Parallel Processing

Lecture 5 - MPI cont.

Felicja Okulicka-Dłużewska

F.Okulicka@mini.pw.edu.pl

Warsaw University of Technology

Faculty of Mathematics and Information Science

Go to full-screen mode now by hitting CTRL-L

Collective communication

- Collective communication actions over a communicator.
- All processes must communicate.
- Synchronization may or may not occur.
- All collective operations are **blocking**.
- There in **no tags**.
- Receive buffers must be exactly the right size.

Collective communication

The communication is involving a group of processes and called by all processes in a communicator:

- Barrier synchronization,
- Broadcast, scatter, gather,
- Global sum, global maximum etc.

Barrier synchronization

Argument of MPI_BARRIER(comm):

IN comm communicator (handle)

C	int MPI_Barrier(MPI_Comm comm)
C++ (in the MPI:: namespace)	void Intracomm::Barrier() const
FORTRAN	MPI_BARRIER(COMM, IERROR) INTEGER COMM, IERROR

Broadcast

Arguments :

INOUT	buffer	starting address of buffer (choice)
IN	count	number of entries in buffer (integer)
IN	datatype	data type of buffer (handle)
IN	root	rank of broadcast root (integer)
IN	comm	communicator (handle)

Broadcast

C	int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
C++ (in the MPI:: namespace)	void Intracomm::Bcast(void* buffer, int cou const Datatype& datatype, int root) const
FORTRAN	MPI_BCAST(BUFFER, COUNT, DATATYP ROOT, COMM, IERROR) <type> BUFFER(*) INTEGER COUNT, DATATYPE, ROOT, COMM,IERROR

Example

Broadcast 100 integers from process 0 to every process in the group.

```
MPI_Comm comm;

    int array[100];
    int root=0;
    ...
    MPI_Bcast( array, 100, MPI_INT, root, comm);
```

Scatter

Distributes distinct messages from a single source task to each task in the group. Arguments of MPI_Scatter:

IN	sendbuf	address of send buffer (choice, significant only at r
IN	sendcount	number of elements sent to each process (integer, significant only at root)
IN	sendtype	data type of send buffer elements (significant only at root) (handle)
OUT	recvbuf	address of receive buffer (choice)
IN	recvcount	number of elements in receive buffer (integer)
IN	recvtype	data type of receive buffer elements (handle)
IN	root	rank of sending process (integer)
IN	comm	communicator (handle)

Scatter

C	int MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
C++ (in the MPI:: namespace)	void Intracomm::Scatter(const void* sendbuf, int sendcount, const Datatype& sendtype, void* recvbuf, int recvcount, const Datatype& recvtype, int root) const

Scatter

FORTRAN	MPI_SCATTER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR) <type> SENDBUF(*), RECVBUF(*) INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

Example of Scatter

Scatter sets of 100 ints from the root to each process in the group:

```
MPI_Comm comm;
        int gsize,*sendbuf;
        int root, rbuf[100];
        ...
        MPI_Comm_size( comm, &gsize);
        sendbuf = (int *)malloc(gsize*100*sizeof(int));
        ...
        MPI_Scatter( sendbuf, 100, MPI_INT, rbuf, 100,
        MPI_INT, root, comm);
```

Example 1 of Scatter

```
#include "mpi.h"
#include <stdio.h>
#define SIZE 4
int main(argc,argv)
int argc;
char *argv[]; {
int numtasks, rank, sendcount, recvcount, source;
float sendbuf[SIZE][SIZE] = {
{1.0, 2.0, 3.0, 4.0},
{5.0, 6.0, 7.0, 8.0},
{9.0, 10.0, 11.0, 12.0},
{13.0, 14.0, 15.0, 16.0} };
float recvbuf[SIZE];
```

