INAUGURAL – DISSERTATION

zur

Erlangung der Doktorwürde

der

Naturwissenschaftlich-Mathematischen Gesamtfakultät

der

Ruprecht – Karls – Universität

Heidelberg

vorgelegt von

M. Sc. Julius Witte

geboren in

Höxter, Deutschland

Datum der mündlichen Prüfung:

.....

Fast and Robust Multilevel Schwarz Methods using Tensor Structure for High-Order Finite Elements

Betreuer: Prof. Dr. Guido Kanschat

Abstract

We design innovative fast and robust numerical solvers by exploiting tensor structure for high-order finite element discretizations of various partial differential equations (PDE). The thesis's main scientific contribution is the careful design and implementation of cost-efficient subspace corrections concerning Schwarz smoothers on vertex patches (i.e., the union of cells sharing a common vertex). Emphasis is put on matrix-free implementations of multilevel solvers that are the method of choice in high-performance computing.

Schwarz smoothers on vertex patches lead to robust numerical solvers. They theoretically enable scalability to extremely large applications on modern supercomputers. However, naïvely computing subspace corrections is prohibitively expensive, negating their mathematical benefits and those of matrix-free operators. To this end, we develop tensor product Schwarz smoothers, exploiting low-rank tensor representations of local solvers. Then, the computational effort per unknown is linear in the polynomial degree, the cost for inverting local matrices is asymptotically negligible, and their memory consumption per unknown is constant. We develop smoothing algorithms for high-order DG and H^1-conforming discretizations of the Laplacian, presenting the prototypical differential operator that preserves separability to finite element operators. We demonstrate the high computational efficiency of our implementations in terms of the number of floating-point operations, (strong) scaling behavior, and time-to-solution. The superiority of multiplicative Schwarz methods on vertex patches (MVS) over simple non-overlapping Schwarz smoothers is shown for higher-order finite elements, arising from their superior mathematical efficiency. The mathematical efficiency of restricted additive Schwarz smoothers comparable to MVS is studied, and extensions to non-Cartesian meshes are discussed. The techniques are then extended to biharmonic and Stokes problems that lack some separability assumptions for fast inversion. We still develop cost-efficient subspace approximations for multilevel C^0-IP methods. Given the stream function method, similar smoothers are applied for $\boldsymbol{H}^{\mathrm{div}}$-IP discretizations of Stokes problems in 2D: we design novel subspace corrections involving local stream functions and local pressure post-processing for Raviart-Thomas elements.

Our implementations are designed with a holistic view on computational efficiency, i.e., carefully balancing arithmetic operations and data transfer to exploit the potential of modern multi-core architectures with SIMD capabilities. Thus, optimal node-level performance is achievable. The C++ software for tensor product Schwarz smoothers is publically available.

Zusammenfassung

Wir nutzen Tensorstrukturen aus, um innovative, schnelle und robuste numerische Löser für Finite-Elemente-Verfahren höherer Ordnung für verschiedene partielle Differentialgleichungen zu entwickeln. Der wissenschaftliche Hauptbeitrag der Dissertation beinhaltet das sorgfältige Entwerfen und Implementieren kosteneffizienter Unterraumkorrekturen hinsichtlich Schwarz-Glättungsverfahren auf Vertex-Patches (d.h. Untergebiete bestehend aus den Gitterzellen mit einem gemeinsamen Eckpunkt). Ein Schwerpunkt liegt auf matrixfreien Implementierungen von Mehrgitter-Lösern, die derzeit der Goldstandard im Bereich High-Performance Computing sind.

