DOI: 10.14529/cmse230204

SOLVING GRID EQUATIONS USING THE ALTERNATING-TRIANGULAR METHOD ON A GRAPHICS ACCELERATOR[*]

© 2023 A.I. Sukhinov[1], V.N. Litvinov[1,2], A.E. Chistyakov[1], A.V. Nikitina[1,3], N.N. Gracheva[1,2], N.B. Rudenko[1,2]

[1]*Don State Technical University (Gagarin Sq. 1, Rostov-on-Don, 344003 Russia),*

[2]*Azov-Black Sea Engineering Institute of Don State Agrarian University (Lenina 21, Zernograd, 347740 Russia),*

[3]*Southern Federal University (Bolshaya Sadovaya 105/42, Rostov-on-Don, 344006 Russia)*

E-mail: sukhinov@gmail.com, litvinovvn@rambler.ru, cheese_05@mail.ru, nikitina.vm@gmail.com, 79286051374@yandex.ru, nelli-rud@yandex.ru

Received: 15.03.2023

The paper describes a parallel-pipeline implementation of solving grid equations using the modified alternating-triangular iterative method (MATM), obtained by numerically solving the equations of mathematical physics. The greatest computational costs at using this method are on the stages of solving a system of linear algebraic equations (SLAE) with lower triangular and upper non-triangular matrices. An algorithm for solving the SLAE with a lower triangular matrix on a graphics accelerator using NVIDIA CUDA technology is presented. To implement the parallel-pipeline method, a three-dimensional decomposition of the computational domain was used. It is divided into blocks along the y coordinate, the number of which corresponds to the number of GPU streaming multiprocessors involved in the calculations. In turn, the blocks are divided into fragments according to two spatial coordinates — x and z. The presented graph model describes the relationship between adjacent fragments of the computational grid and the pipeline calculation process. Based on the results of computational experiments, a regression model was obtained that describes the dependence of the time for calculation one MATM step on the GPU, the acceleration and efficiency for SLAE solution with a lower triangular matrix by the parallel-pipeline method on the GPU were calculated using the different number of streaming multiprocessors.

Keywords: mathematical modeling, parallel algorithm, graphics accelerator.

FOR CITATION

Sukhinov A.I., Litvinov V.N., Chistyakov A.E., Nikitina A.V., Gracheva N.N., Rudenko N.B. Solving Grid Equations Using the Alternating-triangular Method on a Graphics Accelerator. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2023. Vol. 12, no. 2. P. 78–92. DOI: 10.14529/cmse230204.

Introduction

Modeling of any physical processes occurring in the environment and their mathematical description leads to the necessity to solve differential equations in private derivatives. To study dynamic processes in hydrophysics and hydrodynamics, the diffusion-convection-reaction equation is used [1, 2].

The solution to the equations of mathematical physics is based on the approximation of equations of end-and-character schemes. In the case of the use of an implicit, non-exposure scheme, the solution of equation is reduced to solving the system of linear algebraic equations of a large dimension. The largest computational costs in solving differential equations are at solution to the indicated SLAE, therefore, various iterative methods and algorithms are developed and

[*]The paper is recommended for publication by the Program Committee of the International Scientific Conference “Parallel Computational Technologies (PCT) 2023”.

applied [3–6]. One of the effective iterative methods for solving SLAE is the alternating-triangular method. This method is applicable to the high-dimensional SLAE with self-adjoint and non-self-adjoint operators, and has a high convergence rate. The iterative alternating-triangular method is used to solve ill-conditioned SLAE.

The dimensions of resulting SLAE are such that they require large computing performance. Problems of computing performance lack are solved in several ways, in particular, using graphic accelerators (GPUs) for calculation the resources and computing processes [3, 7–12].

Russian scientists applied computational grid decomposition in solving a three-dimensional boundary value problem. The parallelization algorithm was implemented in a heterogeneous computing environment. Due to the use of graphic accelerations, the calculation time was reduced in 60 times, compared to the calculations on the CPU [7]. The three-phase filtration problem was also solved in heterogeneous computing environment using the decomposition of computational domain. The parallel algorithm is performed on C++ with using the CUDA and MPI technology. Due to the use of GPU for calculating the specified problem, the computational costs reduces in several tens of times [3]. Scientists of China University of Petroleum propose a parallel algorithm for modeling hardening in two dimensions based on the domain decomposition. This algorithm was implemented on GPU, which significantly reduces the calculation time [8]. Researchers of Altai State University have developed a parallel algorithm for a numerical solution to the problem of electromagnetic impulse spreading in two-dimensional rectangular field. The algorithm was implemented on the basis of CUDA technology. An analysis of the performance of the developed algorithm showed that the GPU performance is several times higher than the CPU performance at solving the problem [9]. The effectiveness of the use of graphic accelerators for numerical modeling of the tasks of applied hydrodynamics has been proved. The calculation rate when solving the problem of numerical modeling of the hydrodynamic characteristics of mushroom screws was increased 1.4–3 times [10]. The exact calculation of heat transfer coefficients requires powerful computing resources that are not available. The parallelization algorithm for such calculations with implementation in heterogeneous computing environment increases productivity, compared to the CPU, more than ten times [12].

In this paper, the decomposition method of three-dimensional calculated circle to implement the parallel algorithm on the GPU is proposed. The developed parallel-conveyor method for SLAE solving allows to effectively use the GPU resources and reduce the calculation time.

1. Method of Solving Grid Equations

Solving the equations of mathematical physics can be reduced to solving a system of linear algebraic equations of the form:

$$Ax = f,\ A : H \to H, \tag{1}$$

where A is the linear, positive definite operator.

For the grid equation (1), the iterative methods are used, which in canonical form can be represented by the equation [1, 2]:

$$B\frac{x^{m+1} - x^m}{\tau_{m+1}} + Ax^m = f,\ B : H \to H, \tag{2}$$

where m is the iteration number; $\tau_{m+1} > 0$ is the iteration parameter; B is the preconditioner.

The resulting grid equations will be solved using the modified alternating-triangular method of variational type. The preconditioner is formed as follows:

$$B = (D + \omega R_1)\, D^{-1} (D + \omega R_2)\,,\ D = D^* > 0,\ \omega > 0, \tag{3}$$

where D is the diagonal operator; R_1, R_2 are the lower- and upper-triangular operators, respectively.

