Bisections-weighted-by-element-size-and-order algorithm to optimize direct solver performance on 3D *hp*-adaptive grids.

H. AbouEisha[1], V. M. Calo [2,3,4], K. Jopek[5], M. Moshkov[1], A. Paszyńska[6], M. Paszyński[5]

[1] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
mikhail.moshkov@kaust.edu.sa
[2] Chair in Computational Geoscience, Applied Geology Department, Western Australian School of Mines, Faculty of Science and Engineering, Curtin University, Perth, WA, Australia
victor.calo@curtin.edu.au
[3] Mineral Resources, Commonwealth Scientific and Industrial Research Organization (CSIRO), Kensington, WA, Australia 6152
[4] Curtin Institute for Computation, Curtin University, Perth, WA, Australia 6845
[5] AGH University of Science and Technology,
Faculty of Computer Science, Electronics and Telecommunications
al. Mickiewicza 30, 30-059 Krakow, Poland
paszynsk@agh.edu.pl
[6] Jagiellonian University,
Faculty of Physics, Astronomy and Applied Computer Science
Łojasiewicza 11, 30-348 Krakow, Poland
anna.paszynska@uj.edu.pl
http://home.agh.edu.pl/paszynsk

Abstract. The *hp*-adaptive Finite Element Method (*hp*-FEM) generates a sequence of adaptive grids with different polynomial orders of approximation and element sizes. The *hp*-FEM delivers exponential convergence of the numerical error with respect to the mesh size. In this paper, we propose a heuristic algorithm to construct element partition trees. The trees can be transformed directly into the orderings, which control the execution of the multi-frontal direct solvers during the *hp* refined finite element method. In particular, the orderings determine the number of floating point operations performed by the solver. Thus, the quality of the orderings obtained from the element partition trees is important for good performance of the solver. Our heuristic algorithm has been implemented in 3D and tested on a sequence of *hp*-refined meshes. We compare the quality of the orderings found by the heuristic algorithm to those generated by alternative state-of-the-art algorithms. We show 50 percent reduction in flops number and execution time.[1]

Key words: *hp* adaptive finite element method, ordering, nested-dissections, multi-frontal direct solvers, heuristic algorithms

[1] Authors in alphabetical order. The work was supported by National Science Centre, Poland grant no. DEC-2015/17/B/ST6/01867.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

1 Introduction

The finite element method is a widely used approach finding an approximate solution of partial differential equations (PDEs) specified along with boundary conditions and a solution domain. A mesh with hexahedral elements is created to cover the domain and to approximate the solution over it. Then the weak form of the PDE is discretized using polynomial basis functions spread over the mesh. The *hp*-adaptive Finite Element Method (*hp*-FEM) is the most sophisticated version of FEM [9]. It generates a sequence of refined grids, providing exponential convergence of the numerical error with respect to the mesh size. The *hp*-FEM algorithm uses the coarse and the fine meshes in each iteration to compute the relative error and to guide the adaptive refinement process. Selected finite elements are broken into smaller elements. This procedure is called the *h*-refinement. Also, the polynomial orders of approximation are updated on selected edges, faces, and interiors. This procedure is called the *p*-refinement. In selected cases, both *h* and *p* refinements are performed, and this process is called the *hp*-refinement.

The *hp*-FEM is used to solve difficult PDEs, e.g. with local jumps in material data, with boundary layers, strong gradients, generating local singularities, requiring elongated adaptive elements, or utilization of elements with several orders of magnitude difference in dimension. For such kind of meshes iterative solvers deliver convergence problems.

This paper is devoted to the optimization of the element partition trees controlling the LU factorization of systems of linear equations resulting from the *hp*-FEM discretizations over three-dimensional meshes with hexahedral elements. In this paper we focus on a class of *hp* adaptive grids, which has many applications in different areas of computational science and several possible implementations [9, 7, 21, 26, 27, 22, 6, 8, 28]. The LU factorization for the case of *hp*-adaptive finite element method is performed using multi-frontal direct solvers, such as e.g. MUMPS solver [2–4]. This is because the matrices resulting from the discretization over the computational meshes are sparse, and smart factorization will generate a low number of additional non-zero entries (so-called fill-in) [17, 18]. The problem of finding the optimal permutation of the sparse matrix which minimizes the fill-in (the number of new non-zero entries created during the factorization) is NP-complete [29]. In this paper, we propose a heuristic algorithm that works for arbitrary *hp*-adaptive gird, with finite elements of different size and with a different distribution of polynomial orders of approximation spread over finite element edge, faces, and possibly interiors. The algorithm performs recursive weighted partitions of the graph representing the computational mesh and uses these partitions to generate an ordering, which minimizes the fill-in in a quasi-optimal way. The partitions are defined by so-called element partition tree, which can be transformed directly into the ordering.

