Spectral Decomposition to Solve the Elasticity Problem in Quasi-static Formulation

Sergey Solovyev(✉) and Vadim Lisitsa

Institute of Mathematics SB RAS, Koptug ave. 4, 630090 Novosibirsk, Russia
511ssa@mail.ru

Abstract. In this paper we consider the problem of numerical solution of the boundary value problem of the theory of elasticity in static formulation in a rectangle with arbitrary boundary conditions. For this purpose, we use the approach of splitting in the direction of the Laplace operator based on its spectral decomposition, which is similar to the discrete Fourier transform but does not require periodicity of the boundary conditions. A fast matrix-to-vector multiplication algorithm is proposed using an efficiently software-implemented matrix multiplication algorithm. Numerical experiments are performed to show the effectiveness of the proposed method with the ability to solve the elasticity problem on a mesh of 10^9 nodes on systems with 128G RAM.

Keywords: Poroelasticity · Biot equation · Elasticity · finite differences · Laplace operator · spectral decomposition · SLAE · iterative methods · HPC · shared memory systems

1 Introduction

While calculating the effective elasticity tensor of a viscoelastic medium described by the differential equations of Biot [3] in the frequency domain, it is necessary to numerically solve the boundary value problem of the theory of elasticity in the static formulation. This leads to the question of solving the direct problem of finding the components of the displacement vector u, the components of the strain tensor ε and the stress tensor σ according to the given boundary conditions and the inhomogeneous stiffness tensor. Despite the method of discretisation of the boundary value problem by finite difference, volume or finite element methods, the problem of solving the resulting System of Linear Algebraic Equations (SLAE) arises. The solution of SLAE is the most resource consuming part and one of the most effective approaches is the use of preconditioned iterative methods. In addition, the boundary conditions for the initial differential formulation significantly affect the properties of SLAEs. For example, the Neumann-type conditions (the derivatives of the solution along the normal to

S. Solovyev developed the algorithm and did the numerical simulations using the supercomputer facilities of Siberian Supercomputer Center (Cluster NKS-30T) under the support of RSCF grant no. 19-77-20004-*П*.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15888, pp. 343–358, 2026.
https://doi.org/10.1007/978-3-031-97596-7_23

the surface are defined) cause ill-conditioned the SLAE, and it affects the convergence of the iterative process and the construction of the preconditioner. The construction of a preconditioner, and the choice of an iterative algorithm, are independent and important areas of research. The most effective preconditioners are multigrid methods [4,5], both Geometrical Multigrid and Algebraic Multigrid; methods based on incomplete factorisation (IL(0), ILU(p)); and Domain Decomposition methods, which are a set of methods based on the principle of 'divide and conquer' developed mainly for solving differential equations in 2D and 3D domains [8]. The most efficient and most commonly used are the conjugate gradient (CG) methods for SLAEs with self-adjoint matrices and bisadjoint gradients with stabilisation (BCGStab) for square matrices of the general form [8]. These preconditioning methods, as well as the iterative methods, are effectively implemented in software packages such as Intel MKL [1], PETSc [2], J. S. Saad's SPARSKIT [6], Tim Davis's UMFPACK MATLAB [7], and J. S. Sahad's UMFPACK [6]. Saad [6], UMFPACK by Tim Davis (MATLAB [7]). Despite the wide variety of methods implemented, the best approach is often to develop a specialised algorithm for a particular boundary value problem. This paper is devoted to the development of a numerical algorithm for solving the elasticity boundary value problem in a homogeneous medium in the static formulation. This solution can be used as an effective preconditioner both for solving the inhomogeneous elasticity problem and as a stand-alone problem. The main idea of this work is to use the method of spatial decomposition along different axes with the use of the 1D spectral decomposition of the Laplace operator. The method consists in reducing the original problem to a sequence of one-dimensional problems, which leads to a significant reduction in computational resources. The quasi-static formulation of the elasticity problem with constant Lamé coefficients, various boundary conditions such as Dirichlet and Neumann are considered. The equations and boundary conditions are discretised by a second order finite difference approximation on shifted rectangular meshes. This approximation leads to a positive definite SLAE with real coefficients (semidefinite in the case of the Neumann boundary condition). The paper consists of the following sections: The first describes the differential formulation of the elasticity problem, the finite difference approximation, the structure of the obtained SLAE and the solution idea. The second section is devoted to a detailed description of the spectral decomposition of the Laplace operator, as well as the fast multiplication of this operator based on the spatial decomposition method. The third section deals with the mixed derivatives of the elasticity operator, their singular decomposition and the correlation with the spectral decomposition of the Laplace operators, which form the 'diagonal' part of the elasticity operator. The fourth section describes the solution of the elasticity problem itself: based on the results of the previous sections, the splitting of the elasticity operator is presented with a description of the fast multiplication algorithm, and the numerical method for solving the SLAE is described. The last, fifth section is directly dedicated to numerical experiments. The developed algorithm has been tested: the correctness of the solution obtained, the efficiency of memory usage and the

high computational speed have been demonstrated. In addition, a comparison is made with direct methods optimised for solving sparse SLAEs.

2 Problem Statement, Proposed Solution

It is needed to numerically solve the elasticity equation. In two dimensions it looks like this:

$$\begin{gathered} \frac{\partial}{\partial x}\left[(\lambda+2\mu)\frac{\partial u_x}{\partial x}+\lambda\frac{\partial u_z}{\partial z}\right]+\frac{\partial}{\partial z}\left[\mu\left(\frac{\partial u_x}{\partial z}+\frac{\partial u_z}{\partial x}\right)\right]=0, \\ \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial u_x}{\partial z}+\frac{\partial u_z}{\partial x}\right)\right]+\frac{\partial}{\partial z}\left[\lambda\frac{\partial u_x}{\partial x}+(\lambda+2\mu)\frac{\partial u_z}{\partial z}\right]=0 \end{gathered} \tag{1}$$

In the equations (1) λ and μ are the Lame coefficients.

