15. The Software System ParaLab for Learning and Investigations of Parallel Methods

15.1. Introduction

The software system **Parallel Laboratory** (**ParaLab**) provides the possibility of carrying out computational experiments for studying and researching the parallel algorithms of solving complicated computational problems.

The system may be used for organizing a set of laboratory works on various courses in the area of parallel programming. This laboratory works will allow the learners to do the following:

- *To model multiprocessor systems* with various data communication network topologies;
- *To obtain the visual presentations of the computational processes and data communication operations* which take place in case of parallel solving various problems;
- *To construct the efficiency estimations* of the parallel methods to be studied.

This set of laboratory works may be organized on "conventional" single-processor computers, which operate under OS Microsoft Windows 2000 or OS Microsoft Windows XP (the time sharing mode of parallel computation simulation). Besides the simulation mode, the system ParaLab provides remote access to the available multiprocessor system for carrying out the experiments in "real" parallel computation mode. It makes possible to compare the results of simulation to the results of "real" parallel computations.

In general, the system ParaLab is *the integrated environment for studying and researching the parallel algorithms* of solving complicated computational problems. A wide set of available means to visualize the process of carrying out an experiment and to analyze the obtained results allows to study the parallel method efficiency on various computer systems, to make conclusions concerning the scalability of the algorithms and to determine the possible parallel computation speedup.

The processes of study and research realized by the system ParaLab are aimed at mastering the fundamentals of parallel computation theory. They allow the leaners to form the basic concepts of the models and methods of parallel computations through observation, comparison and analysis of various visual graphic forms demonstrated in the course of the experiment execution.

The system ParaLab is mainly used for *training* purposes. It may be used by University professors and students for teaching/studying and researching parallel algorithms of solving complicated computational problems using the set of the laboratory works, applied to various courses in the area of parallel programming. The system ParaLab may be also used to conduct research into estimating the efficiency of parallel computations.

For those who only start to study the problem of parallel computations, the system ParaLab is very useful, as it allows them to master the parallel programming methods. Experienced users may use the system in order to estimate the efficiency of new parallel algorithms, which are being developed.

15.2. System Overview

ParaLab is a software system, which allows the user both to carry out real parallel computations on a multi processor system and to simulate such experiments on a single serial computer visualizing the process of solving a complicated computational problem.

ParaLab provides the following possibilities for the user to carry out the simulation experiments:

- *To determine the topology* of the parallel computer system for carrying out the experiments, *to set the number of processors* in the topology, *to determine the performance* of the processors, *to select the parameters* of the communication network and *the method of communication*;
- *To state the computational problem*, for solving of which in the system there are corresponding parallel algorithms and *to set the problem parameters*;
- *To select the parallel method* for solving the given problem;
- *To set visualization parameters* for the desirable rate of demonstration and *to choose the method of visualizing* the data transmitted among the processors.
- *To carry out the experiment* for parallel solving the given problem. In this case there may be formed several different jobs for executing the experiments with various types of multiprocessor systems, problems or parallel computation methods. The jobs may be carried out simultaneously (in the time sharing mode). Simultaneous executing several jobs allows us to compare the dynamics of solving the problem by various methods, on various topologies, with various parameters of the problem. In case of carrying out *a series of experiments*, which require long computations, the system provides the opportunity to carry out such experiments in the automatic mode. The results are stored in *the experiment log*, which makes possible to analyze the obtained data;
- *To accumulate and analyze the results of the executed experiments.* Using the stored results it makes possible to visualize various graphic forms which allow demonstrating different dependences of parallel computations (execution time, speedup, efficiency) with respect to the problem parameters or computer system.

One of most important feature of the system is that ParaLab provides different modes of performing experiments. They may be carried out in *the simulation mode* using single processor with no any special software like libraries of message passing. Moreover, in the framework of the ParaLab system there are the following possibility of performing *a real computing experiment* using *remote access* to a computing cluster.

In constructing the dependences of the time characteristics with respect to the problem parameters or the computer system for the experiments carried out in the simulation mode, the theoretical estimations are used in accordance with the available parallel computation model (see, for instance, Hockney (1988)). For real parallel experiments on multiprocessor systems the dependences are constructed with the use of the results of the experiments carried out.

Besides, the system provides the possibility to accumulate all the obtained results in *the experiment log*. Stored results allow performing the data analysis. All experiment data can be extracted from the experiment log to restore the previous state of the experiment – that allows to repeat the experiment again. Also it allows to continue computations from the suspended state

The study and investigations provided by ParaLab allow the students to master the parallel computations and assist them to obtain the understanding how to design parallel algorithms for solving complicated time-consuming problems in various areas of applications.

Example

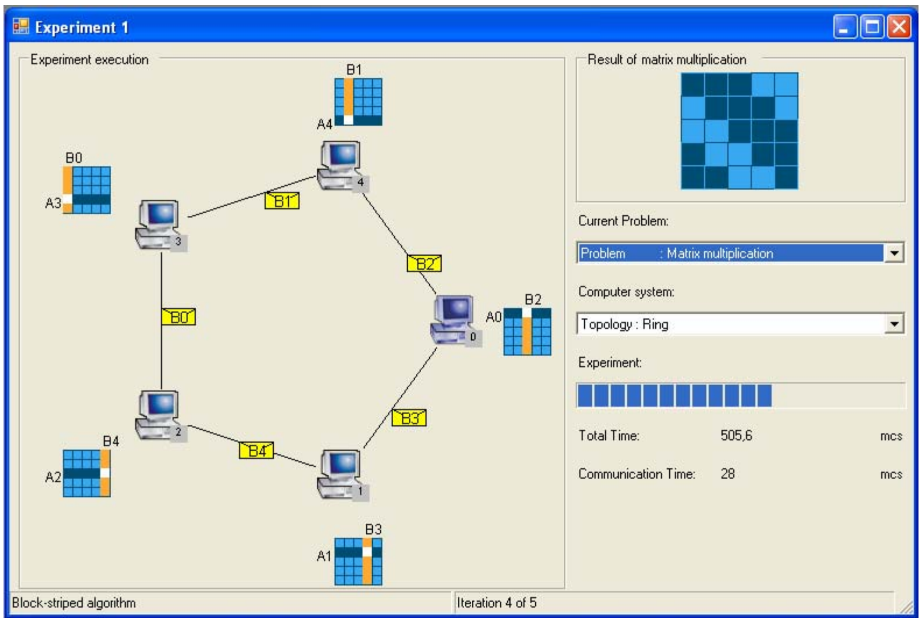
To carry out the example available in the distribution kit of the system, the user should do the following:

- Choose the menu option **Start** and execute the command **Open**,
- Choose the line **first.prl** in the list of file names and press the button **Open,**
- Choose the menu option **Run** and execute the command **In active window.**

As a result of these operations, the window for execution of the computational experiment (see Figure 15.1) will appear on the display. In the window the process of solving the problem of matrix multiplication by means of the block-striped algorithm is shown.

The processors of the computer system and the structure of communication lines are presented in the area "Experiment execution". The data, processed by a processor at any given moment of the algorithm execution, is shown alongside the processor (for the block-striped matrix multiplication algorithm this is several sequential rows of matrix A and several sequential columns of matrix B). The data communications carried out by the processors is shown by means of dynamically moving yellow rectangles ("envelopes").

The current state of the result matrix C is shown in the area "Result of matrix multiplication". As the result of multiplication of the initial stripes of matrices A and B is a block of the matrix C, the resulting matrix has the block structure. The blocks, which have been already computed, are marked by the dark blue color, the blocks, which are to be determined, are marked the light blue color.



Current Problem: Problem : Matrix multiplication

Computer system: Topology : Ring

Experiment:

Total Time: 505,6 mcs

Communication Time: 28 mcs

Block-striped algorithm | Iteration 4 of 5

Figure 15.1. The window of the computational experiment

The list "Current problem" holds the parameters of the problem being executed: the name, the method of solving, and the amount of the initial data. The list "Computer system" holds the characteristics of the given computer system: the topology, the number and the performance of the processors, the network characteristics.

The band indicator "Experiment" shows the current stage of the experiment execution. The lines "Total time" and "Communication time" show the time characteristics of the algorithm.

After the experiment has ended and the main menu is restored, the system execution can be terminated by the command **Exit** of the menu option **File.**

15.3. Simulating the Computer System

In order to simulate the computer system, it is necessary to determine the network topology, the number of processors, the performance of each processor, and the characteristics of the communication network (the latency, the bandwidth and the data communication method). It should be noted that the computer system is assumed to be homogeneous in the ParaLab system, i.e. all the processors possess equal performance, and all the communication lines have the same characteristics.

15.3.1. Choosing the Network Topology

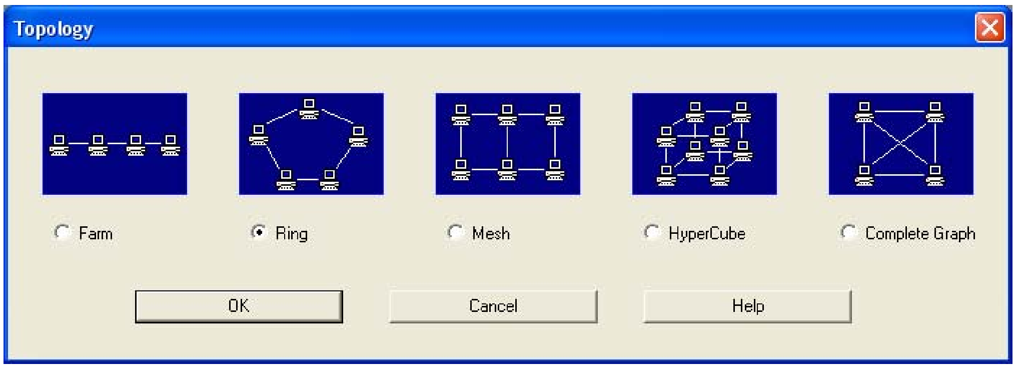
The data communication network topology is defined by the structure of communication lines among the computer system processors. The system ParaLab supports the following network topologies:

- *Completely-connected graph* or *clique*. This is the system, where there is a direct communication line between any pair of processors. As a result, this topology provides the minimum data communication overhead. However, this topology is highly expensive in case of the great number of processors in a system.
- *Linear array* or *farm*. This is the system, each processor of which is connected by communication line only to two neighbors (with the previous and the following ones). This scheme is, on the one hand, easy to realize, and on the other hand, corresponds to the data communication structure in solving many computational problems (for instance, in organizing pipeline computations).
- *Ring*. This topology may be obtained from the linear array by connecting the first and the last linear array processors.
- *Mesh*. The graph of communication lines in this system forms a rectangular two-dimensional grid. This topology may be easily realized. Besides, it may be efficiently used in parallel execution of many numerical algorithms (for instance, in implementation of the chessboard-block matrix multiplication method).
- *Hypercube*. This topology is a particular case of *N*-dimensional grid structure. There are only two processors along each dimension of the grid (i.e. the hypercube contains 2^N processors, if the dimension is *N*). This variant of data communication network is widely used in practice. It has the following distinctive characteristics:
  - Two processors are linked, if the binary presentation of their numbers has only one differing position;
  - Each processor in *N*-dimensional hypercube is connected exactly with *N* neighbors;
  - An *N*-dimensional hypercube may be partitioned into two (*N-1*)-dimensional hypercubes (there exist *N* different partitions of this type);
  - The length of the shortest path between any two processors coincides with the number of differing bit values in the numbers of the processors (this value is known as *the Hamming distance*).