The calculation algorithm of grid equations by the modified alternating-triangular method of the variational type is written in the form:

$$r^m = Ax^m - f,$$

$$(D + \omega R_1)y^m = r^m,\ (D + \omega R_2)w^m = Dy^m,$$

$$\tilde{\omega}_m = \sqrt{\frac{(Dw^m, w^m)}{(D^{-1}R_2 w^m, R_2 w^m)}},$$

$$s_m^2 = 1 - \frac{(A_0 w^m, w^m)^2}{(B^{-1}A_0 w^m)\,(Bw^m, w^m)},\ k_m^2 = \frac{\left(B^{-1}A_1 w^m, A_1 w^m\right)}{(B^{-1}A_0 w^m, A_0 w^m)}, \tag{4}$$

$$\theta_m = \frac{1 - \sqrt{\frac{s_m^2 k_m^2}{(1+k_m^2)}}}{1 + k_m^2\,(1 - s_m^2)},\ \tau_{m+1} = \theta_m \frac{(A_0 w^m, w^m)}{(B^{-1}A_0 w^m, A_0 w^m)},$$

$$x^{m+1} = x^m - \tau_{m+1} w^m,\ \omega_{m+1} = \tilde{\omega}_m,$$

where r^m is the residual vector; w^m is the correction vector; the parameter s_m describes the convergence rate of the method; k_m describes the ratio of the norm of the skew-symmetric operator part to the norm of the symmetric part.

The most labors part of the algorithm is the calculation of the correction vector w^m and reduced to the solution of two SLAE with the lower-triangular and upper-triangular matrix:

$$(D + \omega R_1)y^m = r^m,\ (D + \omega R_2)w^m = Dy^m.$$

The algorithm fragment for solving SLAE with the lower-triangular matrix is given in Algorithm 1. The residual vector is calculated in $14N$ arithmetic operations. The total number of

Algorithm 1 matm(IN: $n_1, n_2, n_3, a_0, a_2, a_4, a_6, \omega$; IN/OUT: r)

```
1: for $k \in [1; n_3 - 2]$ do
2:     for $i \in [1; n_1 - 2]$ do
3:         for $j \in [1; n_2 - 2]$ do
4:             $p_0 \leftarrow i + n_1 \cdot j + n_1 \cdot n_2 \cdot k$
5:             if $a_0[p_0] > 0$ then
6:                 $p_2 \leftarrow p_0 - 1;\ p_4 \leftarrow p_0 - n_1;\ p_6 \leftarrow p_0 - n_1 \cdot n_2$
7:                 $r[p_0] \leftarrow (\omega \cdot (a_2[p_0] \cdot r[p_2] + a_4[p_0] \cdot r[p_4] + a_6[p_0] \cdot r[p_6]) + r[p_0])/((0.5 \cdot \omega +$
                   $1) \cdot a_0[p_0])$
```

arithmetic operations required to solve the SLAE with the seven-diagonal matrix using MATM in the case of known iterative parameters τ_{m+1}, ω_{m+1} is $35N$, where $N = n_1 n_2 n_3$ is SLAE dimension.

2. Decomposition Model of Computational Domain

Let Q be the set of technical characteristics of the video adapter, then we will present the characteristics of the video adapter in the form of a tuple.

$$Q = \left\langle q^1, q^2 \right\rangle, \tag{5}$$

where q^1 is the amount of video memory of the video adapter, GB; q^2 is the number of streaming multiprocessors.

If S is a set of program threads involved in the computational process, then

$$S = \left\{ s_k, k = \overline{1, q^2} \right\}, \tag{6}$$

where s_k is a CUDA streaming block that implements the calculation process on GPU streaming multiprocessor with index k.

Let us take the computational domain with the following parameters: l_x is the characteristic size on the axis Ox, l_y — on the axis Oy, l_z — on the axis Oz.

Let us compare the specified area with a uniform computational grid of the following type:

$$W = \{x_i = ih_x, y_j = jh_y, z_k = kh_z; \\ i = \overline{0, n_x - 1}, j = \overline{0, n_y - 1}, k = \overline{0, n_z - 1}; \\ (n_x - 1)h_x = l_x, (n_y - 1)h_y = l_y, (n_z - 1)h_z = l_z\}, \tag{7}$$

where h_x, h_y, h_z are the steps of computational grid at the corresponding spatial directions; n_x, n_y, n_z are the number of grid nodes at the corresponding spatial directions.

Then the set of nodes of the computational grid can be represented as

$$G = \left\{ g_{i,j,k}, i = \overline{0, n_x - 1}, j = \overline{0, n_y - 1}, k = \overline{0, n_z - 1} \right\}, \\ g_{i,j,k} = \left\langle x_i, y_j, z_k \right\rangle, \tag{8}$$

where $g_{i,j,k}$ is the grid node.

The number of nodes of the computational grid N_G is calculated by the formula:

$$N_G = n_x \cdot n_y \cdot n_z, \tag{9}$$

Under the block of the computational grid $G^{k_1} \subset G$ (further – the block) we will understand the sub-set of nodes of the computational grid G.

$$G = \bigcup_{k_1 \in K_{k_1}} G^{k_1} = \{g^{k_1} | \exists k_1 \in K_{k_1}, g^{k_1} \in G^{k_1}\}, \quad \bigcap_{k_1 \in K_{k_1}} G^{k_1} = \varnothing, \tag{10}$$

where $K_{k_1} = \{1, ..., N_{k_1}\}$ is the set of block indices G^{k_1} of the computational grid G ; N_{k_1} is the number of blocks G_{k_1}, $N_{k_1} = d^2$; $K_{k_1}, N_{k_1} \subset N$; N is the set of natural numbers; k_1 is the block index G^{k_1}.

Since $G^{k_1} \subset G$, then

$$G^{k_1} = \left\{ g^{k_1}_{i,\tilde{j},k}, i = \overline{0, n_x - 1}, j = \overline{0, n_y^{k_1} - 1}, k = \overline{0, n_z - 1} \right\}, \tag{11}$$

where $g^{k_1}_{i,\widetilde{j},k}$ block node k_1; the $\sim$ sign denotes belonging to the block; $\widetilde{j}$ is the block node index k_1 at coordinate y; $n^{k_1}_y$ is the number of nodes in block k_1 at coordinate y.

$$g^{k_1}_{i,\widetilde{j},k} = \langle x_i, y_j, z_k \rangle\,,\ x_i = ih_x, y_j = \left(\sum_{b=1}^{k_1-1} n^b_y + \widetilde{j} \right) \cdot h_y, z_k = kh_z, \tag{12}$$

where n^b_y is the number of nodes at coordinate y of the b-th block.

