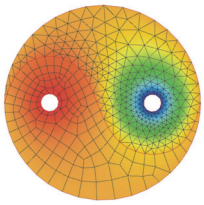
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE
Informatyka w Technologii Materiałów

Vol. 16, 2016, No. 1



# BISECTION WEIGHTED BY ELEMENT SIZE ORDERING ALGORITHM FOR MULTI-FRONTAL SOLVER EXECUTED OVER 3D *h* REFINED GRIDS

MARCIN SKOTNICZNY[1], MACIEJ PASZYŃSKI[1]*, ANNA PASZYŃSKA[2]

[1] *Department of Computer Science AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Cracow, Poland*
[2] *Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, ul. St. Łojasiewicza 11, 30-348 Krakow, Poland*
**Corresponding author: paszynsk@agh.edu.pl*

**Abstract**

In this paper we present an algorithm for generation of ordering over 3D grids h refined towards singularities. The ordering controls the execution of multi-frontal direct solver algorithm on systems of linear equations generated by 3D h adaptive finite element method. The proposed ordering algorithm outperforms other state-of-the-art orderings available through MUMPS interface, namely nested-dissections, AMD and PORD. Our algorithm uses additional knowledge about the structure of the computational mesh, not available to alternative ordering algorithms.

**Key words**: Finite Element Method, hp adaptivity, multi-frontal direct solver; element partition tree; ordering algorithm

## 1. INTRODUCTION

Multi-frontal solver (Amestoy et al., 2001; Amestoy et al., 2006) is the state-of-the-art direct solver algorithm for solving system of linear equations resulting from finite element method simulations. In this paper we focus on systems of linear equations resulting from execution of three dimensional *h* adaptive finite element method. Recently it has been shown theoretically that three dimensional grids *h* refined towards point, edge or face singularities can be solved in linear or quasi-linear computational cost (Paszyński et al., 2015). However, the heuristic algorithms delivering such linear or quasi-linear computational costs are not defined yet The sequential execution of the multi-frontal solver algorithm is controlled by ordering (George & Liu, 1978), defining the order of elimination of rows of the sparse matrix. The state-of-the-art implementation of the multi-frontal direct solver is the MUMPS solver (MUMPS). The MUMPS is interfaced with several algorithms generating different orderings. These are nested-dissections algorithm (George & Liu, 1978) implemented in METIS library (Karypis & Kumar, 1998), PORD algorithm (Schulze 2001) also available through MUMPS interface, and different variations of the Minimum Degree (MD) algorithm (Heggernes et al., 2001), such as Approximate Minimum Degree (AMD) (Amestoy et al., 1996) or Approximate Minimum Fill (AMF) algorithm. In this paper we propose a new heuristic algorithm, *bisection weighted by element size*, that allows to outperform other ordering algorithms by means of the number of floating point operations (FLOPs) for systems of linear equations generated from computational grids *h* refined towards local singularities. Our bisections weighted by element size is the *h* refined grids version of the greedy algo-


ISSN 1641-8581

rithm already proposed for *hp* refined grids (AbouEisha et al., 2016). We test our algorithm on 3D grids refined towards point, edge or face singularities, and compare the FLOPs number with alternative ordering algorithms available through MUMPS interface. Please notice that our ordering algorithm utilizes additional knowledge about the structure of the computational mesh and the history of mesh refinements. These data are ignored by alternative ordering algorithms and it is hard to reconstruct them based only on the sparsity pattern of the matrix. The hexahedral h adapted grid are wildly used with the constrained approximaion and hanging nodes technique in h adaptive finite element method, see page. 32 of (Demkowicz et al., 1999). The extension of the code to p adaptivity also allows for the utilization of the constrained approximation and hanging nodes. For more intuite description in 2D case we refer to Section 4 in (Demkowicz et al., 1998). Additionally, modern isogeometric finite element method (IGA-FEM) works only on regular 2d or 3d patches of elements obtained from geometry mapping from CAD systems. These patches represent master elements from which the geometry map is projecting into particular geometric parts of CAD objects. These regular patches of elements can be also locally refined using T-spline basis functions (Bazilevs et al., 2010). Thus, due to the exponentially growing number of papers and citations of IGA-FEM (Web of Science) we can say that regualr hexahedral grids are currently the state of the art trend in scientific computing. We may even risk a statement that non regular tetrahedral grids and mesh generators will be much less utilized in the future, since the integration of CAD and FEM does not require the unstructured mesh generation, and we just inherit the regular patches of elements from CAD system (Cottrell et al., 2009).