How to use the system ParaLab

1. **<u>The system start</u>.** To start the ParaLab system, select the system pictogram and click it by the left mouse button twice (or press the **Enter** key). Then execute the command **New** (the menu option **Start**) and press **OK** in the dialog window **Experiment name** (the user may change the name of the created window before pressing **OK**).

2. **<u>Choosing the network topology</u>.** In order to choose the network topology, the command **Topology** of the menu option **System**. In the dialog window, which appears on the screen (see Figure 15.2), choose the pictogram of the desired topology by the double click of the left mouse button or in the bottom in the area of the corresponding selection button (radio button). Pressing the button **Help** provides getting the reference information on the implemented topologies. Press **OK** to confirm the choice or **Cancel** to return to the main menu.



Farm [✓ Ring] Mesh HyperCube Complete Graph

Figure 15.2. The dialog window for choosing topology

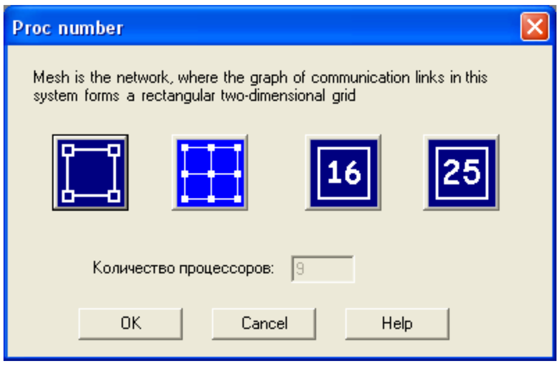
15.3.2. Setting the Number of Processors

The system ParaLab allows the user to set the desirable number of processors for the selected topology. The choice of the system configuration is performed in accordance with the type of the topology used. Thus, for instance, the number of processors in a two-dimensional grid must be a perfect square (the sizes of the grid both vertically and horizontally are the same), while the number of the processors in a hypercube must be a power of 2.

The processor performance in the ParaLab system is measured by the number of floating point operations per second (flops). It should be noted that to esstimate the execution time of the experiment, it is assumed that all the computer instructions correspond to the same floating point operation.

How to use the system ParaLab

1. <u>**Setting the number of processors**</u>. In order to choose the number of processors, it is necessary to select the command **Number of processors** of the menu option **System**. In the dialog window (Figure 15.3), which appears on the display, several pictograms with schematic pictures of the number of processors in the topology are given. To choose the desired pictogram click it by the left mouse button. The pictogram, which corresponds to the current number of processors, is marked by the light blue color.

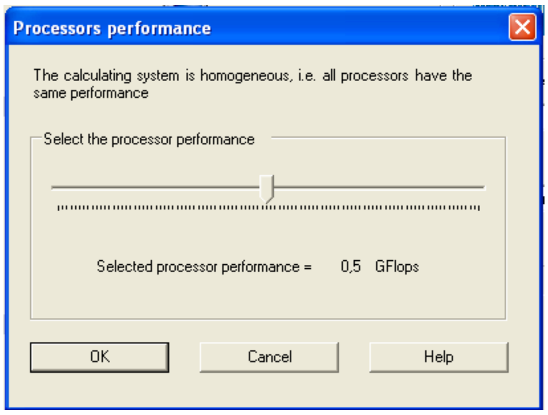

Proc number

Mesh is the network, where the graph of communication links in this system forms a rectangular two-dimensional grid

Figure 15.3. The Dialog window for setting the number of processors

Press **OK** to confirm the choice or **Cancel** to return to the main menu without changing the number of processors.

2. <u>**Setting the Processor Performance**</u>. In order to set the performance of the processors execute the command **Processor Performance** of the menu option **System**. Then at the dialog window **Processor Performance** (Figure 15.4) the desirable value of the performance can be set by the track bar. Press the button **OK** or the key **Enter** to confirm the choice. Otherwise press the button **Cancel** or the key **Escape** to return to the main menu without any changes.


Processors performance

The calculating system is homogeneous, i.e. all processors have the same performance

Select the processor performance

Selected processor performance = 0,5 GFlops

Figure 15.4. The dialog window for setting the processor performance

15.3.3. Setting the Network Characteristics

The time of data transmission among the processors determines the communication overhead of the parallel algorithm execution in a multiprocessor system. The main set of parameters, which makes possible to estimate the data communication time, contains the following values:

– *Latency* (t_s)**.** It is the time, which characterizes the duration of preparing a message for transmission, as well as the duration of the searching for the route in the network etc.

– *Network bandwidth (R)*. It is defined as the maximum amount of data, which can be transmitted in a certain unit of time through a data communication channel. This characteristic is measured, for instance, in Mbits per second.

Among the data communication methods, implemented in the systems ParaLab, there are the following two well-known communication methods (see, for instance, Kumar, et al. (1994)). The first method is aimed at *passing messages* as indivisible information blocks (*store-and-forward routing* or *SFR*). In case of this approach the processor, which contains the original message, prepares all the amount of data for transmission, determines the transit processor, via which the data may be delivered to the target processor, and starts the data transmission operation. The processor, which is to receive the message, performs, first of all, the reception of all the transmitted data, and only then starts to transmit the data further along the route. The time of data transmission T for passing the messages of size m along the route of length l can be determined by the following expression:

$$T=(t_s+m/R)\cdot l\ .$$

The second communication method is based on representing the transmitted messages as a set of information blocks of smaller sizes (*packets*). As a result, the data transmission may be reduced to *passing packets*. In case of this communication method (*cut-through routing* or *CTR*) the transit processor may perform transmitting the data along the chosen route directly after the reception of the next packet without waiting for the termination of receiving all the message data. The number of the transmitted packets is equal to the following:

$$n=\lceil m/(V-V_0)\rceil+1\ ,$$

where V is the packet size, and the value V_0 determines the amount of the control data transmitted in each packet (*the packet header*). As a result, the communication time in this case is equal to the following value:

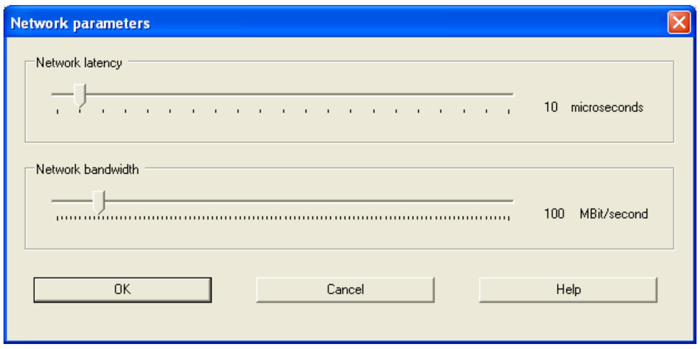
$$T=t_s+(V/R)\left(l+\lceil m/(V-V_0)\rceil\right)=t_s+(V/R)\left(l+n-1\right)$$

(the brackets $\lceil\ \rceil$ denote the rounding operation to the integer from above).

To compare the obtained expressions, it can be noted that in case when the route length is more than one, the method of packet passing leads to faster data communications. This approach, besides, reduces the need for the memory for storing the transmitted data in order to arrange the reception/transmission of messages. Also different communication channels may be used simultaneously for the packet passing.

How to use the system ParaLab

1. **Setting the network characteristics**. To set the network characteristics, execute the command **Network parameters** of the menu option **System**. It is possible to set the latency in microseconds and the network bandwidth in Mbit/sec. by means of the track bars in the dialog window (Figure 15.5). Press **OK** to confirm the choice. In order to return to the main system menu without changing the parameters, press the button **Cancel**.


Network parameters

Network latency

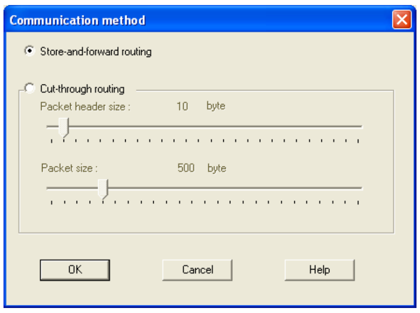
10 microseconds

Network bandwidth

100 MBit/second

Figure 15.5. The dialog window for setting the network parameters

2. **Setting the data communication method**. In order to set the data communication method to be used in the computational experiment, it is necessary to execute the command **Communication Method** of the menu option **System**. It is necessary to click on the left mouse button in the area of the radio button, which corresponds to the desired data communication method in the dialog window (Figure 15.6). If the method of packet passing has been chosen, it is necessary to set the packet length and the packet header length in bytes by the track bars. To confirm the choice of the method and its parameters, press **OK**.


Communication method

(•) Store-and-forward routing

( ) Cut-through routing

Packet header size : 10 byte

Packet size : 500 byte

Figure 15.6. The dialog window for setting the data communication method

3. **<u>The termination of the system execution</u>**. To terminate the execution of the system ParaLab, execute the command **Exit** (the menu option **File**).

15.4. Selecting the Problem and the Parallel Method

For parallel solving computational problems, the process of computations should be presented as a set of independent computational procedures, which may be executed on independent processors.

The general scheme of organizing such computations may be presented in the following way:

- Partitioning the computational process into parts, which may be executed simultaneously;
- Distributing computations among the processors;
- Providing the interactions of the computations executed in parallel.

The following ways to obtain the parallel computation methods are possible:

- Developing new parallel algorithms;
- Parallelizing sequential algorithms.

Parallel algorithm will be efficient, if the following requirements are met:

- Uniform processor loading;
- Low intensity of the processor interaction.

The following widely used parallel algorithms applied to solving complicated computational problems in various scientific and technical applications are implemented in the system ParaLab: the algorithms of data sorting, the algorithms of matrix operations, the algorithms of solving the systems of linear equation and graph processing.

