Konrad Jopek
Maciej Paszyński
Anna Paszyńska
Muhammad Amber Hassan
Keshav Pingali

HYPERGRAPH GRAMMAR-BASED, MULTI-THREAD, MULTI-FRONTAL DIRECT SOLVER SCHEDULED IN PARALLEL GALOIS ENVIRONMENT

Abstract *In this paper, we analyze two-dimensional grids with point and edge singularities in order to develop an efficient parallel hypergraph grammar-based multi-frontal direct solver algorithm. We express these grids by a hypergraph. For these meshes, we define a sequence of hypergraph grammar productions expressing the construction of frontal matrices, eliminating fully assembled nodes, merging the resulting Schur complements, and repeating the process of elimination and merging until a single frontal matrix remains. The dependency relationship between hypergraph grammar productions is analyzed, and a dependency graph is plotted (which is equivalent to the elimination tree of a multi-frontal solver algorithm). We utilize a classical multi-frontal solver algorithm; the hypergraph grammar productions allow us to construct an efficient elimination tree based on the graph representation of the computational mesh (not the global matrix itself). The hypergraph grammar productions are assigned to nodes on a dependency graph, and they are implemented as tasks in the GALOIS parallel environment and scheduled according to the developed dependency graph over the shared memory parallel machine. We show that our hypergraph grammar-based solver outperforms the parallel MUMPS solver.*

Keywords graph grammar, direct solver, h adaptive finite element method, GALOIS

Citation Computer Science 20(1) 2019: 27–55

1. Introduction

The multi-frontal solver [11–13] is the state-of-the-art algorithm for the LU factorization of matrices resulting from adaptive finite element method computations [8]. The input to classical multi-frontal solvers like the MUMPS solver [3–5] is the global matrix obtained by assembling an element's frontal matrices. The computational complexity of the multi-frontal solver algorithm depends on the quality of the constructed elimination tree [18]. In this paper, we show that the parallel multi-frontal solver algorithm can be expressed by hypergraph grammar productions, and the elimination tree can be expressed by a dependency graph between them. The graph grammar model has already been utilized for both expressing the adaptive mesh transformations [26–28, 30, 31] and the multi-frontal solver algorithm itself [20, 32]. However, we utilized composition graph grammars (CP-graph grammars) [26, 27, 30, 31] in our previous work; in this paper, we propose the use of the hypergraph grammar [36]. The hypergraph grammar was originally introduced in [15, 16] and instantiated to model the mesh transformations in [36].

The CP-graph model allows us to reflect on the history of mesh refinements within particular elements. This is an advantage in the case of a solver performing a reutilization of LU factorizations or in the case of adaptation with non-stationary problems requiring unrefinements. The disadvantage of this approach is an expensive search for adjacent elements. The hypergraph model does not allow us to remember the history of the refinements, but the graph representing the finite element mesh is much smaller than the corresponding CP-graph. The advantage of this approach is the possibility of planning the global ordering and scheduling algorithms not bounded to single finite element refinements. For example, it is possible to eliminate level by level in all elements surrounding the singularity area in the case of point singularity located in the common vertex of four elements. In the CP-graph manner, the solver would process the refined elements one by one, from the leaves up to the initial mesh elements level. The cost of such a level-by-level elimination that is natural for hypergraph grammars is cheaper than the elimination of refinement levels inside one element until the initial element level is reached and then repeating the elimination in the neighboring element in the CP-graph-grammar manner.

In this paper, we utilize the classical version of the multi-frontal solver algorithm without any modern modifications (like the use of H-matrices, for example [35]). We employ a hypergraph grammar model to construct an elimination tree, which results in the good performance of the solver.

Namely, we consider hypergraph models of grids with point and edge singularity. We define a sequence of hypergraph grammar productions over the hypergraphs representing the construction of frontal matrices, eliminating the fully assembled nodes and merging the remaining Schur complement matrices recursively up to the moment when there is only one frontal matrix left. We analyze the dependency relationships between these hypergraph grammar productions, and we plot a dependency graph for them. The dependency graph is equivalent to the elimination tree, and the operations

performed by the multi-frontal solver algorithm over the elimination tree are expressed by hypergraph grammar productions.

The multi-frontal solver expressed by the CP-graph grammar productions described in [20] concerns the one-dimensional finite difference method. It doesn't include the ordering generation necessary for two- or three-dimensional grids. The approach presented there can be generalized to two- or three-dimensional model grids with singularities [14] constructed by performing refinements from a single initial mesh element. This, however, does not allow for the straightforward processing of refinements surrounded by several initial mesh elements in a manner that allows the elements to be processed level by level. The CP-graph, grammar-based, multi-frontal solver described in [32] concerns the two-dimensional adaptive finite element method. The CP-graph grammars construct the CP-graph corresponding to the finite element mesh, remembering the history of the refinement. Each element that was h-refined is represented in the graph by an interior node and edges connecting with nodes corresponding to the smaller son elements. The CP-graph grammar productions modeling the solver algorithm must travel the element refinement trees from the leaves up to the root of the tree. These productions are complex, and it is hard to propose general productions that are independent of the depth of the refinement trees. In this paper, we propose a hypergraph grammar that generates graphs without the history of the refinements; thus, hypergraph grammar productions do not need to travel multiple levels of the graph – it is possible to process the elements surrounding the singularities level by level.

Having the solver algorithm expressed by hypergraph grammar productions and its elimination tree by the dependency graph, we need an efficient scheduler for the concurrent executions of sets of tasks identified as hypergraph grammar productions. We implement our hypergraph grammar productions in the GALOIS system [34]. It automatically schedules the tasks (graph grammar productions) according to DAG (directed accyclic graph) represented by the dependency graph.

The GALOIS scheduler was used in the solver described in [14]. In this paper, we utilize the parallel solver over the two- or three-dimensional grids that possess a structure that is topologically equivalent to a 1D mesh. The GALOIS scheduler was also used in the solver described in [29]. In this paper, we use the GALOIS scheduler working on the element partition tree.

In this current paper, we extend these ideas to a hypergraph model of the computational mesh. This has the following advantages:

- The hypergraph is a natural "flat" data structure for storing the refined computational mesh.
- This allows for the easy construction of operations on finite elements surrounding the singularities located inside the finite element mesh and performing eliminations of these elements level by level.

- The expression of the solver algorithm by hypergraph grammar production allows for the decomposition of the solver algorithm into basic undividable tasks (like in the task-based programming approach).
- These tasks can then be scheduled into multiple cores of parallel shared-memory machines.

We compare our hypergraph grammar-based solver using the GALOIS scheduler with the state-of-the-art MUMPS solver [3–5] on model as well as industrial problems.

For a description of the GALOIS system, we refer to [34]. From the practical point of view, GALOIS can execute several tasks (threads) concurrently, utilizing multiple cores on a shared-memory machine in an optimal manner.

2. Expressing solver algorithm by hypergraph grammar productions

Let us focus on the two finite element meshes represented by a hypergraph (see Figure 1). The vertices of the finite element mesh are represented by hypergraph nodes with **v** labels. The edges of the mesh located on the boundary are represented by hypergraph edges with **B** labels, while the common edge is represented by hypergraph edge with a **F1** label. Finally, the interiors of the finite elements are represented by hypergraph hyperedges with **I** labels.

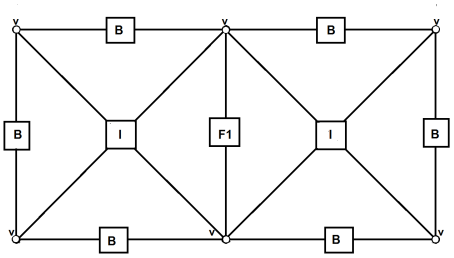


Figure 1. Hypergraph representation of mesh with two elements

Hypergraphs consist of nodes and so-called hyperedges; a hyperedge is an edge that can connect two or more nodes – it has a sequence of nodes assigned to it. The nodes and hyperedges can be labeled with the use of a fixed alphabet. Additionally, each node and hyperedge can have a set of attributes. Figure 1 presents an exemplary hypergraph with six nodes labeled as **v**, one hyperedge labeled as **F1**, two hyperedges labeled as **I**, and six hyperedges labeled as **B**. In the presented graph, no attributes are assigned to the nodes nor hyperedges.

For a formal definition of a hypergraph and how it differs from composition graphs (CP-graphs), for example, we refer to [36].

The hypergraph's grammar was originally introduced in [15, 16]. The instantiation of the hypergraph model for the representation of the computational mesh as well as the instantiation of the hypergraph grammar model for mesh generation is

introduced in [36]. In this paper, the hypergraph grammar is a tool to express the order of generations, mergings, and eliminations performed by the multi-frontal solver algorithm over the computational mesh. To simplify the presentation, we prefer to illustrate the execution of the productions by highlighting the parts of the hypergraph where these operations are performed. Below, we present the explanation on how we understand this notation in the sense of the generation and processing of matrices and assigning them to the nodes in the hypergraph model. The formal definitions of hypergraphs and hypergraph grammars as well as hypergraph productions describing the mesh-generation process can be found in [36].

In our simple example, we use the following hypergraph grammar productions that model the multi-frontal solver algorithm executed over the two finite element mesh:

- Hypergraph-grammar production **(PgenM)** generates the frontal matrix associated to a single element. It is presented in Figure 2. The production labels the element nodes with indices of the rows of the matrix and assigns the matrix to the element interior node, meaning it sets the value-off attribute Matrix of the hyperedge labeled as **I** into the generated element matrix. The rows and columns in the matrix are ordered in such a way that the interior edge rows are localized at the end of the list. The production sets the value of the attribute row of each node and hyperedge of a graph representing a finite element. The values of the attribute row of hyperedges and nodes for the first-generated matrix are set as in Figure 2, where *act_nr* denotes the global variable initialized by 0. After performing the production, global variable *act_nr* is increased by 9 in this case. For the next generated matrices (the matrices for the next elements), some of the nodes or hyperedges can have the value of an attribute row set into a value greater than -1. For these hyperedges and nodes, the value of the attribute row remains the same as on the left-hand side of the production for the corresponding hyperedges and nodes. Additionally, the value of the attribute stat of each hyperedge and node is set to "gen" (which means the generation of the corresponding part of the matrix).