Under the fragment of the computational grid G^{k_1,k_2} (further — the fragment) we will understand a subset of the nodes of the computational grid of block G^{k_1}.

$$G^{k_1} = \bigcup_{k_2 \in K_{k_1,k_2}} G^{k_1,k_2} = \{g^{k_1,k_2} | \exists k_2 \in K_{k_1,k_2}, g^{k_1,k_2} \in G^{k_1,k_2}\},$$

$$\bigcap_{k_2 \in K_{k_1,k_2}} G^{k_1,k_2} = \oslash, \tag{13}$$

where $K_{k_1,k_2} = \{1, ...N_{k_1,k_2}\}$ is a plurality of fragment indexes G^{k_1,k_2} of block G^{k_1}; N_{k_1,k_2} is the number of fragments G^{k_1,k_2}; $K_{k_1,k_2}, N_{k_1,k_2} \subset N$; k_2 is the index of fragment G^{k_1,k_2} of block G^{k_1}.

Since $G^{k_1,k_2} \subset G^{k_1}$ then

$$G^{k_1,k_2} = \left\{ \breve{g}_{\breve{i},\widetilde{j},\breve{k}}, \breve{i} = \overline{0, \breve{n}_x - 1}, \widetilde{j} = \overline{0, \widetilde{n}_y - 1}, \breve{k} = \overline{0, \breve{n}_z - 1} \right\}, \tag{14}$$

where $\breve{g}_{\breve{i},\widetilde{j},\breve{k}}$ is the fragment node; the sign $\smile$ denotes belonging to a fragment; $\breve{i}, \breve{k}$ are indexes of fragment node by coordinates x, z; $\breve{n}_x, \breve{n}_z$ is the number of nodes of the computational grid in the fragment along the coordinates x, z.

Each index k_2 of fragment G^{k_1,k_2} is associated with a tuple of indices $\langle k_3, k_4 \rangle$, designed to store fragment coordinates in plane xOz, where k_3 is the fragment index at coordinate x, k_4 is the fragment index at coordinate z.

$$k_2 = k_3 + K_{k_3} \cdot k_4, \tag{15}$$

where k_3 is the fragment index along the x coordinate; k_4 is the fragment index along the z coordinate; K_{k_3} is the number of fragments along the Ox axis.

The number of fragments G^{k_1,k_2} block G^{k_1} is calculated by the formula

$$K_{k_2} = K_{k_3} \cdot K_{k_4}, \tag{16}$$

where K_{k_4} is the number of fragments at coordinate z.

$$\breve{g}_{\breve{i},\widetilde{j},\breve{k}} = \langle x_i, y_j, z_k \rangle$$

$$x_i = \left(\sum_{b=1}^{k_3-1} \breve{n}_b + \breve{i} \right) \cdot h_x, y_j = \widetilde{j} h_y, z_k = \left(\sum_{b=1}^{k_4-1} \breve{n}_b + \breve{k} \right) \cdot h_z, \tag{17}$$

where $\breve{n}_b$ is the number of nodes in the b-th fragment.

Let us introduce a set of comparisons of computational grid blocks with program currents M.

$$M = \left\{ m_{k_1} = \left\langle G^{k_1}, s_{k_1} \right\rangle, k_1 \in K_{k_1} \right\}, \tag{18}$$

where $s_{k_1} \in S$ — program flow, calculating block G^{k_1}.

For the domain decomposition, it is necessary to take into account the computing performance of device, involved in calculations. Performance refers to the number of nodes of the computational grid, calculated using a given algorithm, per unit of time.

To calculate the number of nodes along the coordinate y in the blocks of the computational grid processed by GPU streaming multiprocessors, we use the formulas

$$n_{yGT} = \left\lfloor \frac{n_y}{N_{k_1} - 1} \right\rfloor, n_{yGTL} = n_y - \sum_{b=1}^{N_{k_1}-1} n_{yGT}^b, \tag{19}$$

where n_{yGT} is the number of computational grid nodes along coordinate y in blocks processed by GPU streaming multiprocessors, except for the last block; n_{yGTL} is the number of nodes at coordinate y in the last block of the computational grid processed by GPU streaming multiprocessors.

The number of the computational grid fragments along the coordinate y is equal to

$$N_y^f = N_{k_1}. \tag{20}$$

Let the number of fragments be N_x^f and N_z^f at coordinates x and z, respectively. Then, the number of nodes of the computational grid along the coordinate x is calculated by the formulas:

$$n_x^f = \left\lfloor \frac{n_x}{N_x^f - 1} \right\rfloor, \ n_x^{fL} = n_x - n_x^f \cdot (N_x^f - 1), \tag{21}$$

where n_x^f is the number of nodes of the computational grid along the coordinate x in all fragments except the last one; n_x^{fL} — the number of nodes of the computational grid along the coordinate x in the last fragment.

Similarly, the number of nodes of the computational grid along the coordinate z is calculated

$$n_z^f = \left\lfloor \frac{n_z}{N_z^f - 1} \right\rfloor, \ n_z^{fL} = n_z - n_z^f \cdot (N_z^f - 1), \tag{22}$$

where n_z^f is the number of nodes of the computational grid along the coordinate z in all fragments except the last one; n_z^{fL} — the number of nodes of the computational grid along the coordinate z in the last fragment.

Let on M it is necessary to organize a parallel process for calculation some function F, and the calculations in each fragment G^{k_1,k_2} depend on the values in neighboring fragments, each of which has at least one of the indices at coordinates x, y and z one less than the current one.

To organize a parallel-pipelining method, let us introduce a set of tuples A that define correspondences a between program flows s_k, processing fragments G^{k_1,k_2}, and the numbers of steps of the parallel-pipelining method r.

$$\forall s_k \in S \ \exists a \in A : a = \left\langle s_k, G^{k_1,k_2}, r \right\rangle, \tag{23}$$

where $r = \overline{1, N_r}$ is the step number of the parallel-pipeline method, N_r is the number of steps of the parallel-pipeline method, calculated by the formula

$$N_r = N_x^f N_z^f + N_y^f - 1. \tag{24}$$

Full download of all calculators in the proposed parallel-pipeline method starts from step $r_{100START} = N_y^f$ and ends at the step $r_{100STOP} = N_x^f N_z^f$. In this case, the total number of steps with a full load of N_{rPAR} calculators will be

$$N_{rPAR} = r_{100STOP} - r_{100START} + 1 = N_x^f N_z^f - N_y^f + 1. \tag{25}$$

The calculation time of some function F by the parallel-pipeline method can be written in the form

$$T_M = \sum_{r=1}^{N_r} max(T_a), \tag{26}$$

where T_a is a vector of time values for fragment processing in parallel mode.