How to use the system ParaLab

1. **<u>Setting the problem</u>**. To choose a problem among those implemented in the system, execute the menu option **Problem** and select one of the following commands: **Sorting**, **Matrix-Vector Multiplication, Matrix multiplication, Solving of linear equation system (SLE), Graph processing**. The selected problem becomes the current one in the active window.

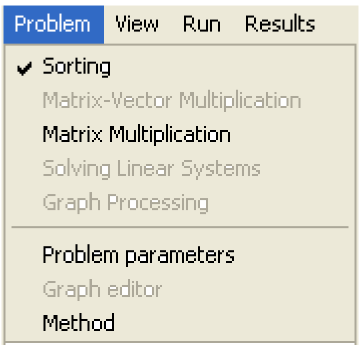
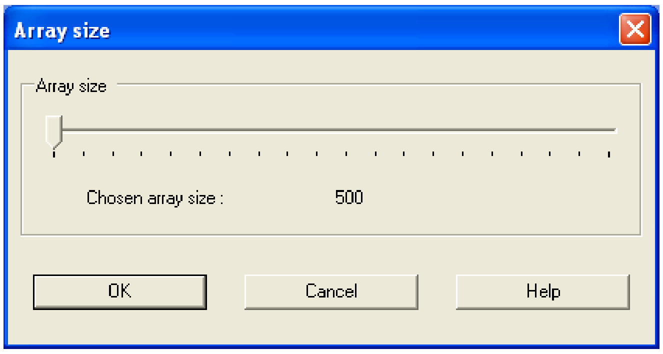


Figure 15.7. Computational problems implemented in ParaLab

2. **<u>Setting the problem parameters</u>.** The main problem parameter in the system ParaLab is the initial data amount. For the problem of sorting this is the size of the array. For the matrix operations and the problem of solving a system of linear equations this is the dimension of the matrices. For the problem of processing graphs this is the number of graph vertices. In order to select the problem parameters it is necessary to execute the command **Problem parameters** of the menu option **Problem**. It is necessary to set the desired initial data amount in the dialog window

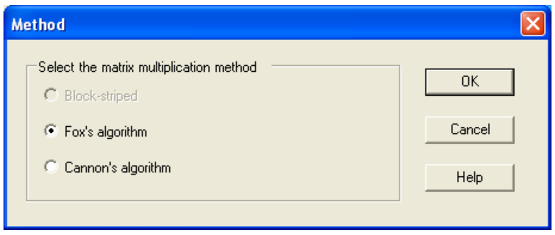
(Figure 15.8) by the track bars. Press **OK** (**Enter**) to confirm the parameter setting. To return to the main system menu without saving the changes, press the button **Cancel** or the key **Escape**.



Chosen array size : 500

Figure 15.8. The dialog window for setting the matrix multiplication problem

3. **Choosing the method for solvig the problem.** To choose the method for solving the problem, execute the command **Method** of the menu option **Problem**. Select the desirable method in the dialog window by the mouse and click **OK** to confirm the choice, press **Cancel** to return to the main menu.



Select the matrix multiplication method

( ) Block-striped
[✓ Fox's algorithm]
( ) Cannon's algorithm

Figure 15.9. The dialog window of choosing the method for solving the matrix multiplication problem

15.4.1. Data Sorting

Data sorting is one of typical problems of data processing and is usually considered to be a problem of redistributing the elements of a given sequence of values

$$S = \{a_1, a_2, ..., a_n\}$$

in the order of monotonic increase or decrease

$$S \sim S' = \{(a_1^{'}, a_2^{'}, ..., a_n^{'}) : a_1^{'} \le a_2^{'} \le ... \le a_n^{'}\}.$$

The computational complexity of the sorting procedure is considerably high. Thus, for a number of well-known methods (bubble sort, insertion sort etc) the number of the necessary operations is determined by the square dependence with respect to the number of the ordered data

$$T_1 \sim n^2.$$

For more effective algorithms (such as merge sort, Shell sort, quick sort) the complexity is determined by the following value:

$$T_1 \sim n \log_2 n.$$

This expression gives the lower estimation of the necessary number of operations for sorting the set of *n* values. The algorithms with lower complexity may be obtained only for particular problem variants.

Data sorting speedup may be provided by means of using several (*p>1*) processors. In this case the data is distributed among the processors. In the course of calculations the data are transmitted among the processors and one part of data is compared to another one. As a rule, the resulting (sorted) data are distributed among the processors. To regulate such distribution a scheme of consecutive numeration is introduced for the processors. It is usually required that after sorting termination the data located on the processors with smaller numbers should not exceed the values on the processors with greater numbers.

The following methods of data sorting are implemented in the ParaLab system: *bubble sort*, *Shell sort*, *quick sort*. The original (sequential) variants of these methods are discussed in many books (see, for instance, Knuth (1997)). The ways of the parallel execution are discussed in Section 10.

15.4.1.1. Bubble Sort

The bubble sort is rather complicated to be parallelized: the comparison of the neighboring element pairs is performed strictly sequentially. The parallel variant of the algorithm is based on *the method of odd-even permutation*. In case of this method the elements of the following pairs are compared at the odd iteration:

$$(a_1, a_2),\ (a_3, a_4),\ \ldots,\ (a_{n-1}, a_n).$$

If the pair is not ordered, its elements are permuted. At the even iteration the following pairs are ordered:

$$(a_2, a_3),\ (a_4, a_5),\ \ldots,\ (a_{n-2}, a_{n-1}).$$

As it can be seen, the sequence may be sorted out after *n-1* iterations. The parallel variant of the algorithm is given in more detail in section 10.

Tasks and Exercises

1. Start the ParaLab system and create a new experiment. Select the menu option **Problem** and make sure that the problem of sorting is the current one in the active window. Open the dialog window of the method selection, and observe, what sorting algorithms may be performed on the current topology. As the current topology in case of creating an experiment on default is the ring topology, the only possible algorithm is the bubble sorting algorithm. Close the dialog window and return to the main menu.

2. Perform several experiments changing the size of the initial data. To carry out the experiment, select the command **In active window** of the menu option **Run**. Analyze the time characteristics of the experiments presented in the right lower part of the window.

3. Carry out several computational experiments changing the number of the computer system processors. Analyze the time characteristics obtained.

15.4.1.2. Shell Sort

The parallel algorithm of Shell sorting may be obtained as the generalization of the parallel bubble sorting method. The main difference between them is that at the first iterations of Shell algorithm the pairs of elements are compared which are located far from each other in the initial data set (to sort such pairs in bubble sorting it might be necessary to make rather a big number of iterations). The parallel generalization of Shell sorting algorithm is described in detail in Section 10.

Tasks and Exercises

1. Start the ParaLab system. Select the ring topology and set the number of processors equal to 8. Carry out an experiment on the execution of the bubble sorting algorithm.

2. Set the hypercube topology and the number of processors equal to 8 in the window of the computational experiment.

3. Open the dialog window of the method selection and observe, which sorting algorithms may be executed on this topology. Select Shell sorting method. Close the window.

4. Carry out the computational experiment. Compare the number of iterations performed in solving the problem by means of the Shell method to the number of the bubble sorting algorithm iterations (the number of iterations is shown to the right in the status bar). Make sure that in case of carrying out the experiment by means of the Shell algorithm, it is necessary to perform a smaller number of iterations for sorting the array.

5. Carry out the experiment with the use of the Shell method several times. Make sure that the number of iterations is not constant and depends on the initial array.

15.4.1.3. Quick Sort

The quick sorting algorithm suggested by Hoare C.A.R. is based on the sequential subdividing the sorted data into blocks of smaller sizes in such a way that the ordering relation is provided among the values of different blocks (for any pair of blocks all the values of one of the blocks do not exceed the values of the other one). The division of the original data into the first two parts is performed at the first iteration of the method. A certain *pivot element* is selected for providing this division, and all the values of the data, which are smaller that the pivot element, are transferred to the first block being formed. All the rest of the values form the second block of the sorted data. These rules are applied recursively for the two created blocks on the second iteration of the sorting etc. If the choice of the pivot elements is adequate, than the initial data array appears to be sorted after the execution of $log_2 n$ iterations.

In case of parallel generalization of the algorithm, the ordering relations are provided among the elements of the sorted set located on the neighboring processors with the hypercube network topology.

Section 10 provides a more detailed description of the quick sorting algorithm.

🖶 *Tasks and Exercises*

1. Start the system ParaLab. Set the hypercube topology and the number of processors equal to 8 in the active window.

2. Carry put three sequential experiments using the three different sorting algorithms: bubble sort, Shell sort, quick sort. Compare the time characteristics of the algorithms. They are shown in the right bottom part of the window. Make sure that quick sorting requires the least algorithm execution time and the least data communication time.

3. Change the amount of the initial data (execute the command **Problem parameters** of the menu option **Problem**). Carry out the experiments again. Compare the time characteristics.

4. Change the number of processors (execute the command **Number of processors** of the menu option **System**). Carry out the computational experiments and compare the time characteristics.

15.4.2. Matrix-Vector Multiplication

The result of *multiplying the matrix A of order* $m \times n$ *by vector b*, which consists of *n* elements, is the vector *c* of size *m*, each *i*-th element of which is the result of inner multiplication of *i*-th matrix *A* row (let us denote this row by a_i) by vector *b*:

$$c_i = (a_i, b) = \sum_{j=0}^{n-1} a_{ij} b_j \text{ , } 0 \le i \le m-1 .$$

Thus, obtaining the result vector *c* can be provided by the set of the same operations of multiplying the rows of matrix *A* by the vector *b*. Each operation includes multiplying the matrix row elements by the elements of vector *b* (*n* operations) and the following summing the obtained products (*n-l* operations). The total number of necessary scalar operations is the value

$$T_1 = m \cdot (2n - 1) .$$

The repetition of the same computational operations for different matrix elements is typical of matrix–vector multiplication. In this case we can say that there exist *data parallelism*. As a result, the problem to parallelize matrix–vector multiplication can be reduced to matrix distributing among the processors of the computer system. The choice of matrix distribution method determines the use of the definite parallel computation method. The availability of various data distribution schemes generates a range of parallel algorithms of matrix–vector multiplication.

The most general and the most widely used matrix distribution methods consist in partitioning data into *stripes* (vertically and horizontally) or rectangular fragments (*blocks*).

In more detail the problem of matrix-vector multiplication is considered in Section 7.

15.4.2.1. Rowwise Block-striped Decomposition

This algorithm is based on representing a matrix as continuous sets (*horizontal stripes*) of rows. The stripes are distributed among the computational system processors. The vector *b* is copied onto all the processors. Multiplying stripes of the matrix *A* and the vector *b* (this operation may be carried out by the processors in parallel) leads to obtaining a block of the result vector *c*. To combine the results of the computations and to obtain the complete vector *c* on each of the computer system processors, it is necessary to carry out the all gather operation.

