Int. J. Appl. Math. Comput. Sci., 2017, Vol. 27, No. 2, 351–365
DOI: 10.1515/amcs-2017-0025

amcs

ELEMENT PARTITION TREES FOR H-REFINED MESHES TO OPTIMIZE DIRECT SOLVER PERFORMANCE. PART I: DYNAMIC PROGRAMMING

HASSAN ABOUEISHA[a], VICTOR MANUEL CALO[b], KONRAD JOPEK[c], MIKHAIL MOSHKOV[a], ANNA PASZYŃSKA[d], MACIEJ PASZYŃSKI[c,*], MARCIN SKOTNICZNY[c]

[a] Computer, Electrical and Mathematical Sciences and Engineering (CEMSE)
King Abdullah University of Science and Technology, Bld. 1 (Al-khawarizmi)
4128-WS03, Thuwal, 23955-6900, Kingdom of Saudi Arabia
e-mail: {hassan.aboueisha,mikhail.moshkov}@kaust.edu.sa

[b] Faculty of Science and Engineering, Western Australian School of Mines
Curtin University, Kent Street, Perth, WA 6102, Australia
e-mail: vmcalo@gmail.com

[c] Faculty of Computer Science, Electronics and Telecommunications
AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: macwozni@agh.edu.pl

[d] Faculty of Physics, Astronomy and Applied Computer Science
Jagiellonian University, ul. Łojasiewicza 11, 30-348 Kraków, Poland
e-mail: anna.paszynska@uj.edu.pl

We consider a class of two- and three-dimensional h-refined meshes generated by an adaptive finite element method. We introduce an element partition tree, which controls the execution of the multi-frontal solver algorithm over these refined grids. We propose and study algorithms with polynomial computational cost for the optimization of these element partition trees. The trees provide an *ordering* for the elimination of unknowns. The algorithms automatically optimize the element partition trees using extensions of dynamic programming. The construction of the trees by the dynamic programming approach is expensive. These generated trees cannot be used in practice, but rather utilized as a learning tool to propose fast heuristic algorithms. In this first part of our paper we focus on the dynamic programming approach, and draw a sketch of the heuristic algorithm. The second part will be devoted to a more detailed analysis of the heuristic algorithm extended for the case of hp-adaptive grids.

Keywords: h-adaptive finite element method, ordering, element partition tree, extensions of dynamic programming, multi-frontal direct solvers.

1. Introduction

The finite element method (FEM) is a popular approach (Hughues, 2000; Zienkiewicz *et al.*, 2013; Demkowicz, 2006) to approximate solutions of partial differential equations (PDEs). It has multiple applications in two and three-dimensions (Barboteu *et al.*, 2013; Strug *et al.*, 2013; Karczewska *et al.*, 2016). The discrete approximation of the solution of a PDE using FEM uses a computational mesh to describe the geometry of the domain and approximating functions to the solution. This discrete approximation results in a global system of linear equations. Using the particular weak form resulting from the integration by parts of the weighted residual, each finite element generates a frontal matrix. The algebraic system is spread over these frontal matrices. The entries in these matrices correspond to the action of the weak form on the basis functions associated with element vertices, edges, and interiors. The global matrix is obtained by merging these frontal matrices according to the mesh topology.

The multi-frontal solver is a state-of-the-art algorithm for solving sparse linear systems resulting

*Corresponding author

from finite element discretizations (Duff *et al.*, 1986; Duff and Reid, 1983; 1984). This algorithm works in both single- and multi-processor environments, either in the shared-memory (Fiałko, 2009a; 2009b; 2010) or distributed-memory (Amestoy *et al.*, 2000; 2001; 2006), parallel machines.

Classical multi-frontal solvers, such as MUMPS (Amestoy *et al.*, 2000; 2001; 2006) use an ordering constructed based on the analysis of the sparsity of the global matrix (Liu, 1990). We build an element partition tree based on the computational mesh, and construct our ordering based on this element partition tree.

We focus on classes of two-dimensional h-refined meshes, obtained through an iterative adaptive process, where some finite elements are selected for h refinement, and thus broken into smaller finite elements. Algorithms performing h refinements in the areas of local point or edge singularities are essential to solving a variety of engineering problems (Bao *et al.*, 2012; Kardani *et al.*, 2012; Niemi *et al.*, 2012; Patro *et al.*, 2013). The refinement process is controlled by one of many available adaptive algorithms (Babuška and Rheinboldt, 1978; Becker *et al.*, 2000; Belytschko and Tabbar, 1993; Errikson and Johnson, 1991), and the goal of the process is to increase the accuracy of the numerical solution by adding a limited number of new basis functions associated with new elements generated in the areas of a large numerical error.

A refined 2D mesh with rectangular elements can be considered a finite set of vertical and horizontal line segments, vertices (intersections of line segments) and rectangles bounded by these line segments. An element partition tree is a binary rooted tree that describes the partition of the mesh by these line segments.

We present a dynamic programming algorithm for the optimization of element partition trees for a class of 2D meshes. The goal of the optimization is to find element partition trees that results in an ordering providing a minimal number of floating point operations of the multi-frontal solver algorithm. Cost functions for the element partition trees are defined based on the mesh topological information. The optimization problem we consider lies at the intersection of discrete geometry and combinatorial optimization.

The construction of the element partition trees by the dynamic programming approach is expensive. These generated optimal trees cannot be used in practice, but rather guide our heuristic methods. In this paper we describe our optimization procedure and present a class of element partition trees obtained from our optimization procedure. From the analysis of the automatic optimization results, we outline a heuristic algorithm to construct element partition trees, which is based on multilevel recursive bisections. The second part of our paper (AbouEisha *et al.*, 2016) will be devoted to a more detailed analysis of the heuristic algorithm extended for the case of hp-adaptive grids.

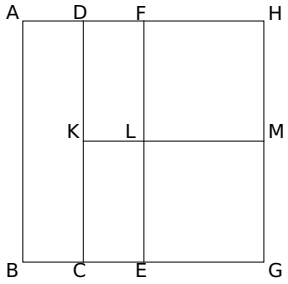

Fig. 1. Example mesh.

The plan of the paper is the following. First, we define the class of meshes suitable for optimization. Second, we define partitions of the mesh. Third, we define our notion of an element partition tree. Next, we define our cost function related to the optimization of floating point operations. Later, we describe our optimization algorithms and present their computational cost. Finally, we discuss the results of our dynamic programming procedure for optimization of element partition trees.

2. Dynamic programming algorithm for optimization of element partition trees

2.1. Class of meshes. The class of *meshes* investigated is constructed as follows. We start with a rectangle. We call the sides of this initial rectangle *boundary sides* and its vertices are called *corners*. Further vertical and horizontal straight line segments (we refer to these segments including sides of the initial rectangle as *lines*) may be added as follows. We select two points that lie on different existing parallel lines and which can be connected by a vertical or horizontal line and connect them by adding a line. The process may be repeated until the desired mesh structure is obtained. Once the construction process is finished, we have a mesh that is mainly a rectangle with a set of vertical and horizontal lines that lies inside it.

The following example illustrates the description of an instance of this class of meshes presented in Fig. 1. Initially, the rectangle boundary $ABGH$ is specified. The order of points is assumed to follow a counter-clockwise order, and the selection of the starting point is arbitrary. Then, lines CD and EF are added where their end points belong to the horizontal sides of the mesh boundary (AH and BG). Finally, line LK is drawn between lines CD and EF and KM are drawn to connect points K and M.

As a result of this construction process, we obtain a set of points that are either vertices of the initial rectangle, endpoints of the lines added or points that result from the intersection of the lines added. We call this set of points *mesh points*. Any segment of a straight horizontal or

vertical line in the mesh that connects two different mesh points is called a *mesh line*. A *boundary line* is a mesh line which belongs to a boundary side. For example, EK, KF, and LM are mesh lines, and CE is a boundary line (see Fig. 1).

We define *dividing lines* that are used to partition a given mesh. We denote the set of vertical lines that extend from the borders of a mesh M by $DL_V(M)$, horizontal lines that extend between the borders of M by $DL_H(M)$ and the union of both sets by $DL(M)$. We do not consider vertical border sides of the mesh M among $DL_V(M)$ and similarly for its horizontal border sides. The mesh M can be partitioned using a dividing line l that belongs to the set $DL(M)$. This partitioning step results in two sub meshes: $M(l,0)$ that represents the submesh which lies below the horizontal (left of the vertical) line l and $M(l,1)$ denotes the submesh which is above the horizontal (right of the vertical) line l. In Fig. 1, $DL_V(ABGH) = \{CD, EF\}$, $DL_H(ABGH) = \emptyset$ and $DL(ABGH) = \{CD, EF\}$. The mesh $ABGH$ can be partitioned using the dividing line EF resulting in the two submeshes: $ABEF = ABGH(EF,0)$ and $EGHF = ABGH(EF,1)$.

