Proceedings of the Federated Conference on Computer Science and Information Systems pp. 497–506
DOI: 10.15439/2017F64
ISSN 2300-5963 ACSIS, Vol. 11

Block Subspace Projection Preconditioned Conjugate Gradient Method for Structural Modal Analysis

Sergiy Fialko
Tadeusz Kościuszko Cracow University of Technology
ul. Warszawska 24 St., 31-155 Kraków, Poland
Email: sergiy.fialko@gmail.com

Viktor Karpilovskyi
IT company SCAD Soft
ul. Osvity 3a, office 1, 2, Kiev, Ukraine
Email: kvs@scadsoft.com

***Abstract*— The method for extracting natural vibration frequencies and modes of design models arising when the finite element method is applied to the problems of structural and solid mechanics is proposed. This approach is intended to be used on multicore SMP computers and is an alternative to the conventional block Lanczos and subspace iteration methods widely used in modern FEA software. We present the main idea of the method as well as the parallel fast block incomplete factorization approach for creating efficient preconditioning, the shift technique and other details accelerating the solution and improving the numerical stability. Real-life examples are taken from the computational practice of SCAD Soft IT company and approve the efficiency of the proposed method.**

I. Introduction

THE application of the finite element method to the problems of natural vibrations of structures results in a generalized algebraic eigenvalue problem

$$\mathbf{KV} - \mathbf{MV\Lambda} = 0 , \tag{1}$$

where **K** and **M** are the symmetric positive definite stiffness and semidefinite mass sparse matrices, $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$ – matrix of eigenvectors $\mathbf{v}_i$, located in **V** column-by-column, **Λ** is a diagonal matrix of eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$, $\lambda_i = \omega_i^2$, $\omega_i = 2\pi f_i$, $i \in [1, n]$, ω_i is a cyclic frequency in s^{-1} and f_i is a frequency in Hz. The dimension of the problem is N and the number of required eigenpairs is $n << N$.

For large finite element design models the problem dimension N reaches 200 000 – 6 000 000 equations and more. The required number of eigenpairs $\{\lambda_i, \mathbf{v}_i\}$, $i \in [1, n]$ depends on the type of dynamic analysis and properties of construction. Usually n = 20 – 100, but in the case of seismic analysis it can be 1 000 – 3 000 and more. Some of the constructions have a lot of local vibration modes in the lower part of the spectrum. Such modes produce very small contributions in a seismic response of the structure, but create huge difficulties for eigenvalue solvers, because a very large number of eigenpairs are required in such cases.

The block Lanczos method or block subspace iteration method is used most often in contemporary FEA software. But these powerful methods use the inverse iteration procedure on each iteration step which requires the twice reading of the factorized stiffness matrix [10]. Most users prefer to solve these problems on laptops and desktops, which have the amount of RAM 8 – 16 GB. In the case of a large dimensionality of the problem, the amount of core memory is insufficient for storing a factorized stiffness matrix, which is stored on the disk. Therefore, when performing forward and backward substitutions at each iteration, we need to read twice from the disk the amount of data on the order of 6 – 20 or more GB. The above methods work with the speed of a slow disk, not a fast processor, and it takes many hours to solve the problem.

Therefore, it seems interesting to develop a method that would solve the problem (1) in a core memory. Our choice is based on the preconditioned conjugate gradient method (PCG). It is known that for many problems of structural mechanics, which are poorly conditioned for a number of reasons [21], the conjugate gradient method demonstrates unacceptably slow convergence. In order to correct the situation, it is necessary to create efficient preconditioning [2], [3], [5], [6], [8], [9], [13] – [15], [19], [20], [22]. Our experience shows that a stable convergence of the eigenvalue problem (1) is much more difficult to obtain than when solving a system of linear algebraic equations

$$\mathbf{Kx} = b , \tag{2}$$

where **x** and **b** are respectively the unknown vector and the right-hand part vector. Therefore, the successful solution of the problem (1) usually requires more effective preconditioning than when solving the problem (2). In addition, in order to obtain the stable convergence in the presence of multiple and close eigenvalues, it is required to introduce the shift into preconditioning

$$(B - \sigma M) z_i^k = r_i^k , \tag{3}$$

where **B** is a preconditioning operator without shift, σ is a shift, $\mathbf{z}_i^k$ – residual vector for a preconditioned problem and k is an iteration step number. The preconditioned algebraic eigenvalue problem is formulated as

$$B_\sigma^{-1}(\mathbf{KV} - \mathbf{MV\Lambda}) = 0 , \tag{4}$$

This work was supported by IT company SCAD Soft (www.scadsoft.com)

where $\mathbf{B}_\sigma = \mathbf{B} - \sigma\mathbf{M}$. The residual vector of an initial problem (1) is:

$$\mathbf{r}_i^k = \lambda_i^k \mathbf{M}\mathbf{x}_i^k - \mathbf{K}\mathbf{x}_i^k . \quad (5)$$

Here subscript i denotes a mode number and $\mathbf{x}_i^k$, λ_i^k are the approximations of the i-th eigenmode and eigenvalue on iteration step k.

The article [5] presents PCG method with element-by-element aggregation multilevel preconditioning [6] and shift technique. This method does not use multithreading, it was implemented in SCAD software in 2004 and enables to extract a relatively small number of eigenpairs (5 – 30). A parallel version of PCG method has been proposed in [9], but acceleration with the increasing number of threads was poor.

The local block PCG method (LOBPCG – [17], [18], [25]) uses the following approximation:

$$\begin{cases} \mathbf{x}_i^{k+1} = \sum_{j=1}^{m} \alpha_j^k \mathbf{z}_j^k + \sum_{j=1}^{m} \tau_j^k \mathbf{x}_j^k + \sum_{j=1}^{m} \gamma_j^k \mathbf{p}_j^k \\ \mathbf{p}_i^{k+1} = \sum_{j=1}^{m} \alpha_j^k \mathbf{z}_j^k + \sum_{j=1}^{m} \gamma_j^k \mathbf{p}_j^k \end{cases}, \quad i \in [1, m] \quad (6)$$

where $\mathbf{p}_j^k$ is a conjugate direction vector and m is a dimension of the block. The dimension of the block $m \geq n$ and is constant until all required eigenpairs are extracted. For the problems of structural mechanics, we found that as soon as the first eigenpair begins to converge, the method loses the computational stability (see section IV, *A*), because for a converged eigenpair the residual vector $\mathbf{r}_i^k$ in (5) tends to zero, vector $\mathbf{z}_i^k$ tends to zero too (3) and a zero column appears in the projection matrix $\mathbf{Q}_k = \{\mathbf{Z}_k, \mathbf{X}_k, \mathbf{P}_k\}$, where $\mathbf{Z}_k = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_m\}^k$, $\mathbf{X}_k = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}^k$ and $\mathbf{P}_k = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}^k$.

