BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 66, No. 6, 2018
DOI: 10.24425/bpas.2018.125938

Heuristic algorithm to predict the location of C^0 separators for efficient isogeometric analysis simulations with direct solvers

A. PASZYŃSKA[1], K. JOPEK[2], M. WOŹNIAK[2], and M. PASZYŃSKI[2]*

[1]Jagiellonian University, Faculty of Physics, Astronomy and Applied Computer Science, 11 Łojasiewicza St., 30-348 Krakow, Poland
[2]AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, 30 Mickiewicza Ave., 30-059 Krakow, Poland

Abstract. We focus on two and three-dimensional isogeometric finite element method computations with tensor product C^k B-spline basis functions. We consider the computational cost of the multi-frontal direct solver algorithm executed over such tensor product grids. We present an algorithm for estimation of the number of floating-point operations per mesh node resulting from the execution of the multi-frontal solver algorithm with the ordering obtained from the element partition trees. Next, we propose an algorithm that introduces C^0 separators between patches of elements of a given size based on the stimated number of flops per node. We show that the computational cost of the multi-frontal solver algorithm executed over the computational grids with C^0 separators introduced is around one or two orders of magnitude lower, while the approximability of the functional space is improved. We show O(NlogN) computational complexity of the heuristic algorithm proposing the introduction of the C^0 separators between the patches of elements, reducing the computational cost of the multi-frontal solver algorithm.

Key words: refined isogeometric analysis, finite element method, multi-frontal direct solver, heuristic algorithms.

1. Introduction

The finite element method (FEM) is a popular approach [1–4] to approximate solutions of partial differential equations (PDEs). The discrete approximation of the solution of a PDE using FEM uses a computational mesh to describe the geometry of the domain and basis functions spread over the mesh to approximate the solution of the PDE. This discrete approximation results in a sparse global system of linear equations.

Basis functions spread over the computational mesh. Some of the basis functions overlap, and they generate non-zero entries in the global system of linear equations. The weak form of PDE prescribes a way how overlapping basis functions interact and generate non-zero entries in the global matrix. The weak form of the PDE is obtained by taking the L2 scalar products with test functions and performing the integration by parts [1–4].

The multi-frontal solver is the state-of-the-art algorithm for solving sparse linear systems resulting from finite element method discretizations [5–8].

Classical multi-frontal solvers [5–7] use an ordering constructed based on the analysis of the sparsity of the global matrix [12]. Based on the ordering, they construct the elimination tree and perform the multi-frontal elimination of rows of the sparse global matrix, using the tree.

We have shown that reverse approach is possible, namely the construction of the element partition tree based on the structure of the mesh [13, 8], and elimination of matrix rows by browsing the element partition tree. The corresponding ordering can be obtained by postorder traversal of the element partition tree. The resulting number of floating-point operations is similar, with some advantage of the element partition tree in the case of adaptive, non-uniform grid [13].

The main idea of the isogeometric analysis (IGA) [14] is to apply tensor products of higher order global continuity B-splines or NURBS [30] basis functions for finite element method (FEM) simulations. The IGA-FEM has multiple applications in time-dependent simulations, including phase field models [15, 16], cancer growth simulations [18], or phase-separation simulations [19, 20], wind turbine aerodynamics [22], incompressible hyper-elasticity [17], turbulent flow simulations [26], transport of drugs in cardiovascular applications [21] or the blood flow simulations and drug transport in arteries simulations [24, 25].

Recently, a modification to the IGA has been proposed, namely the refined isogeometric analysis (rIGA) methodology [27]. The rIGA method postulates that introduction of C^0 separators between patches of elements with tensor products B-splines basis functions reduces the computational cost of the multi-frontal direct solvers. The method has been investigated on uniform [27] or adaptive grids [28, 29]. The location of C^0 separators for adaptive grids is somehow natural, since they are located between the refinement levels [28, 29]. However, there is no algorithm for automatic selection of the location of C^0 separators between the patches of elements on uniform grids [27]. Moreover, proper selection of the locations of the separators is important, since they can save up to two orders of magnitude in terms of the execution time and the number of the floating point operations of the resulting direct solver algorithm.

In this paper, we propose an algorithm for estimation of the number of floating-point operations per unknown (namely,

*e-mail: paszynsk@agh.edu.pl

Manuscript submitted 2017-08-20, revised 2018-03-09, initially accepted for publication 2018-04-30, published in December 2018.

the coefficient of the basis function approximating the solution, called the degree of freedom), "flops per dof", resulting from the execution of the multi-frontal solver algorithm over the uniform IGA-FEM grids. Next, we introduce an algorithm that uses the estimated number of flops per dof to propose the locations of the C^0 separators between patches of elements. The proposed algorithm has $\mathcal{O}(NlogN)$ computational complexity with respect to the number of degrees of freedom N. We show that the separators selected by our algorithm allow reducing the computational cost of the multi-frontal solver algorithm. We conclude the paper with a sequence of numerical experiments.

We focus on the optimization of the sequential in-core multi-frontal solver [5–7], although the C^0 separators obtained from our algorithm can be possibly utilizes to speed up shared-memory [31–33] or distributed-memory [9–11] implementations as well. This will be the topic of our future work.

The structure of the paper is the following. We start from introducing the idea of the refined isogeometric analysis in Section 2. Next, we present the algorithm for construction of the element partition trees in Section 3. Later, in Section 4, we derive the mathematical formulas for the estimation of the number of flops per dofs. In Section 5 we introduce the algorithm for selection of the locations of the separators between patches of elements. Finally, we estimate the computational complexity in Section 6 and we conclude the paper after the numerical results presented in Section 7. We also present an Appendix describing the implementation of our flops estimation package.

2. Idea of the refined isogeometric analysis

In this section, we introduce the idea of the refined isogeometric analysis on a simple one-dimensional example. Let us consider a sixteen finite elements, and cubic B-splines. In such the case, we have four B-spline basis functions per each element. All these four B-spline basis functions overlap, so if we generate element frontal matrix, with rows and columns associated to the four local B-splines, the element frontal matrix is full. The entries in the matrix represent interactions between the B-spline basis functions, which overlap. All the entries in the frontal matrix are non-zeros, and the values depend on the weak formulation of the PDE being solved. This means that the local element matrices are of the size four times four, and they are full of non-zero entries, as it is presented in Fig. 1.

