A Multi Level Direct Sub-Structuring Multi-Frontal Parallel Solver for the *hp*-Finite Element Method

Maciej Paszyński[(1)], David Pardo[(2)], Carlos Torres-Verdín[(2)],
Leszek Demkowicz[(3)], Victor Calo[(3)]

[(1)] Department of Computer Science,
AGH University of Science and Technology,
e-mail: paszynsk@agh.edu.pl
[(2)] Department of Petroleum and Geosystems Engineering
[(3)] Institute for Computational Engineering and Sciences
The University of Texas in Austin

Abstract

The paper presents a new parallel direct solver for hp refined meshes, utilized by a 2D hp adaptive Finite Element Method (FEM) to solve 3D Direct Current (DC) borehole resistivity measurement problems. The self-adaptive hp FEM generates in a fully automatic mode a sequence of hp meshes delivering exponential convergence of the error with respect to the number of degrees of freedom (d.o.f.) as well as the CPU time. The new parallel solver works on the three-level elimination tree, distributed into processors: (1) the refinement tree growing from initial mesh elements, (2) the initial mesh element tree, and (3) the sub-domain tree resulting from the domain decomposition. The solver computes Schur complements at every tree node, by performing partial forward elimination, leaving untouched partially assembled (or unassembled) d.o.f. The Schur complements are stored at tree nodes, and can be re-utilized when solving subsequent hp meshes (produced by the self-adaptive strategy), when these hp meshes contain elements that have not been refined, as it typically occurs in real applications. Execution time and memory usage of the solver are compared against those delivered by the parallel MUltifrontl Massively Parallel sparse direct Solver (MUMPS) with (1) centralized entries submitted from the host processor, (2) distributed entries submitted from sub-domains, and (3) the MUMPS based direct sub-structuring method when the parallel MUMPS solver with distributed entries is utilized to solve the interface problem.

Key words: Parallel direct solvers, Finite Element Method, hp adaptivity, 3D borehole resistivity

1 Introduction

The paper presents a new parallel direct solver designed for the hp Finite Element Method (FEM) [5]. Sequential and parallel 2D and 3D hp adaptive FE codes [6, 20, 19, 18] generate automatically a

sequence of optimal FE meshes providing exponential convergence of the numerical error of the solution with respect to the CPU time and the mesh size, expressed in terms of the number of degrees of freedom (d.o.f.). A sequence of meshes is automatically generated by the computer by performing h or p refinements. The h refinement consists of breaking a finite element generating several new smaller elements; p refinement consists of increasing the polynomial order of approximation over some finite element edges, faces, and interiors. As we refine the grid, the number of finite elements increase and the polynomial orders of approximation associated to each edge and interior change.

The fully automatic hp adaptive algorithm [6] delivers a sequence of optimal hp-meshes that enables accurate approximation of challenging engineering problems. However, the computational cost needed to solve the problem of interest over this sequence of meshes is large. Thus, there is a need to utilize efficient parallel direct solvers.

Before describing the idea of our new solver, we begin with a short introduction of existing direct solution methods. Single processor direct solvers for FE computations are typically frontal solvers or multi-frontal solvers. The *frontal solver* [11, 7] browses finite elements, one-by-one, to aggregate d.o.f. Fully assembled d.o.f. are eliminated from the single front matrix. The *multi-frontal solver* [9, 8] constructs the assembly tree based on the analysis of the geometry of the computational mesh. Finite elements are joint into pairs and fully assembled d.o.f. are eliminated within frontal matrices associated to multiple branches of the tree. The process is repeated until the root of the assembly tree is reached. Finally, the common interface problem is solved and partial backward substitutions are recursively called on the assembly tree.

Parallel direct solvers can be either executed on the computational mesh partitioned into sub-domains or on a single piece of data. The domain decomposition can be done with either overlapping or non-overlapping sub-domains [10].

The *sub-structuring method* [10] utilized over non-overlapping sub-domains consists of elimination of the internal d.o.f. of each sub-domain with respect to the interface d.o.f., followed by solving the interface problem, and finally solving the internal problems by backward substitution on each sub-domain.

The elimination of the sub-domains internal d.o.f. can be either performed by the frontal or multi-frontal solver. *Multiple fronts solver* is the parallel solver working on multiple non-overlapping sub-domains, performing partial frontal decomposition on each of the local matrices, summing up the local contributions to the interface problem, and solving the interface problem by using the frontal solver [21].

The interface problem can be solved either by the sequential or parallel solver. The sub-structuring method with a single instance of a parallel direct solver used to solve the interface problem is called the *direct sub-structuring method* [10].

Following [10], the execution of a massively parallel multiple frontal solver on the complete system with distributed entries is referred as *sparse direct method*.

An example of a multi-frontal solver that can be executed either on a single processor, or as a massively parallel solver on multiple processors is the MUMPS solver [1, 2, 3, 14]. A comparison of a direct sub-structuring solver (where several sequential instances of MUMPS are called in parallel to build local Schur complement matrices, and a single parallel instance of MUMPS is called to solve the distributed unassembled global Schur complement system) with a sparse direct method (utilizing a single parallel instance of MUMPS to solve the global problem) is presented in [10]. The paper illustrates how the direct sub-structuring is faster than the execution of the massively parallel multi-frontal MUMPS solver on the complete system with distributed entries (called the sparse direct method by [10]), when applied to a few particular examples.

In this paper we propose a new parallel direct solver. The solver works on the three level elimination trees: the refinement trees that grow from initial mesh elements each time an element is h refined; the tree of connectivities of the initial mesh elements; and the tree of connectivities of sub-domains resulting from the distribution of the computational mesh into multiple sub-domains. Leaves of the refinement trees are the active finite elements. A root of each refinement tree (that is, an initial mesh element) is also a leaf of the initial mesh elements connectivity tree. A root of the tree of initial mesh elements connectivities (that is, an entire sub-domain) is also a leaf of the sub-domains connectivity tree. The solver browses elimination trees starting from the level of active elements, first through the level of refinements, then through the level of initial mesh elements and finally through the level of sub-domains. At every tree node, the partial contributions from children nodes are aggregated, the fully assembled d.o.f. are localized, and the Schur complement of the fully assembled d.o.f. with respect to the unassembled (or only partially assembled) d.o.f. is computed. The procedure is recursively repeated until the root of the sub-domains connectivity tree is reached. Finally, backward substitutions are recursively executed from the root node down to the refinement tree leaves.

The Schur complements computed by the solver are stored at tree nodes. They can be re-utilized by the solver over the unrefined parts of the mesh, for example, for other hp meshes generated by the self-adaptive hp FEM.

