Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 277 – 283

ISBN 978-83-60810-14-9
ISSN 1896-7094

A Sparse Shared-Memory Multifrontal Solver in SCAD Software

Sergiy Fialko
Cracow University of Technology, Poland ul. Warszawska 24, 31-155 Cracow, Poland
Email: sfialko@poczta.onet.pl

***Abstract*—A block-based sparse direct finite element solver for commonly used multi-core and shared-memory multiprocessor computers has been developed. It is intended to solve linear equation sets with sparse symmetrical matrices which appear in problems of structural and solid mechanics. A step-by-step assembling procedure together with simultaneous elimination of fully assembled equations distinguishes this method from the well-known multifrontal solver, so this approach can be interpreted as a generalization of the conventional frontal solver for an arbitrary reordering. This solver is implemented in the commercial finite element software SCAD (www.scadsoft.com), and its usage by numerous users has confirmed its efficiency and reliability.**

I. Introduction

THE solver is based on the idea of a step-by-step assembling of a given structure out of separate finite elements and substructures obtained at the previous assembling steps, and a simultaneous elimination of the fully assembled equations. It is an evolution of the frontal solver [1], and it differs from the classical multifrontal solver [2][3] which is a purely algebraic solver and gets the assembled global matrix in a compressed format as input data.

The key distinctive features of the presented method are:

- Each node is associated with a group of equations (usually, 6 equations per node for shell and space frame finite elements, and 3 equations per node for volumetric elements in unconstrained nodes). Thus, this approach produces the natural aggregation of equations, in this way improving the quality of reordering algorithms and speeding up the performance at the factorization stage. We refer to the elimination of a node of a finite element model at each elimination step. It means that a group of equations associated with the given node will be eliminated.
- An object-oriented approach is applied. A front is a C++ class object that encapsulates eliminated nodes at the current elimination step, a list of nodes and a list of equations for the current front, a pointer to the dense matrix which we denote as a frontal matrix, and so on.
- The whole elimination process is performed on a sequence of frontal matrices of the decreasing dimensionality.

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

Key points of the method are described in [4]. The paper [5] presents an improved version of the solver. The Cholesky block factorization algorithm was applied instead of a low-performance LU factorization from [4], and more efficient reordering algorithms comparing to [4] were implemented.

The current paper presents an extension of this solver onto the class of multi-core and multiprocessor shared-memory computers which are very popular today. The contemporary version of the solver is implemented in SCAD – one of the most popular finite element software applications for analysis and design of building structures in the Commonwealth of Independent States region.

II. Analysis Stage

A. Reordering and Symbolic Factorization

Modern reordering algorithms have a heuristic nature and do not provide an exact solution of the nonzero entries minimization problem. Moreover, we never know in advance what reordering strategy leads to the more optimal result for a given problem. Therefore several reordering methods have been developed. The most efficient methods for problems of structural mechanics usually are the minimum degree algorithm MMD , the nested dissection method ND , and the hybrid approach ND_MMD .

The fast symbolic factorization algorithm calculates the number of nonzero entries for each method of the ones listed above and then chooses the method that produces the least number of nonzero entries.

The use of a nodal adjacency graph instead of a graph of equations reduces the amount of data in a natural way and makes the reordering algorithms and the symbolic factorization procedure very fast: it takes only a few seconds even in very large problems (2,000,000 – 4,000,000 equations) to check 3 reordering methods and do the symbolic factorization.

B. A Process Descriptor Data Structure

The reordering method establishes an order of the node elimination. The next step is to define the order of the finite element assembling.

The node is considered fully assembled if all finite elements, including this very node, have been already coupled. Adding any of the remaining finite elements does not make any change in the coefficients of equations associated with

this node. So, all equations belonging to the fully assembled node should be eliminated.

Let us consider a simple example, a square plate with the mesh 2x2 (Fig. 1). Before starting the elimination process, the whole structure is presented as exploded into separate finite elements.

