Efficient Matrix Computation for Isogeometric Discretizations with Hierarchical B-splines in Any Dimension

Maodong Pan[a,b,*], Bert Jüttler[b,c], Felix Scholz[c]

[a]*Department of Mathematics, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China*
[b]*Institute of Applied Geometry, Johannes Kepler University Linz, Austria*
[c]*Johann Radon Institute for Computational and Applied Mathematics, Austrian Academy of Sciences, Austria*

Abstract

Hierarchical B-splines, which possess the local refinement capability, have been recognized as a useful tool in the context of isogeometric analysis. However, similar as for tensor-product B-splines, isogeometric simulations with hierarchical B-splines face a big computational burden from the perspective of matrix assembly, particularly if the spline degree p is high. To address this issue, we extend the recent work [43] – which introduced an efficient assembling approach for tensor-product B-splines – to the case of hierarchical B-splines. In the new approach, the integrand factor is transformed into piecewise polynomials via quasi-interpolation. Subsequently, the resulting elementary integrals are pre-computed and stored in a look-up table. Finally, the sum-factorization technique is adopted to accelerate the assembly process. We present a detailed analysis, which reveals that the presented method achieves the expected complexity of $\mathcal{O}(p^{d+1})$ per degree of freedom (without taking sparse matrix operations into account) under the assumption of mesh admissibility. We verify the efficiency of the new method by applying it to an elliptic problem on the three-dimensional domain and a parabolic problem on the four-dimensional domain in space-time, respectively. A comparison with standard Gaussian quadrature is also provided.

Keywords: Isogeometric analysis; Hierarchical B-splines; Assembling matrices; Spline projection, looking up and sum-factorization; Complexity

1. Introduction

Isogeometric analysis (IgA) introduced by Hughes et al. [32] makes it possible to integrate computer aided design (CAD) and numerical simulations into a unified framework [19]. It employs the smooth non-uniform rational B-splines (NURBS) that are used in CAD geometry descriptions to represent the unknowns of partial differential equations (PDEs),

*Corresponding author.
Email addresses: mdpan@mail.ustc.edu.cn (Maodong Pan), bert.juettler@jku.at (Bert Jüttler), felix.scholz@ricam.oeaw.ac.at (Felix Scholz)

thereby allowing direct approximations of high-order PDEs and improving the accuracy per degree of freedom when compared to the classical finite element method (FEM) with C^0 continuity [9, 24]. Consequently, as pointed out in the literature [13, 14, 21, 23, 34], the smoothness of NURBS (or in other words, the relatively large support of the basis functions) entails more non-zero entries in the system matrix, thereby causing a big computational burden to isogeometric simulations at the point of matrix assembly.

This effect becomes more pronounced for high polynomial degrees and more spatial dimensions. In particular, when a standard FEM framework is adopted to implement isogeometric methods, the simplest way is to use the standard Gaussian quadrature with an element-by-element assembly strategy. E.g., for solving an elliptic boundary value problem with splines of degree p in a d-dimensional space, each local system matrix has $\mathcal{O}(p^{2d})$ elements, which are computed by quadrature at $\mathcal{O}(p^d)$ Gauss nodes. The total complexity of using standard Gauss quadrature for matrix assembly amounts to $\mathcal{O}(Np^{3d})$ floating-point operations, where N is the number of degrees of freedom.

Consequently, the resulting costs grow too fast with respect to the degree p, limiting IgA to low spline degrees (primarily quadratics and cubics) from the viewpoint of computational efficiency. This is an unfavorable factor in isogeometric k-refinement [46], which possesses several advantages such as higher accuracy per degree of freedom and improved spectral behaviour [20, 45]. So far, substantial research has been carried out to reduce the computational effort required for assembling the system matrices:

- First, many works [2, 4–6, 15, 16, 22, 30, 33, 48] focused on exploring *specialized* or *reduced quadrature rules* that improve Gaussian quadrature for making the integrations more efficient. By exploiting the built-in smoothness of splines, the number of evaluations can be reduced, thereby lowering the computational time needed to assemble the matrices.

- Second, the technique of *sum-factorization*, which originates in spectral methods and high-order finite element methods, has been successfully adapted to the formation of isogeometric Galerkin matrices [1, 10, 16, 43]. It arranges the computations in a way that exploits the tensor-product structure of multivariate splines, dramatically cutting down the computational costs. Moreover, the technique of *low-rank approximation* was shown to be a useful tool for isogeometric matrix assembly [31, 41, 49]. It is observed that the system matrices arising from isogeometric discretizations can be well approximated by a sum of few Kronecker products of matrices obtained from univariate integrals. These approaches have a complexity that scales sub-linearly with the number of degrees of freedom. Last but not least, the method of *integration by interpolation and look-up* (IIL) [40] relies on spline projections of the weighting functions appearing in the integrals and uses the built look-up tables of tri-product B-spline integrals to calculate the resulting approximate integrands. An improved version of this method was proposed in [43] by exploring the combination with sum-factorization. Some of these approaches are summarized in Table 1.

- Third, other approaches have been successfully explored, which include *GPU pro-*

Method	Complexity	Remark
Gauss quadrature with sum-factorization (GQSF) [1]	$\mathcal{O}(Np^{2d+1})$	
Integration by interpolation and look-up (IIL) [40]	$\mathcal{O}(Np^{2d})$	
Tensor decomposition [41]	$\mathcal{O}(RNp^{d})$	truncated tensor rank R, R depends on the geometry
Weighted quadrature [16]	$\mathcal{O}(Np^{d+1})$	the symmetry of the matrices is not preserved
Partial tensor decomposition [49]	$\mathcal{O}(rNp^{d})$	truncated matrix rank r, r depends on the geometry
Improved GQSF [10]	$\mathcal{O}(Np^{d+2})$	
Improved IIL [43]	$\mathcal{O}(Np^{d+1})$	

Table 1: Some of the existing matrix assembly approaches.

gramming [36] and the *surrogate matrix methodology* [23]. Furthermore, *collocating* the strong form of PDEs [3, 47] is an alternative way that reduces the cost of assembly. Although this approach possesses the minimal complexity, i.e., $\mathcal{O}(1)$ per degree of freedom, it lacks the theoretical guarantee of optimal convergence rates. To tackle this problem, *variational collocation* methods [29, 42] that establish a direct connection between the Galerkin method and the classical collocation approaches were investigated.

As a natural generalization of tensor-product B-splines, *hierarchical B-splines* (HB-splines), which were introduced in the doctoral thesis [37], inherit all the properties of classical B-splines and allow for an effective local control of the refinement. In recent years, local refinement with HB-splines has become an active topic in the context of isogeometric analysis [11, 17, 18]. However, similar to tensor-product B-splines, isogeometric simulations with HB-splines face the computational challenge of matrix formation. Therefore, developing optimized assembly procedures tailored for HB-splines is an important research task in this field. To this end, Pan et al. [44] recently carried over the improved IIL method [43] to the formation of the system matrices arising from isogeometric discretizations that are based on *bivariate* HB-splines. However, extending this approach to the case of d-variate ($d > 2$) HB-splines is challenging, since the construction and efficient evaluation of the auxiliary tensors would need further study.

In the present paper we aim at developing an efficient methodology for assembling the isogeometric Galerkin matrices with respect to HB-splines in any dimension. We achieve this by adopting the quasi-interpolation, looking-up and sum-factorization techniques. A detailed theoretical analysis demonstrates that the proposed method maintains the same order of complexity as for the improved IIL approach, provided that the hierarchical meshes satisfy certain admissibility assumptions.

The remainder of the paper consists of five sections. We firstly recall the admissibility conditions for HB-splines and introduce the integrals that arise in isogeometric discretizations. Section 3 is devoted to a detailed discussion on the new method. In Section 4, we provide a complexity analysis of the assembly algorithm. Section 5 presents several numerical experiments to verify the effectiveness of our method. Finally, the conclusions and future work are listed in Section 6.

2. Preliminaries

We start by recalling the basic concepts of HB-splines with admissibility and by deriving the isogeometric discretizations considered in this paper.

2.1. Hierarchical B-splines and the admissibility

We consider a finite sequence of nested d-variate tensor-product B-spline spaces

$$V^0 \subseteq V^1 \subseteq \cdots \subseteq V^L$$

defined on the bounded domain $\Omega^0 = [0,1]^d$, where the upper index is called the *level.* The spline spaces V^ℓ have the same polynomial degree p and are spanned by the normalized tensor-product B-splines

$$\mathcal{B}^\ell = \{\boldsymbol{\beta}_{\boldsymbol{i}}^\ell \mid \boldsymbol{i} \in \mathcal{F}^\ell\}$$

with index sets $\mathcal{F}^\ell$. These sets are called the *full index sets*, since they contain the indices of all the tensor-product B-splines that are available at a given level. Each basis function

$$\boldsymbol{\beta}_{\boldsymbol{i}}^\ell(\hat{\boldsymbol{x}}) = \prod_{k=1}^{d} \beta_{i_k}^\ell(\hat{x}_k), \quad \boldsymbol{i} = (i_1, \ldots, i_d), \quad \hat{\boldsymbol{x}} = (\hat{x}_1, \ldots, \hat{x}_d)$$

is a product of d univariate B-splines $\beta_{i_k}^\ell$ of degree p. For each level ℓ and coordinate direction k ($k \in \{1, \ldots, d\}$), the basis

$$\mathcal{B}_k^\ell = \{\beta_{i_k}^\ell \mid i_k = 0, 1, \ldots, n_k^\ell - 1\},$$

which spans a univariate spline space, is defined by $n_k^\ell - p - 1$ single inner knots and $(p+1)$-fold boundary knots 0 and 1. We assume that each knot span of the B-splines $\mathcal{B}_k^\ell$ is refined into two knot spans to create the splines $\mathcal{B}_k^{\ell+1}$ (dyadic refinement), where uniform and non-uniform refinement are both allowed.

Summing up, we have

- $\mathcal{B}^\ell = \mathcal{B}_1^\ell \otimes \cdots \otimes \mathcal{B}_d^\ell$,
- $n_k^{\ell+1} = 2n_k^\ell - p$,
- the size of the full index set $\mathcal{F}^\ell$ is $n_1^\ell \times \cdots \times n_d^\ell$, and
- the spline spaces V^ℓ possess the maximum smoothness C^{p-1} inside Ω^0.

In order to introduce the HB-splines defined on Ω^0, we consider an associated finite sequence of inversely nested domains $\{\Omega^\ell\}_{\ell=0,1,\dots,L}$ satisfying

$$[0,1]^d = \Omega^0 \supseteq \Omega^1 \supseteq \cdots \supseteq \Omega^L,$$

where $\Omega^\ell \subset \mathbb{R}^d$ denotes the region selected to be refined to level ℓ and its boundary $\partial\Omega^\ell$ must be aligned with the knot lines of $V^{\ell-1}$.

The construction of HB-splines is based on the selection mechanism proposed by Kraft [37], which eliminates those B-splines that can be represented as linear combinations of selected B-splines at finer levels. More precisely, HB-splines $\mathcal{H}$ are defined as

$$\mathcal{H} = \{\boldsymbol{\beta}_{\boldsymbol{i}}^\ell \in \mathcal{B}^\ell \mid \boldsymbol{i} \in \mathcal{I}^\ell,\ \ell = 0,1,\dots,L\}$$

with the index set

$$\mathcal{I}^\ell = \{\boldsymbol{i} \in \mathcal{F}^\ell \mid \operatorname{supp} \boldsymbol{\beta}_{\boldsymbol{i}}^\ell \subseteq \Omega^\ell \wedge \operatorname{supp} \boldsymbol{\beta}_{\boldsymbol{i}}^\ell \nsubseteq \Omega^{\ell+1}\},$$

where $\operatorname{supp}\beta$ denotes the support of β. It follows that any basis function $\boldsymbol{\beta}_{\boldsymbol{i}}^\ell \in \mathcal{H}$ can be identified by the pair $(\ell, \boldsymbol{i})$ that combines the level ℓ and the index $\boldsymbol{i}$ of $\boldsymbol{\beta}_{\boldsymbol{i}}^\ell$. Similarly, each univariate B-spline $\beta_{i_k}^\ell$ that constitutes $\boldsymbol{\beta}_{\boldsymbol{i}}^\ell$ can be identified by the pair (ℓ, i_k), $k = 1,\dots,d$.

Assumption 1. The basis functions in $\mathcal{H}$ that take nonzero values over any element of the hierarchical mesh belong to at most two consecutive levels.

HB-splines that admit the above assumption are called *admissible* HB-splines of class 2 [11]. They are equivalently defined by requiring that

$$\operatorname{supp} \boldsymbol{\beta}_{\boldsymbol{i}}^\ell \cap \Omega^{\ell+2} = \emptyset, \quad \forall \boldsymbol{i} \in \mathcal{I}^\ell,\ \ell = 0,\dots,L-2. \tag{1}$$

References [8, 11, 12, 25] present refinement algorithms that generate admissible meshes. The core operation in both algorithms is the enlargement of some nested domains such that the input HB-splines satisfy the admissibility condition (1).

Admissible HB-splines of class 2 possess the following useful properties:

- First, the number of basis functions in $\mathcal{H}$ that are nonzero over any element of the hierarchical mesh does not exceed $2(p+1)^d$, and this upper bound is independent of the overall number L of levels in the hierarchy.

- Second, under the assumption of dyadic refinement, the number of basis functions in $\mathcal{H}$ that have a non-empty support intersection with any given basis $\boldsymbol{\beta}_{\boldsymbol{i}}^\ell \in \mathcal{H}$ amounts to $\mathcal{O}(p^d)$.

  This fact has also been shown in [44, Section 2.2].

These two properties are the key ingredients for hierarchical isogeometric methods [11]. In particular, the latter one is critical for ensuring the sparsity of the matrices arising in isogeometric discretizations with HB-splines. This will be confirmed in the subsequent sections.