First, rewrite it in the form (2) and simplify it by considering the operator consisting of 2-dimensional Laplace operators and mixed derivative operators (3).

$$\begin{gathered} \left[\frac{\partial}{\partial x}(\lambda+2\mu)\frac{\partial}{\partial x}+\frac{\partial}{\partial z}\mu\frac{\partial}{\partial z}\right]u_x+\left[\frac{\partial}{\partial x}\lambda\frac{\partial}{\partial z}+\frac{\partial}{\partial z}\mu\frac{\partial}{\partial x}\right]u_z=0, \\ +\left[\frac{\partial}{\partial z}\lambda\frac{\partial}{\partial x}+\frac{\partial}{\partial x}\mu\frac{\partial}{\partial z}\right]u_x+\left[\frac{\partial}{\partial z}(\lambda+2\mu)\frac{\partial}{\partial z}+\frac{\partial}{\partial x}\mu\frac{\partial}{\partial x}\right]u_z=0. \end{gathered} \tag{2}$$

$$\begin{gathered} \left[\frac{\partial}{\partial x}\frac{\partial}{\partial x}+\frac{\partial}{\partial z}\frac{\partial}{\partial z}\right]u_x+\left[\frac{\partial}{\partial z}\frac{\partial}{\partial x}\right]u_z=0, \\ \left[\frac{\partial}{\partial x}\frac{\partial}{\partial z}\right]u_x+\left[\frac{\partial}{\partial z}\frac{\partial}{\partial z}+\frac{\partial}{\partial x}\frac{\partial}{\partial x}\right]u_z=0. \end{gathered} \tag{3}$$

The computational domain is rectangular; the boundary conditions are of Dirichlet or Neumann. The discretisation of the differential equation is performed by a second order finite difference on staggered grids with constant steps in each direction. The u_x components are located in the centre of the vertical edges and the u_z components are located in the centre of the horizontal edges. The boundary of the computational domain lies along the centres of the edges. An example of the grid $N_x = 6 \times N_z = 4$ is shown in Fig. 1.

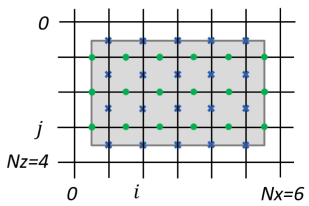

Fig. 1. Example of the computational mesh $N_x = 6 \times N_z = 4$.

As a result of finite-difference approximation, we obtain a system of linear algebraic equations (SLAE) that is positive definite (in the case of Dirichlet and Neumann type boundary conditions) or semi-definite (Neumann conditions only).

This SLAE can be represented as a large-block 2×2 form where diagonal blocks correspond to approximations of Laplace operators and non-diagonal blocks correspond to mixed derivatives (4). In the general case, the number of unknown components u_x is not equal to u_z, and the blocks A_{11} and A_{22} corresponding to the Laplace approximations for these components are square of different sizes, while the blocks A_{12} and A_{21} are rectangular and equal with transpose precision: $A_{21} = A_{12}^t$. For the given example (A_{11} is a matrix of size 20×20 obtained by their two-dimensional Laplace approximation on a mesh of 5×4, A_{22} is a matrix of size 18×18 for a mesh of 6×3, matrices A_{12} and A_{21} are 18×20 and 20×18, respectively.

$$\begin{bmatrix} \frac{\partial}{\partial x}\frac{\partial}{\partial x} + \frac{\partial}{\partial z}\frac{\partial}{\partial z} & \frac{\partial}{\partial z}\frac{\partial}{\partial x} \\ \frac{\partial}{\partial x}\frac{\partial}{\partial z} & \frac{\partial}{\partial z}\frac{\partial}{\partial z} + \frac{\partial}{\partial x}\frac{\partial}{\partial x} \end{bmatrix} \Rightarrow A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{4}$$

The idea of the solution the SLAE, derived from the Eq. 3, is based on the spectral decomposition of the Laplace 1D and on the spatial decomposition along different axes of the 2D (and, in general, 3D) Laplace operator. In the initial formulation of the elasticity problem, direct application of the splitting method to the elasticity operator is not possible because of the mixed derivatives and their off-diagonal blocks A_{21} and A_{12}, Therefore, we assume to use the decomposition of the Laplace operators corresponding to the diagonal blocks A_{11} and A_{22} of the original matrix to transform the matrix A to a large-block 2×2 matrix with diagonal blocks consisting only of the main diagonal. First, we describe the spectral decomposition of the Laplace operator along one direction. Then we apply the splitting scheme to the 2D Laplace operator as well as to the mixed derivative operator.

3 Spectral Decomposition of the Laplace Operator

3.1 Laplace 1D Singular and Spectral Decomposition

As a result of finite-difference approximation of the second derivative $-\frac{\partial^2}{\partial x^2}$ on a uniform mesh (without restriction of generality with grid spacing equal to one), we obtain the well-known symmetric three-diagonal matrix with values equal "2" on the main diagonal and "-1" on the two nearest diagonals. The values of the first and the last elements on the main diagonal will be different (depending on the type of boundary conditions). In the example of the matrix (5, left), $a = b = 1$ corresponds to the Dirichlet condition, and $a = b = 0$ corresponds to the Neumann condition. Also, the matrix A_n can be represented as products of some other operator B_n by its conjugate: $A_n = B_n B_n^*$. It is easy to see that this matrix is a finite-difference approximation of the first-order operator $-\frac{\partial}{\partial x}$ (5, right).

$$A_n = \begin{pmatrix} a^2+1 & -1 & & & 0 \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ 0 & & & -1 & b^2+1 \end{pmatrix}, B_n = \begin{pmatrix} -a & 1 & & & 0 \\ 0 & -1 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & & -1 & 1 \\ 0 & & & & -1 \; b \end{pmatrix} \quad (5)$$

For the matrix A_n we know how to obtain its spectral decomposition: $A_n = V_n D_n^2 V_n^*$ (diagonal matrix D_n^2, dense orthogonal matrix V_n). Based on this decomposition, it is easy to compute the singular decomposition $B_n = V_n D_n W_n^*$ associated with the spectral decomposition of A_n. The matrices B_n and W_n are rectangular, the number of nodes in the mesh n determines the dimensions of the matrices:

$$B_n^{[n\times(n+1)]}, A_n^{[n\times n]}, V_n^{[n\times n]}, D_n^{[n\times n]}, W_n^{[(n+1)\times n]} \quad (6)$$

Further, the orthogonality properties of these matrices are used: $V_n V_n^* = I, V_n^* V_n = I, W_n^* W_n = I$.

