index</th>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>.1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>0,2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1,3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>6</td>
<td>2</td>
</tr>
</tbody>
</table>

Input node=2  Output my_neighbors=2 my_edges={1,3}
Graph Topology

- Main MPI Graph Functions

  - **MPI_GRAPH_GET:**

    - Retrieves graph topology information associated with a communicator
    - **Usage:**

      ```
      int MPI_Graph_get(MPI_Comm comm, int maxindex, int maxedges, int *index, int *edges);
      ```

    - **Parameters**
      - `comm` [in] communicator with graph structure (handle)
      - `maxindex` [in] length of vector index in the calling program (integer)
      - `maxedges` [in] length of vector edges in the calling program (integer)
      - `index` [out] array of integers containing the graph structure
      - `edges` [out] array of integers containing the graph structure
Graph Topology

- Main MPI Graph Functions

- **MPI_GRAPHDIMS_GET**:
  - Retrieves graph topology information associated with a communicator
  - **Usage**:
    ```c
    int MPI_Graphdims_get( MPI_Comm comm, int *nnodes, int *nedges );
    ```
  - **Parameters**
    - `comm`
      - [in] communicator for group with graph structure (handle)
    - `nnodes`
      - [out] number of nodes in graph (integer)
    - `nedges`
      - [out] number of edges in graph (integer)
Graph Topology

- Main MPI Graph Functions

C:
int nnodes, nedges, index[4], edges[6];
.. 
.. 
MPI_Graphdims_get(graph_comm, &nnodes, &nedges);
MPI_Graph_get(graph_comm, nnodes, nedges, index, edges);

Output

nnodes=4
nedges=6
index={1,3,5,6}
edges={1,0,2,1,3,2}
Graph Topology

Main MPI Graph Functions

- **MPI_TOPO_TEST**:  
  ✓ Determines the type of topology (if any) associated with a communicator  
  ✓ Usage:  
    ```c
    int MPI_Topo_test( MPI_Comm comm, int *topo_type );
    ```  
  ✓ Parameters
    - *comm*
      - [in] communicator (handle)
    - *topo_type*
      - [out] topology type of communicator comm (integer).
      - If the communicator has no associated topology, returns MPI_UNDEFINED.
Graph Topology

Example:

```c
#include "stdafx.h"
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main( int argc, char *argv[] )
{
    int errs = 0, i, k, neighbourNumber,j;
    int wsize = 5;
    int topo_type;
    int *index, *edges, *outindex, *outedges,*neighbours;
    MPI_Comm comm1, comm2;
    MPI_Init( &argc, &argv );  //preparation of environment of MPI
    MPI_Comm_size( MPI_COMM_WORLD, &wsize );  // Get the number of Processors
```
Graph Topology

• Example:

```c
if (wsize >= 3) {  // If Processor number is more than 3 we can make a graph.
    index = (int*)malloc(wsize * sizeof(int));
    edges = (int*)malloc(wsize * 2 * sizeof(int));
    // allocate memory for arrays

    if (!index || !edges) {
        printf("Unable to allocate %d words for index or edges\n", 3 * wsize);
        //Error Control if we cannot allocate memory
        fflush(stdout);//bufferı boşaltır
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
```
Graph Topology

- Example:

```plaintext
index[0] = 2; // We are filling index values of the graph
index[1] = 5;
index[2] = 6;
index[3] = 8;
index[4] = 10;

edges[0] = 1; // We are filling edge values of the graph
edges[1] = 4;
edges[2] = 0;
edges[3] = 2;
edges[4] = 3;
edges[5] = 1;
edges[6] = 1;
edges[7] = 4;
edges[8] = 0;
edges[9] = 3;
```

<table>
<thead>
<tr>
<th>Node</th>
<th>Nneighbors</th>
<th>index</th>
<th>edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1.4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>0.2,3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>1.4</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>10</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Graph Topology

- Example:

MPI(Graph_create( MPI_COMM_WORLD, wsize, index, edges, 0, &comm1 );
// We are creating our graph
// MPI_COMM_WORLD is the communicators group we are going to use.
// wsize is the number of processors
// index and edges are the arrays that we are creating our graphs with.
// 0 used if we don’t want to order processes in the group.
// comm1 is the communicator which represents the graph.

MPI_Comm_dup( comm1, &comm2 );
// We duplicated our graph.

MPI_Topo_test( comm2, &topo_type );
// Get the type of Toplogy we are using.

printf( "The Topology Type of Graphs is %s" , &topo_type);
Graph Topology

- Example:

if (topo_type != MPI_GRAPH) { // If Topology type is not graph stop process.
    errs++;
    printf( "Topo type of duped graph was not graph\n" );
    fflush(stdout);
}
else { // If Topology type is graph continue our program
    int nnodes, nedges;
    MPI_Graphdims_get( comm2, &nnodes, &nedges );
    // With using Graphdims we are getting dimensions of index array and edge array.
    if (nnodes != wsize) {
        // And we are controlling if Node number obtained from graphdims same with the number
        // of processors
        errs++;
        printf( "Nnodes = %d, should be %d\n", nnodes, wsize );
        fflush(stdout);
    }
    if (nedges != 2*wsize) {
        errs++;
        printf( "Nedges = %d, should be %d\n", nedges, 2*wsize );
        fflush(stdout);
    }
}
Graph Topology

- **Example:**

  // We are going to obtain arrays that we created graph with. We will use Graphget functions.
  
