High-Performance Computing Using the Application of the Q-determinant of Numerical Algorithms

Valentina N. Aleeva Rifkhat Zh. Aleev
South Ural State University (National Research University)
Chelyabinsk, Russia
aleevavn@susu.ru aleevrz@susu.ru

***Abstract*—The conception of *Q*-determinant is one of the approaches to parallelizing numerical algorithms. The basic concept of the conception is *Q*-determinant of the algorithm. Here *Q* is the set of operations used by the algorithm. The *Q*-determinant consists of *Q*-terms. Their number is equal to the number of output data. Each *Q*-term describes all possible ways to calculate one of the output data based on the input data. Any numerical algorithm has a *Q*-determinant and can be represented in the form of a *Q*-determinant. This representation is a universal description of numerical algorithms. The representation algorithm in the form of *Q*-determinant makes the numerical algorithm in terms of structure and implementation clearer. Although *Q*-determinant contains only machine-independent properties of the algorithm it can be used to implement algorithms on parallel computing systems. The paper describes the application of the *Q*-determinant to determine the parallelism resource of numerical algorithms and to develop *Q*-effective programs. *Q*-effective program uses the parallelism resource of algorithm completely. At the present time the available theoretical results have tested for numerical algorithms with various structures of *Q*-determinants. For example, they include multiplication algorithms for dense and sparse matrices, Gauss–Jordan, Jacobi, Gauss–Seidel methods for solving systems of linear equations, the sweep method and the Fourier method for solving grid equations, and others. The results of the research can be used to increase the efficiency of implementing numerical algorithms on parallel computing systems.**

Keywords—Q-determinant of algorithm; algorithm representation as Q-determinant; Q-effective implementation of algorithm; Q-effective program; parallelism resource of algorithm; parallel computing system; parallel program

I. INTRODUCTION

A. The Relevance of Research

High-performance computations using parallel computing systems are becoming increasingly necessary in various fields, including smart factory. There is a significant gap between the computational potential of parallel computing systems and its use. The solution of this problem is extremely important. One of the reasons for the problem is that the implementation of algorithms on parallel computing systems is not effective. This is due to the fact that the entire resource of algorithm parallelism is not used. So, the algorithm implementations don't use the computing resources of a parallel system completely. Thus, the research of the parallelism resource of algorithms and its use in the implementation of algorithms on parallel computing systems are very relevant. In this paper we describe the research of the parallelism resource of any numerical algorithm and develop a program using the parallelism resource of the numerical algorithm completely.

B. The Research Significance and Review

A lot of scientific research has been devoted to solving the problem of effective implementation of algorithms on parallel computing systems. Here is a brief overview of some of them. A very important and developed direction to investigate the parallel structure of algorithms and programs for the purpose of their implementation on parallel computing systems is the direction, the bases of which are set forth in [1,2]. The Internet encyclopedia AlgoWiki [3] uses the results of this research. The encyclopedia describes the properties, features, static and dynamic characteristics of specific algorithms. This helps to implement the described algorithms effectively. However, the software study of the parallelism resource of algorithms is not considered under this direction. Also, there is no proposed technology for creating parallel programs using the entire parallelism resource of algorithm. The various approaches to the development of parallel programs have been proposed, dozens of parallel programming languages and many different tools have been created during the development of parallel computing. The T-system [4,5] is one of these developments. It represents a programming environment with support of automatic dynamic parallelization of programs. However, it can't be asserted parallel programs created with the help of the T-system use the parallelism resource of algorithm completely. Another approach to creating parallel programs is the parallel program synthesis. The parallel programs synthesis method is to construct new parallel algorithms using the knowledge base of parallel algorithms for solving more complex problems. The technology of fragmented programming, its implementing language, and programming system LuNA are developed on the base of parallel programs synthesis method. Now this research is developing [6,7]. This approach doesn't solve the problem of research and use of the algorithm parallelism resource, despite the fact that it is universal. To overcome resource limitations, the author of paper [8] suggests methods of constructing parallel programs using a functional programming language independent of computer architecture. This approach is developing also. However, there is not shown

The reported study was funded by RFBR according to the research project № 17-07-00865 a. The work was supported by Act 211 Government of the Russian Federation, contract № 02.A03.21.0011.

the created programs use the entire parallelism resource of algorithms. There are many studies of the development of parallel programs take into account the specific nature of algorithms and the architecture of parallel computing systems. Examples of such studies are [9-17]. Those studies increase the efficiency of implementing specific algorithms or implementing algorithms on parallel computing systems of a particular architecture. However, they do not provide general universal approach. I would like to note that in such studies there is no information to what extent is the parallelism resource of algorithm used. So, a new study can be appeared with the better parallelism resource of algorithm.

C. The Purpose and Objectives of the Research

The purpose of this paper is the description of solutions of the parallelism resource investigation for any numerical algorithm and the parallel programs development use the parallelism resource of algorithm completely. Also we show the practical implementation of these solutions. For this purpose we use the conception of the *Q*-determinant and solve the following objectives.

1. Development of methods for the parallelism resource investigation of any numerical algorithm.
2. Development of a method for the parallel program design using the parallelism resource of numerical algorithms completely.
3. Practical applications of methods the parallelism resource investigation of numerical algorithms and executions of computational experiments.
4. Practical application of the method of the parallel program design and executions computational experiments.

