DOI: 10.14529/jsfi260202

Study of Graphics Accelerators Performance for Numerical Solution of the Poisson Equation Using the Chebyshev Method

Sergey V. Polyakov[1], ***Marina A. Kornilina***[1], ***Tatiana A. Kudryashova***[1]

The problem of the efficient numerical solution of boundary value problems for elliptic equations on high-performance computing systems is discussed. In the context of this problem, the use of a special Chebyshev iterative method to solve such problems by the multi-core central processors and graphics accelerators is analyzed. One of the objective functions of such analysis is to compare the time of solving an elliptic boundary value problem by one node with several central processors and one graphics accelerator. The motivation for this consideration is that in many practical applications, the numerical solution of an elliptic problem significantly slows down the overall algorithm. Taking this circumstance into account, as an example, a two-dimensional Dirichlet boundary value problem for the Poisson equation with constant and variable coefficients is considered. Its solution is based on the well-known "cross" scheme on the most detailed Cartesian grid. For this statement, a modified Chebyshev iterative algorithm is proposed. Such approach significantly reduces the requirements to the used RAM. A parallel software implementation of the algorithm, designed for both central processing units (CPUs) and graphics accelerators (GPUs), is examined. The study investigates the algorithm's convergence and the performance of the used computing systems. It analyzes their dependence on the dimension of the grid problem and other parameters. Numerical experiments are used to determine the limits of the algorithm's applicability and to discuss the efficiency of its implementation on CPUs and GPUs.

Keywords: Dirichlet boundary problem for Poisson equation, cross difference scheme, Chebyshev preconditioned method, parallel implementation for central and graphics processor units.

Introduction

The problem of efficient numerical solving boundary value problems for elliptic equations is encountered in many current applications. Fast and reliable numerical algorithms for solving elliptic problems are most in demand in computational fluid dynamics, electrodynamics, strength mechanics, and other fields. The general formulation of the differential problem is as follows:

$$Lu = f, \quad \mathbf{x} \in \Omega \subset E^m, \quad lu = g, \quad \mathbf{x} \in \partial\Omega. \tag{1}$$

Here L is the elliptic differential operator with respect to coordinates x_k of m-dimensional spatial vector $\mathbf{x} = (x_1, \ldots, x_m)$, defined in a compact region Ω of Euclidean space E^m, $u = u(\mathbf{x})$ is the desired function, $f = f(\mathbf{x})$ is specified in Ω function, l is boundary differential-algebraic operator, containing spatial derivatives of no higher than the 1st order, $\partial\Omega$ is border of the region Ω. It is assumed that the input data of problem (1), the boundary for the domain and the coefficients of the equations, have the required properties. Then a solution to problem (1) exists and is unique [1].

The general numerical procedure for solving problem (1) is as follows. First, the domain Ω is discretized. As a rule, a grid Ω_h is built for this purpose with a border $\partial\Omega_h$, based on a system of individual nodes or cells. Then, based on the chosen discretization, an approximation of equations (1) is constructed. Most often, either the finite volume method [2] or the finite element method [3] is used for this aim. The approximation results in a discrete formulation of

[1]Keldysh Institute of Applied Mathematics of RAS, Moscow, Russian Federation

the form:

$$\Lambda_h y_h = f_h, \quad \mathbf{x} \in \Omega_h, \quad l_h y_h = g_h, \quad \mathbf{x} \in \partial\Omega_h. \tag{2}$$

Here Λ_h, y_h, f_h, l_h and g_h are discrete analogues of the above-mentioned continuous objects defined on sets Ω_h and $\partial\Omega_h$. Equations (2) are a system of algebraic linear (quasilinear) equations of the form:

$$\mathrm{A}\mathbf{Y} = \mathbf{F}, \tag{3}$$

here $\mathbf{Y}$ and $\mathbf{F}$ are unknown and given vectors from the real vector space $\mathbb{R}^N$, A is a real matrix from the matrix space $\mathbb{R}^{N\times N}$, N is the number of unknowns which equal to either the number of nodes/cells in used grid for the finite volume method, or the number of basic functions for the finite element method. In the quasi-linear case, it is assumed that the elements of the matrix A and vector $\mathbf{F}$ depend on the components of the vector $\mathbf{Y}$.

In this paper, we take into account many years of experience in solving algebraic problems of the form (3), obtained on the basis of approximations of elliptic equations (1). Accumulated since the early 1950s, this experience has been systematized in extensive literature (see, for example, [4–33]) and summarized in [34]. In addition to many practical methods for accelerating the solving algebraic problems of the form (3), special iterative methods are currently being developed and used, oriented towards parallel computations using high-performance systems with CPUs and GPUs. At the same time, it is known that the use of a large number of computing devices does not yet guarantee a significant acceleration. The reasons for this are the limitations of the architectures of computing systems, including insufficient communication speed between computers, an imbalance in the performance of CPUs and GPUs, as well as different RAM bandwidths.

In the current situation, the following approach can be recommended for most specific applications. For moderately sized problems (within a billion unknowns in the solution vector), it is proposed to use a separate device for its numerical solution. This device can be a computing node equipped with either several multi-core, high-performance CPUs with sufficient shared RAM, or one or more GPUs with sufficient graphics memory. This approach eliminates the problem of network communications slowing down computations, but the relatively low data transfer rate over the buses within the node remains. The second problem can be partially solved by using specialized iterative algorithms focused on parallelization on shared memory.

This paper examines the practical implementation of the proposed approach using the example of solving a two-dimensional Dirichlet problem for the Poisson equation with constant and variable coefficients. This example is the most complex due to the strong sparseness of the matrix of the algebraic system (3). In the two-dimensional case, the convergence rate of the iterations is significantly lower than, for example, when solving the spatially three-dimensional problem (1). The goal of the study is to develop and test an iterative algorithm that would allow us to solve the problems (3) of maximum dimension using a single computing node equipped with a CPU and a GPU.

The article is organized as follows. Section 1 is devoted to the problem formulation and the numerical approach to its solution. In Section 2 the iterative schemes of the Chebyshev method are discussed. Section 3 contains a description of parallel algorithms for solving the discrete problem. Section 4 describes the test problems and analyzes the results of their solution. Conclusion summarizes the study and points directions for further work.

1. Mathematical Problem and Difference Scheme

Let us consider a linear Poisson equation in a finite simply connected rectangular area of the form:

$$\frac{\partial}{\partial x_1}\left(\kappa_1 \frac{\partial u}{\partial x_1}\right) + \frac{\partial}{\partial x_2}\left(\kappa_2 \frac{\partial u}{\partial x_2}\right) = -f\left(x_1, x_2\right), \quad (x_1, x_2) \in \Omega, \tag{4}$$

with Dirichlet boundary conditions:

$$u\left(x_1, x_2\right) = g\left(x_1, x_2\right), \quad (x_1, x_2) \in \partial\Omega. \tag{5}$$

We assume that the elliptic operator used in (4) satisfies the conditions of strict ellipticity at all points. The domain Ω, κ_α coefficients and the function f are quite smooth, the border of the region $\partial\Omega$ is piecewise smooth except for a finite number of points, the boundary function g is continuous. Then the solution of the problem (4), (5) exists and is unique [1].