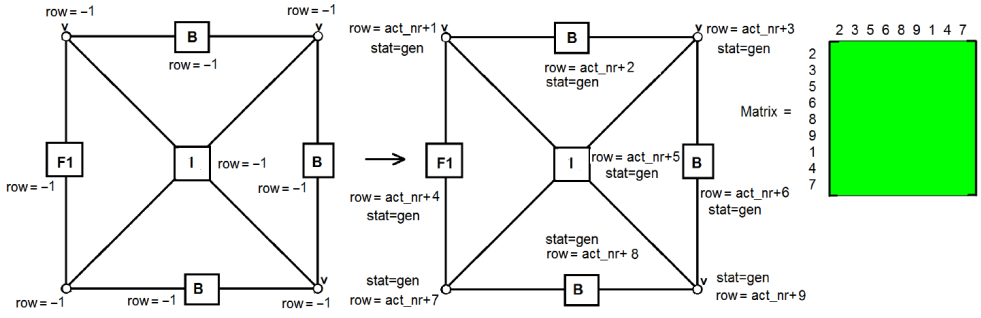


Figure 2. Hypergraph grammar production **(PgenM)** for generation of element matrix

- Hypergraph-grammar production **(PelimM)** eliminates the rows of the element matrix associated with the interior and boundary nodes. The production is presented in Figure 3.

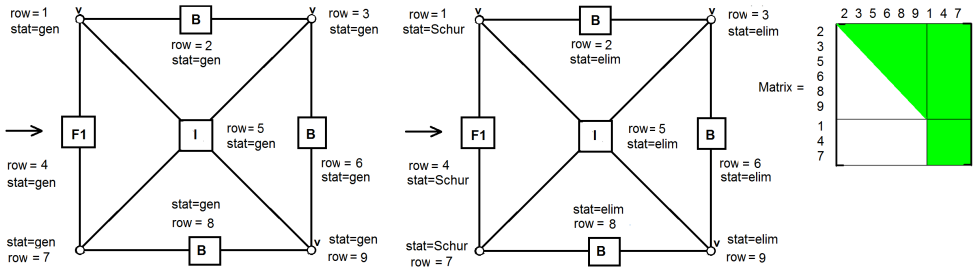


Figure 3. Hypergraph grammar production **(PelimM)** for computing Schur complement of element matrix (also called static condensation procedure)

The procedure is called the static condensation. It leaves the bottom-right part of the matrix as the so-called Schur complement sub-matrix. These rows are associated with the interior edge of the element. The value of the attribute stat is set to "elim" for the hyperedges or nodes representing the eliminated rows and Schur for the remaining hyperedges and nodes.
For the finite element represented by the graph in Figure 4, only the hyperedge and nodes with attribute rows =4,1,7 (respectively) have an attribute stat equal to Schur because they represent common edge and the common vertex of two neighboring elements.

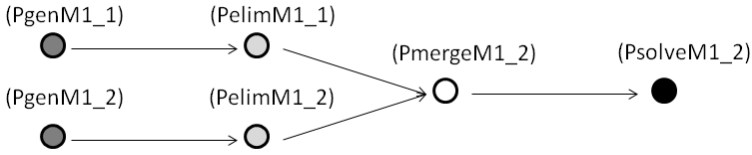


Figure 4. Coloring of dependency graph

For simplicity of presentation, we provide exemplary values of the attribute row of each hyperedge and node in Figure 4. The production remains the same values of the attribute row set earlier by production **(PgenM)** and denoted in the left-hand-side graph of production **(PelimM)** for the corresponding hyperedges and nodes of the right-hand side of the production.

- Hypergraph-grammar production **(PmergeM)** merges the two Schur complement matrices from the two element matrices after static condensation over the two adjacent elements. The production is presented in Figure 5. The merging

follows the association of the rows of the matrices with the nodes of the edge where the Schur complements are computed.

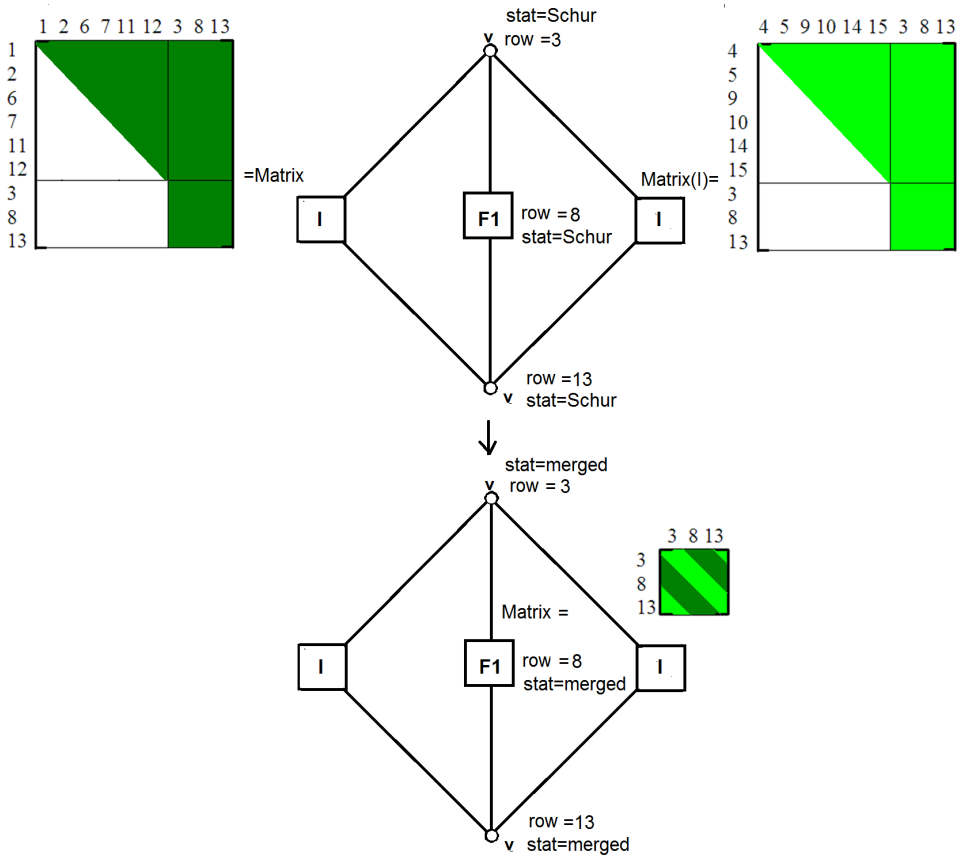


Figure 5. Hypergraph grammar production **(PmergeM)** for merging two Schur complement matrices from two adjacent elements

The production sets the value of the attribute Matrix of the hyperedge labeled as **F1** into the matrix corresponding to the merging of the two Schur complements. The production also sets the value of the attribute stat of this hyperedge and the two nodes into the merged.

- Hypergraph-grammar production **(PsolveM)** factorizes the matrix obtained by merging the Schur complement matrices over the shared element edge. The production is presented in Figure 6. The production sets the value of the attribute Matrix of the hyperedge labeled as **F1** into the factorized matrix corresponding to the merging of two Schur complements. The production also sets the value of the attribute stat of this hyperedge and the two nodes into "solved."

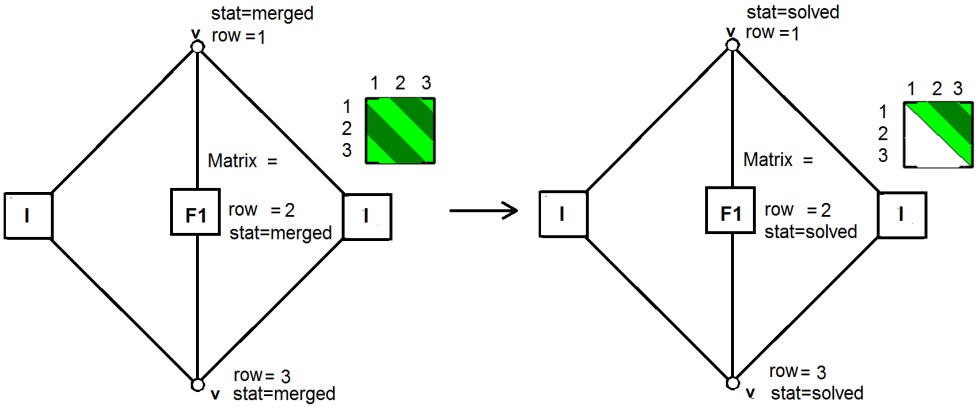


Figure 6. Hypergraph grammar production **(PsolveM)** for solving common edge problem

To simplify the presentation, we illustrate the execution of the hypergraph grammar productions by coloring the nodes of the hypergraph. We highlight the nodes corresponding to the execution of the generation, merging, and elimination productions.

Considering the simple two-finite-element-mesh example, we can express the multi-frontal solver algorithm by hypergraph-grammar productions in the following manner:

- The generation of the frontal matrix associated with the right element is presented in Figure 7(a). The hyperedges and nodes representing the matrix rows and columns are dark gray in color.
- The elimination of the fully-assembled nodes is illustrated in Figure 7(b). The hyperedges and nodes corresponding to eliminated mesh nodes are light gray in color.
- The generation of the frontal matrix associated with the left element is illustrated in Figure 7(c). We need to use a third color to denote the nodes affected by the generation of the second frontal matrix.
- The elimination of the fully-assembled nodes is illustrated in Figure 7(d). The hyperedges and nodes corresponding to eliminated mesh nodes are light gray in color.
- The merging of the two frontal matrices is presented in Figure 7(e). The hyperedges and nodes corresponding to the rows and columns of the merged matrices change colors.
- Finally, the solution of the interface problem is presented in the bottom-right panel of Figure 7(f). The hyperedges and nodes corresponding to the eliminated mesh nodes are light gray in color.

