ISBN 978-83-60810-51-4

Parallel Finite Element Solver for Multi-Core Computers

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

Abstract— **The sparse direct parallel finite element solver PARFES, which is based on L·S·L^T factoring of symmetric stiffness matrix, where S is the sign diagonal, is developed for analysis of problems of structural and solid mechanics using multi-core desktop computers. The subdivision of sparse matrix into rectangular blocks and the use of procedures from level 3 BLAS leads to high-performance factorization. Comparisons against multi-frontal solvers and PARDISO from the Intel Math Kernel Library when solving real problems from the SCAD Soft [1] collection demonstrate that PARFES is a good alternative to the multi-frontal method and, unlike PARDISO, allows solving large-scale problems using computers with limited amount of RAM, owing to virtualization.**

I. Introduction

Nowadays, large-scale engineering problems are more and more frequently solved on desktop computers instead of workstations, clusters, computer grids and other high-performance hardware.

Desktop computers have a relatively small amount of core memory and narrow memory system bandwidth. Therefore, many algorithms developed for high-performance computers are often less efficient on multi-core individual PCs, which are classified as symmetrical multiprocessing (SMP) computers. Therefore, developing software for this type of computers often requires taking into account some of their specifics mentioned above.

The scope of this work will be limited to finite element solvers for problems of solid and structural mechanics, implemented in software for desktop computers. The sparse direct solver presented in this article constructs decomposition:

$$\mathbf{K} = \mathbf{L}\cdot\mathbf{S}\cdot\mathbf{L}^T, \tag{1}$$

where **K** is a symmetric non-singular matrix, **L** is a lower triangle matrix and **S** is a diagonal matrix of signs, consisting of +1 or –1 on the diagonal.

Today, the multi-frontal solver [1], [3] is the most widespread in FEA software [6] among other types of sparse direct solvers, because it utilizes the core memory carefully and involves the disk (HD) if the dimension of the problem exceeds the RAM capacity. On the other hand, the multi-frontal solver results in a large transfer of data between different memory areas, which occurs when the remaining part of the frontal matrix is stored in the stack and restored from it. Aggregation of the current frontal matrix also requires a lot of data transfer [3]. On shared-memory desktop computers with narrow bandwidth, such operations are classified as low-performance ones, and are very poorly accelerated with the increase in the number of processors. This leads to a decrease in performance and restriction of speed up, or acceleration of the method with the increase in the number of processors.

Solvers from high-performance libraries, for instance [12], demonstrate high performance and good speed up on such computers, but their area of application is limited to the tasks that can be placed in the core memory. These problems have relatively small dimension (up to about 300 – 500 thousand equations) because the amount of RAM of desktop computers is usually restricted. Recently, particular attention has been paid to PARDISO – parallel direct solver for shared-memory computers [14] from the Intel Math Kernel Library (Intel MKL) [12]. According to its manual, the method uses the OOC (out-of-core) mode. However, neither the author of this paper nor his colleagues could solve any large-scale problem using PARDISO. Only relatively small tasks were solved successfully [11] in the OOC mode.

This paper presents the direct finite element solver PARFES for FEA on desktop computers. PARFES is more efficient than multi-frontal solvers and, unlike PARDISO and other methods from high-performance libraries, uses the disk space when the scale of the problem is too large to be solved in the core memory.

The presented version of this solver, in contrast to the previously published one [11], contains generalization for the case of indefinite matrices, which is very important in solving problems of structural dynamics and stability. The OOC1 mode was also added. While in the OOC mode, performance is slightly decreased compared to the core mode because of the number of I/O operations with the disk is

This work was supported by Narodowy Centrum Nauki on the basis of resolution DEC-2011/01/B/ST6/00674

[1] SCAD Soft (www.scadsoft.com) is a Software Company developing software for civil engineering. SCAD is FEA software, which is widely used in the CIS region and has a certificate of compliance to local regulations.

minimal, the OOC1 mode performs plenty of I/O operations with the disk. As a result, productivity and speed up of the method are significantly reduced, but the large problems become solvable on computers with small amounts of RAM.

II. Sparse Matrix Analysis Stage

The adjacency graph for the nodes of finite element model is prepared. Each node of this graph corresponds to a block-column in the sparse stiffness matrix, because each node of a finite element model comprises several degrees of freedom (DOFs), denoted as l_b. For unconstrained nodes $l_b \in [2 - 6]$ and depends on the type of the design model. Therefore, we are dealing with an initial grouping of equation into respectively small blocks, and should enlarge their size in order to increase the performance. In addition, the adjacency graph for the nodes of a finite element model is much smaller than the same for the sparse stiffness matrix. Therefore, the analysis stage with the adjacency graph for the nodes is much faster than that with the adjacency graph for the sparse matrix.

Reordering the nodal adjacency graph reduces the number of fill-inns. The reordering method is chosen among the minimal degrees algorithms MMD, METIS [13], or nested dissection [4]. The fast symbolic factorization procedure [4] allows to select a method that produces a minimal number of fill-inns and creates a non-zero structure of the factorized stiffness matrix *L*.

