Optimization of the Computational Process for Solving Grid Equations on a Heterogeneous Computing System

Alexander Sukhinov[1], Vladimir Litvinov[1,2](✉), Alexander Chistyakov[1], Alla Nikitina[1,3], Natalia Gracheva[2], and Nelli Rudenko[2]

[1] Don State Technical University, Rostov-on-Don, Russia
litvinovvn@rambler.ru
[2] Azov-Black Sea Engineering Institute of Don State Agrarian University, Zernograd, Russia
[3] Southern Federal University, Rostov-on-Don, Russia

Abstract. To predict emergencies and the irreversible consequences of human activity, scientists widely use mathematical modeling. In the event of an emergency, it is important that the time for its elimination be the shortest. 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. A large amount 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 research aim is to develop a software module that implements an algorithm for solving a system of linear algebraic equations (SLAE) of large dimensions by the modified alternately triangular iterative method (MATM), applicable in heterogeneous computing systems. The decomposition method of the computational domain in the three-dimensional case is described. A graph model for organizing a parallel pipeline computing process focused on heterogeneous computing systems is proposed. Based on the results of the research, a regression equation is obtained with a coefficient of determination equal to 0.86. 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 show that the minimum calculation time of one MATM step is achieved with the largest available value of the CUDA computing block size along the Ox axis.

Keywords: System of linear algebraic equations · Heterogeneous Computing System · Parallel algorithm

1 Introduction

The prediction of environmental risks allows one to reduce the damage from adverse situations and emergencies in nature, in particular, in the coastal water

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

zone. Research in this area requires the construction of mathematical models and the study of the influence of various factors on the research object.

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

Both Russian and foreign researchers study the processes occurring in various reservoirs and water systems. 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. Studies on modeling hydrophysical processes are performed 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 A. Bonaduce, J. Staneva [1]. Scientists P. Marchesiello [2], A. Androsov [3], etc. are engaged in improving the ocean models. The existing standard software often includes 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 executed on both the CPU (Central Processing Unit) and 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 analyze the possibility of using the GPU to accelerate the calculation of multiphase flows. They determine that the calculation time on the GPU (single GTX280) is 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 propose an algorithm and a software module that implements it, which allows using functions from the NVIDIA CUDA library to select the optimal solution for a large-dimensional SLAE in the case of self-adjoint and non-self-adjoint operators. The developed software tools make it possible to more efficiently utilize the heterogeneous computing system resources used to computationally solve spatial and three-dimensional problems of hydrophysics.

2 Method for Solving Grid Equations

It becomes necessary to solve a high-dimensional SLAE in the mathematical modeling process of hydrodynamics and hydrobiology problems

$$Ax = f, \tag{1}$$

where A is the linear, positive definite operator ($A > 0$) in the finite-dimensional Hilbert space H.

To solve SLAE (1) by iterative methods, the canonical form is used [9,10]

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

where m is the iteration number, $\tau_{m+1} > 0$ is the iteration parameter, B is the preconditioner, which is formed as follows

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

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

MATM calculation steps:

1. Calculation of the residual vector
$$r^m = Ax^m - f.$$
2. Calculation of the correction vector w^m
$$B(\omega_m)w^m = r^m.$$
3. Calculation of the convergence rate of the method
$$s_m^2 = 1 - \frac{\left(A_0 w^m, w^m\right)^2}{\left(B^{-1}A_0 w^m\right)\left(Bw^m, w^m\right)}.$$
4. Calculation of the ratio of the norm of the skew-symmetric part of the operator to the norm of the symmetric part
$$k_m^2 = \frac{\left(B^{-1}A_1 w^m, A_1 w^m\right)}{\left(B^{-1}A_0 w^m, A_0 w^m\right)},$$
5. Calculation of the coefficient θ_m
$$\theta_m = \frac{1 - \sqrt{\frac{s_m^2 k_m^2}{(1+k_m^2)}}}{1 + k_m^2\left(1 - s_m^2\right)}.$$
6. Calculation of the iteration parameter
$$\tau_{m+1} = \theta_m \frac{\left(A_0 w^m, w^m\right)}{\left(B^{-1}A_0 w^m, A_0 w^m\right)}.$$