We will solve the problem (4), (5) numerically using the finite difference method. To do this, we will assume that Ω coincides with an open square $(0,1) \times (0,1)$. Then a uniform Cartesian grid $\bar{\Omega}_h = \bar{\omega}_{x_1} \times \bar{\omega}_{x_2}$, here $\bar{\omega}_{x_\alpha} = \{x_{\alpha,i_\alpha} = i_\alpha h_\alpha = i_\alpha / N_\alpha, \quad i_\alpha = 0, \ldots, N_\alpha\}$, $\alpha = 1, 2$, can be introduced in its closure $\bar{\Omega} = \Omega \cup \partial\Omega$. On this grid, equation (4) is approximated by the well-known "cross" difference scheme supplemented by Dirichlet grid conditions:

$$\begin{gathered}
\frac{1}{\hbar_1}\left\{\kappa_{1,i_1+1/2,i_2}\frac{y_{i_1+1,i_2}-y_{i_1,i_2}}{h_1} - \kappa_{1,i_1-1/2,i_2}\frac{y_{i_1,i_2}-y_{i_1-1,i_2}}{h_1}\right\} + \\
\frac{1}{\hbar_2}\left\{\kappa_{2,i_1,i_2+1/2}\frac{y_{i_1,i_2+1}-y_{i_1,i_2}}{h_2} - \kappa_{2,i_1,i_2-1/2}\frac{y_{i_1,i_2}-y_{i_1,i_2-1}}{h_2}\right\} = f_{i_1,i_2}, \\
i_\alpha = 1, \ldots, N_\alpha - 1, \\
y_{i_1,i_2} = g_{i_1,i_2}, \quad i_\alpha = 0, N_\alpha, \quad \alpha = 1, 2.
\end{gathered} \tag{6}$$

In discrete equations (6), the grid function y_{i_1,i_2} is an approximation to the exact solution on the grid $u_{i_1,i_2} = u\left(x_{1,i_1}, x_{2,i_2}\right)$, $\kappa_{1,i_1\pm1/2,i_2}$ and $\kappa_{2,i_1,i_2\mp1/2}$ are expressions $0.5\left[\kappa_1\left(x_{1,i_1}, x_{2,i_2}\right) + \kappa_1\left(x_{1,i_1\pm1}, x_{2,i_2}\right)\right]$ and $0.5\left[\kappa_2\left(x_{1,i_1}, x_{2,i_2}\right) + \kappa_2\left(x_{1,i_1}, x_{2,i_2\pm1}\right)\right]$, $f_{i_1,i_2} = f\left(x_{1,i_1}, x_{2,i_2}\right)$, $g_{i_1,i_2} = g\left(x_{1,i_1}, x_{2,i_2}\right)$, $\hbar_1$ and $\hbar_2$ are the average steps of the grid, in this case equal to h_1 and h_2.

Before finding a solution of the difference scheme, a well-known transformation of equations (6) to a special form is performed:

$$
\begin{gathered}
d_{i_1,i_2} y_{i_1,i_2} - a_{1,i_1,i_2} y_{i_1-1,i_2} - b_{1,i_1,i_2} y_{i_1+1,i_2} - a_{2,i_1,i_2} y_{i_1,i_2-1} - b_{2,i_1,i_2} y_{i_1,i_2+1} = \varphi_{i_1,i_2}, \\
a_{1,i_1,i_2} = \frac{\hbar_2}{h_1} \kappa_{1,i_1-1/2,i_2}, \quad b_{1,i_1,i_2} = \frac{\hbar_2}{h_1} \kappa_{1,i_1+1/2,i_2}, \\
a_{2,i_1,i_2} = \frac{\hbar_1}{h_2} \kappa_{2,i_1,i_2-1/2}, \quad b_{2,i_1,i_2} = \frac{\hbar_1}{h_2} \kappa_{2,i_1,i_2-1/2}, \\
d_{i_1,i_2} = a_{1,i_1,i_2} + b_{1,i_1,i_2} + a_{2,i_1,i_2} + b_{2,i_1,i_2}, \\
\varphi_{i_1,i_2} = \hbar_1 \hbar_2 f_{i_1,i_2}, \quad i_\alpha = 1, \ldots, N_\alpha - 1, \\
y_{i_1,i_2} = g_{i_1,i_2}, \quad i_\alpha = 0, N_\alpha, \quad \alpha = 1, 2.
\end{gathered}
\tag{7}
$$

As a result of transformation (7), the matrix of system (3) becomes symmetric and positive definite. The portrait of the matrix depends on the ordering of the unknowns in the vector $\mathbf{Y}$ and the order of the equations. With a natural ordering of the view:

$$
\mathbf{Y} = \left(y_{0,0}, y_{1,0}, \ldots, y_{N_1,0},\ y_{0,1}, y_{1,1} \ldots, y_{N_1,1},\ \ldots,\ y_{0,N_2}, y_{1,N_2}, \ldots, y_{N_1,N_2}\right),
$$

the portrait of the matrix has a block tri-diagonal structure (see Fig. 1). This implies the preliminary exclusion of the function's boundary values y_{i_1,i_2} from consideration using the standard dimensionality reduction procedure. Below, we will use precisely this ordering of unknowns and equations.

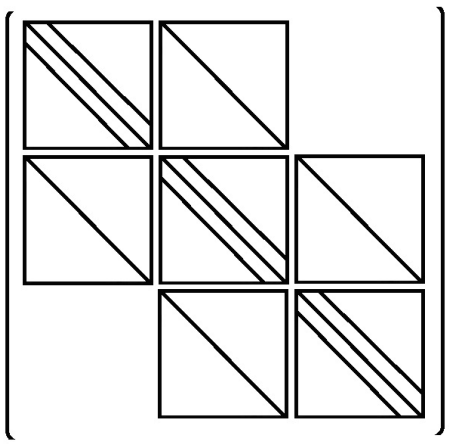

Figure 1. Portrait of the matrix of the system (3)

2. Iterative Algorithms

Let us now describe the proposed iterative approach. It is based on the Chebyshev iterative algorithm [6, 10]. We formulate it using the well-known two-layer iterative scheme [10]:

$$
\mathrm{B}_{s+1} \frac{\mathbf{Y}^{s+1} - \mathbf{Y}^s}{\tau_{s+1}} + \mathrm{A}\mathbf{Y}^s = \mathbf{F}, \quad s = 0, 1, \ldots. \tag{8}
$$

This scheme assumes setting some initial vector $\mathbf{Y}^0$ and calculating subsequent approximations to the solution of problem (3) using equations (8). The transition to the next approximation is carried out by inverting the matrices B_{s+1}. The convergence rate is adjusted by iterative parameters τ_{s+1}. A specific iterative algorithm is determined by the choice of a sequence of transition operators B_{s+1} and iterative parameters τ_{s+1}. It is known (see, for example, [10]) that scheme (8) converges to the solution of system (3) in a finite number of steps if the following conditions are met:

$$\mathrm{A} = \mathrm{A}^T > 0, \quad \mathrm{B}_{s+1} = \mathrm{B}_{s+1}^T > 0, \quad \tau_{s+1} > 0, \quad \mathrm{B}_{s+1} - 0.5\tau_{s+1}\mathrm{A} > 0, \quad s = 0, 1, \ldots. \tag{9}$$

Before solving system (3), a preconditioner matrix M is often chosen, which leads system (3) to a modified form:

$$\mathrm{M}^{-1}\mathrm{A}\mathbf{Y} \equiv \tilde{\mathrm{A}}\mathbf{Y} = \mathrm{M}^{-1}\mathbf{F} \equiv \tilde{\mathbf{F}}. \tag{10}$$

For example, in the iterative Jacobi method, the matrix M coincides with the main diagonal D_{A} of the matrix A. The purpose of such a transformation is to significantly reduce the number of iterations $n = n(\varepsilon)$ required to achieve accuracy $\varepsilon > 0$. Such a decrease occurs due to a change in the spectral properties of the matrix of the system (10). The criterion for convergence of iterations will be considered to be the achievement of a norm in the space $C(\Omega_h)$ of the discrepancy of a given value:

$$||R|| = ||\tilde{\mathbf{F}} - \tilde{\mathrm{A}}\mathbf{Y}|| \le \varepsilon. \tag{11}$$

The Chebyshev iterative method applied to the system (10) implies the use of unit transition matrices ($\mathrm{B}_{s+1} = \mathrm{I}$) and iterative parameters determined by the formulas [6, 10]:

$$\tau_{s+1} = \frac{\tau_0}{1 - \frac{\mu-1}{\mu+1}\cos\frac{\pi(2s+1)}{2n}}, \quad s = 0, \ldots, n-1, \tag{12}$$

here $\tau_0 = 2/\left(\lambda_{min}\left(\tilde{\mathrm{A}}\right) + \lambda_{max}\left(\tilde{\mathrm{A}}\right)\right)$ is the optimal parameter, $\mu = \lambda_{max}\left(\tilde{\mathrm{A}}\right)/\lambda_{min}\left(\tilde{\mathrm{A}}\right)$ is the spectral number of conditionality, $\lambda_{min}\left(\tilde{\mathrm{A}}\right)$ and $\lambda_{max}\left(\tilde{\mathrm{A}}\right)$ are minimum and maximum eigenvalues of the matrix $\tilde{\mathrm{A}}$. In this case, calculations are performed in a series of iterations $n = 2^m$, and the iterative parameters are ordered in a certain way [6].