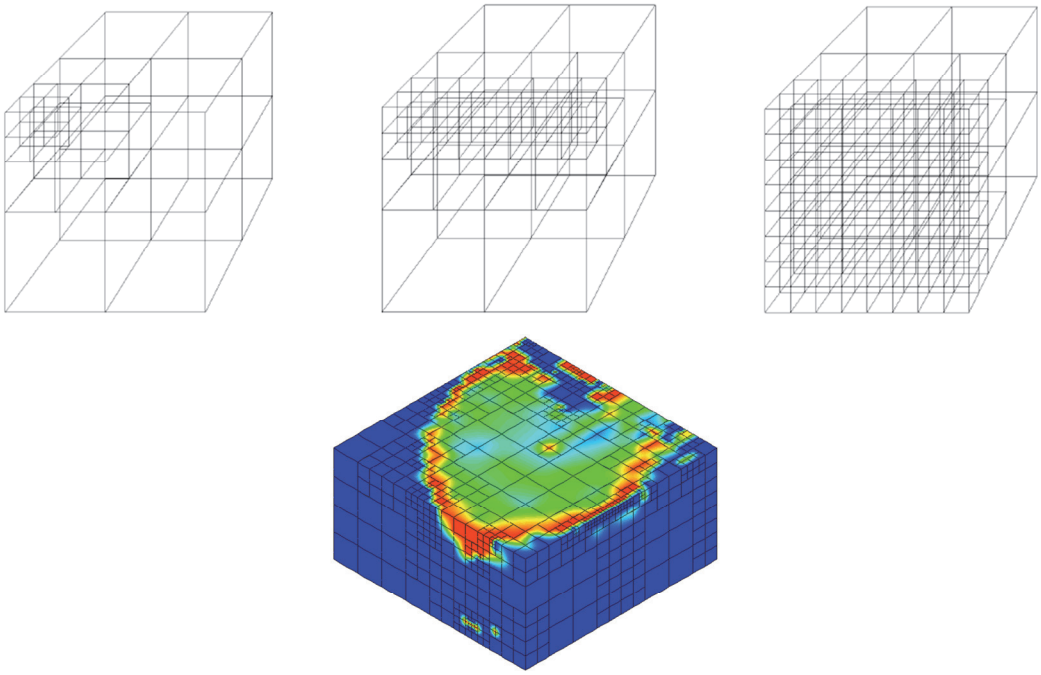


***Fig. 1.*** *Left panel: 3D mesh with point singularity. Middle panel: 3D mesh with edge singularity. Right panel: 3D mesh with face singularity. Bottom panel: Exemplary cross-section of the 3D mesh from the sequence of grids utilizes for the solution of the problem of projection of human head based on MRI scan data (Schaefer et al., 2015).*

## 2. ISECTION WEIGHTED BY ELEMENT SIZE

In this section we introduce our algorithm for constructing of quasi-optimal orderings. We call it *bisection weighted by element size*. Our algorithm targets 3D meshes refined towards singularities. The algorithm was tasted on three exemplary 3D meshes: mesh with point singularity, mesh with edge singularity and mesh with face singularity, see figure 1. We utilize h adaptive finite element method with constrained approximation and hanging nodes technique, as described by (Demkowicz et al., 1999; Demkowicz et al., 1998; Demkowicz et al., 2007). The algorithm constructs first the element partition tree, and later it constructs the ordering based on the obtained element partition tree.

The input for the algorithm is the computational mesh. The algorithm constructs an undirected graph



out of the finite element mesh and performs recursive partition of a graph in order to construct element partition tree. The algorithm creates an undirected graph for the input finite element mesh in such a way, that each finite element of a mesh has corresponding node in a graph. If two elements of a mesh have common face, there exists an edge between nodes corresponding to this elements. Additionally each node has attribute *'size'*. On the regular meshes, as the one we consider in this paper, the size attribute for an element is defined as the number of smallest elements (smallest cubes) it consists. For non-regular *h* refined 3D grids, the size attribute can be defined as the function of a refinement level of an element.

$$\text{size} = 2\text{^}(3*(\text{maximum refinement level} - \text{refinement level})) \quad (1)$$

Figure 2 presents a simple finite 3D mesh with maximum refinement level equal to 2, and the corresponding undirected graph. The attribute *size* is equal to 8 for elements with numbers 1,2,3. They refinement level is equal to 1. Remaining elements have refinement level equal to 2, and the attribute *size* equal to 1. The proposed algorithm works on the connectivity graph, where nodes of the graph represent particular finite elements, and edges in the graph represent the adjacency between elements. We do not assume anything about the regularity of the graph, thus direct extension to unstructured hexahedral or tetrahedral meshes is possible. After creating of a directed graph which represents a finite mesh, the bisection algorithm for the graph is used. Algorithm computes the balanced partitioning of a graph, i.e. such partitioning of a graph into two parts that the number of edges that straddle the graph is minimized and the sums of values of attribute *size* in both parts of a graph are equal. The algorithm for a graph from figure 2 will calculate two possible partitions. One partition will create two subgraphs of sum of attribute *size* equal to 16 and will cut five edges: {1,8},{1,9}, {1,10}, {1,11}, {2,3}. The second partition will also create two subgraphs of sum of attribute *size* equal to *16* and will cut five edges: {1,2},{9,10}, {8,11}, {5,6}, {4,7}.

