Parallel Self-Adaptive
hp Finite Element Method Algorithm with Shared Data Structure

Maciej Paszyński¹, David Pardo², Carlos Torres-Verdin³

¹ Department of Computer Science
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Kraków, Poland
e-mail: paszynsk@agh.edu.pl
², ³ Department of Petroleum and Geosystems Engineering,
The University of Texas at Austin,
1 University Station C0300, TX, 78712

ABSTRACT

In this paper we present a new parallel algorithm of the self-adaptive hp Finite Element Method (hp-FEM) with shared data structures. The algorithm generates in a fully automatic mode (without any user interaction) a sequence of meshes delivering exponential convergence of the prescribed quantity of interest with respect to the mesh size (number of degrees of freedom). The sequence of meshes is generated from the prescribed initial mesh, by performing h (breaking elements into smaller elements) or p (adjusting polynomial orders of approximation) or hp (both) refinements on selected finite elements. The new parallel implementation utilizes a computational mesh shared between multiple processors. All computational algorithms, including automatic hp adaptivity and the solver, work fully in parallel. We will present details of the parallel self-adaptive hp-FEM algorithm with shared computational domain, as well as its efficiency measurements. The presentation will be enriched by numerical results of the problem of 3D AC/DC borehole resistivity measurement simulations.

INTRODUCTION

The parallelization of the adaptive hp-FEM algorithm belongs to the current state of the art, difficult research area. The hp adaptation consists in breaking selected finite elements into smaller elements (thus h stands for an element size) and changing polynomial order of approximation accordingly (thus p stands for a polynomial order of approximation). There are the following mesh adaptation techniques (Demkowicz 2006):

a) Uniform h adaptation, where all finite elements are uniformly broken into smaller elements.

b) Uniform p adaptation, where polynomial order of approximation is increased uniformly over the entire mesh, e.g. by adding bubble shape functions of the higher orders over element edges and interiors.
c) *Non-uniform h adaptation*, where some finite elements are broken into smaller elements, only in parts of the mesh with high numerical error.

d) *Non-uniform hp adaptation*, where some finite elements are broken into smaller elements, and polynomial orders of approximation are increased over some finite elements, only in these parts of the mesh where numerical error is high.

The non-uniform hp adaptation is the most sophisticated strategy, in the sense it provides an exponential convergence of the numerical error with respect to the mesh size. For non-uniform h or hp adaptation it is necessary to localize finite elements with high numerical error. In the case of the non-uniform hp adaptation it is also necessary to select optimal refinements for such finite elements. The non-uniform h or hp adaptation can be realized in the following ways.

   a) The selection of the finite elements to be refined and the refinement kind is performed by the user, by utilizing some interface with the adaptation code.

   b) The selection of the finite elements to be refined and the refinement kind is performed by some algorithm utilizing the knowledge of the structure of the solution.

   c) The selection of the finite elements to be refined and the refinement kind is performed by the self-adaptive algorithm, without the particular knowledge of the structure of the solution, in the fully automatic mode, without any user interaction.

   The first and the second algorithm are called the *non-automatic adaptation*, while the third algorithm is called the *automatic adaptation*. In particular, the non-uniform hp adaptation with the automatic adaptation is called the *self-adaptive hp Finite Element Method* (self-adaptive hp-FEM).

   There are some parallel algorithms supporting non-automatic mesh adaptation. Among major undertakings to develop the algorithms supporting h refinements over the computational mesh distributed into sub-domains, one has to list first of all the Sierra Environment (Edwards 1997, Edwards 2002, Edwards and Stewart and Zepper 2002). The algorithms implemented in the framework have been used to parallelize several finite element applications developed at Sandia National Laboratories (USA) (Edwards and Stewart and Zepper 2002). The environment allows for an arbitrary domain partitioning of a current mesh. The parallel hp adaptive algorithms for Discontinuous Galerkin (DG) methods were developed by Remacle and Xiangrong and Shephard and Flaherty 2000 and Banaś 2003. The only parallel hp adaptive algorithms for Continuous Galerkin method that we are aware of were developed by Patra 1999, Laszlofy, Long, Patra, 2000, Bauer, Patra 2004. None of these algorithms support automatic hp adaptation.

