Graph grammar based multi-thread multi-frontal direct
solver with Galois scheduler

Damian Goik¹, Konrad Jopek¹, Maciej Paszyński¹, Andrew Lenharth², Donald
Nguyen², and Keshav Pingali²

¹ AGH University of Science and Technology, Krakow, Poland
² Institute for Computational and Engineering Science, The University of Texas in Austin USA

Abstract
In this paper, we present a multi-frontal solver algorithm for the adaptive finite element method expressed by graph grammar productions. The graph grammar productions construct first the binary elimination tree, and then process frontal matrices stored in distributed manner in nodes of the elimination tree. The solver is specialized for a class of one, two and three dimensional $h$ refined meshes whose elimination tree has a regular structure. In particular, this class contains all one dimensional grids, two and three dimensional grids refined towards point singularities, two dimensional grids refined in an anisotropic way towards edge singularity as well as three dimensional grids refined in an anisotropic way towards edge or face singularities. In all these cases, the structure of the elimination tree and the structure of the frontal matrices are similar. The solver is implemented within the Galois environment, which allows parallel execution of graph grammar productions. We also compare the performance of the Galois implementation of our graph grammar based solver with the MUMPS solver.

Keywords: $h$ adaptive finite element method, multi-frontal direct solver, graph grammar, Galois environment

1 Introduction
In this paper we present a graph grammar based multi-frontal direct solver algorithm for adaptive finite element method computations.

The multi-frontal solver is a state-of-the-art direct solver algorithm for solving sparse systems of linear equations resulting from finite element method simulations. The multi-frontal solver is based on a generalization of the frontal solver algorithm [8]. The frontal solver algorithm browses the computational mesh element by element, assembles element matrices one by one into a single element frontal matrix, and eliminates fully assembled rows. The rows in the frontal matrix are associated with mesh nodes, and the fully assembled rows are associated with those nodes whose elements have been already assembled into the single frontal matrix. The multi-frontal algorithm proposed by [5, 6, 7] generalizes the idea of the frontal solver into several frontal...
matrices browsing finite elements, assembling element matrices into several frontal matrices, and eliminating fully assembled rows. The multi-frontal algorithm is controlled by an elimination tree [11]. The tree is usually constructed by using graph partitioning algorithms like nested dissections implemented within METIS [10] library. Finite elements are joined into pairs, and fully assembled unknowns are eliminated within frontal matrices associated to multiple branches of the tree. The process is repeated until the root of the elimination tree is reached. Finally, the common interface problem is solved, and partial backward substitutions are recursively called on the elimination tree.

The computational cost of the sequential multi-frontal solver algorithm over regular grids is $O(N)$, $O(N^{1.5})$ and $O(N^2)$ for one, two and three dimensional grids, respectively [2, 19]. Some modern direct solvers [22, 23] utilize the concept of $h$-matrices approximating the off-diagonal entries of the matrix, as well as special properties of PDEs in order to reduce the computational cost down to $O(N)$. However, these methods provide approximate solutions only, and they work only for special classes of PDEs.

There is a class of two and three dimensional $h$ refined grids [20], for which it is possible to construct elimination tree in such a way that the multi-frontal direct solver algorithm delivers linear computational cost. The linear cost in such the case is guaranteed by the structure of the computational grid and does not depend on the PDE being solved.

Graph grammars have been used for modeling of the mesh adaptation process [16, 15, 17, 18] as well as for the modeling of the one dimensional multi-frontal direct solver algorithm [14] with linear computational cost for sequential execution and logarithmic computational costs for parallel shared memory machines (assuming an unbounded number of processors). Graph grammars have also been used for modeling of the two dimensional multi-frontal direct solver algorithm [21, 13, 12], delivering computational performance comparable to the state-of-the-art MUMPS solver [1].

In this paper, we concentrate on the class of $h$ refined two and three dimensional grids for which, according to our recent research results [20], it is possible a linear computational cost multi-frontal direct solver algorithm. This is possible by a special construction of the elimination tree controlling the execution of the multi-frontal solver algorithm. This class includes two and three dimensional grids with point singularities, two dimensional grids refined in an anisotropic way towards edge singularity as well as three dimensional grids refined in an anisotropic way towards edge and face singularity. For all these grids, the structure of the elimination tree and the structure of the graph grammar generating and processing the tree is similar.

Given the graph grammar productions expressing the multi-frontal solver algorithm, we need an efficient scheduler algorithm that will identify the graph grammar productions that can be executed in parallel and will execute them efficiently in a shared memory parallel machine. For this purpose, we utilize the Galois environment [9], which has proven to be an efficient tool for scheduling of the solver algorithm on shared memory multi-core machines.

The numerical experiments presented in this paper have been executed over GILBERT shared memory machine from the Institute for Computational Engineering and Sciences (ICES). The machine has four Intel(R) Xeon(R) CPU E7-4860 with 2.27GHz, each one with 10 cores and additional 10 hyperthreading cores. The total available memory was 128 GB.

