My version of great Optimization of the Computational Process for Solving Grid Equations on a Heterogeneous Computing System*

Bukiraevskiy Kibita[1], Kokov Panton[1,2(✉)][0000−0001−8234−3194],
Vsepolozhina Kalena[1], Hekererov Gerbariy[1,3], Segavanedam Pareton[2], and
Sisitoev Razdam[2]

[1] Don State Technical University, Rostov-on-Don, Russia,
sukhinov@gmail.com,
litvinovvn@rambler.ru,
cheese_05@mail.ru,
[2] Azov-Black Sea Engineering Institute
of Don State Agrarian University, Zernograd, Russia
79286051374@yandex.ru,
nelli-rud@yandex.ru
[3] Southern Federal University, Rostov-on-Don, Russia
nikitina.vm@gmail.com

Abstract. To predict emergencies and irreversible consequences of human activity, scientists widely use mathematical modeling. At the same time, it is very important to minimize the time to make a decision. It is necessary to develop effective methods for solving systems of large-dimensional grid equations with a non-self-adjoint operator in the numerical solution of hydrophysics and biological kinetics problems. The large volume of processed information and the complexity of calculations lead to the need to use computing clusters, which include video adapters to increase the computing system performance and the data conversion rate. The aim of the research is to develop a software module that implements an algorithm for solving a system of linear algebraic equations (SLAE) by the modified alternately triangular iterative method (MATM) (self-adjoint and non-self-adjoint cases) using NVIDIA CUDA technology. The decomposition method of computational domain in the three-dimensional case is described. A graph model of the organization of a parallel pipeline computing process focused on heterogeneous computing systems is proposed. As a result of experimental data processing, a regression equation with a determination coefficient of 0.86 is obtained. The parameters of the obtained regression equation are the size of the CUDA computing block along the Oy axis and the size ratio along the Ox and Oz axes. The performed numerical experiments have shown that the minimum calculation time of one MATM step is achieved

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

with the largest available value of the CUDA computing block size along the Ox axis.

Keywords: Mathematical modeling · Parallel algorithm · Graphics accelerator

1 Introduction

The prediction of environmental risks allows to reduce damage from adverse and emergencies in nature, in particular, in the coastal water zone. Research in this area requires the construction of mathematical models and the study of influence of various factors on the research object.

Currently, computer modeling is becoming more relevant, which replaces complex systems and physical models, as well as allows us to predict various phenomena and processes in nature. Computer modeling is usually based on mathematical models, the discretization of which leads to a large-dimensional SLAE with self-adjoint and non-self-adjoint operators. Solving such systems of grid equations requires a lot of computing capacity.

Many Russian and foreign scientists are engaged in research and forecasting of aquatic ecosystems. Scientists of the Marchuk Institute of Computational Mathematics of the Russian Academy of Sciences and the Keldysh Institute of Applied Mathematics are engaged in the analysis and modeling of complex systems (in ecology, environment, etc.), modeling of hydrodynamic processes, and forecasting of climate changes in the world ocean. Researches is performed on modeling hydrophysical processes on the example of the Azov Sea under the leadership of G.G. Matishov. Mathematical models of sea level dynamics are described in the papers of Bonaduce A., Staneva J. [1]. Scientists Marchesiello P. [2], Androsov A. [3], etc. are engaged in improving the ocean models. Existing standard software often include simplified mathematical models that do not take into account the spatially inhomogeneous water transport, and have insufficient accuracy in modeling the vortex structures of water flow currents, shore and bottom topography [1–4]. The actual direction of improving software systems is the development of parallel algorithms that are executed both on the CPU (Central Processing Unit) and on the GPU (Graphics Processing Unit). Scientists Weicheng Xue and Christopher J. Roy are engaged in research, related to optimizing computing performance at solving fluid dynamics problems on multiple GPUs, improving the performance of multi-GPUs on structured grids. In their work, the use of GPUs improves the performance by 30-70 times [5,6]. Researchers Taku Nagatake and Tomoaki Kunugi analyzed the possibility of using the GPU to accelerate the calculation of multiphase flows. They determined that the calculation time on the GPU (single GTX280) was about 4 times faster than the calculation time on the CPU (Xeon 5040, 4 parallelized threads) [7]. David J. Munk and Timoleon Kipouros describe the acceleration of the optimization process of multi-physical topology on the GPU architecture [8].

