DOI 10.1007/s10559-017-9996-5
Cybernetics and Systems Analysis, Vol. 53, No. 6, November, 2017

SOFTWARE–HARDWARE SYSTEMS

HYBRID ALGORITHMS FOR SOLVING THE ALGEBRAIC EIGENVALUE PROBLEM WITH SPARSE MATRICES

A. N. Khimich,[1†] A. V. Popov,[1‡] and O. V. Chistyakov[1††] UDC 519.6

Abstract. *Hybrid algorithms for solving the partial generalized eigenvalue problem for symmetric positive definite sparse matrices of different structures by hybrid computers with graphic processors are proposed, coefficients for the efficiency of the algorithms are obtained, and approbation of the developed algorithms for test and practical problems is carried out.*

Keywords: *algebraic eigenvalue problem, computer of hybrid architecture, hybrid algorithm, subspace iteration method, conjugate gradient methods, efficiency of parallel algorithms.*

INTRODUCTION

A great number of practical problems, such as structural stability analysis, dynamic analysis of stress–strain state of objects of various nature, etc., reduce to the partial algebraic eigenvalue problem (AEP) with sparse symmetric positive definite matrices [1].

Development of parallel algorithms of AEP solution is based on well-known, well-behaved sequential algorithms. The papers [2, 3] present explicit and implicit one-step iterative methods to solve the partial eigenvalue problem, among which are the method of steepest descent, inverse iteration method, power method, and alternate triangular method. The paper [4] considers the generalized conjugate gradient method on the basis of the symmetric upper relaxation method to solve systems of linear algebraic equations. The study [5] outlines theoretical aspects of the subspace iteration method, the Lanczos algorithm for finding several minimum eigenvalues.

The requirements for high-performance computing are much more ahead of the capabilities of traditional parallel computers, even despite multi-core processors. This problem is being solved by using hybrid computers that combine MIMD and SIMD architecture, which use computers with multi-core processors (CPU) and graphics accelerators (GPU). The fullest use of the potential of such computers is only possible if algorithms and software are available that take into account not only properties of the problem but the features of hybrid architecture as well.

The most well-known algorithms and programs to solve problems of linear algebra for various types of parallel computers were developed under the guidance of Prof. J. Dongarra [6, 7]. To solve AEP with large sparse matrices, parallel software library SLEPc (Scalable Library for Eigenvalue Problem Computations) is available [8]. LIS (Library of Iterative Solvers) [9] allows solving linear algebra problems, in particular, systems of linear algebraic equations and AEP, with matrices of various formats by parallel computers with the use of floating-point arithmetic.

The study [10] considers some special features of developing parallel algorithms of the subspace iteration method for computers of different architectures. In [11], parallel algorithms are developed for alternate triangular method.

[1]V. M. Glushkov Institute of Cybernetics, National Academy of Sciences of Ukraine, Kyiv, Ukraine, [†]*khimich505@gmail.com*; [‡]*alex50popov@gmail.com*; [††]*alexej.chystyakov@gmail.com*. Translated from Kibernetika i Sistemnyi Analiz, No. 6, November–December, 2017, pp. 132–146. Original article submitted July 13, 2017.

SOME ISSUES OF CREATING HYBRID ALGORITHMS

In the paper, we will propose and investigate hybrid algorithms for the solution of the generalized eigenvalue problem

$$Ax = \lambda Bx, \tag{1}$$

where A and B are symmetric positive definite sparse matrices of order n for computers with multi-core processors and graphics accelerators. We will consider the approach to creating hybrid algorithms that is based on structural regularization of a sparse matrix of general form due to reordering of variables, reducing it to the profile, bordered block-diagonal, or band form with further use of hybrid algorithms on the basis of subspace iteration method and implicit conjugate gradient method.

The main problems of creating efficient hybrid algorithms are related to coordinating the distribution of calculations on CPU and GPU and to optimizing the overhead expenses for communication interactions between them. By an efficient problem solution algorithm we understand an algorithm that ensures solution reliability under optimal use of computer resources.

In what follows, we will use the architecture of hybrid computer (unless otherwise specified): each of the p computing processes on CPU interact with one (its) GPU. Such architecture is often denoted by $p\,\text{CPU} + p\,\text{GPU}$.

The efficiency of solution techniques of the majority of problems in linear algebra, including AEP, depends heavily on the efficient performance of matrix–matrix and matrix–vector operations and on the solution of systems of linear algebraic equations (SLAE). To carry out these operations, it is expedient to use GPU whose maximum performance is attained when similar mathematical operations are executed over big amount of data under complete load of each GPU.

Let us formulate the following requirements for the development of efficient hybrid algorithms of AEP solution:

- decomposition into subproblems to be executed on CPU and GPU;
- efficient method of factorization of sparse matrices;
- uniform loading of CPU processes and synchronization of exchanges between them;
- solution of subproblems that require the greatest computing time on GPU;
- minimization of exchanges between CPU processes as well as between CPU and GPU.

In the distribution of matrices between processes on CPU, it is important to provide an efficient computer configuration (topology) of processes and their uniform loading (load balance) to avoid Haydn's effect [12], which reduces the efficiency of parallel algorithm. As studies have shown, block-cyclic decomposition of matrix elements among CPU processes as well as topology of "hypercube" connections between computers being used are the most efficient for the hybrid algorithms under study.

HYBRID ALGORITHM OF THE SUBSPACE ITERATION METHOD

Subspace Iteration Method. This method is used to find r minimum eigenvalues and corresponding to them eigenvectors of problem (1) with symmetric band matrix. This method is a generalization of the inverse iteration method. For problem (1), it constructs the sequence of subspaces E_t $(t = 1, 2, \ldots)$ that converges to subspace E_∞, which contains the required eigenvectors [5]. At the tth iteration, the orthogonal basis of subspace E_t is calculated, and if the required accuracy of the approximate solution is attained, the required eigenpairs are found.

