Architecture Solutions for DDM Numerical Environment*

Yana Gurieva[1] and Valery Ilin[1,2]

[1] Institute of Computational Mathematics and Mathematical Geophysics SB RAS, Novosibirsk, Russia
[2] Novosibirsk State University, Novosibirsk, Russia

Abstract. Domain decomposition methods (DDM)for parallelized solving of large SLAEs with sparse matrices, arising from approximations of multi-dimensional boundary value problem (BVP) at quasi-structured grids, on a heterogeneous multi-processor system (MPS) with distributed and hierarchical shared memory, are proven to be a manifold actual computational area. In an algebraic sense, it includes a two-level multi-preconditioned iterative process in Krylov subspaces. The efficiency of a numerical environment and the corresponding software performance are significantly defined by the quality of various parallelization techniques' usage: Message Passing Interface (MPI) system to arrange the upper level of iterations over subdomains, multi-threaded computations in a simultaneous solution to the auxiliary subsystems by OpenMP (technologies for multi-core CPUs), vectorization of the machine's operations by AVX instruction set, and a usage of graphic accelerators GPGPU with CUDA Nvidia architecture. This paper describes some architecture solutions used in Krylov software library of algebraic solvers.

Keywords: domain decomposition, distributed matrix, iterative algorithm, exchange buffers

1 Introduction

Solution to systems of linear algebraic equations (SLAEs) is an actual indispensable stage of various processes and phenomena mathematical modelling. The most important and interesting case is one with large sparse matrices which arise in a finite difference, finite element, and finite volume method approximations of the multi-dimensional boundary values problems (BVPs) at some non-structured grid. For a big number (10^9 and more) of the degrees of freedom (d.o.f.) such problem is a bottleneck for a large-scale numerical experiment because of the non-linear dependence of the computational resources on the SLAE's dimension.

The main universal approach to the parallel solution of such a problem is based on the additive domain decomposition method (DDM) in the Krylov subspaces, which presents a two-level iterative process of the block Jacobi-Schwarz

* The work has been supported by grant N 16-29-15122-ofi-m of Russian Foundation for Basic Research and by Russian Science Foundation grant N 15-11-10024

type [1]. In this approach, speaking in the geometric language, the grid computational domain is divided into its grid subdomains with a parametrized overlapping, in general. The original grid-algebraic problem

$$Au = f, \;\; A = \{a_{i,j}\} \in \mathcal{R}^{N,N}, \;\; u = \{u_i\}, \;\; f = \{f_i\} \in \mathcal{R}^N, \tag{1}$$

is transformed into an equivalent set of sub-problems each in its "own" subdomain and corresponding "'sub-solutions" for the neighboring blocks are connected via an interface conditions of Poincare-Steklov type. Historically, DDM was described as a tool to solve differential BVPs, see [1], and a lot of literature cited therein, but we consider only a discretized version of these approaches. The idea of a parallel DDM is very simple: all sub-problems in subdomains are solved synchronously on a corresponding processor.

Let Ω be a grid computational domain (or in other words, some set of grid nodes) which can be identified with a set of grid numbers $i = 1, \ldots, N$. We define a decomposition of the domain Ω into its P subdomains without overlapping as

$$\Omega = \bigcup_{q=1}^{P} \Omega_q, \;\; \Omega_{q'} \bigcap \Omega_{q''} = \emptyset, \;\; q' \neq q'', \tag{2}$$

and suppose that each block component of the vectors $u = \{\overline{u}_q\}$, $f = \{\overline{f}_q\}$ corresponds to its own grid subdomain $\Omega_q \in \Omega$ with the number of nodes N_q, $\sum_{q=1}^{P} N_q = N$. For the sake of simplicity, we consider a decomposition without overlapping. For a block form of the original matrix $A = \{A_{q,r}\}$, the simplest iterative DDM can be written as

$$\begin{array}{l} (A_{q,q} + D_q^{(n)})(\overline{u}_q^{(n+1)} - \overline{u}_q^{(n)}) = (f - A\overline{u}^{(n)})_q = \overline{r}_q, \\ q = 1, \ldots, P, \;\; n = 0, 1, \ldots; A_{q,q}, D_q \in \mathcal{R}^{N_q}, \end{array} \tag{3}$$

