On parallel algebraic multilevel preconditioner with approximate inversion by basis matrix method

V.O. Bohaienko[1][0000-0002-3317-9022]

[1] V.M. Glushkov Institute of Cybernetics of NAS of Ukraine, Glushkov ave., 40, Kyiv, 03187, Ukraine

sevab@ukr.net

Abstract. The paper presents a parallel calculation scheme for distributed memory systems for a modification of the AMLI preconditioner that performs incomplete inversions of matrix blocks by a technique that produces approximate inverse of a matrix by an orthogonalization procedure, referred to as the incomplete basis matrix method. Theoretical estimates of algorithm's performance and the results of its experimental testing are presented. The results of experimental analysis of preconditioner parameters influence on the convergence of the iterative algorithm are given. The characteristics of the proposed algorithm are compared with those of known algorithms implemented in the *hypre* software package. The tests on the matrices obtained from finite element discretization of soil stress-strain state modelling problem showed that the proposed algorithm has better performance for significantly ill-conditioned linear systems when solving them on a small number of computational resources.

Keywords. parallel algorithm, distributed memory systems, linear algebraic systems, preconditioner, algebraic multilevel iteration method

1 Introduction

The need to solve linear algebraic systems arises in mathematical modelling of most physical processes, in particular when modelling soil stress-strain state. In many cases, linear systems have ill-conditioned sparse matrices of large size. In these situations, iterative methods, such as conjugate gradient method (CG) or biconjugate gradient stabilized method (BiCGstab), are most commonly used [1].

The main method used to improve the convergence and the accuracy while solving ill-conditioned linear systems is the use of preconditioners [1, 2] - matrices, whose multiplication on linear system matrix leads to the decrease of condition number. The most commonly used methods for constructing preconditioners are incomplete matrix decomposition (e.g. ILU decomposition [1]), incomplete matrix inversion (e.g. polynomial preconditioners [3]), Geometric and Algebraic Multigrid (AMG) [4] methods, the Algebraic Multilevel Iteration method (AMLI) [5,6], and wavelet preconditioners [7,8].

The so-called basis matrix method [9, 10] is one of the methods developed to analyse ill-conditioned and rectangular linear systems. This method constructs iteratively the inverse of a given matrix using some orthogonalization procedure. Based on this method, an incomplete inversion algorithm is created, that can be used as a preconditioner. Further, it is combined [11] with the AMLI preconditioner and applied for solving elasticity problems.

The speed of the AMLI preconditioner, similarly to the multigrid ones, in particular the AMG, is low comparing to the incomplete decomposition and the incomplete inversion preconditioners and its application takes most of the time spent to obtain solutions of linear systems. This makes crucial the development of parallel schemes for these algorithms. While the AMLI is shown to have good scalability on vector and parallel-vector machines [12, 13], its parallel implementations on cluster systems are poorly studied.

2 Preconditioner on the base of the AMLI and the basis matrix method

The AMLI preconditioner [5] is based on the representation of an input matrix as having 2x2 blocks with its subsequent LU factorization:

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}=\begin{bmatrix} I & \\ A_{21}A_{11}^{-1} & I \end{bmatrix}\begin{bmatrix} A_{11} & A_{12} \\ & S \end{bmatrix} \tag{1}$$

where $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ is the Schur complement of A.

After approximating the Schur complement with a polynomial and the recursive application of the expansion (1) to the matrix A_{22}, the following algorithm for constructing a preconditioner that approximates the matrix A^{-1} is obtained:

$$\begin{cases} S^{(k-1)} \approx A_{22}^{(k-1)}\left[I - P_v\left(\left(M^{(k-1)}\right)^{-1} A_{22}^{(k-1)}\right)\right]^{-1} \\ M^{(k)} = \begin{bmatrix} I & \\ A_{21}^{(k)}\left(A_{11}^{(k)}\right)^{-1} & I \end{bmatrix}\begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & S^{(k-1)} \end{bmatrix} \end{cases}. \tag{2}$$

