ARCHIVES OF CIVIL AND MECHANICAL ENGINEERING 14 (2014) 190–203

Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/acme

Original Research Article

Iterative methods for solving large-scale problems of structural mechanics using multi-core computers

S.Yu. Fialko

Department of Physics, Mathematics and Applied Computer Science, Cracow University of Technology, Kraków 31-155, Poland

ARTICLE INFO

Article history:
Received 18 January 2013
Accepted 30 May 2013
Available online 6 June 2013

Keywords:
Finite element method
Preconditioned conjugate gradient method
Sparse matrices
Multi-core computers
Multistory buildings

ABSTRACT

The paper studies the conjugate gradient method for solving systems of linear algebraic equations with symmetric sparse matrices that arise when the finite-element method is applied to the problems of structural mechanics. The main focus is on designing effective preconditioning and parallelizing the method for multi-core desktop computers. Preconditioning is based on the incomplete Cholesky "by value" factorization method and implemented based on the technique of sparse matrices, which allows increasing convergence considerably without a significant increase of the computer's resources. Parallelization is implemented for the incomplete factorization as well as for iterative process stages. The method is integrated into the SCAD software package (http://www.scadsoft.com). The paper includes a discussion of the results of calculations done with direct and iterative methods for large-scale design models of tall buildings, originally from the SCAD Soft[1] problem collection.

1. Introduction

Intense development of multistory construction, combined with the rapid development of computation equipment used for strength calculations in small and medium-sized engineering bureaus, created a necessity to improve methods for solving linear algebraic equations (solvers) that arise when the finite element method is applied to problems of structural mechanics. Since the dimension of the design model of a contemporary multistory building is between 800,000 and 5,000,000 equations, a solver for a contemporary FEA software must be highly efficient.

Modern desktop computers are progressively more capable of solving large-scale and complicated engineering problems, eliminating the need for clusters, powerful and expensive workstations, and computer networks. However, they have a limited amount of RAM and small system bus bandwidth. This makes it necessary to develop computational methods that take into consideration the above specifics of computers with such architecture, because methods proven to be effective on distributed-memory computers (clusters or computer networks) are not always effective on desktop computers with SMP (symmetrical multiprocessing) architecture. In other words, the solver must be able to use the disc memory if the dimension of the problem exceeds the RAM capacity, demonstrate high performance while working in the RAM, and speed up reliably as the number of cores (processors) is increased.

E-mail address: sfialko@poczta.onet.pl

[1]SCAD Soft (http://www.scadsoft.com)—IT company, developer of the SCAD FEA software, one of the most popular software used in the CIS countries for structural analysis and design, certified according to the regional norms.

1644-9665/$ - see front matter © 2013 Politechnika Wrocławska. Published by Elsevier Urban & Partner Sp. z o.o. All rights reserved.
http://dx.doi.org/10.1016/j.acme.2013.05.009

Contemporary FEA software most frequently employs the sparse direct solvers for analysis of problems of structural mechanics. The decisive factor here is employing an effective ordering method that significantly decreases the number of non-zero entries during matrix factorization [4], as well as the solver's ability to use the disc memory when solving large-scale problems on computers with small amounts of RAM, and to maintain high performance and speed up when processing data blocks stored in the RAM.

In the latter case, the key issue is extracting rectangular dense submatrices from a sparse matrix and subjecting them to high-performance BLAS level 3 routines [2], particularly matrix multiplication procedures. Furthermore, with the use of direct methods, solution time does not depend on the conditioning number associated with the linear equation set and depends only slightly on the number of right hand parts, if the number of the latter is relatively small. Direct methods also allow detecting the geometric instability of the design model.

Until recently, the multifrontal solver [1,7,14], has been the most widespread direct method. It has all of the above advantages, however, it also contains an excess number of memory–memory and memory–disc–memory data transfer. On multi-core SMP computers, this shortcoming prevents achieving maximum possible performance and speeding up with the increase in the number of processors.

In the recent years, the PARDISO method [22] from the high-performance Intel Math Kernel Library (Intel MKL) [10] has become widespread, demonstrating high performance with one processor and good speed up on desktop multi-core computers. However, the OOC (out of core) mode, which utilizes disc memory, does not work for large-scale problems and demonstrates low performance with small ones [13]. Therefore, in practice, this method can only be used for problems solved in RAM.

The above became a stimulus for developing the PARFES (parallel finite element solver) method, intended for solving finite-element problems with multi-core desktop computers [13]. PARFES demonstrates the performance and speed up that approximate those of PARDISO, but, unlike the latter, includes two disc usage modes, OOC and OOC1. In the OOC mode, the number of I/O operations is minimal, resulting in only a small decrease in performance and speed up compared to CM (core mode—utilizing RAM only). However, if the requirements to RAM are still exceeded in the OOC mode, PARFES switches to the OOC1 mode. In this mode, the number of I/O operations is increased considerably, and the decrease in performance and speed up is also considerable—however, this mode allows solving large-scale problems on computers with small amounts of RAM [17]. The mode is selected automatically.

The common disadvantage of direct methods includes the quadratic dependence of the number of operations on the dimension of the problems. Plus, for problems with the dimension exceeding the RAM capacity, the matrix factorization time and forward-back substitution time are considerably increased.

Iterative methods do not guarantee detecting the geometric instability of the design model; the iterative process is started anew for each right hand side; and when solving poorly conditioned problems, iterative methods result in slower or no convergence.

Design models of tall multistory buildings contain different types of finite elements (FE)—thin-walled plate and shell FE, FE of spatial frame, volumetric FE, specialized FE (elastic supports, rigid links that carry penalty parameters in SCAD, compatible nodes, etc.). This also results in a wide dispersion of stiffness values. In realistic problems, due to complex geometry, mesh generators cannot always provide for the optimal ratios between FE sides and angles for shells and plates. Due to this, design models for multistory buildings usually result in poorly conditioned stiffness matrices and therefore require iterative methods that are stable against ill conditioning [21]. The use of preconditioning is an effective way for overcoming of poor conditioning [6,8].

The main operations of iterative methods are matrix–vector multiplication and solving the system of linear algebraic equations regarding preconditioning. These operations are classified as BLAS level 2 routines. Unlike matrix multiplication used in direct methods, which employs cache reuse (data, once placed in the cache, is read into the processor registers multiple times, from the fast cache rather than the slow RAM), register blocking, vectorization of computation with the use of XMM and YMM registers, and other high-performance techniques, BLAS level 2 routines are carried out at the speed of the slow memory system rather than the fast processor [8]. The use of multithreading in the SMP architectures allows for good speed up of the matrix multiplication routine, since the memory system is not overloaded thanks to the cache reuse. In BLAS level 2 routines, on the other hand, the number of memory–cache–memory transfers is of the same order as the number of arithmetic operations. The memory system is incapable of efficiently serving several processors, and each of them is forced to perform many idle cycles while waiting for the required data to be loaded into the cache from the RAM. Due to this, the performance and speed up of matrix–vector multiplication and solving of linear equation sets regarding preconditioning is far less compared to matrix multiplication (see Fig. 1).

Here, $S_p = T_1/T_p$ is speed up, T_1 is solution time with one processor, T_p is solution time with p processors. The results were obtained on a computer with Intel® Core™2 Quad CPU Q6600 @2.40 GHz processor, RAM DDR2 333.7 MHz, 8 GB,

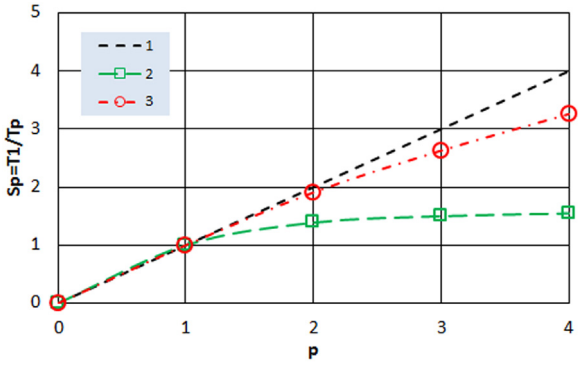

Fig. 1 – Speed up (S_p) with the increase in the number of processors (p). 1—ideal, 2—for dense matrix–vector multiplication algorithm, 3—for dense matrix multiplication algorithm.

chipset Intel P35/G33/G31, OS Windows Vista™ Business (64-bit), Service Pack 2.

Similar results were achieved for the sparse matrix–vector multiplication algorithm in [3,18], where the speed up through multithreaded parallelization was 10–40%. Quite rarely, due to a specific structure of a sparse matrix, twofold speed up is achieved. The main technique for increasing performance is grouping sparse matrix columns (rows) into twos, which allows register blocking. This, however, increases the amount of data, because such joining causes zero elements to be interpreted as non-zero.