If we define the cubic B-spline basis functions over these sixteen finite elements, the local element matrices overlap to the greatest extent, as it is presented in Fig. 2. This is because the B-spline basis functions of order p spread over $p+1$ elements,

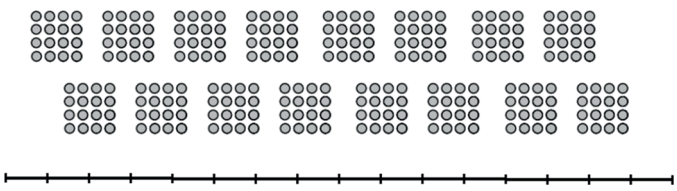

Fig. 1. 16 finite elements with 16 element matrices

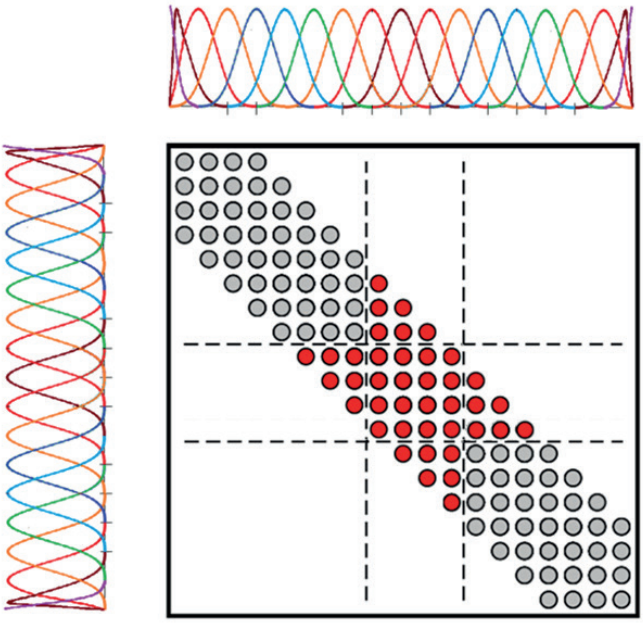

Fig. 2. One-dimensional discretization with standard finite element method with cubic polynomials

and they overlap with three other B-splines. In our case, we obtain the global matrix of size $N = 16 + 3 = 19$, with dense diagonals. This is because each element local matrix overlaps with other matrices by three rows and columns. The IGA matrices are smaller than rIGA and FEM matrices, but they are denser.

On the other hand, if we define the hierarchical polynomials of the third order, like the Lagrange polynomials, over these sixteen finite elements, the element local matrices will overlap minimally, as it is presented in Fig. 3. When we look at the

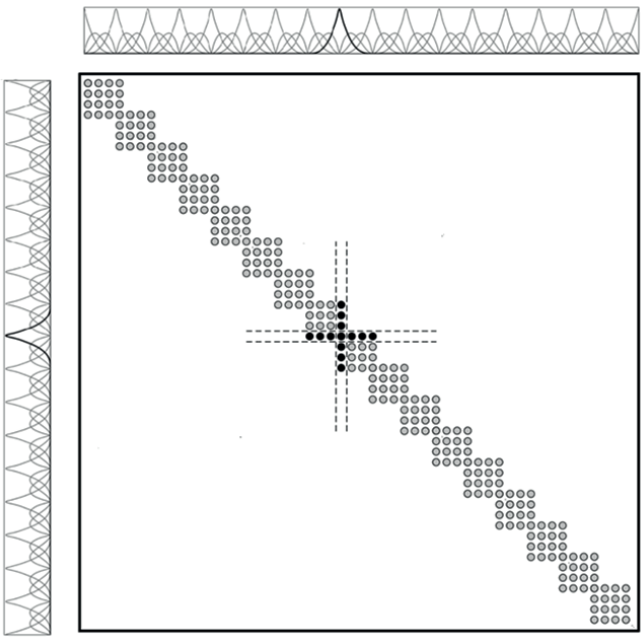

Fig. 3. One-dimensional discretization with standard finite element method with cubic polynomials

basis functions, it is equivalent to perfoming knot insertion on the original B-spline to obtain C^0 separators. Please note that by introducing these C^0 separators we have enriched the space of the basis functions, so the approximability of the new system is better. The higher continuity cubic B-splines cannot approximate with zero error the cubic functions with a corner. The basis refined with C^0 separators can approximate with zero error the functions with the corners located in between the finite elements. Both cubic B-spline basis and the refined cubic B-spline basis functions can approximate with zero error a highly continuous polynomial of the third order. If we look at the global matrix, we see that element matrices overlap just by one row and column. Thus, we have largest matrix $N = 3*16 + 1 = 49$, but the diagonals are sparse, as it is presented in Fig. 3.

Summing up, the traditional FEM generates larger but sparser matrices, and IGA method generates smaller but denser matrices. Moreover, IGA provides higher global continuity of the solution, but it cannot approximate C^0 functions.

Now, the refined isogeometric analysis (rIGA) comes into the picture. We can make a compromise between the size of the matrix and its density. Introducing C^0 separators every four elements, we obtain the matrix that is smaller than traditional finite element method, and sparser than isogeometric finite element method. This is illustrated in Fig. 4. The rIGA matrices are the compromise between small but dense IGA matrices, and large but sparse FEM matrices.

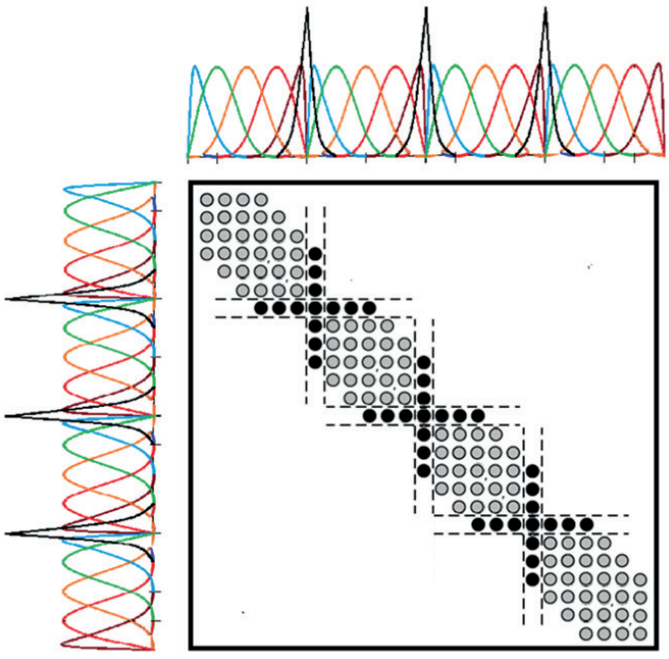

Fig. 4. One-dimensional discretization with refined isogeometric analysis with cubic B-splines and C^0 separators

The idea presented in this section can be generalized into two or three-dimensional tensor product structure grids.

Let us consider a three-dimensional patch with 16×16×16 elements, with quadratic B-splines. Such the patch can be obtained by introducing a knot-vector [0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 16 16] to any software compatible with IGA technology. We introduce such knot-vector for every direction x, y, and z. The repetition of the first and the last knot-points three times implies that quadratic B-splines are used uniformly over the mesh in every direction, x, y, and z. The introduced B-splines have C^1 continuity along the entire mesh. Another example involves the knot-vector defining cubic B-splines. Here we have [0 0 0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 16 16 16], and the cubic B-splines have C^2 continuity along the entire mesh, in a direction where we have introduced this knot-vector. In general, the repetition of the knot-point $p + 1$ times in the beginning and the end of the knot-vector implies B-splines basis functions of the order p. The introduced B-splines has C^{p-1} continuity across the entire mesh, in the direction defined by the knot-vector.