To increase the efficiency of using GPU computing resources, we proposed an algorithm and a software module that implements it, which allows using functions from the NVIDIA CUDA library to select the optimal solution for large-dimensional SLAE in the case of self-adjoint and non-self-adjoint operators. The developed software tools make it possible to more efficiently use the heterogeneous computing system resources, used to solve computationally labors spatial and three-dimensional problems of hydrophysics.

2 Grid equations solving method

Let A be is linear, positive definite operator ($A > 0$) and in a finite-dimensional Hilbert space H it is necessary to solve the operator equation

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

For the grid equation (1), iterative methods are used, which in canonical form can be represented by the equation [9,10]

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

where m is the iteration number, $\tau_{m+1} > 0$ is the iteration parameter, B is the preconditioner. Operator B is constructed proceeding from the additive representation of the operator A_0 – the symmetric part of the operator A

$$A_0 = R_1 + R_2, R_1 = R_2^*, \tag{3}$$

where $A = A_0 + A_1$, $A_0 = A_0^*$, $A_1 = -A_1^*$.

The preconditioner is formed as follows

$$B = (D + \omega R_1)\, D^{-1} \left(D + \omega R_2\right), D = D^* > 0, \omega > 0, \tag{4}$$

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:

$$\begin{gathered}
r^m = Ax^m - f, B(\omega_m)w^m = r^m, \tilde{\omega}_m = \sqrt{\frac{(Dw^m, w^m)}{(D^{-1}R_2w^m, R_2w^m)}}, \\
s_m^2 = 1 - \frac{(A_0w^m, w^m)^2}{(B^{-1}A_0w^m)\,(Bw^m, w^m)}, k_m^2 = \frac{\left(B^{-1}A_1w^m, A_1w^m\right)}{(B^{-1}A_0w^m, A_0w^m)}, \\
\theta_m = \frac{1 - \sqrt{\frac{s_m^2 k_m^2}{(1+k_m^2)}}}{1 + k_m^2\,(1 - s_m^2)}, \tau_{m+1} = \theta_m \frac{(A_0w^m, w^m)}{(B^{-1}A_0w^m, A_0w^m)}, \\
x^{m+1} = x^m - \tau_{m+1}w^m, \omega_{m+1} = \tilde{\omega}_m,
\end{gathered} \tag{5}$$

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

3 Software implementation of the method for solving grid equations

The solution of grid equations using the MATM (modified alternately triangular iterative method) involves the use of a three-dimensional uniform computational grid [11]:

$$\begin{aligned}\bar{w}_h = \{t^n = n\tau, x_i = ih_x, y_i = jh_y, z_k = kh_z,\\ n = \overline{0, n_t - 1}, i = \overline{0, n_1 - 1}, j = \overline{0, n_2 - 1}, k = \overline{0, n_3 - 1},\\ (n_t - 1)\,\tau = T, (n_1 - 1)\,h_x = l_x, (n_2 - 1)\,h_y = l_y, (n_3 - 1)\,h_z = l_z\},\end{aligned}$$

where τ is the time step; h_x, h_y, h_z are space steps; n_t is the time layers number; T is the upper bound on the time coordinate; n_1, n_2, n_3 are the nodes number by spatial coordinates; l_x, l_y, l_z are space boundaries of a rectangular parallelepiped in which the computational domain is inscribed.

The system of grid equations are obtained at discretization of mathematical physics models, in particular, hydrodynamics. Each equation of the system can be represented in a canonical form, and we will use a seven-point pattern:

$$c(m_0)u(m_0) - \sum_{i=1}^{6} c(m_0, m_i)u(m_i) = F(m_0),$$

$m_0(x_i, y_j, z_k)$ is the template center, $M'(P) = \{m_1(x_{i+1}, y_j, z_k), m_2(x_{i-1}, y_j, z_k), m_3(x_i, y_{j+1}, z_k), m_4(x_i, y_{j-1}, z_k), m_5(x_i, y_j, z_{k+1}), m_5(x_i, y_j, z_{k-1})\}$ is a surrounding area of the center, $c_0 \equiv c(m_0)$ is a template center ratio, $c_i \equiv c(m_0, m_i)$ are coefficients of the template center neighborhood, F is the vector of the right parts, u is the calculated vector.