The fact that making iterative solvers parallel on multi-core computers is an ongoing issue is confirmed, first of all, by the lack of corresponding parallel procedures in libraries of high performance. For example, the Intel Math Kernel Library (Intel MKL) [10,11] contains no parallel procedures for iterative methods for solving systems of linear algebraic equations with sparse matrices (Sparse Solver Routines section) or for forward-back substitution (Sparse Triangular solvers, Sparse BLAS routines Level 2 section) [11]. This is also the case for the IMSL library [9]. The last realizes of the both libraries: Intel MKL 10.3 and IMSL 7.0 are considered.

As the dimension of the problem increases, solution time for iterative method increases quasi-linearly (the term "quasi-linear" is applied because is impossible to ensure in practice the constant spectral properties of preconditioning with increasing dimension of the problem), and not quadratically, as in the case of direct methods. Furthermore, iterative methods only use the RAM, therefore excluding slow I/O operations.

The objective of this work is developing a parallel finite element iterative solver for systems of linear algebraic equations with sparse symmetric positive definite matrices, oriented at solving structural mechanics problems with multi-core desktop computers. The approach is based on the incomplete Cholesky conjugate gradient "by value" method, implemented in the sparse matrix techniques. The basis for parallelizing in the SMP architecture is the fact that the majority of problems in this class contain a large number of right hand sides—the load cases. Virtually any engineering structure is subjected by action of its own weight (dead load), the operating loads, including permanent and temporary continuous loads (e.g., load of the entire story of a building or part thereof, etc.), the wind load, snow load, etc. The iterative process for each right hand side is processed in a separate thread.

2. Preconditioned conjugate gradient method

Let us consider a system of linear algebraic equations

$$\mathbf{K}\mathbf{x}=\mathbf{b}, \tag{1}$$

where $\mathbf{K}$ is a sparse symmetric positive definite matrix, $\mathbf{x}$, $\mathbf{b}$ are solution and right hand side matrices $N\times nrhs$, where N is the number of equations and *nrhs* is the number of right hand sides.

Preconditioning is used to accelerate convergence. Let $\mathbf{B}$ be a symmetric positive definite matrix with a sparser structure compared to the $\mathbf{K}$ matrix, while $C(\mathbf{B}^{-1}\mathbf{K})<C(\mathbf{K})$, where C is the conditioning number. Then, the following system of equations is solved instead of (1):

$$\mathbf{B}^{-1}\mathbf{K}\mathbf{x}=\mathbf{B}^{-1}\mathbf{b} \tag{2}$$

This system is solved as follows. First, in order to approximate solution $\mathbf{x}_k^s$ at iteration step k (where s is the right hand side number), residual vector is calculated as $\mathbf{r}_k^s=\mathbf{b}^s-\mathbf{K}\mathbf{x}_k^s$; then, the residual vector of problem (2) is calculated as $\mathbf{z}_k^s=\mathbf{B}^{-1}(\mathbf{b}^s-\mathbf{K}\mathbf{x}_k^s)=\mathbf{B}^{-1}\mathbf{r}_k^s$ by solving the system of equations related to preconditioning:

$$\mathbf{B}\mathbf{z}_k^s=\mathbf{r}_k^s \tag{3}$$

The system of Eq. (3) should be solved much faster than (1). The preconditioned conjugate gradient method algorithm for several right hand parts is shown below. For each right hand side, $s=1, 2,\ldots, nrhs$, the solution vector is calculated according to [6,15,20].

The algorithm of preconditioned conjugate gradient method for multiple right hand parts (sequential version) is presented below.

Algorithm 1. Preconditioned conjugate gradient method scheme

$s=1$

1. Initial approximation: $k=0$; $\mathbf{x}_0^s=0$; $\mathbf{r}_0^s=\mathbf{b}^s$; $\mathbf{p}_0^s=\mathbf{z}_0^s=\mathbf{B}^{-1}\mathbf{r}_0^s$
2. Line search, adjustment of solution and residual vector:
   $\mathbf{w}_k^s=\mathbf{K}\mathbf{p}_k^s$; $\alpha_k^s=\frac{(\mathbf{r}_k^s)^T\mathbf{z}_k^s}{(\mathbf{p}_k^s)^T\mathbf{w}_k^s}$;
   $\mathbf{x}_{k+1}^s=\mathbf{x}_k^s+\alpha_k^s\mathbf{p}_k^s$; $\mathbf{r}_{k+1}^s=\mathbf{r}_k^s-\alpha_k^s\mathbf{w}_k^s$;
3. Checking convergence:
   if $(\|\mathbf{r}_{k+1}^s\|_2\le tol\cdot\|\mathbf{b}^s\|_2 \wedge \|\mathbf{r}_{k+1}^s\|_{\inf}\le tol\cdot\|\mathbf{b}^s\|_{\inf})$ $s++$; *if* $(s>nrhs)$ *stop*; *else goto* 1
4. Finding the new conjugated direction:
   $\mathbf{z}_{k+1}^s=\mathbf{B}^{-1}\mathbf{r}_{k+1}^s$; $\beta_k^s=\frac{(\mathbf{r}_{k+1}^s)^T\mathbf{z}_{k+1}^s}{(\mathbf{r}_k^s)^T\mathbf{z}_k^s}$; $\mathbf{p}_{k+1}^s=\mathbf{z}_{k+1}^s+\beta_k^s\mathbf{p}_k^s$; $k++$;
5. *goto* 2

Here, $\mathbf{p}_k^s$ is the vector of conjugated direction for right hand side s. During checking of convergence, the magnitude of residual vector is evaluated based on two norms, which allows to assume the *tol* to be within 10^{-3}–10^{-4}.

As of today, the issues of theoretical convergence of preconditioned iterative methods for task class in question have been studied thoroughly. Nevertheless, the author found it necessary to give some considerations which, though not open new fundamental results in the theory of convergence of iterative methods, but may be useful in interpreting them.

For iteration step $k+1$, we have

$$\mathbf{B}\mathbf{z}_{k+1}=\mathbf{r}_{k+1}=\mathbf{b}-\mathbf{K}\mathbf{x}_{k+1}=\mathbf{b}-\mathbf{K}(\mathbf{x}_k+\alpha_k\mathbf{p}_k) \tag{4}$$

Here, the upper index s is omitted for the sake of visual simplicity, because the number of the right hand part is not relevant for the purposes at hand. To simplify the presentation, let us consider the steepest descent method with preconditioning (omitting the conjugation of direction vectors, see p. 4 of the above algorithm). Then $\mathbf{p}_k=\mathbf{z}_{k+1}$ and (4) takes the following form:

$$\mathbf{B}\mathbf{z}_{k+1}=\mathbf{b}-\mathbf{K}(\mathbf{x}_k+\alpha_k\mathbf{z}_k)=\mathbf{b}-\mathbf{K}\mathbf{x}_k-\alpha_k\mathbf{K}\mathbf{z}_k=\mathbf{r}_k-\alpha_k\mathbf{K}\mathbf{z}_k,$$

or

$$\mathbf{z}_{k+1}=\mathbf{B}^{-1}\mathbf{r}_k-\alpha_k\mathbf{B}^{-1}\mathbf{K}\mathbf{z}_k=\mathbf{z}_k-\alpha_k\mathbf{B}^{-1}\mathbf{K}\mathbf{z}_k.$$

This results in the following recurring relation:

$$\mathbf{z}_{k+1}=(\mathbf{I}-\alpha_k\mathbf{B}^{-1}\mathbf{K})\mathbf{z}_k \tag{5}$$

Because $\mathbf{r}_{k+1}=\mathbf{B}^{-1}\mathbf{z}_{k+1}$, then $\|\mathbf{r}_{k+1}\|_2=\|\mathbf{B}^{-1}\mathbf{z}_{k+1}\|_2\le\|\mathbf{B}^{-1}\|_2\cdot\|\mathbf{z}_{k+1}\|_2$.

Since $\|\mathbf{B}^{-1}\|_2$ is a positive value, then $\|\mathbf{r}_k\|_2\to 0\big|_{k\to\infty}$ when and only when $\|\mathbf{z}_k\|_2\to 0\big|_{k\to\infty}$; therefore, the convergence of the iterative process follows from the latter condition.