The algorithm called bisection weighted by element size can be described in the following way:

1. Create a graph *G* for input finite element mesh (with nodes attributed by *size* of corresponding element) and assign it to the root of the element partition tree *n*

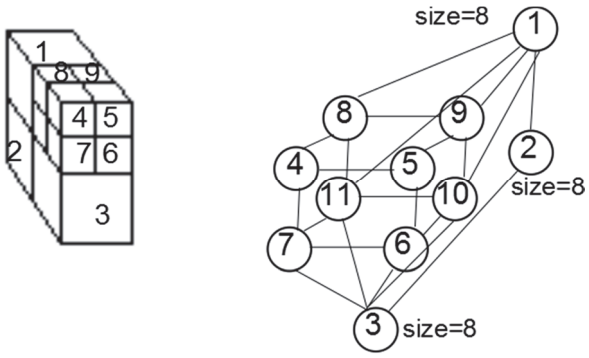



***Fig. 2.*** *A simple finite 3D mesh and the corresponding undirected graph.*

2. **if** number of nodes in *G* is equal to *1*, **then** stop
3. Calculate the balanced weighted partition of a graph *G* into two parts *G1* and *G2* and assign them to nodes *n1*, *n2* of the element partition tree.
4. Make the nodes two children of node *n*
5. Recursively calculate balanced weighted partition of graphs *G1* and *G2* and make them children of *n1* and *n2*

Figure 3 presents step by step execution of the bisection weighted by element size algorithm for the case of a mesh with point singularity. In order to make the example more readable, only the partition of the mesh (not partition of the corresponding graph) are presented. The upper part of the figure presents the mesh with point singularity. First call of bisection weighted by element size algorithm will part the mesh into two parts balanced in the sense of the value of attribute *size*, as presented in second level in figure 3 (vertical partition of a mesh). This partition cuts 10 edges of a corresponding undirected graph. However, the algorithm will also find another well balanced partition (horizontal partition of a mesh), which will cut 10 edges. The second partition is not presented in figure 3. In the next recursive call the mesh from second level from figure 3 will be part into two well balanced parts as presented in Figure 3, level 3 (horizontal partition of a mesh). This partition will cut 2 edges in one part, and 8 edges in the other part. Alternative partition for the right part, not presented in this picture, will be also considered. Third recursive call of the algorithm will part the right part of the mesh into two well balanced parts and cut 7 edges. The algorithm continues until the leaves with single elements. The following figures 4 and 5 present step by step execution of the bisection weighted by element size for the case study of a mesh with edge and face singularity.



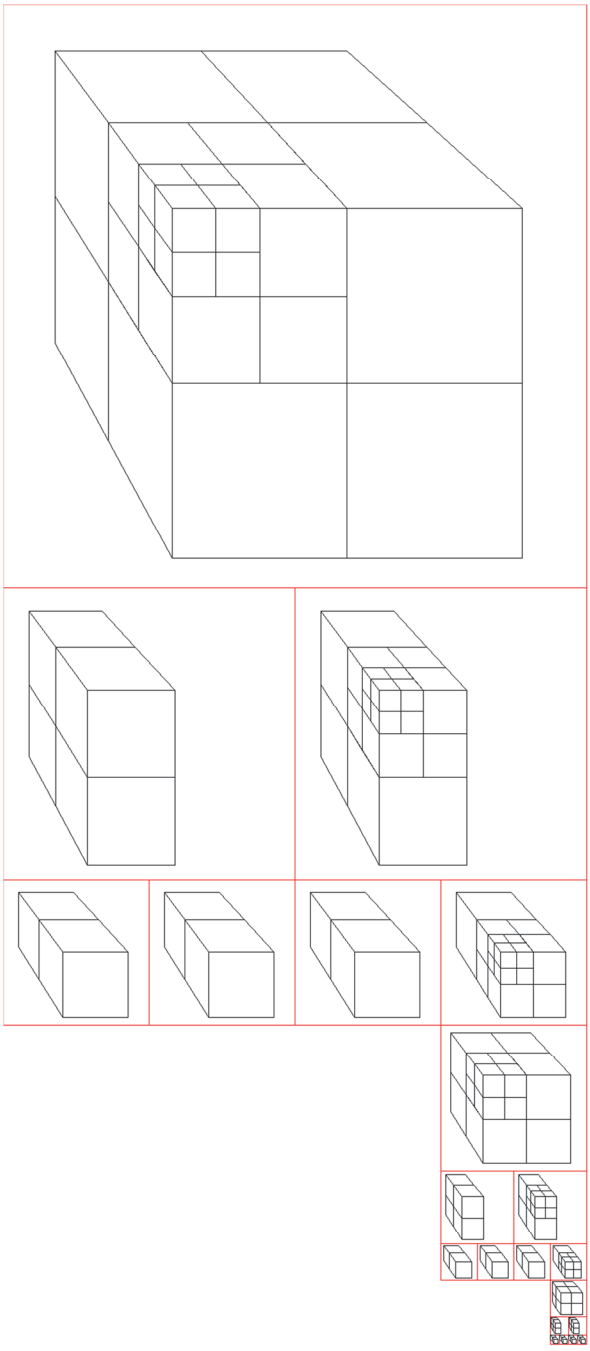


*Fig. 3. 3D mesh with point singularity. Recursive partitions by bisection weighted by element size.*

