Computer Science • 17 (4) 2016 http://dx.doi.org/10.7494/csci.2016.17.4.439

Paweł Lipski
Maciej Paszyński

ONE-DIMENSIONAL FULLY AUTOMATIC H-ADAPTIVE ISOGEOMETRIC FINITE ELEMENT METHOD PACKAGE

Abstract *This paper deals with an adaptive finite element method originally developed by Prof. Leszek Demkowicz for hierarchical basis functions. In this paper, we investigate the extension of the adaptive algorithm for isogeometric analysis performed with B-spline basis functions. We restrict ourselves to h-adaptivity, since the polynomial order of approximation must be fixed in the isogeometric case. The classical variant of the adaptive FEM algorithm, as delivered by the group of Prof. Demkowicz, is based on a two-grid paradigm, with coarse and fine grids (the latter utilized as a reference solution). The problem is solved independently over a coarse mesh and a fine mesh. The fine-mesh solution is then utilized as a reference to estimate the relative error of the coarse-mesh solution and to decide which elements to refine. Prof. Demkowicz uses hierarchical basis functions, which (though locally providing C^{p-1} continuity) ensure only C^0 on the interfaces between elements. The CUDA C library described in this paper switches the basis to B-spline functions and proposes a one-dimensional isogeometric version of the h-adaptive FEM algorithm to achieve global C^{p-1} continuity of the solution.*

Keywords finite element method, isogeometric analysis, parallel computing, h-adaptivity, B-splines

Citation Computer Science 17 (4) 2016: 439–460

1. Introduction

The main motivation of this paper is to introduce a software package developed for the adaptive isogeometric finite element method with the NVIDIA CUDA framework.

We show how to extend the idea of automatic adaptivity (originally developed by the group of Prof. Leszek Demkowicz) for hierarchical basis functions [8] into the case of an isogeometric analysis with B-spline basis functions [7]. When applied to FEM, hierarchical basis functions deliver local C^{p-1} continuity over the elements yet only C^0 on the interfaces between the elements. Switching to B-splines means gaining global C^{p-1} continuity of the solution [4, 7].

The original hp-adaptivity with a hierarchical base delivers exponential convergence rates [1, 2]. However, when we use B-spline basis functions specific to isogeometric analysis, the polynomial order of approximation is fixed, and we must restrict ourselves to h-adaptivity only.

Efficient h-refinement strategies are essential in solving different engineering problems [10, 11, 15]. There are several different versions of h-adaptive algorithms designed for this purpose [3, 5, 6, 9, 16]. In this work, we extend the existing results, delving into the parallel efficiency of the algorithm stages, including both FEM computations and the h-adaptation itself. We point out the opportunities for further speedup and analyze the dependency of the resultant error on computing time and B-spline order. In particular, we assess the efficiency of parallel shared-memory algorithms for computing stiffness matrices and load vectors and compare them to their CPU implementations. We include certain practical observations from the implementation that helped us leverage the GPU potential for the isogeometric analysis.

The process of solving the equation system is not discussed in this paper, since appropriate solutions are already well-described in [13]. There are also some alternative GPGPU-based solvers available [12, 14, 19]; however, they are general-purpose solvers and are not specific to the isogeometric finite element method (like the one presented in [12]). They deliver logarithmic computational cost with respect to the problem size, as presented in [13, 18].

2. Isogeometric finite element method

Let us briefly go through the basics of ISO-FEM, focusing on a simple one-dimensional elliptic model problem with Dirichlet boundary conditions.

$$-\frac{d}{dx}\left(\frac{du\,(x)}{dx}\right) = f(x) \tag{1}$$

$$u\,(0) = \alpha \tag{2}$$

$$u\,(1) = \beta \tag{3}$$

The elliptic ODE (1)–(3) must be restated into an equivalent weak formulation (4)–(9) before proceeding to an actual one-dimensional hierarchical-based FEM [8]:

$$\begin{aligned} \text{Find} \quad & w = u + \hat{u}, w(x) \in H^1((0,1)) && (4)\\ \text{where} \quad & u(x) \in H_0^1((0,1)) && (5)\\ \text{and} \quad & \hat{u}(x) = (1-x)\alpha + x\beta && (6)\\ \text{such that} \quad & b(v,w) = l(v), \forall v \in H_0^1((0,1)) && (7)\\ \text{where} \quad & b(v,w) = \int_0^1 \frac{dv}{dx}\frac{du}{dx}dx && (8) \end{aligned}$$

$$l(v) = \int_0^1 f(x)v(x)dx - \int_0^1 \frac{dv}{dx}\frac{d\hat{u}}{dx}dx = \int_0^1 f(x)v(x)dx - \int_0^1 \frac{dv}{dx}(\beta-\alpha)dx \tag{9}$$

Assuming e_i are quadratic B-spline base functions, the global system of equations takes the following form:

$$B = \begin{bmatrix} b(e_0,e_0) & b(e_0,e_1) & 0 & \dots & & & \\ b(e_1,e_0) & b(e_1,e_1) & b(e_1,e_2) & 0 & \dots & & \\ 0 & b(e_2,e_1) & b(e_2,e_2) & b(e_2,e_3) & 0 & \dots & \\ \vdots & & & \ddots & & & \\ 0 & \dots & 0 & b(e_i,e_{i-1}) & b(e_i,e_i) & b(e_i,e_{i+1}) & \dots \\ \vdots & & & & & \ddots & \\ 0 & & \dots & & 0 & b(e_n,e_{n-1}) & b(e_n,e_n) \end{bmatrix} \tag{10}$$

$$B \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_i \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} 0 \\ l(e_1) \\ l(e_2) \\ \vdots \\ l(e_i) \\ \vdots \\ 0 \end{bmatrix} \tag{11}$$

Isogeometric FEM employs B-splines to approximate the solution of (1)–(3):

$$u(x) \approx \sum_i N_{i,p}(x)\, a_i \qquad v \in \{N_{j,p}\}_j \tag{12}$$

$N_{i,k}$ is a B-spline of the order k, with the index (offset on the mesh) being i. For a given i and k, the value of $N_{i,k}$ is defined as:

$$N_{i,0}(x) = I_{[\xi_i,\xi_{i+1}]} \tag{13}$$

$$N_{i,k}(x) = \frac{x-\xi_i}{\xi_{i+k}-\xi_i}N_{i,k-1}(x) + \frac{\xi_{i+k+1}-x}{\xi_{i+k+1}-\xi_{i+1}}N_{i+1,k-1}(x) \tag{14}$$

where ξ_i are control points defined as

$$\xi_i = \frac{i}{N}, i = 0, ..., N \tag{15}$$

and $I_{[\xi_i,\xi_{i+1}]}$ is the identity function defined as 1 over the interval $[\xi_i, \xi_{i+1}]$ and 0 everywhere else.

