University of Nizhni Novgorod
Faculty of Computational Mathematics & Cybernetics

Introduction to Parallel Programming

Section 4. *Part* 3.

Parallel Programming with MPI

Gergel V.P., Professor, D.Sc.,
Software Department

Contents

Managing Groups of Processes and Communicators…

❑ Managing Groups…

– Processes are united into *groups*. The group may contain all the processes of a parallel program or a part of the available processes only. The same process may belong to several groups,

– The groups of processes are formed in order to create communicators on their basis,

– The groups of processes may be defined on the basis of the available groups only. The group associated with the predetermined communicator *MPI_COMM_WORLD* may be used as the source group:

```
int MPI_Comm_group ( MPI_Comm comm, MPI_Group *group );
```

Managing Groups of Processes and Communicators…

❑ Managing Groups…

– New groups may be created on the basis of the existing groups:

- It is possible to create a new group ***newgroup*** on the basis of the group ***oldgroup***, which includes *n* processes. The ranks of the processes to be included in ***newgroup*** are enumerated in the array ***ranks***:

```
int MPI_Group_incl(MPI_Group oldgroup,int n,
   int *ranks, MPI_Group *newgroup);
```

- It is possible to create a new group ***newgroup*** on the basis of the group ***oldgroup***, which includes *n* processes. The ranks of the processes that have not to be included in ***newgroup*** are enumerated in the array ***ranks***:

```
int MPI_Group_excl(MPI_Group oldgroup,int n,
  int *ranks, MPI_Group *newgroup);
```

Managing Groups of Processes and Communicators…

❑ Managing Groups…

– New groups may also be created by the following operations:

- Creating a new group ***newgroup*** by uniting the groups ***group1*** and ***group2***:

```
int MPI_Group_union(MPI_Group group1, MPI_Group group2,
       MPI_Group *newgroup);
```

- Creating a new group ***newgroup*** from the common processes of the groups ***group1*** and ***group2***:

```
int MPI_Group_intersection ( MPI_Group group1,
      MPI_Group group2, MPI_Group *newgroup );
```

- Creating a new group ***newgroup*** by the difference of the groups ***group1*** and ***group2***:

```
int MPI_Group_difference ( MPI_Group group1,
      MPI_Group group2, MPI_Group *newgroup );
```

Managing Groups of Processes and Communicators…

❑ Managing Groups:

– The following MPI functions provide obtaining information of the group of processes:

- Obtaining the number of processes in the group:

```
int MPI_Group_size ( MPI_Group group, int *size );
```

- Obtaining the rank of the current process in the group:

```
int MPI_Group_rank ( MPI_Group group, int *rank );
```

– After the termination of its use, the group must be deleted:

```
int MPI_Group_free ( MPI_Group *group );
```

Managing Groups of Processes and Communicators…

- **Managing Communicators…**
  - A *communicator* in MPI is a specially designed control object, which unites in its contents a group of *processes* and a number of additional parameters (*context*), which are used in data communication operations,
  - This subsection discusses managing the *intracommunicators*, which are used for data communication operation within a group of processes

Managing Groups of Processes and Communicators…

❑ Managing Communicators…

– To create new communicators the two main methods are used:
  - The duplication of the available communicator:

```
int MPI_Comm_dup ( MPI_Comm oldcom, MPI_comm *newcomm );
```

  - The creation of a new communicator from the subset of the processes of the available communicator:

```
int MPI_comm_create (MPI_Comm oldcom, MPI_Group group,
      MPI_Comm *newcomm);
```

– It should be noted that the operation of creating communicators is collective and must be executed by all the initial communicator processes,

– After the termination of its use, the communicator should be deleted:

```
int MPI_Comm_free ( MPI_Comm *comm );
```

Managing Groups of Processes and Communicators…

❑ Managing Communicators…

– The following function provides a fast and useful method of simultaneous creation of several communicators:

```
int MPI_Comm_split ( MPI_Comm oldcomm, int split, int key,
  MPI_Comm *newcomm ),
where
- oldcomm  – the initial communicator,
- split    – the number of the communicator, to which the process
             should belong,
- key      – the rank order of the process in the communicator being
             created,
- newcomm  – the communicator being created.
```

– The function *MPI_Comm_split* should be called in each process of the communicator ***oldcomm***

Managing Groups of Processes and Communicators

- **Managing Communicators:**
  - The execution of the function *MPI_Comm_split* leads to separating the processes into disjoint groups, each new group is formed from processes which have the same values of the parameter ***split***. On the basis of the created groups a set of communicators is created. The order of enumeration for the process ranks is selected in such a way that it corresponds to the order of the values ***key*** (the process with the greater value ***key*** should have a higher rank)