The algorithmic differences of the proposed solver with respect to the multi-frontal parallel solver are the following. First, the proposed solver utilizes the knowledge of the history of initial mesh element refinements to construct its refinement tree utilized during the elimination. Second, the solver computes Schur complements at every node of the elimination tree, whilst the multi-frontal solver performs eliminations within frontal matrices. Finally, the proposed solver is able to re-utilize partial Schur complements on unrefined parts of the mesh.

The proposed solver has been used on a sequence of automatically generated meshes based on a new formulation [15] utilized for simulating 3D Direct Current (DC) resistivity borehole measure-

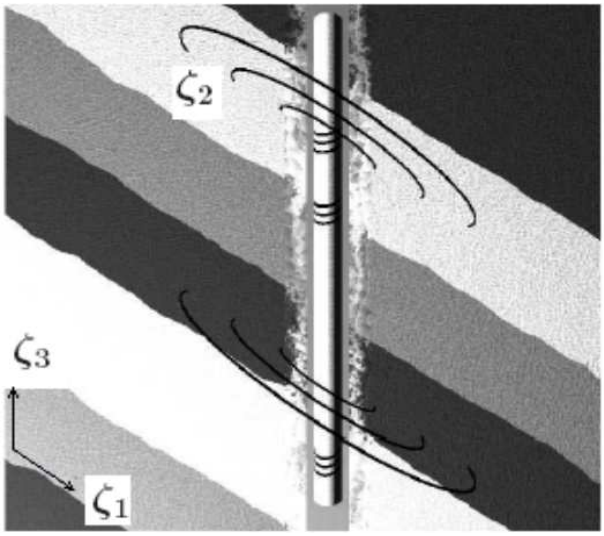

Figure 1: The logging instrument, borehole, and formation layers with different resistivities. Non-orthogonal system of coordinates $(\zeta_1, \zeta_2, \zeta_3)$ utilized to reduce 3D resistivity logging measurements in deviated wells to 2D.

ments [16]. We compare the execution time and memory usage of our solver against the parallel MUMPS solver (version $4.7.3$ with patch obtained from MUMPS Authors [4]) with (1) distributed or (2) centralized entries executed over the entire problem, and (3) the direct sub-structuring method with parallel MUMPS solver utilized to solve the interface problem. All tests are performed on three different hp meshes, with finite elements of various sizes and different polynomial orders of approximation.

2 Problem of interest

We apply our solver to simulate 3D DC resistivity logging measurements in deviated wells, a challenging petroleum engineering problem. The problem consists of solving the conductive media equation

$$\nabla \cdot (\sigma \nabla u) = -\nabla \cdot J^{imp} \tag{2.1}$$

in the 3D domain with different formation layers presented in Fig. 1. The resistivity logging tool with receiver and transmitter electrodes is moving along the borehole. The electromagnetic waves generated by the transmitter electrode are reflected from formation layers and recorded by the receiver electrodes. Of particular interest to the oil industry are 3D simulations of resistivity measurements in deviated wells, where the angle between the borehole and the formation layers is not perpendicular ($\theta_0 \neq 90\,\mathrm{deg}$).

Three non-orthogonal systems of coordinates presented in Fig. 1 are introduced [15]. The variational formulation with respect to the electric potential u in the new system of coordinates can

be expressed in the following way

$$Find \quad u \in u_D + H_D^1(\Omega): \\ < \frac{\partial u}{\partial \xi}, \hat{\sigma} \frac{\partial v}{\partial \xi} > = < v, \hat{f} > \quad \forall v \in H_D^1(\Omega). \tag{2.2}$$

Here the electric conductivity of media $\hat{\sigma} := J^{-1}\sigma J^{-1^T}$, the right-hand-side $\hat{f} = f\|J\|$ with $f = \nabla J^{imp}$ being gradient of the impressed current, are defined by using

$$J = \frac{\partial(x_1, x_2, x_3)}{\partial(\zeta_1, \zeta_2, \zeta_3)} \tag{2.3}$$

the Jacobian matrix of the change of variables from the Cartesian reference system of coordinates (x_1, x_2, x_3) to the non-orthogonal system of coordinates $(\zeta_1, \zeta_2, \zeta_3)$. The Fourier series expansion in terms of the quasi-azimuthal ζ_2 direction is taken with respect to the potential field, material data, and source term

$$u(\zeta_1, \zeta_2, \zeta_3) = \sum_{l=-\infty}^{+\infty} u_l(\zeta_1, \zeta_3) e^{jl\zeta_2}, \tag{2.4}$$

$$\sigma(\zeta_1, \zeta_2, \zeta_3) = \sum_{m=-\infty}^{+\infty} \sigma_m(\zeta_1, \zeta_3) e^{jm\zeta_2}, \tag{2.5}$$

$$f(\zeta_1, \zeta_2, \zeta_3) = \sum_{n=-\infty}^{+\infty} f_n(\zeta_1, \zeta_3) e^{jn\zeta_2}. \tag{2.6}$$

The final variational formulation can be expressed in the following way [15]

$$Find \quad u \in u_D + H_D^1(\Omega): \\ \sum_{n=k-2}^{k+2} < \left(\frac{\partial u}{\partial \xi}\right)_k, \hat{\sigma}_{k-n} \left(\frac{\partial v}{\partial \xi}\right)_n >_{L^2(\Omega_{2D})} = < v_k, \hat{f}_n >_{L^2(\Omega_{2D})} \quad \forall v_k. \tag{2.7}$$

We use 2D hp-adaptive grids over the meridian components (ζ_1, ζ_3) of the solution, and a uniform grid over the quasi-azimuthal component (ζ_2) of the solution, with a fixed number of basis functions. These basis functions for the quasi-azimuthal component (ζ_2) correspond to the basis functions of a Fourier series expansion in a non-orthogonal system of coordinates. See [15] for details.

3 hp finite element meshes and the elimination tree

As mentioned before, the solver presented in this paper is applied to optimal 2D hp-meshes with a fixed number of basis functions in the third (quasi-azimuthal) spatial dimension. The rectangular *2D hp finite element* consists of four vertices, four edges and one interior. The polynomial orders of approximation on the four edges of an element K are denoted by p_i^K $i = 1, ..., 4$. The interior

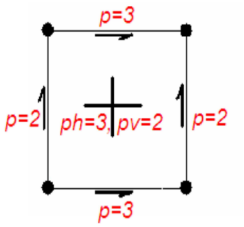

Figure 2: An example of the rectangular hp finite element. Four vertices, two horizontal edges with polynomial order of approximation $p = 3$, two vertical edges with $p = 2$ and the interior with $p_h = 2$ and $p_v = 2$.

of an element has two polynomial orders of approximation, in the horizontal and vertical directions, denoted as $\left(p_h^K, p_v^K\right)$, respectively. An example of hp finite element with horizontal orders of approximation equal to 2 and vertical orders equal to 3 is presented in Fig. 2.