Now, we can partition our 3D patch of elements into eight patches with 8×8×8 elements. The partition is performed by repeating the knot-points on the interface between the subpatches. For example, in the first knot-vector defining quadratic B-splines, we repeat the knot-point 8 one time [0 0 0 1 2 3 4 5 6 7 8 8 9 10 11 12 13 14 15 16 16 16]. In the second knot-vector defining the cubic B-splines, we need to repeat the central knot-point three times [0 0 0 0 1 2 3 4 5 6 7 8 8 8 9 10 11 12 13 14 15 16 16 16 16]. Such the repetition of knot-points has the following interpretation. It reduces the global continuity of the B-splines basis functions from C^{p-1} down to C^0 at the point. It is equivalent to the introduction of one additional basis functions, the so-called C^0 separator, at the point of the repeated knots. For more details on the knot-vectors interpretation we refer to [14].

The computational cost of the multi-frontal direct solver executed on rIGA matrices is lower than for corresponding IGA and FEM matrices. We would like to emphasize that this idea of the reduction of the computational cost of the direct solver has been already presented in [27] in the context of sequential direct solvers. However, the paper [27], lacks the heuristic algorithm for selection of the location of the C^0 separators between parches of elements. We will propose such algorithm in the following sections.

3. Generation of the element partition tree based on two or three-dimensional mesh

Having the two or three-dimensional mesh prescribed by the knot vectors, we define first the algorithm generating the element partition tree (EPT). The EPT is a binary tree. Each leaf of the EPT has one finite element assign. The parent nodes of EPT have lists of finite elements assign. These lists are obtained by merging lists from the children nodes. A formal definition can be found in [13].

The algorithm is introduced with a pseudo-code below. The input to the algorithm is the list of all elements forming the two or three-dimensional mesh.

We provide the pseudo-code below. We first call the `create_root(`*root*`)` routine which creates the root node,

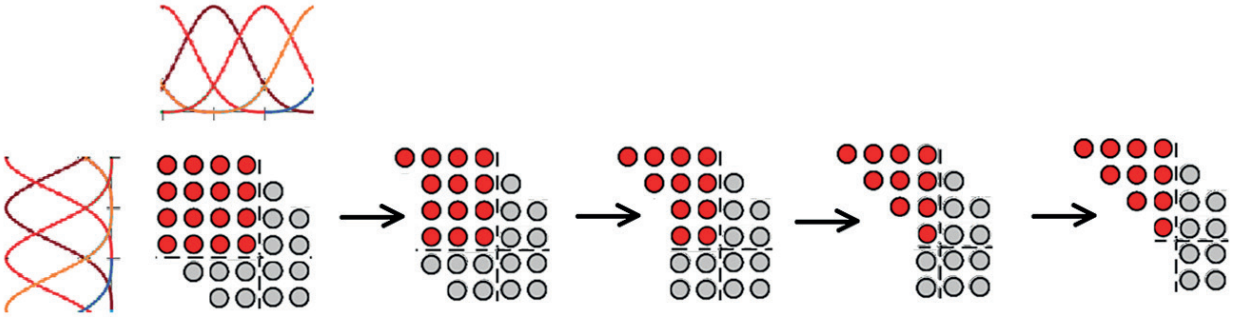

Fig. 5. C^{p-1} separators and elimination of two elements in 1D

prepares the list of all elements, from the first one `nbeg = 1` to the last one `nend = number_of_elements`. We assume that all the elements in the mesh are numbered in a continuous way and that each element from the mesh has its unique identifier. The list of all elements is stored in the *root* node. The `create_tree` routine passes the `root` node with the list of all elements to the `bisection` routine, which constructs the element partition tree in a recursive manner. It sorts the elements along the longest direction, by call `sort_elements_along_direction`. Then, it partitions the list of elements into two equally numbered parts, along with the selected directions, by calling `weighted_half`. Next, the two children nodes are created, and the two constructed sub-lists are assigned to the left and the right children accordingly. The routine `create_root` is called recursively until only one element remains in the sub-list.

```
node routine create_tree(root)
nbeg = 1; nend = number_of_elements;
nlevel = 1;
create root node;
assign elements [nbeg,nend] to root node;
call bisection(nbeg,nend,root,nlevel)
return root

recursive subroutine bisection
    (ibeg,iend,node,ilevel)
if (ibeg = = iend) then
  assign element [ibeg] to node;
  return;
endif
direction = (1,0,0);
sizex = sum_weights(direction,ibeg,iend);
direction = (0,1,0);
sizey = sum_weights(direction,ibeg,iend);
direction = (0,0,1);
sizez = sum_weights(direction,ibeg,iend);
if (sizex = = min(sizex,sizey,sizez) then
  direction = (1,0,0); size = sizex;
else if (sizey = = min(sizex,sizey,sizez) then
  direction = (0,1,0); size = sizey;
else if (sizez = = min(sizex,sizey,sizez) then
  direction = (0,0,1); size = sizez;
endif
call sort_elements_along_direction
    (direction,ibeg,iend)
call weighted_half
    (direction,ibeg,iend,size,ihalf);
create two children nodes,child1,child2
make child1 and child2 children of node;
if (ihalf>ibeg) then
  nbeg = ibeg; nhalf = ihalf;
  nlevel = ilevel + 1;
  assign elements [nbeg,nhalf] to child1;
  call bisection(nbeg,nhalf,child1,nlevel
endif
if (ihalf<iend) then
  nend = iend; nhalf = ihalf + 1;
  nlevel = ilevel + 1;
  assign elements [nhalf,nend] to child2;
  call bisection(nhalf,nend,child2,nlevel
endif
```

We assign weights to elements and use these weights to partition the elements recursively. The elements are partitioned into two halves in such a way, that elements weights on both sides are almost identical (equal if possible). We assume that the separator is a straight line (in 2D) or a plane (in 3D). This is because the C^0 separators on uniform grids can be only introduced along the x, y and z (in 3D) axes. There are different weights for each direction, x, y and z. The weights reflects the locations of the separators between elements along a given direction. To define the weights correctly, we use the following intuition. We analyze the complexity of elimination of elements in 1D, as it is presented in Fig. 5.

Let us focus on B-splines of order p. First, we assume that there are no separators between elements, and B-splines have C^{p-1} continuity. This is illustrated in Fig. 5 for cubic B-splines. In such the case, the element matrices overlap on the block $O(p^2)$. Each element contributes now to elimination of one $(p+1)\times(p+1)$ block. When we have n elements, the complexity of processing of all element $O(np^2+p^3)$, so the complexity per element is $O(p^2)$.

