University of Nizhni Novgorod
Faculty of Computational Mathematics & Cybernetics

Introduction to Parallel Programming

Section 6.

Principles of Parallel Algorithm Design

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

Contents

Introduction…

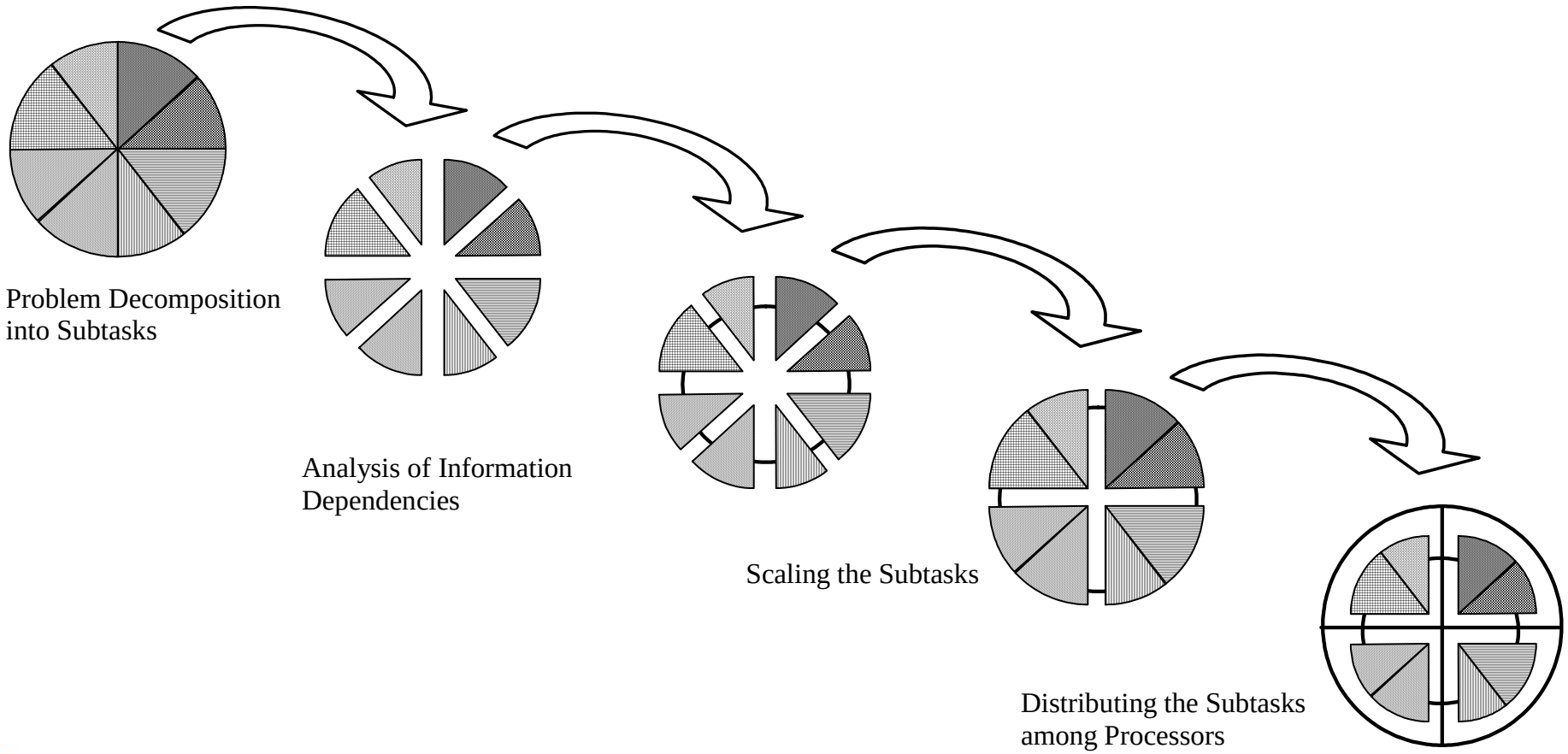
- Developing the efficient methods of parallel computation can be implemented as follows:
  - To analyze the available computational schemes and subdivide (*decompose*) them into parts (*subtasks*), which can be calculated to substantial degree independently,
  - To find out the information dependencies between the obtained subtasks; the information interactions should be carried out in the course of solving the originally formulated problem,
  - To determine the computer system, which is necessary (or available) for solving the problem, and distribute the subtasks among the processors.