In order to facilitate the presentation of the proposed algorithm in the subsequent section, we introduce the *projected index sets*

$$\mathcal{P}_k^\ell = \{i_k \mid \exists i_1, \ldots, i_{k-1}, i_{k+1}, \ldots, i_d \ : \ \boldsymbol{i} \in \mathcal{I}^\ell\}, \quad k = 1, \ldots, d$$

and

$$\widetilde{\mathcal{P}}_k^\ell(i_{k+1}, \ldots, i_d) = \{i_k \mid \exists i_1, \ldots, i_{k-1} \ : \ \boldsymbol{i} \in \mathcal{I}^\ell\}, \quad k = 1, \ldots, d-1$$

of $\mathcal{I}^\ell$. We also introduce the univariate *neighbor index sets*

$$\mathcal{N}_k^{\ell'}(\ell, i_k) = \{i'_k \mid \operatorname{supp}(\beta_{i_k}^\ell \beta_{i'_k}^{\ell'}) \neq \emptyset, \ i'_k = 0, 1, \ldots, n_k^{\ell'} - 1\} \tag{2}$$

that consist of the indices associated with the univariate B-splines of level ℓ' possessing a non-empty support intersection with $\beta_{i_k}^\ell \in \mathcal{B}_k^\ell$, $k = 1, \ldots, d$, and their intersection with the projections,

$$\mathcal{M}_k^{\ell'}(\ell, i_k) = \mathcal{N}_k^{\ell'}(\ell, i_k) \cap \mathcal{P}_k^{\ell'} \ . \tag{3}$$

From [44, Lemma 1], we already know that there are at most $\lceil 2^{\ell'-\ell}(p+1)\rceil + p + 1$ univariate B-splines of level ℓ' that have a non-empty support intersection with a given B-spline $\beta_{i_k}^\ell$. Also, since we assume that the HB-splines considered in this paper are admissible of class 2, it suffices to consider the univariate B-splines of the at most three adjacent levels

$$\ell' = \max(0, \ell-1), \ldots, \min(\ell+1, L) \ . \tag{4}$$

We have the following lemma:

Lemma 2. *The total cardinality of the index sets $\mathcal{N}_k^{\ell'}(\ell, i_k)$ for the at most three adjacent levels, cf. Eq.* (4)*, satisfies*

$$\sum_{\ell'=\max(0,\ell-1)}^{\min(\ell+1,L)} |\mathcal{N}_k^{\ell'}(\ell, i_k)| \leq \sum_{\ell'=\max(0,\ell-1)}^{\min(\ell+1,L)} (\lceil 2^{\ell'-\ell}(p+1)\rceil + p + 1) = \mathcal{O}(p) \ .$$

2.2. Isogeometric discretizations with HB-splines

In order to illustrate the framework of isogeometric Galerkin discretizations with HB-splines, let us consider the following elliptic problem with Dirichlet boundary conditions

$$\begin{cases} \mathcal{D}u = f & \text{in } \Omega, \\ \quad u = g & \text{on } \partial\Omega \end{cases} \tag{5}$$

on a bounded domain $\Omega \subset \mathbb{R}^d$. The differential operator $\mathcal{D}$ is defined as

$$\mathcal{D}u = -\nabla \cdot (A \nabla u) + \sigma u \tag{6}$$

with a SPD (symmetric positive definite) matrix-valued function $A \in \mathbb{R}^{d \times d}$ and a non-negative scalar function σ. Moreover, A and σ are assumed to be smooth functions.

In the isogeometric setting, Ω is typically parameterized by a d-dimensional NURBS mapping

$$\boldsymbol{F} : \Omega^0 \to \Omega,$$

which we assume to be represented by a single-patch spline parameterization, in order to keep the presentation simple.

Following the isoparametric paradigm of IgA, we apply the operator $\mathcal{D}$ to the function $\hat{u} = u \circ \boldsymbol{F}$ that is defined on the parametric domain Ω^0, arriving at

$$\mathcal{D}(\hat{u} \circ \boldsymbol{F}^{-1}) = \frac{-1}{|\det \hat{\nabla} \boldsymbol{F}|} \hat{\nabla} \cdot \left(|\det \hat{\nabla} \boldsymbol{F}| \hat{\nabla} \boldsymbol{F}^{-1} \hat{A} \hat{\nabla} \boldsymbol{F}^{-T} \hat{\nabla} \hat{u} \right) + \hat{\sigma} \hat{u}$$

with $\hat{A} = A \circ \boldsymbol{F}$ and $\hat{\sigma} = \sigma \circ \boldsymbol{F}$, where the symbols $\hat{\nabla}\cdot$ and $\hat{\nabla}$ represent the pull-back of the divergence and gradient operator to the parametric domain Ω^0 respectively. By setting $\hat{\mathcal{D}}\hat{u} = \mathcal{D}(\hat{u} \circ \boldsymbol{F}^{-1})$, $\hat{f} = f \circ \boldsymbol{F}$ and $\hat{g} = g \circ \boldsymbol{F}$, we obtain an equivalent version of the problem (5)

$$\begin{cases} \hat{\mathcal{D}}\hat{u} = \hat{f} & \text{in } \Omega^0, \\ \quad \hat{u} = \hat{g} & \text{on } \partial\Omega^0. \end{cases} \tag{7}$$

Let $\hat{U} = \{\hat{u} | \hat{u} \in H^1(\Omega^0), \hat{u}|_{\partial\Omega^0} = \hat{g}\}$ and $\hat{V} = H_0^1(\Omega^0)$. We arrive at the weak formulation of the transformed problem (7): Find $\hat{u} \in \hat{U}$ such that

$$\hat{a}(\hat{u}, \hat{v}) = \hat{\tau}(\hat{v}) \text{ for all } \hat{v} \in \hat{V}, \tag{8}$$

where

$$\hat{a}(\hat{u}, \hat{v}) = \int_{\Omega^0} \left(\hat{\nabla} \hat{u}^T W \hat{\nabla} \hat{v} + w \hat{u} \hat{v} \right) \mathrm{d}\hat{\boldsymbol{x}} \quad \text{and} \quad \hat{\tau}(\hat{v}) = \int_{\Omega^0} |\det \hat{\nabla} \boldsymbol{F}| \hat{f} \hat{v} \mathrm{d}\hat{\boldsymbol{x}}.$$

Here we rewrite the matrix $|\det \hat{\nabla} \boldsymbol{F}| \hat{\nabla} \boldsymbol{F}^{-1} \hat{A} \hat{\nabla} \boldsymbol{F}^{-T}$ as W and $|\det \hat{\nabla} \boldsymbol{F}| \hat{\sigma}$ as w, respectively. The function $\hat{u} \in \hat{U}$ is a solution of the weak formulation (8) if and only if $u = \hat{u} \circ \boldsymbol{F}^{-1}$ is a weak solution of the original problem (5).

The key idea of Galerkin method for solving problem (8) lies in replacing the infinite-dimensional spaces $\hat{U}$ and $\hat{V}$ with finite-dimensional spaces $\hat{U}_h$ and $\hat{V}_h$, and solving the following discrete problem: Find $\hat{u}_h \in \hat{U}_h$ satisfying

$$\hat{a}(\hat{u}_h, \hat{v}_h) = \hat{\tau}(\hat{v}_h) \text{ for all } \hat{v}_h \in \hat{V}_h. \tag{9}$$

To this end, we choose the spaces $\hat{U}_h$ and $\hat{V}_h$ as $\operatorname{span} \mathcal{H}$ and $\operatorname{span} \mathcal{H}_0$ respectively, where $\mathcal{H}_0 \subset \mathcal{H}$ is defined as

$$\mathcal{H}_0 = \{\beta \,|\, \beta \in \mathcal{H},\ \beta|_{\partial\Omega^0} = 0\},$$

and represent $\hat{u}_h$ as linear combinations of the basis functions in $\mathcal{H}$,

$$\hat{u}_h = \sum_{\ell=0}^{L} \sum_{\boldsymbol{i} \in \mathcal{I}^\ell} u_{\boldsymbol{i}}^\ell \boldsymbol{\beta}_{\boldsymbol{i}}^\ell, \tag{10}$$

with unknown coefficients $\boldsymbol{u}$. Note that the coefficients associated with the boundary basis of $\mathcal{H}$ are determined by spline approximation at the boundary points of Ω^0 such that $\hat{u}_h|_{\partial\Omega^0} = \hat{g}$, thus the vector $\boldsymbol{u}$ collects the coefficients $\{u_{\boldsymbol{i}}^{\ell}\}$ with indices

$$\mathcal{I}_0 = \{(\ell, \boldsymbol{i}) | \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \in \mathcal{H}_0\}.$$

By setting $\hat{v}_h = \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'}, (\ell', \boldsymbol{i}') \in \mathcal{I}_0$, the discrete problem (9) leads to the linear system of algebraic equations

$$(S + M)\boldsymbol{u} = \boldsymbol{b} \tag{11}$$

of size $|\mathcal{I}_0| \times |\mathcal{I}_0|$, where S is the Galerkin stiffness matrix with the elements

$$S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} = \int_{\Omega^0} \hat{\nabla}\boldsymbol{\beta}_{\boldsymbol{i}}^{\ell^T} W \hat{\nabla}\boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \mathrm{d}\hat{\boldsymbol{x}}, \tag{12}$$

M is the Galerkin mass matrix with the entries

$$M_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} = \int_{\Omega^0} w \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \mathrm{d}\hat{\boldsymbol{x}}, \tag{13}$$

and the right-hand side vector $\boldsymbol{b}$ is composed of the elements

$$\boldsymbol{b}_{(\ell,\boldsymbol{i})} = \int_{\Omega^0} \Big(|\det \hat{\nabla}\boldsymbol{F}| \hat{f} \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} - \sum_{(\ell',\boldsymbol{i}') \in \mathcal{I}_\partial} u_{\boldsymbol{i}'}^{\ell'} \big(\hat{\nabla}\boldsymbol{\beta}_{\boldsymbol{i}}^{\ell^T} W \hat{\nabla}\boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} + w \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \big) \Big) \mathrm{d}\hat{\boldsymbol{x}}$$

with the index set

$$\mathcal{I}_\partial = \{(\ell, \boldsymbol{i}) | \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \in \mathcal{H} \setminus \mathcal{H}_0\}.$$

Therefore, the process of solving the elliptic problem (5) via isogeometric Galerkin approach includes assembling the stiffness matrix S, mass matrix M and solving the linear system (11). In this work, we focus on assembling the system matrices S and M. More precisely, we develop efficient algorithms for computing the integrals appearing in (12) and (13).

3. Integration via spline projection, look-up and sum-factorization

The elements of the stiffness and mass matrices displayed in (12) and (13) are multidimensional integrals in a unit cube. Their evaluation via Gauss quadrature is computationally expensive, especially for high polynomial degree p. In order to evaluate these integrals in an efficient way, we propose a three-stage approach which is quadrature-free. First, the weight functions W and w, which involve the mapping $\boldsymbol{F}$, its partial derivatives and the coefficients of the elliptic equation in (5), are projected into the hierarchical spline space. Consequently, the elements of the system matrices can be approximated by sums of integrals of tri-product B-splines. Second, three compact look-up tables are built for evaluating these integrals exactly. Finally, the sum-factorization method is exploited to speed up the procedure of matrix assembly.

3.1. Overall framework

We describe the proposed algorithm for the stiffness matrix and mass matrix, respectively.

- We firstly approximate the weight functions W and w with hierarchical spline functions contained in the space span $\mathcal{H}$, obtaining

$$W(\hat{\boldsymbol{x}}) \approx \sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}'' \in \mathcal{I}^{\ell''}} W_{(\ell'',\boldsymbol{i}'')} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''}(\hat{\boldsymbol{x}}) \tag{14}$$

  with the $d \times d$ coefficient matrices $W_{(\ell'',\boldsymbol{i}'')} = \left(W_{(\ell'',\boldsymbol{i}'')}^{\theta,\phi}\right)_{\theta,\phi=1,\dots,d}$ and

$$w(\hat{\boldsymbol{x}}) \approx \sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}'' \in \mathcal{I}^{\ell''}} w_{(\ell'',\boldsymbol{i}'')} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''}(\hat{\boldsymbol{x}}) \tag{15}$$

  with coefficients $w_{(\ell'',\boldsymbol{i}'')}$. Then the elements of the stiffness matrix S are transformed into

$$\begin{aligned} S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} &\approx \int_{\Omega^0} \hat{\nabla} \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell^T} \Big(\sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}'' \in \mathcal{I}^{\ell''}} W_{(\ell'',\boldsymbol{i}'')} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \Big) \hat{\nabla} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \mathrm{d}\hat{\boldsymbol{x}} \\ &= \sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}'' \in \mathcal{I}^{\ell''}} \sum_{\theta=1}^{d} \sum_{\phi=1}^{d} W_{(\ell'',\boldsymbol{i}'')}^{\theta,\phi} \int_{\Omega^0} \frac{\partial \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell}}{\partial \hat{x}_\theta} \frac{\partial \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'}}{\partial \hat{x}_\phi} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \mathrm{d}\hat{\boldsymbol{x}}, \end{aligned} \tag{16}$$

  and the elements of the mass matrix M can be rewritten as

$$\begin{aligned} M_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} &\approx \int_{\Omega^0} \Big(\sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}'' \in \mathcal{I}^{\ell''}} w_{(\ell'',\boldsymbol{i}'')} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \Big) \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \mathrm{d}\hat{\boldsymbol{x}} \\ &= \sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}'' \in \mathcal{I}^{\ell''}} w_{(\ell'',\boldsymbol{i}'')} \int_{\Omega^0} \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \mathrm{d}\hat{\boldsymbol{x}}. \end{aligned} \tag{17}$$

- From (16) and (17), the elements of the system matrices are approximately expressed as a linear combination of the integrals of tri-product B-splines

$$\int_{\Omega^0} \frac{\partial \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell}}{\partial \hat{x}_\theta} \frac{\partial \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'}}{\partial \hat{x}_\phi} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \mathrm{d}\hat{\boldsymbol{x}}, \quad \int_{\Omega^0} \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \mathrm{d}\hat{\boldsymbol{x}}. \tag{18}$$

  To compute these integrals accurately and efficiently, we build three compact tables that are used for subsequent look-up. These tables consist of the values of univariate integrals of tri-product B-splines with overlapping supports.

- Following spline projection and building look-up tables, the last step is to calculate the linear combinations of the integrals (18). To this end, we employ the sum-factorization approach to speed up the assembly process.

3.2. Approximating the weight functions W and w

We choose $\operatorname{span}\mathcal{H}$ as the spline space that is used for approximating the weight functions. In the case of tensor-product B-splines [40, 43], it has been observed that this choice guarantees the optimal order of approximation. It will also be shown that the sizes of the look-up tables generated in the next stage are relatively small in this case.

