Hindawi Publishing Corporation
Scientific Programming
Volume 2015, Article ID 303024, 18 pages
http://dx.doi.org/10.1155/2015/303024

Research Article

Quasi-Optimal Elimination Trees for 2D Grids with Singularities

A. Paszyńska,[1] M. Paszyński,[2] K. Jopek,[2] M. Woźniak,[2] D. Goik,[2] P. Gurgul,[2] H. AbouEisha,[3] M. Moshkov,[3] V. M. Calo,[3,4] A. Lenharth,[5] D. Nguyen,[5] and K. Pingali[5]

[1] *Jagiellonian University, 31007 Krakow, Poland*
[2] *AGH University of Science and Technology, 30059 Krakow, Poland*
[3] *Applied Mathematics & Computational Science, King Abdullah University of Science and Technology (KAUST), Thuwal 23955-6900, Saudi Arabia*
[4] *Earth Science & Engineering and Center for Numerical Porous Media, King Abdullah University of Science and Technology (KAUST), Thuwal 23955-6900, Saudi Arabia*
[5] *Institute for Computational Engineering and Science, University of Texas, Austin, TX 78712-1229, USA*

Correspondence should be addressed to M. Paszyński; maciej.paszynski@agh.edu.pl

Received 16 October 2013; Revised 28 April 2014; Accepted 25 November 2014

Academic Editor: Ron Perrott

We construct quasi-optimal elimination trees for 2D finite element meshes with singularities. These trees minimize the complexity of the solution of the discrete system. The computational cost estimates of the elimination process model the execution of the multifrontal algorithms in serial and in parallel shared-memory executions. Since the meshes considered are a subspace of all possible mesh partitions, we call these minimizers quasi-optimal. We minimize the cost functionals using dynamic programming. Finding these minimizers is more computationally expensive than solving the original algebraic system. Nevertheless, from the insights provided by the analysis of the dynamic programming minima, we propose a heuristic construction of the elimination trees that has cost $\mathcal{O}(N_e \log(N_e))$, where N_e is the number of elements in the mesh. We show that this heuristic ordering has similar computational cost to the quasi-optimal elimination trees found with dynamic programming and outperforms state-of-the-art alternatives in our numerical experiments.

1. Introduction

We present a dynamic programming algorithm to find quasi-optimal elimination tree for two-dimensional grids with point and edge singularities. We consider two cost functions: one models the sequential solver execution, while the other models the execution cost of a parallel shared-memory solver. The dynamic programming algorithm finds elimination trees that minimize the appropriate cost function for a multifrontal solver. These minimizers belong to a class of elimination trees obtained by recursive partition of the computational mesh along straight lines. These quasi-optimal trees are expressed as graph-grammar productions which define our solver. To optimize execution time, we use the GALOIS scheduler [1]. From the analysis of the quasi-optimal trees we propose a heuristic algorithm that constructs in $\mathcal{O}(N_e \log(N_e))$ time (where N_e denotes the number of elements of the mesh) the elimination trees with similar performance to the one constructed with the dynamic programming algorithm. To determine the efficiency of our algorithms, we compare our elimination trees to popular alternatives. The comparison is performed using MUMPS as an efficient interface to commonly used elimination algorithms, such as approximate minimum fill, approximate minimum degree, SCOTCH, PORD, METIS, and AMD with quasi-row detection. In particular, we estimate the number of FLOPs (floating-point operations) in sequential execution of our graph-grammar solver using elimination trees generated by our dynamic programming and heuristic algorithms. We compare these to the FLOPs resulting from execution of MUMPS using execution time as a proxy for FLOPs. We also use numerical experiments to compare the execution times of our sequential and parallel solvers against those of sequential and parallel MUMPS. Seeking to improve the parallel performance, we

consider the tree rotation algorithm to well balance the elimination trees for parallel computations. We show that both sequential and parallel graph-grammar-based solver with GALOIS scheduler, using the elimination trees constructed by the dynamic programming optimization as well as the heuristic algorithm, outperform MUMPS with any ordering.

In this paper we present new contributions in the following research areas.

Multifrontal Solvers. The computational cost of the multifrontal solver algorithm depends on the quality of the elimination tree (which in sequential mode can be called an ordering). The problem of finding of an optimal elimination tree for a given mesh resulting in minimal computational cost of either sequential or parallel multifrontal solver algorithm is NP-complete [2]. However, for a fixed grid it is possible to define a large class of elimination trees, estimate the computational costs for the solver algorithm for each tree, and select the best one in each class. We introduce a dynamic programming algorithm to find elimination trees for a given mesh that minimizes the computational cost of sequential multifrontal direct solver algorithm. The elimination trees obtained by our dynamic programming algorithm are obtained by considering recursive partitions of the computational mesh along straight lines. Thus we call the resulting trees quasi-optimal, since we do not consider all possible elimination trees. We also restrict our research to the case of initially structured two-dimensional grids with rectangular finite elements, where the partitions along straight lines are possible to implement. From our experience deriving quasi-optimal elimination trees for ordering of meshes, we developed insights and abstractions that allowed us to propose a heuristic algorithm that constructs elimination trees with similar properties to the trees constructed by dynamic programming algorithm. The heuristic algorithm can construct the trees in $\mathcal{O}(N_e \log(N_e))$ computational cost, where N_e is the number of elements in the mesh. We have executed the multifrontal solver algorithm for some representative grids, namely, for grids with a point singularity and grids with an edge singularity. We estimated the number of FLOPs of the multifrontal solver algorithm for the elimination trees constructed by both the dynamic programming and heuristic algorithms. We compare them to the FLOPs resulting from execution of the state-of-the-art ordering algorithms. These are approximate minimum fill, approximate minimum degree, SCOTCH, PORD, METIS (nested-dissection), and AMD with quasi-row detection. We show that our elimination trees resulting from both dynamic programming and the heuristic algorithms outperform the alternative ordering algorithms in terms of FLOPs for sequential solver execution.

Graph-Grammar-Based Solvers. We express our elimination trees and the resulting multifrontal solver as a sequence of graph-grammar productions. Namely, the graph-grammar productions construct frontal matrices, merge Schur complement matrices of children nodes of each node of the binary elimination tree, eliminate fully assembled degrees of freedom, and execute backward substitutions. The graph-grammar productions are implemented in the GALOIS environment [1]. The dependency relation between graph-grammar productions follows the structure of the elimination tree and can be expressed as a directed acyclic graph (DAG). The graph-grammar productions are implemented as GALOIS tasks working on the DAG obtained directly from the elimination tree. We compare the execution time of our graph-grammar-based GALOIS solver with the execution time of sequential MUMPS and show that our solver outperforms this implementation.

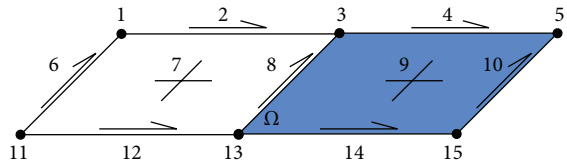

Figure 1: Sample computational domain for the frontal solver.

Parallelism. We execute the dynamic programming algorithm with a modified cost function that reflects the computational cost of the parallel shared-memory multifrontal solver algorithm. The dynamic programming algorithm finds elimination trees that minimize the computational cost for the parallel shared-memory multifrontal solver, within the class of elimination trees obtained by recursive partitions of the computational mesh along the straight lines. We use tree rotation to improve the balancing of the obtained elimination trees. As before, we express the resulting solver as a sequence of graph-grammar productions which can be optimally scheduled using the DAG analysis of GALOIS. These optimally scheduled graph-grammar productions are run on multithreaded execution on a shared-memory machine. We use four different elimination trees: the quasi-optimal dynamic programming using a multithreaded cost function and its rotated tree as well as a heuristic elimination tree and its rotated counterpart. We compare these four solvers against parallel MUMPS on the same machine. All four solvers outperform MUMPS.

1.1. Finite Element Method (FEM), Multifrontal Solver, and Elimination Trees. In this paper we focus on a class of two dimensional structured meshes with rectangular finite elements, subject to h refinement as it is described by Demkowicz in [3]. Let us focus on a simple 2D finite element mesh. The domain Ω is described by two elements and fifteen supernodes, that is, two interiors, seven edges, and six vertices (see Figure 1). In the 2D adaptive FEM, described in [3], we utilize basis functions related to abstract element supernodes. In this example (see Figure 2), we have linear basis functions associated with element vertices, namely, with supernodes 1, 3, 5, 11, 13, and 15, quadratic basis functions associated with element edges, namely, with supernodes 2, 4, 6, 8, 10, 12, and 14, and quadratic basis functions associated with element interiors, namely, with supernodes 7 and 9. This case corresponds to the polynomial order of approximation $p = 2$. In the general case of order p, we have $p - 1$ basis functions related to each element edge and $(p - 1)^2$ basis functions

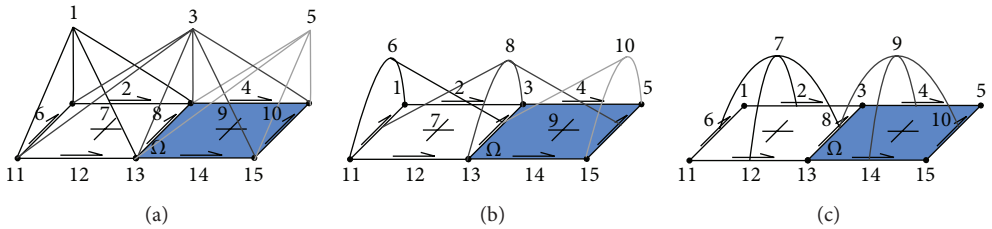

FIGURE 2: Exemplary basis functions spread over element supernodes: (a) basis function associated with vertex supernode 1 (black), vertex supernode 3 (dark gray), and vertex supernode 5 (light gray); (b) basis function associated with edge supernode 6 (black), edge supernode 8 (dark gray), and edge supernode 10 (light gray); (c) basis function associated with interior supernode 7 (black) and interior supernode 9 (dark gray).

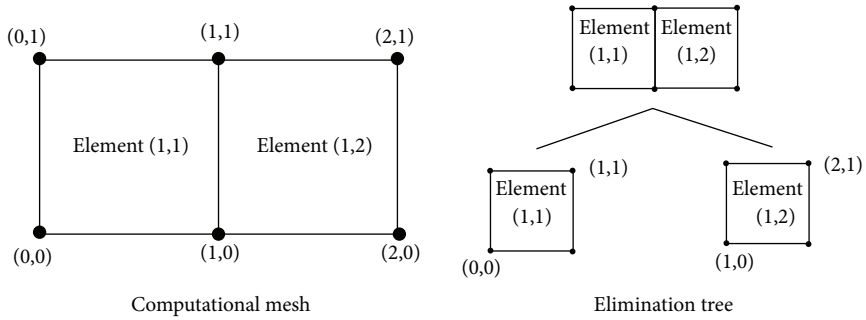

FIGURE 3: Computational domain expressed as an elimination tree.

related to each element interior. In 2D FEM [3], we construct the algebraic system by computing inner products of these basis functions or their derivatives over the analyzed domain. Thus, each entry of the resulting matrix system corresponds to the interaction between particular basis functions. The numerical values of these interactions are determined by the choice of weak form and the support of the basis functions. The connectivity, which is the controlling characteristic of the computational complexity, is only determined by the supports and the weak form. Interior basis functions have support over an element only, edge basis functions have support over one or two elements, and vertex basis functions have their support spread over one or many elements, depending on the grid connectivity.