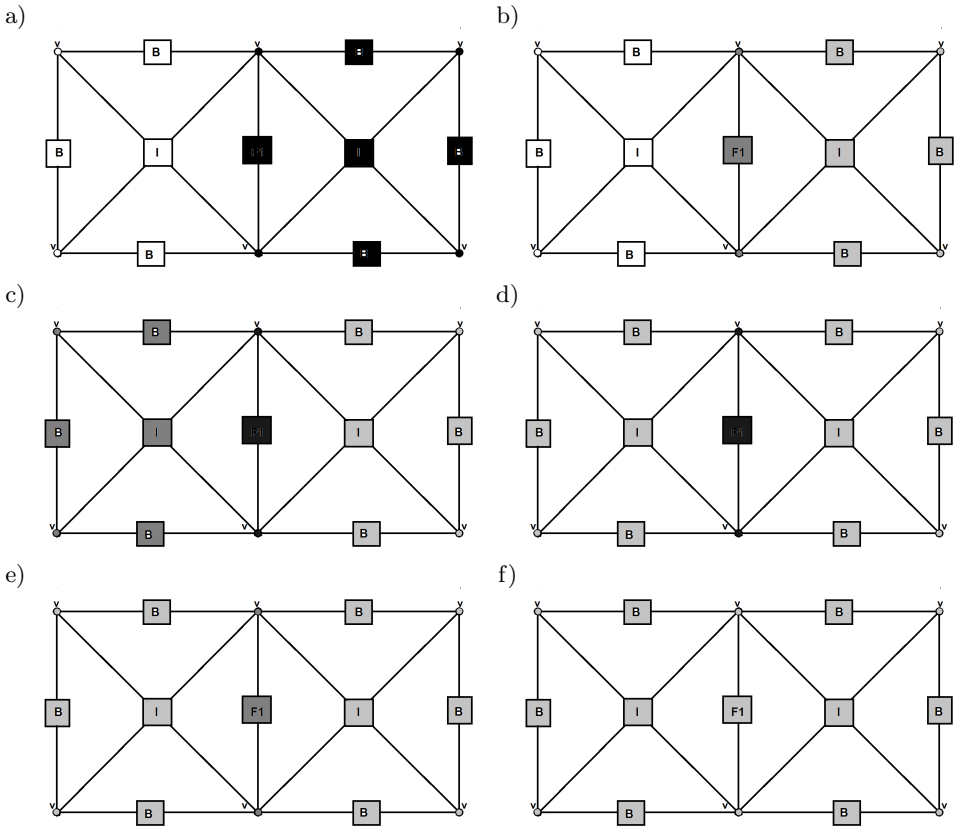


Figure 7. Execution of hypergraph grammar productions **(PgenM)** to the right element (a), **(PelimM)** to the right element (b), **(PgenM)** to the left element (c), **(PelimM)** to the left element (d), **(Pmerge)** for both elements (e) and **(Psolve)** for both elements (f). The starting graph is presented in Figure 1

Finally, we can perform a concurrency analysis [9,20]. The application of the concrete graph grammar production to a concrete hypergraph can be seen as the so-called task. Let us denote the application of production **(PgenM)** to the graph from Figure 1 by **(PgemM1_1)**, the application of production **(Pelim)** to the graph from Figure 7a by **(PelimM1_2)**, the application of production **(PgenM)** to the graph from Figure 7b by **(PgenM1_2)**, the application of production **(Pelim)** to the graph from Figure 7c by **(PelimM1_2)**, the application of production **(PmergeM)** to the graph from Figure 7d by **(PmergeM1_2)**, and the application of production **(PsolveM)** to the graph from Figur 7e by **(PsolveM1_2)**.

The dependency relationship between these tasks can be represented in the form of a dependency graph (see Figure 4). The two tasks are "dependent" when the second one can be started only after the first one is finished. Based on this dependency

graph, we can set up the partial order where the tasks are grouped into sets of independent tasks that can be scheduled set by set into a parallel shared memory machine. This can be obtained from the coloring of the dependency graph shown in Figure 4. From this figure, we can read that tasks **(PgenM1_1)** and **(PgenM1_2)** can be executed concurrently; when they finish, then we can execute **(PelimM1_1)** and **(PelimM1_2)** in concurrent tasks. Later, we can execute task **(PmergeM1_2)** and, finally, task **(PsolveM1_2)**.

3. Analysis of the concurrency for different grids with singularities

3.1. Two-dimensional point singularity

We start with the theoretical analysis of the concurrency for the case of the mesh with point singularity. The exempary deriviation of the hypergraph grammar productions representing the multi-frontal solver algorithm for a mesh with a point singularity are presented in Figure 8. We utilize multiple colors to illustrate the parts of the hypergraph where the particular tasks are executed. The differing shades of colors correspond to the nodes and hyperedges of the hypergraph associated with the eliminated and Schur complement rows in the corresponding matrices.

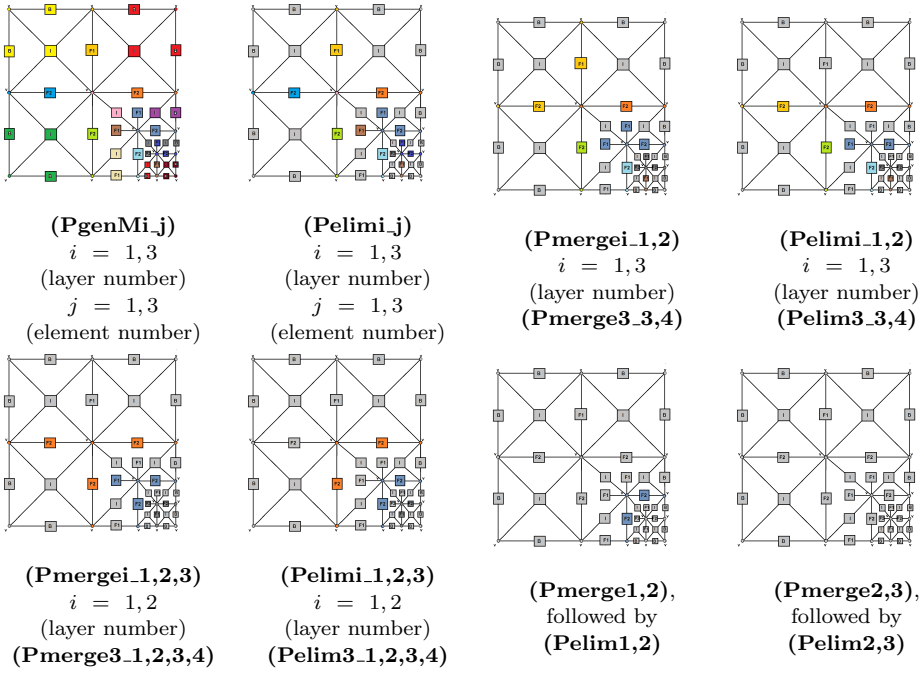


Figure 8. Exempary deriviation of hypergraph representing multi-frontal algorithms for mesh with point singularity

In Figure 9, we construct the dependency graph by analyzing the dependency relationships between these tasks. The nodes in the dependency graph represents the tasks related to the execution of the hypergraph grammar productions on the different parts of the mesh represented by a hypergraph. There is a one-to-one relationship between the hypergraph grammar production executed on a given part of the hypergraph and the task in the dependency graph.

We color the dependency graph layer by layer. Particular sets of tasks are obtained by collecting all of the tasks with the same color on the graph. The graph colors point out the sets of tasks that can be executed concurrently.

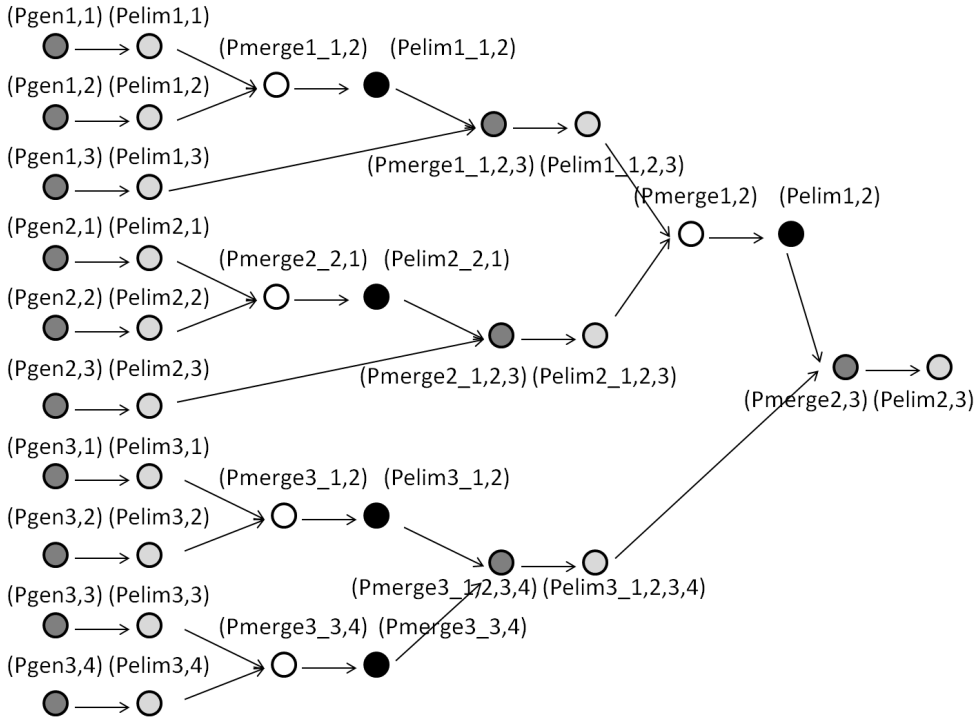
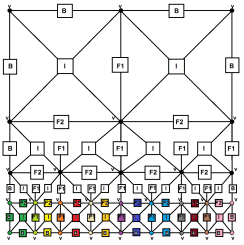


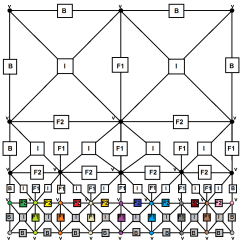
Figure 9. Colored dependency graph for point singularity

3.2. Two-dimensional edge singularity

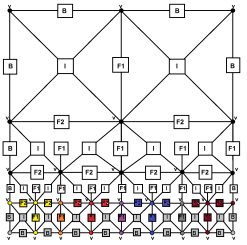
The exemplary derivation of the hypergraph grammar productions that express the multi-frontal solver algorithm for a mesh with an edge singularity is presented in Figures 10 and 11.