Decomposition of a sparse matrix into respectively large rectangular blocks is based on the analysis of the non-zero structure of each column. Let us consider a simple example (fig. 1).

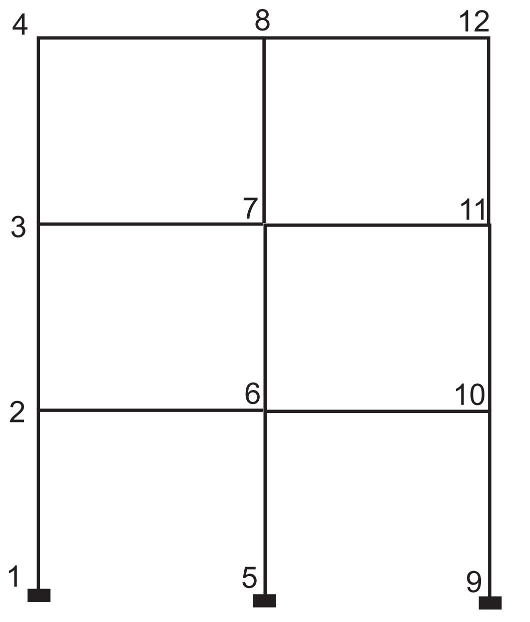

Fig. 1 Finite element model of plane frame

The adjacency graph after reordering using the MMD algorithm is presented in fig. 2.

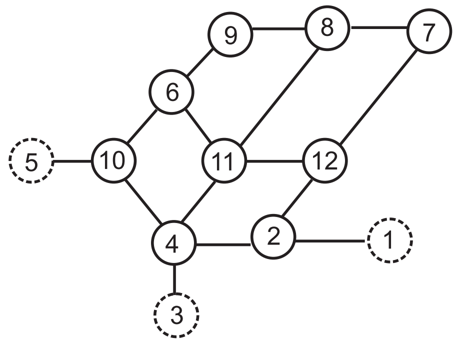

Fig. 2 Reordered adjacency graph

The reordered node numbers are shown. Nodes 1, 3, 5 (their old numbers, corresponding to numbering in fig. 1, are respectively 1, 5, 9) are supported and do not contain any degree of freedom.

The portrait of the factorized stiffness matrix and its division into blocks is shown in fig. 3.

$$\left(\begin{array}{cc|c|cc|cccc}
2 & & & & & & & & \\
x & 4 & & & & & & & \\
\hline
 & & 6 & & & & & & \\
\hline
 & & & 7 & & & & & \\
 & & & x & 8 & & & & \\
\hline
 & & x & & x & 9 & & & \\
 & x & x & & & x & 10 & & \\
 & x & x & & x & x & x & 11 & \\
x & x & & x & x & x & x & x & 12
\end{array}\right)$$

Fig. 3 Non-zero structure of the factorized matrix and its division into blocks

The node numbers, which correspond to the vertices of the adjacency graph (fig. 2), are located on the diagonal of the matrix. The off-diagonal non-zero entries are marked "*x*". Each non-zero element produces dense 3×3 submatrices, because each node contains 3 degrees of freedom. The supported nodes do not generate any equation and, therefore, are skipped. The following algorithm is used to create blocks (fig. 4).

```
while (j <= Neq)
   first = j; blcksize = 1
   do i ϵ Lj
      if(i != j+1 ∨ blcksize >= 120)
         last = first + blcksize-1
         j = last+1
         put first, last to descriptor of block
         break
      else
         blcksize++
         j++;
      end if
   end do
end while
```

Fig. 4 Algorithm for the creation of blocks

The algorithm processes the columns from left to right (*while* loop). For the current column *j*, it processes the non-zero structure of *Lj* (*i* ϵ *Lj*) (*do* loop), and if the next element is non-zero (*i == j+1 – else* block), it means that the block under the diagonal is dense, in which case column *j+1* is added to this block, and *blcksize* block size is incremented. This process is stopped if element with subscripts *i = j+1, j* is zero, or the size of the block exceeds the limit value 120 (*if* block). The first (*first*) and last (*last*) column numbers are pushed to the block descriptor, and further analysis is performed. In contrast to the classical approach based on the super-nodal technique, this algorithm does not allow for overshoot of zero elements in the diagonal blocks.

Blocks located below the diagonal can be completely empty, partially filled or completely filled. The completely empty blocks are skipped. For partially filled blocks, only non-zero rows are stored. A row is considered non-zero if it contains at least one non-zero element. The non-zero rows give rise to the bands in partially filled blocks. The elements are located in column wise manner within non-zero bands, as well as in completely filled blocks [11].