As these definitions are substituted into the weak form, we obtain a discrete weak formulation; stated as:

$$\sum_i b\left(N_{j,p}\left(x\right), N_{i,p}\left(x\right)\right) a_i = l\left(N_{j,p}\left(x\right)\right), \forall j \tag{16}$$

We utilize B-splines to solve (1)–(3) over the computational mesh delivered by the h-adaptation process. The knot vectors are always open (i.e., with $p + 1$ equal knot values at each end of the mesh for B-splines of the polynomial order p). Please note that, when substituting into the Cox-de Boor formula (13-14), we do not use the knots (which usually go from 0 to N) but the control points instead (which go over the interval; i.e., from 0 to 1 in our case, as defined in [15]).

3. Algorithms for h-adaptive ISO-FEM

This section describes two strategies for h-adaptive ISO-FEM – two-grid and residue-based.

In h-adaptation, the mesh is locally refined to obtain higher accuracy on singularities. Various algorithms help to pick the refinements that are expected to increase the overall accuracy of the solution.

On the other hand, p-adaptation involves local increases of the polynomial order. This approach is not easily feasible for B-splines, since a non-uniform base would break the partition-of-unity rule [17]. We decided to stick to one polynomial order for the whole of the base and use it consistently across the mesh.

In the two-grid approach, an approximate solution is computed on both coarse and fine grids (the latter being actually a double-densified coarse grid; i.e., each element split into halves). The results are then compared; and based on the relative difference between the coarse-grid and fine-grid solutions, the grid is appropriately locally refined (as explained below).

3.1. Two-grid approach

The algorithm 1 outlines a single iteration of the two-grid adaptation algorithm. The iterations are performed until certain criteria are met. The typical stop condition is that we iterate until the value of an overall error (which can be residual error, for example, as defined in the section 4, integrated over the entire mesh) becomes lower than some arbitrarily chosen threshold. $\widetilde{u}_i^{coarse}$ and $\widetilde{u}_i^{fine}$ stand for the values of the approximate solutions (computed on the coarse and fine grids, respectively) in the

Algorithm 1 Single iteration of two-grid h-adaptation algorithm

Solve the problem over the coarse mesh
Break each coarse mesh element into two son elements (i.e., double-densify the mesh)
Solve the problem over the fine mesh
for each element E_i **do**
  ▷ compute the relative error r_i
  $r_i \leftarrow \left| \frac{\widetilde{u}_i^{fine} - \widetilde{u}_i^{coarse}}{\widetilde{u}_i^{fine}} \right|$
end for
$r_{max} \leftarrow$ find the maximal r_i
for each element E_i **do**
  if $r_{max} \cdot \tau < r_i$ **then**
    locally refine the grid at E_i (i.e., permanently break E_i into two son elements)
  end if
end for

geometric center of i-th element E_i. τ, in turn, is an arbitrarily chosen threshold constant between 0 and 1.

As a side note, the experiments show that the adaptation threshold (τ) for both the two-grid and residue-based approach should be kept within $[0.1, 0.2]$ rather than fixed at the 0.33 point (which is the threshold value generally recommended for h- and p-adaptation with hierarchical polynomials).

Let us take the model elliptic equation (17) as a sample to illustrate the progress of the algorithm — how the mesh is gradually refined and how the approximate solutions go toward the exact analytical solution with each single iteration.

$$\text{Let} f \text{be defined as} f(x) = -\sin \frac{a}{1 + e^{-k(x-\mu)}} \tag{17}$$

$$a = 10\pi, k = 10, \mu = 0.5 \tag{18}$$

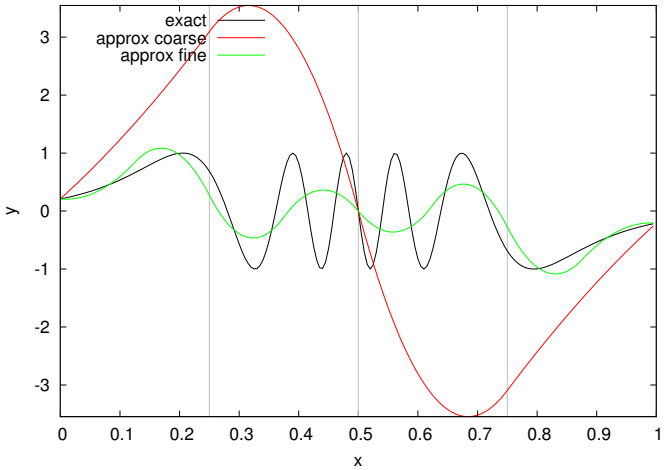
$$-u'' = f''(x) \tag{19}$$

$$u(0) = f(0) \tag{20}$$

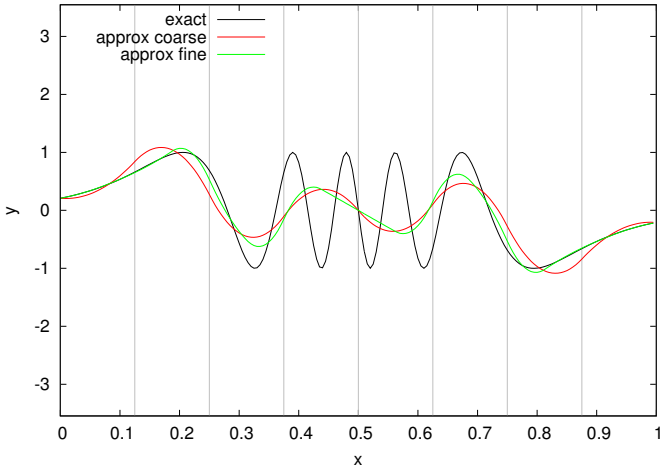
$$u(1) = f(1) \tag{21}$$

We actually let the solver look for the already-known function $f(x)$, the solver being provided only with its second derivative $f''(x)$ as the right-hand side of the equation. The benefit of such an approach is that we have a clear view on how the exact analytical solution should look, thus enabling us to easily assess the error.

The sequence of Figures 1a through 1e shows iterations of the algorithm applied to solve the equation. The solution is approximated with quadratic B-splines. From this point in the paper, the element borders of the coarse grid are marked with light gray vertical lines in the background.