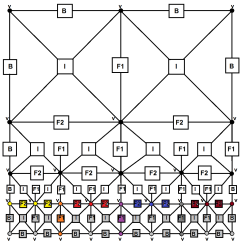
(Pgen1_j)
$j = 1, 8$ (element number)



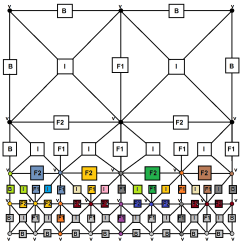
(Pelim1_j)
$j = 1, 8$ (element number)



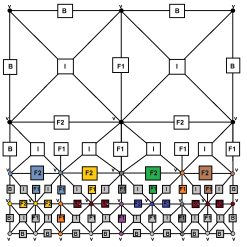
(Pmerge1_j)
$j = 1, 4$ (patch number)



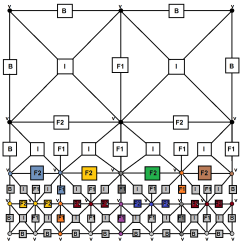
(Pelim1_j)
$j = 1, 4$ (patch number)



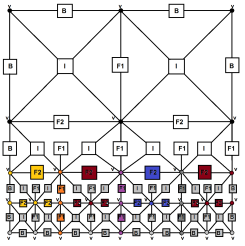
(Pgen2_j)
$j = 1, 8$ (element number)



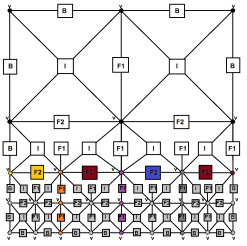
(Pelim2_j)
$j = 1, 8$ (element number)



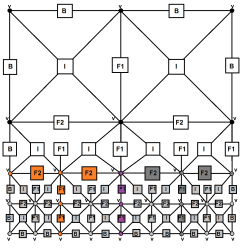
(Pmerge2_2j-1)
(Pelim2_2j-1)
$j = 1, 4$ (patch number)



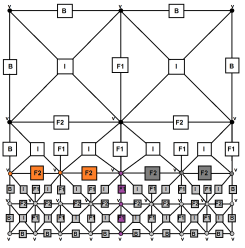
(Pmerge1,2_j)
$j = 1, 4$ (patch number)



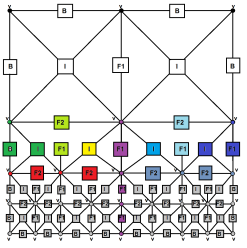
(Pelim1,2_j)
$j = 1, 4$ (patch number)



(Pmerge1,2bis_j)
$j = 1, 2$ (patch number)



(Pelim1,2bis_j)
$j = 1, 2$ (patch number)



(Pgen3_j)
$j = 1, 4$ (element number)

Figure 10. Exempary deriviation of hypergraph representing multi-frontal algorithms for mesh with edge singularity

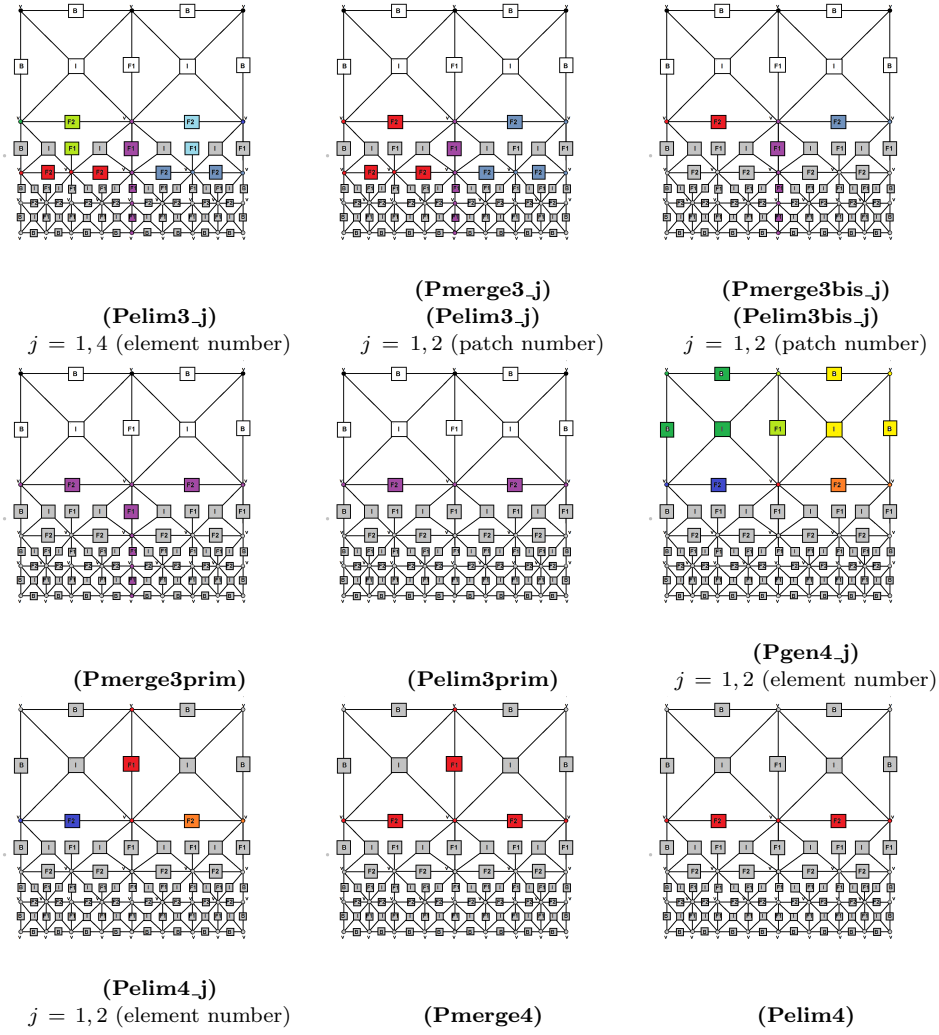


Figure 11. Exempary deriviation of hypergraph representing multi-frontal algorithms for mesh with edge singularity

To simplify the presentation, we illustrate the execution of the hypergraph grammar productions by highlighting the nodes and hyperedges of the graph. Again, we highlight the hyperedges and nodes to which the hypergraph production will be applied.

In Figure 12, we construct the dependency graph by analyzing the dependency relationship between these tasks. The graph is colored to point out the sets of tasks that can be executed concurrently. After the concurrent processing of both parts of the mesh (observe the analogous colors in both graphs), we need to merge the

two resulting frontal matrices and finish the elimination processes by executing hypergraph grammar productions **(Pgen4_1), (Pgen4_2), (Pelim4_1), (Pelim4_2), (Pmerge4), and (Pelim4)**.

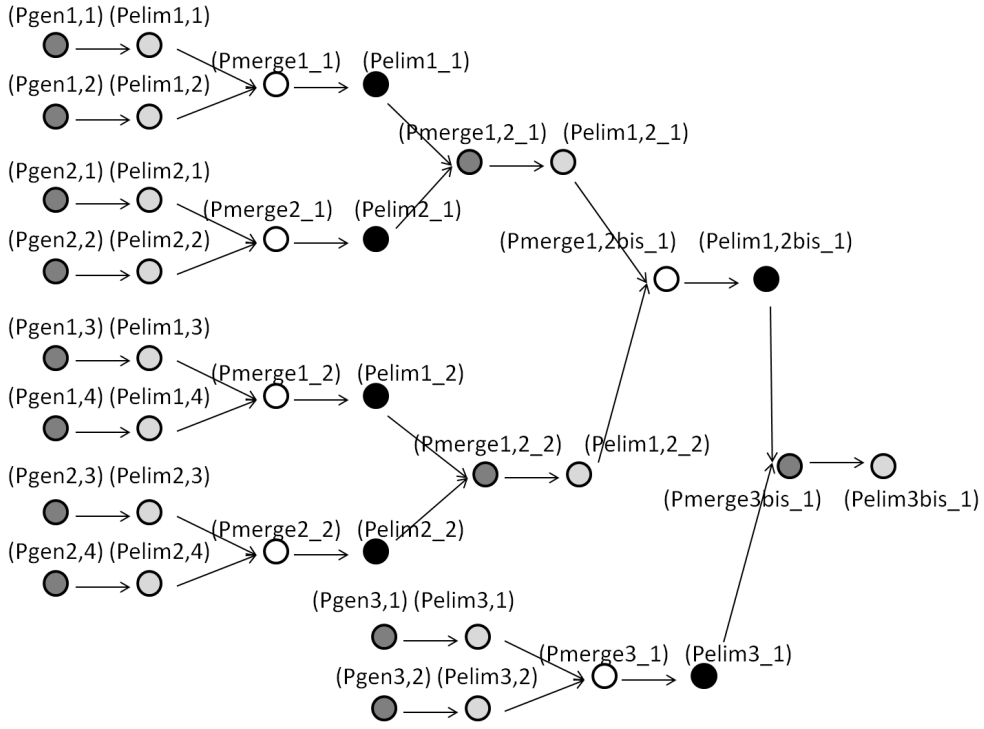


Figure 12. Colored dependency graph for edge singularity for left-hand side of mesh

4. Numerical results

4.1. Model problems

In this section, we present a detailed comparison of the execution time, efficiency, and speedup for the implementation of our hypergraph grammar solver with GALOIS scheduler [34] for two-dimensional grids with point and edge singularities. The singularities are located in the center of the domain in order to investigate the gain associated with the execution of the hypergraph grammar-based solver that eliminates elements level by level.

We do not provide a PDE here since we assume that we are solving any PDE that has a singularity at a point or at an edge that requires several refinements. For elliptic scalar value problems, the sparsity of the resulting discretization matrix is identical; thus, the processing of the tasks follows the same pattern and has the same complexity.

From the comparison presented in Figures 13–16 for the edge singularity, we can read that our sequential GALOIS solver has a faster execution time than the sequential MUMPS solver.