The developed software module uses one-dimensional arrays. The transition from a three-dimensional representation of a grid node (i, j, k) to a one-dimensional form (node number) is performed using the following formula:

$$m_0 = i + jn_1 + kn_1n_2.$$

The node numbers in the surrounding area of the template center $m_i, i = \overline{1, 6}$ are calculated using the formulas:

$$m_1 = m_0 + 1, m_2 = m_0 - 1, m_3 = m_0 + n_1,$$

$$m_4 = m_0 - n_1, m_5 = m_0 + n_1n_2, m_6 = m_0 - n_1n_2.$$

The sequential version of the MATM algorithm consists of four stages and is described in detail in [10]. 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 a lower-triangular and upper-triangular matrix:

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

Algorithm 1 matm(IN: $n_1, n_2, n_3, c_0, c_2, c_4, c_6, \omega$; IN/OUT: r)

```
for $k \in [1; n_3 - 2]$ do
    for $i \in [1; n_1 - 2]$ do
        for $j \in [1; n_2 - 2]$ do
            $m_0 \leftarrow i + n_1 \cdot j + n_1 \cdot n_2 \cdot k$
            if $c_0[m_0] > 0$ then
                $m_2 \leftarrow m_0 - 1;\ m_4 \leftarrow m_0 - n_1;\ m_6 \leftarrow m_0 - n_1 \cdot n_2$
                $r[m_0] \leftarrow (\omega \cdot (c_2[m_0] \cdot r[m_2] + c_4[m_0] \cdot r[m_4] + c_6[m_0] \cdot r[m_6]) +$
                $+r[m_0])/((0.5 \cdot \omega + 1) \cdot c_0[m_0])$
```

The fragment of the algorithm 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 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.

4 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 us to fully utilize all available computing resources, including high-performance graphics accelerators. The distinctive feature of the proposed algorithms is the possibility of using the calculators with different performance. This allows us to organize the distributed computing using different models of central processing units (CPUs) on different nodes and even different video accelerators inside a separate computing node. It is assumed that each computing node of the system can contain from 1 to 2 central processing units (CPUs) containing from 1 to 32 cores, and from 1 to 12 NVIDIA video accelerators with support for CUDA technology (GPU), containing from 192 (NVIDIA GeForce GT 710) to 5120 (NVIDIA Tesla V100) CUDA cores. Data exchange between nodes is performed using MPI technology (Message Passing Interface).

In the process of solving computational problems, it is necessary to dynamically distribute the computational load between dissimilar computers. Therefore, a class library was developed in C++, which allows describing the structure and hardware of a computing cluster. The class library contains the following classes:

- ComputingCluster, describes the structure of a computing cluster. Stores objects describing computational nodes in the std::map container. The class implements a number of auxiliary methods that allow you to manage the list of computational nodes, determine the total performance of the cluster and display detailed information about it.
- ComputingNode, describes the structure and characteristics of a computing node. The methods of the class allow you to manage the list of computing

devices, determine the total performance and the size of the random access memory of the computing node.

- ComputingDevice is an abstract class that describes the general characteristics of computing devices located in a separate computing node of a cluster.
- ComputingDeviceCPU, a heir of the abstract ComputingDevice class, describing the characteristics of the CPU.
- ComputingDeviceGPU, a heir of the abstract ComputingDevice class, describes the characteristics of the GPU.

The organization of calculations on each node is performed by an algorithm that controls all available CPU and GPU threads (calculators). Each calculator performs calculations only for its own fragment of the computational domain. For this, the computational domain is divided into subdomains that are assigned to individual computing nodes (Fig. 1). Next, each subdomain is divided into blocks assigned to each computing device (CPU or GPU). After that, each block is divided into fragments assigned to calculators (CPU cores and GPU streaming multiprocessors). Notations in Fig. 1: $Node1$, $Node2$, $Node3$ are computing nodes; $Device1$, $Device2$, $Device3$ are blocks of the computational domain, calculated on separate computing devices of the node.

