East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.300921.210522

Vol. **13**, No. 1, pp. 47-58
February 2023

Fast Solvers for Systems of Linear Equations with Block-Band Matrices

B.Ya. Steinberg, O.B. Steinberg, P.A. Oganesyan, A.A. Vasilenko, V.V. Veselovskiy and N.A. Zhivykh[*]

Institute of Mathematics, Mechanics and Computer Sciences, Southern Federal University, Rostov-on-Don, Russia.

Received 30 September 2021; Accepted (in revised version) 21 May 2022.

Abstract. This article deals with parallel iterative algorithms for linear systems with block-band matrices. The algorithms can be used in mathematical modeling of the problems involving finite difference and finite element methods. The solvers are adjusted to the problem and to the computing systems, which use special precompilers. Applications of the algorithms to the ACELAN-COMPOS software package focused on the new material modeling, is described. To achieve a high performance, both parallel programming techniques and the optimization of the processor memory hierarchy are used. The results of numerical experiments confirm the efficiency of the methods and algorithms.

AMS subject classifications: 65F08, 65F50

Key words: High performance computing, parallel computing, iterative algorithm, sparse matrix, system of linear algebraic equations.

1. Introduction

In many problems of mathematical modeling, there is a need to solve systems of linear algebraic equations with large sparse matrices. Iterative algorithms are used to solve such problems [9, 23].

This work continues the series of publications of the authors [26, 28, 30]. The solvers presented in this article are embedded in the ACELAN COMPOS application software package for modeling the properties of new materials [18, 20]. In recent years, the main distinguishing feature of processors is that the execution time of arithmetic operations is more than an order of magnitude bigger than the time of reading the arguments of such operations from RAM [16]. In this work, we use structures for efficient data storage and fast algorithms. Numerical experiments demonstrate the high efficiency of the methods

[*]Corresponding author. *Email addresses:* borsteinb@mail.ru (B.Ya. Steinberg), olegsteinb@gmail.com (O.B. Steinberg), oganesyan@hey.com (P.A. Oganesyan), avas@sfedu.ru (A.A. Vasilenko), veselovsky@sfedu.ru (V.V. Veselovskiy), zhivyh@sfedu.ru (N.A. Zhivykh)

presented. While in [26, 28] the solver was applied to the matrices obtained by a finite difference method, we extend the capabilities of the solver to matrices arising in finite element methods.

Since the solver in the application software package is focused on a certain set of similar problems, the solver is assumed to be parameterized. The parameters of the solver are both the characteristics of the input data of the problem (for example, the number of unknown functions and the number of differential equations in the model, the numerical solution of which leads to the system of linear algebraic equations) and the characteristics of the computing system (the amount of cache memory, parallelization capabilities).

Block-band matrices arise when regular grids are used on the domains having the form of rectangular parallelepipeds. If the region has a different shape, then curved orthogonal grids can sometimes be used to obtain a system of linear algebraic equations with a block-band matrix [15, 17]. The solution of linear systems with such matrices can be accelerated by employing specific data structures and parallel methods that can operate with blocks [1, 2, 11]. Such solvers are usually iterative due to the need to keep low memory profile, which is usually not possible for direct solvers. An important part of the development of such iterative solver is convergence analysis [10] and multi-stage performance analysis [3, 5]. Asynchronous parallel nonlinear multi-splitting programs and their convergence analysis were presented in [1]. Designing proper preconditioner is another vital part of solver development for specific matrix structures [4, 6–8].

An important feature of the ACELAN COMPOS package solver presented in this work is that not only the initial data of the mathematical model but also the solver is formed in the package. In addition, the precompiler — i.e. the preliminary compiler in [26, 28], is modified. The presented precompiler merges loops that have different for-loop headers [27]. Information dependency analysis is used for the correctness and effectiveness of the merge. The use of the precompiler is justified by the fact that the modern optimizing compilers such as GCC, LLVM, ICC do not optimize programs efficiently enough [14]. A precompiler is a preliminary compiler that converts a C program into a faster program of the same language. The precompiler presented in the paper, is developed on the basis of OPS (Optimizing Parallelizing System) [13, 21] and shows acceleration by 1.25 times. Ways to get further performance are outlined.