🖶 *Tasks and Exercises*

1. Create a new window of the computational experiment in the system ParaLab. Select the problem of matrix-vector multiplication for this window (select the command **Matrix-vector multiplication** of the menu option **Problem**).

2. Open the dialog window of the method selection and check whether the selected method is based on the rowwise matrix partition.

3. Carry out several computational experiments. Pay attention to the dependence of the algorithm execution time with respect to the initial data amount and the number of processors.

15.4.2.2. Columnwise Block-striped Decomposition

The other approach to parallel matrix-vector multiplication is based on the partitioning the initial matrix into continuous sets (*vertical stripes*) of columns. In case of this approach the vector *b* is also partitioned into blocks. The vertical stripes of the initial matrix and vector blocks are distributed among the processors.

At the beginning of the parallel algorithm of matrix –vector multiplication each processor multiplies its vertical stripe of the matrix A by the element block of the vector b_i. As a result, the block of the intermediate result vector $c'(i)$ is obtained on each processor. To obtain the elements of the result vector c, the processors must further exchange their intermediate data.

15.4.2.3. Checkerboard Decomposition

Let us consider the parallel algorithm of matrix-vector multiplication, which is based on another approach to data partitioning. This is partitioning a matrix into rectangular fragments (*blocks*). If data is partitioned in this way, the initial matrix A is represented as a set of rectangular blocks. The vector b is divided into blocks as well. The data blocks are distributed among the processors. The logical (virtual) computer system topology in this case is a rectangular two-dimensional grid. The sizes of the processor grid correspond to the number of the rectangular blocks obtained after partitioning matrix A. The matrix block $A_{i,j}$ and the vector block b_j are located on the processor $p_{i,j}$, which is at the intersection of the i-th row and the j-th column of the processor grid.

After multiplying the blocks of the matrix A and the vector b, each processor $p_{i,j}$ will hold the vector of intermediate products $c'(i,j)$. Element by element summation of the intermediate vectors for each horizontal processor grid makes possible to obtain the result vector c.

Tasks and Exercises

1. Start the system ParaLab. Set the number of processors equal to 4.

2. Carry out three sequential experiments using the three matrix-vector multiplication algorithms based on, correspondingly, horizontal, vertical and chessboard block matrix partitioning. Compare the time characteristics of the algorithms, given in the right bottom part of the window. Make sure that the time characteristics of the algorithms based on block-striped matrix partitioning are practically the same while the execution time of the algorithm based on the chessboard block matrix partitioning has a little bit worse time characteristics.

3. Change the amount of the initial data (execute the command **Problem parameters** of the menu option **Problem**). Repeat the experiments. Compare the time characteristics of the algorithms.

4. Change the number of the processors, set the number of processor equal to 16 (execute the command **Number of processors** of the menu option **System**). Carry out computational experiments and compare the time characteristics.

15.4.3. Matrix Multiplication

The problem of matrix-matrix multiplication is determined by the following expression:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj},\ 1 \le i, j \le n$$

(let us assume for simplicity that the multiplied matrices A and B are square and have the order $n \times n$). As it follows from the given relations, the computational complexity of the problem is the order n^3 .

As it is clear from the definition of matrix multiplication, all elements of the matrix C may be computed independently, if n^2 processors are available. In this case a row of the matrix A and a column of the matrix B will be located on each processor. If the number of processors is less than n^2 this approach leads to the block-striped data decomposition scheme. In this case several rows and columns (stripes) of the initial matrices are located on the processors.

Another widely used approach for developing parallel methods of matrix multiplication is based on the checkerboard block matrix decomposition. In this case the initial matrices A and B and the result matrix C can be considered as a rectangular grid of square blocks of the size $m \times m$. Then the matrix multiplication may be caculated in the following way:

$$\begin{pmatrix} A_{11}A_{12}...A_{1k} \\ \cdots \\ A_{k1}A_{k2}...A_{kk} \end{pmatrix} \times \begin{pmatrix} B_{11}B_{12}...B_{1k} \\ \cdots \\ B_{k1}B_{k2}...B_{kk} \end{pmatrix} = \begin{pmatrix} C_{11}C_{12}...C_{1k} \\ \cdots \\ C_{k1}C_{k2}...C_{kk} \end{pmatrix},$$

where each block C_{ij} of the matrix C is determined by the following expression:

$$C_{ij} = \sum_{l=1}^{k} A_{il} B_{lj} .$$

Calculations of the blocks C_{ij} are also independent. As a result, it is possible to carry out parallel computations by distributing the calculations of separate blocks C_{ij} among the processors of the computer system. This approach provides to develop many effective parallel multiplication methods.

The parallel algorithm of matrix multiplication with the block-striped data decomposition scheme and the two methods (the Fox method and the Cannon algorithms) based on the checkerboard matrix representation are implemented in the system ParaLab.

In more detail the problem of matrix multiplication is considered in Section 8.

15.4.3.1. Block-striped Decomposition

In case of *the block-striped data decomposition scheme* the initial matrices are partitioned into horizontal (for matrix *A*) and vertical (for matrix *B*) stripes. The obtained stripes are distributed among the processors. Multiplication of these stripes (this operation may be executed by the processors in parallel) leads to obtaining a part of the blocks of the result matrix *C*. To compute the remaining blocks of the matrix *C* the combinations of the stripes of the matrices *A* and *B* on the processors are changed. In the simplest way it may be provided, for instance, in case of the ring topology of the computational network and the number of the processors is equal to the number of stripes. Then the changes necessary to perform matrix multiplication may be performed by the cyclic ring shift of the stripes of the matrix *B*. After iterative executions of these operations (the number of the necessary iterations is equal to the number of processors) each processor obtains the set of blocks, which forms a horizontal stripe of the matrix *C*.

The block-striped matrix multiplication algorithm is described in detail in section 8.

Tasks and Exercises

1. Create a new computational experiment window in the system ParaLab. Select the problem of matrix multiplication (select the command **Matrix multiplication** of the menu option **Problem**).

2. Open the dialog window of the method selection and check if the block-striped matrix multiplication method has been selected.

3. Carry out a series of computational experiments. Study the dependence of the algorithm execution time with respect to the initial data amount and the number of processors.

15.4.3.2. Checkerboard Decomposition

In case of *the checkerboard decomposition scheme* it is reasonable to represent the network topology is a rectangular processor grid which corresponds to the block structure of the multiplied matrices (at least it can be provided at the logical level). Then the parallel block-oriented matrix multiplication methods may be described as follows:

- Each grid processor is responsible for computing a block of matrix *C*,
- One block of each processed matrix (matrices *A*, *B* and *C*) is located on each processor at any time of computations,
- The blocks of the matrix *A* are sequentially shifted along the rows and the blocks of the matrix *B* are shifted along the columns of the processor grid in the course of the matrix multiplication,
- As a result of the computations, a block of matrix *C* is computed on each processor. The total number of the algorithm iterations is equal to $\sqrt{p}$ (where p is the number of processors).

Section 8 provides the description of the parallel block-oriented matrix multiplication methods (*methods proposed by Fox and Cannon correspondingly*) is given in more detail.

Tasks and Exercises

1. Set the topology **Grid** in the active window of the computational experiment. Choose the number of processors equal to 9. Select the problem of matrix multiplication as the current problem of the window.
2. Choose the Fox method of matrix multiplication and carry out a computational experiment.
3. Open the additional experiment window. Choose the Cannon algorithm of matrix multiplication and carry out a computational experiment. Analyze the data communication scheme for the execution of the algorithms. Compare the time characteristics of the algorithms.
4. Change the number of processors in the grid topology and make it to be equal to 16. Carry out sequentially computational experiments using the Fox method and the Cannon method. Compare the time characteristics of the experiments.

15.4.4. Solving Linear Equation Systems

The systems of linear equations appear in solving a number of applied problems described by systems of non-linear (transcendent), differential or integral equations. Such systems may appear in the problems of mathematical programming, statistical data processing, function approximation, discretization of boundary differential problems by the method of finite differences or by the finite element method etc.

The linear equation with n unknowns $x_0, x_1, ..., x_{n-1}$ may be determined by means of the following expression:

$$a_0x_0 + a_1x_1 + ... + a_{n-1}x_{n-1} = b$$

where the values $a_0, a_1, ..., a_{n-1}$ and b are constant.

The set of n linear equations

$$\begin{array}{llll} a_{0,0}x_0 & +a_{0,1}x_1 & +...+a_{0,n-1}x_{n-1} & = b_0 \\ a_{1,0}x_0 & +a_{1,1}x_1 & +...+a_{1,n-1}x_{n-1} & = b_1 \\ ... & & & \\ a_{n-1,0}x_0 & +a_{n-1,1}x_1 & +...+a_{n-1,n-1}x_{n-1} & = b_{n-1} \end{array}$$

is called *a system of linear equations* or *a linear system*. In the shorter (matrix) form the system may be represented as

$$Ax = b ,$$

where $A=(a_{i,j})$ is the real matrix of $n \times n$ size, and b and x are vectors of n elements.

The problem of solving a linear equation system for the given matrix A and the vector b is considered to be the problem of searching the value of unknown vector x whereby all the system equations hold.

The following two algorithms are implemented in the ParaLab system: the widely known *Gauss method* and *the conjugate gradient method*. The Gauss method is a direct algorithm of solving systems of linear equations, it finds the exact solution to the system with the nondegenerate matrix in a finite number of steps. The conjugate gradient method belongs to the large class of iterative methods for solving linear equation systems. More detailed information on algorithms for solving systems of linear equations may be found in Section 10.

15.4.4.1. The Gauss Algorithm

The Gauss method consist in sequential execution of two stages. At the first stage – *the Gaussian elimination* - the initial system of linear equations is transformed to the upper triangle form by means of sequential elimination of unknowns (with the help of equivalent transformations). At the *back substitution* (the second stage of the algorithm) the values of the unknowns are calculated. Namely, it is possible to compute the value of the unknown x_{n-1}, from the last equation of the transformed system. After that it is possible to determine the unknown x_{n-2} from the equation next to last and so on.

The close consideration of Gauss method reveals that all the computations are reduced to computational operations over the matrix rows of the linear equation system coefficients. These operations are of the same type. As a result, the parallel implementation of the Gauss algorithm can be based on the principle of data parallelizing. In this case the matrix A may be partitioned into horizontal stripes and the vector b may be partitioned into blocks. The matrix stripes and the vector blocks are distributed among the processors.

Executing of the Gaussian elimination of the method, $(n-1)$ iterations of eliminating unknowns should be performed in order to transform the matrix A into its upper triangle form.

