Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE
Informatyka w Technologii Materiałów

Vol. 11, 2011, No. 2

PARALLEL SELF-ADAPTIVE *hp* FINITE ELEMENT METHOD WITH SHARED DATA STRUCTURE

MACIEJ PASZYNSKI[1*], DAVID PARDO[2]

[1] *Department of Computer Science, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Krakow, Poland*
[2] *Departamento de Matemática Aplicada, Estadística e Investigación Operativa, UPV/EHU, Campus de Leioa, Vizcaya, and IKERBASQUE (Basque Foundation for Sciences), Bilbao, Spain*
**Corresponding author: maciej.paszynski@agh.edu.pl*

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), *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 present details of the parallel self-adaptive hp-FEM algorithm with shared computational domain, as well as its efficiency measurements. The presentation is enriched by numerical results of the 3D DC borehole resistivity measurement simulations.

Key words: parallel computing, *hp*-FEM, borehole resistivity measurement simulations

1. INTRODUCTION

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 et. al. 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 figure 1 (Paszyński et. al. 2006; Paszyński & Demkowicz 2006). The main disadvantage of the distributed domain decomposition based parallel code was the huge complexity of the mesh transformation algorithms executed over the computational mesh stored in distributed manner. In particular, there exists the following mesh regularity rules: (1) *the one irregularity rule*, imposing that a finite element cannot be broken for the second time without first breaking larger adjacent elements, and (2) *the minimum rule,* which guarantees 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 (Demkowicz et. al. 2007; Paszyński & Demkowicz 2006). 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 figure 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.

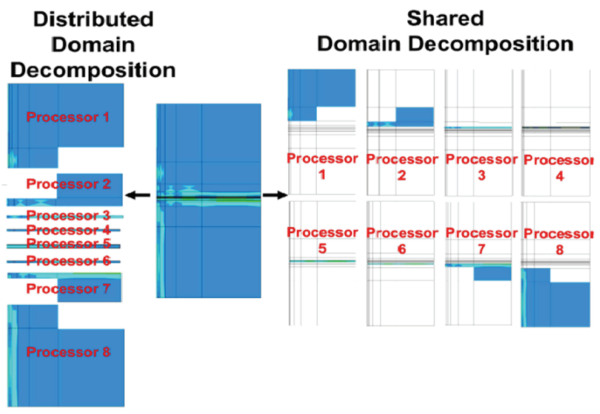

Fig. 1. *The shared domain decomposition as opposed to the distributed domain.*

2. THE DATA STRUCTURE SUPPORTING MESH REFINEMENTS

In this section we present a 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 by using the Unified Modeling Language (UML) diagrams (Booch et. al. 1994). The `node` object type varies between `'medg'` for an element edge and `'mdlq'` for and 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 figure 2. An element is broken in two steps.

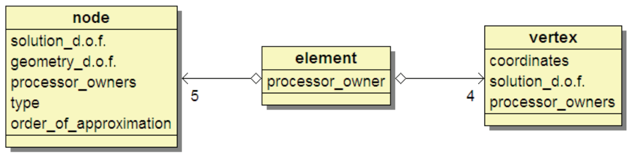

Fig. 2. *The UML diagram presenting initial mesh elements structure.*

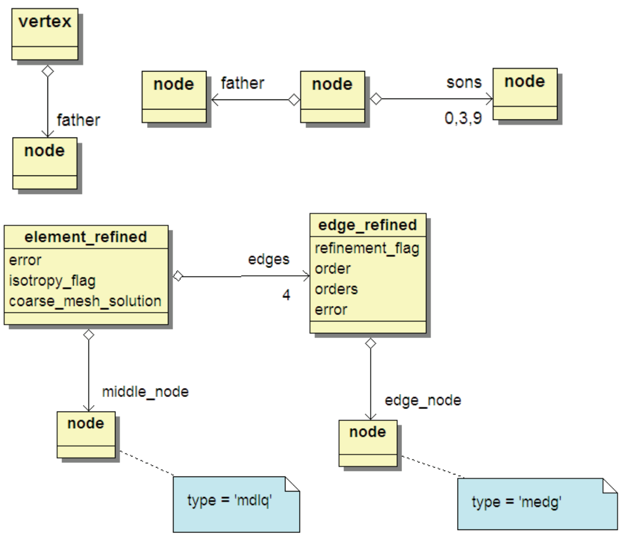

Fig. 3. *The UML diagram presenting the father-son relations utilized for mesh refinements as well as the data structure managing mesh refinements.*

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. 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 figure 3. In the new parallel version of the code the entire data structure is stored on every processor, but initial mesh elements are assigned to different processor owners, as it is illustrated in figure 1. Thus, the element object has `processor_owner` attribute, compare figure 2. 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.