The multifrontal solver introduced by Duff and Reid [4, 5] is a popular solver for systems of linear equations, which is a generalization of the frontal solver algorithm described in [6]. In a multifrontal solver, connectivity analysis is performed using an *elimination tree*. The elimination tree in classical solvers is obtained from a planar graph analysis. The graph is constructed based on the sparsity of the global matrix. In our solver, however, we construct the elimination tree by analyzing the mesh. A computational domain is decomposed into a hierarchy of subdomains, which defines an elimination tree (Figure 3). The construction of the elimination tree for an arbitrary mesh is a complex task. The elimination tree is constructed using the graph representing the connectivities of the mesh. This graph is partitioned using algorithms such as *nested-dissection* from the METIS library [7, 8] or minimum degree algorithms [9]. Usually, solvers like MUMPS [10–12] are not aware of the structure of the mesh, and they need to reconstruct the connectivity pattern by analyzing

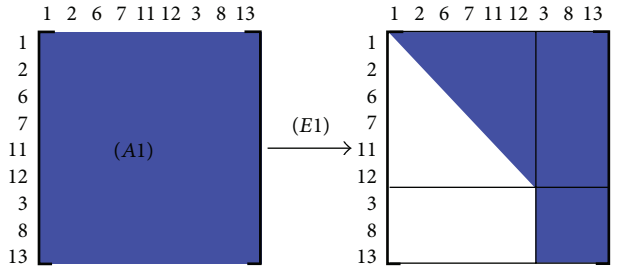

FIGURE 4: Assembly and partial forward elimination on the left element. These operations are expressed by two graph-grammar productions. ($A1$) represents the process of generation and assembly of the frontal matrix and ($E1$) represents the partial forward eliminations.

the sparsity pattern of the matrix given to the solver. The sparse representation of the mesh connectivity for linear order FE method, finite differences, and particle methods directly implies the mesh (or the topological structure of the mesh). For the high-order FE methods, the sparse representation does not precisely reflect the mesh structure (it represents the discretization explicitly). In the multifrontal approach, the solver generates a frontal matrix for each element of the mesh. This is illustrated in Figures 4 and 5. Fully assembled supernodes are eliminated within each frontal matrix, and the resulting Schur complement matrices are merged at the parent level of the tree. This is illustrated in Figure 6. Finally, the solver computes the solution at the elimination tree root node followed by backward substitutions at child nodes. This process is presented in Figure 7.

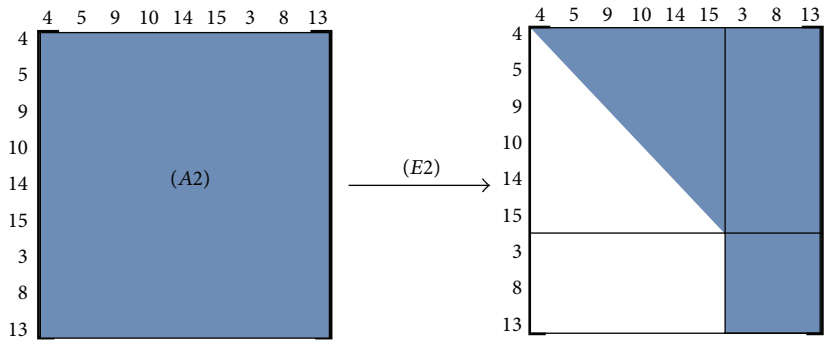

Figure 5: Partial forward elimination on the right element. This operation is expressed by two graph-grammar productions. ($A2$) represents again the process of generation and assembly of the frontal matrix and ($A2$) represents the partial forward eliminations.

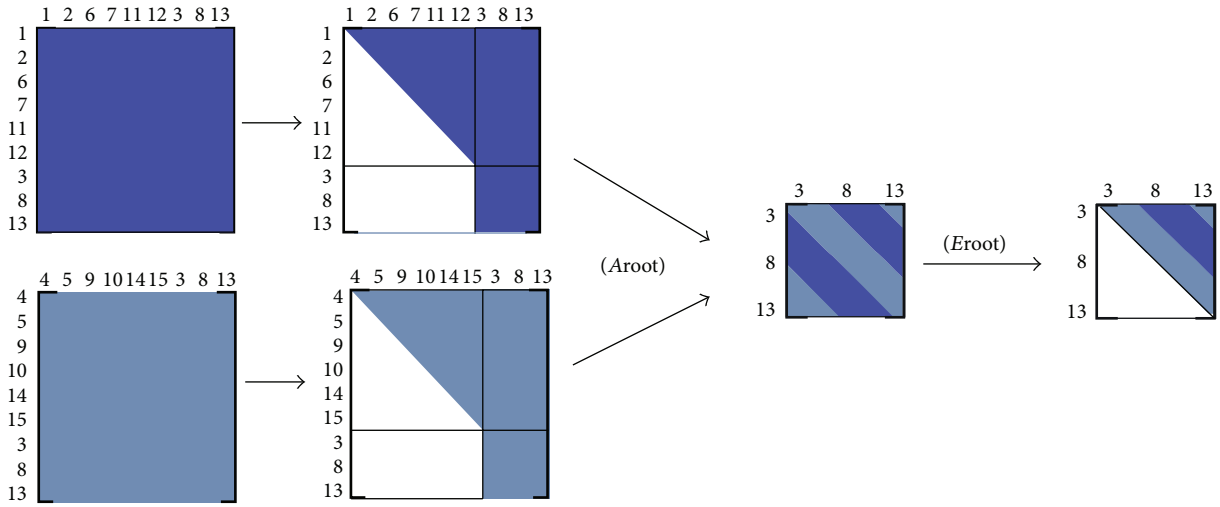

Figure 6: Assembly and full forward of the interface problem matrix. These operations are expressed by two graph-grammar productions. (Aroot) represents the process of merging of the Schur complements from the son nodes and (Eroot) represents full elimination at the root node.

1.2. Graph-Grammar-Based Solver. The topological structure of the mesh [13–16] as well as the multifrontal solver algorithm [17, 18] can be expressed by graph-grammar productions [19–22]. In this section we express the multifrontal solver algorithm by graph-grammar productions that directly follow the structure of the elimination tree. We present the implementation of the multifrontal solver in the GALOIS system for sequential and concurrent execution of graph-grammar productions. The input for our solver is the elimination tree, coded in the following way:

```
2 <- polynomial order of approximation
2 <- number of elements
1 1 0 0 1 1 <- first element id (1, 1)
level 1, element 1, and its coordinates
(0,0), (1,1)
1 2 1 0 2 1 <- first element id (1, 2)
level 1, element 2, and its coordinates
(1, 0), (2, 1)
3 <- number of nodes in elimination tree
1 2 1 1 1 2 2 3 <- tree node id = 1,
2 elements, (1, 1) and (1, 2), pointers
to son nodes 1, 2
2 1 1 1 <- tree node id = 2, 1 element
(1, 1) no son nodes
3 1 1 2 <- tree node id = 3, 1 element
(1, 2) no son nodes.
```

This example tree corresponds to the case presented in Figure 3.

Given the elimination tree, the operations performed by the solver can be coded as graph-grammar productions, working over the elimination tree. In particular, each merging and elimination operation can be represented as a single

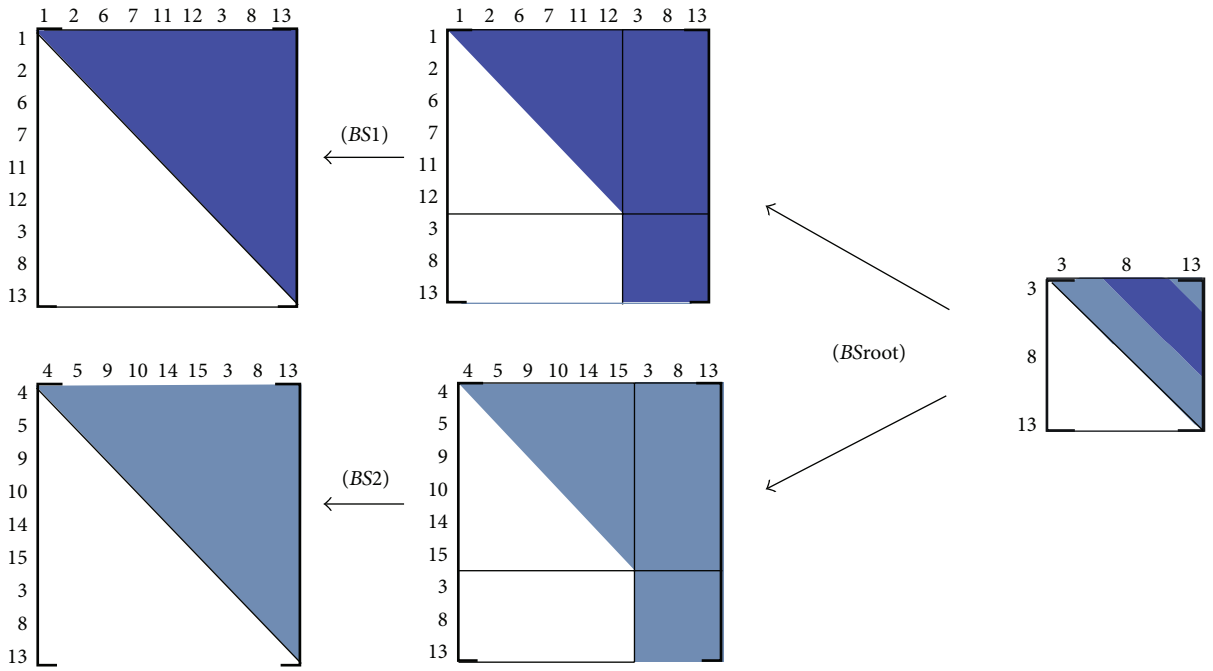

Figure 7: Backward substitution at root node followed by backward substitutions at child nodes. These operations are expressed by three graph-grammar productions. (*BS*root) represents the process of backward substitution at root node and (*BS*1), (*BS*2) represent backward substitutions at child nodes.