Virtual Topologies…

- The *topology* of a computer system is the structure of the network nodes and communication links, which connect them. The topology may be presented as a graph, where the vertices are the system processors (processes), and the arcs correspond to the available communication links (channels)
- Point-to-point data communication operations may be executed for any processes of the same communicator. All the processes of the communicator participate in collective operations. In this respect, the *logical topology* of the communication links in a parallel program is a *complete graph*
- We may organize the logical presentation of any necessary virtual topology. For this purpose it is sufficient to form additional process addressing

Virtual Topologies…

- **Cartesian Topologies (*Grids*)…**
  - *Cartesian topologies* assume the presentation of a set of processes as a rectangular grid and the use of Cartesian coordinate system for pointing to the processes,
  - The following function is used in MPI for creating the Cartesian topology (grid):

```
int MPI_Cart_create(MPI_Comm oldcomm, int ndims, int *dims,
 int *periods, int reorder, MPI_Comm *cartcomm),
where:
- oldcomm   -  the initial communicator,
- ndims     -  the Cartesian grid dimension,
- dims      -  the array of ndims length, it defines the number of processes
               in each dimension of the grid,
- periods   -  the array of ndims length, which defines whether the grid is
               periodical along each dimension,
- reorder   -  the parameter for pointing out  if the process ranks can be reodered,
- cartcomm  –  the communicator being created with the Cartesian
               process topology
```

Virtual Topologies…

❑ Cartesian Topologies (Grids)…

– In order to determine the Cartesian process coordinates according to its rank, the following function can be used:

```
int MPI_Card_coords ( MPI_Comm comm, int rank, int ndims,
  int *coords),
where:
 - comm   –  the communicator with grid topology,
 - rank   -  the rank of the process, for which Cartesian coordinates are
             determined,
 - ndims  -  the grid dimension,
 - coords-  the Cartesian process coordinates calculated by the function
```

Virtual Topologies…

- Cartesian Topologies (Grids)…

  – The reverse operation, i.e. determining the process rank according to its Cartesian coordinates, is provided by means of the following function:

```
int MPI_Cart_rank ( MPI_Comm comm, int *coords, int *rank ),
where
 - comm    – the communicator with grid topology,
 - coords  - the Cartesian coordinates of the process,
 - rank    - the process rank calculated by the function
```

Virtual Topologies…

❑ Cartesian Topologies (Grids)…

– The procedure of splitting the grids into subgrids of smaller dimension, which is useful in many applications, is provided by the following function :

```
int MPI_Card_sub(MPI_Comm comm, int *subdims,
  MPI_Comm *newcomm),
where:
 - comm     -  the initial communicator with grid topology,
 - subdims  –  the array for pointing the subgrid coordinates that can vary,
 - newcomm  -  the created communicator with the subgrid.
```

The function *MPI_Cart_sub* defines, while it is being carried out, the communicators for each combination of the coordinates of the fixed dimensions of the initial grid

Virtual Topologies…

❑ Cartesian Topologies (Grids)…

– ***Example***: Creating the two-dimensional grid *4x4*, the rows and columns of which have a ring structure (the last process is linked with the first one) . Then the communicators with the Cartesian topology is determined for each grid row and column separately. Eight communicators are created for the grid of 4x4 size in this example. For each process the defined communicators RowComm and *ColComm* correspond to the row and the column of the processes, to which the given process belongs

Code

Virtual Topologies…

❑ Cartesian Topologies (Grids)…

– The additional function MPI_Cart_shift provides the support of shift communications along a grid dimension:

- The ***cyclic shift*** on ***k*** elements along the grid dimension. The data from the process ***i*** is transmitted to the process ***(i+k) mod n***, where ***n*** is the size of the dimension, along which the shift is performed,
- *The **linear shift*** on ***k*** positions along the grid dimension. In this variant of the operation the data from the processor ***i*** is transmitted to the processor ***i+k*** (if the latter is available)

Virtual Topologies…

- **Cartesian Topologies (Grids)…**
  – The function *MPI_Cart_shift* provides obtaining the ranks of the processes, which are to exchange the data with the current process (the process, which has called up the function *MPI_Cart_shift*):

```
int MPI_Card_shift(MPI_Comm comm, int dir, int disp,
  int *source, int *dst),
where:
```

- **comm** – the communicator with grid topology,
- **dir** - the number of the dimension, along which the shift is performed,
- **disp** - the shift value (<0 – is the shift towards the beginning of the dimension),
- **source** – the rank of the process, from which the data should be obtained,
- **dst** - the rank of the process, to which the data should be sent

