Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 285 (2015) 817–828
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization

P. Antolin[a], A. Buffa[b], F. Calabrò[c], M. Martinelli[b], G. Sangalli[d,b,*]

[a] *Dipartimento di Ingegneria Civile ed Architettura, Università degli Studi di Pavia, Italy*
[b] *Istituto di Matematica Applicata e Tecnologie Informatiche "E. Magenes" del CNR, Pavia, Italy*
[c] *DIEI, Università degli Studi di Cassino e del Lazio Meridionale, Italy*
[d] *Dipartimento di Matematica, Università degli Studi di Pavia, Italy*

Received 1 July 2014; received in revised form 2 December 2014; accepted 4 December 2014
Available online 15 December 2014

Highlights

- We discuss the use of the sum-factorization for the calculation of the integrals arising in Galerkin isogeometric analysis.
- We give an estimate of the quadrature computational cost and compare with the standard approach.
- We perform numerical tests.
- Sum-factorization significantly reduces the quadrature computational cost.

Abstract

In this paper we discuss the use of the sum-factorization for the calculation of the integrals arising in Galerkin isogeometric analysis. While introducing very little change in an isogeometric code based on element-by-element quadrature and assembling, the sum-factorization approach, taking advantage of the tensor-product structure of splines or NURBS shape functions, significantly reduces the quadrature computational cost.

Keywords: Numerical integration; Isogeometric analysis; Splines; NURBS; Sum-factorization

1. Introduction

Isogeometric analysis (IGA) is a computational technique for the solution of boundary value problems. It is recent and at the same time well known in the computational engineering academic community, as an extension of the classical Finite Element Method (FEM). IGA was proposed in the seminal paper [1], based on the idea of using the functions adopted in Computer Aided Design (CAD), that is, splines and Non-Uniform Rational B-Splines (NURBS),

* Corresponding author at: Dipartimento di Matematica, Università degli Studi di Pavia, Italy.
E-mail addresses: pablo.antolinsanchez@unipv.it (P. Antolin), annalisa@imati.cnr.it (A. Buffa), calabro@unicas.it (F. Calabrò), martinelli@imati.cnr.it (M. Martinelli), giancarlo.sangalli@unipv.it (G. Sangalli).

http://dx.doi.org/10.1016/j.cma.2014.12.013
0045-7825/© 2014 Elsevier B.V. All rights reserved.

not only to describe the domain geometry, but also to represent the numerical solution of the problem, in the isoparametric framework. For the interested reader, we refer to the book on IGA [2]. A recent overview on the mathematical aspects of IGA is [3], that covers the known mathematical theory of IGA but also contains an updated bibliography with references to the major contributions and applications of IGA in various engineering fields.

One of the interesting features of IGA, compared to high order FEM, is that it allows for higher global regularity of the shape functions, up to C^{p-1} inter-element continuity for p-degree splines and NURBS. This leads to a higher accuracy per degree-of-freedom (see [4,5]), improved spectrum properties of the discrete operators (see [6]), and the possibility of constructing smooth discretizations of the fundamental structures of the differential operators (such as De Rham diagrams, see e.g. [7,8]).

IGA can be implemented re-using the existing finite element technology. This may be not the most efficient way to use IGA but surely is one key reason of its fast diffusion and the easiest way to apply IGA on complex problems. In particular, the construction of the matrix of the linear system arising in a Galerkin isogeometric method is typically made by the element-by-element quadrature and assembling as in FEMs. However for high regular and high degree ($p \geq 3$) splines and NURBS, it is experienced that most of the CPU time goes in the quadrature and assembling itself. This may be understood comparing IGA with C^0 to C^{p-1} p-degree splines (or, equivalently, FEM and typical IGA) on the same mesh: element-wise quadrature has the same computational cost in the two cases, even though the C^{p-1} case results in a much smaller linear system. The high cost of quadrature has motivated the research on quadrature rules that keep into account the interelement regularity of IGA functions, see [9–11], improving efficiency w.r.t. Gauss quadrature. In this paper we consider another significant improvement: one can exploit the tensor-product structure of multivariate splines by adopting the so called *sum-factorization*, a well-known technique for spectral elements or some high-degree finite elements (see e.g. [12–14]), but never used with IGA, at our knowledge.

The aim of this paper is to discuss and benchmark the use of sum-factorization in IGA. We show that there is a clear advantage versus the standard quadrature approach, and show that the cost of quadrature (by sum-factorization) is balanced with the cost of the linear solver for high degree IGA. We also discuss the implementation of the proposed sum-factorization within our isogeometric library igatools [15]. We do not consider parallel implementation though this is clearly a key ingredient for a modern and efficient isogeometric code (see for example [16]).

There are other possibilities to circumvent the element-by-element quadrature issue that however require a change of paradigm. For example, if the mesh is uniform, one can efficiently and directly compute the entries of the linear system matrix (see [17–20]) or switch from Galerkin to a collocation formulation [21,22].

The outline of the paper is as follows. In Section 2 we set up the notation and briefly describe the setting of an academic problem. Section 3 introduces the sum-factorization algorithm and discuss its computational cost in terms of the degree p. Section 4 is devoted to the numerical testing and comparison with other strategies. Finally, we draw conclusions in Section 5. An Appendix is included in order to describe the treatment of the linear elasticity stiffness matrix.

2. Preliminaries

We consider the elliptic problem

$$\begin{cases} -\mu \nabla^2 u + \sigma u = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \tag{2.1}$$

as a model problem. Its Galerkin approximation on a discrete space V requires the computation of the following matrix entries:

- the mass matrix (or mass integrals)

$$M_{i,j} = \int_\Omega \sigma(\boldsymbol{x}) R_{\boldsymbol{i}}(\boldsymbol{x}) R_{\boldsymbol{j}}(\boldsymbol{x})\, d\boldsymbol{x}; \tag{2.2}$$

- the stiffness matrix (or stiffness integrals)

$$S_{i,j} = \int_\Omega \mu(\boldsymbol{x}) \nabla R_{\boldsymbol{i}}(\boldsymbol{x}) \cdot \nabla R_{\boldsymbol{j}}(\boldsymbol{x})\, d\boldsymbol{x}; \tag{2.3}$$

where $R_{\boldsymbol{i}}$ and $R_{\boldsymbol{j}}$ denote two basis function in V, $\mu : \Omega \to \mathbb{R}$ and $\sigma : \Omega \to \mathbb{R}$.

In IGA, Ω is given by a spline or NURBS parametrization. For the sake of simplicity, we assume Ω is given by a d-dimensional single patch spline representation:

$$\Omega = \boldsymbol{F}([0,1]^d), \quad \text{with } \boldsymbol{F}(\hat{\boldsymbol{x}}) = \sum_{\boldsymbol{i}} \boldsymbol{C_i} \hat{B}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}),$$

where $\boldsymbol{C_i}$ are the control points and $\hat{B}_{\boldsymbol{i}}$ are p-degree tensor-product B-spline basis functions defined on the parametric patch $[0,1]^d$.

Being IGA based on the isoparametric paradigm, the basis functions $R_{\boldsymbol{i}}$ are defined as $R_{\boldsymbol{i}} = \hat{B}_{\boldsymbol{i}} \circ \boldsymbol{F}^{-1}$. The integrals above are then computed by change of variable and element-by-element assembling. For more details, see [2].