To ensure the computational stability of the method, we keep a constant dimension of the block $m < n$, and as soon as some vectors in the block converge, we immediately remove them, store them as the final results and replace them with the new start vectors. In addition, when the columns in the projection matrix $\mathbf{Q}_k$ become almost linearly dependent, we orthogonalize all the vectors in the block using the modified Gram-Schmidt method.

This article is a continuation of [10], and we focus our attention on a block parallel sparse Cholesky incomplete factorization method used for a fast creation of efficient preconditioning, shift technique, allowing to improve the computational stability of PCG method and other important moments of the proposed approach.

II. Block subspace projection preconditioned conjugate method

The details of our approach have been presented in [10], therefore here we briefly mention their general stages.

A. Initialization

To ensure a load balance between threads, we accept that dimension of block m is multiple to the number of threads np: $m\%np = 0$. Usually, we accept $m \in [16, 64]$. There are no strict recommendations, and the value of m depends on the number of required eigenpairs and peculiarities of the problem. We prepare the block of linearly independent start vectors $\mathbf{X}_0$, set $\mathbf{P}_0 = 0$ and start the iteration process $k = 0, 1, \ldots$, until all required eigenpairs are extracted.

B. Computing of residual vectors

On the k iteration step we obtain the residual vectors using (5) and replacing the subscript i by $j \in [1, m]$. The approximations of eigenvalues are computed applying the Rayleigh quotient

$$\lambda_j^k = \left(\left(\mathbf{x}_j^k\right)^T \mathbf{K}\mathbf{x}_j^k\right) \Big/ \left(\left(\mathbf{x}_j^k\right)^T \mathbf{M}\mathbf{x}_j^k\right). \quad (7)$$

Using (3) ($i \to j$), we receive the block of vectors $\mathbf{Z}_k$. The procedures (5), (7) and (3) are produced in a parallel region because for each mode $j \in [1, m]$ we can run all computations separately.

The check of convergence is performed. If $\|\mathbf{r}_j^k\|_2 / \lambda_i^k < tol$, where tol is a required tolerance, convergence is achieved, all approximations of eigenpairs in the block, $\{\lambda_j^k, \mathbf{x}_j^k\}$, satisfying this condition, are stored as the final results. The new start linearly independent vectors $\mathbf{x}_j^k$ are generated and put instead of the converged vectors. The orthogonalization of these vectors against all converged eigenvectors is performed. The conjugate direction vectors, corresponding to new start vectors, are accepted as $\mathbf{p}_j^k = 0$. The evaluation of approximations of eigenvalues (7) for the new start vectors, residual vectors $\mathbf{r}_j^k$ (5) and vectors $\mathbf{z}_j^k$ (3) are produced in a parallel region because it often turns out that several vectors are converged. The new vectors $\mathbf{z}_j^k$, $\mathbf{x}_j^k$ and $\mathbf{p}_j^k$ are located on positions of converged and removed vectors in blocks $\mathbf{Z}_k$, $\mathbf{X}_k$ and $\mathbf{P}_k$.

C. Projection of mass and stiffness matrices on the subspace

The reduced matrix $\mathbf{m} = \mathbf{Q}_k^T \mathbf{M} \mathbf{Q}_k$ is prepared in a parallel region. The developed algorithm bypasses zero columns $\mathbf{p}_j^k$ in the projection matrix $\mathbf{Q}_k$, if any appeared at the previous step *B*. If Cholesky factorization of matrix $\mathbf{m}$ is successful, we evaluate the reduced matrix $\mathbf{k} = \mathbf{Q}_k^T \mathbf{K} \mathbf{Q}_k$ in a parallel region. Otherwise, the total reorthogonalization of columns in the projection matrix $\mathbf{Q}_k$ is applied to ensure the linear independence of basis vectors. In comparison with the previous version [10] we have parallelized this algorithm (section V). After reorthogonalization procedure, we recalculate the matrix $\mathbf{m}$ and prepare the matrix $\mathbf{k}$ in a parallel region. After this, we solve the reduced eigenproblem

$$\mathbf{k}\mathbf{q} - \mathbf{m}\mathbf{q}\boldsymbol{\mu} = 0 , \quad (8)$$

where $\mathbf{q}$ is a matrix of eigenvectors located column by column, and $\boldsymbol{\mu}$ is a diagonal matrix of eigenvalues. The LAPACK procedures from Intel Math Kernel Library (Intel MKL) [26] are applied. We omit a subscript k denoting the iteration number.

D. Evaluation of basis vectors at the next iteration step

After solving (8), the eigenvectors in matrix $\mathbf{q}$ have been sorted in the ascending order of eigenvalues. Then, we compute:

$$\mathbf{X}_{k+1} = \mathbf{Q}_k \overline{\mathbf{q}}, \quad \mathbf{P}_{k+1} = \mathbf{Z}_k \mathbf{q}_z + \mathbf{P}_k \mathbf{q}_p, \tag{9}$$

where $\overline{\mathbf{q}}$ – the first m eigenvectors from $\mathbf{q}$, $\mathbf{q}_z$ and $\mathbf{q}_p$ – the blocks of subvectors from $\overline{\mathbf{q}}$, related to residual vectors and conjugate direction vectors respectively.

E. Orthogonalization of basis vectors against all converged eigenpairs.

We perform the orthogonalization of columns in subblocks $\mathbf{X}$ and $\mathbf{P}$ against all converged modes in a parallel region.

III. The Block Parallel Incomplete Cholesky Factorization

We found that a stable solution of the partial algebraic generalized eigenvalue problem (1) by the PCG method requires a more efficient preconditioned technique than the solution of the linear algebraic equations with the same stiffness matrix $\mathbf{K}$. It means that in the case of the incomplete Cholesky factorization approach we must accept a much smaller drop parameter ψ than when we solve the linear equations. In the last case, we apply the incomplete Cholesky factorization using a sparse matrix technique [8]. However, it turned out that in order to solve many large-scale real-time problems of natural vibrations, the drop parameter ψ must be so small that the time of incomplete factorization by this method becomes unacceptably large. Therefore, there was an urgent need to develop a left-looking two-level incomplete Cholesky solver “by value” for multi-core SMP computers.

First of all, we prepare a nodal adjacency graph and perform its reordering with the help of the METIS or MMD reordering method [16]. Then, we assemble a stiffness matrix, the lower triangular part of which is packed in a compressed column format (CCF). It is source information for the solver.

