RECOMMENDATIONS FOR DOING OF LABORATORY WORKS

The order of performance of laboratory works

1. To study the theoretical material on the basis of materials of lectures and recommended literature sources.
2. Develop a program according to the task for laboratory work. The variant of the task is computed by the equation v = k mod n + 1, where k is the number of the student in the journal, n is the number of options. In exceptional cases, is allowed to change the variant with the permission of the teacher.
3. Conduct experiments on a computing cluster, present the results in a visual form (graphs, diagrams, tables, etc.). Verify the correctness of the results obtained (see Annex 2).
4. Analyze the results of experiments, formulate conclusions.
5. Answer the control questions given in the assignment for laboratory work.
6. Write an electronically report.
7. Defend laboratory work.

Requirements for the programs

1. Programs are developed for the GNU/Linux operating system in the programming language specified in the task.
2. Programs must be compiled without errors and warnings. For the GCC compiler (C/C ++), it is recommended to use the -O2 and -Wall options.
3. The design of the source code must correspond to the agreements accepted in the framework of the course or similar: K & R style, GNU coding standards, Google C ++ Style Guide, Intel Coding Guidelines, etc.
4. The source code of the program must be developed independently. Solutions completely borrowed from the Internet, and the twin work are not allowed for defense. In the case of partial borrowing of the source code, this should be indicated by comments in the program and reflected in the report. It is desirable to cite references to the source of borrowing.

Carrying out experiments

1. The experiments are performed on the cluster computer system (VS). To obtain the account on the cluster computer system, you need to contact the teacher.
2. The execution of programs on the computing cluster is carried out in strict accordance with the manual of the user of the cluster system (an example of the job file for the task to be run on the computation cluster is given in Appendix 3).
3. During the experiments, it is recommended to create a script (for example, on the bash language) to run programs with different parameters (an example of such script is given in Appendix 4). Methods for measuring the execution time of programs are presented in Appendix 6.
4. After the experiments, it is necessary to perform statistical processing of the results obtained (see Appendix 2).
5. The results of the experiments should be reflected in the form of graphs of the investigated indicators. When constructing graphs, it is recommended to use gnuplot or another freely distributed program (see Appendix 5).
6. It is necessary to analyze the results of experiments and draw conclusions. Conclusions should be reflected in the report.
7. Criteria for evaluating laboratory work

Criteria for evaluating laboratory work

1. The performance of the work is assessed on a five-point scale.
2. Laboratory work should be protected at a fixed time.
3. When evaluation of the work, the completeness and originality of the solution of the problem, the quality of the presentation of the material in the report, the correctness of the conducted experiments and the reasoning of the conclusions, the answers to control questions are taken into account.

Procedure for the protection of laboratory work

1. Demonstrate the workability of the program on a computing cluster. Explain the method of solving the problem, explain the purpose of the main components of the program. Answer the questions about the source code of the program.
2. Demonstrate the results of the experiments and the report. Explain the behavior of curves on graphs (increasing, decreasing, extremes) and values in tables.
3. Answer the control questions given in the lab work assignment, and other questions on the topic of the work.

4. If necessary, modify the program or report and go on re-protection.

Reporting requirements

The report is prepared in electronical form. The main text of the report is made in the same font, for example, Times New Roman, 12-14 pt. Line spacing is single or one-and-a-half. The indent of the first line of each paragraph is 1-2 cm. The contents of the main paragraphs of the text are aligned in width, the headings are centered or left. The source code of the programs and output to the console are made in monospaced font, for example, Consolas, Source Code Pro, Courier New.

Figures, graphs, tables must be aligned to the center of the page and have signatures.

The format of the report is PDF, ODT, RTF. For the preparation of the report it is recommended to use packages Google Docs, LibreOffice, Latex. It is desirable to make out the graphs using gnuplot, Google Docs, LibreOffice, Asymptote, MetaPost, R.

The report should contain the following parts.

1. Title page.
2. The title page contains the title of the course, the number and subject of laboratory work, the full name of the student and the full name of the teacher. An example of the title page is given in Appendix 1.
3. Task for laboratory work.
4. Description of the solution of the problem.
5. Description of the method of solving the task for laboratory work, a description of the developed algorithm (or justification of the choice of the algorithm) and, if necessary, data structures.
6. Organization of experiments.
7. The description of test tasks is described, the investigated parameters, input and output data are described:

- computing system: the number of computational nodes (the total number and number used in experiments), the number of processor cores in each node, the processor model, the amount of RAM;
- system software: version of the operating system, compiler, compilation keys, libraries used;
- description of input and output data of programs, parameters of algorithms.