3. Parallel Implementation

The numerical implementation of the MATM for solving SLAE with the high dimension is based on the developed parallel algorithms that implement the pipeline computing process. The use of these algorithms allows to fully utilize all available streaming multiprocessors of graphics accelerator.

A class library was developed in C++ for describing the domain decomposition. The class library contains the following classes:

- Grid3D, describes the parameters of the computational grid (number of nodes n_x, n_y, n_z, and step sizes h_x, h_y, h_z in spatial coordinates) and contains an array of objects of the GridBlock3D class.
- GridBlock3D, describes the parameters of the computational grid block and contains an array of objects of the GridFragment3D class.
- GridFragment3D, describes the parameters of a computational grid fragment and contains data arrays.

The organization of calculations is performed by an algorithm that controls all available streaming multiprocessors of GPU (calculators). Each calculator performs calculations only for its own block of the computational domain. For this, the computational domain is divided into blocks that are assigned to individual calculators (Fig. 1). Next, each block is divided into fragments. Notations in Fig. 1: SM_1, SM_2, SM_3 are streaming multiprocessors of GPU.

A graph model was used to describe the relationships between adjacent fragments of the computational grid and the organization of the pipeline calculation process (Fig. 2). Each graph node is an object of a class GridFragment3D that describes a fragment of the computational domain. This class contains the following fields: the dimensions of the fragment along the Ox, Oy, and Oz axes; the index of the zero node of the fragment in the global computational domain; pointers to adjacent fragments of the computational grid; pointers to objects that describe the parameters of calculators. The computational process is a graph traversal from the root node with parallel launch of calculators that process the graph nodes in accordance with the value of the calculation step counter r.

An algorithm and its program implementation in the CUDA C language are developed to improve the calculation efficiency of the computational grid fragments assigned to the graphics accelerator [13–17].

We present an algorithm for searching the solution for the system of equations with the lower-triangular matrix (straight line) on CUDA C.

The input parameters of the algorithm are the vectors of the coefficients of grid equations a_0, a_2, a_4, a_6 and the constant ω. The output parameter is the vector of the water flow velocity v. Before running the algorithm, it is necessary to programmatically set the dimensions of the

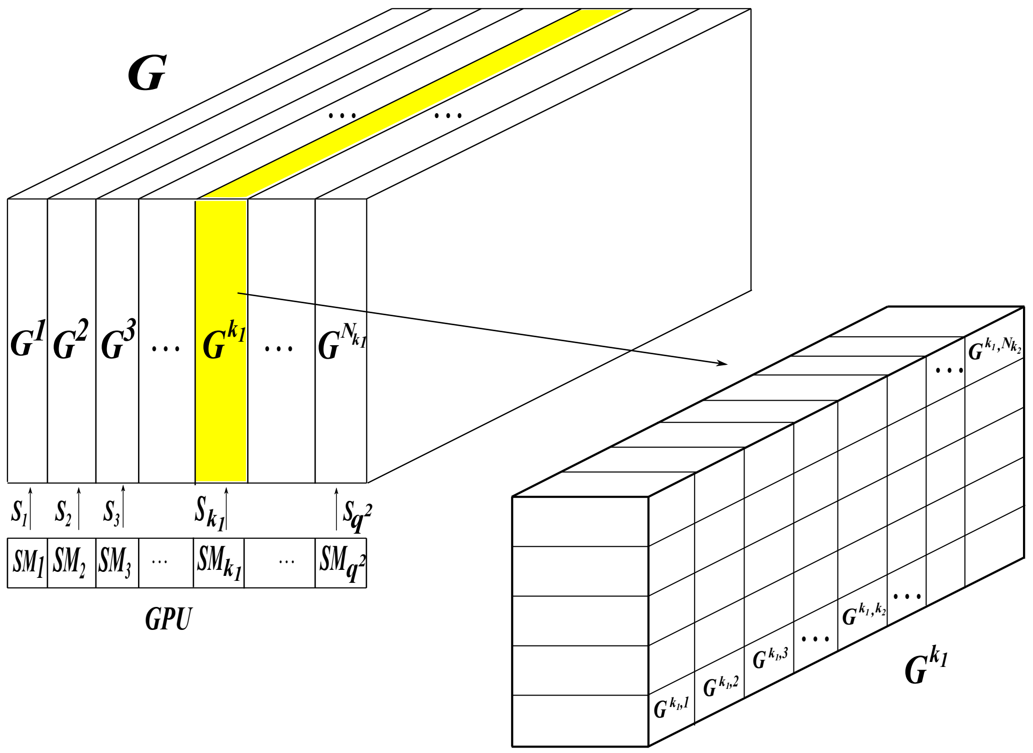
G
G^1 G^2 G^3 … G^{k_1} … $G^{N_{k_1}}$
S_1 S_2 S_3 S_{k_1} S_{q^2}
SM_1 SM_2 SM_3 … SM_{k_1} … SM_{q^2}
GPU
$G^{k_1,N_{k_2}}$
$G^{k_1,1}$ $G^{k_1,2}$ $G^{k_1,3}$ … G^{k_1,k_2} …
G^{k_1}

Fig. 1. Decomposition of the third-dimensional computational domain

CUDA computing block *blockDim.x*, *blockDim.z* according to the spatial coordinates x, z, respectively. The CUDA framework runs this algorithm for each thread, and the variable values *threadIdx.x*, *threadIdx.z*, *blockIdx.x*, *blockIdx.z* are automatically initialized by the indexes of the corresponding threads and blocks. Global thread indexes are calculated in rows 1 and 2. The row index i and the layer index k that the current thread processes are calculated in row 3. A variable j is initialized that represents a counter by coordinate y. The calculation pipeline is organized as a loop in line 4. The indexes of the central node of the grid pattern p_0 and the surrounding nodes p_2, p_4, p_6 are calculated in line 8. The two-dimensional array *cache* is located in the GPU shared memory and designed to store the calculation results on the current layer by the coordinate y. This allows us to reduce the number of reads from slow global memory and accelerate the calculation process by up to 30 %.

The performed researches show a significant dependence of the algorithm implementation time for calculation the preconditioner on the ratio of threads in spatial coordinates. A series of experiments is preperformed to calculate the performance of calculators, which is the 95th percentile of the calculation time in terms of 1000 nodes of the computational grid.