Let us expand $\mathbf{z}_k$, $\mathbf{z}_{k+1}$ in the eigenvectors of problem

$$\mathbf{K}\varphi_i=\lambda_i\mathbf{B}\varphi_i,\quad i=1,\ 2,\ \dots,N, \tag{6}$$

of dimension N, and substitute this decomposition into (5).

$$\sum_{i=1}^{N}\gamma_i^{k+1}\varphi_i=\sum_{i=1}^{N}(\mathbf{I}-\alpha_k\mathbf{B}^{-1}\mathbf{K})\gamma_i^k\varphi_i=\sum_{i=1}^{N}\left(\gamma_i^k\varphi_i-\alpha_k\gamma_i^k\underbrace{\mathbf{B}^{-1}\mathbf{K}\varphi_i}_{\lambda_i\varphi_i}\right)$$

By multiplying the left hand and right hand side of the equation by φ_j^T and taking into consideration the eigenvector orthogonality condition and the normalizing $\varphi_j^T\varphi_i=\delta_{ij}$, where δ_{ij} is Kronecker's delta, we achieve $\gamma_i^{k+1}=(1-\alpha_k\lambda_i)\gamma_i^k$, $i=1,2,\dots,N$. To simplify the presentation, let us assume that the convergence acceleration parameter α does not depend on the iteration step k. The resulting expression shows that

$$\gamma_i^k=(1-\alpha\lambda_i)^k\gamma_i^0,\quad i=1,2,\dots,N \tag{7}$$

In order for the iterative process to converge, the following is necessary and sufficient for all modes:

$$\left|1-\alpha\lambda_i\right|<1,\quad i=1,2,\dots,N \tag{8}$$

Solving inequality (8) results in the following condition:

$$0<\alpha\lambda_i<2,\quad i=1,2,\dots,N \tag{9}$$

The line search routine (p. 2) shows that

$$\alpha_k=\frac{\mathbf{r}_k^T\mathbf{z}_k}{\mathbf{z}_k^T\mathbf{K}\mathbf{z}_k}=\frac{(\mathbf{B}\mathbf{z}_k)^T\mathbf{z}_k}{\mathbf{z}_k^T\mathbf{K}\mathbf{z}_k}=\frac{\mathbf{z}_k^T\mathbf{B}\mathbf{z}_k}{\mathbf{z}_k^T\mathbf{K}\mathbf{z}_k}, \tag{10}$$

because matrix $\mathbf{B}$ is symmetric. Parameter α_k is Rayleigh's quotient for eigenproblem $\mathbf{B}\varphi_i-(1/\lambda_i)\mathbf{K}\varphi_i=0$, which follows from (6). Thus,

$$\frac{1}{\lambda_N}\le\alpha\le\frac{1}{\lambda_1},\quad \lambda_1\le\lambda_2\le\cdots\le\lambda_N \tag{11}$$

Index k is omitted here. In order to satisfy inequality (9), all eigenvalues of (6) must be positive, which means that matrix $\mathbf{B}$ must be a positive definite matrix. If this is not so, condition (9) will be unsatisfied for at least one of values i, and the convergence of the iterative process will not be guaranteed. Additionally, if $\mathbf{B}\to\mathbf{K}$, then $\alpha\to 1$.

By substituting the minimal value of α from (11) into (8), we achieve

$$\gamma_i^k=\left(1-\frac{\lambda_i}{\lambda_N}\right)^k\gamma_i^0,\quad i=1,2,\dots,N,\quad 0<\lambda_1\le\lambda_2\le\cdots\le\lambda_N$$

Then, convergence is achieved in one iteration for mode $i=N$, and the worst convergence is for $i=1$. This result confirms the known fact that the problem of convergence of gradient methods lies in slow convergence of the lower modes. The closer the $\mathbf{B}$ is to $\mathbf{K}$, the closer λ_1 is to λ_N and $1-\lambda_i/\lambda_N$ to zero, and the faster is the convergence.

In practice, preconditioning is built in a variety of ways, some of which work better for some problems, and some for others. It is impossible to determine in advance which preconditioning will be best for a specific problem.

This paper uses incomplete Cholesky factorization [23] implemented in the sparse matrix technique [16] to create a preconditioning $\mathbf{B}$. In this case, $\mathbf{B}=\mathbf{H}\cdot\mathbf{H}^T$, where $\mathbf{H}$ is incomplete Cholesky factor. Small elements are dropped in the incomplete factorization process:

$$H_{ij}<\psi H_{ii}H_{jj}, \tag{12}$$

where $0\le\psi\le 1$, $i,j\in[1,N]$. Here ψ is the rejection parameter. When element H_{ij} is rejected, the error matrix is formed, consisting of four non-zero elements:

$$\mathbf{E}^{(k)}=\begin{pmatrix} D_{ii} & \vdots & -H_{ij}\\ \cdots & \cdots & \cdots\\ -H_{ij} & \vdots & D_{jj}\end{pmatrix},\quad D_{ii}=\sqrt{\frac{H_{ii}}{H_{jj}}}\cdot\left|H_{ij}\right|,\quad D_{jj}=\sqrt{\frac{H_{jj}}{H_{ii}}}\cdot|H_{ij}|$$

When the first element is rejected, incomplete Cholesky factor is presented as $\mathbf{H}_j^{(1)}=\mathbf{L}+\mathbf{E}^{(1)}$, where $\mathbf{L}$ is the complete Cholesky factor, and the upper index indicates the sequential number of rejection. Since matrix $\mathbf{L}$ is positive definite and matrix $\mathbf{E}^{(1)}$ is semi-definite, matrix $\mathbf{H}^{(1)}$ is positive definite. With the second rejection, $\mathbf{H}_j^{(2)}=\mathbf{H}_j^{(1)}+\mathbf{E}^{(2)}$. Matrix $\mathbf{H}^{(2)}$ is again positive definite, being the sum of a positive definite matrix and a semi-definite matrix, and so on. Thus, this approach, suggested in [12] and used in [16,23] and other works, provides for positive definiteness of incomplete Cholesky factor $\mathbf{H}$ and the resulting positive definiteness of matrix $\mathbf{B}$, which, in turn, is the condition necessary for the iterative method convergence [6,23].

The closer ψ is to zero, the less iteration are required to arrive at the solution with the desired precision, yet the larger amount of RAM is necessary to store the incomplete $\mathbf{H}$ factor, and the more time will be used for forward-back substitutions during iterations. In order to allow for the smallest possible value of ψ, we use the technique of sparse matrices.

First of all, the nodes of the original finite-element model are ordered, thus considerably decreasing the amount of non-zero entries in the factorized matrix. This also decreases the number of elements rejected in the process of incomplete factorization, so the average expectation should be that this approach provides for a smaller error in the incomplete factor $\mathbf{H}$, compared to lack of ordering (see Example 1 in Section 5).

Multiple tests have established that the best results are obtained when the MMD minimal degree algorithm [5] is used for ordering, because the matrix density only increases at the end of its decomposition, and rejection of small elements at the beginning of the matrix has a bigger impact on the values of retained elements located at the end of the matrix, and a smaller impact on the values of retained elements in the middle of the matrix, therefore limiting the accumulation of rejection-related errors. Other popular ordering methods (nested dissection method, METIS, factor-tree method, etc. [4]) do not have this property and normally result in inferior preconditioning properties.

We shall refer to this method as PSICCG—parallel sparse incomplete Cholesky conjugate gradient.

3. Parallel algorithm for incomplete Cholesky factorization in the technique of sparse matrices

The non-zero structure of the sparse matrix is represented by the *Space* and *ind* arrays that contain non-zero off-diagonal elements located in columns, and row indices i, respectively.

Diagonal elements are stored in the separate *Diag* array. The *Pos*[*j*] array indicates the position of the first non-zero element of column *j* in the *Space* array and its row index in the *ind* array [4]. This sparse matrix storage method is similar to the Compressed Column Storage (CCS) method, which is widely used in high-performance library procedures [9,10]. The simple example, presented below, illustrates applied here sparse matrix storage format.

$$\begin{pmatrix} a_{11} & & & & \\ 0 & a_{22} & & & \\ 0 & 0 & a_{33} & & \\ a_{41} & a_{42} & 0 & a_{44} & \\ 0 & a_{52} & 0 & a_{54} & a_{55} \end{pmatrix} \quad \begin{cases} pos & : 0\ 1\ 2\ 3\ 4 \\ Space & : a_{41}\ a_{42}\ a_{52}\ a_{54} \\ ind & : 4\ 4\ 5\ 5 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots & \\ j & : 1\ 2\ 3\ 4\ 5 \\ Pos & : 0\ 1\ 3\ 3\ 4 \\ Diag & : a_{11}\ a_{22}\ a_{33}\ a_{44}\ a_{55} \end{cases}$$

where *pos* is an index of the current off-diagonal non-zero entry in array *Space* and their first subscript *i* in array *ind*. The arrays *Space* and *ind* have the same number of elements. Index *j* denotes the column number. The a_{41} is a first non-zero entry in column $j=1$ and is located in a zero position of array *Space*, therefore *Pos*[1]=0. Element a_{42} is a first non-zero entry of column $j=2$ and corresponds to first position of array *Space*, hence *Pos*[2]=1. And so on.

The above method is used to store both the original matrix **K** and the incomplete Cholesky factor. The algorithm for incomplete factorization is given below.

