Computer Assisted Methods in Engineering and Science, **20**: 15–41, 2013.

TWENTY YEARS OF THE CAMES

Minimizing the memory usage with parallel out-of-core multi-frontal direct solver

Maciej Paszyński
AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
al. A. Mickiewicza 30, 30-059 Krakow, Poland
e-mail: maciej.paszynski@agh.edu.pl

This paper presents the out-of-core solver for three-dimensional multiphysics problems. In particular, our study focuses on the three-dimensional simulations of the linear elasticity coupled with acoustics. The out-of-core solver is designed with three principles in mind. First, to store the dense matrices associated with the nodes of the elimination tree with blocks related to nodes of the mesh, where many degrees of freedom may be located in the case of multiphysics computations with high order polynomials. The second principle is to minimize the memory usage. This is obtained by dumping out all local systems from the entire elimination tree to the disk during the elimination stage. The local systems are reutilized later during the backward substitution stage. The third principle is that the communication in the parallel version of the out-of-core solver occurs through the parallel file system. The memory usage of the solver is compared against the state- of- the- art MUMPS solver.

Keywords: multi-frontal direct solver, finite element method, out-of-core, parallel simulations, multiphysics.

1. Introduction

This paper presents the parallel out-of-core multi-frontal direct solver for multiphysics problems solved by using *hp*-finite element method (*hp*-FEM). Multiphysics problems deliver a non-uniform structure of the stiffness matrix resulting from utilizing different number of equations on different parts of the mesh. The solver can work with 3D problems, where finite elements of different kind, such as tetrahedral, hexahedral or prism elements can be utilized. The solver is dedicated to *hp*-adaptive computations, where the polynomial orders of approximation as well as the number of equations can vary locally for different finite element edges, faces, and interiors. Additional complication is the fact the *h*-refined meshes require usage of pyramid elements, since tetrahedral elements are broken into four tetrahedral and two pyramid elements.

The solver is the generalization of the ideas utilized in the two-dimensional solver [1–3] for three-dimensional multiphysics problems. The main challange when switching to three dimensions is the memory usage.

The out-of-core solver is designed with three principles in mind. First, to store the dense matrices associated with the nodes of the elimination tree in block structure, with blocks related to nodes of the mesh, where many degrees of freedom may be located in the case of multiphysics computations and high polynomial orders of approximation. The second principle is to minimize the memory usage. It is obtained by dumping out all local systems from the entire elimination tree to the disk during the elimination stage. The local systems are reutilized later during the backward substitution stage. The third principle is that the communication in the parallel version of the out-of-core solver occurs through the parallel file system. In other words, the Schur complement matrices are not

exchanged via MPI send/receive protocol, rather one processor dumps out the matrix to the disc, and the other processor dumps it in.

The solver is interfaced with multiphysics extension of the *hp*-FEM application *hp3d* [5, 6]. The out-of-core version of the solver is based on the in-core version of the algorithm presented in [7]. The solver is tested on a challenging computational problem: the acoustics of the human head, modeled as linear elasticity coupled with linear acoustics [8].

The reason why we focus on the direct solver instead of considering iterative solvers is the following: multiphysics problems usually generate huge linear systems of equations, which are not well conditioned; thus, the applicability of iterative solvers is typically limited. In addition, most iterative solvers exhibit a lack of robustness (in presence of high-contrast materials, elongated elements, and so on [9]). Moreover, iterative solvers may be slower than direct solvers when a problem with several right- hand sides needs to be solved, as it occurs in the case of goal-oriented adaptivity (it is necessary to solve the dual problem [10]) and inverse problems (when computing the Jacobian and Hessian matrices).

The memory usage of the out-of-core solver is compared against the state-of-the art MUMPS solver [11–13].

We do not compare the execution times, since our solver strongly utilizes the parallel file system and its execution time is one order of magnitude larger than the MUMPS solver, thus we use less memory.

The motivation behind the comparison to the MUMPS solver is the following. In paper [14], Fig. 2 compares PARDISO and MUMPS solver of the multiphysics problem considered in this paper. If we compare the execution time, which is not the point of this paper, the PARDISO is little faster than MUMPS. However, for the largest problem considered in [14], for 3.2 million degrees of freedom PARDISO crashes because of memory allocation problems. But, if we compare the memory usage, MUMPS is more efficient than PARDISO, and it does not crash for the largest problem.

In paper [15], there is a detailed comparison of solvers, including MUMPS and PARDISO, for positive definite problem, for ANALYSIS, FACTORIZATION and SOLUTION phases, in terms of execution time and memory usage. In general, PARDISO performs faster than MUMPS, but it utilizes more memory, compare Fig. 7 in [15]. From the experiments performed in these papers it follows that MUMPS is a representative state of the art solver in terms of memory usage, and comparison of our solver to MUMPS indeed makes sense.

In addition, it makes no sense to compare the solver presented in this paper against commercial solvers like ANSYS. This is because the computations presented in this paper utilize special hierarchical shape functions introduced in Sec. 2 of this paper, allowing for local p refinement. The commercial solvers like ANSYS do not have the same hierarchical shape functions, and thus the structure of the matrix as well as the sparsity of the system generated by ANSYS is different and comparison to ANSYS indeed makes no sense.

The out-of-core solver presented in this paper utilizes block structure of the matrix following the nodes of the mesh, with different number of degrees of freedom over finite element vertices, edges, faces and interiors, as it occurs in multiphysics computations. This allows performing the management of the matrices on the level of nodes of the mesh in the multiphysics application. This approach is similar to the one presented in [16–18] where the sizes of the blocks depend on types of finite elements assembled in this block and constrains which are imposed on it (applied to it).

2. Problem formulation

The out-of-core solver algorithm will be tested on two three-dimensional model problems: the Fichera model problem being the three-dimensional generalization of the two-dimensional L-shape domain problem [6, 23, 24] and the linear elasticity coupled with acoustics [6, 8, 25]. The problems are defined in Appendix A.

The variational formulations for both problems have been discretized with hierarchical shape functions allowing for p-adaptive computations with $hp3d$ code [6].

1D element space shape functions over $\widehat{K} = [\,0,\,1\,]$ are defined as follows:

$$\widehat{\chi}_1(\xi) = 1 - \xi, \tag{1}$$

$$\widehat{\chi}_2(\xi) = \xi, \tag{2}$$

$$\widehat{\chi}_3(\xi) = (1-\xi)\,\xi, \tag{3}$$

$$\widehat{\chi}_l(\xi) = (1-\xi)\,\xi\,(2\xi-1)^{l-3} \quad \text{for} \quad l = 4,\ldots,p+1. \tag{4}$$

In other words, 1D element of order p has $p-1$ shape functions.

Then, we extend the definition to 3D element space shape functions over $\widehat{K} = [0,1]^3$ with uniform p.

We define vertex shape functions as follows:

$$\widehat{\phi}_1(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_1(\xi_3), \tag{5}$$

$$\widehat{\phi}_2(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_1(\xi_3), \tag{6}$$

$$\widehat{\phi}_3(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_1(\xi_3), \tag{7}$$

$$\widehat{\phi}_4(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_1(\xi_3), \tag{8}$$

$$\widehat{\phi}_5(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_2(\xi_3), \tag{9}$$

$$\widehat{\phi}_6(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_2(\xi_3), \tag{10}$$

$$\widehat{\phi}_7(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_2(\xi_3), \tag{11}$$

$$\widehat{\phi}_8(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_2(\xi_3), \tag{12}$$

and the edge shape functions as

$$\widehat{\phi}_{9,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+j}(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_1(\xi_3) \quad j = 1,...,p-1, \tag{13}$$

$$\widehat{\phi}_{10,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_{2+j}(\xi_2)\,\widehat{\chi}_1(\xi_3) \quad j = 1,...,p-1, \tag{14}$$

$$\widehat{\phi}_{11,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+j}(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_1(\xi_3) \quad j = 1,...,p-1, \tag{15}$$

$$\widehat{\phi}_{12,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_{2+j}(\xi_2)\,\widehat{\chi}_1(\xi_3) \quad j = 1,...,p-1, \tag{16}$$

$$\widehat{\phi}_{13,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+j}(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_2(\xi_3) \quad j = 1,...,p-1, \tag{17}$$

$$\widehat{\phi}_{14,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_{2+j}(\xi_2)\,\widehat{\chi}_2(\xi_3) \quad j = 1,...,p-1, \tag{18}$$

$$\widehat{\phi}_{15,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+j}(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_2(\xi_3) \quad j = 1,...,p-1, \tag{19}$$

$$\widehat{\phi}_{16,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_{2+j}(\xi_2)\,\widehat{\chi}_2(\xi_3) \quad j = 1,...,p-1, \tag{20}$$

$$\widehat{\phi}_{17,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad j = 1,...,p-1, \tag{21}$$

$$\widehat{\phi}_{18,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad j = 1,...,p-1, \tag{22}$$

$$\widehat{\phi}_{19,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad j = 1,...,p-1, \tag{23}$$

$$\widehat{\phi}_{20,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad j = 1,...,p-1; \tag{24}$$

the face shape functions as

$$\widehat{\phi}_{21,i,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+i}(\xi_1)\,\widehat{\chi}_{1+j}(\xi_2)\,\widehat{\chi}_1(\xi_3) \quad i=1,...,p-1, \quad j=1,...,p-1, \tag{25}$$

$$\widehat{\phi}_{22,i,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+i}(\xi_1)\,\widehat{\chi}_{1+j}(\xi_2)\,\widehat{\chi}_2(\xi_3) \quad i=1,...,p-1, \quad j=1,...,p-1, \tag{26}$$

$$\widehat{\phi}_{23,i,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+i}(\xi_1)\,\widehat{\chi}_1(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad i=1,...,p-1, \quad j=1,...,p-1, \tag{27}$$

$$\widehat{\phi}_{24,i,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+i}(\xi_1)\,\widehat{\chi}_2(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad i=1,...,p-1, \quad j=1,...,p-1, \tag{28}$$

$$\widehat{\phi}_{25,i,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_1(\xi_1)\,\widehat{\chi}_{2+i}(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad i=1,...,p-1, \quad j=1,...,p-1, \tag{29}$$

$$\widehat{\phi}_{26,i,j}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_2(\xi_1)\,\widehat{\chi}_{2+i}(\xi_2)\,\widehat{\chi}_{2+j}(\xi_3) \quad i=1,...,p-1, \quad j=1,...,p-1, \tag{30}$$

and, finally, the interior shape functions as

$$\widehat{\phi}_{27,i,j,k}(\xi_1,\xi_2,\xi_3) = \widehat{\chi}_{2+i}(\xi_1)\,\widehat{\chi}_{2+j}(\xi_2)\,\widehat{\chi}_{2+k}(\xi_3)$$
$$i=1,...,p-1, \quad j=1,...,p-1, \quad k=1,...,p-1. \tag{31}$$

In other words, a 3D element with uniform order p has 1 shape functions at each vertex, $p-1$ shape functions at each edge, $(p-1)(p-1)$ shape functions at each face, and $(p-1)(p-1)(p-1)$ shape functions at interior.