For the matrix A of the size $dimA = [K*L, K*L]$ that is divided into $L*L$ blocks $\overline{A}_{ij}, i, j = 1,...,L$ of the size $dim\overline{A}_{ij} = [K,K]$ we have

$$A_{11}^{(k)} = \overline{A}_{L-k,L-k}, dimA_{11}^{(k)} = [K,K],$$
$$A_{12}^{(k)} = (\overline{A}_{L-k,L-k+1},...,\overline{A}_{L-k,L}), dimA_{12}^{(k)} = [K,kK],$$
$$A_{21}^{(k)} = (\overline{A}_{L-k+1,L-k},...,\overline{A}_{L,L-k}), dimA_{21}^{(k)} = [kK,K],$$

$$A_{22}^{(k)} = \begin{bmatrix} \bar{A}_{L-k+1,L-k+1} & \cdots & \bar{A}_{L-k+1,L} \\ \cdots & \cdots & \cdots \\ \bar{A}_{L,L-k+1} & \cdots & \bar{A}_{LL} \end{bmatrix}, dimA_{22}^{(k)} = [kK, kK],$$

$$dimS^{(k-1)} = dimA_{22}^{(k)}, S^{(0)} = \bar{A}_{LL}^{-1}, dimM^{(k)} = [(k+1)K,(k+1)K], k = 1,...,L-1$$

and $M^{(L-1)}$ is an approximate inverse of A.

The AMLI algorithm can be optimally stabilized by choosing the degree of an approximating polynomial [12]. However, the applicability of this theory while solving stress-strain problems described by hyperbolic differential equations wasn't proved. As such problems are of our specific interest, we consider the simplest case of $P_v(t) = P_1(t) = 1-t$. Then, the Schur complement approximation becomes $S^{(k-1)} \approx M^{(k-1)}$ and the recursive scheme (2) simplifies to

$$M^{(k)} = \begin{bmatrix} I & \\ A_{21}^{(k)}\left(A_{11}^{(k)}\right)^{-1} & I \end{bmatrix} \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ & M^{(k-1)} \end{bmatrix}. \tag{3}$$

In the scheme (3), the matrices $\left(A_{11}^{(k)}\right)^{-1}$ can be approximated by incomplete decomposition methods such as the ILU factorization [14] or incomplete Gram-Schmidt method [15]. To make the scheme more efficient while solving ill-conditioned linear systems, we propose to approximate them by the incomplete basis matrix method that is specifically developed for this case and is described in detail in [16].

Summarizing, the sequential computational scheme for the preconditioning procedure (3) is as follows. The matrix A is divided into $(N+1)\times(N+1)$ blocks the following way:

$$A = \begin{bmatrix} A_{11}^{(0)} & A_{121}^{(0)} & \cdots & A_{12N}^{(0)} \\ A_{211}^{(0)} & \cdots & \cdots & \cdots \\ \cdots & \cdots & A_{11}^{(N-1)} & A_{12N}^{(N-1)} \\ A_{21N}^{(0)} & \cdots & A_{21N}^{(P-1)} & A_{11}^{(N)} \end{bmatrix},$$

$$A_{12}^{(i)} = \left(A_{12,i+1}^{(i)},...,A_{12N}^{(i)}\right), A_{21}^{(i)} = \left(A_{21,i+1}^{(i)},...,A_{21N}^{(i)}\right).$$

Considering the application of the preconditioner (3) as a solution of the linear system $M^{(N)}x = b, x = (x_0,...,x_N), b = (b_0,...,b_N)$, it can be performed by a two-stage procedure, V-cycle, consisting of a "direct" and a "reverse" part.

On the ``direct`` part of the procedure, values of the following vectors are sequentially computed:

$$w_0 = \tilde{A}_{11}^{(0)} b_0, w_i = b_i - A_{21}^{(i)} \tilde{A}_{11}^{(i)} w_{i-1}, i = 1,...,N. \tag{4}$$

The ``reverse`` part consists in the sequential computation of the resulting vector

$$x_N = \tilde{A}_{11}^{(N)} w_N, x_i = w_i - \tilde{A}_{11}^{(i)} A_{12}^{(i)} (x_{i+1},...,x_N), i = 0,...,N-1. \tag{5}$$

In (4),(5), matrices $\tilde{A}_{11}^{(i)}$ are the approximates to the corresponding inverse matrices obtained by the incomplete basis matrix method [16].