The parallel solver introduced in this paper is applied to *hp meshes* typically containing many hp finite elements. The hp mesh is decomposed into multiple sub-domains. The load balancing is performed based on the computational cost estimation over each finite element, related to the cost of integration and the cost of elimination of interior d.o.f. [17] which can be expressed by

$$c_K = \left(p_h^K + 1\right)^3 \left(p_v^K + 1\right)^3 . \tag{3.8}$$

The presented computations are performed on regular rectangular geometries, thus, we simply order all finite elements row-wise, and compute a sum of load estimations (3.8) over all finite elements $K \in T_h$.

$$c_{total} = \sum_K \left(p_h^K + 1\right)^3 \left(p_v^K + 1\right)^3 . \tag{3.9}$$

The total cost (3.9) is divided by the number of processors $NRPROC$ in order to obtain the average load per processor.

$$c_{ave} = \frac{c_{total}}{NRPROC}; \tag{3.10}$$

All elements are browsed row-wise, assigned to processors, and the element loads are summed up. Each time the summed load is greater than the average load (3.10), the load counter is re-initialized and further elements are assigned to the next available processor.

The solver constructs the *three levels elimination tree* with the level of sub-domains resulting from the domain decomposition, the level of initial mesh elements and the level of mesh refinements. An example of the three-level elimination tree is described in Fig. 3. In this example, the initial mesh with four elements has been distributed into two sub-domains. Each finite element has been broken into four element sons. Different processors are assigned to the leaves of the sub-domain tree. The parent nodes inherit all processors assigned to child nodes. The lists of processors assigned to tree nodes are then utilized in the parallel solver algorithm.

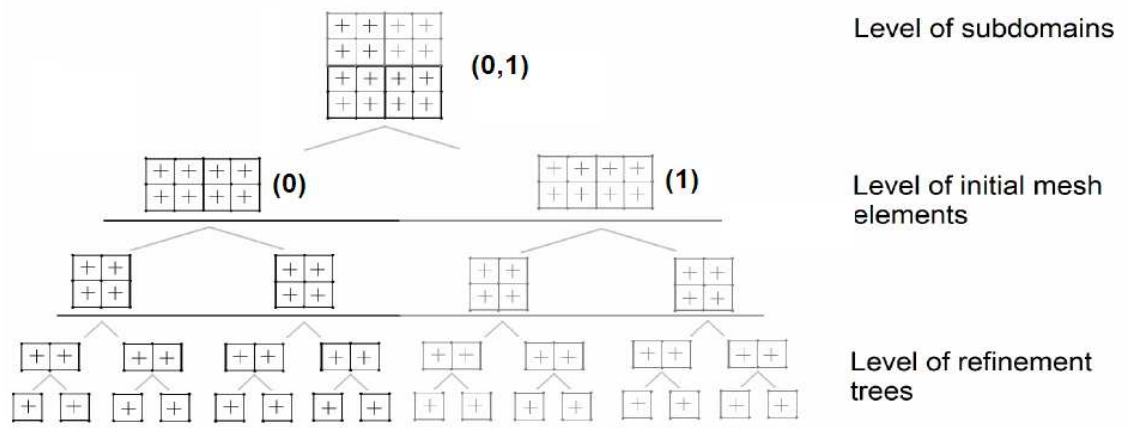

Figure 3: Tree of sub-domains, tree of initial mesh elements, and the refinements tree.

4 Multi level direct sub-structuring multi-frontal parallel solver

Our new parallel solver is described by the following algorithm. The input for the algorithm is the elimination tree with assigned processors, built on every sub-domain. The procedure is called on every processor with the root of the elimination tree on the level of sub-domains.

```
matrix function recursive_solver(tree_node,level)
select case level
case level of sub-domains tree
  if only one processor is assigned to tree_node then
    get root_node of tree of initial mesh elements assigned
      to current processor
    new_level = initial mesh elements tree
    return recursive_solver(root_node,new_level)
  else
    reset new_matrix
    do for each son_node of tree_node
      if son_node is assigned to current processor then
        matrix_contribution = recursive_solver(son_node,level)
        merge matrix_contribution into new_matrix
        if current processor k is the first processor assigned
            to tree_node then
          merge matrix_contribution into new_matrix
          receive matrix_contribution from processor 2k+1
          merge matrix_contribution into new_matrix
        else if current processor is the n/2+1 processor in the group
```

```
            of n processors assigned to tree_node then
          send matrix_contribution to the first processor in the group
        endif
      endif
    enddo
    if current processor k is the first processor assigned
        to tree_node then
      decide which d.o.f.  can be eliminated from new_matrix
      compute Schur complement and store at tree node
      return Schur complement sub-matrix
    endif
case level of initial mesh elements connectivity tree
  if only one initial mesh element is assigned to tree_node then
    get root_node of tree of refinements tree assigned to
      current initial mesh element
    new_level = refinements tree
    return recursive_solver(root_node,new_level)
  else
    do for each son_node of tree_node
      matrix_contribution = recursive_solver(son_node,level)
      merge matrix_contribution into new_matrix
    enddo
    decide which d.o.f.  can be eliminated from new_matrix
    compute Schur complement and store at tree node
    return Schur complement sub-matrix
  endif
case level of refinements tree
  if this is an active element, a leaf of refinement tree then
    compute active element local stiffness matrix
    decide which d.o.f.  can be eliminated from new_matrix
    compute Schur complement and store at tree node
    return Schur complement sub-matrix
  else
    do for each son_node of tree_node
      matrix_contribution = recursive_solver(son_node,level)
      merge matrix_contribution into new_matrix
    enddo
    decide which d.o.f.  can be eliminated from matrix
```

return `Schur complement sub-matrix`
endif

The algorithm stores partial Schur complements at elimination tree nodes. It is possible to re-utilize these partial Schur complements on subsequent meshes generated by the self-adaptive hp FEM. Specifically, each time the hp mesh is refined, the partial Schur complements can be re-utilized on unrefined parts of the mesh. In other words, it is not necessary to recompute Schur complements within branches of the elimination tree associated with unrefined parts of the mesh.