III. Numerical Factorization

The main idea of numerical factorization for simplification of understanding will be illustrated on example of a dense block matrix and then generalized on the sparse matrices consisting of dense rectangular blocks.

$$\begin{pmatrix} \mathbf{A}_{1,1} & & & & & \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} & & & & \\ \cdots & \cdots & \cdots & & & \\ \mathbf{A}_{jb,1} & \mathbf{A}_{jb,2} & \vdots & \mathbf{A}_{jb,jb} & & \\ \cdots & \cdots & \vdots & \cdots & \cdots & \cdots \\ \mathbf{A}_{Nb,1} & \mathbf{A}_{Nb,2} & \vdots & \mathbf{A}_{Nb,jb} & \vdots & \mathbf{A}_{Nb,Nb} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{1,1} & & & & & \\ \mathbf{L}_{2,1} & \mathbf{L}_{2,2} & & & & \\ \cdots & \cdots & \cdots & & & \\ \mathbf{L}_{jb,1} & \mathbf{L}_{jb,2} & \vdots & \mathbf{L}_{jb,jb} & & \\ \cdots & \cdots & \vdots & \cdots & \cdots & \cdots \\ \mathbf{L}_{Nb,1} & \mathbf{L}_{Nb,2} & \vdots & \mathbf{L}_{Nb,jb} & \vdots & \mathbf{L}_{Nb,Nb} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{S}_{1} & & & & & \\ & \mathbf{S}_{2} & & & & \\ \cdots & \cdots & \cdots & & & \\ & & \vdots & \mathbf{S}_{jb} & & \\ \cdots & \cdots & \vdots & \cdots & \cdots & \cdots \\ & & \vdots & & \vdots & \mathbf{S}_{Nb} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{L}^T_{1,1} & \mathbf{L}^T_{2,1} & \vdots & \mathbf{L}^T_{jb,1} & \vdots & \mathbf{L}^T_{Nb,1} \\ & \mathbf{L}^T_{2,2} & \vdots & \mathbf{L}^T_{jb,2} & \vdots & \mathbf{L}^T_{Nb,2} \\ & & \cdots & \cdots & \cdots & \cdots \\ & & & \mathbf{L}^T_{jb,jb} & \vdots & \mathbf{L}^T_{Nb,jb} \\ & & & & \cdots & \cdots \\ & & & & \vdots & \mathbf{L}^T_{Nb,Nb} \end{pmatrix} . \qquad (2)$$

Expression (2) presents the $\mathbf{L}\cdot\mathbf{S}\cdot\mathbf{L}^T$ factoring of the dense symmetric matrix, divided into rectangular blocks $\mathbf{A}_{ib,jb}$, $jb \in [1,Nb]$, $ib \in [jb, Nb]$, where *Nb* is a number of block-columns, $\mathbf{L}_{ib,jb}$ is a block of lower triangle matrix and $\mathbf{S}_{jb}$ is a diagonal block, consisting of +1 or -1 on main diagonal.

The looking-left algorithm is applied for factorization of block columns with left to right – *jb* = 1, 2, … , *Nb*. On step *jb* all block columns to the left of the column *jb* are fully factorized, and the columns to the right include elements of the original matrix.

From (2) on the basis of the known relations of the algebra of matrices should be

$$\mathbf{A}_{qb,jb} = \sum_{kb=1}^{jb-1} \mathbf{L}_{qb,kb}\mathbf{S}_{kb}\mathbf{L}^T_{jb,kb} + \mathbf{L}_{qb,jb}\mathbf{S}_{jb}\mathbf{L}^T_{jb,jb},\ qb \in [jb, Nb]. \qquad (3)$$

Hence, for *qb* = *jb* we find blocks $\mathbf{L}_{jb,jb}$ and $\mathbf{S}_{jb}$.

$$\mathbf{L}_{jb,jb}\mathbf{S}_{jb}\mathbf{L}^T_{jb,jb} = \mathbf{A}_{jb,jb} - \sum_{kb=1}^{jb-1} \mathbf{L}_{jb,kb}\mathbf{S}_{kb}\mathbf{L}^T_{jb,kb} . \qquad (4)$$

The right hand side of (4) is computed because elements of corresponding blocks are known and after this the $\mathbf{L}\cdot\mathbf{S}\cdot\mathbf{L}^T$ decomposition is produced.

Then from (3) for *qb* = *jb*+1, …, *Nb*

$$\mathbf{L}_{qb,jb}\mathbf{S}_{jb}\mathbf{L}^T_{jb,jb} = \mathbf{A}_{qb,jb} - \sum_{kb=1}^{jb-1} \mathbf{L}_{qb,kb}\mathbf{S}_{kb}\mathbf{L}^T_{jb,kb} \qquad (5)$$

and after transposition $\mathbf{L}^T_{qb,jb}$ is obtained:

$$\mathbf{L}_{jb,jb}\mathbf{S}_{jb}\mathbf{L}^T_{qb,jb} = \left(\mathbf{A}_{qb,jb} - \sum_{kb=1}^{jb-1} \mathbf{L}_{qb,kb}\mathbf{S}_{kb}\mathbf{L}^T_{jb,kb} \right)^T \qquad (6)$$

So, we need to update the right hand side of (6) and solve the linear equation set with lower triangle matrix $\mathbf{L}_{jb,jb}$, diagonal scaling presented by $\mathbf{S}_{jb}$, and multiple right hand sides.