7. Recalculation of the vector x at the next iteration

$$x^{m+1} = x^m - \tau_{m+1} w^m.$$

8. Recalculation of the coefficient ω at the next iteration

$$\omega_{m+1} = \sqrt{\frac{(Dw^m, w^m)}{(D^{-1}R_2 w^m, R_2 w^m)}}.$$

3 Software Implementation of the Method for Solving Grid Equations

The software implementation of the MATM for solving high-dimensional SLAEs is based on the developed parallel algorithms that implement a pipeline computing process. The use of these algorithms allows one to fully utilize all available computing resources, including high-performance graphics accelerators. A distinctive feature of the proposed algorithms is the possibility of using calculators with different performance. This allows one to organize distributed computing using different models of central processing units (CPUs) on different nodes and even different video accelerators (GPUs) inside a separate compute node. The software implementation enables to indicate the number and technical characteristics of the CPU and GPU at the initial stage of the decomposition of the computational grid for each compute node of the cluster. For each CPU, the number of cores is set. For the GPU, the number of streaming multiprocessors is specified. Calculations are performed on the K60 hybrid supercomputer installed at the Supercomputer Centre of Collective Usage of KIAM RAS.

In the process of solving computational problems, it is necessary to dynamically distribute the computational load between dissimilar computers. Therefore, a class library is developed in C++, it 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 compute nodes in the std::map container. The class implements a number of auxiliary methods that allow one to manage the list of compute nodes, to determine the total performance of the cluster and display detailed information about it.
– ComputingNode, describes the structure and characteristics of a compute node. The methods of the class allow one to manage the list of computing devices, to determine the total performance and the size of the random access memory of the compute node.
– ComputingDevice is an abstract class that describes the general characteristics of computing devices located in a separate compute 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, describing the characteristics of the GPU.

A multi-threaded computing process is controlled by an algorithm that allows each node to manage all available program threads (calculators) running on both the CPU and GPU. Each calculator handles only for its own fragment of the computational domain. For this, the computational domain is divided into subdomains assigned to individual compute 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 compute nodes; $Device1$, $Device2$, $Device3$ are blocks of the computational domain calculated on separate computing devices of the node. $Thread1$, $Thread2$, $Thread3$, $Thread4$ are fragment arrays of the computational domain calculated by separate threads of the computing device.

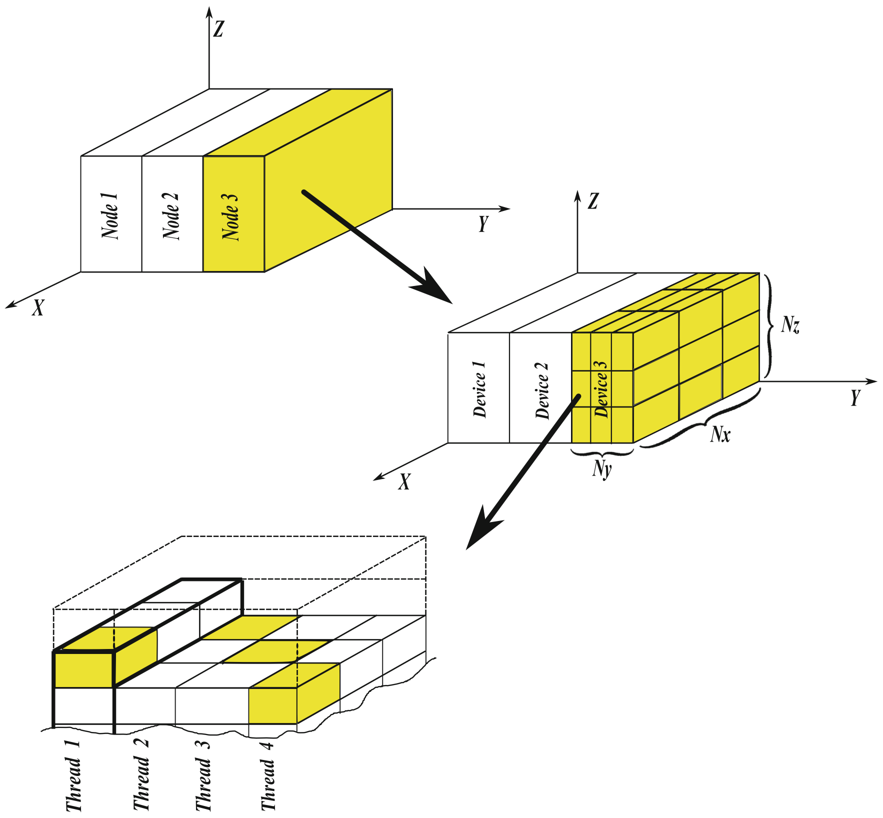

Fig. 1. Computational domain fragments distribution across the compute nodes, devices and threads

The subdivision of subdomains into fragments mapped to each calculator inside a separate compute 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 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 Ox axis; Ny_0, Ny_1, Ny_2, Ny_3 is the number of nodes of the computational domain on the Oy axis for the calculator with indexes 0, 1, 2 and 3, respectively; z is the layer index on the Oz axis; 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 a 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.