Thus, realization of the subspace iteration method reduces to solving the following subproblems for $t = 1, 2, \ldots$:

- solving SLAE:

$$AX_t = Y_{t-1}; \tag{2}$$

- calculating the rectangular matrix:

$$W_t = BX_t; \tag{3}$$

- calculating the projections of matrices A and B onto the subspace E_t:

$$A_t = X_t^T Y_{t-1} \equiv X_t^T AX_t,\ B_t = X_t^T W_t \equiv X_t^T BX_t; \tag{4}$$

- solving the full eigenvalue problem for projections:

$$A_t Z_t = B_t Z_t \Lambda_t; \tag{5}$$

- calculating the approximation

$$Y_t = W_t Z_t. \tag{6}$$

If after c iterations the conditions of termination of the iterative process are satisfied, for example, $\left| \frac{\lambda_i^{(c)} - \lambda_i^{(c-1)}}{\lambda_i^{(t)}} \right| \le \varepsilon$, then additional iteration is made and $\lambda_i^* = \lambda_i^{(c+1)}$ $(i = 1, 2, \ldots, r)$ and the first r columns of matrix $X^* = X_{c+1} Z_{c+1}$ are taken as approximate solutions of problem (1) (we mean that the eigenvalues are ordered in increasing order: $0 < \lambda_1 \le \lambda_2 \le \ldots \le \lambda_r \le ...$).

As is generally known [5], the iterative process converges linearly, and the convergence rate λ_i is defined by the relation λ_q / λ_1, where q is the dimension of the iterated subspace E_q. In successive realizations of the algorithm, it is recommended to choose $q = \min(2r, r+8)$.

Since at each iteration SLAE (2) is solved with the same matrix A, the triangular expansion, for example LL^T, of this matrix is performed once before the beginning of the iterative process.

Data Distribution between Processor Devices. To solve problem (1), matrix A is divided into square blocks $A_{I,J}$ of order s. Elements of the principal diagonal and of the lower or upper triangle (depending on the algorithms being used) of nonzero blocks of this sparse symmetric matrix are distributed among CPU according to one-dimensional block-cyclic scheme. According to this scheme, block $A_{I,J}$ is stored in CPU with logic number $(I + l) \bmod p$ (the result of operation $k \bmod j$ is the remainder of dividing k by j, $-1 \le l \le p-2$ is shift, usually $l = -1$). Blocks of the lower triangular matrix L or of the upper triangular matrix L^T are distributed similarly. Thus, we have rows of square matrix blocks of order s.

The same block-cyclic distribution scheme is used for elements of matrix B and rectangular matrices of iterated vectors X_t, Y_t, and W_t. It will suffice to distribute and store only nonzero elements of matrix B in the following sequence: under-diagonal, diagonal, and above-diagonal ones. This considerably simplifies the algorithm of multiplication of such matrix by rectangular one and insignificantly increases the total amount of data.

GPU memory stores copies of the blocks (according to distribution among CPU) of matrices A, B, X_t, Y_t, and W_t, which are used to execute operations on GPU according to the variant of hybrid algorithm being used (the variant is chosen according to the parameters of the problem being solved and technical features of GPU).

Distribution of Computations among Processor Devices. According to the aforesaid, solution of partial AEP (1) by the subspace iteration method can be divided into four subproblems (of different computing complexity).

1. Generating matrix Y_0 of initial iterated vectors, distributed among the processes, such that matrix B_t from (4) is positive definite. This operation is executed by CPU, without data exchange between processes, using, for example, the parallel algorithm proposed in [10].

2 LL^T-factorization of sparse symmetric positive definite matrix A (on CPU and GPU), using, for example, hybrid tiling algorithm [13] for the case of band matrix.

3. Executing the iterative process (2)–(6), where for each $t = 1, 2, \ldots$ the computations are distributed among processor devices as follows:

- to solve SLAE (2), LL^T-decomposition of matrix A obtained earlier is used, i.e., two systems with triangular matrices are solved: $LV = Y_{t-1}$ and $L^T X_t = V$;
- calculations (3) of rectangular matrix $W_t = BX_t$ depending on the structure (sparse or diagonal) of matrix B, dimension of the subspace being iterated, number of the GPU and their memory are executed either by CPU, or by CPU and GPU, or by GPU;
- calculations (4) of the products of rectangular matrices to form projections of matrices A and B onto the subspace are executed on GPU, and matrices of projections (if more than one GPU is used) are assembled by each process on CPU; such approach substantially reduces amount of data in CPU–GPU exchange; products of rectangular matrices can be calculated on GPU asynchronously with other computing operations or exchanges;

- to solve complete generalized AEP (5) with regard for rather small order of matrices of projections, the Jacobi method is used; the problem is solved by each process on CPU; in that case, there is no need in data exchange between computers;
- conditions of the termination of iterative process are checked by each process on CPU;
- calculation (6) of the new matrix of iterated vectors Y_t (or matrix of approximated eigenvectors X^*) is carried out on GPU according to data distribution (a submatrix of matrix Y_t or X^* is calculated), and there is no need in data exchange between processor devices.

4. Reliability analysis of the results: calculating the error estimates of the obtained solution (see, for example, [10]). To estimate the relative error of the calculated approximations to the eigenvalues, the operations similar to those carried out in the iterative process are executed. To implement these operations, the same hybrid algorithms as in the iterative process are used.

Efficiency of the Hybrid Algorithm. Efficiency analysis of the proposed hybrid algorithm is based on the methodologies from [10, 15]. As was mentioned above, we consider a computer of hybrid architecture with p CPU and p GPU.

The efficiency of parallel algorithms is characterized by acceleration factors $S_p = T_1 / T_p$ and efficiency coefficients $E_p = S_p / p$. Here (for hybrid architecture), T_1 is the time of problem solution for architecture 1 CPU + 1 GPU, T_p is the time of solution of the same problem for architecture p CPU + p GPU. Solution time can be calculated by the formulas

$$T_1 = O_1 t_C + O_{1G} t_G / n_o + O_{oG} t_{oG} + O_{cG} t_{cG},$$
$$T_p = O_p t_C + O_{pG} t_G / n_o + O_o t_o + O_{oG} t_{oG} + O_c t_c + O_{cG} t_{cG}, \quad (7)$$

where t_C and t_G are average time of executing one floating-point arithmetic operation on CPU and GPU, respectively; n_o is the number of such operations that can be executed on GPU simultaneously; t_o and t_{oG} are time necessary to exchange one machine word between MPI-processes on CPU or between CPU and GPU, respectively; O_o and O_{oG} are the volumes (numbers of machine words) of exchanges executed by one CPU and GPU, respectively; t_c and t_{cG} are time necessary to synchronize two MPI processes or an MPI process and its GPU, respectively; O_c and O_{cG} are the numbers of such synchronizations executed by one MPI process. When using hybrid architectures in formulas (7), it is necessary to take into account the synchronism or asynchronism of CPU and GPU operation. In case of asynchronous work, it is necessary to replace summation with finding the maximum value.