3.2 Laplace: 2D-Decomposition

To obtain the 2D spectral decomposition of the Laplace operator approximated on a rectangular mesh of n horizontal and m vertical nodes (Fig. 3, left), let us consider the second derivatives and their matrices. Schematically, the mesh corresponding to the operator $-\frac{\partial^2}{\partial x^2}$ does not contain horizontal connections (Fig. 2, right), and mesh of the operator $-\frac{\partial^2}{\partial y^2}$ does not contain vertical ones (Fig. 2, centre).

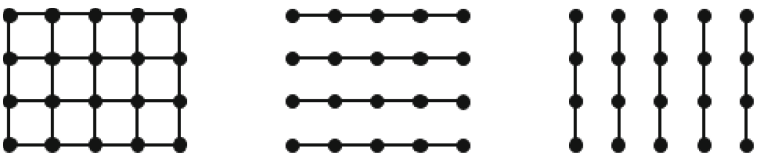

Fig. 2. On the left: rectangular mesh for approximating the Laplace $\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$; in the centre—the mesh for approximating the second derivative in the horizontal direction ($\frac{\partial^2}{\partial x^2}$); on the right—in the vertical direction ($\frac{\partial^2}{\partial y^2}$).

Using horizontal node numbering, the operator $\frac{\partial^2}{\partial x^2}$ yields a block-diagonal matrix $\overline{A}_{nm}$ of m identical A_n blocks, while $\frac{\partial^2}{\partial y^2}$ yields a block-tri-diagonal matrix $\overline{\overline{A}}_{nm}$ with blocks – unit matrices multiplied by the elements of the matrix

obtained from the 1D Laplace approximation on a grid of m nodes A_m. The result of the sum of these two matrices is the finite-difference 2D Laplace operator $\hat{A}_{nm}$ (7).

$$\left.\begin{aligned} A_n = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \Rightarrow \overline{A}_{nm} = \begin{pmatrix} A_n & & & \\ & A_n & & \\ & & A_n & \\ & & & A_n \end{pmatrix} \\ A_m = \begin{pmatrix} a_{11} & \dots & a_{1\,m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mm} \end{pmatrix} \Rightarrow \overline{\overline{A}}_{nm} = \begin{pmatrix} a_{11}I & \dots & a_{1\,m}I \\ \vdots & \ddots & \vdots \\ a_{m1}I & \dots & a_{mm}I \end{pmatrix} \end{aligned}\right\} \Rightarrow \hat{A}_{nm} = \overline{A}_{nm} + \overline{\overline{A}}_{nm} \tag{7}$$

This representation of a matrix as a direct sum can also be done for the matrices of its spectral decomposition. That is, the matrices $\overline{V}_{nm}$ and $\overline{D}_{nm}$ of the spectral expansion $\overline{A}_{nm} = \overline{V}_{nm}\overline{D}_{nm}\overline{V}^t_{nm}$ (and also $\overline{\overline{A}}_{nm}$ and $\overline{\overline{D}}_{nm}$ for $\overline{\overline{A}}_{nm} = \overline{\overline{V}}_{nm}\overline{\overline{D}}_{nm}\overline{\overline{V}}^t_{nm}$) are obtained by the same way from the matrices V_n, D_n, V_m, V_m of the spectral expansion of the 1D Laplace (matrices A_n and V_m).

It is easy to check that any matrix $\overline{M}_{nm} \in \{\overline{A}_{nm}, \overline{V}_{nm}, \overline{D}_{nm}\}$ is permutable with $\overline{\overline{M}}_{nm} \in \{\overline{\overline{A}}_{nm}, \overline{\overline{V}}_{nm}, \overline{\overline{D}}_{nm}\}$, i.e. $\overline{M}_{nm}\overline{\overline{M}}_{nm} = \overline{\overline{M}}_{nm}\overline{M}_{nm}$. Using this property and the orthogonality property of the matrices $\overline{V}_{nm}$ and $\overline{\overline{V}}_{nm}$, we obtain the spectral decomposition for the 2D Laplace $\hat{A}_{nm} = \hat{V}_{nm}\hat{D}_{nm}\hat{V}^t_{nm}$ (8), where $\hat{V}_{nm} = \overline{V}_{nm}\overline{\overline{V}}_{nm}$, and $\hat{D}_{nm} = \overline{D}_{nm} + \overline{\overline{D}}_{nm}$.

A few remarks about the properties of the above matrices. Despite the fact that $\hat{V}_{nm}$ is a dense matrix and at large sizes of its storage, computational operations with it are extremely inefficient, this can be overcome by implicitly representing it. That is, the operation of multiplication of $\hat{V}_{nm}$ by a vector can be performed by successive multiplications of two matrices $\overline{\overline{V}}_{nm}$ and $\overline{V}_{nm}$. Let us consider this operation in details.

$$\hat{A}_{nm} = \overline{V}_{nm}\overline{\overline{V}}_{nm}(\overline{D}_{nm} + \overline{\overline{D}}_{nm})\overline{\overline{V}}^t_{nm}\overline{V}^t_{nm}) \tag{8}$$

A schematic representation of the multiplication of a matrix $\overline{V}_{nm}$ by some vector x and the multiplication of $\overline{\overline{V}}_{nm}$ by a vector y is shown by the formula (9).

$$\begin{pmatrix} z_1 \\ \vdots \\ z_{nm} \end{pmatrix} = \begin{pmatrix} v_{11}I & \dots & v_{1\,m}I \\ \vdots & \ddots & \vdots \\ v_{m1}I & \dots & v_{mm}I \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_{nm} \end{pmatrix}, \quad \begin{pmatrix} y_1 \\ \vdots \\ y_{nm} \end{pmatrix} = \begin{pmatrix} V_nI & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_nI \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{nm} \end{pmatrix} \tag{9}$$

Let us write vectors x and y as matrices X and Y of n rows and m columns with column-wise storage, i.e., the element at the intersection of the i-th row and j-th column in matrix X has index $k = i + j * n$ in vector x, and the k-th element

of the vector is equal to the (i, j)-th element of the corresponding matrix. So, the matrix-vector multiplication operation (9, right) is rewritten as matrix-matrix operation $Y = V_n X$ (Fig. 3, right).

