Contents lists available at ScienceDirect

Journal of Computational Science

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

Parallel multi-frontal solver for *p* adaptive finite element modeling of multi-physics computational problems

Maciej Paszyński[a,*], David Pardo[b], Anna Paszyńska[c]

[a] Department of Computer Science, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland
[b] BCAM (Basque Center for Applied Mathematics) and IKERBASQUE (Basque Foundation for Sciences), Bizkaia Technology Park, Building 500, E-48160 Bilbao, Spain
[c] Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, Reymonta 4, 30-059 Kraków, Poland

ARTICLE INFO

Article history:
Received 14 March 2010
Accepted 26 March 2010

Keywords:
Parallel multi-frontal solver
Multi-physics problems
Finite element method
p adaptivity
Hypermatrix

ABSTRACT

The paper presents a parallel direct solver for multi-physics problems. The solver is dedicated for solving problems resulting from adaptive finite element method computations. The concept of finite element is actually replaced by the concept of the node. The computational mesh consists of several nodes, related to element vertices, edges, faces and interiors. The ordering of unknowns in the solver is performed on the level of nodes. The concept of the node can be efficiently utilized in order to recognize unknowns that can be eliminated at a given node of the elimination tree. The solver is tested on the exemplary three-dimensional multi-physics problem involving the computations of the linear acoustics coupled with linear elasticity. The three-dimensional tetrahedral mesh generation and the solver algorithm are modeled by using graph grammar formalism. The execution time and the memory usage of the solver are compared with the MUMPS solver.

1. Introduction

The paper focuses on the development of an efficient parallel solver for multi-physics problems solved by using the *hp* adaptive finite element method (*hp*-FEM) [1,2]. This is the first step towards the extension of our previous two-dimensional version of the solver [3,4] into higher dimensional problems. Parallel multi-physics problems usually generate huge linear systems of equations, which are not well conditioned, and thus, the applicability of iterative solvers is typically limited. In addition, iterative solvers typically exhibit lack of robustness (in presence of high-contrast materials, elongated elements, etc. [5]). 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 [6]) and inverse solvers (when computing the Jacobian and Hessian matrices). Thus, the main focus of this research is based on using direct solvers.

The large size of these problems typically requires the use of parallel direct solvers. The current state-of-the-art is the parallel multi-frontal solver, e.g. MUMPS solver [7–10]. However, the usage of general purpose solvers for multi-physics problems solved by adaptive *hp*-FEM is also limited, since some special domain decomposition, ordering of degrees of freedom and reutilization of partial LU factorizations [11] algorithms must be developed. Our preliminary study on the coupled linear elastic-acoustics problems shows that even relatively simple three-dimensional geometries – concentric spheres – with uniformly growing polynomial order of approximation resulting from the global *p* refinement procedure requires massive parallel computations. For large *p* computations are out of range for the MUMPS solver, see Table 1.

Preliminary numerical results clearly show that for this class of multi-scale multi-physics problems it is essential to use a special version of the solver that incorporates a specific domain decomposition ordering and a reutilization algorithm intended to minimize the memory usage of the solver.

2. Multi-frontal parallel direct solver algorithm

2.1. Overview

In this section we introduce a new parallel multi-frontal solver interfaced with the FEM code utilizing the domain decomposition paradigm. The computational mesh stored by the FEM code is divided into multiple sub-domains. Each sub-domain is assigned to a single processor, possibly with multiple cores, allowing for the multi-thread execution.

The decision about the partition of the computational mesh into sub-domains can be supported by libraries like ZOLTAN [13]. ZOLTAN (as well as other mesh partitioning libraries) allows for

* Corresponding author. Tel.: +48 12 633 9406; fax: +48 12 633 9406.
E-mail addresses: maciej.paszynski@agh.edu.pl, paszynsk@agh.edu.pl (M. Paszyński).

1877-7503/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.jocs.2010.03.002

Table 1
Execution of the parallel MUMPS solver over 24,192 finite element mesh on LONESTAR linux cluster (from Texas Advanced Computing Center) for the coupled linear elasticity/acoustics problem. The MUMPS solver requires huge amount of memory and it crashes for $p = 5$, for any number of processors.