We utilize the sequential MUMPS solver [1, 2, 3, 14] to compute Schur complements at tree nodes. In the current version, the right-hand-side is stored as an additional column of the matrix, to enforce inclusion of the right-hand-side into the Schur complement procedure, so the matrix is assumed to be non-symmetric. The Schur complement procedure can be understood as partial forward elimination that is stopped before processing unassembled d.o.f.

$$\left[\begin{array}{ccc} A_{11} & A_{12} & b_1 \\ A_{21} & A_{22} & b_2 \\ 0 & 0 & 1 \end{array}\right] \mapsto \left[\begin{array}{ccc} U_{11} & U_{12} & \hat{b}_1 \\ 0 & \hat{A}_{22} & \hat{b}_2 \\ 0 & 0 & 1 \end{array}\right]. \tag{4.11}$$

The MUMPS solver returns to the user the $\hat{A}_{22}$ and $\hat{b}_2$ parts of (4.11), together with the last row of the matrix. The backward substitution can be executed after replacing the Schur complement sub-matrix by the identity matrix I and placing the known solution $\hat{x}_2$ coming from the parent tree node into the right-hand-side:

$$\left[\begin{array}{cc} U_{11} & U_{12} \\ 0 & I \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \end{array}\right] = \left[\begin{array}{c} \hat{b}_1 \\ \hat{x}_2 \end{array}\right]. \tag{4.12}$$

Each time we get the Schur complement at any tree node, we turn off the current sequential instance of the MUMPS solver to reduce the memory usage. Since MUMPS does not return U_{11} and U_{12} [4], we recompute U_{11} and U_{12} during backward substitution, by performing full forward elimination on the tree node sub-matrix. Note that this full forward elimination is much faster than obtaining a Schur complement, since the MUMPS has much more freedom to select optimal ordering of d.o.f. (this time non-assembled d.o.f. are also eliminated).

5 Numerical experiments

5.1 Measurements for our new parallel solver

The proposed solver has been tested on the following hp meshes on the LONESTAR [13] cluster from the Texas Advanced Computing Center (TACC).

1. The first mesh has 2304 active finite elements and uniform $p = 3$ throughout the entire mesh.

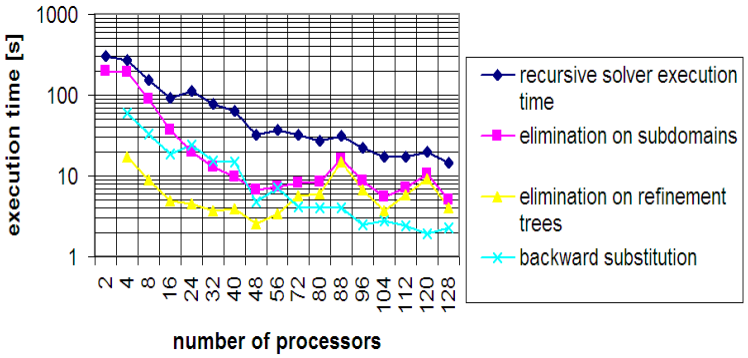

Figure 4: Execution times of our new parallel solver, measured on the first mesh.

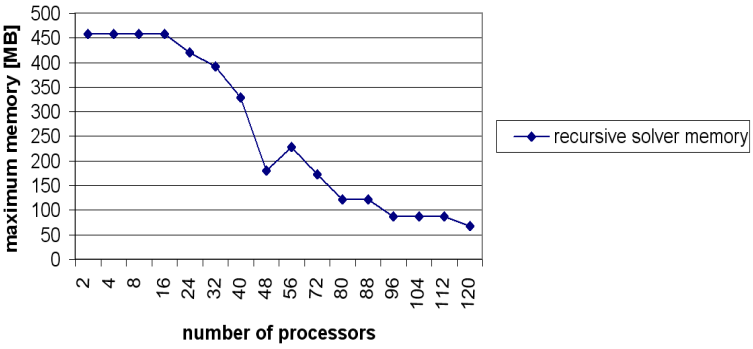

Figure 5: Maximum memory usage of our new parallel solver, measured on the first mesh.

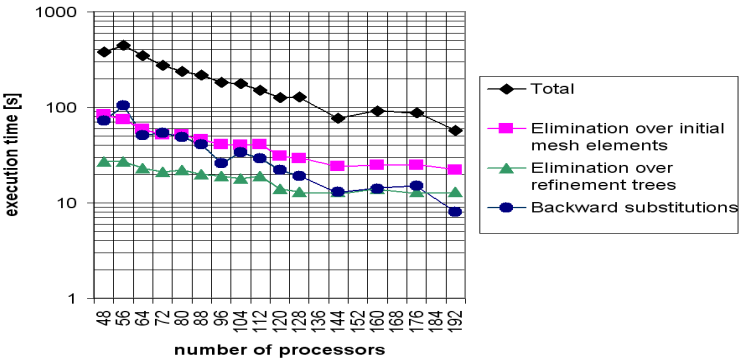

Figure 6: Execution times of our new parallel solver, measured on the second mesh.

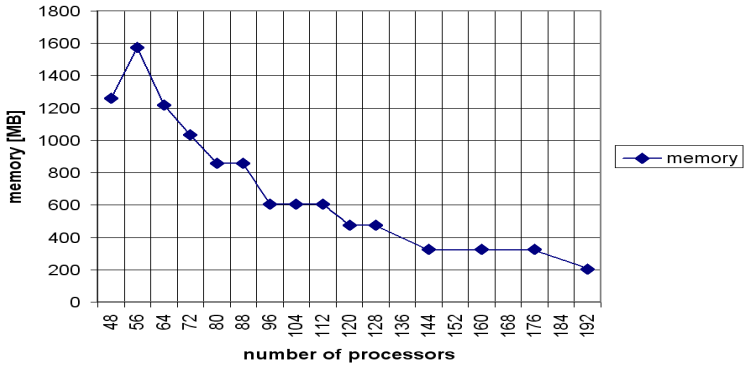

Figure 7: Maximum memory usage of our new parallel solver, measured on the second mesh.

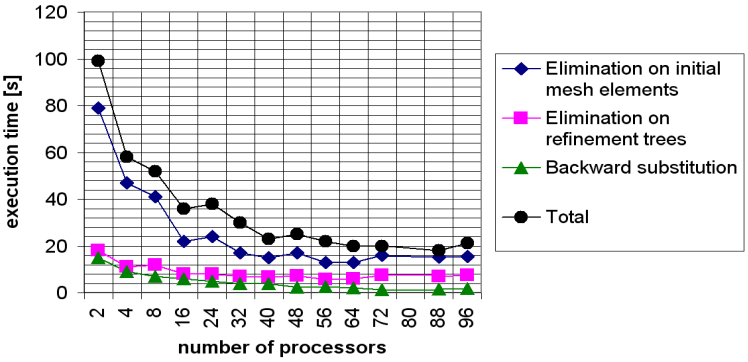

Figure 8: Execution times of our new parallel solver, measured on the third mesh.

