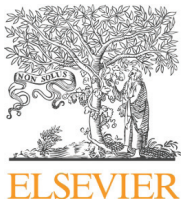

Procedia Computer Science

Volume 51, 2015, Pages 994–1002

ICCS 2015 International Conference On Computational Science

Ordering of elements for the volume & neighbors algorithm constructing elimination trees for 2D and 3D *h*-adaptive FEM

Anna Paszyńska[a]

[a]*Jagiellonian University, Krakow, Poland*

Abstract

In this paper we analyze the optimality of the volume and neighbors algorithm constructing elimination trees for three dimensional *h*-adaptive finite element method codes. The algorithm is a greedy algorithm that constructs the elimination trees based on the bottom up analysis of the computational mesh. We compare the results of the volume and neighbors greedy algorithm with the global dynamic programming optimization performed on a class of elimination trees. The comparison is based on the Directed Acyclic Graph (DAG) constructed for model grids. We construct DAGs for two model grids: a two dimensional grid refined towards point singularitiy and a two dimensional grid refined towards edge singularity. We show that the quasi-optimal trees created by the volume and neighbors algorithm for the model grids are also captured by the dynamic programming procedure. It means that created elimination trees are optimal in the considered class of elimination trees. We show that different element orderings at the input of the volume and neighbors algorithm result in different computational costs of the multi-frontal solver algorithm executed over the resulting elimination trees. Finally we present the ordering of elements that results in optimal (in the considered class) elimination trees. The theoretical results are verified with numerical experiments performed on a three dimensional grids with point, edge and face singularities.

Keywords: multi-frontal direct solver; h-adaptive finite element method; greedy algorithms, dynamic programming, DAGs

Corresponding author. Tel.: +4812-664-45-92. *E-mail address*: anna.paszynska@uj.edu.pl.

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.242

1 Introduction

The multi-frontal solver [2] is the state of the art algorithm for factorization of the linear systems resulting from mesh based computations with the finite element method. It is also a core of the multi-grid method [1]. It is computationally much more expensive than the integration [3, 20, 22]. The execution of the multi-frontal solver algorithm is controlled by the elimination tree [4]. The computational cost of the multi-frontal algorithm depends on the quality of the elimination tree constructed based on the computational mesh. Recently, in the paper [5] a new "volume&neighbors" algorithm is proposed. It outperforms alternative algorithms over the computational grids *h* refined towards singularities. In particular, the elimination trees generated by other state-of-the-art algorithms, such as nested-dissections [6], PORD algorithm [7], minimum degree algorithms [8] with several variations [9, 10] results in worse computational cost of the multi-frontal solver algorithm.

In this paper we analyze the optimality of the volume & neighbors algorithm. We do that by considering a dynamic programming algorithm [11] searching for quasi-optimal elimination trees in a large class of elimination trees constructed for a given computational mesh. The class of elimination trees includes the trees constructed by partitions of the computational meshes along horizontal or vertical lines. The computational cost of the multi-frontal solver algorithm over these grids is estimated by using the dynamic programming approach executed over the Directed Acyclic Graph (DAG). The binary sub-tree of the search tree with minimal computational cost is the optimal tree constructed by the dynamic programming algorithm. In this paper the heuristic volume & neighbors algorithm is considered over model grids, generated by two and three dimensional h-adaptive finite element method [12, 13, 19, 23], namely, on grids refined towards point, edge or face singularities, and the relations between orderings of elements, heuristic and dynamic programming are analyzed.

2 Model problems

We focus on two and three dimensional model grids refined towards point or edge singularities, presented in Figures 1 and 2. The algorithm utilized for generation of the ordering of elements will be summarized in Section 5.

3 Volume & neighbors algorithm

The volume & neighbors algorithm can be summarized in the following way:

```
1 Create list of Ne one-element trees, with attributes neighbors
   and volume, sort them according to volume
2 for i=1 to Ne-1
3   Find a pair (v,w) of "best neighbors"
4   Create new root r
5   Copy trees v and w as a children of r
6   Compute attribute of the root : volume(r)=volume(v)+volume(w)
7   Merge lists of neighbors of v and w as a new list for r
8   Add a new tree to the list of trees
9    Sort the list according to volume
```

Algorithtm 1. Volume & neighbors.

