University of Nizhni Novgorod
Faculty of Computational Mathematics & Cybernetics

Introduction to Parallel Programming

Section 4. *Part* 1.

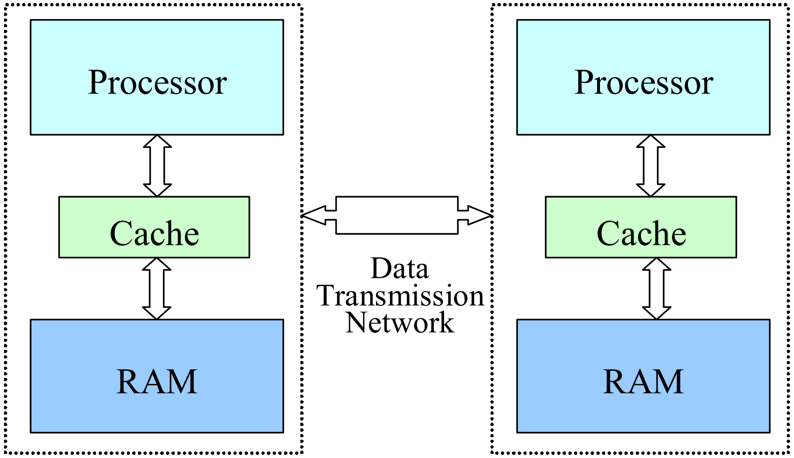
Parallel Programming with MPI…

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

Contents

Introduction…

The processors in the computer systems with distributed memory operate independently.



It is necessary to have a possibility:

– to *distribute* the computational load,

– to *organize the information communication* (data transmission) among the processors.

*The solution of the above mentioned problems is provided by the **MPI** (message passing interface)*

Introduction…

- MPI uses the simple approach: a program is developed for solving the stated problem and this single program is copied on all the available processors
- In order to obtain the different computations on different processors:
  - It is possible to substitute different data for executing the program on different processors,
  - It is possible to vary computations using the processor identifier, on which the program is executed
- This method of implementation of parallel computations is referred to as the model *single program multiple processes* or *SPMP*

Introduction…

- There are many data transmission functions in MPI:
  - They provide various techniques of data passing,
  - They implement practically all the communication operations.

These possibilities are the main advantages of MPI (in particular, the very name of MPI testifies to it)

Introduction…

Understanding of MPI

- MPI is a standard for organizing the message passing
- MPI is the software, which should provide the possibility of message passing and correspond to all the requirements of MPI standard:
  - This software should be arranged as program module libraries (*MPI libraries*),
  - This software should be comprehensible for the most widely used algorithmic languages C and Fortran

Introduction…

The advantages of MPI

- MPI makes possible to a considerable extent to decrease the complexity of the parallel program portability among different computer systems
- MPI contributes to the increase of parallel computation efficiency, as there are MPI library implementations for practically every type of computational system nowadays
- MPI decreases the complexity of parallel program development:
  - The greater part of the basic data communication operations are provided by MPI standard,
  - There are many parallel numerical libraries available nowadays developed with the use of MPI

Introduction

MPI History

1992. The start of investigations on the message passing interface library (Oak Ridge National Laboratory, Rice University).

November, 1992. The publication of the working variant of the standard MPI 1.

November, 1993. The discussion of the standard during conference Supercomputing'93.

May 5, 1994. The final version of MPI 1.0 standard.

June 12, 1995. New version of standard - MPI 1.1.

July 18, 1997. Standard MPI-2 was published: Extensions to the Message-Passing Interface.

Developing the MPI standard is provided by the international consortium ***MPI Forum***

MPI: Basic Concepts and Definitions…

The Concept of Parallel Program

- Within the framework of MPI a *parallel program* means a number of simultaneously carried out *processes*:
  - The processes can be carried out on different processors, several processes may be located on a processor,
  - Each parallel process is generated on the basis of the copy of the same program code (*SPMP model*)
- The source program code for the program being executed is developed in the algorithmic languages C or Fortran with the use of a MPI library implementation
- The number of processes and the number of the processors used are determined at the moment when the parallel program start by the means of MPI program execution environment. These numbers must not be changed in the course of computations.
- All the program processes are sequentially enumerated. The process number is referred to as *the process rank*