As the initial matrix is sparse, we use the compressed row storage (CSR) scheme for all the matrices.

The approximation accuracy is controlled on the iterations of the algorithm by calculating $\varepsilon_1 = \max_i \left(A\tilde{A} \right)_{ii}$ for all the processed rows. When ε_1 becomes bigger than a given threshold value, we perform complete inversion. As the resulting matrices aren't always dense, we also use the CSR storage scheme doing complete inversion.

3 Parallel implementation of the preconditioner

To make the AMLI algorithm combined with matrix blocks incomplete inversions using the basis matrix method run in parallel on distributed memory systems, the following data partitioning scheme can be applied. Each of i row blocks

$$\left(A_{21i}^{(0)}, ..., A_{21i}^{(i-1)}, A_{11}^{(i)}, A_{12,i+1}^{(i)}, ..., A_{12N}^{(i)} \right)$$

is distributed on a system of P processes by blocks of rows. Thus, each process stores blocks of rows of the submatrices $A_{11}^{(i)}, A_{12}^{(i)}, A_{21j}^{(i)}$ and can perform multiplications that arise while applying the preconditioner (3) in parallel.

The computational scheme that we use for multiplying a sparse matrix A of size $M \times M$ by a vector v has the following form:

1. The process p stores the block $B_p = [r_{p0,} r_{p1}],\ \bigcup_{i=0}^{P} B_i = [0, ..., M],\ B_i \bigcap B_j = \varnothing, \forall i, j$ of the matrix A rows and the corresponding elements of the vector v;
2. Denote a set of columns of the matrix A in which non-zero elements are present in the row block B_p as C_p;
3. The process p sends values of the elements of the vector v from the index set $C_i \bigcap B_p$ to the processes $i, C_i \bigcap B_p \neq \varnothing$ and receives the values of the elements from the index set $C_p \bigcap B_j$ from the processes $j, C_p \bigcap B_j \neq \varnothing$;
4. After the synchronization is performed on the step 3, each of the processes independently multiplies the block of rows of the matrix A by the vector v. The result of this multiplication has the same partitioning as the vector v has.

Performing the ``direct`` part (4) of the AMLI algorithm, when vector multiplications by the matrices $A_{21}^{(i)}$ are organized as successive multiplications by the submatrices $A_{21j}^{(i)}$, the synchronizations are done once according to the fill-in pattern of the matrix $A_{21}^{(i)}$. Similarly, when the non-changeable component x_i of the result vector x is multiplied by the submatrices $A_{12i}^{(j)}, j < i$ during the ``reverse`` part (5), the synchronization can be performed only once according to the combined fill-in pattern of the submatrices $A_{21i}^{(i)}$.

The matrices $\tilde{A}_{11}^{(i)}$ can be obtained in parallel the following way:

- The process p handles the sequence $k_{p2} > k \ge k_{p1}$ of the matrices $A_{11}^{(k)}$. The sequences are constructed to make the processes evenly loaded as far as it is possible. This is ensured by approximately the same number of non-zero elements in the matrices $A_{11}^{(k)}$ that are processed by each of the processes.
- The processes exchange the values of the elements of the matrices $A_{11}^{(i)}$ to make the matrices $A_{11}^{(k)}, k_{p2} > k \ge k_{p1}$ completely stored in the memory of the process p. The process p sends row blocks B_p of the matrices $A_{11}^{(k)}, k_{i2} > k \ge k_{i1}$ to the processes $i, i \ne p$ and receives row blocks B_i of the matrices $A_{11}^{(k)}, k_{p2} > k \ge k_{p1}$ from them.
- Each of the processes independently performs a complete or incomplete inversion of the matrices.
- The processes exchange the values of the elements of the matrices $\tilde{A}_{11}^{(i)}$ to restore the initial data partitioning. The process p sends row blocks B_i of the matrices $\tilde{A}_{11}^{(k)}, k_{p2} > k \ge k_{p1}$ to the processes $i, i \ne p$ and receives row blocks B_p of $\tilde{A}_{11}^{(k)}, k_{i2} > k \ge k_{i1}$ matrices from them.