For the definition of shape function over tetrahedral and prism elements we refer to [6].

The hierarchical shape functions are utilized over the computational mesh to discretize the variational formulations (A8)–(A10) or (A12)–(A19), respectively. The resulting global stiffness matrix is symmetric in both cases.

The interface to the solver algorithm requires a sequence of element frontal matrices with block structure described in the next section, with shape functions restricted to particular finite elements, grouped to mesh nodes, with indices denoting the global numbering of mesh nodes. The interface to the solver algorithm is summarized in Appendix B.

3. The block structure of the matrices

The matrices managed by the solver are stored in the hypermatrix module. The hypermatrix is a matrix of matrices.

```
      type hyper_matrix
c
c     type: -2 = rectangular matrix of hyper-matrices
c           -1 = lower-triangular matrix of hyper-matrices
c            0 = zero (null) matrix
c            1 = dense lower-triangular matrix
c            2 = dense rectangular matrix
c     mrow: number of rows
c     ncol: number of columns
c     D:    pointer to memory for dense block
c     H:    recursive pointer to hyper_matrix blocks
      integer :: type
      integer :: mrow,ncol
      double precision,   pointer :: D(:,:)
      type(hyper_matrix), pointer :: H(:,:)
c
      endtype hyper_matrix
```

The exemplary code constructing the 2×2 hypermatrix with 2×2 blocks filled with zeros is listed below:

```
type(hyper_matrix) :: matrix
matrix%mrow=2; matrix%ncol=2; matrix%type=-2
allocate(matrix%H(2,2)); nullify(matrix%D)
do i=1,2
  do j=1,2
    allocate(matrix%H(i,j)%D(2,2)); nullify(matrix%H(i,j)%H)
    matrix%H(i,j)%D(1:2,1:2)=0.d0
    matrix%H(i,j)%mrow=2; matrix%H(i,j)%ncol=2; matrix%H(i,j)%type=2
  enddo
endddo
```

This hierarchical structure of the matrix allows for working on the level of blocks associated with particular nodes of the mesh. Having the matrix stored in blocks associated with nodes of the mesh is particularly important in multiphysics computations as well as in the case of the usage of high polynomial orders of approximations.

This is so because the merging of matrices and elimination of rows can be managed on the level of blocks, and not necessarily on the level of degrees of freedom. For example, the interior nodes with order p have $(p\text{-}1)^3$ degrees of freedom, and for high polynomial orders of approximation this simplifies the analysis phase of the solver (the construction of the elimination tree with order of elimination of nodes).

4. Domain decomposition and construction of the elimination tree

The computational mesh is partitioned recursively into subdomains, see Fig. 1. It is done be executing the following recursive algorithm:

```
// each element identified with global id
nbeg = 1; nend = number_of_elements;
// level of recursive partition
nlevel = 1;
call partition(nbeg,nend,nlevel)
recursive subroutine partition
                    (ibeg,iend,ilevel)
// construct the adjacency graph
    // and partition into two
call partition_elements_into_2_with_metis
                                (ibeg,iend)
ihalf = (iend+ibeg)/2
// we have two subdomains
    [ibeg,ihalf] and [ihalf+1,iend]
if(ihalf>ibeg)then
      nbeg = ibeg;
      nhalf = ihalf;
      nlevel = ilevel+1;
      // recursive call for the 1st subdomain
      call partition(nbeg,nhalf,nlevel)
endif
if(ihalf<iend)then
      nend = iend;
      nhalf = ihalf+1;
      nlevel = ilevel+1;
      // recursive call for the 2nd subdomain
      call partition(nhalf,nend,nlevel)
endif
```

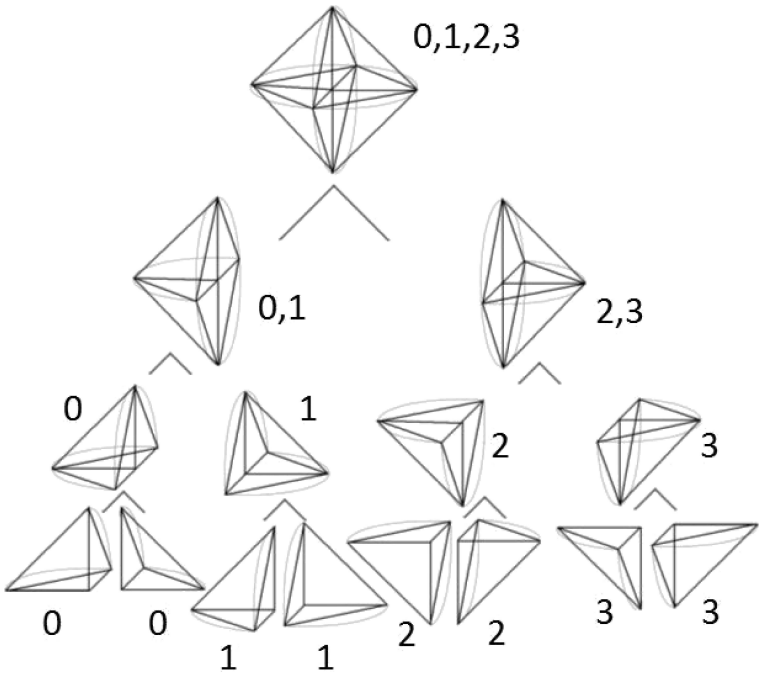

Fig. 1. Four levels of the elimination tree representing recursive partition of the computational domain, attributed with four processors.

The finite elements from the mesh are numbered. The partition subroutine is called with list of all elements. It calls METIS library that generates the graph representation of the connectivities between finite elements, and finds equal partition into two sub-graphs (sub-domains). The procedure is repeated recursively for each subdomain, until we obtain sub-domains with single elements. The resulting binary tree is the elimination tree for the solver algorithm. The elimination tree nodes are attributed by the processor numbers in such a way that parent node inherits processors numbers from the son nodes.

The above algorithm executed on the human head model results in the elimination tree with the number of leaves equal to the number of elements, that is, 19 288.

This algorithm is actually the implementation of the nested-dissection algorithm [19–21]. The nested dissection method as mentioned in the paper is built by recursive calls to METIS library, with the graph representing the sub-part of the mesh for the partition, with edges labeled by polynomial orders of approximation over element faces. This algorithm is suitable for complex geometries like the model of the human head considered in the paper.

The interface to the solver algorithm, summarized in Appendix B, enables to implement other methods for construction of the ordering and elimination tree, like the minimum degree algorithm [22] that may perform better than nested dissections for some class of problems.

5. Sequential out-of-core solver algorithm

The sequential out-of-core solver browses the elimination tree in the post-order, starting from the left bottom leaf. In the leaf node, the fully assembled internal nodes are eliminated, leaving the Schur complement of interface nodes. Later, the Schur complement for the left bottom leaf is dumped out and deallocated. The solver moves to the right neighbor of the left bottom leaf, computes the Schur complement at the node, and dumps it out to the disc. Next, the solver moves to the parent node, and constructs the new system based on two already dumped out systems. In the new system fully assembled nodes are identified and eliminated, and the newly created Schur complement is also dumped out to the disc. The process is repeated until we reach the root of the elimination tree. The algorithm is summarized below:

```
function out_of_core_sequential(node, proc, iret)
if node is a leaf then
   generate local system assigned to node
   find internal nodes at node
   eliminate internal nodes at node
else
   loop through son_node's
     if proc is assigned to son_node then
      iret =0
      call out_of_core_sequential(son_node,proc,iret)
      if iret==1 then return
      compute schur1 complement at son_node
      dumpout the system from son_node
      deallocate the system at son_node
      if proc is 1st proc
          assigned to 2nd son of node then
        BUFFER = schur1; deallocate schur1;
        iret =1; return
      else if proc is 1st proc
          assigned to node then
        dumpout schur1; deallocate(schur1)
        proc_org = proc
        iret =1; proc =1st proc
                  assigned to 2nd son of node
        call out_of_core_sequential(son_node,proc,iret)
        iret =0;proc = proc_org
        schur2 = BUFFER; deallocate(BUFFER)
      endif
     endif
   end loop
   if proc is 1st proc assigned to node then
     dumpin previously dumpout schur1
     merge schur1,2 into new system at node
     deallocate(schur1,schur2)
     find nodes to eliminate at node
   endif
   compute schur complement at node
   dumpout the system at node
endif
```

6. Schur complement computations

In this section, we present the implementation of the Schur complement computations over a local dense system of equations associated with a single node of the elimination tree. Let us assume we have the following 2×2 system:

$$\mathtt{A}(1,1) * \mathtt{x}(1) + \mathtt{A}(1,2) * \mathtt{x}(2) = \mathtt{b}(1),$$

$$\mathtt{A}(2,1) * \mathtt{x}(1) + \mathtt{A}(2,2) * \mathtt{x}(2) = \mathtt{b}(2),$$

where `A(i,j)` are square matrices, `x(i)`, `b(i)` are vectors. The Schur complement computations can be implemented in the following steps:

1) $\mathtt{A}(1,1) = \mathtt{L} * \mathtt{U}$;

2) $\mathtt{A}(1,2) = (\mathtt{L} * \mathtt{U})^{-1} * \mathtt{A}(1,2)$;

3) $\mathtt{b}(1) = (\mathtt{L} * \mathtt{U})^{-1} * \mathtt{b}(1)$;

4) $\mathtt{A}(2,2) = \mathtt{A}(2,2) - \mathtt{A}(2,1) * \mathtt{A}(1,2)$;

5) $\mathtt{b}(2) = \mathtt{b}(2) - \mathtt{A}(2,1) * \mathtt{b}(1)$.