Let us assume that we have a patch of two times two finite elements with basis functions obtained from tensor products

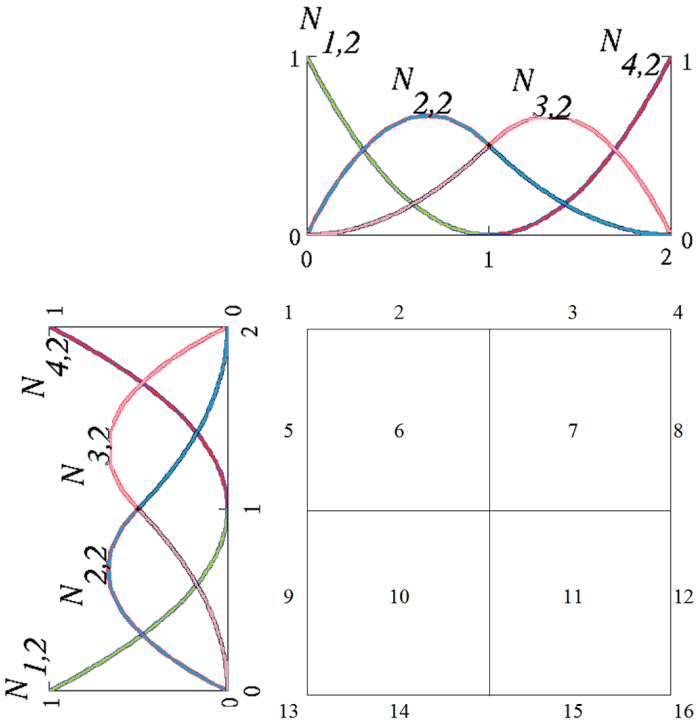

Fig. 6. Exemplary 2D IGA-FEM mesh. Global numbering of quadratic B-splines

of one dimensional B-splines defined by the knot vectors $\{0,0,0,1,2,2,2\}\times\{0,0,0,1,2,2,2\}$. This mesh is illustrated in Fig. 6.

Our implementation of the algorithm generates the EPT and stores the EPT in the following format. In our case, we have a total of 16 basis functions, and there are no nodes in the mesh, only the anchors where the B-spline basis functions are assigned and where they have maximum values. We assume that each node is one B-spline basis function, and there is one degree of freedom per node. We also use this assumption for higher order isogeometric grids. We also need to assign B-splines to elements where they spread their supports.

The syntax of the tree file *files/tree* is desrcibed in Appendix.

4. FLOPS estimator

The EPT file is the input for the flops estimator algorithm that computes a number of dof per degree of freedom (related to B-spline basis functions). In the following section, we define the formulas used for the flops estimation.

4.1. Equations for computing number of FLOPs per degree of freedom. Multi-frontal solver driven by partitioning tree maintains matrix M associated with every node of the tree. The matrix represents coefficients for degrees of freedom (DOFs) that are divided into two main parts:

1. DOFs that will be fully eliminated in given tree node.
2. DOFs being a part of Schur complement and will be merged into the matrix associated with the parent of a given tree node.

The arity of the above groups will be denoted as a and b for the number of DOFs to be eliminated and size of Schur complement respectively.

Therefore, content of the matrix M can be splitted into four main submatrices as it was illustrated by equation (1):

$$M_{n\times n} = \begin{bmatrix} A_{a\times a} & B_{a\times b} \\ C_{b\times a} & D_{b\times b} \end{bmatrix} \tag{1}$$

where: $n = a + b$.

In every node of the tree, solver computes Schur complement. Accounting above assumptions the Schur complement in our current notation can be expressed using equation (2):

$$D = D - CA^{-1}B. \tag{2}$$

Above we have one LU or Cholesky factorization, backward and forward substitution, one matrix by matrix multiplication $C\times\left(A^{-1}B\right)$ and one substraction. The costs for all of these operations will be described in the following subsections[1].

4.2. Factorization. Factorization is performed only on matrix $A_{a\times a}$. Exact cost of LU factorization is given by equation (3).

$$f_{\mathrm{lu}}(n) = \frac{2}{3}n^3 - \frac{1}{2}n^2 + \frac{5}{6}n \tag{3}$$

In case of symmetric, positively-definite matrix, we can use Cholesky decomposition which cost is given by the equation (4).

$$f_{\mathrm{chol}}(n) = \frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n \tag{4}$$

In case of LU factorization (typical for nonsymmetric or not positively-definite matrices) our cost expressed in terms of size of matrix A will be:

$$f_{\mathrm{lu}}(a) = \frac{2}{3}a^3 - \frac{1}{2}a^2 + \frac{5}{6}a \tag{5}$$

4.3. Backward and forward substitution. To compute: $A^{-1}B$ we need only to use factors of matrix A, that is, for LU factorization we will compute:

$$A^{-1}B = U^{-1}L^{-1}B$$

and for Cholesky decomposition:

$$A^{-1}B = \left(L^T\right)^{-1}L^{-1}B.$$

The cost for LU factorization is

$$f(n) = \mathrm{RHS}\left(n^2 - n\right).$$

[1] www.netlib.org/lapack/lawnspdf/lawn41.pdf, page 120

What directly leads to complexity of computing $A^{-1}B$ expressed in sizes of matrices A and B.

$$f_{\text{bfs|lu}}(a,b) = b\left(2a^2 - a\right) \tag{6}$$

and for Cholesky factorization we have similar equation

$$f_{\text{bfs|chol}}(a,b) = 2ba^2. \tag{7}$$

4.4. Matrix-by-matrix multiplication. Number of floating-point operations for multiplication of two matrices of well-known sizes is trivial, assuming $C_{b\times a}$ and $\left(A^{-1}B\right)_{a\times b}$ we have:

$$f_{\text{mult}}(a,b) = 2ab^2. \tag{8}$$

4.5. Substraction step. The last stage is subtraction step that in case of two matrices of size $b\times b$ the total number of FLOPS is:

$$f_{\text{sub}}(b) = b^2. \tag{9}$$

4.6. Merging step. Transferring data towards the root of the partitioning tree requires additional b^2 additions. We can take them also into consideration:

$$f_{\text{merge}}(b) = b^2. \tag{10}$$

4.7. Estimation of the total number of FLOPS in partitioning tree node. Finally we can sum up all of the above complexities to get total number of FLOPS for LU (11):

$$\begin{aligned} f(a,b) &= f_{\text{lu}}(a) + f_{\text{bfs|lu}}(a,b) + f_{\text{mult}}(a,b) + \\ &\quad + f_{\text{sub}}(b) + f_{\text{merge}}(b) = \\ &= \frac{2}{3}a^3 - \frac{1}{2}a^2 + \frac{5}{6}a + b\left(2a^2 - a\right) + \\ &\quad + 2ab^2 + 2b^2. \end{aligned} \tag{11}$$

Similarly, for Cholesky decomposition, the total number of FLOPS per matrix is given by the following equation (12):

$$\begin{aligned} f(a,b) &= f_{chol}(a) + f_{bfs|chol}(a,b) + f_{mult}(a,b) + \\ &\quad + f_{sub}(b) + f_{trans}(b) = \\ &= \frac{1}{3}a^3 - \frac{1}{2}a^2 + \frac{1}{6}a + 2ba^2 + 2ab^2 + 2b^2. \end{aligned} \tag{12}$$

4.8. FLOPS distribution. Computing number of FLOPS is trivial when comparing to the problem of assigning FLOPS to unknowns. The important goal is to distribute FLOPS as equally as possible. Equality means that every operation performed on submatrix should modify unknowns associated with both rows and columns of given block. This rule will be later referenced as "rule of equality".