Polynomial order of approximation	Number of degrees of freedom	Number of non-zero entries	Number of processors	Memory [MB]/per processor	Execution time
$p = 3$	538,123	76,649,280	16	5458–7687	15 min
$p = 4$	1,236,831	250,119,936	64	5245–7810	24 min
$p = 5$	2,313,069	–	–	–	–

the implementation of user defined mesh partitioning algorithms. In the numerical experiments presented in this paper we utilize a simple domain decomposition algorithm, cutting the three-dimensional ball shape domain into slices (compare Fig. 1).

The parallel multi-frontal solver utilizes the elimination tree presented in Fig. 1. The leaves of the elimination tree are associated with sub-domains resulting from the partition of the computational mesh into multiple sub-domains. The leaves of the elimination tree – single sub-domains – are assigned to single processors.

```
system function recursive_solver1(tree_node)
  system = 0
// section #1
// leaf node computes Schur complement of sub-domain internal nodes
// with respect to sub-domain interface nodes
  if only 1 proc is assigned to tree_node then
    system = Schur complement
      of sub-domain internal nodes
      with respect to sub-domain interface nodes
  else
// section #2
// other nodes: send/recv contributions between son nodes
    system = 0
    do ison for each son_node of tree_node
      if MYRANK is assigned to node then
        system_contributions(1) = recursive_solver1(son_node)
        if MYRANK is n/2+1 on the list of
          n processors assigned to node then
          send system_contributions(1) to 1st processor from the list
        else if MYRANK is 1st processor on the list of
          n processors assigned to node then
          receive system_contributions(2)
            from n/2+1 processor from the list of n processors
        endif
      endif
    enddo
// section #3
// eliminate fully assembled nodes
    create NODE_COMMUNICATOR with processors assigned to tree_node
    barrier(NODE_COMMUNICATOR)
    if MYRANK is 1st processor on the list of
       n processors assigned to node then
      create system
      decide which nodes from system_contributions can be eliminated
    endif
    system = resulting Schur complement computed from system
        using all processors from NODE_COMMUNICATOR
    if MYRANK is 1st processor on the list of
       n processors assigned to node then
// section #4
// store Schur complement at the node
      store system at tree_node
    endif
    delete NODE_COMMUNICATOR
  endif
  return system
end
```

2.2. Solver algorithm

Several sequential solvers are executed, each one assigned to a leaf—a single sub-domain (see Section 1). The solvers compute partial LU factorizations to get the Schur complement of the sub-domain internal unknowns (called degrees of freedom) with respect to the interface degrees of freedom. The LU factorizations are stored at nodes of the elimination tree for possible future reutilization (see Section 4).

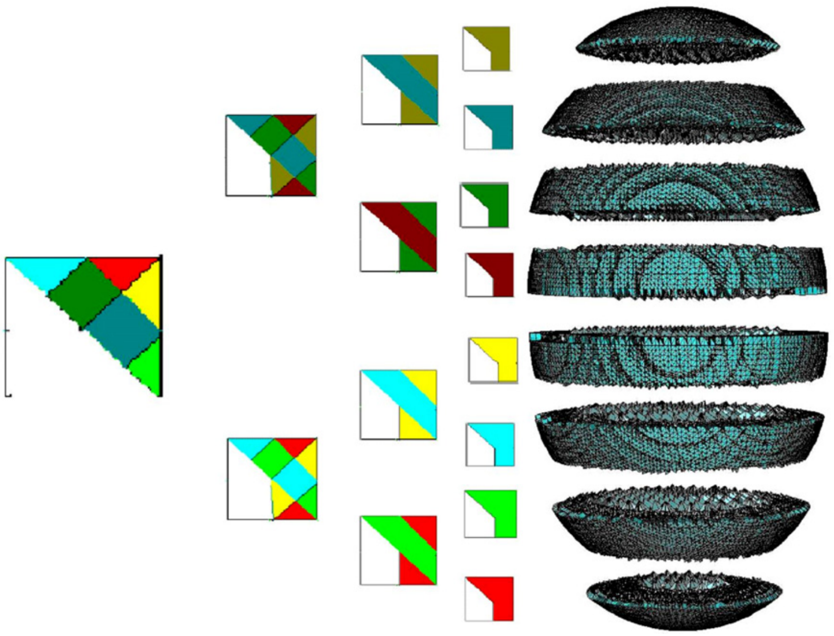

Fig. 1. . The elimination tree (left panel) resulting from partition of the computational mesh into sub-domains.