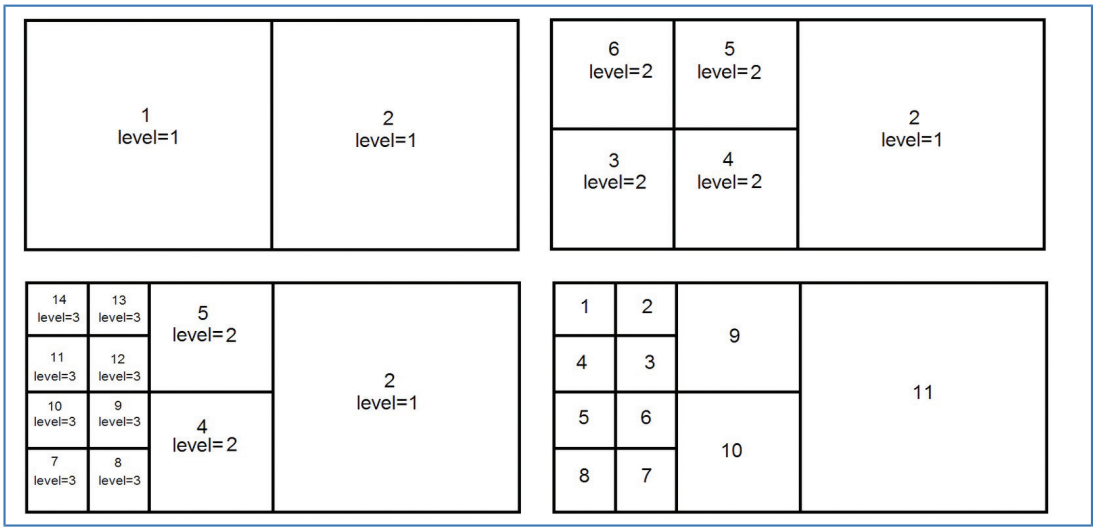

Figure 1. Generation of the 2D mesh with ordering of elements. Top panel: Two dimensional meshes with point singularities. Bottom panel: Two dimensional meshes with edge singularities.

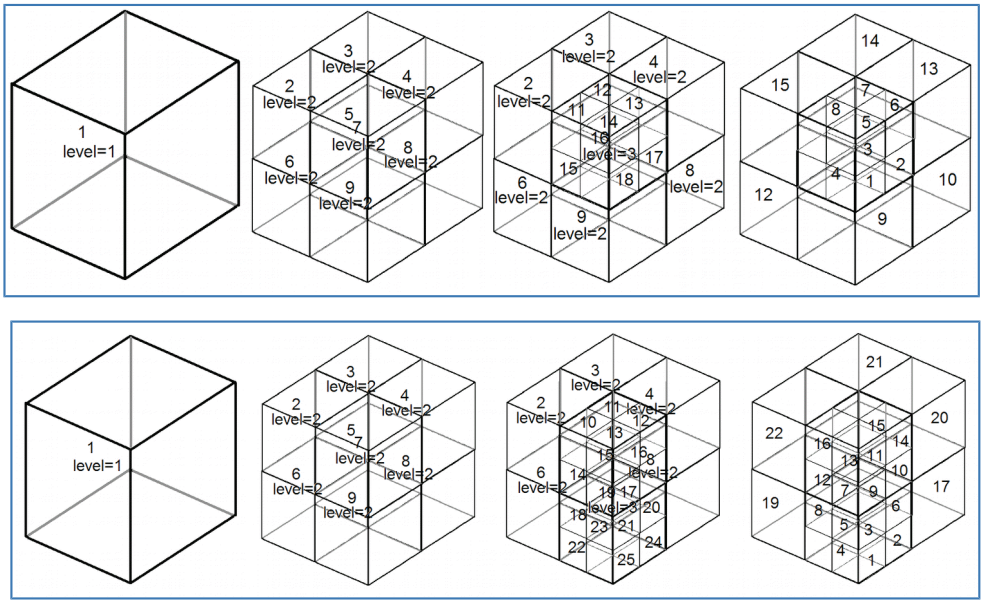

Figure 2. Generation of the 3D mesh with ordering of elements. **Top panel:** Three dimensional meshes refined towards point singularity. **Bottom panel: Three** dimensional meshes refined towards edge singularity.