II. The Parallelism Resourse Investigation of Numerical Algorithms and Design of Parallel Programs

A. The conception of Q-determinant

The theoretical basis of this study is the conception of the *Q*-determinant. It was first described in [18]. We will give the basic definitions of that conception for a better understanding of our work. Let α be an algorithm for solving an algorithmic problem $\bar{y} = F(N, B)$, where $N = \{n_1, \dots, n_k\}$ is a set of dimension parameters of the problem or N is empty set, B is a set of input data, $\bar{y} = (y_1, \dots, y_m)$ is a set of output data. Let $\bar{N}$ be a vector $(\bar{n}_1, \dots, \bar{n}_k)$, where $\bar{n}_i$ is some assigned value of parameter n_i $(i = 1, \dots, k)$, $\{\bar{N}\}$ be a set of possible vectors $\bar{N}$, Q be a set of operations used by algorithm α. Here, the expression over B and Q should be understood as the interpretation of the term of the signature Q in the standard sense of mathematical logic by set of variables B [19]. Let V be a set of all expressions over B and Q. Every single-valued mapping $w: \{\bar{N}\} \to V$ is called unconditional *Q*-term. If the expression $w(\bar{N})$ takes a value of a logical type for all $\bar{N} \in \{\bar{N}\}$ and interpretation of the variables B then w is called an unconditional logical *Q*-term. Let $u_1, \dots, u_l$ be unconditional logical *Q*-terms, $w_1, \dots, w_l$ be unconditional *Q*-terms. We denote the set of pairs (u_i, w_i) $(i = 1, \dots, l)$ as $(\bar{u}, \bar{w}) = \{(u_i, w_i)\}_{i=1,\dots,l}$ and call conditional *Q*-term to length l. Assume we have a countable set of pairs unconditional *Q*-terms $(\bar{u}, \bar{w}) = \{(u_i, w_i)\}_{i=1,2,\dots}$ such that $\{(u_i, w_i)\}_{i=1,\dots,l}$ is conditional *Q*-term for each $l < \infty$ then we call it conditional infinite *Q*-term. We will call just *Q*-term if it does not matter whether the *Q*-term unconditional, conditional or conditional infinite. We mean the computation of unconditional *Q*-term w in each interpretation of B is the computation of the expression $w(\bar{N})$ in a certain vector $\bar{N} \in \{\bar{N}\}$. To compute the conditional *Q*-term $(\bar{u}, \bar{w}) = \{(u_i, w_i)\}_{i=1,\dots,l}$ for a given interpretation of B and some $\bar{N} \in \{\bar{N}\}$, it is necessary to find a pair $u_{i_0}(\bar{N}), w_{i_0}(\bar{N})$ such that $u_{i_0}(\bar{N})$ is $true$, and $w_{i_0}(\bar{N})$ is defined. In this case, the value of $(\bar{u}, \bar{w})$ is equal to $w_{i_0}(\bar{N})$. If it is determined that the pair of expression $u_{i_0}(\bar{N}), w_{i_0}(\bar{N})$ does not exist, then the value $(\bar{u}, \bar{w})$ for this interpretation of B and the given $\bar{N}$ is undefined. The calculation of a conditional infinite *Q*-term can be defined similarly. Suppose that I_1, I_2, I_3 are the subsets of the set $I = (1, \dots, m)$, satisfying the following condition. One or two subsets I_i $(i = 1,2,3)$ may be empty, $I_1 \cup I_2 \cup I_3 = I$, $I_i \cap I_j = \emptyset$ $(i \neq j; i, j = 1,2,3)$. Also suppose that there is a following set of *Q*-terms $\{f_i\}_{i \in I}$:

1. f_{i_1} is unconditional *Q*-term for $i_1 \in I_1$, $f_{i_1} = w^{i_1}$;
2. f_{i_2} is conditional *Q*-term for $i_2 \in I_2$, $f_{i_2} = \{(u_j^{i_2}, w_j^{i_2})\}_{j=1,\dots,l_{i_2}}$, l_{i_2} is computable function of parameters N;
3. f_{i_3} is a conditional infinite *Q*-term for $i_3 \in I_3$, $f_{i_3} = \{(u_j^{i_3}, w_j^{i_3})\}_{j=1,2,\dots}$.

Suppose that the algorithm α is the determination y_i $(i \in I)$ by calculating the *Q*-term f_i $(i \in I)$. Then the set of *Q*-terms f_i $(i \in I)$ is called the *Q*-determinant of the algorithm α, and the representation of the algorithm α in the form

$$y_i = f_i \ (i \in I)$$

is called the representation of the algorithm α in the form of a *Q*-determinant. We give examples of the algorithm representations in the form of a *Q*-determinant.

Consider the algorithm of multiplication of matrices $A = [a_{ij}]_{i=1,\dots,n;j=1,\dots,k}$ and $B = [b_{ij}]_{i=1,\dots,k;j=1,\dots,m}$. The result is matrix $C = [c_{ij}]_{i=1,\dots,n;j=1,\dots,m}$, where $c_{ij} = \sum_{s=1}^{k} a_{is} b_{sj}$. Thus, the algorithm of matrix multiplication is represented in the form of the *Q*-determinant already. *Q*-determinant is composed of nm unconditional *Q*-terms.

Let us consider the Gauss–Jordan method for solving of linear equation systems $A\bar{x} = \bar{b}$ also. We assume that $A = [a_{ij}]_{i,j=1,\dots,n}$ is a matrix of dimension $n \times n$ with a nonzero determinant, $\bar{x} = (x_1, \dots, x_n)^T$, $\bar{b} = (a_{1,n+1}, \dots, a_{n,n+1})^T$. $\bar{A}$ is augmented matrix of the system. Gauss-Jordan method consists of n steps. We select the first non-zero element in the row k of the matrix A as the leading element in step k $(k = 1, \dots, n)$. We denote the column number of the leading element by j_k. We denote $\bar{A}^{j_1 \dots j_k} = [a_{ij}^{j_1 \dots j_k}]_{i=1,\dots,n;j=1,\dots,n+1}$ that is the augmented matrix of the system of linear equations obtained after step k $(k = 1, \dots, n)$. We get system of equations $\bar{A}^{j_1 \dots j_n} \bar{x} = \bar{b}^{j_1 \dots j_n}$ after step n, where $\bar{A}^{j_1 \dots j_n} =$

$[a_{ij}^{j_1\dots j_n}]_{i=1,\dots,n;j=1,\dots,n+1}$, $\bar{b}^{j_1\dots j_n} = (a_{1,n+1}^{j_1\dots j_n}, \dots, a_{n,n+1}^{j_1\dots j_n})^T$. We denote by

$$L_{j_1} = \bigwedge_{j=1}^{j_1-1}(a_{1j} = 0), \text{ if } j_1 \neq 1,\ L_{j_1} = true, \text{ if } j_1 = 1,$$
$$L_{j_l} = \bigwedge_{j=1}^{j_l-1}\left(a_{lj}^{j_1\dots j_{l-1}} = 0\right), \text{ if } j_l \neq 1,\ L_{j_l} = true, \text{ if } j_l = 1 \quad (l = 2, \dots, n).$$

Permutations of elements $(1, \dots, n)$ may be numbered. Let i be a number of permutation $(j_1, \dots, j_n)$. Then the terms

$$w_i^{j_l} = a_{i,n+1}^{j_1\dots j_n}\ (l = 1, \dots, n),$$
$$u_i = L_{j_1} \wedge (a_{1j_1} \neq 0) \wedge \left(\bigwedge_{l=2}^{n}(L_{j_l} \wedge (a_{lj_l}^{j_1\dots j_{l-1}} \neq 0))\right)$$

are unconditional Q-terms. Q-determinant of Gauss–Jordan method consists of n conditional Q-terms and

$$x_j = \left\{\left(u_1, w_1^j\right), \dots, \left(u_{n!}, w_{n!}^j\right)\right\}\ (j = 1, \dots, n)$$

is the representation in the form of Q-determinant.