When analyzing and solving partial AEP (1) with symmetric matrices, a part of stages (generating the matrix of initial iterated vectors, LL^T-decomposition of matrix A, calculation of the estimates of approximate solution) has a fixed number of arithmetic operations, and for the iterative process (2)–(6) the number of arithmetic operations is proportional to the number of performed iterations. The number of iterations necessary to find r minimum eigenvalues is as a rule defined by $O(r)$, and the greater the dimension q of the iterated subspace, the less this quantity. Thus, $T_p = T_p^{(F)} + c_I T_p^{(1I)}$, and it is possible to represent the acceleration factor of the proposed hybrid algorithm as

$$S_p = \frac{T_1}{T_p} = \frac{T_p^{(F)}}{T_p} S_p^{(F)} + \frac{c_I T_p^{(1I)}}{T_p} S_p^{(It)}, \quad (8)$$

where c_I is the number of iterations, $T_p^{(F)}$ and $T_p^{(1I)}$ are, respectively, solution time for subproblems with a fixed number of arithmetic operations and execution time of one iteration on p CPU + p GPU; $S_p^{(F)}$ and $S_p^{(It)}$ are acceleration factors of the algorithms used to solve these subproblems.

For architecture p CPU + p GPU, let us introduce the notation: $T_p^{(LLT)}$ is time for LL^T-decomposition of matrix A; $T_p^{(SS)}(k)$ is time for solution of SLAE (2) with k right-hand sides (using the LL^T-decomposition of the matrix of system calculated earlier); $T_p^{(Fo)}$ and $T_p^{(Io)}(k)$ are, respectively, the run times of the other operations in solution of subproblems with a fixed number of arithmetic operations and execution of one iteration. Then $T_p^{(F)}$ and $T_p^{(1I)}$ can be represented as

$$T_p^{(F)} = T_p^{(LLT)} + T_p^{(SS)}(q) + T_p^{(Fo)}, \quad T_p^{(1I)} = T_p^{(SS)}(q) + T_p^{(Io)}(q). \quad (9)$$

Hereinafter, it is possible to obtain formulas for all $T_1^{(*)}$ by substituting $p=1$ into the formulas for $T_p^{(*)}$, and for all such values of time formulas (7) are true. Then based on (8), we can calculate the acceleration factor of the hybrid algorithm of the subspace iteration method for architecture $p\,\mathrm{CPU}+p\,\mathrm{GPU}$ by the formula

$$S_p=\frac{T_p^{(F)}}{T_p}\left(\alpha_p^{(LLT)}S_p^{(LLT)}+\alpha_p^{(SS)}(q)S_p^{(SS)}(q)+\alpha_p^{(Fo)}S_p^{(Fo)}\right)$$
$$+\frac{c_I T_p^{(1I)}}{T_p}\left(\beta_p^{(SS)}(q)S_p^{(SS)}(q)+\beta_p^{(Io)}(q)S_p^{(Io)}(q)\right), \tag{10}$$

where $\alpha_p^{(*)}=T_p^{(*)}/T_p^{(F)}$, $\beta_p^{(*)}=T_p^{(*)}/T_p^{(1I)}$, $S_p^{(*)}$ is the acceleration factor of the hybrid algorithm of the solution of respective subproblem (execution of operations), and superscripts and parameters coincide with the respective superscripts and parameters in (9) for times $T_p^{(*)}$.

Consider the case where matrices of problem (1) are band ones. Let us introduce the notation: m_A and m_B are band half-widths of matrices A and B, respectively (we assume that $n=O(10^k m_A)$, $k>1$, $m_A \ge m_B$, and $m_A >> q$); η_B is average number of nonzero elements in one row of matrix B (as a rule, $\eta_B << m_B$). For $p>1$, exchanges between MPI processes (CPU) are multi-scatter and multi-gather operations, which are supposed to be implemented by the tree algorithm. We will also introduce times for exchange with an array of d double words between CPU pair: $t_C^{(E)}(d)=t_s+dt_o$, and for $\mathrm{CPU}+\mathrm{GPU}$ architecture we get $t_G^{(E)}(d)=t_{sG}+dt_{oG}$.

For the case of hybrid algorithms used to solve SLAE with symmetric band matrix (see [13, 14]), we can formulate the lemma (we neglect the values of the smallest orders).

LEMMA 1. Under the condition $n>>q>r$ and $m_A>sp$ for the times it takes hybrid algorithm to LL^T-decompose the band symmetric matrix and to solve two SLAE with lower and upper triangular band matrices on the architecture $p\,\mathrm{CPU}+p\,\mathrm{GPU}$, the following estimates are true:

$$T_p^{(LLT)}\approx\frac{n}{p}\left(\frac{s^2}{3}pt_C+\max\left\{m_A^2\frac{t_G}{n_o},t_C^{(E)}(sm_A)\frac{p}{s}\log_2 p\right\}+2psm_A\frac{t_G}{n_o}+t_G^{(E)}(sm_A)\frac{p}{s}\right), \tag{11}$$

$$T_p^{(SS)}(k)=\frac{n}{p}\left(kt_C p\log_2 p+(4m_A+2ps)k\frac{t_G}{n_o}+(t_C^{(E)}(sk)\log_2 p+2t_G^{(E)}(sk))\frac{2p}{s}\right). \tag{12}$$