Having the Schur complements computed, the global interface problem must be solved now. It is done by utilizing the mutli-level elimination pattern described in Fig. 1. First, Schur complements are joined into pairs, being two new contributions to the new system of equations (see Section 2). Fully assembled degrees of freedom are eliminated, and new Schur complements are obtained (see Section 3) with these degrees of freedom that have not been eliminated yet. The pattern is repeated until one common interface problem is obtained. Notice that at every node of the top elimination tree there are several processors available. Thus, we utilize several processors at every tree node to avoid idle processors. This is done by constructing a NODE_COMMUNICATOR involving all available processors for the node.

2.3. *Graph grammar model for tetrahedral meshes*

The order of elimination of degrees of freedom over a single sub-domain can be deducted by using the similar pattern that has been already applied on the level of sub-domains. In order to do so, we introduce the graph representation of the finite element mesh. It is done by using the graph grammar defined in Fig. 2. This is an extension of the graph grammar already introduced for the two-dimensional meshes [12]. The graph grammar consists of a set of graph transformations, called graph grammar productions. The sequence of productions starts with an initial graph with a single vertex called *S*. Each production replaces a sub-graph from its left-hand side with a sub-graph from its right-hand side. The production (**Pinit**) is the initial transformation that generates a single tetrahedral finite element. The generated graph vertices **vv**, **FF**, **II** and **In** represents a finite element vertex, edge, face and interior, respectively. All graph vertices representing a single face are collected as sons of a graph vertex **FA**. All graph vertices representing a single tetrahedral element are collected as sons of a graph vertex **TH**.

The production (**Padd**) generates a new tetrahedral element and merges it to a face of one of already generated elements. The production (**Pclose**) identifies faces of two elements having the same single edge and identical geometrical coordinates of corresponding vertices.

2.4. *Graph grammar model for the solver execution*

At this point we can introduce the graph grammar productions modeling the solver algorithm, based on the graph grammar model already introduced for two-dimensional meshes [4]. The graph grammar productions presented in Fig. 3 are responsible for attributing graph vertices with frontal matrix identifiers and for denoting nodes that have been already eliminated.

The first production (**Pelimint**) generates a new frontal matrix for an element, and eliminates the element interior. The new frontal matrix is identified by α_i index. The second production (**Pelimface**) is responsible for merging two frontal matrices from two adjacent elements, and eliminating the unknowns associated with the common face. The set of graph grammar productions is completed by additional productions which are not presented here. These productions are responsible for assembling and elimination of edges and vertices.

3. Numerical results

3.1. *Weak form of linear elasticity coupled with acoustics*

We focus here on numerical simulations of the exemplary challenging multi-physics problem, involving the linear elasticity coupled with acoustics [1]. The final variational formulation for the linear elasticity coupled with acoustics is the following: we seek for elastic velocity $\boldsymbol{u} \in \tilde{\boldsymbol{u}}_D + \boldsymbol{V}$ and pressure scalar field $p \in \tilde{p}_D + V$ such that:

$$b_{ee}(\boldsymbol{u}, v) + b_{ae}(p, v) = l_e(v), \forall v \in \boldsymbol{V} \quad (1)$$

$$b_{ea}(\boldsymbol{u}, q) + b_{aa}(p, q) = l_a(q), \forall q \in V \quad (2)$$

where

$$b_{ee}(\boldsymbol{u}, v) = \int_{\Omega_e} (E_{ijkl} u_{k,l} v_{i,j} - \rho_s \omega^2 u_i v_i)\, d\boldsymbol{x} \quad (3)$$

