PERFORMANCE OF A MULTI-FRONTAL PARALLEL DIRECT SOLVER FOR HP-FINITE ELEMENT METHOD

Maciej Paszyński
Department of Computer Science,
AGH University of Science and Technology, Kraków, Poland
e-mail: paszynsk@agh.edu.pl

David Pardo
Basque Center for Applied Mathematics, Bilbao, Spain
e-mail: dzubiaur@gmail.com

Carlos Torres-Verdín
Department of Petroleum and Geosystems Engineering
The University of Texas in Austin, TX, USA
e-mail: cverdin@uts.cc.utexas.edu

ABSTRACT

In this paper we present the performance of our parallel multi-frontal direct solver when applied to solve linear systems of equations resulting from discretizations of a hp Finite Element Method (hp-FEM). The hp-FEM generates a sequence of computational meshes delivering exponential convergence of the numerical error with respect to the mesh size (number of degrees of freedom). A sequence of meshes is obtained by performing several hp refinements starting from an arbitrary initial mesh. The solver constructs initial elimination tree for an arbitrary initial mesh, and expands the elimination tree each time the mesh is refined. The solver has been tested on 3D Direct Current (DC) borehole resistivity measurement simulations problems. We compare the solver with two versions of the MUMPS parallel solver: with (1) distributed entries executed over the entire problem, and (2) the direct sub-structuring method with parallel MUMPS solver utilized to solve the interface problem. We show that by providing to the solver the knowledge about the structure of the hp-FEM, the order of elimination is obtained straightforward, and leads to a better performance than by submitting the entire matrix to the solver and executing a connectivity graph based ordering algorithm.

KEY WORDS

direct solvers, parallel algorithms, hp adaptivity, finite element method

1 Introduction

The paper presents performance measurements of our new parallel multi-frontal direct solver designed for the hp Finite Element Method (FEM) [1]. Parallel 2D and 3D hp adaptive FE codes [11, 12] in a fully automatic mode generate a sequence of finite element meshes delivering exponential convergence of the numerical error with respect to the mesh size (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. 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 parallel direct solvers, efficiently utilizing the structure of the refined computational meshes.

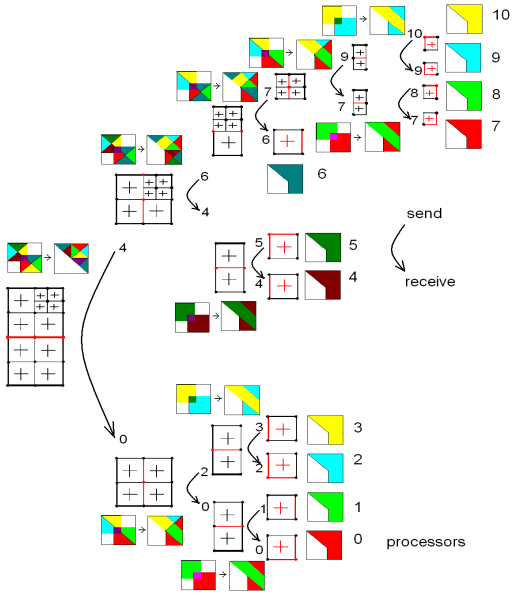

Figure 1. Execution of the parallel solver over the elimination tree constructed for the refined mesh.

The presented solver is an extension of the multi-level parallel direct solver [10]. The previous solver utilized the frontal elimination pattern [4] over sub-domains. It browsed finite elements, one-by-one, to aggregate d.o.f.. Fully assembled d.o.f. were eliminated from the single front matrix. The new solver utilizes the multi-frontal elimination pattern [2] over the computational mesh distributed into sub-domains. The solver algorithm has been summarized in Figure 1. It constructs the elimination tree based

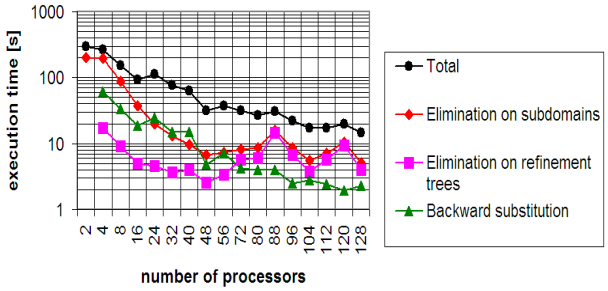

Figure 2. Execution times of our new parallel solver, measured on the first mesh.