## 3. FROM ELEMENT PARTITION TREE TO ORDERING

Having the element partition trees, as the one presented in figures 3, 4 and 5, we can obtain ordering for the multi-frontal solver algorithm. The way of construction of the ordering from the element partition trees is presented in chapter 8 of book (Paszyński, 2016). The input for the ordering algorithm are:

1. The global numbering of nodes over the mesh, like the one presented in figure 6 (a)-(d).
2. List of element local matrices, with local numbering of rows and columns in terms of global nodes numbering. One way to order rows and columns in element matrices is to follow numbering of interior, faces, edges and vertices. So the left element in figure 6 has matrix with rows / columns numbering: 44 (interior) 33, 35, 37, 38, 41, 42 (faces) 13, 14, 16, 17, 20, 21, 23, 24, 27, 28, 29, 30 (edges) 1, 2, 3, 4, 7, 8, 9, 10 (vertices). The right element in figure 6 has matrix with rows / columns numbering: 45 (interior) 34, 36, 39, 40, 42, 43 (faces) 14, 15, 18, 19, 21, 22, 25, 26, 29, 30, 31, 32 (edges) 3, 4, 5, 6, 9, 10, 11, 12 (vertices). Notice that some nodes are repeated in both element matrices.
3. List of elements per node (see table 1).

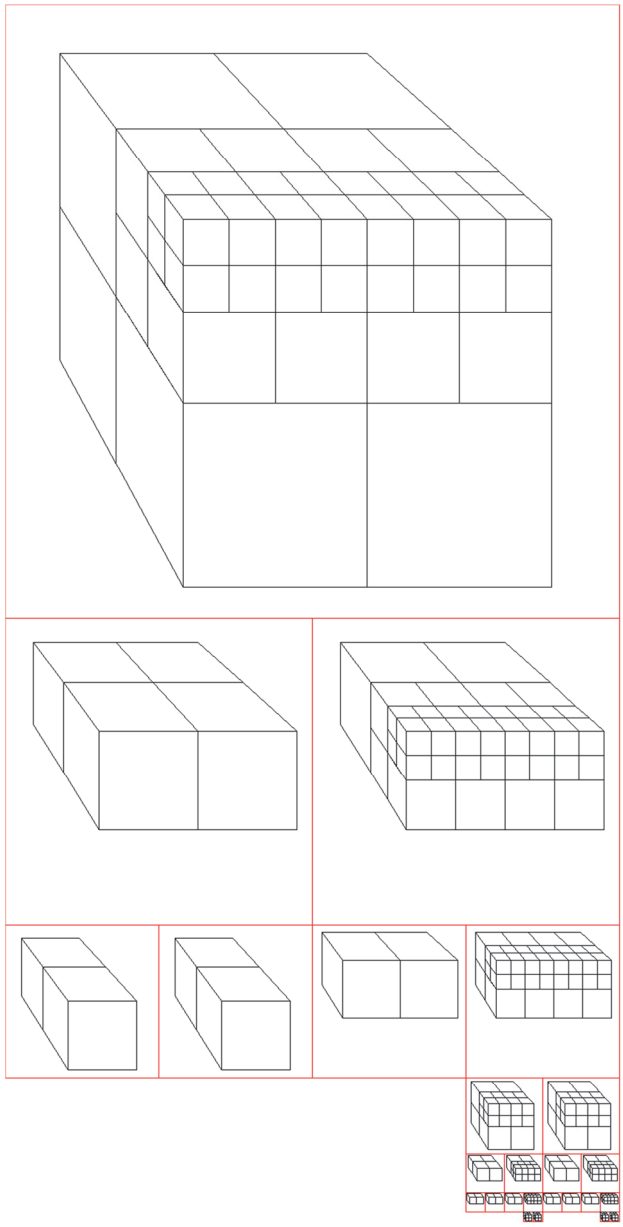


*Fig. 4. 3D mesh with edge singularity. Recursive partitions by bisection weighted by element size.*