This idea is extended to the factorization of sparse symmetric matrices **K**, divided into rectangular blocks (fig. 5).

Nb is the number of block-columns. For the core mode (CM), the initial stiffness matrix is assembled before factorization and stored in *L*. Factorization processes block-columns from left to right (p. 2, fig. 5). The current block-column *jb* is assembled if the OOC or OOC1 mode is used (p. 3). In the CM mode, this step is skipped.

For the current block-column, a parallel correction is made (p. 4) by block-columns located on the left (subscript *kb* < *jb*); only block-columns which contain non-zero blocks $\mathbf{L}_{jb,kb} \neq 0$ in block-row *ib* = *jb*, take part in this correction (*kb* ϵ *List*[*jb*]). The subscript *ib* points to non-zero blocks of column *kb* (*ib* ≥ *jb*, *ib* ϵ *L*). After this update, the diagonal block is factorized, and under-diagonal blocks are updated (p. 5). In the OOC and OOC1 modes, the factorized block-column is stored to disk; in the CM mode, this step is skipped. Non-zero structure of block-column *jb* allows predicting the set of columns located on the right (*sb* > *jb*), which will be updated by column *jb* and create the structure *List*[*sb*] (p. 6).

```
1. if(CM)
       prepare block-columns jb ϵ [1, Nb]
2. do jb = 1, Nb
3.   if(OOC ∨ OOC1)
         prepare block-column jb .
4.   Parallel correction of block-column jb:
       $\mathbf{A}_{ib,jb} = \mathbf{A}_{ib,jb} - \sum_{kb\in List[jb]} \mathbf{L}_{ib,kb}\cdot\mathbf{S}_{kb}\cdot\mathbf{L}^T{}_{jb,kb};\quad ib \ge jb, ib \in L$
5.   Factoring of block-column jb:
         $\mathbf{A}_{jb,jb} = \mathbf{L}_{jb,jb}\cdot\mathbf{S}_{jb}\cdot\mathbf{L}^T{}_{jb,jb};$
         parallel loop ib ≥ jb, ib ϵ L:
           $\mathbf{L}_{jb,jb}\cdot\mathbf{S}_{jb}\cdot\mathbf{L}^T{}_{ib,jb} = \mathbf{A}^T{}_{ib,jb} \quad\rightarrow\quad \mathbf{L}_{ib,jb};$
         end of parallel loop
         if(OOC ∨ OOC1)
           write block-column jb to disk and free RAM for
           block-row ib = jb.
6.   Add jb to List[sb], sb > jb, if block-column jb corrects
     the block-column sb.
7. end do
```

Fig. 5 Algorithm for numerical factorization

Step 4 produces about 99% floating point operations; therefore, it must be parallelized first of all. Each block-row is mapped on the same thread to avoid synchronization during writing in block $\mathbf{A}_{ib,jb}$. The load balance is achieved by taking into account the weights for each block-row and specific mapping algorithm [11]. As result, the queues *Q[ip]* with elements serving as pointers to appropriate blocks in block-columns are prepared. Each of these queues defines the processing order for blocks for threads *ip ϵ [0, np-1]*, where *np* is the number of processors. The parallel update (fig. 6). is performed in the parallel region OpenMP [11].

Several *while* loops run in the parallel region simultaneously until the appropriate queue *Q[ip]* is non-empty. For each queue, the first element is extracted and immediately removed $\left(Q[ip]/\left(\mathbf{L}_{ib,kb};\mathbf{L}_{jb,kb};kb\right)\right)$. The *dgemm* procedure from Intel MKL is used to achieve top performance during matrix multiplication [7].

```
# pragma omp parallel ( ip ϵ [0, np-1])
  while(Q[ip] is not empty)
    $\left(\mathbf{L}_{ib,kb};\mathbf{L}_{jb,kb};kb\right) \leftarrow Q[ip];\ \left(Q[ip]/\left(\mathbf{L}_{ib,kb};\mathbf{L}_{jb,kb};kb\right)\right);$
    $\mathbf{A}_{ib,jb} = \mathbf{A}_{ib,jb} - \mathbf{L}_{ib,kb}\cdot\mathbf{S}_{kb}\cdot\mathbf{L}^T{}_{jb,kb};$
  end while
end of parallel region
```

Fig. 6 Algorithm for parallel update

The virtualization approaches in the OOC and OOC1 modes are presented in fig. 7.

The OOC mode ensures a minimum of I/O operations (only the factorized column *jb* is stored), but requires the RAM amount shown in grey in fig. 7.