After these operations, the Schur complement is stored in `A(2,2)` and `b(2)`. The particular steps of the algorithm are implemented with calls to level 2 and 3 BLAS routines. The detailed implementation of these steps is provided in Appendix C.

7. Numerical results for the sequential solver with the Fichera model problem

The sequential out-of-core solver has been tested on the Fichera model problem. The tests have been performed on the single node of the Linux cluster ATARI from the AGH Department of Computer Science, with 16 cores with 2.4 GHz and 16 GB of RAM. In these computations only single core has been used. The tests concerned the comparison of the memory usage and execution time with MUMPS solver. The Fichera model problem has been solved for various polynomial orders of approximations, for various sizes of the mesh structure. The numerical results are summarized in Tables 1–5.

From the performed experiments we can draw the following conclusions. Our solver utilizes less memory than MUMPS. The memory usage of our solver with respect to the MUMPS memory

Table 1. Number of degrees of freedom for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3 \ldots 8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$	$p=7$	$p=8$
$4 \times 4 \times 4$	665	1981	4401	8261	13 897	21 645	31 841
$8 \times 8 \times 8$	4401	13 897	31 841	60 921	103 825	163 241	241 857
$16 \times 16 \times 16$	31 841	103 825	241 857	467 441	802 081	1 267 281	1 884 545

Table 2. Execution time in seconds of the out-of-core solver for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3 \ldots 8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$	$p=7$	$p=8$
$4 \times 4 \times 4$	1.1	2.21	3.34	6.66	13.32	24.4	43.96
$8 \times 8 \times 8$	62.07	90.67	162.91	374.94	718	1329	2235
$16 \times 16 \times 16$	3913	5787	8218	14 007	23 765	42 339	

Table 3. Memory usage in MB of the out-of-core solver for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3 \ldots 8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p=2$	$p=3$	$p=4$	$p=5$	$p=6$	$p=7$	$p=8$
$4 \times 4 \times 4$	0.16	0.22	0.36	0.68	1.51	2.81	5.4
$8 \times 8 \times 8$	22	28	42	90	147	327	476
$16 \times 16 \times 16$	1592	2189	3001	5073	8216	14 060	

Table 4. Execution time in seconds of the MUMPS solver for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3...8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$
$4 \times 4 \times 4$	0.041	0.083	0.2	0.66	2.44	7.89	31.44
$8 \times 8 \times 8$	0.29	1.1	4.38	15.1	46.03	163	351
$16 \times 16 \times 16$	4.9	35.4	169	619	2649		

Table 5. Memory usage in MB for the MUMPS solver for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3...8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$
$4 \times 4 \times 4$	1	4	12	32	84	171	347
$8 \times 8 \times 8$	9	44	141	381	888	2074	3916
$16 \times 16 \times 16$	121	614	1927	4972	10 947		

usage is summarized in Table 6. The memory usage of our solver is between 9–22 times better than MUMPS memory usage in this case. For the comparison, we examine the memory usage reported by MUMPS (TOTAL MEMORY) which is defined as amount of MB used to store all non-zero entries during the factorization. We compare the memory usage reported by MUMPS to the amount of RAM memory used by our solver to store the factors. Thus, the total amount of memory for MUMPS and our solver is actually higher, since we do not count the memory occupied by the rest of the *hp3d* application.

Table 6. MUMPS memory usage in MB divided by our solver memory usage in MB for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3...8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$
$4 \times 4 \times 4$	1976285	15.64492	14.57924	15.46828	18.92397	19.98195	22.76821
$8 \times 8 \times 8$	10.03034	9.525678	9.16336	9.656079	10.37117	12.40738	13.0933
$16 \times 16 \times 16$	9.031625	9.177692	8.663048	9.736382	9.515527		

It is worth mentioning that MUMPS with METIS ordering crashes for $n = 16 \times 16 \times 16$ for $p = 7$ and $p = 8$ due to a lack of memory, while our solver computes $n = 16 \times 16 \times 16$ with $p = 7$ and crashes for $p = 8$.

With respect to the execution time, MUMPS is faster than our solver, which is summarized in Table 7. But the goal of this research was to minimize the memory usage, with intensive out-of-core operations. From the comparison of the execution times it can be implied that MUMPS is significantly faster than our solver for low polynomial orders of approximation. This is the price to

Table 7. Our solver execution time in seconds divided by MUMPS execution time in seconds for the Fichera problem for polynomial orders of approximation varying from $p = 2, 3...8$ and mesh sizes $n = 2 \times 2 \times 2$ or $n = 4 \times 4 \times 4$ or $n = 8 \times 8 \times 8$ (without 1/8 of the mesh for the Fichera singularity).

	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$
$4 \times 4 \times 4$	26.82927	26.62651	16.7	10.09091	5.459016	3.092522	1.398219
$8 \times 8 \times 8$	214.0345	82.42727	37.19406	24.83046	15.59852	8.153374	6.367521
$16 \times 16 \times 16$	798.5714	163.4746	48.62722	22.62843	8.97131	12.16638	

be paid for the block structure of the matrices, where, for low polynomial orders, the processing of the blocks slows down the solver. For high polynomial orders of approximation the difference goes down, it is between 1–12 times slower than MUMPS.

8. Parallel out-of-core solver algorithm

The parallel version of the out-of-core solver algorithm browses the elimination tree level by level, from the level of leaves to the root level, using multiple processors. The communication between processors is actually happening through parallel file system. Two son nodes compute the two Schur complements and dump them out to the disc. The parent node constructs new system by reading the two already dumped out Schur complements. The synchronization of the read/write operations is performed by sending a single flag through MPI message from son nodes to the parent node.

When the number of processors is less than number of leaves in the elimination tree, the parallel out-of-core solver switches to sequential out-of-core solver whenever number of processors assigned to the node is equal to one.

The parallel out-of-core solver algorithm is summarized below:

```
function out_of_core(node, proc)
 loop through son_node's
  if proc is assigned to son_node then
   if number of processors assigned to son_node is 1 then
    call out_of_core_sequential(son_node, proc)
   else
    call out_of_core(son_node,proc)
   endif
   rewrite schur complement from system at son_node into schur1
   dumpout the system at node
   deallocate the system at son_node
   if proc is 1st proc assigned to 2nd son of node then
     dumpout schur1; deallocate(schur1)
     dest_proc = 1stproc assigned to 2ndson of node
     mpi_send("dumped out",dest_proc)
   else if proc is 1st proc assigned to node then
     proc_source = 1st proc assigned to 2nd son of node
     mpi_recv("dumped out",proc_source)
   endif
  endif
 end loop
 if proc is 1st proc assigned to node then
  merge schur1 withschur2 previously dumped out at disc
   into new system at node
  deallocate schur1
  find nodes to eliminate at node
 endif
 compute schur complement at node
endif
```

9. Numerical results

The numerical experiments concern three computational meshes summarized in Table 8. The first mesh is the coarse mesh with uniform order of approximation $p = 2$ on edges, faces and interiors of tetrahedral finite elements and with polynomial orders of approximation $p = 3$ on prism elements.

Table 8. Computational meshes summarized in the numerical experiments.

	First mesh	Second mesh	Third mesh
Number of finite elements	19 288	19 288	154 304
Number of degrees of freedom	125 754	287 079	1 058 622
Number of non-zero entries	15 321 155	40 215 303	119 942 591
Maximum memory out-of-core solver	1.2 GB	2 GB	19 GB
Memory MUMPS	922 MB	3 GB	35.9 GB

There are 19 288 finite elements and 125 754 degrees of freedom. This is three-dimensional problem and what really matters is not the size of the matrix but the number of non-zero entries in the matrix. The matrix resulting from the first mesh contains 15 millions non-zero entries.

The second mesh is the same coarse mesh but after uniform p refinement. The mesh has the uniform order of approximation $p = 3$ over tetrahedral elements and $p = 4$ over prism elements. The number of finite elements is the same, but the number of degrees of freedom increases to 287 079. Also the sparsity of the new matrix implies 40 millions of non-zero entries.

The third mesh is obtained after global h refinement of the first mesh. The mesh still contains the uniform order of approximation $p = 2$ over tetrahedral elements and $p = 3$ over prism elements. The number of finite elements after uniform h refinement increases up to 154 304. The number of non-zero entries is huge and is equal to 119 millions.

Notice that number of non-zero entries presented in Table 8 concerns the sequence of element frontal matrices submitted to the solver algorithm (or to MUMPS). It does not contain the new non-zero factors generated during the factorization process. This is why $119\,942\,591 \times 8 = 915$ MB and it is not 19 GB for the third mesh. The maximum memory for the out-of-core solver is computed as the amount of RAM occupied by the factors generated during the factorization. For our solver, the maximum amount of RAM is equal to the disc's space occupied by a largest file with largest local dense system processed by the solver algorithm since we report amount memory used to store double precision numbers from frontal matrices and dumped out to disc in a formatted way.

The memory usage for the MUMPS solver is reported as provided by the MUMPS for TOTAL MEMORY used during the factorization. It is equal to the amount of RAM occupied by the factors generated during the factorization process.