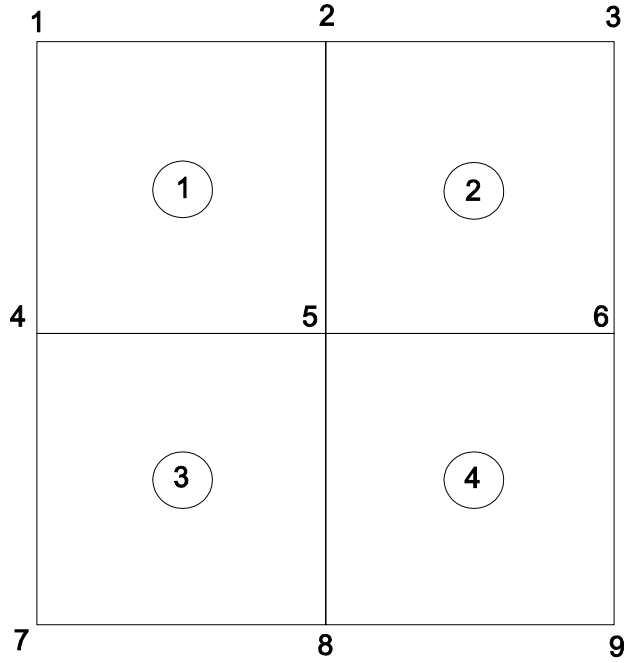

Fig. 1 Square plate 2x2

The sequence of nodal elimination produced by the reordering algorithm is: 1, 3, 7, 9, 2, 6, 8, 4, 5. Node 1 is the first node to be eliminated. This node belongs only to finite element 1. So, we take the finite element 1 and node 1 as fully assembled. The respective finite element matrix will be:

$$\begin{matrix} 1 & 2 & 5 & 4 \end{matrix}$$
$$\begin{pmatrix} s_{11} & & & \\ s_{21} & s_{22} & & \\ s_{31} & s_{23} & s_{33} & \\ s_{41} & s_{24} & s_{34} & s_{44} \end{pmatrix}, \qquad (1)$$

where s_{ij} is a 6x6 matrix block because we suppose the design model to have six degrees of freedom per node. The global node numbers (global indexes) are shown at the top. The first block column contains the fully assembled equations. The partial factorization, covering only fully assembled equations, is then performed and the fully decomposed part of matrix is moved to a special buffer that contains the fully factorized global matrix. The remaining part of the matrix is an incomplete front; it will wait to be used at the following factorization steps. Local indexes are used during the partial factorization. Matrix (1) is a frontal matrix.

In the same way the nodes 3, 7, 9 are eliminated, and their respective incomplete fronts are created.

During the elimination of node 2 all finite elements have been already involved, and now we look through the lists of global indexes in the incomplete fronts. Node 2 is present in the global indexes of previous (incomplete) fronts 1, 2 (Table I), so we must assemble incomplete fronts 1, 2 to obtain a frontal matrix containing fully assembled equations for node 2.

This process is illustrated by Table I which shows a process descriptor data structure. The number of elimination steps is equal to the number of nodes of the design model.

We search for the node to be eliminated in the lists of nodes of each remaining finite element and in the lists of global indexes for each of the previous fronts. The fronts from the preceding elimination steps, which contain this eliminated node number, are the previous fronts. The previous fronts and their corresponding finite elements, pointed to by the last column of Table I, should be assembled to obtain the frontal matrix for the current front.

C. Frontal Tree

The process descriptor data structure allows us to create a frontal tree. We take the last front from the bottom of Table I (it is Front 9) and put it at the top of the frontal tree (Fig. 2). Front 8 is the previous front for Front 9 – we put it under Front 9. For Front 8, Front 7 is its previous one – we put Front 7 under Front 8. And so on.

We reorder the fronts to reduce the storage memory required for incomplete fronts. The new front numbers are shown in italic under the original front numbers.

TABLE I.
A PROCESS DESCRIPTOR DATA STRUCTURE

No. of front, elimination step	Node being eliminated	List of nodes in the frontal matrix	List of previous fronts	List of FEs fed to the assembling
1	1	1,2,4,5	—	1
2	3	3,6,5,2	—	2
3	7	7,4,5,8	—	3
4	9	9,8,5,6	—	4
5	2	2,4,5,6	1,2	—
6	6	6,8,5,4	5,4	—
7	8	8,5,4	6,3	—
8	4	4,5	7	—
9	5	5	8	—

The frontal tree consists of (a) nodal fronts which have more than one previous front, (b) sequential fronts which have only one previous front, and (c) start fronts which do not have any previous front.

The core memory is allocated dynamically for objects of the nodal and start fronts. Each sequential front inherits the address of the frontal buffer from its previous front – this helps avoid the time-consuming memory allocation and copying of incomplete front. Moreover, if the sequence of sequential fronts does not have any assembled finite element, it is possible to consolidate such fronts to enlarge the block size of fully assembled equations and improve the performance. The consolidated frontal tree is presented in Fig. 3.