1. The degree of detail of the description of experiments should be such that an outsider can repeat experiments.
2. Results of experiments.
3. The results obtained in the laboratory work are graphs, diagrams, tables. All figures must have correct format (an example of the design of the graph is given in Appendix 5). Give the results of statistical processing of measurements (see Appendix 2).

4. Analysis of the results of experiments and conclusions.
5. Conclusions should be specific. They should explain the results of the experiments and the main points of the program development.
6. List of used literature sources.

MULTITHREADED PROGRAMMING

Laboratory work № 1. Development of simple multithreaded programs

1. Develop a program that implements a parallel algorithm. Development tools: POSIX Threads, C++-threads.
2. Analyze the effectiveness of the parallel program: plot the acceleration and efficiency coefficient against the number of parallel threads and estimate the acceleration according to the Amdahl law (for non-recursive algorithmic), to estimate the scalability.

Variants of assignments

1. Generation of the Mandelbrot set.

   The input of the program are the numbers a, b, which determine the size of the rectangular area for finding points that belong to the Mandelbrot set. As a result of the program, it is necessary to obtain a subset of n points belonging to the Mandelbrot set. To do this, the rectangular area is evenly divided among the threads, and each thread searches for the Mandelbrot set in its area. To verify the belonging of a point to the Mandelbrot set, it is recommended to use the Escape Time algorithm. It is necessary to take into account the imbalance problem of processor cores and make sure that the execution time of the parallel program is correct.

2. The algorithm of quick sort.

   The input data for the program is an unsorted numeric array which is filled with random values. The result of the program execution is a sorted array a. It is necessary to implement a recursive algorithm for quick sorting. Every time you call the quick sort function, you must generate a thread. Generation of threads must be stopped when the number of threads reaches the specified number, which varies from 1 to the number of processor cores on the compute node. As the pivot element, select the first element of the sequence.

3. Algorithm of Shell sorting.

   The input of the algorithm is an unsorted array a of length n, filled with random numerical values. The result of the program is the sorted array a. The lengths of d_i intervals should be chosen as follows: $d_1 = n / 2$, $d_2 = n / 4$, $d_i = d_{i-1} / 2$, $d_k = 1$. During the algorithm, the sorting of each subarray (for the current length d of the gap) in a separate thread. The number of threads should not exceed the number of processor cores in the system.

4. Algorithm of binary search.

It is necessary to implement two versions of the search algorithm: for an unsorted array and for a sorted array. The keys can be found several times in the array. The input is a random numeric array a (sorted or unsorted), in which the search is performed, and a set of search numbers represented by the array b. Output: if the element of array b occurs in array a, then you need to print the index of this element in array a. In the case of a non-sorted array, the elements of array a are evenly distributed among the threads, each thread performs a search for elements in its subarray. In the case of a sorted array, it is advisable to distribute the search numbers represented by the array b between the threads. The number of threads should not exceed the number of process cores.

5. Solution of the system of linear algebraic equations by the Gauss method.

   The program input receives a randomly generated system of linear equations $Ax = b$ which contains n equations. The type of the matrix is arbitrary. Output data: vector x of the solutions of the system of linear equations. When parallelizing, the elements of the matrix are uniformly distributed among the threads by the rows. It is necessary to implement a parallel execution of both the direct and reverse motion of the Gauss method.

6. Prim's algorithm for constructing a minimal spanning tree in a graph.

   Input: random connected undirected graph $G = (V, E)$. For each edge of the graph, its cost is given. Output: a set of vertices forming a minimal spanning tree. At each iteration of the Prim's algorithm, the finest search with the least cost is performed in parallel. To do this, the entire set of edges, incident to the vertices of the current spanning tree, is evenly distributed among the streams. The number of threads should not exceed the number of processor cores.

7. Floyd's algorithm for finding all minimal paths in a graph.

   Input: random non-oriented graph $G = (V, E)$. Each edge of the graph has specified cost. Output: all the minimum paths in the graph. At each iteration of the algorithm, the set of all vertices k, through which the shortest path can pass, is evenly distributed between the threads. The number of threads should not exceed the number of processor cores.