Figures 2, 3 and 4 present memory usage of the sequential version of the out-of-core solver executed over the first, second and third meshes. The memory usage is related to the sizes of local systems of equations processed by the solver in consequtive nodes of the elimination tree browsed in post order. The maximum memory usage for the first mesh is equal to 1.2 GB, for the second mesh is equal to 2.0 GB, and for the third mesh is equal to 19 GB. The state-of-the-art MUMPS

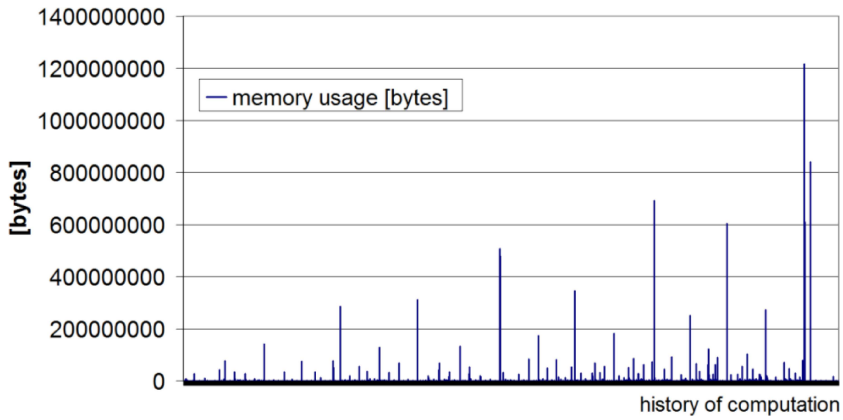

Fig. 2. Memory usage up to 1.2 GB during execution of the sequential out-of-core solver on the first mesh.

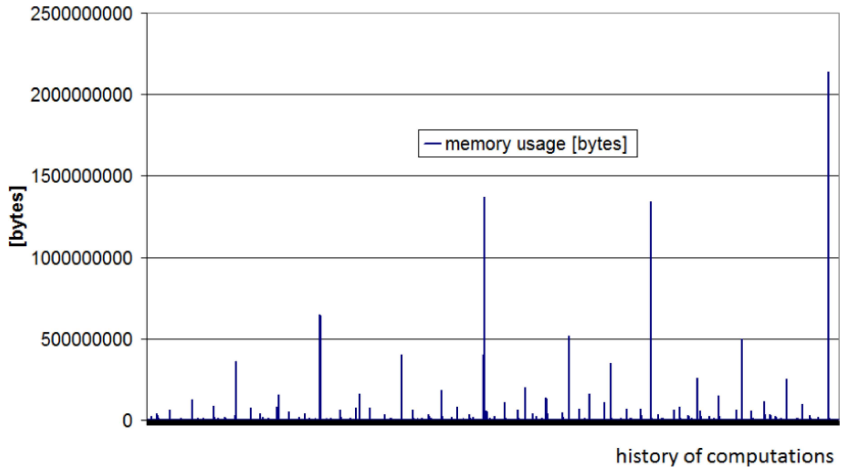

Fig. 3. Memory usage up to 2.0 GB during execution of the sequential out-of-core solver on the second mesh.

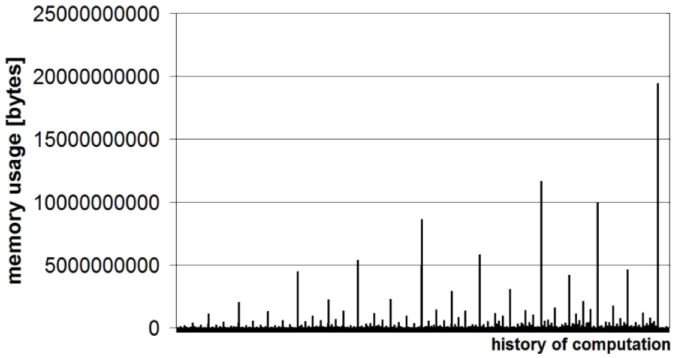

Fig. 4. Memory usage up to 19 GB during execution of the sequential out-of-core solver on the third mesh.

solver requires 922 MB for the first mesh, 3 GB for the second mesh and 35.9 GB for the third mesh (compare Table 8). This implies that our solver uses less memory than the MUMPS solver.

The multi-frontal solver algorithm generates a tree of dense matrices. The dense matrices are stored at nodes of the elimination tree. The solver presented in this paper uses the block structure of the sub-matrices, with blocks related to mesh nodes. The size of the dense system grows when we go up the tree. The smallest systems are located at leaves of the elimination tree, and the size is growing towards the root node. The Figs. 5–7 report the number of mesh nodes in the local systems within the elimination tree, when browsing the tree level by level, from root to the leaves. The level 1 corresponds to the root node; the level 17 corresponds to the leaf nodes. From Figs. 5–7 we can draw interesting conclusion that the maximum size of the system is actually located on the third level of the elimination tree. Each local system from the elimination tree has some number of mesh nodes fully assembled, and some number of nodes not fully assembled yet. The fully assembled nodes can be eliminated; the not fully assembled nodes are merged for the next level systems. These nodes for the next level are called the interface nodes. In Figs. 5–7 it is also assumed that the top problem has only interface nodes, but this time all of these nodes can be eliminated. The recursive calls to METIS library has been utilized to generate the elimination trees which statistics are presented in Figs. 5–7.

Figures 5, 6 and 7 present the number of nodes over particular levels of the elimination tree, for the first, second and third mesh. The figures allow to predict the sizes of the local systems of equations encountered while browsing the elimination trees. The elimination tree has 19 288 leaves so it contains 17 levels. The level 17 corresponds to the leaves, while level 1 corresponds to the root of the elimination tree. Notice that the root level corresponds to the single cross-section of the

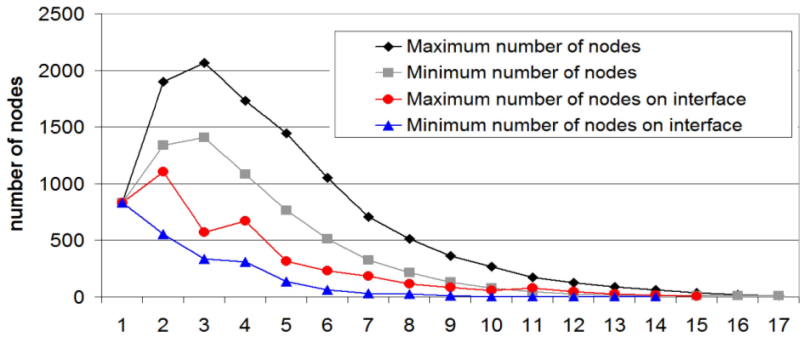

Fig. 5. Statistics for the ordering obtained by recursive calls to METIS package, for the first mesh, for polynomial order of approximation uniformly set to $p = 2$ over tetrahedral elements and to $p = 3$ on prism elements. Maximum and minimum number of nodes estimated for each level of the elimination tree. Maximum and minimum number of nodes on the interface (not fully assembled yet) estimated for each level of the elimination tree.

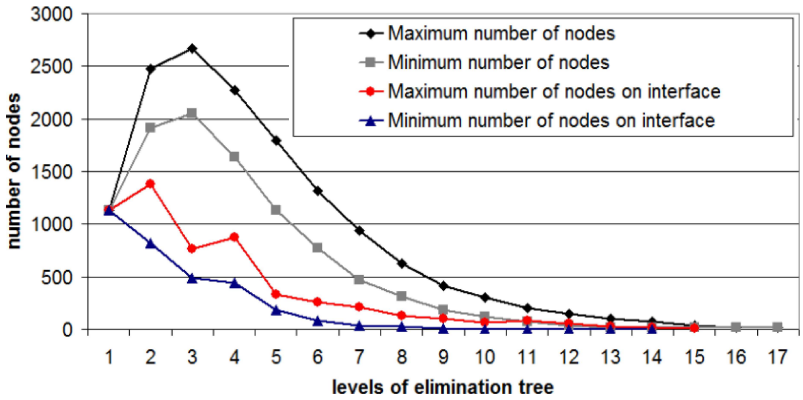

Fig. 6. Statistics for the ordering obtained by recursive calls to METIS package, for the second mesh, for polynomial order of approximation uniformly set to $p = 3$ over tetrahedral elements and to $p = 4$ on prism elements. Maximum and minimum number of nodes estimated for each level of the elimination tree. Maximum and minimum number of nodes on the interface (not fully assembled yet) estimated for each level of the elimination tree.

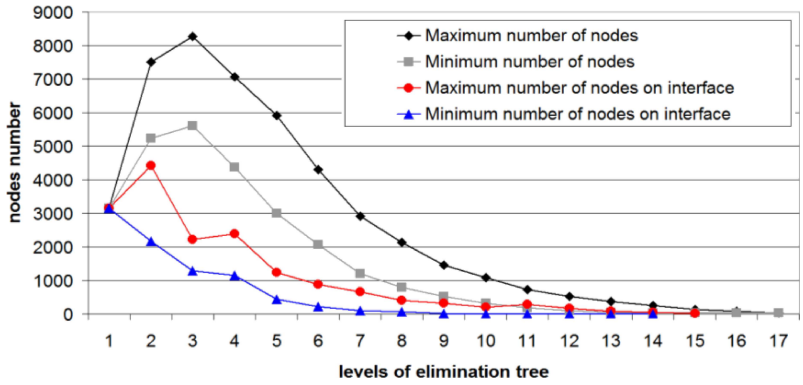

Fig. 7. Statistics for the ordering obtained by recursive calls to METIS package, for the third mesh, obtained by breaking each tetrahedral element from the third mesh into four son tetrahedrals and two piramids, and each prism element into eight new prism elements. Maximum and minimum number of nodes estimated for each level of the elimination tree. Maximum and minimum number of nodes on the interface (not fully assembled yet) estimated for each level of the elimination tree.

computational domain, the second level corresponds to the interface of the quarters of the domain, while the third level corresponds to the 1/8 parts of the domain. It can be read from the figures that the maximum sizes of systems of equations are present at the third level.

The local systems of equations stored at nodes of the elimination tree are "almost" dense, in a sense that they may contain zeros, not related to the topology of the mesh, but rather to the structure of the element frontal matrices resulting from the structure of the variational formulation and the shape function used. The sparsities of these systems are reported in Figs. 8–10. The zeros in the systems follow from the tensor product structure of the shape functions used. The sparsity pattern increases when we increase the polynomial order of approximation. Despite the fact that we have some sparsities in the systems, we utilize block structure of the matrix and dense algebra routines BLAS level 2 and 3 to proceed with the Schur complement computations. Please refer to Appendix C for more details.