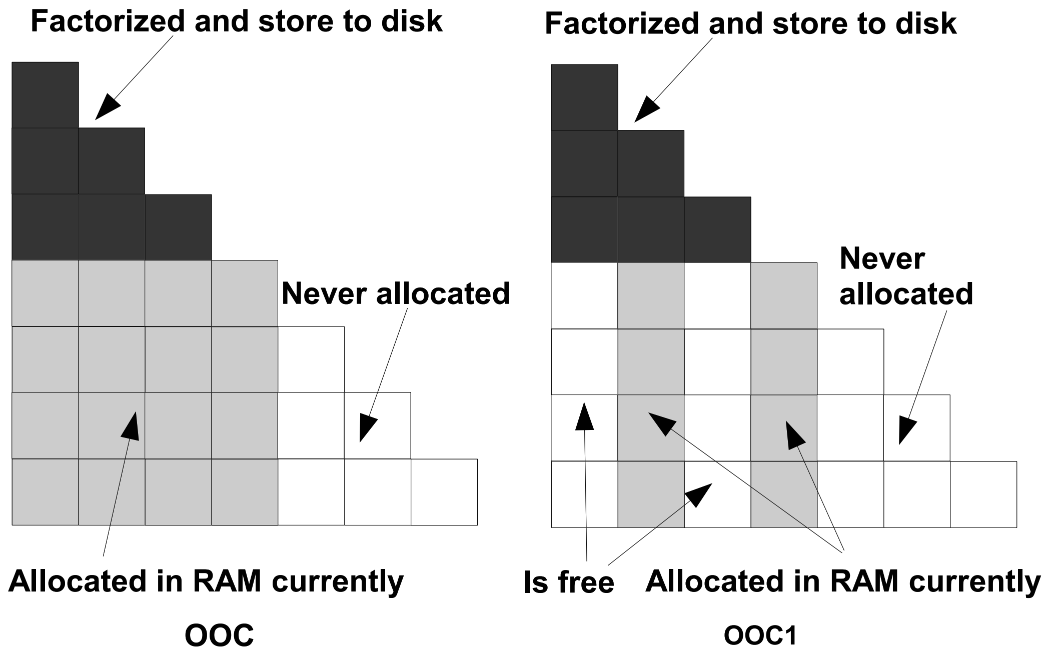
Factorized and store to disk
Never allocated
Allocated in RAM currently
OOC
Factorized and store to disk
Never allocated
Is free
Allocated in RAM currently
OOC1

Fig. 7 Virtualization scheme for the OOC and OOC1 modes

The OOC1 mode requires RAM only for two block-columns. It is essentially less than what the OOC mode requires, but during the update stage, each *kb* column must be read from disk. The number of I/O operations considerably increases, and performance and speed up decrease.

IV. NUMERICAL RESULTS

All numerical results are obtained on the following computers:

- Four-core computer Intel® Core™2 Quad CPU Q6600 @2.40 GHz, cache L1 – 32 KB, L2 – 4096 KB, RAM – DDR2 800 MT/s, 8 GB core memory, chipset – Intel P35/G33/G31, OS – Windows Vista™ Business (64-bit), Service Pack 2.
- Four-core computer AMD Phenom™ II x4 995 3.2 GHz, L1 – 4x64 KB, L2 – 4x512 KB, L3 – 6 MB, RAM – DDR3 1066 MT/s, 16 GB core memory, chipset – AMD 790X, OS – Windows Vista™ Business (64-bit), Service Pack 2.
- Workstation DELL with two processors Intel Xeon X5660 @ 2.8 GHz /3.2 GHz (2×6 = 12 cores), RAM – DDR3, 24 GB core memory, OS – Windows 7 (64-bit).
- Notebook Toshiba Satellite: two-core processor Intel® Pentium® Dual CPU T3200 @ 2.00 GHz, cache L1 – 32 KB, L2 – 1024 KB, RAM – DDR2, 667 MT/s, 4 GB core memory, chipset – Intel GL40 rev. 07, OS – Windows Vista™ Business (64-bit), Service Pack 2.

We compare PARFES with PARDISO and BSMFM – block substructure multi-frontal method [8] – [10]. Comparison of BSMFM with the multi-frontal solver of ANSYS v.11 software demonstrates that the BSMFM method is not inferior [10], [11] and, therefore, it can be selected for this comparison as a good example of a multi-frontal solver.

A large problem from the SCAD Soft collection is considered. The finite element model of a multi-storey building of 3D concrete blocks (fig. 8) contains 3 198 609 equations. METIS reordering is used for all solvers. The results obtained with Core™ 2 Quad based computer are presented in Table 1 ([11], Table 7).