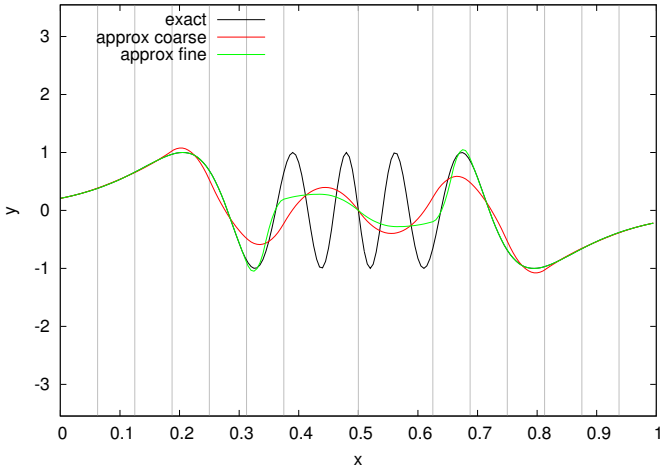


(a) Iteration 1. The solution is computed both on a coarse grid (consisting of 4 elements) and a fine grid (8 elements, since each element of the coarse grid gets halved to obtain the fine grid). The red curve indicates the solution computed on the coarse grid, and the green curve – on the fine grid. The black curve shows the exact analytic solution. Since the error for every single element exceeds the maximal error times threshold τ (here, set to 20%), the algorithm decides to split all four elements, which leads to the mesh shown in Figure 1b.

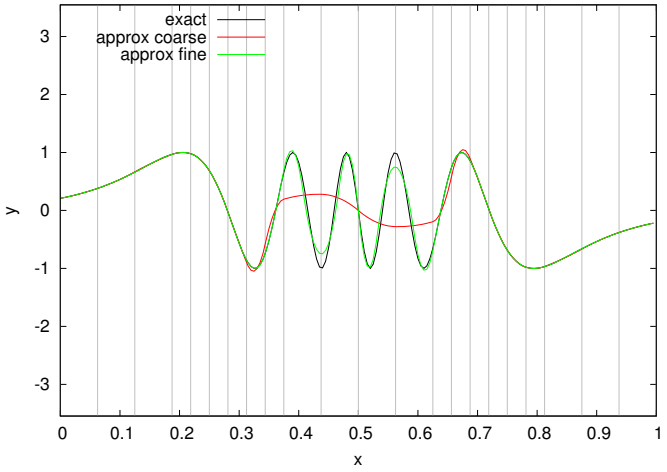


(b) Iteration 2. On the central elements (i.e., E_4 and E_5), the approximate solution computed on the coarse mesh and that on the fine mesh lie very close to each other, thus generating a low relative error (marked as r_i in algorithm 1). At the same time, this error is larger for the other (non-central) elements — E_1, E_2, E_3, E_6, E_7 and E_8. Obviously, the difference between the approximate and exact solutions is much higher at the central elements. The algorithm, however, has no knowledge of the expected exact solution, and judging by r_i values, picks the non-central elements for the split, effecting in the mesh as shown in Figure 1c.

Figure 1. Exact and approximate solutions on the coarse grid in iterations 1 through 5 of the two-grid h-adaptation algorithm. Quadratic B-splines, equation (17).

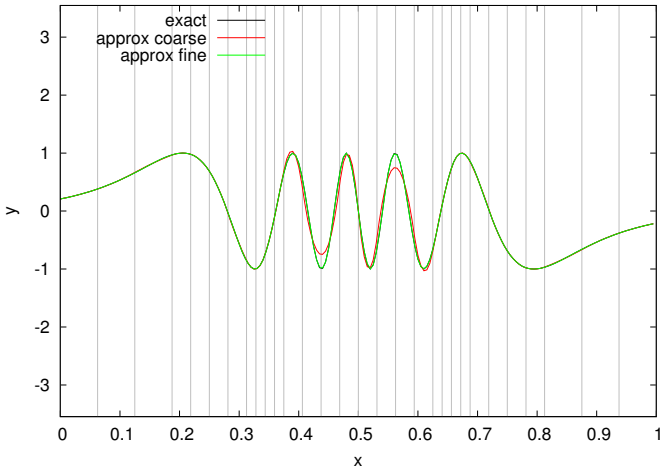


(c) Iteration 3



(d) Iteration 4.Only now, the central elements are split; but still, even after refinement, the distance to the exact solution in the central elements is significantly larger than in the non-central ones.

Figure 1



(e) Iteration 5. The showcase is finished at this moment, since the curves overlap too closely starting with iteration 6, and the plots would lose readability.

Figure 1

3.2. Residue-based approach

This section focuses on another approach to h-adaptation; namely, the residue-based approach. For an elliptic equation (boundary conditions are of no significance and, therefore, are skipped):

$$au''(x) + bu'(x) + cu(x) = f(x) \tag{22}$$

and its given approximate solution $\widetilde{u}(x)$, the residue ρ is a function defined as

$$\rho(x) = |f(x) - a\widetilde{u}''(x) - b\widetilde{u}'(x) - c\widetilde{u}(x)| \tag{23}$$

It might be considered as the distance between the exact right-hand side $f(x)$ (and thus, the expected value of u'') to the second derivative of the approximation; i.e., $\widetilde{u}''$. Just for an example, see Figure 2, showing the left-hand side $a\widetilde{u}''(x)+b\widetilde{u}'(x)+c\widetilde{u}(x)$ compared to the right-hand side $f(x)$ for the equation (17). The approximate solution under consideration $\tilde{u}$ is the one obtained on the coarse grid in the second iteration of the two-grid h-adaptation algorithm with quadratic B-splines. This exactly corresponds to what is shown in Figure 1b.

The refinement in this approach (again, basically splitting the elements into halves) is performed on the elements with the highest residue values. It needs to be emphasized that this approach takes just one grid instead of two (as required in the two-grid strategy). The details are explained in algorithm 2.

Just like before, the sequence of Figures 3a through 3e shows iterations of the algorithm applied to solve a sample elliptic equation. The solution is approximated with quadratic B-splines.

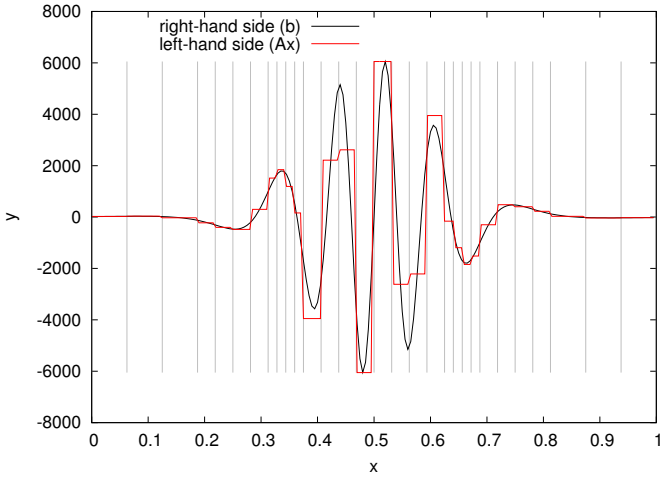


Figure 2. Approximate solution to a sample elliptic equation computed with quadratic B-splines — right-hand side (u'') vs. left-hand side ($\widetilde{u}''$)