We consider the Gauss–Seidel method for solving a system of linear equations $A\bar{x} = \bar{b}$ as an example of an algorithm whose Q-determinant contains conditional infinite Q-terms. Let $A = [a_{ij}]_{i,j=1,\dots,n}$, $a_{ii} \neq 0\ (i = 1, \dots, n)$, $\bar{x} = (x_1, \dots, x_n)^T$, $\bar{b} = (a_{1,n+1}, \dots, a_{n,n+1})^T$. We denote as $c_{ij} = -a_{ij}/a_{ii}$ and $d_i = b_i/a_{ii}\ (i, j = 1, \dots, n)$. Let $\bar{x}^0 = (x_1^0, \dots, x_n^0)$ be an initial approximation. Then the iteration process can be written as

$$x_i^{k+1} = \sum_{j=1}^{i-1} c_{ij}x_j^{k+1} + \sum_{j=i+1}^{n} c_{ij}x_j^k + d_i \quad (i = 1, \dots, n; k = 1,2, \dots)$$

The criterion of the iterative process ending is the condition $\|x^{k+1} - x^k\| < \varepsilon$. There ε is the calculation precision. Q-determinant of the Gauss–Seidel method consists of n conditional infinite Q-terms. Presentation of the Gauss–Seidel method in the form of Q-determinant is written as

$$x_i = \left\{(\|\bar{x}^1 - \bar{x}^0\| < \varepsilon, x_i^1), \dots, \left(\|\bar{x}^k - \bar{x}^{k-1}\| < \varepsilon, x_i^k\right), \dots\right\} \quad (i = 1, \dots, n).$$

Let the algorithm α be represented in the form of the Q-determinant $y_i = f_i\ (i \in I)$. The computation process of Q-terms $f_i\ (i \in I)$ for a given interpretation of B and some $\bar{N} \in \{\bar{N}\}$ is called an implementation of the algorithm α. The implementation of the algorithm is called Q-effective if it is such that the expressions

$$W(\bar{N}) = \{w^{i_1}(\bar{N})(i_1 \in I_1); u_j^{i_2}(\bar{N}), w_j^{i_2}(\bar{N})(i_2 \in I_2, j = 1, \dots, l_{i_2});\ u_j^{i_3}(\bar{N}), w_j^{i_3}(\bar{N})\left(i_3 \in I_3,\ j = 1,2, \dots\right)\}$$

are computed simultaneously, and the operations are performed straightway their operands are computed. If the algorithm allows parallelization, then its Q-effective implementation uses the internal parallelism resource of algorithm completely, therefore Q-effective implementation is the most parallel implementation of the algorithm. The expression and its operations have nesting levels. We will denote the number of nesting levels of the expression $w(\bar{N})$ by $T^{w(\bar{N})}$. We will name a chain of expressions to length n if this expression is obtained by applying one of the associative operations of the set Q without the use of parentheses $n - 1$ times. When determining the level of nesting of an expression and its subexpressions, the order of execution of chain operations is specified by a doubling scheme. For example, the doubling scheme of calculating the chain $a_1 + a_2 + a_3 + a_4$ is the following. First, we calculate $b_1 = a_1 + a_2$ and $b_2 = a_3 + a_4$ simultaneously, after that we calculate $c = b_1 + b_2$. The implementation of algorithm α is called realizable if it is necessary to perform a finite number of operations simultaneously. There are algorithms, for which the Q-effective implementation is not realizable.

Consider the characteristics of the parallel complexity of a realizable Q-effective implementation of the algorithm α:

1. $D_\alpha(\bar{N})$ is the maximum number of nesting levels of the expressions $W(\bar{N})$, i.e. $D_\alpha(\bar{N}) = \max_{w(\bar{N})\in W(\bar{N})} T^{w(\bar{N})}$.
2. $P_\alpha(\bar{N})$ is the maximum number of operations of all nesting levels of all expressions $W(\bar{N})$, i.e. $P_\alpha(\bar{N}) = \max_{1\le r\le D_\alpha(\bar{N})} \sum_{w(\bar{N})\in W(\bar{N})} O_r^{w(\bar{N})}$, where $O_r^{w(\bar{N})}$ is the number of operations of the nesting level r of the expression $w(\bar{N})$.

$D_\alpha(\bar{N})$ characterizes the execution time of the Q-effective implementation of the algorithm, and $P_\alpha(\bar{N})$ characterizes the number of processors necessary to perform the Q-effective implementation of the algorithm. $D_\alpha(\bar{N})$ is called the height of the algorithm, and $P_\alpha(\bar{N})$ is called the width of the algorithm.

B. The Methods to Investigation the Parallelism Resource of Numerical Algorithms

The following methods are developed to investigation the parallelism resource of numerical algorithms.

1. The method of constructing of the Q-determinant of the algorithm based on the flowchart of the algorithm.
2. The method of obtaining of Q-effective implementation of the algorithm based on the Q-determinant of the algorithm.
3. The method of calculating of the parallel complexity characteristics of the Q-effective implementation of the algorithm.
4. The method of comparing the parallel complexity characteristics of Q-effective implementations of algorithms.