The fragments of the computational domain 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 is used to describe the relationships between the adjacent fragments of the computational grid and the organization of a 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 the adjacent fragments of the computational grid, and pointers to the objects that describe the parameters of the calculators. The computational process is a graph traversal from the root node with a 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 computational grid fragments assigned to graphics accelerators [12–16].

A fragment of the algorithm for solving a SLAE with a lower triangular matrix contains the following steps:

1. Computation of global thread indexes
$$tdX = bkDim.x \cdot bkIdx.x + tdIdx.x;$$
$$tdZ = bkDim.z \cdot bkIdx.z + tIdx.z.$$
2. Calculation of the indexes of the row, layer and initialization of the counter by the coordinate (variable) processed by the current thread
$$i = tdX + 1; k = tdZ + 1; j = 1.$$
3. Initialization of the loop parameter for calculating the residual vector: $s = 3$.

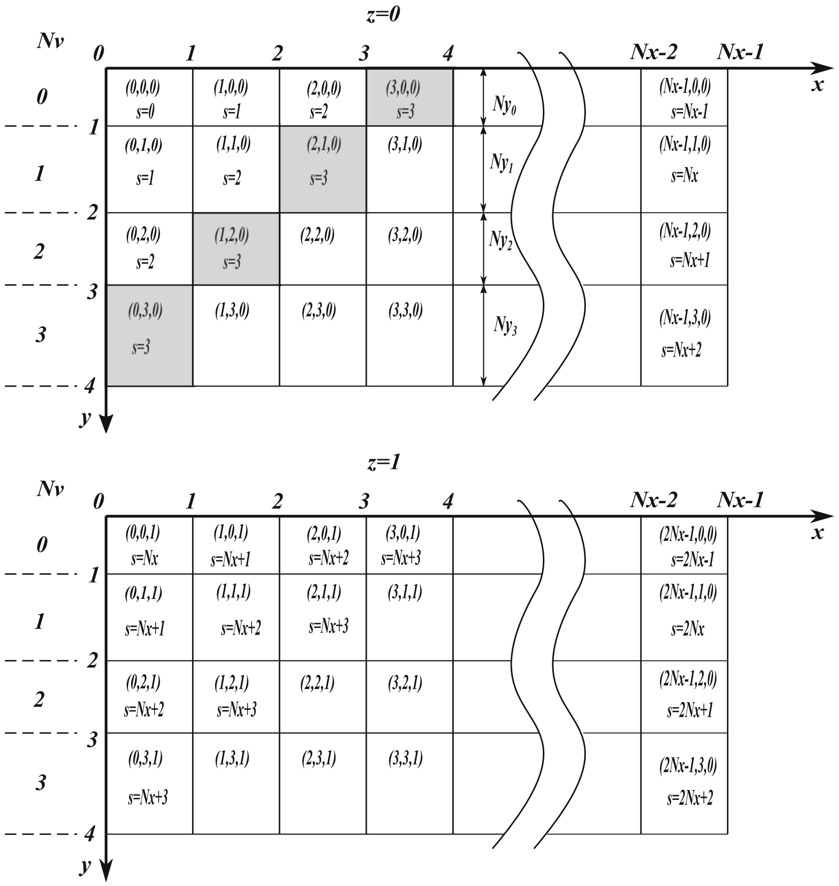

Fig. 2. Decomposition of the computational subdomain calculated by a separate compute node with the organization of a parallel pipeline computing process

4. Calculation of the indexes of the nodes of a seven-point grid pattern

$$mIdx_0 = i + (bkDim.x + 1) \cdot j + n_1 \cdot n_2 \cdot k;$$

$$mIdx_2 = mIdx_0 - 1;$$

$$mIdx_4 = mIdx_0 - n_1;$$

$$mIdx_6 = mIdx_0 - n_1 \cdot n_2.$$

5. Initialization of the residual vector value at the template point $mIdx_4 = 0$.
6. Checking the condition $(s > 3 + tdX + tdZ)$ for calculating the value of the residual vector at the template point $mIdx_4$. If the condition is met, then $rmIdx_4 = cmem[tdX][tdZ]$. Otherwise, $rmIdx_4 = r[mIdx_4]$.
7. Initialization of the residual vector value at the template point $mIdx_2 = 0$.