Diese Art von Schwarz-Verfahren führt zu robusten numerischen Lösern, sodass theoretisch eine Skalierbarkeit auf sehr große Anwendungsprobleme für modernste Supercomputer ermöglicht wird. Allerdings ist die naive Berechnung von Unterraumkorrekturen äußerst rechenintensiv oder gar unmöglich, sodass sowohl die mathematischen Vorteile als auch die Vorteile der matrixfreien Verfahren konterkariert werden. Deshalb entwickeln wir Tensorprodukt-Schwarz-Glätter, die die Tensordarstellungen der lokalen Löser niedrigen Ranges ausnutzen. Dann ist der Rechenaufwand pro Freiheitsgrad linear zum Polynomgrad, die Invertierungskosten lokaler Matrizen sind asymptotisch vernachlässigbar und ihr Speicherbedarf pro Freiheitsgrad ist konstant. Wir entwickeln Glättungsalgorithmen bezüglich DG und H^1-konformen Diskretisierungen höherer Ordnung für den Laplace-Operator. Dieser stellt einen prototypischen Differentialoperator dar, der die Separabilitätsbedingung für zugehörige Finite-Elemente-Operatoren erhält. Wir weisen eine hohe Recheneffizienz unserer Implementierungen nach, indem wir die Anzahl von Gleitkommaoperationen, das (starke) Skalierungsverhalten und die Zeit-zum-Lösen untersuchen. Eine Überlegenheit der multiplikativen Schwarz-Methoden auf Vertex-Patches (MVS) gegenüber einfachen nicht-überlappenden Schwarz-Glättern lässt sich für Finite Elemente höherer Ordnung zeigen. Diese resultiert aus der überlegenen mathematischen Effizienz. Wir zeigen, dass die mathematische Effizienz von uns entwickelter eingeschränkter additiver Schwarz-Glätter vergleichbar mit MVS ist. Zudem diskutieren wir Erweiterungen unserer Glättungsverfahren auf nicht-kartesische Gitter. Unsere Methoden werden danach für biharmonische und Stokes Probleme erweitert. Für diese Probleme sind die Separabilitätsannahmen für schnelles Invertieren lokaler Matrizen nicht gegeben. Dennoch ist es möglich kosteneffiziente Unterraumapproximationen für Mehrgitter-Methoden bezüglich C^0-IP Diskretisierungen eines biharmonischen Problems

zu entwickeln. Mit Hilfe von Stromfunktionen lassen sich ähnliche Glätter auch für $\boldsymbol{H}^{\mathrm{div}}$-IP Diskretisierungen eines zwei-dimensionalen Stokes Problems anwenden: wir entwerfen dazu neue Unterraumkorrekturen basierend auf lokalen Stromfunktionen und einer (lokalen) Nachberechnung des Drucks für Raviart-Thomas-Elemente.

Unsere Implementierungen sind mit einem ganzheitlichen Blick auf Rechenleistung konzipiert, d.h. ein sorgfältiges Ausbalancieren arithmetischer Operationen und des Datentransfers, um das ganze Potential modernster Multi-Core-Architekturen mit SIMD-Fähigkeiten auszuschöpfen. Auf diese Weise kann eine optimale Leistung auf Knotenebene erzielt werden. Die C++ Software für unsere Tensorprodukt-Schwarz-Glätter ist öffentlich verfügbar.

Danksagung

Zuerst möchte ich herzlich meinem Doktorvater, Prof. Dr. Guido Kanschat, danken. Du hast mir diese Promotion in einem spannenden Forschungsbereich und exzellentem Umfeld ermöglicht. Neben deiner unbestrittenen, fachlichen Expertise, die die Basis für meine Forschungstätigkeiten war, schätze ich in besonderem Maße deine aufrichtige Motivation beim Forschen, deine moderne Art zu lehren und Software zu entwerfen sowie deinen offenen, herzlichen Umgang mit Menschen. Ich hatte das Privileg, Forschungsschwerpunkte frei zu wählen. Ich danke dir für deine Geduld und das Vertrauen in meine Arbeit. Auch abseits fachlicher Themen durfte ich viel über das Leben sowie ferne Länder und Kulturen von dir lernen. Großer Dank gilt Daniel Arndt für seinen wissenschaftlichen Rat, fachliche Diskussionen und selbstlose Unterstützung. Ich bewundere deinen Scharfsinn, enormen Wissensdurst und unbändigen Ehrgeiz. Danke für gemeinsame "fit for fun"-Kurse, gesellige Stunden und deine Freundschaft. Ich danke Prof. Dr. Peter Bastian für das Lesen und Begutachten meiner Dissertation.