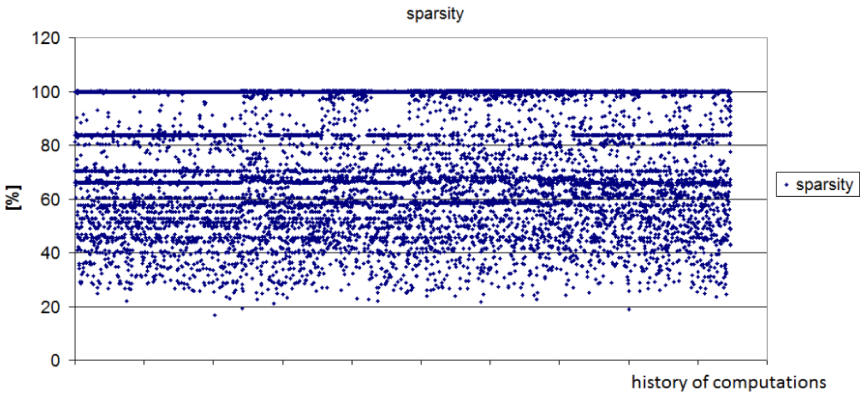

Fig. 8. Sparsity of the matrices for the first mesh during the sequential execution of the out-of-core solver (during browsing the elimination tree in the post order).

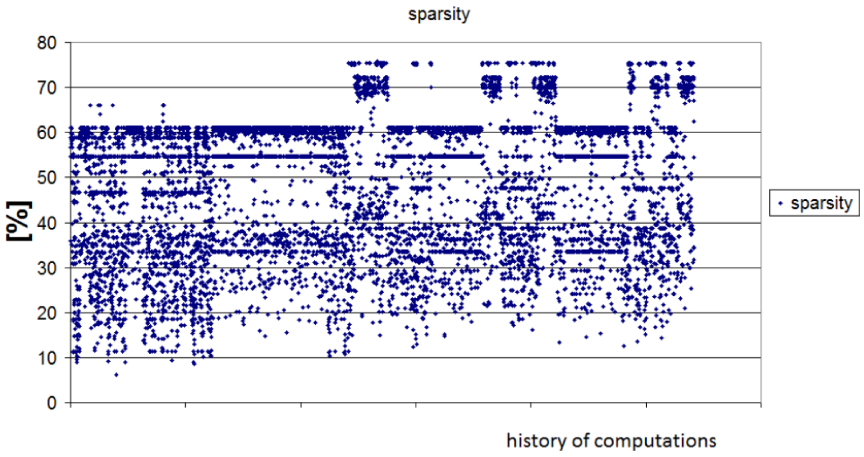

Fig. 9. Sparsity of the matrices for the second mesh during the sequential execution of the out-of-core solver (during browsing the elimination tree in the post order).

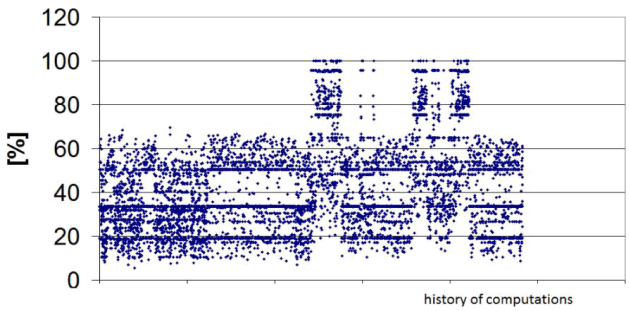

Fig. 10. Sparsity of the matrices for the third mesh during the sequential execution of the out-of-core solver (during browsing the elimination tree in the post order).

Figures 8, 9 and 10 denote the sparsities of the local systems of equations, from all nodes of the elimination tree, for the first, second and third meshes. Each dot corresponds to sparsity of a single system from a single node of the elimination tree. From the figures it follows that the sparsity of the systems goes down when we perform global p or h refinements. This implies that the additional memory saving may be obtained by utilizing compressed matrix storage.

The speed-up and efficiency of our parallel solver reported in Figs. 12 and 14 are defined as follows. The speed-up is defined as $E = T_1/Tp$, and the efficiency is defined as $E = T_1/(pTp)$, where T_1 is the execution time of the sequential solver, p is the number of processors, Tp is the execution time for p processors.

Figures 11 and 13 present the maximum execution time taken from all procesors participating in the parallel computations. Figures 15–17 present the execution times for particular processors. Execution time is non-uniform, since the elimination tree is a binary tree, with all processors assigned to the branches of the tree; however, when the algorithm processes the tree all the way to the root, at some level of the elimination tree the number of tree nodes is less than the number of processors. In such a case, some processors become idle.

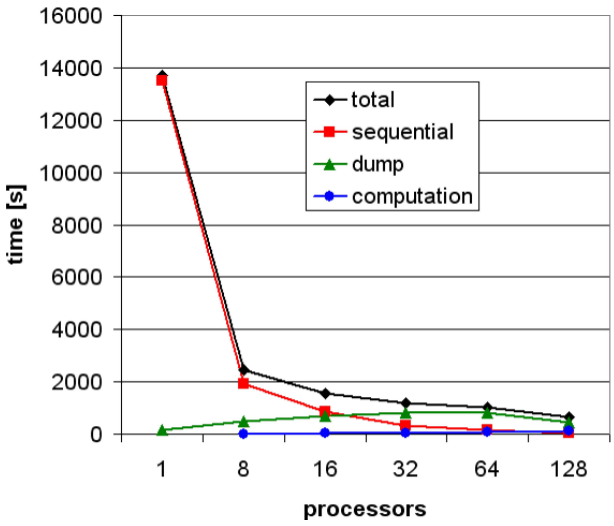

Fig. 11. Execution time for parallel out-of-core solver for the first mesh with uniform $p = 2$. The black line entitled "total" denotes the total execution time for different number of processors. The red line entitled "sequential" denotes the time spent by each processor to process its local branch of the elimination tree. The line includes also time spent by each processor on dumping out and in the Schur complement on local SCRATCH directory on LONESTAR cluster. The green line entitled "dump" denotes the time spent by each processor on "dumping out" and "dumping in" matrices with the Schur complements to be exchanged between processors through WORK parallel file system on LONESTAR cluster. The blue line entitled "computation" denotes the time spent during processing of the parallel part of the elimination tree by parallel MUMPS solver used to perform the partial Schur complements.

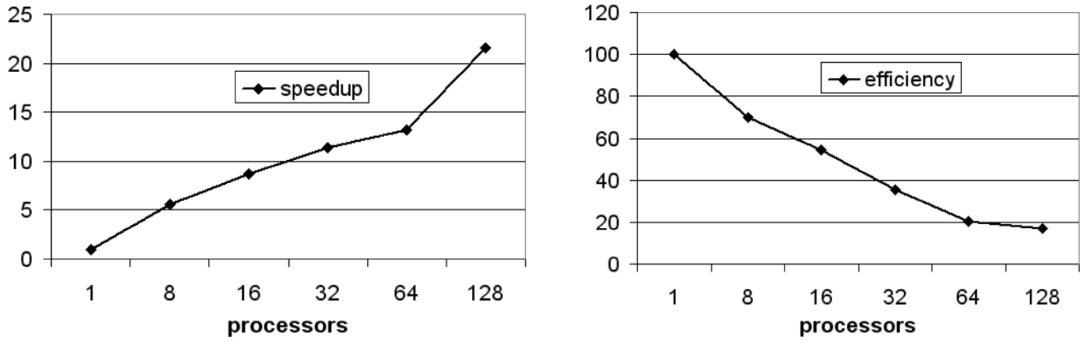

Fig. 12. Speed-up and efficiency of the parallel out-of-core solver for the first mesh with $p = 2$.

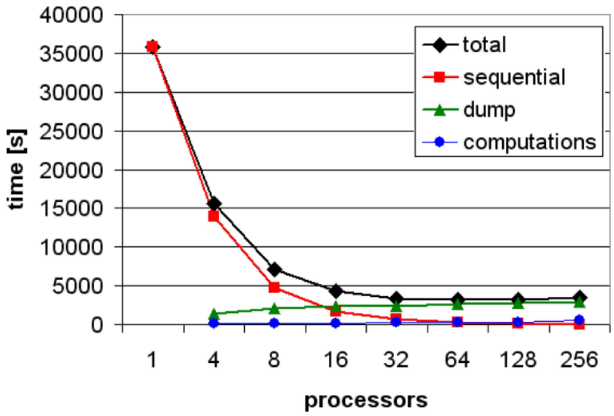

Fig. 13. Execution time for parallel out-of-core solver for the second mesh with uniform $p = 3$. The black line entitled "total" denotes the total execution time for different number of processors. The red line entitled "sequential" denotes the time spent by each processor to process its local branch of the elimination tree. The line includes also time spent by each processor on dumping out and in the Schur complement on local SCRATCH directory on LONESTAR cluster. The green line entitled "dump" denotes the time spent by each processor on dumping out and dumping in matrices with the Schur complements to be exchanged between processors through WORK parallel file system on LONESTAR cluster. The blue line entitled "computation" denotes the time spent during processing of the parallel part of the elimination tree by parallel MUMPS solver used to perform the partial Schur complements.

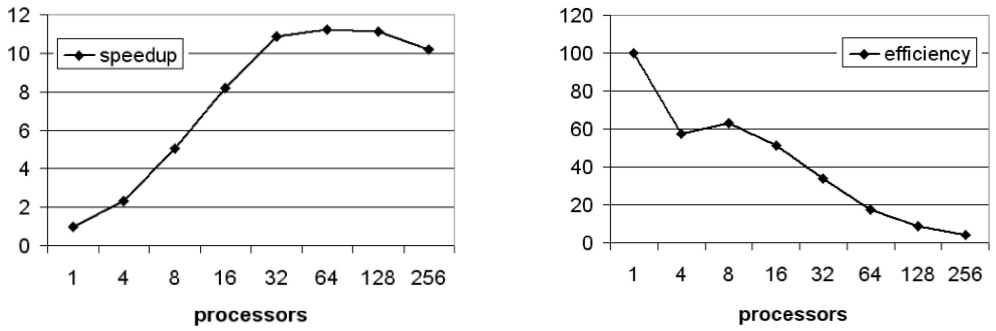

Fig. 14. Speed-up and efficiency of the parallel out-of-core solver for the second mesh with $p = 3$.

Figure 18 presents the execution times for the third problem, and the reason why there are not times reported for low number of processors is the following: while conducting the third problem for low number of processors (less than 8) we ran out of memory and it was not possible to solve the problem with small number of computing nodes without distribution of data into many cluster nodes.

