Petri nets controlled direct solver for 3D DC borehole resistivity measurements simulations

Arkadiusz Szymczak\(^1\), Maciej Paszyński\(^*\)

\(^1\)Department of Computer Science, AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: maciej.paszynski@agh.edu.pl

Abstract

We present a new multi-frontal direct solver algorithm for hp refined Finite Element Method meshes. The order of elimination in the solver is controlled by a Petri net. The Petri net controls the order of elimination of the solver to avoid grow in the number of degrees of freedom (d.o.f.) located on the interface between adjacent refined elements. It forces the solver to eliminate d.o.f. from the interface between adjacent refined elements going up the refinement tree. The algorithm is tested on 3D DC borehole resistivity measurements simulations in deviated wells, solved on 2D hp refined mesh and Fourier series expansions in the azimuthal direction.

Keywords: resistivity logging measurement simulations, multi-frontal direct solver, Petri nets, graph grammar

1. Introduction

In this paper we present how by using Petri nets controlled solver [1] we can speed up the computations of a 3D DC borehole resistivity measurements simulations in deviated wells [2, 3].

2. Strong formulation

Find \( u: R^3 \supset \Omega \ni x \rightarrow u(x) \in R \) the electrostatic scalar potential such that

\[ -\sum_{i=1}^{3} \sigma \frac{\partial^2 u}{\partial x_i^2} = \nabla \cdot J \text{ in } \Omega \]  

(1)

(conductive media equation) where \( \nabla \cdot J \) is the load (divergence of the impressed current [4]) and \( \sigma \) represents the conductivity of the media.

3. Weak formulation

To solve the above 3D problem (1), we introduce the new quasi-cylindrical non-orthogonal system of coordinates shown in Figure 1. This chapter presents a summary of the derivation presented in detail in [2, 3]. The notation from these papers is also used here. The variational formulation with respect to the electric potential \( u \) in new system of coordinates can be expressed in the following way: Find \( u \in V \) such that:

\[
\int_{\Omega} \sum_{n=1}^{3} \frac{\partial u}{\partial \zeta_n} \frac{\partial v}{\partial \zeta_n} d\zeta = \int_{\Omega} \nu \nabla \cdot \tilde{J} d\zeta \quad \forall v \in H^1_0(\Omega)  
\]

(2)

where

\[
V = \left\{ v \in L^2(\Omega) : \int_{\Omega} \left[ \|v\|^2 + \|\nabla v\|^2 \right] dt < \infty \text{ and } v|_{\Gamma_D} = 0 \right\}  
\]

The electric conductivity of media in new system of coordinates is equal to \( \tilde{\sigma} := \text{Jac}^{-1} \sigma \text{Jac}^{-1} \), and

\[
\nabla \cdot \tilde{J} := \nabla \cdot J \text{Jac} \]  

with \( \text{Jac} \) being the Jacobian matrix of...
the change of coordinates with respect to the Cartesian reference system of coordinates \((x_1, x_2, x_3)\), namely,

\[
\text{Jac} = \frac{\partial (\zeta_1, \zeta_2, \zeta_3)}{\partial (x_1, x_2, x_3)}
\]

(3)

Then, we take a Fourier series expansion of the solution, material and \(\zeta_2\) direction

\[
u(\zeta_1, \zeta_2, \zeta_3) = \sum_{l=-\infty}^{l=\infty} u_l(\zeta_1, \zeta_3) e^{jl\zeta_2}
\]

(4)

\[
\sigma(\zeta_1, \zeta_2, \zeta_3) = \sum_{m=-\infty}^{m=\infty} \sigma_m(\zeta_1, \zeta_3) e^{jm\zeta_2}
\]

(5)

\[
\nabla \sigma(\zeta_1, \zeta_2, \zeta_3) = \sum_{l=-\infty}^{l=\infty} \nabla \sigma_l(\zeta_1, \zeta_3) e^{jl\zeta_2}
\]