2. Features of Block-Band Matrices

The block-band matrices are a kind of sparse matrices. Sparse matrices are stored in memory [22] only by values of non-zero elements and their row and column numbers (3 numbers). Block-band matrices can be stored only by non-zero diagonals [26, 28], which are stored as arrays. In this case, column and row numbers should not be stored. The amount of memory used is 2 times less than for sparse matrices of the general form.

Band matrices arise in finite-difference and finite-element methods — cf. Figs. 1 and 2, when covering a rectangular parallelepiped with a Cartesian grid [2–8, 10, 11]. If we consider a system of differential equations, the matrix of the system of linear algebraic equations is a block-band matrix — cf. Fig. 3. The elements of this matrix are blocks (matrices)

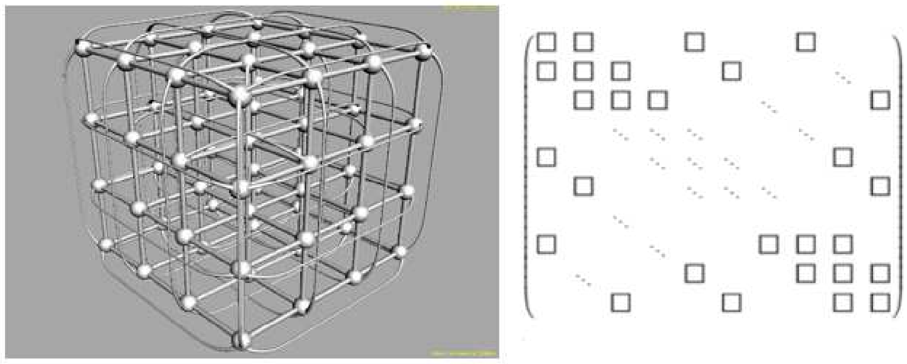

Figure 1: View of the matrix created by the finite difference method with Cartesian grid on a rectangular parallelepiped. The matrix contains 7 nonzero diagonals.

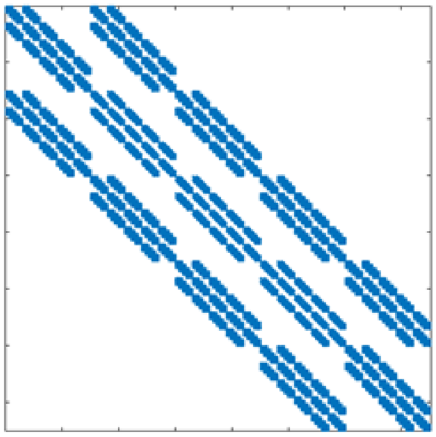

Figure 2: Matrix created by ACELAN COMPOS package using the finite element method. The matrix contains 9 non-zero block diagonals.

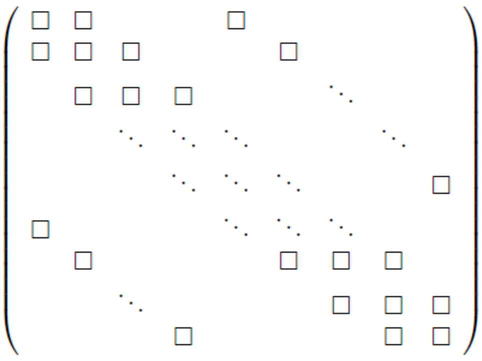

Figure 3: A five-diagonal block-band matrix obtained by the finite difference method for a system of differential equations of functions of two variables.

Figure 4: One of the block diagonals of a matrix.

whose dimension is equal to the number of equations in the original system of differential equations. Block-band matrices also arise when using the finite element method.

Each block diagonal — cf. Fig. 4, of a block-band matrix is stored in the memory as a one-dimensional array. Elements of each block are placed in RAM row by row. Such placement leads to minimization of cache misses. The diagonal of matrix A consists of all elements A_{ij} for which $(i-j)=k$. This makes it easy to find the column number by the row number of an element. Elements of block diagonals can be obtained through simple generalization.

3. Iterative Algorithms for Linear Systems

Consider a system of linear equations $Ax=b_0$ and the iterative algorithm for solving it

$$\begin{aligned} x^{(k+1)} &= Bx^k + b, \\ B &= I - C^{-1}A, \\ b &= C^{-1}b_0, \end{aligned} \tag{3.1}$$

where k is an iteration number, b_0 is an initial approximation, C is a nonsingular matrix.