A. Analysis stage

Taking the nonzero structure of the stiffness matrix from CCF, we prepare an equation adjacency graph and produce a symbolic factorization procedure [4] to create a nonzero structure of the completely factorized matrix, which is accepted as an initial nonzero structure for the incomplete factorization.

Then, we create an elimination tree and reorder the sequence of elimination in accordance with moving along the elimination tree from the leaves to the root. Such reordering does not change the amount of fill-in, but puts the columns of the matrix, processed consequently, in the close memory addresses and slightly improves the work with the caches of processors. The symbolic factorization procedure runs again to correct the nonzero structure of the factorized matrix for a changed sequence of elimination and CCF for source matrix is repacked.

After this, we combine the neighbor vertexes of the elimination tree in supernodes, where it is possible [12], and create a structure of levels for the supernodal elimination tree. It is well-known that the supernodal elimination tree is poorly balanced to ensure a load balance between processors. In order to balance the supernodal elimination tree, we apply the algorithm 1 (Figure 1).

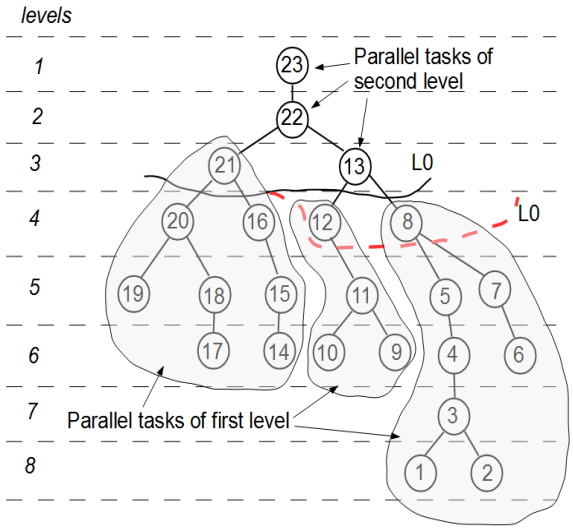

Fig. 1 Balancing of supernodal elimination tree

Algorithm 1. The task scheduling for the first parallelization level.

level = 1;
root ← *str_lev*(*level*); put vertexes of *level* to *root*
L0 ← *root*; put *root* to L0
while(1)
{
  sum_weights ← 0;
  ∀v ∈ L0: $weight_v$ ← *get_weight_subtree*(v);
  Find: *max_weight* = *max*{$weight_v$} and v_{max_weight};
  L0 ← (L0\ v_{max_weight});
  L0 ← *all descendants of* v_{max_weight}
  sort{L0};
  ∀v ∈ L0:
    sum_weights[*ip_min*] ← *get_weight_subtree*(v);
    stack[*ip_min*] ← v;
  if((*max*{ *sum_weights* } – *min*{ *sum_weights* })/

max{ *sum_weights* } < tol_1) break;
}

Here we denote: *str_lev* is a structure of levels for supernodal elimination tree and *str_lev*(*level*) returns all vertexes belonging to *level*; *get_weight_subtree*(v) returns the sum of weights for vertexes of all subtrees, outgoing from vertex v; weight of vertex is equal to the number of columns in the given supernode, L0 – "level zero", sort{L0} is a sorting of all vertexes belonging to L0 in the descending order of their weights. All vertexes of supernodal elimination tree, placed above L0, create a second level of parallelization. Remaining vertexes constitute the first level. Parameter *ip_min* means the thread number which comprises a minimum sum of weights of all supernodes, mapped onto this thread.

Algorithm 1 searches for a set of L0 until an imbalance of weights for vertices of the first level is less than the required tolerance tol_1. The general operations include finding a vertex with the maximum weight of subtrees among all vertexes of L0 (∀v ∈ L0: $weight_v$ ← *get_weight_subtree*(v); Find: *max_weight* = *max*{$weight_v$} and v_{max_weight};), removing this vertex from L0 (L0 ← (L0\ v_{max_weight});) and adding all descendants of removed vertex to L0 (L0 ← all descendants of v_{max_weight}) (see Figure 1). A similar algorithm but with cyclic mapping onto threads has been used in [1]. We found that sorting vertexes in L0 in the descending order of weights and mapping of the current vertex from L0 to the thread which has a minimum sum of weights, results in a better load balance than cyclic mapping [11], [12].

After algorithm 1 is finished, we add the vertexes of all subtrees to vertexes of L0, belonging to *stack*[*ip*], where *ip* ∈ [0, *np*-1] and *ip* is a current thread number. So, we obtained the set of parallel tasks in *stack*[*ip*], *ip* ∈ [0, *np*-1] for the first level of parallelism. Each vertex, added to any stack[ip], *ip* ∈ [0, *np*-1], is marked.

Then, we run Algorithm 2 to prepare the parallel tasks for the second level of parallelism.

Algorithm 2. The task scheduling for the second parallelization level.

sum_weights ← 0; L1 ← 0;
while(*until all vertexes are marked*)
{
  loop ∀v ∈ L0: → v_{next};
  if(v_{next} *is unmarked*)
  {
    marks v_{next};
    L1 ← v_{next};
    $weight_v$ ← *weight*(v_{next})
    queue[*ip_min*] ← v_{next};
    sum_weights[*ip_min*] += $weight_v$;
  }
  end of **loop**
  L0 ← 0; L0 ← L1; L1 ← 0;
}

Loop *while* runs until all vertexes of supernodal elimination tree are marked as added to the parallel tasks. For each vertex belonging to L0 (**loop** ∀v ∈ L0), we obtain its parent vertex v_{next}. If v_{next} is marked, pass to the next vertex of L0. Otherwise, we mark v_{next} , add it to the set L1, map onto thread *ip_min* (*queue*[*ip_min*] ← v_{next}), which has a minimum sum of weights of mapped vertexes, and add the weight of v_{next} to *sum_weights*[*ip_min*] += $weight_v$. After **loop** ∀v ∈ L0 is finished, we reset L0 and L1 to the next iteration.

The parallel tasks of the second level of parallelism are presented by queues *queue*[*ip*], *ip* ∈ [0, *np*-1]. After Algorithms 1 and 2 have been run, all vertexes of the supernodal elimination tree must be marked.

B. Numerical factorization stage

Algorithm 3 presents the numerical factorization stage.