The paper concludes with numerical results presenting the scalability of the Galois based graph grammar solver as well as comparison to the state-of-the-art MUMPS solver [1] with nested dissection algorithm from METIS library [10].
2 Galois environment for parallel execution of graph grammar productions

The Galois environment [9] is a software environment to ease the development of parallel programs on shared-memory machines. Application developers are given a library of concurrent data structures, such as concurrent graphs and trees, and they code their algorithms by specifying (i) the graph grammar rules (known as operators in Galois terminology) and (ii) the schedules for applying these rules in parallel. Each graph grammar rule can be considered to be an action on a data structure; the data structure library and runtime system work together to ensure that the execution of each rule has transactional semantics. Many complex parallel applications including mesh generation, refinement, and partitioning algorithms, graph analytics algorithms, and n-body simulation algorithms have been written in this system, and shown to scale well on 512 core shared memory machines like the SGI Ultraviolet.

In the current system, this programming model is implemented in C++. Users write their programs in sequential C++ using iterators over work-lists containing data structure elements, and specify the scheduling policy for executing iterations in parallel. The body of the iterator is an implementation of the graph grammar rules. This iterator construct can be used to implement elimination tree traversals described in the following sections. The work-list initially contains the leaves of the tree. When all the children of a node have been processed, that node is added to the work-list. The iterator terminates when the work-list is empty.

3 Linear computational cost elimination trees for grids with singularities

Graph grammar productions generating the binary elimination tree and expressing the multifrontal direct solver algorithm for simple one dimensional problem were discussed in [14]. The mesh elements in that paper consist of two vertex nodes. The elimination tree presented there was a binary tree with leaves assigned to nodes of the one dimensional mesh, and each pair of leaves had associated with it one element with two vertex nodes. For the linear case considered in the paper, the element matrices had dimension of $2 \times 2$, and they were merged into $3 \times 3$ matrices with one central node fully assembled. This enabled the elimination of the central node, and the resulting Schur complement $2 \times 2$ matrices were merged at the parent level to form once again a $3 \times 3$ matrix with the central node again fully assembled. The process was repeated up to the root of the tree, where the fully assembled $3 \times 3$ system was formulated and solved. The entire procedure was followed by backward substitutions, executed from the root of the tree down to the leaves.

In this paper, we present a similar structure of the graph grammar that models two and three dimensional grids with some singularities. In our case, the two dimensional elements consist of four vertex node, four edge nodes, and one interior node. We utilize second order polynomials over element edges and interior; thus, each node is associated with only one corresponding row in the element frontal matrix. For the definition of the two dimensional shape functions, we refer to [3]. In the three dimensional case, the finite elements consists of eight vertex nodes, twelve edge nodes, six face nodes and one interior node. We also utilize second order polynomials over element edges, faces and the interior; thus, each node has only one corresponding row in the element frontal matrix. For the definition of the three dimensional shape functions, we refer to [4].
3.1 Two dimensional point singularity

For the two dimensional mesh with point singularity, we can still utilize the binary elimination tree, this time with pairs of leaves assigned to levels surrounding the point singularity, as in Figure 1. At the leaves, all the nodes from the interiors of each level, including the three interior nodes and four edge nodes, are fully assembled and can be eliminated, as in Figure 1. What remains are the two interfaces, two edge and three vertex nodes in our example. This is because the external interfaces have two constrained edges (see [3]), and thus both internal and external interfaces have same number of nodes, namely three vertex nodes and two edge nodes. The interfaces are merged at parent level of the elimination tree into a single matrix of size $3 \times 5 \times 3 \times 5$. At this point, we can eliminate the 5 nodes associated with the common interface, and we end up with $2 \times 5 \times 2 \times 5$ Schur complement matrices, as is illustrated in Figure 2. The process is repeated until the root node, where we have fully assembled $3 \times 5 \times 3 \times 5$ matrix to be solved, as in Figure 3.

We have replaced the graph grammar productions from [14] working with $2 \times 2$ and $3 \times 3$ matrices into the one working with $2 \times 5 \times 2 \times 5$ and $3 \times 5 \times 3 \times 5$ matrices, and implemented the resulting graph grammar in Galois. The resulting efficiency and speedup of the solver are presented in Figure 4.

![Binary elimination tree for 2D mesh with point singularity: elimination of interiors of layers](image)

3.2 Two dimensional anisotropic edge singularity

The case of the two dimensional mesh with anisotropic edge refinements is even simpler than the case of the point singularity. This is because the topology of this mesh is one dimensional,
and the only difference from the one dimensional grid is that the mesh has its thickness, namely the elements have four vertices, four edges and one interior instead of just two vertices as in pure one dimensional case, and there are two vertices and one edge node on the interface between elements instead of just one vertex on the interface in the pure one dimensional case. At the leaf nodes, the element matrices have a size of $4 + 5 \times 4 + 5$, and the two edges and one interior can be eliminated there. The resulting Schur complements are of size $2 \times (2 + 1) \times 2 \times (2 + 1)$ since they correspond to the four vertices and two edges from left and right side of the element. At parent level, these Schur complements are merged into one $3 \times (2 + 1) \times 3 \times (2 + 1)$ matrix, and the central fully assembled two vertices and one edge can be eliminated. The process is repeated until the root of the elimination tree.