The following figures summarize the execution times of the solver over the first, second and third meshes.

From the results for the first mesh summarized in Figs. 11 and 12 we can draw the following conclusions:

- The efficiency of the parallel out-of-core solver presented in Fig. 12 is about 60 percent on 16 processors, 40 procent on 32 processors, and it goes down to 20 percent on 64 and 128 processors.
- The execution times of the particular parts of the parallel out-of-core solver listed in Fig. 11 imply that starting from 32 processors, the dumping out and in matrices into the parallel file system becomes the dominating part of the solver algorithm and this is the bottleneck for improving the efficiency of the solver

From the results for the second mesh summarized in Figs. 13–17 we can draw the following conclusions:

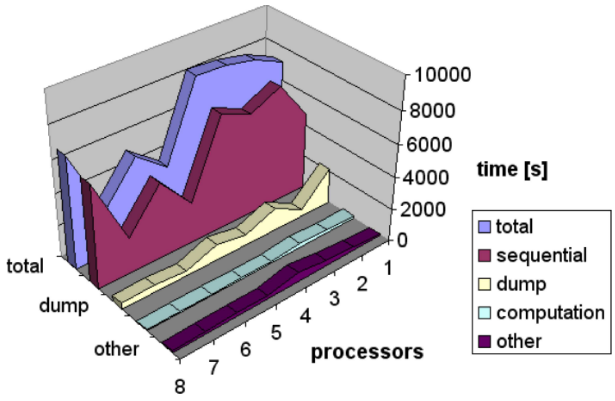

Fig. 15. Profile for 8- processor parallel execution of the out-of-core solver for the second mesh with polynomial order of approximation $p = 3$. The blue line entitled "total" denotes the total execution time for different number of processors. The light pink line entitled "sequential" denotes the time spent by each processor to process its local branch of the elimination tree. The line includes also time spent by each processor on dumping out and in the Schur complement on local SCRATCH directory on LONESTAR cluster. The yellow line entitled "dump" denotes the time spent by each processor on dumping out and dumping in matrices with the Schur complements to be exchanged between processors through WORK parallel file system on LONESTAR cluster. The blue line entitled "computation" denotes the time spent during processing of the parallel part of the elimination tree by parallel MUMPS solver used to perform the partial Schur complements. The dark pink line entitled "other" denotes time spent by processor on other routines including management of nodes connectivity.

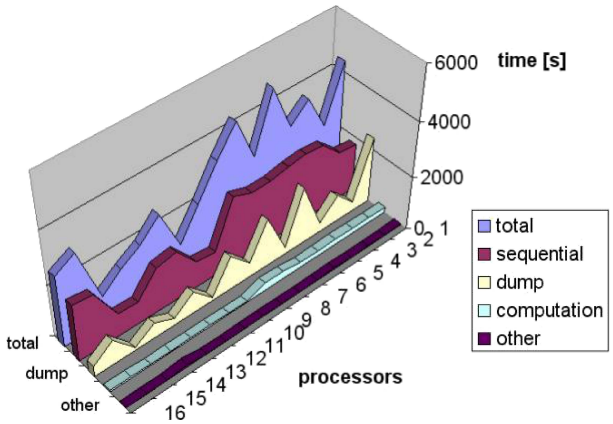

Fig. 16. Profile for 16- processor parallel execution of the out-of-core solver for the second mesh with polynomial order of approximation $p = 3$. The blue line entitled "total" denotes the total execution time for different number of processors. The light pink line entitled "sequential" denotes the time spent by each processor to process its local branch of the elimination tree. The line includes also time spent by each processor on dumping out and in the Schur complement on local SCRATCH directory on LONESTAR cluster. The yellow line entitled "dump" denotes the time spent by each processor on dumping out and dumping in matrices with the Schur complements to be exchanged between processors through WORK parallel file system on LONESTAR cluster. The blue line entitled "computation" denotes the time spent during processing of the parallel part of the elimination tree by parallel MUMPS solver used to perform the partial Schur complements. The dark pink line entitled "other" denotes time spent by processor on other routines including management of nodes connectivity.

- The efficiency of the parallel out-of-core solver presented in Fig. 14 is about 50–60 percent up on 16 processors, 35 procent on 32 processors, and it goes down to 20 percent on 64 and 10 percent on 128 and 256 processors.

- The execution times of the particular parts of the parallel out-of-core solver listed in Fig. 13 imply that starting from 32 processors, the dumping out and in matrices into the parallel file system becomes the dominating part of the solver algorithm and this is the bottleneck for improving the efficiency of the solver.

- The careful analysis of the components of parallel execution for 8, 16 and 64 processors presented in Figs. 15, 16 and 17 confirms the thesis that the dominating parts of the out-of-core solver algorithm are dumping out the matrices into the disc.

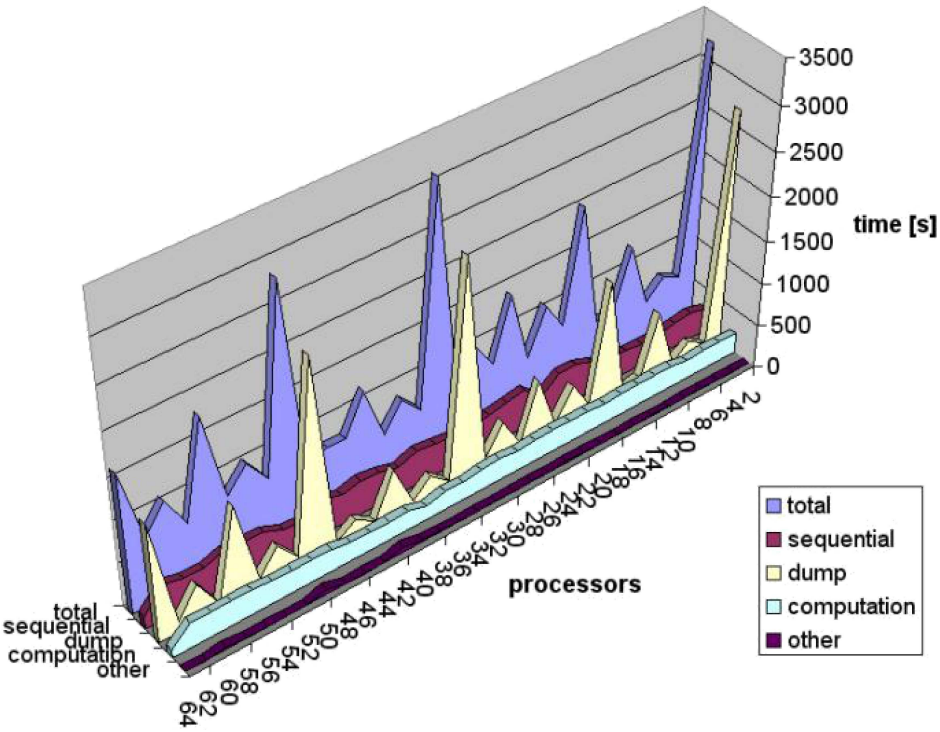

Fig. 17. Profile for 64- processor parallel execution of the out-of-core solver for the second mesh with polynomial order of approximation $p = 3$. The blue line entitled "total" denotes the total execution time for different number of processors. The light pink line entitled "sequential" denotes the time spent by each processor to process its local branch of the elimination tree. The line includes also time spent by each processor on dumping out and in the Schur complement on local SCRATCH directory on LONESTAR cluster. The yellow line entitled "dump" denotes the time spent by each processor on dumping out and dumping in matrices with the Schur complements to be exchanged between processors through WORK parallel file system on LONESTAR cluster. The blue line entitled "computation" denotes the time spent during processing of the parallel part of the elimination tree by parallel MUMPS solver used to perform the partial Schur complements. The dark pink line entitled "other" denotes time spent by processor on other routines including management of nodes connectivity.

From the results for the third mesh summarized in Fig. 18 as well as from Fig. 4 presenting the memory usage we can draw the following conclusions:

- The third mesh is difficult to solve, bacause of huge amount of required memory. Actually, the largest local problem requires 19 GB of memory to solve.
- Solution of the third problem requires huge amount of memory. LONESTAR Linux cluster consists in computational nodes with 24 GB of RAM with 12 cores per node. The maximum amount of memory that can be utilized per node is 24 GB when we execute a single MPI task per node. The memory usage required by the third problem results in necessity of utilizing entire node per processor. Each node has 12 cores, so it implies that when we need to solve the problem on 64 processors, we actually need to allocate $64 \times 12 = 768$ processors (64 nodes with 12 cores per node).
- We believe that further scalability of the solver could be possible, but we had access to maximum 64 nodes with 12 cores = 768 processors. It should be emphasized that current version of the solver utilizes only one core during the LU factorization, and further improvement of the scalability of the solver can be obtained by implementing multi-core LU factorization.

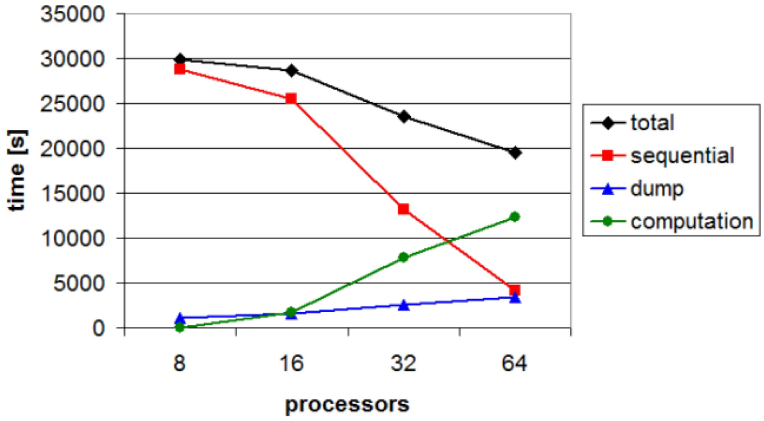

Fig. 18. Execution time for parallel out-of-core solver for different number of processors for the third mesh.