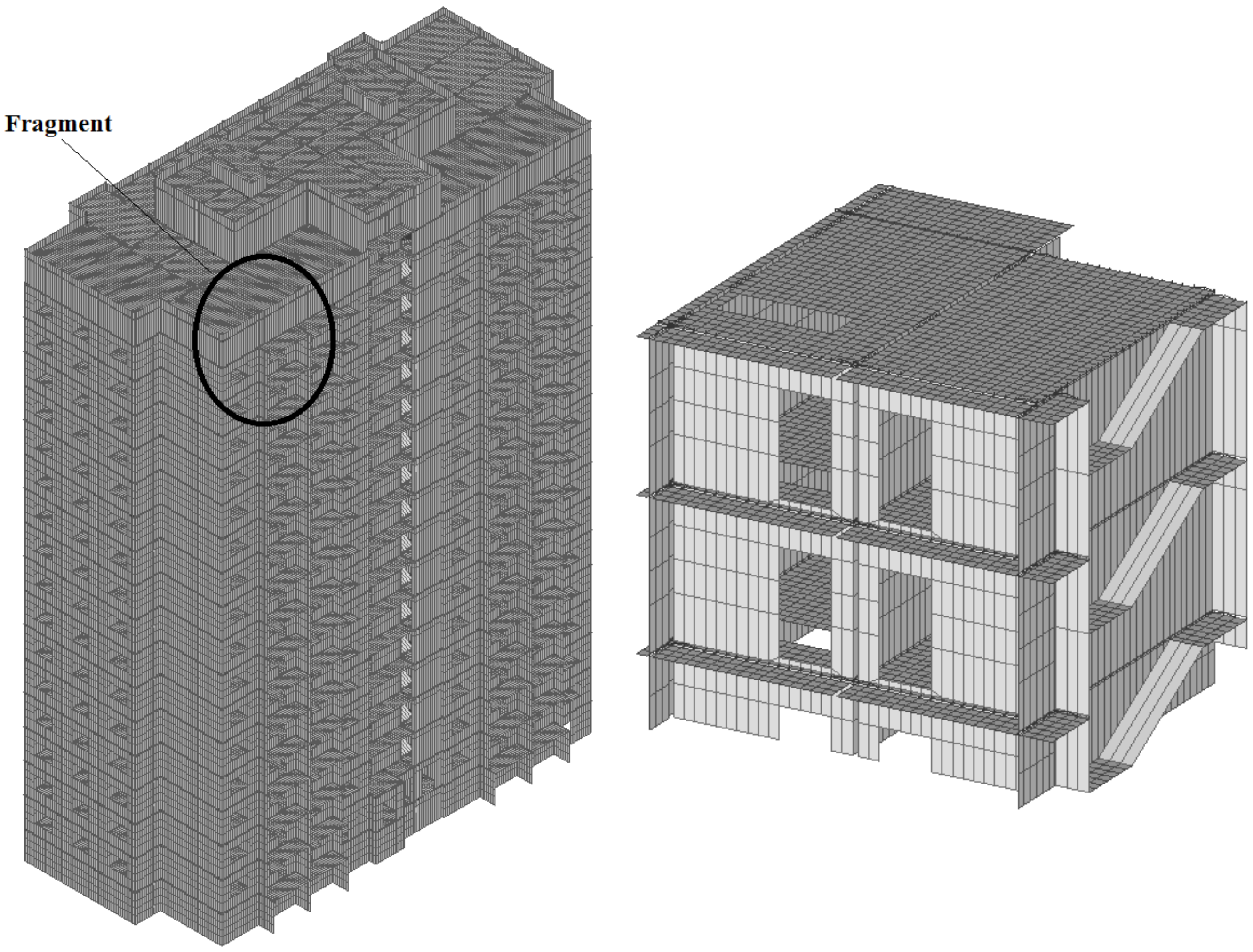

Fig. 8 Design model of multistorey building of 3D concrete blocks (3,198,609 equations) and it fragment

TABLE 1.
DURATION OF THE SOLVING PHASES USING A CORE^{TM} 2 QUAD BASED COMPUTER ([11], TABLE 7).

Method	L, MB	Analysis stage, s	Numerical factorization, s				Phase of forward-back substitutions, s	
			Number of processors				Number of processors	
			1	2	3	4	1	4
PARFES	12,186	23.6	1,190	802	594	475	804	526
PARDISO	10,662	61.4	Numer. factorization phase: error = -11					
BSMFM	10,869	9.0	2,011	1,482	1,286	1,232	497	