Algorithm 2. Column incomplete Cholesky factorization

$\mathbf{v}_{ip,i} \leftarrow 0$, $ip=0, 1,\ldots, np-1$, $i=1,\ldots,N$

1. **do** $j=1, N$
2. $\quad \mathbf{v}_{0,i} \leftarrow \mathbf{K}_{ij}$, $i \epsilon L_j$.
3. $\quad$ *parallel for* $k \epsilon List_j$
   $\quad\quad \mathbf{v}_{ip,i} = \mathbf{v}_{ip,i} - \mathbf{H}_{i,k}\ \mathbf{H}_{j,k}$, $i \epsilon L_k$
   end parallel for
4. *if*($\mathbf{H}_{j,j} <= 0$) stop;
5. **do** $ip=1, np-1$
   $\quad \mathbf{v}_{0,i} += \mathbf{v}_{ip,i}$, $i \epsilon L_j$
   end do
6. *if*$((\mathbf{v}_{0,i})^2 < \psi\ \mathbf{H}_{i,i}\ \mathbf{H}_{j,j})$, $i \epsilon L_j$
   $\quad \mathbf{H}_{i,i} \leftarrow \sqrt{\frac{\mathbf{H}_{i,i}}{\mathbf{H}_{j,j}}} \left|\mathbf{H}_{i,j}\right|$, $\mathbf{H}_{j,j} \leftarrow \sqrt{\frac{\mathbf{H}_{j,j}}{\mathbf{H}_{i,i}}} \cdot \left|\mathbf{H}_{i,j}\right|$, $\mathbf{v}_{0,i} = 0$
   else
   $\quad L_j \leftarrow \mathbf{v}_{0,i} / \sqrt{\mathbf{H}_{j,j}}$, $List_i \leftarrow j$, $\mathbf{v}_{0,i} \leftarrow 0$,
7. **end do**

Here *ip* and *np* are the thread number and the number of threads, respectively. The matrix factorization is done column by column (item 1).

Non-zero elements of column *j* of the initial matrix **K** are copied into vector $\mathbf{v}_{0,i}$ (item 2). The $i \epsilon L_j$ expression means that index *i* belongs to the non-zero structure of column *j* of the factorized matrix **H**.

Then column *j* is corrected by the *k* columns located at left (item 3). The only participating columns are those for which element $\mathbf{H}_{j,k} \neq 0$ ($k \epsilon List_j$, where $List_j$ is the list of columns that are located to the left of column *j* and correct it). Seeing as over 90% of operations are performed at this stage, it should be the first to be parallelized. To this end, the loop is parallelized by index *k*, based on OpenMP. To avoid the situation where several threads modify the same element of vector $\mathbf{v}_i$ at the same time, each *ip* thread writes into its own copy of vector $\mathbf{v}_{ip,i}$. The index of row *i* belongs to the non-zero structure of column *k* of matrix **H** ($i \epsilon L_k$). This means that only non-zero entries of column *k* are used.

After the end of the *parallel for* loop, the value of the diagonal element $\mathbf{H}_{j,j}$ is checked (item 4). If $\mathbf{H}_{j,j}=0$, matrix **K** is singular. This indicates geometric instability of the design model. Unlike complete Cholesky decomposition, incomplete decomposition does not always identify such errors of the design model, but with $\psi < 10^{-9}$ such errors are often identifiable. A negative value of the diagonal element indicates a program error.

The corrections for column *j* in vector $\mathbf{v}_{0,i}$ are summed up (item 5). The condition $i \epsilon L_j$ means that the loop only uses the index *i* values that correspond to non-zero entries of column *j*.

The elements of column *j* are analyzed (12), and a decision is made to reject or retain the current element (item 6). If the element is rejected, the appropriate diagonal elements of matrix **H** are corrected; if the current element is retained, it is added to the non-zero structure of column *j*, and index *j* is added to the list of columns, which are located at right from column *j*, and will be corrected by column *j* at the subsequent factorization steps. If the RAM capacity exhausts when the current element is added to the non-zero structure of column *j*, the algorithm multiplies ψ by ten and returns to start.

After incomplete factorization is finished, secondary rejection of 'small' non-diagonal entries of the **H** matrix takes place [23]. 'Small' elements are $\mathbf{H}_{ij}^2 < \psi_1 \mathbf{H}_{ii} \mathbf{H}_{jj}$, where $0 \le \psi \le \psi_1 \le 1$. Therefore, the ψ value for incomplete factorization should be as small as possible, allowing for better approximation of 'large' coefficients of the **H** matrix ($\mathbf{H}_{ij}^2 \ge \psi_1 \mathbf{H}_{ii} \mathbf{H}_{jj}$) to the values of appropriate coefficients of the **L** matrix—the lower triangular matrix of complete Cholesky decomposition. Secondary rejection and data compression allows freeing up a part of RAM and speeding up forward-back substitutions at the iteration stage, with only a small lowering of the ability of preconditioning to speed up convergence.

4. Parallel algorithm at the iteration stage

If only one right hand side is present, only the sparse matrix–vector multiplication algorithm is parallelized, leading to a slight increase in computation. However, most real problems of structural mechanics contain 5–50, or even more right hand sides. This is the basis for parallelizing this method. The algorithm for parallelizing the iteration stage is given below.

Algorithm 3. Parallel execution of the iteration process in the conjugate gradient method

1. Preparation of queue of tasks Q for each right hand side—Q: {1, 2, …, *nrhs*}
2. *#pragma omp parallel*
3. $\quad$ *while*(Q *is not empty*)
4. $\quad\quad$ *lock* 1

```
        s←Q; Q←Q/s;
        prepare b^s
      end lock 1
5.    start iteration process for sth right hand side
6.    lock 2
        put x^s as a solution of sth right hand side
      end lock 2
7.  end while
8.  end of parallel region
```

At first, the task queue Q is created, where each right hand side $s\epsilon[1,nrhs]$ has a corresponding task—finding solution vector $\mathbf{x}^s$ for the given load $\mathbf{b}^s$, using Algorithm 1 (item 1).

Then, in the parallel region, each thread executes the loop *while* until all tasks in the Q queue have been executed. In the critical section 1 (item 4), the nearest task s is popped from the Q queue, and deleted from it (Q←Q/s). Then the right hand side vector $\mathbf{b}^s$ is taken.

For the given right hand side s, the iteration procedure represented by Algorithm 1 is initiated (item 5).

In the critical section 2, the solution $\mathbf{x}^s$ is stored in the data structure that stores solution vectors.

The use of high-performance techniques will accelerate BLAS level 2 algorithms until the memory system bandwidth is exhausted. For the same reason, using naïve code for low-performance procedures with multithread parallelizing on desktop computers is often as efficient as using high-performance library procedures. Based on this, when designing the solver, we used our own procedures for sparse matrix–vector multiplication and forward-back substitution, without using similar procedures from high-performance libraries.

5. Computational results

Let us consider examples of real problems from the SCAD Soft collection. The original problem names given by the authors of appropriate projects were preserved. The research was done using the following hardware:

A. Laptop, DELL XPS L502X, 4-core Intel® Core™ i7-27600QM CPU 2.4/3.4 GHz processor, RAM DDR3 8 GB, OS Windows 7 (64 bit) Professional, SP1.
B. Desktop computer, AMD Phenom™ II x4 995 3.2 GHz processor, RAM DDR3, 16 GB, OS Windows Vista™ Business (64-bit), SP 2.
C. Workstation, DELL, two 6-core Intel Xeon X5660 @ 2.8/3.2 GHz processors, RAM DDR3 24 GB, OS Windows 7 (64 bit) Professional, SP1.

Computer A has a fast 4-core processor, but restricted amount of RAM. Their processor supports AVX (advanced vector extension), what allows to essentially enlarge of performance of sparse direct methods. In spite of that for large tasks the direct methods will be run in out-of-core mode that would slow down the solution of the problem. In this situation, the iterative methods operating in RAM can be more preferable.

Computer B is a typical desktop, which has 4-core processor AMD Phenom™ and 16 GB RAM. For problems about 1,500,000–2,000,000 equations the sparse direct solvers operates in RAM. Therefore, this computer is selected to compare the effectiveness of sparse direct and iterative methods.

Computer C has the essentially greater number of cores than previous computers, and 24 GB of core memory.

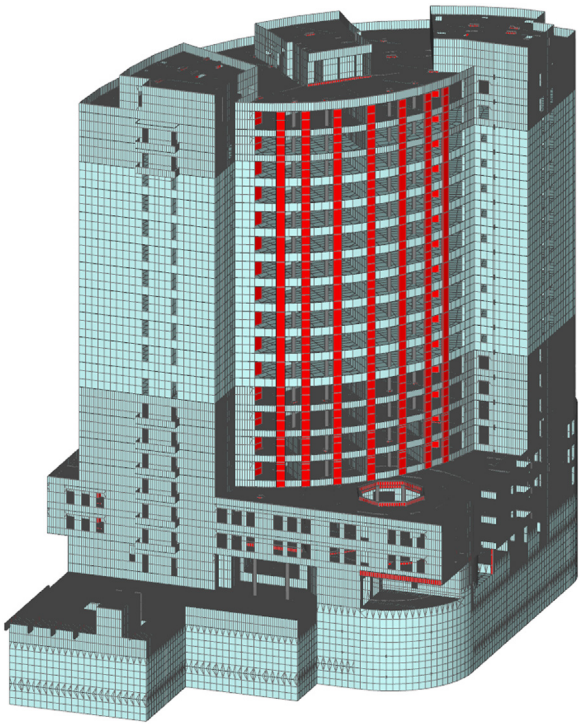

Fig. 2 – Design model of residential block, Raketnyj bulvar project (2,002,848 equations).