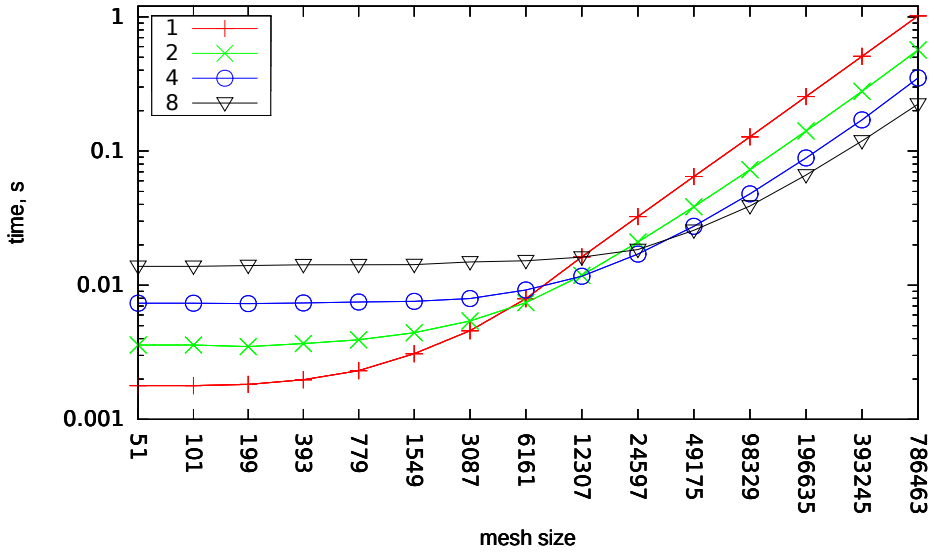


Figure 13. Execution time of GALOIS solver for edge singularity

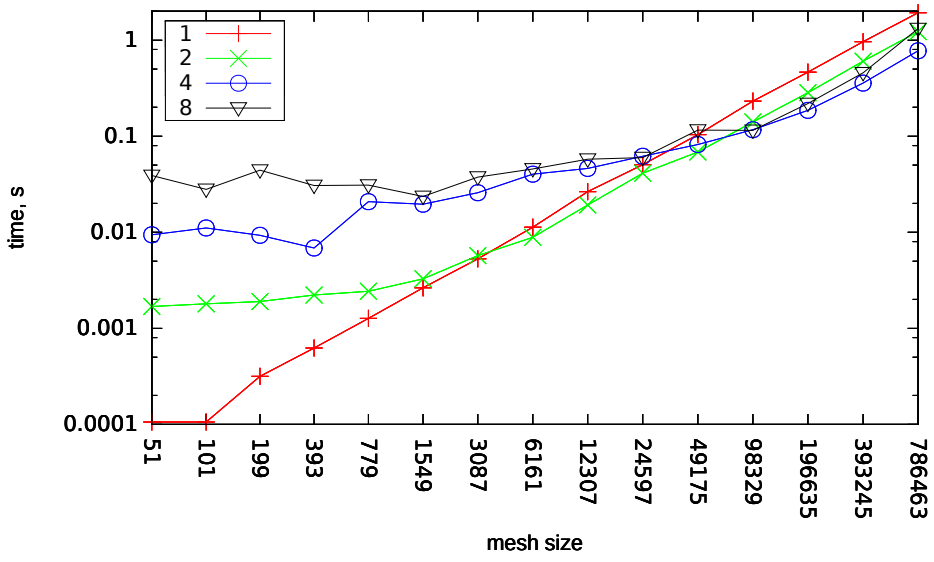


Figure 14. Execution time of MUMPS solver for edge singularity

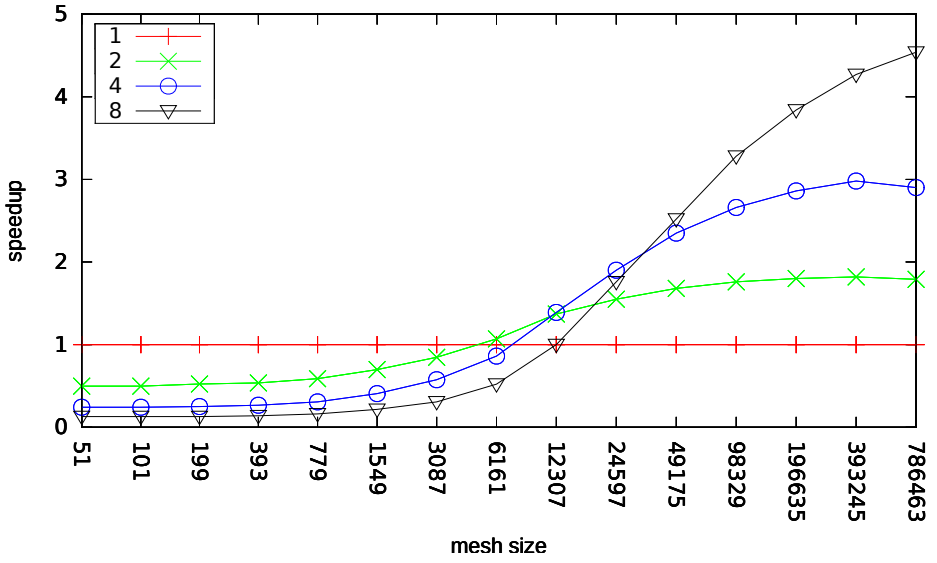


Figure 15. Speedup of GALOIS solver for edge singularity

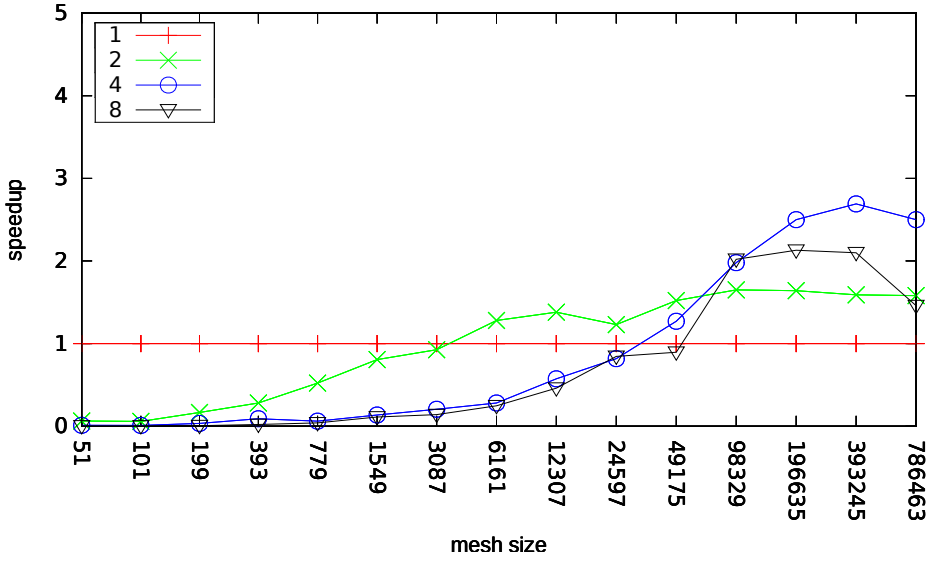


Figure 16. Speedup of MUMPS solver for edge singularity

The MUMPS solver utilizes Cholesky factorization (the problem is symmetric and positive definite), but our solver utilizes LU factorization. Our solver has better

granularity in the sense that the hypergraph grammar model allows us to naturally identify multiple independent tasks that can be grouped together and executed concurrently. These tasks perform a partial LU factorization. These LU factorizations performed on the small task matrices are slower than the Cholesky factorization performed inside the MUMPS solver on parts of the global matrix. However, our total execution time is shorter, as we can perform a better concurrent scheduling of more small tasks resulting from the hypergraph grammar structure.

Our GALOIS solver is a pure C code and has been compiled with gcc-4.8.0. The MUMPS solver has been compiled with gfortran-4.8.0 and linked to metis-4.0.3, atlas-3.10.1, LAPACK-3.4.2, and ScaLAPACK-2.0.2. Thus, both solvers have been compiled in an optimized way. The tests were performed on a single node of an ATARI Linux cluster with an 8-core Intel(R) Xeon(R) CPU with 2.4 GHz and 16 GB RAM. However, the MUMPS solver uses MPI as the communication mechanism, but the GALOIS uses its own mechanism for multi-thread communication.

From the comparison presented in Figures 17–20 for the point singularity, we can read that the grids for the point singularities are very small and that both solvers lost their efficiency there. The MUMPS solver is faster in sequential mode; however, our GALOIS multi-thread solver has a better speedup. This implies that our solver has better mechanisms to manage the thread scheduling and communication, and its overhead is smaller than that of MUMPS, which uses MPI as the communication mechanism over a shared-memory multi-core machine.