$$b_{ae}(p, v) = \int_{\Gamma_1} p v_n \, dS \quad (4)$$

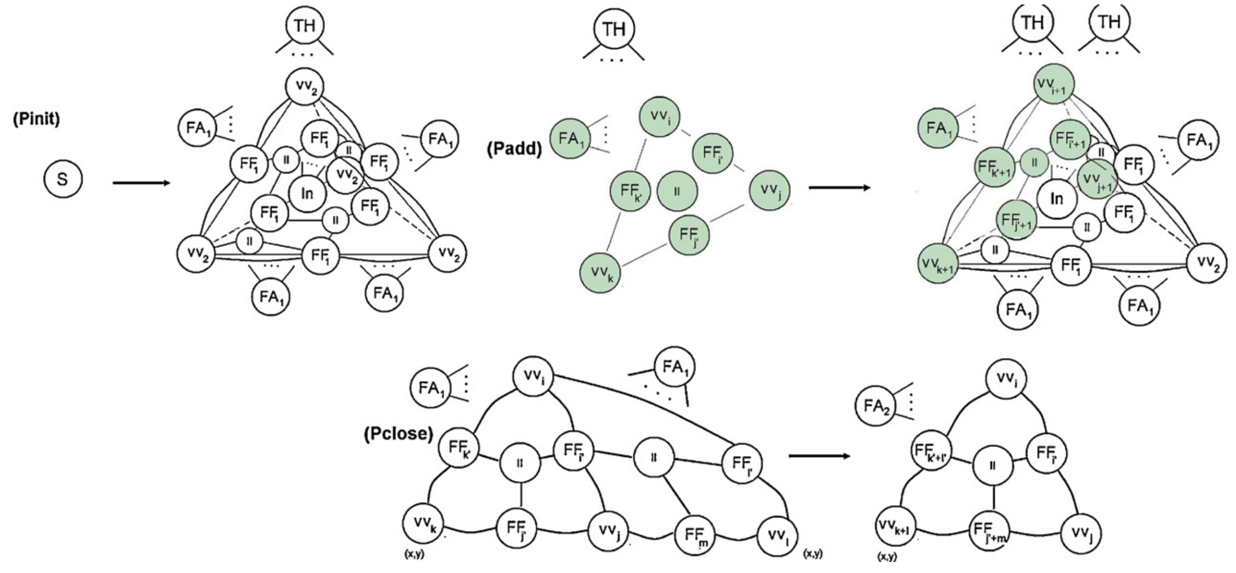

Fig. 2. Graph grammar productions responsible for tetrahedral mesh generation.

$$b_{ea}(\boldsymbol{u}, q) = -\omega^2 \rho_f \int_{\Gamma_1} u_n q \, dS \tag{5}$$

$$b_{aa}(p, q) = \int_{\Omega_a} (\nabla p \cdot \nabla q - k^2 pq) \, d\boldsymbol{x} \tag{6}$$

$$l_e(v) = \int_{\Omega_e} p_{inc} v_i \, d\boldsymbol{x} \tag{7}$$

$$l_a(q) = 0 \tag{8}$$

$\tilde{\boldsymbol{u}}_D = 0$, $\tilde{p}_D \in H^1(\Omega_a)$ 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:

$$\boldsymbol{V} = \{v \in \boldsymbol{H}^1(\Omega_a) : \mathrm{tr} v = \boldsymbol{0} \text{ on } \Gamma_{D_e}\} \tag{9}$$

$$V = \{q \in H^1(\Omega_a) : \mathrm{tr} q = 0 \text{ on } \Gamma_{D_a}\} \tag{10}$$

Here ρ_f is the density of the fluid, ρ_s is the density of the solid, $E_{ijkl} = \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) + \lambda\delta_{ij}\delta_{kl}$ 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 $p_{inc} = e^{-ik\boldsymbol{e}x}$ $\boldsymbol{e} = (-1, 0, 0)$. For more details we refer to [2].