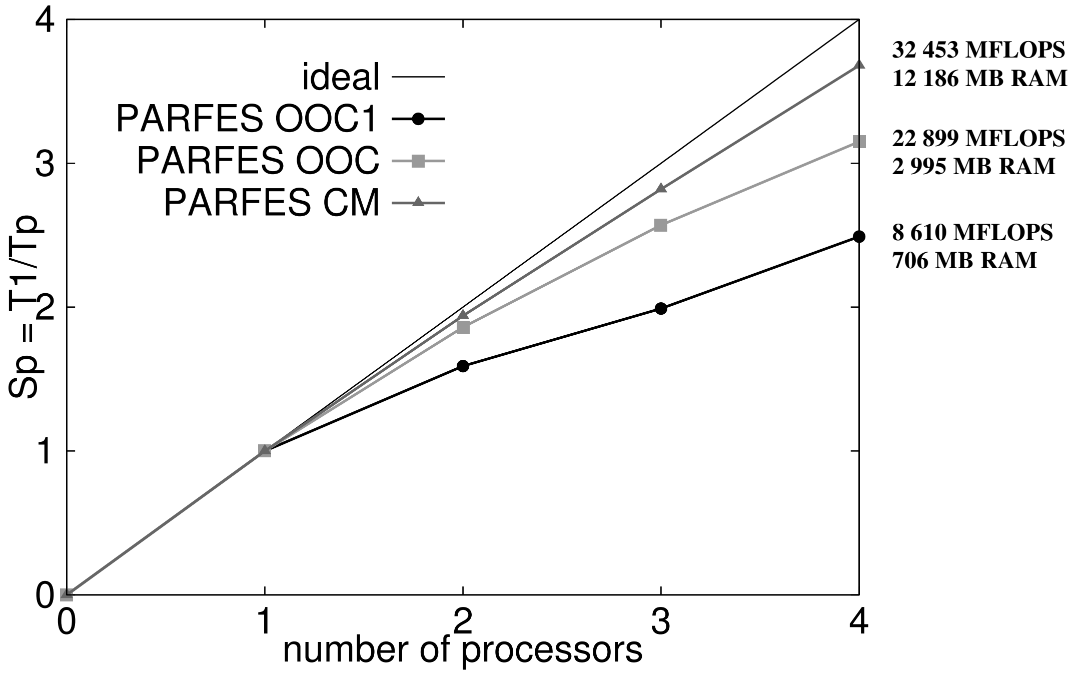

Fig. 9 Speed up of PARFES in the CM, OOC and OOC1 modes on the AMD Phenom™ II 4 × 995 based computer (16 GB RAM)

All solvers run in the OOC mode. The size of the factorized stiffness matrix in MB is shown in the second column. The duration of sparse matrix analysis stage is shown in the third column.

PARDISO failed, producing error code -11. PARFES, run on four cores, solves this problem approximately 2.5 times faster than BSMFM.

The results of solving with the AMD Phenom™ II 4 × 995 based computer are presented in fig. 9.

The performance and amount of required RAM is presented for 4 cores. PARFES demonstrates a good speed up and achieves the best performance in the CM mode. In the OOC mode, the speed up decreases, but the required amount of RAM is approximately 3 GB, instead of 12 GB. In the OOC1 mode, the speed up and performance are the lowest, but as little as 706 MB is sufficient. Therefore, in the OOC1 mode, this large-scale problem can be solved even as a 32-bit application. Both OOC and OOC1 modes demonstrate a stable speed up within four cores. The time of numerical factoring in the CM mode on a single processor is 729 s for PARFES and 697 s for PARDISO; on four processors – 197 s for PARFES and 208 s for PARDISO. On a single processor, PARDISO is slightly faster than PARFES, but on four processors, PARFES is slightly faster than PARDISO. PARDISO results in a slightly smaller size of the factorized matrix, but PARFES demonstrates a slightly higher performance (32,453 MFLOPS – PARFES on four cores and 26,181 MFLOPS – PARDISO). PARFES exhibits an essentially less sparse matrix analysis stage time than PARDISO.

The comparison of solvers on the DELL workstation is presented in Table 2. All solvers run in core mode. PARFES and PARDISO exhibit a very similar numerical factoring time. The minimal time is achieved with 11 processors. The best time of BSMFM is about 9.6 times worse compared to PARDISO and PARFES.

The speed up for all solvers is presented in fig. 10. Since the processors support a turbo boost mode, the ideal speed up (*ideal)* is not achievable. The *id_tb_fun* curve is a quadratic parabola, passing through points (0, 0), (1, 1) and (12, 10.5). The ordinate of the last point is calculated as 12·2.8/3.2=10.5, where 3.2 is the maximal clock frequency and 2.8 – the minimal one. The *ideal-tb* curve presents the results of the test program, which produces intense computations without reading of RAM. The id_tb_fun and *ideal-tb* values are very close. The speed up of PARFES is very close to the one of PARDISO. BSMFM demonstrates a poor speed up for the given problem.

TABLE 2.
DURATION OF THE SOLVING PHASES USING A DELL WORKSTATION (12 CORES).