where $\overline{r}_q$ is a residual vector and $D_q^{(n)}$ are some matrices introduced (may be in a dynamic way, i.e. depending on the iteration number n) to accelerate the iterative process [2]. If the matrices $B_q^{(n)} = A_{q,q} + D_q^{(n)}$ are invertible and the iterative process (3) converges then $\overline{u}_q^{(n)}$ for $n \to \infty$ satisfies the preconditioned equation

$$\begin{array}{l} \overline{A}u = \overline{f}_n, \;\; \overline{A} = B_n^{-1}A, \;\; \overline{f}_n = B_n^{-1}f, \\ B_n = \textit{block-diag}\{A_{q,q} + D_q^{(n)}\}. \end{array} \tag{4}$$

In fact, we solve, instead of (4), the SLAE (1) by some block dynamically preconditioned iterative method in Krylov subspaces. If matrix A is a non-symmetric one, it is natural to use the flexible generalized minimal residual (FGMRES) or semi-conjugate residual (SCR) methods with restarts, see [2] – [3]. Solution for the auxiliary SLAEs in subdomains, i.e. the inversion of the block-diagonal preconditioning matrix B_n is carries out by some direct or iterative method. If the number P of the subdomains is too large then the convergence rate of the iterative process (4) is very slow. For example, for 3D SLAEs we can have $N = 1000^3 = 10^9$ and the condition number of matrix A can be about

10^{13}. In such cases, for $P \approx 100^3 = 10^6$ we will have the value of the number of iterations $n(\varepsilon)$ around several thousands. Here $\varepsilon \approx 10^{-7}$ is the conventional "accuracy", or a tolerance, of the iterative solution. In practice, the usual stopping criterion of the iterative process is

$$||r^{n(\varepsilon)}|| \le \varepsilon ||r^0||, \;\; ||r^0|| = (r^0, r^0)^{1/2}.$$

Let us notice, that the standard double precision arithmetics is supposed to be used during all computations. In order to decrease the value of $n(\varepsilon)$, a special additional acceleration, which will be discussed below, is applied to the iterative process.

An efficiency of a parallel implementation of such a multi-level computational process is supported by various tools: Message Passing Interface (MPI) system to arrange the upper level of iterations over subdomains, multi-threaded computations in a simultaneous solution to the auxiliary subsystems by OpenMP (technologies for multi-core CPUs), vectorization of the machine's operations by AVX instruction set, and a usage of graphic accelerators GPGPU with CUDA Nvidia architecture.

The paper is organized as follows. Section 2 contains a general description of the computational scheme and the data flows in a parallel implementation of DDM. In Section 3 we present some architecture solutions for hybrid programming applied to the problem under consideration intended to be run on a heterogeneous multi-processor system (MPS) with distributed and hierarchical shared memory. The final Section is devoted to some concluding discussion on the technological issues and on the creating of the integrated numerical environment for computational algebra of large sparse SLAEs.

2 DDM computational structure

We present the general scheme of a multi-preconditioned semi-conjugate residual (SCR) method [2]

$$\begin{array}{l} u^{n+1} = u^n + P_n \overline{\alpha}_n, \;\; \overline{\alpha}_n = (\alpha_1^n, \ldots, \alpha_{m_P}^n)^T \in \mathcal{R}^m \\ r^{n+1} = r^n - AP_n \overline{\alpha}_n, \;\; P_n = (\overline{p}_1^n, \ldots, \overline{p}_{m_P}^n) \in \mathcal{R}^{N,m}. \end{array} \tag{5}$$

Here P_n are some rectangular matrices whose columns consist of the direction vectors which are obtained from the recurrent relations

$$\begin{array}{l} \overline{p}_l^0 = B_{0,l}^{-1} r^0, \;\; \overline{p}_l^{n+1} = B_{n+1,l}^{-1} r^{n+1} + \sum\limits_{k=n-m_l}^{n} \sum\limits_{l=1}^{m_P} \beta_{k,l}^n p_l^k, \\ n = 0, 1, \ldots, m_r; \;\; l = 1, \ldots, m_P, \end{array} \tag{6}$$

where $B_{n,l}$ are some predefined preconditioning matrices. The iterative coefficients are computed via the scalar products:

$$\begin{array}{l} \alpha_l^n = (AB_{n,l}^{-1} r^n, r^n)/(Ap_l^n, Ap_l^n) \\ \beta_{k,l}^n = (Ap_l^k, Ap_l^{n,k})/(Ap_l^n, Ap_l^n), \\ p_l^{n,k} = B_{n+1,l}^{-1} r^{n+1} - \sum_{i=0}^{k-1} \beta_{i,l}^n p_l^n, \end{array} \tag{7}$$