Using (11) and (12), taking into account that it is possible to execute some operations of calculation and exchange on CPU and GPU simultaneously or by different flows on GPU, and neglecting the terms of smaller order, we obtain the following estimates of acceleration factors of hybrid algorithms of LL^T-decomposition of band symmetric matrix and solution of SLAE (2) on the architecture $p\,\mathrm{CPU}+p\,\mathrm{GPU}$:

$$S_p^{(LLT)}\approx p\left(1-\frac{2s(p-1)}{m_A+2ps}\right),\ t_G^{(m)}(s)\ge t_C^{(E)}(sm_A),$$
$$S_p^{(LLT)}\approx p\left(\frac{2ps}{m_A+2s}+\frac{p\log_2 p}{m_A s(m_A+2s)}\frac{n_o t_C^{(E)}(sm_A)}{t_G}\right)^{-1}, \tag{13}$$
$$t_G^{(m)}(s)<t_C^{(E)}(sm_A),\ t_G^{(m)}(s)=\frac{m_A^2 s}{p\log_2 p}\frac{t_G}{n_o};$$
$$S_p^{(SS)}(k)\approx p\left(1+\frac{(p-1)s}{2m_A+s}+\frac{p\log_2 p}{ks(2m_A+s)}\frac{n_o t_C^{(E)}(ks)}{t_G}\right)^{-1}.$$

When deriving the estimates $T_p^{(Fo)}$ and $T_p^{(Io)}(k)$, it is necessary to take into account that in the general case the majority of arithmetic operations are performed in calculating the product of sparse matrix B by $(n\times k)$-matrix and in calculating matrices or vectors whose elements are scalar products of n-dimensional vectors. The number of other arithmetic operations executed by one of the p GPU is much less and can be neglected. Then the following statement is true.

LEMMA 2. Provided that $n>>q>r$, $m_A \geq m_B$, and $1\leq \eta_B << m_B$, the following estimates are true for times $T_p^{(Fo)}$ and $T_p^{(Io)}(k)$:

$$T_p^{(Fo)} = \frac{n}{p}\left(\frac{n+2(n+2)p\log_2 p}{n} qt_C + (2\eta_B+6)q\frac{t_G}{n_o}\right) + (2t_C^{(E)}(sq)\log_2 p + 2t_G^{(E)}(sq))\frac{n}{s}, \quad (14)$$

$$T_p^{(Io)}(k) = \frac{n}{p}\left(\frac{(n+2k)\log_2 p + O(k^2)}{n} pkt_C + (2\eta_B+4k)k\frac{t_G}{n_o}\right) + (t_C^{(E)}(sk)\log_2 p + 2t_G^{(E)}(sk))\frac{n}{s}. \quad (15)$$

From (14) we can see that time $T_p^{(Fo)}$ is much less than the values of $T_p^{(LLT)}$ and $T_p^{(SS)}(q)$ from (9); therefore, we can neglect it by assuming in (10) $\alpha_p^{(Fo)} \cong 0$.

The influence of $T_p^{(Io)}(q)$ depends on the relation among m_A, η_B, and q: if q is much greater than η_B, then in (15) we can neglect the term that contains η_B; if $m_A >> \eta_B$ and $m_A >> q$, then in (9) we can neglect $T_p^{(Io)}(q)$. From (15) the estimate also follows

$$S_p^{(Io)}(k) \approx p\left(1+\frac{p\log_2 p}{2k(\eta_B+2k)}\frac{n_o}{t_G}(kt_C + t_C^{(E)}(sk))\right)^{-1}. \quad (16)$$

Now we can estimate the acceleration and efficiency of the proposed hybrid algorithm as a whole.

THEOREM 1. For the hybrid algorithm of the subspace iteration method under the conditions $n>>q$, $m_A \geq m_B$, $m_A >> q$, $m_A > sp$, and $\eta_B << m_B$, the following estimate of acceleration factors takes place:

$$S_p = \frac{\gamma_p}{\gamma_p + c_I}(\alpha_p^{(LLT)} S_p^{(LLT)} + \alpha_p^{(SS)}(q) S_p^{(SS)}(q)) + \frac{c_I}{\gamma_p + c_I}(\beta_p^{(SS)}(q) S_p^{(SS)}(q) + \beta_p^{(Io)}(q) S_p^{(Io)}(q)), \quad (17)$$

where $\gamma_p = (T_p^{(LLT)} + T_p^{(SS)}(q))/T_p^{(1I)}$; $T_p^{(1I)}$ is represented in (9); to calculate the coefficients $\alpha_p^{(*)} = T_p^{(*)}/T_p^{(F)}$ and $\beta_p^{(*)} = T_p^{(*)}/T_p^{(1I)}$, formulas (11), (12), (14), and (15) are used, and the terms that are small with respect to other terms are discarded in the obtained expressions; the acceleration factors $S_p^{(*)}$ are represented by formulas (13) and (16).

The efficiency of the algorithm is estimated by $E_p = S_p/p$.

As analysis of the obtained estimates (11)–(17) shows, the efficiency of the proposed algorithm notably depends on the efficiency of the algorithm of LL^T-decomposition, for example, of hybrid tiling algorithm [16], as well as on the efficiency of the algorithms of solution of SLAE with the lower and upper sparse triangular matrices. If the time spent for data transfer between computing devices is less than computing time, these algorithms (because of the use of asynchrony) will have efficiency coefficients close to one.

The number c_I of iterations performed to attain the necessary accuracy renders a significant influence on the efficiency of the hybrid algorithm of the solution of partial generalized AEP by the subspace iteration method. Usually, for constant r, the greater the dimension of the iterated subspace q, the less the value of c_I. As follows from estimates

TABLE 1. Test Band Matrices from the University of Florida Collection

Problem Name	Problem Domain	Matrix Order (Band Half-Width)
G2_circuit	Structural problem	150 102 (65 956)
Bone010	Model reduction problem	986 703 (26 572)
Emila_923	Structural problem	923 136 (34 175)