8. The simplest algorithm for finding substrings in a string.

   Input: string s, line d. Output: the character numbers from which the string d begins, found in the string s. Elements of the string s are evenly distributed among the threads, each thread performs a search in its substring. The number of threads should not exceed the number of processor cores.

Control questions

1. Characterize the features of the architecture of modern multiprocessor computer systems. Give definitions of the terms Hyper-threading, SMP, NUMA, ccNUMA.
2. Explain the difference between instruction-level parallelism and thread-level parallelism.
3. What are the differences between threads and processes? Which resources for threads are common, which are unique for each thread?

4. Explain the fork-join model of the life cycle of threads. Illustrate the creation, completion, connection, disconnection and forced cancellation of flows using the POSIX Threads standard as an example.
5. What are the features of passing arguments to the flow? Can I pass a local variable as an argument?
6. Explain the events that occur when planning on-streams. Explain the flow state diagram. What are the flow planning policies? What are the features of real-time streams?

References

For assignments 1:

- **[McCool, 2012]** – *Chapter 4.3. Mandelbrot*.
- https://en.wikipedia.org/wiki/Model_Mandelbrot
- http://elementy.ru/posters/fractals/Mandelbrot
- https://en.wikipedia.org/wiki/Mandelbrot_set#Escape_time_algorithm

For assignments 2:

- **[Breshears, 2009]** – *Chapter 8. Sorting – quick sort*.
- Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. – 3rd ed. – MIT Press, 2009. – 1312 p. – *Chapter 7. Quicksort*.

For assignments 3:

- **[Breshears, 2009]** – *Chapter 8. Sorting – shellsort*.
- Knut D.E. The art of computer programming, volume 3: sorting and searching. – Addison-Wesley, 1998. – 782 p. – *Chapter 5.2.1. Sorting by insertion*.

For assignments 4:

- **[Breshears, 2009]** – *Chapter 9. Searching – binary search*.
- Knut D.E. The art of computer programming, volume 3: sorting and searching. – Addison-Wesley, 1998. – 782 p. – *Chapter 6.2.1. Searching in ordered table*.

For assignments 5:

- Bakhvalov N.S. Numerical methods: analysis, algebra, ordinary differential equations. – Moscow: MIR, 1977. – 663 p. – *Chapter 6. Numerical methods of algebra*.

For assignments 6:

- **[Breshears, 2009]** – *Chapter 10. Graph algorithms – minimum spanning tree*.
- Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. – 3rd ed. – MIT Press, 2009. – 1312 p. – *Chapter 23. Minimum spanning trees*.

For assignments 7:

- **[Breshears, 2009]** – *Chapter 10. Graph algorithms – all-pairs shortest path*.

- Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. – 3rd ed. – MIT Press, 2009. – 1312 p. – *Chapter 25. All-Pairs Shortest Paths*.

For assignments 8:

- Galil Z. Optimal parallel algorithms for string matching // Information and Control. – 1985. – V. 67. – No. 1. – P. 144–157.
- Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms. – 3rd ed. – MIT Press, 2009. – 1312 p. – *Chapter 32. String Matching*.

APPENDIX

1. Example of title page

MINISTRY FOR EDUCATION AND SCIENCE OF RUSSIA

_____

SAINT PETERSBURG ELECTROTECHNICAL UNIVERSITY «LETI»

_____

Laboratory work № 7
Lock-free programming

Student
group 9999
Ivanov I.I.

Teacher
PhD, associate prof.
of CE department
Paznikov A.A.

Saint Petersburg
ETU «LETI»
2017

2. Formulas for computations and processing of measurement results

Sample mean

Let there be a random variable x with a mathematical expectation *M* and variance *D*. Over the value of *x*, *n* independent experiments were performed, resulting in a set of numerical results x_1, x_2, ..., x_n. As an unbiased estimate of the mathematical expectation of x, one can use the sample mean, which is defined as the arithmetic average of the observed values:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{1}{n}(x_1 + x_2 + \ldots + x_n)$$

Sample variance

Variance is a measure of the spread of a random variable, i.e. Its deviation from the mathematical expectation. The unbiased estimate of variance is computed by the formula:

$$s^2 = \frac{1}{n-1}\left(\sum_{i=1}^{n}(x_i - \bar{x})^2\right)$$

Standard deviation

The standard deviation is used to estimate the scattering of values of a random variable with respect to its mathematical expectation. The standard deviation is measured in units of the most random variable and is used in calculating the standard error of the arithmetic mean, in constructing confidence intervals, etc. The value of the standard deviation is estimated based on an unbiased estimate of the variance of the random variable:

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Standard error of the mean

This value characterizes the accuracy with which the average value is obtained. Calculated by the following formula:

$$s_{\bar{x}} = \frac{s}{\sqrt{n}} = \sqrt{\left(\sum_{i=1}^{n}(x_i - \bar{x})^2\right)/(n(n-1))}$$

Wellford's method for calculating the arithmetic mean and standard deviation

If the values x_1, x_2, ..., x_n are floating-point values, then the associativity law is not satisfied, and the order of the operations in the standard deviation formulas (mathematical expectation) matters. Then for the

calculation of these indicators it is better to use the following recurrence formulas [Knut D.E. The art of computer programming, volume 2: Seminumerical Algorithms. – Addison-Wesley, 1998. – 782 p.]:

$$M_1 = x_1, \quad M_k = M_{k-1} + (x_k - M_{k-1})/k,$$
$$S_1 = 0, \quad S_k = S_{k-1} + (x_k - M_{k-1})(x_k - M_k),$$
$$k = 2, 3, \ldots, n,$$
$$\bar{x} \approx M_k,$$
$$s \approx \sqrt{S_n/(n-1)}$$

Coefficient of variation

The coefficient of variation of a random variable is a measure of the relative spread of a random variable; Shows how much the average value of this value is its average spread. In contrast to the standard deviation, it measures not the absolute, but the relative measure of the spread of values.

$$V = \frac{s}{\bar{x}}$$

Confidence interval

When conducting experiments for each measured value of x, it is necessary to construct a confidence interval. A confidence interval is an interval $[\bar{x} - \Delta x, \bar{x} + \Delta x]$, with a given degree of certainty, contains the true value of x. Trust probability (probability) is a probability with which the confidence interval includes the true value of the measured quantity.

To calculate the absolute error, a coefficient is introduced that depends on the confidence (probability) p and the number of measurements n, called the Student's coefficient t. Student's ratio is determined by the table (Table).

The absolute error Δx is calculated by the formula

$$\Delta x = s_{\bar{x}} \cdot t$$

The confidence interval is written as

$$x = \bar{x} \pm \Delta x$$

Table. Values of the Student's ratio t

n	p			
	0,80	0,95	0,99	0,999
2	3,078	12,706	63,657	636,61
4	1,638	3,182	5,841	12,941
6	1,476	2,571	4,032	6,859
8	1,415	2,365	3,499	5,405
10	1,383	2,262	3,250	4,781
20	1,328	2,093	2,861	3,883
30	1,311	2,045	2,756	3,659
40	1,303	2,021	2,704	3,551
60	1,296	2,000	2,660	3,460
120	1,289	1,980	2,617	3,373

3. Example of a task passport for running on a cluster

Listing 1 shows the task's passport for running the program on one compute node using 8 processor cores under the control of the TORQUE batch task manager.

Listing 1. Example of a task passport for running on a computing cluster, the resource management system TORQUE

```
#PBS -N program_name
#PBS -l nodes=1:ppn=8
#PBS -j oe

cd $PBS_O_WORKDIR

./prog
```

#PBS -N program_name

– the name of the task;

#PBS -l nodes = 1: ppn = 8

– allocated subsystem for program execution (one node, eight processor cores);

./prog

– the name of the executable.

Listing 2 gives an example of a task passport for running the program on one compute node under the control of the batch task processing manager SLURM.

Listing. 2. Example of a task passport for running on a computing cluster, a resource management system SLURM

```
#!/bin/bash
#SBATCH --nodes=1 --ntasks-per-node=8
./prog
```

#SBATCH --nodes = 1 --ntasks-per-node = 8

– allocated subsystem for program start (one node, eight processor cores);

./prog

– the name of the executable file.

4. Example of bash-script for experiments

Listing 3 shows a script that allows you to run the prog program to calculate the value of a function (mean time to failure) with different parameters *n* and *m*. For each parameter *n*, a separate file with the results of the experiments is formed (the dependence of the value of the function from *m*).