N of processors	PARFES		PARDISO		BSMFM	
	Analysis, s	Numerical factoring, s	Analysis, s	Numerical factoring, s	Analysis, s	Numerical factoring, s
1	16.9	654	31.06	596	13	1,406
2	16.9	337.8	23.59	305.3	13	1,015
3	16.9	232.1	25.33	208.6	13	869
4	16.9	177.9	23.26	163.3	13	793
5	16.9	145.7	23.79	135.7	13	786
6	16.9	125.6	25.68	116	13	777
7	16.9	110.6	23.11	100.9	13	772
8	16.9	100.5	23.43	90.83	13	807
9	16.9	92.5	23.98	83.85	13	770
10	16.9	86.3	23.71	79.6	13	796
11	16.9	82.1	29.86	77.36	13	825
12	16.9	87.5	28.58	78.45	13	839

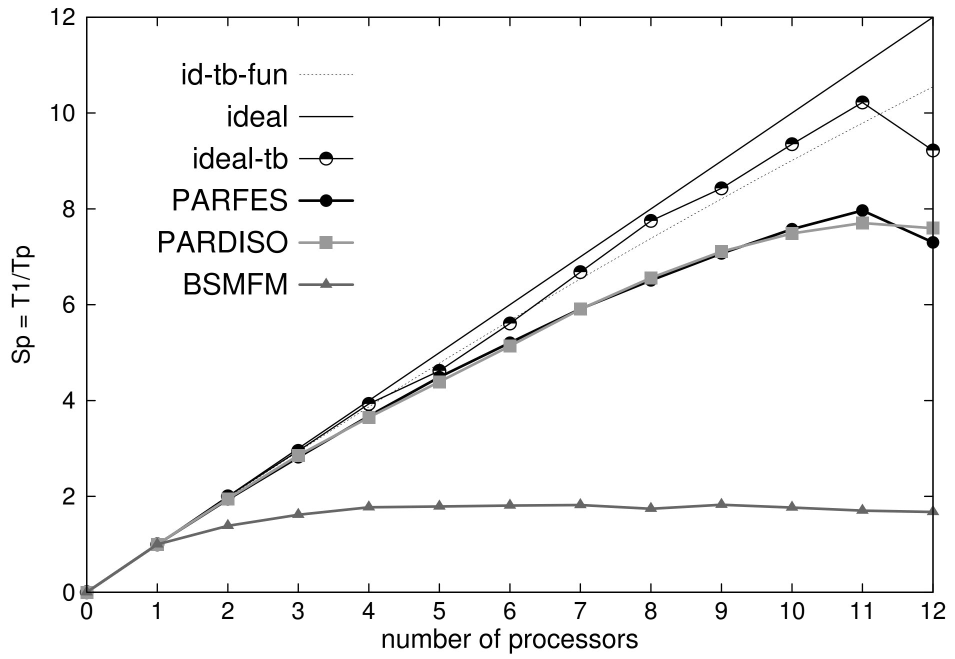

Fig.10 Speed up of solvers on the 12-core DELL workstation

The last result is obtained using the Toshiba Satellite laptop computer. PARFES runs as a 32-bit application in the OOC1 mode. The duration of the sparse matrix analysis stage is 26 s, stiffness matrix assembly 3 min 16 s, numerical factoring 51 min 51 s, solving 19 min 54 s, and total solution time is 75 min 32 s. This problem cannot be solved by BSMFM solver as a 32-bit application; only a 64-bit application ran successfully.

V. Conclusion

PARFES can solve large-scale problems on workstations as well as on desktop and laptop computers. When the factorized matrix is fully stored in the core memory, performance and speed up are close to those of PARDISO. If the scale of the problem exceeds the RAM capacity, PARFES automatically chooses the OOC or OOC1 mode. The performance and speed up decrease in that case, but the problem can be solved even on a laptop computer. PARFES is a good alternative to multi-frontal solvers.

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.

[2] J. W. Demmel, *Applied Numerical Linear Algebra*. Philadelphia: SIAM, 1997.

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

[5] G. H. Golub, C. F. Van Loan, *Matrix Computations*. Third edition. John Hopkins University Press, 1996.

[6] 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.

[7] K. Goto, R. A. Van De Geijn, "Anatomy of High-Performance Matrix Multiplication," *ACM Transactions on Mathematical Software*, 34 (3), pp. 1–25, 2008.

[8] S. Fialko, "A Sparse Shared-Memory Multifrontal Solver in SCAD Software," In *Proc. of the International Multiconference on Computer Science and Information Technology*; October 20-22, 2007, Wisła, Poland, 3, 2008, ISSN 1896-7094, ISBN 978-83-60810-14-9, IEEE Catalog Number CFP0864E-CDR, pages 277–283, 2008. (http://www.proceedings2008.imcsit.org/pliks/47.pdf).

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

[10] S. Fialko, *The direct methods for solution of the linear equation sets in modern FEM software*. Moscow: SCAD SOF , 2009. (in Russian).

[11] S. Fialko, "PARFES: A method for solving finite element linear equations on multi-core computers," *Advances in Engineering Software*, 41, pp. 1256–1265, 2010.

[12] Intel® Math Kernel Library Reference Manual. Document Number: 630813-029US. http://software.intel.com/sites/products/documentation/hpc/mkl/mklman/index.htm. Accessed 2.05.2012.

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

[14] O. Schenk, K. Gartner, "Two-level dynamic scheduling in PARDISO: Improved scalability on shared memory multiprocessing systems," *Parallel Computing*, 28, pp. 187–197, 2002.