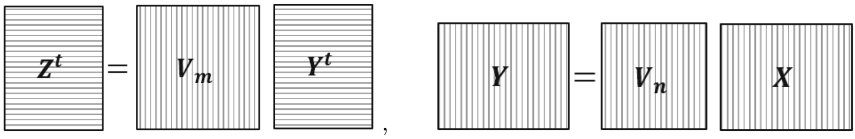

Fig. 3. Schematic representation of fast matrix multiplication with column-wise storage of matrix elements Y, V_n and X in memory. Left: $Y = V_n X$, right: $Z^t = V_m Y^t$.

When multiplying a block matrix $\overline{\overline{V}}_{nm}$ by a vector y (Fig. 9 on the left), with the above order of entry of elements and representing vectors y and z as matrices Y and Z, each element of the first column V_m is multiplied by all the first n-elements of y. Each element of the second column V_m is multiplied by the second group of n-elements of y, and so on. This algorithm is equivalent to multiplying the matrix V_m by the transposed matrix Y^t, with row-wise storage (Fig. 3, left).

Using an additional transpose, the operation of multiplying the matrix $\hat{V}_{nm} = \overline{V}_{nm}\overline{\overline{V}}_{nm}$ by the vector x can be rewritten as $Z = V_n X V_m^t$ (Fig. 4).

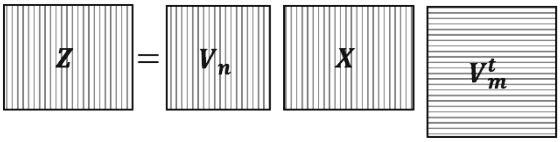

Fig. 4. Schematic representation of the fast multiplication $Z = V_n X V_m^t$.

An important property of this representation is that when changing the numbering direction in matrices (from column-wise to row-wise storage), it is not necessary to physically perform transposition (copying of matrix elements). It is sufficient to use matrix multiplications from already developed and well optimised matrix multiplication implementations that work with different numeration of dense matrices.

4 Elasticity Operator: Mixed Derivatives

4.1 Splitting Scheme

Let us write the matrix A_n (discrete 1D operator of the second derivative) as the product of the matrix obtained from the approximation of the first derivative by its adjoint (the elements of the matrix are real and we will use the "transpose" operator): $A_n = B_n B_n^t$.

In 2D space, the second derivative operators will have the form: $\overline{A}_{nm} = \overline{B}_{nm}\overline{B}^t_{nm}$ and $\overline{\overline{A}}_{nm} = \overline{\overline{B}}_{nm}\overline{\overline{B}}^t_{nm}$. Here $\overline{B}_{nm}$ – a block-diagonal matrix contains of m rectangular bidiagonal blocks B_n, $\overline{\overline{B}}_{nm}$ –a block-diagonal matrix whose blocks are unit matrices of size n multiplied by the elements of matrix B_m (see 10).

$$\overline{B}_{nm} = \begin{pmatrix} B_n & & & \\ & B_n & & \\ & & B_n & \\ & & & B_n \end{pmatrix}, B_n = \begin{pmatrix} b_{11} & b_{12} & 0 & \dots & 0 \\ 0 & b_{22} & b_{23} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b_{nn} & b_{nn+1} \end{pmatrix};$$

$$\overline{\overline{B}}_{nm} = \begin{pmatrix} b_{11}I & b_{12}I & 0 & \dots & 0 \\ 0 & b_{22}I & b_{23}I & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b_{mm}I & b_{mm+1}I \end{pmatrix}, B_m = \begin{pmatrix} b_{11} & b_{12} & 0 & \dots & 0 \\ 0 & b_{22} & b_{23} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & b_{mm} & b_{mm+1} \end{pmatrix} \tag{10}$$

We use such a decomposition for the blocks $A_{11}, A_{22}, A_{12}, A_{21}$ of the matrix A (4). Because of the staggered grids, the Laplace operator for the horizontal component of the solution has the dimension of n along the horizontal axis is $N_x - 1$ and $m = N_z$ along the vertical axis; for the vertical component—$n = N_x$, and $m = N_z - 1$. Thus, the diagonal blocks A_{11}, A_{22} can be represented as (11)):

$$\begin{aligned} A_{11} &= \overline{B}_{Nx-1,Nz}\overline{B}^t_{Nx-1,Nz} + \overline{\overline{B}}_{Nx-1,Nz}\overline{\overline{B}}^t_{Nx-1,Nz} \\ A_{22} &= \overline{B}_{Nx,Nz-1}\overline{B}^t_{Nx,Nz-1} + \overline{\overline{B}}_{Nx,Nz-1}\overline{\overline{B}}^t_{Nx,Nz-1}. \end{aligned} \tag{11}$$

The off-diagonal blocks A_{12}, A_{21} can be writen are expressed in a same maner. The matrix A_{12} (mixed derivatives $\frac{\partial}{\partial x}\frac{\partial}{\partial z}$) is represented as $A_{12} = \overline{B}_{nm}\overline{\overline{B}}^t_{nm}$. It is important to note that the matrix $\overline{\overline{B}}^t_{nm}$ operates on the vertical components in the space of dimension ($n = N_x$, $m = N_z - 1$), and $\overline{B}_{nm}$ operates on the horizontal components in the space of another dimension ($n = N_x - 1$, $m = N_z$). That is, $A_{12} = \overline{B}_{Nx-1,Nz}\overline{\overline{B}}^t_{Nx,Nz-1}$ (Fig. 5, left). To check the correct use of dimensions: if in the formula (10) we set N_x and N_z in the matrix representation, we obtain in the product the number of rows equal to the dimension of the horizontal component space, and the number of columns – vertical. The same reasoning can be done for the block A_{21} (Fig. 5, right).