10. Conclusions and future work

The solver obtained 50–60 percent efficiency on up to 16 processors, then the efficiency went down to 35–40 percent on 32 processors, and went down to 20 and less percent on 64 and more processors. The main reason for loosing the efficiency is the performance of the parallel file-system, since the solver is performing multiple concurrent "dump outs" and "dump ins" of local system from particular nodes of the elimination tree.

The future work will include

- Further minimizing the memory usage by storing the systems of equations from particular nodes of the elimination tree in distributed manner.
- Further improving the scalability of the solver by utilizing multicore algorithms for partial LU factorizations performed by the solver.
- Further improving the scalability of the solver by minimizing the sizes of the dumped out matrices. This can be done by switching to compressed matrix storage as well as by dumping out and in a single local system of equations into multiple files, as it is stored in distributed manner.

Appendix A

The Appendix presents the details of the two numerical problems utilized for testing of the solver algorithm.

L-shape domain model problem

The L-shape domain problem is a model academic problem formulated by Babuška in 1986 [23, 24], to test the convergence of the *p*- and *hp*- adaptive algorithms. The problem consists in solving the temperature distribution over the L-shape domain, presented in Fig. 19 with fixed zero temperature in the internal part of the boundary, and the Neumann boundary condition prescribing the heat transfer on the external boundary. There is a single singularity in the central point of the domain (the gradient of temperature goes to infinity at point O), so the accurate numerical solution requires a sequence of adaptations in the direction of the central point. The problem can be summarized as follows:

find $u : R^2 \supset \Omega x \rightarrow u(x) \in R$ the temperature distribution such that

$$\sum_{i=1}^{2} \frac{\partial^2 u}{\partial x_i^2} = 0 \text{ in } \Omega \tag{A1}$$

with boundary conditions

$$u = 0 \quad \text{on} \quad \Gamma_D, \tag{A2}$$

$$\frac{\partial u}{\partial n} = g \quad \text{on} \quad \Gamma_N, \tag{A3}$$

with n being the unit normal outward to $\partial\Omega$ vector, and

$$g\left(r,\theta\right) = r^{2/3}\sin^{2/3}\left(\theta + \frac{\Pi}{2}\right) \tag{A4}$$

is defined in the radial system of coordinates with the origin point O presented in Fig. 19. The formula (4) is actually based on the exact solution for the L-shape problem.

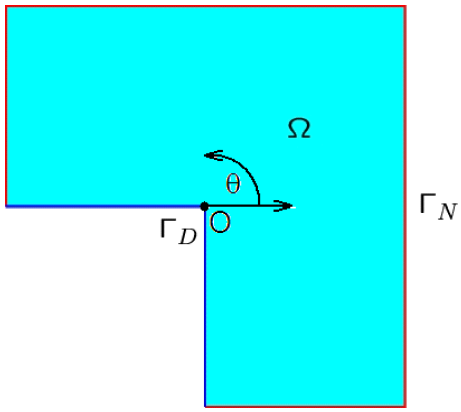

Fig. 19. The L-shape domain.

Fichera problem

The first problem considered in this paper is the Fichera model problem. It is the generalization of the L-shape domain problem into three dimensions. It can be summarized in the following way: find $u : R^3 \supset \Omega x \rightarrow u(x) \in R$ the temperature distribution over the domain presented in Fig. 20 such that

$$\sum_{i=1}^{3} \frac{\partial^2 u}{\partial x_i^2} = 0 \quad \text{in} \quad \Omega \tag{A5}$$

with boundary conditions

$$u = 0 \quad \text{on} \quad \Gamma_D, \tag{A6}$$

$$\frac{\partial u}{\partial n} = g \quad \text{on} \quad \Gamma_N \tag{A7}$$

with n being the unit normal outward to $\partial\Omega$ vector, and g is the exact solution of the L-shape problem.

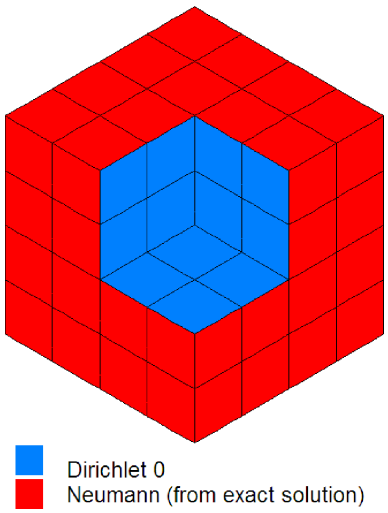

Fig. 20. Domain for the Fichera problem.

The weak variational formulation is obtained by multiplying by test function $v \in V$ and integrating by parts to get

$$b(u,v) = l(v) \qquad \forall v \in V, \tag{A8}$$

$$b(u,v) = \int_{\Omega} \sum_{i=1}^{3} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} dx, \tag{A9}$$

$$l(v) = \int_{\Gamma_N} g v dS, \tag{A10}$$

where

$$V = \left\{ v \in L^2(\Omega) : \int_{\Omega} \|v\|^2 + \|\nabla v\|^2 \, dx < \infty : tr(v) = 0 \text{ on } \Gamma_D \right\}. \tag{A11}$$

Linear elasticity coupled with acoustics

The second problem considered in this paper is the challenging multiphysics problem, involving the linear elasticity coupled with acoustics for the modeling of the acoustic of the simplified model of human head [6, 8, 25]. The final variational formulation is the following: we seek for elastic velocity $\mathbf{u} \in \widetilde{\mathbf{u}}_D + \mathbf{V}$ and pressure scalar field $p \in \widetilde{p}_D + V$ such that

$$b_{ee}(\mathbf{u}, \mathbf{v}) + b_{ae}(p, \mathbf{v}) = l_e(\mathbf{v}), \qquad \forall \mathbf{v} \in \mathbf{V}, \tag{A12}$$

$$b_{ea}(\mathbf{u}, q) + b_{aa}(p, q) = l_a(q), \qquad \forall q \in V, \tag{A13}$$

where

$$b_{ee}(\mathbf{u}, \mathbf{v}) = \int_{\Omega_e} \left(E_{ijkl} u_{k,l} v_{i,j} - \rho_s \omega^2 u_i v_i \right) d\mathbf{x}, \tag{A14}$$

$$b_{ae}\left(p,\mathbf{v}\right)=\int\limits_{\Gamma_{\mathrm{I}}}pv_{n}dS, \tag{A15}$$

$$b_{ea}\left(\mathbf{u},q\right)=-\omega^{2}\rho_{f}\int\limits_{\Gamma_{\mathrm{I}}}u_{n}qdS, \tag{A16}$$

$$b_{aa}\left(p,q\right)=\int\limits_{\Omega_{a}}\left(\nabla p\cdot\nabla q-k^{2}pq\right)d\mathbf{x}, \tag{A17}$$

$$l_{e}\left(\mathbf{v}\right)=\int\limits_{\Omega_{e}}p_{inc}v_{i}d\mathbf{x}, \tag{A18}$$

$$l_{a}\left(q\right)=0. \tag{A19}$$

$\widetilde{\mathbf{u}}_{D}=0$, $\widetilde{p}_{D}\in H^{1}\left(\Omega_{a}\right)$ is a finite energy lift of pressure prescribed on Γ_{D_a}, where Ω_a part is occupied by an acoustical fluid, Ω_e part is occupied by a linear elastic medium, Γ_I is the interface separating the two sub-domains, Γ_{D_a} is the Dirichlet boundary of the acoustic part. The spaces of test functions are defined as

$$\mathbf{V}=\left\{\mathbf{v}\in\mathbf{H}^{1}\left(\Omega_{a}\right):\mathrm{tr}\mathbf{v}=\mathbf{0}\ \text{ on }\ \Gamma_{D_{e}}\right\}, \tag{A20}$$

$$V=\left\{q\in H^{1}\left(\Omega_{a}\right):\mathrm{tr}q=0\ \text{ on }\ \Gamma_{D_{a}}\right\}. \tag{A21}$$

Here ρ_f is the density of the fluid, ρ_s is the density of the solid, $E_{ijkl}=\mu\left(\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk}\right)+\lambda\delta_{ij}\delta_{kl}$ is the tensor of elasticities, ω is the circular frequency, c denotes the sound speed, $k=\omega/c$ is the acoustic wave number and p_{inc} is the incident wave impinging from the top $p_{inc}=e^{-ik\mathbf{e}x}\quad\mathbf{e}=(-1,0,0)$. For more details we refer to [8].

Appendix B

The Appendix summarizes the interface to the out-of-core solver algorithm. The solver calls the following routines that have to be implemented by the user:

```
subroutine distribute_mesh
c Distribute finite elements into processors
end subroutine

subroutine get_subdomain_system(s)
     use hyper_matrix_mod
     use supernodes_system_mod
     type(supernodes_system) :: s
c Gets local system of equations in the hypermatrix format
end subroutine

subroutine return_solution_to_subdomain(sol)
     use supernodes_system_mod
     type(supernodes_solution) :: sol
c Stores the solution in the local subdomain
end subroutine return_solution_to_subdomain

subroutine get_schur_complement(s)
     use supernodes_system_mod
     use communicators
```

```
      type(supernodes_system),pointer :: s
c Executes the Schur complement dense solver over the local system
end subroutine get_schur_complement

subroutine execute_backward_substitution(s, sol)
      use supernodes_system_mod
      use communicators
      type(supernodes_solution) :: sol
      type(supernodes_system) :: s
c Executes the backward substitution dense solver over the local system
end subroutine execute_backward_substitution
```

In other words, the solver allows for user-defined ordering/generation of the elimination tree, for user defined interface to dense algebra solvers, as well as it requires preparation of the subdomain local system with global numbering of mesh nodes, as it is defined in the `supernodes_system` module.

```
type supernodes_system
c number of nodes in rows / columns
      integer :: nr_rows
      integer :: nr_columns
      type(hyper_matrix) :: A
      type(hyper_matrix) :: b
c global ids of nodes, with pointers to matrix supernodes
      type(supernode_in_matrix), dimension(:),pointer ::row_supernodes
      type(supernode_in_matrix), dimension(:),pointer ::column_supernodes
c number of nodes to be eliminated (=0 if full system)
      integer :: ncount
end type supernodes_system
```

The optimal way is to associate the leaves of the elimination tree with particular finite elements, and then the subdomain systems will represent element frontal matrices.