MPI: Basic Concepts and Definitions…

There are four main concepts at the core of MPI:

- The type of data passing operations,
- The type of data, which are transmitted,
- The concept of communicator,
- The concept of virtual topology

MPI: Basic Concepts and Definitions…

Data Communication Operations:

- Data communication operations form the core of MPI
- The functions provided within MPI usually differentiate between:
  - *point-to-point operations*, i.e. operations between two processors,
  - *collective operations*, i.e. communication procedures for the simultaneous interaction of several processes

MPI: Basic Concepts and Definitions…

Communicators…

- The *communicator* in MPI is a specially designed control object, which unites within itself a *group of processes* and a number of complementary parameters (*context*):
  - Point-to-point data transmission operations are carried out for the processes, which belong to the same communicator,
  - Collective operations are applied simultaneously to all the processes of the communicator
- It is necessary to point to the communicator being used for data communication operations in MPI

MPI: Basic Concepts and Definitions…

Communicators

- In the course of computations new communicators can be created and the already existing communicators can be deleted
- The same process can belong to different communicators
- All the processes available in a parallel program belong to the communicator with the identifier MPI_COMM_WORLD, which is created on default
- If it is necessary to transmit the data among the processors, which belong to different groups, an *intercommunicator* should be created

MPI: Basic Concepts and Definitions…

Data Types

- It is necessary to point to the type of the transmitted data in MPI data passing functions
- MPI contains a wide set of the basic data types. These data types largely coincide with the data types of the algorithmic languages C and Fortran
- MPI has possibilities for creating new derived data types for more accurate and precise description of the transmitted message content

MPI: Basic Concepts and Definitions

Virtual Topologies

- The logical topology of the communication links among the processes is a complete graph (regardless of the availability of real physical communication channels among the processors)
- MPI provides an opportunity to present a number of processes as a *grid* of arbitrary dimension. The boundary processes of the grids can be referred to as neighboring, and thus, the structures of *torus* type can be defined on the basis of the grids
- MPI provides for the possibility to form *logical (virtual) topologies* of any desirable type

The Fundamentals of MPI…

- **MPI Program Initialization and Termination…**
  - *The first MPI function*, which is called, must be the following:

```
int MPI_Init ( int *agrc, char ***argv );
```

(it is called to initialize MPI program execution environment; the parameters of the function are the number of arguments in the command line and the command line text),

  - *The last MPI function* to be called must be the following one:

```
int MPI_Finalize (void);
```

The Fundamentals of MPI…

- **MPI Program Initialization and Termination:**
  - The structure of the MPI-based parallel program should look as follows:

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  <program code without the use of MPI functions>
  MPI_Init ( &agrc, &argv );
    <program code with the use of MPI functions>
  MPI_Finalize();
  <program code without the use of MPI functions>
  return 0;
}
```

The Fundamentals of MPI…

- **Determining the Number and the Rank of the Processes…**
  - The *number of the processes* in the parallel program being executed can be obtained by means of the following function:

```
int MPI_Comm_size ( MPI_Comm comm, int *size );
```

  - The following function is used to determine the *process rank*:

```
int MPI_Comm_rank ( MPI_Comm comm, int *rank );
```

The Fundamentals of MPI…

❑ Determining the Number and the Rank of the Processes…

– As a rule, the functions *MPI_Comm_size* and *MPI_Comm_rank* are called right after *MPI_Init* :

```
#include "mpi.h"
int main ( int argc, char *argv[] ) {
  int ProcNum, ProcRank;
  <program code without the use of MPI functions>
  MPI_Init ( &agrc, &argv );
  MPI_Comm_size ( MPI_COMM_WORLD, &ProcNum);
  MPI_Comm_rank ( MPI_COMM_WORLD, &ProcRank);
    <program code with the use of MPI functions>
  MPI_Finalize();
  <program code without the use of MPI functions>
  return 0;
}
```

The Fundamentals of MPI…

- **Determining the Number and the Rank of the Processes:**
  - Communicator *MPI_COMM_WORLD*, as it has been previously mentioned, is created on default and presents all the processes carried out by a parallel program,
  - The rank obtained by means of the function *MPI_Comm_rank* is the rank of the process, which has called this function, i.e. the variable *ProcRank* will accept different values in different processes

The Fundamentals of MPI…

- **Message Passing…**
  – In order to transmit data, the sending process should carry out the following function:

```
int MPI_Send(void *buf, int count, MPI_Datatype type,
  int dest, int tag, MPI_Comm comm),
where
  - buf    – the address of the memory buffer, which contains the data of the
             message to be transmitted,
  - count  – the number of the data elements in the message,
  - type   - the type of the data elements in the message,
  - dest   - the rank of the process, which is to receive the message,
  - tag    - tag-value, which is used to identify the message,
  - comm   - the communicator, within of which the data is transmitted
```

Introduction into MPI Based Development of Parallel Programs…

The Fundamentals of MPI

- **Message Passing…**

The basic (predefined) MPI data types for the algorithmic language C

MPI_Datatype	C Datatype
MPI_BYTE	
MPI_CHAR	signed char
MPI_DOUBLE	Double
MPI_FLOAT	Float
MPI_INT	Int
MPI_LONG	Long
MPI_LONG_DOUBLE	long double
MPI_PACKED	
MPI_SHORT	short
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_SHORT	unsigned short

The Fundamentals of MPI…

Message Passing:

- The message to be sent is determined by pointing to the memory block (*buffer*), which contains the message. The triad, which is used to point to the buffer (*buf, count, type*), is included into the parameters of practically all data passing functions,
- The processes, among which data is passed, should belong to the communicator, specified in the function *MPI_Send*,
- The parameter *tag* is used only when it is necessary to differentiate among the messages being passed. Otherwise, an arbitrary integer number can be used as the parameter value

The Fundamentals of MPI…

❑ Message Receiving…

– In order to receive messages the receiving process should carry out the following function:

```
int MPI_Recv(void *buf, int count, MPI_Datatype type,
  int source,int tag, MPI_Comm comm, MPI_Status *status),
where
```

- **buf, count, type** – the memory buffer for the message receiving,
- **source** - the rank of the process, from which message is to be received,
- **tag** - the tag of the message, which is to be received for the process,
- **comm** - communicator, within of which data is passed,
- **status** – the pointer of the data structure, which contains the information of the results of carrying out the data passing operation.

The Fundamentals of MPI…

- **Message Receiving…**
  - Memory buffer should be sufficient for data reception and the element types of the sent and the received messages must coincide. In case of memory shortage a part of the message will be lost and in the code of the function termination there will be an overflow error registered,
  - The value *MPI_ANY_SOURCE* may be given for the parameter source, if there is a need to receive a message from any sending process,
  - If there is a need to receive a message with any tag, then the value *MPI_ANY_TAG* may be given for the parameter tag

The Fundamentals of MPI…

Message Receiving…

– The parameter *status* makes possible to define a number of characteristics of the received message:

```
-status.MPI_SOURCE – the rank of the process, which has sent the received
                     message,
-status.MPI_TAG    - tag of the received message.
```

The function

```
MPI_Get_count(MPI_Status *status, MPI_Datatype type, int *count );
```

returns in the variable *count* the number of type elements in the received message

The Fundamentals of MPI…

- **Message Receiving**

  The function *MPI_Recv* is a blocking one for the receiving process, i.e. carrying out of the process is suspended till the function terminates its operation. Thus, if due to any reason the expected message is missing, then the parallel program execution will be blocked forever

The Fundamentals of MPI…

- **The First MPI Based Parallel Program…**

  Code

  – Each process find out its rank, and after that all the operations in the program are separated (different processes execute different calculations),

  – All the processes, except the process with the rank 0, send the value of its rank to the process 0,

  – The process 0 first prints the value of its rank and then receives the messages from the other processes and prints their ranks sequentially,

  – A possible variant of the program results can consist in the following:

```
Hello from process 0
Hello from process 2
Hello from process 1
Hello from process 3
```

The Fundamentals of MPI…

- **The First MPI Based Parallel Program (remarks)…**
  - It should be noted that the order of message receiving is not predetermined. It depends on the execution conditions for parallel program (moreover, the order can change from execution to execution). If it does not lead to efficiency losses, it is necessary to provide the unambiguity of computations in case of parallel computations:

```
MPI_Recv(&RecvRank, 1, MPI_INT, i, MPI_ANY_TAG,
        MPI_COMM_WORLD, &Status)
```

    Setting the rank of the sending process regulates the order of message reception.

The Fundamentals of MPI…

- **The First MPI Based Parallel Program (remarks)…**
  - It is recommended to carry out the program code of different processes into separate program modules (*functions*), if the amount of the developed programs is significant. The general scheme of an MPI based program in this case looks as follows:

```
MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
if ( ProcRank == 0 ) DoProcess0();
else if ( ProcRank == 1 ) DoProcess1();
else if ( ProcRank == 2 ) DoProcess2();
```

The Fundamentals of MPI…

- **The First MPI Based Parallel Program (remarks):**
  - All the MPI functions return the termination code as their value. If the function is completed successfully the return code is *MPI_SUCCESS*. The other values of the termination code testifies to the fact that some errors have been discovered in the course of function execution. To find out the type of the discovered error predetermined named constants are used. Among these constants there are the following ones:

    - `MPI_ERR_BUFFER` – incorrect buffer pointer,
    - `MPI_ERR_COMM` – incorrect communicator,
    - `MPI_ERR_RANK` – incorrect process rank,

    and others.

Evaluating of MPI Program Execution Time:

– The execution time needs to know for estimating the obtained speedup of parallel computation,

– Obtaining the time of the current moment of the program execution is provided by means of the following function:

```
double MPI_Wtime(void);
```

– The accuracy of time measurement can depend on the environment of the parallel program execution. The following function can be used in order to determine the current value of time measurement accuracy:

```
double MPI_Wtick(void);
```

(time between two sequential values of the computer system hardware timer in seconds)

Introduction into Collective Data Communication Operations…

- To demonstrate the example of MPI function applications the problem of summation is considered:

$$S = \sum_{i=1}^{n} x_i$$

- To develop the parallel algorithm it is necessary to divide the data into equal blocks, to transmit these blocks to the processes, to carry out the summation of the obtained data in the processes, to collect the values of the computed partial sums on one of the processes and to add the values of partial sums to obtain the general result of the problem,
- This algorithm will be simplified - all the vector being summed up, not only separate blocks of the vector, will be transmitted to the processes

Introduction into Collective Data Communication Operations…

- **Data Broadcasting…**
  - There is the need for transmitting the values of the vector *x* to all the parallel processes,
  - An evident way is to use the above discussed MPI communication functions to provide all required data transmissions:

```
MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
for (i=1; i<ProcNum; i++)
  MPI_Send(&x,n,MPI_DOUBLE,i,0,MPI_COMM_WORLD);
```

  - The repetition of the data transmissions leads to summing up the latencies of the communication operations,
  - The required data transmissions can be executed with the smaller number of iterations

Introduction into Collective Data Communication Operations…

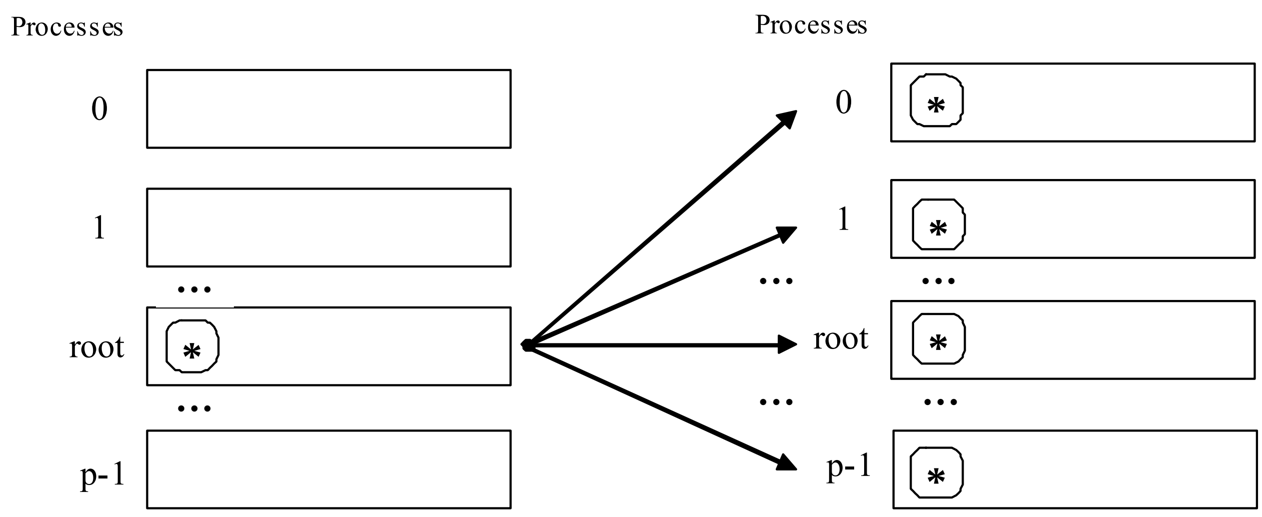
- **Data Broadcasting…**
  - To achieve efficient broadcasting the following MPI function can be used:

```
int MPI_Bcast(void *buf,int count,MPI_Datatype type,
      int root,MPI_Comm comm),
where
```

- **buf, count, type** – memory buffer, which contains the transmitted message (process with the rank *root*) and for message receiving for the rest of the processes,
- **root** - the rank of the process, which carries out data broadcasting,
- **comm** - the communicator, within of which data broadcasting is executed.

Introduction into Collective Data Communication Operations…

- **Data Broadcasting…**
  – The function *MPI_Bcast* carries out transmitting the data from the buffer *buf*, which contains *count* type elements, from the processor with the rank *root* to the processes within the communicator *comm*



a) Before the operation b) After the operation

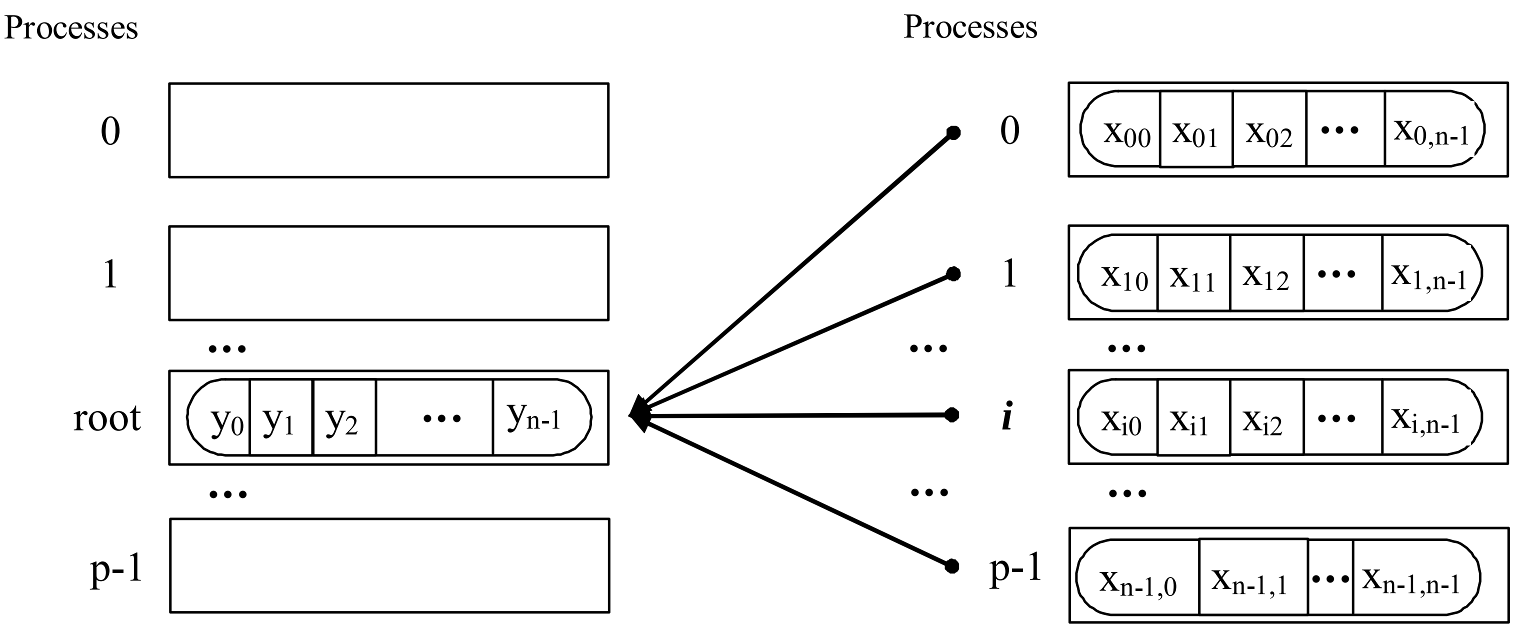
Introduction into Collective Data Communication Operations…

- **Data Broadcasting:**
  - The function *MPI_Bcast* is the collective operation, and thus, the call of the function MPI_Bcast, when the necessary data should be transmitted, is to be executed by all the processes of the communicator *comm*,
  - The memory buffer pointed in the function MPI_Bcast has different designations in different processes:
    - For the root process, from which data broadcasting is performed, this buffer should contain the transmitted message,
    - For the rest of the processes the buffer is intended for data receiving

Code

Introduction into Collective Data Communication Operations…

- Data Transmission from All the Processes to a Process…
  - The procedure of collecting and further data summation available in the above described program is an example of the widely used operation of transmitting data from all the processes to a process



a) After the operation b) Before the operation

$$y_j = \bigotimes_{i=0}^{n-1} x_{ij},\ 0 \le j < n$$

Introduction to MPI Based Development of Parallel Programs…

Introduction into Collective Data Communication Operations…

- **Data Transmission from All the Processes to a Process**

