Towards green multi-frontal solver for adaptive finite element method

H. AbbouEisha¹, M. Moshkov¹, K. Jopec², P. Gepner³, J. Kitowski², and M. Paszyński²

¹ King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
² AGH University of Science and Technology, Krakow, Poland
³ Intel Corporation, Swindon, United Kingdom

Abstract
In this paper we present the optimization of the energy consumption for the multi-frontal solver algorithm executed over two dimensional grids with point singularities. The multi-frontal solver algorithm is controlled by so-called elimination tree, defining the order of elimination of rows from particular frontal matrices, as well as order of memory transfers for Schur complement matrices. For a given mesh there are many possible elimination trees resulting in different number of floating point operations (FLOPs) of the solver or different amount of data transferred via memory transfers. In this paper we utilize the dynamic programming optimization procedure and we compare elimination trees optimized with respect to FLOPs with elimination trees optimized with respect to energy consumption.

Keywords: multi-frontal direct solver, adaptive finite element method, green computing

1 Introduction
The problem of optimization of energy consumption during large scale scientific computations is of the great importance to the scientific community. There are some works addressing the optimization of the computational environments to reduce the energy consumed during the scientific computations [20, 27]. Some recent papers address also the energy consumption for different implementations of the computational algorithms. In particular it is emphasized that memory access is more energy consuming than FLOPs [16, 30]. In this paper we focus on the optimization of energy consumption during the Finite Element Method (FEM) computations.

FEM is a widely used approach [22, 35, 15, 19] to find approximate solutions for partial differential equations (PDEs). Those PDEs are specified along with boundary conditions and the domain of the solution. First, a mesh of rectangular or triangular elements is used to cover the specified domain and approximates the solution over it. Second, the PDEs are transformed into so-called weak form and discretized using basis of polynomial functions spread over finite elements vertices, edges and interiors. Afterwards, we specify a linear combination of a set
of basis functions to approximate the solution of the discretized PDEs. Finally, we can use a solver that finds the coefficients of the linear combination of the basis functions to obtain an approximate solution of the input PDE.

As the result of the discretization of the PDE over the computational mesh, a global system of linear equations is generated. The system is spread over several frontal matrices, one frontal matrix per one finite element over the mesh. The rows and columns in these matrices correspond to basis functions associated with element vertices, edges and interiors. A frontal matrix over a single finite element contains rows and columns corresponding to basis functions associated with this element vertices, edges and interior. The global matrix can be obtained by merging these frontal matrices according to common mesh nodes. However the solver algorithm presented in this paper works with the global matrix distributed over particular finite elements and performs partial factorizations followed by merging of pairs of frontal matrices. The pattern how frontal matrices are factorized and merged is prescribed in the elimination tree, with leaves of the tree associated with frontal matrices generated over particular mesh nodes. The goal of the solution process is to solve the considered PDE in its discretized form over the geometry represented by the mesh.

The computational mesh can be further refined, by breaking selected finite elements into smaller one. In the case of two dimensional regular meshes, where initial elements cover blocks with different parameters of the PDE, we can distinguish point, edge, and mixed singularities. In these cases mesh refinements are performed towards one or several points, one or several lines located at the border of elements, or towards mixture of such objects.

In this paper we focus on multi-frontal direct solver algorithm [17, 18], the state-of-the-art solver for obtaining the exact solution of a linear systems generated by FEM. The multi-frontal solver algorithm constructs many frontal matrices, usually associated with particular finite elements. It eliminates fully assembled rows in these matrices, and merges resulting sub-matrices (called Schur complements) into new frontal matrices. This process is repeated recursively until all the rows are fully assembled and can be eliminated.

Elimination trees define the order in which the solver manage the process of eliminations. Number of floating point operations (FLOPs), memory usage, memory transfers and the energy consumption of the solver depend essentially on the structure of the elimination tree. The problems of construction of the optimal elimination trees, with respect to either FLOPs, memory usage, memory transfers or energy consumption, or a combination of the above factors, are very complicated. There are some problems related to the optimization of the elimination trees which are NP complete [32, 33]. There are several heuristic algorithms utilized for construction of the elimination tree optimal with respect to the number of FLOPs. The most important heuristic algorithms are available with MUMPS multi-frontal direct solver [5, 6, 7, 28]. These are the nested-dissections [23], implemented in METIS library [24], minimum degree algorithms [21], approximate minimum degree [3, 4, 14], quasi-approximate minimum degree [12, 13] or PORD algorithm [31]. There are several variations of these algorithms like approximate minimum degree AMD etc. [3, 4]. It is well known fact that the nested-dissections algorithm provides elimination tree delivering optimal number of FLOPs for uniform grids [26].