In the above algorithm, N_e stands for the number of elements. By "best neighbors" we mean a pair, having minimum volume and maximum number of common neighbors. The intuition behind the algorithm is the following. We would like to minimize the size of matrices processed by the multi-frontal solver. The size of matrices corresponds to the number of mesh nodes, which again corresponds to the volume of mesh blocks

associated with a frontal matrix. Additionally, we would like to maximize the number of eliminated rows, which again correspond to the number of nodes over the common edges (in 2D) or faces (in 3D) when we merge the blocks. For additional details, we refer to [5].

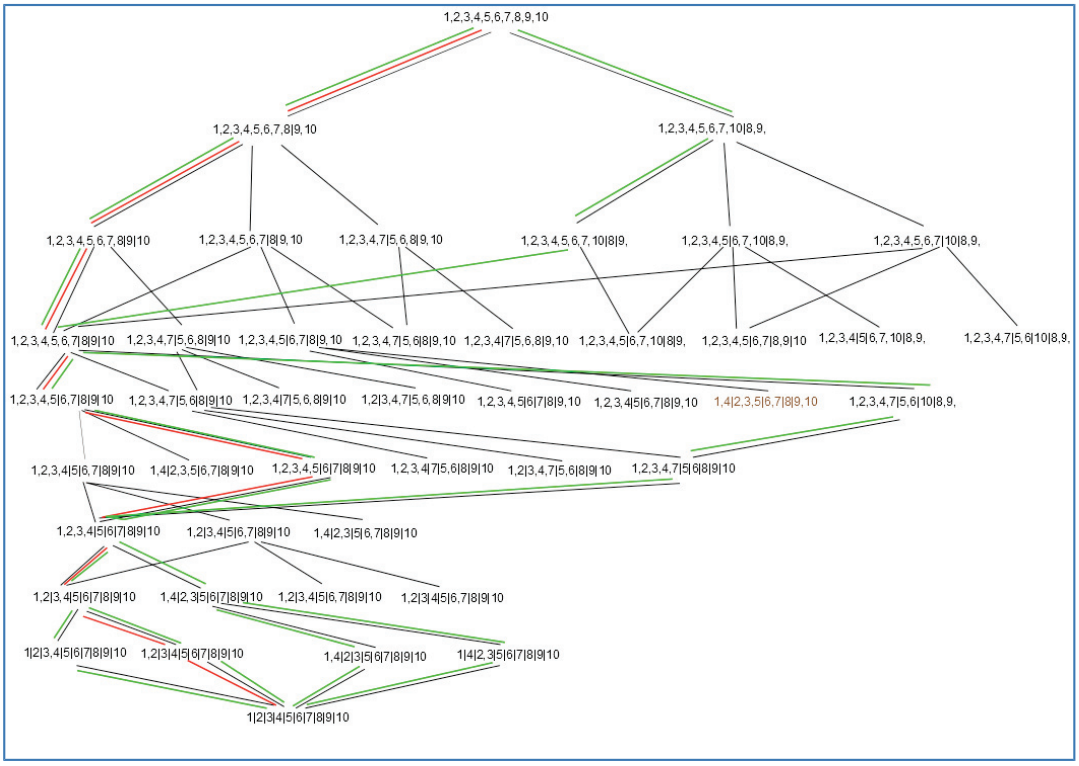

Figure 3. Part of the DAG for elimination trees for the mesh refined towards point singularity

4 Analysis of the algorithm

In this section, we analyse the volume & neighbors algorithm and compare its results to the one obtained from the dynamic programming approach [11]. The green colour in Figure 3 denotes all the cases that are considered by the volume & neighbors algorithm, when we change the initial ordering of elements in the sorted list processed by the algorithm. Notice that the algorithm requires that elements are sorted according to the volume, but this can be done considering different ordering for the subsets of elements with identical volume. The black colour in Figure 3 denotes all the cases browsed by the dynamic programming algorithm, where the partitions of the mesh are allowed only along the straight lines. The red colour denotes the solution found out by the heuristic volume & neighbors algorithm, under the assumed ordering of elements, presented in Figure 1.

We start with the two dimensional mesh refined towards point singularity, presented in Figure 1. We do that by constructing a DAG for the algorithm. We start with the root of the tree, where we have all elements of the mesh, 1,2,3,4,5,6,7,8,9,10. Next, we consider the partitions of the mesh into parts, along the straight lines, the same like in the dynamic programming algorithm. At this point, we can partition the mesh into parts 1,2,3,4,5,6,7,8 | 9,10 (by cutting out the element 9 and 10) or 1,2,3,4,5,6,7,10 | 8,9 (by cutting out the element 8 and 9).