   The self-adaptive hp Finite Element Method (hp-FEM) for two and three dimensional elliptic and Maxwell problems were designed and implemented by the group of Leszek Demkowicz (Demkowicz 2006, Demkowicz and Kurtz and Pardo and Paszyński and Rachowicz and Zdunek 2007). The codes generate a sequence of hp meshes providing exponential convergence of the numerical solution with respect to the mesh size. The parallel version of the two and three dimensional algorithms have been designed and implemented based on the *distributed domain decomposition* paradigm, illustrated on the left panel in Fig. 1. (Paszyński and Kurtz and Demkowicz 2006, Paszyński and Demkowicz 2006).

   The main disadvantage of the distributed domain decomposition base parallel code were huge complexity of the mesh transformation algorithms executed over the computational mesh stored in distributed manner. There are the following mesh regularity rules: 1) *the one irregularity rule*, telling that a finite element cannot be broken for the
second time without first breaking larger adjacent elements, and 2) the minimum rule, telling that the order of approximation over a face must be equal to the corresponding orders of approximation from adjacent element interiors, and the order of approximation over an edge must be equal to corresponding orders of approximation from adjacent faces. The main technical difficulty was to maintain these mesh regularity rules over the computational mesh partitioned into sub-domains, e.g. a refinement performed over one sub-domain may require a sequence of additional refinements over adjacent elements, possibly located at adjacent sub-domains. A partial solution to the problem was the introduction of the ghost elements, in order to simplify mesh reconciliation algorithms (Paszyński and Demkowicz 2006, Demkowicz and Kurtz and Pardo and Paszyński and Rachowicz and Zdunek 2007). However the introduction of the ghost elements increased the communication cost, especially after many refinements, since a layer of initial mesh elements, possibly broken into many smaller elements, must be exchange between adjacent sub-domains.

In this paper we propose an alternative parallelization technique, based on the shared domain decomposition paradigm, illustrated on the right panel in Fig. 1. The entire data structure with the computational mesh is stored on every processor. However, the computations performed over the mesh are shared between processors. It is done by assigning so-called processor owners to particular mesh elements, and executing computations over these elements by assigned processors. This is usually performed by sharing the algorithm’s loops by many processors, followed by mpi_allreduce call merging results. We will present how the self-adaptive hp-FEM algorithm can be parallelized in this way, and conclude the presentation by performance experiments.

Fig. 1. The shared domain decomposition as opposed to the distributed domain decomposition.

Some preliminary comparison between the distributed domain decomposition and shared domain decomposition paradigms for the discontinuous Galerkin method with non-automatic hp adaptivity was performed by Płażek and Banaś and Kitowski and Boryczko 1997, with promising scalability results. We extend this comparison to continuous Galerkin method, for the self-adaptive hp-FEM.

AUTOMATIC HP ADAPTIVITY

A general sequential algorithm for the fully automatic hp adaptation can be described is the following steps.
Algorithm starts with the coarse initial mesh with uniform order of approximation.

The computational problem is solved over the coarse mesh and the approximate solution \( u_{hp} \) is obtained.

The coarse mesh is globally \( hp \)-refined in order to produce the fine mesh. It is done by breaking each finite element into four son elements and increasing the polynomial order of approximation by one. This will be the reference mesh used for calculation of the interpolation error on the coarse mesh.

The computational problem is solved over the fine mesh and the approximate solution \( u_{hp+1} \) is obtained.

As the relative error estimator for the coarse mesh, the difference (in \( H^1 \)-seminorm) between the coarse and the fine mesh solutions is taken.