Algorithm 3. The two-level left-looking numerical factorization.

parallel region
{
  //*the first level of parallelization*
  while(*stack*[*ip*] is not empty)
  {
    jb ← *stack*[*ip*] (*stack*[*ip*]*jb*)
    alloc_block(*jb*);
    aggreg_block(*jb*);
    if(!*update_block*(*jb*))
      { push_front: *stack*[*ip*] ← *jb*; *continue*; }
    factor_block(*jb*);
  }

  //the second level of parallelization
  while(*queue*[*ip*] is not empty)
  {
    jb ← *queue*[*ip*] (*queue*[*ip*]*jb*)
    alloc_block(*jb*);
    aggreg_block(*jb*);
  if(!*update_block*(*jb*))
      { push_back: *queue*[*ip*] ← *jb*; *continue*; }
    factor_block(*jb*);
  }
}

In a parallel region of the first parallelization level, we run the loops **while** until the *stack*[*ip*], corresponding to the

thread $ip \in [0, np\text{-}1]$, is empty. On each iteration, we extract from $stack[ip]$ the last element jb (the number of block-column in the matrix, corresponding to the supernode in the supernodal elimination tree), and remove it from $stack[ip]$ ($stack[ip] \setminus jb$).

Then, we allocate memory for sparse block-column jb, calling the procedure *allock_block*(jb), assemble a block-column jb (*aggreg_block*(jb)) and update it by columns placed at left (*update_block*(jb)).

The procedure *allock_block*(jb) prepares the list of global equation numbers for non-zero rows of block-column jb. The nonzero rows of the block-column jb of the initial stiffness matrix as well as the fill-in that appeared at the previous steps of the incomplete factorization take part in the creation of this list. Then we apply the parallel dynamic allocation of memory, using a separate heap for each thread and the Thread Local Storage technique [24].

The *aggreg_block*(jb) procedure puts the elements of the initial stiffness matrix stored in CCF to the non-zero structure of block-column jb.

The *update_block*(jb) procedure in the critical section extracts from $List[jb]$ the block-column number kb, $kb \in List[jb]$, placed at left from the block-column jb and updating it, removes kb from $List[jb]$ and checks, whether the block-column kb is currently factorized. If yes, the update of block-column jb is performed using the *dgemm* procedure from Intel MKL (see Figure 2):

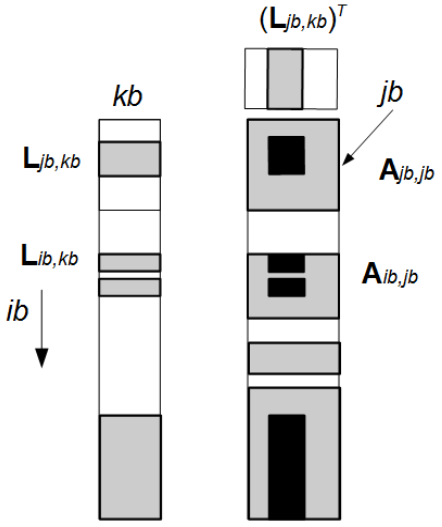

Fig. 2 An update of the block-column jb by block-column kb, placed at left. The black areas correspond to the updated elements of the block-column jb.

$$\mathbf{A}_{ib,jb} = \mathbf{A}_{ib,jb} - \mathbf{L}_{ib,kb} \cdot \mathbf{L}^{T}_{jb,kb}, \tag{10}$$

where ib belongs to the non-zero structure of the block-column jb.

Otherwise (the block-column kb is not currently factorized) the number kb puts to the end of $List[jb]$, and the next element of $List[jb]$ is extracted. When all the block-columns from the $List[jb]$ have participated in the correction of the block-column jb and $List[jb]$ is empty, the procedure *update_block*(jb) returns *true*. Otherwise, the *update_block*(jb) returns *false*, and the given block-column jb is pushed in $stack[ip]$ and we have to wait until the remaining threads factorize all the block-columns from $List[jb]$. Such a technique is similar to [23].

The *factor_block*(jb) procedure produces a Cholesky factorization of the diagonal block using the *dpotrf* procedure from Intel MKL [26]:

$$\mathbf{A}_{jb,jb} = \mathbf{L}_{jb,jb} \cdot \mathbf{L}^{T}_{jb,jb}, \tag{11}$$

and updates the off-diagonal part of the block-column jb, applying the *dtrsm* procedure from Intel MKL [26]:

$$\mathbf{L}_{jb,jb} \cdot \mathbf{L}_{ib,jb} = \mathbf{A}_{ib,jb} \quad \rightarrow \quad \mathbf{L}_{ib,jb}, \tag{12}$$

where a lower triangular matrix $\mathbf{L}_{jb,jb}$ has been obtained in (11) and ib belongs to the nonzero structure of the block-column jb.

After this, we run the *drop_proc*(jb) procedure, presented by algorithm 4.

Algorithm 4. The drop procedure.

```
do iloc = 0, M – 1
{
  d_jj = ∑H_ij, j ∈ [N_s, N_e]
  s_i = ∑H_ij^2
  d_ii = B_ii
  if(s_i < ψ^N d_jj d_ii)
  {
    reject_list ← iloc;   s ← 0;
    for(j= N_s; j≤ N_e; ++j)
    {
      H_jj += |H_ij|√(H_jj/d_ii);
      s += |H_ij|/√(H_jj/d_ii)
    }
    critical section
      diag_add[i] = s;
    end critical section
  }
}
```

Here M is the number of non-zero rows in the block-column jb, $iloc$ – the current local row number in block, N_s – the global number of the first column in the block, N_e – the global number of the last column in the block, d_{jj} is a sum of diagonal elements in the block-column jb, s_i is a sum of elements in the i-th row (i belongs to the nonzero structure of the block-column jb), H_{ij} is an element of the factorized matrix, i and j are the global subscripts, $B_{ii} = H_{ii}$, if the block-column comprising an element H_{ii} has been already factorized, and $B_{ii} = A_{ii}$ – an element of the initial

matrix otherwise. If the sum of squares of elements in the row i is less than $\psi^N d_{jj} d_{ii}$, we put i number to *reject_list*, and correct the diagonal elements to ensure the positive definiteness of the lower triangular part of the incomplete factorization **H**. The value s corrects the diagonal element Dii, which at the current time is not factorized. Such a correction is accumulated in the array *diag_add* and will be used later. Several threads can produce this correction simultaneously, therefore we use a critical section.

After loop **do** is over, we remove from the nonzero structure of the block-column jb the entire rows, stored in *reject_list*, compress the block-column jb and reallocate the memory in order to free up the amount of memory occupied by the rejected rows of the block-column.

When *stack*[*ip*] is empty (see Algorithm 3), the thread ip begins to run the second loop **while** from the second level of parallelization. The nearest element of *queue*[*ip*] is extracted and removed from the queue. If *update_block*(*jb*) returns a *false*, jb is pushed at the end of *queue*[*ip*] and will be processing later.