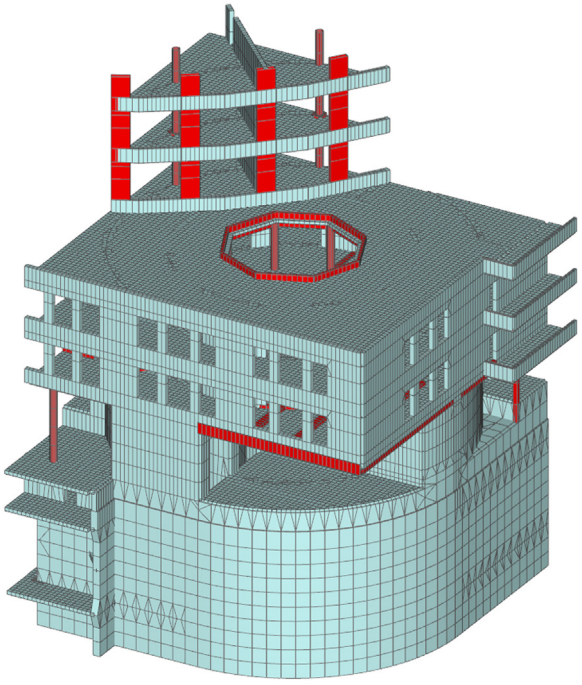

Fig. 3 – Fragment of design model, Raketnyj bulvar project.

Table 1 – Raketnyj bulvar task parameters, solved using the PSICCG method on a computer A and different ordering methods.

Ordering method	Non-zero entries in matrix **L**	Non-zero entries in matrix **H**	No. of rejected elements	Total iterations	Solution time (s)
RCM [4]	55,251,566,592	135,865,674	55,115,700,918	1497	576
METIS [19]	1,041,388,044	113,739,804	927,648,240	1035	394
MMD [5]	1,298,457,600	105,853,122	1,192,604,478	981	357

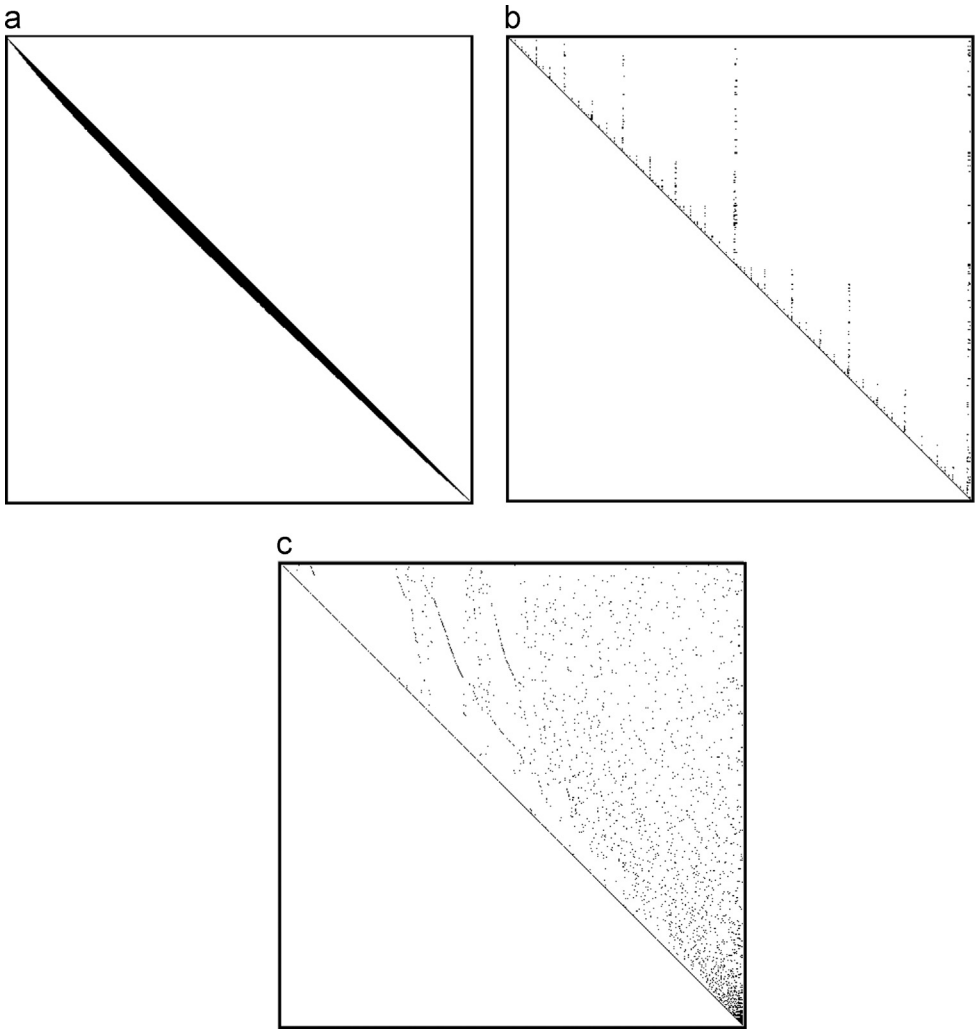

Fig. 4 – Stiffness matrix portrait after ordering with (a) the reverse Cuthill–McKee algorithm (RCM) [4], (b) METIS [19], and (c) MMD minimal degree algorithm [5].

Therefore, it will significantly increase the number of threads and appreciate speed up selected for analysis methods.

The following methods are compared: PSICCG, BSMFM, PARFES and ICCG0 (of which PSICCG and PARFES were mentioned earlier). BSMFM, the block substructure multifrontal method [14], is implemented by the author in the SCAD software. Unlike the known realizations of the classic multifrontal method, such as [1], BSMFM uses effective search of separators (groups of nodes that separate the structure into disconnected substructures if removed) based on the analysis of nodal adjacency graph and modern ordering methods. In tests, BSMFM displayed results that were
not inferior to those of the renowned multifrontal solver ASNSYS v. 11.0, proving that it can be used in the scope of the comparison as a good realization of the multifrontal method [13].

ICCG0 is the incomplete Cholesky conjugate gradient “by position” method, where the incomplete Cholesky factor **H** would only store the elements located in the positions of non-zero entries of the original stiffness matrix. To ensure the positive definiteness of incomplete factor **H**, the following

approach is used [23]. Instead of using the original matrix **K**, matrix $\mathbf{D}+1/(1+\gamma)\mathbf{S}$ is used before factorization, where $\mathbf{D}=Diag(\mathbf{K})$, $\mathbf{S}=\mathbf{K}-\mathbf{D}$, and $\gamma\geq 0$ is a small parameter that decreases the values of off-diagonal elements of the original matrix. Before the first factorization, $\gamma=0$. If incomplete factorization is successful, and $H_{ii}>0$, $i\in[1,N]$, the matrix **H** is positive definite, and the convergence condition for the conjugate gradient method is satisfied. If, however, $H_{ii}<0$, $i\in[1,N]$ at any step of incomplete factorization, the matrix **H** is indefinite. In this case, we increase the value of γ (decreasing the off-diagonal elements) and repeat factorization. This continues until matrix **H** becomes positive definite. After this, ICCG0 uses the same Algorithms 1 and 3 as PSICCG.

For the PSICCG method, the value of ψ is as small as the amount of the computer's RAM allows. We study the speed up of the method with the increase in the number of processors. For PARFES and BSMFM, the maximum possible number of threads is used.

Example 1. A multistory residential block, Raketnyj bulvar project. The problem contains 7 right hand sides and 2,002,848 equations. The design model and its fragment are shown in Figs. 2 and 3. The design model includes triangular and quadrilateral shell finite elements, finite elements of spatial frame and specialized finite elements, such as rigid links used to create rigid connections between nodes. There is significant stiffness dispersion, as the thicknesses of shell elements modeling floor structures and walls vary from 0.2 to 1.3 m. For rectangular cross-section pillars in the underground part of the building, the cross-section height is up to 1.5 m, and the joints between these pillars and floor structure slabs produce substructures with very high stiffness. Furthermore, implementation of rigid links uses the penalty function method, and the penalty parameters, assigned automatically, is between 1000 and 10,000. This may lead to poor conditioned problem and slow convergence of iterative methods.