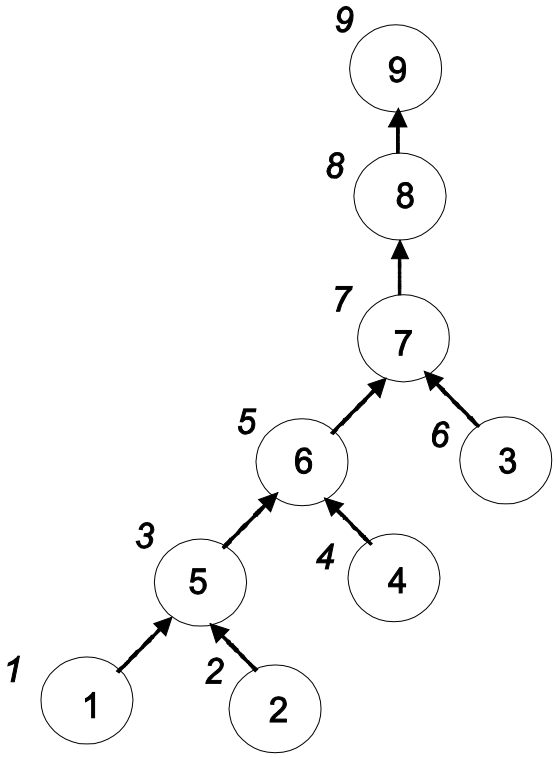

Fig. 2 A frontal tree

The buffer for storing the incomplete fronts (BuffFront), allocated in the core memory, is virtualized. When the size of the model exceeds the capacity of the random access memory (RAM), the buffer for the incomplete fronts is uploaded to hard disk. If all required previous fronts are in RAM during the assembling of the current nodal front, they are taken from BuffFront.

Otherwise, the previous fronts are taken from hard disk. The BuffFront is compressed from time to time to get rid of slow I/O operations during virtualization.

The presented method is a substructure-by-substructure approach, because each front presents a collection of coupled finite elements – a substructure, and the elimination process consists of a step-by-step assembling of these substructures and a simultaneous elimination of fully assembled equations.

III. Factorization of Frontal Matrix

Generally, the structure of the frontal matrix can be very complex (Fig. 4). The matrix is symmetrical, and only the lower triangle part of it is stored in RAM. The fully assembled equations (grayed) are stored continuously but in arbitrary parts of the matrix (not necessarily at the top). This peculiarity restricts the direct application of the popular Lapack high-performance packages (Linpack, BLAS) and forces us to develop our own high-performance software for factorizations in frontal matrices.

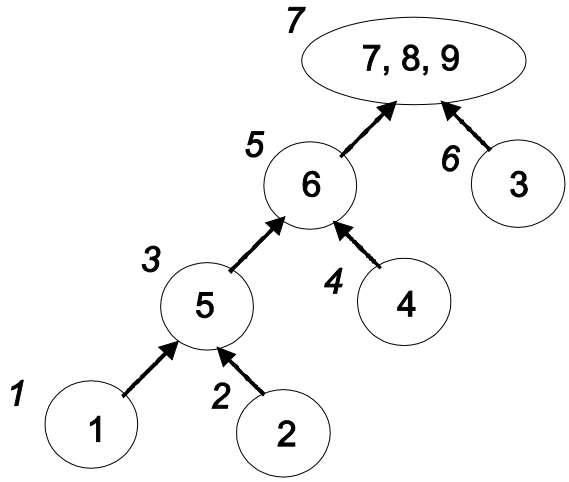

Fig. 3 Consolidated frontal tree

Thus, a Cholesky block method has been developed, which is a generalization of the frontal elimination approach [1], [4], where the fully assembled equations are stored in arbitrary parts of the matrix, onto the block version. It allows us to achieve the BLAS level 3 performance.