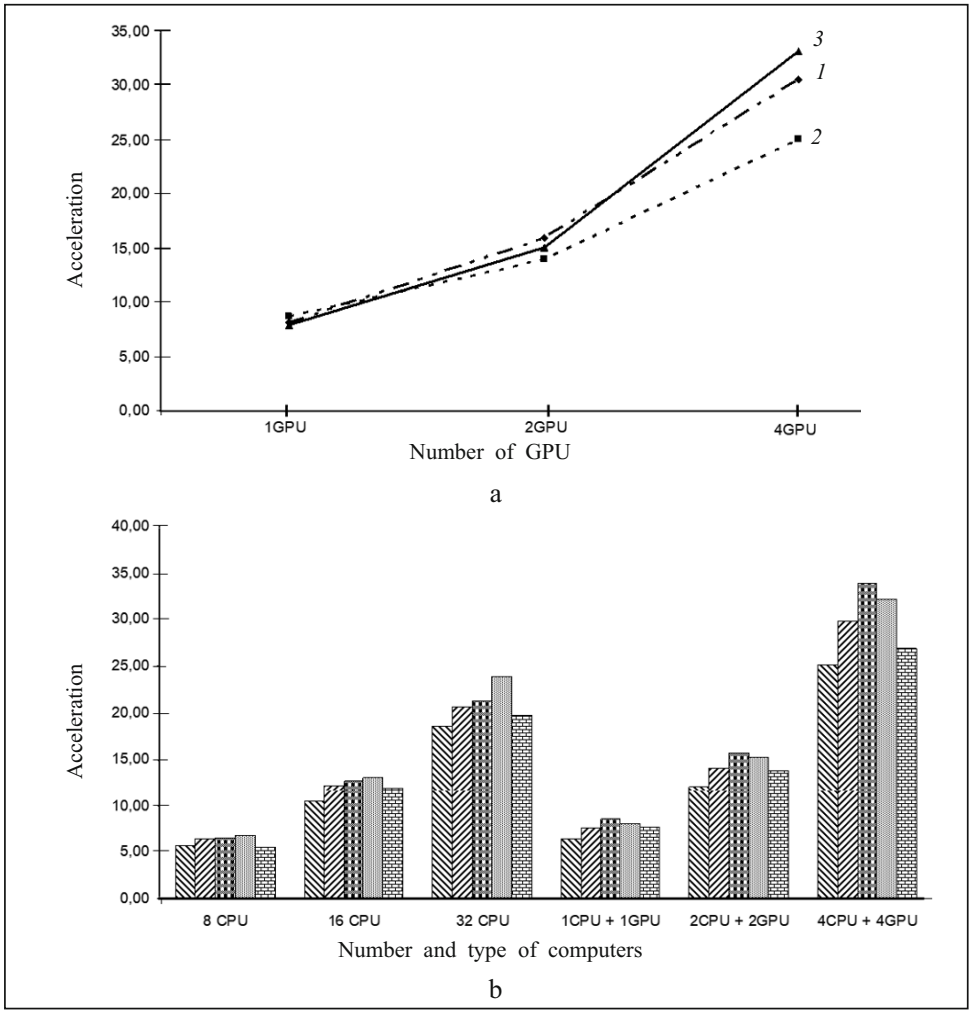

Fig. 1 (a) graphs of acceleration of the hybrid iteration algorithm on the subspace for different numbers of GPU for problems G2_circuit (*1*), Bone010 (*2*), Emila_923 (*3*); (b) diagram specifying the acceleration of the subspace iteration algorithm for different numbers and types of computers for blocks — size 64, — size 128, — size 192, — size 256, — size 512.

(12) and (15), the amount of computation is proportional to q. However, in some cases, for rather small values of q GPU load is insufficient (i.e., the value of n_o actually decreases) as the number of exchanges increases, which reduces actual efficiency of the algorithm. In such cases, increase in the dimension of the iterated subspace (up to attaining the maximum value of n_o) reduces problem solution time by reducing the number of iterations.

Experimental Efficiency Analysis of the Hybrid Algorithm. The experimental analysis was performed by band matrices of different order; matrices from the University of Florida sparse matrix collection [17] were also used (Table 1).

Figure 1a shows the graphs of dependence of acceleration on the number of GPU. The experiments testify good scalability, i.e., problem solution time proportionally decreases as the number of computing devices grows. Also, higher efficiency is provided for matrices with large band half-width (the efficiency of the algorithm improves as the band half-width of the original matrices increases). Figure 1b shows the diagram and specifies the acceleration of the iteration algorithm for different number and types of tiles. Using tiles of optimal size in LL^T-decomposition and in the iteration process allows obtaining good balance of loading of the computing devices being used, substantially reduces the solution time, and increases the computing efficiency. Depending on the type of computing device and its performances, the recommended order of tiles is 512 for CPU and 192 to 256 for GPU. The developed hybrid algorithm allows efficient use of modern computers of various types and adaptation to the structure of matrix of the problem.

HYBRID ALGORITHM OF THE CONJUGATE GRADIENT METHOD

In many practical calculations, generalized AEP (1) with diagonal positive definite matrix B occurs. This problem can be easily reduced to a standard AEP, the eigenvalue problem for symmetric positive definite matrix $C = B^{-1/2}AB^{-1/2}$: $Cy = \lambda y$, where $y = B^{1/2}x$. The sparse structure of the original matrix remains the same. Below, we will consider the iterative method to solve such problem.

The generalized conjugate gradient method to solve the partial eigenvalue problem

$$Ax = \lambda x \tag{18}$$

is represented in [4] by the following formulas:

$$x^{k+1} = \begin{cases} x^k + a_k p^k, \ k \neq N-1, \ 2N-1, \ldots, \\ \dfrac{x^k + a_k p^k}{\| x^k - a_k p^k \|_2}, \ k = N-1, \ 2N-1, \ldots, \end{cases} \tag{19}$$

$$p^k = Z(x^k) f(x^k) - \beta_k p^{k-1}; \ f(x) = [A - \mu(x) I] x / \| x \|_2, \tag{20}$$

$$\beta_k = \begin{cases} 0, \quad k = 0, \ 2N, \ \ldots, \\ \dfrac{(Z(x^k) f(x^k), [A - \mu(x^k) I] p^{k-1})}{(p^{k-1}, \ [A - \mu(x^k) I] p^{k-1})}, \ k = 1, 2, 3, \ldots, \end{cases} \quad \mu(x) = \frac{(Ax, x)}{(x, x)}.$$

Here, parameter α_k can be chosen as a local minimum point of functional $\mu(x^k - \alpha p^k)$, N is restoration moment, matrix $Z(x^k)$ is chosen from the conditions of acceleration of convergence of the iterative process and special features of parallel computer architecture. For example, $Z(x)$ can be chosen on the basis of the upper symmetric relaxation method for the solution of systems of linear algebraic equations. In this case, if we represent the original matrix as

$$A = I - L - L^T$$

in terms of identity, lower, and upper triangular matrices, then

$$Z(x) = \omega(2 - \omega)(I - \omega \hat{L}^T(x))^{-1} (I - \omega \hat{L}(x))^{-1}, \tag{21}$$

where $\hat{L}(x) = \dfrac{1}{1 - \mu(x)} L$ and ω is relaxation parameter. In [4], the convergence of the iteration process is established and the choice of parameter ω is justified.