We describe an arbitrary *submesh* N of M by a sequence of binary partitioning steps. Formally, a submesh N of a mesh M is an expression of the kind

$$N = M(l_1,\delta_1)\dots(l_n,\delta_n),$$

where $\delta_1,\dots,\delta_n \in \{0,1\}$, $l_1 \in DL(M)$ and $l_i \in DL(M(l_1,\delta_1)\dots(l_{i-1},\delta_{i-1}))$ for $i = 2,\dots,n$.

The resulting submesh is described as follows. First, a dividing line l_1 is used to partition the mesh M and then the submesh $M(l_1,\delta_1)$ is acquired. The line l_2 is a dividing line of this submeshh ($l_2 \in DL(M(l_1,\delta_1))$) which is used to partition $M(l_1,\delta_1)$ again until the desired submesh is obtained, etc. For example, $KMHF = ABGH(EF,1)(KM,1)$. We denote by SUB(M) the set of submeshes of the mesh M including the mesh M. In our example, SUB$(ABGH) = \{ABGH,\ ABCD,\ ABEF,\ DCGH,\ DCEF,\ FEGH,\ DLKF,\ LCEK,\ KEGM,\ FKMH,\ DLMH,\ LCGM\}$.

The submesh N can be considered a mesh that is represented as a rectangle, whose sides are called *border sides*, that may contain a set of vertical and horizontal lines inside it. For the submesh N, we can define the notion of dividing lines in a similar way as for M, so we can use the notation $DL_V(N)$, $DL_H(N)$ and $DL(N)$. As for the initial mesh M, it is possible that a dividing line is constructed indirectly in a series of steps. For example, $DL_V(DLMH) = \{KF\}$ and $DL_H(DCGH) = \{LM\}$ (see Fig. 1).

A submesh N of the mesh M is a *unitary submesh* if and only if it does not have any dividing lines, i.e., $DL(N) = \emptyset$. We assume that each unitary submesh N of M has a unique *identifier* $\varphi(N)$. Unitary submeshes correspond to single finite elements.

2.2. Mesh and line partitions. Let M be a mesh. For each mesh line l of M, we describe a set $P(M,l)$ of *distinct points of the line* l. If l is a boundary line, then the set of points includes the endpoints of l and any point of l that results from any other mesh line touching l. If l is not a boundary line, then the set of points includes all the points of l that result from another mesh line cutting l in addition to the endpoints of l. End points of other lines that start or finish on l are not included. That is, a touching line does not generate a point for l. For example, $P(ABGH,AH) = \{A,D,F,H\}$, $P(ABGH,CD) = \{C,D\}$, and $P(ABGH,LM) = \{L,K,M\}$. Each pair of consecutive points in $P(M,l)$ forms an *edge*. The number of edges on a line l is denoted by $E(M,l)$ and $E(M,l) = |P(M,l)| - 1$. For example, $E(ABGH,CD) = 1$ for the mesh in Fig. 1.

Let N be a submesh of M and $l \in DL(N)$. The line l represents a *common border side* of two submeshes: $N(l,0)$ and $N(l,1)$. We define now a number of parameters of the submesh N and the dividing line l for N:

- $EP(M,N,l)$ represents the number of endpoints of l that lie on a boundary side.
- $B(M,N)$ is the number of edges on the border sides of N. These edges result from the mesh lines that cut the border sides of N or touch them if they are on boundary sides, i.e., border sides of the initial mesh M, as described in the previous paragraph.
- $BE(M,N)$ is the number of edges that lie in the border sides of N which are on the boundary sides, where $0 \le BE(M,N) \le B(M,N)$.
- $BV(M,N)$ represents the number of vertices of N that are endpoints of boundary sides, i.e., corners, where $0 \le BV(M,N) \le 4$.

In Fig. 1, the corresponding values are

$$\begin{aligned} EP(ABGH,DCEF,LK) &= 0,\\ EP(ABGH,ABEF,CD) &= 2,\\ B(ABGH,CEFD) &= 5,\\ BE(ABGH,ABCD) &= 3,\\ BV(ABGH,ABCD) &= 2. \end{aligned}$$

We denote by $s(M)$ the number of vertical and horizontal straight lines for each of which there exists a mesh line of M which is a segment of the straight line considered. The number $s(M)$ is the *size* of the mesh M.

Lemma 1. *For any mesh* M, $|\text{SUB}(M)| \le s(M)^4$.

Proof. Each submesh of M is defined by its four border sides, i.e., by four straight lines. The number of possible straight lines is equal to the size $s(M)$ of the mesh M. Therefore $|\text{SUB}(M)| \le s(M)^4$. ∎

2.3. Element partition trees. An element partition tree is a labeled finite directed tree with a root. We define the notion of an *element partition tree* for a submesh N of the mesh M by induction. Let N be a unitary mesh. Then there exists only one element partition tree for N that contains exactly one node which is labeled with $\varphi(N)$. We denote this element partition tree by $etree(\varphi(N))$. Let N be a non-unitary mesh. Then any element partition tree for N can be represented in the form $etree(l, \Gamma_0, \Gamma_1)$ where $l \in DL(N)$, Γ_δ is an element partition tree for the submesh $N(l, \delta)$, $\delta \in \{0, 1\}$, and $etree(l, \Gamma_0, \Gamma_1)$ is a tree in which the root is labeled with l, and two edges start from the root which are labeled with 0 and 1 and enter the roots of element partition trees Γ_0 or Γ_1, respectively. We denote by $ET(M, N)$ the set of all element partition trees for the submesh N of the mesh M.

Let Γ be an element partition tree for the submesh N of the mesh M. Any terminal node (leaf) of this tree is labeled as a unitary submesh of N. Any internal node is labeled with a line. Each internal node has exactly two edges that start from it and are labeled with 0 and 1, respectively. Figure 2 shows an element partition tree for the mesh presented in Fig. 1. The corresponding recursive notation for the tree is $etree(EF, etree(CD, etree(\varphi(ABCD)), etree(LK, etree(\varphi(CEKL)), etree(\varphi(LKFD)))), etree(KM, etree(\varphi(KEGM)), etree(\varphi(FKMH))))$.

We associate to each node v of the element partition tree Γ a submesh $N_\Gamma(v)$ of the submesh N. If v is the root of Γ then $N_\Gamma(v) = N$. If v is not the root and the path from the root to v consists of nodes labeled with lines $l_1, \ldots, l_m$ and edges labeled with the numbers $\delta_1, \ldots, \delta_m$, respectively, then $N_\Gamma(v) = N(l_1, \delta_1) \ldots (l_m, \delta_m)$.

For each internal node v of Γ, this node is labeled with a line from $DL(N_\Gamma(v))$. For each terminal node v, the submesh $N_\Gamma(v)$ is a unitary mesh and v is labeled with the identifier $\varphi(N_\Gamma(v))$ of $N_\Gamma(v)$.

For a node v of Γ, we denote by $\Gamma(v)$ the subtree of Γ with the root in v. Thus, $\Gamma(v)$ is an element partition tree for the submesh $N_\Gamma(v)$.

Let M be a mesh and N be a non-unitary submesh of M. For each $l \in DL(N)$, let $ET(M, N, l) = \{etree(l, \Gamma_0, \Gamma_1) : \Gamma_\delta \in ET(M, N(l, \delta)), \delta = 0, 1\}$. The next statement follows immediately from the definition of element partition tree.

Proposition 1. *Let M be a mesh and N be a submeshh of M. Then $ET(M, N) = \{etree(\varphi(N))\}$ if N is unitary, and $ET(M, N) = \bigcup_{l \in DL(N)} ET(M, N, l)$ if N is non-unitary.*

2.4. Cost function for element partition trees. In this section, we define the notion of a cost function for element partition trees.

Cost function ψ has values from the set $\mathbb{R}$ of real numbers and is defined on triples (M, N, Γ) where M is a mesh, N is a submesh of M, and Γ is an element partition tree for N. The function ψ is given by two operators ψ^0 and F. The value of ψ is defined by induction. The operator ψ^0 represents the cost of processing a unitary submesh, and the operator F represents the cost of recursive merging of the submeshes.

If N is a unitary submesh of M and $\Gamma = etree(\varphi(N))$, then $\psi(M, N, \Gamma) = \psi^0(M, N)$, where ψ^0 is a real-valued function which is defined on pairs (M, N), M is a mesh and N is a unitary submesh of M.

Let N be a non-unitary mesh and $\Gamma = etree(l, \Gamma_0, \Gamma_1)$, where we have $l \in DL(N)$, $\Gamma_\delta \in ET(M, N(l, \delta))$, $\delta \in \{0, 1\}$. Then

$$\psi(M, N, \Gamma) = F(M, N, l, \psi(M, N(l, 0), \Gamma_0), \psi(M, N(l, 1), \Gamma_1)),$$

where F is a real-valued function which is defined on quintuples (M, N, l, x_1, x_2) such that M is a mesh, N is a non-unitary submesh of M, $l \in DL(N)$, $x_1, x_2 \in \mathbb{R}$.

Using this inductive definition, we can compute the cost of given element partition tree beginning from terminal nodes and terminating in the root.