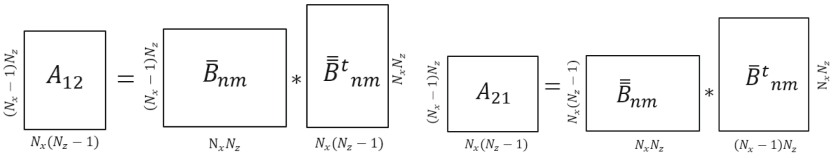

Fig. 5. Schematic illustration of diagonal blocks A_{12} and A_{21}.

4.2 SVD Decomposition

Using one-dimensional singular expansions $B_n = V_n S_n W_n^t$ and $B_m = V_m S_m W_m^t$, we obtain the block representation of the matrices $\overline{B}_{nm} = \overline{V}_{nm}\overline{D}_{nm}\overline{W}^t_{nm}$ and $\overline{\overline{B}}_{nm} = \overline{\overline{V}}_{nm}\overline{\overline{D}}_{nm}\overline{\overline{W}}^t_{nm}$ (Fig. 6).

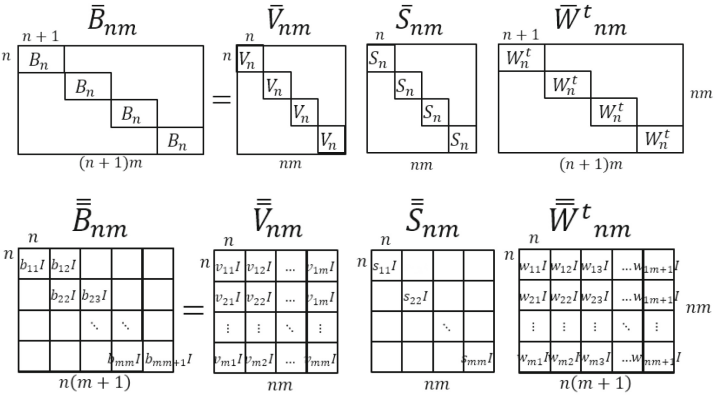

Fig. 6. Block representation of the singular expansion of the 2D operators $\frac{\partial}{\partial x}$ and $\frac{\partial}{\partial z}$.

The matrices $\overline{V}_{nm}$ and $\overline{\overline{V}}_{nm}$ – consist of the eigenvectors of the matrices $\overline{A}_{nm}$ and $\overline{\overline{A}}_{nm}$, and the matrices $\overline{S}_{nm}$ and $\overline{\overline{S}}_{nm}$ are diagonal, consisting of singular values of these matrices (roots from the diagonal matrices of the eigenvalues of $\overline{D}_{nm}$ and $\overline{\overline{D}}_{nm}$).

5 Inversion of the Elasticity Operator

5.1 Splitting Scheme for Elasticity Operator

Let us summarise the above conclusions with respect to the matrix A:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} \overline{B}_{Nx-1,Nz}\overline{B}^t_{Nx-1,Nz} + \overline{\overline{B}}_{Nx-1,Nz}\overline{\overline{B}}^t_{Nx-1,Nz} & \overline{B}_{Nx-1,Nz}\overline{\overline{B}}^t_{Nx,Nz-1} \\ \overline{\overline{B}}_{Nx,Nz-1}\overline{B}^t_{Nx-1,Nz} & \overline{B}_{Nx,Nz-1}\overline{B}^t_{Nx,Nz-1} + \overline{\overline{B}}_{Nx,Nz-1}\overline{\overline{B}}^t_{Nx,Nz-1} \end{pmatrix} \tag{12}$$

Using the spectral and singular expansions we obtain:

$$\begin{aligned}
A_{11} &= \overline{V}_{Nx-1Nz}\overline{\overline{V}}_{Nx-1Nz}(\overline{D}_{Nx-1Nz}+\overline{\overline{D}}_{Nx-1Nz})\overline{\overline{V}}^{t}_{Nx-1Nz}\overline{V}^{t}_{Nx-1Nz}\\
A_{12} &= \overline{V}_{Nx-1Nz}\overline{S}_{Nx-1Nz}\overline{W}^{t}_{Nx-1Nz}\overline{\overline{W}}_{NxNz-1}\overline{\overline{S}}_{NxNz-1}\overline{\overline{V}}^{t}_{NxNz-1}\\
A_{21} &= \overline{\overline{V}}_{NxNz-1}\overline{\overline{S}}_{NxNz-1}\overline{\overline{W}}^{t}_{NxNz-1}\overline{W}_{Nx-1Nz}\overline{S}_{Nx-1Nz}\overline{V}^{t}_{Nx-1Nz}\\
A_{22} &= \overline{V}_{NxNz-1}\overline{\overline{V}}_{NxNz-1}(\overline{D}_{NxNz-1}+\overline{\overline{D}}_{NxNz-1})\overline{\overline{V}}^{t}_{NxNz-1}\overline{V}^{t}_{NxNz-1}
\end{aligned} \tag{13}$$