```
int MPI_Reduce(void *sendbuf, void *recvbuf,int count,
      MPI_Datatype type, MPI_Op op,int root,MPI_Comm comm),
```

where

- **sendbuf** - memory buffer with the transmitted message,
- **recvbuf** – memory buffer for the resulting message (only for the root process),
- **count** - the number of elements in the messages,
- **type** – the type of message elements,
- **op** - the operation, which should be carried out over the data,
- **root** - the rank of the process, on which the result must be obtained,
- **comm** - the communicator, within of which the operation is executed.

Introduction into Collective Data Communication Operations…

- The Basic MPI Operation Types for Data Reduction Functions…

Operation	Description
MPI_MAX	The maximum value calculation
MPI_MIN	The minimum value calculation
MPI_SUM	The calculation of the sum of the values
MPI_PROD	The calculation of the product of the values
MPI_LAND	The execution of the logical operation "AND" over the message values
MPI_BAND	The execution of the bit operation "AND" over the message values
MPI_LOR	The execution of the logical operation "OR" over the message values
MPI_BOR	The execution of the bit operation "OR" over the message values
MPI_LXOR	The execution of the excluding logical operation "OR" over the message values
MPI_BXOR	The execution of the excluding bit operation "OR" over the message values
MPI_MAXLOC	The calculation of the maximum values and their indices
MPI_MINLOC	The calculation of the minimum values and their indices

Introduction into Collective Data Communication Operations…

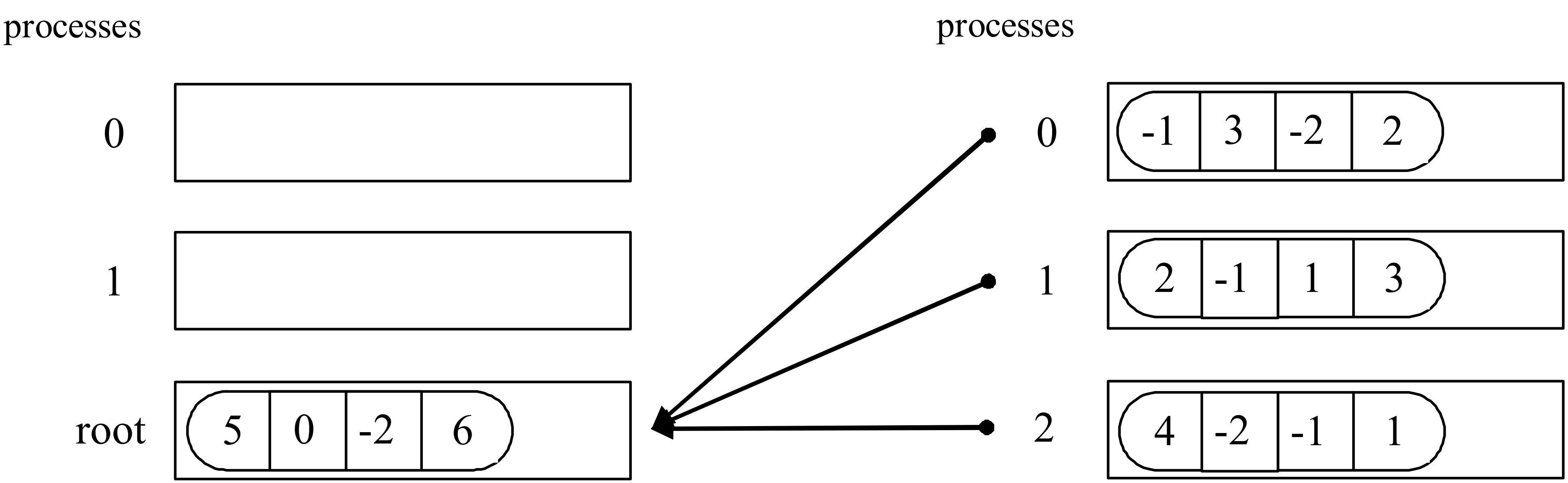
- Types of MPI operations for the data reduction functions:
  - **MPI_MAX** and **MPI_MIN** calculates the maximum and minimum values,
  - **MPI_SUM** calculates the sum of the values,
  - **MPI_PROD** calculates the product of the values,
  - **MPI_LAND**, **MPI_BAND**, **MPI_LOR**, **MPI_BOR**, **MPI_LXOR**, **MPI_BXOR** – logical and bit operations AND, OR, XOR,
  - **MPI_MAXLOC** and **MPI_MINLOC** - calculates the maximum and minimum values and their positions

Introduction into Collective Data Communication Operations…

- **Data Transmission from All the Processes to a Process:**
  - The function *MPI_Reduce* is the collective operation, and thus, the function call should be carried out by all the processes of the communicator *comm*. All the calls should contain the same values of the parameters *count*, *type*, *op*, *root*, *comm*,
  - The data transmission should be carried out by all the processes. The operation result will be obtained only by *root* process,
  - The execution of the reduction operation is carried out over separate elements of the transmitted messages

Introduction into Collective Data Communication Operations…

- **Data Transmission from All the Processes to a Process** (example for calculating the sum of the values)



a) After the operation

b) Before the operation

Introduction into Collective Data Communication Operations

- **Computation Synchronization:**
  - Process synchronization, i.e. simultaneous achieving the specified points of the parallel program by various processes is provided by means of the MPI function:

```
int MPI_Barrier(MPI_Comm comm);
```

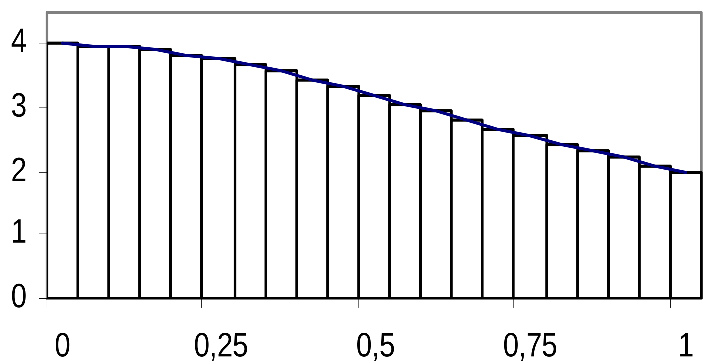
  - The function *MPI_Barrier* is collective operation. Thus, this function should be called by all the processes of the communicator *comm*,
  - When the function *MPI_Barrier* is called, the process execution is blocked. The computations of the process will continue only after the function *MPI_Barrier* is called by all the communicator processes

***Example*: Computation of the constant π**

- The value of constant π can be computed by means of the integral

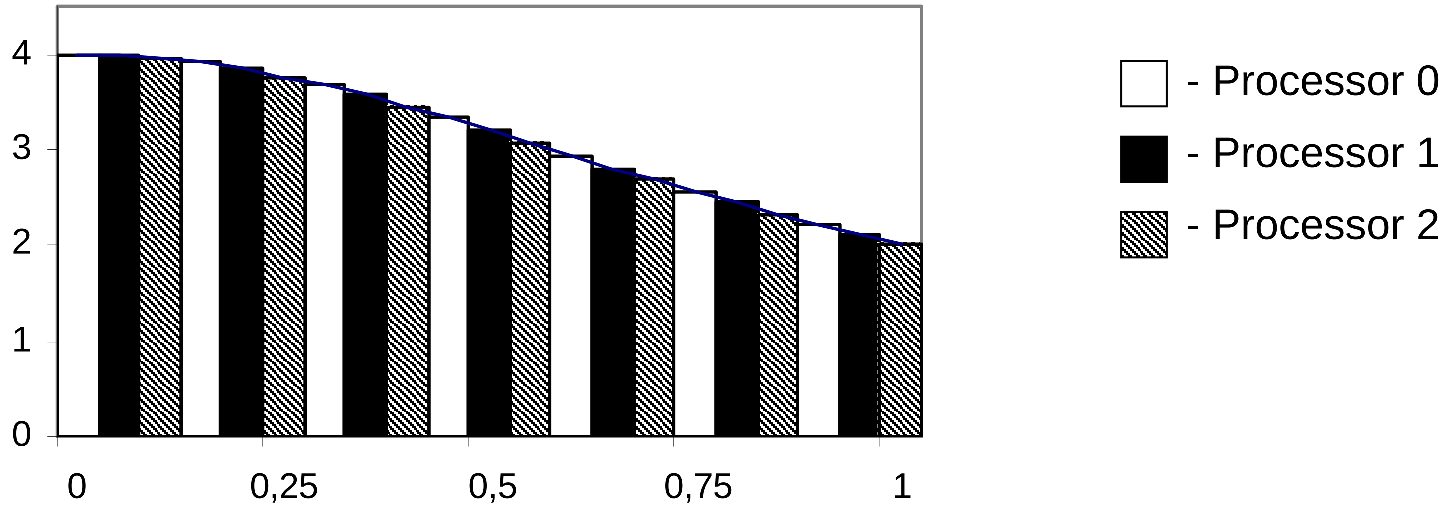
$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

- To compute this integral the method of rectangles can be used for numerical integration



<u>Example</u>: *Computation of the constant* π

- Cyclic scheme can be used to distribute the calculations among the processors
- Partial sums, that were calculated on different processors, have to be summed



Example: Computation of the constant π