Listing 3. Example bash-script for experiments

```
N="10 20 30 40 50"
M="1 2 3 4 5"
NRUNS=3
function measure_time {
   time=`$@ | egrep "Result time: " \
         | awk '{print $3}'`
   echo $time >>results.tmp
}
function measure {
   n=$1
   m=$2
   datfile=$3
   cat /dev/null >results.tmp
   for ((i = 1; i <= NRUNS; i++)); do
      cmd="./prog $n $m"
      echo $cmd
      measure_time $cmd
   done
   avg=`awk '{ total += $1; count++ } END \
          { print total / count }' results.tmp`
   echo -e "$m \t $avg" >>$datfile
   rm results.tmp
}
function time_to_failure {
   for n in $N; do
      datfile="n${n}_time.dat"
      echo -e "m \t time, s" >$datfile
      for m in $M; do
         measure $n $m $datfile
```

```
      done
   done
}
time_to_failure
```

```
N="10 20 30 40 50"
M="1 2 3 4 5"
```

– sets of parameters *n* and *m*;

```
NRUNS=3
```

– the number of repeated runs of the program for each set of parameters to obtain the mean value;

```
function measure_time
```

– he time measurement function executes programs, searches for output data after the line "Result time:" and output to a file results.tmp;

```
function measure
```

– conducting an experiment with the specified parameters *n* and *m* with the output of the result in a file;

```
n=$1
m=$2
datfile=$3
```

– store in variable function arguments measure;

```
cat /dev/null >results.tmp
for ((i = 1; i <= NRUNS; i++)); do
   cmd="./prog $n $m"
   echo $cmd
   measure_time $cmd
done
```

– execution of $NRUNS runs with the same parameters, the results of the runs are written to the results.tmp;

```
avg=`awk '{ total += $1; count++ } END \
        { print total / count }' results.tmp`
echo -e "$m \t $avg" >>$datfile
```

```
rm results.tmp
```

– computation of the average value based on the results of the starts and saving it to the file $datfile, deleting the time file results.tmp;

```
function time_to_failure
```

– function of carrying out experiments for different input parameters;

```
for n in $N; do
  datfile="n${n}_time.dat"
  echo -e "m \t time, s" >$datfile
  for m in $M; do
     measure $n $m $datfile
  done
done
```

– running experiments for each pair of values of *n* and *m* from the sets $N and $M; the results for each value of n are written to a separate file $datfile;

```
time_to_failure
```

– performing the function of conducting experiments.

5. Example gnuplot-script for plotting

The script, shown in Listing 4, allows using the gnuplot program to build three acceleration curves on one coordinate plane, the data for which lie in three different files (Figure 1).

Listing 4. An example of a gnuplot script for plotting

```
set term pngcairo transparent enhanced \
   font "Times,34" size 1200,800
set xlabel "{/Times-Italic p}, number of threads"
set ylabel "{/Times-Italic s}, speedup"
set output "speedup.png"
set key inside bottom nobox
set border lw 3
set grid lw 3
plot "good.dat" using 1:2
   ti "{/Times-Italic n} = 10000" \
   with lp dt 1 lw 5 pt 4 ps 3 lc rgb '#C40D28', \
   "ugly.dat" using 1:2  \
   ti "{/Times-Italic n} = 20000" \
   with lp dt "_" lw 5 pt 7 ps 3 lc rgb '#003399', \
   "bad.dat" using 1:2 \
   ti "{/Times-Italic n} = 30000" \
   with lp dt "_.." lw 5 pt 5 ps 3 lc rgb '#007B00'
```

```
set term pngcairo transparent enhanced \
   font "Times,34" size 1200,800
```

– type of output file (pngcairo), background (transparent), support for special characters (enhanced), head-set and font size (Times, 34), output file sizes (1200, 800);

```
set xlabel "p, number of threads"
set ylabel "speedup"
```

– axis titles;

```
set output "speedup.png"
```

– title of output file;

```
set key inside bottom nobox
```

– alignment of the key of the graph: inside the graph, from below, without the frame;

```
set border lw 3
set grid lw 3
```

– thickness of the axis and grid lines;

```
plot "good.dat" using 1:2  ti "n = 10000" \
    with lp dt 1 lw 5 pt 4 ps 3 lc rgb '#C40D28', \
    "ugly.dat" using 1:2  ti "n = 20000" \
    with lp dt "_" lw 5 pt 7 ps 3 lc rgb '#003399', \
    "bad.dat" using 1:2  ti "n = 30000" \
    with lp dt "_.." lw 5 pt 5 ps 3 lc rgb '#007B00'
```

– command for plotting on one coordinate plane three curves from data from files good.dat, ugly.dat, bad.dat, using for the construction 1 and 2 columns (using 1:2) with the parameters specified after the attributes: ti – title of the graph, with lp – type of curve mapping (markers connected by broken lines), dt – line type, lw – line thickness, pt – market type, ps – market size, lc – color of the curve.