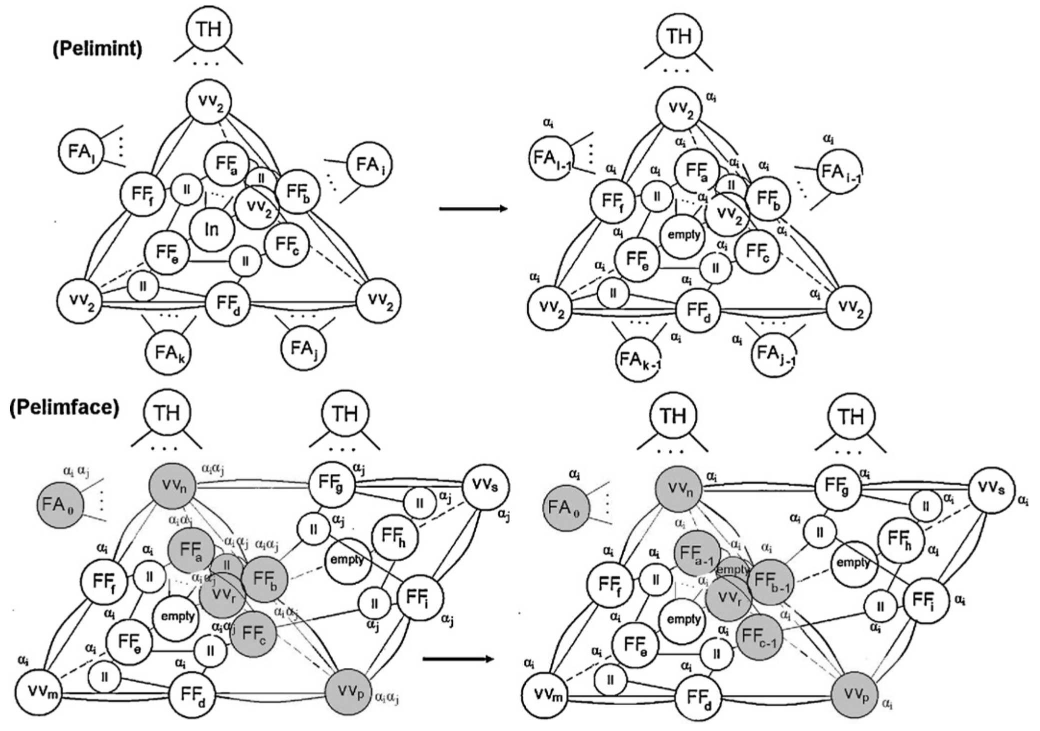

Fig. 3. Graph grammar productions responsible for elimination of element's interior and face's interior.

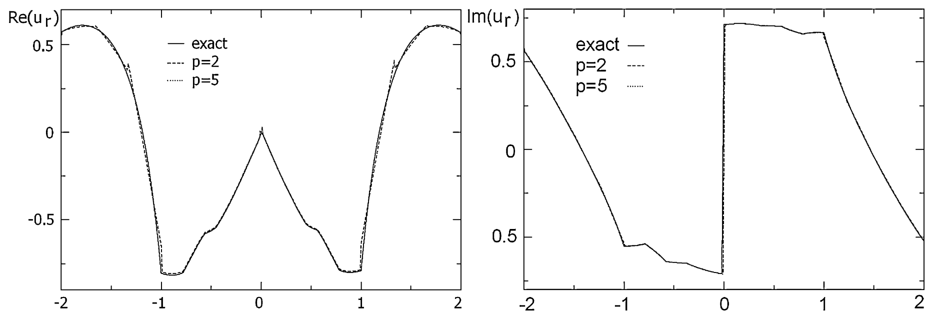

Fig. 4. (Left panel) Real part of the radial component of the displacement field and (right panel) imaginary part of the radial component of the displacement field.

Table 2
Material data for the first numerical problem.

Layers	ρ_s/ρ_f	E	ν	Range
Tissue	1.0	0.625	0.2	$0 < r < 0.7$
Skull	1.0	2.5	0.25	$0.7 < r < 0.784$
Cork	0.3	0.5	0.2	$0.784 < r < 0.964$
Steel	2.0	5.0	0.3	$0.964 < r < 1.0$
Air	1.0	–	–	$1.0 < r < 2.0$

3.2. Problem formulation

The model presented in this paper is the preliminary step towards the finite element method modeling of the acoustics of the human head. The computational domain is defined as a ball filled with tetrahedral finite elements. The three-dimensional ball shape domain is extended by adding six additional concentric layers of prismatic finite elements. The most inner ball represents the tissue, the second layer represents the skull, the third and the fourth layers represent the helmet, with cork and steel, and last three layers represent the air with the last layer used to truncate the domain by utilizing the perfectly matching layer (PML) technique [14]. The material constants for the domain layers are summarized in Tables 2 and 3.

It should be emphasized that this is a multi-scale valued problem, with three components of the unknown elastic velocity over the elastic domain (tissue, skull, cork and steel), one component of the unknown pressure scalar field over the acoustic domain (air with PML) and four unknowns over the interface. Moreover, we utilize the global p adaptation technique to increase the accuracy of the solution over the elastic domain, where p stands for the polynomial order of approximation utilized over an element edge, face or interior. Thus, the number of unknowns at each vertex node is equal to one or three (depending on the acoustic/elasticity domain type), the number of unknowns at each edge is $p-1$ or $3^*(p-1)$ (acoustic/elasticity), the number of unknowns at each face is of the order of $(p-1)^2$ or $3^*(p-1)^2$ (acoustic/elasticity) and the number of unknowns at each interior is of the order of $(p-1)^3$ or $3^*(p-1)^3$ (acoustic/elasticity). Thus the solver algorithm has to deal with different sizes of matrices at different nodes of the elimination trees.