Example 1 of Scatter

```
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks == SIZE) {
source = 1;
sendcount = SIZE;
recvcount = SIZE;
MPI_Scatter(sendbuf,sendcount,MPI_FLOAT,recvbuf,recvcount,
            MPI_FLOAT,source,MPI_COMM_WORLD);
printf("rank= %d Results: %f %f %f %f\n",rank,recvbuf[0],
            recvbuf[1],recvbuf[2],recvbuf[3]);}
else
printf("Must specify %d processors. Terminating.\n",SIZE);
MPI_Finalize();}
```

Example 1 of Scatter

Sample program output:

rank= 0 Results: 1.000000 2.000000 3.000000 4.000000

rank= 1 Results: 5.000000 6.000000 7.000000 8.000000

rank= 2 Results: 9.000000 10.000000 11.000000 12.000000

rank= 3 Results: 13.000000 14.000000 15.000000 16.000000

Gather

Gathers distinct messages from each task in the group to a single destination task. This routine is the reverse operation of MPI_Scatter.

Gather

Arguments of MPI_Gather:

IN	sendbuf	starting address of send buffer (choice)
IN	sendcount	number of elements in send buffer (integer)
IN	sendtype	data type of send buffer elements (handle)
OUT	recvbuf	address of receive buffer (choice, significant only at root)
IN	recvcount	number of elements for any single receive (integer, significant only at root)
IN	recvtype	data type of recv buffer elements (significant only at root) (handle)
IN	root	rank of receiving process (integer)
IN	comm	communicator (handle)

Gather

C	int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
C++ (in the MPI:: namespace)	void Intracomm::Gather(const void* sendbuf, int sendcount, const Datatype& sendtype, void* recvbuf, int recvcount, const Datatype& recvtype, int root) const

Gather

FORTRAN	MPI_GATHER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR) <type> SENDBUF(*), RECVBUF(*) INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

example

Gather 100 ints from every process in group to root.

```
MPI_Comm comm;
        int gsize,sendarray[100];
        int root, *rbuf;
        ...
        MPI_Comm_size( comm, &gsize);
        rbuf = (int *)malloc(gsize*100*sizeof(int));
        MPI_Gather( sendarray, 100, MPI_INT, rbuf, 100,
        MPI_INT, root, comm);
```

MPI_Reduce

Used to compute a result involving data distributed over a group of processes, for example:

- Global sum or product,
- Global maximum or minimum,
- Global user defined operation.

MPI_Reduce

Arguments of MPI_Reduce:

IN	sendbuf	address of send buffer (choice)
OUT	recvbuf	address of receive buffer (choice, significant only at
IN	count	number of elements in send buffer (integer)
IN	datatype	data type of elements of send buffer (handle)
IN	op	reduce operation (handle)
IN	root	rank of root process (integer)
IN	comm	communicator (handle)

MPI_Reduce

C	int MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
C++ (in the MPI:: namespace)	void Intracomm::Reduce(const void* sendbuf, void* recvbuf, int count, const Datatype& datatype, const Op& op, int root) const

MPI_Reduce

FORTRAN	MPI_REDUCE(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, ROOT, COMM, IERROR) <type> SENDBUF(*), RECVBUF(*) INTEGER COUNT, DATATYPE, OP, ROOT, COMM, IERROR

Example of MPI_Reduce

Integer global sum - the sum of all the x values is placed in result. The result is only placed there on processor 0.

MPI_Reduce (&x, &result, 1, MPI_INT, MPI_SUM, 0 , MPI_COMM_WORLD)

Global reduction operation is used to compute a result involving data distributed over a group of processes. There is a set of predefined Reduction Operations. User-defined reduce is possible.

Predefined Reduction Operations

MPI Name	Function
MPI_MAX / MPI_MIN	Maximum / Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC / MPI_MINLOC	Maximum / Minimum and location

Variants of MPI_REDUCE

- MPI_ALLREDUCE - no root process
- MPI_REDUCE_SCATTER - result is scattered
- MPI_SCAN - 'parallel prefix'

Collective operations must be executed in an order so that no cyclic dependencies occur.