In this paper we focus on the optimization of the sequential in-core multi-frontal solver [11–13], although the orderings obtained from our element partition trees can be possibly utilized to speed up shared-memory [14–16] or distributed-memory [2–4] implementations as well. This will be the topic of our future work.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

The heuristic algorithm proposed in this paper is based on the insights we gained in [1], where we proposed a dynamic programming algorithm to search for quasi-optimal element partition trees. These quasi-optimal trees obtained in [1] are too expensive to generate, and they cannot be used in practice, but rather guide our heuristic methods. From the insights garnered from this optimization process, we have proposed a heuristic algorithm that generates quasi-optimal element partition trees for arbitrary h-refined grids in 2D and 3D. In this paper, we generalize the idea presented in [1] to the class of hp-adaptive grids. The heuristic algorithm uses multilevel recursive bisections with weights assigned to element edges, faces, and interiors. Our heuristic algorithm has been implemented and tested in three-dimensional case. It generates mesh partitions for arbitrary hp-refined meshes, by issuing recursive calls to *METIS_WPartGraphRecursive*. That is, we use the multilevel recursive bisection implemented in METIS [20] available through the MUMPS interface [2–4], to find a balanced partition of a weighted graph. We construct the element partition tree by recursive calls of the graph bisection algorithm. Our algorithm for the construction of the element partition tree and the corresponding ordering differs from the orderings used by the METIS library (nested dissection) as follows. First, we use a smaller graph, built from the computational mesh, with vertices representing the finite elements and edges representing the adjacency between elements. Second, we weight the vertices of the graph by the volume of finite elements multiplied by the polynomial orders of approximations in the center of the element. Third, we weight the edges of the graph by the polynomial orders of approximations over element faces.

Previously [23, 24], we have proposed bottom-up approaches for constructing element partition trees for h-adaptive grids. Herein, we propose an alternative algorithm, bisections-weighted-by-element-size-and-order, to construct element partition trees using a top-down approach, for hp-adaptive grids. The element size in our algorithm is a proxy for refinement level of the element. The order is related to the polynomial degrees used on finite element edges, faces and interiors.

The plan of the paper is the following. We first define the computational mesh and basis functions which illustrate how these computational grids are transformed into systems of linear equations using the finite element method. Then, we describe the idea of a new heuristic algorithm which uses bisections weighted by elements sizes and polynomial orders of approximation. We show how the ordering can be generated from our element partition tree. The next section includes numerical tests which compare the number of floating point operations and wall-clock time resulting from the execution of the multi-frontal direct solver algorithm on the alternative orderings under analysis.

2 Meshes, matrices and orderings for the *hp*-adaptive finite element methods

We introduce a class of computational meshes that results from the application of an adaptive finite element method [9]. For our analysis, we start from a three-

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

dimensional boundary-value elliptic partial differential equation problem in its weak (variational) form given by (1): Find $u \in V$ such that

$$b\left(u,v\right)=l\left(v\right) \quad \forall v \in V \tag{1}$$

where $b\left(u,v\right)$ and $l\left(v\right)$ are some problem-dependent bilinear and linear functionals, and

$$V=\{v : \int_{\Omega} \|v\|^2+\|\nabla v\|^2 dx<\infty, tr\left(v\right)=0 \text{ on } \Gamma_D\} \tag{2}$$

is a Sobolev space over an open set Ω called the domain, and Γ_D is the part of the boundary of Ω where Dirichlet boundary conditions are defined.

For a given domain Ω the hp-FEM constructs a finite dimensional subspace $V_{hp} \subset V$ with a finite dimensional polynomial basis given by $\{e^i_{hp}\}_{i=1,...,N_{hp}}$. The subspace V_{hp} is constructed by partitioning the domain Ω into three-dimensional finite elements, with vertices, edges, faces, and interiors, as well as shape functions defined over these objects.

Namely, we introduce one-dimensional shape-functions

$$\hat{\chi}_1(\xi)=1-\xi; \quad \hat{\chi}_2(\xi)=\xi; \quad \hat{\chi}_l(\xi)=(1-\xi)\xi(2\xi-1)^{l-3}, l=4,...,p+1 \tag{3}$$

where p is the polynomial order of approximation, and we utilize them to define the three-dimensional hexahedral finite element $\{(\xi_1,\xi_2,\xi_3) : \xi_i \in [0,1], i=1,3\}$. We define eight shape functions over the eight vertices of the element:

$$\begin{aligned}
&\hat{\phi}_1(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_1(\xi_3) \quad \hat{\phi}_2(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_1(\xi_3)\\
&\hat{\phi}_3(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_1(\xi_3) \quad \hat{\phi}_4(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_1(\xi_3)\\
&\hat{\phi}_5(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_2(\xi_3) \quad \hat{\phi}_6(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_2(\xi_3)\\
&\hat{\phi}_7(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_2(\xi_3) \quad \hat{\phi}_8(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_2(\xi_3)
\end{aligned} \tag{4}$$