This is one of the motivations for developing the node-based solver. Having the node-based data structure, the concept of finite element is actually replaced by the concept of the node. The computational mesh consists of several nodes, related to element vertices, edges, faces and interiors. The concept of the node can be efficiently utilized in order to recognize degrees of freedom that can be eliminated at a given node of the elimination tree. The node-based solver uses the concept of a hypermatrix. The hypermatrix consist of several sub-matrices (called p-blocks) related to particular nodes, with number of degrees of freedom associated with the polynomial order of approximation utilized at the node. The nodes will keep the links (in the hash table manner) to the p-blocks related to the nodes. The degrees of freedom to be eliminated can be identified by browsing nodes and following the links to the p-blocks, and by recognizing those p-blocks which have been fully assembled at this point.

3.3. Numerical experiments

We have performed three numerical experiments. The goal of the first experiment was to test the convergence of the uniform p adaptation algorithm for the problem described in the previous section, but with smooth (not-real) material data, summarized in Table 2. In Fig. 4, we plot the resulting pressure distribution obtained from the post-processing from fully three-dimensional results over the cross-section of the domain. We compare the results obtained for $p=2$ with the results obtained for $p=5$. The size of the $p=2$ mesh was 29,760 finite elements and 213,999 degrees of freedom. The size of the $p=5$ mesh was again 29,760 finite elements and 2,313,069 degrees of freedom. The problem has been solved on LONESTAR linux cluster [15] with 16 and 64 processors. We can clearly see the convergence of the uniform p method.

The goal of the second experiment was to test the stability of the model with real material data, summarized in Table 3. We have compared the results obtained for $p=2$ with the results obtained for $p=3$, since there is no large difference between $p=2$ and $p=3$ results, and increasing further the polynomial order of approximation does not seem to be necessary here. The size of the $p=2$ mesh was 29,760 finite elements and 213,999 degrees of freedom. The size of the $p=3$ mesh was again 29,760 finite elements and 538,123 degrees of freedom. In Fig. 5, we plot the resulting pressure distribution. In this more difficult case we have also obtained the convergence of the uniform p method.

The goal of the third test was to compare the execution time and memory usage of our solver with the MUMPS parallel solver.

Table 3
Material data for the second numerical problem.

Layers	ρ_s/ρ_f	E	ν	Range
Tissue	835.0	$30{,}000 + 2.0\,i$	$0.0000138 + 0.0000287\,i$	$0 < r < 0.7$
Skull	1.0	2.6	0.3	$0.7 < r < 0.784$
Cork	150.0	539.0	0.173	$0.784 < r < 0.964$
Steel	6666.7	6,000,000.0	0.355	$0.964 < r < 1.0$
Air	1.0	–	–	$1.0 < r < 2.0$