Then, we consider the computation of the local matrices related to a parametric element $\hat{\Omega} \subset [0,1]^d$, $\hat{\Omega} := [a_1, b_1] \times \cdots \times [a_d, b_d]$. With an abuse of notation, we denote by $\hat{B}_{\boldsymbol{\alpha}}$ the B-splines that are non-zero on $\hat{\Omega}$ and index them by the local multi-index $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$, $\alpha_j \in \{1, \ldots, p+1\}$ $\forall j = 1, \ldots, d$, and set $\hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_d)$. The total number of local basis functions is $(p+1)^d$, i.e. $\#\{\boldsymbol{\alpha}\} = (p+1)^d$.

The local mass matrix, that corresponds to the global one (2.2) after change of variable $\hat{\boldsymbol{x}} = \boldsymbol{F}^{-1}(\boldsymbol{x})$, is $\mathbb{M} = \{m_{\boldsymbol{\alpha},\boldsymbol{\beta}}\} \in \mathbb{R}^{(p+1)^d \times (p+1)^d}$ where:

$$m_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \int_{\hat{\Omega}} \hat{B}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{x}})\, \hat{B}_{\boldsymbol{\beta}}(\hat{\boldsymbol{x}})\, c(\hat{\boldsymbol{x}})\, d\hat{\boldsymbol{x}}, \tag{2.4}$$

with $c(\hat{\boldsymbol{x}}) = \hat{\sigma}(\hat{\boldsymbol{x}}) |\hat{\boldsymbol{D}}\boldsymbol{F}(\hat{\boldsymbol{x}})|$.

Moreover, for the local stiffness matrix $\mathbb{S} = \{s_{\boldsymbol{\alpha},\boldsymbol{\beta}}\} \in \mathbb{R}^{(p+1)^d \times (p+1)^d}$ we have:

$$s_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \int_{\hat{\Omega}} \left(\hat{\boldsymbol{D}}\boldsymbol{F}^{-T} \hat{\nabla} \hat{B}_{\boldsymbol{\alpha}}\right)^T \left(\hat{\boldsymbol{D}}\boldsymbol{F}^{-T} \hat{\nabla} \hat{B}_{\boldsymbol{\beta}}\right) \hat{\mu}\, |\hat{\boldsymbol{D}}\boldsymbol{F}|\, d\hat{\boldsymbol{x}} = \int_{\hat{\Omega}} \hat{\nabla} \hat{B}_{\boldsymbol{\alpha}}^T \left(\left[\hat{\boldsymbol{D}}\boldsymbol{F}^{-1} \hat{\boldsymbol{D}}\boldsymbol{F}^{-T}\right] \hat{\mu}\, |\hat{\boldsymbol{D}}\boldsymbol{F}| \right) \hat{\nabla} \hat{B}_{\boldsymbol{\beta}}\, d\hat{\boldsymbol{x}}.$$

Also in this case we change notations and we write:

$$s_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \sum_{i,j=1}^{d} \int_{\hat{\Omega}} \frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_i}(\hat{\boldsymbol{x}}) c_{i,j}(\hat{\boldsymbol{x}}) \frac{\partial \hat{B}_{\boldsymbol{\beta}}}{\partial \hat{x}_j}(\hat{\boldsymbol{x}})\, d\hat{\boldsymbol{x}}, \tag{2.5}$$

so that $C(\hat{\boldsymbol{x}}) := \left\{c_{i,j}(\hat{\boldsymbol{x}})\right\}_{i,j=1,\ldots,d}$ is the matrix:

$$C(\hat{\boldsymbol{x}}) = \left[\hat{\boldsymbol{D}}\boldsymbol{F}^{-1}(\hat{\boldsymbol{x}}) \hat{\boldsymbol{D}}\boldsymbol{F}^{-T}(\hat{\boldsymbol{x}})\right] \hat{\mu}(\hat{\boldsymbol{x}})\, |\hat{\boldsymbol{D}}\boldsymbol{F}(\hat{\boldsymbol{x}})|.$$

Finally, we express the tensor-product structure of the B-splines basis functions writing:

$$\hat{B}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{x}}) = \hat{B}_{\alpha_1,\ldots,\alpha_d}(\hat{x}_1, \ldots, \hat{x}_d) = \prod_{k=1}^{d} \hat{B}_{\alpha_k}(\hat{x}_k). \tag{2.6}$$

Remark 2.1. If NURBS are used instead of splines, the situation is similar. Writing NURBS in terms of B-splines then sum-factorization can be used to integrate the B-splines that form the basis. For the mass matrix, the use of NURBS only affects the coefficient $c(\hat{\boldsymbol{x}})$, that takes into account the NURBS weight function, and the structure is still the one in (2.4). For the stiffness matrix instead, the derivation rule will produce a few terms, similar to (2.4) or (2.5).

3. Computation of the local matrices by quadrature

The elements of the local matrices are multidimensional integrals in a d-rectangular domain so that in order to calculate the integrals we need a multidimensional quadrature rule. For the sake of simplicity we consider a quadrature formula constructed via the same univariate quadrature rule in each direction (properly scaled) but everything can be straightforwardly replaced by any tensor-product quadrature scheme. With this aim we introduce a r-points univariate quadrature rule in the reference domain $[0,1]$: we will denote by ζ_γ, $\gamma = 1, \ldots, r$ the quadrature nodes and with ω_γ the relative weights. This formula, then, will be rescaled to be used in the domains $[a_i, b_i]$: $\zeta_\gamma^{(i)} = (b_i - a_i)\zeta_\gamma + a_i$, $\omega_\gamma^{(i)} = (b_i - a_i)\omega_\gamma$. In the d-dimensional domain $\hat{\Omega}$ we will have the set of r^d quadrature points $\boldsymbol{\zeta_\gamma} = (\zeta_{\gamma_1}^{(1)}, \ldots, \zeta_{\gamma_d}^{(d)})$ and relative weights $\omega_{\boldsymbol{\gamma}} = \prod_{i=1}^{d} \omega_{\gamma_i}^{(i)}$.

In order to recover the expected convergence rates we take a quadrature rule of degree of exactness $2p+1$, see [23, Theorem 4.1.6]. In the following we will use r-points Gaussian quadrature. In this case the usual choice is $r = p+1$, which ensures exact integration of (2.4)–(2.5) when the coefficients $c(\hat{\boldsymbol{x}})$ and $\{c_{i,j}(\hat{\boldsymbol{x}})\}_{i,j=1,\dots,d}$ are element-wise constant. For non-constant but regular (element-by-element) coefficients, the choice $r = p+1$ is proved to guarantee optimality of the Galerkin formulation with numerical quadrature (see, e.g., [23, Section 4.1]).

The entries of the local mass and stiffness matrices are $(p+1)^{2d}$ thus if the quadrature procedure is applied directly one obtains that the overall cost for the local matrix construction results in $r^d(p+1)^{2d}$. Because $r = O(p)$, one expects to have an overall cost of $O(p^{3d})$ for the construction of each local matrix, as usually reported. Aim of this paper is to reduce this cost to $O(p^{2d+1})$ by exploiting the tensor product structure via the so-called sum factorization approach [24]. This technique is widely employed in spectral methods, and has also been applied also for high order FEMs [12,13]. We will discuss in this paper its' application to the case of IGA, maintaining the element-by-element assembly procedure.