Table 1 presents the following parameters for different ordering methods: number of non-zero entries in the complete

Table 2 – Solution time for Raketnyj bulvar, using the PSICCG method on the computer A.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	76	487	563
2	58	333	391
3	58	323	381
4	59	293	357

Table 3 – Solution time for Raketnyj bulvar, using different methods on the computer A.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	357	4	CM	159/150/144/147/124/114/143
PARFES	492	4	OOC	–
BSMFM	695	4	OOC	–
ICCG0	4332	4	CM	5033/5649/6754/6833/4062/6203/5804

Table 4 – Solution time for Raketnyj bulvar, using the PSICCG method on the computer C.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	107	766	873
2	80	454	534
3	79	354	433
4	79	275	354
5	79	265	320
6	79	211	290
7	79	177	256

Table 5 – Solution time for Raketnyj bulvar, using different methods on the computer C.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	256	7	CM	159/150/144/147/124/114/143
PARFES	135	8	CM	–
BSMFM	391	8	CM	–
ICCG0	3352	7	CM	5033/5649/6754/6833/4062/6203/5804

Cholesky factor **L** and the incomplete factor **H**, number of entries rejected during incomplete factorization, total number of iterations for all right hand sides, and total solution time for PSICCG used in 4 threads on a computer A with Intel® Core™ i7-27600QM processor. Fig. 4a–c shows the portraits of stiffness matrices for each ordering methods; taking the symmetry into consideration, only the upper triangular part is shown. The following values were assumed: $\psi=10^{-10}$, $\psi_1=10^{-7}$, $tol=10^{-4}$.

The above results show a strong dependence of the number of elements rejected during incomplete factorization on the ordering method, and the former's impact on the quality of preconditioning. Based on the considerations set out in Section 2, the MMD ordering algorithm [5] is used for all problems reviewed in this paper.

The results of solving this problem using the computer A with Intel® Core™ i7-27600QM processor are shown in Tables 2 and 3, and the results for the DELL workstation (computer C) with two 6-core Intel Xeon X5660 processors, in Tables 4 and 5. The values used for the PSICCG method on both computers were $\psi=10^{-10}$, $\psi_1=10^{-7}$ and $tol=10^{-4}$.

The number of threads used for PARFES and BSMFM methods was assumed to be equal to the number of processors, are not occupied by threads, which run own processes of operating system. The number of threads for iterative methods was determined by the number of right hand sides. The speeding up with the increased amount of threads for the PSICCG method is shown in Fig. 5 (computer C with Intel Xeon X5660 processor).

Example 2. Model of multistory office block, Atrium_4_1. The problem contains 5 right hand sides and 7,328,394 equations. The design model and its fragment are shown in Figs. 6 and 7. The design model includes triangular 6-node shell finite elements, rectangular 8-node shell finite elements, triangular and rectangular transit finite elements, finite elements of spatial frame, and specialized finite elements—elastic supports that simulate the work of elastic foundation.

The thicknesses of shell elements are between 0.13 m and 0.6 m, which leads to stiffness dispersion up to 100 times.

Solving this problem on the computer A with the Intel® Core™ i7-27600QM processor turned out inefficient because of the insufficient amount of RAM (8 GB). The value of $\psi=10^{-5}$ did not allow for fast convergence with this method. The solution time with 4 threads was 23,219 s; in order to achieve convergence with the desired degree of precision, 5739/3437/5521/5573/5433 iterations were done for each right hand side, respectively.

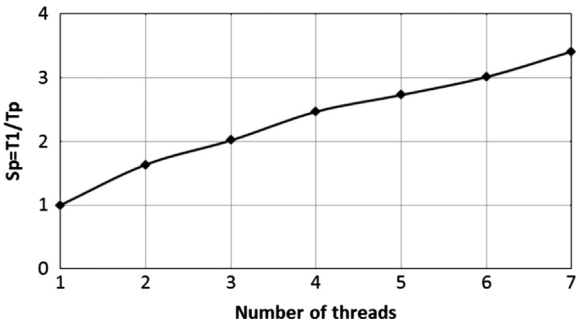

Fig. 5 – Speeding up of the Raketnyj bulvar problem solving with the PSICCG method, with increased number of threads. Computer C with Intel Xeon X5660 processor.

The results of solving this problem using the computer B with AMD Phenom™ II x4 995 processor are shown in Tables 6 and 7, and using the DELL workstation with two 6-core Intel Xeon X5660 processors (computer C), in Tables 8 and 9.

When solving the problem on the computer B with AMD Phenom™ II x4 995 processor, the following values were used: $\psi=10^{-11}$, $\psi_1=10^{-7}$, $tol=10^{-4}$.

When solving the problem on the computer C with Intel Xeon X5660 processor, the following values were used: $\psi=10^{-12}$, $\psi_1=10^{-7}$ and $tol=10^{-4}$. The large amount of RAM allowed decreasing the ψ value tenfold compared to the calculations on the computer B with AMD Phenom™ II x4 995 processor, which resulted in a smaller amount of

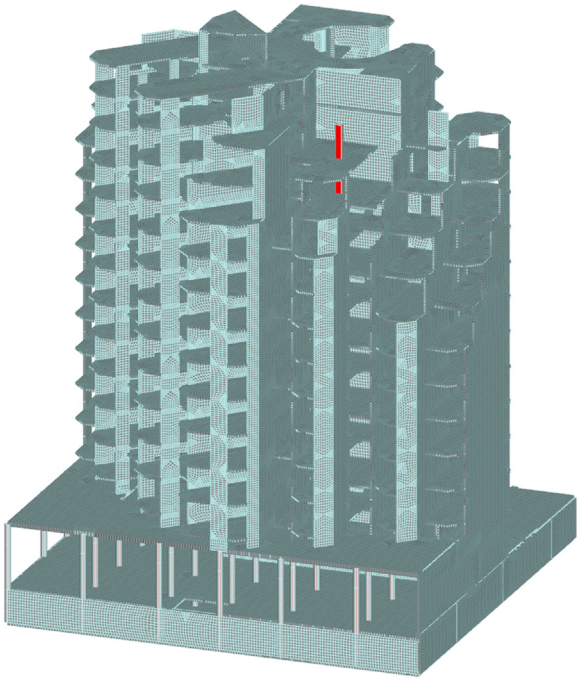

Fig. 6 – Atrium 4_1, design model of office building (7,328,394 equations).

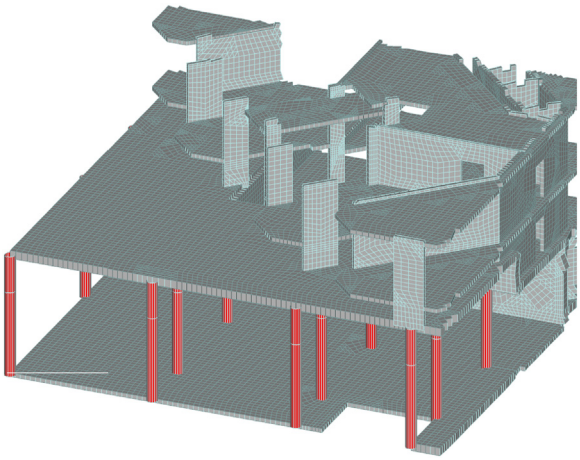

Fig. 7 – Fragment of Atrium 4_1 design model.

Table 6 – Solution time for Atrium 4_1 using the PSICCG method on the computer B.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	612	5228	5840
2	596	3232	3828
3	428	2587	3015
4	407	2306	2713

Table 7 – Solution time for Atrium 4_1, using different methods on the computer B.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	2713	4	CM	131/214/205/205/201
PARFES	17,445	4	OOC	–
BSMFM	Not solved—not enough RAM			
ICCG0	57,769	4	CM	10,521/15,149/15, 371/16,136/14,969

Table 8 – Solution time for Atrium 4_1 using the PSICCG method on the computer C.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	862	3186	4048
2	597	2068	2665
3	514	1447	1961
4	496	1350	1848
5	522	825	1347

Table 9 – Solution time for Atrium 4_1, using different methods on the computer C.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	1347	5	CM	140/188/179/178/174
PARFES	2699	10	OOC	–
BSMFM	6048	10	OOC	–
ICCG0	35,048	5	CM	10,521/15,149/15,371/16,136/14,969

iterations required to achieve convergence with the desired degree of precision.

The speeding up of the solving with the increase in the number of threads for the PSICCG method is shown in Fig. 8.

Example 3. Model of a multistory building with soil interaction (Oster-RS-34-PS2-36). The problem contains 4 right hand sides and 1,929,544 equations. The design model and its fragment are shown in Figs. 9 and 10.

The design model includes triangular and quadrilateral shell finite elements, and finite elements of spatial frame. The soil is modeled with 6- and 8-node volumetric finite elements. The building is supported by a monolith foundation slab. The thicknesses of shell elements modeling floor structures and walls are within 0.2 and 0.6 m. The dispersion of cross-sections of columns is from 0.1×0.1 m to 0.35×0.35 m.

Using the PSICCG method on the computer A with Intel® Core™ i7-27600QM processor, the following values were used: $\psi = 10^{-9}$, $\psi_1 = 10^{-7}$, $tol = 10^{-4}$. The solving results are shown in Tables 10 and 11.

