Computers and Mathematics with Applications 65 (2013) 1140–1151

Contents lists available at SciVerse ScienceDirect

Computers and Mathematics with Applications

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

A direct solver with reutilization of LU factorizations for *h*-adaptive finite element grids with point singularities

Maciej Paszynski [a,*], David Pardo [b,c], Victor M. Calo [d]

[a] *AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunication, Department of Computer Science, al. A. Mickiewicza 30, 30-059 Krakow, Poland*

[b] *Department of Applied Mathematics, Statistics and Operational Research, University of the Basque Country UPV/EHU, Leioa, Spain*

[c] *IKERBASQUE (Basque Foundation for Science), Bilbao, Spain*

[d] *Center for Numerical Porous Media, Applied Mathematics & Computational Science and Earth Science & Engineering, King Abdullah University of Science and Technology, Thuwal, Saudi Arabia*

ARTICLE INFO

Article history:
Received 27 July 2012
Received in revised form 11 December 2012
Accepted 9 February 2013

Keywords:
Direct solver
Linear computational cost
Reutilization
Finite element method
h adaptation

ABSTRACT

This paper describes a direct solver algorithm for a sequence of finite element meshes that are *h*-refined towards one or several point singularities. For such a sequence of grids, the solver delivers linear computational cost $O(N)$ in terms of CPU time and memory with respect to the number of unknowns N. The linear computational cost is achieved by utilizing the recursive structure provided by the sequence of *h*-adaptive grids with a special construction of the elimination tree that allows for reutilization of previously computed partial LU (or Cholesky) factorizations over the entire unrefined part of the computational mesh. The reutilization technique reduces the computational cost of the entire sequence of *h*-refined grids from $O(N^2)$ down to $O(N)$. Theoretical estimates are illustrated with numerical results on two- and three-dimensional model problems exhibiting one or several point singularities.

1. Introduction

Direct solvers are at the core of many engineering analyses based on the finite element method (FEM) [1–3]. The finite element (FE) solution process starts by describing a physical phenomenon in terms of partial differential equations (PDE) with the corresponding boundary and initial conditions over a prescribed domain. Then, this domain is discretized using a finite element mesh, over which the FE solution of the PDE system is obtained by solving a system of linear equations. Finally, the error of the solution is estimated, and if it is above a prescribed tolerance error, the mesh is further refined and the corresponding system of equations is solved until the solution accuracy is within certain error bounds [1,2].

The multi-frontal solver is the state-of-the-art algorithm for solving linear systems of equations [4,5] when using a direct solver. It is a generalization of the frontal solver algorithm proposed in [6]. The multi-frontal algorithm constructs an assembly tree based on the analysis of the connectivity data. FEs are joint into pairs, and fully assembled unknowns are eliminated within frontal matrices associated with 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 executed over the assembly tree.

The direct solver algorithm can be generalized to the use of matrix blocks associated with nodes of the computational mesh (called supernodes) [7]. This allows for a reduction of the computational cost related to the construction of the

* Corresponding author. Tel.: +48 123283400; fax: +48 126175172.
E-mail addresses: paszynsk@agh.edu.pl, maciej.paszynski@agh.edu.pl (M. Paszynski).

0898-1221/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2013.02.006

elimination tree, since the analysis can then be performed at the level of matrix blocks rather than at the level of particular scalar values. There also exist different implementations of the multi-frontal solver algorithm that target specific computer architectures (see, *e.g.*, [8–11]). Other significant advances in the area of multi-frontal solvers include the design of a hybrid solver, where the elimination tree is cut at some level, and the remaining Schur complements are submitted to an iterative solver [12]. A modification of a direct solver that produces linear computational cost is based on the use of H-matrices [13] with compressed non-diagonal blocks. The main limitation of these solvers is that they produce a non-exact solution.

In this paper, we focus on two and three dimensional finite element problems exhibiting point singularities. The corresponding grids are subject to a sequence of h refinements towards the existing singularities. The resulting recursive structure of these grids enables one to build a direct solver algorithm that has linear computational cost for a *single* grid in a sequence of h-refined grids [14]. In this paper, we describe a special algorithm for constructing the elimination tree that results in a linear computational cost of the direct solver algorithm for the *entire* sequence of h-refined grids.

Most available multi-frontal solvers construct the elimination tree by analyzing the structure of the global matrix (e.g., MUMPS solver [15–17]). In this paper, we present an alternative approach, where the elimination tree is constructed by directly analyzing the computational mesh. Thus, the input data used during the analysis phase are just the computational mesh. This approach was already proposed in [18], where the elimination tree was constructed by browsing the refinement trees from the smallest to the largest elements following the natural ordering provided by the FE method. Unfortunately, such an ordering results in a non-linear computational cost for h-adaptive grids. In our approach, we browse the refinement trees from the largest to the smallest element and we merge frontal matrices from refinement trees adjacent to a common point singularity. By doing so, we achieve linear computational cost for each h-refined grid towards one or several point singularities. We have already proposed this ordering in terms of the graph grammar model of the computational mesh [19,20]; however, in those papers we did not realize that this ordering implies linear computational cost. Moreover, the elimination tree can be constructed in such a way that previously computed LU (or Cholesky) factorizations can be reutilized over all unrefined parts of the mesh. This feature provides us with a direct solver algorithm that delivers linear computational cost with respect to the number of unknowns, not only for a single h-refined mesh, but also for the entire sequence of meshes constructed by executing several h-refinements towards one or several point singularities. Thus, the reutilization technique reduces the total computational cost from $O(N^2)$ to $O(N)$.

The approach followed in this work can also be interpreted as a combination of (a) an iterative substructuring technique with reutilization of LU factorizations, and (b) an optimal selection of each substructure for each grid within a given sequence of h-refined meshes driven by point singularities. We also provide the corresponding optimal cost estimates.

Theoretical estimates are illustrated with several numerical results, including 2D problems discretized with the so-called radical meshes [21,22] towards one or two point singularities, and a 3D Fichera model problem [2]. All the problems considered in this paper are symmetric and we do not apply any pivoting algorithm.

