A Linear Complexity Direct Solver for H-adaptive Grids With Point Singularities

Piotr Gurgul

Department of Computer Science, AGH University of Science and Technology, Krakow, Poland
pgurgul@agh.edu.pl

Abstract
In this paper we present a theoretical proof of linear computational cost and complexity for a recently developed direct solver driven by hypergraph grammar productions. The solver is specialized for computational meshes with point singularities in two and three dimensions. Linear complexity is achieved due to utilizing the special structure of such grids. We describe the algorithm and estimate the exact computational cost on an example of a two-dimensional mesh containing a single point singularity. We extend this reasoning to the three dimensional meshes.

Keywords:

1 Introduction

Direct solvers are the core part of many engineering analyses performed using the Finite Element Method [4, 5]. Since they are one of the most computationally expensive part of the process, our main objective is to deliver a solver that is by an order of magnitude faster than the existing ones in terms of both sequential and parallel execution. The finite element solution process starts with the generation of a mesh approximating the geometry of the computational problem. Next, the physical phenomena governing the problem are described using Partial Differential Equation (PDEs). In addition to this differential system, boundary and initial conditions may need to be specified to guarantee uniqueness of the solution. The resulting PDE system is discretized into a system of linear equations by using the Finite Element Method. This algebraic system is inverted using a solver algorithm. The computational complexity of the state-of-the-art multifrontal solver [6, 7] algorithm is \( O(Np^4 + N^{1.5}) \) for two dimensional problems in regular and uniform grid. \( N \) refers to the number of degrees of freedom and \( p \) is a global polynomial order of approximation, which we assume to be constant, yet arbitrary.

In this paper we present a direct solver algorithm delivering \( O(Np^3) \) computational complexity for meshes with point singularities in two and three dimensions. The computational cost and memory usage for the direct solver estimated in this paper uses the ideas expressed by the graph grammar based solver [19] prescribing the order of elimination for grids with point
singularities. The graph grammar formalism is utilized for modeling of mesh adaptation by means of CP-graphs [11, 12, 13, 18] or hypergraphs [17]. It can be also used for expressing the solver algorithm [16, 15]. Finally, the results presented in this paper can be also extended for the idea of reutilization [14], which further generalizes a concept of a linear solver for a sequence of grids.

2 2D multi-frontal sequential solver algorithm

Problems with singularities usually require direct solvers, since they are not suitable for iterative solvers due to their numerical unstability [10]. There exist iterative solvers that converge for a certain class of variational formulations, but there are no iterative solvers per se that converge for every FEM problem (as opposed to direct solvers). Moreover, even the multi-grid iterative solvers [3], more suitable for processing adaptive grids, usually still utilize a direct solver over a coarser grid. This makes the use of direct solvers absolutely necessary for designing a general purpose FEM solver. Since the matrices resulting from FEM discretization are sparse, we focus on algorithms designed for sparse matrices such as a multi-frontal solver. The multi-frontal solver introduced by Duff and Reid [6, 7] is the state-of-the-art direct solver algorithm for solving systems of linear equations, which is a generalization of the frontal solver [9] algorithm. In the multi-frontal approach, the solver generates multiple frontal matrices for all elements of the mesh. It eliminates fully assembled nodes within each frontal matrix, and merges the resulting Schur complement matrices at the parent level of the tree. The frontal matrix assembles element frontal matrices to a single frontal matrix and eliminates what is possible from the single matrix, whereas the multi-frontal solver utilizes multiple frontal matrices and thus allows to reduce the size of the matrices at parent nodes of the tree.

3 Linear complexity solver for 2D problems with point singularities

In this section we describe briefly the idea behind solvers for problems with point singularities and what are their advantages over multi-frontal solvers like MUMPS [1, 2]. The proposed linear complexity algorithm assumes an \( h \)-refined computational mesh structure and browses the elements of the mesh level by level, from the lowest (most refined) level up to the top level. The algorithm utilizes a single frontal matrix, which size is proportional to the number of browsed degrees of freedom. There are \( p^2 \) degrees of freedom associated with element interiors, \( p \) degrees of freedom associated with element edges and one degree of freedom for each element vertex. Each row of the frontal matrix is associated with one degree of freedom. The solver algorithm finds the fully assembled degrees of freedom and places them at the top of the frontal matrix. Then, these rows are eliminated by subtracting them from all the remaining rows. Intuitively, we can say that element interior degrees of freedom are fully assembled once the solver touches the element, an element edge degrees of freedom are fully assembled once the solver algorithm touches all elements adjacent to an edge, and finally element vertex degrees of freedom are fully assembled once the solver algorithm touches all elements sharing the vertex. The remaining degrees of freedom must wait until the solver algorithm browses the elements from the next level. The size of the frontal matrix is limited by the number of elements on a particular level, and this implies the linear computational cost of the solver algorithm. Moreover, the behavior of the solver described above has been prescribed by the means of hypergraphs [19] introduced
by Habel and Kreowski [8] and hypergraph grammars which were their extension proposed by Ślusarczyk [17] for mesh transformations.