If $m_l = n$ and $m_r >> N$ then formulae (5)–(7) provide the orthogonalization of all different vectors Ap_l^k as well as a minimization of the residual norm $||r^{n+1}||$

$$\begin{array}{l}\mathcal{K}_{\sum_{n+1}}(r^0, A) = Span\{B_{0,1}^{-1}r^0, ..., B_{0,M_0}^{-1}r^0, \\ AB_{1,1}^{-1}r^1, ..., AB_{1,M_1}^{-1}r^1, ..., A^nB_{n,1}^{-1}r^n, ..., A^nB_{n,m_P}^{-1}r^n\},\end{array} \tag{8}$$

in the block Krylov subspace [4] of the dimension $\sum_{n+1} = M_0 + ... + M_n$.

In order to save memory, we need to use restarts after each $m_r << N$ iterations, i.e. to define on the iteration number $n = qm_r + 1$ the residual vector from the original equation instead of the recurrent formula

$$r^{qm_r+1} = f - Au^{qm_r+1}$$

and then follow again the recurrent formulas for the next iteration number n. Besides, we can apply the limitation of the orthogonalization and save only last $m_l < n$ direction vectors $p_l^n, p_l^{n+1}, \ldots, p_l^{n-m_l}$ for $l = 1, \ldots, m_p$.

The rate of convergence of the iterative process (5) depends on the number of the subdomains, or more precisely, on the diameter of a graph representing a macro-grid formed by the decomposition. This can be clearly explained by the fact that on a single iteration the solution perturbation in one subdomain is transmitted only to the neighboring, or adjacent, subdomains. To speed up the iterative process, it is natural to use not only the nearest but also the remote subdomain couplings at every step. For this purpose, different approaches are used in decomposition algorithms: deflation, coarse grid correction, aggregation, etc., which to some extent are close to the multigrid principle as well as the low-rank approximations of matrices, see numerous publications cited at a special site [5].

We will consider the following approach based on an interpolation principle. Let Ω_c be a coarse grid with the number of nodes $N_c \ll N$ in the computational domain Ω, moreover, the nodes of the original grid and the coarse grid may not match. Let us denote by $\varphi_1, ..., \varphi_{N_c}$ a set of the basis interpolating polynomials of order M_c on the grid Ω_c which are supposed to have a finite support and without loss of generality form an expansion of the unit, i.e.

$$\sum_{k=1}^{N_c} \varphi_k(x, y) = 1.$$

Then a sought for solution vector of SLAE (1) can be presented in the form of an expansion in terms of the given basis:

$$u = \{u_{i,j} \approx u_{i,j}^c = \sum_{k=1}^{N_c} c_k \varphi_k(x_i, y_j)\} = \Phi\hat{u} + \psi, \tag{9}$$

where $\hat{u} = \{c_k\} \in \mathcal{R}^{N_c}$ is a vector of the coefficients of the expansion in terms of the basis functions, ψ is an approximation error, and $\Phi = [\varphi_1 ... \varphi_{N_c}] \in \mathcal{R}^{N,N_c}$ is a rectangular matrix with every k-th column consisting of the values of the basis

function $\varphi_k(x_i, y_j)$ at N nodes of the original grid Ω (most of the entries of Φ equal zero in virtue of the finiteness of the basis). The columns, or the functions φ_k, can be treated to be the orthonormal ones but not necessarily. If at some k-th node P_k of the coarse grid Ω_c only one basis function is a non-zero one ($\varphi_k(P_{k'}) = \delta_{k,k'}$), then $\hat{u}_k = c_k$ is the exact value of the sought for solution at the node P_k. With a substitution of (9) into the original SLAE, one can obtain the system

$$A\Phi\hat{u} = f - A\psi, \tag{10}$$

and if to multiply it by Φ^T one can obtain

$$\hat{A}\hat{u} \equiv \Phi^T A\Phi\hat{u} = \Phi^T f - \Phi^T A\psi \equiv \hat{f} \in \mathcal{R}^{N_c}. \tag{11}$$