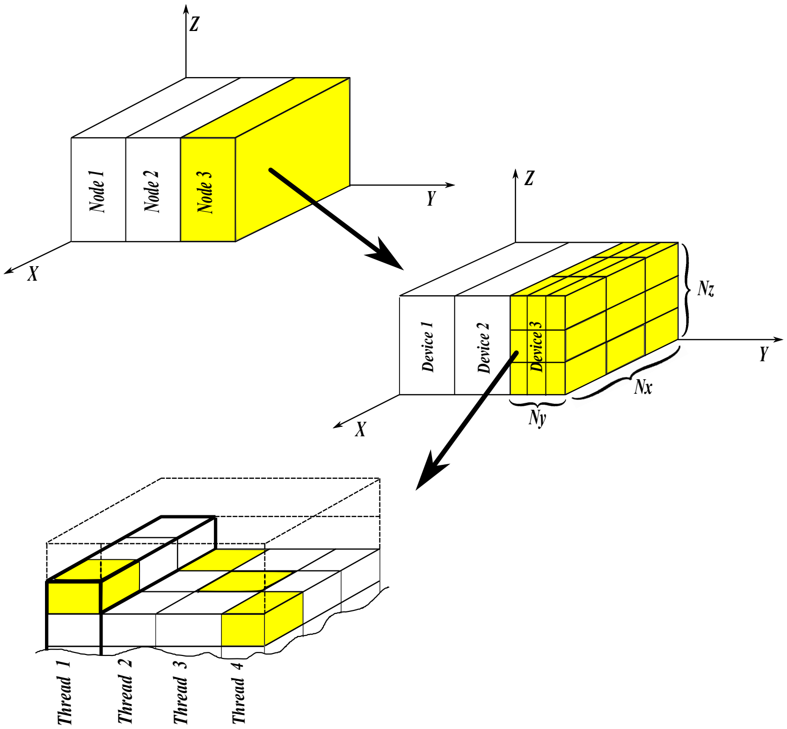

Fig. 1. Decomposition of the computational domain

The subdivision of subdomains into fragments that are mapped to each calculator inside a separate computing node is performed as follows: the number of fragments of the computational domain along the Oz axis is selected as the smallest common multiple of the optimal dimensions of the CUDA computing blocks for all video accelerators involved in the cluster (Fig. 2). In Fig. 2: Nv is the calculator index; Nx is the number of nodes of the computational domain on the axis Ox; Ny_0, Ny_1, Ny_2, Ny_3 is the number of nodes of the computational domain on the axis Oy for calculator with indexes 0, 1, 2 and 3, respectively; z is the layer index on the axis Oz; s is the index of the pipeline calculation stage. The number of fragments of the computational domain on the Ox axis in the block (Nx) is selected in such way that their number is greater than the number of calculators in the cluster and they are the same. The number of fragments of the computational domain along the Oy axis in the block is selected so that the calculation time of each block by different calculators is approximately the same. For this, 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.

Fragments of the computational domain that are processed in parallel are highlighted in gray. Note that the calculator index coincides with the fragment index of the computational domain on the Oy axis.

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. 3). Each graph node is an object of a class 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, and pointers to objects that describe the parameters of the 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 $s = ki + j$.

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 graphics accelerators [12–16].

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 c_0, c_2, c_4, c_6 and the constant ω. The output parameter is the vector of the water flow velocity r. Before running the algorithm, it is necessary to programmatically set the dimensions of the 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 rows 3 and 5. In line 4, a