The execution of the iteration i of the Gaussian elimination, $0 \le i < n-1$, includes a number of sequential operations. The processor where the row i of the matrix is located must send it and the corresponding element of vector b to all the processors, which hold the rows with the numbers k, $k > i$. After obtaining of the leading row, the processors perform the subtraction of rows, thus, providing the elimination of the corresponding unknown x_i.

At the back substitution the processors perform the necessary computations for obtaining the unknown values. As soon as a processor determines the value of its variable x_i, the value is transmitted to all the processors, which contain the rows with the numbers k, $k < i$. Further the processors substitute the value of the new unknown and update the values for the vector b.

More detailed information on the parallel implementation of the Gauss method may be found in Section 10.

Tasks and Exercises

1. Set the topology **Complete Graph** in the active window of the computational experiment. Choose the number of processors equal to 10. Choose the problem of solving a system of linear equations as the current problem of the window.

2. Select **The Gauss method** for solving the problem and carry out a computational experiment. Analyze the data communication scheme in the experiment.

15.4.5. Graph Processing

Mathematical models in the form of graphs are widely used in modeling various phenomena, processes, and systems. As a result, many theoretical and applied problems may be solved with the help of various procedures of analyzing graph models. Among these procedures it is possible to select a certain set of typical graph processing algorithms.

The two following parallel algorithms for solving two typical problems are implemented in the system ParaLab: *the Prim algorithm of searching for the minimum spanning tree* and *the Dejkstra algorithm of searching for the shortest path.*

The system ParaLab allows to execute various operations with a graph – it is possible to create a graph, to edit it, to save the graph in a file, to load the file from a file.

More detailed information on algorithms for graph processing may be found in Section 11.

How to use the system ParaLab

1. **Starting the graph editor.** In order to start the graph editor mode the command **Forming a graph** of the menu option **Problem** should be executed (the command is available only if the problem of processing graphs is the current one).

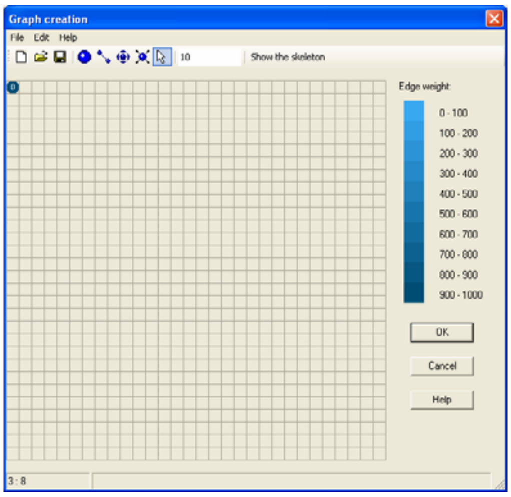


Figure 15.10. The window of the ParaLab graph editor

After the command **Forming a graph** has been executed, a new window appears on the display (Figure 15.10). The graph of the active experiment is shown in the working area of the window. If the graph in the experiment has not been formed the working area of the graph editor is empty.

It is possible to create a new graph, to edit the already existing graphs, to save the graph in a file, and to load a graph into the current experiment.

2. **Creating a new graph.** To create a new empty graph, you should select the menu option **File** and execute the command **New** (or click the left mouse button on the icon of the toolbar). If the graph, which is shown in the active area, has been changed, the user will be offered an opportunity to save the changed graph in a file.

3. **Saving the graph.** To save the graph in a file, select the menu option **File** and execute the command **Save** (or click the left mouse button on the icon of the toolbar). In the dialog window type the new file name or choose any of the existing files for saving the graph in it. Press the button **Save**.

4. **Opening the existing graph.** To load a graph from the file, you should select the menu option **File** and execute the command **Download** (or click the left mouse mutton on the icon of the toolbar). In the dialog window, which appears in this case, select the file name (the graph files in the ParaLab system have the extension plg) and press the button **Open**.

5. **Editing the graph.** It is possible to perform the following operations over the graph located in the working area of the graph editor:

- In order to add one or more new vertices to the graph, select the menu option **Edit** and execute the command **Add vertex** (or click the left mouse button on the icon of the toolbar). The mouse pointer will change in this case: there will appear the symbolic representation of the vertex. The working area of the graph editor is represented as the grid. Click the left mouse button at the grid nodes, where new vertices should be placed.
- In order to link to graph vertices with an edge, choose the menu option **Edit** and execute the command **Add/remove edge** (or click the left mouse button on the icon of the toolbar). The mouse pointer will change and there will appear the image of two vertices linked with an edge. Select one of the graph vertices by clicking it by the

left mouse button. The vertex color will change to the dark red color. Select another graph vertex. There will appear an edge between the two selected vertices. The edge weight is determined randomly. If there was an edge between the first and the second vertices to be selected the edge will be deleted.

- In order to change the graph vertex location select the menu option **Edit** and execute the command **Move vertex** (or click the left mouse button on the icon of the toolbar). After that the mouse pointer will change the form and will look as it is shown on the button of the toolbar. Select one of the graph vertices by clicking it by the left mouse button. The color of the vertex will change into the dark red color. Move the cursor along the working area of the graph editor –the vertex will be moving with the mouse pointer. Click any empty node of the working area and the selected vertex will be moved to this point.
- In order to delete the graph vertex, choose the menu option **Edit** and execute the command **Delete vertex** (or click the left mouse button on the icon of the toolbar). After that the mouse pointer will change its form and there will appear the image of crossed vertex. Click any graph vertex by the left mouse button to delete it.

To terminate any of the modes (**Add vertex, Delete vertex, Move vertex, Add/remove edge**) click the left mouse button on the icon of the toolbar.

6. **Forming a graph by random generator.** A graph may be set randomly. In order to do so, set the number of the graph vertices in the editor on the toolbar, then select the option **Edit** and execute the command **Random graph**.

7. **Editing the graph edge weight.** The colors of the graph edges are of different intensity. The darker the color, the greater the edge weight is. In order to determine the edge weight by sight it is possible to compare the edge color with the scale located to the right. In order to change the edge weight, click it by the right mouse button. A track bar is appeared near the edge. Moving it to the right, the edge weight is increased, and vice versa moving it to the left, the weight is decreased. To save the changes, click any point of the working area or press any key.

8. **Closing the graph editor.** To load the current graph into the current experiment, press the button **OK**. To close the editor without saving changes, press the button **Cancel**.

15.4.5.1. The Prim Algorithm

The spanning tree of the non-oriented graph *G* is the subgraph *T* of the graph *G*, which is a tree and contains all the vertices of *G*. The subgraph weight for the weighed graph is the sum of the edge weights of the subgraph. *The minimum spanning tree (MST)* can be defined as the spanning tree of the minimum weight. The comprehensive interpretation for the problem of finding MST may be, for instance, the practical example of developing a local network of personal computers, which involves the minimum number of linking communication lines.

To solve the given problem *the Prim method* starts from an arbitrary vertex of the graph chosen as the tree root. In the course of sequential iterations the algorithm expands the tree up to MST. The data distribution among the processors is provided so that each graph vertex is located on a processor together with all the information related with the vertex.

With regard to this way of data distribution, the iteration of the Prim algorithm parallel variant is as follows:

- The vertex, which is located at the shortest distance from the current of approximation of MST, that has already been formed, is evaluated (the operations of computing the distances for the graph vertices, that are not included into MST, are independent and may be executed in parallel;
- The vertex is included into MST.

More detailed information on the parallel implementation of the Prim algorithm may be found in Section 11.

15.4.5.2. The Dejkstra Algorithm

The problem of searching for the shortest path in a graph consists in determining the paths of minimum weights from a certain given vertex *S* to all the other graph vertices. The statement of this problem is of great practical importance in various applications, when the edges weights denote time, cost, distance, expenses etc.

The Dejkstra algorithm can be applied to solve this problem. It coincides practically with Prim method. The difference is only in the interpretation and the rule of estimating distances to the graph vertices. In the Dejkstra algorithm this values mean the summary weight of the path from the initial vertex to all the other graph vertices. As a result, at each algorithm iteration such the vertex is chosen as the next one for which the distance from this vertex to the tree nodes is minimal. Further this vertex is included into the tree of the shortest path.

More detailed information on the parallel implementation of the Dejkstra algorithm may be found in Section 11.

Tasks and Exercises

1. Start the system ParaLab. Set the topology **Complete graph** in the active window of the computational experiment. Make the problem of graph processing as the current problem of the window.

2. Execute the command **Graph editor** of the menu option **Problem**. In the graph editor form randomly a graph with ten vertices.

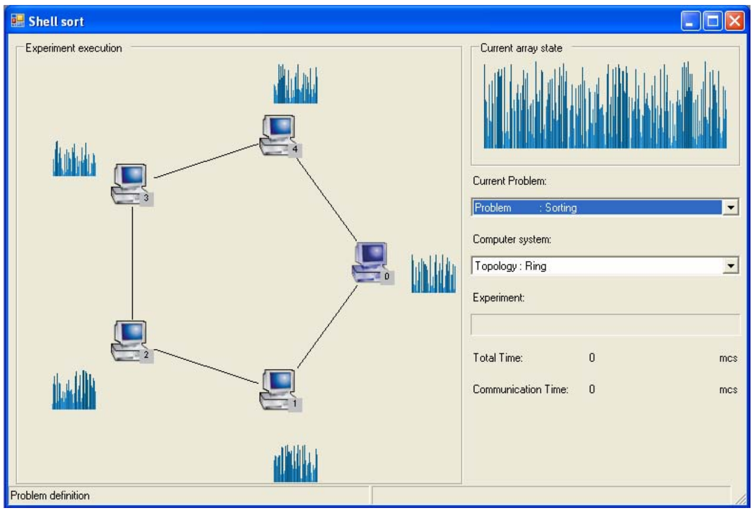
3. Carry out the computational experiment of searching for the minimum spanning tree using the Prim algorithm (carry out the command **Method** of the menu option **Problem**, in the dialog window select **The Prim method**).

4. Carry out several experiments changing the number of processors. Study the dependence of the Prim algorithm time characteristics with respect to the number of processors.

5. Carry out the same number of experiments in order to study the time characteristics of the Dejkstra method.

15.5. Visualizing the Parallel Computations

The system ParaLab provides various forms of graphical demonstration of parallel computation results in order to observe the process of carrying out a computational experiment of solving complicated time consuming computational problems.