3. PARALLEL AUTOMATIC HP ADAPTIVITY

In this section, we present the shared memory parallel version of the self-adaptive *hp*-FEM

1. The coarse initial mesh is generated on every processor. The elements are assigned to different processors, by filling `processor_owner` attribute of the `element` object. It is done either by interfacing with the ZOLTAN library (ZOLTAN). The element's `processor_owner` attribute is filled on every processor.
2. The additional data structure presented in figure 3 is initialized. The `element_refined` objects are created for each active finite element. The `middle_node` links are related with interior nodes of active finite elements, represented by `node` objects with `type='mdlq'`. The `edge_refined` objects are created for all active finite element edges. The `edge_node` links are related with element edge nodes, represented by `node` objects with `type='medg'`.
3. The computational problem is solved over the current coarse mesh, by utilizing MUMPS parallel direct solver (Amestoy et. al. 2000; Amestoy et. al. 2001). Each processor stores the local solution vector at its active finite element `node` and `vertex` objects, in the `solution_d.o.f.` attribute. The coarse mesh solution d.o.f. are also recorded at `coarse_mesh_solution` arrays of `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. This is done by executing isotropic *h* refinement (breaking an element into four) 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, to expressed to uniform *p* refinement.
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
$$\left\| u_{hp} - u_{h/2,p+1} \right\|_{H^1} / \left\| u_{h/2,p+1} \right\|_{H^1}, \tag{1}$$
with u_{hp} the coarse mesh solution restored from the `element_refined` objects, and $u_{h/2,p+1}$ 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
$$\frac{\left\| u_{h/2,p+1} - u_{h,p} \right\|_{1/2} - \left\| u_{h/2,p+1} - w \right\|_{1/2}}{\Delta nrdof} \tag{2}$$
with u_{hp} the coarse mesh solution restored from the `element_refined` objects, $u_{h/2,p+1}$ 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 $u_{h/2,p+1}$ into the considered edge refinement. The $H^{1/2}$ seminorm is utilized over an

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 with relative error larger than 33% of the maximum relative error. The optimal refinement is stored in distributed manner in `element_refined` and `edge_refined` objects.

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 by using

$$\frac{\left\| u_{h/2,p+1} - u_{h,p} \right\|_1 - \left\| u_{h/2,p+1} - w \right\|_1}{\Delta nrdof} \tag{3}$$

with u_{hp} the coarse mesh solution restored from the `element_refined` objects, $u_{h/2,p+1}$ 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 $u_{h/2,p+1}$ into the considered element refinement. The H^1 seminorm is utilized over an element. The number of considered element refinements is restricted be already executed edge refinements. These estimations are performed by every processor over active finite elements with relative error larger than 33% of the maximum relative error. 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. The optimal refinements information is stored in distributed manner in `element_refined` and `edge_refined` objects.
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_refined` object. Some elements are *p* refined, and the new orders of approximation are taken from `orders` attribute of `element_refined` 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.
18. The `element_refined` and `edge_refined` 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.

4. NUMERICAL EXPERIMENTS

In this section we present parallel simulations for the 3D DC resistivity logging measurement simulation problem. The problem is formulated by utilizing 2D mesh, with non-ortogonal system of coordinates and the Fourier series expansion in the aximuthal direction. Thus, we are going to solve full 3D problem but on the 2D mesh. The problem consists in solving the conductive media equation

$$\nabla \circ (\sigma \nabla u) = -\nabla \circ J^{imp} \tag{4}$$

in the 3D domain with different formation layers, presented in figure 4. 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 $\theta_0 \neq 90$. This fully 3D problem can be reduced to 2D by considering the non-orthogonal system of coordinates presented in figure 4. Following (Pardo et. al. 2009) the variational formulation in the new system of coordinates consists in finding $u \in u_D + H_D^1(\Omega)$ such that:

$$\left\langle \frac{\partial u}{\partial \xi}, \hat{\sigma} \frac{\partial v}{\partial \xi} \right\rangle_{L^2(\Omega)} = \left\langle v, \hat{f} \right\rangle_{L^2(\Omega)} \quad \forall v \in H_D^1(\Omega) \tag{5}$$