GeForce GTX 1650 video adapter was used in experimental researches. The GeForce GTX 1650 video adapter has 4 GB of video memory, core and memory clock frequency of 1485 MHz and 1665 MHz, and a video memory bus bit rate of 128 bits. The computing part consists of 14 streaming multiprocessors (SM).

The purpose of the experiment is to determine the distribution of flows along the Ox and Oz axes of the computational grid at different values of its nodes along the Oy axis so that the implementation time on the GPU of one MATM step is minimal. Two values are taken as factors: $k = X/Z$ is the ratio of the number of threads on the Ox (X) axis to the number of threads on the Oz (Z) axis; Y is the number of threads on the axis Oy. Values of the objective

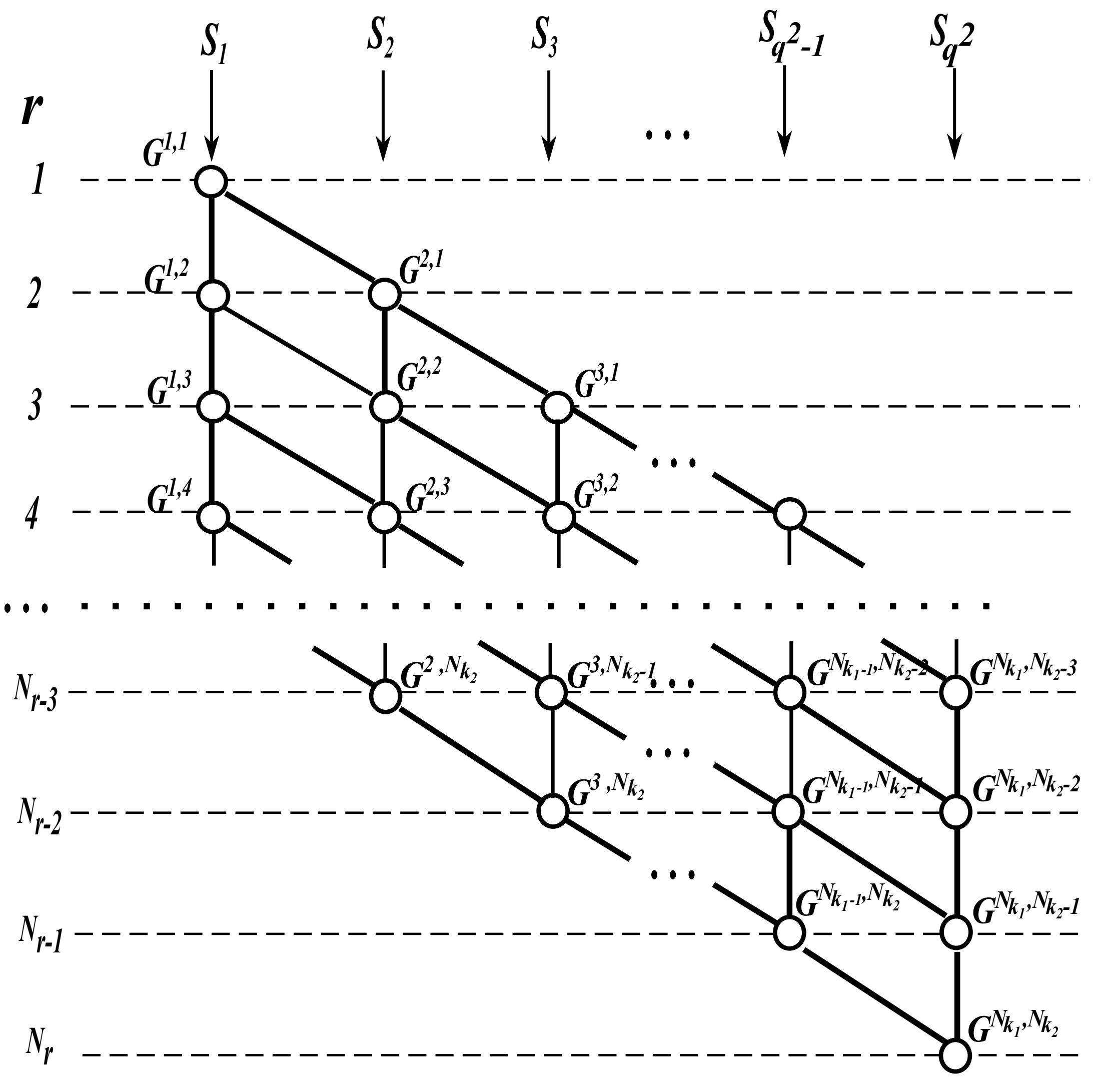

Fig. 2. A graph model that describes the relationships between adjacent fragments of the computational grid and the process of pipeline calculation

function: T_{GPU} is the calculation time of one MATM step on the GPU in terms of 1000 nodes of the computational grid, ms.

The regression equation was obtained as a result of the experimental data processing (Fig. 3):

$$T_{GPU} = a - b \cdot Y - c \cdot ln(k) - d \cdot ln(Y), \tag{27}$$

where T_{GPU} is the implementation time of one MATM step on the GPU in terms of 1000 nodes of the computational grid, ms. The determination coefficient was 0.86; $a = 0.026$; $b = 2 \cdot 10^{-7}$; $c = 16 \cdot 10^{-5}$; $d = 77 \cdot 10^{-5}$.

To evaluate the effectiveness of the parallel-pipeline method for solving SLAE with a lower triangular matrix, a numerical experiment was performed. The dimensions of the three-dimensional uniform computational grid along the spatial coordinates x, y, and z were respectively set equal to 640, 224, and 448, respectively. The amount of video memory was 3.8 GB. In the course of experimental researches, we changed the number of streaming multiprocessors N_{k_1} involved in the calculations and fixed the computation time T_M.