4 Estimation of the computational cost

In this section we estimate the total number of operations for the 2D sequential solver described above. This will be used to obtain computational complexity. In order to compute the number of operations, we define variable $a$ as the number of degrees of freedom that can be eliminated for matrix $M$ and $b$ as the total count of degrees of freedom for matrix $M$ (see Figure 1). For a given node, the total number of operations needed for its elimination $C(a, b)$ is equal to:

$$C(a, b) = \sum_{m=b-a+1}^{b} m^2$$

(1)

This is because we have a matrix with size $b$ and there are $a$ rows to be eliminated. The elimination of the first row involves $b^2$ subtractions, the elimination of the second row involves $(b-1)^2$ subtractions, and so on, up to the last row to be eliminated, which involves $(b-a+1)^2$ subtractions. Rewriting the sum in Equation 1 as a difference of two sums, we obtain Equation 2.

$$\sum_{m=b-a+1}^{b} m^2 = \sum_{m=1}^{b} m^2 - \sum_{m=1}^{b-a} m^2$$

(2)

which can be explicitly written using the following sum of squares:

$$\sum_{m=1}^{b} m^2 = \frac{b(b + 1)(2b + 1)}{6}$$

(3)

$$\sum_{m=1}^{b-a} m^2 = \frac{(b-a)(b-a+1)(2b-a+1)}{6}$$

(4)

Finally, we perform the subtraction:

$$\sum_{m=1}^{b} m^2 - \sum_{m=1}^{b-a} m^2 = \frac{a(6b^2 - 6ab + 6b + 2a^2 - 3a + 1)}{6}$$

(5)
Thus:

\[ C(a, b) = \frac{a(6b^2 - 6ab + 6b + 2a^2 - 3a + 1)}{6} \]  

(6)

Using this formula, we can directly compute the costs for the sample 2D domain presented in the previous sections.

In order to prove that the computational complexity is \( O(N) \), where \( N \) denotes the total number of unknowns, first, in Lemma 4.1, we express the exact computational cost as a function of polynomial approximation level \( p \) and refinement level \( k \).

**Lemma 4.1.** The number of operations incurred by solving the adaptive problem with a point singularity for a two element mesh using a hypergraph grammar driven solver is equal to:

\[
T(p, k) = \frac{16p^6 + 96p^5 + 264p^4 + 864p^3 + 533p^2 + 93p}{6} \\
+ k \frac{12p^6 + 72p^5 + 198p^4 + 1558p^3 - 441p^2 - 277p - 24}{6}
\]

where \( k \) stands for the number of \( h \) refinement levels and \( p \) is the global polynomial order of approximation. Parameter \( p \) is assumed to be uniform, yet arbitrary, over the entire mesh.

**Proof.** With Equation 6 it is possible to evaluate the cost of each step of the elimination process. The proof can be split into two independent steps. First, we compute the constant part associated with the elimination of the initial, unrefined mesh (\( k = 0 \)) and then the count of operations incurred by the each increase of \( k \) by one. This is sufficient, since each layer produced as a result of incrementing \( k \) is identical and contains six new elements.

**Step 1.** \( k = 0 \). We begin by computing the cost of eliminating the fully assembled nodes. This process is called *static condensation*. To eliminate entries produced by these nodes we do not need to know about any other degrees of freedom. Static condensation usually occurs just after attaching an element to the given structure. As such entries are self-contained and can be processed independently, we can count their contributions to the total number of operations independently. For each element 1, 2, 5, 6 (see Figure 2a) the interior, two edges and a vertex are fully assembled and can be eliminated immediately (or at any time). For each element 3, 4, 7, 8 (see Figure 2b) interior and only one edge are fully assembled. The number of operations incurred by each of two cases above has been summarized in Table 1. Column TOTAL is the
sum of contributions for each of the elements in the Table 1 (which means multiplying the value in the first row by four and adding it to the value in the second row, also multiplied by four). The next step is to count the operations incurred by the interface elimination, still for the $k = 0$ two element grid. This time, we need to obey the proper order for the elimination, starting from what is operationally called Interface 1-2. In general, during this step we follow the previously described order of elimination, which is concisely summarized in Figure 3. The results are presented in Table 2. Now we have finished the first step of computations and received the constant part for $k = 0$, which can be expressed as $T(p, 0)$ in Equation 7.