2. Elimination tree of the direct solver

For illustration purposes, we consider the grid described in Fig. 1. It is composed of two elements that have been h-refined towards a point singularity located in the middle of the bottom edge. Isotropically h-refining the two elements closest to the singular point and applying the process recursively, we generate the sequence of grids. We assume a global uniform order of approximation p over the entire mesh. In order to build an elimination tree of the above model problem based on the connectivity information that exists within every FE code, the idea of building an elimination tree that follows the natural ordering of elements provided by the FE code may seem to be attractive. In such a case, the solver generates two frontal matrices. The first one is associated with the left refinement tree, while the second one is associated with the right refinement tree. The solver follows the order of elimination from leaves of the refinement tree up to the root. Namely, the solver aggregates elements 19, 20, 21, and 22 to the first frontal matrix, and elements 23, 24, 25, and 26 to the second frontal matrix. Subsequently, the solver aggregates elements 11, 12, and 13 to the first frontal matrix and elements 15, 16, and 18 to the second matrix. Unfortunately, such an approach produces the following undesired situation. In the last step of the solver, where the first frontal matrix has aggregated elements 3, 4, and 5 and the second frontal matrix has aggregated elements 7, 8, and 10, all unknowns associated with the vertical mid-edges of the grid (lying at line $z = 0$ in Fig. 2) remain invariant, since none of them have been eliminated. As a result, the CPU time cost associated with the LU factorization performed on this last step of the solver scales as $O(N^{1.5})$, which is prohibitively expensive.

Thus, we propose an alternative approach for building the elimination tree, based on ordering elements level by level, as described in Fig. 3. The solver algorithm browses the refinement trees from the top level down to the leaves, and it uses only one frontal matrix. Within each level, we eliminate only those nodes that are fully assembled.

This proposed elimination tree produces a grid structure that is topologically equivalent to a mesh that grows *only* in one dimension as we perform additional h refinements towards the point singularity. As a result, the elimination tree ensures that the size of the frontal matrix involved in the solver algorithm remains constant. To illustrate this, we observe that during the first elimination step (top level), the largest-size elements (namely, elements 1, 2, 3, 4, 5, and 6) are factorized with respect to all interface unknowns (grey stars in Fig. 3). After these elements have been eliminated, we obtain a grid with the same topological structure. In the second step of the algorithm, we eliminate elements 7, 8, 9, 10, 11, and 12 with respect to the interface unknowns (dots in Fig. 3). This second step has the same computational cost as the first one, since the number of eliminated and interface unknowns is the same in both cases. Indeed, it is easy to observe that all steps require

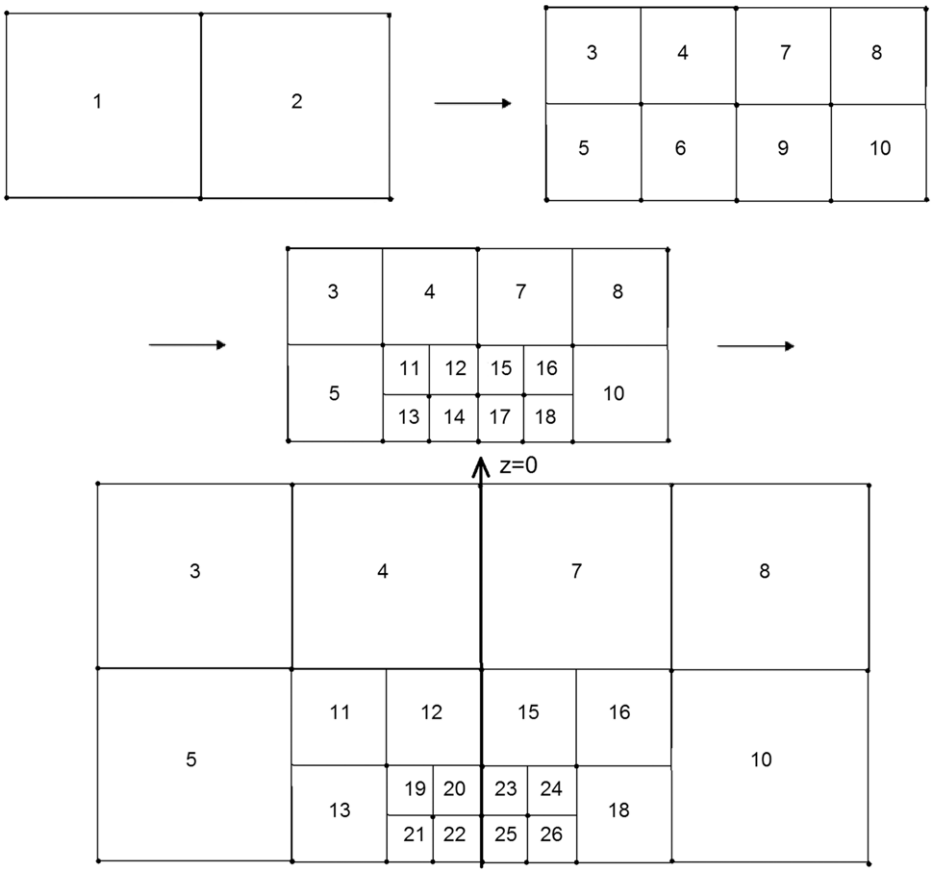

Fig. 1. Example of a radical mesh generated by a sequence of *h* refinements.