Introduction…

- The amount of computations for each processor being used should be approximately the same. It provides for the uniform computational processor loading (*load balancing*)
- The distribution of subtasks among the processors should be carried out in such a way that the number of information links (*communication interactions*) between the processors should be minimal

Introduction…

❑ The general scheme of parallel algorithm design



Introduction…

- After carrying out all the design stages, it is necessary to evaluate the efficiency of the developed parallel methods
- According to the results of the analysis, it can appear to be necessary to review some (or even all) design stages:
  - e.g., in some cases the spectrum of the subtasks needs to be corrected - the subtasks can be *aggregated*, if only a small number of processors are available, or vice versa *detailed*. In general these actions can be defined as *scaling* the developed algorithm.

Introduction

- To apply the parallel method it is necessary to develop programs for solving the formed set of subtasks and distribute the developed programs among the processors
- To carry out the information interactions, the program should have the means of data exchange (*data transmission channels*)
- Each processor is appointed for solving a single subtask. However, if there are many subtasks or the number of processors in use is limited, several programs (*processes*) can be executed on each processor

Parallel Program Modeling…

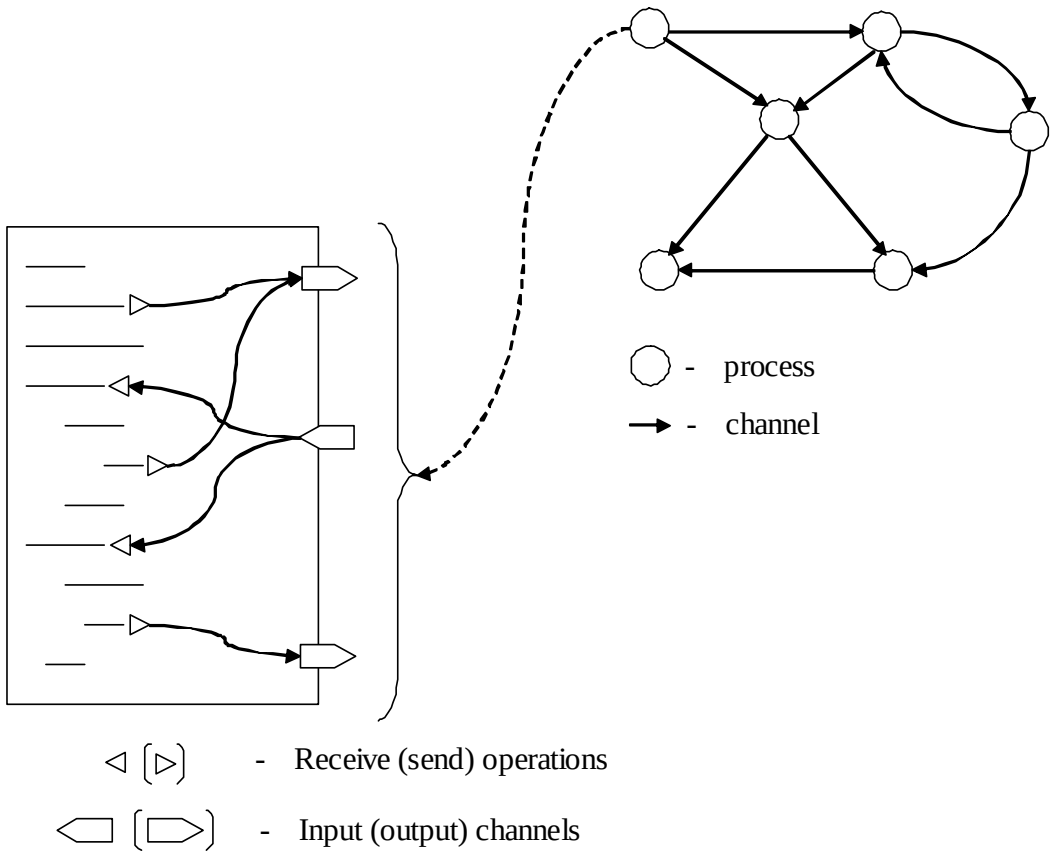
- At the design stage the parallel algorithm can be represented as a ***"subtasks-messages"*** graph. This graph is the aggregated representation of the graph of information dependencies (the *"operations-operands"* graph)
- "Subtasks-Messages" model allows:
  - To concentrate on the problems of forming the subtasks of equal computation complexity,
  - To provide a low level of information dependencies among the subtasks.