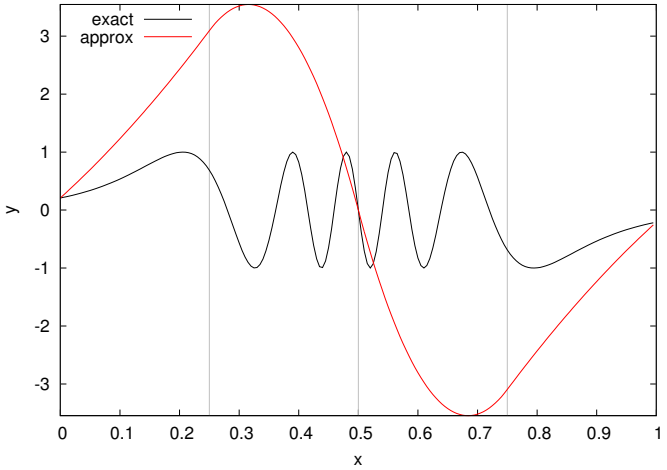
Algorithm 2 Residue-based adaptation

Solve the problem over the mesh
for each element E_i **do**
  $x_i \leftarrow$ center of E_i
  $\triangleright$ compute the residue ρ_i at x_i
  $\rho_i \leftarrow |f(x_i) - a\widetilde{u}''(x_i) - b\widetilde{u}'(x_i) - c\widetilde{u}(x_i)|$
end for
$\rho_{max} \leftarrow$ find the maximal ρ_i
for each element E_i **do**
  if $\rho_{max} \cdot \tau < r_i$ **then**
    locally refine the grid at E_i
  end if
end for

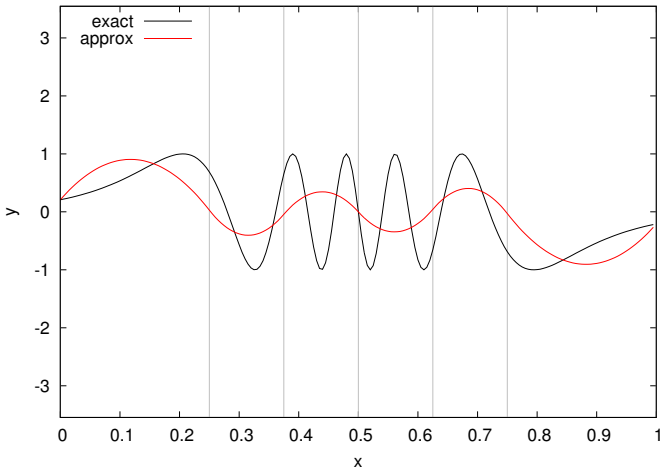
4. Convergence analysis

The following section focuses on an analysis of the residual error and convergence of the solution for h-adaptation algorithm.

First, we observe a convergence for sample equation (17). Then, we discuss the influence of the utilized polynomial order on the convergence rate. To assess the convergence in the following sections, we utilize residual error (which is actually residue) as defined in (23).

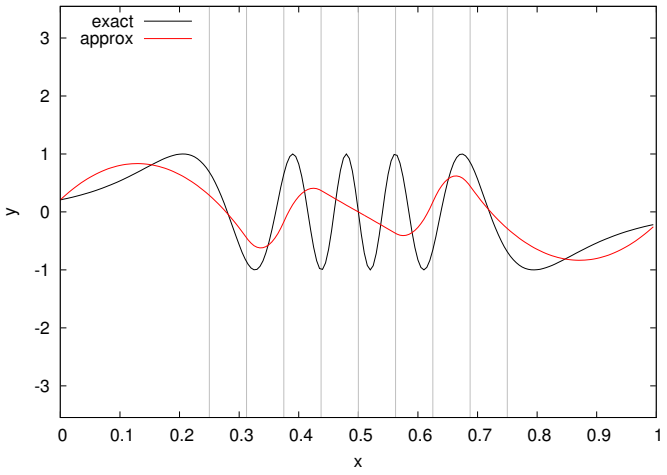


(a) Iteration 1. The solution is computed on a single four-element mesh. The red curve indicates the approximate solution, and the black line shows the exact analytic solution. The residue at element (ρ_i in algorithm 2) exceeds the maximal residue times threshold τ (here, fixed at 20%) only for the two central elements (E_2 and E_3). The algorithm, therefore, decides to refine these two elements, which leads to the mesh shown in Figure 3b.

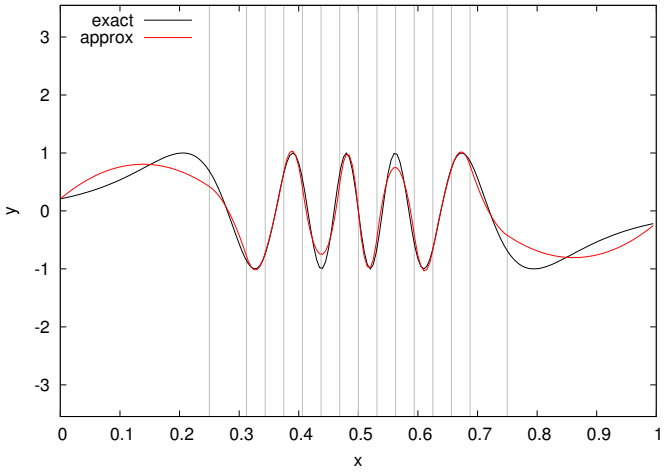


(b) Iteration 2. After this iteration, the algorithm decides to split the central elements once more, resulting in the mesh shown in Figure 3c.

Figure 3. Exact and approximate solutions on the grid in iterations 1 through 5 of the residue-based h-adaptation algorithm. Quadratic B-splines, equation (17) — the very same configuration as in the two-grid approach showcased in Figures 1a through 1e.

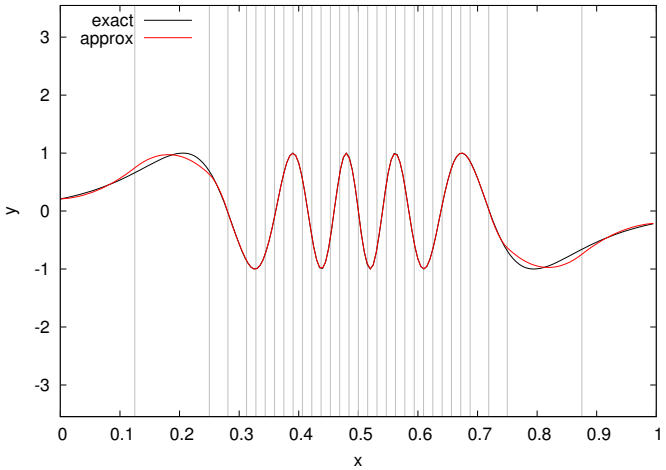


(c) Iteration 3



(d) Iteration 4. Let us compare the current results with Figure 1d, which is the mesh left after the fourth iteration of the two-grid algorithm. For this particular sample equation (17), the two-grid approach performs weaker at picking the elements to refine. The residue-based method tends to converge quicker in this case.