$$\begin{pmatrix} \mathbf{A} & \mathbf{W}_1^T & \mathbf{B}^T \\ \mathbf{W}_1 & \mathbf{F} & \mathbf{W}_2^T \\ \mathbf{B} & \mathbf{W}_2 & \mathbf{C} \end{pmatrix} =$$

$$= \begin{pmatrix} \tilde{A} & \tilde{W}_1 & \tilde{B}^T \\ 0 & L & 0 \\ \tilde{B} & \tilde{W}_2 & \tilde{C} \end{pmatrix} \cdot \begin{pmatrix} I & & \\ & S_L & \\ & & I \end{pmatrix} \cdot \begin{pmatrix} I & 0 & 0 \\ \tilde{W}_1^T & L^T & \tilde{W}_2^T \\ 0 & 0 & I \end{pmatrix} \tag{2}$$

where $\mathbf{W_1}$, $\mathbf{F}$, $\mathbf{W_2}$ are blocks of fully assembled equations, **A, B, C** are blocks of partially assembled equations, which make up an incomplete front (Fig. 5) after the partial Cholesky factorization,

$$\begin{pmatrix} \widetilde{\mathbf{A}} & 0 \\ \widetilde{\mathbf{B}} & \widetilde{\mathbf{C}} \end{pmatrix} \tag{3}$$

The sign diagonal **I,** $\mathbf{S}_L$**, I** allows one to generalize the classic Cholesky factorization method onto a class of problems with indefinite matrices. The symbol "~" means that the

corresponding matrix block is modified during the partial factorization.

The fully assembled blocks are moved to the buffer for the decomposed part of the matrix, and next the factorization of it is performed in the address space of this buffer.

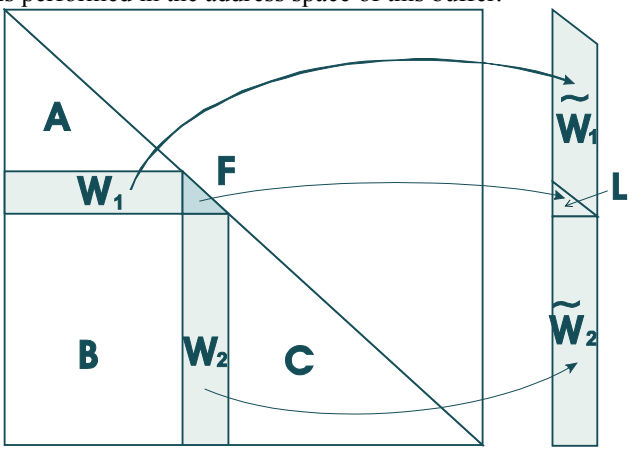

Fig. 4 The general structure of a frontal matrix. The fully assembled equations are moved to a buffer for the decomposed matrix

The incomplete front consists of sectors **A, B, C**. The symmetrical storage scheme is used. The coefficients of sector **A** remain in their positions, sector **B** moves up, and sector **C** moves up and to the left by the width of the grey strip. The final structure of the incomplete front is presented in Fig. 5.

The partial block factorization is performed in several steps:

- Factorize block **F**:

$$\mathbf{F} = \mathbf{L} \cdot \mathbf{S}_{\mathrm{L}} \cdot \mathbf{L}^{\mathrm{T}} \tag{4}$$

- Update blocks $\mathbf{W_1}$, $\mathbf{W_2}$:

$$\begin{aligned} \mathbf{L} \cdot \mathbf{S}_L \cdot \widetilde{\mathbf{W}}_1^T &= \mathbf{W}_1 \to \widetilde{\mathbf{W}}_1^T \\ \mathbf{L} \cdot \mathbf{S}_L \cdot \widetilde{\mathbf{W}}_2^T &= \mathbf{W}_2^T \to \widetilde{\mathbf{W}}_2^T \end{aligned} \tag{5}$$

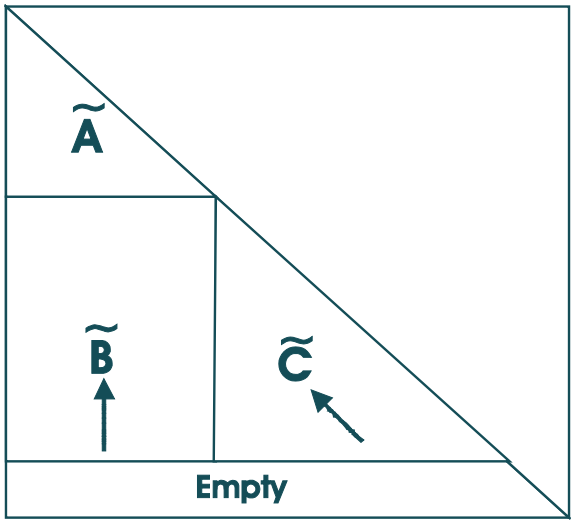

Fig. 5 The structure of the frontal matrix after the partial factorization. An empty strip appears at the bottom of matrix.