Parallel Program Modeling…

- At the execution stage the parallel algorithm can be represented as a ***"processes–channels" graph***, where the concept of processes and channels instead of subtasks and messages is used
- "Processes-Channels" model allows:
  - To optimally distribute subtasks among the processors for decreasing the communication time of information interactions between the subtasks by assigning the groups of intensively interacting processes to the same processors,
  - To analyze the efficiency of the developed parallel methods,
  - To describe the process of parallel computations more adequately.

Parallel Program Modeling…

- **Parallel program model in the form of the "processes-channels" graph**



Parallel Program Modeling

- **The "Processes-Channels" Graph Model…**
  - ***Process*** is a *program*, which is executed on a processor. It uses for its operation a part of the processor's local memory and contains a number of data communication operations for the information interactions among the parallel executed processes

Parallel Program Modeling

- **The "Processes-Channels" Graph Model:**
  - ***Data transmission channel***. From the logical point of view a *data transmission channel* can be considered as a message queue, to which one or more processes can send the transmitted data, and from which the addressee process can get the messages sent by the other processes:
    - Channels appear dynamically at the moment when the first send/receive operation of the channel is executed,
    - A channel can correspond to one or several operations of data receiving of the addressee process,
    - The channel capacity is unlimited,
    - Message receiving operation can lead to delays, if the data, requested from the channel, has not been sent yet by the sending processor

Stages of Parallel Algorithm Design…

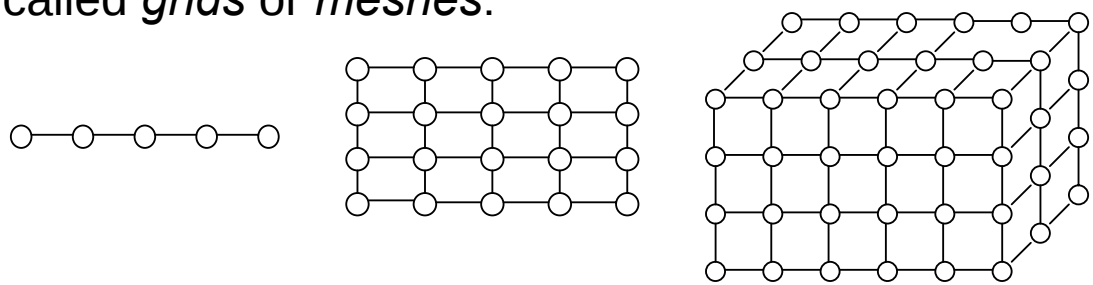
- Developing the parallel algorithm can include the stages:
  - Problem Decomposition into Subtasks,
  - Analysis of Information Dependencies,
  - Scaling the Subtasks,
  - Distributing the Subtasks among Processors
- To demonstrate the formulated scheme, all further discussion is given with the use of the problem of searching the maximal value among matrix ***A*** elements:

$$y = \max_{1 \le i, j \le N} a_{ij}$$

Stages of Parallel Algorithm Design…

Problem Decomposition into Subtasks…

- Two relevant computational schemes:
  - Carrying out the identical processing of a large set of data - ***data parallelism***. In this case the subtasks selection comes up to partitioning the available data:
    - For a large number of problems, data partitioning leads to forming one -, two-, three-dimensional sets of subtasks. The information links in these subtasks exist only among the nearest neighbors. Such schemes are usually called *grids* or *meshes*:

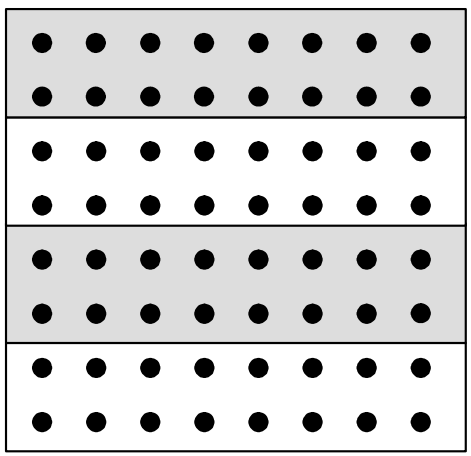


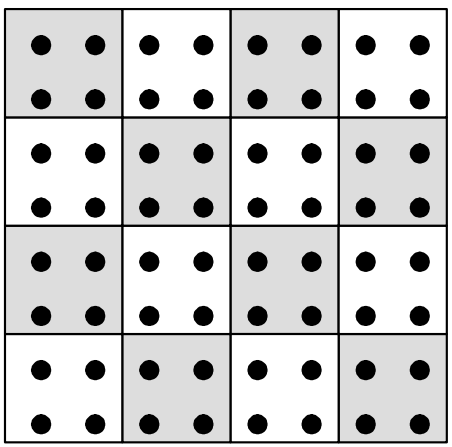
  - Carrying out different operations over the same set of data - ***functional parallelism***.

Stages of Parallel Algorithm Design…

Problem Decomposition into Subtasks…

- ***Example***: The problem of searching the maximal value among matrix ***A*** elements:
  - divide matrix ***A*** into a set of separate rows – *rowwise block-striped* decomposition;
  - divide matrix ***A*** into a set of rectangular blocks – *checkerboard* decomposition.





Stages of Parallel Algorithm Design…

Problem Decomposition into Subtasks…

- **Decomposition Levels**…
  - Getting the maximum possible amount of subtasks (*fine-grained parallelism*):
    - Provides the use of maximum achievable parallelism for the problem being solved,
    - Complicates the parallel computation analysis.
  - Using the large-sized subtasks (*coarse-grained parallelism*):
    - Leads to a clear parallel computation scheme,
    - Can make it more difficult to use a large number of processors efficiently.

Stages of Parallel Algorithm Design…

Problem Decomposition into Subtasks…

- **Decomposition Levels**:
  - ***Intermediate level*** - the use of only those subtasks, for which the methods of parallel computations are known as the constructive decomposition components. The subtasks selected by means of this approach will be further referred to as the *basic* ones. They can be *elementary* (*nondivisible*), if they do not allow further partitioning, or *compound* if they do.

Stages of Parallel Algorithm Design…

Problem Decomposition into Subtasks

- The list of questions can be suggested for estimating the correctness of the decomposition stage:
  - Does the performed decomposition increase the amount of computations and the necessary memory size?
  - Is the balanced loading of all the available processors possible with the chosen decomposition level?
  - Are the selected parts of computation process sufficient for efficient loading of the available processors (taking into account the possibility to increase their number)?