Figure 3



(e) Iteration 5. Only now, the algorithm decided to refine the extreme left and right elements for the very first time.

Figure 3

4.1. Convergence for the sample

Let us, for a moment, go back to sample equation (17) and its two-grid solution process illustrated by the sequence of Figures 1a through 1e. Plots 4a through 4e show the residual error over the interval for iterations 1 - 5 of the two-grid algorithm. Just to clarify — as $\tilde{u}$ in the definition of residual error (see [23]), we consider the coarse-mesh solution, not the fine-mesh.

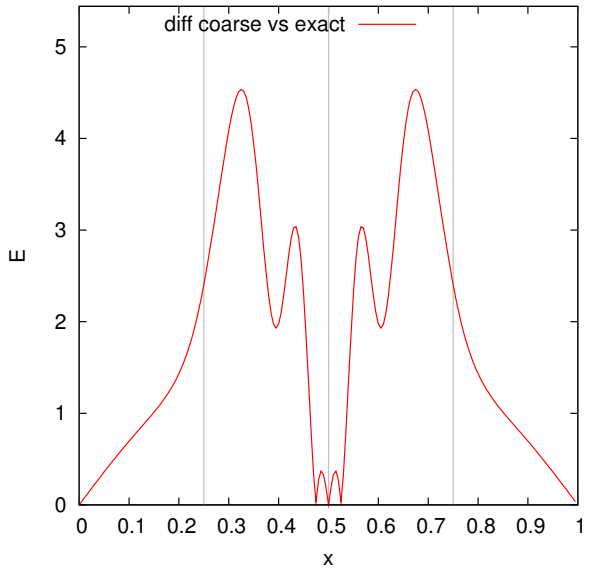
4.2. Comparison of convergence by polynomial order

Let us investigate how the chosen polynomial order affects the convergence of the solution.

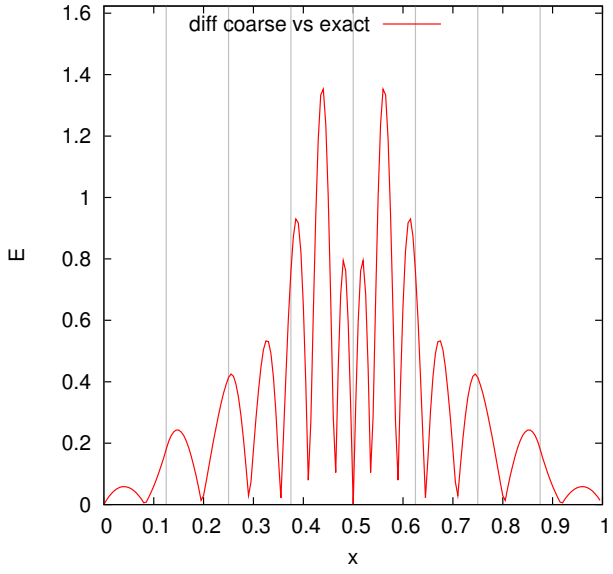
A tradeoff between accuracy and performance arises when increasing the polynomial order (see Figure 5). Since the partition-of-unity rule enforces a single polynomial order for all B-splines, we decided to leave the choice as to the actual order up to the end-user of the library.

The required computation time significantly increases with higher orders. On the other hand, with the use of higher orders, the solution reaches better convergence rates.

Plot 6 shows the time required to achieve the specified level of relative error for various values of the polynomial order. For higher orders, the code is harder to parallelize, and therefore a substantial overhead is incurred for low problem sizes — it only pays off for larger problems to utilize high-order polynomials.

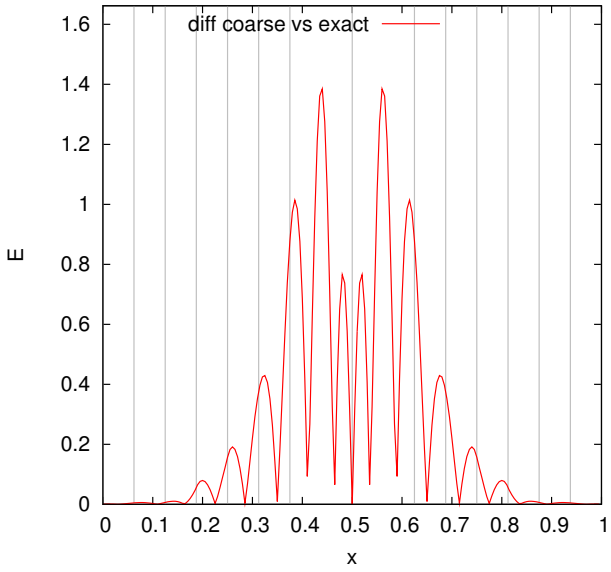


(a) Iteration 1

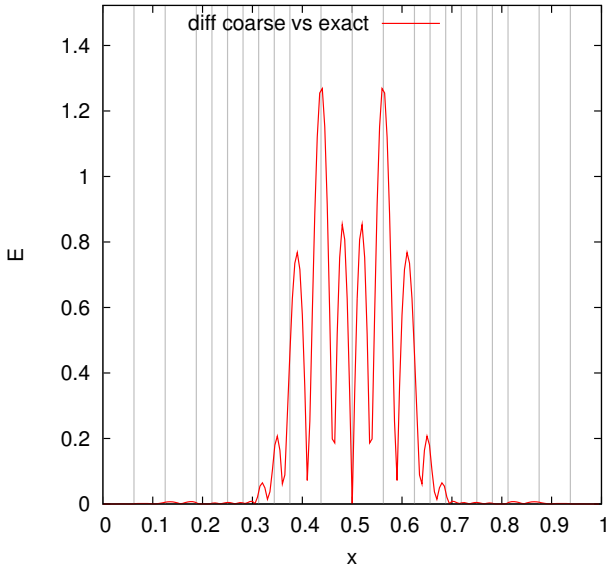


(b) Iteration 2

Figure 4. Residual error in iterations 1 through 5 of the two-grid h-adaptation algorithm. Quadratic B-splines, equation (17).