In the tensor-product case, the spline projections (14) and (15) can be easily done by B-spline interpolation at the associated Greville abscissas. A fast way to compute the coefficients $W_{(\ell'',\boldsymbol{i}'')}$ and $w_{(\ell'',\boldsymbol{i}'')}$ of the interpolants is to use de Boor's method [7] which makes full use of the tensor-product structure. Unfortunately, interpolation is not directly applicable in the hierarchical case, since the choice of interpolation nodes is not yet fully understood. Two alternative approaches for tackling this problem are least-squares approximation and quasi-interpolation.

The least-squares approximation to a given function s by a spline function $h \in \operatorname{span}\mathcal{H}$ is found by solving the following optimization problem:

$$\min_{h \in \operatorname{span}\mathcal{H}} \|h - s\|^2_{L^2(\Omega^0)}.$$

To address this problem, we need to assemble the matrices by some quadrature rule (e.g., Gauss quadrature) and solve linear equations. Both steps usually require a large computational effort, especially when the dimension of the space $\operatorname{span}\mathcal{H}$ is high.

Quasi-interpolation is another general approach for constructing accurate approximations of a given function. Compared to least-squares approximation, it requires a lower computational effort. Given a smooth function s, the quasi-interpolant of s in the hierarchical spline space $\operatorname{span}\mathcal{H}$ has the general formulation

$$\mathcal{Q}(s) = \sum_{\ell=0}^{L} \sum_{\boldsymbol{i} \in \mathcal{I}^\ell} \lambda^\ell_{\boldsymbol{i}}(s) \boldsymbol{\beta}^\ell_{\boldsymbol{i}} \tag{19}$$

with linear functionals $\lambda^\ell_{\boldsymbol{i}}$. These functionals can be defined in various ways, but they are usually required to be local, i.e., each $\operatorname{supp}\lambda^\ell_{\boldsymbol{i}}$ is contained in some specific domain of interest. The functionals $\lambda^\ell_{\boldsymbol{i}}(s)$ are often designed either as a linear combination of values of s at suitable points or as an appropriate integral of s, such that the operator (19) possesses certain reproduction properties. Several quasi-interpolants for hierarchical splines [28, 37, 50, 51] with different properties have been established. For a more detailed comparison of various quasi-interpolation approaches, we refer the readers to the previous paper [44], in particular to Table 1.

3.3. Building compact look-up tables

With the help of spline projections, the problem of computing the system matrices S and M is transformed into evaluating the linear combinations of the integrals of tri-product B-splines (18). In order to perform an exact calculation of these integrals, we build three compact look-up tables that are composed of integrals of tri-products of univariate B-splines. More specifically, we rewrite the integrals (18) as

$$\int_{\Omega^0} \frac{\partial \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell}}{\partial \hat{x}_\theta} \frac{\partial \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'}}{\partial \hat{x}_\phi} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \mathrm{d}\hat{\boldsymbol{x}} = \prod_{k=1}^{d} I_{(\ell,i_k),(\ell',i'_k),(\ell'',i''_k)}^{\delta_{\theta k},\delta_{\phi k}}, \quad \int_{\Omega^0} \boldsymbol{\beta}_{\boldsymbol{i}}^{\ell} \boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'} \boldsymbol{\beta}_{\boldsymbol{i}''}^{\ell''} \mathrm{d}\hat{\boldsymbol{x}} = \prod_{k=1}^{d} I_{(\ell,i_k),(\ell',i'_k),(\ell'',i''_k)} \tag{20}$$

with

$$I_{(\ell,i_k),(\ell',i'_k),(\ell'',i''_k)}^{\delta_{\theta k},\delta_{\phi k}} = \int_0^1 \beta_{i_k}^{\ell\ (\delta_{\theta k})} \beta_{i'_k}^{\ell'\ (\delta_{\phi k})} \beta_{i''_k}^{\ell''} \mathrm{d}\hat{x}_k, \quad I_{(\ell,i_k),(\ell',i'_k),(\ell'',i''_k)} = \int_0^1 \beta_{i_k}^{\ell} \beta_{i'_k}^{\ell'} \beta_{i''_k}^{\ell''} \mathrm{d}\hat{x}_k, \tag{21}$$

where $\beta_{i_k}^{\ell\ (\delta_{\theta k})}$ denotes the $\delta_{\theta k}$th-order derivative of the B-spline $\beta_{i_k}^{\ell}$ and $\delta_{\theta k}$ is the Kronecker delta.

Under the assumption that the HB-splines considered in this paper are admissible of class 2, the B-splines $\beta_{i_k}^{\ell}$, $\beta_{i'_k}^{\ell'}$ and $\beta_{i''_k}^{\ell''}$ with non-empty overlapping supports belong to two consecutive levels or to the same level, i.e., the level indices ℓ, ℓ' and ℓ'' differ at most by 1. This is an important property that ensures the small size of the look-up tables built in this section.

As analyzed in the previous paper [44], for uniform B-splines, the tri-product integrals (21) that do not involve boundary B-splines can be derived from the following standardized tri-product integrals

$$\int_{\mathbb{R}} \tilde{\beta}_{[0,1,\dots,p+1]}^{(\mu)}(\hat{x})\ \tilde{\beta}_{[i,i+1,\dots,i+p+1]}^{(\mu')}(\hat{x})\ \tilde{\beta}_{[i',i'+1,\dots,i'+p+1]}^{(\mu'')}(\hat{x})\ \mathrm{d}\hat{x}, \tag{22}$$

$$\begin{aligned}\int_{\mathbb{R}} \tilde{\beta}_{[\frac{i}{2},\frac{i+1}{2},\dots,\frac{i+p+1}{2}]}^{(\mu)}(\hat{x})\ \tilde{\beta}_{[\lfloor\frac{p+1}{2}\rfloor,\lfloor\frac{p+1}{2}\rfloor+1,\dots,\lfloor\frac{p+1}{2}\rfloor+p+1]}^{(\mu')}(\hat{x}) \\ \cdot\tilde{\beta}_{[\frac{i'}{2},\frac{i'+1}{2},\dots,\frac{i'+p+1}{2}]}^{(\mu'')}(\hat{x})\ \mathrm{d}\hat{x}\end{aligned} \tag{23}$$

or

$$\begin{aligned}\int_{\mathbb{R}} \tilde{\beta}_{[\frac{i}{2},\frac{i+1}{2},\dots,\frac{i+p+1}{2}]}^{(\mu)}(\hat{x})\ \tilde{\beta}_{[\lfloor\frac{p+1}{2}\rfloor,\lfloor\frac{p+1}{2}\rfloor+1,\dots,\lfloor\frac{p+1}{2}\rfloor+p+1]}^{(\mu')}(\hat{x}) \\ \cdot\tilde{\beta}_{[i'+\lfloor\frac{p+1}{2}\rfloor,i'+\lfloor\frac{p+1}{2}\rfloor+1,\dots,i'+\lfloor\frac{p+1}{2}\rfloor+p+1]}^{(\mu'')}(\hat{x})\ \mathrm{d}\hat{x},\end{aligned} \tag{24}$$

where $\tilde{\beta}_{\Theta}^{(\mu)}$ represents the μth-order derivative of the B-spline of degree p and with knot vector Θ. In (22) the three B-splines belong the same level; in (23) two of the B-splines are from the finer level, the remaining one belongs to the coarser level; in (24) two of the B-splines belong to the coarser level, the other one is from the finer level. The evaluation of the integrals (21) that involve boundary B-splines can be exactly performed via Gaussian quadrature. This does not compromise the efficiency of our approach, since it only accounts for a small part of the overall computations. These standardized integrals (22), (23) and (24) can be precomputed, constituting three compact tables of size $7(p+1)^2$, $7(2\lfloor 3(p+1)/2\rfloor)^2$ and $7(p+1)(2\lfloor 3(p+1)/2\rfloor)$, respectively. Our method is also applicable to non-uniform B-splines. In this case, the look-up tables for the integrals (21) are built via Gaussian quadrature. Different from the uniform case, the sizes of the look-up tables for non-uniform B-splines not only depend on the polynomial degree p but also on the dimensions of the univariate spline spaces and on the number of levels L. Fortunately, the computational costs for both uniform and non-uniform B-splines are negligible, compared to the overall complexity of the proposed method. For further details, the reader may wish to consult the reference [44].

3.4. Matrix assembly via sum-factorization

When the look-up tables are available, we substitute (20) into (16) and (17), obtaining

$$S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} \approx \sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}''\in\mathcal{I}^{\ell''}} \sum_{\theta=1}^{d}\sum_{\phi=1}^{d} W^{\theta,\phi}_{(\ell'',\boldsymbol{i}'')} \prod_{k=1}^{d} I^{\delta_{\theta k},\delta_{\phi k}}_{(\ell,i_k),(\ell',i'_k),(\ell'',i''_k)} \tag{25}$$

and

$$M_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} \approx \sum_{\ell''=0}^{L} \sum_{\boldsymbol{i}''\in\mathcal{I}^{\ell''}} w_{(\ell'',\boldsymbol{i}'')} \prod_{k=1}^{d} I_{(\ell,i_k),(\ell',i'_k),(\ell'',i''_k)}. \tag{26}$$

Consequently, the elements $S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')}$ and $M_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')}$ can be evaluated as a weighted sum of values derived from the built look-up tables. The whole procedure requires not more than $\mathcal{O}(Np^{2d})$ flops, where N represents the dimension of the space $\mathcal{H}$. This complexity can be easily achieved by suitably adapting the IIL approach [40] to the hierarchical case.

However, the use of sum-factorization leads to a further significant speed-up. The method of sum-factorization is based on an expansion of the quantities (25) and (26) into nested summations, arriving at

$$\begin{aligned} S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} \approx \sum_{\ell''=0}^{L}\sum_{\theta=1}^{d}\sum_{\phi=1}^{d}\sum_{i''_d\in\mathcal{P}^{\ell''}_d} I^{\delta_{\theta d},\delta_{\phi d}}_{(\ell,i_d),(\ell',i'_d),(\ell'',i''_d)} \Bigg[\sum_{i''_{d-1}\in\widetilde{\mathcal{P}}^{\ell''}_{d-1}(i''_d)} I^{\delta_{\theta d-1},\delta_{\phi d-1}}_{(\ell,i_{d-1}),(\ell',i'_{d-1}),(\ell'',i''_{d-1})} \\ \times\cdots\times \Bigg[\sum_{i''_1\in\widetilde{\mathcal{P}}^{\ell''}_1(i''_2,\ldots,i''_d)} W^{\theta,\phi}_{(\ell'',\boldsymbol{i}'')} I^{\delta_{\theta 1},\delta_{\phi 1}}_{(\ell,i_1),(\ell',i'_1),(\ell'',i''_1)} \Bigg]\Bigg] \end{aligned} \tag{27}$$

and

$$\begin{aligned} M_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} \approx \sum_{\ell''=0}^{L}\sum_{i''_d\in\mathcal{P}^{\ell''}_d} I_{(\ell,i_d),(\ell',i'_d),(\ell'',i''_d)} \Bigg[\sum_{i''_{d-1}\in\widetilde{\mathcal{P}}^{\ell''}_{d-1}(i''_d)} I_{(\ell,i_{d-1}),(\ell',i'_{d-1}),(\ell'',i''_{d-1})} \\ \times\cdots\times \Bigg[\sum_{i''_1\in\widetilde{\mathcal{P}}^{\ell''}_1(i''_2,\ldots,i''_d)} w_{(\ell'',\boldsymbol{i}'')} I_{(\ell,i_1),(\ell',i'_1),(\ell'',i''_1)} \Bigg]\Bigg]. \end{aligned} \tag{28}$$

We only describe the procedure of assembling the stiffness matrix using this method, since the mass matrix can be dealt with in a similar way.

For the convenience of presentation, we rewrite the quantities appearing in the brackets of the formula (27) as

$$\begin{aligned} &K^{(1,\theta,\phi)}_{(\ell),(\ell'),(\ell'',i''_1,\ldots,i''_d)} = W^{\theta,\phi}_{(\ell'',\boldsymbol{i}'')}, \\ &K^{(j+1,\theta,\phi)}_{(\ell,i_1,\ldots,i_j),(\ell',i'_1,\ldots,i'_j),(\ell'',i''_{j+1},\ldots,i''_d)} \\ &= \sum_{i''_j \in \widetilde{\mathcal{P}}^{\ell''}_j(i''_{j+1},\ldots,i''_d)} I^{\delta_{\theta j},\delta_{\phi j}}_{(\ell,i_j),(\ell',i'_j),(\ell'',i''_j)} K^{(j,\theta,\phi)}_{(\ell,i_1,\ldots,i_{j-1}),(\ell',i'_1,\ldots,i'_{j-1}),(\ell'',i''_j,\ldots,i''_d)}, \quad j = 1,\ldots,d, \end{aligned} \tag{29}$$

where we introduce the $d+1$ auxiliary tensors $\boldsymbol{K}^{(1,\theta,\phi)}, \ldots, \boldsymbol{K}^{(d+1,\theta,\phi)}$. The index tuple $(\ell'', i''_{j+1}, \ldots, i''_d)$ corresponds to the B-splines $(\beta^{\ell''}_{i''_{j+1}}, \ldots, \beta^{\ell''}_{i''_d})$, and the index tuple $(\ell', i'_1, \ldots, i'_j)$ corresponds to the B-splines $(\beta^{\ell'}_{i'_1}, \ldots, \beta^{\ell'}_{i'_j})$.