Since the data partitioning of the blocks of rows within the matrices $A_{12}^{(i)}, A_{21i}^{(j)}$ may differ for the different values of i, an additional synchronization should be performed when vectors are multiplied by these matrices. Denote a block of rows from the submatrices $A_{12}^{(i)}, A_{21i}^{(j)}$ that is handled by the process p as B_{pi}. Then, before multiplying a vector v with the partitioning $B_i = (B_{0i}, \ldots, B_{Pi})$ by the matrix $A_{21}^{(i)}$, we need to synchronize its elements to make each of the processes store the block $B_{pi} \bigcup \ldots \bigcup B_{pN}$. To do so, the process p sends the block $B_{pi} \bigcap B_{rj}$ of the elements of the vector to the process r if $B_{pi} \bigcap B_{rj} \ne \varnothing, j \ne i$. Then, the process p receives a corresponding block from the process r if $B_{pj} \bigcap B_{ri} \ne \varnothing, j \ne i$.

Similar procedures should be performed when a vector $v = (v_{i+1}, \ldots, v_N)$ is multiplied by the matrix $A_{12}^{(i)}$ for each of its components. When the non-changeable component x_j is multiplied by the matrices $A_{12}^{(i)}$ during the ``reverse`` part (5) of the AMLI algorithm, the synchronization can be performed once before calculating the result of the first multiplication operation.

When the preconditioner is used with an iterative method for solving a linear algebraic system, we perform a calculation of the matrices $\tilde{A}_{11}$ and the sets of the elements that need synchronization once on the first iteration. The execution of these operations will be further denoted as ``initialization procedure".

4 Estimation of algorithms performance

We estimate the performance of the considered algorithms on the base of the following assumptions:

- the matrices $A_{11}^{(i)}$ have the same fill-in factor k_{11}, and the matrices $A_{12j}^{(i)}, A_{21j}^{(i)}$ - the same fill-in factor k_2 which does not dependent on the block size;
- A percentage k_f of the matrices $A_{11}^{(i)}$ for which the full inversion procedure is applied does not depend on the block size;
- The inverse matrices of $A_{11}^{(i)}$ are completely filled;
- The number of elements that each of the processes must synchronize before performing a parallel multiplication is considered proportional (with the coefficient of proportionality equal to k_s) to the number of non-zero elements in the block of matrix rows. We assume that each of the processes performs only one pair of data exchange operations doing the synchronization;
- The submatrices are considered evenly distributed across the system.

Under these assumptions, the time sequential algorithm spend to perform calculations on (4), (5) can be estimated as

$$T_1(n,N) = k_p[\frac{n^2}{N}\left(2(k_f + (1-k_f)k_{11}) + (N-1)k_2\right) + 4n + n(N-1)] \tag{6}$$

where k_p is the performance coefficient.

The minimal time is achieved here for the number of blocks

$$N = N_m = \sqrt{\left(2\left(k_f + (1-k_f)k_{11}\right) - k_2\right)n}. \tag{7}$$

The time spent on computation of the matrices $\tilde{A}_{11}^{(k)}$ with an assumption that the computational complexity of one step of the incomplete basis matrix method is proportional to the maximal number of non-zero elements in the rows of the matrices $A_{11}^{(k)}$ can be estimated as follows:

$$T_{i1}(n,N) = Nk_{p2}\left(k_f\frac{n^3}{N^3} + k_{11}\frac{n^2}{N^2}\right) \tag{8}$$

where k_{p2} is the performance coefficient.

We evaluate the computation time of the parallel algorithm using the linear model of a pairwise data exchange operation:

$$T_s(n) = n/k_e + k_a \tag{9}$$

where k_e is the bandwidth, n is the data size, k_a is the latency.

Taking into account (9), the total time spent by the parallel algorithm on performing calculations on (4), (5) can be estimated as

$$\begin{aligned} T_P(n,N) = \frac{1}{P}T_1(n,N) + 2\frac{n}{k_e}\left[\frac{k_s}{P}((1-k_f)k_{11} + \frac{N-1}{N}k_2) + \frac{P-1}{P}k_f\right] + \\ +(2N(2+k_f P - 2k_f) - 2)k_a \end{aligned} \tag{10}$$

where P is the number of processes.