Example

The following is erroneous.

```
switch(rank) {
  case 0:
    MPI_Bcast(buf1, count, type, 0, comm);
    MPI_Send(buf2, count, type, 1, tag, comm);
    break;
  case 1:
    MPI_Recv(buf2, count, type, 0, tag, comm, status);
    MPI_Bcast(buf1, count, type, 0, comm);
    break;
}
```

Timers

Time is measured in seconds by calling MPI_Wtime or MPI_Wtick:

C	int double MPI_Wtime(void)
C++ (in the MPI:: namespace)	double Wtime()
FORTRAN	DOUBLE PRECISION MPI_WTIME(IERROR)

The ticks can be get by MPI_Wtict function.

C	int double MPI_Wtick(void)
C++ (in the MPI:: namespace)	double Wtick()
FORTRAN	DOUBLE PRECISION MPI_WTICK(IERROR)

Datatypes in MPI

A message contains a number of elements of some particular datatypes. MPI datatypes are:

- basic types
- derived types: vectors, structs.

Derived types can be built up from basic types. C types are different from Fortran types.

Basic MPI datatypes

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	Signed short int
MPI_INT	Signed int
MPI_LONG	Signed long int
MPI_UNSIGNED_CHAR	Unsigned char
MPI_UNSIGNED_SHORT	Unsigned short int

Basic MPI datatypes

MPI datatype	C datatype
MPI_UNSIGNED	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG DOUBLE	Long double
MPI_BYTE	
MPI_PACKED	

Derived data in MPI

- Contiguous Data,
- Vector Datatype,
- Extend of Datatype,
- Struct Datatype.

Contiguous Data

The simplest derived datatype consists of a number of contiguous item of the same datatype.

C	int MPI_Type_contiguous(int count, MPI_Datatype oldtype, MPI_Datatype newtype)
C++ (in the MPI:: namespace)	Datatype Datatype::Create_contiguous(int count) const
FORTRAN	MPI_TYPE_CONTIGUOUS(COUNT, OLDTYPE, NEWTYPE, IERROR) INTEGER COUNT, OLDTYPE, NEWTYPE, IERROR

Contiguous Data

Arguments :

IN	count	replication count (nonnegative integer)
IN	oldtype	old datatype (handle)
OUT	newtype	new datatype (handle)

Vector Datatype

C	int MPI_Type_vector(int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype)
C++ (in the MPI:: namespace)	Datatype Datatype::Create_vector(int count, int blocklength, int stride) const
FORTRAN	MPI_TYPE_VECTOR(COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR) INTEGER COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR

Vector Datatype

Vector Datatype allows for regular gaps (stride) in the displacements.

Arguments :

IN	count	number of blocks (nonnegative integer)
IN	blocklength	number of elements in each block (nonnegative integer)
IN	stride	number of elements between start of each block (integer)
IN	oldtype	old datatype (handle)
OUT	newtype	new datatype (handle)

Example

Count =2 (2 vectors - blocks)

Stride = 5 (5 element stride between blocks)

Blocklength = 3 (3 element per block)

Extend of Datatype

Returns the size in bytes of the specified data type. Useful for the MPI subroutines that require specification of offsets in bytes.

Arguments :

IN	datatype	datatype (handle)
OUT	extent	datatype extent (integer)

Extend of Datatype

C	int MPI_Type_extent(MPI_Datatype datatype, MPI_Aint *extent) Note: This function is deprecated.
C++ (in the MPI:: namespace)	void MPI::Datatype::Get_extent(MPI::Aint& lb, MPI::Aint& extent) const
FORTRAN	MPI_TYPE_EXTENT(DATATYPE, EXTENT, IERROR) INTEGER DATATYPE, EXTENT, IERROR Note: This function is deprecated.