$j=1,...,p_i-1$ shape functions over each of the twelve edges of the element

$$\begin{aligned}
&\hat{\phi}_{9,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_1(\xi_3) \quad \hat{\phi}_{10,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_1(\xi_3)\\
&\hat{\phi}_{11,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_1(\xi_3) \quad \hat{\phi}_{12,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_1(\xi_3)\\
&\hat{\phi}_{13,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_2(\xi_3) \quad \hat{\phi}_{14,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_2(\xi_3)\\
&\hat{\phi}_{15,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_2(\xi_3) \quad \hat{\phi}_{16,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_2(\xi_3)\\
&\hat{\phi}_{17,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_{2+j}(\xi_3) \quad \hat{\phi}_{18,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_{2+j}(\xi_3)\\
&\hat{\phi}_{19,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_{2+j}(\xi_3) \quad \hat{\phi}_{20,j}(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_{2+j}(\xi_3)
\end{aligned} \tag{5}$$

where p_i is the polynomial order of approximation utilized over the i-th edge. We also define $(p_{ih}-1)\times(p_{iv}-1)$ shape functions for $j=1,...,p_{ih}-1$ and $k=1,...,p_{iv}-1$, over each of six faces of the element

$$\begin{aligned}
&\hat{\phi}_21(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_{2+k}(\xi_2)\hat{\chi}_1(\xi_3) \quad \hat{\phi}_22(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_{2+k}(\xi_2)\hat{\chi}_2(\xi_3)\\
&\hat{\phi}_23(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_1(\xi_2)\hat{\chi}_{2+k}(\xi_3) \quad \hat{\phi}_24(\xi_1,\xi_2,\xi_3)=\hat{\chi}_2(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_{2+k}(\xi_3)\\
&\hat{\phi}_25(\xi_1,\xi_2,\xi_3)=\hat{\chi}_{2+j}(\xi_1)\hat{\chi}_2(\xi_2)\hat{\chi}_{2+k}(\xi_3) \quad \hat{\phi}_26(\xi_1,\xi_2,\xi_3)=\hat{\chi}_1(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_{2+k}(\xi_3)
\end{aligned} \tag{6}$$

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

where p_{ih}, p_{iv} are the polynomial orders of approximations in two directions in the i-th face local coordinates system. Finally, we define $(p_x-1)\times(p_y-1)\times(p_z-1)$ basis functions over an element interior

$$\hat{\phi}_{27,ij}(\xi_1,\xi_2) = \hat{\chi}_{2+i}(\xi_1)\hat{\chi}_{2+j}(\xi_2)\hat{\chi}_{2+k}(\xi_3) \tag{7}$$

where (p_x, p_y, p_z) are the polynomial orders of approximation in three directions, respectively, utilized over an element interior. The shape functions from the adjacent elements that correspond to identical vertices, edges, or faces, they are merged to form global basis functions.

The support interactions of the basis functions defined over the mesh determine the sparsity pattern for the global matrix.

In the example presented in Figure 1 there are first order polynomial basis functions associated with element vertices, second order polynomials associated with element edges, and second order polynomials in both directions, associated with element interiors. For more details we refer to [9].

We illustrate these concepts with two-dimensional example. Figure 1 presents an exemplary two-dimensional mesh consisting of rectangular finite elements with vertices, edges and interiors, as well as shape functions defined over vertices, edges and interiors of rectangular finite elements of the mesh.

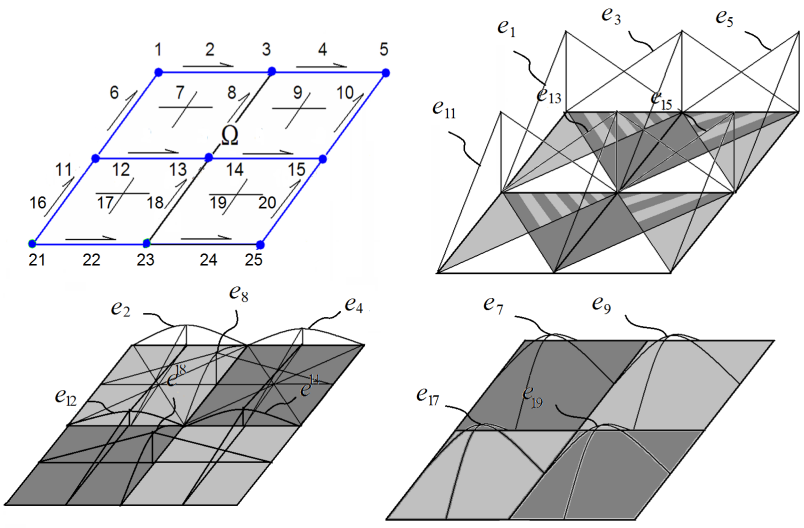

Fig. 1. Examplary four element mesh and basis functions spread over the mesh

The interactions of supports of basis functions defined over the mesh define the sparsity pattern for the global matrix. In other words, i-th row and j-th column of the matrix is non-zero, if supports of i-th and j-th basis functions

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

overlap. For example, for the $p = 1$ case the global matrix looks like it is presented in Figure 2. In this case, only vertex functions are present. For $p = 2$, all the basis functions are interacting, and this corresponds to the case presented in Figure 3.