(6)

where

\[
u_l = \frac{1}{2\Pi} \int_0^{2\Pi} u e^{-jl\zeta_2} d\zeta_2, \quad \sigma_m = \frac{1}{2\Pi} \int_0^{2\Pi} \sigma e^{-jm\zeta_2} d\zeta_2
\]

and

\[
\nabla \sigma_l = \frac{1}{2\Pi} \int_0^{2\Pi} \nabla \sigma e^{-jl\zeta_2} d\zeta_2
\]

and \(j\) is the imaginary unit. We introduce symbol \(F_l\) such that applied to a scalar function \(u\) it produces the \(l\)th Fourier modal coefficient \(u_l\), and when applied to a vector or matrix, it produces a vector or matrix of the components being \(l\)th Fourier modal coefficients of the original vector or matrix components.

Using the Fourier series expansions we get the following variational formulation:

Find \(F_l(u) \in V\) such that:

\[
\left[ \int_{\Omega} \sum_{l, m=-\infty}^{\infty} F_l \left( \frac{\partial u}{\partial \zeta} \right) F_m \left( \frac{\partial \sigma}{\partial \zeta} \right) e^{j(l+m)\zeta} \, d\zeta \right] v = 0
\]

\[
\forall v \in V
\]

(7)

The summation is applied with respect to \(-\infty \leq l, m \leq \infty\). We select a mono-modal test function \(v = v_k e^{jk\zeta_2}\).

Thanks to the orthogonality of the Fourier modes in \(L^2(\Omega)\) the variational problem (7) reduces to:

Find \(F_l(u) \in V\) such that:

\[
\left[ \int_{\Omega_{k+2}} \sum_{k=-\infty}^{k=\infty} F_l \left( \frac{\partial u}{\partial \zeta} \right) F_{k-1} \left( \frac{\partial \sigma}{\partial \zeta} \right) e^{j(k-l)\zeta} \, d\zeta_1 d\zeta_3 \right] v_k = 0
\]

\[
\forall v_k \in V
\]

(8)

since five Fourier modes are enough to represent exactly the new material coefficients. For more details, see [2].

4. Numerical results

The simulations described in this chapter provide a highly accurate value of the potential of the electromagnetic field for different position of the transmitter and receiver electrodes. In order to achieve this goal, we must generate highly non-uniform meshes with high polynomial order of approximation in the
entire domain. An exemplary mesh providing such a high accuracy solution for a single position of receiver and transmitter electrodes is presented in Figure 2.

Figure 4. Real part of the solution at the receiver electrode, for different positions of the logging tool.

Figure 5. Imaginary part of the solution at the receiver electrode, for different positions of the logging tool.

The main goal of the simulation is to generate the so-called resistivity logging curves which are of great interest to the oil industry. The curves are obtained by solving either DC or AC formulation, for many positions of the receiver and transmitter antenna, for different dip angles. The simulation reflects the process of resistivity measurements by the tool shifted along the borehole.

Figures 3 presents the assumed resistivities of the formation layers. Figures 4 and 5 present the solution of an exemplary 3D AC problem, the real and the imaginary part of the electric field at the receiver electrode, for various positions of the tool.

5. Multi-frontal solver algorithm

Sequential direct solvers for finite element computations are typically frontal solvers or multi-frontal solvers. The frontal solver [5, 6] browses finite elements, one-by-one, to aggregate degrees of freedom. Fully assembled degrees of freedom are eliminated from the single front matrix. The multifrontal solver [7] is an extension of the idea of the frontal solver. It constructs the so-called elimination tree based on the analysis of the connectivity data or the geometry of the computational mesh. Finite elements are joint into pairs and fully assembled degrees of freedom are eliminated within frontal matrices associated to multiple branches of the tree. The process is repeated until the root of the assembly tree is reached. Finally, the common interface problem is solved and partial backward substitutions are recursively called on the assembly tree. The number of operations performed by the multi-frontal solver depends on shape of the elimination tree which translates directly into the sizes of the frontal matrices.