The cost function ψ is called *increasing* if, for any mesh M, its non-unitary submesh N, line $l \in DL(N)$, and $x_1, x_2, y_1, y_2 \in \mathbb{R}$, if $x_1 \leq y_1$ and $x_2 \leq y_2$ then $F(M, N, l, x_1, x_2) \leq F(M, N, l, y_1, y_2)$. The cost function ψ is called *strictly increasing* if, for any mesh M, its non-unitary submesh N, line $l \in DL(N)$, and $x_1, x_2, y_1, y_2 \in \mathbb{R}$, if $x_1 \leq y_1$, $x_2 \leq y_2$, and $(x_1, x_2) \neq (y_1, y_2)$ then $F(M, N, l, x_1, x_2) < F(M, N, l, y_1, y_2)$. Each strictly increasing cost function is an increasing cost function.

Our cost function depends on a parameter p (the polynomial order), defined as a natural number which identifies the number of basis functions spread over vertices, edges, and interiors of unitary submeshes (finite elements): one per vertex, p per edge, and p^2 per interior. Another parameter considered is q that indicates how expensive a data transfer operation is with respect to an arithmetic operation. We assume that one data transfer operation is 107 times more expensive than an arithmetic operation with respect to time. This parameter can be set to adjust the cost of data transfer operations according to the architecture considered. Note that setting the parameter q to zero neglect the cost of data transfer operations.

In our optimization procedure we assume that we have dense frontal matrices within the element partition tree. For leaf nodes this is true, since element frontal matrices form dense blocks. For higher level nodes, when we merge two frontal matrices, the fully assembled nodes to be subtracted from other nodes have non-zeros

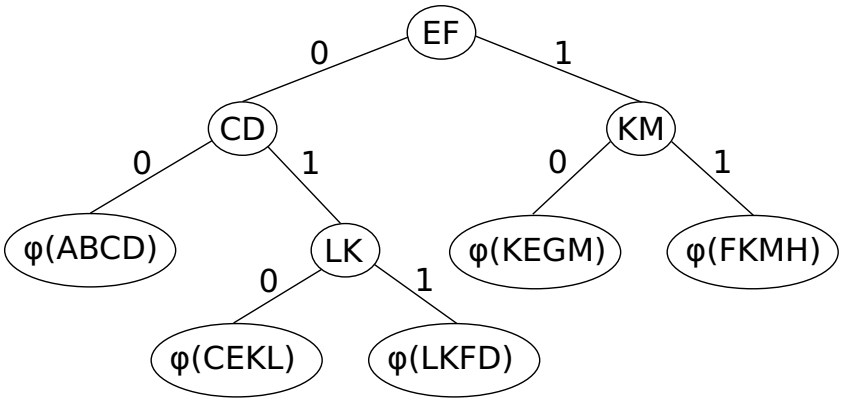

Fig. 2. Element partition tree for the mesh in Fig. 1.

in the entire row and the column, so the subtractions may make the entire matrix full. We are not able to track zeros obtained by some subtractions, or resulting from orthogonality of the basis functions, and we treat all the entries as non-zero entries. We use $W(\mu,\rho) = \sum_{i=1}^{\rho}(2\mu-2i+1)(\mu-i) = 2\mu^2\rho - 2\mu\rho^2 - \mu\rho + \frac{2}{3}\rho^3 + \frac{1}{2}\rho^2 - \frac{1}{6}\rho$ as the number of arithmetic floating point operations for partial factorization of a $\mu\times\mu$ dense matrix with ρ fully assembled rows.

Let N be a unitary submesh of M, $\Gamma = etree(\varphi(N))$, and A' be an $n\times n$ matrix with r fully assembled rows corresponding to N. Then, during the processing of the submesh N, the algorithm makes $W(n,r)$ arithmetic floating point operations for partial factorization of A'. We send $(n-r)^2$ numbers corresponding to the Schur complement to the parent node if $N\neq M$ and keep the n^2 numbers which result from factorization. Therefore $\psi(M,N,\Gamma) = \sigma q(n-r)^2 + W(n,r)$ Here $\sigma = 1$ if $N\neq M$ and $\sigma = 0$ otherwise since all the nodes of the element partition tree except the root node must communicate Schur complement data to the parent node. In summary, $\psi^0(M,N) = \sigma q(n-r)^2 + W(n,r)$.

For a unitary submesh N, the matrix A' contains $n = p^2+4p+4$ rows corresponding to base functions spread over four vertices, four edges, and the interior of N.

Let N be a non-unitary submesh, $\Gamma = etree(l,\Gamma_0,\Gamma_1)$, where $l\in DL(N)$ and $\Gamma_\delta \in ET(M,N(l,\delta))$, $\delta\in\{0,1\}$, and A'' be an $m\times m$ matrix with t fully assembled rows corresponding to N. Then, during the processing of the submesh N, the algorithm makes $W(m,t)$ arithmetic floating point operations during partial factorization of A'', it should send $(m-t)^2$ numbers corresponding to the Schur complement to the parent node if $N\neq M$ and store m^2 numbers to keep the factorization.

Therefore

$$\begin{aligned}&\psi(M,N,\Gamma)\\ &\quad = \psi(M,N(l,0),\Gamma_0) + \psi(M,N(l,1),\Gamma_1)\\ &\qquad + \sigma q(m-t)^2 + W(m,t).\end{aligned}$$

Here $\sigma = 1$ if $N\neq M$ and $\sigma = 0$ otherwise.

The matrix A'' contains only rows corresponding to basis functions spread over edges and vertices that belong to the border of N or to the line l. The fully assembled rows of A'' correspond to basis functions spread over edges and some vertices that belong to the line l.

2.5. Single-processor time cost function. We characterize the time complexity of element partition trees in the case when we perform computations on a single processor. First, we present common formulas with parameters for operators ψ^0 and F. Then we specify the differentiating parameters.

Let M be a mesh and N be a submesh of this mesh. If N is a unitary mesh then

$$\psi^0(M,N) = \sigma q(n-r)^2 + W(n,r). \tag{1}$$

Let N be a non-unitary submesh, $l\in DL(N)$, and $x_1,x_2\in\mathbb{R}$. Then

$$\begin{aligned}F(M,N,l,x_1,x_2) = x_1 + x_2 + \sigma q(m-t)^2\\ + W(m,t).\end{aligned} \tag{2}$$

As above, $\sigma = 1$ if $N\neq M$ and $\sigma = 0$ if $N = M$.

The operator ψ^0 is given by formula (1) with $r = p^2 + BE(M,N)p + BV(M,N)$ and $n = p^2+4p+4$, and the operator F is given by formula (2) with $t =$

$E(M,l)(p+1)+EP(M,N,l)-1$ and

$$m=\begin{cases}(B(M,N)-BE(M,N)\\ \quad +E(M,l))(p+1)+EP(M,N,l)+\gamma\\ \qquad \text{if} \quad 1\le BE(M,N)<B(M,N),\\ (B(M,N)-BE(M,N)\\ \quad +E(M,l))(p+1)+EP(M,N,l)-1\\ \qquad \text{otherwise,}\end{cases} \tag{3}$$

where $\gamma\in\{0,1\}$ and $\gamma=1$ if and only if N has exactly two boundary sides which are parallel, e.g., submesh $CEFD$ in Fig. 1.

From (2) it follows that the cost function considered is strictly increasing.

2.6. Optimization of element partition trees. In this section, we consider three algorithms: (i) to construct a directed acyclic graph used for the description and optimization of element partition trees, (ii) to count the number of all or only optimal element partition trees, and (iii) to optimize the element partition trees.

2.6.1. Directed acyclic graph $\Delta(M)$. Let M be a mesh. We now describe Algorithm 1 for the construction of a directed acyclic graph $\Delta(M)$ which is used to describe and optimize the element partition trees. The set of nodes of this graph coincides with the set SUB(M) of submeshes of the mesh M. During each iteration (with the exception of the last one) the algorithm processes one node. It starts with the graph that consists of one node M which is not processed and finishes when all nodes of the constructed graph are processed.

Proposition 2. *Algorithm 1 has polynomial time complexity with respect to the size of the input mesh.*

Proof. The number of iterations of Algorithm 1 (each iteration includes Step 2 and Step 3 or 4) is at most $|\text{SUB}(M)|+1$. By Lemma 1, $|\text{SUB}(M)|\le s(M)^4$. One can show that the time complexity of each iteration is polynomial with respect to the size of M. Therefore, Algorithm 1 has polynomial time complexity with respect to the size of the input mesh. ■

A node of a directed graph is called *terminal* if there are no edges starting in this node, and *internal* otherwise. A node N of the graph $\Delta(M)$ is terminal if and only if N is a unitary mesh. A *proper subgraph* of the graph $\Delta(M)$ is a graph G obtained from $\Delta(M)$ by removal of some l-pairs of edges such that each internal node of $\Delta(M)$ keeps at least one l-pair of edges starting from this node. By definition, $\Delta(M)$ is a proper subgraph of $\Delta(M)$. Proper subgraphs of the graph $\Delta(M)$ arise as results of the tree optimization procedure applied to $\Delta(M)$ or to its proper subgraphs.

Algorithm 1. Construction of a directed acyclic graph $\Delta(M)$.