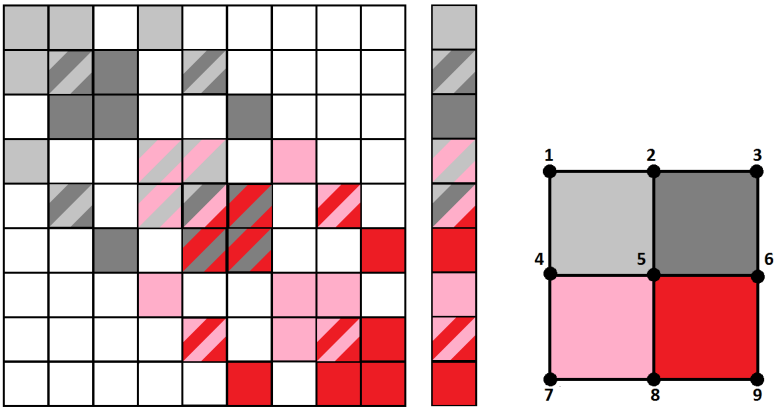

Fig. 2. Matrix resulting from four element mesh with $p = 1$ vertex basis functions.

Traditional sparse matrix solvers construct the ordering based on the sparsity pattern of the global matrix. This is illustrated in the top path in Figure 4. The sparse matrix is submitted to an ordering generator, e.g., the nested-dissections [20] or the AMD [5] algorithms from the METIS library. The ordering is utilized later to permute the sparse matrix, which results in less non-zero entries generated during the factorization, and lower computational cost of the factorization procedure. In the meantime, the elimination tree is constructed internally by the sparse solver, which guides the elimination procedure [2]

The alternative approach is discussed in this paper. We construct the element partition tree based on the structure of the computational mesh, using the weighted bisections algorithm. The element partition tree is then browsed in post-order, to obtain the ordering, which defines how to permute the sparse matrix. This is illustrated on the bottom path presented in Figure 4. For a detailed description on how to construct ordering based on an element partition tree, we refer to chapter 8 of the book [25].

The sparsity pattern of the matrix rather not depend on the elliptic PDE being solved over the mesh. It strongly depends on the basis functions and the topology of the computational mesh.

[2] In [25] the name elimination tree was also used for the element partition tree.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

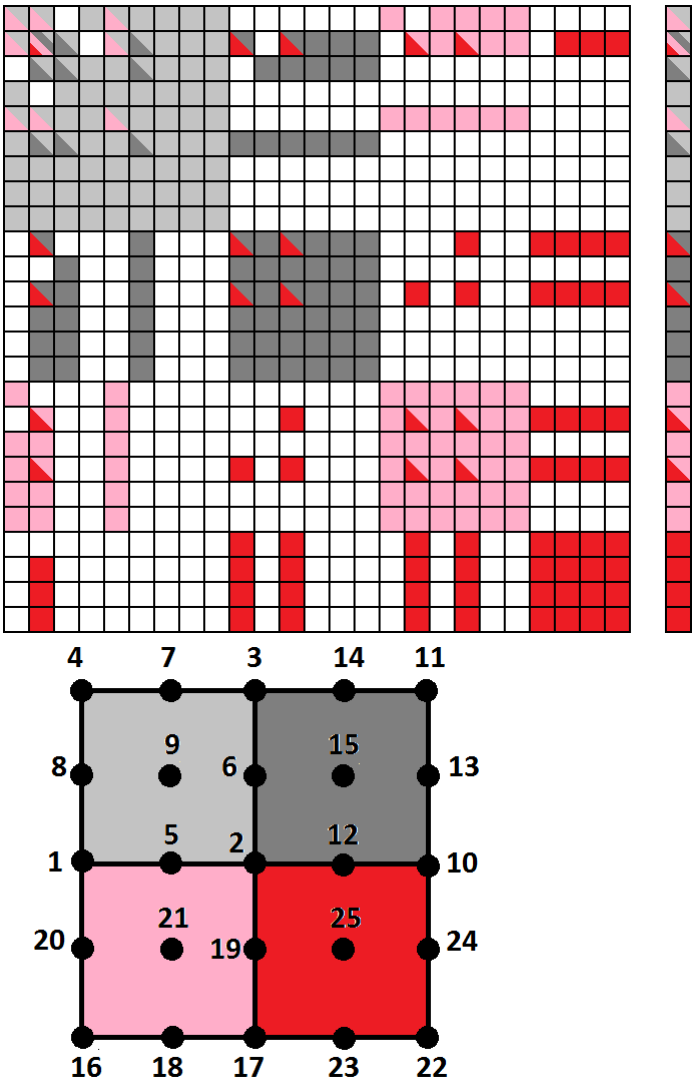

Fig. 3. Matrix resulting from four element mesh with $p = 2$ basis functions related to element vertices, edges, faces and interios.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