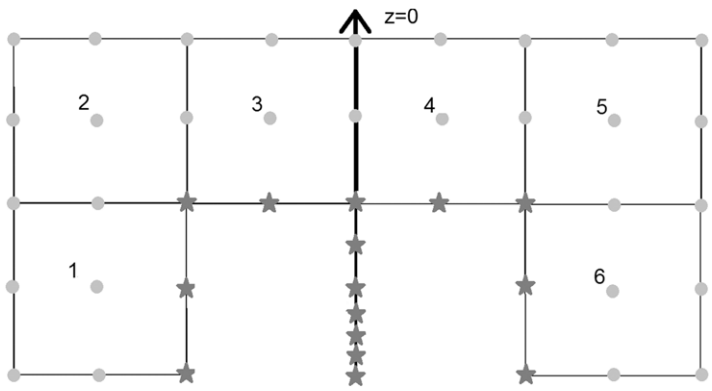

Fig. 2. Last step of the elimination pattern following the natural order of elements.

the same computational cost with the exception of the final one where we eliminate the remaining elements and has an even smaller cost than each of the previous steps. Since the cost at each level is constant, the total cost of the algorithm is proportional to the number of levels, which by grid construction is proportional to the number of unknowns. As a result, we obtain a solver algorithm with linear computational cost with respect to the number of unknowns. For a detailed analysis of the computational cost of an analogous version of the algorithm that browses elements level by level in a reverse bottom-up fashion, we refer to [14].

The main advantage of constructing a top-to-bottom elimination tree (as opposed to the bottom-to-top elimination tree proposed in [14]) is related to the reutilization of previously computed LU factorizations. If one solves the problem in a given grid and then performs an additional *h*-refinement towards the singular point, the top-to-bottom approach enables a full reutilization of previously computed LU factorizations, since new refined elements appear at the top of the elimination

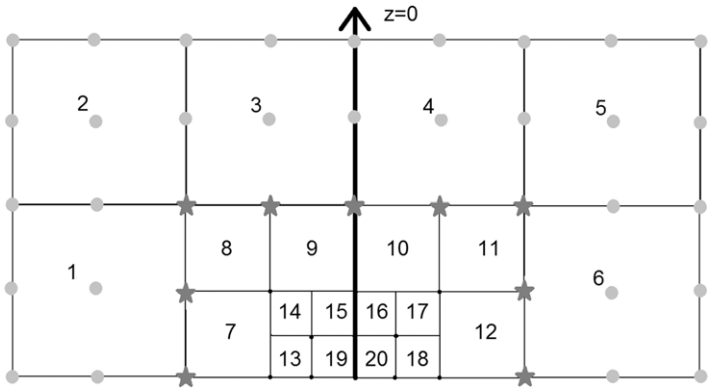

Fig. 3. Elimination pattern on a single level.

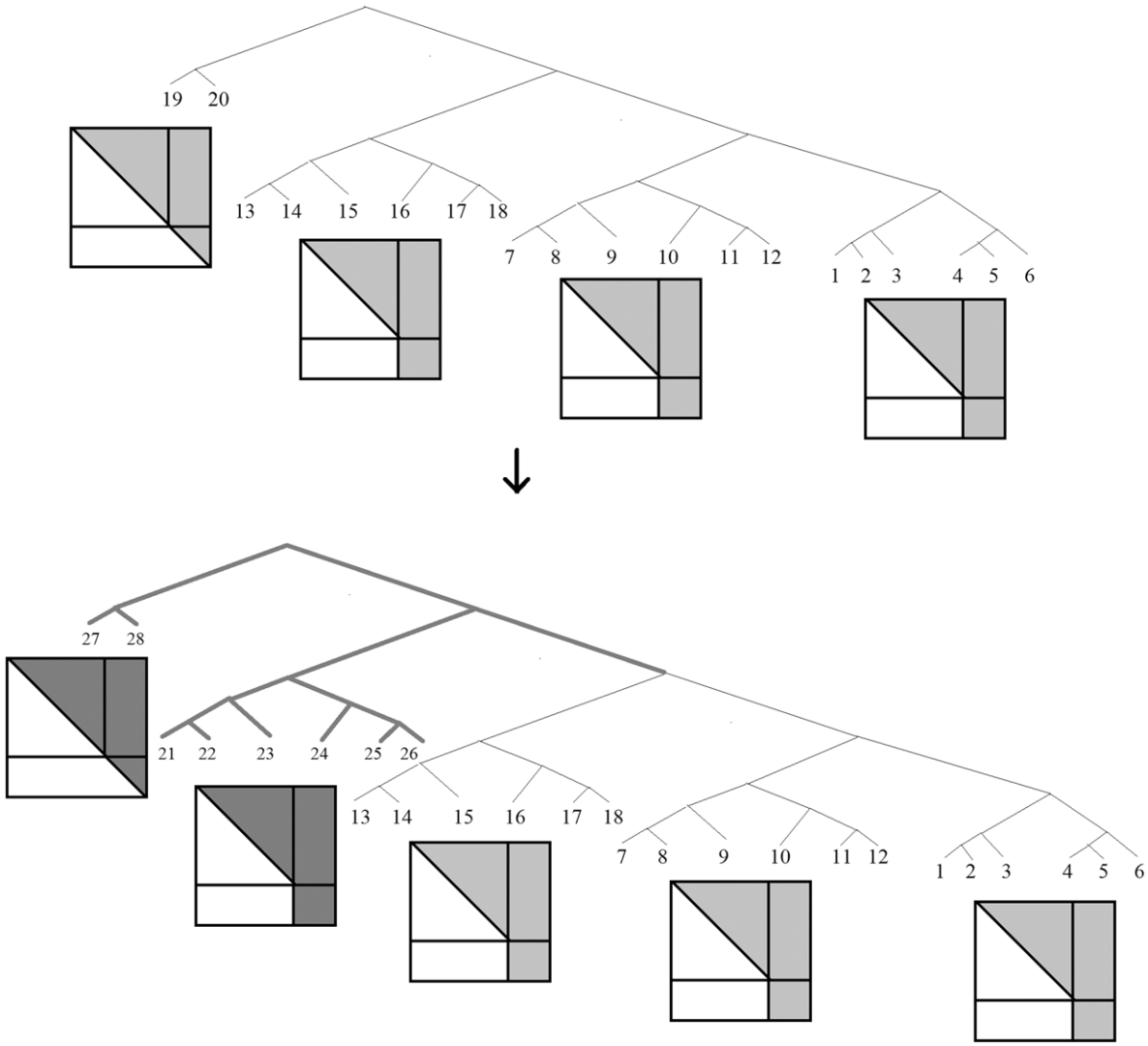

Fig. 4. Elimination tree based on top-to-bottom ordering, which enables efficient reutilization of previously computed LU factorizations.

tree. For a schematic representation of the process, see Fig. 4. Thus, one only needs to re-compute the LU factorization corresponding to the newly generated elements. This is not possible in the bottom-to-top approach proposed in [14],

since newly generated elements appear at the leaves of the elimination tree, which requires re-computation of nearly all previously computed partial LU factorizations.