After the numeric factorization is finished, we start the post-factorization drop procedure, which is similar to the one, presented in Algorithm 4. The post-drop procedure uses a post-drop parameter ψ_1 instead of the drop parameter ψ ($0 \le \psi \le \psi_1 < 1$) and does not produce the correction of the diagonal entries. This approach allows us to maintain a low level of error accumulation if the value of the parameter ψ is small because every rejection in the incomplete factorization process leads to the accumulation of errors [8]. And only after the factorization is completed, secondary dropping is performed to reduce the amount of data in the preconditioning and accelerate the procedure (3) without considerable degradation of the quality of preconditioning. In addition, if $\psi = 0$ and $\psi_1 > 0$, the proposed method produces the complete Cholesky factorization and then makes a post-factorization dropping. For large poorly conditioned problems, the application of this approach can be very efficient if the capacity of the core memory allows the allocation of the completely factorized matrix.

IV. SHIFT TECHNIQUE

It is widely known ([5], [7], [9] and others) that the application of a properly selected shift accelerates the convergence of methods based on the iteration by the inverse matrix. In the PCG method, based on minimizing of Rayleigh quotient, we introduce a shift into preconditioning – see (3), (4). We do not evaluate $\mathbf{B}_\sigma$ directly and use the iterative procedure [5], [9] when solving the system of linear equations (3). Let us assume that $\hat{\mathbf{z}}_i^k$ is an approximation of the exact vector $\mathbf{z}_i^k$ and $\mathbf{q}$ is a small correction:

$$\hat{\mathbf{z}}_i^k = \mathbf{z}_i^k + \mathbf{q} . \tag{13}$$

Then, after substituting (13) in (3) we obtain:

$$\mathbf{Bq} = \sigma\mathbf{M}\hat{\mathbf{z}}_i^k + \underbrace{\mathbf{r}_i^k - \mathbf{B}\hat{\mathbf{z}}_i^k + \sigma\mathbf{Mq}}_{\text{is dropped comparing with the first term}} . \tag{14}$$

Due to the assumption that $\mathbf{q}$ is a *small* correction, we neglect $\sigma\mathbf{Mq}$ in comparison with other terms and accept $\mathbf{B}\hat{\mathbf{z}}_i^k \approx \mathbf{r}_i^k$. In addition, the iterative procedure for the solution of (3) is:

Algorithm 5. Iterative solution of (3)

$\mathbf{B}\hat{\mathbf{z}}_i^k = \mathbf{r}_i^k \quad \rightarrow \quad \hat{\mathbf{z}}_i^k$
do $s = 1, 2, ,\ldots, S$
  $\mathbf{Bq} = \sigma\mathbf{M}\hat{\mathbf{z}}_i^k \quad \rightarrow \quad \mathbf{q}$
  $\hat{\mathbf{z}}_i^k = \hat{\mathbf{z}}_i^k + \mathbf{q}$
end do

Usually, 1 – 2 iterations are required. We start the natural vibration problem analysis with $\sigma = 0$. The shift value is corrected at the iteration step k, where the convergence of at least one eigenpair is achieved, and new starting vectors are added to the block, or when five iterations are performed, on which no eigenpair has converged. The new value of shift is taken as a current approximation of the eigenvalue $\lambda_{i\_shift}^k$, where $i\_shift = (m - 1)/4 + 1$ and m is the dimension of the block.

V. THE ALGORITHM OF TOTAL REORTHOGONALIZATION

When the Cholesky factorization of the reduced mass matrix **m** has failed (section II,*C*), we perform the total reorthogonalization (Algorithm 6) of the columns in the matrix $\mathbf{Q}_k$.

Algorithm 6. The parallel reorthogonalization of the columns of the matrix $\mathbf{Q}_k$

Parallel loop for i =1, 3m
{
  $\mathbf{q}_i = \mathbf{q}_i / \sqrt{(\mathbf{q}_i^T \mathbf{M} \mathbf{q}_i)}$
}

do *i =2, 3m*
  $\mathbf{w} \leftarrow \mathbf{Mq}_i$
  Parallel loop for j = 1, i – 1
  {
    $\beta_{ij} = \mathbf{q}_j^T \cdot \mathbf{w}$
  }

  Parallel loop for l = 1, *N*, *schedule* (*static*, *chunk*)

```
    {
      do j = 1, i – 1
        q^l_i ← q^l_i – β_ij q^l_j
      end do
    }

    q_i ← q_i/√(q_i^T M q_i)
  end do
```

In the first parallel loop, we perform the normalization of all columns in the matrix $\mathbf{Q}_k$. Then, we apply the modified Gram-Schmidt orthogonalization method (loop **do** *i =2, 3m*). The second parallel loop (*Parallel loop for* $j = 1, i - 1$) calculates $\beta_{ij} = \mathbf{q}_j^T\mathbf{M}\mathbf{q}_i$. Here it is very important to ensure the coherence in caches of different processors, and we make a padding of the array for β_{ij} to exceed the dimension of the cache line – 64B. The third parallel loop (*Parallel loop for* $l = 1, N$) covers all elements of vectors $\mathbf{q}_i$ and $\mathbf{q}_j$. It is very important to ensure the coherence in caches too, and we accept a chunk size as the 16 words of double. The inner loop **do** covers the number of vectors j. Finally, each corrected vector $\mathbf{q}_i$ should be normalized.

We underline that it is very important to ensure the coherence in caches of different processors because otherwise, we obtain a drastic degradation of performance at least on the AMD Opteron 6276 processor, not protected at the hardware level. The second column in Table III demonstrates the efficiency of the proposed parallel algorithm.

VI. Numerical Results

Let us consider examples taken from the collection of SCAD Soft (http://www.scadsoft.com) — IT Company, developer of the SCAD FEA software, one of the most popular softwares used in the CIS countries for structural analysis and design, certified according to the regional standards.

We use the computer A with 16-core processor AMD Opteron 6276, 2.3/3.2 GHz, 64 GB DDR3 RAM, OS Windows Server 2008 R2 Enterprise SP1, 64 bit, and computer B with 4-core processor Intel® Core™ i5 – 2500 CPU 3.30 GHz, 24 GB DDR3 RAM, OS Windows 7, 64 bit.

Computer A is a workstation and computer B – usual desktop.

A. Problem 1

The uniform beam (Figure 3) is considered.