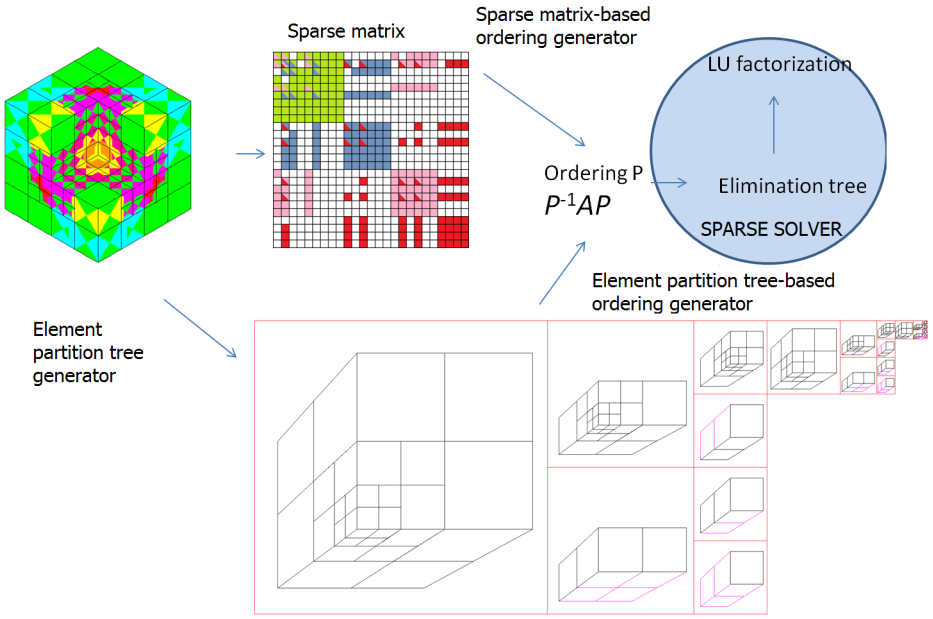

Fig. 4. The construction of the ordering based on sparsity pattern of the matrix, and based on the element partition tree.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

3 Bisections-weighted-by-element-size-and-order

The algorithm of bisections-weighted-by-element-size-and-order creates an initial undirected graph G for finite element mesh. Each node of the graph corresponds to one finite element from the mesh. An edge in the graph G exists if the corresponding finite elements have a common face. Additionally, each node of the graph G has an attribute *size* that is defined as follows. For the regular meshes, as considered in this paper, the size of an element is defined as the volume of the element times the order of the element. For general three-dimensional grids, the volume attribute is defined as the function of a refinement level of an element:

$$volume = 2^{(3*(max\_refinement\_level - refinement\_level))}(p_x - 1)(p_y - 1)(p_z - 1) \quad (8)$$

Moreover, each vertex of graph G has an attribute *weight* defined as the polynomial order of approximation of the face between two neighboring elements. The elements in the three-dimensional mesh may be neighbors through a vertex, an edge, or a face. In these cases, the weight of the edge corresponds to the vertex order (always equal to one), the edge order (defined as $p_{edge} - 1$) or the face order (defined as $(p_{ih} - 1) \times (p_{iv} - 1)$. This is illustrated in Figure 5.

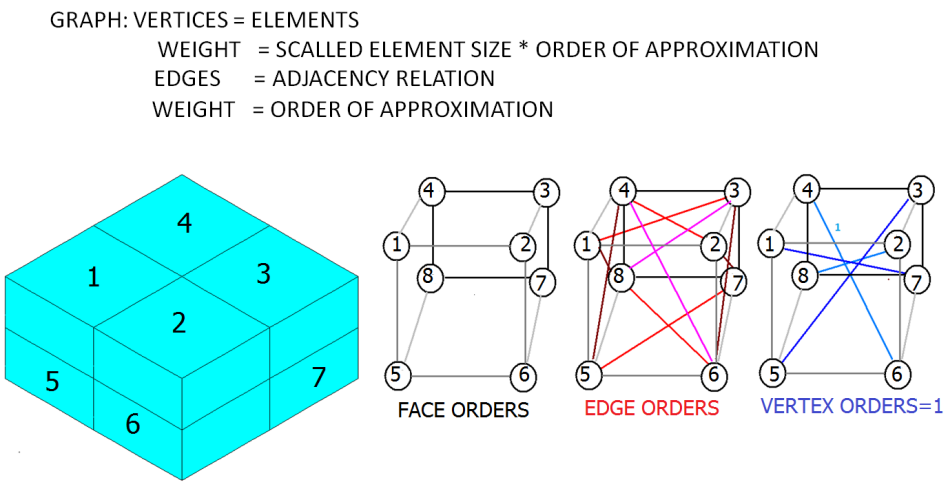

Fig. 5. The exemplary three-dimensional mesh and its weighted graph representation.