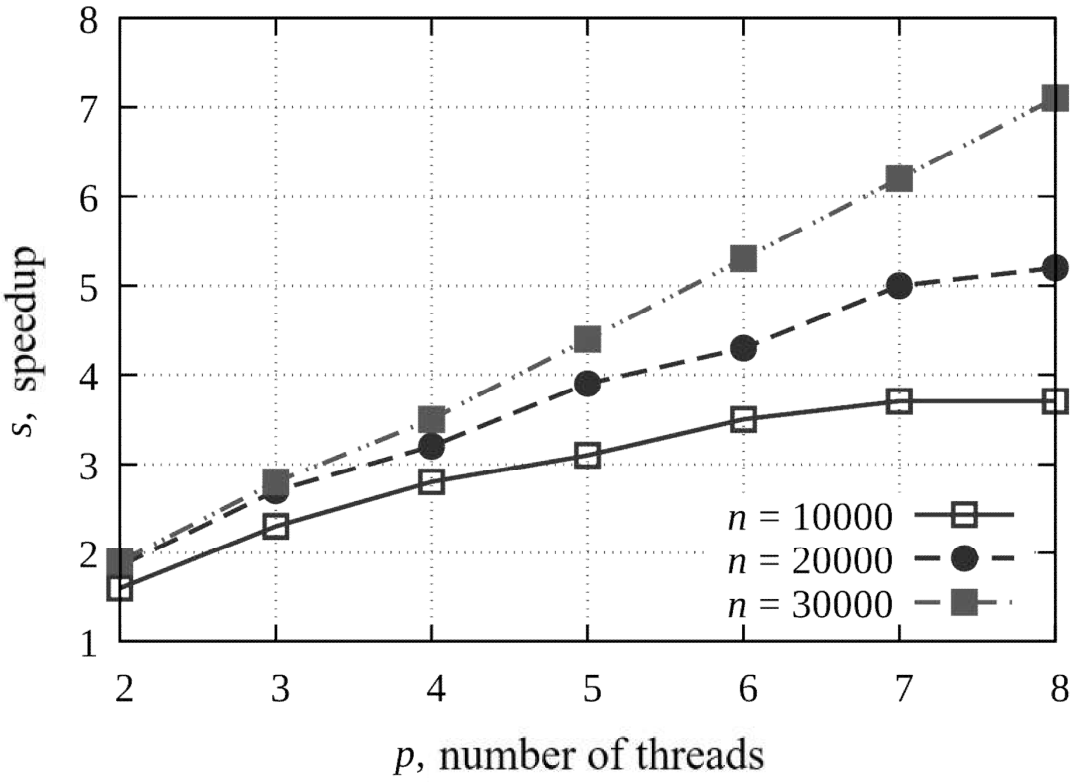

Fig. 1. An example of a graph constructed with gnuplot

Rules for graphs

An example of the graph is shown in Fig. 1. Requirements for drawing up schedules:

1. The font size of all the inscriptions, the thickness of the lines, the size of the markers on the curves, the resolution of the image must be sufficient for good readability of the graph.
2. The axes of the coordinates must be signed, in the middle or at the end of the axes, in the presence of units of measurement, they must be indicated.
3. Curves should be constructed without using anti-aliasing; The measuring points must be marked with markers. Curves should be different in color and shape of markers.
4. All variables are typed in italics, all other labels are in a direct font.
5. The legend of the graph is located on the coordinate plane, under the figure or in the caption of the figure.
6. Each graph must have a caption. In the caption, specify the figure number and the title of the graph.

6. Methods of measuring time

Measuring program execution time

The execution time of the program can be measured with the help of the utility time:

```
$ \time –f %e –o file ./prog
```

Here -f% e – the format of the output of time (in seconds), -o file – the indication to output the measurement result to a file.