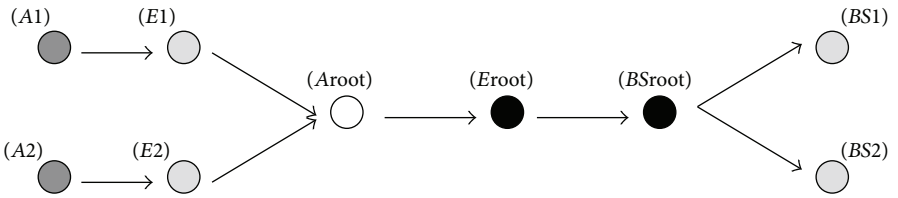

Figure 8: Directed acyclic graph for the elimination tree for a two-finite-element mesh.

graph-grammar production. The above example contains the following graph-grammar productions:

$$(A1)\text{ - }(E1)\text{ - }(A2)\text{ - }(E2)\text{ - }(A\text{root})\text{ - }(E\text{root})\text{ - }(BS\text{root})\text{ - }(BS1)\text{ - }(BS2)\,. \tag{1}$$

The dependency relation between these graph-grammar productions strictly follows the elimination tree and it is represented as a directed acyclic graph (DAG). This representation is equivalent to the one obtained by the trace theory [17, 23]. These graph-grammar productions are implemented as GALOIS tasks working on the DAG representing the elimination tree. The DAG for our simple example is presented in Figure 8. We start from graph-grammar productions located at the leaves of the elimination tree, then we go up to the root, and finally we go back down to the leaves. The tasks are then managed and scheduled by GALOIS [1]. That is, there is a direct relation between the structure of the elimination tree, the graph-grammar productions, the dependency relation between them, and the scheduling based on DAG in GALOIS. Thus, we present the elimination trees generated by our algorithms and we refrain from listing the graph-grammar productions.

2. Computational Cost Estimates for Sequential and Parallel Multifrontal Solver Algorithm

In order to estimate the number of floating-point operations (FLOPs) executed by the multifrontal algorithm we start with estimation of the FLOPs number during elimination of a rows from square matrix M of size $b \times b$ (see Figure 9). The FLOPs number as derived in [24] is equal to

$$C(a,b) = \frac{a\left(6b^2 - 6ab + 6b + 2a^2 - 3a + 1\right)}{6}. \tag{2}$$

To estimate the sequential execution cost of the multifrontal solver we sum up the costs of all the nodes of the elimination tree. To estimate the parallel shared-memory execution cost of the multifrontal solver we sum the maximum cost of each level of the elimination tree. Additionally we assume that we have enough cores to process all frontal

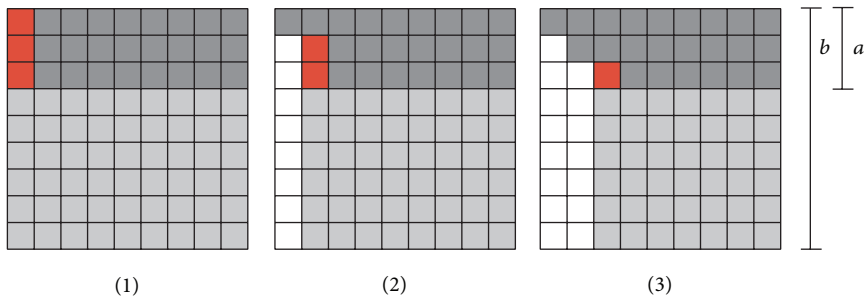

Figure 9: Elimination of a fully assembled rows from matrix M of size $b \times b$. In this example $a = 3$ and $b = 9$, and the row subtraction is performed in three steps, denoted on panels (1), (2), and (3). The dark gray squares denote rows to be eliminated, the red squares denote the value checked by partial pivoting, and the white squares denote zeros generated during the row subtractions.

Table 1: Estimation of computational cost on a two-element domain for graph-grammar productions expressing the multifrontal solver algorithm for $p = 2$, for sequential and parallel shared-memory solver executions.

Graph-grammar production	a	b	OPS(a, b)
(PelimM1_1)	6	9	271
(PelimM1_2)	6	9	271
(PsolveM1_1,2)	3	3	27
Total sequential			271 + 271 + 27 = 569
Total parallel			max(271, 271) + 27 = 288

matrices from all levels of the elimination tree in parallel, which is not always the case in practical computations. This is illustrated in Table 1.

Our solver uses partial pivoting. The pivoting is performed over the local frontal matrices at all the levels of the elimination tree. Partial pivoting compares the values of all the entries located below the diagonal within the rows that are fully assembled, which can be eliminated at this level. We do not pivot with nonfully assembled rows. From the point of view of the FLOPs, pivoting does not require FLOPs; it requires a few comparisons followed by a swift of the integers in the vector representing rows order. Thus we do not include the cost of pivoting in the cost function. The implementation of the pivoting requires just one loop through diagonal column, followed by swap of the two indexes in the row indexes vector. The cost of pivoting is negligibly small in comparison to the factorization itself. In our previous papers [18, 25] we have developed one two-dimensional solver and one three-dimensional solver using such partial pivoting algorithm and we have solved a number of computational problems with h, p, and hp adaptivity, including linear elasticity [26], Poisson equation [27], Maxwell equations [28], propagation of acoustics waves over the human head [29], and the Stokes flow problem [30]. We have not encountered convergence or round-off error problems.

3. Quasi-Optimal Elimination Trees by Dynamic Programming

The search for the optimal elimination tree can be represented by the directed acyclic graph (DAG) presented in Figure 10. The DAG root node represents the entire computational mesh, while child DAG nodes represent possible partitions of the mesh. We consider partitions of the mesh along straight lines which can be either horizontal or vertical. Thus, child DAG nodes of a root DAG node represent all possible partitions of the root along straight lines. We repeat this partition process recursively, until we reach leaves representing single finite elements. Some subbranches of the DAG are identical. For example, we identify identical branches in Figure 10 by red or green. These subbranches do not need to be regenerated, since we can use the pointer to an already generated identical subbranch. The elimination trees are represented as binary subtrees of this DAG. The optimization procedure is executed twice, once for each cost function defined recursively below. One cost function corresponds to sequential solver cost; the other one corresponds to parallel solver cost. The cost of processing the internal DAG node is defined as

$$\begin{aligned}&\text{cost of processing internal DAG node}\\&\quad=\text{cost of processing first child DAG node}\\&\qquad+\text{cost of processing second child DAG node}\\&\qquad+\text{cost of elimination of common interface}\end{aligned}\tag{3}$$

for the optimization performed for the sequential solver execution and

$$\begin{aligned}&\text{cost of processing internal DAG node}\\&\quad=\max\{\text{cost of processing first child DAG node},\\&\qquad\qquad\text{cost of processing second child DAG node}\}\\&\qquad+\text{cost of elimination of common interface}\end{aligned}\tag{4}$$

for the optimization performed for the parallel shared-memory solver execution. Again, this is only true under

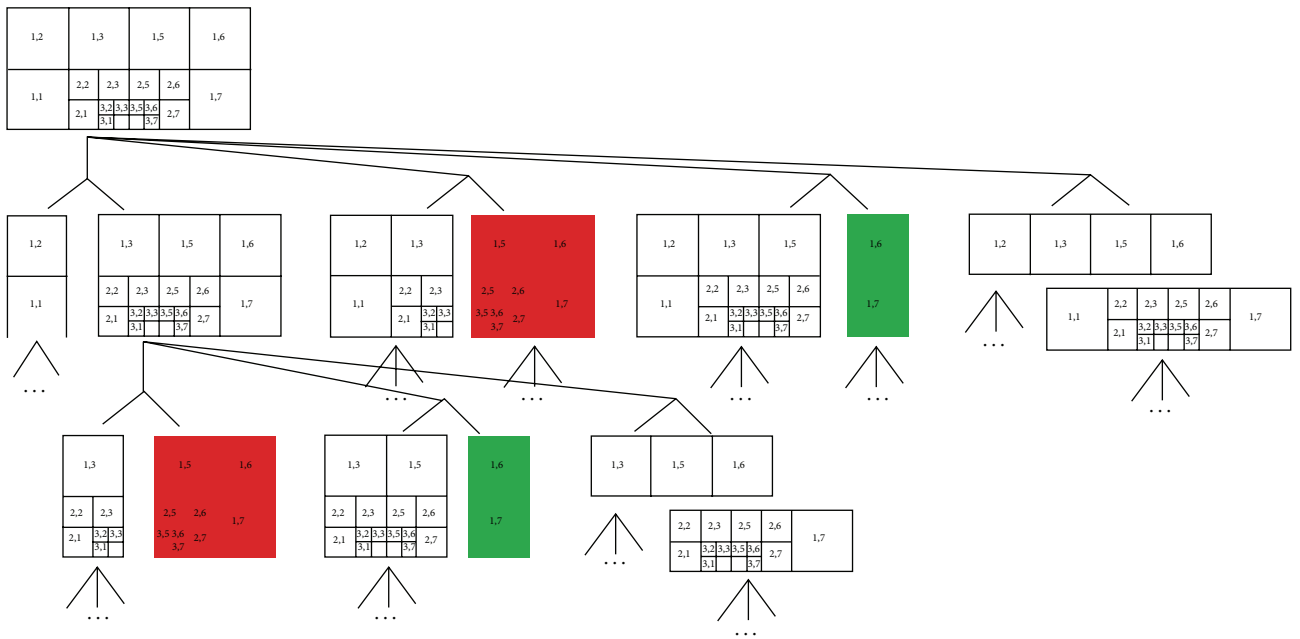

Figure 10: Tree of partitions used by the dynamic programming algorithm.

the assumption that we have enough available threads to process all frontal matrices from a given level of the elimination tree at the same time, which is not always the case in practical application. Each node of the elimination tree contains a frontal matrix with size b having some number a of fully assembled degrees of freedom. Leaf nodes contain element frontal matrices with fully assembled internal supernodes which can be eliminated. The cost $C(a, b)$ of elimination of a fully assembled supernodes from frontal matrix of size b has been defined in (2). This is just the number of operations for the partial forward elimination algorithm. Given a geometric description of the finite element mesh, the dynamic programming algorithm works in two steps. In the first step, the DAG representing the subproblems and dependency relations between them is constructed. Then, the DAG is optimized in a bottom-up approach. The DAG is constructed as follows. The algorithm adds a first DAG node to the DAG corresponding to the initial mesh. At any subsequent step $t > 1$, any unprocessed DAG node is processed and this DAG node is marked as processed. The algorithm terminates once all DAG nodes are processed. To process a DAG node we list all possible bisections of the (sub)mesh the DAG node represents, which we denote as a nodal mesh. The nodal submesh bisections use straight vertical or horizontal lines to split the mesh into two. These straight lines are called separators. For each separator (bisection of the submesh) two children DAG nodes are assigned to the parent DAG node under analysis which are formed by an edge of the graph. Once all possible separators are applied the DAG node analysis is complete and is marked as processed.

After completing the construction of the DAG, we start the optimization stage based on a cost analysis that is built as follows. First, we assign to each DAG node with zero out degrees the cost of evaluating its Schur complement. These DAG nodes are called sinks and correspond to individual finite elements. All DAG nodes with descendants are called parents. The cost assigned to each parent corresponds to the child DAG nodes of partitions with minimal cost, that is, for a DAG node with only sink children, the cost corresponding to the sum of the children in serial execution or the cost of the most expensive children in parallel as listed in Table 1. For parent DAG nodes with children which have out-node connections, the cost corresponds to the path to sinks with minimal cost. The optimization began by assigning the cost to each sink. Then, for each parent whose children's cost has been fully processed, each partition is assigned a cost based on the separator used and the full cost of the submeshes. The partitions with minimum cost are kept and all other children DAG nodes are removed. The optimization procedure continues this way until reaching the root of the tree.