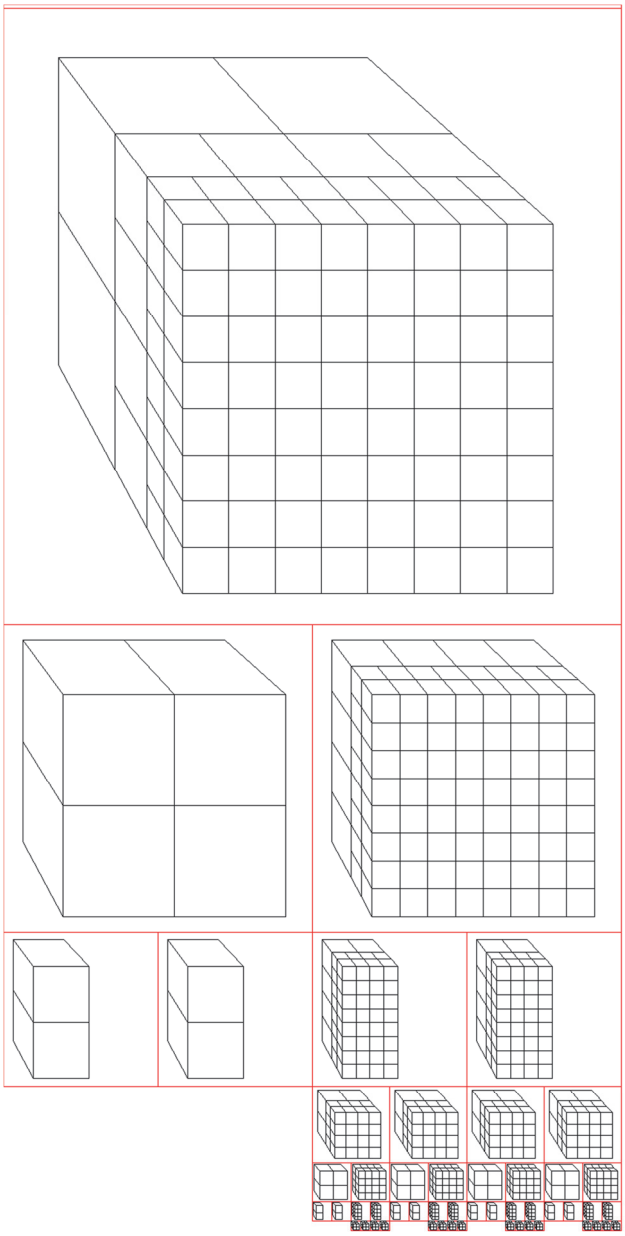


*Fig. 5. 3D mesh with face singularity. Recursive partitions by bisection weighted by element size*

The ordering algorithm pass the element partition tree in post-order. So, in our example from figure 1, it will visit first left child (element 1), then right child (element 2) then parent (elements 1,2). For each visited tree node, it localizes mesh nodes to be eliminated. The condition is the following:

*When processing tree node T from element partition tree, having list of elements LE(T) assigned, we eliminate all the mesh nodes n belonging to elements from LE(T), such that list of elements of mesh node n is a subset of lists of elements of tree node T.*

In our example:

1. The algorithm will visit left child tree node of elimination tree with LE(T)={1}. It will check nodes from element matrix assigned to element 1, namely 44 (interior) 33, 35, 37, 38, 41, 42 (faces) 13, 14, 16, 17, 20, 21, 23, 24, 27, 28, 29, 30 (edges) 1, 2, 3, 4, 7, 8, 9, 10 (vertices) and it will eliminate nodes 44 (interior), 33, 35, 37, 38, 41 (faces) 13, 16, 17, 20, 23, 24, 27, 28 (edges) 1, 2, 7, 8 (vertices), since their LE(n ={1}= LE(T)
2. The algorithm will visit right child tree node of elimination tree with LE(T)={2}. It will check nodes from element matrix assigned to element 2, namely 45 (interior) 34, 36, 39, 40, 42, 43 (faces) 14, 15, 18, 19, 21, 22, 25, 26, 29, 30, 31, 32 (edges) 3, 4, 5, 6, 9, 10, 11, 12 (vertices) and it will eliminate nodes 44 (interior), 33, 35, 37, 38, 41 (faces) 13, 16, 17, 20, 23, 24, 27, 28 (edges) 1, 2, 7, 8 (vertices), since since their LE(n)={2}=LE(T)

***Table 1.*** *Mappings from nodes to elements*

| Node | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Elements | 1 | 1 | 1,2 | 1,2 | 2 | 2 | 1 | 1 | 1,2 | 1,2 | 2 | 2 | 1 | 1,2 | 2 |
| Node | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Elements | 1 | 1 | 2 | 2 | 1 | 1,2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 1,2 | 1,2 |
| Node | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| Elements | 2 | 2 | 1 | 2 | 1 | 2 | 1 | 1 | 2 | 2 | 1 | 1,2 | 2 | 1 | 2 |

4. Element partition tree, like the one presented in figure 6 (e). The leaves of the element partition tree have element number assigned. The parent nodes sum up the lists from child nodes. Thus, the root node has list of elements assigned.

3. The algorithm will visit root tree node of elimination tree with LE(T)={1, 2}. The lists of uneliminated nodes from both child nodes will be merged to obtain 42 (face) 14, 21, 29, 30 (edges), 3, 4, 9, 10 (vertices). All the nodes will be eliminated, since theire LE(n)={1, 2}=LE(T).