$$T(p, 0) = \frac{728p^3 + 552p^2 + 106p + 16p^6 + 96p^5 + 264p^4 + 136p^3 - 19p^2 - 13p}{6}$$

**Step 2.** The second step of the proof is to compute the linear increase of the operation count depending on $k$. Increasing $k$ by one adds one more layer of elements as presented in Figure 4. Again, for simplicity, we introduce temporary naming for the consecutive interfaces (Interface 1-2 to Interface 7-8) that will be processed (which corresponds to $3 + 6k$ to $8 + 6k$). Let $k \geq 1$. 

### Table 1: Operations incurred by static condensation for $k = 0$

<table>
<thead>
<tr>
<th>Elements</th>
<th>$a$ for arbitrary $p$</th>
<th>$b$ for arbitrary $p$</th>
<th>$C(a, b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 4, 7, 8</td>
<td>$1 + 2(p - 1) + (p - 1)^2$</td>
<td>$4 + 4(p - 1) + (p - 1)^2$</td>
<td>$\frac{2p^6 + 12p^5 + 33p^4 + 36p^3 + 13p^2}{6}$</td>
</tr>
<tr>
<td>1, 2, 4, 6</td>
<td>$p - 1 + (p - 1)^2$</td>
<td>$4(p - 1) + (p - 1)^2$</td>
<td>$\frac{2p^6 + 12p^5 + 33p^4 + 2p^3 - 32p^2 - 13p}{6}$</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td>$\frac{16p^6 + 96p^5 + 264p^4 + 136p^3 - 19p^2 - 13p}{6}$</td>
</tr>
</tbody>
</table>

### Table 2: Computational cost incurred by element elimination for $k = 0$

<table>
<thead>
<tr>
<th>Interface</th>
<th>$a$ for arbitrary $p$</th>
<th>$b$ for arbitrary $p$</th>
<th>$C(a, b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-2</td>
<td>$1 + p - 1$</td>
<td>$6 + 5(p - 1)$</td>
<td>$\frac{122p^3 + 81p^2 + 13p}{6}$</td>
</tr>
<tr>
<td>2-3</td>
<td>$1 + p - 1$</td>
<td>$6 + 5(p - 1)$</td>
<td>$\frac{122p^3 + 81p^2 + 13p}{6}$</td>
</tr>
<tr>
<td>3-4</td>
<td>$1 + p - 1$</td>
<td>$6 + 5(p - 1)$</td>
<td>$\frac{122p^3 + 81p^2 + 13p}{6}$</td>
</tr>
<tr>
<td>4-5</td>
<td>$2 + 2(p - 1)$</td>
<td>$6 + 5(p - 1)$</td>
<td>$\frac{196p^3 + 144p^2 + 26p}{6}$</td>
</tr>
<tr>
<td>5-6</td>
<td>$2 + 2(p - 1)$</td>
<td>$5 + 4(p - 1)$</td>
<td>$\frac{112p^3 + 108p^2 + 26p}{6}$</td>
</tr>
<tr>
<td>6-7</td>
<td>$1 + (p - 1)$</td>
<td>$4 + 3(p - 1)$</td>
<td>$\frac{38p^3 + 45p^2 + 13p}{6}$</td>
</tr>
<tr>
<td>7-8</td>
<td>$3 + 2(p - 1)$</td>
<td>$0$</td>
<td>$\frac{16p^3 + 12p^2 + 2p}{6}$</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td>$\frac{728p^3 + 552p^2 + 106p}{6}$</td>
</tr>
</tbody>
</table>
Figure 3: Interface elimination order, $k = 0$
As in Step 1, we begin with the contribution of static condensation. In the case of elements $3+6k$ and $8+6k$ we can eliminate for each one edge and its interior independently. For the remaining elements the elimination of their interiors is the only possibility at this point. The results has been summarized in Table 3. In terms of eliminating interfaces, as in Step 1, we have to follow the order, but the direction (left to right or right to left) is not relevant to the result, so we may assume that we are proceeding from right to left. For a summary of the results, see Table 4. Having all the terms, it is now possible to assemble the final formula:

$$\text{TOTAL} = 12p^6 + 72p^5 + 198p^4 - 308p^3 - 60p^2 + 62p + 24$$
\[ T(p, k) = T(k, 0) + k \frac{12p^6 + 72p^5 + 198p^4 + 1558p^3 - 441p^2 - 277p - 24}{6} \]  

(8)

Thus:

\[ T(p, k) = \frac{16p^6 + 96p^5 + 264p^4 + 864p^3 + 533p^2 + 93p}{6} + k \frac{12p^6 + 72p^5 + 198p^4 + 1558p^3 - 441p^2 - 277p - 24}{6} \]  

(9)

which completes the proof.