The Chebyshev method can be combined with the Jacobi method. Such a combined method can be written either in operator form:

$$\frac{\mathbf{Y}^{s+1} - \mathbf{Y}^s}{\tau_{s+1}} + D_{\mathrm{A}}^{-1}\mathrm{A}\mathbf{Y}^s = D_{\mathrm{A}}^{-1}\mathbf{F}, \quad s = 0, 1, \ldots. \tag{13}$$

or in the form of a numerical scheme using the notation used in (7):

$$\begin{gathered}
y_{i_1,i_2}^{s+1} = (1 - \tau_{s+1})\, y_{i_1,i_2}^s + \tau_{s+1} d_{i_1,i_2}^{-1} r_{i_1,i_2}^{s+1}, \\
r_{i_1,i_2}^{s+1} = a_{1,i_1,i_2} y_{i_1-1,i_2}^s + b_{1,i_1,i_2} y_{i_1+1,i_2}^s + a_{2,i_1,i_2} y_{i_1,i_2-1}^s + b_{2,i_1,i_2} y_{i_1,i_2+1}^s + \varphi_{i_1,i_2}, \\
i_\alpha = 1, \ldots, N_\alpha - 1, \quad \alpha = 1, 2, \quad s = 0, 1, \ldots, \\
y_{i_1,i_2}^{s+1} = g_{i_1,i_2}, \quad i_\alpha = 0, N_\alpha, \quad \alpha = 1, 2, \quad s = 0, 1, \ldots.
\end{gathered} \tag{14}$$

Since one of the goals of the work is to apply the proposed iterative method in conditions of limited RAM, we will monitor the required volume for this. In particular, in the case of

problem (4) with the Laplace operator ($\kappa_\alpha \equiv 1$, $\alpha = 1, 2$), matrix storage A is not required when implementing the iterative scheme (13). RAM will be allocated only for vectors $\mathbf{Y}^{s+1}$, $\mathbf{Y}^s$, $\mathbf{F}$, which will amount to $3N$ cells, where $N = (N_1 + 1)(N_2 + 1)$. More generally, when $\kappa_\alpha \neq \text{const}$, $\alpha = 1, 2$, 4 more side diagonals of the matrix A are stored, which will require a total of $7N$ cells.

Then let us consider the issue of computational costs in the implementation of the iterative scheme (13). As is well known (see, for example, [10]), the general number of arithmetic operations for this scheme is $Q_1 = q_1 n(\varepsilon)$, where q_1 is the number of operations required to implement one step of the iterative scheme, $n(\varepsilon)$ is the number of such steps required to achieve accuracy ε. In the case of highly sparse matrices A value $q_1 = O(N)$, and it is not possible to lower it. Value $n(\varepsilon)$ depends on the scheme of the iterative algorithm and its parameters. In the general situation $n(\varepsilon) = O(N\ln(2/\varepsilon))$. This estimate is valid for two-layer schemes with a diagonal transition operator $\mathrm{B}_{s+1} = \mathrm{D}$. However, when using the Chebyshev parameter set or the three-layer Chebyshev method [10], the estimate of the number of iterations $n(\varepsilon)$ is reduced to $O\left(\sqrt{N}\ln(2/\varepsilon)\right)$. This fact explains the choice of this method as the basis for further analysis.

A variety of approaches are used to achieve a more significant acceleration of iteration convergence. The most popular of them are methods such as conjugate and biconjugate gradients [5–16, 20, 24] using Krylov subspaces of various dimensions. These methods are three-layered and automatically estimate the rate of convergence of iterations $O\left(\sqrt{N}\ln(2/\varepsilon)\right)$. When factorized matrices are used as transition operators B_{s+1} in them, for example, those close to Gaussian decomposition $\mathrm{A} = \mathrm{LU}$, an even higher convergence rate (of the order $O\left(N^{1/4}\ln(2/\varepsilon)\right)$ and lower) is achieved. However, the price for this acceleration is an increase in the required RAM and the use of additional information about the spectrum of the matrix A. In the most difficult-to-implement variants (see, for example, [20, 24]), not only additional RAM is required, but there is also an increase in labor intensity (increase q_1), which does not adapt well to the architectures of GPU systems.

It is also worth to mention the methods of variable directions [10], which have a maximum convergence rate (of the order), but are suitable only for solving problems in rectangular areas on Cartesian grids. Similar asymptotics can be obtained using multigrid iterative methods [31–33]. However, even here, complex logic and the re-sorting of unknowns and equations are difficult to implement on GPU systems.

Without disputing the advantages of the above methods, we will not consider them in this paper due to the available objective function – saving computing RAM when solving large-dimensional problems and simplicity of parallel implementation on hybrid nodes with CPU and GPU.

Another possibility to accelerate the convergence of iterations within the framework of the Chebyshev approach is the use of the alternating triangular method [4, 6, 10, 15, 28, 30, 35–40]. Its formulation is obtained if a factorized matrix of the form is used as the preconditioner matrix $\mathrm{M} = (\mathrm{D_A} + \mathrm{A_-})\,\mathrm{D_A^{-1}}\,(\mathrm{D_A} + \mathrm{A_+})$, here $\mathrm{A_-}$ and $\mathrm{A_+}$ are lower and upper triangular matrices, which together with $\mathrm{D_A}$ make up the original matrix $\mathrm{A} = \mathrm{D_A} + \mathrm{A_-} + \mathrm{A_+}$. Considering that the inversion of the matrix factors M to obtain M^{-1} does not increase the asymptotic value q_1 (it still amounts to $O(N)$), the scheme of the method can also be written as:

$$\left(\mathrm{I} + \mathrm{D_A^{-1}A_-}\right)\left(\mathrm{I} + \mathrm{D_A^{-1}A_+}\right)\frac{\mathbf{Y}^{s+1} - \mathbf{Y}^s}{\tau_{s+1}} + \mathrm{D_A^{-1}A}\mathbf{Y}^s = \mathrm{D_A^{-1}}\mathbf{F}, \quad s = 0, 1, \ldots. \tag{15}$$

Scheme (15) is transformed into a four-stage algorithm that requires additional storage of one vector (the residual vector $\mathbf{R}^s = \mathrm{D}_\mathrm{A}^{-1}\mathbf{F} - \mathrm{D}_\mathrm{A}^{-1}\mathrm{A}\mathbf{Y}^s$), compared with algorithm (13). Then the total required memory is either $4N$, or $8N$ cells. The convergence rate increases, including through the use of Chebyshev acceleration. Scheme (15) will be used mainly for the case of variable coefficients in equation (4).

3. Parallel Realization

The above-mentioned iterative schemes (13) and (15) have been adapted to parallel computing. In this context, scheme (13) has been discussed in many papers, scheme (15) has been studied in [35–40]. In this paper, the issue of software implementation of iterative schemes (13) and (15) on a node with multi-core processors and on a graphics accelerator was considered. As a result, the following solutions were implemented.

OpenMPI was chosen as the parallel programming standard for nodes with CPUs [41]. The motivation for this solution was that the most efficient calculations on shared memory are implemented when each core interacts with its individual RAM block. The OpenMPI utilities provide such work discipline due to their focus on distributed computing. In addition, the modern implementation of OpenMPI takes into account the hardware features of computing nodes, up to the automatic allocation of NUMA devices [42]. When using other computing standards (POSIX Threads [43], OpenMP [44]), these problems have to be solved independently and, as a rule, inefficiently.