The function named BisectionWeightedByElementSizeOrder() is called initially with the entire graph G, and later it is called recursively with sub-graphs of G. It generates the element partition tree. The *BisectionWeightedByElementSizeOrder* function is defined as follows:

function `BisectionWeightedByElementSizeOrder(G)`
If `number of nodes in` G

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

```
  is equal to 1 then
  create one element tree t with the node v ∈ G; return t;
else
  Calculate the balanced weighted partition of G into G1 and G2;
  //calling METIS WPartGraphRecursive() for G
  t1 = BisectionWeightedByElementSizeOrder(G1);
  t2 = BisectionWeightedByElementSizeOrder(G2);
  create new root node t with left child t1 and right child t2
  return t
endif
```

Once the algorithm generates the element partition tree, we extract the ordering and call a sequential solver. Herein, we use `METIS_WPartGraphRecursive` [20] function to find a balanced partition of a graph, where weights on vertices are equal to the size value of the corresponding mesh elements. The `METIS_WPartGraphRecursive` uses the Sorted Heavy-EdgeMatching method during the coarsening phase, the Region Growing method during partitioning phase and the Early-Exit Boundary FM refinement method during the un-coarsening phase.

4 Numerical results

In this section, we compare the number of flops of the MUMPS multi-frontal direct solver [2–4] with the ordering obtained from the element partition trees generated by the bisections-weighted-by-element-size-and-order algorithm, and the MUMPS with automatic selection of the ordering algorithm, compiled with *icntl(7)=7*. The MUMPS solver chooses either nested-dissection [20] or approximate minimum degree algorithm [5] for this kind of problem, depending on the properties of the sparse matrix. We focus on the model Fichera problem [9, 10]: Find u temperature scalar field such that $\nabla u = 0$ on Ω being 7/8 of the cube, with zero Dirichlet b.c. on the internal 1/8 boundary, and Neumann b.c. on the external boundary, computed from the manufactured solution. This model problem has strong singularities at the central point, and along the three internal edges, thus the intensive refinements are required.

The *hp*-FEM code generates a sequence of *hp*-refined grids delivering exponential convergence of the numerical error with respect to the mesh size, as presented in Figure 6. The comparison of flops and wall time concerns the last two grids, the coarse, and the corresponding fine grids, generated by the *hp*-FEM algorithm, with various polynomial orders of approximation, and element sizes, as presented in Figure 7. It is summarized in Table 1.

To verify the flops and the wall-time performance of our algorithm against alternative ordering provided by MUMPS, we use the PERM IN input array of the library. The *hp*-FEM code generates a sequence of optimal grids. The decisions about the optimal mesh refinements are performed by using the reference solution on the fine grids, obtained by the global *hp*-refinement of the coarse grids. We compare the flops and the wall time-performance on the last two iterations performed by the adaptive algorithm, where the relative error, defined