Assuming further that the error ψ in (9) is sufficiently small and omitting it, one can obtain a system for an approximate coarse grid solution $\check{u}$:

$$\hat{A}\check{u} = \Phi^T f \equiv \check{f}. \tag{12}$$

If A is a non-singular matrix and Φ is the full-rank matrix (the rank is much less than N), we assume these facts to hold further, then from (12) we have

$$u \approx \tilde{u} = \Phi\check{u} = \Phi\hat{A}^{-1}\hat{f} = B_c^{-1}f,\ B_c^{-1} = \Phi(\Phi^T A\Phi)^{-1}\Phi^T.$$

The matrix B_c^{-1} introduced above can be regarded as a low rank approximation of the matrix A^{-1} and used as a preconditioner to build an iterative process. In particular, for an arbitrary vector u^{-1} we can choose an initial guess as

$$u^0 = u^{-1} + B_c^{-1}r^{-1},\ \ r^{-1} = f - Au^{-1}. \tag{13}$$

In doing so, the corresponding initial residual $r^0 = f - Au^0$ will be orthogonal in the sense of fulfilling the condition

$$\Phi^T r^0 = \Phi^T(r^{-1} - A\Phi\hat{A}^{-1}\Phi^T r^{-1}) = 0. \tag{14}$$

Such relations are the basis for the deflated conjugate gradient method (DCG) to solve symmetric SLAEs, wherein an initial direction vector is chosen by the formula

$$p^0 = (I - B_c^{-1}A)r^0, \tag{15}$$

which ensures that the following orthogonality condition holds:

$$\Phi^T Ap^0 = 0. \tag{16}$$

Further iterations are implemented using the following relations:

$$\begin{array}{l} u^{n+1} = u^n + \alpha_n p^n,\ \ r^{n+1} = r^n - \alpha_n Ap^n, \\ p^{n+1} = r^{n+1} + \beta_n p^n - B_c^{-1}Ar^{n+1}, \\ \alpha_n = (r^n, r^n)/(p^n, Ap^n),\ \ \beta_n = (r^{n+1}, r^{n+1})/(r^n, r^n). \end{array} \tag{17}$$

In this method, at every step the following relations hold:

$$\Phi^T r^{n+1} = 0,\ \ \Phi^T Ap^{n+1} = 0. \tag{18}$$

3 Technological and performance issues

As was mentioned above, the essence of a parallel DDM is to solve all sub-problems in subdomains synchronously, each on its "own" processor. To start the iterative process, one should have all sub-problems' submatrices distributed among the processors or MPI processes (we consider a DDM program flow on some cluster MPS having some MPI system on it, so we suppose that each subdomain is treated in its "own" MPI process). Here we have just named two important moments: the matrix of the system should be distributed and each computational process has its own subdomain to carry out calculations over it. Naturally, to make a calculation load uniform on all the processes, matrix division into submatrices should be as balanced as possible. One can do it manually, but a conventional approach is to represent the matrix as a graph and use a graph theory for the graph partitioning. If one use free software, the best choice is to use METIS [6], a set of serial programs to partition graphs (its parallel analogue, ParMETIS [7] which is an MPI-based parallel library that extends the functionality provided by METIS requires prior approval). A balanced data distribution provides more uniform calculation load for the processes and a downtime reduction.

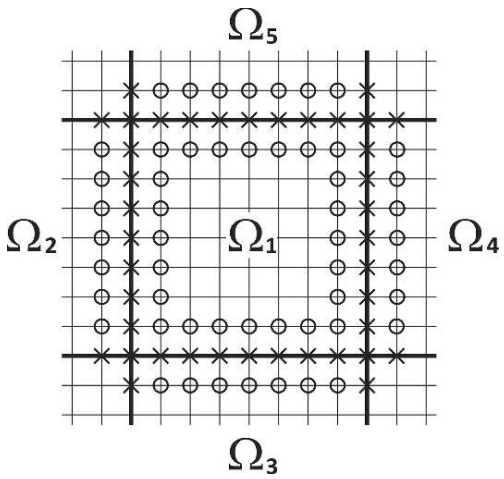

Fig. 1. Decomposition of 2D domain with a subdomain-divider

The next point needed to be mentioned is about a type of the domain division, which can be carried out with or without intersections between adjacent subdomains. The simplest way to start here is to make the division of the computational domain into subdomains firstly without any intersections and then add, if necessary, these intersections ("overlapping") in a layer-by-layer manner. Here a term "layer" stands for the layer consists of the grid nodes, adjacent (having links, or grid edges) to the previous subset of the grid nodes. E.g., let one have