The parallelization technique was based on the area separation method [45]. At the same time, the topology of the initial problem was taken into account, namely, the two-dimensionality and configuration of the area Ω. As a result, the difference grid was split into two spatial directions and compared with a two-dimensional grid of computers, which were CPU cores. The implementation of iterative schemes (13) and (15) did not require the convolution of the grid function into a linear vector. The final code was developed in the ANSI C language.

When developing parallel implementations of iterative schemes (13) and (15) for GPU computing, the OpenCL parallel programming standard was used [46]. This made it possible to use a single code for calculations on NVidia and AMD graphics accelerators. At the same time, the main program was developed in C++ and used thread separation based on the OpenMP standard to speed up work. In fact, within the framework of the general iterative scheme implemented on the CPU, time-consuming operations of matrix-vector multiplication and scalar product calculations were performed on the GPU. This approach has been proposed and successfully tested in many papers (see, for example, [47–55]). The parallelization technique in this case required convolution of the tabular grid function $y_h = \{y_{i_1,i_2}\}$ in a linear vector $\mathbf{Y}$. It was divided into adjacent blocks in accordance with the number of GPU multiprocessors. If there are several GPUs, you can distribute the implementation of each iteration between them. However, this will not be efficient enough when using the PCI Express bus.

4. Testing of Parallel Iterative Algorithms

Three tasks were used to test the proposed parallel iterative algorithms. The task data is shown in Tab. 1.

Table 1. The initial data of the test tasks

Task No.	View of coefficients κ_α	The exact solution formula
1	$\kappa_\alpha \equiv 1, \alpha = 1, 2$	$u = \sin(\pi k_1 x_1) \sin(\pi k_2 x_2)$
2	$\kappa_\alpha \equiv 1, \alpha = 1, 2$	$u = \sin(\pi k_1 x_1) \sin(\pi k_2 x_2) \cosh(a(x_1 - x_2))$
3	$\kappa_\alpha = 1 + b \cdot \kappa(x_1)\kappa(x_2),\ \alpha = 1, 2$ $\kappa(x) = \exp\left[-(x - 0.5)^4 / c^4\right]$	$u = \sin(\pi k_1 x_1) \sin(\pi k_2 x_2) \cosh(a(x_1 - x_2))$

The right-hand side of equation (4) was calculated based on a given exact solution:

$$f(x_1, x_2) = -\frac{\partial}{\partial x_1}\left(\kappa_1 \frac{\partial u}{\partial x_1}\right) - \frac{\partial}{\partial x_2}\left(\kappa_2 \frac{\partial u}{\partial x_2}\right). \tag{16}$$

In tasks 1–3, the parameters are $k_1 = 2$, $k_2 = 2$; in tasks 2 and 3 $a = 2$, in task 3 parameter b is ranged from 1 to 10^3, parameter $c = 0.1$. The general view of the solutions to problem 1 is shown in Fig. 2, tasks 2 and 3 are shown in Fig. 3 and Fig. 4 shows the dependence of the coefficients on the coordinate at a fixed value of the coordinate $x_2 = 0.5$. The dependence κ_α on the coordinate x_2 was similar. The type of exact solutions determined their values at the boundary: $g(x_1, x_2) \equiv 0$.

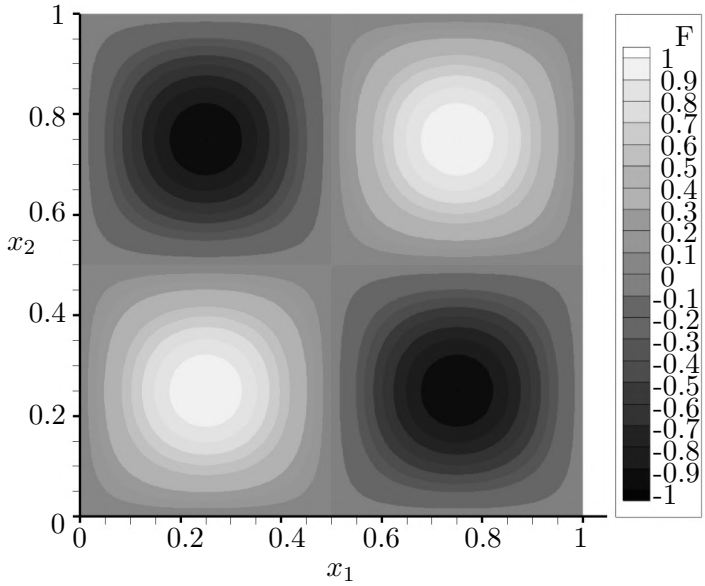

Figure 2. General view of the solution of task 1

Let us explain the choice of test tasks. The first two tasks use the Laplace operator as a differential operator. This is convenient because all the parameters of the spectrum at the differential and discrete levels are known for this operator. At the same time, task 1 corresponds to the case of separation of variables, in which the solution is represented by a finite Fourier series. In task 2, separation of variables is impossible, and the Fourier series becomes infinite. The task 3 differs from task 2 in the presence of dependencies of coefficients κ_α simulating an insert made of a material with significantly different properties. The latter allows us to study the influence of the number of conditionality of the corresponding matrix of the system (3) on the convergence of iterative algorithms. It should also be noted that even fast convergence of

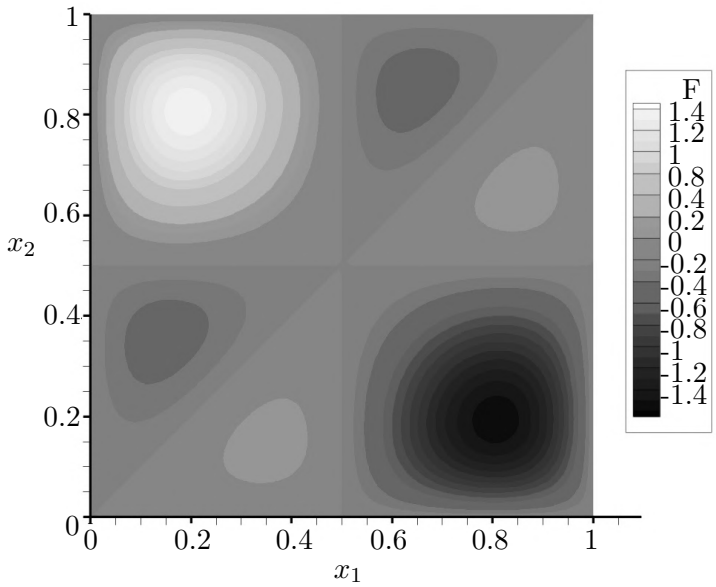

Figure 3. General view of solving tasks 2 and 3

iterations is not a guarantee of obtaining a solution to the numerical scheme with an accuracy corresponding to the order of approximation.