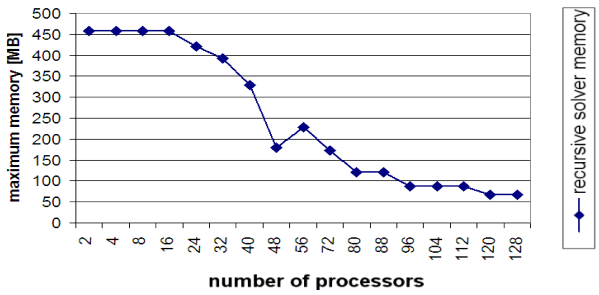

Figure 3. Maximum memory usage of our new parallel solver, measured on the first mesh.

on the analysis of the finite elements connectivity data. The elements are joint into pairs and fully assembled d.o.f. are eliminated within frontal matrices associated to multiple branches of the tree. Finite elements are assigned to processors, and Schur complement contributions are sent between processors assigned to adjacent branches in the tree. The technical details on the solver algorithms and its implementation are presented in the paper [13]. The main advantage of our solver is that the order of elimination follows the refinement trees, as illustrated in Figure 1. The mesh presented in Figure 1 has been obtained by breaking two initial mesh elements into four son elements and by breaking one of the son elements again into four new elements.

In this paper we focus on the comparison of our solver with other reliable multi-frontal parallel solver [8]. The parallel solver has been tested on a challenging problem consisting of simulations of 3D Direct Current (DC) borehole resistivity measurement [9]. The problem has been solved on a sequence of 3D meshes, where we utiliye a Fourier series expansion in the azimuthal direction. We refer to [9] for more details. We compare the solver with two versions of the multi-frontal massively parallel sparse direct solver (MUMPS) [8]: with (1) distributed entries executed over the entire problem, and (2) the direct substructuring [3] method with parallel MUMPS solver utilized to solve the interface problem. We show that by providing to the solver the knowledge about the structure of the hp-FEM, the order of elimination is obtained straightforward, and leads to a better performance than by submitting the entire matrix to the solver and executing a connectivity graph based ordering algorithm.

2 Numerical experiments

2.1 Measurements for our new parallel solver

The proposed solver has been tested on the following hp meshes on the LONESTAR [7] cluster from the Texas Advanced Computing Center (TACC).

1. The first mesh with 2304 finite elements and uniform

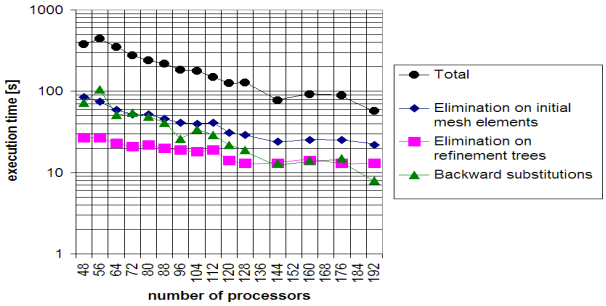

Figure 4. Execution times of our new parallel solver, measured on the second mesh.

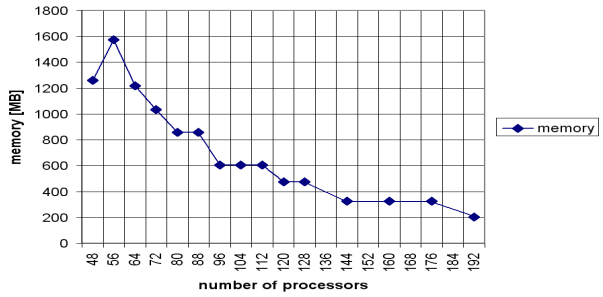

Figure 5. Maximum memory usage of our new parallel solver, measured on the second mesh.

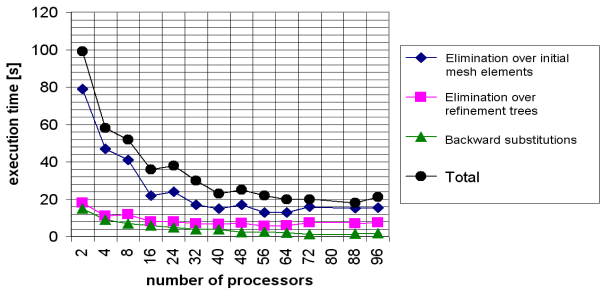

Figure 6. Execution times of our new parallel solver, measured on the third mesh.