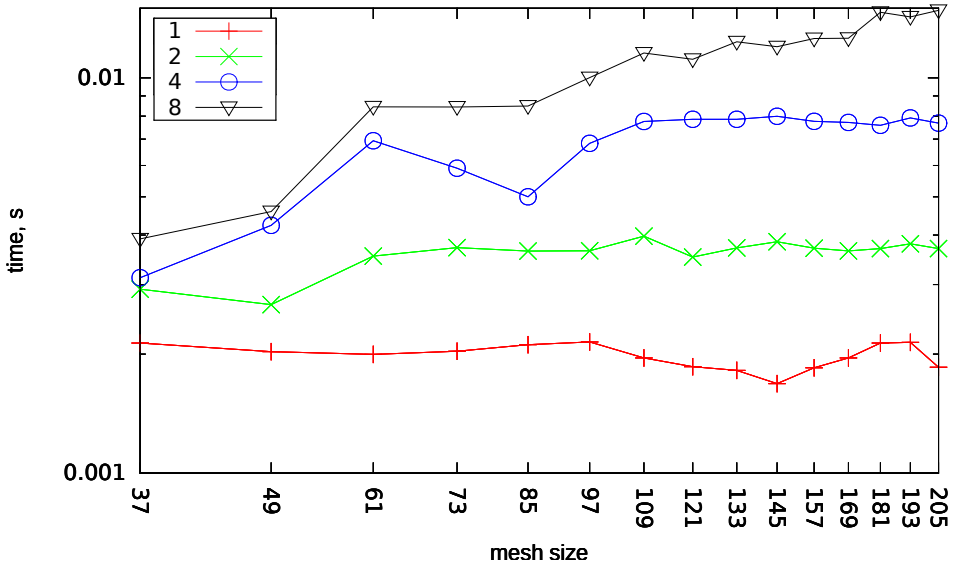


Figure 17. Execution time of GALOIS solver for point singularity

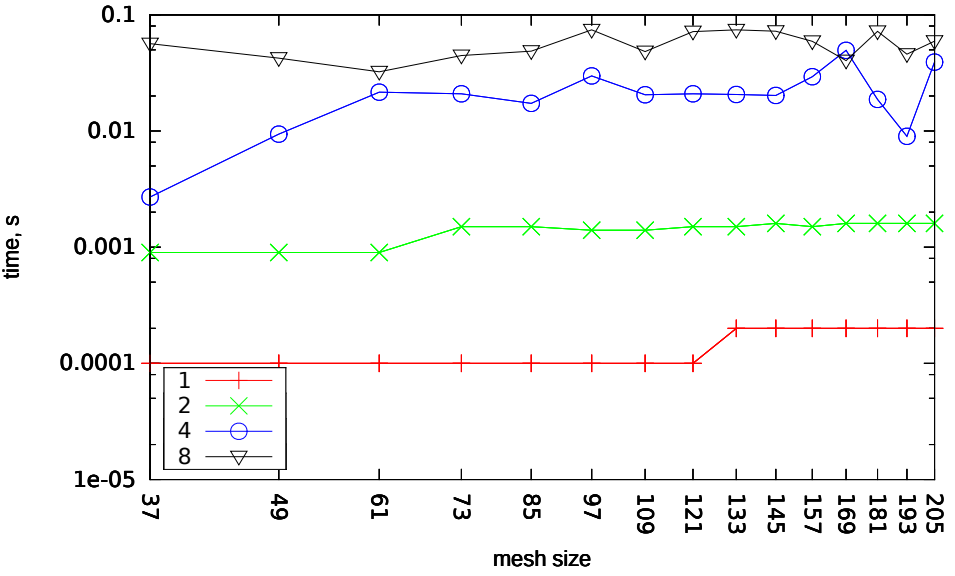


Figure 18. Execution time of MUMPS solver for point singularity

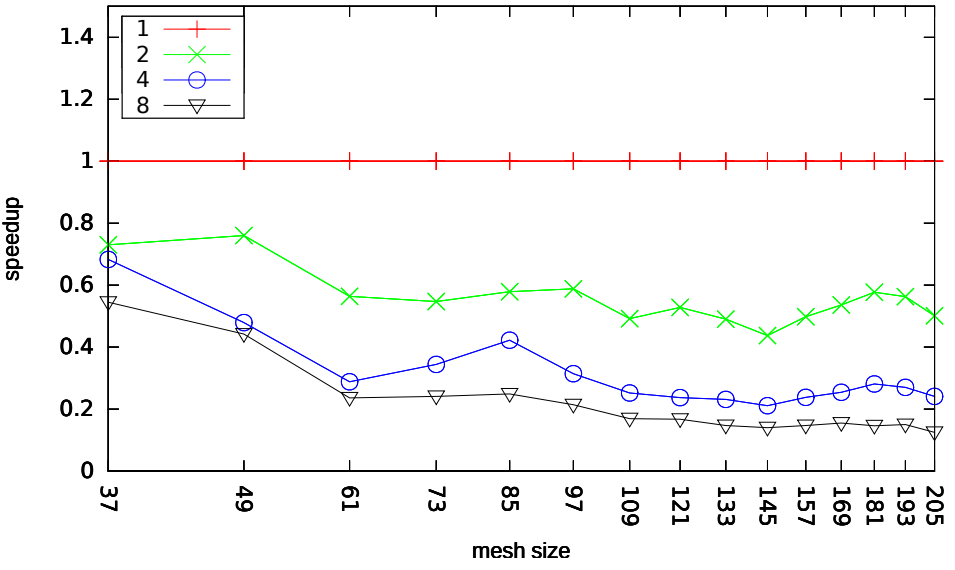


Figure 19. Speedup of GALOIS solver for point singularity

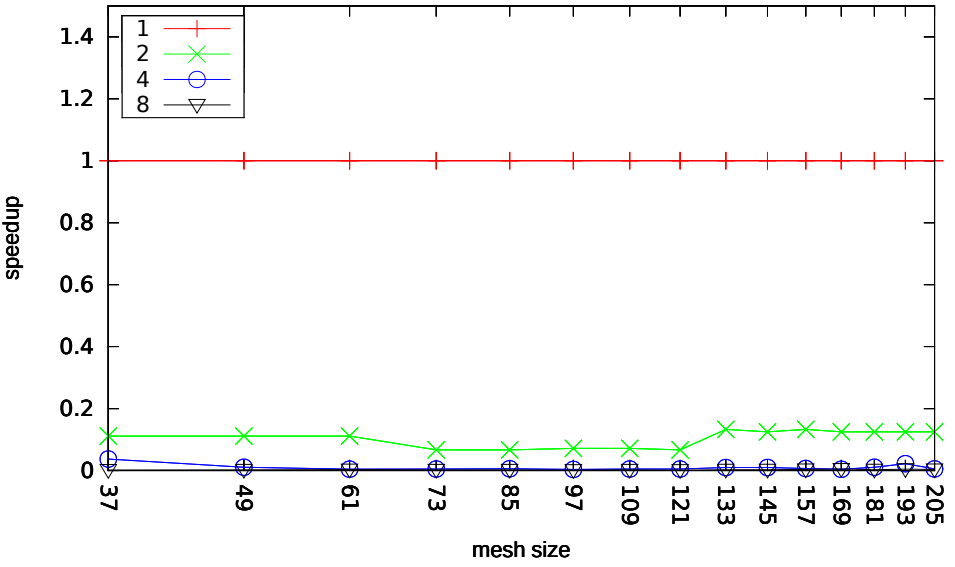


Figure 20. Speedup of MUMPS solver for point singularity

4.2. Quality of ordering resulting from hypergraph grammar sequential processing of mesh

Another issue is how the sequential hypergraph grammar-based solver performs as compared to traditional multi-frontal solvers.

In order to investigate this issue, it is necessary to transform the hypergraph grammar-based processing of the mesh into the ordering for the sequential execution. The ordering may be obtained by considering the sequence of tasks and not from the sets of tasks processed in a concurrent manner. Each task corresponding to **(Pelim*)** production eliminates some degrees of freedom, and this prescribes the ordering.

The method of how we order the tasks in a sequential manner over an arbitrary mesh is prescribed by the bisections weighted by an element-size algorithm [1]. It is worth mentioning that this ordering is possible to obtain with the hypergraph model but not with the CP-graph model due to the structure of the graphs.

This ordering can then be passed to the sequential MUMPS solver, and the execution times and flops can be compared with the traditional ordering algorithms employed by the MUMPS solver (e.g., nested-dissections [17] or AMD [2]). A comparison of the nested-dissection ordering as implemented by the METIS library [17] and the AMD ordering as used by MUMPS [3–5] with the ordering resulting from our hypergraph grammar solver is presented in Figures 21 and 22. We compare the number of floating-point operations performed during the elimination using particular orderings.

The data structure and the ordering that allows for a layer-by-layer elimination of the elements surrounding the singularities results in a lower number of flops.

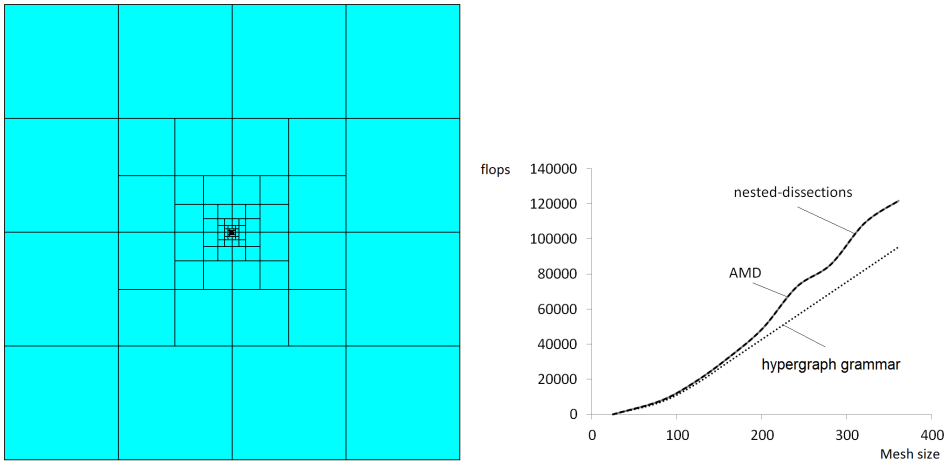


Figure 21. Comparison of hypergraph grammar ordering, nested-dissections, and AMD ordering for point singularity

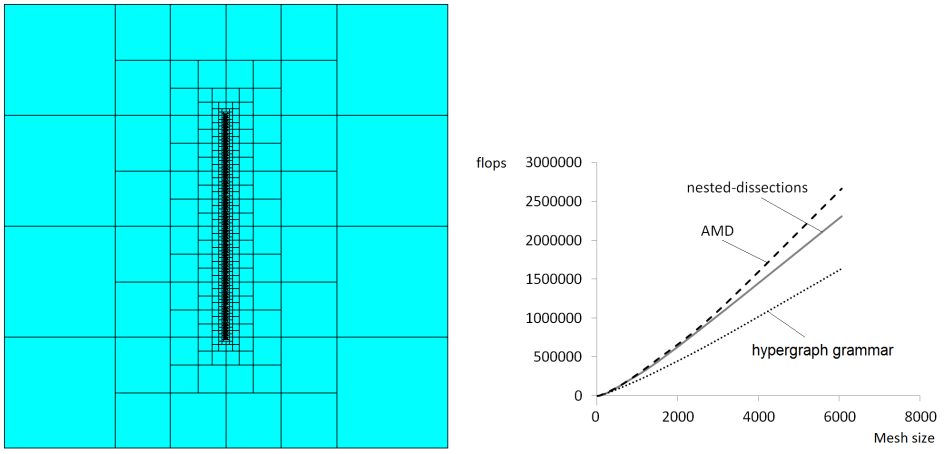


Figure 22. Comparison of hypergraph grammar ordering, nested-dissections, and AMD ordering for edge singularity

4.3. Propagation of electromagnetic waves in formation layers

In this section, we provide the numerical results of our solver for a real engineering problem related to simulations of the propagation of electromagnetic waves in the formation layers.

Several numerical methods exist for the simulation of forward resistivity measurements (i.e. [6, 7, 10, 19, 24, 33, 37–39]). In this section, the forward problem is solved by means of the goal-oriented self-adaptive *hp* Finite Element Method (*hp*-FEM) developed in [23, 25]. We employ our solver as a key component of the goal-oriented strategy. The governing equations are described as follows:

The direct current flow in the continuum 3D conductor is governed by *time--harmonic Maxwell's equations* with an angular frequency of $\omega \neq 0$

$$\begin{cases} curl\mathbf{H} &= (\sigma + i\omega\epsilon)\mathbf{E} + \mathbf{J}^{imp} & \text{(Ampére's law)} \\ curl\mathbf{E} &= -i\omega\mu\mathbf{H} & \text{(Faraday's law)} \\ div(\epsilon\mathbf{E}) &= \chi & \text{(Gauss's law of electricity)} \\ div(\mu\mathbf{H}) &= 0 & \text{(Gauss's law of magnetism),} \end{cases} \tag{1}$$

where $\mathbf{H}$ is the magnetic field intensity (magnetizing field), $\mathbf{E}$ is the electric field, $\mathbf{J}^{imp}$ is a prescribed impressed electric current density, χ is the electric free charge distribution, and ϵ, μ, and σ stand for the permittivity, permeability, and electrical conductivity of the considered medium, respectively.