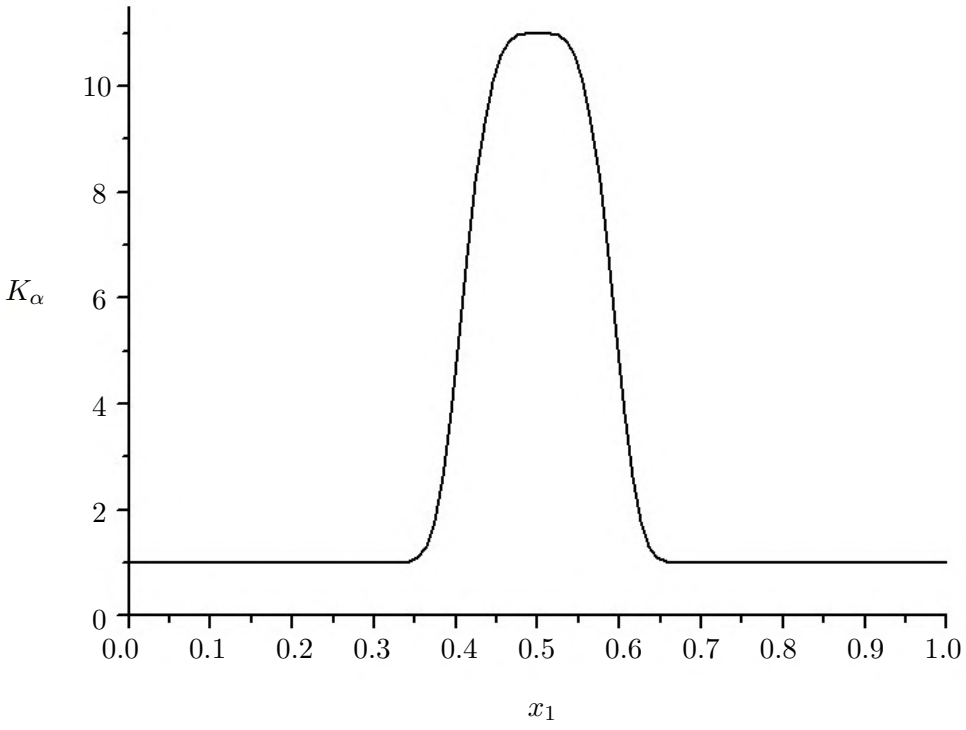

Figure 4. Dependence κ_α from x_1 for $x_2 = 0.5$ in task 3

During the test calculations, the IMM23, K60GPU, K120AMD, and K120NVI computing systems installed at the Keldysh IPM RAS Supercomputer Center for Collective Use were used. The characteristics of the systems used are shown in Tab. 2.

Table 2. Characteristics of the computing systems used

Name	Quantity CPU	Model and characteristics CPU	Quantity GPU	Model and characteristics GPU
Server IMM23	2	AMD EPYC 9124, 1500 MHz, 16 Cores, 32 Threads, Mem 128 Gb	1	gfx908 (AMD Instinct MI100 32Gb, PCIe Gen4), 1502 MHz, 120 CU, 7680 stream cores
Node K60GPU	2	Intel(R) Xeon(R) Gold 6142, 2600 MHz, 16 Cores, 32 Threads, Mem 384 Gb	4	GV100 (Nvidia V100 32Gb, PCIe Gen4), 1230–1380 MHz, 84 SM, TMU 640, CUDA cores 5120
Node K120AMD	2	Intel(R) Xeon(R) Gold 6444Y, 4000 MHz, 16 Cores, 32 Threads, Mem 512 Gb	4	gfx90a (AMD Instinct MI210 64Gb HBM2e, PCIe Gen4), 1700 MHz, 104 CU, 6656 stream cores
Node K120NVI	2	Intel(R) Xeon(R) Gold 6444Y, 4000 MHz, 16 Cores, 32 Threads, Mem 512 Gb	4	GH100 (NVidia H100 80GB, PCIe Gen5), 1095-1755 MHz, 114 SM, TMU 456, CUDA cores 14592

Let us consider the test results for solving task 1. Calculations were performed using both iterative schemes on the K60GPU cluster node and on the IMM23 server on central and graphics processors. The calculations used codes implemented in the ANSI C/C++ language using the OpenMPI and OpenCL parallel computing standards. The purpose of the calculations was to demonstrate Chebyshev acceleration of convergence of implemented iterative methods. Table 3 and Table 4 show the results of the calculations performed. For square grids, the number of nodes was chosen equal 2^k. When analyzing the above results, it is necessary to take into account that usually in Chebyshev methods, the accuracy of iteration is consistent with the approximation error of the difference scheme. In this case, we set the accuracy, which is about two decimal orders of magnitude higher than the expected accuracy of the difference scheme.

The data in Tab. 3 show that the applied variant of the conditional Chebyshev method (13) has a root dependence on the total number of grid elements, that is $n(\varepsilon) = O\left(\sqrt{n_1 \cdot n_2} \cdot \ln[2/\varepsilon]\right)$. This result is also illustrated in Fig. 5, where the dependence of the normalized number of iterations is shown $\bar{n}(\varepsilon) = n(\varepsilon)\left(\sqrt{n_1 \cdot n_2} \cdot \ln[2/\varepsilon]\right)^{-1}$.

When analyzing the results shown in Tab. 3, it can be seen that on the 2048×2048 and 4096×4096 grids, the numerical solution was obtained with approximately the same accuracy. Here the reason lies in reaching the threshold of machine accuracy in the second case. This leads to the fact that the right-hand side in equations (7) is calculated with large errors. In this case, it is possible to correct the situation by artificially multiplying the vector of the right side by a coefficient of the form 2^k. The resulting solution is then divided by this coefficient. This technique, let us call it "binary scaling", allows you to get a solution with the required accuracy even on detailed grids. Below, it is used in the calculations of tasks 1–3 for both iterative schemes.

Table 3. Test results for solving task 1 according to scheme (13)

Grid size $n_1 \times n_2$	Number of processes and configuration	Calculation time (sec)	Number of iterations, $n(\varepsilon)$	Iteration convergence criterion, ε	Accuracy of the numerical solution, Δ
K60GPU, CPU, OpenMPI					
128×128	1 (1×1)	0.132	616	1e-6	3.937e-06
256×256	1 (1×1)	0.912	1408	5e-7	1.297e-04
512×512	4 (2×2)	1.926	3362	1e-7	8.117e-06
1024×1024	16 (4×4)	4.752	7681	5e-8	8.773e-06
2048×2048	32 (4×8)	28.437	17438	1e-8	7.152e-07
4096×4096	32 (4×8)	387.560	36862	5e-9	7.646e-07
K60GPU, GPU, OpenCL					
1024×1024	-	0.334	7681	5e-8	8.773e-06
2048×2048	-	2.159	17438	1e-8	7.152e-07
4096×4096	-	38.392	36862	5e-9	7.646e-07
IMM23, CPU, OpenMPI					
128×128	1 (1×1)	0.056	616	1e-6	3.937e-06
256×256	1 (1×1)	0.527	1408	5e-7	1.297e-04
512×512	4 (2×2)	1.297	3362	1e-7	8.117e-06
1024×1024	16 (4×4)	3.153	7681	5e-8	8.773e-06
2048×2048	32 (4×8)	20.921	17438	1e-8	7.152e-07
4096×4096	32 (4×8)	256.834	36862	5e-9	7.646e-07
IMM23, GPU, OpenCL					
1024×1024	-	0.518	7681	5e-8	8.773e-06
2048×2048	-	2.803	17438	1e-8	7.152e-07
4096×4096	-	47.639	36862	5e-9	7.646e-07

The data in Tab. 4 demonstrate a slower convergence of the iterative scheme (15). The reason is that this scheme used a series of Chebyshev parameters of length 2. In this situation, the root dependence of the number of iterations $n(\varepsilon)$ changes to a linear one as the dimension of problem (3) increases (see Fig. 5). For longer series, scheme (15) slows down even more. As a result, its variant with two iterative parameters is further used, which are calculated using formulas (12) with the selected parameter τ_0.

It should also be noted that when computing on processors from different manufacturers, the number of iterations may vary slightly due to different implementations of floating-point arithmetic. This is clearly seen from the data in Tab. 4. Since a more complex chain of calculations is implemented in the iterative scheme (15), rounding errors have a stronger effect on detailed grids when calculating task 1.

Let us now consider the results of solving task 2 using scheme (13) on large grids. The results were obtained on all the above-mentioned computers (see Tab. 5) using parallel technologies OpenMP for CPU and OpenCL for GPU. The calculations were performed with the parameter $\varepsilon = 10^{-6}$. The number of iterations was $n(\varepsilon) = 929, 1773, 3377, 4925, 5770, 7186, 8596, 10001, 11397$ for the grids specified in Tab. 5.