- Update sectors **A, B, C**:

$$\begin{aligned} \widetilde{\mathbf{A}} &= \mathbf{A} - \widetilde{\mathbf{W}}_1 \cdot \mathbf{S}_L \cdot \widetilde{\mathbf{W}}_1^T \\ \widetilde{\mathbf{B}} &= \mathbf{B} - \widetilde{\mathbf{W}}_2 \cdot \mathbf{S}_L \cdot \widetilde{\mathbf{W}}_1^T \\ \widetilde{\mathbf{C}} &= \mathbf{C} - \widetilde{\mathbf{W}}_2 \cdot \mathbf{S}_L \cdot \widetilde{\mathbf{W}}_2^T \end{aligned} \tag{6}$$

Expressions (4) – (6) are derived from (2) by means of the block matrix multiplication. Figs. 4, 5 help us to understand the structure of the frontal matrix. The main idea of the generalized Cholesky block factorization method is similar to the Gauss block elimination approach, reorganized to use the level 3 BLAS [10]. A particular case where a fully assembled part is at the top of the frontal matrix is presented in [5].

Matrix **F** has a small dimensionality, so there is no problem to store all matrix multipliers (4) in the processor cache and achieve a peak performance at this stage.

Stage (5) is a forward reduction performed on the package of right-hand sides – it is possible to achieve a good performance here, too.

The procedure that consumes most of the time is Stage (6). Each matrix multiplier $\mathbf{W_1}$, $\mathbf{W_2}$ is divided into smaller blocks, and the block matrix multiplication ensures the level 3 BLAS .

IV. Parallel Implementation in Frontal Matrix

The bottleneck in the multi-core and multiprocessor shared-memory computing systems is an insufficient bandwidth of the system bus which leads to a weak speedup as the number of processors grows. A typical dependence for the speedup parameter, $S_p = T_1 / T_p$, where T_1 is a computing time on one processor, T_p is that on p processors, for algorithms where the performance factor $q \le 2$ (matrix-vector multiplication, non-block matrix-matrix multiplication) is presented in Fig. 6. The performance factor is $q = f/m$ where f is the number of arithmetic operations for a given algorithm and m is the number of data transfers between RAM and the processor cache.

A totally different dependence takes place on the same computer for the block matrix-by-matrix multiplication; here $q \sim \sqrt{M}$ where M is the cache size (fig. 7).

So, it is possible to achieve an efficient speedup for computers with an insufficient bandwidth of the system bus by increasing the number of processors for level 3 BLAS algorithms, where $q \sim \sqrt{M}$, because these algorithms use significantly fewer RAM – cache – RAM transactions per single arithmetic operation than the algorithms where $q \le 2$.

The results presented in Figs. 6 through 8 have been obtained by special tests prepared by the author.

Therefore the main attention during the development of the parallel code is paid to the Cholesky block factorization in the frontal matrix and the fork-joint parallelization technique, which is used mainly at the stage 6 during the block matrix-by-matrix multiplication.

The Microsoft Visual Studio 2008 environment (the C++ language for most of the code and the C language for "bottleneck" fragments of the code in order to achieve a peak efficiency by compiler optimizations) with the OpenMP support is used. All program code, including the factorization of frontal matrix, was developed by the author.

The local reordering of equations performed in the frontal matrix for the nodal and start fronts allows us to reduce the amount of time-consuming operations of moving sectors **B** , **C** block and thus increase the performance of method (Fig. 8).

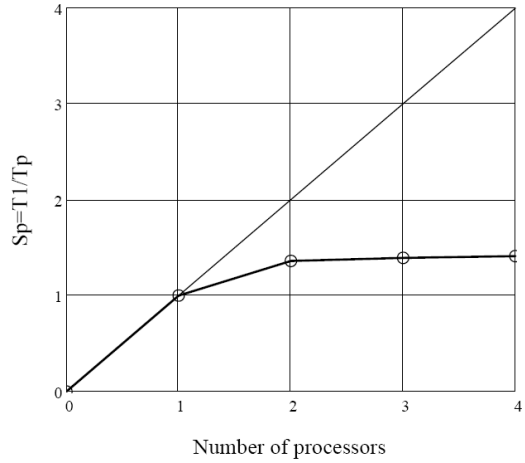

Fig. 6. $Sp = T_1 / T_p$ vs. number of processors p for algorithms where the performance factor $q \le 2$