3. Theoretical estimates on the computational cost

This section compares the number of floating point operations (NFLOPS) required to solve the entire sequence of h-refined grids performed with and without reutilization of previously computed LU factorizations. Again, we consider the radical grid described in Fig. 3.

We denote by L the total number of grids in the sequence, which also corresponds to the number of refinements applied to the last grid in the sequence. For each grid number l, we denote by T_l and M_l the execution time and memory required to solve it, and by N_l the number of unknowns. Let $N = N_L$ be the total number of unknowns at the last grid.

As mentioned before (and also proved in Section 4 of [14]), the execution time of the solution of a single grid is linear, which can be expressed as $T_l = c_1 + N_l c_2$ for the l-th grid, where c_1 and c_2 are constants. We also know that the number of unknowns grows linearly within the sequence; in other words $N_l = c_3 + c_4 l$, where c_3 and c_4 are constants.

3.1. Computational complexity estimates for the factorization of a sequence of h-refined grids

3.1.1. Without reutilization of prior factorizations

The total number of floating point operations (NFLOPS) required for performing the LU factorization of the entire sequence of h-refined grids without reutilizing previously computed LU factorizations is given by

$$\begin{aligned}\sum_{l=1}^{L} T_l &= \sum_{l=1}^{L} (c_1 + N_l c_2) = L c_1 + \sum_{l=1}^{L} N_l c_2 = L c_1 + c_2 \sum_{l=1}^{L} (c_3 + c_4 l) = L c_1 + L c_2 c_3 + c_2 c_4 \sum_{l=1}^{L} l \\ &= L c_1 + L c_2 c_3 + c_2 c_4 \left(\frac{L (L+1)}{2} \right) = O\left(L \ (c_1 + c_2 c_3 + c_2 c_4) + L^2 c_2 c_4\right) = O\left(L + L^2\right) \\ &= O\left(N^2\right).\end{aligned}$$

We can also estimate the memory usage, which is

$$\max_l M_l = \max_l \ (c_5 + N_l c_6) = O(N).$$

3.1.2. With reutilization of prior factorizations

We estimate now the total NFLOPS required for computing the LU factorization over the entire sequence of h-refined grids with the reutilization of previously computed LU factorizations. When the reutilization is turned on, the cost of solving a single grid from the sequence becomes constant, and we obtain

$$\sum_{l=1}^{L} T_l = \sum_{l=1}^{L} c_1 = L c_1 = O(N).$$

We can also estimate the memory usage. When the reutilization is active, the solution consists of generating just one frontal matrix associated with the root of the elimination tree, which implies that the memory usage of a single grid from the sequence is constant ($M_l = c_7$), and we obtain

$$\sum_{l=1}^{L} M_l = \sum_{l=1}^{L} c_7 = L c_7 = O(N).$$

We conclude that the reutilization reduces the execution time from $O(N^2)$ down to $O(N)$ and preserves the memory usage of order $O(N)$.

4. Numerical results

The reutilization solver has been implemented and tested on a number of model problems.

We report here the memory usage expressed as the number of non-zero entries appearing in the LU factorization as well as the execution time utilized by the following solvers:

- the state-of-the-art *MUMPS* direct solver [15–17] with *PORD* [23] ordering,
- our in-house solver called *Hypersolver* [24], which employs the elimination trees proposed in this work but do not reutilize partial LU factorizations, and
- our in-house solver called *Reutilization* that employs both the elimination trees proposed in this work as well as reutilization of previously computed partial LU factorizations.

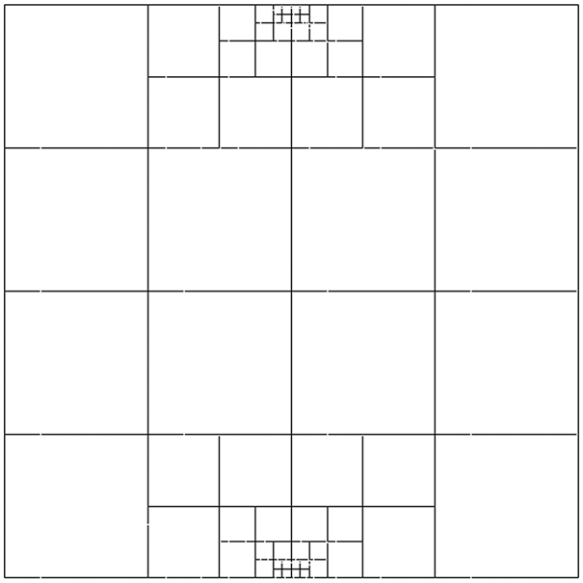

Fig. 5. Radical mesh with two point singularities.

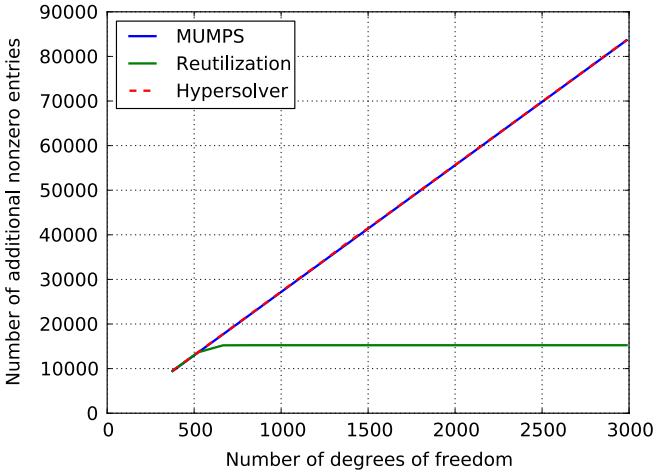

Fig. 6. Comparison of the number of non-zero entries for the radical mesh with one singularity.