Let us consider the hybrid algorithm of the conjugate gradient method for sparse matrices. An ideological premise for using hybrid computing in handling sparse matrices of arbitrary structure is preliminary reduction of the structure of original matrix to sparse bordered block-diagonal form by means of the parallel sectioning method:

$$\widetilde{A} = P^T A P = \begin{pmatrix} D_1 & 0 & 0 & \dots & 0 & C_1 \\ 0 & D_2 & 0 & \dots & 0 & C_2 \\ 0 & 0 & D_3 & \dots & 0 & C_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & D_{p-1} & C_{p-1} \\ C_1 & C_2 & C_3 & \dots & C_{p-1} & D_p \end{pmatrix},$$

where P is permutation matrix, blocks D_i and C_i remain sparse, and p is the number of diagonal blocks in the matrix. For the parallel sectioning method it is natural that the order of the last diagonal block is much less than the orders of other diagonal blocks.

Thus, the original problem (18) reduces to the equivalent problem: $\widetilde{A}y = \lambda y$.

To store data with regard for the structure of the obtained matrix $\widetilde{A}$, it is expedient to use block distribution, and the number of blocks should be chosen with regard for the number of processors involved in the calculations. The following distribution of blocks (submatrices) among processes on CPU is implemented: processes with numbers $0 \le i < p-1$ keep blocks D_{i+1} and C_{i+1} and process with number $p-1$ keeps block D_p. Here, p is the total number of processes.

Parallel realization of the hybrid algorithm of the conjugate gradient method on the basis of the method of symmetric upper relaxation for sparse bordered block-diagonal matrix will be defined according to (19)–(21) mainly by the solution of two systems:

$$(I - \omega\widetilde{\widetilde{L}}(x))y = z(x),\ (I - \omega\widetilde{\widetilde{L}}^T(x))w = y(x),$$

where $z(x) = \omega(2-\omega)f(x)$; here, $w^k = p^k - \beta_k p^{k-1}$.

The sparse bordered block-triangular matrix has the form

$$\widetilde{\widetilde{L}} = \begin{pmatrix} \widetilde{L}_1 \\ \widetilde{L}_2 \\ \widetilde{L}_3 \\ \vdots \\ \widetilde{L}_{p-1} \\ \widetilde{L}_p \end{pmatrix} = \begin{pmatrix} \widetilde{D}_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \widetilde{D}_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & \widetilde{D}_3 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & \widetilde{D}_{p-1} & 0 \\ C_1 & C_2 & C_3 & \dots & C_{p-1} & \widetilde{D}_p \end{pmatrix}.$$

The algorithm of solution of system $(I - \omega\widetilde{L}(x))y = z$ with the lower triangular matrix reduces to simultaneous and independent solution of triangular systems by each process (except for the last one): $(I - \omega\widetilde{D}_q(x))y_q = z_q$; q ($1 \le q < p$) is the number of the process (CPU), and subsequent calculation of $\widetilde{y}_q = C_q(x)y_q$.

Then all the processes send $\widetilde{y}_q$ to the last (pth) process, which calculates vector y_p by solving the system $(I - \omega\widetilde{D}_p(x))y_p = z_p - \sum_{q-1}^{p-1}\widetilde{y}_q$.

The system $(I - \omega\widetilde{L}^T)w = y$ is solved similarly. The pth process solves system $(I - \omega\widetilde{D}_p^T(x))w_p = y_p$ and then sends corresponding components w_p to the other processes, which independently solve systems $(I - \omega\widetilde{D}_q^T(x))w_q = y_q - C_q^T(x)w_p$.

In the hybrid algorithm of the conjugate gradient method, the most complicated arithmetic operations with respect to runtime are multiplication of a sparse matrix by a dense vector and solution of two systems with block-triangular sparse matrices.