Local system of linear equations is constructed as follows:

$$\begin{bmatrix} A_{a\times a} & B_{a\times b} \\ C_{b\times a} & D_{b\times b} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_a \\ \phi_{a+1} \\ \vdots \\ \phi_{a+b-1} \\ \phi_{a+b} \end{bmatrix} = \bar{b}.$$

Please notice that this notation uses a local enumeration of DOFs. Estimator traces mappings between local and global enumerations and fill flops array accordingly.

Assuming LU factorization, the first a variables will be totally eliminated and number of flops mapped to any of them is given by equation (13):

$$f_{\text{fact}(\phi_i)}(a) = \frac{2}{3}a^3 - \frac{1}{2}a^2 + \frac{5}{6}a. \tag{13}$$

For $i \in \{1, \ldots, a\}$. Equation (13) denotes cost of factorization related to unknown ϕ_i.

This is implication of equation (5) divided by number of unknowns (a). Similarly, we can derive the equation for Cholesky factorization.

This approach will give us a fair division of the number of flops between all of the unknowns involved in factorization step.

Situation becomes more complicated in terms of backward and foward substitution step presented in equation (6) and computing $CA^{-1}B$ (8). Estimator should divide number of steps equally between all of the degrees of freedom. Equation (14) shows mapping between flops and unknowns, trying to follow "near-equal" rule:

$$\begin{aligned} f_{\text{bfs|lu}}(a,b) + f_{\text{mult}}(a,b) &= ab(2a-1) + 2ab^2 = \\ &= \underbrace{2b(a,b)}_{\text{For DOFs related to Schur complement}} + \underbrace{ab(2a-1)}_{\text{For DOFs that will be fully eliminated}}. \end{aligned} \tag{14}$$

Computing $D = D - CA^{-1}B$ is simple and affects only degrees of freedom belonging to Schur complement.

Hence:

$$f_{\text{total}(\phi_i)} = \begin{cases} \left\lceil \frac{4a^2 - 3a + 5}{6} \right\rceil + b(2a-1) & \text{for } i \in \{1, \ldots, a\} \\ (2a+2)b & \text{for } i \in \{a+1, \ldots, a+b\} \end{cases}. \tag{15}$$

Please notice that rounding fraction up results in an overestimation of FLOPS.

The FLOPS estimation algorithm executed for the mesh presented in Fig. 6 generates the following output file. Please notice that the degrees of freedom are numbered from 0.

```
 0 = 9
 1 = 91
 2 = 91
 3 = 9
 4 = 200
 5 = 264
 6 = 264
 7 = 200
 8 = 200
 9 = 264
10 = 264
11 = 200
12 = 9
13 = 91
14 = 91
15 = 9
```

Our implementation of the FLOPS estimator generates also the output files in the VTK format.

5. Finding location of separators based on the flops per node map

To estimate the locations of the separators, we sum up the number of flops per 1D B-splines along particular axis. For example, in our exemplary mesh from Fig. 6, we have a tensor product of two knot vectors $\{0,0,0,1,2,2,2\}\times\{0,0,0,1,2,2,2\}$. This implies tensor product structure of four quadratic B-splines, along the x and y axis. Thus, we sum up the flops along the x and y axis. The result is $9+91+91+9=200, 200+264+264+200=928$, $200+264+264+200=928$, and $9+91+91+9=200$, the same for rows and for columns. We introduce the separator if there is a high local jump in the summation of flops, two orders (in 2D) and three orders (in 3D) of magnitude higher. We do not have such the local jumps here, so in this case the C^0 separator is not needed.

6. Computational complexity of the heuristic algorithm for selection of C^0 separators

The computational complexity of the algorithm is equal to the sum of the computational complexities of the following components

- Algorithm constructing the element partition tree. This algorithm partitions recursively a list of size N_e, constructing an equally weighted binary tree with weighted mesh elements at leafs. The computational complexity of this algorithm is $\mathcal{O}(N_e log N_e)$ where N_e is the number of elements.
- The algorithm estimating the number of flops by post-order traversal of the element partition tree. The algorithm has to visit all the nodes of the element partition tree, and update the cost for all the degrees of freedom The number of visited nodes is $\mathcal{O}(N_e log N_e)$, and the number of degrees of freedom updated is $\mathcal{O}(N)$.
- Finally the algorithm that sums up the costs in rows and columns. It has to visit all the degrees of freedom, so the computational complexity is $\mathcal{O}(N)$.

For the case of IGA-FEM, $\mathcal{O}(N_e)=\mathcal{O}(N)$, so the total computational complexity of the algorithm is $\mathcal{O}(NlogN)$.

7. Numerical results

We have executed our algorithm on representative two and three-dimensional grids with cubic B-splines. We plot the map of flops per node using our implementation of the interface with ParaView. When plotting the maps of flops from ParaView, it is necessary to setup Coloring = Results, Representation to either = Slice for 2D or = SurfacewithEdges. It is also necessary to 'Use log scale when mapping data to colors'. The map of flops per nodes is presented in Figs 7 and 8.

The generated maps of flops per mesh nodes enable to identify the locations of separators. From the plots we can read the location of separators are red lines, visible on logarithmic scale. We can discover them by summing up the flops along x, y and