We obtain the final stiffness matrix approximation from

$$S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} \approx \sum_{\ell''=0}^{L} \sum_{\theta=1}^{d} \sum_{\phi=1}^{d} K^{(d+1,\theta,\phi)}_{(\ell,i_1,\ldots,i_d),(\ell',i'_1,\ldots,i'_d),(\ell'')} .$$

3.5. Algorithms for stiffness matrix assembly

We initialize $\boldsymbol{K}^{(1,\theta,\phi)}$ with the quasi-interpolation coefficients with respect to the weight function W. In the subsequent steps, the stiffness matrix S is assembled in a recursive way, using d loops, for $j = 1, \ldots, d$, each loop consists of two steps:

Firstly, an index set for the evaluation of the tensor $\boldsymbol{K}^{(j+1,\theta,\phi)}$ is computed. Since we know which elements of $\boldsymbol{K}^{(j+1,\theta,\phi)}$ are potentially nonzero, the index tuples of these elements can be identified. We collect them in the index set $\mathcal{J}^{(j)}$. These index tuples

$$(\ell, i_1, \ldots, i_j, \ell', i'_1, \ldots, i'_j, \ell'', i''_{j+1}, \ldots, i''_d)$$

are characterized by the following conditions:

- $\exists i_{j+1}, \ldots i_d$ such that $(i_1, \ldots, i_d) \in \mathcal{I}^\ell$,
- $|\ell - \ell'| < 2, \ |\ell - \ell''| < 2$,
- $\mathrm{supp}(\beta^\ell_{i_1}\beta^{\ell'}_{i'_1}) \neq \emptyset, \ldots, \mathrm{supp}(\beta^\ell_{i_j}\beta^{\ell'}_{i'_j}) \neq \emptyset$,
- $\mathrm{supp}(\beta^\ell_{i_{j+1}}\beta^{\ell''}_{i''_{j+1}}) \neq \emptyset, \ldots, \mathrm{supp}(\beta^\ell_{i_d}\beta^{\ell''}_{i''_d}) \neq \emptyset$,
- $i'_1 \in \mathcal{P}^{\ell'}_1, \ldots, i'_j \in \mathcal{P}^{\ell'}_j$, and
- $i''_{j+1} \in \mathcal{P}^{\ell''}_{j+1}, \ldots, i''_d \in \mathcal{P}^{\ell''}_d$.

As we shall see later, the size of the index set $\mathcal{J}^{(j)}$ is bounded by $\mathcal{O}(Np^d)$.

Secondly, with the index set $\mathcal{J}^{(j)}$ at hand, we are able to evaluate all the potentially nonzero elements

$$\begin{aligned} &K^{(j+1,\theta,\phi)}_{(\ell,i_1,\ldots,i_j),(\ell',i'_1,\ldots,i'_j),(\ell'',i''_{j+1},\ldots,i''_d)} \\ &\qquad = \sum_{i''_j \in \widetilde{\mathcal{P}}^{\ell''}_j(i''_{j+1},\ldots,i''_d)} I^{\delta_{\theta j},\delta_{\phi j}}_{(\ell,i_j),(\ell',i'_j),(\ell'',i''_j)} K^{(j,\theta,\phi)}_{(\ell,i_1,\ldots,i_{j-1}),(\ell',i'_1,\ldots,i'_{j-1}),(\ell'',i''_j,\ldots,i''_d)} \end{aligned}$$

of the sparse tensor $\boldsymbol{K}^{(j+1,\theta,\phi)}$. It suffices to visit all the indices

$$(\ell, i_1, \ldots, i_j, \ell', i'_1, \ldots, i'_j, \ell'', i''_{j+1}, \ldots, i''_d) \in \mathcal{J}^{(j)}$$

and to identify all the indices (ℓ'', i''_j) that satisfy

$$\operatorname{supp}(\beta^{\ell}_{i_j} \beta^{\ell'}_{i'_j} \beta^{\ell''}_{i''_j}) \neq \emptyset$$

for the summation in (29).

The entire procedure is summarized in the pseudocode listed below. Note that we use the notation

$$\mathcal{J} \cup\!= \{\Xi\} \quad \text{as an abbreviation for the assignment} \quad \mathcal{J} = \mathcal{J} \cup \{\Xi\} \ ,$$

where Ξ is some index tuple. This notation generalizes the addition assigment operator += to sets.

```
**Loop no.** $j$
Firstly, we generate the index set $\mathcal{J}^{(j)}$. Note that each index tuple may be added more than once, but only one instance is kept!
$\mathcal{J}^{(j)} = \emptyset$                                                    ▷ index set initialization
for $\ell = 0$ to $L$ do
    for $\boldsymbol{i} \in \mathcal{I}^{\ell}$ do
        for $\ell' = \max(0, \ell-1)$ to $\min(\ell+1, L)$ do
            for $i'_1 \in \mathcal{M}^{\ell'}_1(\ell, i_1), \ldots, i'_j \in \mathcal{M}^{\ell'}_j(\ell, i_j)$ do      ▷ $j$ loops for $i'_1, \ldots, i'_j$
                for $\ell'' = \max(0, \ell-1)$ to $\min(\ell+1, L)$ do
                    for $i''_{j+1} \in \mathcal{M}^{\ell''}_{j+1}(\ell, i_{j+1}), \ldots, i''_d \in \mathcal{M}^{\ell''}_d(\ell, i_d)$ do      ▷ $d-j$ loops for $i''_{j+1}, \ldots, i''_d$
                        $\mathcal{J}^{(j)} \cup\!= \{(\ell, i_1, \ldots, i_j, \ell', i'_1, \ldots, i'_j, \ell'', i''_{j+1}, \ldots, i''_d)\}$      ▷ index set creation

Secondly, we use the index set to evaluate the elements of the auxiliary tensor $\boldsymbol{K}^{(j+1,\theta,\phi)}$:
for $(\ell, i_1, \ldots, i_j, \ell', i'_1, \ldots, i'_j, \ell'', i''_{j+1}, \ldots, i''_d) \in \mathcal{J}^{(j)}$ do      ▷ loop through index set
    for $\theta = 1$ to $d$ do
        for $\phi = 1$ to $d$ do
            $K^{(j+1,\theta,\phi)}_{(\ell,i_1,\ldots,i_j),(\ell',i'_1,\ldots,i'_j),(\ell'',i''_{j+1},\ldots,i''_d)} = 0$      ▷ initialization
            for $i''_j \in \mathcal{N}^{\ell''}_j(\ell, i_j) \cap \mathcal{N}^{\ell''}_j(\ell', i'_j) \cap \widetilde{\mathcal{P}}^{\ell''}_j(i''_{j+1}, \ldots, i''_d)$ do      ▷ loop through summation index
                $K^{(j+1,\theta,\phi)}_{(\ell,i_1,\ldots,i_j),(\ell',i'_1,\ldots,i'_j),(\ell'',i''_{j+1},\ldots,i''_d)}$ +=
                        $I^{\delta_{\theta j},\delta_{\phi j}}_{(\ell,i_j),(\ell',i'_j),(\ell'',i''_j)} K^{(j,\theta,\phi)}_{(\ell,i_1,\ldots,i_{j-1}),(\ell',i'_1,\ldots,i'_{j-1}),(\ell'',i''_j,\ldots,i''_d)}$      ▷ summation
```

It should be emphasized that the matrix-assembly approach proposed in this section is not a trivial extension of the method developed for the bivariate case [44]. As described above, in order to guarantee the expected complexity of $\mathcal{O}(p^{d+1})$ per degree of freedom, in the first stage of the assembly procedure, we generate an index set $\mathcal{J}^{(j)}$ for storing the indices of the potentially nonzero elements of $\boldsymbol{K}^{(j+1,\theta,\phi)}$ that are needed for evaluating the next tensor $\boldsymbol{K}^{(j+2,\theta,\phi)}$. The second stage evaluates the nonzero elements of $\boldsymbol{K}^{(j+1,\theta,\phi)}$ by summing over the indices i''_j. The complexity analysis is provided in the following section.

4. Computational costs

In this section, the detailed complexity analysis of assembling the system matrices by Gaussian method and the proposed method is given. We focus on the stiffness matrix, in order to keep the presentation concise. In fact, the computational cost of assembling the mass matrices has the same order of magnitude.

When using *Gaussian quadrature*, an efficient implementation includes two steps. The first step is pre-computing and storing the values of the weight function W and derivatives of the B-splines at each Gauss node. As reported in [40, 43], the cost of this step does not dominate the total complexity. The remaining task (the core operation) is the evaluation of the local stiffness matrix with at most $2(p+1)^d \times 2(p+1)^d$ entries element by element. Each element involves $\mathcal{O}(p^{2d})$ loops at each of the $\mathcal{O}(p^d)$ Gauss nodes, since there exist $\mathcal{O}(p^d)$ B-splines that do not vanish at this node. Thus the total computational cost of Gaussian quadrature is $\mathcal{O}(p^{3d})$ per degree of freedom.

In the matrix-assembly stage of the *new method*, we need to precompute various index sets, such as $\mathcal{M}_j^{\ell'}(\ell, i_j)$, $\mathcal{N}_j^{\ell'}(\ell, i_j)$ and $\mathcal{P}_j^{\ell'}$ for any given B-spline $\beta_{i_j}^{\ell}$. Also, we need to compute the index sets $\widetilde{\mathcal{P}}_j^{\ell''}(i''_{j+1}, \ldots, i''_d)$ which appear in the summation of Eq. (29). Unlike the tensor-product case, these operations for HB-splines incur certain computational costs. In addition, the assembly process needs a substantial number of sparse matrix accessing operations, which also require some time. However, these costs exist in any assembly approach for HB-splines, and they depend heavily on the data structure used in the implementation. In order to facilitate the complexity analysis, we make the following assumption:

Assumption 3. The time needed to compute the index sets and to perform the sparse matrix accessing operations is negligible compared to the total computational cost of the presented approach.

Under this assumption, we present the main result of the paper:

Theorem 4. *When solving the problem* (5) *using isogeometric discretizations with admissible HB-splines of class 2, the total computational cost of the proposed algorithm for the formation of stiffness matrices* (12) *amounts to* $\mathcal{O}(p^{d+1})$ *flops per degree of freedom, where* p *represents the spline degree.*

Proof. Let us recall the presented approach. It consists of three parts: spline projection, building look-up tables and formation of the system matrices.

Spline quasi-interpolation is a common and powerful tool devoted to the constructions of accurate approximations to a given function. So far, several quasi-interpolants with different properties, including interpolants that work in hierarchical spline spaces, have been created. We adopt the recently invented local HB-spline projector [28] to perform spline projection. It requires $\mathcal{O}(p^d)$ flops per degree of freedom, while preserving the same accuracy as global approximation.

The look-up tables for d-variate ($d > 2$) HB-splines are essentially the same as the ones built for bivariate case. Based on the analysis in [44, Theorem 1], we already know that the complexity of this step, both for uniform and non-uniform B-splines, does not significantly increase the total computational cost.

For studying the stage of matrix assembly, we first note that the total cardinality of the index sets from the at most three adjacent levels satisfies

$$\sum_{\ell'=\max(0,\ell-1)}^{\min(\ell+1,L)} |\mathcal{M}_j^{\ell'}(\ell, i_j)| \leq \sum_{\ell'=\max(0,\ell-1)}^{\min(\ell+1,L)} |\mathcal{N}_j^{\ell'}(\ell, i_j)| = \mathcal{O}(p)$$

under the admissibility assumption for HB-splines, according to Lemma 2. Therefore, the j-th loop iterates over

$$N \cdot \underbrace{\mathcal{O}(p) \cdot \ \cdots \ \cdot \mathcal{O}(p)}_{j \text{ loops for } i'_1,\ldots,i'_j} \cdot \underbrace{\mathcal{O}(p) \cdot \ \cdots \ \cdot \mathcal{O}(p)}_{d-j \text{ loops for } i''_{j+1},\ldots,i''_d}$$

index tuples $(\ell, i_1, \ldots, i_d, \ell', i'_1, \ldots, i'_j, \ell'', i''_{j+1}, \ldots, i''_d)$, omits $(\ell, i_{j+1}, \ldots, i_d)$ and adds the remainder of each tuple to the index set $\mathcal{J}^{(j)}$. Note that each tuple may be added more than once, but only one copy is kept in the index set! Indeed, a set stores at most one instance of each element. Thus the size of the resulting index set $\mathcal{J}^{(j)}$ amounts to $\mathcal{O}(Np^d)$.

The evaluation of the tensor $\boldsymbol{K}^{(j+1,\theta,\phi)}$ proceeds by visiting $\mathcal{O}(p)$ indices i''_j for each of the index tuples in the index set $\mathcal{J}^{(j)}$, and requires two flops per term. Summing up, the overall cost of the presented approach is equal to $\mathcal{O}(p^{d+1})$ per degree of freedom. □

5. Numerical results

We implemented the new algorithm in C++ using the G+Smo library for IgA [35] and tested its performance on an elliptic problem on the three-dimensional domain and a parabolic problem on the four-dimensional domain in space-time, respectively. All the experiments were done on a laptop computer with an Intel Core i5-7300HQ CPU (2.5GHz). In addition, each test was repeated 10 times and we choose the average value to make the result more accurate.

Firstly, we verify the theoretical complexity of the current approach summarized in Theorem 4. This is done by studying the dependence of the computational time (the number of flops) on the number of degrees of freedom and by investigating the dependence of the computational time (the number of flops) on the spline degree. Secondly, the convergence behaviour of the new approach is demonstrated by solving the PDEs with an adaptive refinement strategy. Finally, a comparison with the standard Gaussian quadrature in terms of the computational time and required number of flops is provided to demonstrate the superiority of our method.

In the experiments, we do not take the time needed for spline projection into account, since the implementation of this stage has not yet been fully optimized. Instead we show the required number of flops (which confirms the computational complexity with repect to p) for the quasi-interpolant [28] in the elliptic model problem (see Figure 7(b)). It should be noted that the total complexity of our approach is dominated by the last stage – matrix assembly via sum-factorization. As mentioned before, the computational cost of building the look-up tables is negligible with respect to the overall complexity. Therefore, the computational time and the number of flops reported in this section (besides Figure 7) only include the ones required for assembling the matrices via sum-factorization. In addition, the numbers of flops needed by some black-box operations (e.g., filling in a sparse matrix) – which are available in the G+Smo library – are not obvious, hence we do not take these numbers into account.