In order to clarify the use of the sum factorization algorithm in our framework, we will now present a general application in the 3D case for the calculation of the integral $\int_{\hat{\Omega}} \hat{B}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{x}})\hat{B}_{\boldsymbol{\beta}}(\hat{\boldsymbol{x}})c(\hat{\boldsymbol{x}})\,d\hat{\boldsymbol{x}}$. Simple application of the quadrature rule leads to:

$$\int_{\hat{\Omega}} \hat{B}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{x}})\hat{B}_{\boldsymbol{\beta}}(\hat{\boldsymbol{x}})c(\hat{\boldsymbol{x}})\,d\hat{\boldsymbol{x}} \approx \sum_{\boldsymbol{\gamma}\in\{1,\dots,r\}^3} \omega_{\boldsymbol{\gamma}} \hat{B}_{\boldsymbol{\alpha}}(\boldsymbol{\zeta}_{\boldsymbol{\gamma}})\hat{B}_{\boldsymbol{\beta}}(\boldsymbol{\zeta}_{\boldsymbol{\gamma}})c(\boldsymbol{\zeta}_{\boldsymbol{\gamma}}) = \mathbb{I}_M(\boldsymbol{\alpha},\boldsymbol{\beta}). \tag{3.1}$$

In order to exploit the tensor product structure we start from (2.6) and notice that:

$$\begin{aligned}\int_{\hat{\Omega}} \hat{B}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{x}})\hat{B}_{\boldsymbol{\beta}}(\hat{\boldsymbol{x}})c(\hat{\boldsymbol{x}})\,d\hat{\boldsymbol{x}} &= \int_{a_1}^{b_1} \hat{B}_{\alpha_1}(\hat{x}_1)\hat{B}_{\beta_1}(\hat{x}_1)\left[\int_{a_2}^{b_2} \hat{B}_{\alpha_2}(\hat{x}_2)\hat{B}_{\beta_2}(\hat{x}_2)\right.\\ &\quad\left.\times\left[\int_{a_3}^{b_3} \hat{B}_{\alpha_3}(\hat{x}_3)\hat{B}_{\beta_3}(\hat{x}_3)c(\hat{x}_1,\hat{x}_2,\hat{x}_3)\,d\hat{x}_3\right]d\hat{x}_2\right]d\hat{x}_1.\end{aligned}$$

Then, if we apply the univariate quadrature rules in this nested integrals we obtain:

$$\begin{aligned}\mathbb{I}_M(\boldsymbol{\alpha},\boldsymbol{\beta}) &= \sum_{\gamma_1=1}^{r} \omega_{\gamma_1}^{(1)} \hat{B}_{\alpha_1}(\zeta_{\gamma_1}^{(1)})\hat{B}_{\beta_1}(\zeta_{\gamma_1}^{(1)})\left[\sum_{\gamma_2=1}^{r} \omega_{\gamma_2}^{(2)} \hat{B}_{\alpha_2}(\zeta_{\gamma_2}^{(2)})\hat{B}_{\beta_2}(\zeta_{\gamma_2}^{(2)})\right.\\ &\quad\left.\times\left[\sum_{\gamma_3=1}^{r} \omega_{\gamma_3}^{(3)} \hat{B}_{\alpha_3}(\zeta_{\gamma_3}^{(3)})\hat{B}_{\beta_3}(\zeta_{\gamma_3}^{(3)})c(\zeta_{\gamma_1}^{(1)},\zeta_{\gamma_2}^{(2)},\zeta_{\gamma_3}^{(3)})\right]\right].\end{aligned}$$

To calculate this sums $\forall\boldsymbol{\alpha},\boldsymbol{\beta}$, we introduce the following recursion:

(i) $\mathbb{C}^{(0)}(\zeta_{\gamma_1}^{(1)},\zeta_{\gamma_2}^{(2)},\zeta_{\gamma_3}^{(3)}) = c(\zeta_{\gamma_1}^{(1)},\zeta_{\gamma_2}^{(2)},\zeta_{\gamma_3}^{(3)})$

(ii) $\mathbb{C}^{(1)}(\zeta_{\gamma_1}^{(1)},\zeta_{\gamma_2}^{(2)};\alpha_3,\beta_3) = \sum_{\gamma=1}^{r} \omega_{\gamma}^{(3)} \hat{B}_{\alpha_3}(\zeta_{\gamma}^{(3)})\hat{B}_{\beta_3}(\zeta_{\gamma}^{(3)})\mathbb{C}^{(0)}(\zeta_{\gamma_1}^{(1)},\zeta_{\gamma_2}^{(2)},\zeta_{\gamma}^{(3)})$

(iii) $\mathbb{C}^{(2)}(\zeta_{\gamma_1}^{(1)};\alpha_2,\beta_2;\alpha_3,\beta_3) = \sum_{\gamma=1}^{r} \omega_{\gamma}^{(2)} \hat{B}_{\alpha_2}(\zeta_{\gamma}^{(2)})\hat{B}_{\beta_2}(\zeta_{\gamma}^{(2)})\mathbb{C}^{(1)}(\zeta_{\gamma_1}^{(1)},\zeta_{\gamma}^{(2)};\alpha_3,\beta_3)$

(iv) $\mathbb{I}_M(\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathbb{I}_M(\alpha_1,\beta_1;\alpha_2,\beta_2;\alpha_3,\beta_3) = \sum_{\gamma=1}^{r} \omega_{\gamma}^{(1)} \hat{B}_{\alpha_1}(\zeta_{\gamma}^{(1)})\hat{B}_{\beta_1}(\zeta_{\gamma}^{(1)})\mathbb{C}^{(2)}(\zeta_{\gamma}^{(1)};\alpha_2,\beta_2;\alpha_3,\beta_3)$.

Then we can count the operations:

1. Step (i) costs one function evaluation in r^3 quadrature points; these r^3 values have to be stored.
2. Step (ii) costs $3r$ multiplications and $(r-1)$ additions in r^2 quadrature points and for $(p+1)^2$ coefficients α_3,β_3; $r^2\times(p+1)^2$ values have to be stored.
3. Step (iii) costs $3r$ multiplications and $(r-1)$ additions in r quadrature points and for $(p+1)^4$ coefficients $\alpha_3,\beta_3,\alpha_2,\beta_2$; $r\times(p+1)^4$ values have to be stored.

4. Step (iv) costs $3r$ multiplications and $(r-1)$ additions for $(p+1)^6$ coefficients $\boldsymbol{\alpha}, \boldsymbol{\beta}$; the final $(p+1)^6$ values are the requested approximations of the integrals (2.4).

The overall cost is then $O(rp^6)$. We notice also that the symmetries can be exploited in each direction. Indeed:

$$\begin{aligned}
&\mathbb{C}^{(1)}(\zeta^{(1)}_{\gamma_1}, \zeta^{(2)}_{\gamma_2}; \alpha_3, \beta_3) = \mathbb{C}^{(1)}(\zeta^{(1)}_{\gamma_1}, \zeta^{(2)}_{\gamma_2}; \beta_3, \alpha_3)\\
&\mathbb{C}^{(2)}(\zeta^{(1)}_{\gamma_1}; \alpha_2, \beta_2; \alpha_3, \beta_3) = \mathbb{C}^{(2)}(\zeta^{(1)}_{\gamma_1}; \alpha_2, \beta_2; \beta_3, \alpha_3) = \mathbb{C}^{(2)}(\zeta^{(1)}_{\gamma_1}; \beta_2, \alpha_2; \alpha_3, \beta_3)\\
&\mathbb{I}_M(\alpha_1, \beta_1; \alpha_2, \beta_2; \alpha_3, \beta_3) = \mathbb{I}_M(\alpha_1, \beta_1; \alpha_2, \beta_2; \beta_3, \alpha_3) = \mathbb{I}_M(\alpha_1, \beta_1; \beta_2, \alpha_2; \alpha_3, \beta_3)\\
&\qquad = \mathbb{I}_M(\beta_1, \alpha_1; \alpha_2, \beta_2; \alpha_3, \beta_3).
\end{aligned}$$

Remark 3.1. If, instead of Gaussian quadrature, one adopts the optimal quadrature proposed in [10,9], the overall computational cost can be reduced up to a factor 1/2.