We use MUMPS solver with *PORD* ordering rather than with *METIS* [25] ordering, since PORD ordering provides better performance of MUMPS on the examples considered in this paper. This improved performance can be observed in Figs. 8, 12 and 16.

These solvers have been tested using the following examples:

- the two dimensional radical mesh with one artificially enforced singularity described in Fig. 3,
- the two dimensional radical mesh with two artificially enforced singularities described in Fig. 5,
- the two dimensional L-shape domain problem described [1,21,22], and
- the three dimensional Fichera problem introduced in [2].

In all grids, we employ a uniform polynomial order of approximation $p = 5$.

We start by comparing the number of new non-zero entries in the LU factorization for the sequence of h-refined grids towards point singularities. The comparisons for all model problems are displayed in Figs. 6–9. Both MUMPS and our in-house implementation, Hypersolver, have to re-compute the entire LU factorization for each new mesh. Thus, the number of non-zero entries grows as the sequence progresses. However, the Reutilization solver only re-computes non-zero entries of the LU factorization in those newly refined elements surrounding the point singularity.

Figs. 10–13 compare the execution times of different solvers when applied to our model problems. The Hypersolver is significantly slower in the pre-asymptotic regime than the highly optimized MUMPS solver. Our solver utilizes block

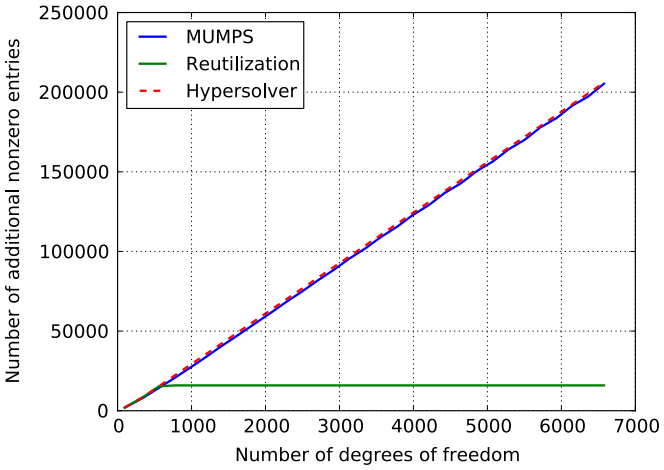

Fig. 7. Comparison of the number of non-zero entries for the L-shape domain problem.

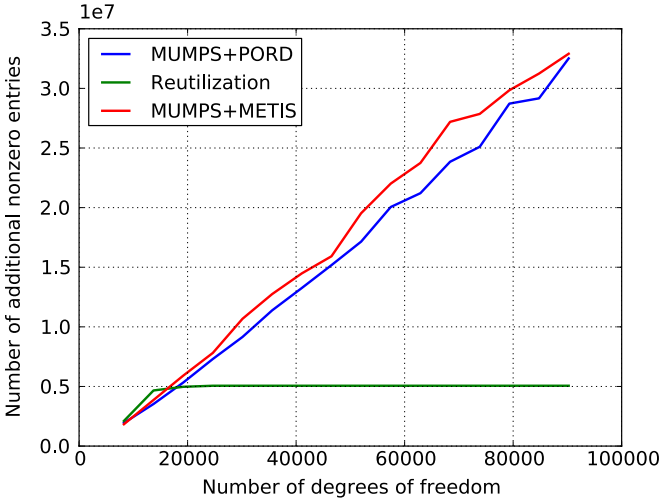

Fig. 8. Comparison of the number of non-zero entries for the Fichera problem.

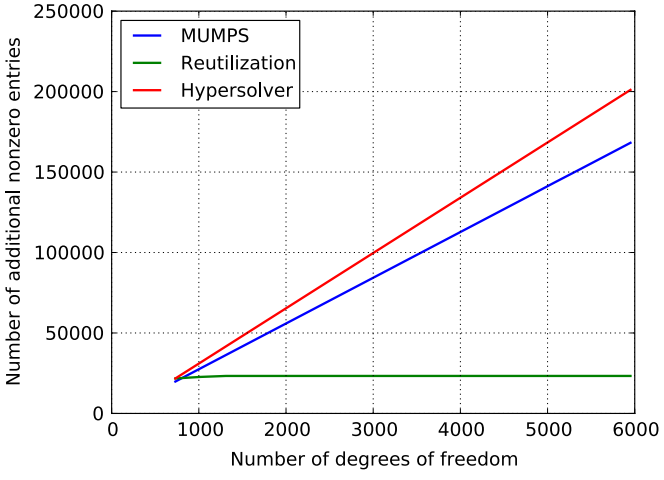

Fig. 9. Comparison of the number of non-zero entries for the radical mesh with two point singularities.

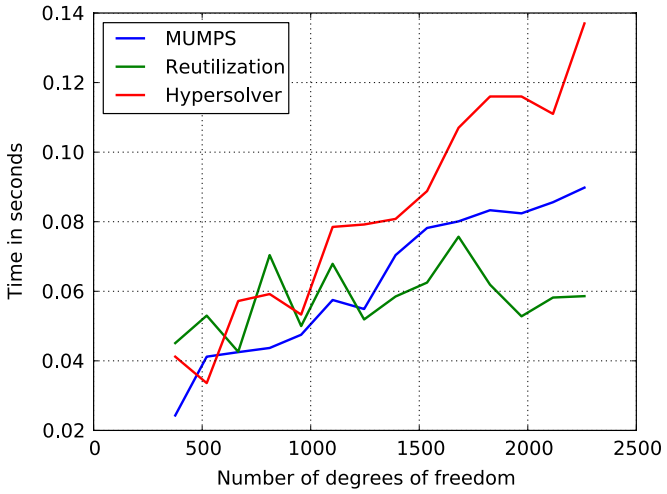

Fig. 10. Comparison of the execution time for the radical mesh with one point singularity.