Virtual Topologies…

- **Cartesian Topologies (Grids):**
  – It should be noted, that the function *MPI_Cart_shift* only determines the rank of the processes, which are to exchange data in the course of shift operation. The execution of data transmission may be carried out, for instance, by means of the function *MPI_Sendrecv*

Virtual Topologies…

❑ Graph Topology…

– To create a communicator with the graph topology the following function is intended in MPI:

```
int MPI_Graph_create(MPI_Comm oldcomm, int nnodes,
  int *index, int *edges, int reorder, MPI_Comm *graphcomm),
where:
```

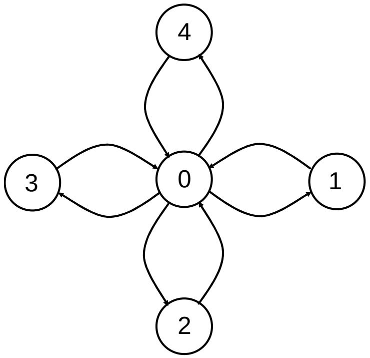
- **oldcomm** - the initial communicator,
- **nnodes** - the number of the graph vertices,
- **index** - the number of the arcs proceeding from each vertex,
- **edges** - the sequential list of the graph arcs,
- **reorder** - the flag for pointing out if the process ranks can be reodered,
- **cartcomm** – the created communicator with the graph type topology.

– Creating the topology is a collective operation and should be carried out by all the processes of the initial communicator

Virtual Topologies…

Graph Topology (example)…

- The number of processes is equal to 5, the graph vertices orders are (4,1,1,1,1), and the incidence matrix looks as follows:



Processes	Communication Lines
0	1,2,3,4
1	0
2	0
3	0
4	0

- To create the topology with the graph of this type, it is necessary to perform the following program code:

```
int index[] = { 4,1,1,1,1 };
int edges[] = { 1,2,3,4,0,0,0,0 };
MPI_Comm StarComm;
MPI_Graph_create(MPI_COMM_WORLD, 5, index, edges, 1,
  &StarComm);
```

Virtual Topologies

❑ Graph Topology:

– The number of the neighboring processes, which contain the outgoing arcs from the current process, may be obtained by the following function:

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

– Obtaining the ranks of the neighboring vertices is provided by the following function:

```
int MPI_Graph_neighbors(MPI_Comm comm,int rank,
      int mneighbors, int *neighbors);
```

(where ***mneighbors*** is the size of the array ***neighbors***)

Additional information on MPI…

- **Programming with MPI in Fortran…**
  - The subprograms of the library MPI are procedures, and thus, they are called by means of the procedure call statement CALL,
  - The termination codes are obtained though the additional parameter of the integer type, which is located last in the list of the procedure parameters,
  - The variable *status* is the integer type array, which consists of *MPI_STATUS_SIZE* elements,
  - The types *MPI_Comm* and *MPI_Datatype* are presented by the type INTEGER

Additional information on MPI…

Programming with MPI in Fortran

```
PROGRAM MAIN
  include 'mpi.h'
  INTEGER PROCNUM, PROCRANK, RECVRANK, IERR
  INTEGER STATUS(MPI_STATUS_SIZE)
  CALL MPI_Init(IERR)
  CALL MPI_Comm_size(MPI_COMM_WORLD, PROCNUM, IERR)
  CALL MPI_Comm_rank(MPI_COMM_WORLD, PROCRANK IERR)
  IF ( PROCRANK.EQ.0 )THEN
  ! operations carried out only with 0 rank process
    PRINT *,"Hello from process ", PROCRANK
    DO i = 1, PROCNUM-1
      CALL MPI_RECV(RECVRANK, 1, MPI_INT, MPI_ANY_SOURCE,
        MPI_ANY_TAG, MPI_COMM_WORLD, STATUS, IERR)
      PRINT *,"Hello from process ", RECVRANK
    END DO
  ELSE ! The message sent by all the processes,  except 0 rank process
    CALL MPI_SEND(PROCRANK,1,MPI_INT,0,0,MPI_COMM_WORLD,IERR)
  END IF
  CALL MPI_FINALIZE(IERR)
  STOP
END
```

Additional information on MPI…

- **Overview of MPI Program Execution Environment…**
  - The MPI program execution environment must be installed in a computer system in order to carry out parallel computations:
    - To provide the development, compilation, linkage and execution of parallel program the usual means of program development (such as, for instance, Microsoft Visual Studio) and a version of MPI library are sufficient,
    - In order to carry out the parallel programs, the environment should have the tool of indication of the processors being used, the utilities for starting the remote programs, etc.,
    - It is also desirable to have the tools of profiling, tracing and debugging parallel program in the environment