The numerical algorithm is rarely presented in the form of a Q-determinant. So, it is necessary to have a method of constructing the Q-determinant based on the generally accepted representation of the algorithm, for example, the flowchart. We will analyze the flowchart of the algorithm for fixed dimension parameters of the algorithmic problem. The features of various algorithms were investigated to develop the method of constructing the Q-determinant of the algorithm based on the algorithm flowchart. Let the Q-determinant of the algorithm consist of unconditional Q-terms. Then it has the feature that the pass through the flowchart of the algorithm should be carried out sequentially, as if the algorithm were being executed. After passing through the flowchart of the algorithm, the expressions $w^{i_1}(\bar{N})\ (i_1 \in I_1)$ will be obtained as the values of the Q-terms $f_{i_1}(i_1 \in I_1)$. These expressions are formed using the contents of the blocks involved in the calculation of $y_{i_1}(i_1 \in I_1)$. Let the Q-determinant of the algorithm consist of

conditional *Q*-terms. Then the passage through the flowchart of the algorithm branches out and each branch is processed separately. When the processing of one branch is completed, the flowchart handler returns to the nearest block where the branch has occurred and continues processing with the opposite condition written in this block. In this case the generated *Q*-term is a list of pairs $(u_j^{i_2}(\bar{N}), w_j^{i_2}(\bar{N}))\left(i_2 \in I_2, j = 1, \dots, l_{i_2}\right)$. The same goes for algorithms whose *Q*-determinant consists of conditional infinite *Q*-terms. The problem of storing conditional infinite *Q*-terms is solved by limiting the number of iterations of the algorithm. It is assumed that for such algorithms it is possible to determine the necessary number of iterations. The result of applying the described method to the numerical algorithm is the set of *Q*-terms f_i $(i \in I)$ for some value $\bar{N} \in \{\bar{N}\}$. Consequently, a set of expressions $W(\bar{N})$ is formed. The method of constructing the *Q*-determinant of the algorithm on the basis of the algorithm flowchart is described in [20] in detail.

Nesting levels of operations that occur in the expression $W(\bar{N})$ can be calculated by determining the nesting level of the operation. The method of obtaining a *Q*-efficient implementation of the algorithm based on the *Q*-determinant is designed to perform these calculations. The set of expressions $W(\bar{N})$ with the calculated nesting levels of operations is the execution plan of the *Q*-effective implementation of the algorithm. The method of calculating the characteristics of the parallel complexity of *Q*-effective implementation of the algorithm uses the execution plan of *Q*-effective implementation of the algorithm. The method determines the maximum value of the nesting level of the operations into the expressions $(\bar{N})$ to calculate $D_\alpha(\bar{N})$. The method finds the number of operations $\sum_{w(\bar{N}) \in W(\bar{N})} O_r^{w(\bar{N})}$ for each the each nesting level r $(1 \le r \le D_\alpha(\bar{N}))$, and determines the maximum value of them and then we calculate $P_\alpha(\bar{N})$. We note if we obtain the width estimate of the algorithm $P_\alpha(\bar{N})$ with the help of this method then we can often exceed the number of processors used simultaneously in the implementation of the algorithm on a parallel computing system. This is due to the fact that it is necessary to attempt for the algorithm implementation to do the computation without duplication the same expressions and subexpressions, of which *Q*-terms consist. However, it should also be remembered that the use of distributed memory may cause eliminating duplication is not always advisable, since it can lead to slow performance due to additional transfers between processor nodes.

We propose to save the results of the calculations obtained with the methods described above in the database. Thus, the database must contain a unique identifier, a name, a text description of the algorithm being investigated, its *Q*-determinants for various values of the dimension parameters $\bar{N} \in \{\bar{N}\}$, together with the characteristics of the parallel complexity of the *Q*-effective implementation of the algorithm $D_\alpha(\bar{N})$ and $P_\alpha(\bar{N})$. The method of comparing the characteristics of the parallel complexity of *Q*-effective implementations of algorithms uses the information contained in the database. He compares the characteristics of the parallel complexity of *Q*-effective implementations of two algorithms that solve the same algorithmic problem for identical sets of values of the dimension parameters N. This makes it possible to define an algorithm with a better characteristic $D_\alpha(\bar{N})$ or $P_\alpha(\bar{N})$.

C. Method of Parallel Program Designing

The model of the *Q*-determinant conception allows us to investigate the machine-independent properties of numerical algorithms. However, it does not take into account the specific features of their implementation on parallel computing systems. For example, it does not take into account the dependence of the implementations of algorithms on the losses when we use memory. So, we extend the model of the *Q*-determinant concept by addition of two submodels. These submodels are parallel computing models that present the architectures of target computing systems with shared and distributed memory in an abstract form. The initial model of the *Q*-determinant will be called the base model. We will use PRAM (Parallel Random Access Machine) [21] as a model of parallel computing for computational systems with shared memory. Now OpenMP technology is most widely used for the organization of parallel computing on multiprocessor systems with shared memory. This technology will be used in the development of parallel programs based on the extended model of the conception of *Q*-determinant for computing systems with shared memory. We will use the BSP (Bulk-Synchronous Parallel) model [22] as a model of parallel computing for distributed memory computing systems. One of the frequently used frameworks in parallel and distributed programming is the master-slaves paradigm [23]. Here, we limit ourselves to the master-slaves paradigm with a configuration that includes one master and many slaves for the extended *Q*-determinant model in the submodel for distributed memory. This configuration is the most popular. The MPI (Message Passing Interface) [24] is de-facto a standard for parallel programming on distributed memory. This technology will be used in the development of parallel programs based on the extended model of the conception of the *Q*-determinant for distributed memory computing systems.

The extended model of the conception of the *Q*-determinant is base of the method of designing parallel programs that use the parallelism resource of numerical algorithms completely. This method has three stages.

1. The construction of *Q*-determinant of algorithm.
2. The execution plan description of *Q*-effective implementation of algorithm.
3. The program development for execution of *Q*-effective implementation of algorithm, if it is realizable.

We don't fix the values of the parameters of dimension *N* under construction of the *Q*-determinant. The first two stages of the method use the basic model of the *Q*-determinant conception. The third stage is done with the use of the submodels of the extended model for computational systems with shared and distributed memory. We use the execution plan of the *Q*-effective implementation obtain in stage 2 to develop a program for shared memory computing systems in stage 3. We must supplement the execution plan of the *Q*-effective implementation with a plan of the computation distribution over the nodes if the distributed memory use. We

can evaluate the parameters of the models of parallel computing PRAM and BSP by use the execution plan of the *Q*-effective implementation of the algorithm and the plan for computation distribution over the nodes give. A program is called *Q*-effective if it creates with use of the described method of designing parallel programs. The process of creation of *Q*-effective programs is called *Q*-effective programming. *Q*-effective program uses the parallelism resource of algorithm completely because it execution a *Q*-effective implementation of the algorithm.