The multi frontal solver algorithm can be visualized in the following way. The degrees of freedom are related to finite element vertices, edges and interiors. Each degree of freedom has related basis function that must be integrated over elements containing the degree of freedom. Once the integrations are performed, the degree of freedom is fully assembled and can be eliminated from the frontal matrix. This procedure is illustrated in Figures 6 and 7. The figures present two finite element mesh that has been refined several times in the direction of the singularity located in the bottom part of the mesh in the middle. The Figure 6 presents the approach implemented in [JPDS]. First the smallest elements (“leaves” of the elimination tree) are eliminated, by performing the integration over them, resulting in their integral degrees of freedom to be fully assembled. This first step is denoted in Figure 7 by simply removing fully assembled the elements 14 and 17. In this example, two frontal matrices have been created, and degrees of freedom from element 14 have been assembled into the first frontal matrix while degree of freedom from element 17 have been assembled into the second frontal matrix.

The procedure will continue now with integrating over elements 11, 12 and 13 and assembling resulting degree of freedom into the first frontal matrix. Next, the integration will be performed over elements 15, 16 and 17 and resulting degree of freedom
will be assembled into the second frontal matrix. Notice however that degree of freedom located over the common edge are not fully assembled since their contributions are distributed into the first and second frontal matrix. The degrees of freedom from that interface will remain un-eliminated until we reach the top of the elimination tree and merge the two resulting frontal matrices. This is very unfortunate, since the size of both frontal matrices is constantly growing and the number of operations is the cube of the size of the frontal matrix.

In this paper we investigate the alternative solution, where we actually merge all frontal matrices adjacent to the local singularity from the very beginning so we can eliminate the common edge degree of freedom from the very beginning. In this case the solver has linear execution time $O(d^3 n)$ where $n$ is the constant size of the common frontal matrix and $d$ is the depth of the elimination tree.

6. Petri net controlling the order of elimination

The solver algorithm has been expressed as a graph grammar whereof productions attribute appropriate vertices in the elimination tree. Figures 8 – 12 present subsequent productions of the graph grammar.

Figure 7. The ordering algorithm browsing all the refinement trees together results in increasing in a constant size of the frontal matrix

Figure 8. Production P0 of the graph grammar modelling the solver algorithm

Figure 9. Production P1 of the graph grammar modelling the solver algorithm

Figure 10. Production P3 of the graph grammar modelling the solver algorithm

Figure 11. Production P4 of the graph grammar modelling the solver algorithm

Figure 12. Production P5 of the graph grammar modelling the solver algorithm

Figure 13 presents a coloured Petri net controlling the execution of the graph grammar productions realizing solver eliminations in the proper order.
The Petri net models the class of problems where the FE mesh contains multiple singularities, each of them surrounded by four mesh elements. All four elements surrounding a given singularity must be at the same adaptation level, while surroundings of different singularities can be at different adaptation levels. The adaptation levels of all singularities are encoded in the initial marking of place $P_28$. Different singularities are represented by different values of tokens of each type. The exemplary marking in Figure 13 represents two distinct singularities (distinguished with different colours). The Petri net transitions are named after the graph grammar productions (corresponding to solver eliminations) they represent.

The Petri net model utilizes three types (colours) of tokens: $m$ – type of tokens used to ensure mutual exclusion of operations performed on adjacent mesh elements (mutex); $p$ – type of tokens used to control the flow of execution of graph grammar productions represented by fired Petri net transitions; $c$ – type of tokens used to denote the adaptation level of mesh elements surrounding particular singularities.

In the initial marking the place $P_{28}$ contains $N-2$ tokens for each singularity where $N$ is the adaptation level of the mesh elements surrounding the given singularity. This Petri net model assumes that eliminations for different singularities are performed concurrently.

**Definition of the transition guard function.**

Let $In(t)$ denote the set of input arcs for transition $t$. Let $W(a)$ denote the weight of arc $a$. Let $P(a)$ denote the source place of arc $a$. Let $S = \{s_1, s_2, ..., s_N\}$ denote the set of mesh singularities. Let $Ms(p)$ denote the number of tokens corresponding to singularity $s$, present in place $p$.