Table 4. Test results for solving task 1 according to scheme (15)

Grid size $n_1 \times n_2$	Number of processes and configuration	Calculation time (sec)	Number of iterations, $n(\varepsilon)$	Iteration convergence criterion, ε	Parameter τ_0	Accuracy of the numerical solution, Δ
K60GPU, CPU, OpenMPI						
128×128	1 (1×1)	0.643	1092	1e-6	0.94	1.644e-04
256×256	1 (1×1)	7.622	4526	5e-7	0.97	2.761e-05
512×512	4 (2×2)	125.571	20646	1e-7	0.98	7.958e-06
1024×1024	16 (4×4)	629.533	94353	5e-8	0.98	2.180e-06
2048×2048	32 (4×8)	2536.435	416113	1e-8	0.99	5.904e-07
4096×4096	32 (4×8)	4980.037	1738673	5e-9	0.99	9.507e-08
K60GPU, GPU, OpenCL						
1024×1024	-	212.522	94353	5e-8	0.98	2.180e-06
2048×2048	-	609.667	416113	1e-8	0.99	5.904e-07
4096×4096	-	352.039	1738673	5e-9	0.99	9.507e-08
IMM23, CPU, OpenMPI						
128×128	1 (1×1)	0.375	1092	1e-6	0.94	1.644e-04
256×256	1 (1×1)	6.488	4526	5e-7	0.97	2.761e-05
512×512	4 (2×2)	30.236	20646	1e-7	0.98	7.958e-06
1024×1024	16 (4×4)	139.570	94352	5e-8	0.98	2.180e-06
2048×2048	32 (4×8)	2571.582	416119	1e-8	0.99	5.904e-07
4096×4096	32 (4×8)	5933.479	1733057	5e-9	0.99	9.572e-08
IMM23, GPU, OpenCL						
1024×1024	-	33.623	94352	5e-8	0.98	2.180e-06
2048×2048	-	453.943	416119	1e-8	0.99	5.904e-07
4096×4096	-	951.392	1733057	5e-9	0.99	9.572e-08

Analyzing the data in Tab. 5, we see that modern graphics accelerators can solve two-dimensional elliptic boundary value problems of sufficiently large dimensions, making them suitable for most current applications. Among the GPUs we used, the GH100 supports the maximum elliptic problem dimension. For variable coefficients, it reaches 32768×32768. Table 5 also shows that the main limiting factor in computational speed is RAM bandwidth. This applies to both CPUs and GPUs. The relative speedup achieved using a GPU versus a CPU node is shown in Fig. 6 for all four systems. They demonstrate that as the problem size increases, the GPU increasingly outperforms the CPU node.

Let us now discuss the application of scheme (15) for the case of variable coefficients of equation (4) using the example of solving task 3. The calculations were performed on the IMM23 server. Table 6 presents the data obtained on grids of different dimensions and for different values b. As expected, the convergence of scheme (15) when using large grids deteriorates with increasing b. This is again due to rounding errors associated with the calculations of matrix elements. Multigrid technology will likely help overcome this dependence.

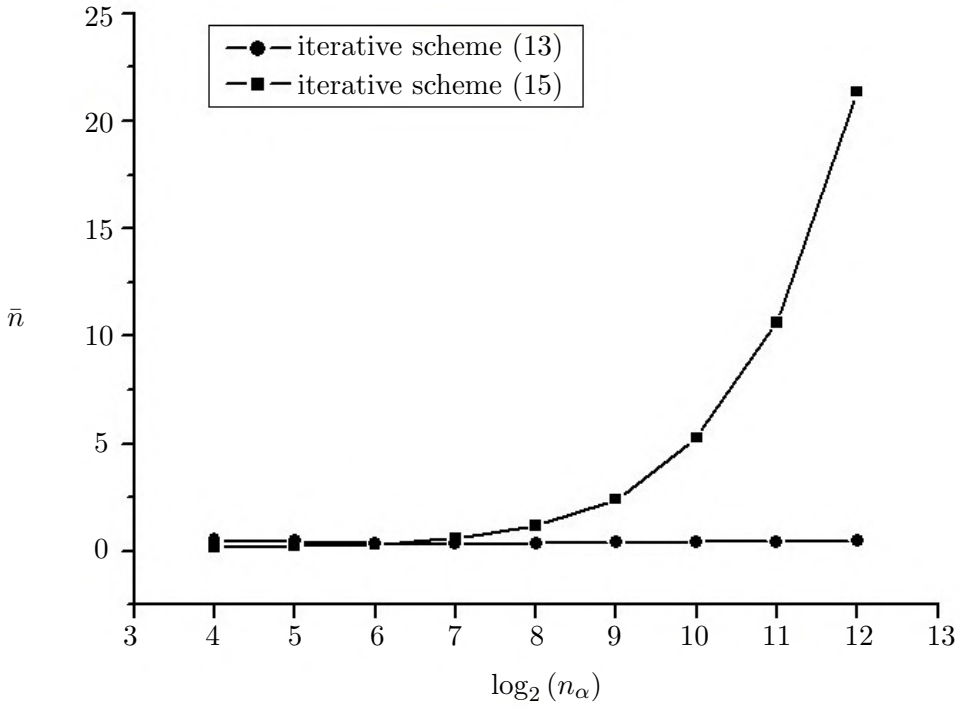

Figure 5. Dependence of the normalized number of iterations on the grid size when solving task 1

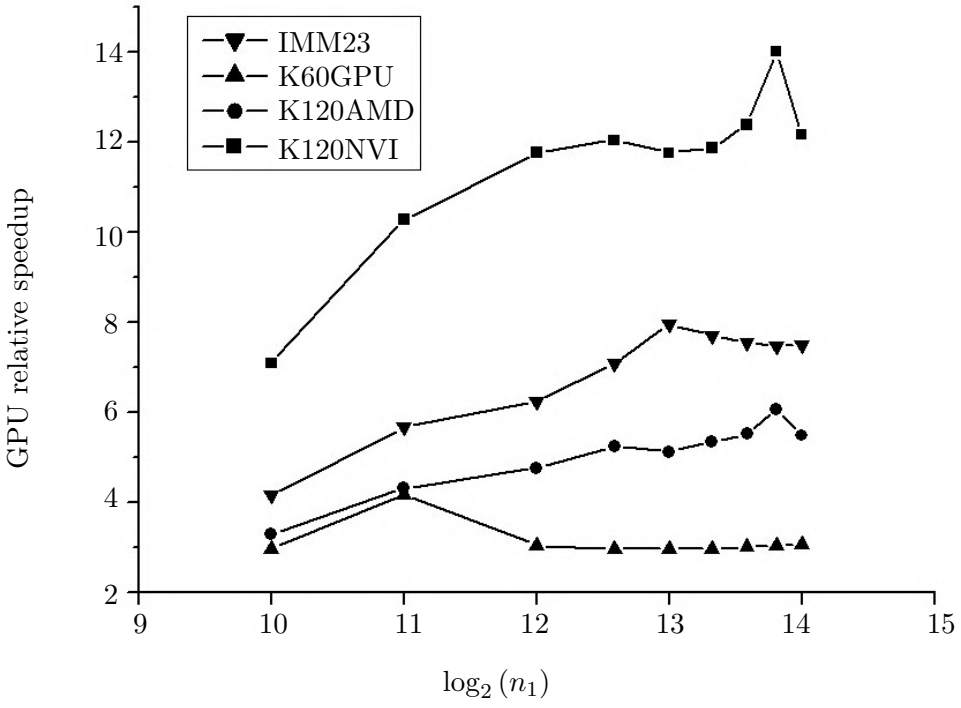

Figure 6. Relative acceleration ($S = GPU\_time/CPU\_time$) dependencies on grid size, obtained when solving task 2

Table 5. Test results for solving task 2 according to scheme (13)