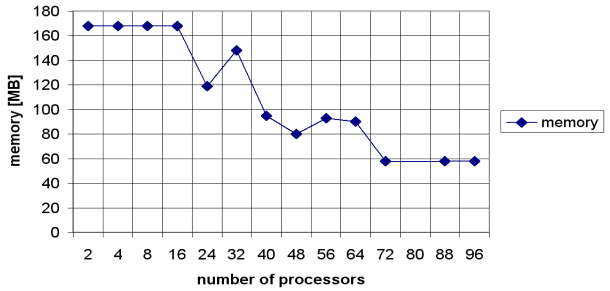

Figure 7. Maximum memory usage of our new parallel solver, measured on the third mesh.

polynomial order of approximation $p = 3$ throughout the entire mesh. There are $315,555$ d.o.f. over the mesh with $10,745,846$ total mumber of non-zeros entries in the matrix.

2. The second mesh with 9216 finite elements and uniform $p = 4$. There are $1,482,570$ d.o.f. over the mesh with $68,826,475$ total mumber of non-zeros entries in the matrix.

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 $p = 1, ..., 8$. There are $51,290$ d.o.f. over the mesh with $1,666,190$ total mumber of non-zeros entries in the matrix.

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 Figure 1). The second mesh has been obtained by performing two global hp refinements, thus the depth of the refinement tree is equal to 3. The third non-uniform mesh has been obtained by performing multiple h, p or hp refinements (selected by the self-adaptive hp FEM algorithm).

The measurements presented in Figures 2, 4, and 6 describe the maximum (over processors) time spent for sequential elimination over refinement trees and over subdomains, 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 figures 2 and 4. The figures 3, 5, and 7 display the maximum memory usage, where the maximum is taken over all nodes of the distributed elimination tree.

The following conclusions can be drawn from the presented measurements. The number of utilized processors is lower than the number of finite elements. It implies that each processor is assigned to a sub-domain with several finite elements. We utilize nested-dissection [6] algorithms to partition the computational mesh into sub-domains. The first and the second meshes have been uniformly hp refined, so each initial mesh element is uniformly loaded.

The maximum speedup of the solver as well as the minimum memory usage are obtained when the structure of the elimination tree is nice (the lengths of all paths, going from the root of the elimination tree down to the leaves, are the same). If the sub-domains have a regular pattern, then the elimination tree also has a regular pattern: the depth of the elimination tree is uniform (all paths from the tree root down to every leaf have the same length). The performance of the solver is worse when the structure of the elimination tree is not uniform, e.g. there is a single longest path from the root of the elimination tree to a single deepest leaf.

For the first mesh, the maximum speedup is obtained for 16 or 48 processors (when there are 16 rectangular subdomains with $576/16 = 6 \times 6$ elements or 48 rectangular sub-domains with $576/48 = 4 \times 3$ elements). Also, the memory usage rapidly decreases for 48 or 96 processors (when there are 48 or 96 rectangular sub-domains with $576/96 = 3 \times 2$ elements). For the second mesh, the maximum speedup is obtained for 144 or 192 processors (when there are 144 rectangular sub-domains with $576/144 = 2 \times 2$ elements or 192 rectangular sub-domains with ($576/192 = 1 \times 3$ elements). The memory usage decreases rapidly for 96, 144 or 192 processors.

The third mesh is not uniformly hp refined, and the mesh partition can be highly non-uniform, so the structure of the elimination tree can be also non-uniform. In this case, the maximum speedup or decrease of memory usage do not follow the above pattern.

2.2 Comparison with different versions of parallel MUMPS solver

Our new solver has been compared with two versions of parallel MUMPS solver with METIS [5] ordering:

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

2. 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 “Integration” in Figures 8, 12 and 16 stands for integration of local matrices over hp finite elements, performed by the interface routine, preparing assembled list of non-zero entries for MUMPS. In the case of the MUMPS-based direct sub-structuring method presented in Figures 10, 14 and 18 the “Integration” stands for the integration over active finite elements, and the ”Preparation” stands for transferring Schur complement outputs into lists of non-zero entries for the MUMPS parallel solver with distributed entries. Notice that the “Integration” stage for MUMPS solver stands

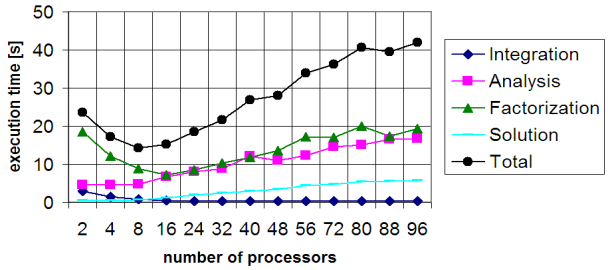

Figure 8. Execution time of the parallel MUMPS solver with distributed entries, measured on the first mesh.