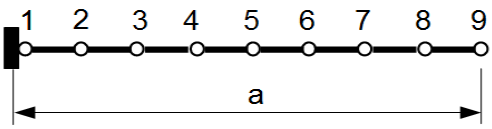

Fig. 3 The clamped beam

We accept: a = 2 m, E = 200 000 MPa, ρ = 7 600 kg/m^3, A = 0.001 m^2, I = 0.0001 m^4, where E is the Young's modulus, ρ – the material density, A – the cross-sectional area and I – the moment of inertia. Three eigenpairs are extracted (n = 3). The dimension of the problem N = 24 and the dimension of the block m = 3. The preconditioning parameters: reordering method is MMD (multiple minimum degree), $\psi = 10^{-16}$, $\psi_1 = 10^{-13}$, $tol = 10^{-6}$ (section II.*B*). Figure 4 presents the comparison of convergence for both: the LOBPCG method [17], [18] and the proposed block subspace projection PCG (BSPPCG) method. The minimal error for the iterated approximations of eigenvectors in the block is depicted on the vertical axis.

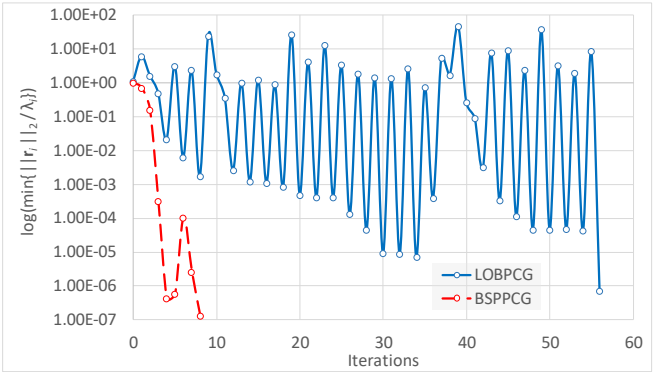

Fig. 4 The convergence of the LOBPCG and BSPPCG methods

The proposed BSPPCG method performs the control of the linear independence of the basis vectors in the subspace projection matrix $\mathbf{Q}_k$, decomposing the reduced mass matrix $\mathbf{m}$ by the Cholesky method (section II.*C*). If the Cholesky factorization of the $\mathbf{m}$ matrix has failed, the total reorthogonalization of columns in the projection matrix $\mathbf{Q}_k$ ensures the linear independence of the basis vectors. As soon as the first eigenpair begins to converge, the residual vector $\mathbf{r}_i^k$ corresponding to this pair tends to zero, since for an exact solution this vector must be strictly zero. The vector of the conjugate direction $\mathbf{p}_i^k$ behaves similarly since at the stationary point of the Rayleigh functional the gradient vector is also zero and its direction is not defined. Hence it follows that the convergence of eigenpairs leads to a linear dependence between the columns of the projection matrix $\mathbf{Q}_k$. Therefore, we remove the converged eigenvectors from the block, replacing them with the new start vectors, and restore the linear independence of the columns of the matrix $\mathbf{Q}_k$, if necessary. As a result, we obtain a fast and stable convergence for the BSPPCG method, and the given example demonstrates it.

B. Problem 2

The design model of the multi-storey building, resting on the soil, is shown in Figure 5. The number of equations is 2 989 476. Solid finite elements simulating the soil

behavior contribute a relatively dense part in the sparse stiffness matrix. The size of the completely factorized stiffness matrix using the METIS reordering [16] is 36.53 GB. Therefore, this problem is very hard for a direct solution. We accept the required number of eigenpairs n = 100, the dimension of the block m = 32. Parameters of preconditioning are as follows: METIS reordering, $\psi = 10^{-50}$, $\psi_1 = 10^{-13}$. The required tolerance is $tol = 10^{-3}$.

We compare the performance and speed up with an increase of the number of threads of the proposed incomplete block Cholesky solver with PARDISO from Intel MKL 11.3 accepting a complete factorization mode ($\psi = \psi_1 = 0$) because in this mode the incomplete solver processes the maximum amount of data.

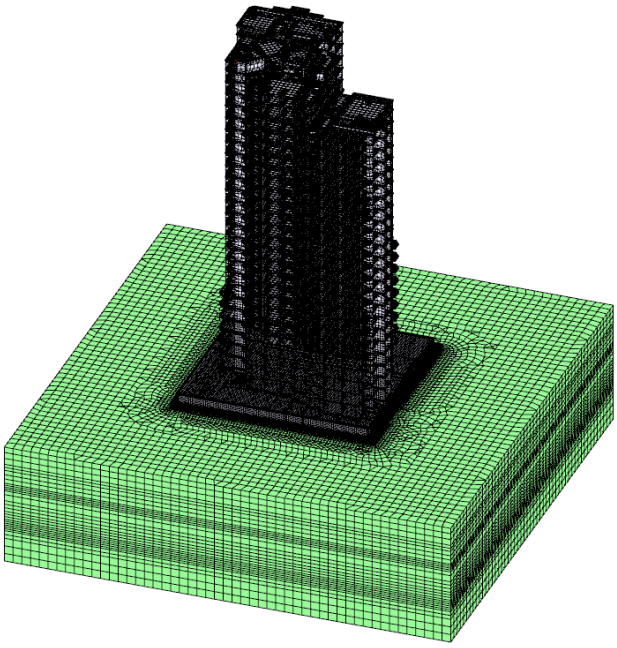

Fig. 5 Multi-storey building, based on a prism of soil

Moreover, for a lot of large poorly conditioned problems of structural and solid mechanics, we must take the parameter ψ as very small. Therefore, the mode of incomplete solver in such a case will be close to the complete factorization, which allows us on the scheduling of load on the processors based on the non-zero structure of the completely factorized matrix. If the problem admits a relatively large value of the dropping parameter ψ, the duration of the incomplete factorization considerably decreases and ceases to be critical in comparison with other stages of the solution of the problem, and we apply in such a situation the non-block version of the incomplete Cholesky solver [8], which results in better properties of preconditioning than the block Cholesky method due to dropping of single elements in columns instead of the rejection of the entire rows in the block-columns.

The Table I presents a comparison of the complete factorization time and speed up, when the number of threads increases, for the proposed block Cholesky solver and PARDISO from Intel MKL 11.3. As it turned out, on computer A with AMD Opteron processor the PARDISO solver limits the number of threads to eight. The solver proposed by us demonstrates a steady acceleration of up to 15 threads, and only when the number of threads is equal to the number of processor cores, there is a slight decrease in performance. As a result, we achieve on 15 threads a slightly shorter total factorization time than PARDISO. In general, both solvers show close results, and this fact allows to conclude, that the developed incomplete block Cholesky solver can be used successfully for the solution of real-life large problems.

TABLE I
COMPARISON OF THE COMPLETE FACTORIZATION TIME AND SPEED UP. PROBLEM 2, COMPUTER A.