```
#include "mpi.h"
#include <math.h>
double f(double a) {
  return (4.0 / (1.0 + a*a));
}
int main(int argc, char *argv) {
  int ProcRank, ProcNum, done = 0, n = 0, i;
  double PI25DT = 3.141592653589793238462643;
  double mypi, pi, h, sum, x, t1, t2;
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&ProcNum);
  MPI_Comm_rank(MPI_COMM_WORLD,&ProcRank);
  while (!done ) { // main computational loop
    if ( ProcRank == 0) {
      printf("Enter the number of intervals: ");
      scanf("%d",&n);
      t1 = MPI_Wtime();
    }
```

Example: *Calculating* π

```
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  if (n > 0) { // calculating the local sums
    h = 1.0 / (double) n;
    sum = 0.0;
    for (i = ProcRank + 1; i <= n; i += ProcNum) {
      x = h * ((double)i - 0.5);
      sum += f(x);
    }
    mypi = h * sum;
    // reduction
   MPI_Reduce(&mypi,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
    if ( ProcRank == 0 ) { // printing results
      t2 = MPI_Wtime();
      printf("pi is approximately %.16f, Error is
                 %.16f\n",pi, fabs(pi - PI25DT));
      printf("wall clock time = %f\n",t2-t1);
    }
  } else done = 1;
 }
 MPI_Finalize();
}
```

Summary

- It is discussed a number of concepts and definitions, which are the basic ones for the standard MPI (parallel program, message passing operations, data types, communicators, virtual topologies)
- A brief and simple introduction into the development of MPI based parallel programs is given
- Examples of the MPI based parallel programs are presented

Discussions

- The complexity of the MPI based parallel programs
- The problem of debugging the parallel programs

Exercises

- Develop a program for finding the minimum (maximum) value of the vector elements
- Develop a program for computing the inner product of two vectors
- Develop a program, where two processes repeatedly exchange messages of *N* byte length. Carry out the experiments and estimate the dependence of the data operation execution time with respect to the message length. Compare it to the theoretical estimations obtained according to the Hockney model

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 MPI…

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.