The optimal refinements are selected based on the calculated error estimators, for the subset of the coarse mesh elements with higher relative error estimators. Selected elements are either broken into smaller son elements (this is so called \( h \)-refinement) isotropically (4 sons) or anisotropically (2 sons in the same direction) or the polynomial order of approximation is increased on element edges or interiors (this is so called \( p \)-refinement), or both. This is illustrated in Fig. 2.

![Fig. 2. Many possible refinements of a coarse mesh element.](image)

The optimal refinements are selected independently over each coarse mesh element. It is done in a way to provide maximal error decrease rate given by:

\[
rate = \max \left\{ \frac{\| u_{hp+1} - u_{h, p} \|_1 - \| u_{hp+1} - w \|_1}{\Delta \text{ndof}} \right\}
\]

where \( \Delta \text{ndof} \) is the number of added degrees of freedom during the considered refinement, \( w \) is the solution for proposed refinement strategy, obtained by utilizing the projection based interpolation (Demkowicz 2004) from the fine mesh solution \( u_{hp+1} \) into the consider element refinement, \( \| u_{hp+1} - u_{h, p} \|_1 \) is the relative error estimation over the current coarse mesh with respect the fine mesh and
\( u_{y_{p+1}} - u \) is the relative error estimation for the refinement strategy proposed for the coarse mesh element with respect to the fine mesh. Thus we seek for such a refinement which gives the best error decrease rate by the possible low increase of the number of degrees of freedom.

(7) The selected refinements are executed over the coarse mesh to obtain the new optimal mesh.

(8) The new optimal mesh becomes a coarse mesh for the next iteration, and the entire procedure is repeated as long as the global relative error estimation is larger than the required accuracy of the solution.

The algorithm is also illustrated in Fig. 3. The selection of the optimal refinements for the coarse mesh finite elements is actually performed in two steps, in order to limit the number of possibilities considered in point 6). First, the optimal refinements are selected for finite element edges, and then the optimal refinements for elements interiors are selected, with restriction to known optimal refinements for element edges. The relative error measurements over element edges are performed in the \( H^{1/2} \) seminorm.

![Fig. 3. The algorithm for the sequential self-adaptive hp-FEM.](image)

THE DATA STRUCTURE SUPPORTING HP MESH REFINEMENTS

In this section we present a derived data structure supporting mesh refinements. The regular initial mesh is generated as consisting of many element objects, with each
element having four vertex objects, four edge node objects, and one interior node. These relations are presented in Fig. 4 by using the Unified Modeling Language (UML) diagrams (Booch and Rumbaugh and Jacobson 1998). The node object type varies between ‘medg’ for an element edge and ‘mdlq’ for an element interior.

However, the element objects are created only for the initial mesh, and mesh refinements are obtained by creating new node and vertex objects, and adding them as sons of broken edge and interior node objects. This is illustrated in Fig. 5. An element is broken in two steps. First an element interior node is broken, and then element edges are broken. An element interior can be broken into one new edge node object and two new element interior node objects (which corresponds to anistropic \( h \) refinement) or into one new vertex object, four new edge node objects, and four new element interior node objects (which corresponds to isotropic \( h \) refinement). An element edge node object can be broken into one new vertex and two new edge node objects. An element vertex is never broken. The actual active elements over the refined mesh are dynamically located by browsing the tree-like structure connecting initial mesh element nodes with new nodes and vertices, constructed as a result of mesh refinements.

An additional data structure is utilized to collect error estimations over the coarse mesh elements, to be able to perform optimal mesh refinements. This data structure is illustrated in Fig. 6.