Struct Datatype

The new data type is formed according to completely defined map of the component data types.

Example

The arguments can be followings:

Count =2

array_of_blocklength[0] = 1

array_of_types [0] = MPI_INT

array_of_blocklength[1] = 3

array_of_types [1] = MPI_DOUBLE

Struct Datatype

C	int MPI_Type_struct(int count, int *array_of_blocklengths, MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types, MPI_Datatype *newtype) Note: This function is deprecated
C++ (in the MPI:: namespace)	static MPI::Datatype MPI::Datatype:: Create_struct(int count, const int array_of_blocklengths[], const MPI::Aint array_of_displacements[], const MPI::Datatype array_of_types[])

Struct Datatype

FORTRAN	MPI_TYPE_STRUCT(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS, ARRAY_OF_TYPES, NEWTYPE, IERROR) INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*), ARRAY_OF_TYPES(*), NEWTYPE, IERROR Note: This function is deprecated

Struct Datatype

Arguments :

IN	count	number of blocks (integer) - also number of entries in arrays array_of_types, array_of_displacements and array_of_blocklengths
IN	array_of_blocklength	number of elements in each block (array of integer)
IN	array_of_displacements	byte displacement of each block (array of integer)

Struct Datatype

Arguments :

IN	array_of_types	type of elements in each block
		(array of handles to datatype objects)
OUT	newtype	new datatype (handle)

Struct Datatype - Example

```
#include "mpi.h" #include <stdio.h> #define NELEM 25
int main(argc,argv)
int argc;
char *argv[]; {
int numtasks, rank, source=0, dest, tag=1, i;
typedef struct {
        float x, y, z;
        float velocity;
        int n, type;
} Particle;
Particle p[NELEM], particles[NELEM];
MPI_Datatype particletype, oldtypes[2];
int blockcounts[2];
```

Struct Datatype - Example

```
/* MPI_Aint type used to be consistent with syntax of */
/* MPI_Type_extent routine */
MPI_Aint offsets[2], extent;
MPI_Status stat;
MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
/* Setup description of the 4 MPI_FLOAT fields x, y, z, velocity */
offsets[0] = 0;
oldtypes[0] = MPI_FLOAT;
blockcounts[0] = 4;
```

Struct Datatype - Example

```
/* Setup description of the 2 MPI_INT fields n, type */
/* Need to first figure offset by getting size of MPI_FLOAT */
MPI_Type_extent(MPI_FLOAT, &extent);
offsets[1] = 4 * extent;
oldtypes[1] = MPI_INT;
blockcounts[1] = 2;
/* Now define structured type and commit it */
MPI_Type_struct(2, blockcounts, offsets, oldtypes, &particletype);
MPI_Type_commit(&particletype);
```

Struct Datatype - Example

```
/* Initialize the particle array and then send it to each task */
if (rank == 0) {
for (i=0; i<NELEM; i++) {
        particles[i].x = i * 1.0;
        particles[i].y = i * -1.0;
        particles[i].z = i * 1.0;
        particles[i].velocity = 0.25;
        particles[i].n = i;
        particles[i].type = i % 2;
}
for (i=0; i<numtasks; i++)
        MPI_Send(particles, NELEM, particletype, i, tag,
        MPI_COMM_WORLD);
}
```

Struct Datatype - Example

```
MPI_Recv(p, NELEM, particletype, source, tag,
MPI_COMM_WORLD,&stat);
/* Print a sample of what was received */
printf("rank= %d %3.2f %3.2f %3.2f %3.2f %d %d\n", rank,p[3].x,
p[3].y,p[3].z,p[3].velocity,p[3].n,p[3].type);
MPI_Type_free(&particletype);
MPI_Finalize();
}
```

Sample program output:

```
rank= 0 3.00 -3.00 3.00 0.25 3 1
rank= 2 3.00 -3.00 3.00 0.25 3 1
rank= 1 3.00 -3.00 3.00 0.25 3 1
rank= 3 3.00 -3.00 3.00 0.25 3 1
```

Commiting

Commits new datatype to the system. Required for all user constructed (derived) datatypes.