In our previous work [1, 2] we proposed a dynamic programming algorithm finding alternative elimination trees for adapted non-uniform grids, refined towards point or edge singularities. We considered the optimization with respect to FLOPs number.

In this paper we perform an automatic search for alternative elimination trees providing optimal energy consumption of the multi-frontal solver algorithm for the class of non-uniform refined meshes. We compare our optimal elimination trees to the one delivering optimal FLOPs number.
2 Finite Element Method, Multi-Frontal Direct Solvers and Elimination Trees

In this section we introduce the state-of-the-art multi-frontal solver algorithm on the simple example of four finite element mesh with second order polynomial basis functions spread over finite elements.

The FEM [15] starts with two dimensional boundary-value elliptic PDE transformed into so-called weak (variational) formulation of the form (1): Find $u \in V$ such that

$$b(u, v) = l(v) \quad \forall v \in V$$

where $b(u, v)$ and $l(v)$ are some problem dependent bilinear and linear functionals, and

$$V = \{v : \int_{\Omega} \left\| v \right\|^2 + \left\| \nabla v \right\|^2 dx < \infty, \text{tr}(v) = 0 \text{ on } \Gamma_D\}$$

is the functional Sobolev space over an open set $\Omega$ called the domain, and $\Gamma_D$ is the part of boundary of $\Omega$ where Dirichlet boundary conditions are defined.

For a given domain $\Omega$ the $hp$-FEM consists in constructing a finite dimensional subspace $V_{hp} \subset V$ with a finite dimensional polynomial basis $\{e_{hp}^i\}_{i=1,...,N_{hp}}$. The subspace $V_{hp}$ is constructed by partitioning the domain $\Omega$ into so-called finite elements. In this paper we restrict our considerations to rectangular elements in two dimensions or hexahedral elements in three dimensions. The basis functions are constructed by gluing together the so-called shape functions constructed over vertices, edges, and interiors of finite elements in two dimensions, or over vertices, edges, faces, and interiors of finite elements in three dimensions.

Figure 1 presents an exemplary two-dimensional mesh consisting of rectangular finite elements with vertices, edges and interiors, as well as shape functions defined over vertices, edges and interiors of rectangular finite elements of the mesh. We introduce four shape functions over the four vertices of the two dimensional rectangular element, $p_i - 1$ shape functions over each of the four edges of the element, and $(p_n - 1) \times (p_n - 1)$ shape functions over an element interior [15]. These basis functions are utilized to approximate a solution of the weak form of the PDE being solved.
In the example presented in Figure 1 there are first order polynomial basis functions associated with element vertices, second order polynomials associated with element edges, and second order polynomials in both directions, associated with element interiors. The Figure presents only selected basis functions, associated to vertices number 1, 3, 5, 11, 13 and 15, edges 2, 4, 8, 12, 14 and 18 as well as all interiors. For more details we refer to [15].

2.1 Multi Frontal Solver Algorithm and Elimination Trees

In this section we present an overview of the multi-frontal solver algorithm executed over the exemplary mesh presented in the previous section. Among several possible elimination trees we select one exemplary presented in Figure 2. It has to be emphasized, that this is only one possible elimination tree for the mesh from Figure 1, and this elimination tree has been selected here just for the illustration of the solver algorithm and cost function definition. The root of the elimination tree is the entire mesh, the son nodes contains recursive partitions of the mesh, and the leaves contain particular finite elements.