Stages of Parallel Algorithm Design…

Analysis of Information Dependencies…

- **Main Definitions**…
  - *Local and global schemes* of data communications:
    - For local schemes the data communications at every moment are executed only among a small number of subtasks (placed, as a rule, on neighboring processors),
    - For global data communication operations all subtasks participate in the communication process.
  - *Structural and nonstructural communication methods:*
    - For structural methods data transmissions lead to forming some standard communication schemes (for instance, in the form of a ring, a rectangular mesh, etc.),
    - The scheme of data communication operation for nonstructural interaction structures is not homogeneous.

Stages of Parallel Algorithm Design…

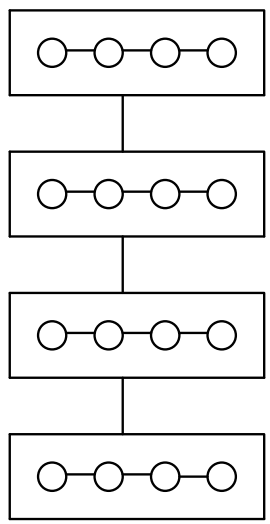
Analysis of Information Dependencies…

- **Main definitions:**
  - *Static or dynamic schemes* of data communications:
    - For the static schemes the information communication moments and participants are fixed at the stages of design and parallel program development,
    - For the dynamic variant of communications the structure of data transmission operation is determined in the course of computations.
  - *Synchronous* and *asynchronous* communication methods:
    - Data communication operations for synchronous methods are carried out only if all the interaction participants are ready for communication and they are ended only after the completion of all the communication actions,
    - If the operations are carried out asynchronously, the communication participants do not have to wait for the total completion of data communication operations.

Stages of Parallel Algorithm Design…

Analysis of Information Dependencies…

- ***Example***: Searching the maximal value among matrix ***A*** elements:
  - the procedures of maximal value search in separate rows (*rowwise striped decomposition*) of the original matrix ***A*** can be used as *the basic subtasks*,
  - the structure of information dependencies looks as follows:



Stages of Parallel Algorithm Design…

Analysis of Information Dependencies

- As in the previous case, for estimating the correctness of the stage implementation it will be useful to answer the questions:
  - Does the computational complexity of subtasks corresponds to the intensity of their information communications?
  - Is the intensity of information communications the same for different subtasks?
  - Is the scheme of information communications local?
  - Will the information dependencies hinder parallel solution of subtasks?

Stages of Parallel Algorithm Design…

Scaling the Subtasks…

- Decomposition scheme of parallel computations has to be scaled, if the number of available subtasks differs from the number of processors planned to be used
- To decrease the number of subtasks, it is necessary to perform the *aggregation* of calculations:
  - The anew determined subtasks should have close computational complexity and the number and the intensity of information communications among subtasks should stay at the minimum possible level,
  - The first candidates for aggregation are subtasks with a high degree of information interdependence.
- If the available set of subtasks is not enough to load all the available processors, it is necessary to perform further decomposition of computations.

Stages of Parallel Algorithm Design…

Scaling the Subtasks

- The list of questions for evaluation of the scalability stage correctness looks as follows:
  - Will the computation locality deteriorate after scaling the available subtask set?
  - Are the subtasks after scalability of the same computation and communication complexity?
  - Does the number of subtasks correspond to the number of the available processors?
  - Do scalability schemes depend parametrically on the number of processors?

Stages of Parallel Algorithm Design…

Distributing the Subtasks among Processors…

- The management of load distribution for processors is possible only for computer system with the distributed memory. For multiprocessors (the systems with shared memory) load distribution is usually performed by the operating system automatically
- Stage of distributing the subtasks among processors is redundant, if the number of subtasks coincides with the number of available processors, and the network topology of a computer system is the complete graph

Stages of Parallel Algorithm Design…

Distributing the Subtasks among Processors…

- *Efficiency of processor utilization* is the relative fraction of time, during which the processors were being used for solving the original problem
- The ways to achieve good results in efficiency:
  - Uniform distribution of computational load among processors,
  - Minimal number of messages transmitted among processors.
- The optimal solution of the subtask distribution problem is based on the connectivity analysis of the *"subtasks-messages"* graph