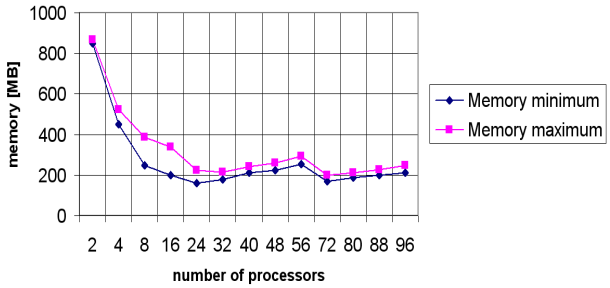

Figure 9. Minimum and maximum memory usage of the parallel MUMPS solver with distributed entries, measured on the first mesh.

for the integration over finite elements, and this integration is performed in a loop, while preparing assembled list of non zero entries in the interface to MUMPS routine. In our parallel solver, the integration over active finite elements is performed in the leaves of the elimination tree. These operations are included in the "Elimination over refinement trees". The "Analysis" stage for MUMPS involves execution of the connectivity graph algorithm (e.g. METIS in this example). In our solver there is no such stage, since the order of elimination is directly obtain from the mesh data structure (the binary tree constructed for the sub-domains and initial mesh elements, and the order of elimination over refinement trees follows the history of refinements, stored in our data structure). The "Factorization" stage for MUMPS involves all "Elimination over initial mesh elements" and "Elimination over refinement trees". There is no way to distinguish these two parts within the MUMPS solver. Our "Backward substitution" is related to the "Solution" stage for MUMPS.

3 Conclusions

We draw the following conclusions from presented measurements. Our solver is slower than parallel MUMPS solvers for a low number of processors. This is because our algorithms generating local numbering of matrices at

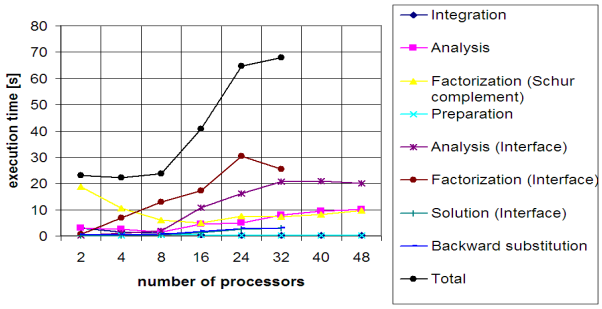

Figure 10. Execution times of particular parts of the MUMPS-based direct sub-structuring method, measured on the first mesh.

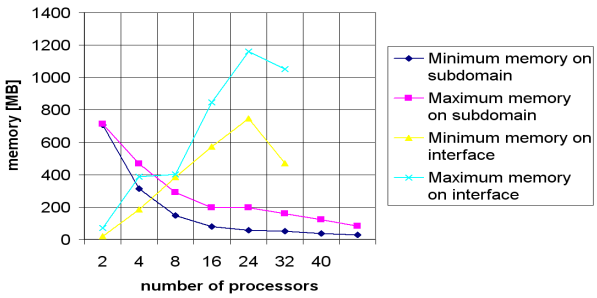

Figure 11. Minimum and maximum memory usage of the MUMPS-based direct sub-structuring method, measured on the first mesh.

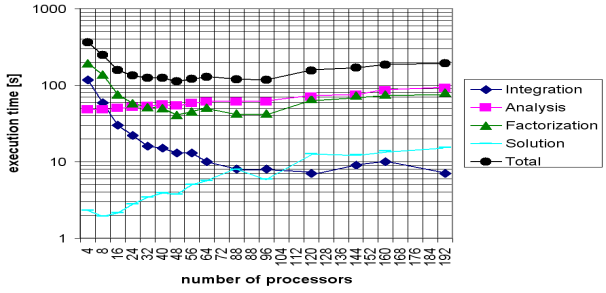

Figure 12. Execution time of the parallel MUMPS solver with distributed entries, measured on the second mesh.