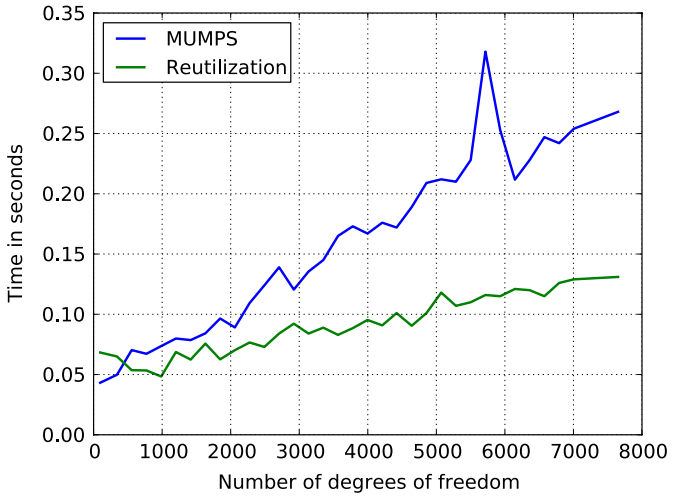

Fig. 11. Comparison of the execution time for the L-shape domain problem.

structure of the matrix, with blocks related to nodes of the mesh. The size of a block is equal to one for each vertex, $p-1$ for each edge, $(p-1)^2$ for each face, and $(p-1)^3$ for each interior. The blocks are transferred from the hypermatrix module to the BLAS routines. We believe that our solver is slower than MUMPS because of these memory transfers, and because the BLAS operations are performed on the different size blocks related to the nodes of the mesh. However, the purpose of this work is to show the better scalability of the proposed algorithm, which eventually translates to a better performance, as illustrated in the numerical results shown in Figs. 10–13. These results demonstrate that the reutilization solver outperforms MUMPS solver and it delivers constant execution time for each grid in the sequence of refinements. For the Fichera problem, we distinguish between the forward elimination and backward substitution parts of the solver, and we conclude that the backward substitution time is negligible here, as expected. The computational cost for backward substitution is linear for a single grid of the sequence, with a relatively small constant.

Finally, Figs. 14–17 compare the total execution time for the entire sequence of h-refined grids. From the comparison, it follows that only our reutilization solver delivers linear execution time.

5. Limitations of the method

The linear computational cost of the direct solver algorithm is obtained only in the presence of point singularities. Thus, we have utilized the 3D Fichera problem to test only a sequence of grids refined towards a central point singularity, as displayed in the left panel of Fig. 18. In the case of a sequence of refinements towards point and edge singularities, presented on the right panel of Fig. 18, the solver does not exhibit linear computational cost. This is because the number of degrees of

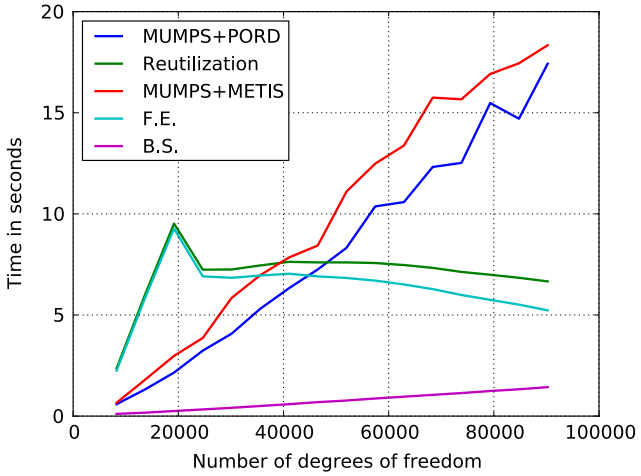

Fig. 12. Comparison of the execution time for the Fichera problem. Additionally, we describe the forward elimination part (F.E.) and the backward substitution part (B.S.) of the reutilization solver. Notice that the total time of the reutilization solver is the sum of the F.E. and B.S. times.

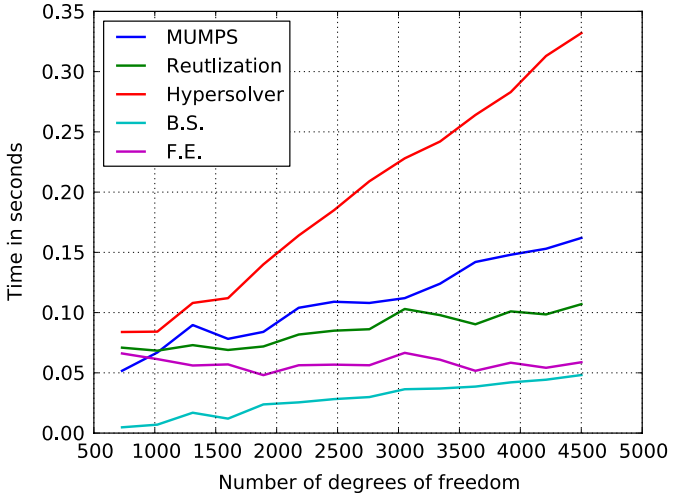

Fig. 13. Comparison of the execution time for the radical mesh with two singularities. Additionally, we describe the forward elimination part (F.E.) and the backward substitution part (B.S.) of the reutilization solver. Notice that the total time of the reutilization solver is the sum of the F.E. and B.S. times.

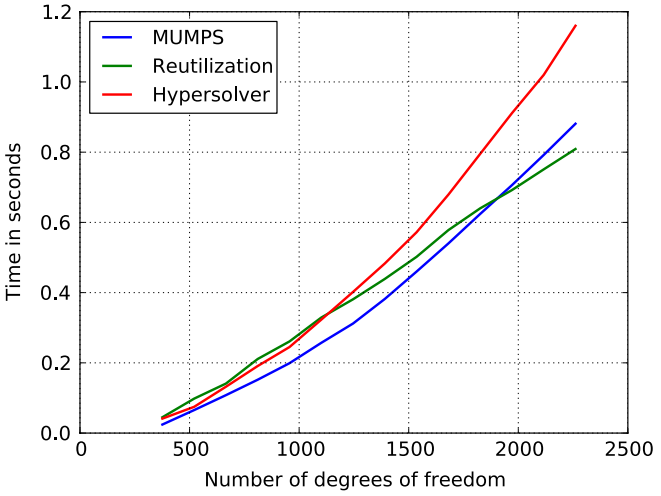

Fig. 14. Comparison of the total execution time for the entire sequence of grids for the radical mesh with one point singularity.