Algorithm 2 matmKernel(IN: $a_0, a_2, a_4, a_6, \omega$ IN/OUT: v;)

```
1: $thX \leftarrow blockDim.x \cdot blockIdx.x + threadIdx.x$
2: $thZ \leftarrow blockDim.z \cdot blockIdx.z + threadIdx.z$
3: $i \leftarrow thX + 1$; $j \leftarrow 1$; $k \leftarrow thZ + 1$
4: for $s \in [3; n_1 + n_2 + n_3 - 3]$ do
5:     if $(i + j + k = s) \wedge (s < i + n_2 + k)$ then
6:         $p_0 \leftarrow i + (blockDim.x + 1) \cdot j + n_1 \cdot n_2 \cdot k$
7:         if $a0[p0] > 0$ then
8:             $p_2 \leftarrow p_0 - 1; p_4 \leftarrow p_0 - n_1; p_6 \leftarrow p_0 - n_1 \cdot n_2$
9:             $vp4 \leftarrow 0$
10:            if $(s > 3 + thX + thZ)$ then
11:                $vp4 \leftarrow cache[thX][thZ]$
12:            else
13:                $vp4 \leftarrow v[p_4]$
14:            $vp2 \leftarrow 0$
15:            if $(thX \neq 0) \wedge (s > 3 + thX + thZ)$ then
16:                $vp2 \leftarrow cache[thX - 1][thZ]$
17:            else
18:                $vp2 \leftarrow v[p_2]$
19:            $vp6 \leftarrow 0;$
20:            if $(thZ \neq 0) \wedge (s > 3 + thX + thZ)$ then
21:                $vp6 \leftarrow cache[thX][thZ - 1]$
22:            else
23:                $vp6 \leftarrow v[p_6]$
24:            $vp0 \leftarrow (\omega \cdot (a2[p_0] \cdot vp2 + a4[p_0] \cdot vp4 + a6[p_0] \cdot vp6) + v[p_0])/((0.5 \cdot \omega + 1) \cdot a_0[p_0])$
25:            $cache[thX][thZ] \leftarrow vp0$
26:            $v[p_0] \leftarrow vp0$
27:        $j \leftarrow j + 1$
```

For each experiment, the computational grid was divided into three-dimensional blocks and fragments. In this case, the number of blocks was set equal to the number of streaming multiprocessors. The number of fragments in blocks along spatial coordinates x, y, and z was set equal to 4, 1, and 7, respectively. The sizes of fragments along spatial coordinates x ($n_x^{k_1}$) and z ($n_z^{k_1}$) were set equal to 160 and 64, respectively. The acceleration $S_p = T_M(1)/T_M(i)$ and efficiency $E_p = S_p(i)/N_{k_1}(i)$ were calculated from the experimental data. The results of numerical experiments are shown in Tab. 1.

Conclusions

To solve grid equations using the MPTM method on the graphics accelerator, the decomposition model of computational domain has been developed. The computational domain is divided into blocks along the spatial coordinate y, and then the blocks are divided into fragments along the spatial coordinates x and z. This model allows each GPU streaming multiprocessor to map a computational domain block and organize a parallel-pipelined computational process. The graph model was proposed that describes the relationship between adjacent fragments of

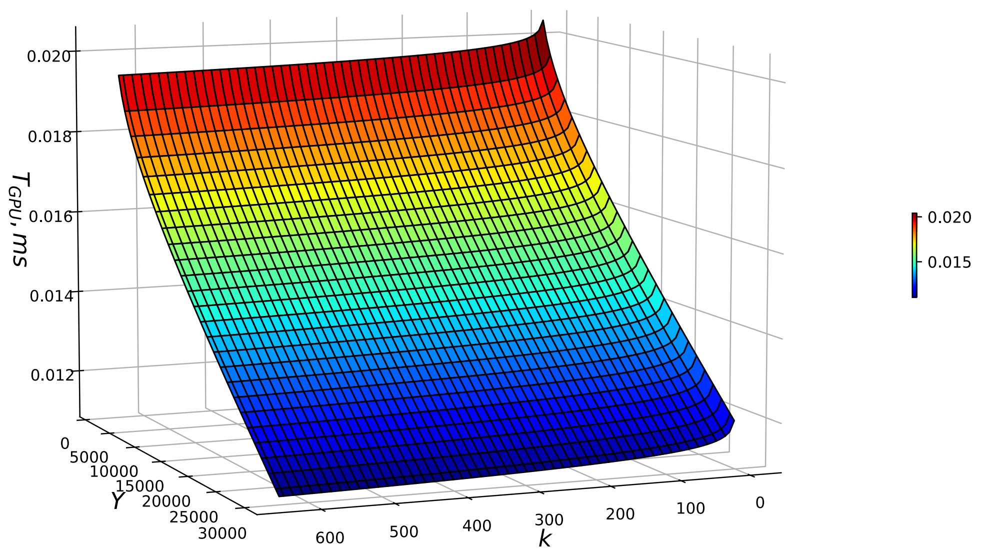

Fig. 3. Surface of the response function $T_{GPU} = f(k, Y)$

Table 1. Results of SLAE calculations with a lower triangular matrix by a parallel-pipeline method on GPU

N_{k_1}	$n_y^{k_1}$	N_r	T_M	S_p	E_p
1	224	–	580	1.0	1.00
2	112	29	304	1.9	0.95
7	32	34	102	5.7	0.81
14	16	41	61	9.5	0.68

the computational grid and the process of conveyor calculation. The algorithm for solving the system of equations with a lower triangular matrix in the CUDA C language was described.

As a result of the experiment, a regression model was obtained: it describes the dependence of the time for calculation one step of the MATM on the GPU. According to the regression model, at $k < 10$ and $Y < 1000$, the calculation velocity slows down, which is explained by the inefficient use of the distributed memory of the graphics accelerator.

The results of calculations of SLAE with the lower triangular matrix by the parallel-pipeline method on the GPU with using the different number of streaming multiprocessors are presented. At $N_{k_1} = 14$, the acceleration S_p was 9.5, and the efficiency E_p was 0.668.

The reported study was funded by the Russian Science Foundation (project No. 21-71-20050).

References

1. Sukhinov A.I., Atayan A.M., Belova Y.V., *et al.* Data processing of field measurements of expedition research for mathematical modeling of hydrodynamic processes in the Azov Sea. Computational Continuum Mechanics. 2020. Vol. 13, no. 2. P. 161–174. DOI: 10.7242/1999-6691/2020.13.2.13.

2. Sukhinov A.I., Litvinov V.N., Chistyakov A.E., *et al.* Computational aspects of solving grid

equations in heterogeneous computing systems. Parallel Computing Technologies. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. P. 166–177. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-86359-3_13.