The dynamic programming algorithm itself for the case of sequential solver optimization has been described in conference proceedings paper [31]. We also refer to [32] for examples on the usage of the GALOIS solver over these trees. In this paper we focus on the trees constructed by the heuristic algorithm presented in the next section, delivering a computational cost similar to the one obtained by the quasi-optimal trees obtained with dynamic programming. Below, we provide a short summary of a quasi-optimal elimination tree found by our dynamic programming algorithm for meshes with point and edge singularities with three refinement levels.

3.1. Description of Quasi-Optimal Orderings Based on Dynamic Programming

3.1.1. Point Singularity in Sequential Execution. The dynamic programming optimization algorithm sequential execution for a mesh with a point singularity results in the elimination

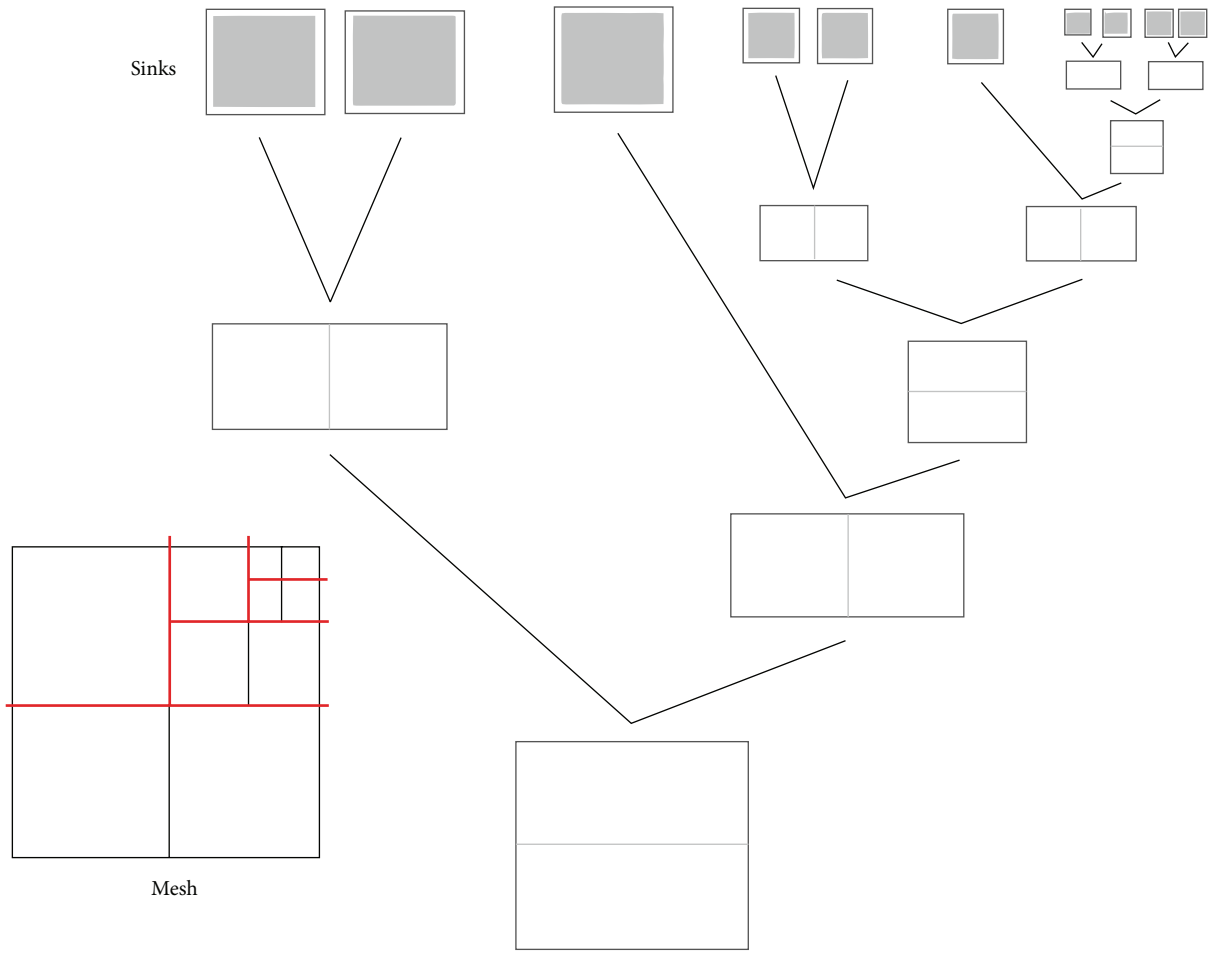

Figure 11: Optimal elimination tree for point singularity for a sequential solver.

tree presented in Figure 11. The pattern of the elimination tree is invariant with the number of refinement levels. The tree follows level-by-level elimination pattern.

3.1.2. Point Singularity for a Parallel Solver. This time we have executed the dynamic programming optimization algorithm for the point singularity with the cost function designed for a parallel shared-memory solver. The obtained elimination tree is presented in Figure 12. The tree is no longer cutting layers, one by one, but rather trying to balance the load over each partition. However, the pattern is not invariant with refinement level.

3.1.3. Edge Singularity in Sequential Execution. We switch now to an edge singularity. For the dynamic optimization algorithm optimizing for sequential solution on a mesh with an edge singularity the optimization procedure resulted in the elimination tree presented in Figure 13. The general elimination tree pattern is invariant with the level of refinements. The optimizer cuts the largest two elements, and then it partitions the remaining mesh into two parts. The optimizer does this sequence recursively until the leaves.

3.1.4. Edge Singularity for a Parallel Solver. As before, switching the cost estimates for parallel execution leads to a tree which is not invariant with the refinement level. The quasi-optimal tree is presented in Figure 14. This result of the optimization on the bisection sequence scales to optimize load balancing for the selected separators.

4. Heuristic Algorithm for Construction of the Elimination Trees

The dynamic programming strategy algorithm described above to check the computational cost of the multifrontal solver resulting from elimination trees is obtained by recursive partitioning of the computational mesh along straight lines. There are many such elimination trees for a single mesh, and the dynamic programming algorithm can only be used as a learning tool, since the cost of finding the elimination tree resulting in minimal cost for a large mesh is actually orders of magnitude larger than the cost of the solution itself. The dynamic programming algorithm allowed us to construct a heuristic algorithm that provides similar elimination trees in $\mathcal{O}(N_e \log(N_e))$ computational cost, where N_e is the number of elements of the mesh. Thus, we propose an *area and neighbors* algorithm for construction elimination trees for multifrontal solvers for h refined grids.

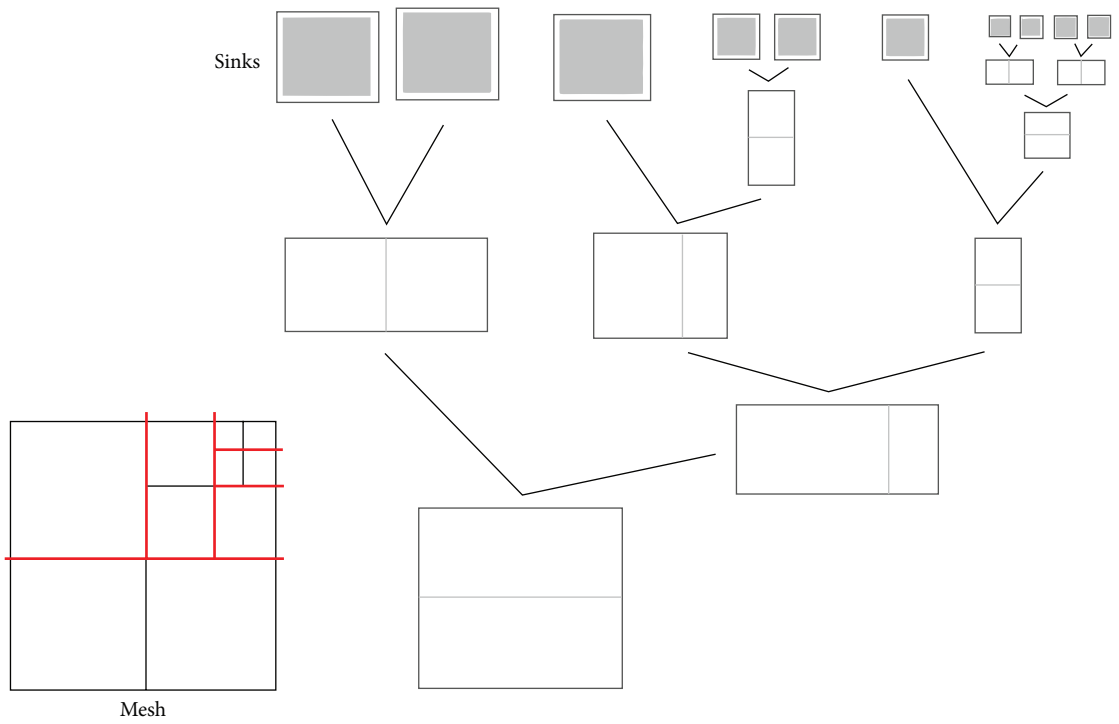

Figure 12: Optimal elimination tree for point singularity for the solver working in parallel shared-memory mode.

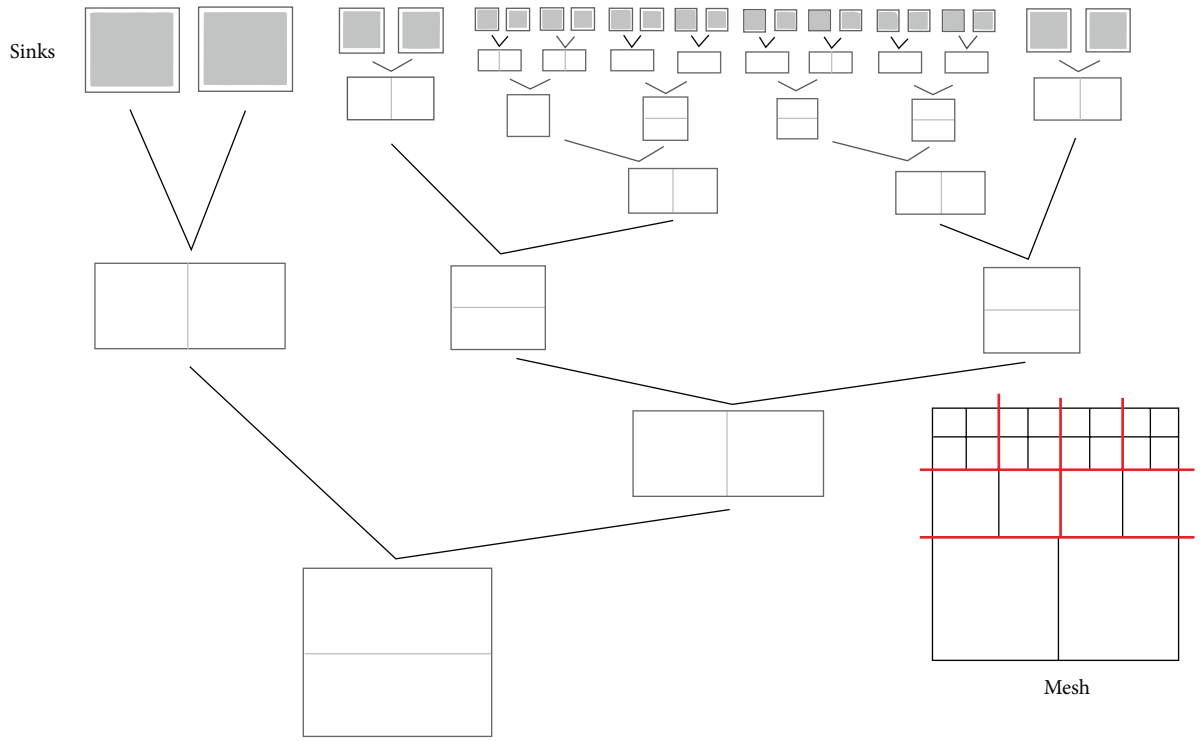

Figure 13: Optimal elimination tree for edge singularity for the solver working in sequential mode.