  ```c
  outindex = (int*)malloc(wsize * sizeof(int)); // allocate memory for arrays
  outedges = (int*)malloc(wsize * 2 * sizeof(int)); // allocate memory for arrays
  
  MPI_Graph_get(comm2, wsize, 2*wsize, outindex, outedges);
  // Comm2 is the Communicator we will get arrays from.
  // wsize and 2*wsize are the lengths of arrays.
  // outindex and outedges are the arrays to write graph information.
  for (i=0; i<wsize; i++) { // We are controlling arrays we obtained with Graph_get if they are same with input
    if (index[i] != outindex[i]) {
      printf("%d = index[%d] != outindex[%d] = %d\n", index[i], i, i, outindex[i]);
      fflush(stdout);
      errs++;
    }
  }
  for (i=0; i<2*wsize; i++) {
    if (edges[i] != outedges[i]) {
      printf("%d = edges[%d] != outedges[%d] = %d\n", edges[i], i, i, outedges[i]);
      fflush(stdout);
      errs++;
    }
  }
  ```
### Graph Topology

- **Example:**

```c
printf( "\n The node count of graph that obtained with MPI_Graphdims_get Function : " );
printf( "%d", nnodes );
printf( "\n Edge count of graph that obtained with MPI_Graphdims_get Function : " );
printf( "%d", nedges );
printf( "\n-------------------------------------\n" );

printf( "Array of indexes that obtained with MPI_Graph_get Function : " );
// We are printing arrays obtained with Graph_Get function.
for (i=0;i<wsize;i++)
{
    printf( "%d ,", outindex[i] );
}
printf( "\nArray of Edges that obtained with MPI_Graph_get Function :" );
for ( i=0;i<wsize;i++)
{
    printf( "%d ,", outedges[i] );
}

free( outindex );//returns the memory which was allocated for outindex to system.
free( outedges ); //returns the memory which was allocated for outedges to system.

printf( "\n-------------------------------------\n" );
```
Graph Topology

Example:

```c
for(i=0;i<wsize;i++) // We are going to print each Nodes and their
// neighbours with using arrays that we obtained.
{
    int temp;
    if(i==0)
        temp=0;
    else
        temp=index[i-1];

    neighbourNumber=index[i]-temp; //Get each node's neighbour
    // number.
    printf("My node no is = %d and I have %d neighbours", i,neighbourNumber);
    printf("My neighbours are : ");
    for( j=temp; j<index[i];j++)
    {
        printf("%s",edges[j]);
    }
    printf("n");
}
```
Graph Topology

Example:

```c
printf( "With Using MPI Commands"");
printf( "\n-------------------------------------\n");

for( k=0;k<wsize;k++)
{
    MPI_Graph_neighbors_count(comm2,k,&neighbourNumber);
    //comm2 is the communicator we get graph’s info.
    //k is the node number.
    // neighbourNumber is number of neighbour of “k”;
    MPI_Graph_neighbors(comm2,k,neighbourNumber,&neighbours);
    //k is the node number.
    // neighbourNumber is number of neighbour of “k”.
    // neighbour is the array neighbours of k will be write.
    printf( "My node no is = %d and I have %d neighbours\n", k,neighbourNumber);
    printf( "My neighbours are : ");
    for(i=0;i<neighbourNumber;i++)
    {
        printf("%s ",neighbours[i]);
    }
}
```
Graph Topology

Example:

free( index ); //return the allocated memory to the system.
free( edges ); // return the allocated memory to the system.
MPI_Comm_free( &comm2 ); // Empty comm2 and give to system.
MPI_Comm_free( &comm1 ); //Empty comm1 and give to system.
}

MPI_Finalize(); //Finish MPI
return 0;
}
Graph Topology

● Example:

The Topology Type of Graphs is MPI_GRAPH

The node count of graph that obtained with MPI_Graphdims_get Function : 5
Edge count of graph that obtained with MPI_Graphdims_get Function :10

Array of indexes that obtained with MPI_Graph_get Function : 2,5,6,8,10,
Array of Edges that obtained with MPI_Graph_get Function : 1,4,0,2,3,1,1,4,0,3,

My node no is 0 and I have 2 neighbours
My neighbours are : 1,4.

My node no is 1 and I have 3 neighbours
My neighbours are : 0,2,3.

My node no is 2 and I have 1 neighbours
My neighbours are : 1.

My node no is 3 and I have 2 neighbours
My neighbours are : 1,4.

My node no is 4 and I have 2 neighbours
My neighbours are : 0,3.

With Using MPI Commands

My node no is 0 and I have 2 neighbours
My neighbours are : 1,4.

My node no is 1 and I have 3 neighbours
My neighbours are : 0,2,3.

My node no is 2 and I have 1 neighbours
My neighbours are : 1.

My node no is 3 and I have 2 neighbours
My neighbours are : 1,4.

My node no is 4 and I have 2 neighbours
My neighbours are : 0,3.
Questions????????
Bibliography

- http://mpi.deino.net/mpi_functions/MPI_Graph_create.html
- http://parallel.ru/docs/Parallel/mpi1.1/node136.html
- http://larshj.dk/morphy/javadoc/mpi/Graphcomm.html
- http://www.mhpcc.edu/training/workshop/mpi/MAIN.html#Virtual_Topologies