Require: Mesh M.
1: Construct the graph that consists of one node M which is not marked as processed.
2: **if** All nodes of the graph are processed **then**
3: **return** the resulting graph as $\Delta(M)$
4: **end if**
5: Choose a node (submesh) N that has not been processed yet
6: **if** N is a unitary mesh **then**
7: Mark N as processed
8: Go to Step 2.
9: **end if**
10: **if** N is non-unitary **then**
11: **for all** $l\in DL(N)$ **do**
12: Draw two edges from the node N (this pair of edges is called *l-pair*)
13: Label these edges with pairs $(l,0)$ and $(l,1)$
14: These edges enter nodes $N(l,0)$ and $N(l,1)$, respectively.
15: **end for**
16: **if** Some of the nodes $N(l,0),N(l,1)$ are not present in the graph **then**
17: Add these nodes to the graph.
18: **end if**
19: Mark the node N as processed
20: Go to Step 2.
21: **end if**

Let G be a proper subgraph of the graph $\Delta(M)$. For each internal node N of the graph G, we denote by $DL_G(N)$ the set of lines l from $DL(N)$ such that an l-pair of edges starts from N in G. For each node N of the graph G, we define the set $Etree(G,N)$ of element partition trees in the following way. If N is a terminal node of G (in this case N is unitary), then $Etree(G,N)=\{etree(\varphi(N))\}$. Let N be a internal node of G (in this case N is non-unitary) and $l\in DL_G(N)$. We write $Etree(G,N,l)=\{etree(l,\Gamma_0,\Gamma_1):\Gamma_\delta\in Etree(G,N(l,\delta)),\delta=0,1\}$. Then $Etree(G,N)=\bigcup_{l\in DL_G(N)}Etree(G,N,l)$.

Proposition 3. *Let M be a mesh. Then*

$$Etree(\Delta(M),N)=ET(M,N)$$

for any node N of the graph $\Delta(M)$.

Proof. We prove this statement by induction on nodes of $\Delta(M)$. Let N be a terminal node of $\Delta(M)$. Then $Etree(\Delta(M),N)=\{etree(\varphi(N))\}=ET(M,N)$. Let now N be an internal node of $\Delta(M)$, and let us assume that $Etree(\Delta(M),N(l,\delta))=ET(M,N(l,\delta))$ for any $l\in DL(N)$ and $\delta\in\{0,1\}$. Then, for any $l\in DL(N)$,

we have $Etree(\Delta(M), N, l) = ET(M, N, l)$, where $DL_{\Delta(M)}(N) = DL(N)$. Using Proposition 1, we obtain $Etree(\Delta(M), N) = ET(M, N)$. ■

In consequence, the set of element partition trees for a mesh M is equal to the set $Etree(\Delta(M), M)$. We show below that the set of strictly optimal element partition trees for a mesh M relative to an increasing cost function ψ can be represented in the form $Etree(G, M)$, where G is a proper subgraph of the graph $\Delta(M)$. In the next section, we describe an algorithm which counts the cardinality of the set $Etree(G, M)$ for a proper subgraph G of the graph $\Delta(M)$.

2.6.2. Cardinality of the set $Etree(G, M)$. Let M be a mesh, and G be a proper subgraph of the graph $\Delta(M)$. We describe an algorithm which counts, for each node N of the graph G, the cardinality $C(N)$ of the set $Etree(G, N)$, and returns the number $C(M) = |Etree(G, M)|$.

Algorithm 2. Count the cardinality $C(N)$.

Require: A proper subgraph G of the graph $\Delta(M)$ for some mesh M.

1: **if** All nodes of the graph G are processed **then**
2: **return** $C(M)$
3: **end if**
4: Choose a node N which is not processed yet and which is either a terminal node of G or an internal node of G such that all nodes $N(l, \delta)$ are processed for each $l \in DL_G(N)$ and $\delta \in \{0, 1\}$.
5: **if** N is a unitary mesh **then**
6: Set $C(N) = 1$
7: Mark N as processed
8: Go to Step 1.
9: **end if**
10: **if** N is non-unitary **then**
11: Set $C(N) = \sum_{l \in DL_G(N)} C(N(l, 0)) \times C(N(l, 1))$
12: Mark the node N as processed
13: Go to Step 1.
14: **end if**

Proposition 4. *Let M be a mesh, and G be a proper subgraph of the graph $\Delta(M)$. Then Algorithm 2 returns the cardinality of the set $Etree(G, M)$ and makes at most $2s(M)^5$ operations of multiplication and addition.*

Proof. We prove by induction on the nodes of the graph G that $C(N) = |Etree(G, N)|$ for each node N of G. Let N be a terminal node of G. Then $Etree(G, N) = \{etree(\varphi(N))\}$ and $|Etree(G, N)| = 1$. Therefore the statement considered holds for N. Let now N be an internal node of G such that the statement considered holds for its children. We know that $Etree(G, N) = \bigcup_{l \in DL_G(N)} Etree(G, N, l)$, where, for $l \in DL_G(N)$,

$$Etree(G, N, l) = \{etree(l, \Gamma_0, \Gamma_1) : \Gamma_\delta \in Etree(G, N(l, \delta)), \delta = 0, 1\}.$$

Then, for any $l \in DL_G(N)$,

$$|Etree(G, N, l)| = |Etree(G, N(l, 0))| \times |Etree(G, N(l, 1))|,$$

and $|Etree(G, N)| = \sum_{l \in DL_G(N)} |Etree(G, N, l)|$. By the inductive hypothesis,

$$|Etree(G, N(l, \delta))| = C(N(l, \delta))$$

for each $l \in DL_G(N)$ and $\delta \in \{0, 1\}$. Therefore $C(N) = |Etree(G, N)|$. Hence the statement considered holds. Thus

$$C(M) = |Etree(G, M)|,$$

i.e., Algorithm 2 returns the cardinality of the set $Etree(G, M)$.

We now evaluate the number of arithmetic operations made by Algorithm 2. By Lemma 1, $|\mathrm{SUB}(M)| \leq s(M)^4$. Therefore the number of internal nodes in G is at most $s(M)^4$. In each internal node N of G, Algorithm 2 performs $|DL_G(N)|$ multiplications and $|DL_G(N)|-1$ additions. One can show that $|DL_G(N)| \leq s(M)$. Therefore, Algorithm 2 performs at most $2s(M)^5$ arithmetic operations. ■

2.7. Optimization procedure. Let M be a mesh, G be a proper subgraph of the graph $\Delta(M)$, and ψ be an increasing cost function for element partition trees defined by operators ψ^0 and F.

Let N be a node of G and $\Gamma \in Etree(G, N)$. One can show that, for any node v of Γ, the element partition tree $\Gamma(v)$ belongs to the set $Etree(G, N_\Gamma(v))$.

An element partition tree Γ from $Etree(G, N)$ is called an *optimal element partition tree for N relative to ψ and G* if $\psi(M, N, \Gamma) = \min\{\psi(M, N, \Gamma') : \Gamma' \in Etree(G, N)\}$.

An element partition tree Γ from $Etree(G, N)$ is called a *strictly optimal element partition tree for N relative to ψ and G* if, for any node v of Γ, the element partition tree $\Gamma(v)$ is an optimal element partition tree for $N_\Gamma(v)$ relative to ψ and G.

We denote by $Etree_\psi^{opt}(G, N)$ the set of optimal element partition trees for N relative to ψ and G. We denote by $Etree_\psi^{s-opt}(G, N)$ the set of strictly optimal element partition trees for N relative to ψ and G.

Let $\Gamma \in Etree_\psi^{opt}(G, N)$ and $\Gamma = etree(l, \Gamma_0, \Gamma_1)$. Then $\Gamma \in Etree_\psi^{s-opt}(G, N)$ if, and only if, $\Gamma_\delta \in Etree_\psi^{s-opt}(G, N(l, \delta))$ for $\delta = 0, 1$.

Proposition 5. *Let ψ be a strictly increasing cost function for element partition trees, M be a mesh, and G be a proper subgraph of the graph $\Delta(M)$. Then, for any node N of the graph G, $Etree_{\psi}^{opt}(G,N) = Etree_{\psi}^{s-opt}(G,N)$.*

Proof. It is clear that

$$Etree_{\psi}^{s-opt}(G,N) \subseteq Etree_{\psi}^{opt}(G,N).$$

Let Γ belong to $Etree_{\psi}^{opt}(G,N)$ and let us assume that $\Gamma \notin Etree_{\psi}^{s-opt}(G,N)$. Then there is a node v of Γ such that $\Gamma(v) \notin Etree_{\psi}^{opt}(G, N_{\Gamma}(v))$. Let $\Gamma_0 \in Etree_{\psi}^{opt}(G, N_{\Gamma}(v))$ and Γ' be the element partition tree obtained from Γ by the replacement of $\Gamma(v)$ with Γ_0. One can show that $\Gamma' \in Etree(G,N)$. Since ψ is strictly increasing and $\psi(M, N_{\Gamma}(v), \Gamma_0) < \psi(M, N_{\Gamma}(v), \Gamma(v))$, we have $\psi(N,\Gamma') < \psi(N,\Gamma)$. Therefore $\Gamma \notin Etree_{\psi}^{opt}(G,N)$, but this is impossible. Thus $Etree_{\psi}^{opt}(G,N) \subseteq Etree_{\psi}^{s-opt}(G,N)$. ∎

We now describe Algorithm 3 which is an *optimization procedure relative to the cost function ψ*. This algorithm attaches to each node N of G the number $c(N) = \min\{\psi(M,N,\Gamma) : \Gamma \in Etree(G,N)\}$ and, probably, removes some l-pairs of edges starting from internal nodes of G. As a result, we obtain a proper subgraph G^{ψ} of the graph G. By construction, G^{ψ} is also a proper subgraph of the graph $\Delta(M)$.