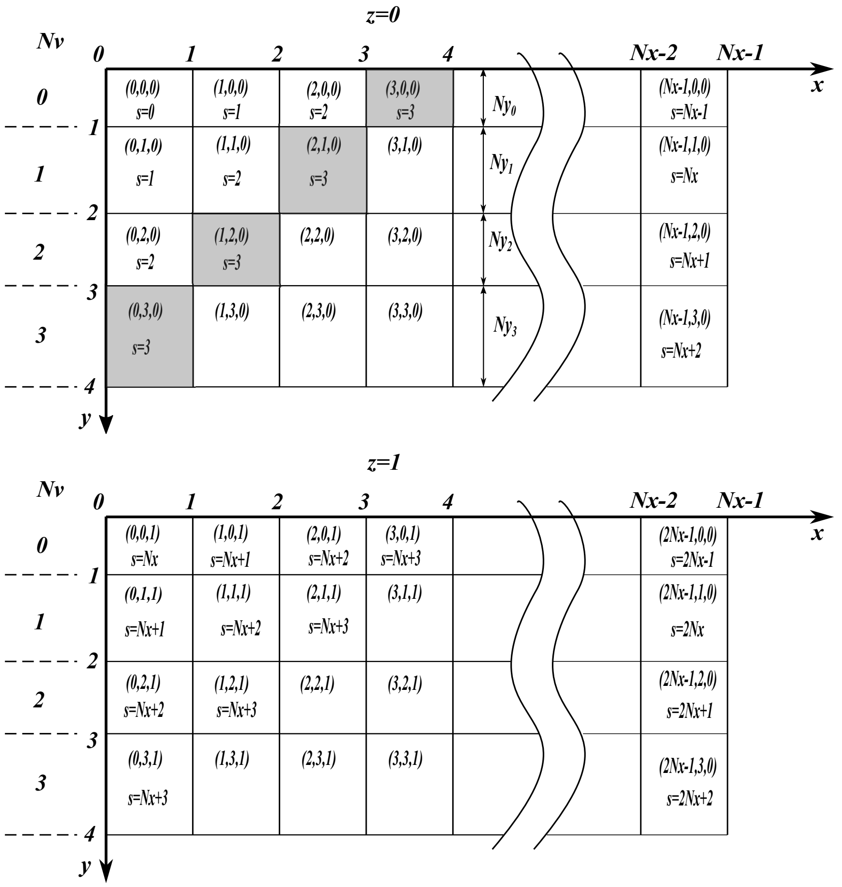

Fig. 2. Decomposition of the computational subdomain, calculated by a separate computing node with organization of a parallel pipeline computing process

variable j is initialized that represents a counter by coordinate y. The calculation pipeline is organized as a loop in line 6. The indexes of the central node of the grid pattern m_0 and the surrounding nodes m_2, m_4, m_6 are calculated in lines 8, 10-12. The two-dimensional array *cache* is located in the GPU shared memory and designed to store the results of calculations 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 conducted researches show a significant dependence of the algorithm implementation time for calculation the preconditioner on the ratio of threads in spatial coordinates.

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 8000 MHz, respectively, and a video memory

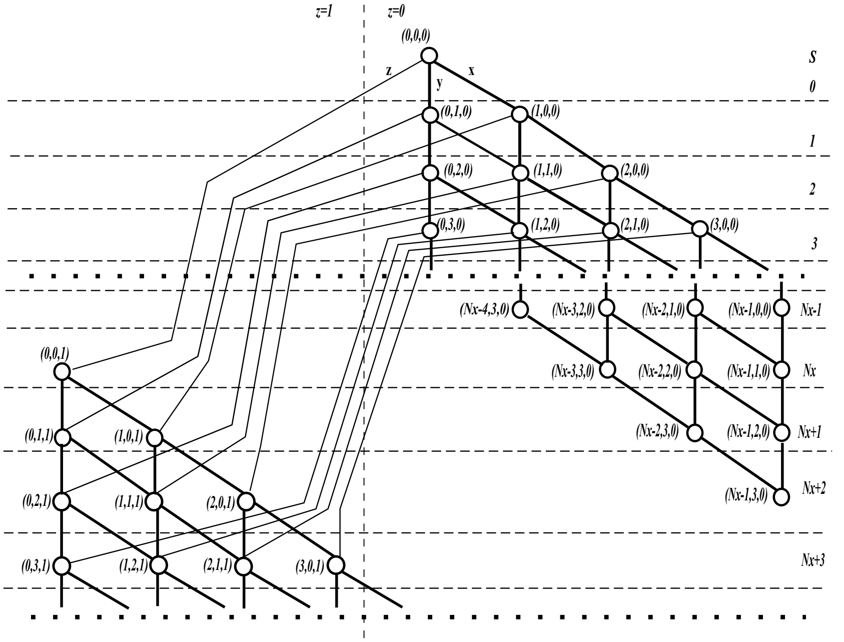

Fig. 3. A graph model that describes the relationships between adjacent fragments of the computational grid and the process of pipeline calculation

bus bit rate of 128 bits. The computing part consists of 56 texture processor clusters (TPC) with 2 multiprocessors (SM) in each. Each multiprocessor contains 8 streaming processors (SP) or CUDA cores. Therefore, the number of CUDA cores for the GeForce GTX 1650 video adapter is 896.