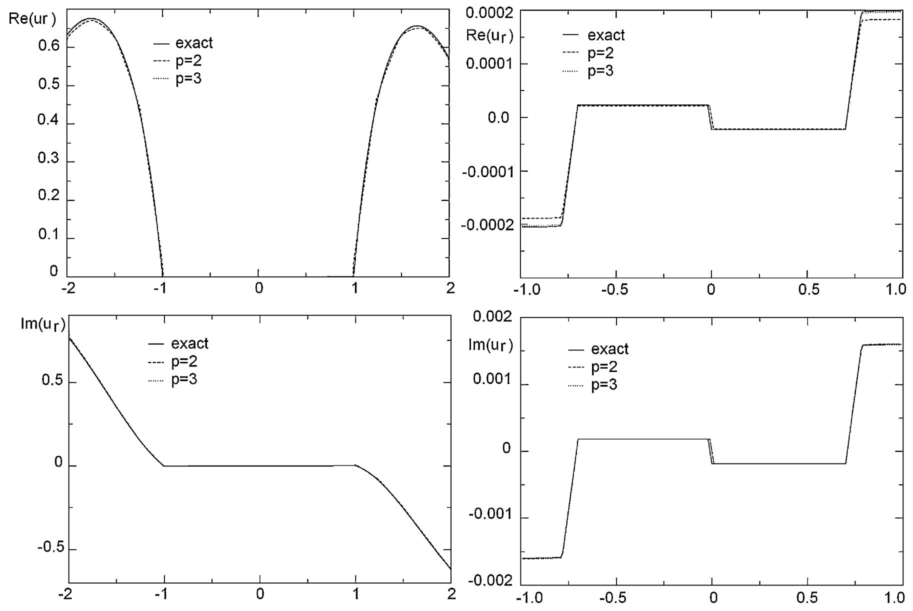

Fig. 5. (Top panels) Real part of the radial component of the displacement field; (bottom panel) imaginary part of the radial component of the displacement field.

This time the computational mesh with $p = 2$ with 29,760 finite elements with 213,999 degrees of freedom has been partitioned into 16 sub-domains. The problem has been solved on a PC with 32 GB of RAM. The solver has been executed in a sequential mode. Thus, the cost of communication can be assumed to be zero, since all Schur complements were created within the shared memory. The solver has been executed with out-of-core version of a sequential MUMPS solver utilized at every tree node, on a single processor. The statistics of the solver execution are summarized in Table 4. The total execution time for the LU factorization was 622 s. The predictive maximum memory usage for our solver is equal to the maximum of memory usages per elimination tree nodes, since each Schur complement matrix could be dumped out to disc after processing. Thus, the predictive maximum memory usage of our solver is 1916 MB. We call it "predictive" since in the current version the Schur complements are not dump-out yet, only the sequential MUMPS solver utilized in every tree node is using the out-of-core feature.

Table 4
Solver execution statistics per 16 sub-domains elimination tree.

Tree node	Problem size	Number of non-zero entries	Execution time [s]	Memory usage [MB]
Sub-domain 0	3557	332,191	3	87
Sub-domain 1	3911	372,952	3	137
Sub-domain 2	4416	442,464	4	166
Sub-domain 3	4414	402,744	4	152
Sub-domain 4	3769	345,943	3	151
Sub-domain 5	3382	301,744	3	162
Sub-domain 6	5033	510,041	5	301
Sub-domain 7	4429	444,631	4	186
Sub-domain 8	3956	352,765	3	166
Sub-domain 9	4205	426,477	4	161
Sub-domain 10	4454	475,695	5	138
Sub-domain 11	4043	392,659	4	145
Sub-domain 12	4112	430,856	4	120
Sub-domain 13	3562	279,616	3	162
Sub-domain 14	4372	431,209	5	159
Sub-domain 15	4212	416,574	5	140
Sub-domains 0,1	2910	4,667,527	4	392
Sub-domains 2,3	3234	6,531,273	11	462
Sub-domains 4,5	3676	5,543,848	3	379
Sub-domains 6,7	4496	10,012,208	19	747
Sub-domains 8,9	3447	7,165,240	13	534
Sub-domains 10,11	3206	6,010,750	9	444
Sub-domains 12,13	3318	5,749,926	6	438
Sub-domains 14,15	3310	6,446,964	11	484
Sub-domains 0,1,2,3	4817	13,832,607	22	1003
Sub-domains 4,5,6,7	6147	16,623,332	52	1327
Sub-domains 8,9,10,11	4772	15,046,955	38	984
Sub-domains 12,13,14,15	5389	15,534,227	45	1329
Sub-domains 0,1,2,3,4,5,6,7,8	7040	35,658,739	170	1916
Sub-domains 9,10,11,12,13,14,15	5389	15,534,227	45	1329
Sub-domains 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15	4753	22,576,729	112	892
Total			622	11,948

These results have been compared with the out-of-core version of the MUMPS solver [10]. The total execution time of the out-of-core LU factorization of the MUMPS solver was 2221 s, and the maximum memory usage of the MUMPS solver was 3998 MB.

4. Conclusions and future work

We have presented the preliminary promising version of the multi-frontal multi-scale parallel direct solver for adaptive FEM. We have proved the potential of the solver for 2 times less memory usage and 3.5 times faster execution time than the MUMPS solver. We intend to improve the solver by implementing our own algorithm partitioning each physics part of the domain separately, in order to avoid mixed physics within a single domain. We consider the design and development of a new mesh partitioning and ordering algorithms in order to minimize the size and the density of the interface problems. We consider to implement the out-of-core version of the solver with reutilization of LU factorizations, based on our previous work [11]. We also consider to make out the mutli-thread versions of the solver, for the case when there are several cores available at every processor. We have some preliminary results for the multi-thread solver for one and two-dimensional finite difference method [16].