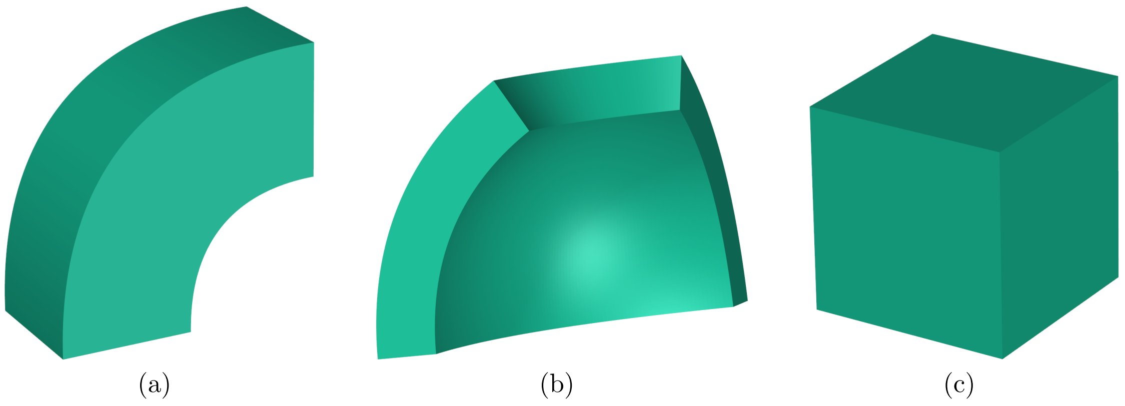


Figure 1: Single-patch B-spline volumes used for testing the performance of different methods. The left one of degree $\boldsymbol{p} = (1, 2, 1)$ is defined by $2 \times 3 \times 2$ control points, the middle one of degree $\boldsymbol{p} = (2, 2, 1)$ is defined by $3 \times 3 \times 2$ control points, and the right one of degree $\boldsymbol{p} = (1, 1, 1)$ is defined by $2 \times 2 \times 2$ control points.

5.1. *$d = 3$: Elliptic problem on a three-dimensional domain*

As a first example it is shown the use of the new approach for the elliptic problem

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ \quad\; u = g & \text{on } \partial\Omega. \end{cases} \tag{30}$$

on the bounded domain $\Omega \in \mathbb{R}^3$, which is defined in (5) with $A = I$ and $\sigma = 0$.

5.1.1. *Complexity with respect to the number of degrees of freedom*

In the first test, we experimentally investigate the dependence of the computational cost required by the new algorithm on the number of degrees of freedom. The test was performed on the single-patch domain depicted in Figure 1(a) using seven HB-spline bases of various degrees ($p = 2, 3, 4$). For each p, these bases have different numbers of degrees of freedom.

Starting from an initial tensor-product mesh with $16\times16\times16$ cells, the i-th $(i=0,\dots,6)$ hierarchical mesh is generated by inserting a box of level 1 and a box of level 2 into the bottom left corner of Ω^0. The boxes of level 1 and level 2 with respect to the ith mesh contain $(12+2i)\times(12+2i)\times(12+2i)$ and $P(i)\times P(i)\times P(i)$ cells respectively, where the function $P(i)$ is defined as

$$P(i)=\begin{cases}8+2i & i\le 3\\ 14 & i>3.\end{cases}$$

Figure 2 illustrates the fourth $(i=3)$ mesh. These meshes are guaranteed to be admissible of class 2 for spline degrees varying from 2 to 4.

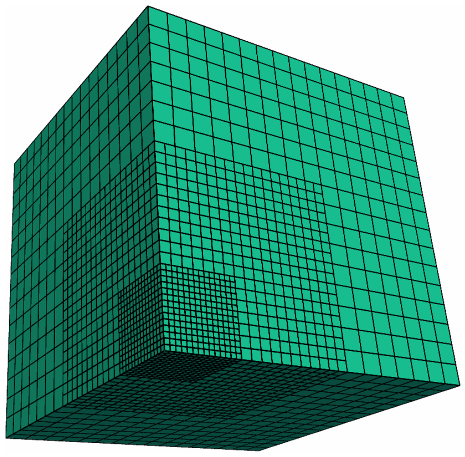


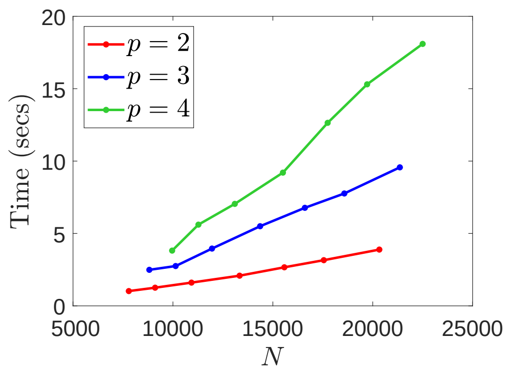
Figure 2: The fourth hierarchical mesh used for investigating the dependence of the computational cost on the number of degrees of freedom.

Figure 3 reports the computing time needed for assembling the stiffness matrices with respect to different HB-splines and degrees p. As already noted in Section 4, the sparse matrix accessing operations and computing the index sets $\mathcal{M}_j^{\ell'}(\ell,i_j)$, $\mathcal{N}_j^{\ell'}(\ell,i_j)$, $\mathcal{P}_j^{\ell'}$, $\widetilde{\mathcal{P}}_j^{\ell''}(i''_{j+1},\dots,i''_d)$ take up a certain portion of the total time. This has an unavoidable effect on the numerical performance. However, this does not have much impact on the overall behaviour. Figure 3 indicates that the computational time scales roughly linearly with the number of degrees of freedom, and this is consistent with Theorem 4.

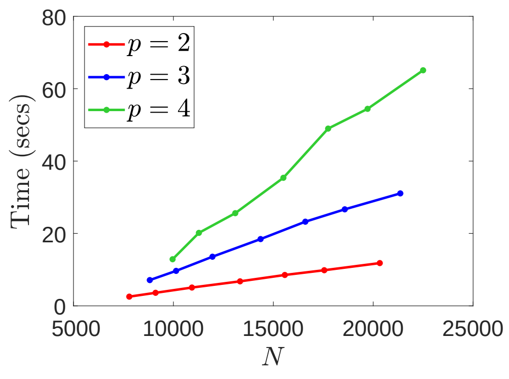
We also measure the cost of the matrix assembly algorithm by counting the number of flops, which excludes the computational effort needed by the accessing operations and computing the index sets (Assumption 3 is satisfied in this case). The result shown in Figure 4 again reveals that the computing cost of the assembly algorithm depends linearly on the number of degrees of freedom. It also indicates that the last loop dominates the complexity of the overall procedure from the perspective of experimental results.

5.1.2. Complexity with respect to spline degrees

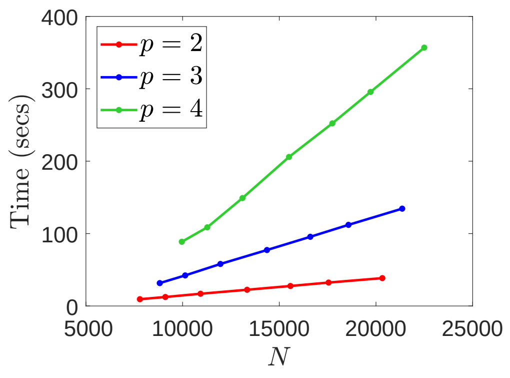
Next we study the relation between the complexity and spline degrees. We assemble the stiffness matrices on the single-patch domain depicted in Figure 1(b) using HB-spline



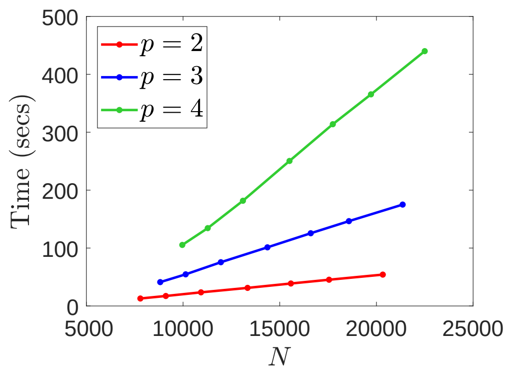
(a) Loop 1



(b) Loop 2



(c) Loop 3



(d) All loops

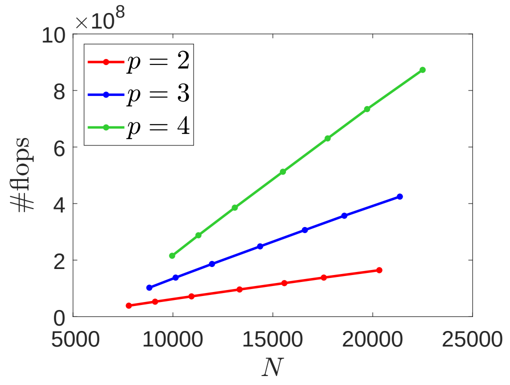
Figure 3: N-dependence of the computational time for assembling the stiffness matrices. The computational time of each loop and all loops are demonstrated.

bases with varying degrees ($p = 1, \dots, 7$). The associated meshes are constructed as follows: They are initialized with a tensor-product mesh consisting of $4p \times 4p \times 4p$ cells, and the hierarchical mesh of degree p is generated by repeatedly refining the up right $2p \times 2p \times 2p$ cells of level ℓ ($\ell = 0, 1, 2$). Clearly, these HB-splines are admissible of class 2. An example of degree 2 is depicted in Figure 5.

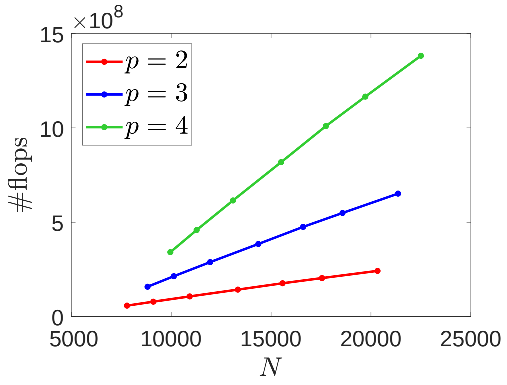
In Figure 6, which uses doubly logarithmic plot, we highlight the dependence of the computation time on the degree p by reference dotted lines. Clearly, the observed growth rates do not yet reach the theoretically predicted asymptotic values for large-enough degrees, where the rate for Gaussian quadrature should be 9 and the rate for our method should be 4.

We also explore the relation between complexity and spline degree by counting the number of flops needed by the spline projection (via the quasi-interpolant [28]) and the assembly algorithm. As described in [44], the quasi-interpolant is implemented via coefficient functionals that perform linear combinations of sampled values. The computational complexity depends on the number thereof (that does not depend on p), and on the number of flops needed to evaluate the coefficient functionals (which has complexity $\mathcal{O}(p^3)$ per degree of freedom). The latter number varies between

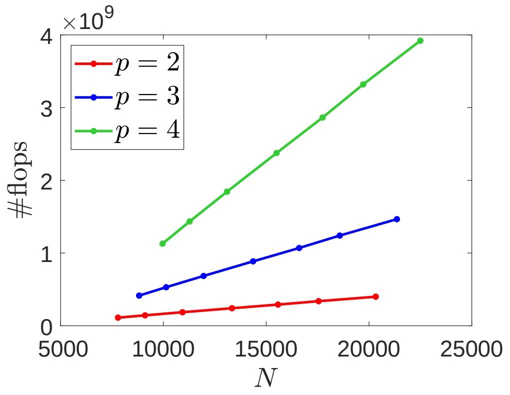
$$2(2p+3)^3 \quad \text{and} \quad 2(6p+7)^3$$



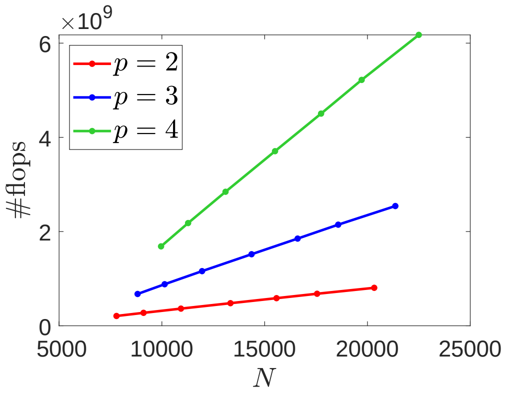
(a) Loop 1



(b) Loop 2



(c) Loop 3



(d) All loops

Figure 4: N-dependence of the number of flops needed for assembling the stiffness matrices. The number of each loop and all loops are shown.

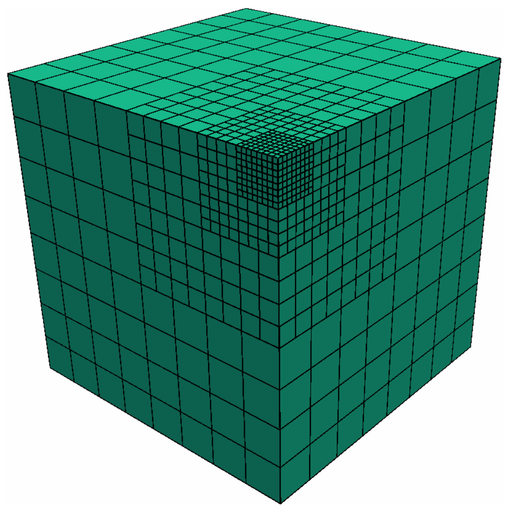
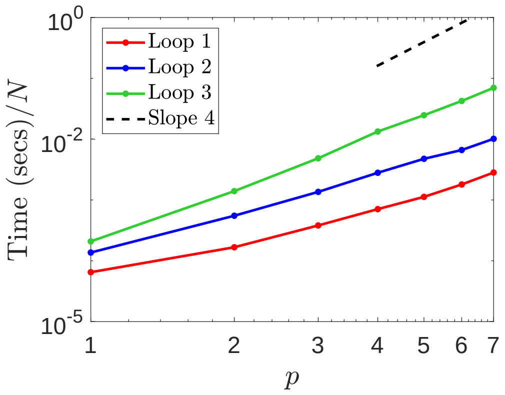
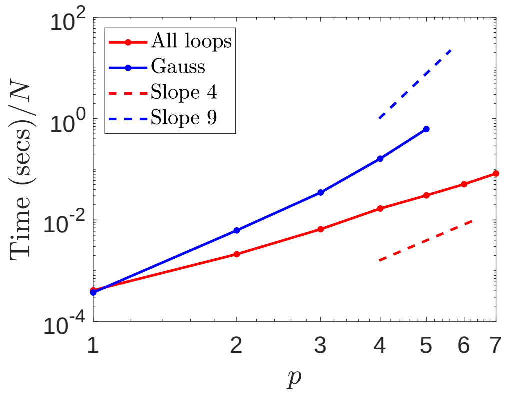


Figure 5: A hierarchical mesh of degree 2 used for verifying the p-dependence of the assembly time.