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

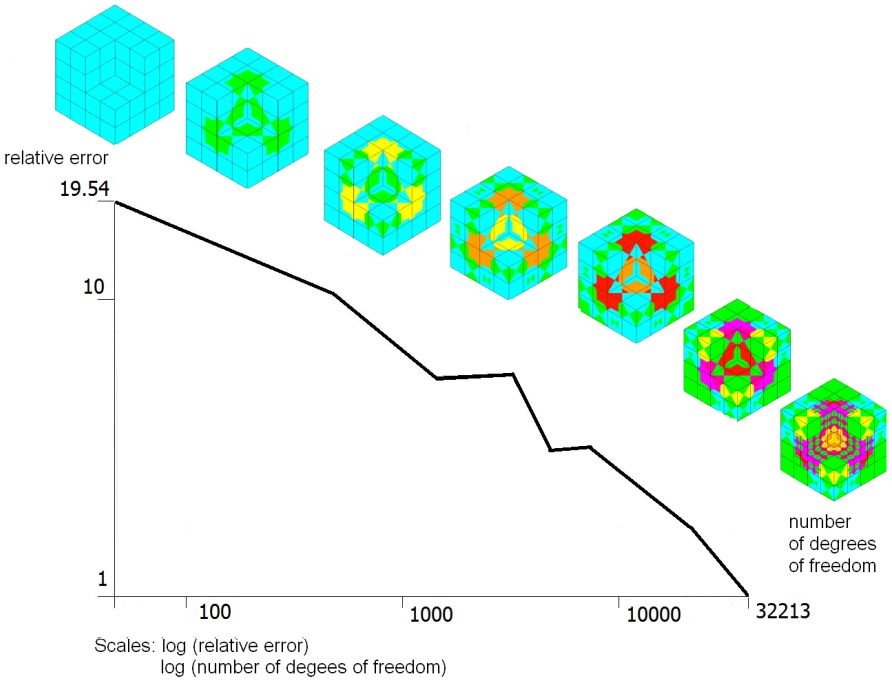

Fig. 6. Exponential convergence of the numerical error with respect to the mesh size for the model Fichera problem, obtained on the generated sequence of coarse grids. The corresponding fine grids are not presented here.

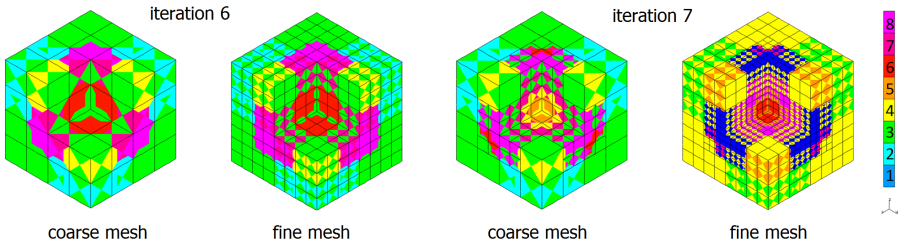

Fig. 7. Coarse and fine meshes of *hp*-FEM code for the Fichera problem. Various polynomial orders of approximation on element edges, faces and interiors are denoted by different colors.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

Table 1. Comparison of flops and execution times between bisection-weighted-by-element-size-and-order, with MUMPS equipped with automatic generation of ordering on different three-dimensional adaptive grids.

N	Weighted bisections flops	MUMPS flops	Ratio flops	Weighted bisections time[s]	MUMPS time[s]	Ratio time[s]
3,958	$119*10^6$	$140*10^6$	1.17	2.7s	4.52s	1.67
32,213	$4{,}797*10^6$	$9{,}469*10^6$	1.90	36.02s	43.21s	1.19
94,221	$56*10^9$	$111*10^9$	1.97	14.49s	28.29s	1.95
139,425	$132*10^9$	$254*10^9$	1.92	33.06s	67.94s	2.05

as the H1 norm difference between the coarse and the fine mesh solutions is less than 1.0 percent. In particular, on the last iteration for the Fichera problem (N=139,425) MUMPS with its default orderings used 67.94 seconds while with our ordering it used 33.06 seconds. The number of floating point operations required to perform the factorizations was $254 * 10^9$ as reported by the MUMPS with automatic ordering, and $111 * 10^9$ as reported by the MUMPS with our ordering. We can conclude that the bisections-weighted-by-element-size-and-order is an attractive alternative algorithm for generation of the ordering based on the element partition trees.

5 Conclusions

We introduce a heuristic algorithm called *bisections-weighted-by-element-size-and-order* that utilizes a top-down approach to construct element partition trees. We compare the trees generated by our algorithm against the alternative state-of-the-art ordering algorithms, on a three-dimensional *hp*-refined grids used to solve the model Fichera problem. We conclude that our ordering algorithm can deliver up to 50% improvement against the state-of-the-art orderings used by MUMPS both in floating-point operations counts as well as wall time.

References

1. AbouEisha H, Calo VM, Jopek K, Moshkov M, Paszyńska A, Paszyński M, Skotniczny M (2017) Element partition trees for two- and three-dimensional *h*-refined meshes and their use to optimize direct solver performance. Dynamic programming, Int J Appl Math Comput Sci (accepted)
2. Amestoy PR, Duff IS (2000) Multifrontal parallel distributed symmetric and unsymmetric solvers, Comput Methods Appl Mech Eng 184:501–520. doi:10.1016/S0045-7825(99)00242-X
3. Amestoy PR, Duff IS, Koster J, L'Excellent J-Y (2001) A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J Matrix Anal Appl, 1(23):15–041. doi:10.1137/S0895479899358194
4. Amestoy PR, Guermouche A, L'Excellent J-Y, Pralet S (2011) Hybrid scheduling for the parallel solution of linear systems, Comput Methods Appl Mech Eng 2(32):136–156. doi:10.1016/j.parco.2005.07.004