Proposition 6. *Let Algorithm 3 use a cost function ψ given by operators ψ^0 and F which have polynomial time complexity with respect to the size of the input mesh. Then Algorithm 3 has polynomial time complexity with respect to the size of the input mesh.*

Proof. By Lemma 1, $|\mathrm{SUB}(M)| \leq s(M)^4$. Therefore the number of nodes in G is at most $s(M)^4$. In each terminal node of the graph G, Algorithm 3 computes the value of ψ^0. In each internal node N of G Algorithm 3 computes $|DL_G(N)|$ times the value of F and makes $|DL_G(N)| - 1$ comparisons. One can show that $|DL_G(N)| \leq s(M)$. Therefore, Algorithm 3 makes at most $s(M)^5$ comparisons and at most $s(M)^5$ computations of operators ψ^0 and F. If ψ^0 and F have polynomial time complexity with respect to $s(M)$ then Algorithm 3 has polynomial time complexity with respect to $s(M)$. ∎

For any node N of the graph G and for any $l \in DL_G(N)$ we write $\psi_G(N) = \min\{\psi(M,N,\Gamma) : \Gamma \in Etree(G,N)\}$ and $\psi_G(N,l) = \min\{\psi(M,N,\Gamma) : Gamma \in Etree(G,N,l)\}$.

Lemma 2. *Let ψ be an increasing cost function for element partition trees, M be a mesh, and G be a proper subgraph of the graph $\Delta(M)$. Then, for any node N of the graph G and for any $l \in DL_G(N)$, Algorithm 3 computes values $c(N) = \psi_G(N)$ and $c(N,l) = \psi_G(N,l)$.*

Algorithm 3. Optimization procedure relative to the cost function ψ.

Require: A proper subgraph G of the graph $\Delta(M)$ for some mesh M, and an increasing cost function ψ for element partition trees given by operators ψ^0 and F.

1: **if** all nodes of the graph G are processed **then**
2: **return** the obtained graph as G^{ψ}
3: **end if**
4: Choose a node N which is not processed yet and which is either a terminal node of G or an internal node of G for which all its children have been processed.
5: **if** N is a unitary mesh **then**
6: set $c(N) = \psi^0(M,N)$
7: Mark N as processed
8: Go to Step 1.
9: **end if**
10: **if** N is non-unitary **then**
11: **for all** $l \in DL_G(N)$ **do**
12: Compute the value

$$c(N,l) = F(M,N,l,c(N(l,0)),c(N(l,1)))$$

13: Set $c(N) = \min\{c(N,l) : l \in DL_G(N)\}$.
14: Remove all l-pairs of edges starting from N for which $c(N) < c(N,l)$.
15: Mark the node N as processed
16: **end for**
17: Proceed to Step 1.
18: **end if**

Proof. We proceed by induction on the nodes of the graph G. Let N be a terminal node of G. Then $Etree(G,N) = \{etree(\varphi(N))\}$ and $\psi_G(N) = \psi^0(M,N)$. Therefore $c(N) = \psi_G(N)$. Let now N be an internal node of G such that the statement considered holds for each node $N(l,\delta)$ such that $l \in DL_G(N)$ and $\delta \in \{0,1\}$. We know that

$$Etree(G,N) = \bigcup_{l \in DL_G(N)} Etree(G,N,l)$$

and, for each $l \in DL_G(N)$,

$$\begin{aligned} &Etree(G,N,l) \\ &\quad = \{etree(l,\Gamma_0,\Gamma_1) : \\ &\qquad \Gamma_{\delta} \in Etree(G,N(l,\delta)), \delta = 0,1\}. \end{aligned}$$

Since ψ is an increasing cost function,

$$\psi_G(N,l) = F(M,N,l,\psi_G(N(l,0)),\psi_G(N(l,1))),$$

where $\psi_G(N) = \min\{\psi_G(N,l) : l \in DL_G(N)\}$. By the inductive hypothesis, $\psi_G(N(l,\delta)) = c(N(l,\delta))$ for

each $l \in DL_G(N)$ and $\delta \in \{0,1\}$. Therefore $c(N,l) = \psi_G(N,l)$ for each $l \in DL_G(N)$, and $c(N) = \psi_G(N)$. ■

Theorem 1. *Let ψ be an increasing cost function for element partition trees, M be a mesh, and G be a proper subgraph of the graph $\Delta(M)$. Then, for any node N of the graph G^ψ, the following equality holds: $Etree(G^\psi, N) = Etree_\psi^{s-opt}(G, N)$.*

Proof. We proceed by induction on nodes of G^ψ. We use Lemma 2 which shows that, for each node N of the graph G and for each $l \in DL_G(N)$, $c(N) = \psi_G(N)$ and $c(N,l) = \psi_G(N,l)$. Let N be a terminal node of G^ψ. Then $Etree(G^\psi, N) = \{etree(\varphi(N))\}$ with $Etree(G^\psi, N) = Etree_\psi^{s-opt}(G, N)$. Let N be an internal node of G^ψ. Then

$$Etree(G^\psi, N) = \bigcup_{l \in DL_{G^\psi}(N)} Etree(G^\psi, N, l)$$

and, for each $l \in DL_{G^\psi}(N)$,

$$\begin{aligned} &Etree(G^\psi, N, l) \\ &\quad = \{etree(l, \Gamma_0, \Gamma_1) : \\ &\qquad \Gamma_\delta \in Etree(G^\psi, N(l,\delta)), \delta = 0,1\}. \end{aligned}$$

Assume that, for any $l \in DL_{G^\psi}(N)$ and $\delta \in \{0,1\}$,

$$Etree(G^\psi, N(l,\delta)) = Etree_\psi^{s-opt}(G, N(l,\delta)).$$

We know that

$$DL_{G^\psi}(N) = \{l : l \in DL_G(N), \psi_G(N,l) = \psi_G(N)\}.$$

Let $l \in DL_{G^\psi}(N)$, and $\Gamma \in Etree(G^\psi, N, l)$. Then $\Gamma = etree(l, \Gamma_0, \Gamma_1)$, where $\Gamma_\delta \in Etree\ (G^\psi, N(l,\delta))$ for $\delta = 0,1$. According to the induction hypothesis, $Etree(G^\psi, N(l,\delta)) = Etree_\psi^{s-opt}(G, N(l,\delta))$ and $\Gamma_\delta \in Etree_\psi^{s-opt}(G, N(l,\delta))$ for $\delta = 0,1$. In particular, $\psi(M, N(l,\delta), \Gamma_\delta) = \psi_G(N(l,\delta))$ for $\delta = 0,1$. Since $\psi_G(N,l) = \psi_G(N)$, we have

$$F(M, N, l, \psi_G(N(l,0)), \psi_G(N(l,1))) = \psi_G(N).$$

Therefore $\Gamma \in Etree_\psi^{opt}(G, N)$, $\Gamma \in Etree_\psi^{s-opt}(G, N)$, and

$$Etree(G^\psi, N) \subseteq Etree_\psi^{s-opt}(G, N).$$

Let $\Gamma \in Etree_\psi^{s-opt}(G, N)$. Since N is an internal node then Γ can be represented in the form $etree(l, \Gamma_0, \Gamma_1)$, where $l \in DL_G(N)$, and $\Gamma_\delta \in Etree_\psi^{s-opt}(G, N(l,\delta))$ for $\delta = 0,1$. Since $\Gamma \in Etree_\psi^{s-opt}(G, N)$, $\psi_G(N,l) = \psi_G(N)$ and $l \in DL_{G^\psi}(N)$. According to the induction hypothesis, $Etree(G^\psi, N(l,\delta)) = Etree_\psi^{s-opt}(G, N(l,\delta))$ for $\delta = 0,1$. Therefore $\Gamma \in Etree(G^\psi, N, l) \subseteq Etree(G^\psi, N)$. As a result, we have $Etree_\psi^{s-opt}(G, N) \subseteq Etree(G^\psi, N)$. ■

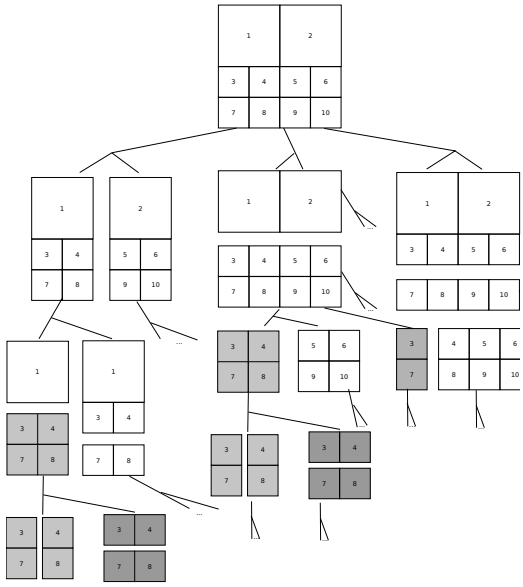

Fig. 3. Tree of partitions created by the optimization algorithm.