C	int MPI_Type_commit(MPI_Datatype *datatype)
C++ (in the MPI:: namespace)	void Datatype::Commit()
FORTRAN	MPI_TYPE_COMMIT(DATATYPE, IERROR) INTEGER DATATYPE, IERROR

Arguments :

INOUT datatype datatype that is committed (handle)

Deallocation of data type

MPI_Type_free deallocates the specified datatype object. Use of this routine is especially important to prevent memory exhaustion if many datatype objects are created, as in a loop.

C	int MPI_Type_free(MPI_Datatype *datatype)
C++ (in the MPI:: namespace)	void Datatype::Free()
FORTRAN	MPI_TYPE_FREE(DATATYPE, IERROR) INTEGER DATATYPE, IERROR

Arguments :

INOUT datatype datatype that is freed (handle)

Virtual topologies

Virtual topology can allow MPI to optimize communications by creating scheme fitting the communication pattern. there may be no relation between the physical structure of the parallel machine and the process topology. The creating a topology produces a new communicator with new interior ranks. Topology types are graph topologies and Cartesian topologies.

Cartesian topologies

Each process is 'connected' to its neighbors in a virtual grid. Boundaries can be cyclic , or not. Processes are identified by the Cartesian coordinates.

C	int MPI_Cart_create (MPI_Comm comm_old, int ndims, int *dims, int *period, int reorder, MPI_Comm *comm_cart)
C++ (in the MPI::namespace):	Cartcomm Intracomm::Create_cart(int ndims, const int dims[], const bool periods[], bool reorder) const

Cartesian topologies

FORTRAN	MPI_CART_CREATE(COMM_OLD, NDIMS, DIMS, PERIODS, REORDER, COMM_CART, IERROR) INTEGER COMM_OLD, NDIMS, DIMS(*), COMM_CART, IERROR LOGICAL PERIODS(*), REORDER

Arguments :

IN	comm_old	input communicator (handle)
IN	ndims	number of dimensions of cartesian grid (integer)
IN	dims	integer array of size ndims specifying the number of processes in each dimension
IN	periods	logical array of size ndims specifying whether the grid is periodic (true) or not (false) in each dimension
IN	reorder	ranking may be reordered (true) or not (false) (logical)
OUT	comm_cart	communicator with new cartesian topology (handle)

MPI_Cart_rank

Mapping process grid coordinates to ranks:

C	int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)
C++ (in the MPI::namespace):	int Cartcomm::Get_cart_rank (const int coords[]) const
FORTRAN	MPI_CART_RANK(COMM, COORDS, RANK, IERROR) INTEGER COMM, COORDS(*), RANK, IERROR

MPI_Cart_rank

Arguments :

IN	comm	communicator with cartesian structure (handle)
IN	coords	integer array (of size ndims) specifying the Cartesian coordinates of a process
OUT	rank	rank of specified process (integer)

MPI_Cart_shift

Computing ranks of neighboring processes.

C	int MPI_Cart_shift (MPI_Comm comm, int direction, int disp, int *rank_source, int *rank_dest)
C++ (in the MPI::namespace):	void Cartcomm::Shift(int direction, int disp, int& rank_source, int& rank_dest) const

MPI_Cart_shift

Arguments :

IN	comm	communicator with Cartesian structure (handle)
IN	direction	coordinate dimension of shift (integer)
IN	disp	displacement (> 0: upwards shift,
		< 0: downwards shift) (integer)
OUT	rank_source	rank of source process (integer)
OUT	rank_dest	rank of destination process (integer)

Cartesian partitioning

Cut a grid up into slices. A new communicator is produced for each slice. Each slice can perform its own communications.