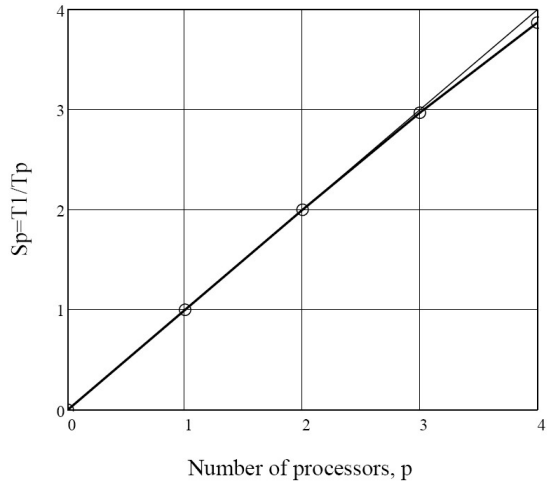

Fig. 7. $Sp = T_1 / T_p$ vs. number of processors p for algorithms where the performance factor $q \sim M^{1/2}$

We reorder the frontal nodes according to the sequence of global reordering and then take the reverse order – each eliminated node occurs at the end of the node list. It ensures such a structure of the frontal matrix that the fully assembled equations are at the bottom and no moving of incomplete fronts is required (only sector A is shown in the figure, sectors B, C are omitted).

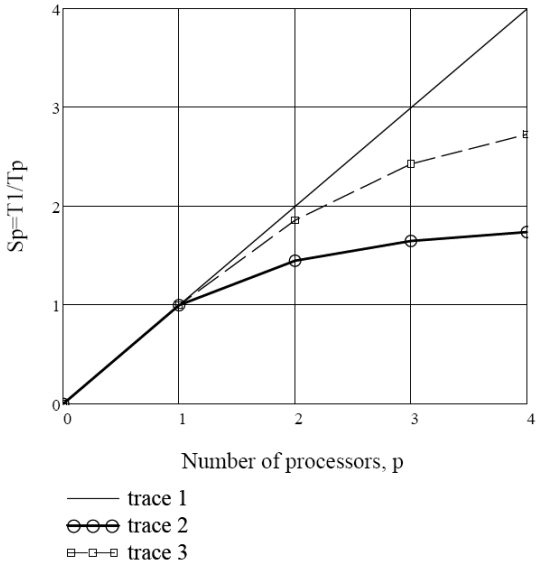

Fig. 8. $Sp = T_1 / T_p$ vs. number of processors p for factorization of frontal matrices. Trace 1 – perfect speedup, trace 2 – local fronts not reordered, trace 3 – reordered fronts.

For instance, in the example of Fig. 1 the order of node processing for front 7 (tab. I) must be 5, 4, 8 rather than 8, 5, 4, because this numbering ensures that at each front elimination step the fully assembled equations stay at the bottom of the matrix and no moving of incomplete fronts is required.

Typical dependencies $S_p = T_1 / T_p$ vs. the number of processors p for factorization of frontal matrices for presented method are shown in Fig. 8. The total time of factorization for all frontal matrices is under consideration. An essential improvement of performance occurs when the local reordering of equations in frontal matrices is performed (the dashed curve corresponds to local reordering and solid curve to no local reordering).

V. Numerical Results

All results presented further below have been obtained on the computer Intel® Core™2 Quad CPU Q6600 @2.40 GHz, cache 4xL1: 32 KB (code), 4xL1: 32 KB (data), 2xL2:4096 KB, random-access memory DDR2 800 MHz 4 GB, operating system Windows XP Professional.

D. Test 1

A cube with the mesh 40x40x40 consisting of brick volumetric finite elements is considered. Four corner nodes of the bottom face are restrained. This model comprises 68 921 nodes, 64 000 finite elements, 206 751 equations; the whole model is stored in RAM. The analysis stage, which includes reordering, symbolic factorization, creation of the process descriptor data structures, reordering and consolidation of the frontal tree, is performed as a serial computation. The computing time is 3 s. The solution stage, including forward – backward reductions, takes 4 s. The parallel computing stage covers only the factorization of frontal matrices

and takes a predominant part of the whole time required by the numerical factorization stage (Table II).