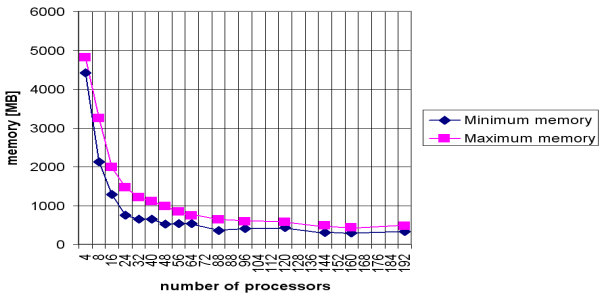

Figure 13. Minimum and maximum memory usage of the parallel MUMPS solver with distributed entries, measured on the second mesh.

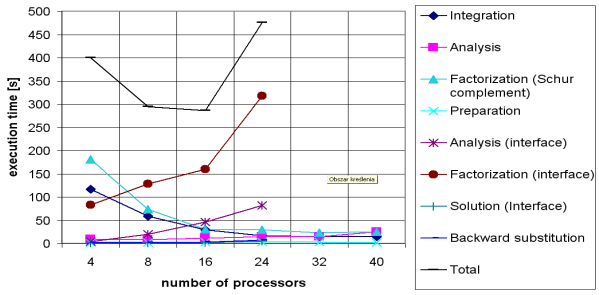

Figure 14. Execution times of particular parts of the MUMPS-based direct sub-structuring method, measured on the second mesh.

tree nodes and making decisions about d.o.f. that can be eliminated are not optimized, and become slow when there are many finite elements assigned to a single sub-domain.

On the other hand, our solver scales very well up to the 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). The MUMPS-based direct sub-structuring method usually runs out of memory for large number of processors (40 or 32 for the first or the second mesh, respectively - it requires more than 8096 MB of memory per processor). Notice that the Schur complements obtained from our solver on the level of leaves of the sub-domains tree *are exactly the same* as the Schur complements requested from the MUMPS solvers in this case. The METIS ordering provided for the MUMPS with the Schur complement restriction over the sub-domain interface seems to be not optimal. On the other side, our algorithm provides quasi-optimal ordering under the same constrains. This results clearly shows that our ordering obtained by utilizing the knowledge about the initial mesh structure, and the history of refinements provides much better scalability than the graph based ordering for this case. The METIS ordering (graph based ordering) is much better when the MUMPS solver gets the entire matrix, but it

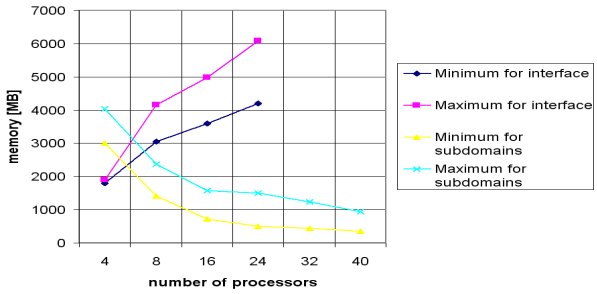

Figure 15. Minimum and maximum memory usage of the MUMPS-based direct sub-structuring method, measured on the second mesh.

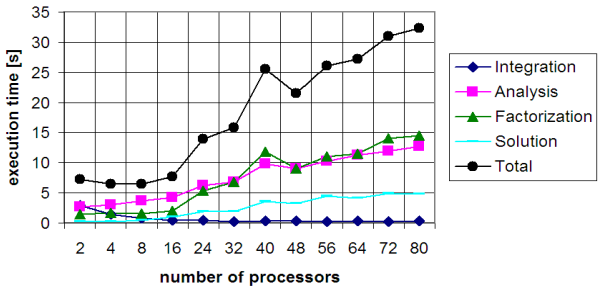

Figure 16. Execution time of the parallel MUMPS solver with distributed entries, measured on the third mesh.

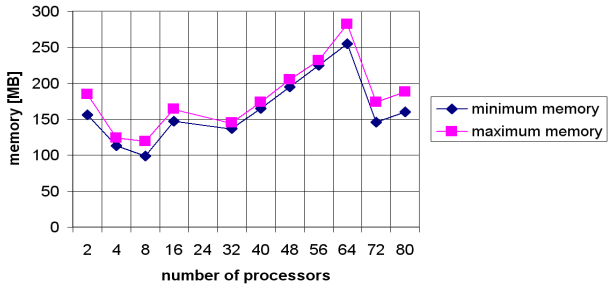

Figure 17. Minimum and maximum memory usage of the parallel MUMPS solver with distributed entries, measured on the third mesh.