In the new parallel version of the code presented in this paper, the entire data structure is stored on every processor, but initial mesh elements are assigned to different processor owners, as it is illustrated in Fig. 1. Thus, the element object has processor_owner attribute, compare Fig. 4. Vertices or edge nodes can be assigned to one or many processors since they are shared between adjacent elements, with possibly different processor owners. Thus, the vertex and element node objects have the processor_owners list.
In this section, we present the parallel version of the fully automatic $hp$ adaptivity, implemented under the shared domain decomposition paradigm. The new parallel algorithm can be summarized in the following steps:

1. The coarse initial mesh is generated on every processor. The initial mesh elements are assigned to different processors, by filling $\text{processor\_owner}$ attribute of the $\text{element}$ object. It is done either by interfacing with the ZOLTAN library (ZOLTAN), or by utilizing simple row-wise mesh partitioners for two dimensional meshes. The element’s $\text{processor\_owner}$ attribute is filled on every processor, in other words, each processor knows processor owners of all elements. The element edges and vertices are assigned to processor owners. It is done by browsing all elements and filling $\text{processor\_owners}$ lists located at $\text{element\_node}$ or $\text{vertex}$ objects.

2. The additional data structure presented in Fig. 6 is initialized. The $\text{element\_refined}$ objects are created for each active finite element. The $\text{middle\_node}$ links are related with interior nodes of active finite elements, represented by $\text{node}$ objects with $\text{type='mdlq'}$. The $\text{edge\_refined}$ objects are created for all active finite element edges. The $\text{edge\_node}$ links are related with element edge nodes, represented by $\text{node}$ objects with $\text{type='medg'}$. Notice, that $\text{element\_refined}$ objects (related to active finite elements) do not correspond to $\text{element}$ objects (related to the initial mesh elements only).

3. The computational problem is solved over the current coarse mesh, by utilizing MUMPS parallel direct solver (Amestoy and Duff and L'Excellent 2000, Amestoy and Duff and L'Excellent 2001, Amestoy and Guermouche and L'Excellent and Pralet 2005). Each processor stores the local solution vector at its active finite element $\text{node}$ and $\text{vertex}$ objects, in the $\text{solution\_d.o.f.}$ attribute. The coarse mesh solution d.o.f. are also recorded at $\text{coarse\_mesh\_solution}$ arrays of $\text{elements\_refined}$ objects.

4. The global $hp$ refinement is executed over the coarse mesh, in order to construct the reference fine mesh. This is performed by every processor over the entire data structure. Each finite element from the coarse mesh is partitioned into four new finite elements, and the polynomial order of approximation is uniformly raised by
This is done by executing isotropic $h$ refinement over each element interior node object, as well as refinement over each element edge node object. Also, the order_of_approximation attribute is increased for each active node (for each leaf node object).

(5) The processor owners of newly created node and vertex objects are filled, based on the information inherited from father node objects.

(6) The computational problem is solved again over the just created fine mesh, by utilizing MUMPS parallel direct solver. Each processor stores the local solution vector at its active finite element node and vertex objects, in the solution_d.o.f. attribute. Notice, that the coarse mesh solution is still stored at parent nodes, as well as at elements_refined objects.

(7) Each processor loops through its active elements, and computes the relative error estimation over the element

$$\frac{\| u_{hp} - u_{h,p+1} \|_{H^1}}{\| u_{h,p+1} \|_{H^1}}$$

with $u_{hp}$ being the coarse mesh solution restored from the element_refined objects, and $u_{h,p+1}$ being the fine mesh solution restored from the solution_d.o.f. attribute of active finite element node and vertex objects. The relative error is stored in the error attribute of the element_refined objects.

(8) The maximum element relative error is computed, and elements with the relative error estimation larger than 33% of the maximum error are denoted to be refined.

(9) For elements with strong gradient of the solution in one direction, the isotropy_flag attribute of the element_refined object is set to enforce the element refinement in one direction.