The minimal time spent on data exchanges for a fixed N is achieved when the number of processes is equal to

$$P_m(N) = \sqrt{\frac{2nk_s((1-k_f)k_{11} + \frac{N-1}{N}k_2)}{2k_e k_a k_f N}}. \tag{11}$$

When doing parallel computations, the time spent to calculate the matrices $\tilde{A}_{11}^{(k)}$ can be estimated with the assumption that each of the processes performs incomplete or complete inversion of $\frac{N}{P}$ submatrices (we consider this number to be integer) as

$$T_{iP}(n,N) = \frac{1}{P}T_{i1}(n,N) + \frac{N}{P}(P-1)(\left(\frac{k_{11}n^2}{N^2 P k_e} + k_a\right) + \left(((1-k_f)k_{11} + k_f)\frac{n^2}{N^2 P k_e} + k_a\right)). \tag{12}$$

5 Experimental analysis of the algorithm's performance

We apply the proposed parallel modification of the preconditioner to solve linear systems arising while modelling stress-strain state of the soil under a dam using the model of dynamic consolidation of saturated soils as described in [11].

The two-dimensional model has the form of the hyperbolic differential equations' system with respect to 8 unknown function - the horizontal and the vertical components u_w, v_w, u_{sk}, v_{sk} of water and soil skeleton displacement vectors and their first derivatives. The used boundary and the initial conditions, solution domain, and parameter values are given in [11]. The problem was discretised by the Galerkin finite element method on a rectangular mesh with respect to the space variables and by the Crank-Nicolson scheme with respect to the time variable.

While forming linear systems, their rows were ordered to form blocks related to a single unknown function. To minimize a number of synchronizations we use the

Cuthill-McKee ordering of mesh nodes. Performing the diagonal scaling of rows we obtain matrices that have the following properties: non-symmetricity; loosely coupled block structure; high fill-in factor; high condition number due to different speeds of the simulated processes. As in the considered model of soil stress-strain state we have 8 unknown functions, the matrices can be easily represented as 2x2 block matrices. Such properties of matrices make them an appropriate testing case for the AMLI preconditioner that is built upon the idea of block matrix subdivision and thus can be efficient working with matrices that have loosely coupled diagonal blocks in their structure. The usage of the incomplete basis matrix method here is beneficial because matrices contain significantly ill-conditioned blocks and other methods like incomplete LU decomposition can be unstable in this case.

The performance and the convergence of the considered algorithm were experimentally studied on the SCIT-3 cluster of the Institute of Cybernetics of National Academy of Sciences of Ukraine. The eight-core nodes based on Intel Xeon 5345 processors with 2GB RAM per core were used.

The algorithm was tested on the matrix which arises on the first time step with a length $\tau = 2hours$. This matrix of this linear system, further denoted as SD, has the size equal to 107856×107856 with an average number of non-zero elements in a row equal to 23.42 . Its condition number estimated by the LSMR algorithm [17] has an order of 10^{12} .

Linear system was solved using the BiCGStab [1] algorithm with the proposed modification of the AMLI preconditioner. The BiCGStab was used because of non-symmetric nature of the matrix. The submatrices $A_{11}^{(k)}$ were completely inverted when $\varepsilon_1 < 0.05$. This resulted in the complete inversion of $: \; 50\%$ of the submatrices. The acceptable accuracy of the calculations (here and further - the value below which the residual has to be reduced on BiCGStab iterations) was equal to 10^{-9} .

The use of the BiCGStab algorithm itself, or with the procedure of a diagonal scaling of rows, did not allow obtaining solutions of the required accuracy. The computation time when the BiCGStab was used without the usage of any preconditioner was equal to 140 ms per iteration.