Nos of threads	Duration of the block Cholesky, factorization s	Duration of the PARDISO, factorization, s	Block Cholesky, $Sp = T_1/T_p$	PARDISO, $Sp = T_1/T_p$
1	10 211	8 282	1	1
2	5 448	6 539	1.87	1.27
4	3 151	3 114	3.24	2.66
8	1 984	1 898	5.15	4.36
12	1 713	–	5.96	–
14	1 635	–	6.25	–
15	1 629	–	6.27	–
16	1 704	–	6.00	–

The sums of weights for each thread for both: first and second parallelization levels (see section III.*A*) are presented in Table II. We take a parameter $tol_1 = 0.05$.

Even on 16 threads, the proposed scheduling algorithms 1 and 2 ensure an acceptable balance of computational load between threads.

The duration of factorization using a conventional incomplete Cholesky solver [8] on eight threads (in such case this method demonstrates the best performance) is 42 638 s, and we believe that such a long time is unacceptable.

Table III presents the duration of the main stages of BSPPCG method. We use the following abbreviation: *Reort. time* – time of the total reorthogonalization when the columns of matrix $\mathbf{Q}_k$ become almost linearly dependent; *Orth. against conv. modes* – orthogonalization of columns in the subblocks **X** and **P** against all converged modes (see section II.*E*); *Evaluation of residuals* – see section II.*B*; *Subsp. project*. – evaluation of the subspace projection matrices **m** and **k** (section II.*C*); *Prolong*. – prolongation procedure (section II.*D*). All these stages are parallelized.

Table IV shows the comparison of computing time required for the extraction of 100 eigenpairs for different methods. The BSPPCG method runs in the core memory, requires 35.3 GB RAM and produces 61 iterations and 17 total reorthogonalization when $\psi = 10^{-50}$, $\psi_1 = 10^{-13}$. When

we assign $\psi = 10^{-8}$, $\psi_1 = 10^{-8}$, the amount of required RAM is 16.4 GB, the ability of preconditioning of accelerating of the convergence is worse than in the previous case, and 142 iterations and 50 total reorthogonalization is required.

TABLE II
THE DISTRIBUTION OF COMPUTATIONAL WORK AMONG THREADS. PROBLEM 2, COMPUTER A.

Thread number	First level of parallelization	Second level of parallelization
1	81 957 982	200 429 150
2	81 992 420	200 070 240
3	81 051 034	200 507 916
4	78 914 973	200 263 442
5	81 610 575	200 230 611
6	81 862 179	200 518 532
7	81 724 138	200 079 318
8	81 287 287	199 915 628
9	79 117 086	200 671 820
10	79 027 153	200 172 374
11	80 780 854	201 390 756
12	81 153 035	200 164 913
13	82 434 985	200 463 860
14	79 223 228	200 221 723
15	81 595 566	200 978 565
16	83 031 954	201 070 606

TABLE III
COMPUTING TIME OF SEVERAL PHASES VIA A NUMBER OF THREADS. PROBLEM 2, COMPUTER A.

np	Reort time, s	Orth. against conv modes, s	Evaluation of residuals, s	Subsp. project. s	Prolong., s	Total, s	S_{np}
1	5241	6420	40039	8515	319	62711	1
2	2384	3197	21861	5303	183	34323	1.82
4	1628	2298	12039	3621	122	20814	3.01
8	1068	2315	9031	2809	104	16390	3.82
16	620	2183	8229	2415	103	14552	4.31

The block Lanczos method with the spectral transformations (SBLANC, [7]) runs in two modes. When the entire amount of the core memory is accessible (100 % RAM), solver PARFES [11], [12] – one from the fastest sparse direct solvers for multicore computers on today – works in a core mode, and the lower triangular factorized matrix **L**, having the size 36.53 GB, is located in RAM. Therefore, the forward and back substitutions, performed at each step of the Lanczos method, run extremely fast. In this case, we obtain practically the same durations for both methods.

When we allow using only 50% of RAM for the Lanczos method, PARFES runs in the out of core mode (OOC), and the matrix **L** is written block-by-block on disk. The forward and back substitutions run very slowly, and the solution time for the Lanczos method increases more than two times.

TABLE IV
COMPARISON OF COMPUTATION TIME FOR DIFFERENT METHODS. PROBLEM 2, COMPUTER A.

Method	Total time, s
BSPPCG (core mode, $\psi = 10^{-50}$, $\psi_1 = 10^{-13}$)	14 552
BSPPCG (core mode, $\psi = 10^{-8}$, $\psi_1 = 10^{-8}$)	16 334
SBLANC (100% RAM)	14 096
SBLANC (50% RAM)	34 660

C. Problem 3.

Figure 6 presents the design model of the multi-storey building, having quite a different topology and construction than the previous problem.

Fig. 6 Multi-storey building, having three towers

The dimension of the problem is $N = 4\ 262\ 958$ equations, number of required eigenpairs – $n = 100$, dimension of the block $m = 32$. The parameters of preconditioning are: $\psi = 0$, $\psi_1 = 10^{-13}$, $tol = 10^{-3}$. We use a usual desktop – computer B.

This problem is very hard for stable convergence due to the presence of a large number of almost multiple frequencies, produced by similar towers, resting on the same stylobate, and we were forced to apply a shift technique. Table V shows that the given approach proved to be stable when the shift correction procedure (algorithm 5) performs at least two iterations.

As a result, we obtain about the same time of solving the problem by both methods: BSPPCG and SBLANC, but BSPPCG method works in RAM, and when Lanczos method is applied, PARFES uses the out of core (OOC) mode.

TABLE V
COMPARISON OF COMPUTATION TIME FOR DIFFERENT METHODS. PROBLEM 3, COMPUTER B.

Method	Number of shift's corrections, *S*	Number of iterations	Number of total reorthogonalization	Total time, s
BSPPCG	0	Lock of convergence after 42 eigenpairs are converged		
BSPPCG	1	Lock of convergence after 53 eigenpairs are converged		
BSPPCG	2	70	19	16 901
SBLANC	—	—	—	16 568

VII. CONCLUSION

The proposed BSPPCG method can compete with the block Lanczos method widely used in FEA software on shared memory multi-core computers. The presented approach, based on the PCG method, uses the block incomplete Cholesky factorization approach which allows to keep a very small value of the rejection parameter ψ and produces a lower triangular matrix $\mathbf{H}$, $\mathbf{B} = \mathbf{H}\mathbf{H}^T$, which possesses a high ability to accelerate the convergence. The use of the iteration technique in a block of fixed dimension, the shift technique and the parallel algorithm for the total re-orthogonalization, when a linear dependence between the columns of the projection matrix $\mathbf{Q}_k$ was detected, ensure the high computational stability of the proposed approach.

ACKNOWLEDGMENT

This work was supported by SCAD Soft IT company.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J-Y. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Comput. Meth. Appl. Mech. Eng.*, 184, pp. 501–520, 2000, https://doi.org/10.1016/S0045-7825(99)00242-X.