where new electrical conductivity of the media $\hat{\sigma} := J^{-1} \sigma J^{-1^T} |J|$ and $\hat{f} := f|J|$ with $f = \nabla J^{imp}$ is the gradient of the impressed current and J 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 ζ_2 direction

$$u(\zeta_1,\zeta_2,\zeta_3) = \sum_{l=-\infty}^{l=+\infty} u_l(\zeta_1,\zeta_3) e^{jl\zeta_2};$$
$$\sigma(\zeta_1,\zeta_2,\zeta_3) = \sum_{m=-\infty}^{m=+\infty} \sigma_m(\zeta_1,\zeta_3) e^{jm\zeta_2}; \tag{8}$$
$$f(\zeta_1,\zeta_2,\zeta_3) = \sum_{l=-\infty}^{l=+\infty} f_l(\zeta_1,\zeta_3) e^{jl\zeta_2};$$

where $u_l = \frac{1}{2\Pi} \int_0^{2\Pi} u e^{-jl\zeta_2} d\zeta_2$, $\sigma_m = \frac{1}{2\Pi} \int_0^{2\Pi} \sigma e^{-jm\zeta_2} d\zeta_2$ and $f_l = \frac{1}{2\Pi} \int_0^{2\Pi} f e^{-jl\zeta_2} d\zeta_2$ 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 $F_l(u) \in F_l(u_D) + H_D^1(\Omega)$ such that:

$$\left\langle F_l\left(\frac{\partial u}{\partial \xi}\right), F_m(\hat{\sigma}) \frac{\partial v}{\partial \xi} e^{j(l+m)\zeta_2} \right\rangle_{L^2(\Omega_{2D})} =$$
$$\left\langle v, F_l(\hat{f}) e^{jl\zeta_2} \right\rangle_{L^2(\Omega_{2D})} \quad \forall v \in H_D^1(\Omega) \tag{6}$$

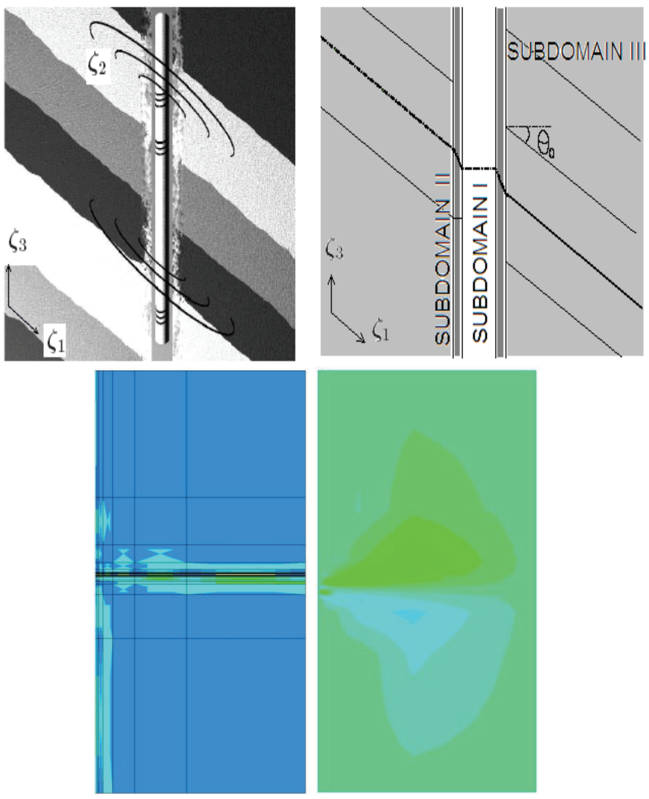

Fig. 4. Left panel: *The non-orthogonal system of coordinates in the borehole and formation layers. Different colors denote different layers in the formation, corresponding to e.g. rock, sand, oil-based mud, oil etc.* ***Down panel:*** *Three subdomains where the non-orthogonal system of coordinates is defined.* ***Right panel:*** *The optimal mesh and the solution for a single position of receiver and transmitter electrodes. Different colors on the optimal mesh refers to different polynomial orders of approximation used. The different colors on the solution refers to different values of the scalar electric potential.*

The Einstein's summation convention is applied with respect to $-\infty \le l, m \le \infty$. We select a monomodal test function $v = v_k e^{jk\zeta_2}$. Thanks to the orthogonality of the Fourier modes in L^2 the variational problem (6) reduces to: Find $F_l(u) \in F_l(u_D) + H_D^1(\Omega)$ such that:

$$\sum_{n=k-2}^{n=k+2}\left\langle F_l\left(\frac{\partial u}{\partial \xi}\right), F_{k-l}(\hat{\sigma})F_l\left(\frac{\partial v}{\partial \xi}\right)\right\rangle_{L^2(\Omega_{2D})} =$$

$$\left\langle F_k(v), F_k\left(\hat{f}\right)\right\rangle_{L^2(\Omega_{2D})} \qquad \forall F_k(v) \in H_D^1(\Omega_{2D}) \qquad (7)$$

since five Fourier modes are enough to represent exactly the new material coefficients. We refer to [9] for more details. The optimal mesh generated in a fully automatic mode and the exemplary solution are presented on right panel in figure 4.