To experimentally assess the accuracy of the estimates (6) and (7) of single preconditioner application execution time on a single processor core without the initialization procedure, the linear system SD was solved with the variable size of blocks on which the matrix is split. The values of the estimates' coefficients were, here and further, found using the least squares method on the base of the measured execution time. The block size which minimizes the computation time was between 125 and 500, which corresponds to the calculated value of the theoretical estimate (7) equal to 179. The obtained experimental data confirm the properties of the algorithm reflected in the estimate (6) whose accuracy was not worse than 25%.

The performance estimate (10) in the similar case for the parallel algorithm was verified measuring the time spent on data exchanges between processes. The estimation error here remains within 30% for the block size ranging from 250 to 3000 and the number of processes ranging from 12 to 24. In other cases, the error increases. For a fixed block size, the optimal number of processes, that ensures the minimal time spent on exchanges, in most cases, was slightly lower than the values calculated by the estimate (11).

The time spent on the calculations of the matrices $\tilde{A}_{11}^{(k)}$ (initialization procedure) depending on the block size in the case of the sequential algorithm was measured for the block size ranging from 125 to 4000. Then, the time was assessed by the estimate (8). The accuracy of the estimate was within 12% for the block size less than 3000, while the block size increase leads to the increase of the error. These inaccuracies may be caused by a change in the percentage of fully inverted submatrices. In the case of the parallel algorithm, the corresponding time was measured and estimated by (12). The general behaviour of the estimate (12) coincides with the measured time (Fig.1) but the estimation errors were high.

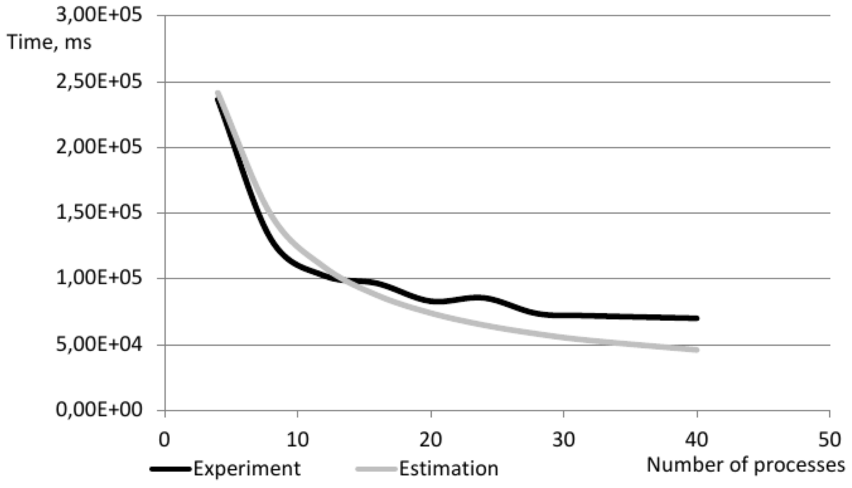

Fig. 1. Time, ms, spent on the calculations of $\tilde{A}_{11}^{(k)}$ matrices and its estimate in the case of the parallel algorithm

Further, we studied an influence of the size of blocks on which the matrix is split on the iterative method's convergence. The number of iterations required to achieve 10^{-9} order of accuracy was measured along with the overall time spent by the BiCG-stab algorithm and the proposed preconditioner running on a single processor core. Here, an increase of the block size leads to a decrease of the fill-in factor of the matrices $A_{11}^{(k)}$. As the incomplete basis matrix method performs better in terms of accuracy on highly filled sparse matrices, the decrease of the fill-in factor results in the increase of the number of completely inverted matrices. This accelerates the convergence of the iterative algorithm (Fig. 2) along with the significant increase of its running time. The block size for which the required accuracy is reached in the minimal time was in the range from 75 to 125 elements.

The decrease of the number of fully inverted submatrices $A_{11}^{(k)}$ reduces the time spent on preconditioner initialization and thus significantly decreases the running time the parallel algorithm in the case when a small number of computational resources are used. On the other hand, the speed-up of the initialization operation is higher for the higher values of k_f, so the time spent on initialization becomes close to equal for the different values of k_f when the number of processes increase. The time spent on the parallel calculations on (4), (5) behaves likewise. The smaller number of completely inverted submatrices, in general, accelerates this operation, but the speed-up decreases with the increase of the number of computing resources involved. All this, given that convergence is faster for a larger number of completely inverted submatrices, leads to the situation when k_f reduction is effective only when a small number of processes are involved.