[2] V. E. Bulgakov, M. E. Belyi and K. M. Mathisen, "Multilevel aggregation method for solving large-scale generalized eigenvalue problems in structural dynamics," *Int. J. Numer. Methods Eng.,* vol. 40. pp. 453 – 471, 1997, http://DOI: 10.1002/(SICI)1097-0207(19970215)40:33.0.CO;2-2.

[3] Y. T. Feng and D. R. J. Owen, "Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems," *Int. J. Numer. Methods Eng.,* vol. 39. pp. 2209 – 2229, 1996, http://DOI: 10.1002/(SICI)1097-0207(19960715)39:13<2209::AID-NME951>3.0.CO;2-R.

[4] A. George, J. W. H. Liu, *Computer solution of sparse positive definite systems*. New Jersey : Prentice-Hall, Inc. Englewood Cliffs, 1981.

[5] S. Yu. Fialko, "Natural vibrations of complex bodies," *Int. Applied Mechanics,* vol. 40, no. 1, pp. 83 – 90, 2004, http://DOI: 10.1023/B:INAM.0000023814.13805.34.

[6] S. Fialko, "Aggregation Multilevel Iterative Solver for Analysis of Large-Scale Finite Element Problems of Structural Mechanics: Linear Statics and Natural Vibrations", in *PPAM 2001*, R. Wyrzykowski et al. (Eds.), *LNCS* 2328, Springer-Verlag Berlin Heidelberg, 2002, pp. 663–670, http://DOI: 10.1007/1-4020-5370-3_41.

[7] S. Yu. Fialko, E. Z. Kriksunov and V. S. Karpilovskyy, "A block Lanczos method with spectral transformations for natural vibrations and seismic analysis of large structures in SCAD software," in *Proc. CMM-2003 – Computer Methods in Mechanics*, Gliwice, Poland, 2003, pp. 129 —130.

[8] S. Yu. Fialko, "Iterative methods for solving large-scale problems of structural mechanics using multi-core computers," Archives of Civil and Mechanical Engineering, vol. 14, pp. 190 – 203, 2014, http:// doi:10.1016/j.acme.2013.05.009.

[9] S. Yu. Fialko, F. Żegleń, "Block Preconditioned Conjugate Gradient Method for Extraction of Natural Vibration Frequencies in Structural Analysis", Proceedings of the FedCSIS. Łódź, 2015. IEEE Xplore Digital Library, pp. 655 – 662. DOI: 10.15439/2015F87. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7321505&tag=1 .

[10] S. Yu. Fialko, F. Żegleń, "Block subspace projection PCG method for solution of natural vibration problem in structural analysis.", Proceedings of the Federated Conference on Computer Science and Information Systems pp. 669–672. DOI: 10.15439/2016F88. URL: http://annals-csis.org/Volume_8/pliks/88.pdf .

[11] S. Yu. Fialko, "PARFES: A method for solving finite element linear equations on multi-core computers," Advances in Engineering software, vol. 40, no. 12, pp. 1256-1265, 2010, http:// doi:10.1016/j.advengsoft.2010.09.002.

[12] S. Yu. Fialko, "Parallel direct solver for solving systems of linear equations resulting from finite element method on multi-core desktops and workstations", Computers and Mathematics with Applications 70, pp. 2968–2987, 2015 doi:10.1016/j.camwa.2015.10.009

[13] G. Gambolati, G. Pini and F. Sartoretto, "An improved iterative optimization technique for the leftmost eigenpairs of large symmetric matrices," *J. Comp. Phys.,* no 74, pp. 41 – 60, 1988, http://doi: 10.1016/0021-9991(88)90067-8.

[14] C. K. Gan, P. D. Haynes and M. C. Payne, "Preconditioned conjugate gradient method for sparse generalized eigenvalue problem in electronic structure calculations," *Computer Physics Communications*, vol 134, nr. 1, pp. 33 – 40, 2001, http://DOI: 10.1016/S0010-4655(00)00188-0.

[15] V. Hernbadez, J. E. Roman, A. Tomas and V. Vidal, "A survey a software for sparse eigenvalue problems," Universitat Politecnica De Valencia, SLEPs technical report STR-6, 2009.

[16] G. Karypis and V. Kumar, "METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System,". Technical report, Department of Computer Science, University of Minnesota, Minneapolis, 1995.

[17] A. V. Knyazev and K. Neymayr, "Efficient solution of symmetric eigenvalue problem using multigrid preconditioners in the locally optimal block conjugate gradient method," *Electronic Transactions on Numerical Analysis*, vol. 15, pp. 38 – 55, 2003. URL: https://eudml.org/doc/123270.

[18] A. V. Knyazev, M. E. Argentati, I. Lashuk, E.E. Ovtchinnikov, "Block Locally Optimal Preconditioned Eigenvalue Solvers (BLOPEX) in HYPRE and PETSC". URL: http://arxiv.org/pdf/0705.2626.pdf.

[19] R. B. Morgan, "Preconditioning eigenvalues and some comparison of solvers," *Journal of computational and applied mathematics*, vol. 123, pp. 101 – 115, 2000, http://doi: 10.1016/S0377-0427(00)00395-2.

[20] M. Papadrakakis, "Solution of partial eigenproblem by iterative methods," *Int. J. Num. Meth Eng*., vol. 20. pp. 2283—2301, 1984, http://DOI: 10.1002/nme.1620201209.

[21] A. V. Perelmuter, S. Yu. Fialko, "Problems of computational mechanics relate to finite-element analysis of structural constructions," *International Journal for Computational Civil and Structural Engineering*, vol. 1, no 2, 2005, pp. 72 – 86.

[22] Y. Saad, *Numerical methods for large eigenvalue problems, Revised edition, Classics in applied mathematics*. SIAM, 2011, http://dx.-doi.org/10.1137/1.9781611970739.

[23] O. Schenk, K. Gartner, "Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems," *Parallel Computing*, 28, pp. 187–197, 2002, https://doi.org/10.1016/S0167-8191(01)00135-1.

[24] Thread Local Storage. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ms686749(v=vs.85).aspx (Last access: 18.04.2017).

[25] S. Tomov, J. Langou, A. Canning, Lin-Wang Wang, J. Dongarra, "Conjugate-gradient eigenvalue solver in computing electronic properties of nanostructure architecture," Int. J. Computational Science and Engineering, vol. 2, nr. 3-4, pp. 205 – 212, 2006. https://doi.org/10.1504/IJCSE.2006.012774.

[26] doclib/iss/2013/mkl/mklman/index.htm (Last access: 16.04.2015).