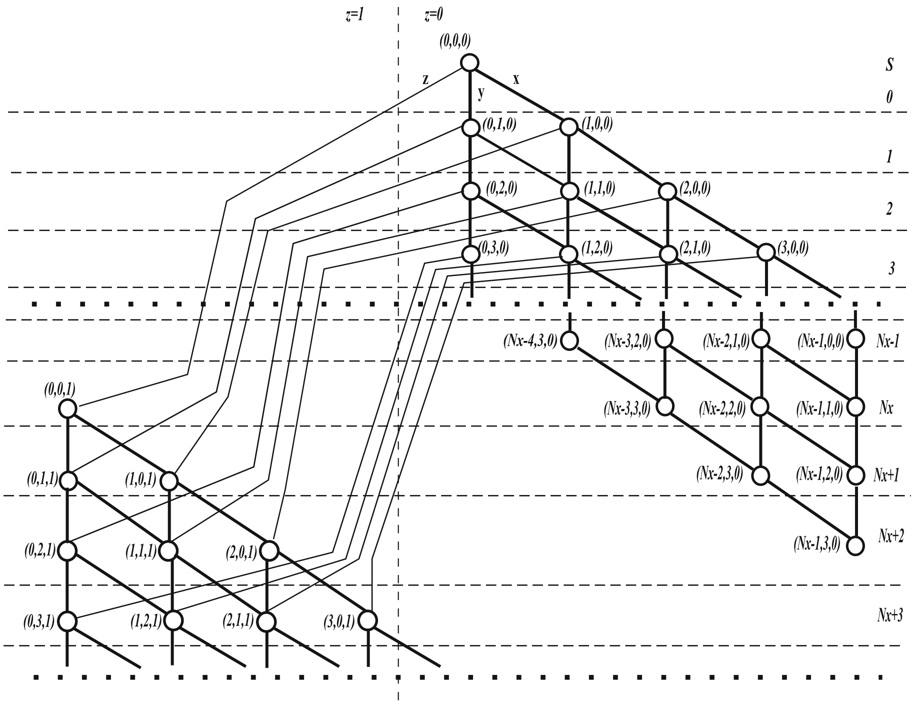

Fig. 3. Graph model that describes the relationships between the adjacent fragments of the computational grid and the process of pipeline calculation

8. Checking the condition $(tdX \neq 0) \wedge (s > 3 + tdX + tdZ)$ for calculating the value of the residual vector at the template point $mIdx_2$. If the condition is met, then $rmIdx_2 = cmem[tdX-1][tdZ]$. Otherwise, $rmIdx_2 = r[mIdx_2]$.
9. Initialization of the residual vector value at the template point $mIdx_6 = 0$.
10. Checking the condition $(tdZ \neq 0) \wedge (s > 3 + tdX + tdZ)$ for calculating the value of the residual vector at the template point $mIdx_6$. If the condition is met, then $rmIdx_6 = cmem[tdX][tdZ-1]$. Otherwise, $rmIdx_6 = r[mIdx_6]$.
11. Calculation of the value of the residual vector at the central point of the seven-point pattern $mIdx_0$

$$rmIdx_0 = (\omega \cdot (ksu_2[mIdx_0] \cdot rmIdx_2 + ksu_4[mIdx_0] \cdot rmIdx_4 +$$

$$+ksu_6[m_0] \cdot rmIdx_6) + r[mIdx_0])/((0.5 \cdot \omega + 1) \cdot ksu_0[mIdx_0]);$$

$$cmem[tdX][tdZ] \leftarrow rmIdx_0;$$

$$r[mIdx_0] \leftarrow rmIdx_0.$$

12. Transition to the next node of the computational grid along the coordinate y: $j = j + 1$.

13. Assigning the loop parameter to the next value $s = s + 1$.
14. Checking the exit condition from the loop $s \leq n_1 + n_2 + n_3 - 3$. If the condition is true, then the transition to step 4 is performed; otherwise, the algorithm exits.

The conducted studies show a significant dependence of the algorithm implementation time for calculating the preconditioner on the ratio of threads in spatial coordinates.

The GeForce GTX 1650 video adapter is used in experimental studies. It has 4 GB of video memory, a core and memory clock frequency of 1485 MHz and 8000 MHz, respectively, and a video memory 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 the number of threads on the Oy axis. Values of the objective function: T_{GPU} is the 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 are chosen taking into account CUDA limitations. For example, the number of threads on the Oy axis varies in the range $[1000, 30000]$. Experimental data analysis for the factor values $X = 1$, $Z = 640$ and $X = 640$, $Z = 1$ shows that the allocated memory is not used when calculating the objective function at the specified points. Therefore, these points must be excluded from regression analysis.

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

$$T_{GPU} = a - b \cdot Y - c \cdot ln(k) - d \cdot ln(Y), \tag{4}$$

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

As a result of experimental data analysis, it is 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.