The heuristic algorithm can be utilized under the following assumptions.

(i) The computational mesh is two-dimensional, and it is obtained by performing a sequence of isotropic refinements from initial structured regular mesh with rectangular elements.

(ii) When constructing the h refined mesh, only isotropic h refinement is allowed. In other words, selected

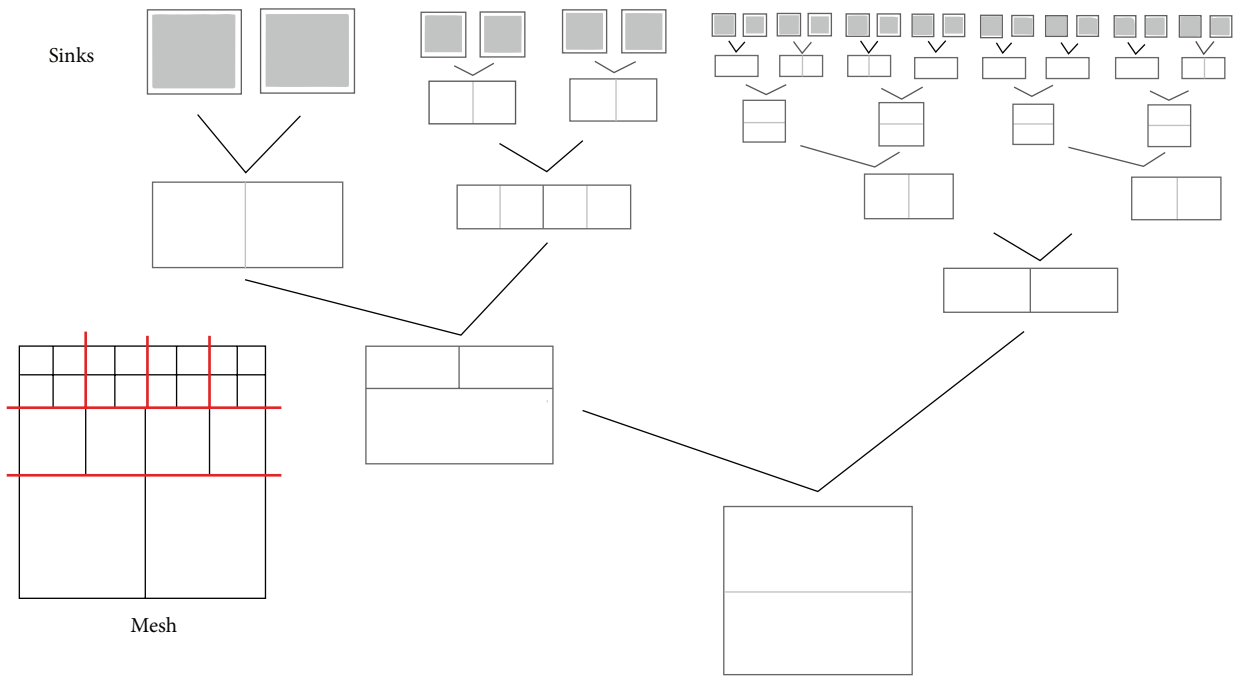

Figure 14: Optimal elimination tree for edge singularity for the solver working in parallel shared-memory mode.

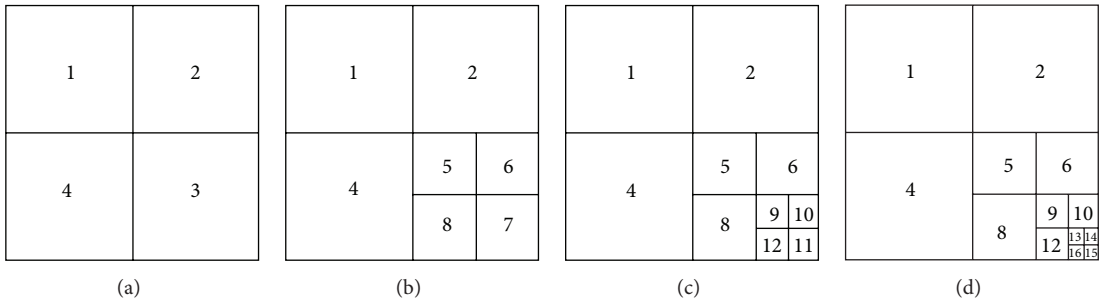

Figure 15: Generation of numbering of elements over the mesh h refined towards point singularity, refined in four steps denoted on panels (a), (b), (c), and (d).

rectangular elements are always broken into four smaller son elements.

(iii) The elements in the initial mesh are numbered, and their numbers are topologically sorted, left to right, row by row.

(iv) Each element of the initial mesh has assigned refinement level equal to 1.

(v) When the adaptive algorithm breaks an element into four son elements, the refinement level of all son elements is equal to the refinement level of the parent element plus one.

(vi) The area of each element is defined as $1/2^{2*\text{refinement level}}$.

(vii) When the adaptive algorithm breaks an element into four son elements, they get the new numbers in the global numbering of elements, and their numbers increase clockwise.

(viii) The mesh fulfills the 1-irregularity rule, telling that an element can be broken only once without breaking an adjacent large element.

(ix) When there are one element on one side of an edge and two elements on the other side of the mesh, we call this common edge a *constrained edge*.

(x) When we compute the maximum number of common edges between two adjacent patches of elements in the mesh, we count each constrained edge as one.

The assumptions listed above correspond to the computational grids generated by two-dimensional hp adaptive finite element method called $hp2d$ described in [3]. However in this paper we consider only uniform polynomial order of approximation $p = 2$ (only h refinement with uniform $p = 2$). The exemplary process of generation of the numbering for point and edge singularity is presented in Figures 15 and 16. The corresponding numbering of refinement levels for particular elements is reported in Tables 2 and 3.

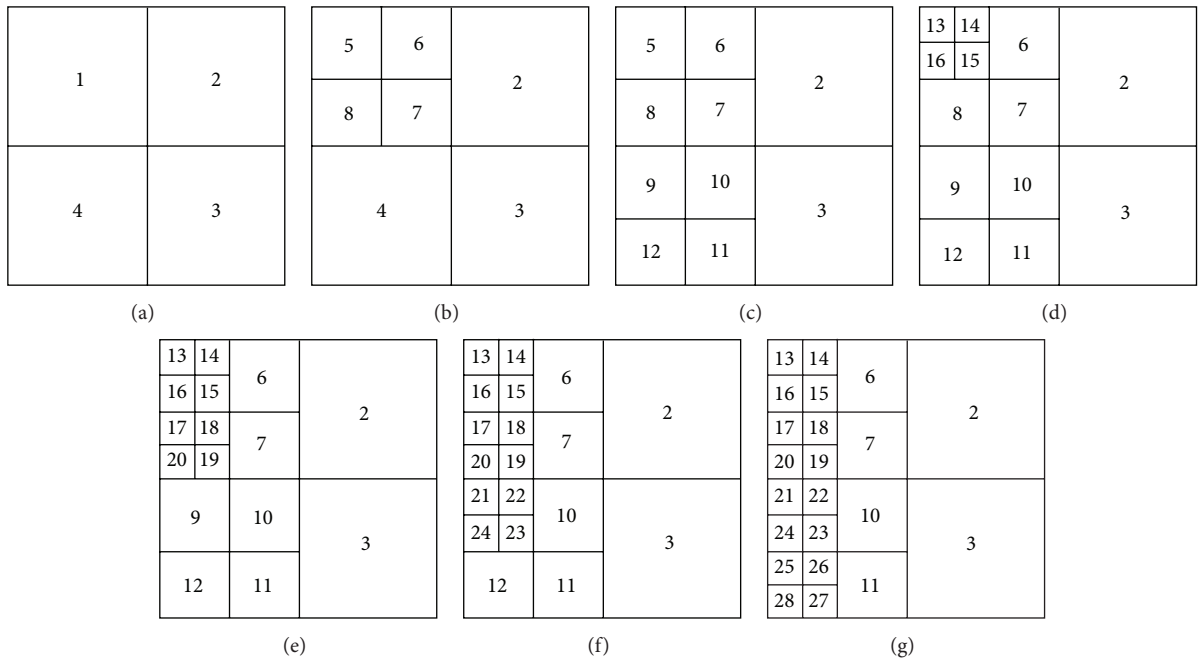

Figure 16: Generation of numbering of elements over the mesh h refined towards edge singularity, refined in seven steps denoted on panels (a), (b), (c), (d), (e), (f), and (g).

Table 2: Refinement levels for particular elements from mesh with point singularity.

Refinement level	Area	Elements
1	1	1, 2, 4
2	1/4	5, 6, 8
3	1/16	9, 10, 11, 12
4	1/256	13, 14, 15, 16

Table 3: Refinement levels for particular elements from mesh with edge singularity.

Refinement level	Area	Elements
1	1	2, 3
2	1/4	6, 7, 10, 11
3	1/16	13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28

The area and neighbors algorithm executed on the computational mesh with N_e elements can be summarized as shown in Algorithm 1.

Notice the following remarks.

(i) Thanks to our definition of numbering of elements, the initial forest sorted according to the numbering of elements is also sorted according to elements area.

(ii) When we rotate the point or edge singularity by 90, 180, or 270 degrees our numbering algorithm followed by the area and neighbors algorithm will generate similar quasi-optimal elimination trees resulting in a similar number of FLOPs.

(iii) In the general case, the algorithm may not deliver quasi-optimal elimination trees, since it is a heuristic algorithm, and the problem of construction of an optimal elimination tree is NP-complete.

Let us illustrate the heuristic algorithm on the mesh examples with point and edge singularities. The meshes are presented in Figures 15 and 16. Let us focus first on the mesh with point singularity, as presented in Figure 17.