The above procedure is used for the computation of the mass matrix with numerical quadrature:

$$m_{\boldsymbol{\alpha},\boldsymbol{\beta}} \approx \sum_{\boldsymbol{\gamma} \in \{1,\ldots,r\}^d} \omega_{\boldsymbol{\gamma}} c(\boldsymbol{\zeta}_{\boldsymbol{\gamma}}) \hat{B}_{\boldsymbol{\alpha}}(\boldsymbol{\zeta}_{\boldsymbol{\gamma}}) \hat{B}_{\boldsymbol{\beta}}(\boldsymbol{\zeta}_{\boldsymbol{\gamma}}). \tag{3.2}$$

With direct application of the procedure seen before we obtain the requested algorithm, reported in the following box.

```
Input: $p+1$ number of basis functions $\hat{B}$; $r =$ number of quadrature points;
Input: $\hat{\Omega} := [a_1, b_1] \times \ldots \times [a_d, b_d] =$ element in the reference domain ;
Input: $(\zeta_\gamma, \omega_\gamma)$ $r$-points quadrature in the domain [0, 1];
  {Computation of the function evaluations}
  Calculate quadrature nodes and weights along directions $i = 1, \ldots, d$:
      $\zeta^{(i)}_\gamma = (b_i - a_i)\zeta_\gamma + a_i ; \qquad \omega^{(i)}_\gamma = (b_i - a_i)\omega_\gamma .$
  Pre-calculate the basis functions at the quadrature nodes (for re-use purpose):
      $\hat{B}_\alpha(\zeta^{(i)}_\gamma) \qquad \forall i \in \{1, \ldots, d\}, \forall \gamma \in \{1, \ldots, r\}, \forall \alpha \in \{1, \ldots, p+1\}$
  Calculate the function $c(\hat{\boldsymbol{x}})$ on the nodes:
      $\mathbb{C}^{(0)}(\zeta^{(1)}_{\gamma_1}, \ldots \zeta^{(d)}_{\gamma_d}) = c(\zeta^{(1)}_{\gamma_1}, \ldots \zeta^{(d)}_{\gamma_d}), \ \mathbb{C}^{(0)} \in \mathbb{R}^{r^d}$ .
  {Computation of the mass terms via sum factorization}
  for $i = 1, \ldots, d$ do
    Calculate $\mathbb{C}^{(i)} \in \mathbb{R}^{r^{d-i} \times (p+1)^{2i}}$:
      $\mathbb{C}^{(i)}\left(\zeta^{(1)}_{\gamma_1}, \ldots, \zeta^{(d-i)}_{\gamma_{d-i}}; \alpha_{d-i+1}, \beta_{d-i+1}; \ldots; \alpha_d, \beta_d\right)$
      $= \sum_{\gamma=1}^{r}\Big[\omega^{(d-i+1)}_\gamma \hat{B}_{\alpha_{d-i+1}}(\zeta^{(d-i+1)}_\gamma)\hat{B}_{\beta_{d-i+1}}(\zeta^{(d-i+1)}_\gamma) \cdot$
          $\mathbb{C}^{(i-1)}\left(\zeta^{(1)}_{\gamma_1}, \ldots, \zeta^{(d-i)}_{\gamma_{d-i}}, \zeta^{(d-i+1)}_\gamma; \alpha_{d+i+2}, \beta_{d+i+2}; \ldots; \alpha_d, \beta_d\right)\Big]$
  end for[$i$]
  Output:
      $m_{\boldsymbol{\alpha},\boldsymbol{\beta}} \approx \mathbb{I}_M(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbb{C}^{(d)}(\alpha_1, \beta_1; \ldots; \alpha_d, \beta_d)$
```

For the calculation of the stiffness matrix, we need to apply the procedure to all the d^2 integrals appearing in Eq. (2.5). Exploiting the tensor product structure of the gradient we can write the following:

$$\frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_i} = \prod_{k=1}^{d} D^{\delta_{ik}} \hat{B}_{\alpha_k}(\hat{x}_k), \tag{3.3}$$

where we have denoted by:

$$D^{\delta_{ik}} f(\hat{x}) \equiv \begin{cases} f'(\hat{x}) & \text{if } i = k \\ f(\hat{x}) & \text{otherwise.} \end{cases}$$

Then the elements of the stiffness matrix can be written as:

$$s_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \sum_{i,j=1}^{d} \int_{\hat{\Omega}} \left[\prod_{k=1}^{d} D^{\delta_{ik}} \hat{B}_{\alpha_k}(\hat{x}_k) D^{\delta_{jk}} \hat{B}_{\beta_k}(\hat{x}_k) \right] c_{i,j}(\hat{\boldsymbol{x}}) d\hat{\boldsymbol{x}}. \tag{3.4}$$

The integrals above are then treated by Gaussian quadrature, for example when $d = 3$:

$$s_{\boldsymbol{\alpha},\boldsymbol{\beta}} \approx \sum_{i,j=1}^{3} \left\{ \sum_{\gamma_1=1}^{r} \omega_{\gamma_1}^{(1)} D^{\delta_{i1}} \hat{B}_{\alpha_1}(\zeta_{\gamma_1}^{(1)}) D^{\delta_{j1}} \hat{B}_{\beta_1}(\zeta_{\gamma_1}^{(1)}) \cdot \left[\sum_{\gamma_2=1}^{r} \omega_{\gamma_2}^{(2)} D^{\delta_{i2}} \hat{B}_{\alpha_2}(\zeta_{\gamma_2}^{(2)}) D^{\delta_{j2}} \hat{B}_{\beta_2}(\zeta_{\gamma_2}^{(2)}) \right.\right.$$
$$\left.\left. \cdot \left(\sum_{\gamma_3=1}^{r} \omega_{\gamma_3}^{(3)} D^{\delta_{i3}} \hat{B}_{\alpha_3}(\zeta_{\gamma_3}^{(3)}) D^{\delta_{j3}} \hat{B}_{\beta_3}(\zeta_{\gamma_3}^{(3)}) c(\zeta_{\gamma_1}^{(1)}, \zeta_{\gamma_2}^{(2)}, \zeta_{\gamma_3}^{(3)}) \right) \right] \right\}. = \mathbb{I}_S(\boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{3.5}$$