Corollary 1. *Let ψ be a strictly increasing cost function for element partition trees, M be a mesh, and G be a proper subgraph of the graph $\Delta(M)$. Then, for any node N of G^ψ, $Etree(G^\psi, N) = Etree_\psi^{opt}(G, N)$.*

This corollary follows immediately from Proposition 5 and Theorem 1.

3. Numerical results

We use a dynamic programming algorithm to find optimal element partition trees for a simple class of meshes. We assume that each division of a mesh follows a straight horizontal or vertical line that cuts the entire sub-domain. The element partition trees use these horizontal or vertical divisions of the whole mesh or mesh parts and the optimal ones are found using dynamic programming. The dynamic programming algorithm seeks to find all optimal element partition trees. The found trees are optimal among the studied class of element partition trees. The algorithm starts with the whole mesh and lists all possible divisions (by horizontal or vertical line) of the mesh. Then at every step, the algorithm finds all possible divisions (by a horizontal or a vertical line) of each previously found part of a mesh. If a logically identical partition of the mesh has been found before, further divisions of this part are omitted in following steps. In this way, the tree of partitions of the whole mesh can be constructed. Figure 3 shows the tree of partitions of an example mesh. Based on the obtained tree of partitions, the best partition (according to the defined cost function) is found. The best partition of each level is used to determine the class of optimal element partition trees for a given mesh.

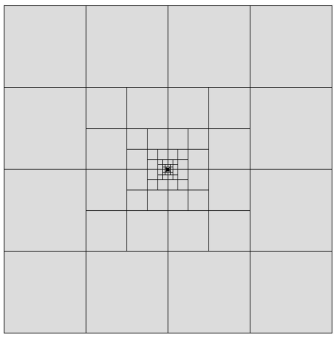

Fig. 4. Point singularity.

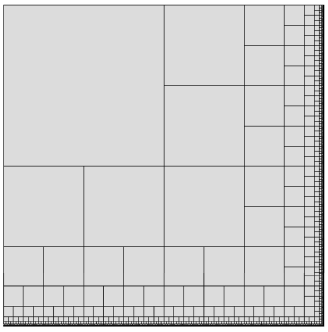

Fig. 5. Two edge singularities.

3.1. Quasi-optimal element partition trees. This section describes the results of the optimization of the element partition trees for meshes with a point singularity and two edge singularities, shown in Figs. 4 and 5.

These results are an extension of the conference presentations (AbouEisha *et al.*, 2015; AbouEisha *et al.*, 2014), where we focused on one representative tree, while in this paper we look at the whole class of trees.

We optimize with respect to the cost function defined in Section 2. The column "# all trees" in Tables 1 and 2 contains the number of all element partition trees for the meshes with point and two-edge singularities, respectively. For the point singularity, we optimize over one-quarter of the mesh. We call the point singularity and the two-edge singularity meshes P_k and A_k, respectively, where k denotes the number of refinement levels. The number of trees increases fast with the number of refinement steps. This prevents a naive search algorithm from solving the problem.

Table 1 summarizes the number of optimal element partition trees for P_k meshes with different polynomial orders p. All optimal trees for a given mesh have the same value of the studied cost function. The P_k meshes are simpler than the other investigated classes, so we are able to understand the behavior of any such optimal tree. For A_k meshes, the situation is more complex. Table 2 presents the number of optimal trees for A_k meshes.

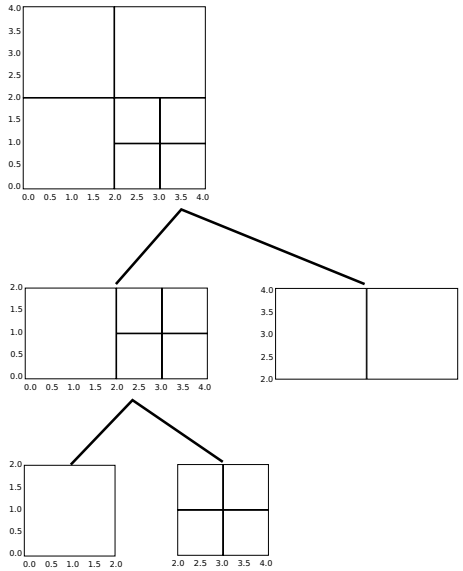

Fig. 6. Quasi-optimal element partition tree for the mesh with a point singularity.

Table 1. Number of optimal element partition trees for mesh P_k for different p and k.

		p				
k	# all trees	1	2	3	4	5
1	2	2	2	2	2	2
2	6	4	4	4	4	4
3	18	8	8	8	8	8
4	54	16	16	16	16	16
5	162	32	32	32	32	32
6	486	64	64	64	64	64
7	1458	128	128	128	128	128
8	4374	256	256	256	256	256
9	13122	512	512	512	512	512
10	39366	1024	1024	1024	1024	1024

We summarize this section with some example optimal element partition trees in Figs. 7 and 8.

4. Sample heuristic algorithm and 3D examples

We present a sketch of a simple heuristic algorithm that can be drawn from the presented dynamic programming results. By looking at the sample optimal element partition trees, generated for the refined grids, we can observe the following property. The levels of the trees contain elements with different sizes, namely the element obtained at various stages of the refinement process. Based on this observation, we can propose the following simple algorithm. We call the algorithm *bi-section weighted by element size*. The algorithm creates an initial undirected graph G for a finite element mesh. Each node of the graph G corresponds to one of finite

Table 2. Number of optimal element partition trees for mesh A_k for different p and k.

k	# all trees	p				
		1	2	3	4	5
1	2	2	2	2	2	2
2	580	6	6	6	6	6
3	2.6×10^8	8	8	8	8	8
4	2.9×10^{20}	8	8	8	8	8
5	1×10^{45}	48	48	48	48	48
6	6.6×10^{94}	32	32	32	32	32
7	9.2×10^{194}	64	64	64	64	64
8	6.5×10^{395}	128	128	128	128	128
9	1.2×10^{798}	256	256	256	256	256
10	1.7×10^{1603}	512	512	2048	2048	2048

elements from the mesh. There is an edge in the graph G connecting two nodes when the corresponding finite elements are neighbors in the mesh. Additionally, each node of graph G has an attribute *weight* with the value defined in the following way. The weight attribute is defined as the following function of the refinement level of an element:

$$weight = 2^{(3 \times refinement\_level)}. \tag{4}$$

The intuition behind the weight is that the smallest weight is associated with the "smallest" elements from the deepest refinement level, while the highest weight is associated to the "largest" elements from the top refinement level. After forming the initial graph G, the function named $BisectionWeightedByElementSize()$ is called with the parameter G. The function $BisectionWeightedByElementSize$ returns the element partition tree, and it is defined as follows: The

Algorithm 4. Bisection weighted by element size.

Require: G

1: **if** number of nodes in G equals 1 **then**
2: create one element tree t with the node $v \in G$;
3: **return** t;
4: **end if**
5: Calculate a balanced weighted partition of graph G into $G1$ and $G2$;
6: $t1$ = BisectionWeightedByElementSize(G1)
7: $t2$ = BisectionWeightedByElementSize(G2)
8: create new root node t with left child $t1$, right child $t2$
9: **return** t

function calls the METIS routine WPartGraphRecursive() to get the partition of the graph with nodes weighted by (4). Having the element partition tree generated by the algorithm, we can extract the ordering and call the sequential solver. For a detailed description on how to

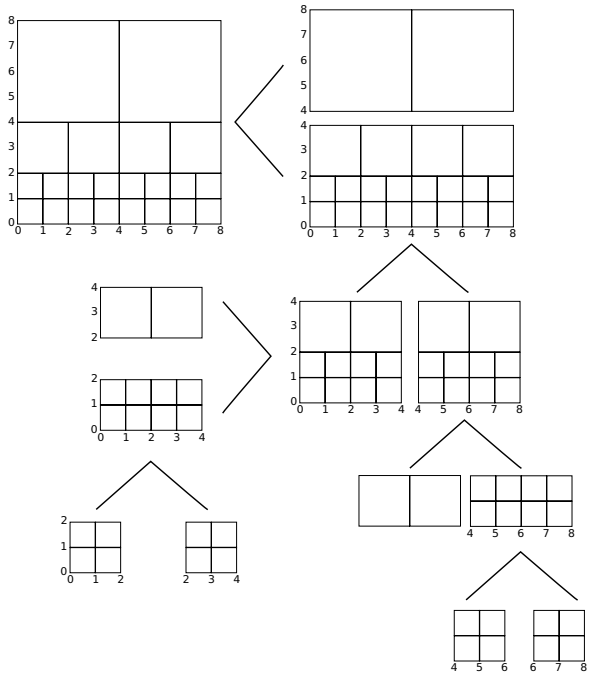

Fig. 7. Quasi-optimal element partition tree for mesh with an edge singularity.

construct ordering based on the element partition tree, we refer the reader to Chapter 9 of the book by Paszyński (2016).