Ein herzlicher Dank gilt meinen ehemaligen Kollegen der Arbeitsgruppe "Mathematische Methoden der Simulation". Ich danke meinen Schreibtischnachbarn Noman Shakir und Stefan Meggendorfer für den (fachlichen) Austausch und die kleinen, unterhaltsamen Ablenkungen. Für die Rückendeckung in Organisationsfragen und den gemütlichen Plausch danke ich herzlich Felicitas Hirsch. Mit Andreas Rupp durfte ich manche Kuchenpause und viele gesellige Stunden im Bräustadl verbringen. Ich danke Niklas Fehn und Martin Kronbichler für die spannende Zusammenarbeit im Rahmen des ExaDG Projektes, gemeinsame Doctoral Retreats und fachliche Treffen in München. Besonders danke ich Martin für die exzellente Implementation matfrixfreier Methoden, die meine Forschung motiviert und meine Softwareentwicklung nachhaltig geprägt hat. Ich danke Conrad Clevenger und Timo Heister für wundervolle Wochen in Clemson und ihre unvergleichliche Gastfreundschaft. Ich danke der `deal.II` Community für ein familiäres Umfeld, indem ich erste Schritte mit C++ tätigen und meine große Leidenschaft zum Programmieren entdecken durfte.

Ich danke meinen Freunden für die Freundschaft und die Unterstützung in den letzten Jahren. Ganz besonders danke ich meinen Eltern und meinem Bruder Moritz, die mir selbstlos Gehör, Rat und Mut gespendet haben. Der größte Dank gilt meiner Frau Melanie für die Hingabe und Opfer, die du erbracht hast. Du bist mit mir gemeinsam diesen langen Weg

gegangen – auch bei Hamburger Schietwetter. Du bist mir ein Vorbild, mein bester Freund und die Liebe meines Lebens.

Table of Contents

List of Figures

List of Tables

Nomenclature

Roman Symbols

$C^\infty(\Omega)$	space of infinitely differentiable functions in Ω
$C_0^\infty(\Omega)$	space of C^∞-functions with compact support in Ω
D	number of spatial dimensions
e	facet index (i.e., an edge in 2D and a face in 3D)
$\boldsymbol{F}_K$	transformation map from unit cell $\hat{K}$ to cell K
$\boldsymbol{H}^{\mathrm{curl}}(\Omega)$	Sobolev space of $L^2(\Omega)$-functions with square-integrable weak curl
$\boldsymbol{H}^{\mathrm{div}}(\Omega)$	Sobolev space of $L^2(\Omega)$-functions with square-integrable weak divergence
$\boldsymbol{H}_0^{\mathrm{div}}(\Omega)$	space of $\boldsymbol{H}^{\mathrm{div}}$-functions with vanishing normal trace on $\partial\Omega$
$H^1(\Omega)$	Sobolev space of $L^2(\Omega)$-functions with square-integrable weak derivative
$H_0^1(\Omega)$	space of $H^1(\Omega)$-functions with vanishing trace on $\partial\Omega$
$\boldsymbol{H}^1(\Omega)$	vector-valued $H^1(\Omega)$-functions (abbreviating $H^1(\Omega;\mathbb{R}^D)$)
$\boldsymbol{H}_0^1(\Omega)$	vector-valued $H_0^1(\Omega)$-functions (abbreviating $H_0^1(\Omega;\mathbb{R}^D)$)
$H^1(\mathcal{T}_h)$	Sobolev space of piecewise $H^1(\Omega)$-functions on mesh $\mathcal{T}_h$
$\hat{J}_K$	Jacobian matrix of $\boldsymbol{F}_K$, i.e., $(\hat{J}_K)_{ij} = \hat{\partial}_j \boldsymbol{F}_{K;i}$
K	mesh cell
k	degree of polynomials
$\hat{K}$	unit mesh cell
L	level index of most-refined mesh
$L^2(\Omega)$	Lebesgue space of scalar-valued square-integrable functions on Ω

$\boldsymbol{L}^2(\Omega)$	vector-valued $L^2(\Omega)$-functions (abbreviating $L^2(\Omega;\mathbb{R}^D)$)
$L^2_0(\Omega)$	space of mean-value free $L^2(\Omega)$-functions, i.e., $L^2(\Omega)/\mathbb{R}$
$\boldsymbol{n}$	unit normal vector
N_{cell}	number of cells
N_{col}	number of colors
N_{dof}	number of degrees of freedom per element
n_{dof}	number of degrees of freedom per element per dimension
N_j	number of degrees of freedom for subdomain Ω_j
N_{FLOP}	number of floating-point operations
N_h	number of degrees of freedom on mesh $\mathcal{T}_h$
N_{quad}	number of quadrature points per element
n_{quad}	number of quadrature points per element per dimension
$P_{\ell;j}$	Ritz projection onto subspace $V_{\ell;j}$
$P_{ad;\ell}$	additive Schwarz operator on level ℓ
$P_{mu;\ell}$	multiplicative Schwarz operator on level ℓ
RT_k	Raviart-Thomas polynomials/elements of degree k
$R_{\ell;j}$	restriction operator onto subspace $V_{\ell;j}$
$R^{\mathsf{T}}_{\ell;j}$	prolongation operator from subspace $V_{\ell;j}$ to V_ℓ
$S_{ad;\ell}$	additive Schwarz smoother on level ℓ
$S_{mu;\ell}$	multiplicative Schwarz smoother on level ℓ
$\boldsymbol{t}$	unit tangential vector
V_ℓ	finite element space on level ℓ
$V_{\ell;j}$	finite element subspace on subdomain Ω_j
$\boldsymbol{x}$	spatial coordinate in real space $\Omega \subset \mathbb{R}^D$
$\hat{\boldsymbol{x}}$	spatial coordinate in unit space $[0,1]^D$