III. Practical Application of the Proposed Methods

A. *The Program System for Investigation the Parallelism Resource of Numerical Algorithms*

At the present time, we have the development of the first version of the software system, which implements methods to investigation the parallelism resource of numerical algorithms. The most important and most complex module of the software system realizes the method of the *Q*-determinant construction of the algorithm based on the flowchart of the algorithm. The software implementation of this module was first described in [20]. We use the JSON format as the format of the algorithm flowchart representation because it allows us to work with a description of flowcharts in text format. *Q*-determinants can be formed both in string format and in JSON format. Examples of description of flowcharts and *Q*-determinants of some algorithms are given in [20]. The program is developed using the C# programming language on the base .NET platform. Development was carried out in accordance with the paradigm of object-oriented programming. Microsoft Visual Studio was used as an integrated development environment. The program testing was performed using algorithms whose *Q*-determinants contain different types of *Q*-terms. We have among them the algorithm for multiplying matrices, the algorithm of the maximum element finding of a number array, the Euclidean algorithm for integers, the Jacobi method for solving a system of linear equations, and others.

Recently, the software implementation of the method of the *Q*-determinant construction of the algorithm based on the flowchart of the algorithm has been further developed. The new version of the program generates a file containing a more detailed representation of the structure of the *Q*-determinant. This provides more opportunities for investigating machine-independent properties of algorithms. We will describe the structure of the generated file. Each conditional *Q*-term corresponds to as many lines of the file as the length l of the *Q*-term. Each of the lines contains the identifier of the output variable computed by the given *Q*-term, equality sign and one pair (u_i, w_i) $(i = 1, \dots, l)$. The *Q*-terms u_i and w_i are described in the JSON format and separated by a semicolon. An unconditional *Q*-term corresponds to one line of the file. In this case there is no logical *Q*-term, so we use the space instead of it. The representation of the algorithm in the form of a *Q*-determinant for fixed parameters of the dimension N of the algorithmic problem is formed with the help of the program actually.

An example of a file describing the *Q*-determinant of the matrix multiplication algorithm $C = A \times B$ is shown in fig. 1. Fig. 2 shows the contents of the file describing the *Q*-determinant of the algorithm for finding of the maximum element max in the collection of numbers $A(i)$ $(i = 1, \dots, n)$. Fig. 3 shows a description of the *Q*-determinant generated by the program computing of the one iteration of the Jacobi method for solving a system of linear equations $AX = B$. Here $X0$ is the initial approximation, e is the computation accuracy. Dimension parameters N are chosen to be minimal for clarity.

```
C(1,1)= ;{"operation": "+", "firstOperand": {"operation": "*",
"firstOperand": "A(1,1)", "secondOperand": "B(1,1)"},
"secondOperand": {"operation": "*", "firstOperand": "A(1,2)",
"secondOperand": "B(2,1)"}}
C(1,2)= ;{"operation": "+", "firstOperand": {"operation": "*",
"firstOperand": "A(1,1)", "secondOperand": "B(1,2)"},
"secondOperand": {"operation": "*", "firstOperand": "A(1,2)",
"secondOperand": "B(2,2)"}}
C(2,1)= ;{"operation": "+", "firstOperand": {"operation": "*",
"firstOperand": "A(2,1)", "secondOperand": "B(1,1)"},
"secondOperand": {"operation": "*", "firstOperand": "A(2,2)",
"secondOperand": "B(2,1)"}}
C(2,2)= ;{"operation": "+", "firstOperand": {"operation": "*",
"firstOperand": "A(2,1)", "secondOperand": "B(1,2)"},
"secondOperand": {"operation": "*", "firstOperand": "A(2,2)",
"secondOperand": "B(2,2)"}}
```

Fig. 1. *Q*-determinant of the matrix multiplication algorithm.

```
max={"operation": "&", "firstOperand": {"operation": "<",
"firstOperand": "A(1)", "secondOperand": "A(2)"},
"secondOperand": {"operation": "<", "firstOperand": "A(2)",
"secondOperand": "A(3)"}};A(3)
max={"operation": "&", "firstOperand": {"operation": "<",
"firstOperand": "A(1)", "secondOperand": "A(2)"},
"secondOperand": {"operation": ">=", "firstOperand": "A(2)",
"secondOperand": "A(3)"}};A(2)
max={"operation": "&", "firstOperand": {"operation": ">=",
"firstOperand": "A(1)", "secondOperand": "A(2)"},
"secondOperand": {"operation": "<", "firstOperand": "A(1)",
"secondOperand": "A(3)"}};A(3)
max={"operation": "&", "firstOperand": {"operation": ">=",
"firstOperand": "A(1)", "secondOperand": "A(2)"},
"secondOperand": {"operation": ">=", "firstOperand": "A(1)",
"secondOperand": "A(3)"}};A(1)
```

Fig. 2. *Q*-determinant of the algorithm for the maximum element finding in the collection of numbers.

```
X(1)={"operation": "<", "firstOperand": {"operation": "+",
"firstOperand": {"operation": "abs", "Operand": {"operation":
"-", "firstOperand": "X0(1)", "secondOperand": {"operation":
"/", "firstOperand": {"operation": "-", "firstOperand": "B
(1)", "secondOperand": {"operation": "*", "firstOperand": "A
(1,2)", "secondOperand": "X0(2)"}}, "secondOperand": "A
(1,1)"}},"secondOperand":{"operation": "-", "firstOperand":
"X0(2)", "secondOperand": {"operation": "/", "firstOperand":
{"operation": "-", "firstOperand": "B(2)", "secondOperand":
{"operation": "*", "firstOperand": "A(2,1)", "secondOperand":
"X0(1)"}}, "secondOperand": "A(2,2)"}}}, "secondOperand":
"e"};{"operation": "/", "firstOperand": {"operation": "-",
"firstOperand": "B(1)", "secondOperand": {"operation": "*",
"firstOperand": "A(1,2)", "secondOperand": "X0(2)"}},
"secondOperand": "A(1,1)"}
X(2)={"operation": "<", "firstOperand": {"operation": "+",
"firstOperand": {"operation": "abs", "Operand": {"operation":
"-", "firstOperand": "X0(1)", "secondOperand": {"operation":
"/", "firstOperand": {"operation": "-", "firstOperand": "B
(1)", "secondOperand": {"operation": "*", "firstOperand": "A
(1,2)", "secondOperand": "X0(2)"}}, "secondOperand": "A
(1,1)"}},"secondOperand":{"operation": "-", "firstOperand":
"X0(2)", "secondOperand": {"operation": "/", "firstOperand":
{"operation": "-", "firstOperand": "B(2)", "secondOperand":
{"operation": "*", "firstOperand": "A(2,1)", "secondOperand":
"X0(1)"}}, "secondOperand": "A(2,2)"}}}, "secondOperand":
"e"};{"operation": "/", "firstOperand": {"operation": "-",
"firstOperand": "B(2)", "secondOperand": {"operation": "*",
"firstOperand": "A(2,1)", "secondOperand": "X0(1)"}},
"secondOperand": "A(2,2)"}
```

Fig. 3. The *Q*-determinant of the Jacobi method.