The factorized matrix, as well as the buffer for incomplete fronts, is stored in RAM, and due to this fact the speedup parameter on four processors is about 3.0. It is a good result for multi-core and shared-memory office computers. For problems of structural mechanics solved by finite element software based on the Intel Math Kernel Library (PARDISO solver –[15]), the speedup parameter on four processors is about 2.9.

TABLE II.
TIME OF NUMERICAL FACTORIZATION FOR CUBE 40X40X40

Number of processors	Numerical factorization – T_p, s	$S_p = T_1 / T_p$
1	519	1.0
2	280	1.85
3	202	2.57
4	167	3.10

However, the PARDISO solver works only with the core memory, so its capabilities are restricted to problems of relatively small dimensions.

B. Test 2

A square plate with two meshes 400x400 and 800x800 is considered. A quadrilateral flat shell finite element is used. The time of numerical factorization was compared with that for the sparse direct solver from ANSYS v 11.0. The first problem (mesh 400x400, Table III) contains 964 794 equations and the second one (mesh 800x800, Table IV) contains 3 849 594.

TABLE III.
TIME OF NUMERICAL FACTORIZATION FOR PLATE 400X400

Number of processors	Numerical factorization for ANSYS v. 11.0, s	Numerical factorization for presented solver, s
1	221	226
2	176	152
4	159	121

TABLE IV.
TIME OF NUMERICAL FACTORIZATION FOR PLATE 800X800

Number of processors	Numerical factorization with ANSYS v. 11.0, s	Numerical factorization with the presented solver, s
1	failed	2 091
2	failed	1 450
4	failed	1 080

For the first model (mesh 400x400), our solver stores the factorized matrix on hard disk, but the buffer for incomplete fronts is stored in the core memory. Due to virtualization of the factorized matrix, the scalability is worse comparing to the previous model – the speedup parameter on four processors is about 1.9. The performance demonstrated by both the sparse direct solver from ANSYS v. 11.0 and our solver is about the same.

For the second model (mesh 800x800), the solver from ANSYS v.11.0 failed due to insufficient RAM. Results for our solver are presented in Table IV. The factorized matrix is stored on hard disk, and virtualization is used for the buffer that contains incomplete fronts. The scalability is weak — the speedup parameter on four processors is about 1.9. The size of the factorized matrix is 9 723 MB.

C. Test 3

The design model of a multistory building complex is presented in Fig. 9.

The number of equations is 1 534 674, the size of the factorized global matrix is 5 355 MB, the numerical factorization time on four processors is 860 sec. The disk memory is used for virtualization of the factorized matrix as well as for virtualization of the buffer where the incomplete fronts are stored.

Test A demonstrates that the scalability of the presented method for multi-core and multiprocessor office computers is similar to the scalability of the sparse direct solver based on Intel Math Kernel Library when the dimensionality of the model permits to use only the core memory. Test B proves that the performance of our solver is similar to the performance of the conventional multifrontal solver from the well-known ANSYS software. In addition, test B shows that the presented solver works very efficiently with the core memory. Test C illustrates that the presented method solves a really large finite element problem of structural mechanics very efficiently.

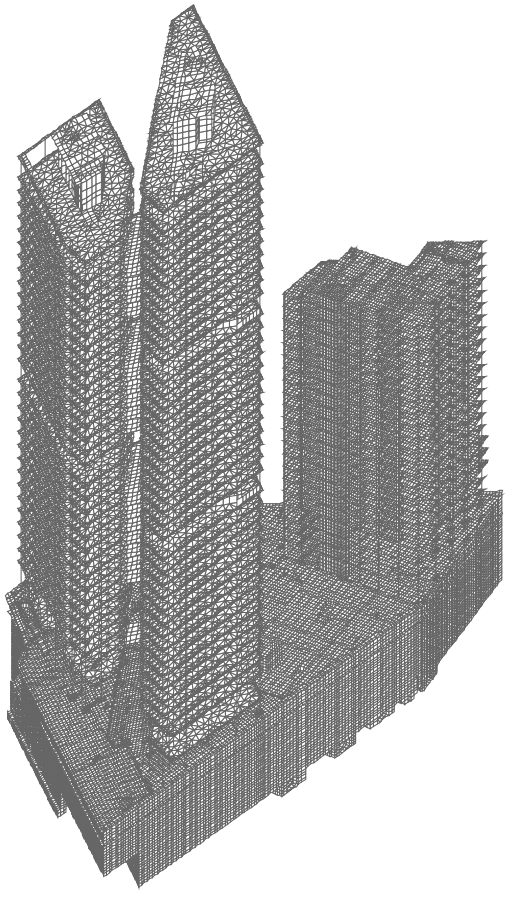

Fig. 9. Finite element model of a multistory complex “Moscow-city”, 255 779 nodes, 347 767 finite elements, 1 534 674 equations