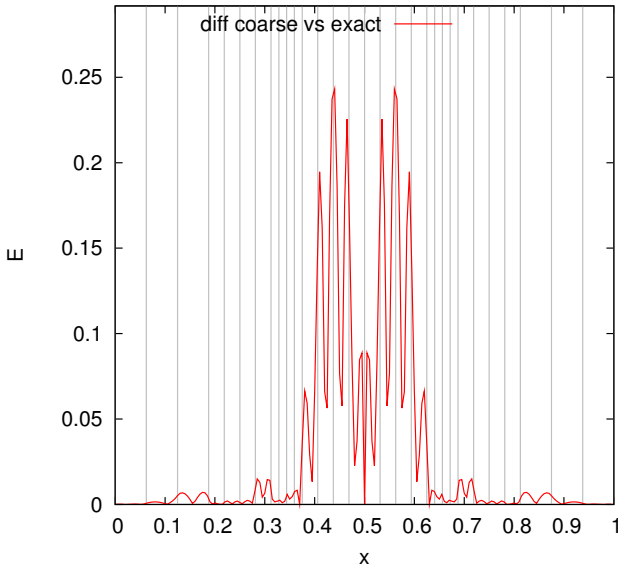


(c) Iteration 3



(d) Iteration 4

Figure 4



(e) Iteration 5

Figure 4

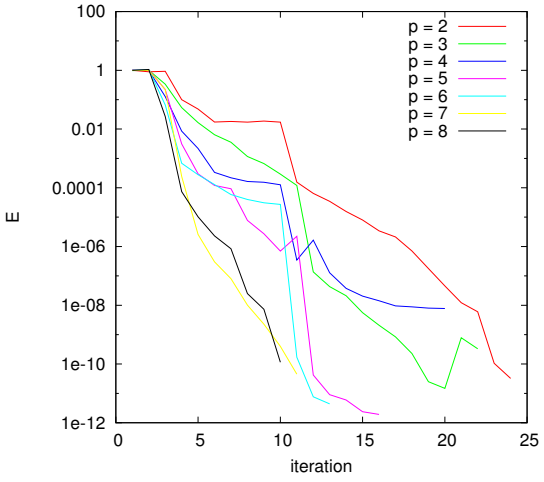


Figure 5. Residual error in h-adaptation algorithm by polynomial order and iteration

5. Parallelization and performance

In this section, we describe the approach we took for parallelization and performance measures.

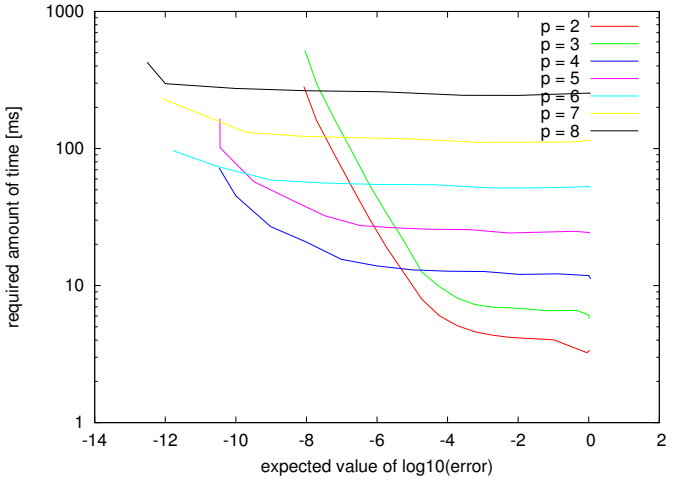


Figure 6. Time required to reach a specified value of residual error for various B-spline orders

Most of the computing time within the stiffness matrix fill-up process is spent inside the quadrature procedure, since each element of the left-hand side matrix is actually computed as an integral. Quadratures, in turn, involve computing B-spline values very frequently. It is, therefore, desirable to reduce the number of direct B-spline computations as much as possible.

5.1. B-spline value caching

In the standard (both parallel and sequential) Gauss-Legendre quadrature implementations, values of all B-splines (except for those located at the interval boundaries) must be computed $p+1$ times at every single quadrature point (p standing for the polynomial order).

Let us take a look at the following fragment of the stiffness matrix for third-order B-splines. N_i, N_j indicate the basis functions whose product must be integrated to obtain the value of each matrix entry. Let us pick an arbitrary quadrature point x on the very first finite element covered by N_7. Then, all of the matrix entries for which it is necessary to compute N_7 at the point x (for the sake of the quadrature) are highlighted.

By definition, each cubic B-spline (except for those located on the boundaries) occupies $3+1=4$ finite elements. For the case above, N_7 needs to be computed at x exactly seven times. In general, each computation of p-ordered B-spline N_i^p at any point x (except for points at a few extreme left and right intervals) must be repeated $p+1$ times.

To avoid repeating costly computations (especially for higher-order bases), B-spline values are cached in a 2-dimensional array before any quadrature is performed.

Table 1

Fragment of the matrix for third order B-splines

$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\iddots$
$\dots$	N_3, N_3	N_3, N_4	N_3, N_5	N_3, N_6	0	0	$\dots$
$\dots$	N_4, N_3	N_4, N_4	$\mathbf{N_4, N_5}$	N_4, N_6	N_4, N_7	0	$\dots$
$\dots$	N_5, N_3	$\mathbf{N_5, N_4}$	$\mathbf{N_5, N_5}$	$\mathbf{N_5, N_6}$	$\mathbf{N_5, N_7}$	N_5, N_8	$\dots$
$\dots$	N_6, N_3	N_6, N_4	$\mathbf{N_6, N_5}$	N_6, N_6	N_6, N_7	N_6, N_8	$\dots$
$\dots$	0	N_7, N_4	$\mathbf{N_7, N_5}$	N_7, N_6	N_7, N_7	N_7, N_8	$\dots$
$\dots$	0	0	N_8, N_5	N_8, N_6	N_8, N_7	N_8, N_8	$\dots$
$\iddots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

The array stores the value of each B-spline at each point of the quadrature that this particular B-spline covers.

The integration procedure, in turn, takes advantage of the precomputed values and never needs to evaluate any B-spline at any point directly. Thus, it saves a massive amount of time that would otherwise be spent on the costly repeated calls of B-spline procedure for the same parameters again and again.

Just to clarify, the caching phase is performed on a per-grid basis, and values computed for a grid cannot be simply utilized for refined versions of this grid. This implies that, in the two-grid approach, the caching phase must be run separately for both coarse and fine grids in every single iteration.

5.2. Performance measures

This section covers the time measures performed for the parallel code in our package as compared with their sequential versions.

Both recently mentioned stages (i.e., caching the values of B-splines at the quadrature points and computing the stiffness matrix) benefit massively from parallelization. Figure 7 compares the time spent for various phases of the stiffness matrix fill-up process in both the parallel and sequential modes of execution. Besides the total left-hand side fill-up time, this includes the time spent for caching B-spline values and the time spent performing actual quadratures with the use of the cached B-spline values.

All of the following parallel (CUDA) results (figures 7, 8, 9, 10) have been obtained on an NVIDIA Tesla K20c GPU. A single core of an Intel i7 processor has been utilized as a sequential machine. In all of the performance-related figures, N corresponds to the grid size.