The database of the software system was created using the Microsoft SQL Server database management system. A server application has been developed to interact with the database. The tasks of the server application are reading, adding, editing and deleting information about algorithms, as well as reading, adding and deleting *Q*-determinants of algorithms. Also, there are developed and implemented algorithms to obtain the execution plan of *Q*-effective implementation of the algorithm, to calculate the characteristics of the parallel complexity of a *Q*-effective implementation of the algorithm, to compare the characteristics of the parallel complexity of *Q*-effective implementations of two algorithms. A client application was created for the users of the software system. The implementation of the database, its applications, as well as the results of computational experiments are described in the article [25] in more detail.

The resource of internal parallelism of any numerical algorithm is studied by the software system that implements the methods to investigation the parallelism resource of numerical algorithms. In addition, it makes it possible to choose an algorithm with the best resource of internal parallelism from several algorithms that solve the same algorithmic problem. The method of designing parallel programs using the complete parallelism resource of numerical algorithms can be applied to the chosen algorithm. It is possible to construct the *Q*-determinant of any numerical algorithm using given software system. This software system makes possible to automate the study of various properties of algorithms by developing new system options because the *Q*-determinant contains all the machine-independent properties of the algorithm, therefore.

B. Development of Q-effective Programs

The possibility of practical application of the method of designing *Q*-effective programs and the experimental study of the developed *Q*-effective programs are described in several works by the examples of algorithms having different structures of *Q*-determinants. The research was performed on the supercomputer «Tornado» of South Ural State University. The programs for shared memory were executed on one compute node. Programs for distributed memory used several compute nodes. The algorithms for the multiplication of dense and sparse matrices were investigated in [26]. The Gauss–Jordan method for solving a system of linear equations is considered in [27]. The results of a study of the Jacobi method for solving a system of linear equations are given in [28]. *Q*-effective programs for shared and distributed memory are developed for these algorithms and estimates of their dynamic characteristics are obtained. Fig. 4 and fig. 6 show the acceleration of *Q*-effective programs of the matrix multiplication algorithm for shared and distributed memory [26]. Also, fig. 5 and fig. 7 show the efficiency of *Q*-effective programs of the matrix multiplication algorithm for shared and distributed memory [26].

The method of designing *Q*-effective programs was tested on algorithms having small and large parallelism resources. The sweep method for solving a system of linear three-point equations is considered in [29]. It has a small parallelism resource. The use of distributed memory is not justified in that case because it can lead to a reduced performance. The Fourier method for solving the system of difference equations is considered in [29] also. It has a large parallelism resource and can be implemented efficiently on distributed memory using the master-slaves principle. The study of the practical application of the *Q*-effective program design method is shown that there are algorithms, for which the principle of master-slaves can be inappropriate, because the number of transfers between computing nodes increases. An example of such an algorithm can be the Jacobi method for solving five-point difference equations. The approbation of the method of designing *Q*-effective programs using algorithms with the complex structure *Q*-determinant has a certain interest also. For example, such algorithms include the Gauss–Jordan method [27] and the Gauss–Seidel method for solving a system of linear equations [30]. These algorithms have complex *Q*-determinants but they have well-structured Q-determinants and *Q*-effective implementations. Also we can apply *Q*-effective programming to other well-known widely used algorithms having similar structures. This fact enables the creation of *Q*-effective programs for them. Fig. 8 and fig. 10 show the acceleration of *Q*-effective programs of the Gauss–Seidel method for shared and distributed memory [30]. Also, fig. 9 and fig. 11 show the efficiency of *Q*-effective programs of the Gauss–Seidel method for shared and distributed memory [30].

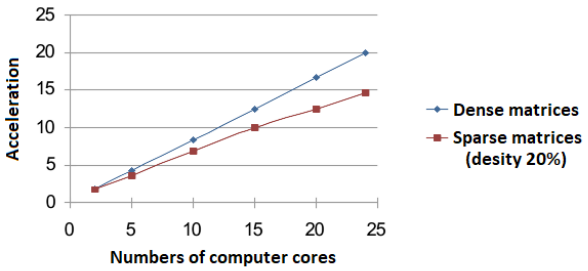

Fig. 4. The acceleration of *Q*-effective program of the matrix multiplication algorithm for shared memory.

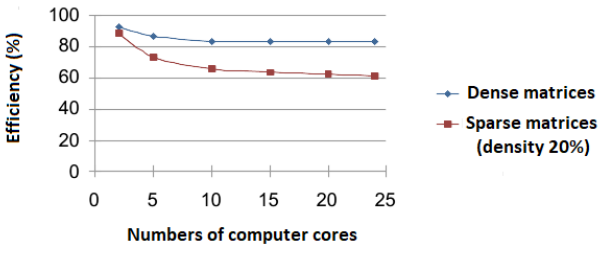

Fig. 5. The efficiency of *Q*-effective program of the matrix multiplication algorithm for shared memory.

IV. Conclusion

We can make a conclusion that the presented research results about the *Q*-determinant make an opportunity to express and evaluate the existing internal parallelism of any numerical algorithm, as well as to show a possible way of its parallel execution. We see that the *Q*-determinant of the algorithm helps to the developer in the creation a program for execution of *Q*-effective implementation of the algorithm as well as the algorithm flowchart enables to developer in the creation a

sequential program. So, it becomes possible to obtain effective implementations of algorithms for real parallel computing systems.

The extended model of the *Q*-determinant conception can be developed further by adding models of parallel computations for computing systems with other architectural features. Also, various programming languages and parallel programming technologies can be used to create *Q*-effective programs. So, a potentially infinite set of *Q*-effective programs can be created for a single numerical algorithm. All of them execute a *Q*-effective implementation of the algorithm that it is the most rapid implementation of the algorithm from a formal point of view. It is quite possible, there is no the best of all *Q*-effective programs for given algorithm in terms of performance, but each of those programs can be most effective in the computing infrastructure, for which it is created.