We have executed the algorithm on sample three-dimensional grids with point, edge and face singularities. The resulting partitions of the mesh are denoted with different colors in Figs. 9–11.

We generated the orderings using our element partition trees and counted the resulting number of floating-point operations (multiplications, subtraction, and additions). We compared the number of floating-point operations with alternating state-of-the-art orderings, namely, the nested-dissections (Karypis and Kumar, 1999), approximate minimum degree AMD (Amestoy *et al.*, 1996), and PORD available through MUMPS interface. The comparison is presented in Tables 3-5.

Table 3. Number of floating-point operations resulting from bisections weighted by element size, METIS, PORD and AMD, for the three-dimensional grid with a point singularity.

k	N	bisections	METIS	PORD, AMD
1	125	72095	84900	81030
2	181	122275	159000	153300
3	237	172455	232400	217600
4	293	222635	300600	289900
5	349	272815	379300	362100
6	405	322995	491200	434400
7	461	373175	566100	506700

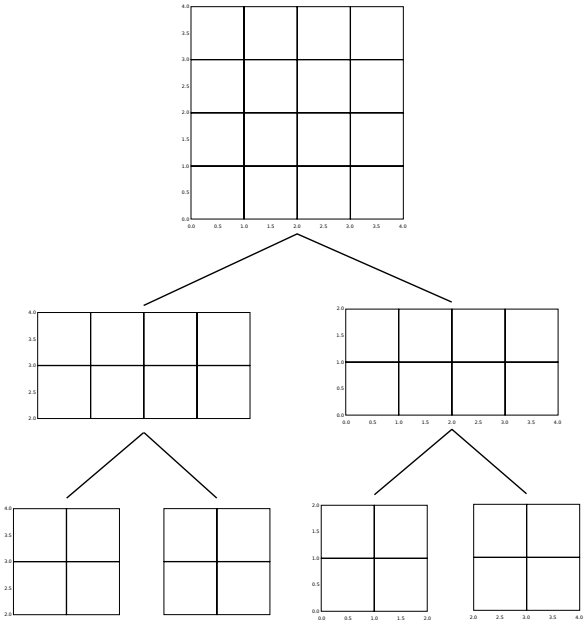

Fig. 8. Quasi-optimal element partition tree for the uniform mesh.

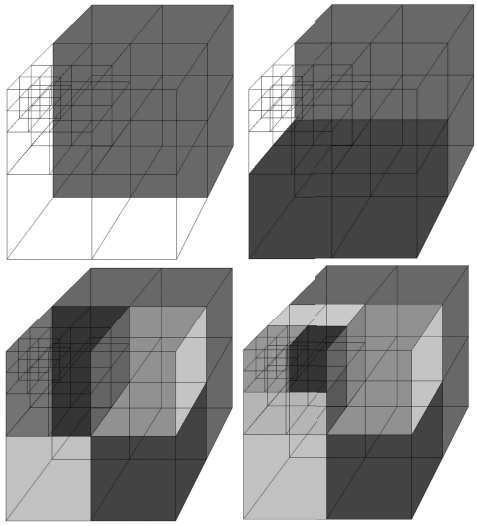

Fig. 9. Three-dimensional mesh with a point singularity partitioned by our algorithm.

Table 4. Number of floating-point operations resulting from bisections weighted by element size, METIS, PORD and AMD, for the three-dimensional grid with an edge singularity.

k	N	bisections	METIS	PORD, AMD
1	125	72095	84900	81030
2	249	244785	296700	281900
3	485	805867	1070000	894600
4	945	2474501	3358000	2662000
5	1853	6952239	9020000	7949000
6	3657	17988361	24040000	21760000
7	7253	43510547	66380000	55070000

Table 5. Number of floating-point operations resulting from bisections weighted by element size, METIS, PORD and AMD, for the three-dimensional grid with a face singularity.

k	N	bisections	METIS	PORD, AMD
1	125	72095	84900	81030
2	399	687516	860800	799400
3	1393	7590013	10990000	13320000
4	5171	79621866	113600000	240300000
5	19893	774584899	1081000000	3105000000

To compare the execution time, we consider a three-dimensional problem of projection of MRI scan data, described in detail by Schaefer *et al.* (2015). The sequence of three-dimensional h-adaptive grids used for the computations of the projection of the MRI scan data in presented in Fig. 12. The corresponding execution times of the MUMPS solver using the ordering resulting from either our bisections weighted by the element size algorithm or the automatic ordering provided by the MUMPS solver are summarized in Table 6.

5. Conclusion

We have described a dynamic programming algorithm with the polynomial time complexity that optimizes element partition trees for multi-frontal solvers for h-adaptive two-dimensional grids. We compare the number of total element partition trees and the number of optimal trees, for different numbers of refinement levels and different polynomial orders of approximation.

An intuitive description on how to construct of the element partition trees based on selected optimal element partition trees obtained from the optimization procedure is the following. We can see that the optimal tree processes the mesh level by level, by slices defined by the particular refinements. The partitions of the mesh usually generates two well-balanced parts, with an identical "volume" of elements. The optimal trees differ mainly by the order of selection of horizontal or vertical partitions on particular levels of the tree. We have used this insight to produce an outline of the heuristic algorithm that constructs the element partition tree by weighted recursive partitions of the mesh, using the refinement level as the element weights. We show that this algorithm allows us to reduce the number of floating-point operations or the execution time up to the factor of two. We are going to use this insight in the second part of the paper (AbouEisha *et al.*, 2016) to propose a more general heuristic algorithm,

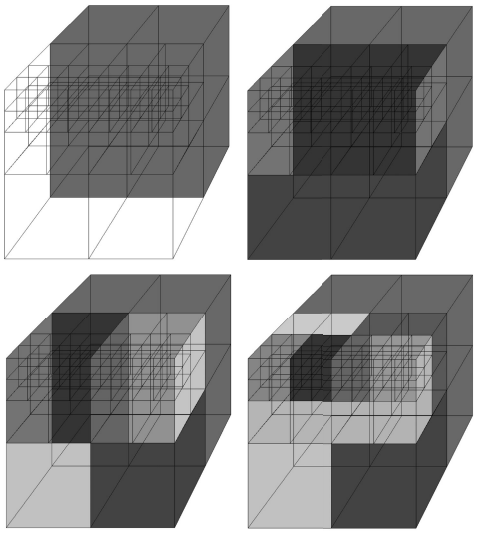

Fig. 10. Three-dimensional mesh with an edge singularity partitioned by our algorithm.

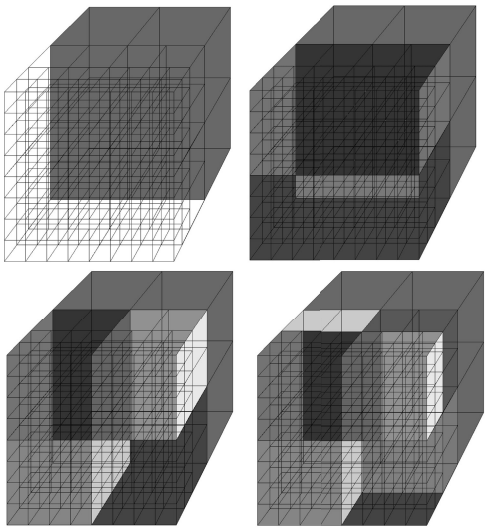

Fig. 11. Three-dimensional mesh with a face singularity partitioned by our algorithm.

Table 6. Execution times [s] of the factorization of the MUMPS solver with the ordering resulting from the bisection weighted by the size of the elements and with default MUMPS ordering.

N	bisections ordering	MUMPS ordering
60025	5,70	5,67
94221	14,49	28,29
139425	33,06	67,94
197173	71,59	142,64

targeting the hp-adaptive computations.

Acknowledgment

The work was partially supported by the Center for Numerical Porous Media, King Abdullah University of Science and Technology (KAUST), and by the National Science Centre, Poland, grant no. DEC-2012/06/M/ST1/00363. This publication also was made possible by a National Priorities Research Program grant 7-1482-1-278 from the Qatar National Research Fund (a member of The Qatar Foundation). This work was partially supported by the European Union's *Horizon 2020* research and an innovation program under the Marie Skłodowska-Curie grant agreement no. 644602. The J. Tinsley Oden Faculty Fellowship Research Program at the Institute for Computational Engineering and Sciences (ICES) of the University of Texas at Austin partially supported the visits of Victor Manuel Calo to the ICES.

References

AbouEisha, H., Calo, V.M., Jopek, K., Moshkov, M., Paszyńska, A., Paszyński, M. and Skotniczny, M. (2016). Element partition trees for two- and three-dimensional h-refined meshes and their use to optimize direct solver performance. Part II: Heuristic algorithms, *Journal of Computational and Applied Mathematics*, (submitted).