Figure 3 presents the processing of the first branch of the multi-frontal solver algorithm associated with the top two elements, followed by processing of the first branch of the multi-frontal solver algorithm associated with the bottom two elements, followed by the solution of the top interface problem. In this case the solver starts with generating element frontal matrices for both elements (panels (a), (b)). The rows and columns in these matrices contain basis functions defined over the mesh. The matrix entries corresponds to integrals of the multiplications of the basis functions, which are PDEs dependent. The solver localizes all the rows of the frontal matrices that are fully assembled. In this case, the interior node is fully assembled, since the basis function associated with the interior nodes 7 and 9 (compare functions $e_7^{hp}$ and $e_9^{hp}$ in Figure 1) have support defined only inside the corresponding single element. Additionally, the rows corresponding to boundary basis functions related with nodes 1, 2, 6, and 4, 5, 10 are fully assembled in the first or second frontal matrix, respectively. This is because their supports are also located only inside the corresponding single element. However, we must keep all the nodes located on the interface with the two bottom elements.

All these rows can be eliminated from the matrix (panels (a), (b)), and what remains are the rows associated with the basis functions 3, 8 located on the common edge and 11, 12, 13, 14, 15 located on the interface with the bottom two elements. The multi-frontal solver merges the resulting sub-matrices (panel (c) reorders them to eliminate first two rows (panel (d)). The same procedure is repeated for the bottom two elements (panels (e)-(h)). Finally, the interface problem associated to nodes 11, 12, 13, 14 and 15 is formulated and solved (panels (i) and (j)).

3 Cost functions

We can define several cost functions measuring performance of the multi-frontal solver algorithm based on the elimination tree.

The first cost function is the one measuring the FLOPs, as reported in [2]. The functions estimates the number of FLOPs during particular partial forward eliminations, performed in steps (a), (b), (e), (f) and (j) as presented in Figure 3. The meaning of these steps have been described in section 2.1. They correspond to processing of the elimination tree presented in Figure 2. The number of FLOPs is estimated by using the formula (3)

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

(3)
Figure 2: Multi-frontal solver algorithm processing the first branch of the elimination tree related to the two first elements.

Figure 3: Multi-frontal solver algorithm processing the first branch of the elimination tree related to the top first elements (panels (a)-(d)), followed by processing the second branch of the elimination tree related to the bottom first elements (panels (e)-(h)), followed by the solution of the common interface problem (panels (i)-(j)).

where \( a \) denotes the number of rows to be eliminated, and \( b \) the local size of the frontal matrix, compare [1]. In our example in steps (a), (b), (e) and (f) \( b = 9 \) and \( a = 4 \), while is step (i) \( a = b = 5 \).

In this paper we propose a new cost function, measuring the memory transfers necessary to form a new frontal matrices out of the obtained Schur complements. The number of memory
Table 1: FLOPs and memory transfers on a sample four element domain for multi-frontal solver

<table>
<thead>
<tr>
<th>Step</th>
<th>a</th>
<th>b</th>
<th>FLOPs(a, b)</th>
<th>Step</th>
<th>a</th>
<th>b</th>
<th>MEM(a, b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>9</td>
<td>374</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>9</td>
<td>374</td>
<td>2</td>
<td>4</td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>133</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>9</td>
<td>374</td>
<td>4</td>
<td>4</td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>9</td>
<td>374</td>
<td>5</td>
<td>4</td>
<td>9</td>
<td>25</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>7</td>
<td>133</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>5</td>
<td>70</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

TOTAL FLOPs 1832 TOTAL memory transfers 150

transfers is given by (4)

\[ MEM(a, b) = (b - a)^2. \]  

This is because the size of the Schur complement is equal to the size of the local system minus the size of the rows that can be eliminated in the local system. The Schur complements are transferred then to the parent level where they are merged into a new local system. Using these formula, we can directly compute the costs for the sample 2D domain presented in the previous sections, compare Table 1. The FLOPs considered above concerns only the factorization step, they do not consider the backward/forward substitutions (the so called solution phase). This is because the factorization step is the most expensive one and the cost of the solution phase can be neglected when we optimize the elimination tree for the multi-frontal solver algorithm.