Greek Symbols

δ_{ij}	Kronecker delta is one if $i = j$, otherwise zero
$\Delta\,\varphi$	Laplace operator of a scalar-valued function φ
$\boldsymbol{\Delta}\,\boldsymbol{v}$	Laplace operator of a vector field $\boldsymbol{v}$
γ_e	(interior) penalty factor on facet e
$\gamma_{0,e}$	(interior) penalty pre-factor on facet e
$\boldsymbol{\nabla}\varphi$	gradient of a scalar-valued function φ
$\nabla \cdot \boldsymbol{v}$	divergence of a vector field $\boldsymbol{v}$
$\boldsymbol{\nabla} \times \varphi$	vector curl of a scalar-valued function φ
ν_{frac}	fractional convergence steps
ω	relaxation factor for smoothers
Ω	physical domain
$\partial\Omega$	boundary of physical domain
Ω_j	subdomain
$\Pi_{h;K}^{\mathsf{T}}$	transfer of cell-local to global coefficients on $\mathcal{T}_h$
$\Pi_{h;K}$	transfer of mesh-global to local coefficients on K

Calligraphic Symbols

$\mathcal{E}_h$	set of facets included in mesh $\mathcal{T}_h$
$\mathcal{E}_h^{\partial}$	set of boundary facets included in mesh $\mathcal{T}_h$
$\mathcal{E}_h^{\circ}$	set of interior facets included in mesh $\mathcal{T}_h$
$\mathcal{E}_{h,j}$	set of facets included in local mesh $\mathcal{T}_j$
$\mathcal{E}_{h,j}^{\partial}$	set of facets at the subdomain's boundary $\partial\Omega_j$
$\mathcal{E}_{h,j}^{\partial\partial}$	subset of $\mathcal{E}_{h,j}^{\partial}$ containing facets at the physical boundary $\partial\Omega$
$\mathcal{E}_{h,j}^{\partial\circ}$	subset of $\mathcal{E}_{h,j}^{\partial}$ containing facets in the interior of domain Ω
$\mathcal{E}_{h,j}^{\circ}$	set of facets in the subdomain's interior Ω_j

$\mathcal{R}(A)$	reshuffling of the matrix A
$\mathcal{T}_h$	mesh with characteristic size h
$\mathcal{T}_j$	local mesh of subdomain Ω_j with characteristic size h

Blackboard Symbols

$\mathbb{N}$	set of natural numbers
$\mathbb{P}_k$	space of polynomials of degree at most k
$\mathbb{Q}_k$	space of tensor product polynomials of degree k
$\mathbb{R}$	set of real numbers

Fraktur Symbols

$\mathfrak{A}$	tensorization of the matrix A
$\mathfrak{v}$	tensorization of the vector v

Subscripts / Superscripts

V'	dual space of a topological vector space V
$W^\perp$	orthogonal complement of Hilbert subspace $W \subset H$
$\boldsymbol{v}^\perp$	normal vector field of a vector field $\boldsymbol{v}$
A^T	transpose of the matrix A
$\boldsymbol{v}^\parallel$	tangential vector field of a vector field $\boldsymbol{v}$