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 number of threads on the axis Oy. Values of the objective function: T_{GPU} is implementation time of one MATM step on the GPU in terms of 1000 nodes of the computational grid, ms. The multiply of threads X and Z must not exceed 640-the number of threads in a single block. Therefore, the levels of variation of the values X and Z were chosen taking into account the CUDA limitations. So, for example, the number of threads on the Oy axis varied in the range $[1000, 30000]$. The experimental data analysis for factor values $X = 1$, $Z = 640$ and $X = 640$, $Z = 1$ shown that the allocated memory is not used at calculation the objective function at the specified points. Therefore, these points must be excluded at performing the regression analysis.

The regression equation was obtained as a result of the experimental data processing:

Algorithm 2 matmKernel(IN: $c_0, c_2, c_4, c_6, \omega$ IN/OUT: r;)

```
$threadX \leftarrow blockDim.x \cdot blockIdx.x + threadIdx.x$
$threadZ \leftarrow blockDim.z \cdot blockIdx.z + threadIdx.z$
$i \leftarrow threadX + 1$; $j \leftarrow 1$; $k \leftarrow threadZ + 1$
for $s \in [3; n_1 + n_2 + n_3 - 3]$ do
    if $(i + j + k = s) \wedge (s < i + n_2 + k)$ then
        $m_0 \leftarrow i + (blockDim.x + 1) \cdot j + n_1 \cdot n_2 \cdot k$
        if $c0[m0] > 0$ then
            $m_2 \leftarrow m_0 - 1; m_4 \leftarrow m_0 - n_1; m_6 \leftarrow m_0 - n_1 \cdot n_2$
            $rm4 \leftarrow 0$
            if $(s > 3 + threadX + threadZ)$ then
                $rm4 \leftarrow cache[threadX][threadZ]$
            else
                $rm4 \leftarrow r[m_4]$
            $rm2 \leftarrow 0$
            if $(threadX \neq 0) \wedge (s > 3 + threadX + threadZ)$ then
                $rm2 \leftarrow cache[threadX - 1][threadZ]$
            else
                $rm2 \leftarrow r[m_2]$
            $rm6 \leftarrow 0;$
            if $(threadZ \neq 0) \wedge (s > 3 + threadX + threadZ)$ then
                $rm6 \leftarrow cache[threadX][threadZ - 1]$
            else
                $rm6 \leftarrow r[m_6]$
            $rm0 \leftarrow (\omega \cdot (c2[m_0] \cdot rm2 + c4[m_0] \cdot rm4 + c6[m_0] \cdot rm6) + r[m_0])/$
            $/((0.5 \cdot \omega + 1) \cdot c_0[m_0])$
            $cache[threadX][threadZ] \leftarrow rm0$
            $r[m_0] \leftarrow rm0$
        $j \leftarrow j + 1$
```

$$T_{GPU} = a - b \cdot Y - c \cdot ln(k) - d \cdot ln(Y), \tag{6}$$

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; $a = 0.026; b = 0.0000002; c = 0.00016; d = 0.00077$. The graph of the objective function is given in Fig. 4.

The analysis of the graph, constructed according to the equation (6), shows a slowdown in the calculation speed at $k < 10$ and $Y < 1000$, explained in this case by the inefficient use of the distributed memory of the graphics accelerator (Fig. 4).