So the final ordering is the following: 44 (interior), 33, 35, 37, 38, 41 (faces) 13, 16, 17, 20, 23, 24, 27, 28 (edges) 1, 2, 7, 8 (vertices); 44 (interior), 33, 35, 37, 38, 41 (faces) 13, 16, 17, 20, 23, 24, 27, 28 (edges) 1, 2, 7, 8 (vertices); 42 (face) 14, 21, 29, 30 (edges), 3, 4, 9, 10 (vertices).

and face singularities, presented in figure 1. The weighted partitions generated by the algorithm are summarized in figure 3, 4 and 5, for point, edge and face singularities, respectively. The element partition trees are processed by our multi-frontal direct solver with GALOIS scheduler (Paszyńska et al., 2015),

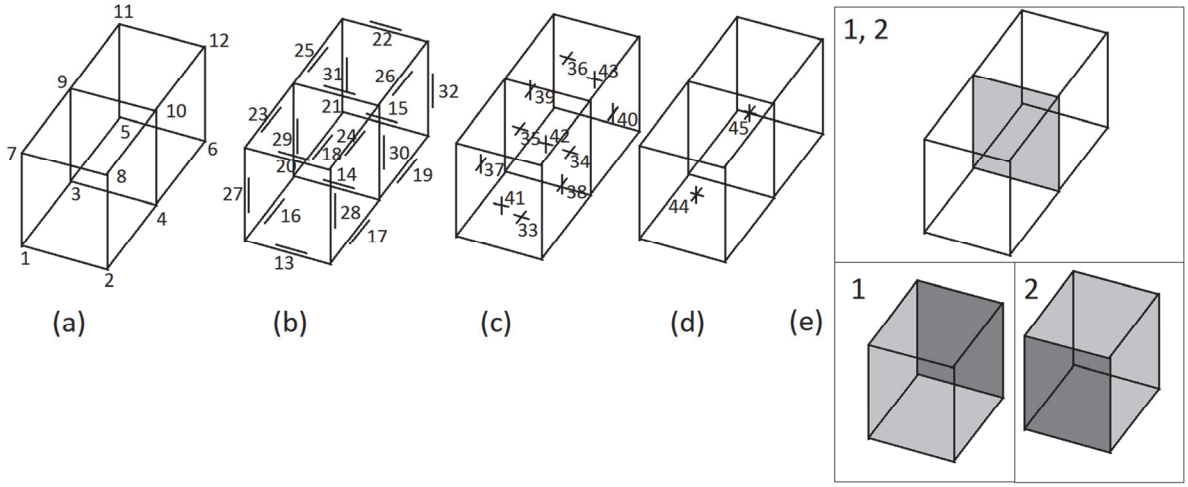



***Fig. 6.*** *Two finite element mesh and its element partition tree (a) Numbering of vertex nodes (b) Numbering of edge nodes (c) numbering of face nodes (d) numbering of interior nodes (e) element partition tree.*

executed in sequential mode (single core). The algorithm generates corresponding orderings, as described in section 3, and produces some FLOPs during rows eliminations. We compare number of FLOPs generated by our element partition tree controlled solver algorithm with AMD, METIS and PORD ordering algorithms available through MUMPS interface. The comparison is presented in figure 7 and tables 2-4. For all the grids, our bisection weighted by element size algorithm outperforms the alternative orderings.

***Table 2.*** *Comparison of FLOPs for point singularity*

| Nodes | Bissections | METIS | AMD, PORD |
|---|---|---|---|
| 125 | 72095 | 84900 | 81030 |
| 181 | 122275 | 159000 | 153300 |
| 237 | 172455 | 232400 | 217600 |
| 293 | 222635 | 300600 | 289900 |
| 349 | 272815 | 379300 | 362100 |
| 405 | 322995 | 491200 | 434400 |
| 461 | 373175 | 566100 | 506700 |

***Table 3.*** *Comparison of FLOPs for edge singularity*

| Nodes | Bissections | METIS | AMD, PORD |
|---|---|---|---|
| 125 | 72095 | 84900 | 81030 |
| 249 | 244785 | 296700 | 281900 |
| 485 | 805867 | 1070000 | 894600 |
| 945 | 2474501 | 3358000 | 2662000 |
| 1853 | 6952239 | 9020000 | 7949000 |
| 3657 | 17988361 | 24040000 | 21760000 |
| 7253 | 43510547 | 66380000 | 55070000 |

***Table 4.*** *Comparison of FLOPs for face singularity*

| Nodes | Bissections | METIS | AMD, PORD |
|---|---|---|---|
| 125 | 72095 | 84900 | 81030 |
| 399 | 687516 | 860800 | 799400 |
| 1393 | 7590013 | 10990000 | 13320000 |
| 5171 | 79621866 | 113600000 | 240300000 |
| 19893 | 774584899 | 1,081E+09 | 3,105E+09 |

## 4. NUMERICAL RESULTS

We verify our bisection weighted by element size algorithm over 3D grids *h* refined towards edge