(1) The elements are sorted according to their numbering, in the reversed order, which is equivalent to sorting according to their area, from smallest to the largest.

(2) We select the subforest with smallest elements (16), (15), (14), (13) with minimum area equal to 1/256 (cf. Figure 17(a)).

(3) We find out that the maximum number of common edges between elements (16), (15), (14), (13) is equal to 1.

(4) The two pairs of elements (16, 15) and (14, 13) with minimum area and maximum number of common edges are formed. The elements (16), (15), (14), (13) are removed from the list; the newly created trees with pairs of elements are put at the beginning of the list (cf. Figure 17(b)).

(5) The subforest with trees built from smallest patches is now (16, 15), (14, 13) and the minimum area is equal to 2/256 (cf. Figure 17(b)).

(6) We find out that maximum number of common edges between patches (16, 15), (14, 13) is equal to 2.

```
(1)  Create forest of N_e one-element trees,
(2)    sorted according to the numbering of elements, from largest to smallest
(3)  Store list of neighbors for each tree
(4)  Compute area = 1/2^(2*refinement level) for each tree
(5)  repeat
(6)    Select sub-forest of elements with minimum area
(7)    Find maximum number of common edges between all pairs in the selected sub-forest
(8)    Loop through all pairs of trees (v,w) in the sub-forest with number of common edges
(9)       equal to the maximum number of common edges found
(10)     Create new root node r
(11)     Assign v and w as children nodes of r
(12)     Update area of r: area(r) = area(v) + area(w)
(13)     Update list of neighbors of r (merge the lists)
(14)     Add new tree r to the forest in such a way
(15)        that the forest is still sorted according to area of trees
(16)   end for
(17) until forest has one element
```

Algorithm 1

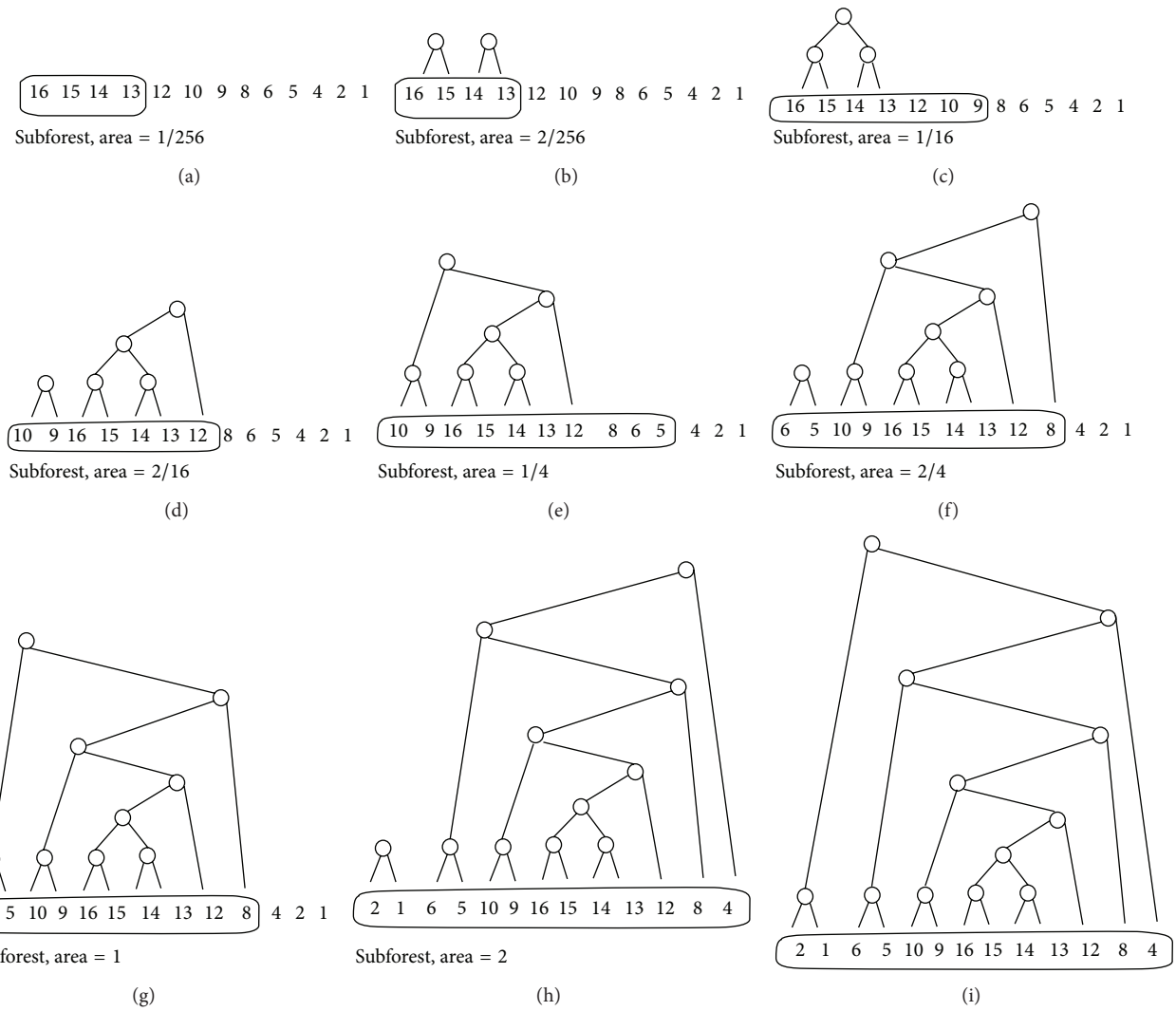

Figure 17: Particular steps of construction of the heuristic elimination tree for mesh with point singularity.

(7) We form a new tree from a pair ((16, 15), (14, 13)); the pairs (16, 15), (14, 13) are removed from the list, and the new tree ((16, 15), (14, 13)) is added at the beginning of the list (cf. Figure 17(c)).

(8) The subforest with trees built from smallest patches is now ((16, 15), (14, 13)), (12), (10), (9) and the minimum area is equal to 1/16 (cf. Figure 17(c)).

(9) We find out that maximum number of common edges between patches ((16, 15), (14, 13)), (12), (10), (9) is equal to 1.

(10) We form a new tree from a patch and element (((16, 15), (14, 13)), (12)) as well as a new tree out of elements ((10), (9)). They are removed from the list, and the new trees are added at the beginning of the list (cf. Figure 17(d)).

(11) The subforest with smallest patches is now ((10), (9)), (((16, 15), (14, 13)), (12)) and the minimum area is 2/16 (cf. Figure 17(d)).

(12) We find out that the maximum number of common edges between patches ((10), (9)), (((16, 15), (14, 13)), (12)) is equal to 2.

(13) We form a new tree from patches (((10), (9)), (((16, 15), (14, 13)), (12))). They are removed from the list, and the new tree is added at the beginning of the list (cf. Figure 17(e)).

(14) The subforest with smallest patches is now (((10), (9)), (((16, 15), (14, 13)), (12))), (8), (6), (5) and the minimum area is 1/4 (cf. Figure 17(e)).

(15) We find out that the maximum number of common edges between patches is equal to 1.

(16) We form new trees ((6), (5)) and ((((10), (9)), (((16, 15), (14, 13)), (12))), (8)). The original patches are removed from the list, and the new trees are added at the beginning of the list (cf. Figure 17(f)).

(17) The subforest with smallest patches is now ((6), (5), (((10), (9)), (((16, 15), (14, 13)), (12)))) and the minimum area is 2/4 (cf. Figure 17(f)).

(18) We find out that the maximum number of common edges between the patches is equal to 2.

(19) We form a new tree from patches ((6), (5), (((10), (9)), (((16, 15), (14, 13)), (12)))). They are removed from the list, and the new tree is added at the beginning of the list (cf. Figure 17(g)).

(20) The scenario is repeated recursively until one tree is formed.

Let us focus now on the case of the mesh with edge singularity, as presented in Figure 18. For the sake of simplicity, we present only a short description for this case.

(1) The elements are sorted according to their area.

(2) The eight pairs of elements with minimum area and maximum number of common edges are selected.

(3) The created eight patches of elements still have minimal areas. They are selected to form four new patches.

Table 4: Comparison of FLOPs for area and neighbors algorithm versus MUMPS with nested-dissection (METIS), approximate minimum fill (AMF), approximate minimum degree (AMD), quasi-approximate minimum degree (QAMD), PORD, and SCOTCH executed over the mesh with point singularity.

N	MUMPS + PORD, AMF, AMD, QAMD, SCOTCH	MUMPS + METIS	Area and neighbors
25	1120	1120	1145
37	2070	2070	1991
49	3020	3448	2837
61	3970	3970	3683
73	4920	5424	4529
85	5870	7282	5375
97	6820	8556	6221
109	7770	9830	7067
121	8720	10780	7913

(4) At this point, patches (5, 6) and (17, 18) have the same minimal area as the four created multielement patches. All these six patches are merged now into two new patches.

(5) At this point patches ((5, 6), (17, 18)) and the two multielement patches have minimal area and they are merged.

(6) Now we have to merge two patches that have minimal area.

(7) The situation is repeated again.

5. Tree Rotation Algorithm

We apply a tree rotation algorithm to improve the balance of the elimination tree [33, 34]. The algorithm browses the tree in breadth-first search order and performs a sequence of local rotations every time a branch is much deeper than the other one. For a detailed description of the tree rotation algorithm please refer to [33, 34].