As a result of the experimental data analysis, it was found that the shortest implementation time of one MATM step in terms of 1000 nodes of the computational grid on the GeForce GTX 1650 video adapter will be obtained with the largest number of threads along the Oy axis and the highest coefficient value k. The implementation time of MATM one step in terms of 1000 nodes of the

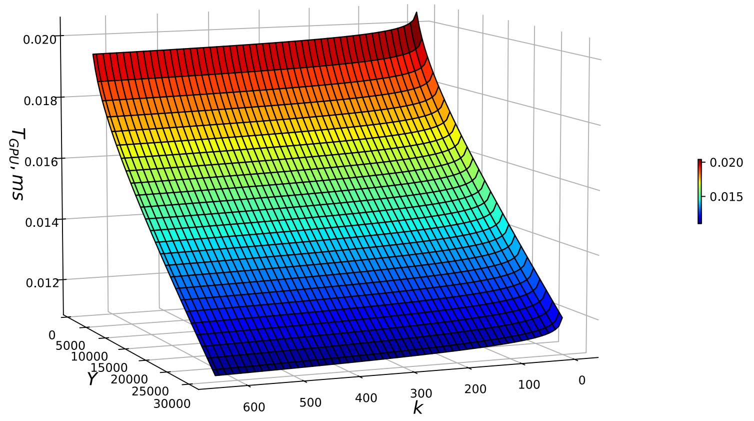

Fig. 4. Surface of the response function $T_{GPU} = f(k, Y)$

computational grid on the GeForce GTX 1650 video adapter is inversely proportional to the number of the computational grid nodes on the Oz axis, i.e., with an increase in the number of nodes on the Oz axis, the calculation time decreases. The highest value of the coefficient k is achieved, when the number of threads on the axis Ox increases and the number of threads on the axis Oz decreases. Therefore, it is advisable to perform the decomposition of the computational domain in the form of parallelepipeds, in which the size on the axis Oz is minimal, and on the axis Ox is maximal. The choice of the decomposition method of computational domain in the form of parallelepipeds must be made taking into account the architecture of the video adapter.

When developing a parallel algorithm that implements the pipeline process of computations, it is necessary to take into account the amount of data transmitted between computational nodes - the size of the transmitted plane (the number of elements in the plane). To dynamically determine the size of the transmitted plane, it is necessary to determine the functional dependence of the time spent on transferring data between computational nodes on the size of the transmitted plane. The resulting dependence will make it possible to obtain such a decomposition of the computational domain, which will reduce the execution time of the entire parallel algorithm.

The purpose of the experiment is to determine the functional dependence of the data transfer time between computational nodes on the number of elements in the transmitted plane. The number of elements in the plane is taken as a factor - V. The value of the objective function $Time$, ms - transmission time of the number of elements V.

To determine the dependence of the time of data transmission between computational nodes on the number of transmitted elements, an algorithm and its software implementation in the C language were developed. The program considered three ranges of dimensions of the transmitted plane: $V \in [1; 100]$ with a step of 1, $V \in [100; 10000]$ with a step of 100 elements, and $V \in [10000; 1000000]$ with a step of 10000 elements.

The developed algorithm was tested on the main computing resource of the Keldysh Institute of Applied Mathematics - the K-60 computing cluster. The specified cluster consists of two sections - one without graphics accelerators k60.kiam.ru, the other with graphics accelerators k60gpu.kiam.ru. The hardware of the GPU section consists of 10 computational nodes. Each node is a dual-processor server with the following characteristics: 2 x Intel Xeon Gold 6142 v4 processors (16 x cores), 4 x nVidia Volta GV100GL GPU, 768GB RAM, 2TB disk.

As a result of processing experimental data for the range of dimensions of the transmitted plane from 10,000 to 1,000,000 elements, a regression equation was obtained:

$$T = a + bV, \tag{7}$$

where T is the transmission time of the number of V elements, ms. The coefficient of determination was 0.995; $a = 278.72$; $b = 0.038$.

The resulting functional dependence is used by the decomposition algorithm to dynamically determine the dimensions of the plane transmitted between the computational nodes.

5 Conclusions

The algorithm and software module implementing it were developed as a result of the performed research, designed to solve the SLAE (self-adjoint and non-self-adjoint cases) that arise during the discretization of spatial-three-dimensional model problems of mathematical physics, with the help of MATM using NVIDIA CUDA technology.

The graph model was proposed that makes it possible to organize a parallel pipeline computing process on the GPU, designed to solve the systems of large-dimensional grid equations.

It was established that the shortest implementation time of one MATM step per 1000 nodes of the computational grid on the GeForce GTX 1650 video adapter will be obtained with the largest number of threads on the Oy axis and the highest value of the coefficient k, directly proportional to the number of threads, on the Ox axis. The optimal decomposition method of the three-dimensional computational grid with the number of nodes from 10^8 to 10^{11} and the number of time layers from 10^4 and more, if we focus on the limitations of computational stability and accuracy of discrete models, applicable for the GPU, was described.