MPI_CART_SUB generates new communicator for the slices.

Arguments :

IN	comm	communicator with Cartesian structure (handle)
IN	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)
OUT	newcomm	communicator containing the subgrid that includes the calling process (handle)

Cartesian partitioning

C	int MPI_Cart_sub(MPI_Comm comm, int *remain_dims, MPI_Comm *newcomm)
C++ (in the MPI:: namespace)	Cartcomm Cartcomm::Sub(const bool remain_dims[]) const
FORTRAN	MPI_CART_SUB(COMM, REMAIN_DIMS, NEWCOMM, IERROR) INTEGER COMM, NEWCOMM, IERROR LOGICAL REMAIN_DIMS(*)

Cartesian partitioning

Arguments :

IN	comm	input communicator (handle)
IN	ndims	number of dimensions of cartesian structure (integer)
IN	dims	integer array of size ndims specifying the number of processes in each coordinate direction
IN	periods	logical array of size ndims specifying the periodicity specification in each coordinate direction
OUT	newrank	reordered rank of the calling process; MPI_UNDEFINED if calling process does not belong to grid (integer)

Cartesian partitioning

C	int MPI_Cart_map(MPI_Comm comm, int ndims, int *dims, int *periods, int *newrank)
C++ (in the MPI:: namespace)	int Cartcomm::Map(int ndims, const int dims[], const bool periods[]) cons
FORTRAN	MPI_CART_MAP(COMM, NDIMS, DIMS, PERIODS, NEWRANK, IERROR) INTEGER COMM, NDIMS, DIMS(*), NEWRANK, IERROR LOGICAL PERIODS(*)

Example

```
// Cartesian topology
#include <stdio.h>
#include <mpi.h>
// Cartesian topology:
//  2 – 5
//  |   |
//  1 – 4
//  |   |
//  0 – 3
```

Example

```
int main(int argc, char **argv)
{
int size,rank;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
                // get number of process
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
                // get rank of the process
int dim [2]=2,3;                    // 2x3 cartesian virtual topology
int per [2]=0,1;                    // period only on second coordiante
MPI_Comm com;
```

Example

```
MPI_Cart_create(MPI_COMM_WORLD,2,dim,per,1,&com);
                //create cartesian virtual topology
                //communicator
                //number of dimentions (2)
                //array of dimentions
                //array of periods 0,1
                //reorder 1
                //new comunicator
```

Example

```
int cord [2]={0,0};
int crank;
MPI_Cart_coords(com,rank,2,cord);
                // finds the coordiante of the rank process
                // cartesian communicator
                // rank of the process
                // number of dimentions
                // array of coordiante
printf("I am %d and my coordinates are (%d,%d)\n", rank, cord[0],
cord[1]);
printf(" my neighbours are:\n");
int c,d,ne;
```

Example

```
for(c=0;c<2;c++)
{
        for(d=-1;d<2;d++)
                { // shift left and right by 1 step (d=-1,d=1)
                if (d!=0)
                { MPI_Cart_shift(com,c,d,&rank,&ne);
                                // finds the rank of the neighbour
                                // coordiante
                                // displacement
                                // my rank
                                // rank of the neighbour
                if (ne>=0)
                    printf(" neighbour [%d,%d] has rank %d\n",c,d,ne);
                 }
```

Example

```
}
}
MPI_Finalize();
return 0;
}
```

Graph topologies - MPI_Graph_create

C	int MPI_Graph_create(MPI_Comm comm_old, int nnodes, int *index, int *edges, int reorder, MPI_Comm *comm_graph)
C++ (in the MPI::namespace)	Graphcomm Intarcomm::Create_graph(int nnodes, const int index[], const int edges[], bool reorder) const
In FORTRAN:	MPI_GRAPH_CREATE(comm_old, int nnodes, index, edges, reorder, comm_graph, ierror) LOGICAL reorder

Graph topologies - MPI_Graph_create

Neutral Binding:

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

Arguments :

IN	comm_old	input communicator (handle)
IN	nnodes	number of nodes in graph (integer)
IN	index	array of integers describing node degrees
IN	edges	array of integers describing graph edges
IN	reorder	ranking may be reordered (true) or not (false) (logical)
OUT	comm_graph	communicator with graph topology added (handle)

MPI_GRAPH_NEIGHBORS

Arguments MPI_GRAPH_NEIGHBORS(comm, rank, maxneighbors, neighbors) :

IN	comm	communicator with graph topology (handle)
IN	rank	rank of process in group of comm (integer)
IN	maxneighbors	size of array neighbors (integer)
OUT	neighbors	ranks of processes that are neighbors to specified process (array of integer)

MPI_GRAPH_NEIGHBORS

C	int MPI_Graph_neighbors(MPI_Comm comm, int rank, int maxneighbors, int *neighbors)
C++ (in the MPI:: namespace)	void Graphcomm::Get_neighbors(int rank, int maxneighbors, int neighbors[]) const
FORTRAN	MPI_GRAPH_NEIGHBORS(COMM, RANK, MAXNEIGHBORS, NEIGHBORS, IERROR) INTEGER COMM, RANK, MAXNEIGHBORS, NEIGHBORS(*), IERROR

Example - graph topology

```
error= MPI_Graph_create( MPI_COMM_WORLD,
        nnodes ,index, edges,0,&new_com);
if(error!=0)
{
printf("error at MPI_Graph_create");
return;
}
error=MPI_Graph_neighbors(new_com,
         my_number,1024,neighbours);
printf(" My number = %d Neighbours = ",my_number);
for(i=0;neighbours[i]!=-1;i++)
         printf(" %d ",neighbours[i]);
```

Example - graph topology

```
// Graph topology #include <stdio.h>
#include <mpi.h>
// Graph topology:
//    0
//   / \
//  1   2
// / \ / \
// 3  4   5
int main(int argc, char **argv)
{
int size,rank;
MPI_Init(&argc,&argv);
```

example

```
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm com;
int deg[6]={2,5,8,9,11,12}; // degree of nodes array
        // deg[i]=sum deg(j) where deg(j) is degree of j-th node
int edg[12]={1,2,0,3,4,0,4,5,1,1,2,2}; // edge array
        // first deg(0) numbers are nodes incident with 0 node
        // next deg(1) numbers are nodes incident width 1 node
```

Example - graph topology

```
MPI_Graph_create(MPI_COMM_WORLD,6,deg,edg,0,&com);
        // create graph virtual topology
        // communicator
        // number of nodes
        // degree of nodes array
        // egde array
        // reorder
        // graph communicator
```

Example - graph topology

```
int nodes,edges;
MPI_Graphdims_get(com,&nodes,&edges);
 hspace10mm // gets number of nodes and edges in graph topology
          // communicator
          // number of nodes
          // double number of edges
if (rank==0)
          printf("Thre are %d nodes and %d edges\n",nodes,edges);
int nneighbours;
```

Example - graph topology

```
MPI_Graph_neighbors_count(com,rank,&nneighbours);
        // finds the number of neighbours for rank node
        // graph comunicator
        // rank
        // number of neighbours
printf("I am %d and have %d neighbours:\n",rank,nneighbours);
int neighbours[100];
MPI_Graph_neighbors(com,rank,nneighbours,neighbours);
        // finds neighbours for rank node
        // graph comunicator
        // rank
        // maximal number of neighbors
        //(must be at most size of the neighbours array)
        // neighbours array
```

Example - graph topology

```
int i;
for(i=0;i<nneighbours;i++)
        printf(" neighbour %d\n",neighbours[i]);
MPI_Finalize();
return 0;
}
```

References

https://computing.llnl.gov/tutorials/mpi/

Thank you for your attention!

Any questions are welcome.