We are looking for solutions to (1) in domain $\Omega \subset \mathbb{R}^3$ being a 3D cylinder surrounding the borehole (see Fig. 23).

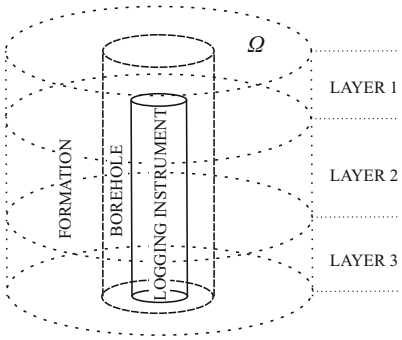


Figure 23. Three-dimensional geometry of logging instrument in vertical borehole penetrating three dipping layers. $x = (x_1, x_2, x_3)$ represents Cartesian system of coordinates, and $\boldsymbol{\zeta} = (\zeta_1, \zeta_2, \zeta_3)$ represents new non-orthogonal system of coordinates. New system of coordinates is different in all three sub-domains. Subdomain I corresponds to logging instrument, Subdomain II to borehole, and Subdomain III to formation. New system of coordinates is globally continuous (as indicated by parameterization)

Notice that such an Ω is a simply connected bounded domain with a Lipschitz boundary. On boundary $\Gamma = \partial\Omega$, we consider homogeneous perfect electric conductor (PEC) boundary conditions

$$\mathbf{n} \times \mathbf{E} = 0 \text{ on } \Gamma \tag{2}$$

with $\mathbf{n}$ being the outward unit normal vector field.

We consider the following Hilbert spaces:

$$H(curl;\Omega) = \left\{\mathbf{F} \in L^2(\Omega;\mathbb{C}^3) : curl\mathbf{F} \in L^2(\Omega;\mathbb{C}^3)\right\} \tag{3}$$

and

$$V = H_0(curl;\Omega) = \{\mathbf{F} \in H(curl;\Omega) : \mathbf{n}\times\mathbf{F} = 0 \text{ on } \Gamma\}\,. \tag{4}$$

Both spaces are endowed with an inner product:

$$(\mathbf{E},\mathbf{F})_{H(curl;\Omega)} = \int_\Omega curl\mathbf{E}\cdot curl\overline{\mathbf{F}}dx + \int_\Omega \mathbf{E}\cdot\overline{\mathbf{F}}dx \tag{5}$$

where $\mathbf{u}\cdot\mathbf{v} = u_1v_1 + u_2v_2 + u_3v_3$ (note that this is not the standard inner product in $\mathbb{C}^3$).

Dividing both sides of Faraday's law by $\frac{1}{\mu}$, multiplying them by $curl\overline{\mathbf{F}}$, integrating (one term by parts) over domain Ω, and finally applying Ampére's law, we obtain the following variational formulation (the so-called **E**-formulation): find $\mathbf{E}\in V$ such that

$$\int_\Omega \frac{1}{\mu} curl\mathbf{E}\cdot curl\overline{\mathbf{F}}dx - \int_\Omega (\omega^2\epsilon - i\omega\sigma)\mathbf{E}\cdot\overline{\mathbf{F}} = -i\omega\int_\Omega \mathbf{J}^{imp}\cdot\overline{\mathbf{F}}dx \tag{6}$$

for every test field $\mathbf{F}\in V$. We assume that $\mathbf{J}^{imp}\in L^2(\Omega;\mathbb{C}^3)$, $(\omega^2\epsilon - i\omega\sigma), \mu \neq 0$, $\mu\in\mathbb{R}, \epsilon\in\mathbb{R}, \mu > 0, \epsilon > 0$, and $\sigma\in L^\infty(\Omega), \sigma\geq\sigma_0 > 0$ a.e. in Ω.

For the case of deviated wells (below 90 degrees) in a horizontally stratified layered media, we employ the *hp*–Fourier Finite Element Method described in [21]. This method performs a non-orthogonal change of coordinates followed by a Fourier series expansion in the azimuthal direction. Using that technique, we obtain fast and accurate forward simulations of the 3D resistivity logging measurements in the deviated wells.

The forward problem is formulated for a 3D-skewed cylinder; therefore, we perform a change of variables from Cartesian to the non-orthogonal system of coordinates (see Figure 23). First, we consider the union of three (possibly rotated) cylindrical systems of coordinates defined over subdomains I, II, and III (respectively) as illustrated in Figure 23. The change of coordinates $\boldsymbol{\zeta} = (\zeta_1,\zeta_2,\zeta_3) = \psi(x)$ is globally continuous and with a positive Jacobian J; therefore, it is suitable for Finite Element computations. In addition, we observe that J (as a function of ζ_2) can be represented as a linear combination of functions $1, \sin(\zeta_2)$, and $\cos(\zeta_2)$ because the change of coordinates is composed of rotations of the cylindrical system of coordinates. It is easy to see that corresponding metric tensor J^TJ (as a function of ζ_2) can be represented in terms of the following five Fourier basis functions: $1, \cos(\zeta_2), \sin(\zeta_2), \cos^2(\zeta_2)$, and $\sin^2(\zeta_2)$. Since the material properties for the geometry described in Figure 23 (deviated wells) are constant with respect to new quasi-azimuthal direction ζ_2 and the metric can be represented exactly with only five Fourier modes, we conclude that, when using a 1D Fourier series expansion in terms of ζ_2, the corresponding stiffness matrix becomes penta-diagonal (as opposed to a dense matrix) with respect to ζ_2, leading to a dramatic reduction of computational complexity. For details, see [21].

The primal direct problem for the conductive media equation can be summarized as follows: find electric potential $u \in H_0^1(\Omega)$ assuming the zero Dirichlet condition on the whole $\partial\Omega$ for the given resistivities of all layers and the borehole subdomain. The influence of the probe is expressed by the assumed displacement of $q = \boldsymbol{\nabla} \cdot \mathbf{J}^{imp}$ in the borehole subdomain.

We have connected our solver to the goal-oriented *hp* adaptive finite element method code solving the problem of the propagation of electromagnetic waves in the borehole and the formation layers. The code uses a 2D mesh expanded in the cylindrical system of coordinates. Thus, the computational mesh is treated like a 2D mesh with the number of unknowns per mesh nodes corresponding to the number elements in the azimuthal direction.

The exemplary meshes after rotating the 2D mesh and transforming it back to the skew cylindrical system are presented in Figures 24–25.

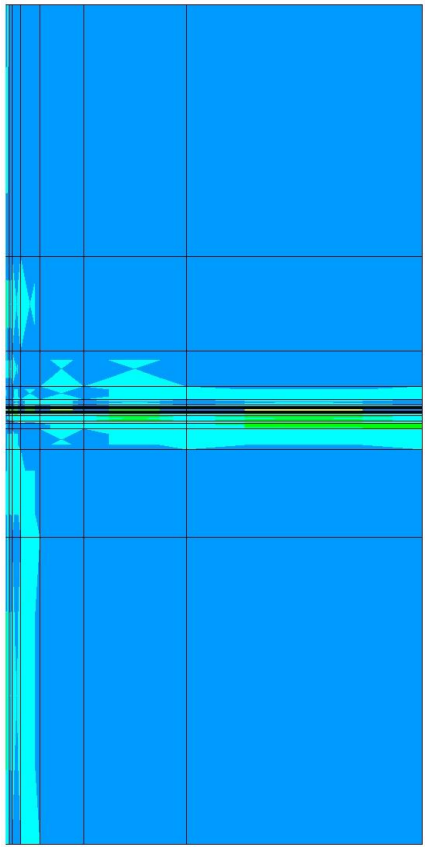


Figure 24. Example of two-dimensional grid generated by goal-oriented *hp*-adaptive finite element method code solving

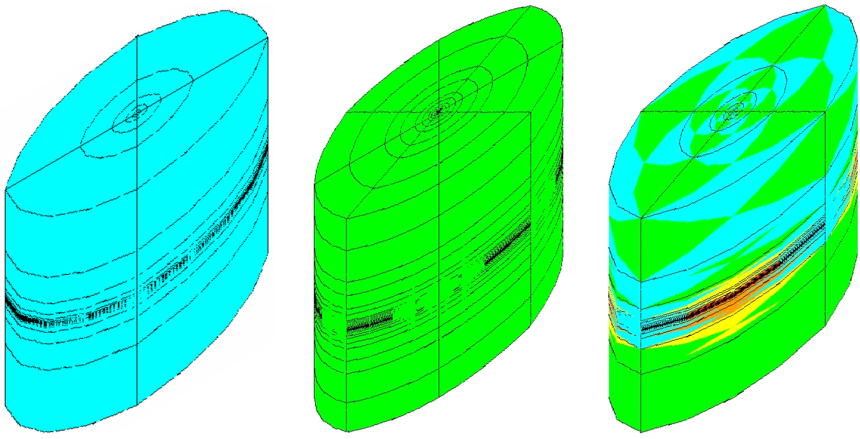


Figure 25. Example of three-dimensional grid obtained by rotating two-dimensional one in azimuthal direction

The code operates in two loops. The first loop iterates through positions of antennas; for each position, the second loop with mesh refinements are executed. We refer to [21] for more details. In Table 1, we present a comparison of an execution time of the MUMPS solver and our solver on a grid generated by the goal-oriented code as well as the total time of the code for all of the positions of the antennas. We can read a similar scalability of our solver with the MUMPS solver and around a two- to three-times-faster execution time of our solver. The largest mesh size from a single antenna solution is equal to 148,257, but the solver is called several times for different positions of the antennas. We break the computations for 32 cores since the solver execution time is larger than for 16 cores. This is due to hyperthreading on the Linux cluster node.

Table 1
Scalability of our solver on problem of propagation of electromagnetic waves

Cores	1	2	4	8	16	32
Hypersolver time [s]	–	14.506	8.124	5.209	3.35	4.038
MUMPS time [s]	–	27.859	17.457	12.747	10.011	11.015
Total time [s]	–	1087	735	555	295	–

5. Conclusions

In this paper, we have analyzed the concurrency of the multi-frontal direct solver executed over model two-dimensional grids with point and edge singularity.

We defined several hypergraph grammar productions over the hypergraph representation of the computational mesh. We analyzed the dependency relationship

between the so-called tasks (defined as application of the production to the subhypergraph of the hypergraph representing the mesh) and constructed dependency graphs that are the equivalents of elimination trees. The hypergraph grammar productions were implemented and executed in the GALOIS system. A sequence of numerical experiments tested the scalability of the resulting multi-thread multi-frontal direct solver.