For the iterative process convergence, the spectral radius of matrix B must be less than 1. In order to perform each iteration quickly, it is necessary to quickly perform the calculation $z=C^{-1}\cdot y$ and $y=b_0-A\cdot x$. We will consider the matrix C to be equal to the submatrix of matrix A consisting of the main matrix and the block-diagonals next to it. Matrix A can be represented as $A=C+O$, where O is a matrix consisting of the remaining elements of matrix A after subtraction of the elements on the main block diagonal and adjacent (on both sides) block diagonals (Fig. 5). After LU-decomposition of the matrix C (or $C=L\cdot D\cdot L^T$ for a symmetric matrix), $C^{-1}\cdot x$ at each iteration of the algorithm can be quickly calculated.

In many applications, the matrix C is symmetric or positively defined or has diagonally dominant properties. Matrix A may not be symmetric or positively defined (for example, with a saddle-point singularity), but C is symmetric and positively defined. If matrix A had diagonally dominant properties, then matrix C has the same feature.

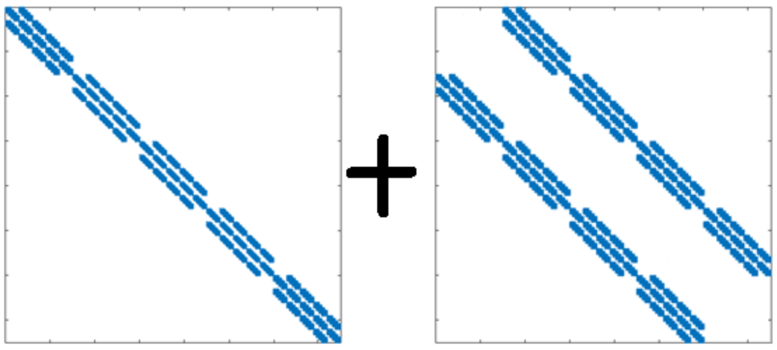

Figure 5: Representation of original matrix as a sum of two matrices $A = C + O$.

Decomposition of the matrix C into a product of two-diagonal (block-two-diagonal) matrices can take a long time. It is important that the calculation of $C^{-1} * y$ should be performed quickly, if possible, with parallelization [30]. If the matrix C is symmetric, then its decomposition $L \cdot D \cdot L^T$ allows you to store only the matrices L and D, which should give acceleration. It is expected that with a larger number of computing cores, the parallel solution of a system of linear equations with a two-diagonal matrix will give greater acceleration. And for a new generation of processors with more than 1000 cores [24], acceleration can also be obtained for block tridiagonal matrices.

4. Exploring the Parallelization Options of ACELAN-COMPOS Solvers

Each iteration of the developed solver contains 2 sequentially called computationally intensive steps: $y = O \cdot h$ and $x = C^{-1} \cdot y$. Each of these steps can be paralleled. Calculation of $y = O \cdot h$ is multiplication of a block-band matrix (with six block diagonals) by a vector. It was decided to store each diagonal of this matrix in a separate array. The peculiarity of the matrices under study is that the upper three diagonals are transposed lower three diagonals. That is why they are stored only once, in three corresponding arrays.

Let us consider possible computation optimizations.

Multiplication of all block diagonals by a vector should be performed simultaneously (inside one loop) to reduce the number of readings. This can be helped by simultaneous (inside one loop) multiplication of two middle blocks marked in Fig. 6 by corresponding fragments of vector h. Note that these blocks use the same data of the three stored diagonal arrays. Then we can proceed to the next middle blocks (using the same data). In doing so, it is possible to parallelize the computations required for the first and the second pair. But on neighboring iterations there is a write access to the common part of the resulting vector. This is supported by OpenMP (reduction function for arrays) since version 4.5.

The calculation of $x = C^{-1} \cdot y$ at each iteration of the algorithm uses the decomposition $C = L \cdot D \cdot L^T$, which is performed beforehand. Thus, the computation of $x = C^{-1} \cdot y$ is reduced to the solution of system of linear algebraic equations with a two-diagonal matrix and the trivial inversion of the diagonal matrix.