Appendix C

The Appendix contains the details of the implementation of the Schur complement computations with calls to BLAS routines

Ad. 1) A(1,1)=L*U

```
SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
INTEGER
INFO, LDA, M, N
INTEGER
IPIV( * )
DOUBLE
PRECISION A( LDA, * )
M (input) INTEGER The number of rows of the matrix A. M >= 0.
N (input) INTEGER The number of columns of the matrix A. N >= 0.
A (input/output) DOUBLE PRECISION array, dimension (LDA,N) On entry, the M-by-N
matrix to be factored.
On exit, the factors L and U from the factorization A = P*L*U; the unit
diagonal elements of L are not stored.
LDA (input) INTEGER The leading dimension of the array A. LDA >= max(1,M).
```

IPIV (output) INTEGER array, dimension (min(M,N)) The pivot indices; for 1 <= i <= min(M,N), row i of the matrix was interchanged with row **IPIV**(i).
INFO (output) INTEGER = 0: successful exit
< 0: if INFO = -i, the i-th argument had an illegal value
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

Ad. 2) A(1,2)=(L*U)^-1*A(1,2)

in other words (where A=A(1,2))
R = U^-1*L^-1*A
L^-1*A = B
U^-1*B=R
Algorithm:
a) Given A and L, solve A=L*B
b) Given B and U, solve B=L*R for R
DTRSM -- ZDTRM(SIDE, UPLO, TRANS, DIAG, M, N, ALPHA, A, LDA, B, LDB)
B<=L^-1*A
A is of size mxn
L is of size mxm
SIDE = 'L'
UPLO = 'L' lower triangular matrix
TRANS = 'N' no transpoze of L
DIAG = indicates if the diagonal of L is to be taken to equal the identity matrix (DIAG = "Unit") or the values in the matrix (DIAG = "Non unit").
M=m, N=n
ALPHA=1
A <- L
B <- A
The leading dimensions of the matrices are given in LDA(for L) and LDB (for A).
R<=U^-1*B
B is of size nxn
U is of size nxn
SIDE = 'L'
UPLO = 'U' upper triangular matrix
TRANS = 'N' no transpoze of U
DIAG = indicates if the diagonal of U is to be taken to equal the identity matrix (DIAG = "Unit") or the values in the matrix (DIAG = "Non unit").
M=n, N=n
ALPHA=1
A <- U
B <- B
The leading dimensions of the matrices are given in LDA(for U) and LDB (for B).

Ad. 3) b(1)=(L*U)^-1*b(1)

in other words (where b=b(1))
d = U^-1*L^-1*b
L^-1*b = e
b=L*e
d=U^-1*e

```
e=U*d
Algorithm:
a) Given b and L, solve b=L*e for e
a) Given e and U, solve e=U*d for d
DTRSM - ZDTRM( SIDE, UPLO, TRANS, DIAG, M, N, ALPHA, A, LDA, B, LDB )
b is of size mx1
L is of size mxm
SIDE = 'L'
UPLO = 'L' lower triangular matrix
TRANS = 'N' no transpoze of L
DIAG = indicates if the diagonal of L is to be taken to equal the identity
matrix (DIAG = "Unit" ) or the values in the matrix (DIAG = "Non unit" ).
M=m, N=1
ALPHA=1
A <- L
B <- b
The leading dimensions of the matrices are given in LDA(for L) and LDB (for b)
d<-U^-1*e
e is of size mx1
U is of size mxm
SIDE = 'L'
UPLO = 'U' upper triangular matrix
TRANS = 'N' no transpoze of U
DIAG = indicates if the diagonal of U is to be taken to equal the identity
matrix (DIAG = "Unit") or the values in the matrix (DIAG = "Non unit" ).
M=m, N=1
ALPHA=1
A <- U
B <- e
The leading dimensions of the matrices are given in LDA(for U) and LDB (for e)

Ad. 4) A(2,2)=A(2,2)-A(2,1)*A(1,2)
DGEMM/ZGEMM (double / complex)
ZGEMM (transa, transb, l, n, m, alpha, a, lda, b, ldb, beta, c, ldc)
C = alpha A *B + beta* C
here alpha = -1, beta = 1, A(2,1)=mult, B=A(1,2), C=A(2,2)
transa = 'N' A is used in the computation.
transb = 'N', B is used in the computation.
l is the number of rows in matrix C.
n is the number of columns in matrix C.
m is the number of columns in matrix A.
alpha is the scalar alpha.
a is the matrix A, where: A has l rows and m columns.
If transa equal to 'N', its size must be lda by (at least) m.
lda is the leading dimension of the array specified for a.
b is the matrix B, where: B has m rows and n columns.
ldb is the leading dimension of the array specified for b.
beta is the scalar beta.
c is the l by n matrix C.
ldc is the leading dimension of the array specified for c.
On Return c is the l by n matrix C, containing the results of the computation.
```

```
Ad. 5) b(2)=b(2)-A(2,1)*b(1)
  ZGEMV
  ( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )
  TRANS = 'N' for y := alpha*A*x + beta*y,
  M = number of rows in A
  N = number of columns in A
  ALPHA=-1
  A <= A(2,1)
  LDA On entry, LDA specifies the first dimension of A as declared in the
  calling (sub) program
  X<=b(1)
  INCX=1
  BETA=1
  Y<=b(1)
  INCY=1
```

Acknowledgement

The work was supported by the Polish Ministry of Science and Higher Education grant no. NN 519 405737 and NN 519 447739 and DEC-2011/01/B/ST6/00674. The visits of the author at ICES were supported by ICES Oden Research Faculty Fellowship program.

References

[1] M. Paszyński, D. Pardo, C. Torres-Verdin, L. Demkowicz, V. Calo. A parallel direct solver for the self-adaptive *hp* finite element method. *Journal of Parallel and Distributed Computing*, **70**(3): 270–281, 2010.

[2] M. Paszyński, R. Schaefer. Graph grammar driven partial differential eqautions solver. *Concurrency and Computations: Practise and Experience*, **22**(9): 1063–1097, 2010.

[3] A. Szymczak, M. Paszyński. Graph grammar based Petri net controlled direct sovler algorithm. *Computer Science*, **11**: 65–79, 2010.

[4] M. Paszyński, R. Schaefer. Reutilization of partial LU factorizations for self-adaptive hp Finite Element Method solver. *Lecture Notes in Computer Science*, **5101**: 965–974, 2008.

[5] M. Paszyński, L. Demkowicz. Parallel, fully automatic, *hp*-adaptive 3D finite element package. *Engineering with Computers*, **22**: 255–276, 2006.

[6] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek. *Computing with hp-Adaptive Finite Elements, Vol. II. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications.* Chapman & Hall/CRC Applied Mathematics & Nonlinear Science, 2007.

[7] M. Paszyński, D. Pardo, A. Paszyńska. Parallel multi-frontal solver for *p*-adaptive finite element modeling of multi-physics computational problems. *Journal of Computational Science*, **1**: 48–54, 2010.

[8] L. Demkowicz, P. Gatto, J. Kurtz, M. Paszynski, W. Rachowicz, E. Bleszyński, M. Bleszyński, M. Hamilton, C. Champlin, D. Pardo. Modeling of bone conduction of sound in human head using hp finite elements. Part I. Code design and modificaton. *Computer Methods in Applied Mechanics and Engineering*, **200**: 1757–1773, 2011.

[9] D. Pardo. Integration of hp-Adaptivity with a Two-Grid Solver. PhD. Dissertation. University of Texas at Austin, 2004.

[10] D. Pardo, L. Demkowicz, C. Torres-Verdin, M. Paszyński. A self-adaptive goal-oriented hp finite element method with electromagnetic applications. Pt. 2, Electrodynamics. *Computer Methods in Applied Mechanics and Engineering*, **196**: 3585–3597, 2007.

[11] P.R. Amestoy, I.S. Duff, J. Koster, J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, **23**(1): 15–41, 2001.

[12] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, **32**(2): 136–156, 2006.

[13] Multi-frontal Massively Parallel Sparse direct solver (MUMPS), http://graal.ens-lyon.fr/MUMPS.

[14] D. Pardo, M.J. Nam, C. Torres-Verdin, M. Hoversten, I. Garay. Simulation of Marine Controlled Source Electromagnetic Measurements Using a Parallel Fourier *hp*-Finite Element Method. *Computational Geosciences*, **15**(1): 53–67, 2011.

[15] N.I.M. Gould, J.A. Scott, Hu Yifan. Numerical Evaluation of Sparse Direct Solvers for the Solution of Large Sparse Symmetric Linear Systems of Equations. *ACM Transactions on Mathematical Software (TOMS)*, **33**(2)10: 300–331, 2007.

[16] S. Fialko. A block sparse shared-memory multifrontal finite element solver for problems of structural mechanics. *Computer Assisted Mechanics and Engineering Sciences*, **16**: 117–131, 2009.

[17] S. Fialko. *A Sparse Shared-Memory Multifrontal Solver in SCAD Software, Proceedings of the International Multiconference on Computer Science and Information Technology*. October 22–28, 2009,Wisła, Poland, **3**: 277–283, 2008.

[18] S. Fialko. The block substructure multifrontal method for solution of large finite element equation SETS. *Technical Transactions*, 1-NP/2009 (8): 175–188, 2009.

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

[20] S. Fialko. The Nested Substructures Method for Solving Large Finite-Element Systems as Applied to Thin-Walled Shells with High Ribs. *International Applied Mechanics*, **39**(3): 324–332, 2003.

[21] P. Gend, J.T. Oden, R. van de Geijn. A parallel multifrontal algorithm and its implementation. *Computer Methods in Applied Mechanics and Engineering*, **149**: 289–301, 1997.

[22] 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.

[23] I. Babuška, B. Guo. The hp-version of the finite element method, Part I: The basic approximation results. *Computational Mechanics*, **1**: 21–41, 1986.

[24] I. Babuška, B. Guo. The hp-version of the finite element method, Part II: General results and applications. *Computational. Mechanics*, **1**: 203–220, 1986.

[25] M. Paszyński, D. Pardo, A. Paszyńska. Parallel multi-frontal solver for p adaptive finite element modeling of multi-physics computational problems. *Journal of Computational Science*, **1**: 48–54, 2010.