VI. Conclusion

A block-based sparse direct multifrontal solver for finite element problems of structural and solid mechanics has been developed. This method is oriented mainly at the usage on common shared-memory multiprocessor and multi-core computers.

The proposed solver has a clear mechanical interpretation as a substructure-by-substructure method. It uses the modern reordering algorithms which effectively reduce the fill-ins.

The Cholesky block algorithm with a generalization onto the class of indefinite matrices is applied here to the factorization of frontal matrix. The complicated structure of the frontal matrix restricts the direct application of the BLAS high-performance package and forces us to develop our own high-performance code based on a subdivision of the frontal matrix into blocks.

The OpenMP technique is applied to produce a fork-joint parallelization. The block subdivision and an appropriate numbering of equations in the frontal matrix, which reduces the amount of low-performance moving operations, allow us to improve the scalability of the method.

A comparison with the sparse direct solver from ANSYS v11.0 demonstrates that the performance of the proposed solver is close to the performance of ANSYS, but the proposed solver allows us to solve essentially larger problems on the same computers than the ANSYS solver is capable of processing.

The reliability and efficiency of the proposed solver has been confirmed by numerous users of the SCAD commercial software.

References

[1] B. M. Irons, "A frontal solution program for finite-element analysis", *International Journal for Numerical Methods in Engineering*, 2, pp. 5 –32, 1970.

[2] I. S. Duff, J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear systems", *ACM Transactions on Mathematical Software*, 9, pp. 302–325, 1983

[3] F. Dobrian, A. Pothen, "Oblio: a sparse direct solver library for serial and parallel computations", *Technical Report describing the OBLIO software library*, 2000.

[4] S. Yu. Fialko, "Stress-Strain Analysis of Thin-Walled Shells with Massive Ribs", *Int. App. Mech.*, 40, N4, pp. 432–439, 2004.

[5] S. Yu. Fialko, "A block sparse direct multifrontal solver in SCAD software", in *Proc. of the CMM-2005 – Computer Methods in Mechanics June 21-24, 2005*, Czestochowa, Poland, pp. 73 – 74.

[6] A. George, J. W.-H. Liu, "The Evolution of the Minimum Degree Ordering Algorithm", *SIAM Rev*., 31, March, pp. 1-19, 1989.

[7] A. George, J. W.-H. Liu, *Computer solution of sparse positive definite systems* , New Jersey : Prentice-Hall, Inc. Englewood Cliffs, 1981, ch. 8.

[8] C. Ashcraft, J. W.-H. Liu, "Robust Ordering of Sparse Matrices Using Multisection", *Technical Report CS 96-01, Department of Computer Science, York University, Ontario, Canada,* 1996.

[9] A. George, J. W.-H. Liu, *Computer solution of sparse positive definite systems*, New Jersey : Prentice-Hall, Inc. Englewood Cliffs, 1981, ch. 5.

[10] J. W. Demmel, *Applied Numerical Linear Algebra*. SIAM, Philadellphia, 1997, ch. 2.

[11] A. Kumbhar, K. Chakravarthy, R. Keshavamurthy, G. V. Rao, "Utilization of Parallel Solver Libraries to solve Structural and Fluid problems", White paper by Cranes Software, http://www.intel.com/cd/software/products/asmo-na/eng/373466.htm.

[12] 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*.

[13] O. Schenk, K. Gartner, "On fast factorization pivoting methods for sparse symmetric indefinite systems", Technical Report CS-2004-004, Department of Computer Science, University of Basel, Switzerland, 2004.

[14] O. Schenk, K. Gartner, "Solving unsymmetric sparse systems of linear equations with PARDISO", Journal of Future Generation Computer Systems, 20(3), 475–487, 2004.

[15] O. Schenk, K. Gartner, W. Fichtner. "Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors", BIT, 40(1), 158–176, 2000.