Other Symbols

$\operatorname{card} S$	cardinality of a set S
$\operatorname{cond} A$	condition number of a matrix A
$\operatorname{diag}(\dots)$	diagonal matrix with entries … from top left to bottom right
$\operatorname{diam} G$	diameter of geometric structure G
$\operatorname{div} \boldsymbol{v}$	divergence of vector field $\boldsymbol{v}$ (alternative)
$\oplus$	direct sum
$\boldsymbol{grad}\, \varphi$	gradient of scalar-valued function φ (alternative)

id	identity operator
$\operatorname{Im}\varphi$	the image/range of a function φ
$\operatorname{Ker}\varphi$	the kernel of a homomorphism φ
$\operatorname{meas} S$	measure of a set S
$\|\cdot\|_2$	Euclidean norm
$\|\cdot\|_F$	Frobenius norm
$\|\cdot\|_\Omega$	norm of $L^2(\Omega)$-functions
$\|\cdot\|_{r,\Omega}$	norm of $H^r(\Omega)$-functions
$\|\cdot\|_{r,h}$	norm of $H^r(\mathcal{T}_h)$-functions
$\operatorname{rank} A$	rank of a matrix A
$\|\cdot\|_{r,\Omega}$	seminorm of $H^r(\Omega)$-functions
$\|\cdot\|_{r,h}$	seminorm of $H^r(\mathcal{T}_h)$-functions
$\operatorname{span} S$	linear span of a set of vectors S
$\rho(A)$	spectral radius of a bounded linear operator A
$\operatorname{supp}\phi$	(essential) support of a function ϕ
tens	tensorization
$\mathrm{Tr}_{\Omega,\Omega'}$	trace operator w.r.t. submanifold Ω' of Ω
Tr	abbreviation of trace operator $\mathrm{Tr}_{\Omega,\Omega'}$ in case of $\Omega' = \partial\Omega$
vect	vectorization

Acronyms / Abbreviations

ACS	Additive Cell-based Smoother
AVS	Additive Vertex patch Smoother
AVX	Advance Vector Extensions
BDM	Brezzi-Douglas-Marini elements
C^0IP	C^0 Interior Penalty method

CFD	Computational Fluid Dynamics
CG	Conjugate Gradient method
CPU	Central Processing Unit
DG	Discontinuous Galerkin method
DoF	Degree of Freedom
FEM	Finite Element Method
FLOP	FLoating-point OPeration
GMRES	Generalized Minimal RESidual method
GPU	Graphics Processing Unit
HPC	High-Performance Computing
ISA	Instruction Set Architecture
KSVD	Kronecker product Singular Value Decomposition
MCS	Multiplicative Cell-based Smoother
MPI	Message Passing Interface
MVS	Multiplicative Vertex patch Smoother
PCG	Preconditioned Conjugate Gradient method
PDE	Partial Differential Equation
RAS	Restricted Additive Schwarz smoother
RAS	Restricted Additive Vertex-patch Schwarz smoother
RT	Raviart-Thomas elements
SEM	Spectral Element Method
SIMD	Single Instruction, Multiple Data
SIPG	Symmetric Interior Penalty Galerking method
SVD	Singular Value Decomposition
TPSS	Tensor Product Schwarz Smoothers

Chapter 1

Introduction

In this thesis, we will design innovative robust and fast numerical solvers through exploiting tensor structure for (non-standard) high-order finite element discretizations of various partial differential equations (PDE). The careful design of cost-efficient subspace approximations for overlapping Schwarz smoothers will be the thesis' guiding principle and major scientific contribution.

1.1 Motivation and Perspective

We develop fast numerical solvers for three kinds of PDE problems, the Poisson, biharmonic, and Stokes problem. They have a direct or indirect impact to computational fluid dynamics (CFD) being part of methods or a preliminary step towards solving prominent incompressible flow problems modeled by the Navier-Stokes equations, respectively. CFD is more than ever a thriving field of research due to the fast-paced development of computer hardware (including the world's fastest supercomputers) over recent decades, providing society with simulations of countless diverse flow problems.

Researching in the field of scientific computing as well as merging aspects of mathematics and computer science, the author's main **motivation** arises from the challenge of developing robust mathematical methods and implementing them with an emphasis on cost-efficiency such that fast numerical solvers on modern supercomputers are obtained. Mathematically, we denote a *numerical solver* as *robust* if it converges uniformly with respect to characteristic sizes of the underlying discretization, in our case, the generic mesh size and polynomial degree. The robustness is essential for the PDE problem's scalability, ensuring (almost) a constant number of the solver's iterative steps for any size of the discrete problem. In theory, we may evaluate a numerical solver based on its convergence speed without considering computational aspects. Thus, we measure mathematical efficiency by the reciprocal number of convergence steps to reach a target accuracy.