(10) Different refinement strategies are considered for element edges, by utilizing the formula

$$\frac{\| u_{h,p+1} - u_{hp} \|_{H^{1/2}}}{\| u_{h,p+1} - w \|_{H^{1/2}}}$$

with $u_{hp}$ the coarse mesh solution restored from the element_refined objects, $u_{h,p+1}$ the fine mesh solution restored from active finite element node and vertex objects, and $w$ being the projection based interpolant of the fine mesh solution $u_{h,p+1}$ into the considered edge refinement. The $H^{1/2}$ seminorm is utilized to measure the relative error over an element edge. The selected refinement is stored in edge_refined object. If an element edge is going to be $p$ refined, the ref_flag attribute for the edge is set to 1, and the proposed order of approximation is stored at order attribute. If an element edge is going to be $h$ refined, the ref_flag attribute for the edge is set to -1, and the
proposed orders of approximation for son edges are stored at orders attribute array. These estimations are performed by every processor over active finite elements assigned to the processor. Thus, the optimal refinement information is stored in distributed manner in element_refined and edge_refined objects. These estimations are performed only for edges of elements with relative error estimation larger than 33% of the maximum relative error.

(11) The proposed refinement data (ref_flag, order and orders attributes of edge_refined object as well as isotropy_flag of element_refined object) are broadcasted to all processors.

(12) The fine mesh is deallocated and the coarse mesh is restored.

(13) The selected optimal refinements are executed for element edges. This is done by all processors over the entire data structure. It can be done, since we broadcasted the proposed refinement data. Some edges are \( h \) refined: one new vertex object and two new edge node objects are created are connected to the original edge node. The order of approximation for new node objects is taken from orders attribute array of edge_refined object. Some edges are \( p \) refined, and the new order of approximation is taken from order attribute of edge_refined object. Some edge refinements are modified based on the isotropy_flag from element_refined objects.

(14) Different element interior node refinements are considered for elements. This is done by utilizing the formula

\[
\frac{\|u_{h,p} - u_{h,p+1}\|}{\Delta t} \leq \frac{\|u_{p} - W_{p}\|}{\Delta t}
\]

(4)

with \( u_{h,p} \) the coarse mesh solution restored from the element_refined objects, \( u_{h,p+1} \) the fine mesh solution restored from active finite element node and vertex objects, and \( w \) being the projection based interpolant of the fine mesh solution \( u_{h,p+1} \) into the considered element refinement. The \( H^1 \) seminorm is utilized to measure the relative error over an element interior. The number of considered element refinements is restricted be already executed edge refinements. These estimations are performed by every processor over active finite elements assigned to the processor. The selected refinement is stored in element_refined object. The type of refinement is coded within refinement_flag, and new orders of approximation for son nodes are coded within orders array. Thus, the optimal refinements information is stored in distributed manner in element_refined and edge_refined objects. These estimations are performed only for edges of elements with relative error estimation larger than 33% of the maximum relative error.

(15) The proposed interiors refinement data (refinement_flag and orders attributes of element_refined objects) are broadcasted to all processors.

(16) The selected optimal refinements are executed for element interiors. This is done by all processors over the entire data structure. It can be done, since we broadcasted the proposed refinement data. Some elements are \( h \) refined: new edge and interior node objects and vertex objects (for isotropic \( h \) refinement) are
created are connected to the original interior node. The order of approximation for new node objects is taken from orders attribute array of element_redefined object. Some elements are $p$ refined, and the new orders of approximation are taken from orders attribute of element_redefined object.

(17) The minimum rule is enforced over the entire data structure: the order of approximation over element edges is set to be equal to the minimum of orders for adjacent element interiors. This is done by all processors over the entire data structure. Thus, the identical copy of the new optimal mesh is stored on every processor.

(18) The element_redefined and edge_redefined objects are deallocated.

(19) If the maximum error is still greater than the required accuracy of the solution, the new optimal mesh becomes a coarse mesh and the next iteration is executed.