***Table 6.*** *Comparison of FLOPS and execution times for a sequence of grids for the projection of the human head*

| N | Bissections | MUMPS | Gain |
|---|---|---|---|
| 8125 | <1s | <1s | |
| 35301 | <1s | <1s | |
| 60025 | 5,70[s] | 5,67[s] | 0,99 |
| 94221 | 14,49 [s] | 28,29 [s] | 1,9 |
| 139425 | 33,06 [s] | 67,94 [s] | 2,05 |
| 197173 | 71,59[s] | 142,64[s] | 1,99 |

The FLOPs comparison makes only sense on sequential processors nowadays, and the processor itself may perform additional vectorization to reduce the effective number of FLOPs. To compare our orderings algorithm in term of the execution time, which at the end is what really counts, we have executed our algorithm on the most complicated case, the three dimensional *h* refined grids utilized in the projection of the MRI scan data of the human head (Schaefer et al., 2015). We have generated our

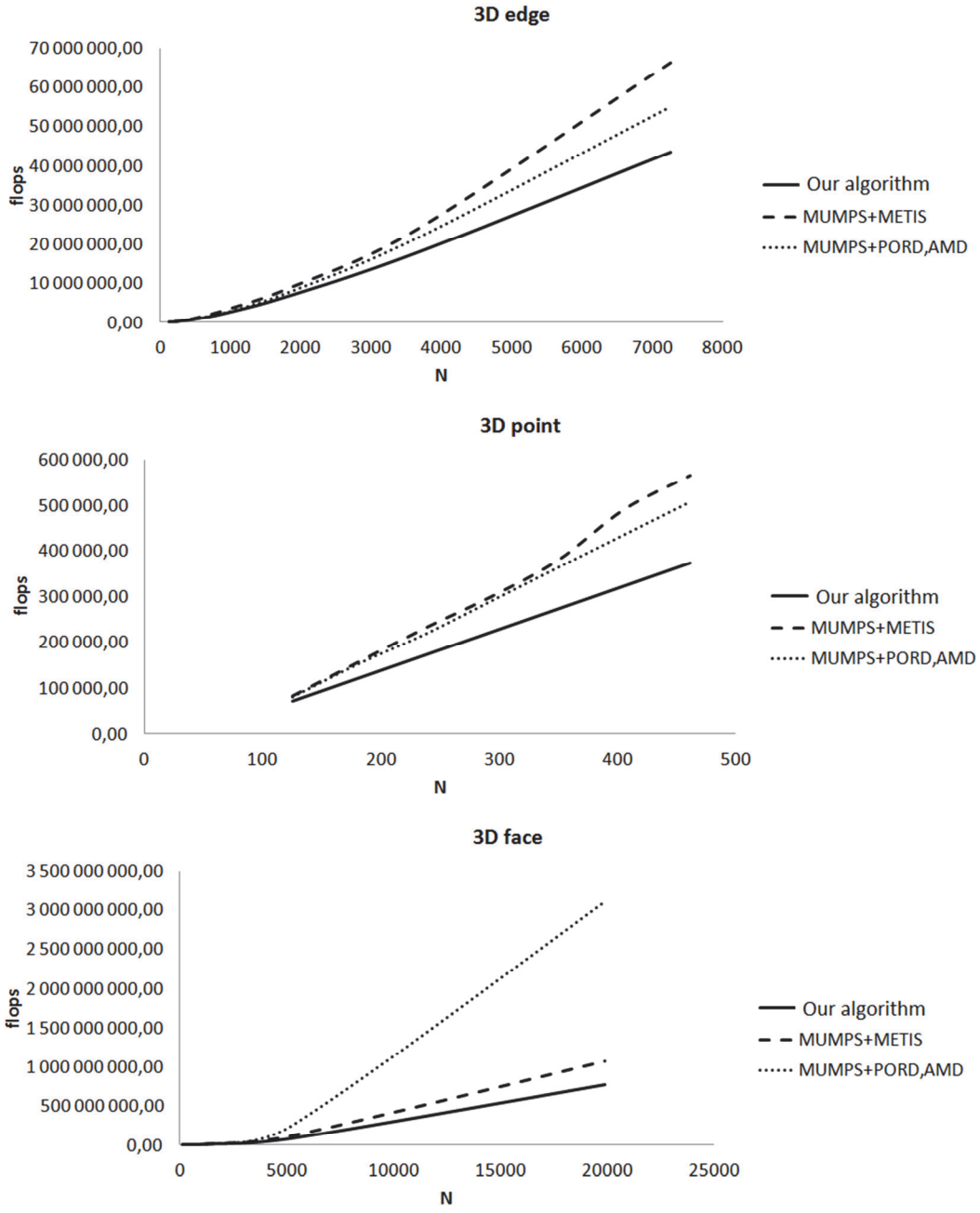



***Fig. 7.*** *Scalability of the bisection weighted by element size algorithm in comparison to MUMPS solver. Comparison of FLOPs. Top panel: 3D edge singularity. Middle panel: 3D point singularity. Bottom panel: 3D face singularity.*



ordering and send it to MUMPS solver through PERM_IN array. We compare the execution time versus the automatic selection of the ordering performed by MUMPS. The results of the comparison are presented in table 6.