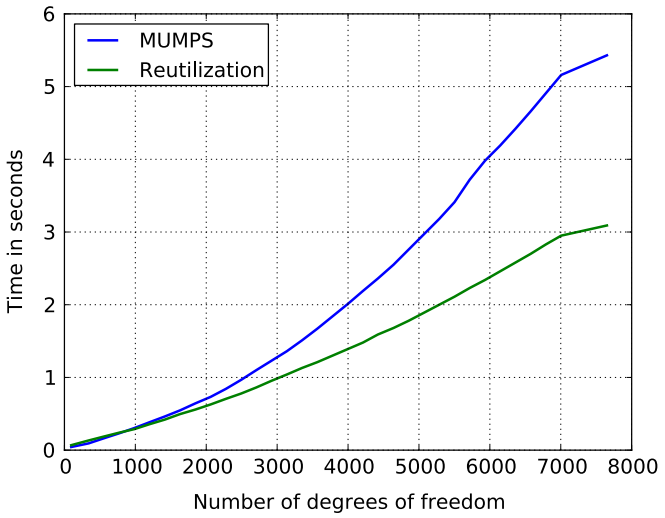

Fig. 15. Comparison of the total execution time for the entire sequence of grids for the L-shape domain problem.

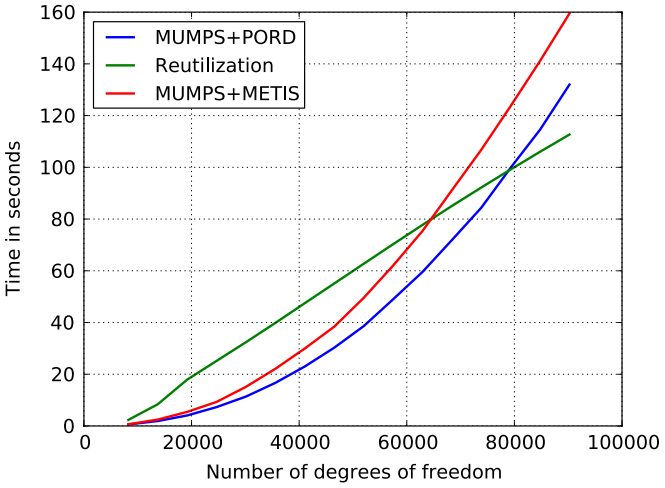

Fig. 16. Comparison of the total execution time for the entire sequence of grids for the Fichera problem with point singularity.

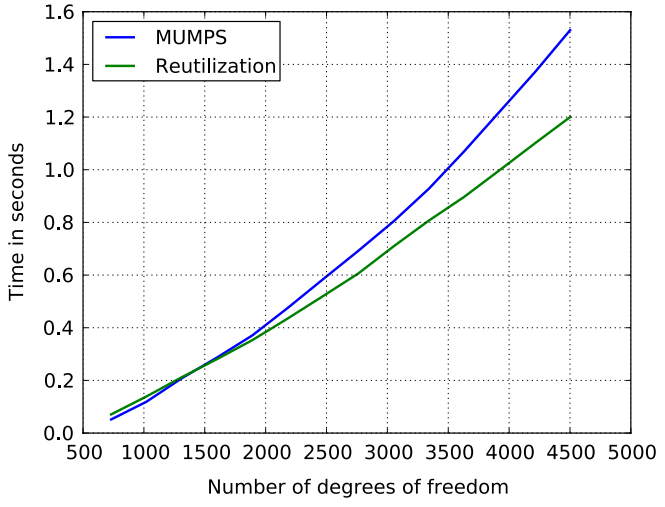

Fig. 17. Comparison of the total execution time for the entire sequence of grids for the radical mesh with two point singularities.

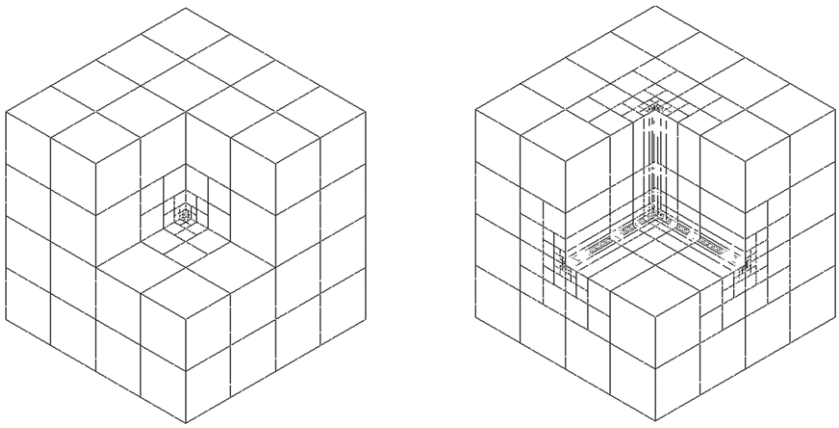

Fig. 18. Refinements towards point singularity versus refinements towards one point and three edge singularities.

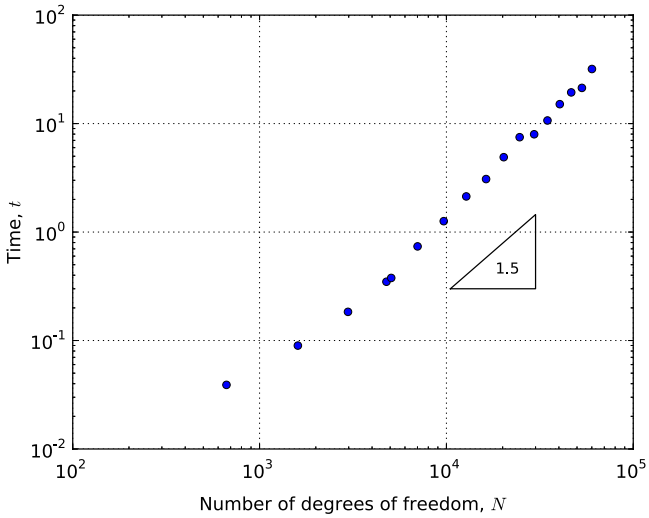

Fig. 19. Non-linear computational cost of the MUMPS solution corresponding to the Fichera problem with one point and three edge singularities.

freedom added during the edge refinement is no longer constant. The numerical experiments with MUMPS solver suggest that the execution time could be of order $O(N^{1.5})$; see Fig. 19.