The resulting efficiency and speedup of the Galois implementation of graph grammar solver are presented in Figure 5.

### 3.3 Three dimensional anisotropic face singularity

The case of the three dimensional anisotropic face singularity is similar to the previous case. The only difference is the number of nodes at leaf matrices (eight vertices, six faces, twelve edges and one interior), the number of fully assembled nodes at leaf vertices (four faces, four edges and one interior) and the number of nodes on the interface between elements (four vertices, four edges and one face on each side of an element). The Schur complements are of size $2 \times (4 + 4 + 1) \times 2 \times (4 + 4 + 1)$ and the merged matrices at parent level as well as the root problem are of size of $3 \times (4 + 4 + 1) \times 3 \times (4 + 4 + 1)$.

The resulting efficiency and speedup of the Galois implementation of graph grammar solver are presented in Figure 6.
3.4 Three dimensional anisotropic edge singularity

The three dimensional anisotropic edge singularity utilizes the same pattern as two dimensional point singularity; however, each layer has its element thickness. Thus, the leaf matrices contain three 3D elements with their interior nodes fully assembled. What remains are the two interfaces, each with two faces (namely two faces, seven edges and six vertices on both sides of each layer). In other words, the Schur complement matrices have size of $2 \times (2+7+6) \times 2 \times (2+7+6)$, and they are merged at parent levels into $3 \times (2+7+6) \times 3 \times (2+7+6)$ matrices having $2+7+6$
Figure 5: Two dimensional anisotropic edge singularity and the scalability of Galois solver

Figure 6: Three dimensional anisotropic face singularity and the scalability of Galois solver

fully assembled nodes.

The resulting efficiency and speedup of the Galois implementation of graph grammar solver are presented in Figure 7.

Figure 7: Three dimensional anisotropic edge singularity and the scalability of Galois solver
3.5 Three dimensional point singularity

Finally, the three dimensional point singularity has similar structure to the two dimensional point singularity, but each layer is now three dimensional and it contains seven three dimensional elements. The interfaces between layers contains three faces on both sides of each layer (notice the constrained faces on the external size of each layer, compare with [4], which implies same number of nodes on both internal and external interfaces of each layer). In other words, the Schur complement matrices contains nodes from three faces; thus, they are of the size $2 \times (3 + 9 + 7) \times 3 \times (3 + 9 + 7)$. The merged matrices at parent levels and at the root node are of size $3 \times (3 + 9 + 7) \times 3 \times (3 + 9 + 7)$, and they contain $3 + 9 + 7$ fully assembled nodes except at the root node where all the nodes are fully assembled.

The resulting efficiency and speedup of the Galois implementation of graph grammar solver are presented in Figure 8.

![Figure 8: Three dimensional point singularity and the scalability of Galois solver](image)

4 Comparison with MUMPS solver

We conclude with a comparison of the Galois solver on the edge singularity case with the state of the art MUMPS solver, shown in Figure 9. Even single thread Galois solver has better execution time as the state of the art MUMPS solver [1]; and when we increase number of threads, the Galois solver is one order of magnitude faster.

5 Conclusions

In this paper, we showed how the Galois environment can be efficiently utilized for implementation of graph grammars modeling the multi-frontal solver algorithm for a class of two and three dimensional grids refined towards singularities. In particular, for all considered cases, including the two and three dimensional point singularities, two dimensional anisotropic edge singularity, as well as three dimensional anisotropic edge and face singularities, the Galois solver delivered good scalability. We compared the execution time of the Galois solver with the state-of-the-art MUMPS solver for the most complicated case, the three dimensional point singularity, and we showed that the one thread Galois solver reproduced the execution time of the MUMPS solver and outperformed MUMPS by the order of magnitude when we increased the number of threads.
Future work will include the generalization of the Galois solver for grids with more complicated singularities, including isotropic edge in two and three dimensions, as well as isotropic face in three dimensions. We also plan to improve the Galois scheduler in order to allow for multi-level concurrency, in particular to be able to implement subtraction of different rows from the same matrix in concurrent, when processing multiple matrices at the same time. We also are considering developing a generalized solver for arbitrary $h$ refined grids, utilizing the library of template elimination trees for parts of the mesh refined in different way towards local singularities.

Acknowledgments The work presented in this paper has been supported by Polish MNiSW grants no. 2012/07/B/ST6/01229 (2D case) and 2012/06/M/ST1/00363 (3D case), and by NSF grants CCF 1337281, CCF 1218568, ACI 1216701, and CNS 1064956. Donald Nguyen was supported by a DOE Sandia Fellowship.

References


Graph grammar solver with Galois scheduler. M. Paszyński, A. Lenharth, D. Nguyen, K. Pingali


[12] Paszynski M. On the parallelization of selfadaptive hp-finite element methods part ii. partitioning communication agglomeration mapping (pcam) analysis. Fundamenta Informaticae, 93.