Shell sort

Experiment execution

Current array state

Current Problem:

Problem : Sorting

Computer system:

Topology : Ring

Experiment:

Total Time: 0 mcs

Communication Time: 0 mcs

Problem definition

Figure 15.11. The window of the computational experiment

The experiment window (Figure 15.11) is used in order to visualize the data in the course of the experiment execution in the working area of the system ParaLab. In the left part of the window there is the area "Experiment execution". All the processors of a multiprocessor system, linked into a topology, the data, located on each processor, and the data communications among the system processors are shown there. In the right upper part of the window there is the area, which demonstrates the information on the current state of the data, which are the result of the experiment. Depending on the type of the experiment this area is named as:

- "Current array state " if a sorting algorithm is executed,
- “Result of matrix-vector multiplication” in case of matrix-vector multiplication,
- "Result of matrix multiplication " in case of matrix multiplication,
- “Current state of linear system matrix” in case of solving linear systems,
- "Result of graph processing " in case of carrying out graph algorithms.

The information on the current problem is given in the middle of the right part of the current window. The list "Computer system" holds such the characteristics of the system as its topology, the number of processors, the processor performance and communication parameters.

In the right hand bottom corner there is a band indicator of the experiment execution and its current time characteristics.

Additionally in a separate window there may be also visually presented the computations performed by one of the available processors (the user may open the window and choose a processor to observe).

15.5.1. The Area "Experiment Execution"

In this area of the window the processors of a multiprocessor system are shown, which are connected by communication lines into the chosen topology. The processors in the topology are enumerated. In order to find out the number of the processor the processor should be indicated by the mouse pointer. The mouse pointer will change its form and the prompt with the processor number will be shown. The processor can be selected by the double click of the left mouse button and the window "Demonstration of processor activity" will be appeared, where the results of the processor operations can be shown in detail.

The data located on a processor at any moment of the experiment execution is schematically shown at each processor. If the experiment consists in studying some parallel sorting algorithm the processor block of the array to be sorted is demonstrated near the processor. Each array element is shown by a vertical line. The line height and its color intensity characterize the value of the element. The higher and darker the line is, the greater the element is. If the experiment concerns the study of matrix multiplication algorithms, then a matrix frame is given near every processor. The parts of the initial data located on the processor are shown in different colors. The blue color shows a part of the matrix *A* on the processor (a block or a horizontal stripe), the orange color shows a part of matrix *B* (a block or a vertical stripe). If the experiment concerns the study of graph processing algorithms, then a subgraph, which consists of the vertices located on the processor, is shown near each processor.

In the course of carrying out the experiment the data communications among the processors is also demonstrateded in the area "Experiment execution". It may take place in the two modes:

- The mode **Channel View** – the line, along which the communication takes place, is marked by the red color,
- The mode **Packet View** is the visualization of the communications by means of a rectangle (envelope), which moves from processor to processor. If the parallel matrix multiplication algorithms are studied, the number of the block, which is being transmitted, is given on the envelope.

If the graph algorithms are performed all the parallel algorithm iterations are of the same type and their amount is considerably high (it is equal to the number of graph vertices). It will take a lot of time to show all the iterations - to reduce the demonstration time the system ParaLab visualizes only some of them.

How to use ParaLab

1. **Choosing the visualization method of data communications.** To set the method of displaying the processor communications, the command **Demo mode** of the menu option **View** should be executed. In the list, which appears on the screen, select the name of the desirable displaying method.

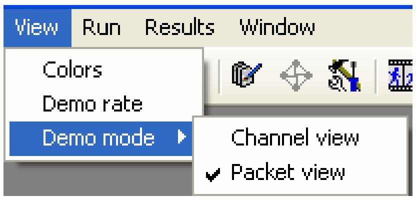
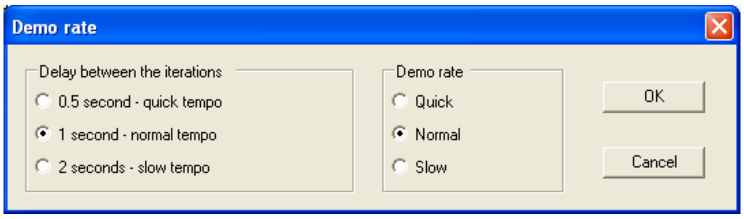


Figure 15.12. Choosing the visualization method of data communications

2. **Choosing the demonstration rate.** In order to choose the demonstration rate it is necessary to execute the command **Demo rate** in the menu option **View**. In the dialog window, which appears on the display (Figure 15.13), the user can choose the value of the delay between the algorithm iterations and the rate of packet movement (time of color highlighting of channels) , while displaying the processor communications.


Demo rate

Delay between the iterations
0.5 second - quick tempo
[✓ 1 second - normal tempo]
2 seconds - slow tempo

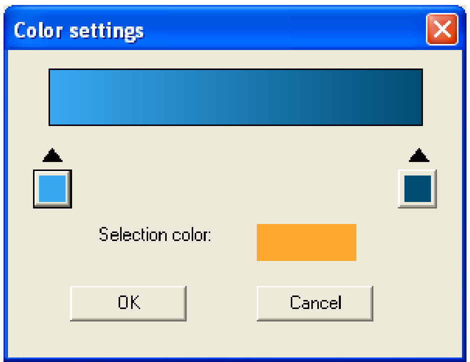
Demo rate
Quick
[✓ Normal]
Slow

OK
Cancel

Figure 15.13. The dialog window for choosing the demonstration rate

Press the button **OK** or the key **Enter** to confirm the choice of the demonstration rate. To return to the main menu of the system ParaLab without saving the changes, press the button **Cancel** or the key **Escape**.

3. **Changing the color palette.** To change the colors, which are used in the system ParaLab to visualize the problem solving process, the command **Colors** of the menu option **View** should be executed. In the dialog window



(Figure 15.14) the rectangle with the spectrum of colors changing smoothly from the light tint to the dark tint (*the color palette*) and the square with the selection color. In case of sorting algorithm the lighter color is used to display the minimum value elements of the sorted array, while the darker color displays the maximum value elements. If algorithms are performed on graphs, the lighter color shows the graph edge, which have the minimum weights, and the darker color is used to display the edge of the maximum weights. In case of matrix multiplication algorithms the selection color is used to display the matrix blocks located on the processors. In the graph algorithms the selection color is used to display the minimum spanning tree (the Prim algorithm) and the tree of the shortest paths (the Dejkstra algorithm).

Figure 15.14. The dialog window for changing of the color palette

To change these colors, click on the button located to the left or to the right of the color scale. As a result, the standard dialog window of choosing the colors will appear on the screen. Choose a color in the window and press **OK**. The color will change. In order to change the selection color click the square displaying this color by the left mouse button. Set the necessary color using the standard dialog window.

To change color palette press the button **OK** or the key **Enter** in the dialog window **Color palette**, In order to return to the main menu of the system ParaLab, press the button **Cancel** or the key **Escape**.

15.5.2. The Area "Current Array State"

This area is located in the right upper part of the window and displays the sequence of the sorted array elements if the experiment for the problem of data sorting is executed. As in case of the area "Experiment execution", each element of the sorted array is displayed by a vertical line. The height and the intensity of color represent the element value. The higher and darker the line is, the greater the element is.

All the parallel sorting algorithms distribute the initial array among the processors. The blocks placed one by one in the order of increasing the number of processors, on which they are located, form the resulting array. After the execution of sorting, the array blocks on each processor must be sorted and, besides, the elements located on the processor with a smaller number must not exceed the elements, located on the processor with a greater number.

Initially the array is a random set of elements. After being sorted, the array (if the initial data amount is big enough) is displayed as a right triangle with smooth changing of the color from light blue to dark blue.

15.5.3. The Area "Result of Matrix-Vector Multiplication"

This area is located in the right upper part of the window and displays the state of the result vector in the process of executing a parallel algorithm of matrix-vector multiplication.

In case of the algorithm based on the rowwise block-striped matrix partitioning, each processor computes a result vector block multiplying a stripe of matrix *A* by the vector *b*. The block computed on the active processor (blue color) is displayed by the dark blue color. After the execution of communications, the total result vector is located on each processor. Thus, all the vector blocks become the dark blue color.

In executing the algorithm based on the column block-striped matrix partitioning, each processor computes the vector of partial results multiplying a stripe of the matrix *A* by a block of vector argument *b*. All the blocks of the result vector in the area "Result of matrix-vector multiplication" are highlighted by the light blue color. After the execution of the communication step, each processor contains a block of the result vector. The block of the active processor is displayed in the area the in dark blue color.

If the algorithm based on the chessboard block partitioning is executed, vector *b* is distributed among the processors, which form the columns of the processor grid. After multiplying a block of matrix *A* by a block of vector *b*, the processor computes the block of the partial result vector. It is highlighted by the light blue color. After exchanging blocks within a processor grid row, each processor of the row holds a result vector block. The block of the active processor is displayed in the area "Result of matrix-vector multiplication" by the dark blue color.

15.5.4. The Area "Result of Matrix Multiplication"

This area is located in the right upper part of the window and displays the matrix state, which is the result of the process of executing the parallel algorithm of matrix multiplication.

The matrix *C* is partitioned into square blocks. Each processor of a multiprocessor system is responsible for computing one block of the result matrix *C* (the Fox method and the Cannon algorithm) or several blocks of the result matrix *C* (the block-striped algorithm).

In case of the block-striped algorithm, the dark blue color marks the blocks, which have already been computed.

In case of the Fox algorithm or the Cannon algorithm, all the blocks of the matrix *C* are computed simultaneously, and none of them can be computed until all the iterations of algorithm are performed. That is why the computational dynamics of the result matrix block, which is located on the active processor, is displayed in the area "Result of matrix multiplication" (this processor in the area "Experiment execution" is marked by the blue color). The summands, which have been computed so far, are shown in the dark blue color, and the summand, which is being computed at the current iteration, is marked by the selection color.

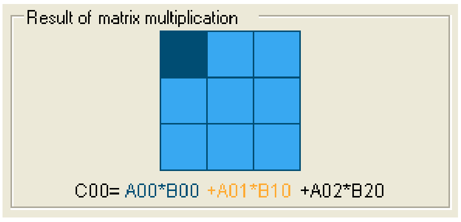

Result of matrix multiplication
C00= A00*B00 +A01*B10 +A02*B20

Figure 15.15. The area "Result of matrix multiplication" in case of the Fox algorithm

15.5.5. The Area "Current State of Linear System Matrix"

This area is located in the right upper part of the computational experiment window. It shows the current state of the linear equation system matrix in the course of the executing the Gauss algorithm. The dark blue color marks the non-zero elements, while the light blue color – the zero ones. After the execution of the Gaussian elimination only zero elements are located under the main diagonal. After the execution of the back substitution all non-zero elements are located on the main diagonal.

15.5.6. The Area "Result of Graph Processing"

This area is located in the upper right part of the computational experiment window and shows the current state of the graph. The graph edges are shown in different colors: the darker the color, the greater the edge weight is.