When it comes to engineering numerical solvers, we are limited by computational resources. Moreover, we need to consider modern computer architectures' capabilities. Developing *fast numerical solvers* is a complex task, and implementing them is even more delicate. It starts with developing the mathematical method to solve a given problem reliably, emphasizing mathematical efficiency. Given the methodology, we want to derive algorithms that theoretically perform numerical procedures with optimal computational complexity in space and time. It ends with application software for a class of target platforms, requiring careful implementation of algorithms to exploit the full potential of underlying computer hardware. The transition of the three fields, i.e., methods, algorithms, and implementations, is relatively smooth, and mutual dependencies emerge. For instance, increasing mathematical efficiency should also increase computational efficiency at first sight. However, to reduce the number of iteration steps, we usually pay the price of a method's increased sophistication, significantly increasing the number of arithmetic operations and memory needed for a single algorithmic step. Potentially, this growth in computational complexity negates all mathematical benefits gained, resulting in prohibitively expensive algorithms for large-scale applications. Choosing a metric to measure computational efficiency, i.e., how fast a numerical solver is, is not trivial. Rooted in the field of algorithms, simple metrics are the ratio of the number of unknowns to floating-point operations or consumed memory. However, these metrics do not factor in computing capabilities. We may measure computational efficiency by the ratio of degrees of freedom to the time-to-solution, where "time" literally refers to elapsed real-time. This metric, in turn, may limit the comparison to a subset of target platforms, in particular, given the heterogeneous computing landscape of the 21$^{\text{st}}$ century. The thesis' core topic is the design of algorithms that achieve high computational efficiency. Nevertheless, we also address the other two related topics: developing new mathematical methods if needed and carefully implementing algorithms focusing on modern (super)computers. We stress the attribute "modern" because our algorithms will particularly account for the trend of arithmetic capabilities evolving faster than memory bandwidth.

Multigrid methods accelerated by Krylov subspace solvers have become the method of choice in high-performance computing. In particular, the success of matrix-free implementations is ever-growing for high-order finite element problems in modern HPC, among others, referring to the state-of-the-art PDE solvers in (Arndt, Fehn, et al., 2020; Bastian, Altenbernd, et al., 2020; Bauer et al., 2020; P. Fischer et al., 2020; Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019). At the beginning of our research, we identified that mainly diagonal-based or nonoverlapping block-Jacobi methods were utilized for multilevel smoothing. We were motivated to take a step beyond these cost-efficient "simple" smoothers, developing fast and powerful Schwarz smoothers using nonoverlapping domain decompositions. To be competitive with matrix-free evaluation of discrete "forward" operators, (local) "inverse" operators (inherent in Schwarz smoothers) need to be computed and applied at a similar pace. In addition, smoothing algorithms shall have a comparable memory footprint

(if stored at all). We strive to overcome these challenges by exploiting tensor structure, advancing well-known concepts from the spectral element community (Couzy and Deville, 1994, 1995; P. F. Fischer and Lottes, 2005; P. F. Fischer, Tufo, et al., 2000; Lottes and P. F. Fischer, 2005) and establishing new alternatives to recent methods for high-order finite elements (Barker and Kolev, 2020; Bastian, Müller, et al., 2018; Pazner, 2020; Pazner and Kolev, 2021; Pazner and Persson, 2018). To this end, the thesis' main topic will be designing and implementing innovative *tensor product Schwarz smoothers* with a holistic view on computational complexity, i.e., (i) satisfying mathematical robustness and parallelism to provide scalability of PDE problems on modern supercomputers, (ii) leveraging tensor structure to obtain cost-efficient algorithms that minimize data transfer in main memory, and (iii) exploiting the full potential of multi-core architectures for optimal node-level performance. Then, solving various classes of PDE problems becomes feasible also at large scales.

Fast inversion is only achievable at the cost of losing generality, i.e., stricter assumptions on the tensor structure exist than for the "forward" matrix-free operators. The critical reader might argue that limiting us to Cartesian meshes and separable physical coefficients is too restrictive. We suggest concepts throughout this monograph on overcoming these limitations. Among others, we may apply our methods only partially in "regions" with (approximately) regular structure. They may be combined with cut-cell or level-set methods allowing for complex geometries, or may act as local preconditioners accelerating the local iterative solvers in (Bastian, Müller, et al., 2018) that apply to generic meshes and differential operators, respectively. We may extend them to the very recent smoothing algorithms of Brubeck and Farrell (2021) that handle unstructured vertex patches and meshes. Besides, our algorithms still prove valuable by achieving optimal computational complexity when regular structure is given, following Albert Einstein's principle: "Alles sollte so einfach wie möglich sein - aber nicht einfacher."