The sum factorization procedure to compute the quantity in Eq. (3.5) is summarized in the following box.

```
Input: $p + 1$ number of basis functions $\hat{B}$; $r$ = number of quadrature points;
Input: $\hat{\Omega} := [a_1, b_1] \times \ldots \times [a_d, b_d]$ = element in the reference domain ;
Input: $(\zeta_\gamma, \omega_\gamma)$ $r$-points quadrature in the domain [0, 1];
  {Computation of the function evaluations}
  Calculate quadrature nodes and weights along directions:
      $\zeta_\gamma^{(i)} = (b_i - a_i)\zeta_\gamma + a_i \; ; \; \omega_\gamma^{(i)} = (b_i - a_i)\omega_\gamma \; .$
  Pre-calculate the basis functions and derivatives at the quadrature nodes (for re-use purpose):
      $D^0 \hat{B}_\alpha(\zeta_\gamma^{(i)}) \equiv \hat{B}_\alpha(\zeta_\gamma^{(i)}) \quad , \quad D^1 \hat{B}_\alpha(\zeta_\gamma^{(i)}) \equiv \hat{B}'_\alpha(\zeta_\gamma^{(i)})$
          $\forall i \in \{1, \ldots, d\}, \forall \gamma \in \{1, \ldots, r\}, \forall \alpha \in \{1, \ldots, p + 1\}$
  Calculate the functions $\{c_{i,j}(\hat{\boldsymbol{x}})\}_{i,j=1,\ldots,d}$ on the nodes:
      $\mathbb{C}_{i,j}^{(0)}(\zeta_{\gamma_1}^{(1)}, \ldots \zeta_{\gamma_d}^{(d)}) = c_{i,j}(\zeta_{\gamma_1}^{(1)}, \ldots \zeta_{\gamma_d}^{(d)}) \, , \; \mathbb{C}_{i,j}^{(0)} \in \mathbb{R}^{r^d} \; .$
  {Computation of the stiffness terms via sum factorization}
  for $i, j = 1, \ldots, d$ do
    for $l = 1, \ldots, d$ do
      Calculate $\mathbb{C}_{i,j}^{(l)} \in \mathbb{R}^{r^{d-l} \times (p+1)^{2l}}$:
        $\mathbb{C}_{i,j}^{(l)} \left( \zeta_{\gamma_1}^{(1)}, \ldots, \zeta_{\gamma_{d-l}}^{(d-l)}; \alpha_{d-l+1}, \beta_{d-l+1}; \ldots; \alpha_d, \beta_d \right)$
          $= \sum_{\gamma=1}^{r} \Big[ \omega_\gamma^{(d-l+1)} D^{\delta(il)} \hat{B}_{\alpha_{d-l+1}}(\zeta_\gamma^{(d-l+1)}) D^{\delta(jl)} \hat{B}_{\beta_{d-l+1}}(\zeta_\gamma^{(d-l+1)}) \cdot$
          $\mathbb{C}_{i,j}^{(l-1)} \left( \zeta_{\gamma_1}^{(1)}, \ldots, \zeta_{\gamma_{d-l}}^{(d-l)}, \zeta_\gamma^{(d-l+1)}; \alpha_{d+l+2}, \beta_{d+l+2}; \ldots; \alpha_d, \beta_d \right) \Big]$
    end for[$l$]
  end for[$i, j$]
  Output:
      $s_{\boldsymbol{\alpha},\boldsymbol{\beta}} \approx \mathbb{I}_S(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i,j=1}^{d} \mathbb{C}_{i,j}^{(d)} (\alpha_1, \beta_1 \ldots \alpha_d, \beta_d)$
```

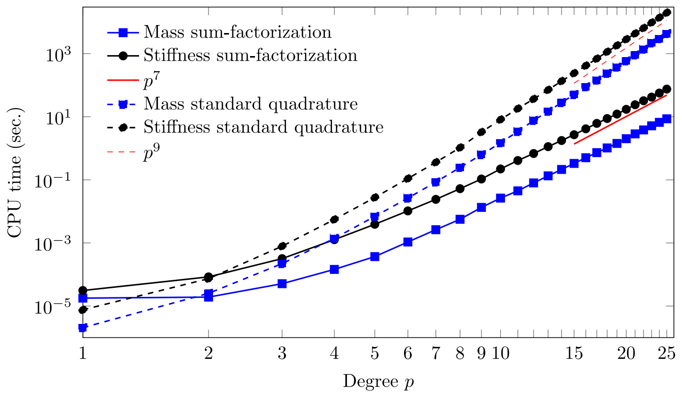

Fig. 1. CPU time vs. degree for calculating the local stiffness-matrix and mass-matrix on a single element using sum-factorization and standard quadrature.

4. Numerical results

In this section, in order to evaluate numerically the behaviour of the proposed procedure we present some tests that compare the standard quadrature procedure with the above described sum factorization.

In Section 4.1 we consider the construction of the mass and stiffness matrices in a single mesh element, comparing the CPU costs when the polynomial degree p increases.

In Section 4.2 we compute the solution of a linear elasticity problem on a mesh of $30 \times 15 \times 30$ elements, while in Section 4.3 we compute the solution of an advection–diffusion problem on a mesh of $60 \times 84 \times 12$ elements. For the elasticity problem and the advection–diffusion problem, we have compared the time needed for the computation of the stiffness matrix with the different quadrature strategies and also the time needed by different solver strategies to converge to the solution.

The results for all cases refers to a Linux workstation equipped with Xeon E5-2470 processors (running at 2.3 GHz), where we have used only one core for the simulations in order to avoid any time saving due to parallelization. The previous algorithms have been implemented in the igatools, for details see Section 4.4.

4.1. Computation of the local matrices

In Fig. 1 we plot the CPU time needed for the calculations on a single 3D element of the mass and stiffness matrices. This test confirms what predicted by the general results in Section 3: the time increases as p^{3d} if no attention is paid to the tensor product structure, while the sum factorization procedure behaves as p^{2d+1}.

Only for $p = 1$ the usual procedure over-performs the sum factorization: this is due to the fact that the complexity of the sum-factorization algorithm is not compensated by the saving due to the smaller number of operations w.r.t. the standard quadrature approach. For degree $p = 2$ the two approaches cost approximately the same amount of CPU time, while for degree $p \geq 3$ the sum-factorization is less expensive. At degree $p = 4$ one can see that the computational cost with the sum factorization procedure is one order of magnitude below the corresponding one obtained with the standard quadrature.

4.2. Linear elasticity

As a first example it is shown the use of sum factorization technique for a linear elasticity problem:

$$\begin{cases} \nabla \cdot \sigma + f = 0 & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega_D, \\ \sigma \cdot n = t & \text{on } \partial\Omega_N, \end{cases} \tag{4.1}$$

being σ and u the stress and deformation displacement fields. For the linear elasticity case the Cauchy stress tensor σ can be expressed as

$$\sigma(u) = \mathcal{C} : \varepsilon(u), \tag{4.2}$$

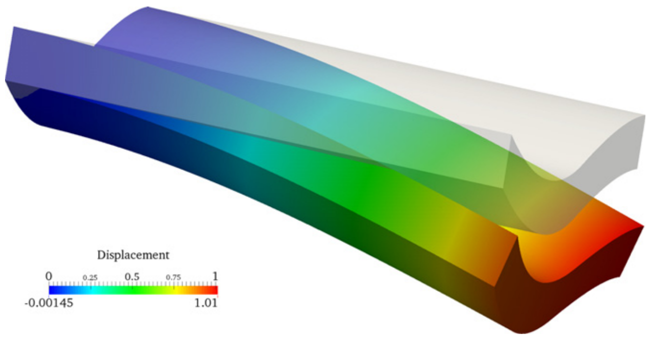

Fig. 2. Geometry (before and after deformation) and displacements solution for the linear elasticity problem (4.1).