Let us focus now on the partition 1,2,3,4,5,6,7,8 | 9,10 . This can be further partitioned into 1,2,3,4,5,6,7,8 | 9 | 10 (by separating elements 9 and 10) or 1,2,3,4,5,6,7 | 8 | 9,10 (by cutting out element 8 in the upper part) or 1,2,3,4,7 | 5,6,8 | 9,10 (by cutting out elements 5,6 and 8).

We can continue with all possible partitions along the straight lines, and we get the tree of partitions that is browsed by the dynamic programming algorithm. Even for the mesh refined towards point singularity this tree is huge, and Figure 3 contains only a small portion of that tree. We end up with the mesh partitioned into single finite elements, namely to 1 | 2 | 3 | 4 | 5 |6 |7 |8 |9 |10. Each binary elimination tree is defined by the path from vertex, 1,2,3,4,5,6,7,8,9,10 to the vertex 1 | 2 | 3 | 4 | 5 |6 |7 |8 |9 |10.

In the DAG presented in Figure 3 we have denoted by black colour combinations (partitions) that are browsed by the dynamic programming algorithm. Notice, that we only present some part of the entire tree.

5 Assumptions of volume & neighbors algorithm

Based on the DAG analysis presented in previous section, we propose the ordering of elements that may improve the quality of the elimination trees constructed by the volume & neighbors algorithm.

Let the heuristic algorithm works under the following assumptions:

- The computational mesh is either two or three dimensional and it is obtained by performing a sequence of refinements from an initial structured regular mesh with rectangular finite elements (in 2D) or hexahedral finite elements (in 3D).
- When constructing the h refined mesh, only isotropic h refinements are allowed. In other words, selected rectangular elements are always broken into 4 smaller child elements (in 2D) or 8 child elements (in 3D).
- The elements in the initial mesh are topologically numbered, counter-clockwise, level by level (in 3D)
- Each element has an assigned refinement level, and the refinement level on the initial mesh is equal to 1.
- When the adaptive algorithm breaks an element into 8 child elements, the refinement level of all child elements is equal to the refinement level of the parent element plus one.
- The volume of each element is defined as $1/2^{(2*(\text{refinement level-1}))}$ (in 2D) or $1/2^{(3*(\text{refinement level-1}))}$ (in 3D).
- When the adaptive algorithm breaks an element into child elements, they get the new numbers in the global numbering of elements, and their numbers are again are topologically sorted, counter-clockwise, level by level (in 3D)
- The mesh fulfils the 1-irregularity rule, telling that an element can be broken only once without breaking adjacent large element.
- When there is one element on one side of an edge (in 2D) or a face (in 3D) and two (in 2D) or four (in 3D) elements on the other side of the edge (in 2D) or face (in 3D), we call this common edge (or face) a constrained edge (or face).
- When we compute the maximum number of common edges (in 2D) or faces (in 3D) between two adjacent patches of elements in the mesh, we count each constrained edge (in 2D) or face (in 3D) as one.
- The ordering of elements for the volume & neighbors algorithm is constructed based on the above non-continuous numbering, in the reverse order, and we make the final numbering continuous.

The exemplary generations of the meshes with orderings of elements are illustrated in Figures 1 for 2D meshes and Figure 2 for 3D meshes. The assumptions listed above corresponds to the computational grids generated by two and three dimensional hp-adaptive finite element method codes called *hp2d* for two dimensional grids [12] or *hp3d* for three dimensional grids described in [13].

6 Numerical results

In this section we compare the computational costs of the volume & neighbors algorithm for different element orderings of initial elements, following the proposed assumption, and selected randomly. We perform the comparison for the three dimensional grids refined towards point, edge and face singularities.

The results for the three dimensional mesh refined towards point, edge and face singularities obtained with elements orderings following the above assumptions, are presented in Figures 4,5 and 6. We report the number of floating point operations and compare with alternative ordering algorithms available through MUMPS interface, such as nested-dissections [6], available through METIS library [14], PORD algorithm [7], minimum degree algorithms [8] and its modifications [9, 10].