Additional information on MPI…

- **Overview of MPI Program Execution Environment:**
  – The start of MPI program also depends on the execution environment. In the majority of cases this operation is carried out by means of the command ***mpirun***. This command may have the following possible parameters:
    - *Execution mode*. It may be local or multiprocessor. The local mode is usually indicated by means of the key *–localonly,*
    - *The number of processes,* which should be created when a parallel program is started,
    - *The list of the processors* being used, which is defined by the configuration file,
    - *The executed file* of the parallel program,
    - *The command string* with the parameters of the executed program

Additional information on MPI

- **Additional Features of the Standard MPI-2**
  - The *dynamic generation of the processes*, which assumes the creation and termination of the parallel program processes in the course of execution,
  - The *single-sided process interaction*, which allows only one process to initiate data communication,
  - The *parallel input/output*, which provides a special interface for the operation of the processors with the file system,
  - The *extended collective operations*, which include, for instance, the procedures for simultaneous interaction of the processes from several communicators,
  - The *C++* interface

Summary

- This part of the presentation focuses on the issues of process and communicator and processes groups management
- The use of virtual topologies are considered
- The additional information on MPI is given. The information includes the issues of MPI based parallel program development in Fortran, an overview of the execution environment for MPI based programs and a survey of the additional possibilities of the standard MPI-2

Discussions

- Advantages of the use of the additional communicators
- Benefits of using the Cartesian and graph virtual topologies

Exercises

- Develop a sample program for the Cartesian topology
- Develop a sample program for a graph topology
- Develop subprograms for creating a set of additional virtual topologies (a star, a tree, etc.)

References…

- The internet resource, which describes the standard MPI: http://www.mpiforum.org
- One of the most widely used MPI realizations, the library MPICH, is presented on http://www-unix.mcs.anl.gov/mpi/mpich
- The library MPICH2 with the realization of the standard MPI-2 is located on http://www-unix.mcs.anl.gov/mpi/mpich2

References…

- **Group, W., Lusk, E., Skjellum, A.** (1994). Using MPI. Portable Parallel Programming with the Message-Passing Interface. –MIT Press.
- **Group, W., Lusk, E., Skjellum, A.** (1999a). Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface (Scientific and Engineering Computation). - MIT Press.
- **Group, W., Lusk, E., Thakur, R.** (1999b). Using MPI-2: Advanced Features of the Message Passing Interface (Scientific and Engineering Computation). - MIT Press.

References

- **Pacheco, P.** (1996). Parallel Programming with MPI. - Morgan Kaufmann.
- **Quinn, M. J.** (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
- **Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.** (1996). MPI: The Complete Reference. - MIT Press, Boston, 1996.

Next Section

- Parallel Programming with OpenMP

Author’s Team

Gergel V.P., Professor, Doctor of Science in Engineering, Course Author

Grishagin V.A., Associate Professor, Candidate of Science in Mathematics

Abrosimova O.N., Assistant Professor (chapter 10)

Kurylev A.L., Assistant Professor (learning labs 4,5)

Labutin D.Y., Assistant Professor (ParaLab system)

Sysoev A.V., Assistant Professor (chapter 1)

Gergel A.V., Post-Graduate Student (chapter 12, learning lab 6)

Labutina A.A., Post-Graduate Student (chapters 7,8,9, learning labs 1,2,3, ParaLab system)

Senin A.V., Post-Graduate Student (chapter 11, learning labs on Microsoft Compute Cluster)

Liverko S.V., Student (ParaLab system)

About the project

The purpose of the project is to develop the set of educational materials for the teaching course “Multiprocessor computational systems and parallel programming”. This course is designed for the consideration of the parallel computation problems, which are stipulated in the recommendations of IEEE-CS and ACM Computing Curricula 2001. The educational materials can be used for teaching/training specialists in the fields of informatics, computer engineering and information technologies. The curriculum consists of **the training course “Introduction to the methods of parallel programming”** and **the computer laboratory training “The methods and technologies of parallel program development”**. Such educational materials makes possible to seamlessly combine both the fundamental education in computer science and the practical training in the methods of developing the software for solving complicated time-consuming computational problems using the high performance computational systems.

The project was carried out in Nizhny Novgorod State University, the Software Department of the Computing Mathematics and Cybernetics Faculty (http://www.software.unn.ac.ru). The project was implemented with the support of Microsoft Corporation.