In the process of carrying out the graph algorithms the selection color marks the vertices and the edges, which have been so far included into the minimum spanning tree (the Prim algorithm) or into the tree of the shortest paths (the Dejkstra algorithm).

15.5.7. Choosing the Processor for Demonstration

For more detailed observation over the process of carrying out the experiment the system ParaLab provides the possibility to visualize the computations of a processor in a separate window.

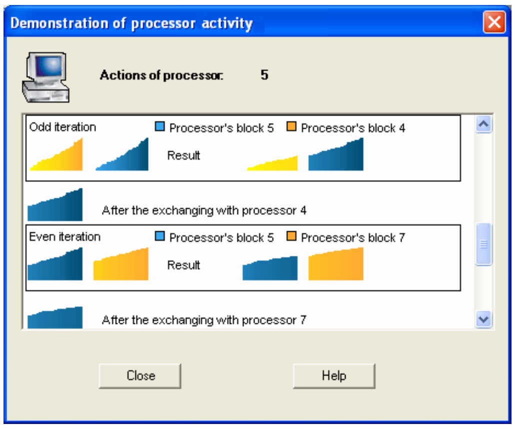


Figure 15.16. The window for demonstrating the processor activity

One of the methods of choosing a processor is to execute the command **Demo processor** of the menu option **View**. Select the number of processors using the track bar in the dialog window and press **OK**. In order to return to the main menu without choosing a processor press **Cancel**.

The second method to choose a processor is to select the processor by the mouse pointer in the active area (the form of the cursor will change and the prompt with the number of the chosen processor will be shown). After that you should click the left button.

The window "Demonstration of processor activity" will allow to observe the processor computations in detail.

15.6. Accumulating and Analyzing the Experiment Results

Carrying out numerical experiments for studying various parallel algorithms of solving complicated computational problems requires in many cases long-continued computations. To found the assumptions being made, it is necessary to carry out a wide range of experiments. These experiments may be carried out on different multiprocessor systems, by different methods, for different initial data. The system ParaLab has various means of accumulating and storing the results of the computational experiments and provides various methods of demonstrating the data in the form, which is suitable for analysis.

15.6.1. Accumulating the Experiment Results

To accumulate the results of the executed experiments, the system ParaLab provides a special memory, which is hereinafter referred to as *the experiment log*. The data stored in the experiment log includes:

- The date and the time of the experiment execution,
- The computer system parameters (the topology, the number of processors, the processor performance, the latency and the network bandwidth, the data transmission method),
- The problem statement (the problem name, the size of the initial data, the method of solving the problem),
- The experiment execution time.

The results are stored in the experiment log either by the user or by the system automatically (if the mode **Autosave** is set up). It should be noted that in case of the Autosave mode the repetition of the experiment with the identical initial data leads to the repeated storing of the results.

Accumulated results can be used for observing and analyzing. The stored data can be also taken to restore the previous state of the experiment – that allows to repeat the experiment again. Also it allows to continue computations from the suspended state.

Data of needless experiments can be deleted, it is possible to erase all accumulated results.

Saving the current experiment in a file, all the accumulated results will be also saved.

The data of the experiment log may be demonstrated in the various table or graph forms.

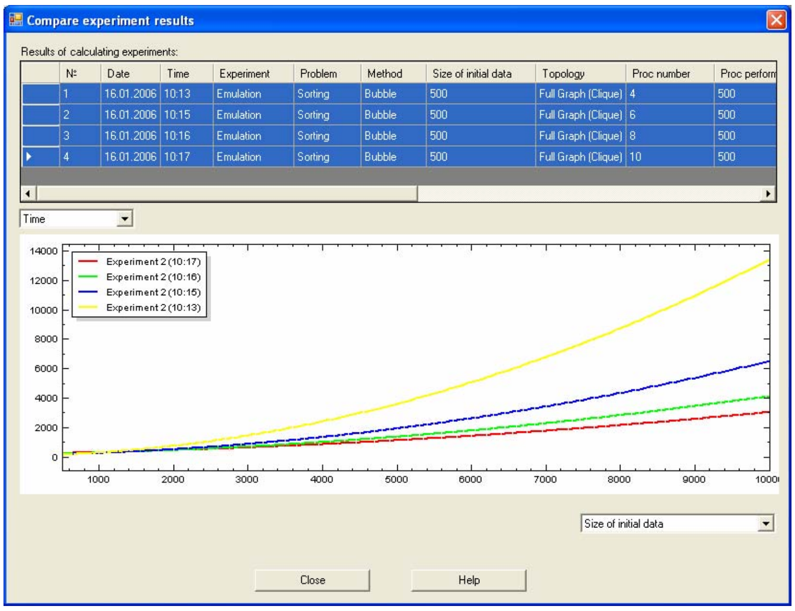
15.6.2. Observing the Experiment Results

To demonstrate the experiment log (the accumulated results of the experiments) in the system ParaLab there is the window **Experiment results**, which has the area **Result Table** (the upper part of the window) and the area **Result Graph** (the lower part of the window).

Each row of the result table (see Figure 15.17) corresponds to one executed experiment. The first row of the table is selected on default for visualizing the graph of the dependence of the experiment execution time with respect to the initial data amount the graph window area. In order to change the demonstrated dependence, it is necessary to select the corresponding items in the lists located in the upper left hand and lower right hand corners of the graph window area. It is possible to visualize the dependence of the experiment execution time and speedup with respect to the initial data amount, the number of processors, the processor performance and the network characteristics. It is possible to observe the graphs corresponding to different experiments, selecting different table rows.

It should be noted that the necessary analytical dependencies (see Sections 7-11) are used for demonstrating graphs of the experiments carried out in the simulation mode. For the experiments carried out on a computational cluster, a set of the results of real experiments, which have been obtained so far, is used.

Several dependencies can be shown in the graph window area, if several rows in the result table are selected.


Compare experiment results

Results of calculating experiments:

	№	Date	Time	Experiment	Problem	Method	Size of initial data	Topology	Proc number	Proc perform
	1	16.01.2006	10:13	Emulation	Sorting	Bubble	500	Full Graph (Clique)	4	500
	2	16.01.2006	10:15	Emulation	Sorting	Bubble	500	Full Graph (Clique)	6	500
	3	16.01.2006	10:16	Emulation	Sorting	Bubble	500	Full Graph (Clique)	8	500
▸	4	16.01.2006	10:17	Emulation	Sorting	Bubble	500	Full Graph (Clique)	10	500

Figure 15.17. The widow of the experiment results

🖶 *How to use the system ParaLab*

1. **Saving the experiment result.** To save the results of the computational experiment in the experiment log, execute the command **Save** of the menu option **Results**. It should be noted that the results of the same experiment cannot be saved twice. However, the experiments carried out with the identical initial data are considered to be different and their results may be saved separately.

2. **Demonstrating the experiment results.** To demonstrate the experiment results accumulated in the experiment log, execute the menu option **Results**, select the command **View**, and carry out one of the following two commands: **From active window** or **From all windows**. The execution of the first command leads to presenting the results accumulated in the active window of the computational experiment. The execution of the second command provides demonstrating the results from all the open experiment windows. The dialog window with the experiment result is shown in Figure 15.17. The window demonstrates the result table and the result graphs.

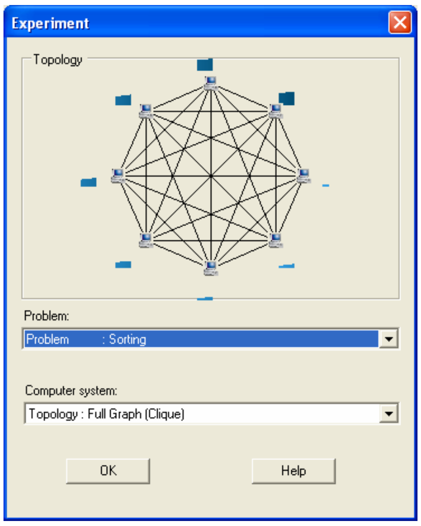
3. **Observing the experiment paramenetrs.** To observe the experiment parameters select the row of the result table by the double click and the window Experiment should be appeared (see Figure 15.18).

In order to change the type of the dependence, select the necessary values in the lists located upper to the left and below to the right of the graph window area. The lower right list allows to choose the argument of the dependence, and the upper left list makes possible to choose the dependence type.

4. **Changing the demonstrated dependence.** In order to change the type of the dependence, select the necessary values in the lists located upper to the left and below to the right of the graph window area. The lower right list allows to choose the argument of the dependence, and the upper left list makes possible to choose the dependence type.

5. **Setting the Autosave mode.** In order to automatically save the results of the experiments being carried out, the user should execute the command **Autosave** of the menu option **Results** (if the mode is fixed, the mark ✓ will be highlighted before the command **Autosave**). The experiment results are saved in the experiment log at the moment of the problem solving termination. To cancel the mode, it is necessary to repeat the command **Autosave**.

6. **Selecting a row in the result table.** Each row of the summary table corresponds to one executed experiment. In order to select a row, indicate at the necessary row with the mouse pointer and press the left button. It is also possible to move the cursor up and down (in this case the previous of the following row will be selected). If only one row is selected, then only the dependence corresponding to this row is shown in the graph window area.



Problem:
Problem : Sorting
Computer system:
Topology : Full Graph (Clique)

Figure 15.18. The window with the experiment parameters from the experiment log

7. **Selecting several rows in the result table.** In order to select several successive rows in the result table, press and hold **Shift** and select the first and the last rows of the desirable range by the mouse. To select several rows, which do not form a continuous sequence, press and hold **Ctrl** and select rows in the arbitrary order. In order to select several rows by means of the arrow keys press and hold **Shift** and move up and down in the table using these keys. If several rows are selected in the result table, then several dependencies will be shown in the graph window area.

8. **Deleting a result table row.** In order to delete the selected row, execute the command **Delete** of the context menu.

9. **Restoring the experiment using the data of the result table.** As it has been mentioned previously, the result table holds exhaustive information concerning the computational experiment being executed. To restrore the

experiment using the data of the result table select the desired row using one of the above described methods, and execute the command **Restore** of the context menu. The experiment will be restored in the active window.

🖶 *Tasks and Exercises*

Do the following exercises to master the rules of using the experiment log:

1. Carry out several experiments using the same matrix multiplication method, but changing the initial data amount and the number of processors. Using the window of the experiment results, analyze the obtained results. Create simultaneously several graphs in the graph window area and compare them.
2. Set up the mode **Autosave** and carry out several computational experiments studying the dependence of the execution time of the sorting problem with respect to the number of processors for various sorting methods.
3. Analyze the results of the experiments accumulated in the Log; select the experiment, which leads to obtaining the best (the worst) time characteristics, and analyze computer system topology for these experiments.