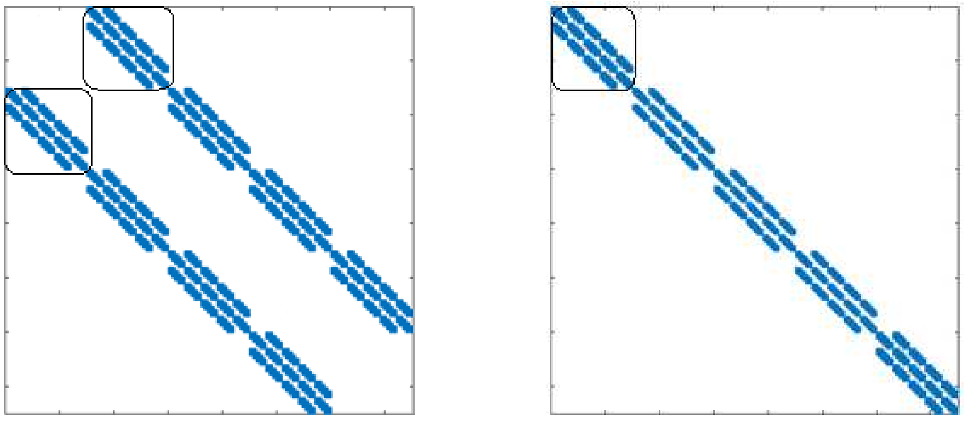

Figure 6: Block 6-diagonal and block 3-diagonal matrix.

Solving system of linear algebraic equations with a bi-diagonal matrix reduces to computing a loop with linear recurrent dependence. Parallel algorithms for computing such loops were described in the articles [29, 30]. In this work, the matrix of the system of linear algebraic equations has a special block structure (periodically zero blocks appear), which significantly improves the efficiency of parallelized. It is possible to solve system of linear algebraic equations for all middle blocks in parallel.

Numerical experiments were carried out for a matrix of size 125 with block size 5 and parallelized to 8 threads for two compilers (GCC 10.1.0 and Intel C++ Compiler 19.1.3.311) with the $-O3$ optimization flag. The results of the experiments (see Table 1) show that using loop parallelized leads to slowdown of the program. This can be explained by the small size of middle blocks which leads to the fact that different parallel threads will use the same cache line which will lead to overhead costs on synchronization.

Table 1: Results of numerical experiments.

	Block size 5	Block size 65
Sequential program	4.145	7.266
Parallel program	6.402	5.662

Numerical experiments were performed for a matrix of size 274625 with block size 65 with 8 threads parallelized for two compilers (GCC 10.1.0 and Intel C++ Compiler 19.1.3.311) with the $-O3$ optimization flag. The results of the experiments show that using loop parallelized leads to speeding up the program — cf. Table 1.

Parallelization does not always speed up programs – sometimes it slows them down. Addition and multiplication operations are performed quickly, and accesses to $DDR4$ memory are slow in modern processors [16].

Modern processors use a hierarchy of memory

$$DDR4->L3->L2->L1->R->ALU.$$

The closer the memory module is to the *ALU* – the faster access to this memory module. Previous publications could not take into account such a hierarchy of memory, since then no such an hierarchy existed.

The solution of a system of equations with a two-diagonal matrix is reduced to the calculation of a recurrent program loop — viz.

```
1. for $(j = 1, j < N, j = j + 1)$
2. {
3. $X[j] = A[j] * X[j - 1] + B[j]$
4. }
```

To speed up the calculation of this loop on an 8-Core processor, the intermediate values

$$X[N/8],\ X[2*N/8],\ X[3*N/8],\ X[4*N/8],\ X[5*N/8],\ X[6*N/8],\ X[7*N/8]$$

have been first calculated in parallel. These intermediate values are stored in the fast register memory. The remaining values of the array X are then calculated in parallel in $(N/8-1)$ step [30].

Only half of the elements of a symmetric or skew-symmetric matrix are stored in computer memory. If this data is placed in cache memory, then it is read from RAM only once, and the second time – from cache memory. This property gives additional acceleration, since the data of the two-diagonal matrix can be placed in the $L2$ or $L1$ cache.

5. Solver Acceleration Using Precompiler

Like in [26,28], we used a preliminary compiler (precompiler) to speed up the solver. The use of a precompiler is justified by the fact that optimizing compilers often poorly optimize high-performance programs [14,25]. We added optimization of linear expressions to the precompiler. These linear expressions can contain real numbers (replacing an arithmetic operation with constant operand containing the result of this operation, replacing the product of zero and another operand with a zero value, replacing the product of one and another operand with this operand), ternary expressions (they may occur in the loop headers when applying a loop merge), modulus and conditional operators.