Grid size $n_1 \times n_2$	Computer type and technology	Time (sec) Server IMM23	Time (sec) Node K60GPU	Time (sec) Node K120AMD	Time (sec) Node K120NVI
1024×1024	CPU, OpenMP	3.408	1.567	1.734	1.644
1024×1024	**GPU, OpenCL**	**0.821**	**0.529**	**0.528**	**0.232**
2048×2048	CPU, OpenMP	23.708	13.413	13.159	12.700
2048×2048	**GPU, OpenCL**	**4.185**	**3.224**	**3.055**	**1.236**
4096×4096	CPU, OpenMP	178.904	70.224	99.038	97.019
4096×4096	**GPU, OpenCL**	**28.686**	**23.118**	**20.797**	**8.246**
6144×6144	CPU, OpenMP	593.188	221.443	311.050	317.346
6144×6144	**GPU, OpenCL**	**83.707**	**74.891**	**59.440**	**26.355**
8192×8192	CPU, OpenMP	1298.320	460.028	655.477	641.901
8192×8192	**GPU, OpenCL**	**163.040**	**155.085**	**128.100**	**54.596**
10240×10240	CPU, OpenMP	2381.910	890.509	1253.330	1254.130
10240×10240	**GPU, OpenCL**	**309.563**	**300.987**	**234.493**	**105.671**
12288×12288	CPU, OpenMP	4188.890	1579.790	2215.260	2266.440
12288×12288	**GPU, OpenCL**	**554.968**	**524.636**	**402.219**	**182.974**
14336×14336	CPU, OpenMP	6604.910	2490.220	3800.570	4047.06
14336×14336	**GPU, OpenCL**	**884.013**	**819.417**	**626.890**	**288.752**
16384×16384	CPU, OpenMP	9494.430	3723.690	4986.160	5222.420
16384×16384	**GPU, OpenCL**	**1266.540**	**1218.240**	**909.693**	**429.445**

Table 6. Test results for solving task 3 according to scheme (15)

Grid size $n_1 \times n_2$	Number of processes and configuration	Calculation time (sec)	Number of iterations, $n(\varepsilon)$	Iteration convergence criterion, ε	Parameter τ_0	Accuracy of the numerical solution, Δ
$b = 1$						
128×128	1 (1×1)	0.617	1763	1e-6	0.93	1.872e-04
256×256	4 (2×2)	2.710	7428	5e-7	0.96	4.493e-05
512×512	16 (4×4)	13.126	35564	1e-7	0.98	1.191e-05
1024×1024	32 (4×8)	114.466	147312	5e-8	0.99	2.871e-06
2048×2048	32 (4×8)	4022.841	650993	1e-8	0.99	7.285e-07
2048×2048	**GPU, OpenCL**	**710.624**	**650993**	**1e-8**	**0.99**	**7.285e-07**
$b = 8$						
128×128	1 (1×1)	0.454	1248	1e-6	0.88	3.939e-04
256×256	4 (2×2)	1.901	5210	5e-7	0.92	1.433e-04
512×512	16 (4×4)	9.349	25633	1e-7	0.97	2.264e-05
1024×1024	32 (4×8)	82.100	107287	5e-8	0.98	7.278e-06
2048×2048	32 (4×8)	2986.452	484223	1e-8	0.98	1.655e-06
2048×2048	**GPU, OpenCL**	**535.015**	**484223**	**1e-8**	**0.98**	**1.655e-06**
$b = 128$						
128×128	1 (1×1)	0.447	1278	1e-6	0.79	2.240e-03
256×256	4 (2×2)	1.838	5046	5e-7	0.74	6.291e-04
512×512	16 (4×4)	8.959	24638	1e-7	0.77	1.499e-04
1024×1024	32 (4×8)	80.111	104803	5e-8	0.77	4.052e-05
2048×2048	32 (4×8)	2958.324	478542	1e-8	0.77	9.888e-06
2048×2048	**GPU, OpenCL**	**529.501**	**478542**	**1e-8**	**0.77**	**9.888e-06**
$b = 1024$						
128×128	1 (1×1)	0.482	1340	1e-6	0.72	5.722e-03
256×256	4 (2×2)	1.833	5040	5e-7	0.71	1.515e-03
512×512	16 (4×4)	7.437	20228	1e-7	0.76	3.960e-04
1024×1024	32 (4×8)	65.711	85708	5e-8	0.77	1.066e-04
2048×2048	32 (4×8)	2461.501	398062	1e-8	0.77	2.594e-05
2048×2048	**GPU, OpenCL**	**430.032**	**398062**	**1e-8**	**0.77**	**2.594e-05**

Conclusion

The problem of effective numerical solution of boundary value problems for elliptic equations on central processors and graphics accelerators is considered. The two-dimensional Dirichlet boundary value problem for the Poisson equation with constant and variable coefficients was chosen as an example. For this formulation, it is proposed to use the classical "cross" difference scheme on a Cartesian grid and the modified Chebyshev iterative method. Based on the Chebyshev method, two iterative schemes using the Jacobi transition operator or alternating triangular factorization have been developed. For these schemes, parallel software implementations are presented, made for both central processors and graphics accelerators. The convergence of iterative schemes and the performance of computing systems have been studied depending on the problem dimension and other parameters. The limits of applicability of the developed approach have been determined in numerical experiments. The main conclusion of the work is that in many applications it is sufficient to allocate one calculator for the numerical solution of an elliptic boundary value problem on a grid of moderate volume. This computing unit can be a node with either multiple CPUs or multiple GPUs. The choice of a specific configuration is determined by the complexity of the task and the availability of the necessary resources to solve it.

References

1. Keldysh, M.V.: On the solvability and stability of the Dirichlet problem. Advances in Mathematical Sciences (8), 171–231 (1941). (in Russian)

2. Eymard, R., Gallouet, T.R., Herbin, R.: The finite volume method. Handbook of Numerical Analysis 7, 713–1020 (2000). https://doi.org/10.1016/S1570-8659(00)07005-8

3. Zienkiewicz, O.C., Taylor, R.L., Zhu, J.Z.: The finite element method: Its basis and fundamentals. Butterworth-Heinemann, Oxford (2013). https://doi.org/10.1016/C2009-0-24909-9

4. Axelsson, O.: A generalazed SSOR method. BIT 12, 443–467 (1972). https://doi.org/10.1007/BF01932955

5. Marchuk, G.I., Kuznetsov, Yu.A.: Iterative methods and quadratic functionals. Nauka. Sib. otdel., Novosibirsk (1972). (in Russian)

6. Bakhvalov, N.S.: Numerical methods: Analysis, algebra, ordinary differential equations. MIR Publishers, Moscow (1977).

7. Meijerink, J.A., van der Vorst, H.A.: An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. Math. Comp. 31(137), 148–162 (1977). https://doi.org/10.1090/S0025-5718-1977-0438681-4

8. Gustafsson, I.: A class of first order factorization methods. BIT 18, 142–156 (1978). https://doi.org/10.1007/BF01931691

9. Ortega, J.M.: Introduction to parallel and vector solution of linear systems. Plenum Press, New York (1988). https://doi.org/10.1007/978-1-4899-2112-3

10. Samarskii, A.A., Nikolaev, E.S.: Numerical methods for grid equations. Birkhauser Verlag, Basel-Boston-Berlin (1989). https://doi.org/10.1007/978-3-0348-9272-8

11. Il'in, V.P.: Iterative incomplete factorization methods. World Sci. Publ., Singapore (1992). https://doi.org/10.1142/1677

12. Van der Vorst, H.A.: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM Journal on Scientific Computing 13, 631–644 (1992). https://doi.org/10.1137/0913035

13. Axelsson, O.: Iterative solution methods. Cambridge Univ. Press, Cambridge (1994). https://doi.org/10.1017/CBO9780511624100

14. Kaporin, I.E.: New convergence results and preconditioning strategies for the conjugate gradient method. J. Numer. Linear. Algebra Applic. 1(2), 179–210 (1994). https://doi.org/10.1002/nla.1680010208

15. Kaporin, I.E.: High quality preconditioning of a general symmetric positive definite matrix based on its UTU+UTR+RTU-decomposition. J. Numer. Linear. Algebra Applic. 5(6), 483–509 (1998). https://doi.org/10.1002/(SICI)1099-1506(199811/12)5:6<483::AID-NLA156>3.0.CO;2-7

16. Axelsson, O., Kaporin, I.: On the sublinear and superlinear rate of convergence of conjugate gradient methods. Numerical Algorithms (25), 1–22 (2000). https://doi.org/10.1023/A:1016694031362

17. Milyukova, O.Yu.: Parallel approximate factorization method for solving discrete elliptic equations. Parallel Computing 27(10), 1365–1379 (2001). https://doi.org/10.1016/S0167-8191(01)00092-8