3. Lyupa A., Morozov D., Trapeznikova M., *et al.* Three-phase filtration modeling by explicit methods on hybrid computer systems. Mathematical Models and Computer Simulations. 2014. Vol. 6. P. 551–559. DOI: 10.1134/S2070048214060088.
4. Mat Ali N.A., Rahman R., Sulaiman J., Ghazali K. Solutions of reaction-diffusion equations using similarity reduction and HSSOR iteration. Indonesian Journal of Electrical Engineering and Computer Science. 2019. Vol. 16, no. 3. P. 1430–1438. DOI: 10.11591/ijeecs.v16.i3.pp1430-1438.
5. Kittisopaporn A., Chansangiam P. The steepest descent of gradient-based iterative method for solving rectangular linear systems with an application to Poisson's equation. Advances in Difference Equations. 2020. Vol. 2020. Article number 259. DOI: 10.1186/s13662-020-02715-9.
6. Yifen K., Ma C. Adaptive parameter based matrix splitting iteration method for the large and sparse linear systems. Computers & Mathematics with Applications. 2022. Vol. 122. P. 19–27. DOI: 10.1016/j.camwa.2022.07.010.
7. Klimonov I.A., Korneev V.D., Sveshnikov V.M. Parallelization technologies for solving three-dimensional boundary value problems on quasi-structured grids using the CPU+GPU hybrid computing environment. Numerical Methods and Programming. 2016. Vol. 17, no. 1. P. 65–71. DOI: 10.26089/NumMet.v17r107.
8. Ding P., Liu Z. Accelerating phase-field modeling of solidification with a parallel adaptive computational domain approach. International Communications in Heat and Mass Transfer. 2020. Vol. 111. P. 104452. DOI: 10.1016/j.icheatmasstransfer.2019.104452.
9. Molostov I., Scherbinin V. Application of NVIDIA CUDA Technology for Numerical Simulation of Electromagnetic Pulses Propagation. Izvestiya of Altai State University. 2015. Vol. 1, no. 1/1(85). DOI: 10.14258/izvasu(2015)1.1-06.
10. Krasnopolsky B., Medvedev A., Chulyunin A. On application of GPUs for modelling of hydrodynamic characteristics of screw marine propellers in OpenFOAM package. Proceedings of the Institute for System Programming of RAS. 2014. Vol. 26, no. 5. P. 155–172. DOI: 10.15514/ISPRAS-2014-26(5)-8.
11. Egorov M., Egorov S., Egorov D. Using graphics accelerator to improve computing performance in the numerical modeling of complex technical systems functioning. Perm National Research Polytechnic University Aerospace Engineering Bulletin. 2015. No. 40. P. 81–91. DOI: 10.15593/2224-9982/2015.40.05.
12. Szenasi S. Solving the inverse heat conduction problem using NVLink capable Power architecture. PeerJ Computer Science. 2017. Vol. 3. P. 138. DOI: 10.7717/peerj-cs.138.
13. Zheng L., Gerya T., Knepley M., *et al.* GPU Implementation of Multigrid Solver for Stokes Equation with Strongly Variable Viscosity. GPU Solutions to Multi-scale Problems in Science and Engineering. Springer, 2013. P. 321–333. Lecture Notes in Earth System Sciences. DOI: 10.1007/978-3-642-16405-7_21.
14. Konovalov A. The steepest descent method with an adaptive alternating-triangular preconditioner. Differential Equations. 2004. Vol. 40. P. 1018–1028.

15. Sukhinov A.I., Chistyakov A.E., Litvinov V.N., *et al.* Computational Aspects of Mathematical Modeling of the Shallow Water Hydrobiological Processes. Numerical methods and programming. 2020. Vol. 21, no. 4. P. 452–469. DOI: 10.26089/NumMet.v21r436.

16. Samarskii A.A., Vabishchevich P.N. Numerical methods for solving convection-diffusion problems. Moscow: URSS, 2009. (in Russian).

17. Browning J.B., Sutherland B. C++20 Recipes. A Problem-Solution Approach. Berkeley, CA: Apress, 2020. 630 p.

УДК 519.6 **DOI: 10.14529/cmse230204**

РЕШЕНИЕ СЕТОЧНЫХ УРАВНЕНИЙ ПОПЕРЕМЕННО-ТРЕУГОЛЬНЫМ МЕТОДОМ НА ГРАФИЧЕСКОМ УСКОРИТЕЛЕ

© 2023 А.И. Сухинов[1], В.Н. Литвинов[1,2], Ф.Е. Чистяков[1], А.В. Никитина[1,3], Н.Н. Грачева[1,2], Н.Б. Руденко[1,2]

[1]*Донской государственный технический университет (344003 Ростов-на-Дону, пл. Гагарина, д. 1),*

[2]*Азово-Черноморский инженерный институт ФГБОУ ВО Донской ГАУ (347740 Зерноград, ул. Ленина, д. 21),*

[3]*Южный федеральный университет (344006 Ростов-на-Дону, ул. Большая Садовая, д. 105/42)*

E-mail: sukhinov@gmail.com, litvinovvn@rambler.ru, cheese_05@mail.ru, nikitina.vm@gmail.com, 79286051374@yandex.ru, nelli-rud@yandex.ru

Поступила в редакцию: 15.03.2023

В статье описана параллельно-конвейерная реализация решения сеточных уравнений модифицированным попеременно-треугольным итерационным методом (МПТМ), получаемых при численном решении уравнений математической физики. Наибольшие вычислительные затраты при использовании указанного метода приходятся на этапы решения системы линейных алгебраических уравнений (СЛАУ) с нижнетреугольной и верхнетреугольной матрицами. Представлен алгоритм решения СЛАУ с нижнетреугольной матрицей на графическом ускорителе с использованием технологии NVIDIA CUDA. Для реализации параллельно-конвейерного метода использовалась трехмерная декомпозиция расчетной области. Она делится по координате y на блоки, количество которых соответствует количеству потоковых мультипроцессоров GPU, задействованных в вычислениях. В свою очередь, блоки разделяются на фрагменты по двум пространственным координатам — x и z. Представленная графовая модель описывает взаимосвязь между соседними фрагментами расчетной сетки и процессом конвейерного расчета. По результатам проведенных вычислительных экспериментов получена регрессионная модель, описывающая зависимость времени расчета одного шага МПТМ на GPU, вычислены ускорение и эффективность расчетов СЛАУ с нижнетреугольной матрицей параллельно-конвейерным методом на GPU при задействовании различного количества потоковых мультипроцессоров.

Ключевые слова: математическое моделирование, параллельный алгоритм, графический ускоритель.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Sukhinov A.I., Litvinov V.N., Chistyakov A.E., Nikitina A.V., Gracheva N.N., Rudenko N.B. Solving Grid Equations Using the Alternating-triangular Method on a Graphics Accelerator // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 2. С. 78–92. DOI: 10.14529/cmse230204.