**Lemma 4.2.** The computational complexity of the hypergraph grammar driven solver for problems with point singularities is equal to \( O(N) \), where \( N \) is the number of unknowns.

**Proof.** The proof relies on Lemma 4.1 and transforms \( T(p, k) \) into a function of \( N \). In order to achieve this goal, we first need to determine the relationship between the number of elements \( N_e \) and the number of refinement levels \( k \). The initial mesh for \( k = 0 \) has eight elements. Each new refinement level adds a layer of six new elements. Thus:

\[ N_e(k) = 8 + 6k \]  

(10)

Since each element contains 4 degrees of freedom associated with its vertex, \( 4(p - 1) \) degrees of freedom associated with its edges and \( (p - 1)^2 \) degrees of freedom associated with its interior, we multiply number of elements and number of degrees of freedom per element.

\[ N(N_e, p) = (8 + 6k)((p - 1)^2 + 4(p - 1) + 4) = O(p^2 + kp^2) \]  

(11)

Substituting this result in \( T(p, k) \) we receive:

\[ T(p, N) = O(Np^4) \]  

(12)

Since \( p \) can be treated as a constant and in fact, due to numerical side-effects is very rarely greater than nine, we receive:

\[ T(N) = O(N) \]  

(13)

which proves the linear complexity.

\[ \square \]

5 Estimation of the memory usage

Memory usage can be estimated following the same pattern as in the previous section. Order of the memory usage of the solver can be well-approximated by computing the count of non-zero entries in the matrix. Since we do not store zero values, the real space usage is the function of non-zero entries (NZ). NZ at a given stage can be expressed as a function of previously defined \( a \) and \( b \), like in Equation 14.

\[ NZ(a, b) = \frac{a(a + 1)}{2} + a(b - a) \]  

(14)

Applying the same reasoning as in terms of computational complexity leads as to the following equation:

\[ MEM(k, p) = NZ(k, p) = \frac{1}{2} (12p^4 + 48p^3 + 11p^2 + 35p + 4) + k \frac{1}{2} (105p^2 - 29p - 8) = O(p^4 + kp^2) = O(N) \]  

(15)
6 Generalization to the three dimensional grids

As an estimation of the exact computational cost for the three dimensional solver would be a very strenuous task, we restrict ourselves to a rough approximation of the computational complexity.

**Lemma 6.1.** Computational complexity of the sequential solver with respect to the number of degrees of freedom $N$ and polynomial order of approximation $p$ for a three dimensional grid with point singularity is equal to $T(p, N) = O(Np^6)$.

**Proof.** A three dimensional element has number of degrees of freedom over an element edge of the order of $O(p)$, the number of degrees of freedom over an element face of the order of $O(p^2)$ and the number of degrees of freedom over an element interior of the order of $O(p^3)$. The computational complexity of elimination of the interior-related degrees of freedom is of the order of $O((p + p^2 + p^3)^2p^3) = O(p^9)$. The computational complexity of the static condensation is of the order of $O(N_e p^9)$, where $N_e$ denotes the number of elements. The remaining faces and edges are eliminated level by level (layer by layer), and the computational complexity of elimination of a single level is of the order of $O((p^2 + p)^3) = O(p^6)$. The number of elements $N_e$ is of the order of $O(N_e) = O(N_e^{1/3})$, and the number of levels $k$ is of the order of $O(k) = O(N_e^{1/3})$ thus the total computational complexity is of the order of $O(N_e p^9 + kp^6) = O(Np^6 + Np^3) = O(Np^6)$ which completes the proof.

7 Conclusions and future work

This paper contains a proof of the linear computational cost of the proposed direct solver algorithm executed on the $h$-refined computational grids with point singularities. This results show that the $h$-refined grids can be solved in linear cost $O(Np^3)$ as opposed to the traditional solver algorithms delivering $O(Np^4 + 1.5)$ cost for regular grids. The theoretical estimates has been proven by a series of experiments for $p = 2$, $p = 3$ and $1 - 8$ point singularities. An interesting attempt would be to estimate solver’s complexity in case of different type of singularities such as edge singularities.

**Acknowledgements** The work presented in this paper has been supported by Dean’s grant no. 15.11.230.106.

References