15.7. Carrying out the Computational Experiments

The system ParaLab provides different schemes of carrying out experiments to give convenient possibilities for studying and using parallel algorithms of solving complicated computational problems. Problems may be solved in the sequential execution mode, in the time sharing mode with the possibility to simultaneously observe the algorithm iterations in all the computational experiment windows. Carrying out series experiments, which requires long-continued computations, may take place in the automatic mode with the possibility of saving the results of solving problems in the experiment log. Experiments may be also carried out in the step-by-step mode.

15.7.1. Carrying out Sequential Experiment

The general purpose of carrying out computational experiments is to estimate the efficiency of parallel method in solving complicated computational problems depending on the multiprocessor computational system parameters and the initial data amount. Carrying out such experiments may be reduced to multiple repetitions of the problem statement and solution stages. The computation may be suspended at any moment of time (for instance, for changing the graphic forms of observation over the process of solving). The process may be continued after that and carried out till the result is obtained. The results of solving computational problems may be saved in the experiment log and demonstrated in the form suitable for analysis.

🖶 *How to use the system ParaLab*

1. **Carrying out the experiment.** To carry out a computational experiment, choose the menu option **Run** and execute the command **In active window**. The computation will start and will go on continuously till the results are obtained. In the course of the experiment execution the main menu is substituted for the menu with the command **Stop**; after the termination of solving problem the main menu is restored.
2. **Stopping the experiment.** To stop the process of carrying out the experiment the user should execute the command **Stop** in the menu line (the command is possible only before the moment of the process termination).
3. **Continuing the experiment.** To continue the previously suspended experiment the command **Continue** of the menu option **Run** should be executed (this command may be executed only in case if the problem statement and the computer system parameters have note been changed after the process was suspended; if the experiment cannot be continued, the command **Continue** is not available and is marked by the grey color).

🖶 *Tasks and Exercises*

1. Set the topology **Ring** and the number of processor equal to 10 in the active window of the computational experiment. Make the problem of sorting with the use of the bubble sorting algorithm as the current problem of the window.
2. Execute the two algorithm iterations and suspend the experiment execution.
3. Change the demonstration rate and the mode for visualizing data communications.
4. Continue the execution of the experiment.

15.7.2. Carrying out Step-by-step Experiment

For more detailed analysis of the parallel algorithm iterations the system ParaLab provides the possibility of carrying out parallel algorithm in the step-by-step mode. The parallel algorithm is suspended after executing each iteration in this mode. It gives the user the opportunity to study the results of the iterations in detail.

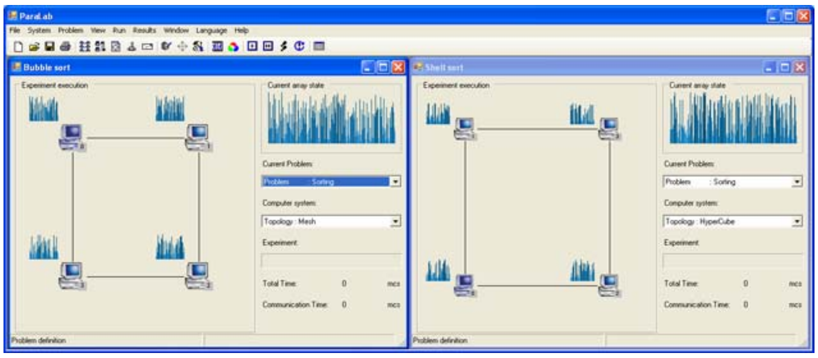
🖶 *How to use the system ParaLab*

1. **Executing the experiment in the Step-by-step mode mode.** To set this mode the command **Step-by-step mode** of the menu option **Run** should be executed. After the execution of that command, the main menu is substituted by the menu of the step-by-step mode with the commands:

- The command **Step** is used in order to carry out the next iteration of the parallel method;
- The command **Run** is used in order to continue carrying out the experiment without stops after iterations;
- The command **Close** is executed when it is necessary to suspend carrying out the experiment and return for executing the main menu commands.

15.7.3. Carrying out Several Experiments Simultaneously

Carrying out several experiments in sequence hinders observing and analyzing the results of parallel algorithms in the dynamics. In order to compare data in detail the system ParaLab provides the possibility to display simultaneously the results of all the compared experiments. In this case the display may be divided into several experiment windows. In each window the results of a separately executed experiment can be observed. The user can open a new window for carrying out a new experiment. The experiment results and the experiment log are formed separately for each available window. The experiment windows when they are visualized may share the screen (in this case the contents of all windows are visualized) or the windows may overlap. The user can make any window active for carrying out the next experiment. However, the computations may be performed in all windows simultaneously in the time sharing mode. In this case each new iteration is performed sequentially in all available windows. Using this mode it is possible to observe the dynamics of carrying out several experiments. The computation results may be visually differed, and their comparison may be done on a convenient visual basis.



Current Problem: Problem : Sorting
Computer system: Topology : Mesh
Experiment:
Total Time: 0 mcs
Communication Time: 0 mcs

Current Problem: Problem : Sorting
Computer system: Topology : HyperCube
Experiment:
Total Time: 0 mcs
Communication Time: 0 mcs

Figure 15.19. The example of demonstrating several experiment windows

It should be noted that the results of experiments carried out in different windows may be demonstrated jointly in the same result table (see Subsection 15.6).

How to use the system ParaLab

1. **Creating a new window.** In order to create a window for carrying out an experiment, the user should execute the command **New** of the menu option **File**. The experiment window may be closed. To close all the windows simultaneously, the command **Close all** of the menu option **Experiment** can be executed.
2. **Carrying out the experiments in all windows.** To carry out computational experiments in all available windows in the time sharing mode (i.e. in passing over to the execution of the next iteration only after the termination of the current one in all available windows) the command **In all windows** of the menu option **Run** should be executed. The control of the computational process is performed in the same way as it is done for a single window (the experiment execution is suspended after the command **Stop**, the computations are continued after the command **Continue** of the menu option **Run**).
3. **Resizing the experiment windows.** For simultaneous demonstrating all the windows without overlapping the command **Tile** of the menu option **Window** should be executed. To select the greater part of the screen for the active window (but preserving the possibility of quick access to all available windows) the command **Cascade** of the menu option **Window** can be applied.
4. **Analyzing the results of all the experiments.** To make a table of all the experiment results from all the experiment windows, execute the sequence of commands **Results → View→ From all windows**.

Tasks and Exercises

1. Open a new experiment window, set up the mode of demonstrating windows without overlapping (the command **Tile**).
2. Select the bubble sort method in the first window. Set the topology **Hypercube** and select the Shell short method in the second window. Set the same topology **Hypercube**.
3. In both windows set the mode of automatic data saving in the experiment log (the mode **Autosave**).

4. Carry out computational experiments simultaneously in both windows. Regulate the visualization by setting the appropriate demonstration rate.

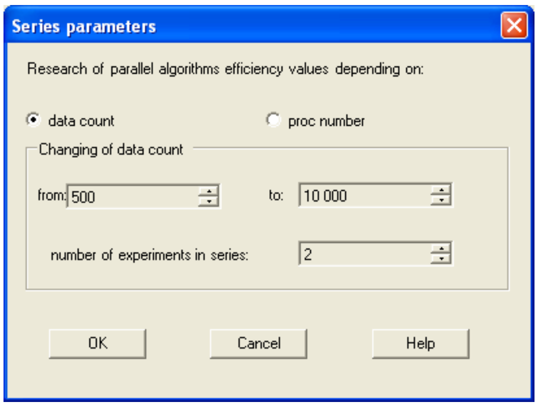
5. Form the result table for both experiments. Compare the algorithm time characteristics of the bubble sort to those of the Shell sorting.

15.7.4. Carrying out a Series of Experiments

ParaLab provides the opportunity to perform automatically a series of experiments, which require long-continued computations. If the user wants to set this mode, it is necessary to choose the window, in which the experiments will be carried out, to set the number of experiments and select the parameter that will change from experiment to experiment (the initial data amount or the number of processors). The results of the experiments may be saved in the experiment log and analyzed afterwards.

How to use the system ParaLab

1. **Carrying out a series of experiments.** Switch to the mode of executing a series of experiments is performed by means of the command **Series** of the menu option **Run**. In the dialog window Series parameters (Figure 15.20) the number of experiments in the series and the type of the series should be set. It should be specified whether it has to investigated the dependence of time and speedup of solving the given problem with respect to the initial data amount or with respect to the number of the processors used. Before setting the mode, it might be useful to set the automatic saving of the experiment results in the experiment log (the command **Autosave** of the menu option **Experiment**).


Series parameters

Research of parallel algorithms efficiency values depending on:

[✓ data count] proc number

Changing of data count

from: 500 to: 10 000

number of experiments in series: 2

Figure 15.20. The dialog window of setting the series parameters

In executing a series of experiments the main menu of the system ParaLab is substituted by the menu of the given mode of computations. The commands of this menu allow to perform the following commands:

- The command **Start** allows the user to carry out a секшуы of experiments;
- The command **Close** is used to suspend the mode and return to the commands of the main menu;
- The command **Help** is used to obtain additional reference information.

The experiment may be suspended during solving a series of problems (after the execution of the command **Start**) by means of the command **Stop** at any moment of time.

15.8. Storing the Experiment

The experiment carried out in the active experiment window may be stored in a file be at any moment of time. The data stored for the experiment window, includes:

- The parameters of the computer system (the topology, the number of processors, the processor performance, the network latency and bandwidth, the method of data communication),
- The problem statement (the problem type, the initial data size, the method of solving the problem),
- The experiment log.

The data stored in the file may be loaded from the archive. Thus, the user may load the experiment from the file and continue carrying out the experiments.

How to use the system ParaLab

1. **Saving the experiment.** To save the experiment, the command **Save** of the menu option **File** should be executed. Then the user should type the file name in the dialog window **Save File As**. The extension of the file name may not be specified (the file name extension of the ParaLAb system is **.prl** on default).

2. **Loading the experiment.** To load the experiment, which have been previously saved in the file, selet the menu option **Load** and execute the command **Open**. The experiment stored in the selected file will be loaded after the execution of the command.

Tasks and Exercises

Carry out computational experiments according to the following scheme:

1. Carry out an experiment and save it in the file.
2. Close the ParaLab system.
3. Start the system again and load the stored experiment from the file.

References

Gergel, V.P., Strongin, R.G. (2001, 2003 - 2 edn.). Introduction to Parallel Computations. - N.Novgorod: University of Nizhni Novgorod (In Russian)

Knuth, D. E. (1997). The Art of Computer Programming. Volume 3: Sorting and Searching, second edition. - Reading, MA: Addison-Wesley.

Kumar V., Grama, A., Gupta, A., Karypis, G. (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)