The matrix A is factorised into the product of the orthogonal matrices $\hat{V}$, $\hat{V}^t$ and the matrix $\hat{A}$ with diagonal blocks consisting only of diagonal elements:

$$\begin{aligned}
A &= \hat{V}\hat{A}\hat{V}^t,\\
\hat{V} &= \begin{pmatrix} \overline{V}_{Nx-1Nz}\overline{\overline{V}}_{Nx-1Nz} & 0\\ 0 & \overline{V}_{NxNz-1}\overline{\overline{V}}_{NxNz-1}\end{pmatrix}\\
\hat{A} &= \begin{pmatrix} \overline{D}_{Nx-1Nz}+\overline{\overline{D}}_{Nx-1Nz} & \hat{A}_{12}\\ \hat{A}_{21} & \overline{D}_{NxNz-1}+\overline{\overline{D}}_{NxNz-1}\end{pmatrix},\\
\hat{A}_{12} &= \overline{\overline{V}}^{t}_{Nx-1Nz}\overline{S}_{Nx-1Nz}\overline{W}^{t}_{Nx-1Nz}\overline{\overline{W}}_{NxNz-1}\overline{\overline{S}}_{NxNz-1}\overline{V}_{NxNz-1}\\
\hat{A}_{21} &= \overline{V}^{t}_{NxNz-1}\overline{\overline{S}}_{NxNz-1}\overline{\overline{W}}^{t}_{NxNz-1}\overline{W}_{Nx-1Nz}\overline{S}_{Nx-1Nz}\overline{\overline{V}}^{t}_{Nx-1Nz}
\end{aligned} \tag{14}$$

As a result, the solution of the system $Ax - b$ leads to the solution the system(15).

$$\begin{aligned}
\hat{V}z &= b\\
\hat{A}y &= z\\
\hat{V}^t x &= y
\end{aligned} \tag{15}$$

Due to the orthogonality of $\hat{V}$, the solution of the first and third equations are trivial, let us focus on the solution of the second equation. The spectral properties of the matrix $\hat{A}$ remain the same as for A, but the off-diagonal blocks $\hat{A}_{12}$ and $\hat{A}_{21}$ are dense. Despite this, the operation of multiplying the matrix $\hat{A}$ by a vector can be performed very efficiently by using an approach similar to the dense matrix multiplication in the method of separation of variables for the 2D Laplace operator (see Fig. 4) to multiply the blocks $\hat{A}_{12}$ and $\hat{A}_{21}$ by vectors. That is, multiplication of the product of the matrices $\overline{\overline{V}}^{t}_{Nx-1Nz}\overline{S}_{Nx-1Nz}\overline{W}^{t}_{Nx-1Nz}\overline{\overline{W}}_{NxNz-1}\overline{\overline{S}}_{NxNz-1}\overline{V}_{NxNz-1}$ by vector u_z is rewritten as a multiplication of matrices $S_{N_x-1}W^t_{N_x-1}V_{N_x}U_zS_{N_z-1}W^t_{N_z-1}V_{N_z-1}$. This is shown schematically in Fig. 7.

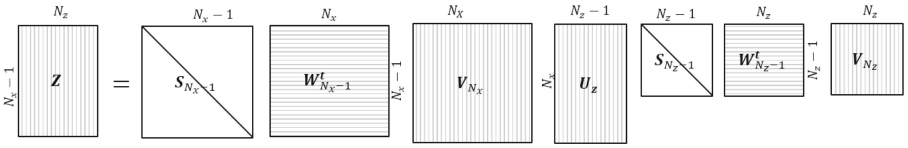

Fig. 7. Schematic representation of the multiplication of the matrix $\hat{A}_{12}$ by the vector u_z.

To simplify the counting of the arithmetic operations, let us consider the square computational domain ($N = N_x = N_z$). Then multiplying of the matrix $\hat{A}_{12}$ by a vector requires $O(8N^3) + O(4N^2)$ matrix of "add" and "multiply" operations, while formally at dense representation $\hat{A}_{12}$ requires $O(2N^4)$ operations.

The total number of operations required to multiply a matrix $\hat{A}$ by a vector is $O(16N^3) + O(12N^2)$. Theoretically, the second term could be ignored, but in practice, from a computational point of view, dense matrix multiplication operations are maximally optimised and parallelized. However, the multiplication of a diagonal matrix by a vector is not optimised. The reason is that the number of memory accesses of these operations is of the same order: $2N^2$ for multiplying two square matrices and N^2 for multiplying a diagonal matrix by a square matrix.

The multiple multiplication of $\hat{A}$ by a vector can be further optimised by pre-multiplying the matrices $F_{N_x-1} = S_{N_x-1} W^t_{N_x-1} V_{N_x}$, and $F_{N_z-1} = S_{N_z-1} W^t_{N_z-1} V_{N_z-1}$. The operation of multiplication of block $\hat{A}_{12}$ by vector u_z is written in the form $F_{N_x-1} U_z F_{N_z-1}$ (see Fig. 8).

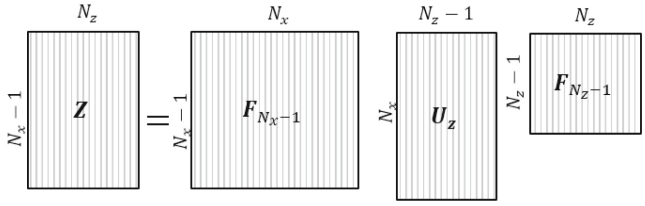

Fig. 8. Schematic representation of the optimised multiplication of matrix $\hat{A}_{12}$ by vector u_z

As a result, the total number of operations when multiplying the matrix $\hat{A}$ by a vector twice smaller is $O(8N^3) + O(4N^2)$ operations.