Литература

1. Sukhinov A.I., Atayan A.M., Belova Y.V., *et al.* Data processing of field measurements of expedition research for mathematical modeling of hydrodynamic processes in the Azov Sea // Computational Continuum Mechanics. 2020. Vol. 13, no. 2. P. 161–174. DOI: 10.7242/1999-6691/2020.13.2.13.

2. Sukhinov A.I., Litvinov V.N., Chistyakov A.E., *et al.* Computational aspects of solving grid equations in heterogeneous computing systems // Parallel Computing Technologies. Vol. 12942 / ed. by V. Malyshkin. Springer, 2021. P. 166–177. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-86359-3_13.

3. Lyupa A., Morozov D., Trapeznikova M., *et al.* Three-phase filtration modeling by explicit methods on hybrid computer systems // Mathematical Models and Computer Simulations. 2014. Vol. 6. P. 551–559. DOI: 10.1134/S2070048214060088.

4. Mat Ali N.A., Rahman R., Sulaiman J., Ghazali K. Solutions of reaction-diffusion equations using similarity reduction and HSSOR iteration // Indonesian Journal of Electrical Engineering and Computer Science. 2019. Vol. 16, no. 3. P. 1430–1438. DOI: 10.11591/ijeecs.v16.i3.pp1430-1438.

5. Kittisopaporn A., Chansangiam P. The steepest descent of gradient-based iterative method for solving rectangular linear systems with an application to Poisson's equation // Advances in Difference Equations. 2020. Vol. 2020. Article number 259. DOI: 10.1186/s13662-020-02715-9.

6. Yifen K., Ma C. Adaptive parameter based matrix splitting iteration method for the large and sparse linear systems // Computers & Mathematics with Applications. 2022. Vol. 122. P. 19–27. DOI: 10.1016/j.camwa.2022.07.010.

7. Klimonov I.A., Korneev V.D., Sveshnikov V.M. Parallelization technologies for solving three-dimensional boundary value problems on quasi-structured grids using the CPU+GPU hybrid computing environment // Numerical Methods and Programming. 2016. Vol. 17, no. 1. P. 65–71. DOI: 10.26089/NumMet.v17r107.

8. Ding P., Liu Z. Accelerating phase-field modeling of solidification with a parallel adaptive computational domain approach // International Communications in Heat and Mass Transfer. 2020. Vol. 111. P. 104452. DOI: 10.1016/j.icheatmasstransfer.2019.104452.

9. Молостов И.П., Щербинин В.В. Применение технологии NVIDIA CUDA для численного моделирования распространения электромагнитных импульсов // Известия Алтайского государственного университета. 2015. Т. 1, № 1/1(85). DOI: 10.14258/izvasu(2015)1.1-06.

10. Краснопольский Б.И., Медведев А.В., Чулюнин А.Ю. Применение графических ускорителей для расчета гидродинамических характеристик гребных винтов в пакете OpenFOAM // Труды Института системного программирования РАН. 2014. Т. 26, № 5. С. 155–172. DOI: 10.15514/ISPRAS-2014-26(5)-8.

11. Егоров М.Ю., Егоров С.М., Егоров Д.М. Применение графических ускорителей для повышения производительности вычислений при численном моделировании функциони-

рования сложных технических систем // Вестник ПНИПУ. Аэрокосмическая техника. 2015. № 40. С. 81–91. DOI: 10.15593/2224-9982/2015.40.05.

12. Szenasi S. Solving the inverse heat conduction problem using NVLink capable Power architecture // PeerJ Computer Science. 2017. Vol. 3. P. 138. DOI: 10.7717/peerj-cs.138.

13. Zheng L., Gerya T., Knepley M., *et al.* GPU Implementation of Multigrid Solver for Stokes Equation with Strongly Variable Viscosity // GPU Solutions to Multi-scale Problems in Science and Engineering. Springer, 2013. P. 321–333. Lecture Notes in Earth System Sciences. DOI: 10.1007/978-3-642-16405-7_21.

14. Konovalov A. The steepest descent method with an adaptive alternating-triangular preconditioner // Differential Equations. 2004. Vol. 40. P. 1018–1028.

15. Sukhinov A.I., Chistyakov A.E., Litvinov V.N., *et al.* Computational Aspects of Mathematical Modeling of the Shallow Water Hydrobiological Processes // Numerical methods and programming. 2020. Vol. 21. P. 452–469. DOI: 10.26089/NumMet.v21r436.

16. Самарский А.А., Вабищевич П.Н. Численные методы решения уравнений конвекции-диффузии. Москва: УРСС, 2009.

17. Browning J.B., Sutherland B. C++20 Recipes. A Problem-Solution Approach Berkeley, CA: Apress, 2020. 630 p.

Сухинов Александр Иванович, чл.-корр. РАН, д.ф.-м.н., профессор, кафедра математики и информатики, Донской государственный технический университет (Ростов-на-Дону, Российская Федерация)

Литвинов Владимир Николаевич, к.т.н., доцент, кафедра математики и информатики, Донской государственный технический университет (Ростов-на-Дону, Российская Федерация); кафедра математики и биоинформатики, Азово-Черноморский инженерный институт ФГБОУ ВО Донской ГАУ (Зерноград, Российская Федерация)

Чистяков Александр Евгеньевич, д.ф.-м.н., кафедра программного обеспечения вычислительной техники и автоматизированных систем, Донской государственный технический университет (Ростов-на-Дону, Российская Федерация)

Никитина Алла Валерьевна, д.т.н., доцент, кафедра программного обеспечения вычислительной техники и автоматизированных систем, Донской государственный технический университет (Ростов-на-Дону, Российская Федерация); кафедра интеллектуальных и многопроцессорных систем, Южный федеральный университет (Ростов-на-Дону, Российская Федерация)

Грачева Наталья Николаевна, к.т.н., доцент, кафедра математики и биоинформатики, Азово-Черноморский инженерный институт ФГБОУ ВО Донской ГАУ (Зерноград, Российская Федерация); кафедра проектирования и технического сервиса транспортно-технологических систем, Донской государственный технический университет (Ростов-на-Дону, Российская Федерация)

Руденко Нелли Борисовна, к.т.н., доцент, кафедра математики и биоинформатики, Азово-Черноморский инженерный институт ФГБОУ ВО Донской ГАУ (Зерноград, Российская Федерация); кафедра медиатехнологий, Донской государственный технический университет (Ростов-на-Дону, Российская Федерация)