4 Optimization algorithm

We utilize the same optimization algorithm as the one presented in [1, 2], however with the cost function updated to reflect both FLOPs and memory transfers. Additionally, we assume that energy consumption for the memory transfers is around 100 time higher than for FLOPs. It is of course a simplified assumption, and it assumes order of magnitude and does not include specific architecture design. However 96% of processors is based on Intel, include the Xeon. The differences of sizes of L1, L2 and L3 caches is not taken into account. The architecture of memory is also not taken into accout, however for both DDR3, and IMC for Xeon the architectures are similar. Taking all that into account our estimation is correct. For intel processor we have around 10-20 pJ per FLOP, and 1000 pJ per Memory Access (DRAM Access). For a Xeon phy we have L1 Hit=39pJ, L2/L3 Hit=389 pJ, and DRAM Access=1320 pJ. Based on these data we can assume that FLOP/Memory Access ratio is around 100.

In our simple example the energy consumption cost is equal \( ENERGY = 1832 + 150 \times 100 = 16832 \).

5 Numerical results

The presented optimization algorithm has been executed on two dimensional computational grids with point and edge singularities. The computations has been performed on 1 core of
Intel(R) Xeon(R) CPU E7- 4860 @ 2.27GHz, with DDR3, 1066 MHz according to [36].

We compare the execution times for the elimination trees optimized with respect to FLOPs to the trees optimized with respect to energy consumption. The elimination trees obtained from our optimization procedures have been submitted to our GALOIS solver [29], and the resulting execution times has been measured. The comparison is presented in Figure 4 for the mesh with edge and point singularity. We have also performed the curve fitting algorithm to estimate the exponent factor of the trends. We can draw the following conclusions from the presented experiments:

- The energy optimal elimination tree for mesh with edge singularity results in $O(N^{1.6831})$ execution time, while the FLOPs optimal elimination tree results in $O(N^{0.963})$ execution time. In other words the price to pay for the green solver is the lost of linear computational cost of the solver.

- The energy optimal elimination tree for mesh with point singularity is practically identical with the mesh optimizing the execution time.

![Figure 4: Execution time of FLOPs vs ENERGY optimal solvers for 2D mesh with edge singularity (left panel) and point singularity (right panel). Both plots uses log-log scales on both axes.](image)

Let us compare the elimination trees for the case of edge singularity. The tree optimizing FLOPs, following the result presented in [1] is presented on left panel in Figure 5. The new tree optimizing the energy is presented on right panel in Figure 5. The tree optimizing energy prefers longer slices of elements.

Let us also compare the elimination trees for the case of point singularity. The tree optimizing FLOPs, following the result presented in [1] is presented on left panel in Figure 6. The new tree optimizing the energy is presented on right in Figure 6. The trees for point singularity are practically identical, and they differ only by order of cutting the levels. Note that the optimization of FLOPs does not necessary leads to faster execution times, since memory transfers also take up to two orders of magnitude more time as the FLOPs.

6 Conclusions

In this paper we compared elimination trees controlling the execution of the multi-frontal direct solver algorithm over refined grids, optimized with respect to the number of floating point operations (FLOPs) as well as energy consumption. As a criterion for energy consumption
cost we assumed that FLOPs results in 100 less energy consumption than memory transfers. The execution time of both FLOPs optimized and energy optimized solvers have been compared on two dimensional grids with point and edge singularity. We conclude that optimizing the solver with respect to energy consumptions results in slight increase of the execution time. The optimization of the energy consumption over model refined 2D grids may be generalized to more complex refined grids by decomposition of the grid into subgrids with either point or edge singularity, and building standard nested-dissection tree on top of it. Additionally, for non-linear or non-stationary problems the energy optimization may be performed in every iteration. Future work may include the extension of the optimization algorithm to different
grids, including 3D grids with point, edge, face and mixed singularities. We may also consider different implementations of the solver algorithm, including different memory transfers, and test them on grids with singularities as well as on the uniform grids, and for transient problems [10]. Another topic is the optimization of energy during numerical integration [34, 11, 25].

Acknowledgement The work presented in this paper has been supported by Polish National Science Center grant no. DEC-2012/07/B/ST6/01229 and by King Abdullah University of Science and Technology (KAUST). The work of Konrad Jopek has been supported by Dean’s grant.

References


Towards green multi-frontal solver for adaptive finite element method

H. AbbouEisha, et. al.

(2006)


[28] MUlti-frontal Massively Parallel Sparse direct solver (MUMPS) http://mumps.enseeiht.fr/