Transition $t$ is active iff $G(t)$ returns $TRUE$, where $G(t)$ is defined as follows. For given transition $t$:

$$\forall s \in S \exists a \in In(t): Ms(P(a)) = W(a)$$

In other words, for a transition to be active, all activeness conditions of given transition have to be fulfilled for the same singularity.

Figure 14 presents a graph representing a single mesh element at forth level of adaptation. Figures 15 – 18 present this element transformed with P1, P3, P4 and P5 graph grammar productions respectively.
Figure 15. Graph representing the mesh element after execution of P1 graph grammar production

Figure 16. Graph representing the mesh element after execution of P3 graph grammar production

Figure 17. Graph representing the mesh element after execution of P4 graph grammar production

Figure 18. Graph representing the mesh element after execution of P5 graph grammar production
7. Comparison of the solver efficiency

We conclude the paper with Tables 1 and 2 presenting the comparison of the proposed solver execution time with classical solver, browsing elimination trees one element after another, eliminating elements from leaf up to the root. Table 1 presents the execution time of the integration algorithm, common for both approaches, while Table 2 presents the elimination algorithm, depending on the selected ordering.

Our Petri nets based solver is slower for low polynomial order of approximation than the original solver. It is because our solver contains not optimized matrix operations routines. However, when we go up with the polynomial order of approximation $p$, the Petri nets solver becomes significantly faster than the original solver. This is because our solver has much better scalability, and it becomes faster even if matrix operations are slower than in the original solver.

<table>
<thead>
<tr>
<th>$p$</th>
<th>Mesh size</th>
<th>Integration algorithm execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16,997</td>
<td>0.22 sec</td>
</tr>
<tr>
<td>3</td>
<td>38,314</td>
<td>0.81 sec</td>
</tr>
<tr>
<td>4</td>
<td>68,217</td>
<td>2.92 sec</td>
</tr>
<tr>
<td>5</td>
<td>106,686</td>
<td>9.03 sec</td>
</tr>
<tr>
<td>6</td>
<td>153,721</td>
<td>21.23 sec</td>
</tr>
<tr>
<td>7</td>
<td>209,232</td>
<td>48.56 sec</td>
</tr>
<tr>
<td>8</td>
<td>273,489</td>
<td>99.47 sec</td>
</tr>
</tbody>
</table>

Table 1. Execution time of the integration algorithm, creating frontal matrices entries for both approaches, for different sizes of meshes, for different polynomial order of approximation

<table>
<thead>
<tr>
<th>$p$</th>
<th>Mesh size</th>
<th>Petri nets solver execution time</th>
<th>Original solver execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>16,997</td>
<td>11.03 sec</td>
<td>2.33 sec</td>
</tr>
<tr>
<td>3</td>
<td>38,314</td>
<td>11.67 sec</td>
<td>2.36 sec</td>
</tr>
<tr>
<td>4</td>
<td>68,217</td>
<td>13.35 sec</td>
<td>6.27 sec</td>
</tr>
<tr>
<td>5</td>
<td>106,686</td>
<td>15.31 sec</td>
<td>14.53 sec</td>
</tr>
<tr>
<td>6</td>
<td>153,721</td>
<td>16.77 sec</td>
<td>28.81 sec</td>
</tr>
<tr>
<td>7</td>
<td>209,232</td>
<td>18.72 sec</td>
<td>49.75 sec</td>
</tr>
<tr>
<td>8</td>
<td>273,489</td>
<td>23.92 sec</td>
<td>91.47 sec</td>
</tr>
</tbody>
</table>

Table 2. Execution time of the forward elimination for different sizes of meshes, for different polynomial order of approximation

8. Conclusions

We presented the Petri net enforcing the quasi-optimal ordering for the multi-frontal solver algorithm. The ordering is enforced by controlling the order of execution of graph grammar productions over the graph representation of the mesh. The graph vertices attributed by graph grammar production denoted the mesh vertices, edges and interiors to be aggregated and eliminated into frontal matrix. The Petri nets controlled solver algorithm has been implemented and tested, showing the better efficiency than traditional ordering browsing elements one after another, eliminating nodes from leaves up to the root.

References