Additionally, we presented comparison of the number of FLOPs for the volume & neighbors algorithm executed with the predefined order of elements as well as with randomly re-sorted order. The results are presented in Tables 1,2 and 3 for point, edge and face singularities. The elimination trees for the random order of elements are no longer optimal. The only exception is the point singularity, where the volume & neighbors algorithm always generates optimal trees.

We compare number of FLOPs and not the execution times, since it may be affected by other factors, like using out-of-core or in-core version of the solver [15]. The number of FLOPs has been measured using numerical experiments.

N	*hp3d*	*random*
181	122275	122275
237	172455	172455
293	222635	222635
349	272815	272815

Table 1. Comparison on the number of floating point operations for the three dimensional mesh refined towards point singularity, with elements ordered randomly and elements sorted according to *hp3d*.

N	*hp3d*	*random*
249	224049	381981
485	716239	1163116
945	2076145	5100271
1853	5570307	25424296

Table 2. Comparison on the number of floating point operations for the three dimensional mesh refined towards edge singularity, with elements ordered randomly and elements sorted according to *hp3d*.

N	*hp3d*	*random*
399	624192	1239132
1393	6883545	30575529
5171	75234102	891583297
19893	770216539	468448007

Table 3. Comparison on the number of floating point operations for the three dimensional mesh refined towards face singularity, with elements ordered randomly and elements sorted according to *hp3d*.

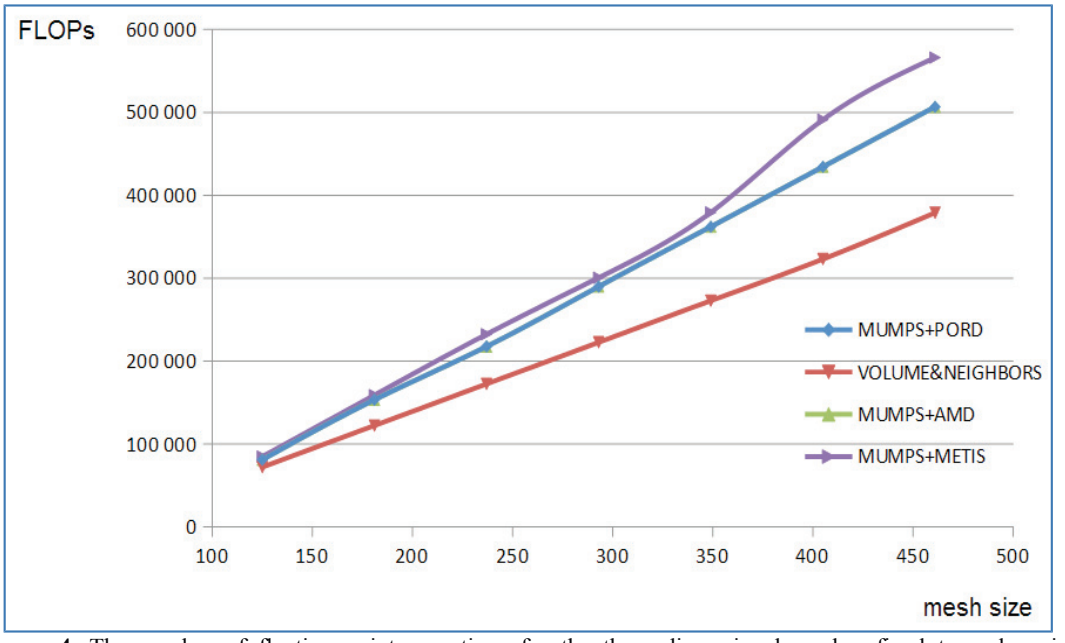

Figure 4. The number of floating point operations for the three dimensional mesh refined towards point singularity, with elements sorted according to *hp3d* ordering. Comparison with alternative algorithms.

Conclusions

In this paper we have shown the importance of the ordering of elements for the volume & neighbors algorithm. When the algorithm follows the ordering prescribed in *hp2d* or *hp3d* codes, it generates quasi-optimal elimination trees, that out-performs alternative ordering algorithms. However, when the ordering of elements is random, nothing can be certain about the performance of the algorithm. We have also compared the orderings generated by the volume & neighbors algorithm with the results of the global dynamic programming procedure. We have shown that the optimal trees found by the algorithm for grids refined towards point and edge singularities are also captured by the dynamic programming procedure. The future work may include development of the graph grammar model for two [16] and three dimensional [17,18] version of the algorithm. It may also include development of the algorithm updating the elimination tree when the mesh is refined from one time step to another [21]

References

[1] K. Banaś, Scalability analysis for a multigrid linear equations solver, Lecture Notes in Computer Science, 4967 (2008) 1265-1274

[2] I. S. Duff, J. K. Reid, The multifrontal solution of indefinite sparse symmetric linear systems. ACM Transactions on Mathematical Software, 9 (1983) 302-325.