AbouEisha, H., Gurgul, P., Paszyńska, A., Paszyński, M., Kuźnik, K. and Moshkov, M. (2015). An automatic way of finding robust elimination trees for a multi-frontal sparse solver for radical 2D hierarchical meshes, *in* R. Wyrzykowski *et al.* (Eds.), *Parallel Processing and Applied Mathematics*, Lecture Notes in Computer Science, Vol. 8385, Springer, Berlin/Heidelberg, pp. 531–540.

AbouEisha, H., Moshkov, M. Calo, V.M., Paszyński, M., Goik, D. and Jopek, K. (2014). Dynamic programming algorithm for generation of optimal elimination trees for multi-frontal direct solver over h-refined grids, *Procedia Computer Science* **29**: 947–959.

Amestoy, P.R., Davis, T.A. and 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. and L'Excellent, J.-Y. (2000). Multifrontal parallel distributed symmetric and unsymmetric solvers, *Computer Methods in Applied Mechanics and Engineering* **184**(2): 501–520.

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

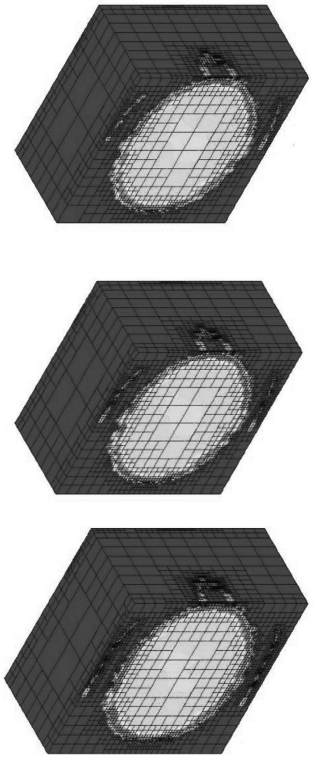

Fig. 12. Cross-sections of a sequence of adaptive three-dimensional grids.

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

Babuška, I. and Rheinboldt, W.C. (1978). Error estimates for adaptive finite element computations, *SIAM Journal of Numerical Analysis* **15**(4): 736–754.

Bao, G., Hu, G. and Liu, D. (2012). An h-adaptive finite element solver for the calculations of the electronic structures, *Journal of Computational Physics* **231**(14): 4967–4979.

Barboteu, M., Bartosz, B. and Kalita, P. (2013). An analytical and numerical approach to a bilateral contact problem with nonmonotone friction, *International Journal of Applied Mathematics and Computer Science* **23**(2): 263–276, DOI: 10.2478/amcs-2013-0020.

Becker, R., Kapp, H. and Rannacher, R. (2000). Adaptive finite element methods for optimal control of partial differential equations: Basic concept, *SIAM Journal on Control and Optimisation* **39**(1): 113–132.

Belytschko, T. and Tabbar, M. (1993). H-adaptive finite element methods for dynamic problems, with emphasis on localization, *International Journal for Numerical Methods in Engineering* **36**(24): 4245–4265.

Demkowicz, L. (2006). *Computing with hp-Adaptive Finite Elements, Vol. I: One and Two Dimensional Elliptic and Maxwell Problems*, Chapman and Hall/CRC, Boca Raton, FL .

Duff, I.S., Erisman, A.M. and Reid, J.K. (1986). *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford.

Duff, I.S. and Reid, J.K. (1983). The multifrontal solution of indefinite sparse symmetric linear, *ACM Transactions on Mathematical Software* **9**(3): 302–325.

Duff, I.S. and Reid, J.K. (1984). The multifrontal solution of unsymmetric sets of linear equations, *SIAM Journal on Scientific and Statistical Computing* **5**(3): 633–641.

Errikson, K. and Johnson, C. (1991). Adaptive finite element methods for parabolic problems. I: A linear model problem, *SIAM Journal on Numerical Analysis* **28**(1): 43–77.

Fiałko, S. (2009a). A block sparse shared-memory multifrontal finite element solver for problems of structural mechanics, *Computer Assisted Mechanics and Engineering Science* **16**: 117–131.

Fiałko, S. (2009b). The block subtracture multifrontal method for solution of large finite element equation sets, *Technical Transactions* **8**: 175–188.

Fiałko, S. (2010). PARFES: A method for solving finite element linear equations on multi-core computers, *Advanced Engineering Software* **40**(12): 1256–1265.

Hughues, T. (2000). *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, Dover, New York, NY.

Karczewska, A., Rozmej, P., Szczeciński, M. and Boguniewicz, B. (2016). A finite element method for extended KdV equations, *International Journal of Applied Mathematics and Computer Science* **26**(3): 555–567, DOI: 10.1515/amcs-2016-0039.

Kardani, M., Nazem, M., Abbo, A.J., Sheng, D. and Sloan, S.W. (2012). Refined h-adaptive finite element procedure for large deformation geotechnical problems, *Computational Mechanics* **49**(1): 21–33.

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

Liu, J. (1990). The role of elimination trees in sparse factorization, *SIAM Journal of Matrix Analysis Applications* **11**(1): 134–172.

Niemi, A., Babuška, I., Pitkaranta, J. and Demkowicz, L. (2012). Finite element analysis of the Girkmann problem using the modern hp-version and the classical h-version, *Procedia Computer Science* **28**: 123–134.

Paszyński, M. (2016). *Fast Solvers for Mesh Based Computations*, Taylor & Francis/CRC Press, Boca Raton, FL.

Patro, S.K., Selvam, P.R. and Bosch, H. (2013). Adaptive h-finite element modeling of wind flow around bridges, *Engineering Structures* **48**: 569–577.

Schaefer, R., Łoś, M., Sieniek, M., Demkowicz, L. and 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.

Strug, B., Paszyńska, A., Paszyński, M. and Grabska, E. (2013). Using a graph grammar system in the finite element method, *International Journal of Applied Mathematics and Computer Science* **23**(4): 839–853, DOI: 10.2478/amcs-2013-0063.

Zienkiewicz, O.C., Taylor, R. and Z., Z.J. (2013). *The Finite Element Method: Its Basis and Fundamentals*, Elsevier, Amsterdam.

Hassan AbouEisha is a PhD student with the CEMSE Division at the King Abdullah University of Science Technology. He graduated from German University in Cairo in the field of computer science and engineering in 2010. He received his MSc in computer science from the KAUST in 2011. His main research interests are multi-frontal solvers, discrete optimization and computational complexity.

Victor Calo is a professor in the Department of Applied Geology of the Western Australian School of Mines in the Faculty of Science and Engineering at Curtin University. He is a highly cited researcher who is actively involved in disseminating knowledge: Dr. Calo has authored over 150 peer-reviewed publications. Also, in the last two years, he has given more than 25 invited presentations and keynotes at conferences and seminars, and organized 15 mini-symposia at international conferences. Dr. Calo holds a professional engineering degree in civil engineering from the University of Buenos Aires. He received his MSc in geomechanics and a doctorate in civil and environmental engineering from Stanford University. Dr. Calo's research interests include modeling and simulation of geomechanics, fluid dynamics, flow in porous media, phase separation, fluid-structure interaction, solid mechanics, and high-performance computing.

Konrad Jopek is a PhD student in the Department of Computer Science of the AGH University of Science and Technology. He is a co-author of several publications about fast solvers for the finite element method. His research interests focus on numerical methods, parallel algorithms, highly scalable software and low-level programming.

Mikhail Moshkov has been a professor in the Computer, Electrical and Mathematical Sciences and Engineering Division at the King Abdullah University of Sience Technology since 2008. He holds an MSc degree from Nizhni Novgorod State University, a doctorate from Saratov State University, and habilitation from Moscow State University. From 1977 to 2004, Dr. Moshkov was with Nizhni Novgorod State University. Since 2003 he has worked in Poland in the Institute of Computer Science, University of Silesia, and since 2006 also in the Katowice Institute of Information Technologies. His main areas of research are the complexity of algorithms, combinatorial optimization, and machine learning. Dr. Moshkov is the author or a coauthor of five research monographs published by Springer.

Anna Paszyńska received her PhD (2007) in computer science from the Institute of Fundamental Technological Research of the Polish Academy of the Sciences in Warsaw, Poland. She currently works as an assistant professor at the Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University in Kraków, Poland. Her research interests include evolutionary algorithms, graph grammars, computer aided design and engineering.

Maciej Paszyński is an associate professor in the Department of Computer Science, AGH University of Science and Technology, Kraków, Poland. He obtained his PhD in mathematics with applications to computer science from Jagiellonian University in 2003, and habilitation in computer science in 2010 from the AGH University of Science and Technology. His research interests include fast solvers for adaptive finite element method simulations. He has co-authored 40 papers in impact factor journals. He has also given over 100 presentation at conferences and workshops.

Marcin Skotniczny is a second year PhD student of computer science at the Faculty of Computer Science, Electronics and Telecommunication of the AGH University of Science and Technology. He is also a chief executive of the software company Software Mansion in Kraków, Poland. His research interests include computational complexities and advanced algorithms.

Received: 11 September 2016
Revised: 30 December 2016
Accepted: 5 February 2017