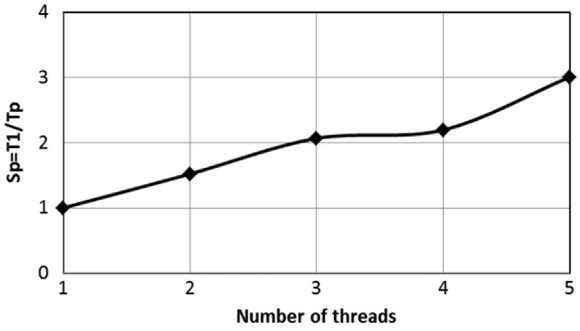

Fig. 8 – Speeding up of the Atrium_4_1 problem solving using the PSICCG method with an increase in the number of threads. Computer C with the Intel Xeon X5660 processor.

The same problem was solved on the computer B with AMD Phenom™ II x4 995 processor, with values of $\psi = 10^{-10}$, $\psi_1 = 10^{-7}$, $tol = 10^{-4}$. The results are shown in Tables 12 and 13.

The speeding up of solving with an increase in the number of threads is shown in Fig. 11.

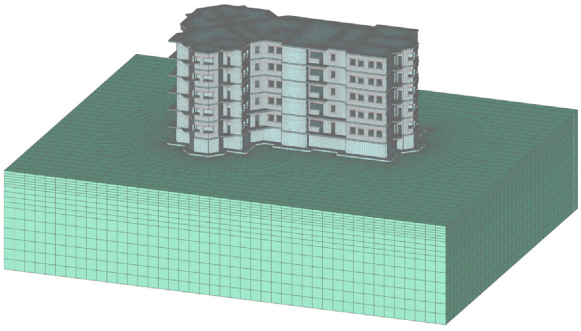

Fig. 9 – Oster-RS-34-PS2-36 design model of multistory building with soil interaction (1,929,544 equations).

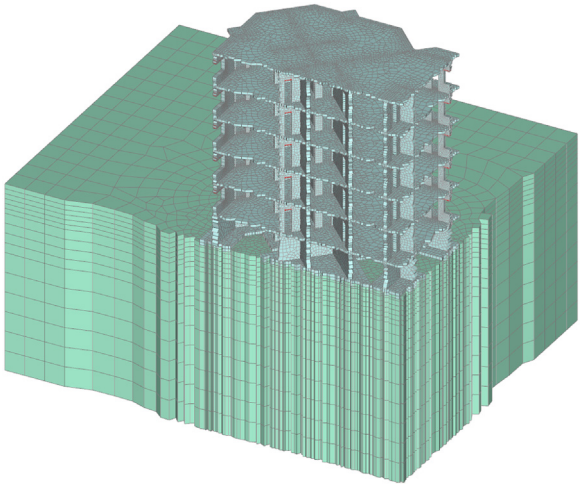

Fig. 10 – Fragment of Oster-RS-34-PS2-36 design model.

Since the problem contains 5 right hand parts, solution times with 3 and 4 threads are virtually identical.

Example 4. Model of multistory building of volumetric concrete blocks (Schema 1). The problem contains 13 right hand sides and 3,198,609 equations. The design model and its fragment are shown in Figs. 12 and 13.

The design model includes 3- and 4-node shell finite elements, and axial finite elements. The thicknesses of walls and floor structures are within 0.1 and 0.3 m.

For the PSICCG method used to solve the problem on the computer B with AMD Phenom™ II x4 995 processors, the values used were $\psi=10^{-13}$, $\psi_1=10^{-13}$ and $tol=10^{-4}$. The results are shown in Tables 14 and 15.

Tables 16 and 17 shows the results of solving the same problem using the computer C with Intel Xeon X5660 processor, with the values of $\psi=10^{-14}$, $\psi_1=10^{-7}$ and $tol=10^{-4}$.

The number of threads was increased up to 10, because 2 cores out of 12 were occupied with OS processes.

The relationship between speeding up and the increased number of threads is shown in Fig. 14.

With the given number of right hand sides, the solution times with 7, 8, 9 and 10 threads are virtually identical, because the task queue Q is resolved in two stages. For 7 threads, 7 right hand sides are iterated at the first stage and 6 at the second; for 8 threads, 8 and 5 right hand sides respectively, and so on.

6. Conclusions

Design models of multistory buildings with various architectural, layout and engineering solutions were considered. For

Table 10 – Solution time for Oster-RS-34-PS2-36 using the PSICCG method on the computer A.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	424	2858	3282
2	275	2032	2307
3	252	1818	2070
4	244	2049	2293

Table 11 – Solution time for Oster-RS-34-PS2-36, using different methods on the computer A.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	2293	4	CM	591/607/604/607/591
PARFES	1079	4	OOC	–
BSMFM	1649	4	OOC	–
ICCG0	12,842	4	CM	13,429/13,429/13,493/13,627/13,493

Table 12 – Solution time for Oster-RS-34-PS2-36 using the PSICCG method on the computer B.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	2031	3769	5800
2	1188	2400	3588
3	922	1802	2724
4	799	1878	2677

Table 13 – Solution time for Oster-RS-34-PS2-36, using different methods on the computer B.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	2677	4	CM	396/393/392/393/392
PARFES	1054	4	OOC	–
BSMFM	1317	4	OOC	–
ICCG0	20,661	4	CM	13,429/13,429/13,493/13,627/13,493

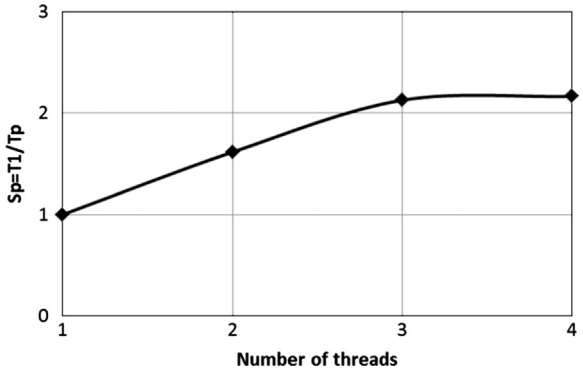

Fig. 11 – Speeding up of Oster-RS-34-PS2-36 problem solving with the PSICCG method as the number of threads increases. Computer B with AMD Phenom™ II x4 995 processor.

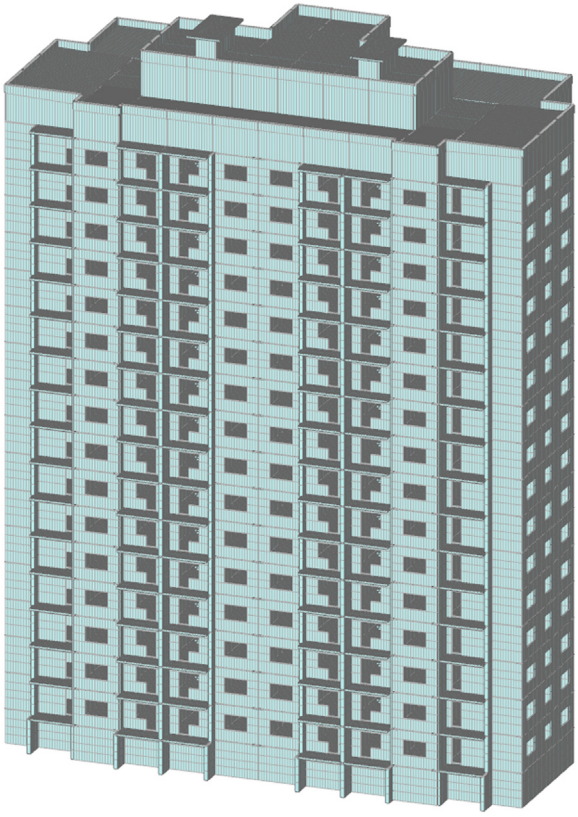

Fig. 12 – Schema 1, design model of multistory building of 3D concrete blocks (3,198,609 equations).

all models considered, the suggested PSICCG iterative method demonstrated reliable convergence and sustainable speeding up with the increase in the number of processors. With a sufficiently large amount of RAM, the suggested method,

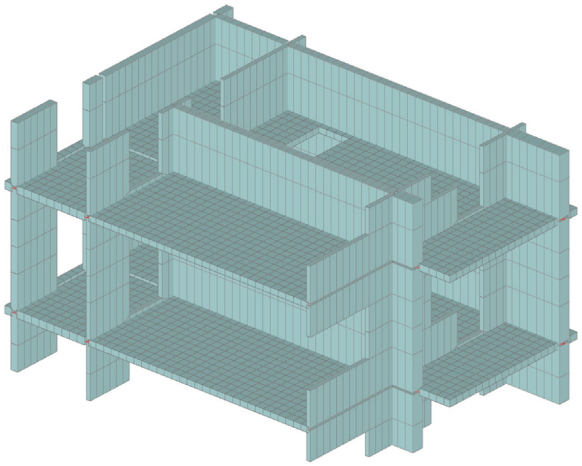

Fig. 13 – Fragment of Schema 1 design model of multistory building of 3D concrete blocks.

based on the technique of sparse matrices, allowed using a value of the rejection parameter ψ that was low enough to allow the desired precision of convergence with a small amount of iterations.