The first half of the code below shows an example of a loop to index expressions in the body of which optimization can be applied. The result will be a loop located at the bottom of the code. The modified expressions are underlined in the original and transformed code fragments. The precompiler speeds up the solver by 1.25 times.

6. Building Block-Band Matrix in ACELAN-COMPOS Package

For electroelastic bodies V with inhomogeneous material properties the following system of differential equations is used [12]:

$$\rho\ddot{u} + \alpha_d\rho\dot{u} - \bigtriangledown\cdot\sigma = f, \qquad \bigtriangledown\cdot D = 0,$$

$$\sigma = c^E : (\varepsilon + \beta_d \dot{\varepsilon}) - e^T \cdot E, \quad \dot{D} + \zeta_d D = e : (\varepsilon + \zeta_d \dot{\varepsilon}) + k^S \cdot E,$$
$$\varepsilon = (\nabla u + \nabla u^T)/2, \qquad E = -\nabla \varphi,$$

where σ is the stress tensor, ρ is the body density, ε is the stain tensor, u is the displacement vector, D is the electric induction vector, E is the electric field vector, f is the mass forces vector, φ is the electric potential, α_d, β_d, ψ_d are the damping coefficients, c^E is the tensor of elastic stiffness moduli of the fourth rank, computed for constant electric filed (E), e is the tensor of piezoelectric moduli of the third rank, k^S is the tensor of dielectric permittivity of the second rank, calculated at constant strains (S), which is also often denoted by ε^S, $(...)^T$ is the transposition operation, $(...) : (...)$ is the double inner product operation for tensors.

Corresponding finite element model in vector form can be presented as

$$u(x,t) = N_u^T(x) \cdot U(t),$$
$$\varphi(x,t) = N_\phi^T(x) \cdot \Phi(t),$$

where N_u is the matrix of the shape functions for the displacement field, N_φ the vector of shape functions for the electric potential, and $U(t)$, $\Phi(t)$ the global vectors of the degrees of freedom.

```
\\source code
for (I_BLK = 0; I_BLK < BLKS_IN_ROW - 2; I_BLK++)
    for (i = 0; i < BL_H; i++)
        for (j = 0; j < BL_W; j++) {
            V[I * B_BL_H + I_BLK * BL_H + 0 * BL_H + i] += B[
    B_BLK_BIAS * BLOCK_INDEX + BLK_SZ_2 * (2 + 0 + 3 * I_BLK) + j *
    BL_H + i] * H[J * B_BL_W + j + BL_W + I_BLK * BL_W];
            V[I * B_BL_H + I_BLK * BL_H + 1 * BL_H + i] += B[
    B_BLK_BIAS * BLOCK_INDEX + BLK_SZ_2 * (2 + 1 + 3 * I_BLK) + j *
    BL_H + i] * H[J * B_BL_W + j + BL_W + I_BLK * BL_W];
            V[I * B_BL_H + I_BLK * BL_H + 2 * BL_H + i] += B[
    B_BLK_BIAS * BLOCK_INDEX + BLK_SZ_2 * (2 + 2 + 3 * I_BLK) + j *
    BL_H + i] * H[J * B_BL_W + j + BL_W + I_BLK * BL_W];
        }

\\result code
for (I_BLK = 0; I_BLK < BLKS_IN_ROW - 2; I_BLK++)
    for (i = 0; i < BL_H; i++)
        for (j = 0; j < BL_W; j++) {
            V[I * B_BL_H + I_BLK * BL_H + i] += B[B_BLK_BIAS *
    BLOCK_INDEX + BLK_SZ_2 * (2 + 3 * I_BLK) + j * BL_H + i] * H[J *
    B_BL_W + j + BL_W + I_BLK * BL_W];
            V[I * B_BL_H + I_BLK * BL_H + BL_H + i] += B[B_BLK_BIAS*
    BLOCK_INDEX + BLK_SZ_2 * (3 + 3 * I_BLK) + j * BL_H + i] * H[J*
    B_BL_W + j + BL_W + I_BLK * BL_W];
            V[I * B_BL_H + I_BLK * BL_H + 2 * BL_H + i] += B[
    B_BLK_BIAS * BLOCK_INDEX + BLK_SZ_2 * (4 + 3 * I_BLK) + j * BL_H
    + i] * H[J * B_BL_W + j + BL_W + I_BLK * BL_W];
        }
```

Figure 7: An example of precompiler using.