PROBLEM FORMULATION

In this section we present exemplary parallel simulations for the 3D DC resistivity logging measurement simulation problem. The problem consists in solving the conductive media equation

$$\nabla \cdot (\sigma \nabla u) = -\nabla \cdot J^{\text{imp}}$$

in the 3D domain with different formation layers, presented in Fig. 7. There is a tool with one transmitter and two receiver electrodes in the borehole. The tool is shifted along the borehole. The reflected waves are recorded by the receiver electrodes in order to determine location of the oil formation in the ground. Of particular interest to the oil industry are 3D simulations with deviated wells, where the angle between the borehole and formation layers is sharp $\theta_0 \neq 90^\circ$. This fully 3D problem can be reduced to 2D by considering the non-orthogonal system of coordinates presented in Fig. 7. Following (Pardo and Calo and Torres-Verdin and Nam 2007) the variational formulation in the new system of coordinates consists in finding $u \in u_\nu + H_D^1(\Omega)$ such that:

$$\left\langle \frac{\partial u}{\partial \xi}, \sigma \frac{\partial v}{\partial \xi} \right\rangle_{L^2(\Omega)} = \left\langle v, J \right\rangle_{L^2(\Omega)} \quad \forall v \in H_D^1(\Omega)$$

(6)

where new electrical conductivity of the media $\hat{\sigma} := J^{-1} \sigma J^{-1} \nabla$ and $\hat{J} := J J^{\text{imp}}$ with $f = \nabla J^{\text{imp}}$ is the gradient of the impressed current and

$$J = \frac{\partial(x_1, x_2, x_3)}{\partial(\xi_1, \xi_2, \xi_3)}$$

(7)
stands for the Jacobian matrix of the change of variables from the Cartesian reference to non-orthogonal systems of coordinates, and $|J| = \det(J)$ is its determinant. We take Fourier series expansions in the azimuthal $\zeta_2$ direction.

Fig. 7. The non-orthogonal system of coordinates in the borehole and formation layers.

$$u(\zeta_1, \zeta_2, \zeta_3) = \sum_{l=\pm \infty} u_l(\zeta_1, \zeta_3) e^{j l \zeta_2};$$  \hspace{1cm} (8)

$$\sigma(\zeta_1, \zeta_2, \zeta_3) = \sum_{m=\pm \infty} \sigma_m(\zeta_1, \zeta_3) e^{j m \zeta_2};$$  \hspace{1cm} (9)

$$f(\zeta_1, \zeta_2, \zeta_3) = \sum_{l=\pm \infty} f_l(\zeta_1, \zeta_3) e^{j l \zeta_2};$$  \hspace{1cm} (10)

where $u_l = \frac{1}{2\pi} \int_0^{2\pi} u e^{-j l \zeta_2} d\zeta_2$, $\sigma_m = \frac{1}{2\pi} \int_0^{2\pi} \sigma e^{-j m \zeta_2} d\zeta_2$ and $f_l = \frac{1}{2\pi} \int_0^{2\pi} f e^{-j l \zeta_2} d\zeta_2$ 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 $F_l(u) \in F_l(u_d) + H_D^1(\Omega)$ such that:

$$\left\langle F_l \left( \frac{\partial u}{\partial \zeta} \right), F_m \left( \frac{\partial \sigma}{\partial \zeta} \right) e^{j (l+m) \zeta_2} \right\rangle_{L^2(\Omega_{id})} = \left\langle \nu, F_l \left( \frac{\partial f}{\partial \zeta} \right) e^{j l \zeta_2} \right\rangle_{L^2(\Omega_{id})} \quad \forall \nu \in H_D^1(\Omega).$$  \hspace{1cm} (11)
The Einstein’s summation convention is applied with respect to \(-\infty \leq l, m \leq \infty\). We select a mono-modal test function \(v = \psi_k e^{\hat{\xi} l}\). Thanks to the orthogonality of the Fourier modes in \(L^2\) the variational problem defined in Eq. (11) reduces to

\[
\text{Find } F_j(u) \in F_j(u_p) + H^1_D(\Omega) \text{ such that:}
\]

\[
\sum_{\xi=0}^{\xi=5} \left( F_i \left( \frac{\partial u}{\partial \xi} \right), F_{k-1}(\hat{\xi} l) \right)_{L^2(\Omega_{2D})} = \left( F_k(v), F_j \right)_{L^2(\Omega_{2D})} \quad \forall F_k(v) \in H^1_D(\Omega_{2D})
\]

(12)

since five Fourier modes are enough to represent exactly the new material coefficients.