For all solved practical problems, both the number of iterations required for convergence and the solution time for each problem was higher for ICCG0, the preconditioned conjugate gradient method with incomplete Cholesky "by position" factorization, compared to the PSICCG method. This shows that the design models of multistory buildings often lead to poorly conditioned problems.

Parallelization at the iteration stage, based on each right hand side is processed on a separate thread, allowed to considerably speed up task-solving for desktop multi-core computers.

An overview of the most popular high-performance libraries, such as Intel MKL and IMSL, showed that their iterative methods for solving systems of linear algebraic equations with sparse symmetric matrices that implement the preconditioned conjugate gradient method do not support multithreading. This is most likely related to the unsolved issue of parallelizing the forward-back substitution procedure on computers using the SMP architecture. The speed up of iterative methods on computers with such architecture is essentially lower than for sparse direct solvers, because leading low-performance algorithms are accelerated with growing of processor number only as long as the system bus bandwidth allows it. The results, presented in Fig. 1, clearly illustrate this fact.

A comparison of solution times for the considered problems between the PSICCG method and the direct

Table 14 – Solution time for Schema 1 using the PSICCG method on the computer B.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	626	2175	2801
2	382	1305	1687
3	320	966	1286
4	291	858	1149

Table 15 – Solution time for Schema 1, using different methods on the computer B.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	1149	4	CM	71/55/62/65/73/62/59/59/18/64/63/65
PARFES	366	4	OOC	–
BSMFM	1033	4	OOC	–
ICCG0	51,256	4	CM	18,673/15,018/10,356/14,242/18,371/18,507/14,462/12,904/12,888/5994/14,042/14,108/14,877

Table 16 – Solution time for Schema 1 using the PSICCG method on the computer C.

Number of threads	Incomplete factorization (s)	Iterative process (s)	Total solution time (s)
1	784	808	1592
2	481	476	957
3	372	354	726
4	337	286	623
5	327	239	566
6	332	235	567
7	331	189	520
8	338	189	527
9	326	189	515
10	326	189	515

Table 17 – Solution time for Schema 1, using different methods on the computer C.

Method	Total solution time (s)	Number of threads	RAM mode	Number of iterations per right hand side
PSICCG	515	10	CM	47/52/57/57/57/58/66/68/39/52/50/55/57
PARFES	230	10	OOC	–
BSMFM	1030	10	OOC	–
ICCG0	22,258	10	CM	18,673/15,018/10,356/14,242/18,371/18,507/14,462/12,904/12,888/5994/14,042/14,108/14,877

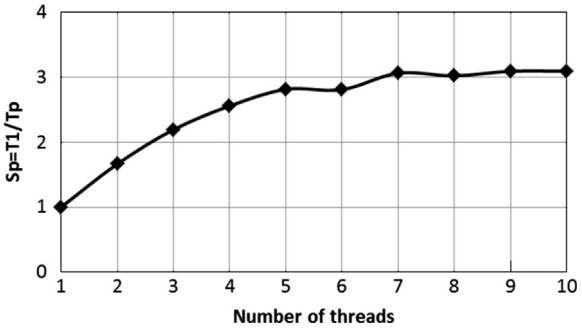

Fig. 14 – Speeding up of Schema 1 problem solving with the PSICCG method as the number of threads increases. Computer C with Intel Xeon X5660 processor.

PARFES method showed that if the factorized stiffness matrix is stored in the RAM, PARFES provides for a quicker solution. However, in case of a RAM deficit, PARFES switches to the OOC or OOC1 mode. Then, in a number of cases, the iterative PSICCG method becomes more efficient.

For all problems considered in this paper, PARFES demonstrates higher performance than the multifrontal method. Furthermore, PARFES uses less RAM working in the OOC1 mode. PARFES was able to successfully solve problem 2, which contained 7,328,394 equations, on a computer with 16 GB RAM, while BSMFM failed at that.

Therefore, modern FEA software should contain both direct and iterative methods.

Acknowledgments

The preparation of this work was supported by National Science Center in Poland, based on the DEC-2011/01/B/ST6/00674 decision. The author is deeply grateful to the SCAD Soft team for the large collection of real problems offered by them.

REFERENCES

[1] P.R. Amestoy, I.S Duff., J.Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, Computer Methods in Applied Mechanics and Engineering 184 (2000) 501–520.
[2] BLAS—Basic Linear Algebra Subprograms, 2008. URL: ⟨http://www.netlib.org/blas/⟩ (accessed 29.04.12).
[3] Eun-Jin Im, K. Yelick, R. Vuduc, Sparsity: optimization framework for sparse matrix kernels, International Journal of High Performance Computing Applications 18 (2004) 135–158.
[4] A. George, J.W.H. Liu, Computer solution of Sparse Positive Definite Systems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
[5] A. George, J.W.H. Liu, The evolution of the minimum degree ordering algorithm, SIAM Review 31 (1989) 1–19.
[6] G.H. Golub, C.F. Van Loan, Matrix Computations, 3rd ed., John Hopkins University Press, 1996.
[7] N.I.M. Gould, Y. Hu, J.A. Scott, A Numerical Evaluation of Sparse Direct Solvers for the Solution of Large Sparse, Symmetric Linear Systems of Equations. Technical Report RAL-TR-2005-005, Rutherford Appleton Laboratory, 2005.
[8] J.W. Demmel, Applied Numerical Linear Algebra, SIAM, Philadelphia, 1997.
[9] IMSL® Fortran Library Features, Fortran Library Documentation, 2012. ⟨http://www.roguewave.com/support/product-documentation/imsl-numerical-libraries/fortran-library.aspx⟩ (accessed 29.04.12).
[10] Intel® Math Kernel Library Reference Manual, Document Number: 630813-029US, 2012. ⟨http://www.intel.com/software/products/mkl/docs/WebHelp/mkl.htm⟩ (accessed 29.04.12).
[11] Intel(R) Math Kernel Library for Windows* OS User's Guide, Document Number: 315930-013 US. Intel(R) MKL 10.3 - Windows* OS. Threaded Functions and Problems, 2012. ⟨http://software.intel.com/sites/products/documentation/hpc/composerxe/en-us/mklxe/mkl_userguide_win/MKL_UG_managing_performance/Threaded_Routines.htm#blas⟩ (accessed 29.04.12).
[12] A. Jennings, Development of an ICCG algorithm for large sparse systems, in: D.J. Evans (Ed.), Preconditioned Methods. Theory and Applications, Gordon and Breach, Science Publishers, Inc., 1983, pp. 425–438.
[13] S. Fialko, PARFES: a method for solving finite element linear equations on multi-core computers, Advances in Engineering Software 40 (12) (2010) 1256–1265.
[14] S. Fialko, The Block Substructure Multifrontal Method for Solution of Large Finite Element Equation SETS, Technical Transactions, 1-NP/2009, issue 8, (2009) 175–188 (in Polish).
[15] S. Fialko, High-performance aggregation element-by-element iterative solver for large-scale complex shell structure problems, Archives of Civil Engineering XLV (2) (1999) 193–207.
[16] S. Fialko, A sparse incomplete Cholesky conjugate gradient method for finite element analysis of large-scale problems in structural mechanics, In: Proceedings of the CMM-2007—Computer Methods in Mechanics, Lodz—Spala, Poland, June 19–22, 2007, pp. 145–146.
[17] S. Fialko, A parallel sparse direct finite element solver for desktop computers, In: Proceedings of the 19th International Conference on Computer Methods in Mechanics, Warsaw, Poland, 9–12 May 2011, pp. 183–184.
[18] K. Malkowski, Ingyu Lee, P. Raghavan, M.J. Irwin, Conjugate gradient sparse solver: performance-power characteristics, In: Proceedings of the Parallel and Distributed Processing Symposium, 2006.
[19] METIS—Serial Graph Partitioning and Fill-Reducing Matrix Ordering. 2012. ⟨http://glaros.dtc.umn.edu/gkhome/metis/metis/overview⟩ (accessed 29.04.12).
[20] M. Papadrakakis, Solving Large-Scale Problems in Mechanics, John Wiley & Sons Ltd., 1993.
[21] A.V. Perelmuter, S.Y. Fialko, Problems of computational mechanics relate to finite-element analysis of structural constructions, International Journal for Computational Civil and Structural Engineering 1 (2) (2005) 72–86.
[22] O. Schenk, K. Gartner, Two-level dynamic scheduling in PARDISO: improved scalability on shared memory multiprocessing systems, Parallel Computing 28 (2002) 187–197.
[23] M. Suarjana, H.Kincho Law, A robust incomplete factorization based on value and space constraints, International Journal for Numerical Methods in Engineering 38 (1995) 1703–1719.