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

5. Amestoy PR, Davis TA, Du IS (1996) An Approximate Minimum Degree Ordering Algorithm, SIAM J Matrix Anal Appl 17(4):886–905. doi:10.1137/S0895479894278952
6. Babuśka I, Rheinboldt WC (1978) Error estimates for adaptive finite element computations, SIAM J Num Anal 15:736–754. doi:10.1137/0715049.
7. Babuska I, Guo BQ (1992) The h, p and hp version of the finite element method: basis theory and applications, Adv Eng Softw 15(3-4): 159-174. doi:10.1016/0965-9978(92)90097-Y
8. Becker R, Kapp J, Rannacher R (2000) Adaptive finite element methods for optimal control of partial differential equations: Basic concept, SIAM J. Control Optim, 39:113–132. doi:10.1137/S0363012999351097.
9. Demkowicz L, Kurtz J, Pardo D, Paszyński M, Rachowicz W, Zdunek A (2007) Computing with hp Adaptive Finite Element Method. Part II. Frontiers: Three Dimensional Elliptic and Maxwell Problems with Applications, Chapmann & Hall, CRC Press, Boca Raton, London, New York
10. Demkowicz L, Pardo D, Rachowicz W (2006) Fully automatic hp-adaptivity in three-dimensions, Comput Methods Appl Mech Eng 196(37-40):4816–4842. doi:10.1023/A:1015192312705
11. Duff IS, Erisman AM, Reid JK (1986) Direct Methods for Sparse Matrices, Oxford University Press, Inc., New York
12. Duff IS, Reid JK (1983) The multifrontal solution of indefinite sparse symmetric linear, ACM Trans Math Softw, 9(3):302–325. doi:10.1145/356044.356047
13. Duff IS, Reid K (1984) The multifrontal solution of unsymmetric sets of linear systems, SIAM J Sci Comput 5:633–641. doi:10.1137/0905045
14. Fiałko S (2009) A block sparse shared-memory multifrontal finite element solver for problems of structural mechanics, Computer Assisted Mechanics and Engineering Science 16:117–131.
15. Fiałko S (2009) The block subtracture multifrontal method for solution of large finite element equation sets, Technical Transactions 1-NP, 8:175–188.
16. Fiałko S (2010) PARFES: A method for solving finite element linear equations on multi-core computers, Adv Eng Softw 40(12):1256–1265. doi:10.1016/j.advengsoft.2010.09.002
17. George A, Lu JW-H (1978) An automatic nested dissection algorithm for irregular finite element problems, SIAM J Num Anal 15:1053–1069. doi:10.1137/0715069
18. Gilbert JR, Tarjan RE (1986/87) The analysis of a nested dissection algorithm, Numer Math 50(4):377–404. doi:10.1007/BF01396660
19. Hughes TJR (1987) The Finite Element Method. Linear Statics and Dynamics Finite Element Analysis, Prentice-Hall, Englewood Cliffs, New-York.
20. Karypis G, Kumar V (1998) A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J Sci Comp 20(1):359–392. doi:10.1137/S1064827595287997
21. Melenk JM (2002) hp-Finite Element Methods for Singular Perturbations, Springer-Verlag, Berlin, Heidelberg.
22. Niemi A, Babuśka I, Pitkaranta J, Demkowicz L (2012) Finite element analysis of the Girkmann problem using the modern hp-version and the classical h-version, Eng Comput 28:123–134. doi:10.1007/s00366-011-0223-0
23. Paszyńska A (2014) Volume and neighbors algorithm for finding elimination trees for three dimensional h-adaptive grids, Comput Math Appl 68(10):1467–1478. doi:10.1016/j.camwa.2014.09.012

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60

24. Paszyńska A, Paszyński M, Jopek K, Woźniak M, Goik D, Gurgul P, AbouEisha H, Moshkov M, Calo VM, Lenharth A, Nguyen D, Pingali K (2015) Quasi-Optimal Elimination Trees for 2D Grids with Singularities, Scientific Programming, Volume 2015 Article ID 303024, 1-18. doi:10.1155/2015/303024
25. Paszyński M (2016) Fast solvers for mesh-based computations, Taylor and Francis, CRC Press, Boca Raton, London, New York
26. Schwab, C (1998) *p* and *hp* Finite Element Methods: Theory and Applications in Solid and Fluid Mechanics, Clarendon Press, Oxford
27. Solin P, Segeth K, Dolezel I (2003) Higher-Order Finite Element Methods, Chapman & Hall/CRC Press, Boca Raton, London, New York
28. Szymczak A, Paszyńska A, Paszyński M, Pardo D (2013) Preventing deadlock during anisotropic 2D mesh adaptation in hp-adaptive FEM, J Comput Sci, 4(3):170–179. doi:10.1016/j.jocs.2011.09.001
29. Yannakakis M (1981) Computing the minimum fill-in is NP-complete, SIAM J Alg Disc Meth 2:77–79. doi:doi.org/10.1137/0602010

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_60