[3] K. Banas, P. Plaszewski, P. Maciol, Numerical integration on GPUs for higher order finite elements, Computers and mathematics with applications 67(6) (2014) 1319-1344

[4] J.W.H. Liu, The multifrontal method for sparse matrix solution: theory and practice, SIAM Review 34 (1992), 82-109.

[5] A. Paszyńska, Volume and neighbors algorithm for finding elimination trees for three dimensional h-adaptive grids, Computers & Mathematics with Applications (2014) 68(10) 1467-1478.

[6] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal of Scientiffic Computing, 20, 1 (1998) 359-392.

[7] J. Schulze, Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods, BIT, 41, 4 (2001) 800.

[8] P. Heggernes, S.C. Eisenstat, G. Kumfert, A. Pothen, The Computational Complexity of the Minimum Degree Algorithm, ICASE Report No. 2001-42, (2001).

[9] P. R. Amestoy, T. A. Davis, I. S. Du, An Approximate Minimum Degree Ordering Algorithm, SIAM Journal of Matrix Analysis & Application, 17, 4 (1996) 886-905.

[10] G.W. Flake, R.E. Tarjan, K. Tsioutsiouliklis, Graph clustering and minimum cut trees, Internet Mathematics 1 (2003), 385-408.

[11] H. AbouEisha, M. Moshkov, V. Calo, M. Paszynski, D. Goik, K. Jopek, Dynamic Programming Algorithm for Generation of Optimal Elimination Trees for Multi-frontal Direct Solver Over H-refined Grids, Procedia Computer Science, (2014) 29, 947-959.

[12] L. Demkowicz, Computing with hp-Adaptive Finite Elements, Vol. I. One and Two Dimensional Elliptic and Maxwell Problems. Chapman & Hall / CRC Applied Mathematics & Nonlinear Science, (2006).

[13] L. Demkowicz, J. Kurtz, D. Pardo , M. Paszynski, W. Rachowicz , A. Zdunek, Computing with hp-Adaptive Finite Elements, Vol. II. Frontiers. Three Dimensional Elliptic and Maxwell Problems with Applications. Chapman & Hall / CRC Applied Mathematics & Nonlinear Science, (2006).

[14] G. Karypis, V. Kumar, METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0, Technical Report (1995)

[15] M. Paszynski, D. Pardo, A. Paszynska, L. Demkowicz, Out-of-core multi-frontal solver for multi-physics hp adaptive problems, Procedia Computer Science, 4 (2011) 1788-1797

[16] A. Paszynska, M. Paszynski, E. Grabska, Graph Transformations for Modeling hp-Adaptive Finite Element Method with Mixed Triangular and Rectangular Elements, Lectures Notes in Computer Science, 5545 (2009) 875-884.

[17] A. Paszynska, E. Grabska, M. Paszynski, A Graph Grammar Model of the hp Adaptive Three Dimensional Finite Element Method. Part I, Fundamenta Informaticae, 114(2) (2012) 149-182.

[18] A. Paszynska, E. Grabska, M. Paszynski, A Graph Grammar Model of the hp Adaptive Three Dimensional Finite Element Method. Part II, Fundamenta Informaticae, 114(2) (2012) 183-201.

[19] P. Gurgul, M. Sieniek, M. Paszynski, L. Madej, N. Collier, Two-dimensional hp-adaptive algorithm for continuous approximations of material data using space projection, Computer Science, 14(1), (2013) 97-112.

[20] F. Kruzel, K. Banaś, Vectorized OpenCL implementation of numerical integration for higher order finite elements, Computers and mathematics with applications 66(10) (2013) 2030-2044.

[21] K. Banaś, L. Demkowicz, Entropy controlled adaptive finite element simulations for compressible gas flow, Journal of computational physics 126(1) (1996) 181-201.

[22] P. Plaszewski, P. Maciol, K. Banas, Finite Element Numerical Integration on GPUs, Lecture Notes in Computer Science, 6067 (2010) 411-420.