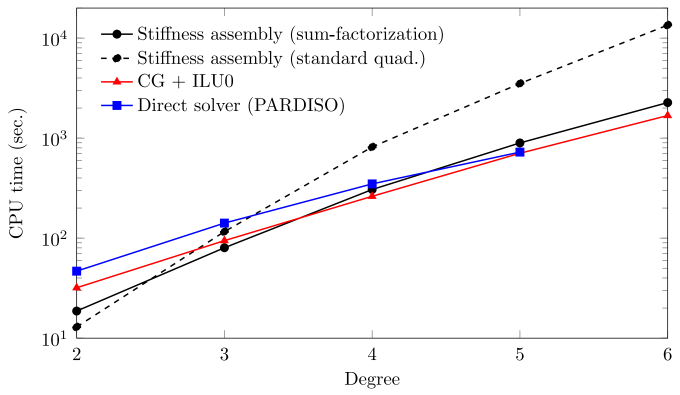

Fig. 3. Linear elasticity problem in 3D: comparison of the assembling and solving CPU time vs. degree for a grid of $30 \times 15 \times 30$ elements. The time of the iterative linear solver is the sum of the preconditioner computation (ILU(0)) and the Conjugate Gradient iterations. Both, direct and iterative solvers, are ran using a single thread. The number of degrees of freedom for each degree p are: $\text{ndof}_{p=2} = 52\,224$, $\text{ndof}_{p=3} = 60\,588$, $\text{ndof}_{p=4} = 69\,768$, $\text{ndof}_{p=5} = 79\,800$ and $\text{ndof}_{p=6} = 90\,720$.

where $\mathcal{C}$ is the fourth-order elasticity tensor and ε is the infinitesimal strain tensor, that is the symmetric part of the gradient of u.

In this example an isotropic material has been chosen, therefore the components of $\mathcal{C}$ are $\mathcal{C}_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk})$, $\forall i, j, k, l = 1, \ldots, d$. λ and μ are the Lamé parameters, which numerical values for this problem are $\lambda = 576.92$ and $\mu = 384.62$ (the corresponding ones to $E = 1.0 \cdot 10^3$ and $\nu = 0.3$). Homogeneous Dirichlet conditions have been applied to all the components of one of the faces, and non homogeneous Neumann conditions to the opposite face, with $t = \{1, 0, 0\}$. In the other faces homogeneous Neumann conditions are applied and $f = \{0, 0, 0\}$.

The problem domain Ω and the solution of the problem are shown in Fig. 2 for a mesh of $n = 30 \times 15 \times 30$ elements (the second parametric direction corresponds to the thickness of the beam). The quadrature cost for a whole mesh of n elements will be obviously proportional to n. The algorithm for the evaluation of the stiffness matrix of the elasticity problem (4.1) using sum-factorization is described in the Appendix.

In Fig. 3 the CPU times for the evaluation of stiffness matrices for both, sum factorization and standard quadrature, are shown together with the solver time using a direct (Intel MKL PARDISO) and an iterative linear solver (Conjugate Gradient + ILU(0) preconditioner), both provided by the Trilinos library [25]. For the discussion of these and other strategies for the linear solvers we refer to [26].

Some important remarks that arise from the results in Fig. 3 are:

- in terms of CPU cost, for $p \geq 3$ the standard quadrature is more demanding that the sum-factorization, and equivalent for $p = 2$, as already noticed and explained in Section 4.1;
- the CPU time ratio between sum-factorization and the iterative solver is constant w.r.t. p;

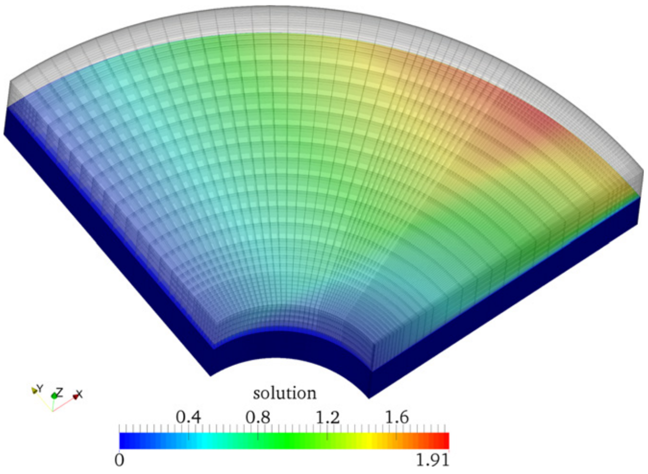

Fig. 4. Geometry and solution at the midplane $z = 0.5$ for the advection–diffusion problem (4.3).

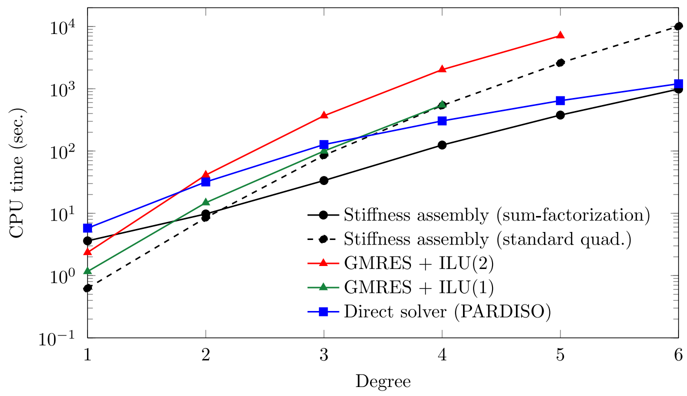

Fig. 5. Advection–diffusion problem in 3D: comparison of the CPU time vs. degree for a grid of $60 \times 84 \times 12$ elements. The time of the iterative linear solver is the sum of the preconditioner computation with the GMRES iterations. The direct solver (PARDISO) is used with a single thread. Convection/diffusion ratio $\frac{b}{\mu} = 10^3$. The number of degrees of freedom for each degree p is: $\mathrm{ndof}_{p=1} = 67\,405$, $\mathrm{ndof}_{p=2} = 74\,648$, $\mathrm{ndof}_{p=3} = 82\,215$, $\mathrm{ndof}_{p=4} = 90\,112$, $\mathrm{ndof}_{p=5} = 98\,345$ and $\mathrm{ndof}_{p=6} = 106\,920$.

- for this problem, the iterative solver outperform the direct solver for $p < 5$ (for $p > 5$ the memory requirement of the direct solver factorization exceeded the available memory on our system).

For the iterative solver the convergence tolerance considered is 10^{-7} and, in all the cases, the convergence was achieved without any restart of the method.

4.3. Advection–diffusion problem

For the tests we have selected the domain Ω to be the region of space with cylindrical coordinates $1 \le \rho \le 4$, $0 \le \theta \le \frac{\pi}{2}$ and $0 \le z \le 1$ (see Fig. 4), and considered the elliptic problem:

$$\begin{cases} -\mu \nabla^2 u + \boldsymbol{b} \cdot \nabla u = 0.5 & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases} \tag{4.3}$$

with $\boldsymbol{b} = (1, 0, 0)^T$ and $\mu = 10^{-3}$.

The problem has been solved using a mesh of $n = 60 \times 84 \times 12$ elements, adapted in order to resolve the boundary layer along the side with cylindrical coordinate $\rho = 4$ and avoid spurious oscillations on the numerical IGA solution of (4.3). The CPU times for the simulation are shown in Fig. 5.

The curves refer to the CPU time needed for the evaluation of the stiffness matrix using the sum factorization and the standard quadrature approaches and the time needed by some linear solvers to converge to the solution.

The stiffness matrix arising from the discretization of the problem (4.3) is non-symmetric, and for the solution of the associated linear system we have tested different solver strategies: both iterative (GMRES+ILU(k) preconditioner) and direct (Intel MKL PARDISO), provided by the Trilinos library [25].

Regarding the iterative strategy, we have experienced that for this problem is difficult to find a unique combination of GMRES parameters + ILU(k) preconditioner that works well for all degrees p (we have considered $p = 1, \dots, 5$). Increasing the fill-in level k of ILU(k) increases robustness but, at the same time, results in a higher CPU cost for the linear solver.