(a)

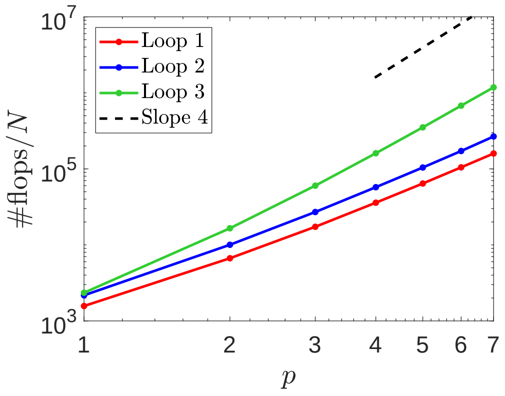


(b)

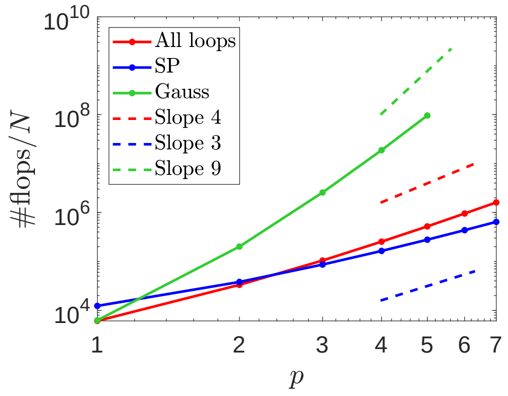
Figure 6: The relation between the computational time and the spline degrees. (a) The computational time of each loop of the new approach. (b) The computational time of Gaussian quadrature and all loops of the proposed algorithm. Here we divide the time by N to eliminate the influence of different numbers of degrees of freedom.

per HB-spline coefficient, depending on the local configuration of the mesh.

Figure 7 reports the numbers of flops per degree of freedom for various spline degrees needed by the new assembly algorithm, and it also includes the corresponding numbers for Gaussian quadrature and the spline projection via the local-fitting based quasi-interpolant [28]. The plots support the theoretical results concerning the computational complexity. It also shows that the spline projection requires a non-negligible computational effort, especially for low degrees ($p = 1, 2$), even if its computational complexity ($\mathcal{O}(p^3)$ flops per degree of freedom) is of lower order than that of the assembly algorithm.



(a)



(b)

Figure 7: The relation between the required number of flops and the spline degrees. (a) The numerical behaviour of each loop of the new method. (b) The cost of spline projection, Gaussian quadrature and all loops of the proposed assembly algorithm. Here we divide the numbers by N to eliminate the influence of different numbers of degrees of freedom.

5.1.3. Adaptive refinement

In this part, we present the results of embedding the new assembly algorithm into an adaptive refinement framework for solving the problem (30) on the domain depicted in Figure 1(a). The exact solution of this problem is given by

$$u_{\text{exact}}(\boldsymbol{x}) = e^{-100((x_1-0.5)^2+(x_2-1.5)^2+(x_3-0.5)^2)} + e^{-100((x_1-1.5)^2+(x_2-0.5)^2+(x_3-0.5)^2)},$$

which has two peaks around the points $(0.5, 1.5, 0.5)$ and $(1.5, 0.5, 0.5)$. Figure 8 shows it on a slice of the computational domain. The source function f can be directly derived from the first equation in (30).

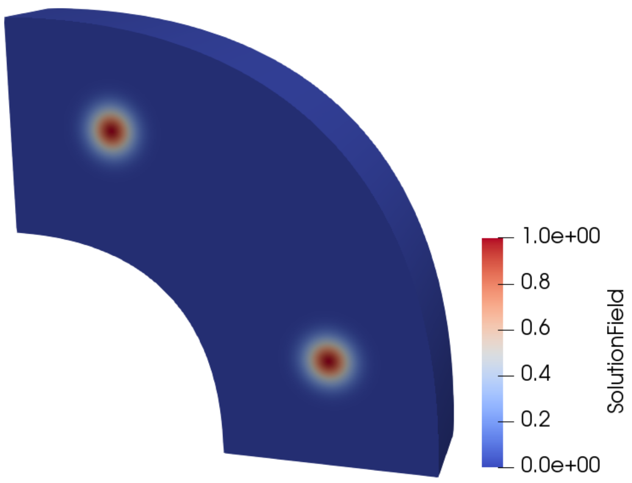


Figure 8: The exact solution on a slice of the physical domain.

Starting from a tensor-product mesh of size $8 \times 8 \times 8$, the adaptive refinement is performed by a marking strategy that is based on the well-known residual-based posteriori error estimate. The estimate ζ_c on the cell $c \in \Omega$ is defined as

$$\zeta_c = h_c ||f + \Delta u_h||_{L^2(c)},$$

where u_h is the computed approximate solution of the problem (30) and h_c denotes the diameter of cell c. A cell is marked for refinement if its error estimate exceeds a certain threshold. In the previous paper [26, Section 4.3], two different strategies with a parameter η for choosing this threshold are introduced. In the experiment, we use the second strategy and set $\eta = 0.25$.

Figure 9 presents a slice of the resulting hierarchical mesh after 4 refinements ($\ell = 4$) using the proposed approach. It can be seen that the adaptive method performs well and generates reasonable refinements around the two peaks.

Figure 10 compares the uniform refinement via Gaussian quadrature and the adaptive refinements via Gaussian quadrature and our method in terms of the convergence rate. The corresponding statistical data is listed in Table 2. As to be expected, the adaptive approach outperforms uniform refinement. It requires fewer degrees of freedom to arrive at a certain

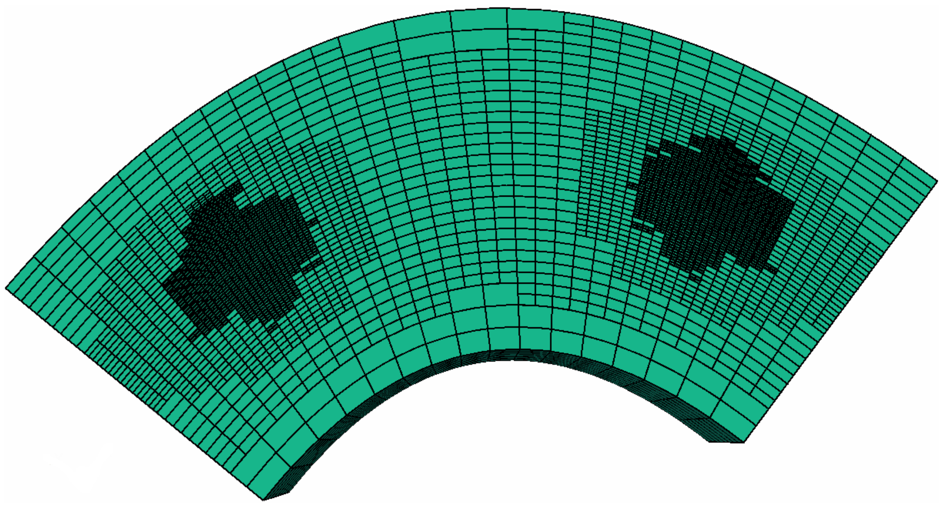
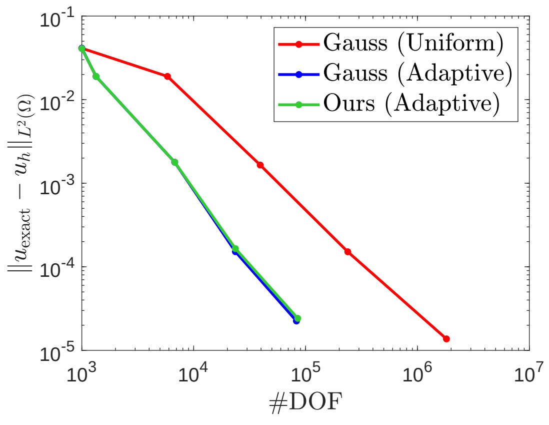
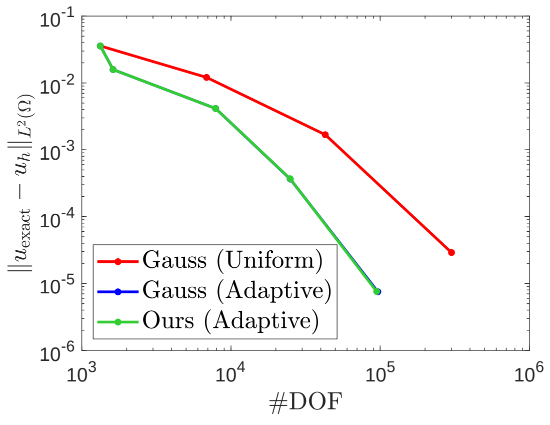


Figure 9: A slice of the hierarchical mesh with $p = 2$ after 4 refinements.



(a) $p = 2$



(b) $p = 3$

Figure 10: Comparison of the convergence behaviour for uniform refinement via Gaussian quadrature and adaptive refinements via Gaussian quadrature and our method.

accuracy. Our new assembly method achieves similar results to Gaussian quadrature with adaptive refinement. In addition, through careful observation, we note that there exists a slight difference in the results of these two approaches. This is mainly caused by the small approximation error incurred in the spline projection stage of the new method. In fact, the spline projection is the only approximation step that concerns the matrix assembly in our approach.

5.1.4. Speedup with respect to Gaussian quadrature

Finally we compare the proposed assembly approach with the classical Gaussian quadrature in terms of the time and required number of flops. As in previous work [40, 43, 44], we choose $p + 1$ nodes in each direction per element for Gaussian quadrature.

Table 3 presents the experimentally observed computational time and numbers of flops

Level ℓ	Uniform refinement		Adaptive refinement			
	Gauss		Gauss		Ours	
	#DOF	L^2 error	#DOF	L^2 error	#DOF	L^2 error
			$p = 2$			
0	1,000	4.12e-2	1,000	4.12e-2	1,000	4.09e-2
1	5,832	1.90e-2	1,340	1.90e-2	1,340	1.90e-2
2	39,304	1.65e-3	6,764	1.79e-3	6,764	1.79e-3
3	238,328	1.51e-4	23,597	1.52e-4	23,583	1.65e-4
4	1,815,848	1.37e-5	82,843	2.25e-5	85,112	2.42e-5
			$p = 3$			
0	1,331	3.58e-2	1,331	3.58e-2	1,331	3.57e-2
1	6,859	1.21e-2	1,622	1.58e-2	1,622	1.58e-2
2	42,875	1.68e-3	7,893	4.15e-3	7,893	4.14e-3
3	300,763	2.90e-5	24,946	3.66e-4	24,932	3.69e-4
4	2,248,091	N/A	96,417	7.53e-6	94,759	7.60e-6

Table 2: The statistical data for the convergence results of uniform refinement via Gaussian quadrature and adaptive refinements via Gaussian quadrature and our method. Here Gaussian quadrature with uniform refinement is too time-consuming for 2,248,091 degrees of freedom and $p = 3$.

for assembling the stiffness matrices via two different methods. The experiment was done on the domain depicted in Figure 1(b) with HB-splines introduced in Section 5.1.2. For higher polynomial degrees ($p > 5$), Gaussian quadrature takes a long time (more than two hours). It can be seen that our approach achieves a significant speedup for degrees larger than 1, ranging from 2.95 to 20.35 with respect to the time and from 6.05 to 184.94 concerning the number of flops. The speedup becomes even more pronounced for higher degrees. Certainly, the speedup achieved for low degrees ($p = 2, 3$) may be more important, since splines of these degrees tend to be widely used in isogeometric applications.

However, there exists an obvious gap between the speedups with respect to the observed time and number of flops (the factors exceed 7 for $p > 3$). This is mainly because the new algorithm contains a large number of unavoidable sparse matrix accessing operations, which greatly influences the efficiency. Nevertheless, the current results still reveal the advantages of our approach.

5.2. $d = 4$: Parabolic problem on a four-dimensional domain in space-time

Parabolic initial-boundary value problems arise in many practical applications, they are often used to describe various physical phenomena such as the heat conduction and diffusion process, the evolution process in life sciences and so on. The typical way for solving these problems is to discretize them in space and time separately. Several papers [38, 39] proposed a discretization scheme in the framework of IgA, which is called *space-time isogeometric discretization*. Simply speaking, this scheme discretizes the problems in both space and time simultaneously, and it treats the time as an additional spatial variable. Consequently, the parabolic problem defined on a $\tilde{d}$-dimensional spatial domain and the time domain is lifted to a $\tilde{d} + 1$-dimensional problem.

Complexity	Degree	#DOF	Gauss	Ours	Gauss/Ours Speedup
Time (secs)	1	293	1.09e-1	1.20e-1	0.91
	2	2,344	1.46e1	4.95e0	2.95
	3	7,911	2.76e2	5.20e1	5.30
	4	18,752	3.04e3	3.16e2	9.64
	5	36,625	2.28e4	1.12e3	20.35
#flops	1	293	1.83e6	1.78e6	1.03
	2	2,344	4.71e8	7.79e6	6.05
	3	7,911	2.03e10	8.26e8	24.57
	4	18,752	3.52e11	4.76e9	73.99
	5	36,625	3.52e12	1.90e10	184.94

Table 3: Comparison of Gaussian quadrature versus our approach in terms of the required time and number of flops for assembling the system matrices.

We consider the heat equation with homogeneous initial-boundary conditions

$$\begin{cases} \partial_t u - \Delta_x u = f & \text{in } \Omega_s \times \Omega_t \\ u = 0 & \text{on } \partial\Omega_s \times \Omega_t \\ u = 0 & \text{in } \Omega_s \times \{0\}, \end{cases} \tag{31}$$

where $\Omega_s \in \mathbb{R}^3$ and $\Omega_t = (0,1)$ are the spatial domain and time domain respectively, and $f \in L^2(0,1,H^{-1}(\Omega_s))$ is a given source function. The detailed derivation of the isogeometric discretizations for this problem can be found in the references [38, 39]. The elements of the corresponding isogeometric Galerkin matrix S have the form

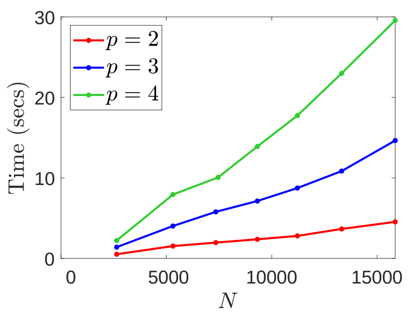
$$S_{(\ell,\boldsymbol{i}),(\ell',\boldsymbol{i}')} = \int_{[0,1]^4} \partial_{\hat{t}}\boldsymbol{\beta}_{\boldsymbol{i}}^{\ell}\boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'}\,|\det J_{\boldsymbol{G}}|\,\mathrm{d}\hat{\boldsymbol{x}}\mathrm{d}\hat{t} + \int_{[0,1]^4} \hat{\nabla}_{\hat{\boldsymbol{x}}}\boldsymbol{\beta}_{\boldsymbol{i}}^{\ell^T} J_{\boldsymbol{G}}^{-1}J_{\boldsymbol{G}}^{-T}\hat{\nabla}_{\hat{\boldsymbol{x}}}\boldsymbol{\beta}_{\boldsymbol{i}'}^{\ell'}\,|\det J_{\boldsymbol{G}}|\,\mathrm{d}\hat{\boldsymbol{x}}\mathrm{d}\hat{t}, \tag{32}$$

where $\boldsymbol{\beta}_{\boldsymbol{i}}^{\ell}$ is a 4-variate basis function with the arguments $(\hat{\boldsymbol{x}},\hat{t})$, and $\boldsymbol{G}$ is the parameterization of the spatial domain Ω_s. We apply the proposed algorithm to assemble the system matrix S (32), and test its numerical performance which is shown in the following sections.