Acknowledgements

The work reported in this paper was supported by the Foundation for Polish Science under Homing Programme, by the Polish MNiSW grant no. NN 519 405737. We acknowledge the Texas Advanced Computing Center (TACC) for the computational time. The work of the second author was partially funded by the Spanish Ministry of Science and Innovation under the projects MTM2008-03541, TEC2007-65214, and PTQ-08-03-08467.

References

[1] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek, Computing with hp Adaptive Finite Elements. Volume 2. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications, Chapmann & Hall/CRC Press, 2007.

[2] M. Paszyński, L. Demkowicz, Parallel fully automatic *hp* adaptive 3D finite element package, Engineering with Computers 22 (3–4) (2006) 255–276.

[3] 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) (2010) 270–281.

[4] M. Paszyński, R. Schaefer, Graph grammar-driven parallel partial differential equation solver, Concurrency & Computations, Practise & Experience, doi:10.1002/cpe.1533.

[5] D. Pardo, Integration of hp-adaptivity with a two-grid solver, Ph.D. dissertation, The University of Texas in Austin, 2004.

[6] 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 (2007) 3585–3597.

[7] P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, Computer Methods in Applied Mechanics and Engineering 184 (2000) 501–520.

[8] 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 (2001) 15–41.

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

[10] MUMPS: http://www.enseeiht.fr/lima/apo/MUMPS version 4.7.3.

[11] M. Paszyński, R. Schaefer, Reutilization of partial LU factorizations for self-adaptive *hp* finite element method solver, Lecture Notes in Computer Science 5101 (2008) 965–974.

[12] M. Paszyński, A. Paszyńska, Graph transformations for modeling parallel *hp*-adaptive finite element method, Lecture Notes in Computer Science 4967 (2008) 1313–1322.

[13] ZOLTAN: http://www.cs.sandia.gov/Zoltan.

[14] C. Michler, L. Demkowicz, J. Kurtz, D. Pardo, Improving the performance of perfectly matched layers by means of hp-adaptivity, in: ICES-Report 06-17, 2006, pp. 359–392.

[15] LONESTAR cluster users' manual http://www.tacc.utexas.edu/services/userguides/lonestar.

[16] P. Obrok, P. Pierchała, A. Szymczak, M. Paszynski, Graph grammar based multi-thread multi-frontal parallel solver with trace theory-based scheduler, Procedia Computer Science, 2010.

Maciej Paszyński received his Ph.D. (2003) in mathematics with applications to computer science from the Jagiellonian University, Kraków, Poland. In 2003–2005, he worked as a postdoctoral fellow at the Institute for Computational Engineering and Sciences (ICES) at The University of Texas at Austin. In 2005–2006, he worked at the Department of Modeling and Information Technology at AGH University of Science and Technology of Kraków. In summer 2006, he worked as a postdoctoral fellow at the Department of Petroleum and Geosystems Engineering at The University of Texas at Austin. Since 2006, he holds a position as an assistant professor in the Department of Computer Science at AGH University of Science and Technology of Kraków. His research interests include parallel self-adaptive hp finite element method, parallel direct solvers, and computational science.

David Pardo Zubiaur received the bachelor of science degree in mathematics from the University of the Basque Country in May 2000. In June 2000, he entered the Graduate School of The University of Texas at Austin for enrollment in the Computational and Applied Mathematics (CAM) Ph.D. program. In summer 2003, he worked at the oil company Baker-Atlas, a division of Baker-Hughes. In the academic year 2003–2004, he was awarded with the Continuing Education Fellowship at The University of Texas at Austin. After graduating in April 2, 2004, he worked as a postdoctoral fellow and research associate until August 2008 at the Department of Petroleum and Geosystems Engineering at the University of Texas at Austin. He currently holds a research professor position at the Basque Center for Applied Mathematics (BCAM).

Anna Paszyńska received her Ph.D. (2007) in computer science from the Institute of Fundamental Technological Research of Polish Academy of Sciences in Warsaw, Poland. She currently works as a postdoctoral fellow at the Jagiellonian University in Kraków, Poland. Her research interests include evolutionary algorithms, graph grammar and computer aided design.