Regarding the solution of the linear system with a direct solver we experienced that, at least for the degrees we have tested (from 1 to 6), the solution is always obtained, and the CPU cost for $p \geq 4$ was lower than the iterative approaches. We were not able to compute a solution for $p > 6$ because the memory required by the solver during the factorization phase exceeded the memory available on our system.

Finally, we have noticed the following on the cost for the computation and assembly of the stiffness matrix:

- analogously to the example above, in terms of CPU cost, for $p \leq 2$ the standard quadrature is less expensive ($p = 1$) or equivalent ($p = 2$) than sum-factorization;
- for $p \geq 3$ the sum-factorization outperform the standard quadrature approach and its CPU cost is always smaller than the CPU cost for the linear solver, while the stiffness matrix computation with standard quadrature costs more than the best linear solver approach when $p \geq 4$ (that is obtained using the direct solver).

As in the previous example, the convergence tolerance of the iterative solver is 10^{-7} and no restart was applied to the method.

4.4. Implementation details

All the previous algorithms and tests are implemented by the use of the igatools library. igatools is an open source general purpose isogeometric library, written in C++11 and designed using the object-oriented paradigm. Currently igatools provides two classes that group the methods for the computation of the local matrices associated with the elliptic operators

- `EllipticOperatorsStdIntegration` for the computation using the standard quadrature technique;
- `EllipticOperatorsSFIntegration` for the computation using the sum-factorization technique.

For further information on the igatools library, see [15] and the official website http://www.igatools.org.

5. Summary and conclusions

We have considered the use of the sum-factorization for the calculation of the integrals arising in Galerkin isogeometric analysis. The minor changes needed in the actual coding in order to implement the proposed technique are presented in general and then detailed in the framework of the igatools code. This modification leads to a saving in terms of computational cost that allows high degree computations. The proposed numerical tests confirm the advantage of the sum-factorization when compared to the standard technique.

Acknowledgements

A. Buffa, F. Calabrò and G. Sangalli were partially supported by the GNCS project "Dall'Approssimazione all'Algebra Lineare: metodi numerici per l'Analisi Isogeometrica". P. Antolin, A. Buffa, M. Martinelli and G. Sangalli were also partially supported by the Italian MIUR through the FIRB "Futuro in Ricerca" Grant RBFR08CZ0S and A. Buffa and G. Sangalli by the Italian MIUR through the PRIN-2012 "Metodologie Innovative Nella Modellistica Differenziale Numerica" Grant, by the European Research Council through the FP7 Ideas Starting Grant *GeoPDEs* (205004) and Consolidator Grant *HIgeoM* (616563), by the European Commission through the FP7 Factories of the Future project *TERRIFIC* (FP7-2011-NMP-ICT-FoF 284981). Finally, P. Antolin and G. Sangalli received support also from the FP7 Ideas Starting Grant *ISOBIO* (259229). This support is gratefully acknowledged.

Appendix. Sum-factorization for the linear elasticity stiffness matrix

In solid mechanics, it is a common practice the use of Voigt notation for expressing the Galerkin approximation of the strong equation (4.1) as $\mathbb{S} = \{K_{\boldsymbol{\alpha},\boldsymbol{\beta}}\} \in \mathbb{R}^{d(p+1)^d \times d(p+1)^d}$, being $K_{\boldsymbol{\alpha},\boldsymbol{\beta}} \in \mathbb{R}^{d\times d}$ such as

$$K_{\boldsymbol{\alpha},\boldsymbol{\beta}} = \int_{\Omega} \mathbb{B}^{\mathrm{T}}_{\boldsymbol{\alpha}}(\boldsymbol{x})\mathbb{D}(\boldsymbol{x})\mathbb{B}_{\boldsymbol{\beta}}(\boldsymbol{x})\, d\boldsymbol{x}, \tag{A.1}$$

see [27] for further details.

For $d = 3$ the matrix $\mathbb{D}$, that is the Voigt representation of the fourth-order elasticity tensor $\mathcal{C}$ in (4.2), can be expressed as

$$\mathbb{D}(\boldsymbol{x}) = \begin{bmatrix} \mathcal{C}_{1111}(\boldsymbol{x}) & \mathcal{C}_{1122}(\boldsymbol{x}) & \mathcal{C}_{1133}(\boldsymbol{x}) & \mathcal{C}_{1112}(\boldsymbol{x}) & \mathcal{C}_{1113}(\boldsymbol{x}) & \mathcal{C}_{1123}(\boldsymbol{x}) \\ & \mathcal{C}_{2222}(\boldsymbol{x}) & \mathcal{C}_{2233}(\boldsymbol{x}) & \mathcal{C}_{2212}(\boldsymbol{x}) & \mathcal{C}_{2213}(\boldsymbol{x}) & \mathcal{C}_{2223}(\boldsymbol{x}) \\ & & \mathcal{C}_{3333}(\boldsymbol{x}) & \mathcal{C}_{3312}(\boldsymbol{x}) & \mathcal{C}_{3313}(\boldsymbol{x}) & \mathcal{C}_{3323}(\boldsymbol{x}) \\ & & & \mathcal{C}_{1212}(\boldsymbol{x}) & \mathcal{C}_{1213}(\boldsymbol{x}) & \mathcal{C}_{1223}(\boldsymbol{x}) \\ & \text{sym.} & & & \mathcal{C}_{1313}(\boldsymbol{x}) & \mathcal{C}_{1323}(\boldsymbol{x}) \\ & & & & & \mathcal{C}_{2323}(\boldsymbol{x}) \end{bmatrix} \tag{A.2}$$

and $\mathbb{B}_{\boldsymbol{\alpha}}$ as

$$\mathbb{B}_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_1}(\boldsymbol{x}) & 0 & 0 \\ 0 & \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_2}(\boldsymbol{x}) & 0 \\ 0 & 0 & \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_3}(\boldsymbol{x}) \\ \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_2}(\boldsymbol{x}) & \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_1}(\boldsymbol{x}) & 0 \\ \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_3}(\boldsymbol{x}) & 0 & \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_1}(\boldsymbol{x}) \\ 0 & \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_3}(\boldsymbol{x}) & \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_2}(\boldsymbol{x}) \end{bmatrix} = \sum_{i=1}^{3} M_i \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_i}(\boldsymbol{x}), \tag{A.3}$$

where

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad M_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad M_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

Moreover, $\frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_i}(\boldsymbol{x}) = \sum_{k=1}^{3} (\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}(\hat{\boldsymbol{x}}))_{i,k} \frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_k}(\hat{\boldsymbol{x}})$ for $i = \{1, 2, 3\}$, therefore

$$\mathbb{B}_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \sum_{i=1}^{3} M_i \frac{\partial B_{\boldsymbol{\alpha}}}{\partial x_i}(\boldsymbol{x}) = \sum_{i=1}^{3} M_i \sum_{k=1}^{3} (\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}(\hat{\boldsymbol{x}}))_{i,k} \frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_k}(\hat{\boldsymbol{x}}) = \sum_{k=1}^{3} \hat{M}_k(\hat{\boldsymbol{x}}) \frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_k}(\hat{\boldsymbol{x}})$$

with $\hat{M}_k(\hat{\boldsymbol{x}}) = \sum_{i=1}^{3} M_i (\hat{\boldsymbol{D}}\boldsymbol{F}^{-T}(\hat{\boldsymbol{x}}))_{i,k}$, $k = \{1, 2, 3\}$.