Note that all of this section is about the simplified elasticity operator (3). For the original one, with constant Lame coefficients (1), the decomposition of the matrix A will have the form (16).

$$\begin{gathered}\hat{A}_{11} = D_{11} = (\lambda + \mathbf{2}\mu)\overline{D}_{Nx-1Nz} + \mu\overline{\overline{D}}_{Nx-1Nz} \\ \hat{A}_{22} = D_{22} = \mu\overline{D}_{NxNz-1} + (\lambda + 2\mu)\overline{\overline{D}}_{NxNz-1} \\ \hat{A}_{12} = (\lambda + \mu)\hat{A}_{12}^t = \overline{\overline{V}}^t_{Nx-1Nz}\overline{S}_{Nx-1Nz}\overline{W}^t_{Nx-1Nz}\overline{\overline{W}}_{NxNz-1}\overline{\overline{S}}_{NxNz-1}\overline{V}_{NxNz-1}\end{gathered} \tag{16}$$

5.2 Solution SLAE Algorithm

To solve the system $\hat{A}u = b$, we use the preconditioned conjugate gradient algorithm. The preconditioner $\hat{B}$ is a matrix consisting of diagonal blocks $\hat{A}_{11}$ and $\hat{A}_{22}$. This preconditioning is equivalent to solving by conjugate gradients (17) the system with matrix $\overline{A} = \hat{B}^{-1/2}\hat{A}\hat{B}^{-1/2}$.

$$\begin{gathered}\overline{A} = \begin{pmatrix} I & \overline{A}_{12} \\ \overline{A}_{21} & I \end{pmatrix} \\ \overline{A}_{12} = \overline{A}^t_{21} = D_{11}^{-1/2}\hat{A}_{12}D_{22}^{-1/2}\end{gathered} \qquad \begin{gathered} r_0 := b - \overline{A}x_0;\ p_0 = r_0 \\ \textit{Loop } j = 0, 1, \ldots \textit{ until converge} \\ q_i = \overline{A}p_j \\ \alpha := \frac{(r_j, r_j}{(q_j, p_j)} \\ x_{j+1} := x_j + \alpha_j p_j \\ r_{j+1} := r_j - \alpha_j q_j \\ \beta_j := \frac{(r_{j+1}, r_{j+1}}{(r_j, r_j)} \\ p_{j+1} := r_{j+1} + \beta_j p_j \end{gathered} \tag{17}$$

As a result, the solution of the original SLAE $Ax = b$ consists of the next steps:

1. Compute: 1D singular decompositions;
2. Solve: $\hat{V}z = b$ using the orthogonality property of $\hat{V}$;
3. Product: $\overline{z} := \hat{B}^{-1/2}z$ using the diagonality property of $\hat{B}$;
4. Solve: SLAE $\overline{A}\overline{y} = \overline{z}$ by conjugate gradients;
5. Product: $\overline{y} := \hat{B}^{-1/2}y$ using the diagonality property of $\hat{B}$;
6. Solve: SLAE $\hat{V}^t x = y$ using the orthogonality property of $\hat{V}$.

The most expensive point is #3, which is implemented in software using optimised functions from mathematical libraries. As mentioned above, to simplify FLOP counting, we assume $N = N_x = N_z$. Then the dimensions of the SLAEs and vectors are equal to $2N^2$. Two groups of operations can be considered: 1) multiplication of dense matrices of size $N \times N$ (blocks appearing in the multiplication of $\overline{A}p_j$), 2) addition of vectors with dimensions of order N^2 (recalculation of x_{j+1}, r_{j+1} and p_{j+1}). The number of arithmetic operations in the first case is $O(N^3)$ (estimates are given in the previous section); for vector addition, the number of operations is $O(N^2)$. It is also important to take into account the number of access to RAM, which is approximately the same in both cases: $O(N^2)$. In the context of numerical experiments, we analyse these operations separately. It is also worth estimating the amount of memory needed to store SLAEs and intermediate data (see Table 1).

Table 1. Amount of memory used by different functions or data structures.

Data	Number of memory elements for storage
Right hand side b	$2N^2$
Matrix A (CSR format)	$10N^2$
Spectral decomposition A	$8N^2$
(modification of $\hat{V}$,$\hat{B}$,$\hat{A}$)	$5N^2$ (square mesh)
Iterative algorithm (vectors x_{j+1}, r_{j+1}, p_{j+1}, q_{j+1})	$8N^2$

Due to symmetry, the matrix A consists of 10 diagonals of size $n = N^2$. However, it is possible to avoid storing the original matrix by using on-the-fly matrix element generation during the operation of multiplication by a vector.

To store the above decomposition of A into multipliers, we need $8N^2$ elements, i.e. $\hat{V}$ – four dense matrices $V_{N_x-1}, V_{N_z}, V_{N_x}, V_{N_z-1}$ of size $N \times N$, a diagonal $\hat{B}$ of size $2N^2$ and two dense blocks F_{N_x-1}, F_{N_z-1} of size $N \times N$ defining the operation of multiplying the matrix $\overline{A}$ by a vector. Also, the iterative algorithm needs to store four vectors $x_{j+1}, r_{j+1}, p_{j+1}, q_{j+1}$ of size $2N^2$, as well as a pair of temporary arrays of size N^2 needed in the process of multiplying $\overline{A}$ by a vector. To estimate the peak load, it is worth noting that the initial matrix A as well as the right-hand side of b are not used in the iterative process; technically, they can be removed from memory and an estimate of $16N^2$ elements for the peak load can be obtained. Recall that these estimates are true for both square and rectangular meshes. In the case of square meshes ($N_x = N_z$) we can save on memory by using two vectors V_{N_x-1}, V_{N_x-1} instead of four, and one vector F_{N_x-1} and get a peak load of $13N^2$.

Next, we present the results of numerical experiments.

6 Numerical Experiments

There are some statements that should be checked in the experiments. The 1st one is the independence of the convergence rate on the mesh size. The second statement – dependence of convergence rate on coefficients λ and μ and on boundary types (Dirichlet or Neuman). The third investigation should be made – comparison of the iterative algorithm with commercial sparse direct solver from Intel MKL. The criteria of comparison are computation time and memory consumption. The quality of the solution is checked by the relative residual $||b - Ax||/||b||$.

The computations were performed in double precision on **Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz, 256G RAM**.

Memory usage optimisation makes it possible to solve the problem on a mesh of 10^9 nodes on a server with 128G RAM. The convergence of the iterative process does not critically depend on both the boundary conditions and the ratio of λ and μ. Although, as expected, the number of iterations increases when the

Table 2. Dependence of the number CG iterations on different physical properties for the largest problem. The stopping criteria is relative residual 10^{-10}.