We refer to Pardo and Calo and Torres-Verdin and Nam 2007 for more details.

![Fig. 8. The optimal mesh generated by the code in the Cartesian 2D reference system of coordinates and the solution for a single position of receiver and transmitter electrodes.](image)

**Fig. 8.**

**NUMERICAL EXPERIMENTS**

We present measurements of the total execution time and memory usage for increasing number of processors, for the parallel self-adaptive \(hp\)-FEM based on the shared domain decomposition paradigm. The optimal mesh generated in a fully automatic mode and the exemplary solution are presented in Fig. 8. The total execution times and maximum memory usage for five consecutive iterations of the algorithm for increasing number of processors are presented in Table 9. The Fig. 10 presents total execution time–the sum of execution times from all five iterations – and maximum memory usage.

<table>
<thead>
<tr>
<th>Number of processors</th>
<th>CPU time</th>
<th>Memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Iteration</td>
<td>Time [s]</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>43</td>
</tr>
</tbody>
</table>
The coarse and fine mesh problem sizes are reported in Table 11. The initial coarse mesh is a rectangular mesh with 16*36=576 finite elements, with polynomial order of approximation $p=2$.

From presented measurements it follows that the code scale well up to 8 processors and than it lost its efficiency. To investigate this problem, we perform more detailed measurements presented in Fig. 12. In this figure we present total execution time (sum over all five iterations of the self-adaptive $hp$-FEM) for all components of the algorithm. This includes the coarse and fine mesh solutions, divided into the analysis, factorization and solution steps of the MUMPS solver (Amestoy and Duff and L'Excellent 2000, Amestoy and Duff and L'Excellent 2001, Amestoy and Guermouche and L'Excellent and Pralet 2005), as well as the execution time of the algorithm making decision about optimal mesh refinements, and performing these refinements over the entire mesh.

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>39</th>
<th>2</th>
<th>1032</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>37</td>
<td>3</td>
<td>1044</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>40</td>
<td>4</td>
<td>1109</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>45</td>
<td>5</td>
<td>1140</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>28</td>
<td>1</td>
<td>678</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24</td>
<td>2</td>
<td>635</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>26</td>
<td>3</td>
<td>662</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>27</td>
<td>4</td>
<td>708</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>28</td>
<td>5</td>
<td>695</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>25</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>22</td>
<td>2</td>
<td>519</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>21</td>
<td>3</td>
<td>422</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>23</td>
<td>4</td>
<td>444</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>25</td>
<td>5</td>
<td>471</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>32</td>
<td>1</td>
<td>365</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>2</td>
<td>349</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>23</td>
<td>3</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>25</td>
<td>4</td>
<td>325</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>26</td>
<td>5</td>
<td>361</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>38</td>
<td>1</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32</td>
<td>2</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>32</td>
<td>3</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>33</td>
<td>4</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>32</td>
<td>5</td>
<td>238</td>
</tr>
</tbody>
</table>

Table 9. Execution time and maximum memory usage for increasing number of processors.
Fig. 10. Total execution time and maximum memory usage after five iterations of the self-adaptive $hp$-FEM, for increasing number of processors.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse</td>
<td>26455</td>
<td>26730</td>
<td>26983</td>
<td>27489</td>
<td>27808</td>
</tr>
<tr>
<td>Fine</td>
<td>231407</td>
<td>233255</td>
<td>234685</td>
<td>239811</td>
<td>241747</td>
</tr>
</tbody>
</table>

Table 11. Coarse and fine mesh problems sizes for particular iterations.