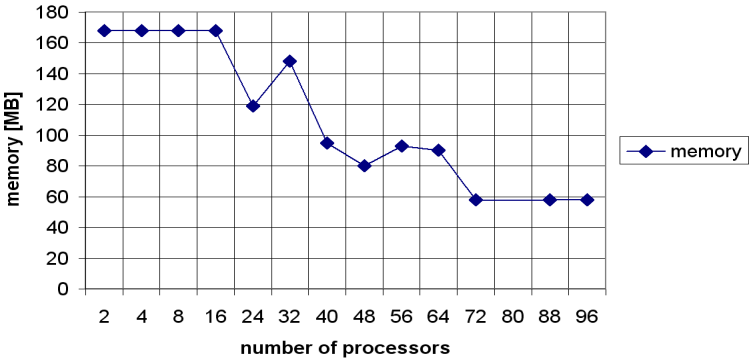

Figure 9: Maximum memory usage of our new parallel solver, measured on the third mesh.

Table 1: Number of degrees of freedom N and number of non-zero entries NZ

Mesh	First, uniform $p = 3$	Second, uniform $p = 4$	Third, non-uniform
N	315,555	1,482,570	51,290
NZ	10,745,846	68,826,475	1,666,190

2. The second mesh has 9216 active finite elements and uniform $p = 4$.

3. The third mesh is the optimal mesh obtained by performing 10 iterations of the self-adaptive hp FEM. The mesh is highly non-uniform with polynomial order of approximation varying from $p = 1, ..., 8$.

We employ 10 basis functions in the azimuthal direction on every mesh. The angle between the borehole and the formation layers is $\theta_0 = 60$ deg.

There are 576 initial mesh elements on every mesh. The first mesh has been obtained by performing one *global hp refinement*, i.e. each initial mesh element has been broken into 4 element sons, and the polynomial order of approximation has been uniformly raised by one (from $p = 2$ to $p = 3$). It implies the depth of the refinement trees to be equal to 2 on the first mesh (see Fig. 3). The second mesh has been obtained by performing two global hp refinements, thus the depth of the refinement tree is equal to 4. The third non-uniform mesh presented in Fig. 10 has been obtained by performing multiple h, p or hp refinements (selected by the self-adaptive hp FEM algorithm). The number of d.o.f. as well as the number of non-zero entries are presented in Tab. 1.

The measurements presented in Figures 4, 6, and 8 describe the maximum (over processors) time spent for sequential elimination over refinement trees and over sub-domains, as well as the total time spent on backward substitutions, including regeneration of LU factorizations by performing full forward eliminations. The logarithmic scale is utilized for time in these figures. Figures 5, 7, and 9 display the maximum memory usage, where the maximum is taken over all nodes of the distributed elimination tree.

From presented measurements it follows that our new parallel solver scales very well up to maximum number of utilized processors, both in terms of the execution time and memory usage.

5.2 Comparison with different versions of parallel MUMPS solver

Our new solver has been compared with three versions of parallel MUMPS solver:

1. The parallel MUMPS solver with distributed entries (the input matrix stored in distributed manner, submitted from all processors in assembled format).

2. The parallel MUMPS solver with centralized entries (the input matrix submitted from the host processor in assembled format).

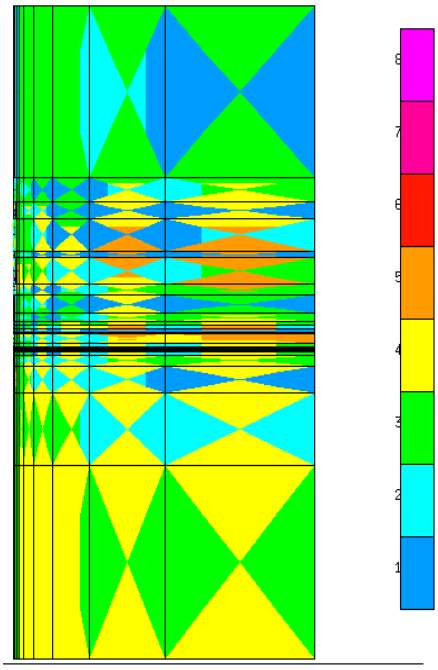

Figure 10: The non-uniform optimal mesh. Different colors denote different polynomial orders of approximation.

3. The direct sub-structuring method with sequential MUMPS solver utilized to compute the Schur complements over sub-domains and parallel MUMPS solver with distributed entries utilized to solve the interface problem.

The ”Preparation” in Figures 11, 13, 17 and 21 stands for integration of local matrices over hp finite elements. In the case of MUMPS-based direct sub-structuring method presented in Figures 15, 19 and 23 the “Preparation on sub-domains” stands for the integration, and the ”Preparation of interface problem” stands for transferring Schur complement outputs into lists of non-zero entries for the MUMPS parallel solver with distributed entries.

We draw the following conclusions from presented measurements.

1. Our solver is slower than parallel MUMPS solvers in a low number of processors because of the following reasons:
   - Our algorithms generating local numbering of matrices at tree nodes and making decisions about d.o.f. that can be eliminated are not optimized.
   - All matrices are assumed to be non-symmetric, to be able to perform the trick described in Section 4 with the right-hand-side as an additional column, whilst the tested MUMPS parallel solvers work on symmetric matrices.

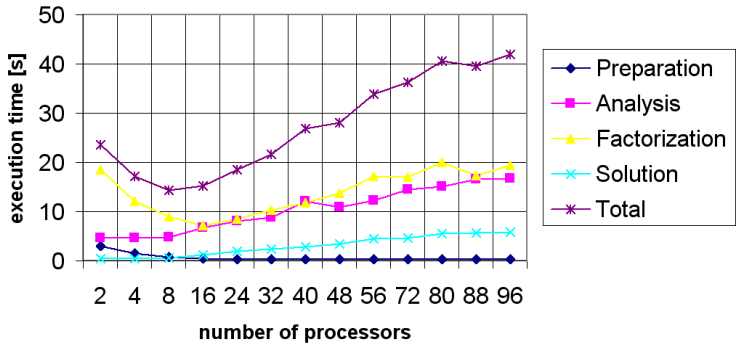

Figure 11: Execution time of the parallel MUMPS solver with distributed entries, measured on the first mesh.

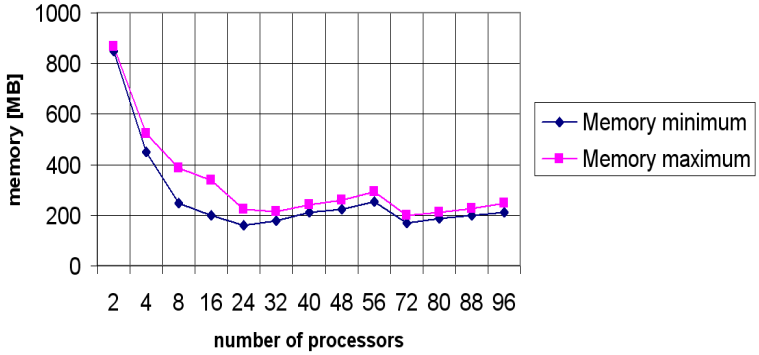

Figure 12: Minimum and maximum memory usage of the parallel MUMPS solver with distributed entries, measured on the first mesh.