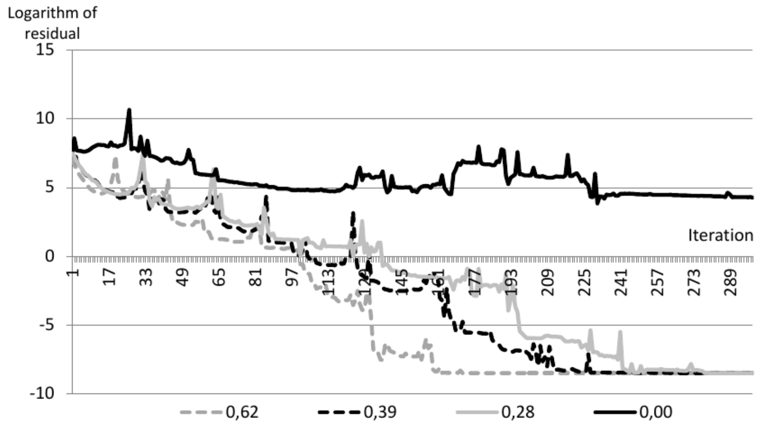

Fig. 2. The logarithm of residual at the iterations of the algorithm for different percentages of completely inverted submatrices

The efficiency of the proposed parallel preconditioner was compared with the efficiency of known algorithms implemented in the hypre software package v.2.10.0b [18]. The only combination of an iterative method and a preconditioner implemented in the hypre which allowed obtaining solutions of 10^{-9} accuracy order was the Gmres [1] and the AMG preconditioner [3] (we will further denote this pair as the Gmres+AMG). The total time needed to obtain the solution of the linear system SD with 10^{-9} order of accuracy is given in Fig. 3. Here the proposed parallel preconditioner was used with the block size equal to 250 and 500. It was faster than the Gmres+AMG when the number of processes was less than 20 with the block size equal to 500. However, the scalability of the proposed parallel algorithm was lower comparing with the Gmres+AMG.

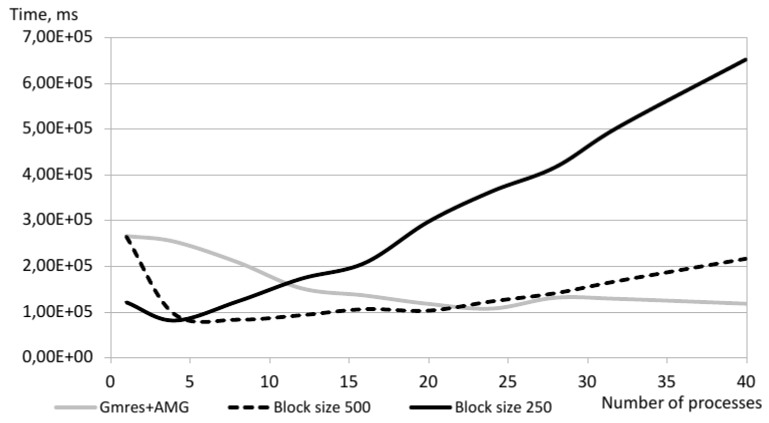

Fig. 3. Computation time, ms, subject to the number of processes

6 Conclusions

The parallel computation scheme was constructed for the AMLI preconditioner with incomplete inversion of matrix blocks using the so-called basis matrix method and the theoretical estimates of its performance were obtained and experimentally verified.

For the linear system obtained while modelling stress-strain state of soil which has the size equal to 107856×107856, the condition number of 10^{12} order, the maximal achieved speed-up was 24% when scaling the algorithm on 8 processes for the block size equal to 500 elements. The low scalability of the algorithm here can be explained by the large number of data exchange operations that must be performed before each matrix-vector multiplication.

We compared the characteristics of the proposed preconditioner with the AMG used as preconditioner in the Gmres algorithm implemented in the hypre software package. The scalability of the proposed algorithm was generally worse, but it had better performance for the considered significantly ill-conditioned linear system when solving it on less than 20 processor cores.