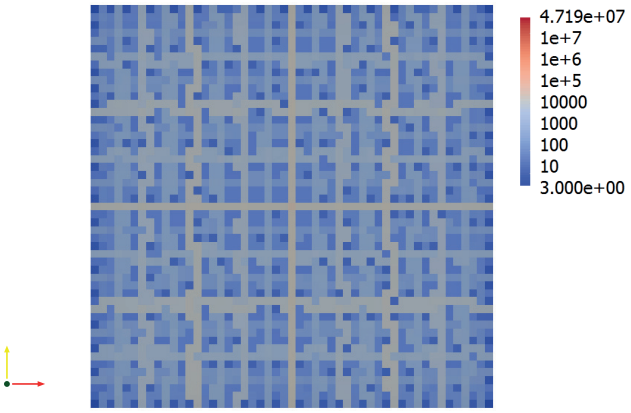

Fig. 7. Screenshot from ParaView for 2D mesh

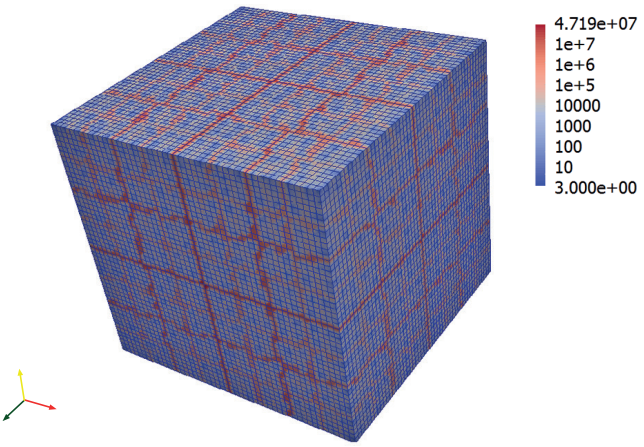

Fig. 8. Screenshot from ParaView for 3D mesh

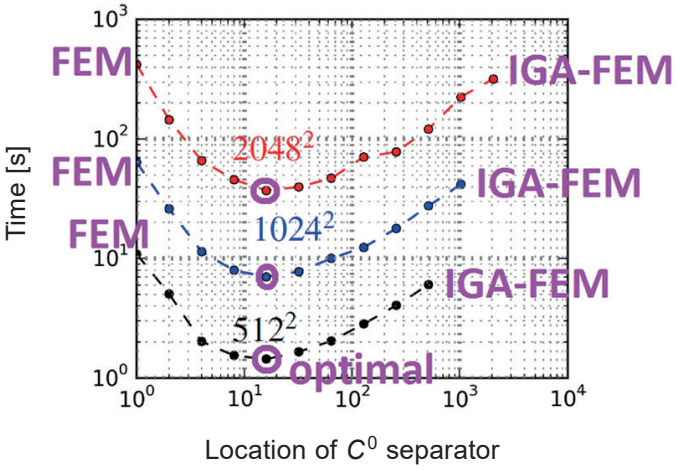

Fig. 9. The speedup of the multi-frontal solver executed on the 2D mesh with cubic B-splines with C^0 separators introduced

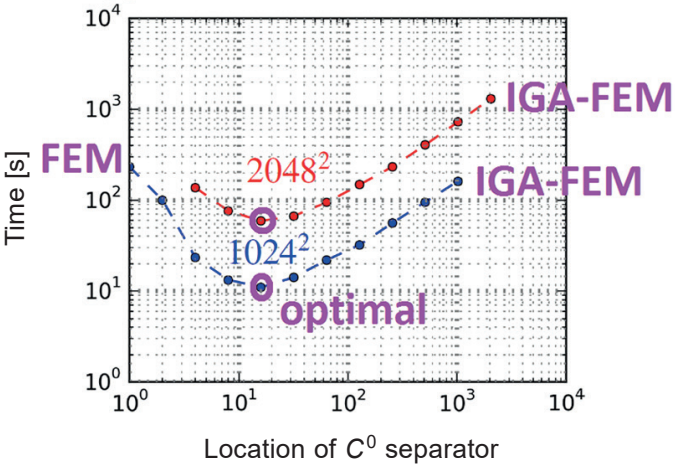

Fig. 10. The speedup of the multi-frontal solver executed on the 2D mesh with quintic B-splines with C^0 separators introduced

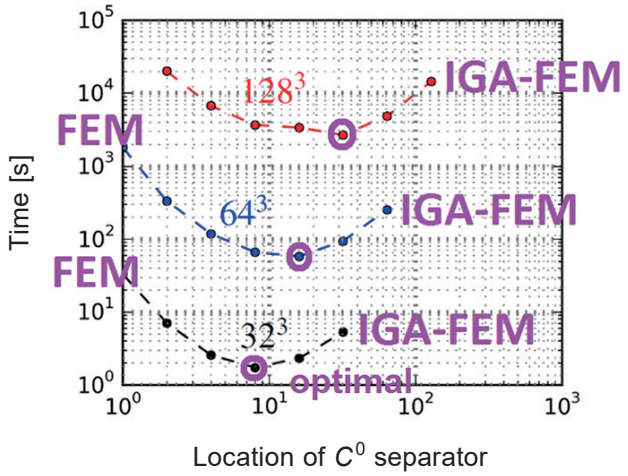

Fig. 11. The speedup of the multi-frontal solver executed on the 3D mesh with cubic B-splines with C^0 separators introduced

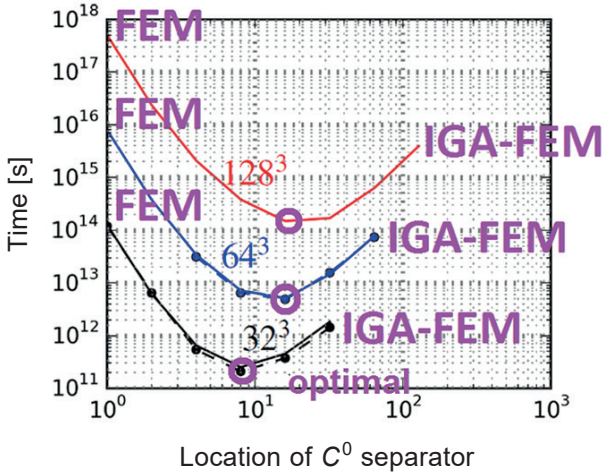

Fig. 12. The speedup of the multi-frontal solver executed on the 3D mesh with quintic B-splines with C^0 separators introduced

z axis (in 3D). This can be done in a linear cost, browsing all the mesh nodes. The local high jumps in the sum denotes the locations of the separators.

We have executed the sequential state-of-the-art MUMPS solver with METIS on the grids with estimated location of the separators. We have compared the resulting execution times for different location of the separators, as well as for the two border cases, the one corresponding to the FEM with Lagrange polynomials (which corresponds to separators between all elements, and to the IGE-FEM with tensor product B-spline polynomials (which corresponds to no separators). This is illustrated in Figs 9–12, where we have used the summation algorithm to find the optimal location of separators. Notice that for 3D quintic B-splines for 128^3 mesh it is not possible to run sequential simulation since MUMPS solver runs out of memory, even for 128 GB of available RAM. Thus, the red curve in Fig. 12 is our graphical estimate only. However, the flops estimator finds the optimal location of the separator, since the estimation does not require large amount of memory (it does not produce the fill-in, new non-zero entries during the factorization, which takes a lot of memory).