Left-hand side fill-up time strongly depends on the B-spline order. The collection of figures 8 shows the total execution time (including both caching and actual integration) depending on the chosen order of B-splines.

The results suggest that increasing the order of B-splines by one increases the execution time for large grids approximately by half an order of magnitude.

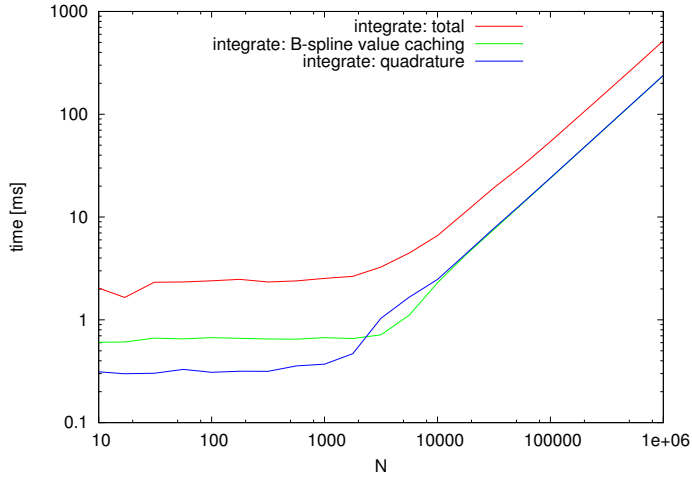


Figure 7. B-spline caching, integration, and total left-hand side fill-up time for quadratic B-splines.

Other parallelizable steps of execution include right-hand side vector fill-up, solving the system of linear equations and h-adaptation itself.

If it comes to the right-hand side (load vector) fill-up phase, parallelization simply means computing each value in a separate thread (since the entries of the vector do not depend on each other in any way).

For the solver phase, the algorithm described above can be wired up to the parallel multifrontal implementation of the solver, as described in [18].

For the refinement phase, the coarse-vs-fine relative errors must be determined for every single element, and it must be decided whether to refine each particular element or not, depending on the maximal error and the threshold τ.

Figures 9 and 10 show a performance comparison between the sequential and parallel versions of right-hand side fill-up and h-adaptation, respectively.

6. Complexity

This section features a brief analysis of the computational complexity of the phases of the algorithm. We consider the time spent for a single iteration, for mesh size N, and polynomial order p. It is assumed that an infinite number of cores is available.

The caching of each B-spline value is performed in a separate logical thread. Computing a B-spline value in a given point takes $O(p^2)$ time, and this is also the overall complexity of the phase.

Thanks to prefetching the B-spline values, integration takes a constant time for each element; thus, the overall complexity also being $O(1)$.

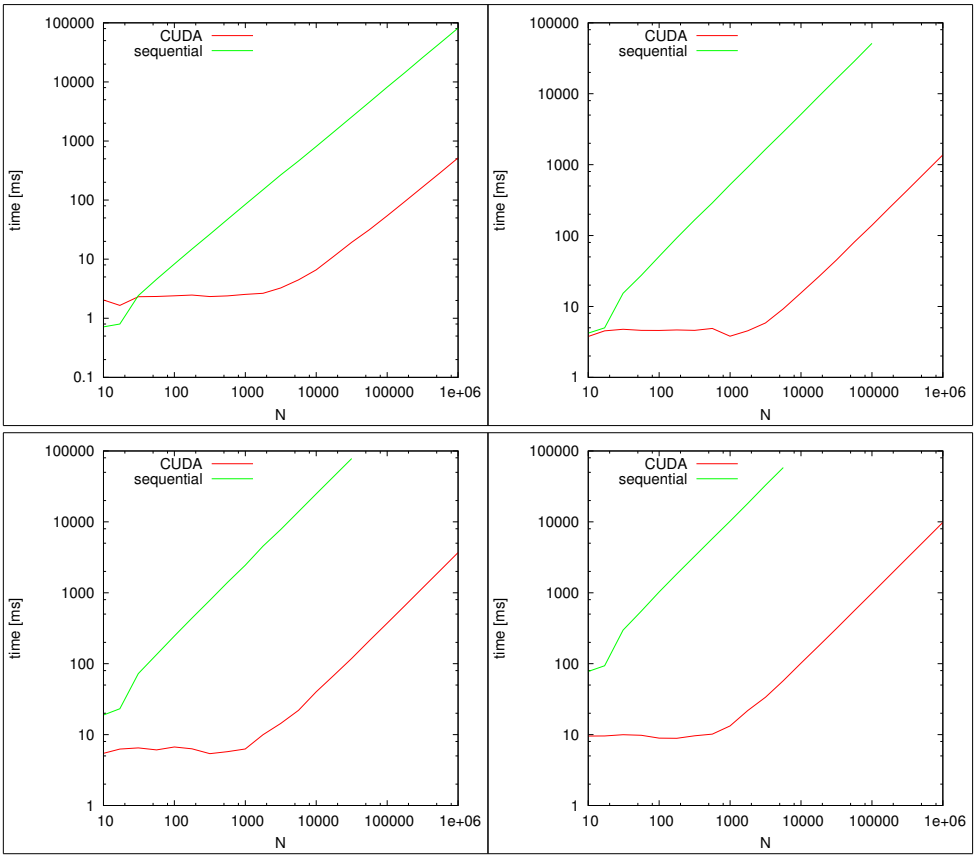


Figure 8. Stiffness matrix fill-up time for CPU and GPU implementations. **Top left panel:** Quadratic B-splines. **Top right panel:** Cubic B-splines. **Bottom left panel:** Quartic B-splines. **Bottom right panel:** Quintic B-splines.

Since the elements do not depend on each other and cached B-spline values are utilized, load vector fill-up also takes only $O(1)$.

Assuming the approach described in [18] is employed, the solver phase is done in $O(p^2 \log N)$ time.

The h-adaptation phase can be done in logarithmic time. More specifically, for each element, it is necessary to compute relative error ($O(1)$), then determine the maximum relative error ($O(\log N)$ with a proper fast parallel reduction algorithm). Afterwards, decide for each element whether it should be refined or not ($O(1)$). Finally, construct the refined grid – again, with the use of a parallel prefix sum algorithm, $O(\log N)$ time can be achieved.

The above facts imply that the total time complexity of a single h-adaptation algorithm adaptation is $O(p^2 \log N)$.