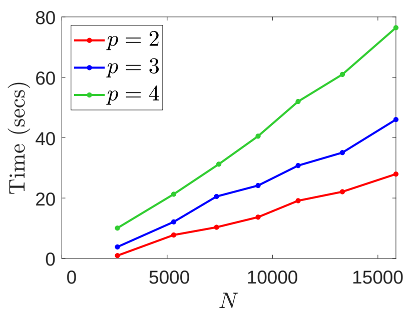
5.2.1. Computational time and number of flops

In the first test, we experimentally investigate the dependence of the computational effort (including the time and number of flops) needed to form the system matrix S on the number of degrees of freedom. The experiment was done on a computational domain using seven HB-splines with three levels for each polynomial degree ($p = 2,3,4$). The spatial part of this computational domain is depicted in Figure 1(a). The HB-spline bases are created in a similar way as described in Section 5.1.1. The plots in Figures 11 and 12 again reveal that the computational cost – with respect to each loop of the assembly algorithm – scales nearly linearly with the number of degrees of freedom. Thereby the total cost of the new algorithm also depends linearly on the number of degrees of freedom.

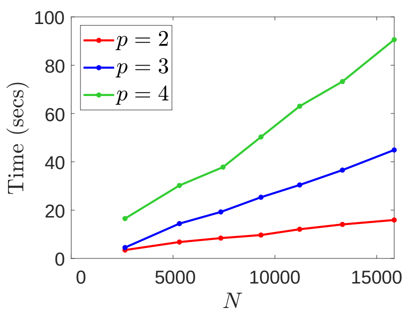
In the second test, we study the growth order of the computational effort relative to the spline degree p. The test was done on a computational domain using a sequence of



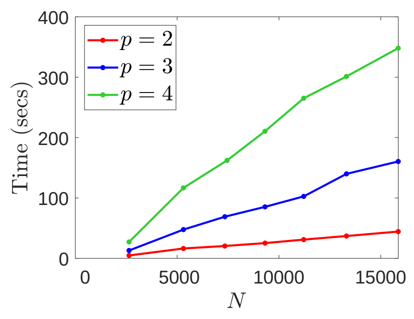
(a) Loop 1



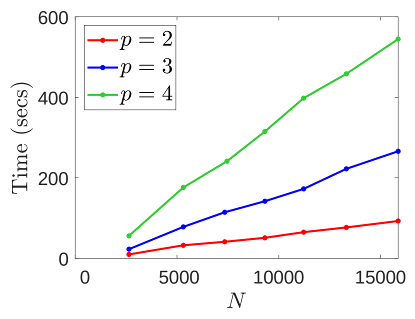
(b) Loop 2



(c) Loop 3

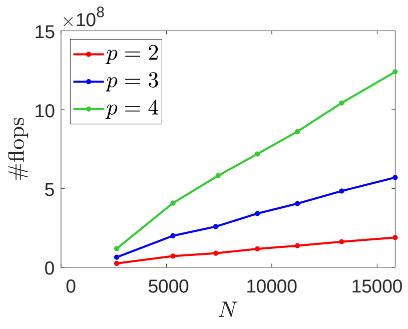


(d) Loop 4

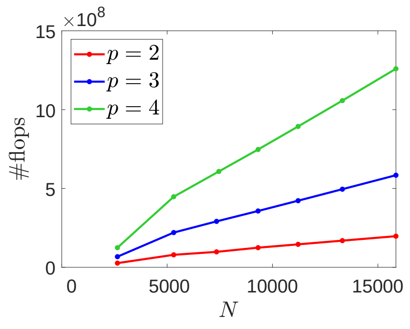


(e) All loops

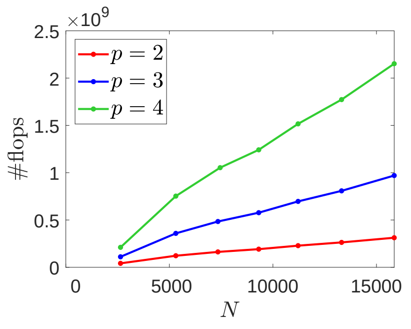
Figure 11: N-dependence of the computational time needed for assembling the stiffness matrices. The time of each loop and all loops are shown.



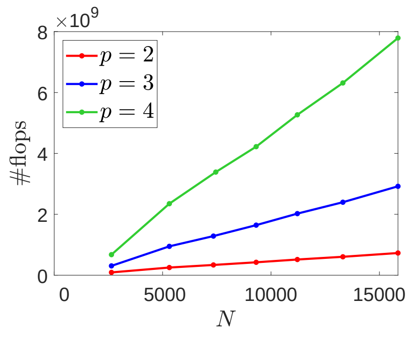
(a) Loop 1, Step 1



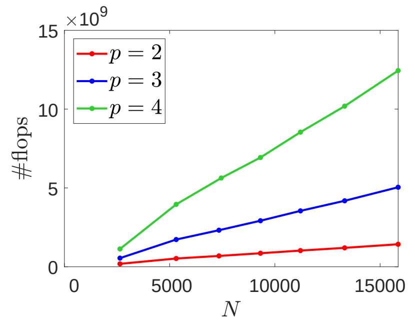
(b) Loop 2



(c) Loop 3



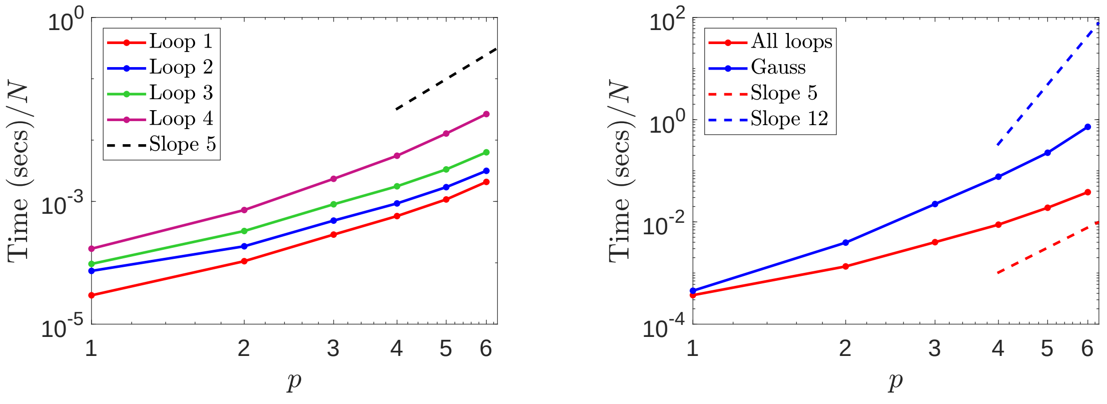
(d) Loop 4



(e) All loops

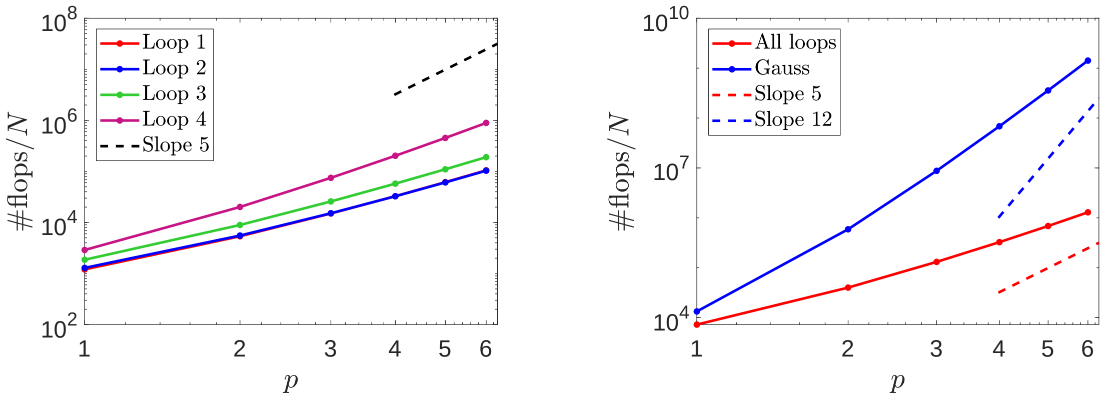
Figure 12: N-dependence of the number of flops needed for assembling the stiffness matrices. The numbers of each loop and all loops are shown.

HB-splines with various degrees ranging from 1 to 6. The spatial part of the computational domain is depicted in Figure 1(b). The HB-spline bases – which possess two levels – are again created in a similar way as described in Section 5.1.1. In Figures 13 and 14, the dependence of the cost needed by the assembly approach on the spline degrees is depicted using a doubly logarithmic plot. As expected, the cost needed by each loop of the proposed algorithm is nearly proportional to p^5. Therefore, the total cost of the new approach evaluates to $\mathcal{O}(p^5)$ per degree of freedom, which is shown in Figure 13(b) and 14(b). This again confirms the results summarized in Theorem 4.



(a) (b)

Figure 13: The relation between the computational time and the spline degrees. (a) The computational time of each loop of the new approach. (b) The assembly time of Gaussian quadrature and the overall procedure of the proposed algorithm. Here we divide the time by N to eliminate the influence of different numbers of degrees of freedom.



(a) (b)

Figure 14: The relation between the required number of flops and the spline degrees. (a) The numerical behaviour of each loop of the new method. Note that the computational costs of the first two loops are almost the same. (b) The cost of Gaussian quadrature and all loops of the proposed assembly algorithm. Here we divide the numbers by N to eliminate the influence of different numbers of degrees of freedom.

5.2.2. *Convergence behaviour*

Next we assess the convergence behaviour of the presented method by solving the problem (31) in both uniform and adaptive refinement modes, where the spatial domain Ω_s is chosen as a unite cube displayed in Figure 1(c), i.e., we have the computational domain $\Omega := \Omega_s \times \Omega_t = (0,1)^4$ which can be geometrically represented by tensor-product B-splines of degree 1 and of size $2 \times 2 \times 2 \times 2$.

The exact solution is defined by a sharp local Gaussian distribution

$$u_{\text{exact}}(\boldsymbol{x},t) = \frac{x_1(1-x_1)x_2(1-x_2)x_3(1-x_3)t(1-t)}{\exp(\sqrt{(5x_1-2.5)^2+(5x_2-2.5)^2+(5x_3-2.5)^2+(5t-2.5)^2})}$$

which has a peak located at the point $(\boldsymbol{x},t) = (0.5, 0.5, 0.5, 0.5)$. Then the right-hand side f is computed by substituting u_{exact} into the equation $\partial_t u - \Delta_x u = f$. The homogeneous initial-boundary conditions of the problem (31) are obviously satisfied.

We firstly investigate the convergence rate of the proposed approach by performing a series of uniform refinements and by using B-splines of degrees $p = 1, 2$. Table 4 reports the errors and the order of convergence (obtained via the dyadic logarithm of the ratio of adjacent errors) with respect to the L^2 norm. We observe that the optimal convergence rate $\mathcal{O}(h^{p+1})$ is achieved, although this is not implied by existing theoretical results, as mentioned in [39, Section 5.1].

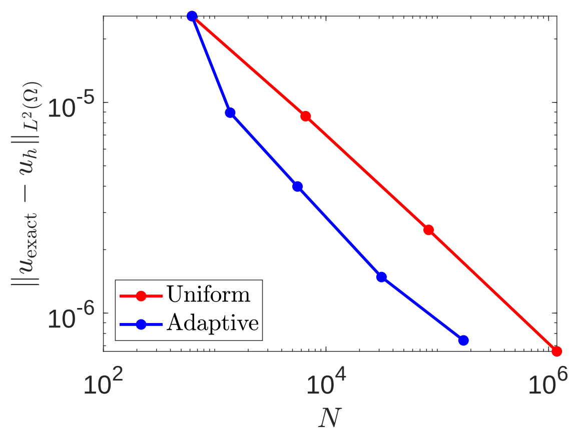
Degree	h	N	L^2 error	Order
$p=1$	0.25	625	2.57e-05	–
	0.125	6,561	8.60e-06	1.58
	0.0625	83,521	2.48e-06	1.79
	0.03125	1,185,921	6.58e-07	1.92
$p=2$	0.5	256	1.30e-04	–
	0.25	1,296	4.29e-05	1.60
	0.125	10,000	6.75e-06	2.67
	0.0625	104,976	8.95e-07	2.91
	0.03125	1,336,336	1.01e-07	3.15

Table 4: Accuracy test for uniform refinement by the proposed method.

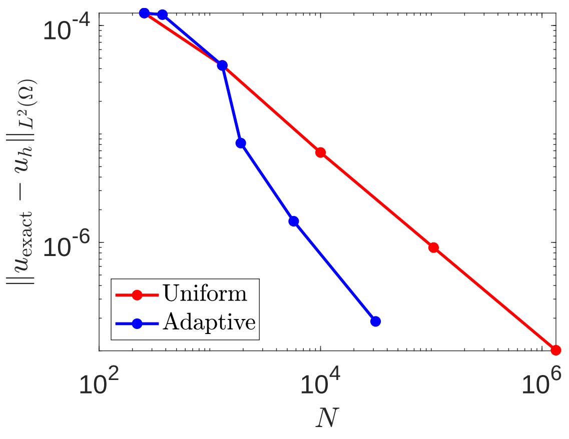
Second, an adaptive refinement strategy guided by error estimates is employed to solve the parabolic problem (31). We start with initial tensor-product B-splines, and continue with several adaptive refinement steps based on a residual-based posteriori error estimator. Assume we have computed an approximate solution u_h of the problem (31), the error on a local cell c is defined as

$$h_c|| - \partial_t u_h + \Delta_x u_h + f||_{L^2(c)},$$

where h_c represents the diameter of the four-dimensional cell c. During the refinement process, marking the elements in Ω is driven by the criterion with a relative threshold $\eta = 0.25$, which is discussed in Section 5.1.3. Figure 15 shows the comparison between uniform refinement and adaptive refinement in terms of the convergence rate, which confirms that adaptive refinement requires fewer degrees of freedom to achieve a certain accuracy.