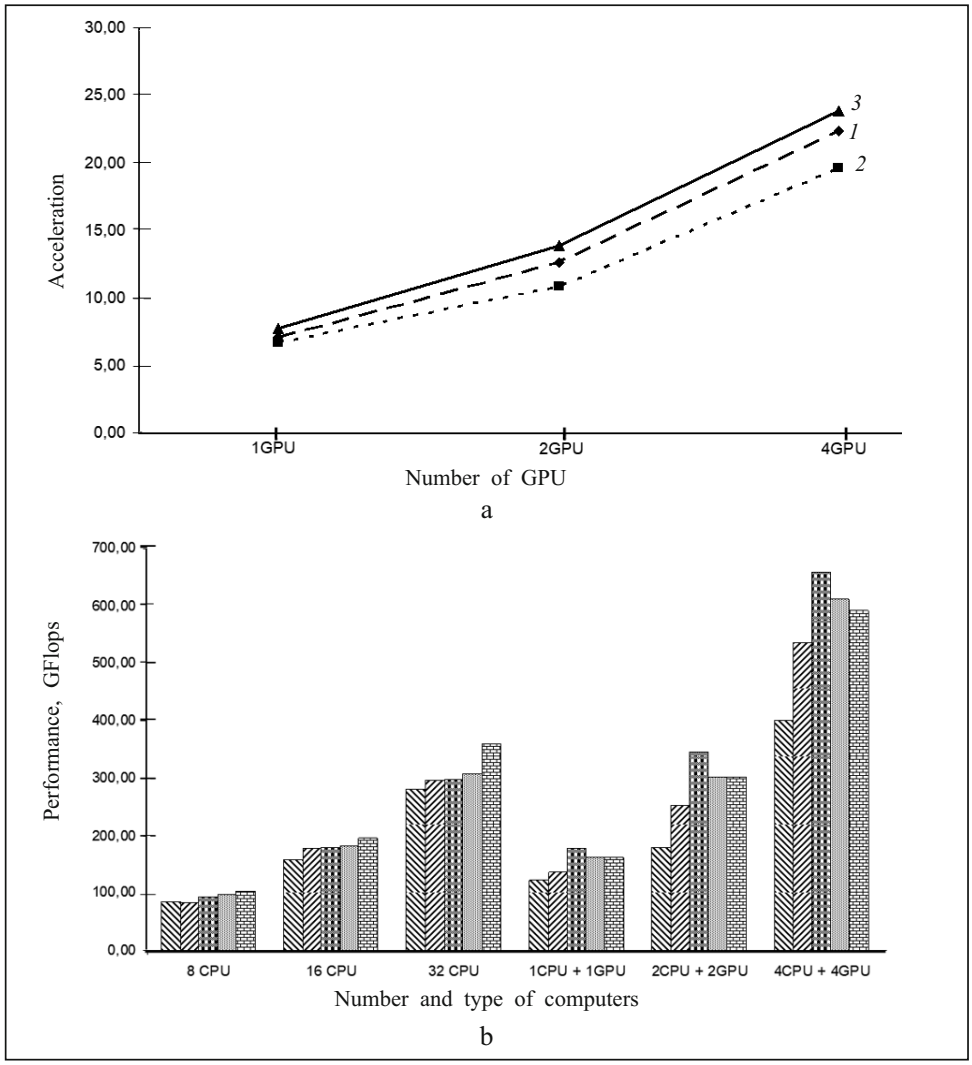

Fig. 2 (a) graphs of acceleration of the hybrid conjugate gradient algorithm for different numbers of GPU for problems G2_circui (*1*), Bone010 (*2*), Emila_923 (*3*); (b) diagram specifying the performance of the hybrid conjugate gradient algorithm for different numbers and types of computers for blocks — size 64, — size 128, — size 192, — size 256, — size 512.

To determine the efficiency of the hybrid algorithm (unless otherwise is specified), we will use the above notation and will in addition introduce the following notation: $nz(D_i)$ is the number of nonzero elements in the *i*th block of triangular matrix and n_i is the order of the respective block.

To find the acceleration, we will use the formula $S_p = T_1 / T_p$. We will take into account that the majority of operations are executed on GPU and we can neglect synchronization time and data transfers between CPU and GPU. Solution of triangular system for each diagonal block, except for the last one, involves $2nz(\widetilde{D}_i)+2nz(C_{pi})+3n_i$ arithmetic operations, multiplication of a sparse matrix by a dense vector involves $2nz(\widetilde{D}_i)+2nz(C_{pi})+n_i$ operations. Hence, the total number of arithmetic operations necessary to implement one iteration of the hybrid algorithm on one GPU can be estimated by $\sum_{i=1}^{p-2} O_{kGi}$, where $O_{kGi} = 10nz(\widetilde{D}_i)+10nz(C_{pi})+9n_i$, and time spent for one iteration makes

$T_1 = \sum_{i=1}^{p-1} O_{kGi} \frac{t_G}{n_o}$; O_k and O_{kG} are the number of arithmetic operations performed by one of k CPU and k GPU, respectively.

The time spent to execute one iteration of the hybrid algorithm on the architecture p CPU + p GPU is defined by $\max_{1 \le i \le p-1} O_{kGi} \frac{t_G}{n_o}$. In solving triangular systems, the amount of data transmitted between CPU is $n_p \log_2 p$, and in multiplying a sparse matrix by a dense vector it is $3(n_p \log_2 p)/2$.

Thus, the following theorem is true for the estimate of acceleration factor of hybrid algorithm.

THEOREM 2. If $n_p << \min_{1 \le i \le p-1} n_i$ and tree algorithm is used for multi-scatter and multi-gather operations, then the acceleration factor of the hybrid algorithm of the conjugate gradient method of the solution of eigenvalue problem of the bordered block-diagonal matrix (19) is

$$S_p \approx \frac{\sum_{i=1}^{p-1} O_{kGi} \frac{t_G}{n_o}}{\max_{1 \le i \le p-1} O_{kGi} \frac{t_G}{n_o} + t_{cG} n_p \log_2 p + t_{cG} 3 (n_p \log_2 p)/2},$$

where $O_{kGi} = 10nz(\tilde{D}_i) + 10nz(C_{pi}) + 9n_i$.

As well as for the algorithm of the method of iteration on a subspace, sparse matrices from [17] were used for the efficiency analysis.

Figure 2a shows the graphs of acceleration obtained in the problem solution for different orders of the matrix and different numbers of CPU and GPU processes used. These results testify that the developed algorithm can be scaled well and ensures uniform loading of computers without regard for their type, which confirms the theoretical results.

Figure 2b shows the diagram that specifies the performance of the hybrid algorithm. These results experimentally prove that the used approaches ensure high efficiency of paralleling of the computing process when several GPU are used.

NUMERICAL SOLUTION OF ONE STABILITY PROBLEM

The developed hybrid algorithms were validated for various practical problems, in particular, the problem of stability of a layered two-component composite of regular structure [18] under uniform monoaxial compression of reinforcing layers by a superficial load of constant intensity was solved.

For stability analysis of composite structures, the static method of three-dimensional linearized theory of stability within the limits of the second variant of the theory of small subcritical strains is applied. Subcritical state is defined by solution of a plane problem of the linear theory of elasticity of piecewise homogeneous bodies for various boundary conditions, which correspond to a three-layer periodicity element in case of a material and to a three-layer structure element with stress-free lateral faces in case of a composite sample.

When Euler's static method is used, the stability problem reduces to generalized eigenvalue problem in which minimum eigenvalue λ determines critical load and corresponding eigenfunction $u = (u_1, u_2)$ determines buckling mode.

The problem was solved with different initial data on a personal supercomputer of hybrid architecture Inparcom_pg [19] (one computational node, two four-core processors Xeon 5606, two GPU Tesla K40). The software that implements hybrid algorithms is written on C++ with the CPU using MPI system and software library Intel MKL [20]; for paralleling on GPU, CUDA technology and software libraries cuBLAS and cuSPARSE [21] were used. Below is the listing of the problem solution protocol: the order of matrices A and B: 12282, band half-width of matrix A: 6212, band half-width of matrix B: 71.