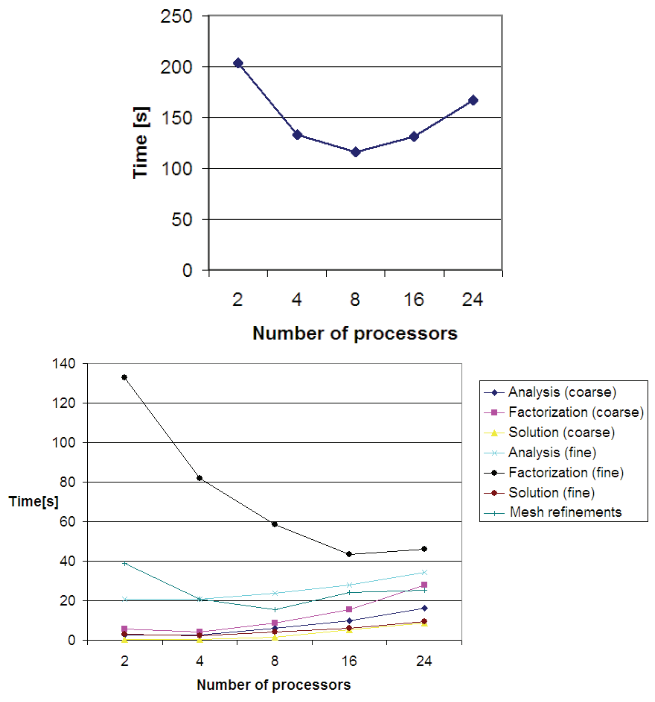

***Fig. 5**. Total execution time after five iteration of the self-adaptive hp-FEM, for increasing number of processors.*

***Table 1.** Coarse and fine mesh problems sizes for particular iterations*

Iteration	1	2	3	4	5
Coarse	26455	26730	26983	27489	27808
Fine	231407	233255	234685	239811	241747

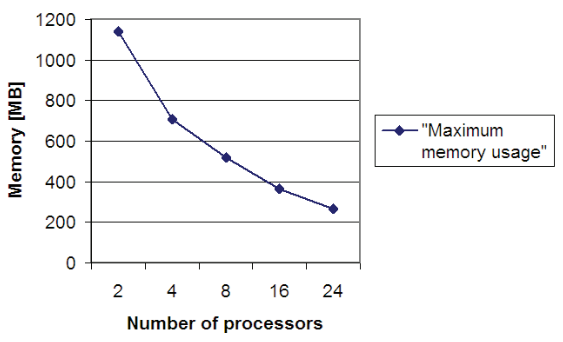

***Fig. 6**. Maximum memory usage after five iteration of the self-adaptive hp-FEM, for increasing number of processors.*

We conclude the section by presenting measurements of the total execution time for increasing number of processors, for the parallel self-adaptive *hp*-FEM based on the shared domain decomposition paradigm. Figure 5 presents total execution time (the sum of execution times from all five iterations), and 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 et. al. 2000; Amestoy et. al. 2001), as well as the execution time of the algorithm making decision about optimal mesh refinements, and performing these refinements over the entire mesh. The coarse and fine mesh problem sizes are reported in table 1. The initial coarse mesh is a rectangular mesh with 16·36 = 576 finite elements, with polynomial order of approximation p = 2. Finally, figure 6 presents the maximum memory usage for increasing number of processors. This concerns the fine grid solver solution, which takes one order of magnitude more memory than other operations.