Conclusions

In this paper we present a direct solver algorithm that constructs a novel elimination tree based on analyzing the structure of the computational mesh. The solver focuses on a class of computational meshes that are *h*-refined towards point singularities. The algorithm delivers linear computational cost of the solution for the entire sequence of meshes. This is achieved by using an elimination tree that enables reutilization of previously computed partial LU factorizations over the entire unrefined part of the mesh. The reutilization technique reduces the computational cost of the solution of the entire sequence of *h*-refined grids from $O(N^2)$ to $O(N)$. Numerical results confirm those theoretical estimates. Future work includes development of a more general algorithm for the construction of the elimination tree that enables reutilization of partial LU factorizations for an arbitrary mesh. We also plan to develop the reutilization algorithm for three dimensional grids with combined point and edge singularities, where we expect a reduction on the computational cost from $O(N^2)$ to $O(N^{1.5})$.

Acknowledgements

MP was supported by Polish National Science Center grants no NN 519 447739 and NN 519 405737. DP was partially funded by the Project of the Spanish Ministry of Sciences and Innovation MTM2010-16511, the Laboratory of Mathematics (UFI 11/52), and the Ibero-American Project CYTED 2011 (P711RT0278). MP and DP were partially supported by the Center for Numerical Porous Media at KAUST.

References

[1] L. Demkowicz, Computing with hp-adaptive finite elements, in: One and Two Dimensional Elliptic and Maxwell Problems, Vol. I, in: Applied Mathematics & Nonlinear Science, Chapman & Hall, CRC, 2006.
[2] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek, Computing with hp-adaptive finite elements, in: Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications, Vol. II, in: Applied Mathematics & Nonlinear Science, Chapman & Hall, CRC, 2007.
[3] T.J.R. Hughes, Linear Static and Dynamic Finite Element Analysis, Dover Publications, 2000.
[4] I.S. Duff, J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear systems, ACM Transactions on Mathematical Software 9 (1983) 302–325.
[5] I.S. Duff, J.K. Reid, The multifrontal solution of unsymmetric sets of linear systems, SIAM Journal on Scientific and Statistical Computing 5 (1984) 633–641.
[6] B. Irons, A frontal solution program for finite-element analysis, International Journal of Numerical Methods in Engineering 2 (1970) 5–32.
[7] T.A. Davis, W.W. Hager, Dynamic supernodes in sparse Cholesky update/downdate and triangular solves, ACM Transactions on Mathematical Software 35 (4) (2009) 1–23.
[8] A. Gupta, G. Karypis, V. Kumar, Highly scalable parallel algorithms for sparse matrix factorization, IEEE Transactions on Parallel and Distributed Systems 8 (5) (1997) 502–520.
[9] A. Gupta, Recent advances in direct methods for solving unsymmetric sparse systems of linear equations, ACM Transactions on Mathematical Software 28 (3) (2002) 301–324.
[10] A. Gupta, F.G. Gustavson, M. Joshi, S. Toledo, The design, implementation, and evaluation of a symmetric banded linear solver for distributed-memory parallel computers, ACM Transactions on Mathematical Software 24 (1) (1998) 74–101.
[11] J.D. Hogg, J.K. Reid, J.A. Scott, A DAG-based Sparse Cholesky Solver for Multicore Architectures RAL-TR-2009-004, 2009.
[12] J. Gaidamour, P. Hénon, A Parallel Direct/Iterative Solver Based on a Schur Complement Approach, in: 2008 11th IEEE International Conference on Computational Science and Engineering, 2008, pp. 98–105.
[13] P. Schmitz, L. Ying, A fast direct solver for elliptic problems on general meshes in 2D, Journal of Computational Physics 231 (2012) 1314–1338.
[14] A. Szymczak, A. Paszyńska, P. Gurgul, M. Paszyński, Graph grammar based direct solver for hp-adaptive finite element method with point singularities, Procedia Computer Science (2013) (in press).
[15] 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.
[16] P.R. Amestoy, I.S. Duff, J. Koster, J.-Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal of Matrix Analysis and Applications 23 (1) (2001) 15–41.
[17] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, S. Pralet, Hybrid scheduling for the parallel solution of linear systems, Parallel Computing 32 (2) (2006) 136–156.
[18] P. Bientinesi, V. Eijkhout, K. Kim, J. Kurtz, R. van de Geijn, Sparse direct factorizations through unassembled hyper-matrices, Computer Methods in Applied Mechanics and Engineering 199 (2010) 430–438.
[19] A. Szymczak, M. Paszynski, D. Pardo, Grach grammar based Petri nets controlled direct solver algorithm, Computer Science 11 (2010) 65–79.
[20] M. Paszynski, On the parallelization of self-adaptive hp finite element methods, part I, composite programmable graph grammar, Fundamenta Informaticae 93 (4) (2009) 411–434.
[21] I. Babuška, B. Guo, The hp-version of the finite element method, part I: the basic approximation results, Computational Mechanics 1 (1986) 21–41.
[22] I. Babuška, B. Guo, The hp-version of the finite element method, part II: general results and applications, Computational Mechanics 1 (1986) 203–220.
[23] J. Schulze, Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods, BIT 41 (2001) 800–841.
[24] M. Paszynski, D. Pardo, A. Paszynska, Parallel multi-frontal solver for p adaptive finite element modeling of multi-physics computational problems, Journal of Computational Science 1 (1) (2010) 48–54.
[25] G. Karypis, V. Kumar, MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, University of Minesota, 2011. http://www.cs.umn.edu/metis.