## 5. CONCLUSIONS

We presented a bisection weighted by element size ordering algorithm for controlling execution of the multi-frontal solver algorithm over 3D computational grids with singularities. The algorithms has been verified numerically and compared against MUMPS solver interfaced with AMD, PORD and METIS ordering algorithms. Our algorithm outperforms alternative ordering algorithms in the number of floating point operations (FLOPs) and with respect to the sequential execution times. Our future work may involve development of the parallel version of these ordering algorithms and working on better measures for the parallel execution, like measurements of the memory transfers and communication costs.

## ACKNOWLEDGEMENTS

The work presented in this paper has been supported by National Science Centre, Poland grant no. DEC-2012/06/M/ST1/00363.

## REFERENCES

AbouEisha, H., Calo, V. M., Jopek, K., Moshkov, M., Paszyńska, A., Paszyński, M., Skotniczny, M., 2016, Element Partition Trees for Two- and Three-Dimensional h-Refined Meshes and Their Use to Optimize Direct Solver Performance, submitted to *Journal of Computational Science.*

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

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

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

Cottrel, J. A., Bazilevs, Y., Hughes, T.J.R., 2009. *Isogeometric Analysis: Toward Integration of CAD and FEA*, Dover.

Bazilevs, Y., Calo, V.M., Cottrell, J.A., Evans, J.A., Lipton, S., Scott, M.A., Sederberg, T.W., 2010. Isogeometric analysis using T-splines, *Computer Methods in Applied Mechanics and Engineering*, (199) 229-263.

Demkowicz, L., Bajer, A., Rachowicz, W., Gerdes, K., 1999. *ICES Report* 99-29, 3D hp-adaptive finite element package (3Dhp90).

Demkowicz, L., Kurtz, J., Pardo, D., Paszyński, M., Rachowicz, W., Zdunek, A., 2007. *Computing with hp-Finite Elements. Vol. II. Frontiers Three Dimensional Elliptic and Maxwell Problems with Applications,* Chapmann & Hall/Crc Applied Mathematics & Nonlinear Science.

Demkowicz, L, Walsh, T., Gerdes, K., Bajer, A., 1998. *ICES Report* 98-14, 2D hp-adaptive finite element package (2Dhp90).

George, A., Liu, J.W.-H., 1978, An automatic nested dissection algorithm for irregular finite element problems, *SIAM Journal of Numerical Analysis* 15, 1053-1069.

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

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

MUMPS, MUlti-frontal Massively Parallel Sparse direct solver MUMPS, available online at: http://mumps.enseeiht.fr/ accessed: 8.07.2016.

Paszyńska A., Paszyński M., Jopek K., Woźniak M., Goik D., Gurgul P., AbouEisha H., Moshkov M., Calo V. M., Lenharth V. M., Nguyen D., Pingali K., 2015, Quasi-Optimal Elimination Trees for 2D Grids with Singularities, *Scientific Programming*, Article ID 303024:1-18.

Paszyński, M., 2016, *Fast Solvers for Mesh Based Computations*, Taylor and Francis, CRC Press.

Paszynski, M., Pardo, D., Calo, V. M., 2015, Direct Solver Performance on h-Adaptive Grids, *Computers & Mathematics with Applications*, 70(3), 282-295.

Schaefer, R., Łoś, M., Sieniek, M., Demkowicz, L., Paszyński, M., 2015, Quasi-linear computational cost adaptive solvers for three dimensional modeling of heating of a human head induced by cell phone, *Journal of Computational Science,* (11), 163-174.

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

## ALGORYTM BISEKCJI WAŻONYCH ROZMIAREM ELEMENTÓW DO GENERACJI PORZĄDKU KIERUJĄCEGO WYKONANIEM SOLWERÓW WIELOFRONTALNYCH DZIAŁAJĄCYCH NA TRÓJWYMIAROWYCH *h* ADAPTOWANYCH SIATKACH OBLICZENIOWYCH

Streszczenie

W artykule prezentujemy algorytm generacji porządku eliminacji kierujący wykonaniem solwera wielo-frontalnego dla trójwymiarowych siatek h adaptowanych do osobliwości punktowych, krawędziowych i ścianowych. Wygenerowany porządek generuje permutacje macierzy układu równań liniowych uzyskanych podczas obliczeń trójwymiarową metodą elementów skończonych. Proponowany algorytm dostarcza porządku eliminacji który pozwala wykonywać faktoryzację z mniejszą liczbą operacji zmienno-przecinkowych niż klasyczne algorytmy generacji porządku dostępne za pośrednictwem solwera MUMPS, takie jak nested-dissections, AMD oraz PORD. Nasz algorytm wykorzystuje dodatkową wiedzę o strukturze siatki obliczeniowej, nie dostępną dla alternatywnych algorytmów generacji porządku.

*Received: April 11, 2016*
*Received in a revised form: July 26, 2016*
*Accepted: August 8, 2016*