The author sees potential in using matrix-free implementations as comprehensive standard in modern finite element software given the current advances in computer hardware and easy access to cloud computing using state-of-the-art resources at various scales. Therefore, developing and implementing fast preconditioners will be as crucial as the matrix-free operator evaluation. We elaborate our design choices for the tensor product Schwarz smoothers, our background in Exascale computing, and the thesis' scientific contribution next.

Taking a methodological **perspective**, multigrid methods have proven to be the method of choice for solving elliptic or parabolic PDE problems over recent decades. We utilize geometric multigrid methods based on a hierarchy of uniformly refined Cartesian meshes. These methods consist of several building blocks: a hierarchy of discretization matrices, smoothers, transfer operators, and a coarse-grid solver. If these components are well-composed according to the underlying PDE problem, linear arithmetic complexity in the number of unknowns N_L can be expected. L indicates the level of the most-refined mesh. Each level-related finite element discretization is characterized by a generic mesh width h_ℓ and a level-independent

polynomial degree k. The number of elements is proportional to h_L^{-D} and the number of degrees of freedom per element to k^D such that $N_L = \mathcal{O}(h_L^{-D} k^D)$. Outer Krylov subspace solvers accelerate the multilevel methods.

Taking a technical perspective, matrix-free implementations of numerical solvers have become the gold standard in HPC, in particular, for high-order finite elements (Arndt, Fehn, et al., 2020; Bastian, Altenbernd, et al., 2020; Bauer et al., 2020; P. Fischer et al., 2020; Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019). In its early days, matrix-free methods were used to fit PDE solvers for large-scale applications into main memory (Bergen et al., 2005). Due to the fast-paced development of computer hardware with processing capabilities increasing faster than the memory throughput, (Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019; May et al., 2014) have demonstrated that well-designed matrix-free implementations are superior over sparse-matrix-based finite element codes for cubic and higher finite element orders. In particular, the efficiency gap grows quickly for increasing finite element orders. Traditionally, we iterate over elements computing local integrals via numerical quadrature and ultimately storing the results in a sparse matrix. In contrast, matrix-free operator evaluation folds both operations, the local numerical integration and the matrix-vector product, into one. At the heart of each operator, a technique called *sum factorization* is utilized, which is the root cause for the method's feasibility and success. Sum factorization leverages the tensor structure of finite elements and quadrature formulas, reducing the arithmetic complexity per element from $\mathcal{O}(k^{3D})$ for naïve processing of folded operations to $\mathcal{O}(Dk^{D+1})$. A sparse-matrix-based code pays the price of $\mathcal{O}(k^{3D})$ operations per element only once during the assembly stage, but a standard matrix-vector product still requires $\mathcal{O}(k^{2D})$ operations per element. Using sum factorization is universal for efficient matrix-free operators. Only their highly-optimized implementation varies in prominent academic software libraries, among others[1], mentioning `deal.II` (Arndt, Bangerth, Blais, et al., 2020; Arndt, Bangerth, Davydov, et al., 2021; Arndt, Fehn, et al., 2020), DUNE (Bastian, Altenbernd, et al., 2020; Bastian, Blatt, et al., 2021; Bastian, Engwer, et al., 2016), and MFEM (Anderson et al., 2021).

Therefore, matrix-free operator evaluation is the computational baseline that we strive to reach with well-designed algorithms and implementations of overlapping Schwarz methods for multilevel smoothing. We focus on multiplicative Schwarz smoothers on vertex patches (MVS). A vertex patch is the local decomposition of cells that share a common vertex, resulting naturally in element-wise overlap. For the Poisson problem, MVS is optional but very beneficial for high-order discretizations (equally for the conforming FEM and DG method) since the solver's convergence speed does not degrade with increasing polynomial degree. On the contrary, it improves convergence speed leading almost to a direct solver. In Chapter 3, we will observe their superiority over cell-based nonoverlapping Schwarz methods even in

[1] A more extensive list is found in Section 2.2.

terms of computational efficiency. For the C^0 and $\boldsymbol{H}^{\mathrm{div}}$ interior penalty discretizations of the biharmonic and Stokes problem, overlapping Schwarz methods on vertex patches were developed in (Arnold, Falk, and Winther, 1997; Arnold, Falk, and Winther, 2000; Kanschat and Mao, 2015; Kanschat and Sharma, 2014) to obtain mathematically robust multilevel solvers. In the same context, nonoverlapping Schwarz smoothers or simple pointwise methods fail to provide mathematical robustness, in particular, for high-order finite elements.