a domain division, then for a grid subdomain, the nodes that have links with the nodes from other subdomains form a "boundary" layer of this subdomain, neighbors (in graph sense) of these boundary nodes that lie in other subdomains form the first outer layer, neighbors of the neighbors form the second outer layer, etc. In such a manner, one can perform layered intersections of the subdomains. It is worthy to notice that in practice, it is enough to use 3–4 grid layers to boost convergence of the outer iterative DDM process, otherwise overheads to save the additional grid and submatrix data grow without giving additional acceleration of the convergence.

A separate question in organizing parallel DDM calculations is in the data management for SLAEs presented in the Compressed Sparse Row format (CSR, see, e.g., [8]). If one have a whole matrix of system (1) on a separate cluster node (on a separate MPI process), then it is necessary to perform the matrix division and submatrices' distribution. In doing so, one can face a memory limit – if the matrix is too big, a memory available within one cluster node could be insufficient even to save the matrix coefficients. The ideal way to overcome this limit is to make submatrices on a stage of grid construction and approximation, every on its own node. A special case here is when one have the problem produced from outside, with its right-hand side and the matrix given in the block form already ready to be distributed between the nodes (probably, each block in a separate file, or all the blocks but in indexed file).

For parallel DDM programming and coding, it is important to understand how a division of the domain is carried out and what namely is a "divider". A domain decomposition can be done either with (Fig. 1) or without (Fig. 2) the divider. A decomposition without a divider is the most natural one - it is just a division into subdomains without intersections as mentioned above. In (Fig. 2), the dividing assumed runs via black bold lines, the neighboring nodes from different subdomains are marked by empty small circles. Data exchange between subdomains is carried out for the adjacent (in the approximation stencil sense) nodes from different subdomains. If exists, a divider is such a set of the grid nodes that every subdomain has the links with the divider only but not with other subdomains. So, the divider binds all the subdomains. In (Fig. 1), nodes of the divider are crosses on the black bold lines, neighboring nodes from adjacent to divider subdomains are empty small circles. One can think of the divider as of some special separate subdomain with a property that only this subdomain has the links with all the rest subdomains of the division; all the subdomains except the divider have the links only with the divider. Therefore, the information transfer between the subdomains is done only via the divider.

To reduce the run time of a parallel DDM program one should reduce both the volume of the transferred information and the number of the data exchanges. The latter depends on the rate of convergence of the iterative process chosen. Besides, an important moment is a necessity to combine computation flow with exchanges as they are carried out by different devices. If one use some MPI-like tool, then the best choice for this is to use non-blocking functions MPI_Isend

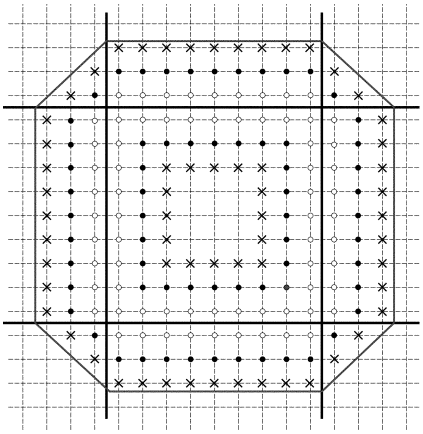

Fig. 2. Decomposition of a grid domain without divider

and MPI_Irecv and make some calculations without additional waiting until communication is complete.

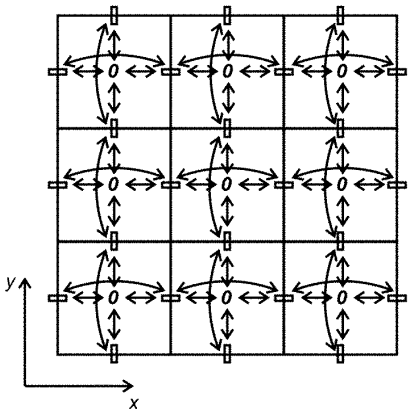

Fig. 3. The grid stencils for the mixed statement at the staggered grid

4 Component architecture of DDM

As one can see from the above, the algebraic DDM approaches for parallel solving of a wide class of big sparse SLAEs on heterogeneous multi-processor systems with the distributed and hierarchical shared memory present a complicated computational problem. A mathematical efficiency of the algorithms used and

high performance of the corresponding code are both very important for vast modelling applications. It is worthy to mention that a program implementation should be adapted to distinctive features of new methods and to the evolution of computer platforms. In general, we can name the following steps in algebraic DDM technologies:

- automatic construction of algebraic balanced decomposition of the given grid domain in two stages: without intersection and with a parametrized overlapping;
- creating the submatrices for corresponding grid subdomain with predefined modifications taking into account the interface conditions of Poincare-Steklov type on the internal boundaries of the adjacent subdomains, as well as a deflation preconditioner;
- organizing the distributed submatrices for subdomains that are saved in its own MPI process;
- forming of buffers for the data exchange between subdomains at each external iteration as well as for a coarse grid correction;
- implementation of external iterations in the Krylov subspaces over subdomains;
- solution to auxiliary SLAEs in subdomains.

At two latter steps, different algebraic solvers from the Krylov library can be used [9]. The concept of the Krylov library consists in support of the integrated numerical environment to solve SLAEs under constraints of strong technical requirements: a flexible possibility to extend library's functionality, i.e. possibility to add new models and algorithms, adapting to evolution of computer platforms, a possibility of some external software product usage, coordinated participation of other groups of developers. These requirements will provide the long life cycle of the project and its wide exploitation by end users.

5 Conclusion

In this paper, architecture solutions to create a parallel integrated DDM environment have been discussed. They are based on modern approaches taking into account geometric and algebraic DDM parallel nature. Theoretical evaluation of the resulting code performance seems to be unreachable because of a high complexity of the computational model and it depends on its every specific implementation in a high-level programming language. Further verification of the approaches could be in a development of the Krylov library to make it, e.g., more adaptable to a given specific problem. Experimental data on some model problems solved with taking into account several of the aspects named can be found in the preceding authors' works, e.g., [4],[10]. Surely, a large related work consists in a methodical comparison of other modern high-performance approaches like OpenMP, AVX, GpGPU+CUDA usage with the current Krylov implementation from a performance point of view. This could be a step to an open question of building a low-cost specialized high-performance DDM software.

References

1. Dolean, V., Jolivert, P., Nataf, F.: An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation, `http://archives-ouvertes.fr/cel-01100932`
2. Il'in, V.P.: Problems of parallel solution of large systems of linear algebraic equations. J. of Matem. Sci. 216(6), 795–804 (2016)
3. Butyugin, D.S., Gurieva, Y.L., Ilin, V.P., Perevozkin, D.V.: Some Geometric and Algebraic Aspects of Domain Decomposition Methods. In: Dickopf, Th., Gander, M.J., Halpern, L., Krause, R., Pavarino, L.F. (eds.) Domain Decomposition Methods in Science and Engineering XXII. LNCSE, vol. 104, pp. 117–125, Springer, Heidelberg (2016)
4. Gurieva, Y.L., Il'in, V.P.: On parallel computational technologies of augmented domain decomposition methods. In: Malyshkin, V. (Ed.) Parallel Computing Technoligies: 13th International Conference, PaCT 2015, Petrozavodsk, Russia, August 31-September 4, 2015, Proceedings. LNCS 9251, pp.35–46, Springer, Heidelberg (2015)
5. Domain Decomposition, `http://www.ddm.org`
6. Karypis, G., Kumar, V.: A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM J. Sci. Comp. 20 (1), 359–392 (1999).
7. ParMETIS, `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`
8. Intel Math Kernel Library (Intel MKL), `http://software.intel.com/en-us/intel-mkl`
9. Butygin , D.S., Gurieva, Y.L., Il'in, V.P., Perevozkin, D.V., Petukhov, A.V., Skopin, I.N.: Functionality and technologies of algebraic solvers in the KRYLOV library, Vestnik YuUrGU, 2(3), 92–105 (2013) (in Russian).
10. Gurieva, Y.L., Il'in, V.P., Perevozkin, D.V.: Algebraic-geometric and information structures of domain decomposition methods. Vychislitelnye Metody i Programmirovanie. Scientific on-line open access journal. 17, 132–146 (2016) (in Russian).