Stages of Parallel Algorithm Design…

Distributing the Subtasks among Processors…

- *Dynamic distribution* of computational load is necessary if the computation scheme can change in the course of solving the problem
- Widely used method of dynamic management, which is usually termed as the *manager-worker scheme*

Stages of Parallel Algorithm Design…

Distributing the Subtasks among Processors…

- **Manager-Worker Scheme:**
  - A single processor-*manager* is selected in the system, this processor has access to the information of all the available subtasks,
  - The rest of the processors are *workers*. They apply to processor-manager to obtain the next computation load,
  - The new subtasks generated in the process of the computations are transmitted back to the processor-manager and can be obtained for solving, if processors-workers apply to the processor-manager,
  - Completion of computations occurs as soon as the processors-workers have accomplished all the subtasks transmitted to them, and the processor-manager does not have any computation jobs for execution.

Stages of Parallel Algorithm Design

Distributing the Subtasks among Processors

- The list of questions for checking the subtask distribution stage is as follows:
  - Will the placement of several subtasks on one processor lead to the increase of additional computational expenses?
  - Is it necessary to balance the calculations dynamically?
  - Can the processor-manager be a bottleneck in manager-worker scheme?

Example: Parallel Algorithm Design for the *n*-Body Problem…

- ***Gravitational problem of N bodies*** as many problems in the field of physics, can be reduced to data processing operations for each pair of objects of the available physical system:
  - Let there be a great number of bodies (planets, stars, etc.) and the mass, the initial position and the velocity are known for each of the bodies,
  - Under the influence of gravity the position of the bodies changes,
  - To solve the problem it is required to calculate all the system's parameters during a certain given time interval.

Example: Parallel Algorithm Design for the *n*-Body Problem…

- For modeling the time interval is usually divided into small time segments. Further on at each modeling step the forces affecting each body are computed. Then the velocities and positions of the bodies get renewed
- The evident algorithm for solving the problem of N bodies consists in consideration at each modeling step of all the pairs of physical system objects and carrying out the necessary computations for each pairs of objects obtained
- As a result, the time of one modeling iteration execution will be the following (τ is the time of parameter recalculation for a pair of bodies):

$$T_1 = \tau N(N-1)/2$$

Example: Parallel Algorithm Design for the *n*-Body Problem…

- **Problem Decomposition into Subtasks:**
  - *Basic subtask* is all the set of computations related with processing data of a physical system's body
- **Analysis of Information Dependencies:**
  - Calculations become possible only if each subtask has data from all the bodies of the physical system,
  - Before each modeling iteration starts, each subtask must get all the necessary data from the other subtasks of the system,
  - This variant of data communication is usually called *multi-node gather* or *all-gather.*

Example: Parallel Algorithm Design for the *n*-Body Problem…

- **Analysis of Information Dependencies. All-gather operation**
  - **Algorithm 1**: At each iteration all the available subtasks split into pairs, and the data exchange is carried out between the subtasks of each pair (*N-1* iterations).
  - **Algorithm 2**: The first iteration is the same as in the algorithm 1. After accomplishing this step, the subtasks will contain not only their own data, but also the data of the subtasks they formed pairs with. As a result, at the second iteration of data gather it will be possible to form pairs of subtasks to exchange data simultaneously about two bodies of the physical system. Thus after accomplishing the second iteration each subtask will contain the information about four bodies of the system, etc. It is evident that this method of information exchange realization allows to terminate the necessary procedure in $log_2 N$ iterations.

Example: Parallel Algorithm Design for the *n*-Body Problem…

- **Scaling and Distributing the Subtasks among Processors**
  - If the number of bodies in a physical system ***N*** exceeds the number of processors ***p***, the subtasks discussed previously must be aggregated. They should be united in the framework of one computation subtask for the group of (***N/p***) bodies,
  - After this aggregation the number of subtasks and the number of processors will coincide,
  - Distributing the subtasks among processors, it will only be necessary to provide direct communication links for processors with subtasks, which execute information exchanges of data gather operations.