5. CONCLUSIONS

From the presented results it follows that the parallel code scales well up to 8 processors, and then it lost its efficiency. This results from (1) the limited scalability of the MUMPS solver (2) the limited scalability of the mesh refinements algorithm. 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, compare Figure 5. This implies that actually the size of the computational problem is too small, and when we increase the number of processors, the communication time is dominating the computation time. In other words, when we increase the number of processors, the computation time goes down, while the communication time goes up, since there are more processors to communicate with. The parallel algorithm making decisions about optimal mesh refinements scales well up to 8 processors and then it execution time stabilizes. This is again because the fact that the communication time becomes to dominate the computation time. In the future we plan to perform larger parallel experiments with larger computational problems. The limited scalability of the MUMPS solver will be also overcome by utilizing our own parallel solver that scale well up to larger number of processors (Paszyński et. al. 2010). Finally, 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 required more than 2 GB of memory, and often runs out of memory.

ACKNOWLEDGEMENTS

The support of Polish MNiSW grant no NN 519 447739 is gratefully acknowledged.

REFERENCES

Demkowicz, L., 2006, *Computing with hp-Adaptive Finite Elements, Vol. I. One and Two Dimensional Elliptic and Maxwell Problems*, Chapmann & Hall / CRC Press.

Demkowicz, L., Rachowicz, W., Pardo, D., Paszyński, M., Kurtz, J., Zdunek, A., 2007, *Computing with hp-Finite Elements. Volume II*, Chapmann & Hall / CRC Press.

Paszyński, M., Kurtz, J., Demkowicz, L., 2006, Parallel Fully Automatic hp-Adaptive 2D Finite Element Package, *Computer Methods in Applied Mechanics and Engineering*, 195, 7-8, 711-741.

Paszyński, M., Demkowicz, L., 2006, Parallel Fully Automatic hp-Adaptive 3D Finite Element Package, *Computers and Mathematics with Applications*, 22, 3-4, 255-276.

Booch, G., Rumbaugh, J., Jacobson, I., 1994, The Unified Modeling Language User Guide, Addison-Wesley Professional, 1st edition.

ZOLTAN: *Data-Management Services for Parallel Applications*, http://www.cs.sandia.gov/Zoltan

Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., 2000, Multifrontal parallel distributed symmetric and unsymmetric solvers, *Computer Methods in Applied Mechanics and Engineering*, 184, 501-520.

Amestoy, P. R., Duff, I. S., Koster, J., L'Excellent, J.-Y., 2001, A fully asynchronous multifrontal solver using distributed dynamic scheduling, *SIAM Journal of Matrix Analysis and Applications*, 23, 1, 15-41.

Pardo, D., Calo, V., Torres-Verdin, C., Nam, M.J., 2007, Fourier Series Expansion in a Non-Orthogonal System of Coordinates for Simulation of 3D Borehole Resistivity Measurements. Part I: DC, *Computer Methods in Applied Mechanics and Engineering*, 197, 1-3, 1906-1925.

Paszyński, M., Pardo, D., Torres-Verdin, C., Demkowicz, L., Calo, V., 2010, A Parallel Direct Solver for the Self-Adaptive hp Finite Element Method, *Journal of Parallel and Distributed Computing*, 70, 270-281.

RÓWNOLEGŁY ALGORYTM HP ADAPTACYJNEJ METODY ELEMENTÓW SKOŃOCZNYCH O WSPÓŁDZIELONEJ STRUKTURZE DANYCH

Streszczenie

Artykuł ten przedstawia nowy algorytm równoległy dla *hp* adaptacyjnej metody elementów skończonych (*hp*-MES) cechujący się rozproszoną strukturą danych. Algorytm ten generuje w sposób w pełni automatyczny (bez żadnej interakcji użytkownika) ciąg siatek obliczeniowych dostarczających eksponencjalnej zbieżności zadanej funkcji celu względem rozmiaru siatki obliczeniowej (ilości stopni swobody). Algorytm generuje ciąg siatek obliczeniowych począwszy od zadanej siatki początkowej. Kolejne siatki otrzymywane są na drodze *h* adaptacji (łamania wybranych elementów) lub *p* adaptacji (zwiększania stopnia aproksymacji wielomianowej) lub *hp* adaptacji (jednocześnie *h* i *p* adaptacji) na wybranych elementach. Algorytm ten pracuje w oparciu o siatkę obliczeniową dzieloną pomiędzy wieloma procesorami. Wszystkie algorytmy obliczeniowe, włączając w to automatyczną *hp* adaptację oraz algorytm solvera, pracują w pełni równolegle. W artykule tym omawiamy algorytm równoległy oraz analizujemy jego efektywność. Prezentacja wzbogacona jest o wyniki numeryczne dotyczące trójwymiarowych symulacji problemu pomiaru oporowości warstw górotworu dla zadań prądu stałego.

Received: August 8, 2010
Received in a revised form: September 9, 2010
Accepted: November 9, 2010