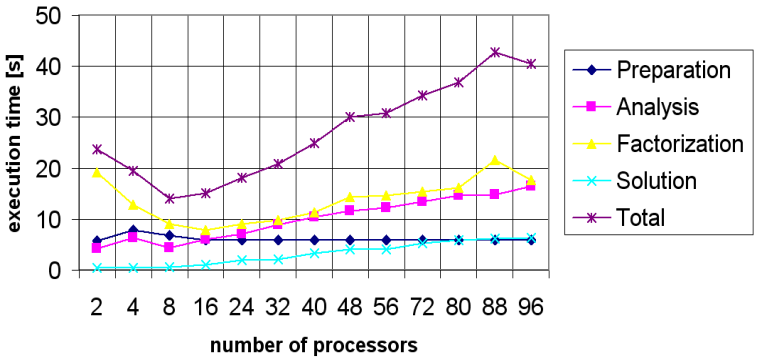

Figure 13: Execution time of the parallel MUMPS solver with centralized entries, measured on the first mesh.

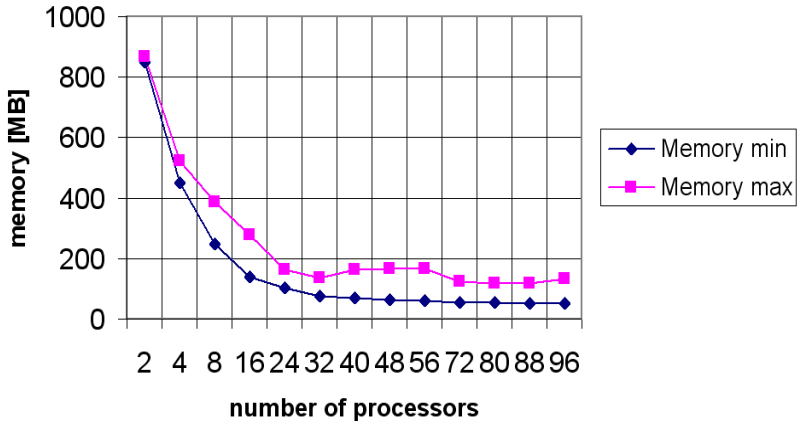

Figure 14: Minimum and maximum memory usage of the parallel MUMPS solver with centralized entries, measured on the first mesh.

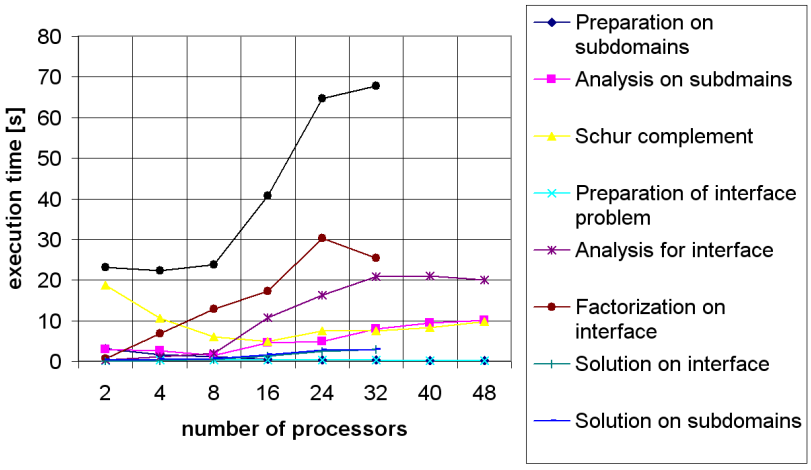

Figure 15: Execution times of particular parts of the MUMPS-based direct sub-structuring method, measured on the first mesh.

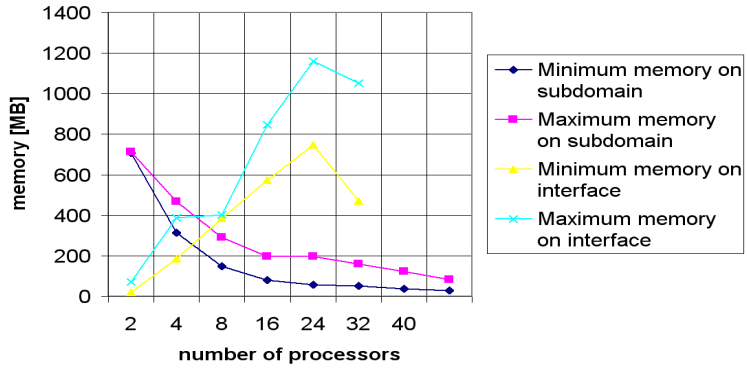

Figure 16: Minimum and maximum memory usage of the MUMPS-based direct sub-structuring method, measured on the first mesh.

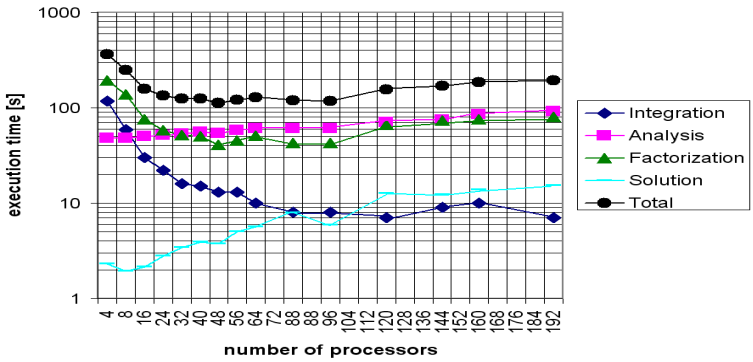

Figure 17: Execution time of the parallel MUMPS solver with distributed entries, measured on the second mesh.