6. Numerical Results

6.1. Comparison of Computational Cost for Different Elimination Trees. In this section we compare the number of floating-point operations (FLOPs) of our heuristic elimination trees constructed for the meshes with point and edge singularities, with alternative ordering algorithms available through MUMPS. In particular, we compare the number of FLOPs of our GALOIS solver based on our heuristic elimination trees with number of FLOPs required by MUMPS using the ordering provided by approximate minimum fill (AMF), approximate minimum degree (AMD), SCOTCH, PORD, METIS, and AMD with quasi-row detection. We utilize sequential version of our GALOIS solver, with heuristic elimination tree, without tree rotations. The results of the comparison are presented in Tables 4 and 5 as well as in Figures 19 and 20 using log-log scale. In all the cases, our

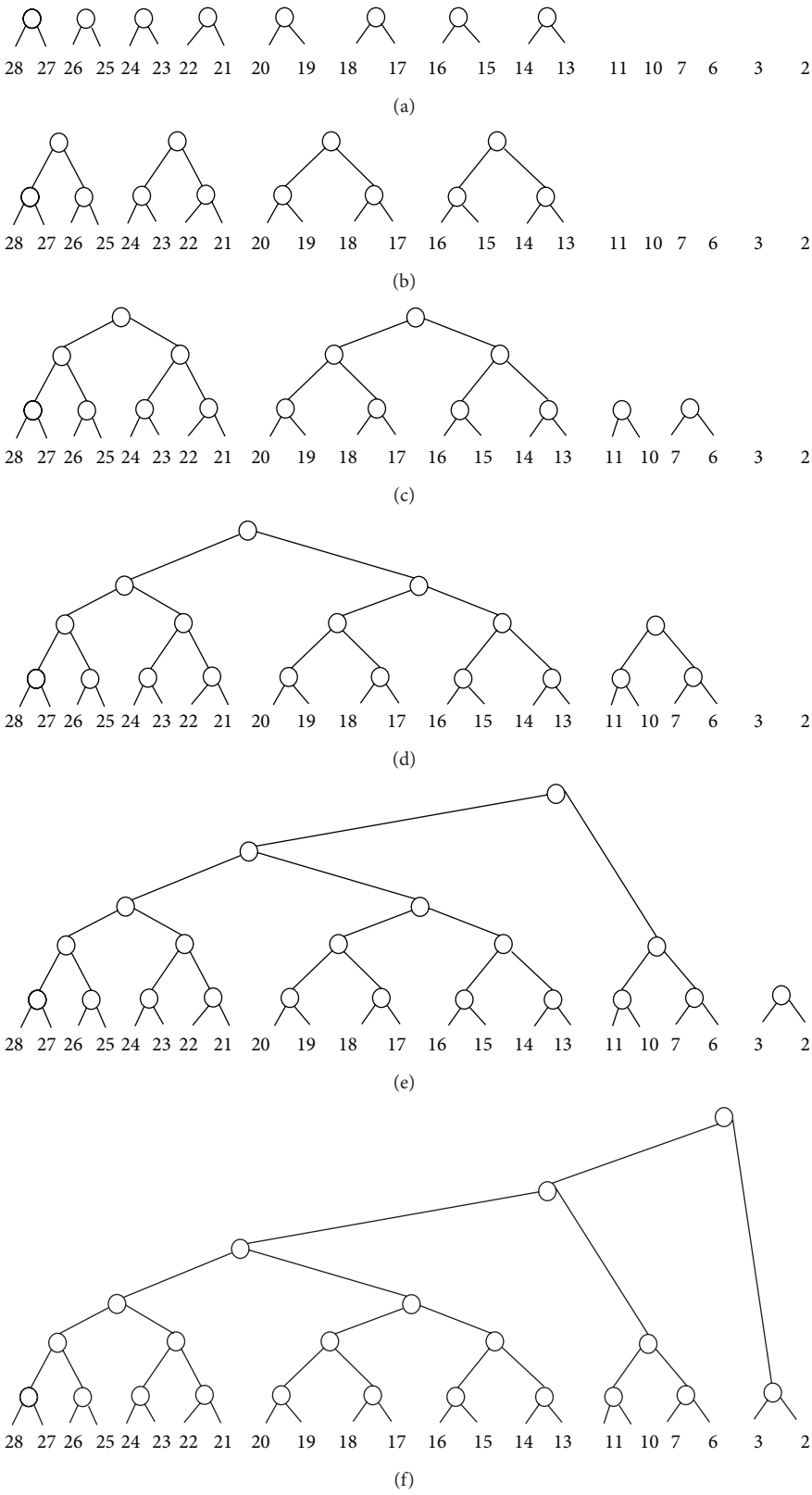

Figure 18: Particular steps (a)–(f) for construction of the heuristic elimination tree for mesh with edge singularity.

Table 5: Comparison of FLOPs for area and neighbors algorithm versus MUMPS with nested-dissection (METIS), approximate minimum fill (AMF), approximate minimum degree (AMD), quasi-approximate minimum degree (QAMD), PORD, and SCOTCH executed over the mesh with edge singularity.

N	MUMPS + PORD, AMF, AMD, QAMD, SCOTCH	MUMPS + METIS	Area and neighbors
25	1120	1120	1145
51	3527	3831	3342
101	10100	10530	9827
199	28970	35710	26204
393	77740	81450	64827
779	197400	204100	153510
1549	486700	513400	348519
3087	1155000	1223000	765176
6161	2656000	2766000	1717203

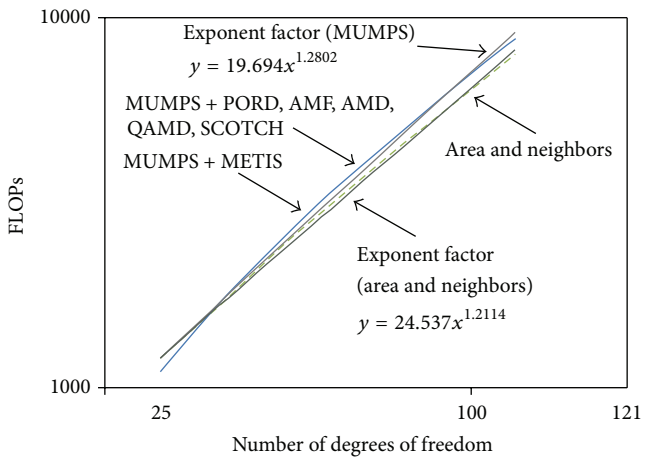

Figure 19: Comparison of the number of FLOPs for the point singularity. No visible difference between MUMPS + METIS and MUMPS + other orderings.

heuristic algorithms deliver a lower number of FLOPs. We also estimated the exponent factors for both algorithms and obtain $\mathcal{O}(N^{1.2114})$ for our algorithm and $\mathcal{O}(N^{1.2802})$ for the best alternative ordering for the point singularity, as well as $\mathcal{O}(N^{1.323})$ for our algorithm and $\mathcal{O}(N^{1.4122})$ for the best alternative ordering for the edge singularity.

6.2. Comparison of Dynamic Programming and Heuristic Algorithms. In this section we analyze the computational performance of the quasi-optimal and heuristic elimination trees in actual implementations. To compare the methods we use the execution time. This allows us to account for FLOPs and memory transfers in this comparison. We schedule the resulting sequences of graph-grammar productions using GALOIS to obtain an optimized performance both in serial

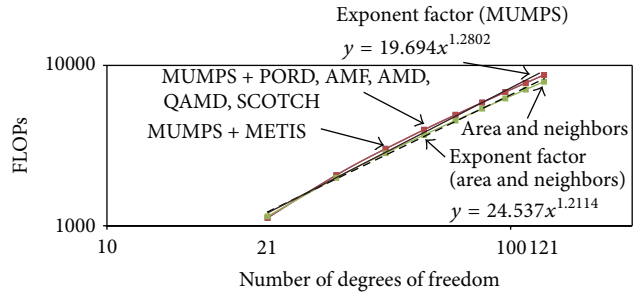

Figure 20: Comparison of the number of FLOPs for the edge singularity.

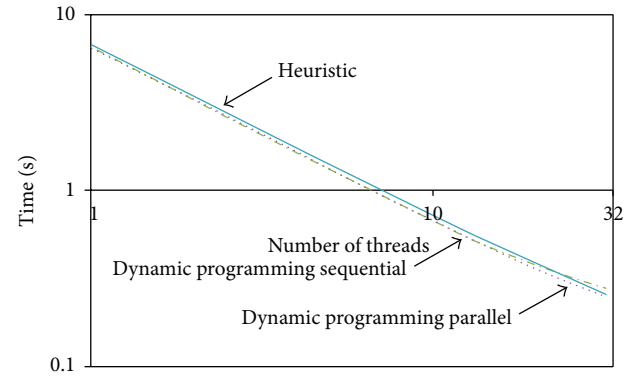

Figure 21: Comparison of the execution times between quasi-optimal and heuristic elimination trees for a mesh with an edge singularity.

and in parallel execution. We compare the solver execution time over three elimination trees:

(i) the dynamic programming tree using the sequential cost function, without tree rotations,

(ii) the dynamic programming tree using the parallel cost function, without tree rotations,

(iii) the heuristic elimination tree, without tree rotations.

The tests are performed on a GILBERT machine with 64 cores. We focus on the mesh with point and edge singularity. The solvers use an increasing number of threads, from 1, 2, 4, 8, 16 to 32. The results are presented in Figure 21. The results indicate that the heuristic algorithms result in similar execution times like the quasi-optimal trees generated by the dynamic programming algorithm.

We can draw the following conclusions from the performed numerical experiments.

(i) Both dynamic programming orderings provide similar execution times; the parallel ordering becomes slightly faster for a large number of threads.

(ii) Our heuristic ordering provides a similar execution time to that of the dynamic programming orderings.

(iii) We conclude that we can safely use the elimination trees generated by the heuristic ordering instead of

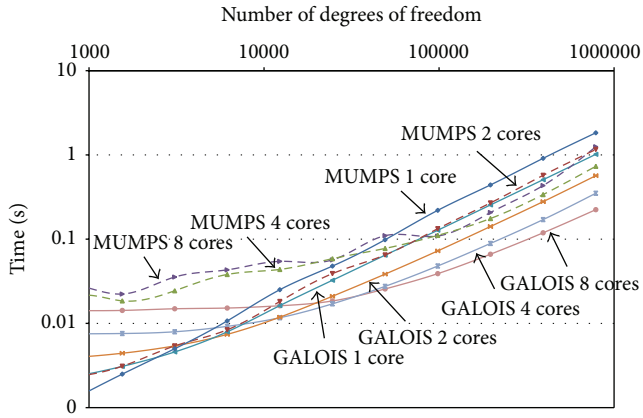

Figure 22: Log-log scale comparison of the execution times of the MUMPS and GALOIS solvers for different numbers of threads, for different numbers of refinement levels, for the mesh with edge singularity.

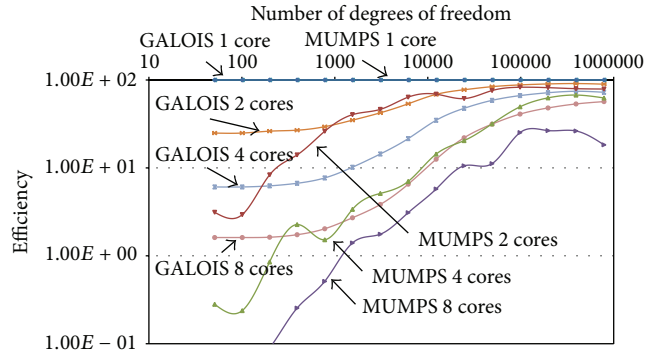

Figure 23: Comparison of the efficiency of the MUMPS and GALOIS solvers for different numbers of threads, for different numbers of refinement levels, for the mesh with edge singularity. The log-log scale is utilized.

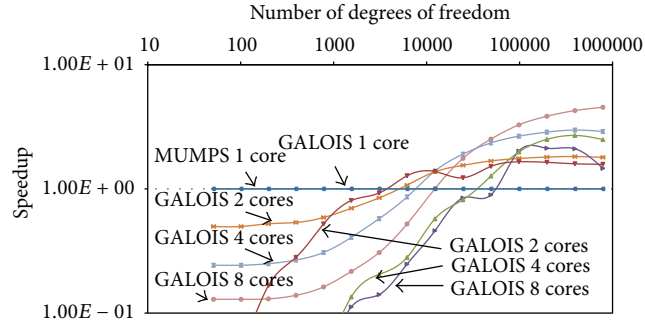

Figure 24: Comparison of the speedup of the MUMPS and GALOIS solvers for different numbers of threads, for different numbers of refinement levels, for the mesh with edge singularity. The log-log scale is utilized.

expensive dynamic programming orderings; however, we will continue comparison of these two approaches for other kinds of meshes and in 3D in our future work.