We show that our GALOIS-based solver outperforms the MUMPS solver on the example of edge singularity. This is because MUMPS is not a task-based programming model solver. The utilization of the graph grammar for the expression of the solver algorithm allows for a better granularity in the computations, and there are more independent tasks to be scheduled for concurrent execution. The hypergraph grammar model allows for a better definition of the computational tasks related to hypergraph grammar productions. Since the hypergraph represents the computational mesh as a flat structure, it is easy to process the elements surrounding the singularities layer by layer. This in turn produces more independent tasks to be scheduled for concurrent execution.

The hypergraph is just a "flat" data structure allowing for a simple construction of tasks surrounding the singularities and processing them level by level. The hypergraph grammar is just a model for expressing the solver algorithm. For more--complicated grids, it requires some control algorithm that chooses the parts of the mesh where the productions will be executed and allow for the selection of sets of tasks by coloring the dependency graph. For the problem of the propagation of electromagnetic waves in formation layers, we employ the bisections weighted by an element size algorithm [1] to identify the tasks. These sets are then scheduled by the GALOIS using the available cores. Our future work will involve a generalization to three-dimensional grids.

Acknowledgements

This work was supported by National Science Center, Poland – Grant No. DEC-2015/17/B/ST6/01867.

References

[1] AbouEisha H., Moshkov M., Calo V.M., Paszyński M., Goik D., Jopek K.: 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.

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

[3] Amestoy P.R., Duff I.S., L'Excellent J.Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers, *Computer Methods in Applied Mechanics and Engineering*, vol. 184(2), pp. 501–520, 2000.

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

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

[6] Avdeev D.B., Kuvshinov A.V., Pankratov O.V., Newman G.A.: Three-dimensional induction logging problems, Part I: An integral equation solution and model comparisons, *Geophysics*, vol. 67(2), pp. 413–426, 2002.

[7] Davydycheva S., Druskin V., Habashy T.: An efficient finite-difference scheme for electromagnetic logging in 3D anisotropic inhomogeneous media, *Geophysics*, vol. 68(5), pp. 1525–1536, 2003.

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

[9] Diekert V., Rozenberg G.: *The book of traces*, World Scientific Publishing, 1995.

[10] Druskin V.L., Knizhnerman L.A., Lee P.: New spectral Lanczos decomposition method for induction modeling in arbitrary 3-D geometry, *Geophysics*, vol. 64(3), pp. 701–706, 1999.

[11] Duff I.S., Reid J.K.: The Multifrontal Solution of Indefinite Sparse Symmetric Linear, *ACM Transactions on Mathematical Software*, vol. 9, pp. 302–325, 1983.

[12] Duff I.S., Reid J.K.: The Multifrontal Solution of Unsymmetric Sets of Linear Equations, *SIAM Journal on Scientific and Statistical Computing*, vol. 5, pp. 633–641, 1984.

[13] Geng P., Oden J., van de Geijn R.: A parallel multifrontal algorithm and its implementation, *Computer Methods in Applied Mechanics and Engineering*, vol. 149(1), pp. 289–301, 1997.

[14] Goik D., Jopek K., Paszyński M., Lenharth A., Nguyen D., Pingali K.: Graph Grammar based Multi-thread Multi-frontal Direct Solver with Galois Scheduler, *Procedia Computer Science*, vol. 29, pp. 960–969, 2014.

[15] Habel A., Kreowski H.J.: May we introduce to you: Hyperedge replacement. In: Ehrig H., Nagl M., Rozenberg G., Rosenfeld A. (eds.), *Graph-Grammars and Their Application to Computer Science. Graph Grammars 1986*, Lecture Notes in Computer Science, vol. 291, pp. 15–26, Springer, Berlin, Heidelberg, 1987.

[16] Habel A., Kreowski H.J.: Some Structural Aspects of Hypergraph Languages Generated by Hyperedge Replacement. In: Brandenburg F.J., Vidal-Naquet G., Wirsing M. (eds.), *STACS 87. STACS 1987*, Lecture Notes in Computer Science, vol. 247, pp. 207–219, Springer, Berlin, Heidelberg, 1987.

[17] Karypis G., Kumar V.: MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, 1998. `http://umn.edu/home/karypis`.

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

[19] Newman G.A., Alumbaugh D.L.: Three-dimensional induction logging problems, Part 2: A finite-difference solution, *Geophysics*, vol. 67(2), pp. 484–491, 2002.

[20] Obrok P., Pierzchala P., Szymczak A., Paszyński M.: Graph grammar-based multi-thread multi-frontal parallel solver with trace theory-based scheduler, *Procedia Computer Science*, vol. 1(1), pp. 1993–2001, 2010.

[21] Pardo D., Calo V., Torres-Verdín C., Nam M.: Fourier series expansion in a non-orthogonal system of coordinates for the simulation of 3D-DC borehole resistivity measurements, *Computer Methods in Applied Mechanics and Engineering*, vol. 197(21), pp. 1906–1925, 2008.

[22] Pardo D., Demkowicz L., Torres-Verdín C., Paszyński M.: A self-adaptive goal-oriented *hp*-finite element method with electromagnetic applications. Part II: Electrodynamics, *Computer Methods in Applied Mechanics and Engineering*, vol. 196(37), pp. 3585–3597, 2007.

[23] Pardo D., Demkowicz L., Torres-Verdín C., Paszynski M.: 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(6), pp. 2085–2106, 2006.

[24] Pardo D., Torres-Verdín C., Paszynski M.: Simulations of 3D DC borehole resistivity measurements with a goal-oriented *hp* finite-element method. Part II: through-casing resistivity instruments, *Computational Geosciences*, vol. 12(1), pp. 83–89, 2008.

[25] Pardo D., Torres-Verdin C., Demkowicz L.F.: Simulation of multifrequency borehole resistivity measurements through metal casing using a goal-oriented *hp* finite-element method, *IEEE Transactions on Geoscience and Remote Sensing*, vol. 44(8), pp. 2125–2134, 2006.

[26] Paszyńska A., Grabska E., Paszyński M.: A Graph Grammar Model of the *hp* Adaptive Three Dimensional Finite Element Method. Part I, *Fundamenta Informaticae*, vol. 114, pp. 149–182, 2012.

[27] Paszyńska A., Grabska E., Paszyński M.: A Graph Grammar Model of the *hp* Adaptive Three Dimensional Finite Element Method. Part II, *Fundamenta Informaticae*, vol. 114, pp. 183–201, 2012.

[28] Paszyńska A., Paszyński M., Grabska E.: Graph Transformations for Modeling *hp*-Adaptive Finite Element Method with Mixed Triangular and Rectangular Elements. In: Allen G., Nabrzyski J., Seidel E., van Albada G.D., Dongarra J., Sloot P.M.A. (eds.), *Computational Science – ICCS 2009. ICCS 2009*, Lecture Notes in Computer Science, vol. 5545, pp. 875–884, Springer, Berlin, Heidelberg, 2009.

[29] Paszyńska A., Paszyński M., Jopek K., Woźniak M., Goik D., Gurgul P., AbouEisha P., Moshkov M., Calo V.M., Lenharth A., Nguyen D., Pingali K.: Quasi-Optimal Elimination Trees for 2D Grids with Singularities, *Scientific Programming*, vol. 2015, pp. 1–18, 2015.

[30] Paszyński M.: On the Parallelization of Self-Adaptive *hp*-Finite Element Methods, Part I. Composite Programmable Graph Grammar Model, *Fundamenta Informaticae*, vol. 93(4), pp. 411–434, 2009.

[31] Paszyński M.: On the Parallelization of Self-Adaptive *hp*-Finite Element Methods, Part II. Partitioning Communication Agglomeration Mapping (PCAM) Analysis, *Fundamenta Informaticae*, vol. 4(93), pp. 435–457, 2009.

[32] Paszyński M., Schaefer R.: Graph grammar-driven parallel partial differential equation solver, *Concurrency and Computation Practice and Experience*, vol. 22, pp. 1063–1097, 2010.

[33] Paszyński M., Demkowicz L., Pardo D.: Verification of goal-oriented HP-adaptivity, *Computers & Mathematics with Applications*, vol. 50(8), pp. 1395–1404, 2005.

[34] Pingali K., Nguyen D., Kulkarni M., Burtscher M., Hassaan M.A., Kaleem R., Lee T.H., Lenharth A., Manevich R., Méndez-Lojo M., Prountzos D., Sui X.: The Tao of Parallelism in Algorithms, *SIGPLAN Not.*, vol. 46(6), pp. 12–25, 2011.

[35] Schmitz P.G., Ying L.: A fast direct solver for elliptic problems on general meshes in 2D, *Journal of Computational Physics*, vol. 231(4), pp. 1314–1338, 2012.

[36] Ślusarczyk G., Paszyńska A.: Hypergraph Grammars in *hp*-adaptive Finite Element Method, *Procedia Computer Science*, vol. 18, pp. 1545–1554, 2013.

[37] Wang T., Fang S.: 3-D electromagnetic anisotropy modeling using finite differences, *Geophysics*, vol. 66(5), pp. 1386–1398, 2001.

[38] Wang T., Signorelli J.: Finite-difference modeling of electromagnetic tool response for logging while drilling, In: *Geophysics*, vol. 69(1), pp. 152–160, 2004.

[39] Zhang J., Mackie R.L., Madden T.R.: 3-D resistivity forward modeling and inversion using conjugate gradients, *Geophysics*, vol. 60(5), pp. 1313–1325, 1995.

Affiliations

Konrad Jopek
AGH University of Science and Technology, Department of Computer Sciences, Krakow, Poland, kjopek@gmail.com

Maciej Paszyński
AGH University of Science and Technology, Department of Computer Sciences, Krakow, Poland, maciej.paszynski@agh.edu.pl, ORCID ID: https://orcid.org/0000-0001-7766-6052

Anna Paszyńska
Jagiellonian University, Krakow, Poland, anna.paszynska@uj.edu.pl, ORCID ID: https://orcid.org/0000-0002-0716-0619

Muhammad Amber Hassan
The University of Texas at Austin, Institute for Computational and Engineering Sciences, USA, amberuet@gmail.com

Keshav Pingali
The University of Texas at Austin, Institute for Computational and Engineering Sciences, USA, amberuet@gmail.com

Received: 08.08.2018
Revised: 17.11.2018
Accepted: 17.11.2018