With these new definitions we can write the integral (A.1) as:

$$\int_{\Omega} \mathbb{B}^{T}_{\boldsymbol{\alpha}}(\boldsymbol{x})\mathbb{D}(\boldsymbol{x})\mathbb{B}_{\boldsymbol{\beta}}(\boldsymbol{x})\, d\boldsymbol{x} = \sum_{i=1}^{3}\sum_{j=1}^{3} \int_{Q} \frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_i}(\hat{\boldsymbol{x}}) \hat{\mathbb{D}}_{i,j}(\hat{\boldsymbol{x}}) \frac{\partial \hat{B}_{\boldsymbol{\beta}}}{\partial \hat{x}_j}(\hat{\boldsymbol{x}})\, d\hat{\boldsymbol{x}} \tag{A.4}$$

where $\hat{\mathbb{D}}_{i,j}(\hat{\boldsymbol{x}})$ is the 3×3 matrix defined by $\hat{\mathbb{D}}_{i,j}(\hat{\boldsymbol{x}}) = \hat{M}^{T}_{i}(\hat{\boldsymbol{x}})\mathbb{D}(\hat{\boldsymbol{x}})|\hat{\boldsymbol{D}}\boldsymbol{F}(\hat{\boldsymbol{x}})|\hat{M}_j(\hat{\boldsymbol{x}})$, $i, j = \{1, 2, 3\}$. Using (A.4) we can use the sum-factorization technique in order to exploit the tensor-product structure of the terms $\frac{\partial \hat{B}_{\boldsymbol{\alpha}}}{\partial \hat{x}_i}(\hat{\boldsymbol{x}})$ and $\frac{\partial \hat{B}_{\boldsymbol{\beta}}}{\partial \hat{x}_j}(\hat{\boldsymbol{x}})$.

An analogous development can be done for $d = 2$.

References

[1] T.J.R. Hughes, J.A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement, Comput. Methods Appl. Mech. Engrg. 194 (2005) 4135–4195.

[2] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs, Isogeometric Analysis. Toward Integration of CAD and FEA, Wiley, 2009.

[3] L. Beiraõ da Veiga, A. Buffa, G. Sangalli, R. Vázquez, Mathematical analysis of variational isogeometric methods, Acta Numer. 23 (2014) 157–287.

[4] J.A. Evans, Y. Bazilevs, I. Babuška, T.J.R. Hughes, n-Width, sup-infs, and optimality ratios for the k-version of the isogeometric finite element method, Comput. Methods Appl. Mech. Engrg. 198 (2009) 1726–1741.

[5] L. Beiraõ da Veiga, A. Buffa, J. Rivas, G. Sangalli, Some estimates for h–p–k-refinement in isogeometric analysis, Numer. Math. 118 (2) (2011) 271–305.

[6] T.J.R. Hughes, A. Reali, G. Sangalli, Duality and unified analysis of discrete approximations in structural dynamics and wave propagation: comparison of p-method finite elements with k-method NURBS, Comput. Methods Appl. Mech. Engrg. 197 (2008) 4104–4124.

[7] A. Buffa, J. Rivas, G. Sangalli, R. Vázquez, Isogeometric discrete differential forms in three dimensions, SIAM J. Numer. Anal. 49 (2) (2011) 818–844.

[8] J.A. Evans, T.J.R. Hughes, Isogeometric divergence-conforming B-splines for the steady Navier–Stokes equations, Math. Models Methods Appl. Sci. 23 (08) (2013) 1421–1478.

[9] T.J.R. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 199 (2010) 301–313.

[10] F. Auricchio, F. Calabrò, T.J.R. Hughes, A. Reali, G. Sangalli, A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 249–252 (2012) 15–27.

[11] D. Schillinger, S.J. Hossain, T.J.R. Hughes, Reduced Bézier element quadrature rules for quadratic and cubic splines in isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 277 (2014) 1–45.

[12] T. Eibner, J.M. Melenk, Fast algorithms for setting up the stiffness matrix in hp-FEM: a comparison, in: HERCMA 2005 Conference Proceedings, Techn. Univ. Chemnitz SFB 393, 2006.

[13] J.M. Melenk, K. Gerdes, C. Schwab, Fully discrete hp-finite elements: fast quadrature, Comput. Methods Appl. Mech. Engrg. 190 (32) (2001) 4339–4364.

[14] M. Ainsworth, G. Andriamaro, O. Davydov, Bernstein–Bézier finite elements of arbitrary order and optimal assembly procedures, SIAM J. Sci. Comput. 33 (6) (2011) 3087–3109.

[15] M.S. Pauletti, M. Martinelli, N. Cavallini, P. Antolìn, Igatools: an isogeometric analysis library, I.M.A.T.I.-C.N.R. Technical Report No. 3PV14/1/0, 2014, pp. 1–27.

[16] A. Karatarakis, P. Metsis, M. Papadrakakis, GPU-acceleration of stiffness matrix calculation and efficient initialization of EFG meshless methods, Comput. Methods Appl. Mech. Engrg. 258 (2013) 63–80.

[17] F. Calabrò, C. Manni, The choice of quadrature in NURBS-based isogeometric analysis, in: M. Papadrakakis, M. Kojic, and Tuncer I. (Eds.), Proceedings SEECCM III, 2013.

[18] F. Calabrò, C. Manni, F. Pitolli, Computation of quadrature rules for integration with respect to refinable functions on assigned nodes, Appl. Numer. Math. (2014) http://dx.doi.org/10.1016/j.apnum.2014.11.010.

[19] A. Mantzaflaris, B. Jüttler, Exploring matrix generation strategies in isogeometric analysis, in: Michael Floater, Tom Lyche, Marie-Laurence Mazure, Knut Mørken, Larry L. Schumaker (Eds.), Mathematical Methods for Curves and Surfaces, in: Lecture Notes in Computer Science, vol. 8177, Springer, Berlin, Heidelberg, 2014, pp. 364–382.

[20] A. Mantzaflaris, B. Jüttler, Integration by interpolation and look-up for Galerkin-based isogeometric analysis, Comput. Methods Appl. Mech. Engrg. 284 (2015) 373–400.

[21] F. Auricchio, L. Beiraõ da Veiga, T.J.R. Hughes, A. Reali, G. Sangalli, Isogeometric collocation methods, Math. Models Methods Appl. Sci. 20 (11) (2010) 2075–2107.

[22] D. Schillinger, J.A. Evans, A. Reali, M.A. Scott, T.J.R. Hughes, Isogeometric collocation: cost comparison with Galerkin methods and extension to adaptive hierarchical NURBS discretizations, Comput. Methods Appl. Mech. Engrg. 267 (2013) 170–232.

[23] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, Elsevier, 1978.

[24] S.A. Orszag, Spectral methods for problems in complex geometries, J. Comput. Phys. 37 (1) (1980) 70–92.

[25] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, J.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, A.G. Salinger, H.K. Thornquist, R.S. Tuminaro, J.M. Willenbring, A. Williams, K.S. Stanley, An overview of the Trilinos project, ACM Trans. Math. Software 31 (2005) 397–423.

[26] N. Collier, L. Dalcin, D. Pardo, V.M. Calo, The cost of continuity: performance of iterative solvers on isogeometric finite elements, SIAM J. Sci. Comput. 35 (2013) A767–A784.

[27] T. Belytschko, W.K. Moran, B. Moran, Nonlinear Finite Elements for Continua and Structures, Wiley, 2000.