References

1. Bonaduce, A., Staneva, J., Grayek, S., Bidlot J.-R., Breivik O.: Sea-state contributions to sea-level variability in the European Seas. Ocean Dynamics, vol. 70, pp. 1547–1569 (2020) doi:10.1007/s10236-020-01404-1
2. Marchesiello, P., Mc.Williams, J.C., Shchepetkin, A.: Open boundary conditions for long-term integration of regional oceanic models. Oceanic Modelling Journal, vol. 3, pp. 1–20 (2001) doi:10.1016/S1463-5003(00)00013-5
3. Androsov, A.A., Wolzinger, N.E.: The straits of the world ocean: a General approach to modelling (In Russian). Nauka, Saint Petersburg (2005)
4. Nieuwstadt, F., Boersma, B.J., Westerweel, J.: Turbulence. Introduction to Theory and Applications of Turbulent Flows. Springer, Cham (2016) doi:10.1007/978-3-319-31599-7
5. Xue, W., Roy, C.J.: Multi-GPU performance optimization of a computational fluid dynamics code using OpenACC. Concurrency and Computation Practice and Experience, vol. 33, pp. 1547–1569 (2021) doi:10.1002/cpe.6036
6. Xue, W., Jackson, C.W., Xue, W., Roy, C. J.: Multi-CPU/GPU Parallelization, Optimization and Machine Learning based Autotuning of Structured Grid CFD Codes. AIAA Aerospace Sciences Meeting, p. 0362 (2018)
7. Nagatake, T., Kunugi, T.: Application of GPU to Computational Multiphase Fluid Dynamics. IOP Conference Series: Materials Science and Engineering, vol. 10 (2010)
8. Munk, D.J., Kipouros, T., Vio, G.A.: Multi-physics bi-directional evolutionary topology optimization on GPU-architecture. Engineering with Computers, vol. 35, pp. 1059–1079 (2019) doi:10.1007/s00366-018-0651-1
9. Sukhinov, A.I., Atayan, A.M., Belova, Y.V., Litvinov, V.N., Nikitina, A.V., Chistyakov, A.E.: Data processing of field measurements of expedition research for mathematical modeling of hydrodynamic processes in the Azov Sea. Computational Continuum Mechanics, vol. 13(2), pp. 161–174 (2020) doi:10.7242/1999-6691/2020.13.2.13
10. Sukhinov, A., Litvinov, V., Chistyakov, A., Nikitina, A., Gracheva, N., Rudenko, N.: Computational aspects of solving grid equations in heterogeneous computing systems. In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science vol. 12942, pp. 166–177 (2021). doi:10.1007/978-3-030-86359-3_13
11. Oyarzun, G., Borrell, R., Gorobets, A., Oliva, A.: MPI-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner. Computers and Fluids, vol. 92, pp. 244–252 (2014) doi: 10.1016/j.compfluid.2013.10.035
12. Zheng, L., Gerya, T., Knepley, M., Yuen, D., Zhang, H., Shi, Y.: GPU Implementation of Multigrid Solver for Stokes Equation with Strongly Variable Viscosity. (2013) doi:10.1007/978-3-642-16405-7_21
13. Konovalov, A.: The steepest descent method with an adaptive alternating-triangular preconditioner. Differential Equations, vol. 40, pp. 1018–1028 (2004)
14. Sukhinov, A.I., Chistyakov, A.E., Litvinov, V.N., Nikitina, A.V., Belova, Yu.V., Filina, A.A.: Computational Aspects of Mathematical Modeling of the Shallow Water Hydrobiological Processes. Numerical methods and programming, vol. 21, pp. 452–469 (2020). doi:10.26089/NumMet.v21r436
15. Samarskii, A.A., Vabishchevich, P.N.: Numerical methods for solving convection-diffusion problems (In Russian). URSS, Moscow (2009)
16. Browning, J.B., Sutherland, B.: C++20 Recipes. A Problem-Solution Approach Apress, Berkeley, CA, p. 630 (2020)