18. Higham, N.J.: Accuracy and stability of numerical algorithms. SIAM, Philadelphia (2002). http://doi.org/10.1137/1.9780898718027

19. Saad, Y.: Iterative methods for sparse linear systems, 2nd ed. SIAM, Philadelphia (2003). http://doi.org/10.1137/1.9780898718003

20. Van der Vorst, H.A.: Iterative Krylov methods for large linear systems. Cambridge Univ. Press, Cambridge (2003). https://doi.org/10.1017/CBO9780511615115.015

21. Il'in, V.P.: Finite element methods and technologies. ICM&MG SBRAS Publ., Novosibirsk, (2007). (in Russian)

22. Knight, P.: The Sinkhorn-Knopp algorithm: convergence and applications. SIAM J. Matrix. Anal. Appl. 30(1), 261–275 (2008). https://doi.org/10.1137/060659624

23. Walkert, H.F., Ni, P.: Anderson acceleration for fixed-point iterations. SIAM J. on Num. Analysis 49(4), 1715–1735 (2011). https://doi.org/10.1137/10078356X

24. Liesen, J., Strakos, Z.: Krylov subspace methods, principles and analysis. Oxford Univ. Press, Oxford (2013). https://doi.org/10.1093/acprof:oso/9780199655410.001.0001

25. Elman, H.C., Silvester, D.J., Wathen, A.J.: Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Numerical mathematics and scientific computation. 2nd ed. Oxford Univ. Press, Oxford (2014). https://doi.org/10.1093/acprof:oso/9780199678792.001.0001

26. Olshanskii, M.A., Tyrtyshnikov, E.E.: Iterative methods for linear systems theory and applications. SIAM, Philadelphia (2014). https://doi.org/10.1137/1.9781611973464

27. Pratara, P.P., Suryanarayana, P., Pask, J.E.: Anderson acceleration of the Jacobi iterative method. An efficient alternative to Krylov methods for large, sparse linear systems. J. Comput. Phys. 306, 43–54 (2016). https://doi.org/10.1016/j.jcp.2015.11.018

28. Vabishchevich, P.N., Zakharov, P.E.: Alternating triangular schemes for convection-diffusion problems. Comput. Math. Math. Phys. 56(4), 576–592 (2016). https://doi.org/10.7868/S0044466916040165

29. Anzt, H., Huckle, T.K., Brackle, J., Dongarra, J.: Incomplete sparse approximate inverses for parallel preconditioning. Parallel Computing 71, 1–22 (2018). https://doi.org/10.1016/j.parco.2017.10.003

30. Vabishchevich, P.N.: Alternating triangular schemes for second-order evolution equations. Comput. Math. Math. Phys. 59(2), 266–274 (2019). https://doi.org/10.1134/S0965542519020155

31. Fedorenko, R.P.: Relaxation method for solving difference elliptic equations. Zh. Vych. Matem. i Matem. Fiz. 1(5), 922–927 (1961). (in Russian)

32. Brandt, A.: Guide to multigrid development. In: Hackbusch, W., Trottenberg, U. (eds.) Multigrid Methods. Lect. Notes in Matem., vol. 960., pp. 220–312. Springer, Berlin, Heidelberg (1982). https://doi.org/10.1007/BFb0069930

33. Xu, J., Zikatanov, L.: Algebraic multigrid methods. Acta Numerica, 26, 591–721 (2017). https://doi.org/10.1017/S0962492917000083

34. Il'in, V.P.: Iterative preconditioned methods in Krylov spaces: trends of the 21st Century. Comput. Math. Math. Phys. 61(11), 1750–1775 (2021). https://doi.org/10.1134/S0965542521110099

35. Milyukova, O.Yu.: A parallel version of the generalized alternating triangular method for elliptic equations. Comput. Math. Math. Phys. 38(12), 1922–1932 (1998). (in Russian)

36. Milyukova, O.Yu.: Parallel versions of the alternating triangular method for solving three-dimensional elliptic equations. Comput. Math. Math. Phys. 42(10), 1472–1481 (2002). (in Russian)

37. Kaporin, I.E.: Using Chebyshev polynomials and approximate inverse triangular factorizations for preconditioning the conjugate gradient method. Comput. Math. Math. Phys. 52(2), 169–193 (2012). https://doi.org/10.1134/S0965542512020091

38. Milyukova, O.Yu.: Combination of numerical and structured approaches to the construction of a second-order incomplete triangular factorization in parallel preconditioning methods. Comput. Math. Math. Phys. 56(5), 699–716 (2016). https://doi.org/10.1134/S096554251605016X

39. Milyukova, O.Yu.: MPI+OpenMP parallel implementation of conjugate gradient method with factored implicit preconditioners. Math. Models Comput. Simul. 14(3), 367–380 (2022). https://doi.org/10.1134/S2070048222030103

40. Milyukova, O.Yu.: Some ways of parallel implementation of the conjugate gradient method with an implicit factorized preconditioner. Math. Models Comput. Simul. 16(4), 638–653 (2024). https://doi.org/10.1134/S2070048224700285

41. Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/, accessed: 2026-05-20

42. Majo, Z., Gross, T.R.: Memory System Performance in a NUMA Multicore Multiprocessor. http://people.inf.ethz.ch/zmajo/publications/11-systor.pdf (2011), accessed: 2026-05-20

43. Barney, B.: POSIX Threads Programming. https://computing.llnl.gov/tutorials/pthreads/, accessed: 2026-05-20

44. Specifications – OpenMP. https://www.openmp.org/specifications, accessed: 2026-05-20

45. Domain Decompositon Methods (DDM). https://www.ddm.org/, accessed: 2026-05-20

46. OpenCL – The Open Standard for Parallel Programming of Heterogeneous Systems. https://www.khronos.org/opencl/, accessed: 2026-05-20

47. Kuo, F.-A., Smith, M.R., Hsieh, Ch.-W., *et al.*: GPU acceleration for general conservation equations and its application to several engineering problems. Computers & Fluids 45(1), 147–154 (2011). https://doi.org/10.1016/j.compfluid.2010.10.007

48. Niemeyer, K.E.: GPU Laplace solver using optimized red-black Gauss-Seidel with SOR solver. https://github.com/kyleniemeyer/laplace_gpu (2012), accessed: 2026-05-20

49. Dominguez, J.M., Crespo, A.J.C., Valdez-Balderas, D., *et al.*: New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. Computer Physics Communications 184(8), 1848–1860 (2013). https://doi.org/10.1016/j.cpc.2013.03.008

50. Phillips, E., Fatica, M.: A CUDA implementation of the high performance conjugate gradient benchmark. In: Jarvis, S., Wright, S., Hammond, S. (eds.) High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation. PMBS 2014. Lecture Notes in Computer Science, vol. 8966, pp. 68–84. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17248-4_4

51. Menshov, I., Pavlukhin, P.: Highly scalable implementation of an implicit matrix-free solver for gas dynamics on GPU-accelerated clusters. J. Supercomput. 73, 631–638 (2017). https://doi.org/10.1007/s11227-016-1800-1

52. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Computer Physics Communications 271, 108231 (2022). https://doi.org/10.1016/j.cpc.2021.108231

53. Gorobets, A.V.: Adapting a scientific CFD code to industrial applications on hybrid supercomputers. Supercomputing Frontiers and Innovations 9(4), 49–54 (2022). https://doi.org/10.14529/jsfi220405

54. Magomedov, A.R., Gorobets, A.V.: Heterogeneous implementation of preconditioners based on Gauss–Seidel method for sparse block matrices. Computational Mathematics and Modeling 33(4), 438–442 (2022). https://doi.org/10.1007/s10598-023-09585-2

55. Khrapov, S.S., Agafonnikova, E.O., Khoperskov, A.V.: Prospects for improving computational efficiency of hydrodynamic simulations on supercomputers by increasing the number of GPUs per compute node. Supercomputing Frontiers and Innovations 12(2), 43–59 (2025). https://doi.org/10.14529/jsfi250204