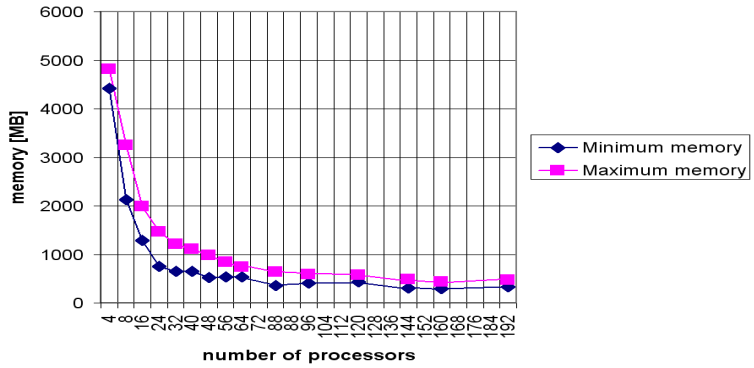

Figure 18: Minimum and maximum memory usage of the parallel MUMPS solver with distributed entries, measured on the second mesh.

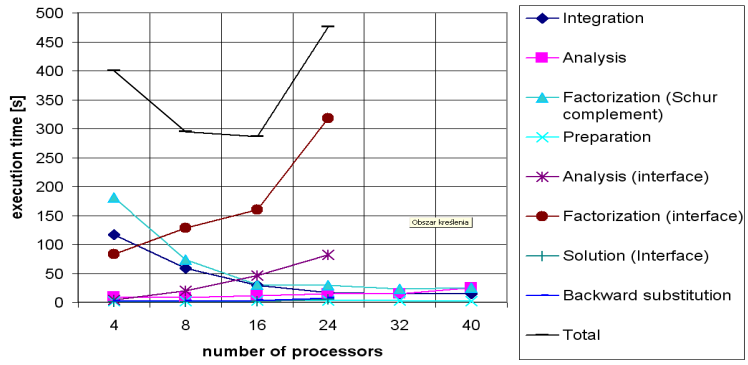

Figure 19: Execution times of particular parts of the MUMPS-based direct sub-structuring method, measured on the second mesh.

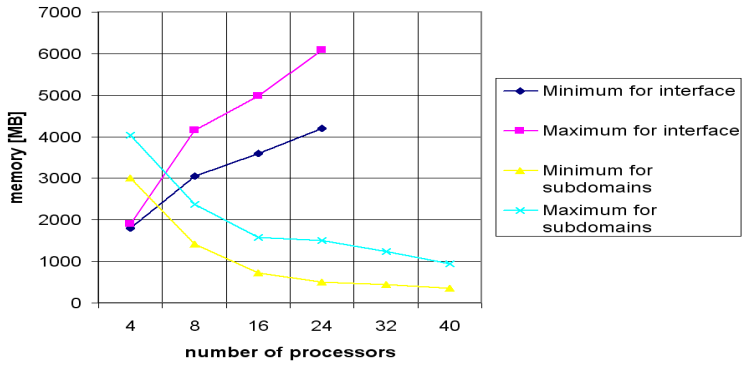

Figure 20: Minimum and maximum memory usage of the MUMPS-based direct sub-structuring method, measured on the second mesh.

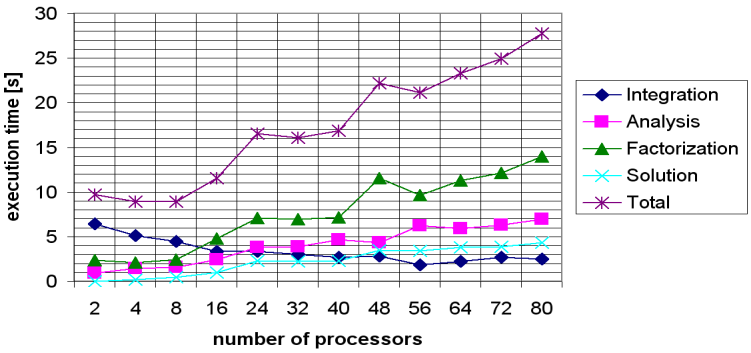

Figure 21: Execution time of the parallel MUMPS solver with distributed entries, measured on the third mesh.

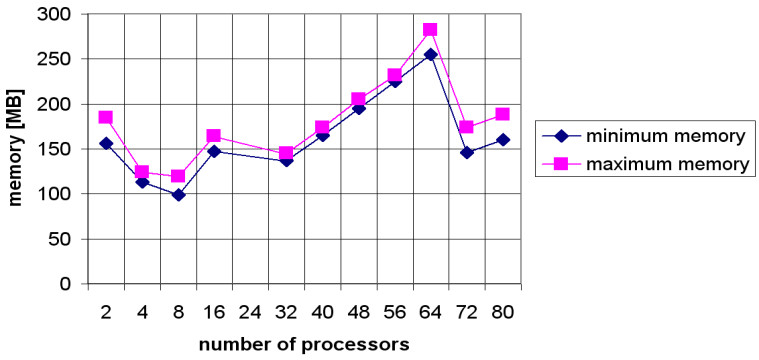

Figure 22: Minimum and maximum memory usage of the parallel MUMPS solver with distributed entries, measured on the third mesh.

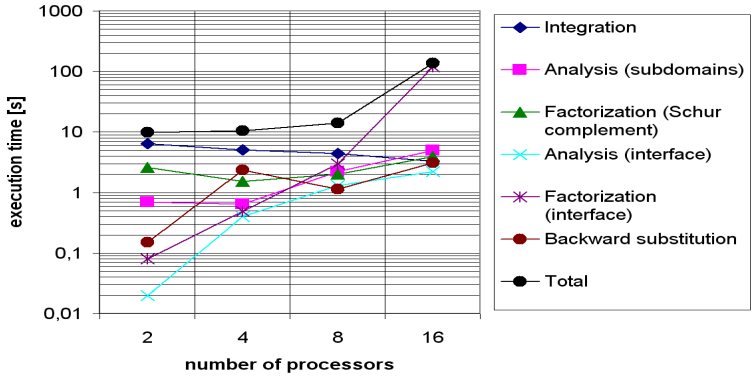

Figure 23: Execution times of particular parts of the MUMPS-based direct sub-structuring method, measured on the third mesh.

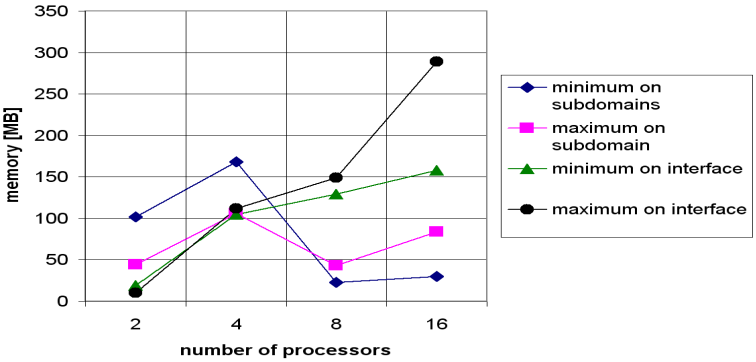

Figure 24: Minimum and maximum memory usage of the MUMPS-based direct sub-structuring method, measured on the third mesh.