References

1. Saad, Y.: Iterative methods for sparse linear systems, 2 edition. Society for Industrial and Applied Mathematics (2003). doi: 10.1137/1.9780898718003
2. Chen, K.: Matrix Preconditioning Techniques and Applications. Cambridge University Press (2005). doi: 10.1017/CBO9780511543258
3. Saad, Y.: Practical use of polynomial preconditionings for the conjugate gradient method. SIAM J. Sci. Stat. Comp. **6**, 865-881 (1985). doi:10.1137/0906059
4. Shapira, Y.: Matrix-Based Multigrid: Theory and Applications. Springer Science & Business Media (2003).
5. Mense, C., Nabben, R.: On algebraic multilevel methods for non-symmetric systems - convergence results. Electron. Trans. Numer. Anal. **30**, 323-345 (2008).

6. Bayramov, N.R., Kraus, J.K.: Multigrid methods for convection–diffusion problems discretized by a monotone scheme. Comput. Method. Appl. M. **317**, 723-745 (2017). doi:10.1016/j.cma.2017.01.004
7. Chan, T., Tang, W., Wan, W.: Wavelet sparse approximate inverse preconditioners. BIT Numerical Mathematics **37**, 644-660 (1997). doi:10.1007/BF02510244
8. Bianchi, D., Buccini, A.: Generalized Structure Preserving Preconditioners for Frame-Based Image Deblurring. Mathematics **8(4)**, 468 (2020). doi:10.3390/math8040468
9. Kudin, V.I., Liashko, S.I., Kharytonenko, N.V., Iatsenko, Iu.P.: Analysis of the properties of a linear system using the method of artificial basis matrices. Cybernetics and Systems Analysis **43(4)**, 563-570 (2007). doi:10.1007/s10559-007-0081-3
10. Bogaienko, V.O., Kudin: V.I., Skopetsky, V.V.: Aspects of the organization of computations using the basis-matrix method. Cybernetics and Systems Analysis **48(4)**, 30-34 (2012). doi:10.1007/s10559-012-9441-8
11. Bohaienko, V.O.: Using Basis Matrix Method with AMLI Preconditioner for Solving Elasticity Problems. In: Proceedings of the 4th International Scientific Conference of Students and Young Scientists ``Theoretical and Applied Aspects of Cybernetics``; Kyiv, Ukraine, pp.24-28 (2014).
12. Axelsson, O., Barker, V.A., Neytcheva, M., Polman, B.: Solving the Stokes Problem on a Massively Parallel Computer. Math Model Anal. **6**, 7-27 (2001).
13. Li, Zh., Wu, Sh., Xu, J., Zhang, Ch.: Toward Cost-Effective Reservoir Simulation Solvers on GPUs. Adv. Appl. Math. Mech. **8(6)**, 971-991 (2016). doi:10.4208/aamm.2015.m1138
14. Saad, Y.: Schur Complement Preconditioners for Distributed General Sparse Linear Systems. In: Domain Decomposition Methods in Science and Engineering XVI, Lecture Notes in Computational Science and Engineering, 55, pp. 127-138 (2007).
15. Liu, Q.H., Zhang, F.D.: Incomplete hyperbolic Gram-Schmidt-based preconditioners for the solution of large indefinite least squares problems. J. Comput. Appl. Math. **250**, 210-216 (2013). doi:10.1016/j.cam.2013.02.016
16. Bohaienko, V.O., Kudin, V.I.: Building preconditioners using basis matrix method. International Journal ``Information Content and Processing`` **1(2)**, 182-187 (2014).
17. Fong, D.C.-L., Saunders, M.: LSMR: An Iterative Algorithm for Sparse Least-Squares Problems. SIAM J. Sci. Comput. **33(5)**, 2950-2971 (2011). doi:10.1137/10079687X
18. Falgout, R.D., Yang, U.M.: hypre: A Library of High Performance Preconditioners. In: Sloot PMA, Hoekstra AG, Tan CJK, Dongarra JJ (editors). Lecture Notes in Computer Science, vol 2331, Wien: Springer Verlag, pp. 632-641 (2002).