Measuring inside the program (C language)

Listing 5. An example of a time measurement in a C program

```
struct timeval start, end;
double elapsed_time;

/* The beginning of measurements */
gettimeofday(&start, NULL);

/* Program code */

/* The end of measurements */
gettimeofday(&end, NULL);
elapsed_time = (end.tv_sec – start.tv_sec) +
  (double)(end.tv_usec – start.tv_usec) / 1000000;
printf("Elapsed time: %f s\n", elapsed_time);
```

Measurement inside the program (C ++ language)

Listing 6. Example of measuring time in a program (C ++)

```
auto get_time = std::chrono::steady_clock::now;
decltype(get_time()) start, end;

// The beginning of measurements
start = get_time();

// Program code

// The end of measurements.
end = get_time();
```

```
auto elapsed = std::chrono::duration_cast
  <std::chrono::milliseconds> (end – start).cout();
std::cout << "Elapsed time: "
        << double(elapsed) / 1000 << " s\n"
```

Measuring time intervals within a multithreaded program (POSIX Threads)

Listing 7. An example of measuring time in a multithreaded program using binary synchronization (POSIX Threads)

```
pthread_barrier_t barrier;
struct timeval start, end;
static pthread_once_t
  timer_start_once = PTHREAD_ONCE_INIT,
  timer_end_once = PTHREAD_ONCE_INIT;
void timer_start(void) {
  gettimeofday(&start, NULL);
}
void timer_end(void) {
  gettimeofday(&end, NULL);
}
void *thread(void *arg) {
  pthread_barrier_wait(&barrier);
  pthread_once(&timer_start_once, timer_start);
  /* Program code */
  pthread_once(&timer_end_once, timer_end);
  return NULL;
}
int main(int argc, const char *argv[]) {
  pthread_barrier_init(&barrier, NULL, NTHREADS);
  pthread_t tid[NTHREADS];
  for (int ithr = 0; ithr < NTHREADS; ithr++)
```

```
    pthread_create(&tid[ithr], NULL,
                &thread, NULL);
  for (int ithr = 0; ithr < NTHREADS; ithr++)
    pthread_join(tid[ithr], NULL);
  pthread_barrier_destroy(&barrier);
  double elapsed_time = (end.tv_sec - start.tv_sec)
 + (double) (end.tv_usec - start.tv_usec) / 1000000;
   printf("Elapsed time: %f s\n", elapsed_time);
}
```

7. Questions for the exam

1. Multithreaded programming

1. Performance indicators of parallel programs: acceleration, efficiency factor. Scalability of parallel programs. Laws of Amdal, Gustavson-Barsis.
2. Evolution of the architectures of multiprocessor computer systems (VS). Characteristic of SMP, NUMA systems.
3. Streams in the operating system. The life cycle of flows. Flow planning. Managing threads in POSIX Threads and C ++. Passing arguments to the thread, waiting for the threads to end, disconnecting the threads.
4. The problem of mutual exclusion, the organization of critical sections. Types of locks. The use of locks in POSIX Threads and C ++. Granularity of critical sections.
5. Problems associated with the use of locks, and methods for their prevention. Deadlocks, fasting, inversion of priorities.
6. Thread-safe data structures based on locks. Difficulties arising from their implementation. Illustrate with the example of a thread safe queue with fine-grained locks.
7. Features of multi-threaded programming in C ++: flow management, organization of critical sections. Future results, asynchronous tasks. The RAII pattern.
8. Atomic operations. Fragmented read-write operations. Extraordinary execution of instructions. Types of consistency. The C ++ memory model. The relationship "occurs before" and "synchronizes with". Barriers of memory.

9. Non-blocking thread-safe data structures: features, advantages and disadvantages. Examples of algoritmov and data structures without the use of blocking. The ABA problem. Methods for solving the problem of ABA.
10. Non-blocking thread-safe data structures: features, advantages and disadvantages. Methods for improving the scalability of non-blocking data structures.