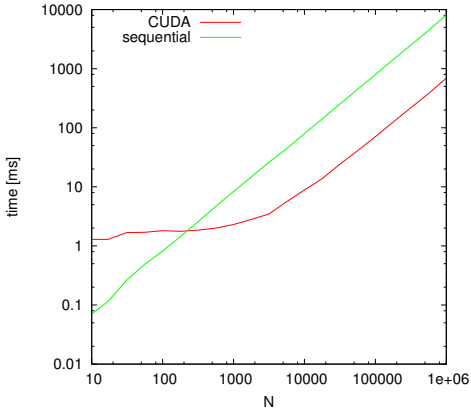


Figure 9. Load vector fill-up time for quadratic B-splines

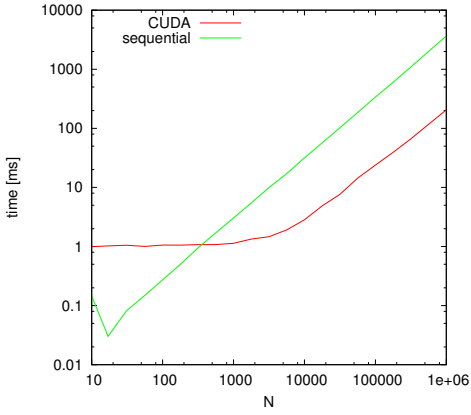


Figure 10. h-adaptation time for quadratic B-splines

7. Conclusions and future works

A tradeoff between accuracy and performance arises when increasing the polynomial order. Since the partition-of-unity rule enforces a single polynomial order for all B-splines, we decided to leave the choice as to the actual order up to the end-user.

If it comes to the stiffness matrix and load vector fill-up, a tremendous speedup might be achieved when a GPU is employed (since each h-adaptation iteration is easily parallelizable).

For h-adaptation with B-splines, la ower adaptation threshold should be chosen than for hierarchical polynomials. For the latter, a value of 33% is recommended; whereas, for isogeometric h-adaptation, values of 10-20% perform best.

At certain cases, the residue-based approach reaches better convergence rates than the two-grid approach, mainly due to the fact that it decides about the elements to split earlier and more adequately. In addition, whereas the latter computes the solution on two grids, the former requires only a single FEM phase to run in a single iteration, leading to a substantial speedup.

Isogeometric h-adaptation may as well be considered in the cases of 2D and 3D problems – instead of a 1D knot vector, we operate on *patches*; i.e., tensor products of 2 (in 2D) or 3 (in 3D) knot vectors. They can be directly mapped to basic surfaces, like a cylinder in 3D. By combining multiple patches, full coverage can be provided even for complex objects. Thus, our future research in the topic will focus on complexity and adaptation strategies for 2D and 3D problems.

Acknowledgements

The work presented in this paper was supported by Polish National Science Center grant no. DEC-2012/07/B/ST6/01229.

References

[1] Babuška I., Guo B.: The hp-version of the finite element method, Part I: The basic approximation results. *Computational Mechanics*, pp. 21–41, 1986.

[2] Babuška I., Guo B.: The hp-version of the finite element method, Part II: General results and applications. *Computational Mechanics*, pp. 203–220, 1986.

[3] Babuška I., Rheinboldt W.C.: Error Estimates for Adaptive Finite Element Computations. *SIAM Journal of Numerical Analysis*, vol. 15(4), pp. 736–754, 1978.

[4] Bazilevs Y., Beirão Da Veiga L., Cottrell J.A., Hughes T.J.R., Sangalli G.: Isogeometric analysis: Approximation, stability and error estimates for h-refined meshes. *Mathematical Models and Methods in Applied Sciences*, vol. 16(7), pp. 1031–1090, 2006.

[5] Becker R., Hartmut K., R. R.: Adaptive finite element methods for optimal control of partial differential equations: Basic concept. *SIAM Journal on Control and Optimisation*, vol. 39(1), pp. 113–132, 2000.

[6] Belytschko T., Tabbara M.: H-Adaptive finite element methods for dynamic problems, with emphasis on localization. *International Journal for Numerical Methods in Engineering*, vol. 36(24), pp. 4245–4265, 1993.

[7] Cottrell J., Hughes T.J.R., Bazilevs Y.: *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley Publishing, 1st ed., 2009, ISBN 0470748737, 9780470748732.

[8] Demkowicz L.: *Computing with hp adaptive finite element methods. Part I. Elliptic and Maxwell problems with applications*. Taylor & Francis, CRC Press, 2006.

[9] Eriksson K., Johnson C.: Adaptive Finite Element Methods for Parabolic Problems I: A Linear Model Problem. *SIAM Journal on Numerical Analysis*, vol. 28(1), pp. 43–77, 1991.

[10] Gang B., Guanghui H., Di L.: An *h*-adaptive finite element solver for the calculations of the electronic structures. *Journal of Computational Physics*, vol. 231(14), pp. 4967–4979, 2012.

[11] Kardani M., Nazem M., Abbo A.J., Sheng D., Sloan S.W.: Refined h-adaptive finite element procedure for large deformation geotechnical problems. *Computational Mechanics*, vol. 49(1), pp. 21–33, 2012.

[12] Krawezik G., Poole G.: Accelerating the ANSYS Direct Sparse Solver with GPUs. *Symposium on Application Accelerators in High Performance Computing, SAAHPC*, 2009.

[13] Kuźnik K., Paszyński M., Calo V.: Grammar-Based Multi-Frontal Solver for One Dimensional Isogeometric Analysis with Multiple Right-Hand-Sides. *Procedia Computer Science*, vol. 18(0), pp. 1574 – 1583, 2013.

[14] Lucas R.F., Wagenbreth G., Davis D.M., Grimes R.: Multifrontal Computations on GPUs and Their Multi-core Hosts. *VECPAR*, *Lecture Notes in Computer Science*, vol. 6449, pp. 71–82, Springer, 2010.

[15] Niemi A.H., Babuška I., Pitkäranta J., Demkowicz L.: Finite element analysis of the Girkmann problem using the modern hp-version and the classical h-version. *Engineering with Computers*, vol. 28(2), pp. 123–134, 2012.

[16] Nochetto R.H., Siebert K.G., Veeser A.: *Multiscale, Nonlinear and Adaptive Approximation*. Springer, 2009.

[17] Piegl L., Tiller W.: *The NURBS Book (Second Edition)*. Springer-Verlag New York, Inc., 1997.

[18] Woźniak M., Kuźnik K., Paszyński M., Calo V., Pardo D.: Computational cost estimates for parallel shared memory isogeometric multi-frontal solvers. *Computers and Mathematics with Applications*, vol. 67(10), pp. 1864–1883, 2014.

[19] Yu C.D., Wang W., Pierce D.: A CPU-GPU Hybrid Approach for the Unsymmetric Multifrontal Method. *Parallel Comput.*, vol. 37(12), pp. 759–770, 2011.

Affiliations

Paweł Lipski
AGH University of Science and Technology, Krakow, Poland, lipski@student.agh.edu.pl

Maciej Paszyński
AGH University of Science and Technology, Krakow, Poland, paszynsk@agh.edu.pl

Received: 8.08.2015
Revised: 2.02.2016
Accepted: 3.02.2016