6.3. Comparison with MUMPS. In this example we compare our GALOIS solver with heuristic trees without the rotation algorithm against MUMPS with AMD ordering provided by METIS, since AMD results in a minimum number of FLOPs. We compile our GALOIS based solver with the gcc-4.8.0 compiler. Our solver *does not use any optimized numerical libraries* and is a pure C code. We compare against MUMPS solver version 4.10.0 compiled with gfortran-4.8.0 and linked with metis-4.0.3, atlas-3.10.1, LAPACK-3.4.2, and ScaLAPACK-2.0.2. In MUMPS we utilize Cholesky factorization (the problem is symmetric, positive definite). We use a simple model problem, the heat transfer equation. In our solver we utilize LU factorization. We compare execution times as well as parallel efficiency and speedup. The tests are performed on a single node of ATARI Linux cluster with 16-core Intel Xeon CPU, with 2.4 GHz, total 16 GB RAM. The point singularity results in very small computational meshes, and the comparison of parallel solvers makes no sense there. In the following experiments we focus on the mesh with an edge singularity. The comparison of the execution times for an edge singularity is presented in Figure 22. The comparison of the parallel efficiency for an edge singularity is presented in Figure 23. The comparison of the parallel speedup for an edge singularity is presented in Figure 24.

We can draw the following conclusions from the performed numerical experiments.

(i) For small problem sizes (less than 10000 degrees of freedom) MUMPS and GALOIS solvers behave like for point singularity case.

(ii) For larger problem sizes both MUMPS and GALOIS speed up when we increase the number of cores.

(iii) For larger problem sizes GALOIS scales well up to 8 cores; however MUMPS scales well up to 4 cores only.

(iv) For larger problems the GALOIS solver with any thread number outperforms multithreaded MUMPS.

6.4. Tests Using Tree Rotation Algorithm. In this section we compare execution times for the meshes with point and edge singularities, using our heuristic elimination trees, before and after execution of the rotation algorithm. The tests are performed on a GILBERT machine with 64 cores. From these experiments we do not observe a significant improvement on the performance of the proposed heuristic elimination trees. That is, for some instances rotation improves performance while in others it does not. In all cases the performance is comparable.

7. Conclusions

In this paper we discussed a dynamic programming algorithm for finding quasi-optimal elimination trees for two-dimensional grids with singularities. We performed the optimization for the cost function reflecting sequential and parallel execution. We introduce a heuristic algorithm to construct the elimination trees. These heuristic elimination trees have similar performance to that of the quasi-optimal trees obtained with dynamic programming. The quasi-optimal

and heuristic elimination trees are compared against state-of-the-art solvers implemented in popular libraries such as MUMPS. We compare number of FLOPs for each solver (using execution times as proxies for MUMPS) and show that our elimination trees deliver better computational costs in terms of the exponent factors. We also executed the algorithm for rotation of our elimination trees to check if they are well balanced and well suited to parallel computations. We verified experimentally that the tree rotation may improve the execution time of the multifrontal solver working with our elimination trees, but this is not always the case. Finally our elimination trees were expressed as graph-grammar productions and implemented in our graph-grammar-based solver using GALOIS scheduler. The solver is compared with MUMPS. We show that the graph-grammar-based solver outperforms MUMPS for large problems and provides comparable execution times for small ones.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The work of Anna Paszyńska, Maciej Paszyński, Konrad Jopek, Maciej Woźniak, Damian Goik, and Piotr Gurgul was supported by the Polish National Science Center Grants nos. DEC-2012/07/B/ST6/01229, DEC-2011/03/B/ST6/01393, and DEC-2012/06/M/ST1/00363. The work of Keshav Pingali and Andrew Lenerth was supported by NSF Grants CCF 1337281, CCF 1218568, ACI 1216701, and CNS 1064956. Donald Nguyen was supported by a DOE Sandia Fellowship. The work of Hassan AbouEisha, Mikhail Moskkov, and Victor Manuel Calo and visits of Anna Paszyńska, Maciej Paszyński, and Maciej Woźniak at KAUST were supported by the Center for Numerical Porous Media at KAUST. The visits of Maciej Paszyński at ICES were supported by J. T. Oden Research Faculty Fellowship.

References

[1] K. Pingali, D. Nguyen, M. Kulkarni et al., "The tao of parallelism in algorithms," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11)*, pp. 12–25, June 2011.

[2] M. Yannakakis, "Computing the minimum fill-in is NP-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[3] L. Demkowicz, *Computing with hp-Adaptive Finite Elements, Volume I: One and Two Dimensional Elliptic and Maxwell Problems*, Applied Mathematics and Nonlinear Science, Chapman & Hall/CRC Press, 2006.

[4] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear equations," *ACM Transactions on Mathematical Software*, vol. 9, no. 3, pp. 302–325, 1983.

[5] I. S. Duff and J. K. Reid, "The multifrontal solution of unsymmetric sets of linear equations," *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 3, pp. 633–641, 1984.

[6] B. M. Irons, "A frontal solution program for finite element analysis," *International Journal for Numerical Methods in Engineering*, vol. 2, no. 1, pp. 5–32, 1970.

[7] G. Karypis and V. Kumar, "METIS—unstructured graph partitioning and sparse matrix ordering system, version 2.0," Tech. Rep., 1995.

[8] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[9] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis & Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[10] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, "Multifrontal parallel distributed symmetric and unsymmetric solvers," *Computer Methods in Applied Mechanics and Engineering*, vol. 184, no. 2–4, pp. 501–520, 2000.

[11] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[12] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, "Hybrid scheduling for the parallel solution of linear systems," *Computer Methods in Applied Mechanics and Engineering*, vol. 2, no. 32, pp. 136–156, 2001.

[13] A. Paszyńska, M. Paszyński, and E. Grabska, "Graph transformations for modeling hp-adaptive finite element method with mixed triangular and rectangular elements," in *Computational Science—ICCS 2009*, vol. 5545 of *Lecture Notes in Computer Science*, pp. 875–884, Springer, Berlin, Germany, 2009.

[14] A. Paszynska, M. Paszynski, and A. Grabska, "Graph transformations for modeling hp-adaptive finite element method with triangular elements," in *Computational Science—ICCS 2008*, vol. 5103 of *Lecture Notes in Computer Science*, pp. 604–613, 2008.

[15] M. Paszynski, "On the parallelization of self-adaptive hp-finite element methods part I. Composite programmable graph grammar model," *Fundamenta Informaticae*, vol. 93, no. 4, pp. 411–434, 2009.

[16] M. Paszynski, "On the parallelization of self-adaptive *hp*-finite element methods. II. Partitioning communication agglomeration mapping (PCAM) analysis," *Fundamenta Informaticae*, vol. 93, no. 4, pp. 435–457, 2009.

[17] P. Obrok, P. Pierzchala, A. Szymczak, and M. Paszynski, "Graph grammar-based multi-thread multi-frontal parallel solver with trace theory-based scheduler," *Procedia Computer Science*, vol. 1, no. 1, pp. 1993–2001, 2010.

[18] M. Paszyński and R. Schaefer, "Graph grammar-driven parallel partial differential equation solver," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 9, pp. 1063–1097, 2010.

[19] E. Grabska, "Theoretical concepts of graphical modeling. Part two: CP-graph grammars and languages," *Machine Graphics and Vision*, vol. 2, no. 2, pp. 149–178, 1993.

[20] A. Habel and H.-J. Kreowski, "May we introduce to you: hyperedge replacement," in *Graph-Grammars and Their Application to Computer Science*, vol. 291 of *Lecture Notes in Computer Science*, pp. 15–26, Springer, Berlin, Germany, 1987.

[21] A. Habel and H.-J. Kreowski, "Some structural aspects of hypergraph languages generated by hyperedge replacement," in *STACS 87*, vol. 247 of *Lecture Notes in Computer Science*, pp. 207–219, Springer, 1987.

[22] G. Ślusarczyk and A. Paszyńska, "Hypergraph grammars in hp-adaptive finite element method," *Procedia Computer Science*, vol. 18, pp. 1545–1554, 2012.

[23] V. Diekert and G. Rozenberg, *The Book of Traces*, World Scientific, River Edge, NJ, USA, 1995.

[24] P. Gurgul, "A linear complexity direct solver for h-adaptive grids with point singularities," *Procedia Computer Science*, vol. 29, pp. 1090–1099, 2014.

[25] M. Paszyński, D. Pardo, and A. Paszyńska, "Parallel multi-frontal solver for p adaptive finite element modeling of multi-physics computational problems," *Journal of Computational Science*, vol. 1, no. 1, pp. 48–54, 2010.

[26] B. Barabasz, E. Gajda-Zagórska, S. Migórski, M. Paszyński, R. Schaefer, and M. Smołka, "A hybrid algorithm for solving inverse problems in elasticity," *International Journal of Applied Mathematics and Computer Science*, vol. 24, no. 4, pp. 865–886, 2014.

[27] E. Gajda-Zagórska, R. Schaefer, M. Smołka, M. Paszyński, and D. Pardo, "A hybrid method for inversion of 3D DC resistivity logging measurements," *Natural Computing*, 2014.

[28] D. Pardo, L. Demkowicz, C. Torres-Verdinn, and M. Paszynski, "Two-dimensional high-accuracy simulation of resistivity logging-while-drilling (LWD) measurements using a self-adaptive goal-oriented *hp* finite element method," *SIAM Journal on Applied Mathematics*, vol. 66, no. 6, pp. 2085–2106, 2006.

[29] L. Demkowicz, P. Gatto, J. Kurtz et al., "Modeling of bone conduction of sound in the human head using *hp*-finite elements: code design and verification," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 21-22, pp. 1757–1773, 2011.

[30] P. Matuszyk and M. Paszyński, "Fully automatic hp adaptive finite element method for the Stokes problem in two dimensions," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, no. 51-52, pp. 4549–4558, 2008.

[31] H. AbouEisha, M. Moshkov, V. Calo, M. Paszynski, D. Goik, and K. Jopek, "Dynamic programming algorithm for generation of optimal elimination trees for multi-frontal direct solver over h-refined grids," *Procedia Computer Science*, vol. 29, pp. 947–959, 2014.

[32] D. Goik, K. Jopek, M. Paszyński, A. Lenharth, D. Nguyen, and K. Pingali, "Graph grammar based multi-thread multi-frontal direct solver with galois scheduler," *Procedia Computer Science*, vol. 29, pp. 960–969, 2014.

[33] M. Fredrik, "An algorithm for computing an elimination tree of minimum height for a tree," in *Proceedings of the 2nd Meeting of the International Linear Algebra Society*, Lisbon, Portugal, 1992.

[34] M. Fredrik, "An algorithm for computing an elimination tree of minimum height for a tree," Tech. Rep. CS-91-59, University of Bergen, Bergen, Norway, 1991.