After the substitution $a = [U, \Phi]^T$, the problem can be written as the system of linear equations

$$M \cdot \ddot{a} + C \cdot \dot{a} + K \cdot a = F,$$

where M is the mass matrix, C the damping matrix, and K the stiffness matrix. In this paper we consider static problems, so that the above equation takes the form

$$K \cdot a = F,$$

where

$$K = \begin{pmatrix} K_{uu} & K_{u\varphi} \\ K_{u\varphi}^T & -K_{\varphi\varphi} \end{pmatrix}.$$

Note that K_{uu} is a symmetric positive semi-definite matrix describing mechanical properties, $K_{u\varphi}$ defines piezoelectric properties, and $K_{\varphi\varphi}$ dielectric properties and is also symmetric positive semi-definite. For mechanical problems, we can consider only K_{uu}, and for electrostatic problems only $K_{\varphi\varphi}$.

The solvers for the systems of linear equations presented in this article are meant to be used in ACELAN-COMPOS package for modeling of new composite materials. The problem of identification of properties of a composite material can be solved using homogenization method [19, 20]. The implementation of homogenization method includes solving a set of boundary problems for a representative volume of material. Representative volumes for composites with connectivity types 3-1, 3-0 and 3-3 are created as regular hexahedron-based finite element meshes in the ACELAN-COMPOS package. The regularity of meshes makes it possible to select specific numeration for nodes and degrees of freedom which leads to block-diagonal sparse global stiffness matrices in static boundary problems (Fig. 2).

Depending on the problem being solved, the structure of the blocks and the properties of the matrix change. In case of electrostatic, magnetostatic, and thermal conductivity problem each node corresponds to one degree of freedom, in problems of the theory of elasticity there are three unknowns at each node, and in coupled problems there could be four or more degrees of freedom. The matrices that arise when solving problems with one and three degrees of freedom turn out to be symmetric and positive definite, and in the case of coupled fields, saddle matrices that require more complex solution are obtained. Fig. 8 shows the block structure for problems with 3 and 4 degrees of freedom at each node of representative volume. It is possible to change structure of matrices by changing the way of numerating degrees of freedom. So, Fig. 9 shows the structure of matrix for the electroelasticity problem with four degrees of freedom at the node for two different numerations of the unknowns. As seen, it is possible to obtain a block-diagonal matrix or a saddle matrix consisting of two symmetric blocks located along the main diagonal and two rectangular blocks. For the problem with n nodes, the sizes of large blocks on the main diagonal will be $3n$ and n. Building blocks of the matrix can be performed independently in parallel.

As can be seen from Figs. 8 and 9 (left), at the level of large blocks, matrices for different physical problems have the same structure, although they have different properties. Changing the numbering of unknowns allows to split a matrix into even larger blocks. At

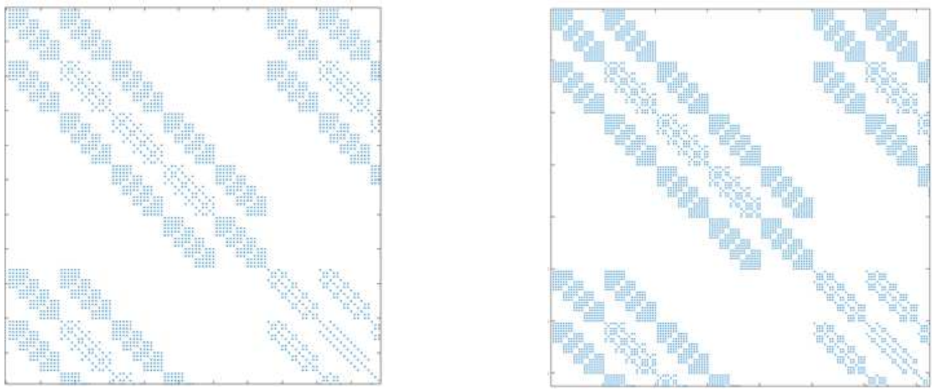

Figure 8: Distribution of non-zero elements in matrices with 3(left) and 4(right) degrees of freedom per node.