8. Conclusions

In this paper, we presented a heuristic algorithm for selection of the location of C^0 separators between patches of elements for IGA-FEM simulations on uniform 2D or 3D patches of elements. The C^0 separators allow reducing the computational cost of the multi-frontal direct solvers executed on the patches of elements, up to two orders of magnitude. Such the patches of elements are natural for IGA-FEM, where the geometrical objects are partition into parts, with each part mapped into the master patch of elements, with the geometry prescribed in terms of B-splines [14] or NURBS [30]. Thus, our method can be directly incorporated into CAD packages executing IGA-FEM simulations with multi-frontal direct solvers. The multi-frontal solvers are still used for badly conditioned problems, or as the

preconditioners for iterative solvers, as well as the kernels for multi-grid solvers. The weights introduced in the algorithm enable the usage of different orders of B-splines in different directions over the patches of elements. The computational complexity of the flops estimating algorithm is $\mathcal{O}(NlogN)$. The complexity of this algorithm is similar to the complexity of the algorithm generating the ordering for the multi-frontal solver, and it is much lower than the computational complexity of factorization itself, which is $\mathcal{O}(N^{1.5})$ in 2D and $\mathcal{O}(N^2)$ in 3D. The introduction of the C^0 separators reduces the constant in front of the complexity formula. Our future work will involve the extension of the algorithm to the parallel multi-frontal direct solvers, or for the correction of the computational grids where some C^0 separators have been already introduced.

Acknowledgements. The work was supported by the National Science Centre, Poland, grant no. DEC-2015/17/B/ST6/01867.

References

[1] T.J.R. Hughes, *The Finite Element Method. Linear Statics and Dynamics Finite Element Method Analysis*, Dover (2000).

[2] O.C. Zienkiewicz, R.Taylor, and J.Z. Ziu, *The Finite Element Method: Its Basis and Fundamentals*, Elsevier, 7th edition (2013).

[3] L. Demkowicz, *Computing with hp adaptive finite element method. Part I*, CRC Press, Boca Raton, FL. (2006).

[4] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, and A. Zdunek, *Computing with hp adaptive finite element method. Part II*, CRC Press, Boca Raton, FL (2007).

[5] I.S. Duff, A.M. Erisman, and J.K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, Inc., New York, NY, (1986).

[6] I.S. Duff and J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear, *ACM Transations on Mathematical Software*, 9(3), 302–325 (1983).

[7] I.S. Duff and K. Reid, The multifrontal solution of unsymmetric sets of linear systems, *SIAM Journal of Scientific Statistical Computing*, 5, 633–641 (1984).

[8] M. Paszyński, *Fast solvers for mesh-based computations*, Taylor & Francis, CRC Press, (2016).

[9] P.R. Amestoy and I.S. Duff, Multifrontal parallel distributed symmetric and unsymmetric solvers, *Computer Methods in Applied Mechanics and Engineering*, 184 501–520 (2000).

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

[11] P.R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, Hybrid scheduling for the parallel solution of linear systems, *Computer Methods in Applied Mechanics and Engineering*, 2(32), 136–156 (2001).

[12] A. George and J.W.-H. Lu, "An automatic nested dissection algorithm for irregular finite element problems", *SIAM Journal of Numerical Analysis* 15, 1053–1069 (1978).

[13] H. AbouEisha, V.M. Calo, K. Jopek, M. Moshkov, A. Paszńska, M. Paszyński, and M. Skotniczny, Element partition trees for hrefined meshes to optimize direct solver performance. P. 1, Dynamic programming, *International Journal of Applied Mathematics and Computer Science*, 27(2) (2017) 351–365.

[14] J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs, *Isogeometric Analysis: Toward Unification of CAD and FEA* JohnWiley and Sons, (2009).

[15] L. Dedè, T.J.R. Hughes, S. Lipton, V.M. and Calo, *Structural topology optimization with isogeometric analysis in a phase field approach*, USNCTAM2010, 16th US National Congree of Theoretical and Applied Mechanics.

[16] L. Dedè, M. J. Borden, and T.J.R. Hughes, *Isogeometric analysis for topology optimization with a phase field model*, ICES REPORT 11–29, The Institute for Computational Engineering and Sciences, The University of Texas at Austin (2011).

[17] R. Duddu, L. Lavier, T.J.R. Hughes, and V.M. Calo, A finite strain Eulerian formulation for compressible and nearly incompressible hyper-elasticity using high-order NURBS elements, *International Journal of Numerical Methods in Engineering*, 89(6) (2012) 762–785.

[18] M. Łoś, M. Paszyński, A. Kłusek, and W. Dzwinel, Application of fast isogeometric L2 projection solver for tumor growth simulations, *Computer Methods in Applied Mechanics and Engineering*, 316 (2017) 1257–1269.

[19] H. Gómez, V.M. Calo, Y. Bazilevs, and T.J.R. Hughes, Isogeometric analysis of the Cahn-Hilliard phase-field model, *Computer Methods in Applied Mechanics and Engineering* 197 (2008) 4333–4352.

[20] H. Gómez, T.J.R. Hughes, X. Nogueira, and V.M. Calo, Isogeometric analysis of the isothermal Navier-Stokes-Korteweg equations. *Computer Methods in Applied Mechanics and Engineering* 199 (2010) 1828–1840.

[21] S. Hossain, S.F.A. Hossainy, Y. Bazilevs, V.M. Calo, and T.J.R. Hughes, Mathematical modeling of coupled drug and drug-encapsulated nanoparticle transport in patientspecific coronary artery walls, *Computational Mechanics*, doi: 10.1007/s00466–011–0633–2, (2011).

[22] M.-C. Hsu, I. Akkerman, and Y. Bazilevs, High-performance computing of wind turbine aerodynamics using isogeometric analysis, *Computers and Fluids*, 49(1) (2011) 93–100.

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

[24] Y. Bazilevs, V.M. Calo, J.A. Cottrell, T.J.R. Hughes, A. Reali, and G. Scovazzi, Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows, *Computer Methods in Applied Mechanics and Engineering* 197 (2007) 173–201.

[25] Y. Bazilevs, V.M. Calo, Y. Zhang, and T.J.R. Hughes: Isogeometric fluid-structure interaction analysis with applications to arterial blood flow, *Computational Mechanics* 38 (2006).

[26] K. Chang, T.J.R. Hughes, and V.M. Calo, Isogeometric variational multiscale large-eddy simulation of fully-developed turbulent flow over a wavy wall, *Computers and Fluids*, 68 (2012) 94–104.

[27] D. Garcia, D. Pardo, L. Dalcin, M. Paszyński, N. Collier, and V.M. Calo, The value of continuity: Refined isogeometric analysis and fast direct solvers, *Computer Methods in Applied Mechanics and Engineering*, 316 (2017) 586–605.

[28] B. Janota and M. Paszyński, Algorithms for construction of Element Partition Trees for Direct Solver executed over h refined grids with B-splines and C^0 separators, *Procedia Computer Science*, 108 (2017) 857–866.

[29] B. Janota and A. Paszyńska, Automatic algorithms for the construction of element partition trees for isogeometric finite element method, accepted to *International Conference on Man-Machine Interactions (ICMMI)* (2017), Kraków, Poland.