Computational mesh: 10^9 nodes ($N_x = N_z = 32\ 000$)	Iteration number	Computational time, s.	Relative residual of initial SLAE
$\lambda = 10\mu$, Neumann	35	13506	$3.5 * 10^{-9}$
$\lambda = \mu$, Neumann	16	6735	$3.7 * 10^{-9}$
$\lambda = \mu$, Dirichlet	8	3905	$1.5 * 10^{-11}$

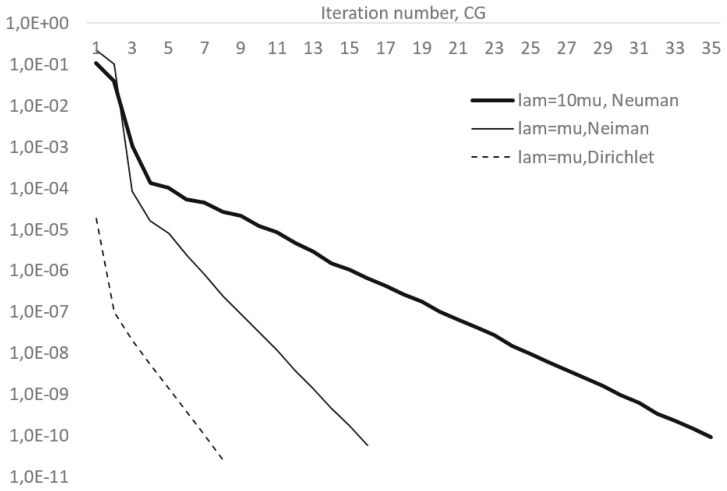

Fig. 9. The relative residual behaviour of the iterative process for the example of three variants of the problem.

Table 3. CG computational process profiling with a stopping criterion on the relative residual 10^{-10}.

$\lambda = 10\mu$, Neumann; computational mesh $N_x = N_z =$	Memory consumption, G	Number of iterations	CG computational time, s.	Time to solve initial SLAE, s.	Initial SLAE relative residual
1 000	0.1	40	1.2	1.3	$0.6 * 10^9$
2 000	0.5	39	6.9	7.4	$0.7 * 10^9$
4 000	1.9	39	37.5	39.9	$0.7 * 10^9$
8 000	7.4	38	238	251	$1.3 * 10^9$
16 000	29.4	36	1671	1767	$2.5 * 10^9$
32 000	117.5	35	12786	13506	$3.5 * 10^9$

ratio λ/μ increases, as well as when the Neumann condition is used (Table 2 and Fig. 9).

The convergence rate does not depend on the mesh size. For significantly different values of the coefficients λ and μ for 8 nested meshes, the characteristics

Table 4. Computational time for operations of different complexity: GEMM (N^3) and vector addition/multiplication (N^2); memory cost as a function of mesh size.

$\lambda = 10\mu$, Neumann; computational mesh $N_x = N_z =$	Memory consumption, G	Number of iterations	GEMM (N^3)	Other functions (N^2 and less)
1 000	0.1	40	0.6	0.6
2 000	0.5	39	4.2	2.7
4 000	1.9	39	27.1	10.4
8 000	7.4	38	198	40
16 000	29.4	36	1504	167
32 000	117.5	35	11818	968

of the computational process are shown in Table 3. The profiling of the counting time for different arithmetic operations is shown in Table 4.

Comparison with direct solver showed the effectiveness of the proposed algorithm, but it should be noted that direct methods are intended mainly for solving problems in inhomogeneous media, as well as in areas with complex geometry.

Table 5. Comparison with direct solver for high contrast λ/μ ratio.

$\lambda = 10\mu$, Neumann;	Memory consumption, G		Time to solve initial SLAE, s.	
computational mesh $N_x = N_z =$	Spectral decomposition	Direct solver	Spectral decomposition	Direct solver
1 000	0.1	5.1	1.3	11.2
2 000	0.5	14.5	7.4	50.4
4 000	1.9	53.9	39.9	254
8 000	7.4	No memory	251	No memory
16 000	29.4	No memory	1767	No memory
32 000	117.5	No memory	13506	No memory

7 Conclusion

For the numerical solution of the elasticity problem in the quasi-static state, an algorithm for the construction of a preconditioner in a rectangular domain with arbitrary boundary conditions has been developed. It uses the approach of splitting scheme in different directions together with the spectral decomposition of the Laplace operator. This approach has allowed the development of an efficient algorithm for solving the elasticity problem in a homogeneous medium. The proposed algorithm is 'memory limited', so its implementation uses the functionality of the high performance dense matrix multiplication (GEMM), giving efficiency close to the peak CPU. Computational experiments confirmed the high performance of the algorithm on various examples. In particular, the elasticity problem was solved on a computational mesh of 10^9 nodes on a server with 128G RAM.

References

1. Intel math kernel library documentation. https://software.intel.com/en-us/intel-mkl/
2. Balay, S., et al.: PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.5, Argonne National Laboratory (2014)
3. Biot, M.A.: Theory of propagation of elastic waves in fluid-saturated porous solid i. low-frequency range. J. Acoust. Soc. Am. **28**, 168–178 (1956)
4. Briggs, W., Henson, V., McCormick, S.: A Multigrid Tutorial, 2nd Edition (2000)
5. Huckle, T.: Multigrid preconditioning and toeplitz matrices (2000)
6. Saad, Y.: A basic tool-kit for sparse matrix computations. Web cite (2021). https://www-users.cse.umn.edu/~saad/software/SPARSKIT/
7. Trujillo-Ortiz, A.: Roystest. MATLAB Central File Exchange (2020). https://www.mathworks.com/matlabcentral/fileexchange/17811-roystest
8. Saad, Y.: Iterative Methods for Sparse Linear Systems. PWS Publishing, New York (1996)