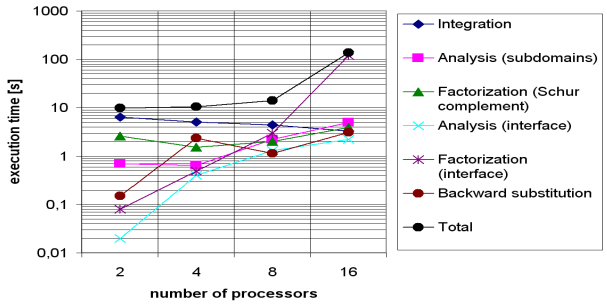

Figure 18. Execution times of particular parts of the MUMPS-based direct sub-structuring method, measured on the third mesh.

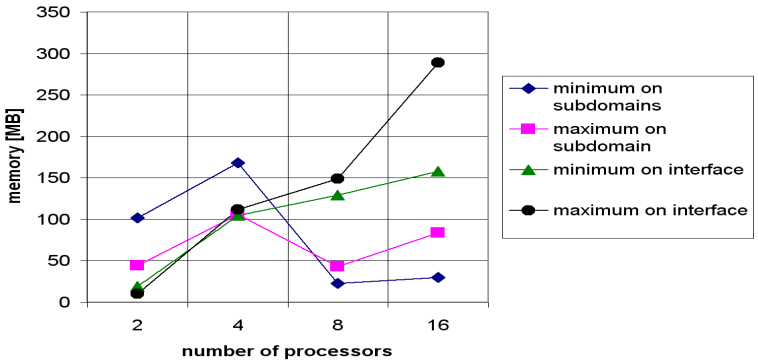

Figure 19. Minimum and maximum memory usage of the MUMPS-based direct sub-structuring method, measured on the third mesh.

is far from the optimal ordering if we provide only a single sub-domain data and request the Schur complement with respect to the sub-domain interface.

All presented problems are 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 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.

Acknowledgements The work presented in this paper has been supported by Polish Ministry of Scientific Research and Information Technology grant no. NN 519 318 635 and by The University of Texas at Austin Research Consortium on Formation Evaluation. We would like to acknowledge the Texas Advanced Computing Center (TACC) for the computational time.

References

[1] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek, *Computing with hp-Adaptive Finite Elements, Vol. II. Frontiers.*, Chapman & Hall/Crc Applied Mathematics & Nonlinear Science (2007)

[2] P. Geng, T. J. Oden, R. A. van de Geijn, A parallel multifrontal algorithm and its implementation, *Computer Methods in Applied Mechanics and Engineering* 149 (2006) 289-301

[3] 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

[4] B. Irons, A frontal solution program for finite-element analysis, *International Journal for Numerical Methods in Engineering*, 2 (1970) 5-32.

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

[6] Khaira M. S., Miller G. L., Sheffler T. J., Nested Dissection: A survey and comparison of various nested dissection algorithms, *CMU-CS-92-106R*, Carnegie Mellon University (1992)

[7] Lonestar Cluster Users' Manual (2008) *http://www.tacc.utexas.edu/services/userguides/lonestar*

[8] A MUltifrontal Massively Parallel sparse direct Solver *http://graal.ens-lyon.fr/MUMPS/* (2008)

[9] D. Pardo, C. Torres-Verdin, M. J. Nam, M. Paszyński, V. Calo, Fourier series expansion in a non-orthogonal system of coordinates for the simulation of 3D DC borehole resistivity measurements, *Computer Methods in Applied Mechanics and Engineering*, 197, 1-3 (2008) 1906-1925

[10] M. Paszyński, Performance of multi level parallel direct solver for hp finite element method, *Lecture Notes in Computer Science* 4967 (2008) 1303-1312

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

[12] M. Paszyński, J. Kurtz, L. Demkowicz, Parallel fully automatic hp-adaptive 2D finite element package, *Computer Methods in Applied Mechanics and Engineering*, 195, 7-8 (2006) 711-741

[13] M. Paszyński, D. Pardo, C. Torres-Verdin, L. Demkowicz, V. Calo, A parallel direct solver for self-adaptive hp-finite element method, submitted to *Journal of Parallel and Distributed Computing* (2008)