[30] L. Piegl, and W. Tiller, *The NURBS Book* (Second Edition), Springer-Verlag New York, Inc., (1997).
[31] S. Fiałko, "A block sparse shared-memory multifrontal finite element solver for problems of structural mechanics", *Computer Assisted Mechanics and Engineering Science* 16, 117– 131 (2009).
[32] S. Fiałko, "The block subtracture multifrontal method for solution of large finite element equation sets", *Technical Transactions*, 1-NP, 8, 175–188 (2009).
[33] S. Fiałko, "PARFES: A method for solving finite element linear equations on multi-core computers", *Advanced Engineering Software* 40(12), 1256–1265 (2010).

9. Appendix: Implementation of the flops estimator

In this section, we describe the software package for estimation of the number of floating-point operations on two- and three-dimensional patches of elements utilized in isogeometric finite element method computations. The code can be obtained from *http://www.ki.agh.edu.pl/FastSolvers/rIGA/rIGA_clean.tar.gz*} The code is also located at *https://bitbucket.org/agha2s/riga/*.

The software package has the following components:

- Main component managing the execution,
- Generator of the element partition tree,
- FLOPS estimator,
- Graphics processor.

The dependency between modules is presented in Fig. 13. The data files generated by the modules are the following:

- *files/tree* – the element partition tree generated by the first module, and read by the second module,
- *files/analysed* – map of flops per node generated by second module,
- *files/pv_step_1.vti* – input for ParaView with plot of the map generated by the second module.
- Graphics processor

The code has the following source files:

- *main.f* – main routine controlling the execution and calling other modules. The required input is *files/knots*;
- *partition_engine.f* – module generating the element partition tree. The input for the module is the patch of elements created in main routine. The output from the module is the element partition tree *files/tree*;
- *flops_estimator_f/*∗*.f95* – fortran module performing the flops estimation. The input for the module is is the element partition tree *files/tree*. The output is the map of flops per dof *files/analysed*;
- *graphics_engine.f* – module generating the ParaView input file. The input is the map of flops per dof *files/analysed*. The output is the paraview file *files/pv_step_1.vti*;
- other routines ∗*.f* – are auxiliary and are necessary for the compilation;
- *makefile* and *flops_estimator_f/Makefile* – makefiles for the compilation.

The structure of the element partition tree is stored in *files/tree* file. The file contains the following data:

- map of dofs per mesh node. In the case of IGA-FEM we assign one node per one B-spline basis function,
- list of nodes per finite element,
- tree of recursive partitions of the finite element mesh.

The example of the element partition tree for the mesh described in Fig. 6 is provided below:

```
16 // number of nodes
 1 1 // 1 dof per B-spline 1
 2 1 // 1 dof per B-spline 2
 3 1 // 1 dof per B-spline 3
 4 1 // 1 dof per B-spline 4
 5 1 // 1 dof per B-spline 5
 6 1 // 1 dof per B-spline 6
 7 1 // 1 dof per B-spline 7
 8 1 // ....
 9 1
10 1
11 1
12 1
13 1
14 1
15 1
16 1
4 // number of elements
// element number 1, refinement level 1,
// nodes: 1,2,3, 5,6,7,9,10,11
// element 1 on this level, 9
1 1 1 9 1 2 3 5 6 7 9 10 11
// element number 2, refinement level 1,
// element 2 on this level,
// 9 nodes: 2,3,4,6,7,8,10,11,12
2 1 2 9 2 3 4 6 7 8 10 11 12
// element number 3, refinement level 1,
```

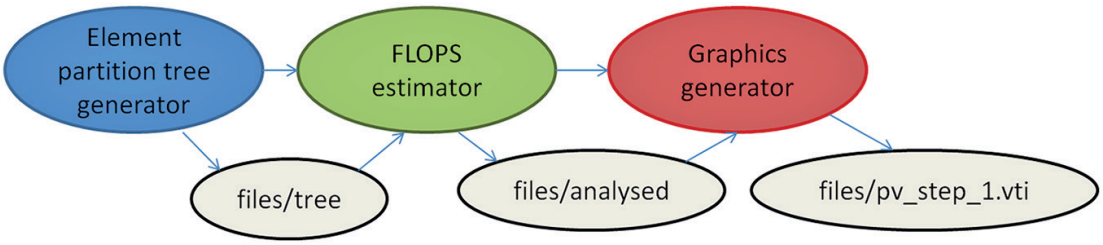

Fig. 13. Dependency between modules and data files generated

```
//element 3 on this level,
// 4 nodes: 5,6,7, 9,10,11, 13,14,15
3 1 3 9 5 6 7 9 10 11 13 14 15
// element number 4, refinement level 1,
// element 4 on this level,
// 9 nodes: 6,7,8, 10,11,12, 14,15,16
4 1 4 9 6 7 8 10 11 12 14 15 16
7 // number of nodes in element partition tree
// tree node id = 1,
// 4 elements: (1, 1) (1, 2) (1, 3) (1, 4)
// pointers to \revision{children} nodes 2, 3
1 4 1 1 1 2 1 3 1 4 2 3
// tree node id = 2, 2 elements: (1, 1) (1, 2),
2 2 1 1 1 2 4 5
// pointers to \revision{children} nodes 4, 5
// tree node id = 3, 2 elements: (1, 3) (1, 4),
// pointers to \revision{children} nodes 6, 7
3 2 1 3 1 4 6 7
// tree node id = 4, 1 element: (1, 1) leaf
4 1 1 1
// tree node id = 4, 1 element (1, 2) leaf
5 1 1 2
// tree node id = 4, 1 element (1, 3) leaf
6 1 1 3
// tree node id = 4, 1 element (1, 4) leaf
7 1 1 4
```

The code does not need any libraries to run. The compilation is straightforward. It requires to invoke make in main directory and in *flops_estimator_f* directory. The output files are stored in *files* directory that needs to be created.

The mesh generator is incorporated into the software. Let us refer to the patch of two times two finite elements with basis functions obtained from tensor products of one dimensional B-splines defined by the knot vectors $\{0,0,0,1,2,2,2\}\times$ $\times\{0,0,0,1,2,2,2\}$, presented in Fig. 6. The knot vectors are defined in the *main.f* routine by preparing knot vectors. To define the 2D mesh like in our example we need to proceed as follows: To define the 3D mesh, we just provide $nz > 1$.

```
c INPUT PARAMETERS
      px = 2
      nx = 3
      py = 2
      ny = 3
      pz = 2
      nz = 1
c ...
c ...prepare the problem dimensions
      allocate(Ux(nx + px + 2)) !knot vector
      allocate(Uy(ny + py + 2)) !knot vector
      allocate(Uz(nz + pz + 2)) !knot vector
c fill the knot vector for x
      Ux(1:px + 1) = 0.d0
      Ux(nx + 2:nx + px + 2) = 1.d0
      do i = px + 2,nx + 1
        Ux(i) = real(i-px-1)/real(nx-px + 1)
      enddo
      nelemx = CountSpans(nx,px,Ux)
c same code for ny,py,py
c ...\\
c same code for nz,pz,pz
c ...
```

However, our code can be incorporated into any software application working with IGA-FEM, using the knots vector standard.