We conclude that the most expensive is the fine mesh solution problem. It scales well up to 16 processors. All other parts of the MUMPS solver don’t scale at all. The algorithm making decision about optimal mesh refinements scale well up to 8 processors, then it lost its efficiency. This is related with the fact, that we actually perform refinements over the entire mesh, and broadcast decisions about optimal mesh refinements.

CONCLUSIONS

We can draw the following conclusions from presented measurements. The scalability of the proposed parallel algorithm based on shared domain decomposition is limited. It scale well up to 8 processors, and than it lost its efficiency. This results from 1) the limited scalability of the MUMPS solver, and 2) the limited scalability of the mesh refinements algorithm. What is surprising is that this lost of efficiency for the MUMPS solver actually results from constantly increasing execution time for the analysis phase of the fine mesh solver, as well as constantly growing coarse mesh solution time. The factorization of the fine mesh problem solution scales well up to 16 processors. In the future version of the parallel code we will utilize our own parallel solver that scale well
up to larger number of processors (Paszyński and Pardo and Torres-Verdin and Demkowicz and Calo 2007).

The parallel algorithm making decisions about optimal mesh refinements scales well up to 8 processors and lost the efficiency for larger number of processors. It results from the fact that decisions about optimal refinements performed locally are exchanged between all processors, as well as the mesh refinements are the sequential algorithm, since they are executed over the entire mesh.

We conclude that the *share domain decomposition* parallel algorithm can be effectively utilized for limited number of processors.

However, the memory usage for the parallel code scales very well, and we are able to reduce the single processors memory usage down to 267 MB per processor. The single processor version of the code required more than 2 GB of memory, and we very often run out of memory in our simulations.

![Graph showing comparison of solver scalability versus parallel algorithm scalability.](image)

Fig. 12. Comparison of the scalability of the solver versus the scalability of the parallel algorithm making decisions about optimal mesh refinements.

ACKNOWLEDGEMENTS

The first author has been supported by the Foundation for Polish Science under Homming Programme and by the Polish Ministry of Scientific Research and Information Technology grant no 3 T08B 055 29. The second and third authors are supported by The University of Texas at Austin’s *Joint Industry Research Consortium on Formation Evaluation* sponsored by Aramco, Baker Atlas, BP, British Gas, ConocoPhilips, Chevron, ENI E&P, ExxonMobil, Halliburton Energy Services, Hydro, Marathon Oil Corporation, Mexican Institute for Petroleum, Occidental Petroleum Corporation, Petrobras, Schlumberger, Shell International E&P, Statoil, TOTAL, and Weatherford.

REFERENCES


KEYWORDS
Finite element method, automatic hp adaptivity, parallel computations

LIST OF CAPTIONS FOR ILLUSTRATIONS

Fig. 1. The shared domain decomposition as opposed to the distributed domain decomposition.

Fig. 2. Many possible refinements of a coarse mesh element.

Fig. 3. The algorithm for the sequential self-adaptive $hp$-FEM.

Fig. 4. The UML diagram presenting initial mesh elements structure.

Fig. 5. The UML diagram presenting the father-son relations utilized for mesh refinements.

Fig. 6. The UML diagram presenting the data structure managing mesh refinements.

Fig. 7. The non-orthogonal system of coordinates in the borehole and formation layers.

Fig. 8. The optimal mesh generated by the code in the Cartesian 2D reference system of coordinates and the solution for a single position of receiver and transmitter electrodes.

Table 9. Execution time and maximum memory usage for increasing number of processors.

Fig. 10. Total execution time and maximum memory usage after five iterations of the self-adaptive $hp$-FEM, for increasing number of processors.

Table 11. Coarse and fine mesh problems sizes for particular iterations.

Fig. 12. Comparison of the scalability of the solver versus the scalability of the parallel algorithm making decisions about optimal mesh refinements.