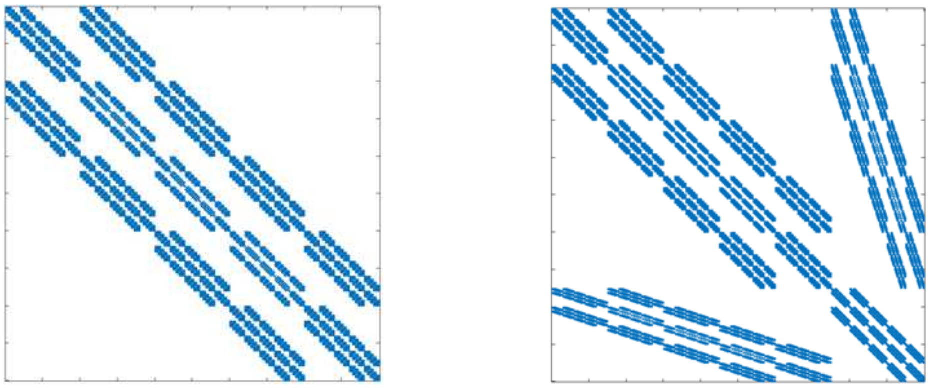

Figure 9: Distribution of non-zero elements in matrices built for electroelastic problem using two different ways of enumerating degrees of freedom.

the same time, in problems with field connectivity, in particular, in the case of electroelasticity, the values of the matrix elements can differ significantly: in the rows corresponding to mechanical variables, the diagonals will contain numbers of the order of 10^{10}, and the electrical variables will correspond to values of the order of 10^{-10}. This negatively affects the accuracy of calculations; various data normalization techniques are used to solve this problem. When numbered corresponding to Fig. 9 (right), values of the same order appear inside each large block. The block located in the upper left corner of the Fig. 9 (right) matrix completely coincides in structure and characteristics with the matrix shown in Fig. 8.

7. Conclusion and Further Research

We consider expanding the set of algorithms, where the proposed acceleration methods can be used. The article describes the construction of a high-performance system of linear algebraic equations solver with block-band matrices for the ACELAN-COMPOS application software package. Numerical experiments demonstrate high performance of the solver on modern processors.

The ACELAN-COMPOS package forms block-band matrices in which blocks have many zero elements in the upper-right and lower-left corners. Splitting these blocks into smaller ones can lead to more efficient matrix storage and performance.

Acknowledgments

The work was supported by the Government of the Russian Federation under Grant No. 075-15-2019-1928.

References

[1] Z.-Z. Bai, *Parallel matrix multisplitting block relaxation iteration methods*, Math. Numer. Sin. **17**(3), 238–252 (1995).

[2] Z.-Z. Bai, *Parallel hybrid iteration methods for block bordered linear systems*, Commun. Appl. Math. Comput. **86**(1), 37–60 (1997).

[3] Z.-Z. Bai, *A class of parallel decomposition-type relaxation methods for large sparse systems of linear equations*, Linear Algebra Appl. **282**(1-3), 1–24 (1998).

[4] Z.-Z. Bai, *A class of modified block SSOR preconditioners for symmetric positive definite systems of linear equations*, Adv. Comput. Math. **10**(2), 169–186 (1999).

[5] Z.-Z. Bai, *A class of parallel hybrid two-stage iteration methods for the block bordered linear systems*, Appl. Math. Comput. **101**(2-3), 245–267 (1999).

[6] Z.-Z. Bai, *Modified block SSOR preconditioners for symmetric positive definite linear systems*, Ann. Oper. Res. **103**(1), 263–282 (2001).

[7] Z.-Z. Bai, G.H. Golub and C.-K. Li, *Convergence properties of preconditioned Hermitian and skew-Hermitian splitting methods for non-Hermitian positive semidefinite matrices*, Math. Comp. **76**(257), 287–298 (2007).

[8] Z.-Z. Bai, G.H Golub, L.-Z. Lu and J.-F. Yin, *Block triangular and skew-Hermitian splitting methods for positive-definite linear systems*, SIAM J. Sci. Comput. **26**(3), 844–863 (2005).

[9] Z.-Z. Bai and J.-Y. Pan, *Matrix Analysis and Computations*, SIAM (2021).