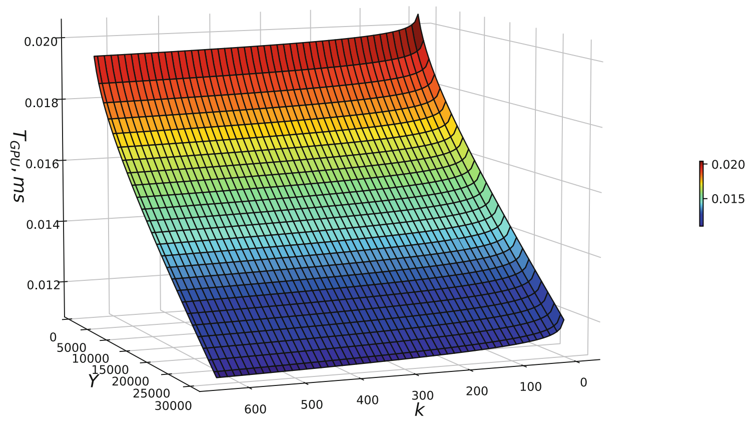

Fig. 4. Surface of the response function $T_{GPU} = f(k, Y)$

The implementation time of one MATM step in terms of 1000 nodes of the computational grid on the GeForce GTX 1650 video adapter is inversely proportional to the number of 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 Ox axis increases, and the number of threads on the Oz axis decreases. Therefore, it is advisable to perform the decomposition of the computational domain in the form of parallelepipeds, in which the size on the Oz axis is minimal, and on the Ox axis is maximal. The choice of the decomposition method of the 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 compute nodes, i.e., the size of the transmitted plane (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 compute nodes on the size of the transmitted plane. The resulting dependence will make it possible to obtain such a decomposition of the computational domain that 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 compute 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 is the time of transmitting the number of elements V.

To determine the dependence of the time of data transmission between compute nodes on the number of transmitted elements, an algorithm and its software implementation in the C language are developed. The program considers 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 10,000 elements.

The developed algorithm is tested on the main computing resource of the Keldysh Institute of Applied Mathematics, namely, 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 compute 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, 768 GB RAM, 2 TB 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 is obtained:

$$T = a + bV, \tag{5}$$

where T is the time of transmitting the number of V elements, ms. The determination coefficient is 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 compute nodes.

4 Conclusions

As a result of the conducted research, an algorithm and a software module implementing it, designed to solve SLAEs that arise during the discretization of spatial-three-dimensional model problems of mathematical physics using the MATM, are developed.

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

It is 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 for the three-dimensional computational grid with the number of nodes up to 10^{11} and the time layers number from 10^4 and more, if we focus on the limitations of computational stability and accuracy of discrete models, applicable to the GPU, is described.

References

1. Bonaduce, A., Staneva, J., Grayek, S., Bidlot, J.-R., Breivik, Ø.: Sea-state contributions to sea-level variability in the European Seas. Ocean Dyn. **70**(12), 1547–1569 (2020). https://doi.org/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. Ocean. Model. **J.** 3, 1–20 (2001). https://doi.org/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. Westerweel, J.T.M., Boersma, B.J.J., Nieuwstadt, F.T.M.: Turbulence. Springer, Cham (2016). https://doi.org/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. Concurr. Comput. Pract. Exp. **33**, 1547–1569 (2021). https://doi.org/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. In: AIAA Aerospace Sciences Meeting, p. 0362 (2018)
7. Nagatake, T., Kunugi, T.: Application of GPU to computational multiphase fluid dynamics. In: 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. Eng. Comput. **35**(3), 1059–1079 (2018). https://doi.org/10.1007/s00366-018-0651-1
9. Sukhinov, A.D., et al.: Data processing of field measurements of expedition research for mathematical modeling of hydrodynamic processes in the Azov Sea. Comput. Conti. Mech. **13**(2), 161–174 (2020). https://doi.org/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. (ed.) PaCT 2021. LNCS, vol. 12942, pp. 166–177. Springer, Cham (2021). https://doi.org/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. Comput. Fluids **92**, 244–252 (2014). https://doi.org/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. In: GPU Solutions to Multi-scale Problems in Science and Engineering, pp. 321-333. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-16405-7_21
13. Konovalov, A.: The steepest descent method with an adaptive alternating-triangular preconditioner. Diff. Equ. **40**, 1018–1028 (2004)
14. Sukhinov, A.I., et al.: Computational aspects of mathematical modeling of the shallow water hydrobiological processes. Num. Methods Program. **21**, 452–469 (2020). https://doi.org/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, p. 630. Apress, Berkeley (2020)