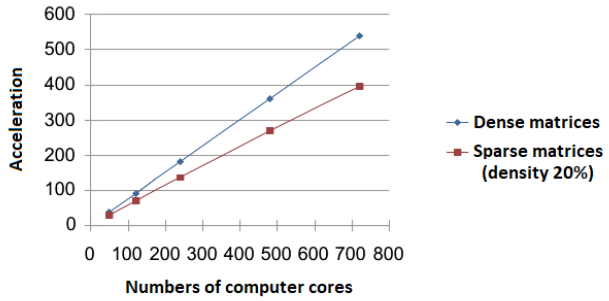

Fig. 6. The acceleration of *Q*-effective program of the matrix multiplication algorithm for distributed memory.

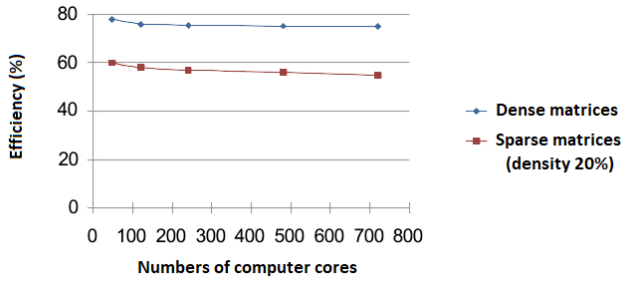

Fig. 7. The efficiency of *Q*-effective program of the matrix multiplication algorithm for distributed memory.

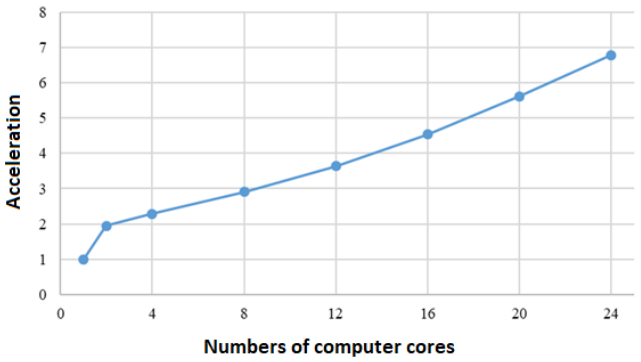

Fig. 8. The acceleration of *Q*-effective program of the Gauss–Seidel method for shared memory.

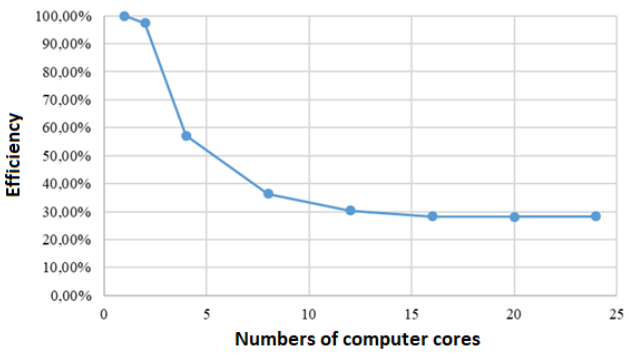

Fig. 9. The efficiency of *Q*-effective program of the Gauss–Seidel method for shared memory.

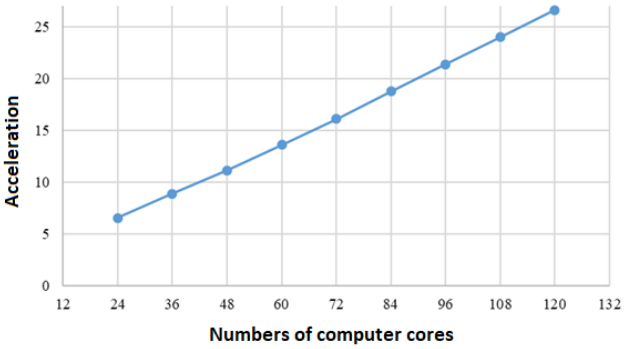

Fig. 10. The acceleration of *Q*-effective program of the Gauss–Seidel method for distributed memory.

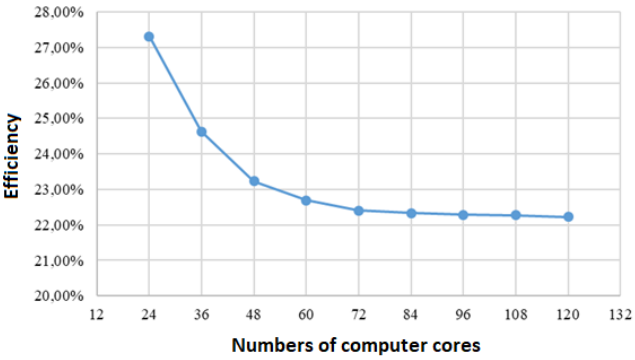

Fig. 11. The efficiency of *Q*-effective program of the Gauss–Seidel method for distributed memory.

REFERENCES

[1] V.V. Voevodin, V.V. Voevodin, “The V-Ray technology of optimizing programs to parallel computers,” In: Vulkov, L.G., Yalamov, P., Waśniewski, J. (eds.) WNAA 1996. LNCS, vol. 1196, Springer, Heidelberg, 1997, pp. 546-556.

[2] V.V. Voevodin, V.V. Voevodin, Parallel computing, BHV-Petersburg St.Petersburg, 2002. (in Pussian)

[3] Open Encyclopedia of Parallel Algorithmic Features. (http://algowiki-project.org/en/Open_Encyclopedia_of_Parallel_Algorithmic_Features)

[4] S.M. Abramov, A.I. Adamovich, and M.R. Kovalenko, “T-system as a programming environment with automatic dynamic support parallelization support. An example with implementation of ray-

tracing algorithm," Programmirovanie, vol. 25(2), 1999, pp. 100-107 (in Russian)

[5] S.M. Abramov, V.A. Vasenin, E.E. Mamchits, V.A. Roganov, and A.F. Slepukhin, "Dynamic parallelization of programs based on parallel graph reduction. A software architecture of new T-system version," Proceedings book of MIPHI scientific session, January 22-26 2001, vol. 2, Moscow, 2001. (in Russian)

[6] V.E. Malyshkin, V.A. Perepelkin, and G.A. Schukin, "Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA," In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2015. Lecture Notes in Computer Science, vol. 9251. Springer, 2015, pp. 80-85.