- The backward substitution is accompanied by additional full forward elimination, necessary to regenerate the structure of the LU factorization.
- The sub-domains tree as well as initial mesh elements trees in the current version of the solver are constructed by a simple algorithm that builds binary trees based on the natural order of elements. In other words, the order of elimination of d.o.f. within these trees may be not optimal.

On the other hand, our solver scales very well up to maximum number of utilized processors, and becomes up to two times faster than the MUMPS solver for a large number of processors (57 seconds for our solver on the second mesh versus 112 seconds of the parallel MUMPS with distributed entries).

2. The MUMPS parallel solver with distributed entries scales in a very similar way to the MUMPS parallel solver with centralized entries.

3. The MUMPS-based direct sub-structuring method usually runs out of memory for large number of processors (it requires more than 8096 MB of memory per processor).

4. All presented problems are relatively sparse, and all MUMPS-based parallel solvers reach the minimum execution time on 8 or 16 processors. However, the memory usage for all three types of MUMPS-based solvers is large. The memory usage usually stabilizes for the parallel MUMPS with distributed or centralized entries, but the execution time increases. On the other hand, the memory usage of our new solver is usually lower than for any MUMPS solver.

6 Conclusions and future work

- We proposed a new parallel direct solver for hp FEM, based on the three level elimination trees: sub-domains, initial mesh elements, and refinement trees. The solver utilizes the knowledge of the history of refinements to construct the third level elimination tree. Future work involves interfacing of the solver with graph partitioning algorithms (like METIS [12]) to optimize the structure of initial mesh elements and sub-domains elimination trees.

- The solver computes partial Schur complements that could be stored at elimination tree nodes to be re-utilized on unrefined parts of the mesh.

- The solver scales well up to the maximum number of available processors (256), reached MUMPS execution time and even became twice as fast for the largest considered mesh.

- The solver allowed to minimize the memory usage, and actually utilized less memory than MUMPS solver. This is crucial in real applications, since we very often run out of memory when computing real-life 3D DC borehole resistivity measurement simulations.

- The new solver could be very easily implemented out-of-core, just by storing partial Schur complements on disk instead of within elimination tree nodes.

7 Acknowledgments

The work reported in this paper was supported by the Foundation for Polish Science under Homing Programme, and by The University of Texas at Austin Research Consortium on Formation Evaluation, jointly sponsored by Aramco, Baker Atlas, BP, British Gas, ConocoPhilips, Chevron, ENI E&P, ExxonMobil, Halliburton Energy Services, Hydro, Marathon Oil Corporation, Mexican Institute for Petroleum, Occidental Petroleum Corporation, Petrobras, Schlumberger, Shell International E&P, Statoil, TOTAL, and Weatherford. We would like to acknowledge the Texas Advanced Computing Center (TACC) used for the parallel computations.

References

[1] P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent, Multifrontal parallel distributed symmetric and unsymmetric solvers, in Comput. Methods in Appl. Mech. Eng. 184 (2000) 501-520.

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

[3] P. R. Amestoy and A. Guermouche and J.-Y. L'Excellent and S. Pralet, Hybrid scheduling for the parallel solution of linear systems. Accepted to Parallel Computing (2005).

[4] P. R. Amestoy, Personal Communication, July 2007.

[5] Demkowicz L., Computing with hp-Adaptive Finite Elements, Vol. I. Chapman & Hall/Crc Applied Mathematics & Nonlinear Science (2006)

[6] Demkowicz L., Pardo D., Rachowicz W., 3D hp-Adaptive Finite Element Package (3Dhp90) Version 2.0, The Ultimate Data Structure for Three-Dimensional, Anisotropic hp Refinements, TICAM Report 02-24 (2002).

[7] I. S. Duff and J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear systems, ACM Trans. on Math. Soft., 9 (1983) 302-325.

[8] I. S. Duff and J. K. Reid, The multifrontal solution of unsymmetric sets of linear systems, SISSC, 5 (1984) 633-641

[9] Geng P., Oden T.J., van de Geijn R.A., A Parallel Multifrontal Algorithm and Its Implementation, Computer Methods in Applied Mechanics and Engineering 149 (2006) 289-301.

[10] L. Giraud, A. Marocco and J.-C. Rioual, Iterative versus direct parallel substructuring methods in semiconductor device modeling, Numerical Linear Algebra with Applications, 12, 1 (2005) 33-55.

[11] B. Irons, A frontal solution program for finite-element analysis, Int. J. Num. Meth. Eng, 2 (1970) 5-32.

[12] Karypis G., Kumar V., A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing, 20, 1 (1999) 359 - 392.

[13] Lonestar Cluster Users' Manual *http://www.tacc.utexas.edu/services/userguides/lonestar* (2007)

[14] A MUltifrontal Massively Parallel sparse direct Solver version 4.7.3 *http://www.enseeiht.fr/lima/apo/MUMPS/* (2007)

[15] Pardo D., Calo V., Torres-Verdin C., Nam M. J., Fourier Series Expansion in a Non-Orthogonal System of Coordinates for Simulations of 3D Borehole Resistivity Measurements. Part I: DC, ICES-Report 2007-09-20

[16] Pardo D., Demkowicz L., Torres-Verdin C., Paszyski M., Simulation of Resistivity Logging-While-Drilling (LWD) Measurements Using a Self-Adaptive Goal-Oriented hp-Finite Element Method, SIAM Journal on Applied Mathematics 66 (2006) 2085-2106.

[17] Paszyński M., Agent Based Hierarchical Parallelization of Complex Algorithms on the Example of hp Finite Element Method, Lecture Notes in Computer Science (2007) in press.

[18] Paszyński, M., Demkowicz, L., Parallel Fully Automatic hp-Adaptive 3D Finite Element Package, *Engineering with Computers* 22, 3-4 (2006) 255-276.

[19] Paszyński M., Kurtz J., Demkowicz L., Parallel Fully Automatic hp-Adaptive 2D Finite Element Package, *Computer Methods in Applied Mechanics and Engineering*, 195, 7-8 (2006) 711-741.

[20] Rachowicz, W., Pardo D., Demkowicz, L., Fully Automatic hp-Adaptivity in Three Dimensions, Computer Methods in Applied Mechanics and Engineering, 195, 37-40 (2006) 4816-4842.

[21] J. A. Scott, Parallel Frontal Solvers for Large Sparse Linear Systems, ACM Trans. on Math. Soft., 29, 4 (2003) 395-417.

[22] "Zoltan: Data-Management Services for Parallel Applications", *http://www.cs.sandia.gov/Zoltan/* (2007)