[23] K. Banas, A modular design for parallel adaptive finite element computational kernels, Lecture Notes in Computer Science, 3037 (2004), 155-162

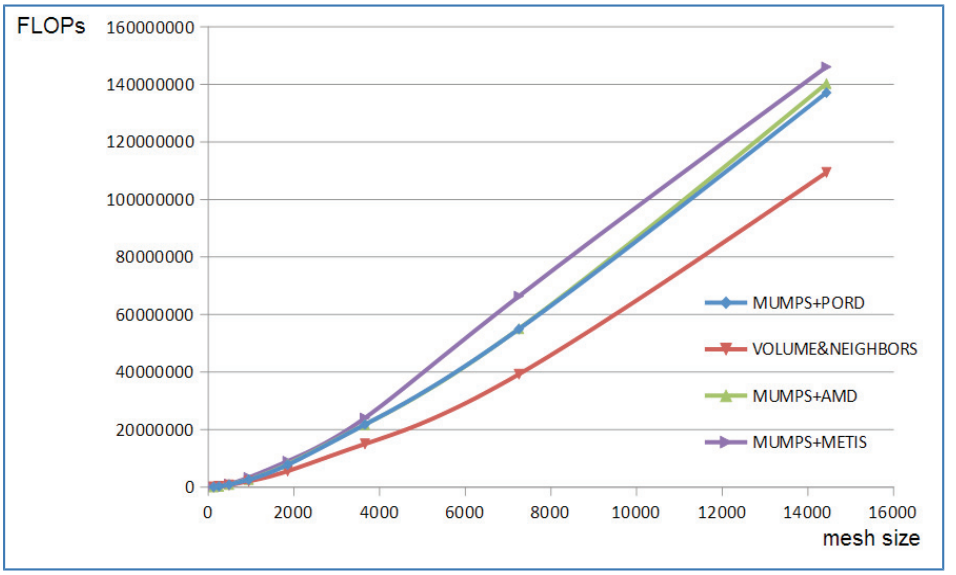

Figure 5. The number of floating point operations for the three dimensional mesh refined towards edge singularity, with elements sorted according to *hp3d* ordering. Comparison with alternative algorithms.

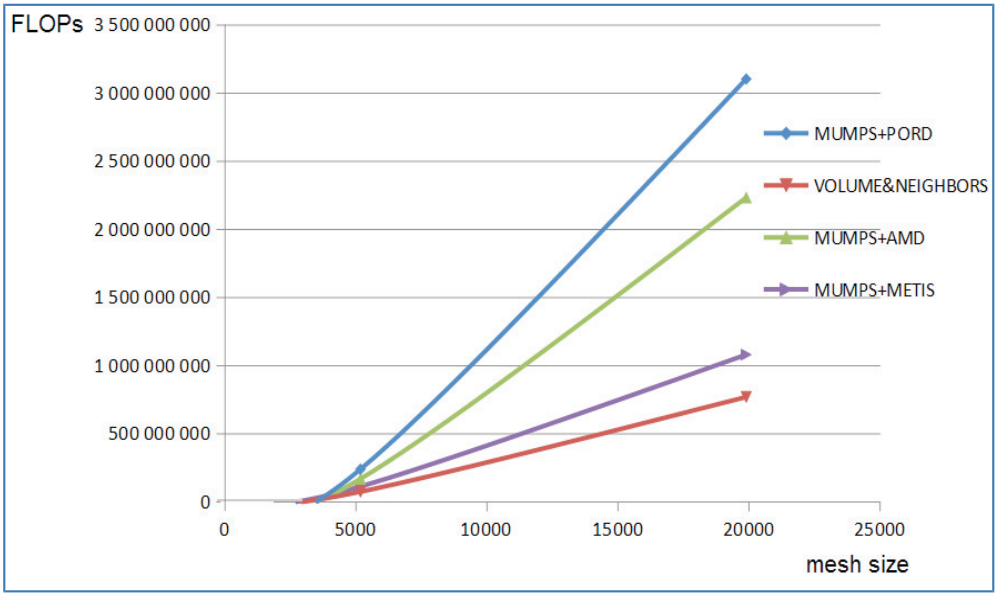

Figure 6. The number of floating point operations for the three dimensional mesh refined towards face singularity, with elements sorted according to *hp3d* ordering. Comparison with alternative algorithms.