[7] V.E. Malyshkin, V.A. Perepelkin, and A.A. Tkacheva, "Control Flow Usage to Improve Performance of Fragmented Programs Execution," In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2015. Lecture Notes in Computer Science, vol. 9251. Springer, 2015, pp.86-90.

[8] A.I. Legalov, "Functional language for creating architecturally independent parallel programs," In: Computational technologies, № 1 (10), 2005, pp. 71-89. (in Pussian)

[9] Y.L. Gurieva, V.P. Il'in, "On Parallel Computational Technologies of Augmented Domain Decomposition Methods," In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2015. Lecture Notes in Computer Science, vol. 9251. Springer, 2015, pp.35-46.

[10] D.A. Suplatov, V.V. Voevodin, and V.K. Svedas, "Robust enzyme design: Bioinformatic tools for improved protein stability," In: Biotechnology journal, vol. 10, № 3. Publishing Wiley - VCH Verlag GmbH & CO. KGaA (Germany), 2015, pp. 344-355.

[11] M.G. Venkata, P. Shamis, R. Sampath, R.L. Graham, and J.S. Ladd, "Optimizing blocking and nonblocking reduction operations for multicore systems: hierarchical design and implementation," In: Proceedings of IEEE Cluster, 2013, pp. 1-8.

[12] Martin Schlueter and Masaharu Munetomo, "Parallelization strategies for evolutionary algorithms for MINLP," In: IEEE Congress on Evolutionary Computation, 2013, pp. 635-641.

[13] Qinglin Wang, Jie Liu, Xiantuo Tang, Feng Wang, Guitao Fu, and Zuocheng Xing, "Accelerating embarrassingly parallel algorithm on Intel MIC," In: IEEE International Conference on Progress in Informatics and Computing, 2014, pp. 213-218.

[14] Yan Li, Wanfeng Dou, Kun Yang, and Shoushuai Miao, "Optimized Data I/O Strategy of the Algorithm of Parallel Digital Terrain Analysis," In: 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, 2014, pp. 34-37.

[15] Jiaxin You, Hongda Kezhang, Huimin Liang, and Bin Xiao, "Research on parallel algorithms for calculating static characteristics of electromagnetic relay," In: IEEE 11th Conference on Industrial Electronics and Applications (ICIEA), 2016, pp. 1421-1425.

[16] Valma Prifti, Redis Bala, Igli Tafa, Denis Saatciu, and Julian Fejzaj, "The time profit obtained by parallelization of quicksort algorithm used for numerical sorting," In: Science and Information Conference (SAI), 2015, pp. 897-901.

[17] Rajashri Awari, "Parallelization of shortest path algorithm using OpenMP and MPI," In: International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2017, pp. 304-309.

[18] V.N. Aleeva, "Analysis of parallel numerical algorithms," Preprint No. 590. Novosibirsk, Computing Center of the Siberian Branch of the Academy of Sciences of the USSR, 1985. (in Pussian)

[19] Yu. L. Ershov, E. A. Palyutin, Mathematical Logic, Nauka, Moscow, 1987, 336 p. (in Russian)

[20] A.R. Bagautdinov, "Development of methods to investigation the parallelism of algorithms based on the conception of the *Q*-determinant and their software implementation," Graduate qualification work of master in direction "Fundamental informatics and information technology": 02.04.02. South Ural State University, Chelyabinsk, 2017. (http://omega.sp.susu.ru/publications/masterthesis/17-Bagautdinov.pdf)

[21] W.F. McColl, "General Purpose Parallel Computing," In: Lectures on Parallel Computation. In: Cambridge International Series on Parallel Computation, USA: Cambridge University Press, 1993, pp. 337-91.

[22] L.G. Valiant, "A bridging model for parallel computation," In: Communications of the ACM, vol. 33, № 8, 1990, pp. 103-111.

[23] J.Y.-T. Leung, H. Zhao "Scheduling problems in master-slave model," Annals of Operations Research, vol. 159, № 1, 2008, pp. 215-231.

[24] W. Gropp, E. Lusk, A. Skjellum. Using MPI: portable parallel programming with the messagepassing interface. Second Edi. MIT Press, 1999.

[25] V.N. Aleeva, N.A. Ivanov, "Investigation of the internal parallelism of numerical algorithms," In: Parallel Computational Technologies (PCT'2018): Proceedings of the International Scientific Conference (Rostov-on-Don, Russia, April, 2-6, 2018). Publishing of the South Ural State University, Chelyabinsk, 2018, pp. 224-234.

(http://omega.sp.susu.ru/pavt2018/short/004.pdf)

[26] N.V. Val'kevich, "*Q*-effective implementation of the algorithm for matrix multiplication on a supercomputer "Tornado SUSU"," Graduate qualification work of bachelor in direction "Fundamental informatics and information technology": 02.03.02. South Ural State University, Chelyabinsk, 2017.
(http://omega.sp.susu.ru/publications/bachelorthesis/17-Valkevich.pdf)

[27] D.E. Tarasov, "*Q*-effective co-design of realization of the Gauss–Jordan method on the supercomputer "Tornado SUSU"," Graduate qualification work of master in direction "Fundamental informatics and information technology": 02.04.02. South Ural State University, Chelyabinsk, 2017. (http://omega.sp.susu.ru/publications/masterthesis/17-Tarasov.pdf)

[28] Yu.S. Lapteva, "*Q*-effective implementation of the Jacobi method for solving SLAE on the supercomputer "Tornado SUSU"," Graduate qualification work of bachelor in direction "Fundamental informatics and information technology": 02.03.02. South Ural State University, Chelyabinsk, 2017.
(http://omega.sp.susu.ru/publications/bachelorthesis/17-Lapteva.pdf)

[29] L.A. Bazhenova, "Application of the method of designing a *Q*-effective program for solving the system of grid equations," Graduate qualification work of bachelor in direction "Fundamental informatics and information technology": 02.03.02. South Ural State University, Chelyabinsk, 2018.
(http://omega.sp.susu.ru/publications/bachelorthesis/18-Bazhenova.pdf)

[30] A.D. Necheporenko, "Development of a *Q*-effective program for solving SLAE by the Gauss–Seidel method," Graduate qualification work of bachelor in direction "Fundamental informatics and information technology": 02.03.02. South Ural State University, Chelyabinsk, 2018. (http://omega.sp.susu.ru/publications/bachelorthesis/18-Necheporenko.pdf)