Example: Parallel Algorithm Design for the *n*-Body Problem…

❑ Efficiency Analysis for Parallel Computations…

- As the suggested algoritms differ from each other only by the methods of carrying out information exchanges, it is enough to determine the duration of multi-node gather operation in order to compare them. Let us use the Hockney model to evaluate the data communication time,
- The duration of data gather operation for the first parallel computation algorithm can be expressed as:

$$T_p^1(comm) = (p-1)(\alpha + m(N/p)/\beta)$$

- In the second algorithm for iteration number i the size of transmitted data is estimated as $2^{i-1}(Nm/p)$. As a result, the duration of data gather operation execution in this case may be determined by the following equation:

$$T_p^2(comm) = \sum_{i=1}^{\log p}(\alpha + 2^{i-1} m(N/p)/\beta) = \alpha \log p + m\,(N/p)(p-1)/\beta$$

Example: Parallel Algorithm Design for the *n*-Body Problem

- **Efficiency Analysis for Parallel Computations**

  The comparison of the obtained expressions shows that the second developed parallel computation method is far more efficient, involves fewer communication expenses and allows better scalability, if the number of processors in use is increased

Summary….

- The main requirements to parallel algorithms are considered, namely it has to be provided:
  - Equal computational load among processors,
  - Low intensity of data communications between subtasks.
- Two models to describe the computation parallel schemes are discussed:
  - The “subtasks-messages” model, which can be used at the stage of parallel algorithm design,
  - The “processes-channels” model, which can be applied at the stage of parallel program development

Summary

- The methodology of parallel algorithm design is proposed, which includes the following stages:
  - Problem Decomposition into Subtasks,
  - Analysis of Information Dependencies,
  - Scaling the Subtasks,
  - Distributing the Subtasks among Processors
- Finally the problem of *N* bodies is considered to demonstrate how to use the proposed methodology

Discussions

- What are the initial assumptions for the possibility to apply the methodology of parallel algorithm design discussed in the section?
- What are the basic stages of parallel algorithm design?
- What basic requirements should be met in parallel algorithm design?
- Which parallel computation method was developed for solving the gravitational problem of *N* bodies?
- Which method of multi-node gather operation is the most efficient?

Exercises…

- Design a scheme of parallel computations using the methodology described in the section for developing the parallel methods:
  - For the problem of searching the maximum value among minimal elements of matrix rows (such a problem occurs in matrix games)

$$y = \max_{1 \le i \le N} \min_{1 \le j \le N} a_{ij}$$

(pay special attention to the situation when the number of processors exceeds the matrix size, i.e. $p>N$),

  - For the problem of computing a definite integral using the method of rectangles

$$y = \int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f_i,\ f_i = f(x_i),\ x_i = i\,h,\ h = (b-a)/N$$

Exercises

- Implement the discussed methods of multi-node gather operation and compare their execution time. Associate the obtained time characteristics with the available theoretical estimations. Compare them with the time of executing the function MPI_Allgather of MPI.

References…

- **Andrews, G. R.** (2000). Foundations of Multithreaded, Parallel, and Distributed Programming.. – Reading, MA: Addison-Wesley
- **Bertsekas, D.P., Tsitsiklis, J.N.** (1989) Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.
- **Buyya, R.** (Ed.) (1999). High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. - Prentice Hall PTR, Prentice-Hall Inc.

References

- **Kahaner, D., Moler, C., Nash, S.** (1988). Numerical Methods and Software. – Prentice Hall
- **Foster, I.** (1995). Designing and Building Parallel Programs: Concepts and Tools for Software Engineering. Reading, MA: Addison-Wesley.
- **Quinn, M. J.** (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
- **Wilkinson, B., Allen, M.** (1999). Parallel programming. – Prenrice Hall.

Next Section

- **Parallel Methods for Matrix-Vector Multiplication**

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.