(a) $p = 1$



(b) $p = 2$

Figure 15: Convergence plots for uniform refinement by our method and adaptive refinement via our method.

5.2.3. Comparison with Gaussian quadrature

In order to demonstrate the superiority of the proposed method, we compare it with the standard Gaussian quadrature by counting the computational time and number of flops required to build the system matrices. As in the 3D tests, we take $p+1$ nodes for Gaussian method in each coordinate direction per element. Table 5 lists the statistical data for the test on HB-splines with degrees up to 6. Here the chosen computational domain and HB-splines are the same as the ones used in Section 5.2.1 for verifying the degree-dependence behaviour. As expected, our approach significantly outperforms Gaussian quadrature in terms of the computational cost, and the advantages become even more obvious as the spline degree increases. This can also be observed in Figures 13(b) and 14(b). However, similar to the 3D case, we also note that there is an unavoidable gap (ranging from 1.51 to 57.97) between the experimental speedups and the predicted ones, which is mainly caused by the implementation aspects.

6. Conclusions and future work

In the previous works [43, 44], it has been shown that the combination of (quasi-) interpolation, looking-up and sum-factorization techniques is an effective way to assemble the isogeometric Galerkin matrices. However, extending these approaches to d-variate ($d > 2$) HB-splines is not a trivial task. In this paper, we introduced a three-stage method that involves these techniques for the formation of system matrices with respect to HB-splines in any dimension. Different from the bivariate case [44], in the first stage of the current assembly procedure, we generate an index set that is needed in the second stage of the algorithm.

The theoretical analysis demonstrated that the new approach maintains the order of the complexity of the method for tensor-product B-splines [43] in any dimension, i.e., the assembly takes $\mathcal{O}(p^{d+1})$ flops per degree of freedom (without taking sparse matrix operations into account), provided that the hierarchical mesh respects the admissibility assumption. We

Computational cost	Degree	#DOF	Gauss	Ours	Speedup (Gauss/Ours)
Time (s)	1	271	1.22e-1	1.00e-1	1.22
	2	640	2.22e0	8.62e-1	2.58
	3	1,311	2.93e1	5.26e0	5.57
	4	2,416	1.85e2	2.14e1	8.64
	5	4,111	9.27e2	7.78e1	11.92
	6	6,576	4.76e3	2.51e2	18.96
#flops	1	271	3.60e6	1.96e6	1.84
	2	640	3.77e8	2.56e7	14.73
	3	1,311	1.15e10	1.72e8	66.86
	4	2,416	1.65e11	7.86e8	209.92
	5	4,111	1.47e12	2.82e9	521.28
	6	6,576	9.32e12	8.48e9	1099.06

Table 5: Comparison of Gaussian quadrature with our approach in terms of the required time and number of flops for assembling the system matrices.

performed several numerical experiments to verify the theoretical results. The comparison with Gaussian quadrature in terms of the computational cost reveals that our approach has a significant advantage.

Future work will be devoted to two aspects. First, THB-splines [26, 27], which are modified from HB-splines via the truncation mechanism, play an important role in adaptive isogeometric analysis and lead to discretizations with improved numerical properties. Exploring the fast formation of the matrices arising from isogeometric discretizations with THB-splines is an interesting topic. Second, we will try to further optimize the implementation such that the gap between the experimentally observed and predicted speedup (see Tables 3 and 5) is reduced.

Acknowledgement

The authors express their appreciation for supports provided by the Austrian Science Fund (Project No. S11708) and by the European Research Council through the CHANGE project (GA No. 694515). Maodong Pan was also supported by the Natural Science Foundation of China (No. 61972368, 12101308). This support is gratefully acknowledged.

References

[1] P. Antolin, A. Buffa, F. Calabrò, M. Martinelli, and G. Sangalli. Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization. *Computer Methods in Applied Mechanics and Engineering*, 285:817–828, 2015.

[2] F. Auricchio, F. Calabrò, T.J.R. Hughes, A. Reali, and G. Sangalli. A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 249:15–27, 2012.

[3] F. Auricchio, L. Beirão da Veiga, T.J.R. Hughes, A. Reali, and G. Sangalli. Isogeometric collocation methods. *Mathematical Models and Methods in Applied Sciences*, 20(11):2075–2107, 2010.

[4] M. Bartoň and V.M. Calo. Optimal quadrature rules for odd-degree spline spaces and their application to tensor-product-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 305:217–240, 2016.

[5] M. Bartoň and V.M. Calo. Gauss–Galerkin quadrature rules for quadratic and cubic spline spaces and their application to isogeometric analysis. *Computer-Aided Design*, 82:57–67, 2017.

[6] M. Bartoň, V. Puzyrev, Q. Deng, and V. Calo. Efficient mass and stiffness matrix assembly via weighted Gaussian quadrature rules for B-splines. *Journal of Computational and Applied Mathematics*, 371:112626, 2020.

[7] C. De Boor. Efficient computer manipulation of tensor products. *ACM Transactions on Mathematical Software (TOMS)*, 5(2):173–182, 1979.

[8] C. Bracco, C. Giannelli, and R. Vázquez. Refinement algorithms for adaptive isogeometric methods with hierarchical splines. *axioms*, 7(3):43, 2018.

[9] A. Bressan and E. Sande. Approximation in FEM, DG and IGA: a theoretical comparison. *Numerische Mathematik*, 143(4):923–942, 2019.

[10] A. Bressan and S. Takacs. Sum factorization techniques in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 352:437–460, 2019.

[11] A. Buffa and C. Giannelli. Adaptive isogeometric methods with hierarchical splines: error estimator and convergence. *Mathematical Models and Methods in Applied Sciences*, 26(01):1–25, 2016.

[12] A. Buffa, C. Giannelli, P. Morgenstern, and D. Peterseim. Complexity of hierarchical refinement for a class of admissible mesh configurations. *Computer Aided Geometric Design*, 47:83–92, 2016.

[13] A. Buffa, T.J.R. Hughes, A. Kunoth, and C. Manni. Mathematical Foundations of Isogeometric Analysis. *Oberwolfach Reports*, 16(3):1981–2032, 2020.

[14] F. Calabrò, G. Loli, G. Sangalli, and M. Tani. Quadrature Rules in the Isogeometric Galerkin Method: State of the Art and an Introduction to Weighted Quadrature. In *Advanced Methods for Geometric Modeling and Numerical Simulation*, pages 43–55. Springer, 2019.

[15] F. Calabrò, C. Manni, and F. Pitolli. Computation of quadrature rules for integration with respect to refinable functions on assigned nodes. *Applied Numerical Mathematics*, 90:168–189, 2015.

[16] F. Calabrò, G. Sangalli, and M. Tani. Fast formation of isogeometric Galerkin matrices by weighted quadrature. *Computer Methods in Applied Mechanics and Engineering*, 316:606–622, 2017.

[17] M. Carraturo, C. Giannelli, A. Reali, and R. Vázquez. Suitably graded THB-spline refinement and coarsening: Towards an adaptive isogeometric analysis of additive manufacturing processes. *Computer Methods in Applied Mechanics and Engineering*, 348:660–679, 2019.

[18] L. Coradello, P. Antolin, R. Vázquez, and A. Buffa. Adaptive isogeometric analysis on two-dimensional trimmed domains based on a hierarchical approach. *Computer Methods in Applied Mechanics and Engineering*, 364:112925, 2020.

[19] J.A. Cottrell, T.J.R. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. John Wiley & Sons, 2009.

[20] J.A. Cottrell, A. Reali, Y. Bazilevs, and T.J.R. Hughes. Isogeometric analysis of structural vibrations. *Computer Methods in Applied Mechanics and Engineering*, 195(41-43):5257–5296, 2006.

[21] L. Beirão da Veiga, A. Buffa, G. Sangalli, and R. Vázquez. Mathematical analysis of variational isogeometric methods. *Acta Numerica*, 23:157–287, 2014.

[22] Q. Deng, M. Bartoň, V. Puzyrev, and V. Calo. Dispersion-minimizing quadrature rules for C1 quadratic isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 328:554–564, 2018.

[23] D. Drzisga, B. Keith, and B. Wohlmuth. The surrogate matrix methodology: Low-cost assembly for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 361:112776, 2020.

[24] J.A. Evans, Y. Bazilevs, I. Babuška, and T.J.R. Hughes. n-Widths, sup–infs, and optimality ratios for the k-version of the isogeometric finite element method. *Computer Methods in Applied Mechanics and Engineering*, 198(21-26):1726–1741, 2009.

[25] G. Gantner, D. Haberlik, and D. Praetorius. Adaptive IGAFEM with optimal convergence rates: Hierarchical B-splines. *Mathematical Models and Methods in Applied Sciences*, 27(14):2631–2674, 2017.

[26] C. Giannelli, B. Jüttler, S.K. Kleiss, A. Mantzaflaris, B. Simeon, and J. Špeh. THB-splines: An

effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 299:337–365, 2016.

[27] C. Giannelli, B. Jüttler, and H. Speleers. THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29(7):485–498, 2012.

[28] A. Giust, B. Jüttler, and A. Mantzaflaris. Local (T)HB-spline projectors via restricted hierarchical spline fitting. *Computer Aided Geometric Design*, 80:101865, 2020.

[29] H. Gomez and L. De Lorenzis. The variational collocation method. *Computer Methods in Applied Mechanics and Engineering*, 309:152–181, 2016.

[30] R.R. Hiemstra, F. Calabrò, D. Schillinger, and T.J.R. Hughes. Optimal and reduced quadrature rules for tensor product and hierarchically refined splines in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:966–1004, 2017.

[31] C. Hofreither. A black-box low-rank approximation algorithm for fast matrix assembly in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 333:311–330, 2018.

[32] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135–4195, 2005.

[33] T.J.R. Hughes, A. Reali, and G. Sangalli. Efficient quadrature for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):301–313, 2010.

[34] T.J.R. Hughes, G. Sangalli, and M. Tani. Isogeometric analysis: Mathematical and implementational aspects, with applications. In *Splines and PDEs: From Approximation Theory to Numerical Linear Algebra*, pages 237–315. Springer, 2018.

[35] B. Jüttler, U. Langer, A. Mantzaflaris, S.E. Moore, and W. Zulehner. Geometry + Simulation modules: Implementing isogeometric analysis. *Proceedings in Applied Mathematics and Mechanics*, 14(1):961–962, 2014.

[36] A. Karatarakis, P. Karakitsios, and M. Papadrakakis. GPU accelerated computation of the isogeometric analysis stiffness matrix. *Computer Methods in Applied Mechanics and Engineering*, 269:334–355, 2014.

[37] R. Kraft. *Adaptive und linear unabhängige Multilevel B-Splines und ihre Anwendungen.* PhD thesis, Universität Stuttgart, 1998.

[38] U. Langer, S.E. Moore, and M. Neumüller. Space–time isogeometric analysis of parabolic evolution problems. *Computer Methods in Applied Mechanics and Engineering*, 306:342–363, 2016.

[39] G. Loli, M. Montardini, G. Sangalli, and M. Tani. An efficient solver for space–time isogeometric Galerkin methods for parabolic problems. *Computers & Mathematics with Applications*, 80(11):2586–2603, 2020.

[40] A. Mantzaflaris and B. Jüttler. Integration by interpolation and look-up for Galerkin-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 284:373–400, 2015.

[41] A. Mantzaflaris, B. Jüttler, B.N. Khoromskij, and U. Langer. Low rank tensor methods in Galerkin-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 316:1062–1085, 2017.

[42] M. Montardini, G. Sangalli, and L. Tamellini. Optimal-order isogeometric collocation at Galerkin superconvergent points. *Computer Methods in Applied Mechanics and Engineering*, 316:741–757, 2017.

[43] M. Pan, B. Jüttler, and A. Giust. Fast formation of isogeometric Galerkin matrices via integration by interpolation and look-up. *Computer Methods in Applied Mechanics and Engineering*, 366:113005, 2020.

[44] M. Pan, B. Jüttler, and A. Mantzaflaris. Efficient matrix assembly in isogeometric analysis with hierarchical B-splines. *Journal of Computational and Applied Mathematics*, 390:113278, 2021.

[45] E. Sande, C. Manni, and H. Speleers. Sharp error estimates for spline approximation: Explicit constants, n-widths, and eigenfunction convergence. *Mathematical Models and Methods in Applied Sciences*, 29(06):1175–1205, 2019.

[46] G. Sangalli and M. Tani. Matrix-free weighted quadrature for a computationally efficient isogeometric k-method. *Computer Methods in Applied Mechanics and Engineering*, 338:117–133, 2018.

[47] D. Schillinger, J.A. Evans, A. Reali, M.A. Scott, and T.J.R. Hughes. Isogeometric collocation: Cost

comparison with Galerkin methods and extension to adaptive hierarchical NURBS discretizations. *Computer Methods in Applied Mechanics and Engineering*, 267:170–232, 2013.

[48] D. Schillinger, S.J. Hossain, and T.J.R. Hughes. Reduced Bézier element quadrature rules for quadratic and cubic splines in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 277:1–45, 2014.

[49] F. Scholz, A. Mantzaflaris, and B. Jüttler. Partial tensor decomposition for decoupling isogeometric Galerkin discretizations. *Computer Methods in Applied Mechanics and Engineering*, 336:485–506, 2018.

[50] H. Speleers. Hierarchical spline spaces: quasi-interpolants and local approximation estimates. *Advances in Computational Mathematics*, 43(2):235–255, 2017.

[51] H. Speleers and C. Manni. Effortless quasi-interpolation in hierarchical spaces. *Numerische Mathematik*, 132(1):155–184, 2016.