[10] Z.-Z. Bai and Y.F. Su, *On the convergence of a class of parallel decomposition-type relaxation methods*, Appl. Math. Comput. **81**(1), 1–21 (1997).

[11] Z.-Z. Bai and D.-R. Wang, *Parallel multilevel iterative methods*, Linear Algebra Appl. **250**, 317–347 (1997).

[12] A.V. Belokon', A.V. Nasedkin and A.N. Solov'yev, *New schemes for the finite-element dynamic analysis of piezoelectric devices*, J. Appl. Math. Mech. **66**(3), 481–490 (2002).

[13] L.R. Gervich et al., *How OPS (Optimizing Parallelizing System) may be useful for Clang*, in: *Proceedings of the 13th Central and Eastern European Software Engineering Conference in Russia*, Association for Computing Machinery (2017).

[14] Z. Gong et al., *An empirical study of the effect of source-level loop transformations on compiler stability*, in: *Proceedings of the ACM on Programming Languages*, **2**, pp. 1–29, OOPSLA (2018).

[15] V.S. Goun, V.S. Morozova and V.L. Polyatsko, *The general purpose system for construction of two–dimensional orthogonal grids*, Math. Models Comput. Simul. **29**(11), 71–88 (2017).

[16] S. Graham et al., *Getting Up to Speed: The Future of Supercomputing*, National Academies Press (2005).

[17] A.A. Korotkov, V.A. Pervichko, I.G. Plontikova and V.V. Chudanov, *Method of Orthogonal Mesh Generating on Plane or Smooth 3D-Surfaces*, IBRAE RAS (2006).

[18] N.V. Kurbatova, D.K. Nadolin, A.V. Nasedkin, P.A. Oganesyan, and A.N. Soloviev, *Finite element approach for composite magneto-piezoelectric materials modeling in ACELAN-COMPOS package*, in: *Analysis and Modelling of Advanced Structures and Smart Systems*, pp. 69–88, Springer (2018).

[19] A.V. Nasedkin, P.A. Oganesyan, and A.N. Soloviev, *Analysis of Rosen type energy harvesting devices from porous piezoceramics with great longitudinal piezomodulus*, ZAMM Z. Angew. Math. Mech. **101** (2020).

[20] A.V. Nasedkin, P.A. Oganesyan, A.N. Soloviev, A.B. Kudimova, and D.K. Nadolin, *Finite element homogenization models of bulk mixed piezocomposites with granular elastic inclusions in ACELAN package*, Mater. Phys. Mech. **37**, 1, 25–33 (2018).

[21] Optimizing Parallelization System (Accessed 30.09.2021), `www.ops.rsu.ru`

[22] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press (1984).

[23] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM (2003).

[24] SoC Esperanto (Accessed 30.09.2021), `https://www.esperanto.ai/technology/`

[25] B.Ya. Steinberg and O.B. Steinberg, *Program transformations are the fundamental basis for creating optimizing parallelizing compilers*, Prog. Sys.: Theor. Appl. **12**(1), 21–113 (2021).

[26] B.Ya. Steinberg, O.B. Steinberg, E.A. Metelitsa, A.A. Vasilenko, V.V. Veselovskiy, N.A. Zhivykh, *Precompiler for the ACELAN-COMPOS package solvers*, in: *PaCT 2021 16th International Conference on Parallel Computing Technologies, September 13-18, 2021, Kaliningrad, Russia*, (2021).

[27] B.Ya. Steinberg, O.B. Steinberg and A.A. Vasilenko, *The loop fusion for data localization*, Prog. Sys. Theor. Appl. **11**(3), 17–31 (2020).

[28] B.Ya. Steinberg, A.A. Vasilenko, V.V. Veselovskiy and N.A. Zhivykh, *Solvers for systems of linear algebraic equations with block-band matrices*, B. SUSU MMCS **14**(3), 106–112 (2021).

[29] O.B. Steinberg, *The parallelizing of recurrent loops with using of non-regular superpositions computations*, B. High. Ed. Inst. N. Cauc. Reg. Nat. Sci. **2**, 16–18 (2009).

[30] O.B. Steinberg, *Parallelization of recurrent loops due to the preliminary computation of superpositions*, B. SUSU MMCS, **13**(3), 59–67 (2020).