Fragment of the protocol of problem solution on Inparkom_pg

```
Problem solving: total time = 10.0033e+00
Results: SOLUTION WAS CALCULATED by 20 iterations
```

```
FIRST  3  EIGENVALUES
  Eigenvalues  (calculated)       Estimates  of  Errors
      1.248665556779830e-01              1.893e-06
      1.248734577580778e-01              8.894e-07
      1.248765574182433e-01              2.837e-06
```

The protocol presents three calculated minimum eigenvalues. It took 340 sec to solve this problem by MATLAB package [22]. The time of problem solution by the hybrid algorithm of the subspace iteration method on Inparkom_pg is 10.0033 sec, which is approximately 30 times faster than with the use of MATLAB.

The comparative analysis of the performance of the developed hybrid algorithms and of the respective parallel algorithms for MIMD-computers was carried out. For the parallel algorithm of the implicit conjugate gradient method, the greatest acceleration received for MIMD-computer with the use of eight processes was five to six times as compared with the sequential version of the program (1 CPU). Acceleration of 6.5 to 7.5 times was received for the hybrid algorithm with the use of one GPU, and 9 to 11 times for two GPU, as compared with the sequential version of the program. The results of solution by parallel and hybrid algorithms of the subspace iteration method have shown that using eight processes on CPU resulted in four times acceleration as compared with the sequential version of the program, and using GPU gave six and nine times acceleration for one and two GPU, respectively.

CONCLUSIONS

Further development of the presented schemes of hybrid algorithms (partition into blocks; distribution among processor devices; handling of blocks that correspond to nonzero blocks) lies in the plane of expansion of the types of treated structures with sparse matrices such as profile ones.

Testing of the proposed hybrid algorithms on the solution of practical problems shows that this direction in the development of algorithms and software is promising for mathematical modeling of problems of stability of materials and structures.

In future, it is expedient to modify the proposed hybrid algorithm of the subspace iteration method by replacing LL^T-decomposition of sparse symmetric matrix with its LDL^T-decomposition. This will allow a deeper a posteriori analysis of solution results (for example, [6]) and finding eigenvalues (and eigenvectors corresponding to them) in the middle of spectrum.

Further analysis of the developed hybrid algorithms will be directed to their realization for graphic accelerators of Maxwell and Pascal architecture, which allow implementing dynamic parallelism and direct data exchange between graphics accelerators without participation of the central processor. Moreover, optimization of these algorithms for new processors Intel Xeon Phi of the second generation is supposed. The developed algorithms are also supposed to be adjusted to using multilength arithmetics in solving ill-conditioned problems.

REFERENCES

1. S. Pissanetzky, Sparse Matrix Tehnology, Elsevier, Academic Press (1984).
2. V. G. Prikazchikov, “Prototypes of iterative processes in the eigenvalue problem,” Diff. Uravneniya, Vol. 16, No. 9, 1688–1697 (1980).
3. O. M. Khimich and O. V. Chistyakov, “Parallel one-step iterative methods to solve the algebraic eigenvalue problem for sparse matrices,” Komp. Matem., No. 2, 81–88 (2014).
4. G. V. Savinov, “Convergence analysis of one generalized conjugate gradient method to find extremum eigenvalues of the matrix,” Trans. Sci. Seminar LOMI, No. 111, 145–150 (1981).
5. B. N. Parlett, The Symmetric Eigenvalue Problem (Classics in Applied Mathematics), Book 20, Society for Industrial and Applied Mathematics (1987).
6. NetLib (2015), http://www.netlib.org/.
7. MAGMA (2015), http://icl.cs.utk.edu/magma/.

8. SLEPc (2015), http://slepc.upv.es/.
9. LIS (2015), http://www.ssisc.org/lis/.
10. A. N. Khimich, I. N. Molchanov, A. V. Popov, T. V. Chistyakova, and M. F. Yakovlev, Parallel Algorithms to Solve Problems in Calculus Mathematics [in Russian], Naukova Dumka, Kyiv (2008).
11. O. V. Chistyakov, "Special features of software development to solve eigenvalue problems with sparse matrices by hybrid computers," Komp. Matematika, No. 1, 75–84 (2015).
12. V. S. Mikhalevich, I. N. Molchanov, I. V. Sergienko, et al., Numerical methods for the ES Multiprocessor Computing Complex [in Russian], VVIA im. N. E. Zhukovskogo, Moscow (1986).
13. A. Yu. Baranov, A. V. Popov, Ya. E. Slobodyan, and A. N. Khimich, "Mathematical modeling of building constructions using hybrid computing systems," J. Autom. Inform. Sci., Vol. 49, Issue 7, 18–32 (2017).
14. O. M. Khimich and V. A. Sidoruk, "Hybrid algorithm to solve linear systems with sparse matrices based on the block LLT method," Komp. Matematika, No. 1, 67–74 (2015).
15. O. V. Popov, "Efficiency analysis of parallel algorithms for computers of hybrid architecture," in: Trans. Intern. Sci. School-Seminar "Optimization of Computations (POO XLII)" (2015).
16. A. N. Khimich, A. V. Popov, A. Yu. Baranov, and O. V. Chistyakov, "Hybrid algorithm to solve eigenvalue problems for band matrices," Teoriya Optym. Rishen', 86–94 (2016).
17. The SuiteSparse Matrix Collection (2015), https://cise.ufl.edu/research/sparse/matrices/.
18. V. A. Dekret, V. S. Zelenskii, and V. M. Bystrov, Numerical Stability Analysis of a Layered Composite Material under Monoaxial Compression of the Filler [in Russian], Naukova Dumka, Kyiv (2008).
19. O. M. Khimich, I. M. Molchanov, O. V. Popov, et al., "Smart personal supercomputer to solve scientific and technical problems," Nauka ta Innovatsii, Vol. 12, No. 5, 17–31 (2016).
20. Intel Maths Kernel Library (2015), https://software.intel.com/en-us/mkl.
21. CUDA library (2015), https://nvidia.com/.
22. MATLAB (2017), https://mathworks.com/.