2. Parallel programming in the MPI standard

1. Architectural features of distributed aircraft. Basic concepts of the MPI standard. Execution of MPI-programs on the computing cluster. Indicators of the effectiveness of MPI-programs.
2. Principle of parallelization in the MPI standard. Problems encountered in programming in MPI, and methods for solving them. Differentiated exchanges: the definition, classification and examples of their use: ping-pong, message transmission on the ring, random walk algorithm.
3. Collective operations in MPI: definition, classification. Examples of use: determining the number of “pi”, calculating the arithmetic mean, standard deviation, the task of ranking processes, Floyd's algorithm for finding the shortest paths in the graph.
4. Virtual topologies: purpose, basic operations, examples of use. Multiplication of a matrix by a vector. Decomposition of elements of matrices and vectors.
5. The task of classifying documents. The Monte Carlo method. Illustrate the method using the example of calculating the number "pi" and the problem of neutron transport.
6. The linear system equation solution by the Gauss method.
7. Iterative methods for solving systems of linear equations: the Jacobi method and the conjugate gradient method.
8. Derived data types: creation of derived types, examples of use. Management of communicators.
9. Differential equations in partial derivatives: definition, classification. The finite difference method for solving partial differential equations. Construction of difference relations. Modeling of string vibration by finite difference method.
10. Differential equations in partial derivatives: definition, classification. Finite difference method. The solution of the stationary heat distribution problem by the finite difference method.

8. Recommended references

Parallel programming

1. **[Herlihy, 2008]** Herlihy M., Shavit N. The Art of Multiprocessor Programming. – Burlington, Morgan Kaufmann Publishers, 2008. – 529 p.
2. **[Breshears, 2009]** Breshears C. The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications. – O'Reilly Media, 2009. – 303 p.
3. **[McCool, 2012]** McCool M., Reinders J., Robison A. Structured Parallel Programming: Patterns for Efficient Computation. – Morgan Kaufmann Publishers, 2012. – 433 p.
4. **[Mattson, 2004]** Mattson T.G., Sanders B.A., Massingill B.L. Patterns for Parallel Programming. – Addison-Wesley, 2004. – 328 p.
5. **[Andrews, 2003] Andrews** G. Foundations of Multithreaded, Parallel, and Distributed Programming. – Pearson, 2003. – 512 p.
6. **[Lozi, 2014]** Lozi J.P. Towards more scalable mutual exclusion for multicore architectures: дис. – Université Pierre et Marie Curie-Paris VI, 2014.

POSIX Threads

1. **[Lewis, 1996]** Lewis B. Threads Primer: A Guide to Multithreaded Programming. – SunSoft Press, 1996. – 370 p.
2. **[Butenhof, 1997]** Butenhof D. Programming with POSIX Threads. – Addison Wesley Longman, 1997. – 398 p.
3. **[Nichols, 1997]** Nichols B., Buttlar D., Farrell J.P. PThreads Programming. – O'Reilly & Associates, 1997. – 195 p.
4. **[Blaise]** Blaise B. POSIX Threads Programming. – Lawrence Livermore National Laboratory [https://computing.llnl.gov/tutorials/pthreads/].
5. **[Stevens, 2007]** Stevens W.R., Rago S.A. Advanced Programming in the UNIX Environment. – Addison Wesley, 2007. – 1040 p.
6. **[Mitchell, 2003]** Advanced Linux Programming. – New Riders Publishing, 2003. – 368 p.

C++

1. **[Williams, 2012**] Williams A. C++ Concurrency in Action. – Manning Publications, 2012. – 530 p.
2. **[Хьюз, 2004]** Hughes C., Hughes T. Parallel and Distributed Programming Using C++. – Addison-Wesley Professional, 2004. – 691p.

3. **[Meyers, 2014]** Meyers S. Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++ 14. – O'Reilly Media, Inc., 2014. – 336 p. – Chapter 8. The concurrency API.

MPI

1. **[MPI]** MPI: A Message-Passing Interface Standard. Version 3.1. [www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf]
2. **[Quinn, 2004]** Quinn M.J. Parallel Programming in C with MPI and OpenMP. – McGraw-Hill Inc., 2004. – 529 p.
3. **[Balaji, 2013]** Balaji P., Hoefler T. MPI for Dummies // ACM Symposium on Principles and Practice of Parallel Programming, 2013.
4. **[Balaji, 2013]** Balaji P., Hoefler T. Advanced Parallel Programming with MPI-1, MPI-2, and MPI-3 // ACM Symposium on Principles and Practice of Parallel Programming, 2013.
5. **[Gropp-1, 1999]** Gropp W., Lusk E., Skjellum A. Using MPI: portable parallel programming with the message-passing interface. – MIT press, 1999. – 336 p.
6. **[Gropp-2, 1999]** Gropp W., Lusk E., Thakur R. Using MPI-2: Advanced features of the message-passing interface. – MIT press, 1999. – 382 p.