We have not addressed the major challenges of deploying Schwarz smoothers (on vertex patches) cost-efficiently in multilevel solvers. Using naïve algorithms,

(i) multiplicative Schwarz smoothers are intrinsically sequential,

(ii) solving local PDE problems by standard methods requires $\mathcal{O}(k^{3D})$ operations,

(iii) if storing local inverses, $\mathcal{O}(k^{2D})$ floating-point numbers are needed, and

(iv) their standard matrix-vector products cost $\mathcal{O}(k^{2D})$ operations,

with Items (ii) to (iv) holding for each subdomain. Standard Schwarz smoothers would completely negate the benefits of matrix-free operator evaluation, dominating the computational complexities in both time and space. For instance, in (Witte et al., 2021) we showed that the one-time computational cost of standard implementations for cell-based Schwarz smoothers (i.e., a block-Jacobi method) are 3000 times higher for tri-cubic polynomials and even 95000 times higher for polynomial degree $k = 7$ than the respective matrix-free Laplace operator. For overlapping Schwarz smoothers on vertex patches, the gap is even more pronounced since k in Items (ii) to (iv) is actually $2k$. Furthermore, Item (i) prevents massively parallel computations.

A remedy for Item (i) is well-known, using the domain decomposition concept called coloring. All subspace corrections which are somewhat decoupled can be colorized equally and processed in parallel. The coupling of sub-problems is determined by the overlap and transmission conditions imposed at interfaces. At the discrete level, the coloring will be derived from the "overlap" of local finite element stencils[2]. We only observe a load imbalance for levels with few, very large sub-problems in computations.

The challenge lies in overcoming Items (ii) to (iv). We identified solving local problems as a main limiting factor for cost-efficiency, given its cubic complexity for standard algorithms. In addition, striving for algorithms with high arithmetic and low memory intensity, it is infeasible to store dense matrices for each subdomain. To this end, we leverage the tensor structure of finite elements, quadrature formulas, (almost) Cartesian mappings, and (almost) separable differential operators through the *fast diagonalization* of local discretization matrices. Compared to forward operators, we additionally pay the price of assumptions on the shape of mesh cells and (local) differential operators (including physical coefficients) to break the

[2]Finite element stencils are understood in the sense of (Arndt, Fehn, et al., 2020, §1).

curse of dimensionality. Fast diagonalization dates back to efficient finite difference methods for certain classes of separable differential operators (Lynch et al., 1964). Their separability is preserved one-to-one for finite difference operators. Unfortunately, the integration-based nature of finite element methods does, in general, not allow such preservation, not even for Cartesian meshes, e.g., see the Bilaplacian. In this regard, a contribution of this thesis is the design of subspace approximations for PDE problems that do not satisfy the separability assumptions completely. Using fast diagonalization, inverting boils down to computing D one-dimensional (generalized) eigenvalue problems at the total cost of $\mathcal{O}(Dk^3)$ arithmetic operations. The obtained inverse is representable as a ternary matrix-matrix product of two rank-1 matrices with a diagonal matrix in between such that only $\mathcal{O}(Dk^2 + k^3)$ floating-point numbers must be stored per inverse. In addition, the matrix-vector product is amenable to sum factorization, thus, requiring $\mathcal{O}(Dk^{D+1})$ operations. Most importantly, the one-time cost for inverting matrices are even one order of magnitude less. Storing local inverses is proportional to the number of unknowns. We refer to this specific class of algorithms as tensor product Schwarz smoothers, overcoming the challenges in Items (ii) to (iv).

The thesis is partly rooted in the field of HPC. My Ph.D. project was funded by the German Research Foundation for the "ExaDG - High-order Discontinuous Galerkin for the Exa-scale" project, being part of the second phase of the German priority programme "Software for Exascale computing" (SPPEXA)[3]. The project ExaDG provides novel contributions in terms of methods and software:

1. robust and accurate discretization methods for CFD-related PDE problems and development of fast matrix-free implementations with a focus on high- order DG methods,

2. developing fast numerical solvers using robust and cost-efficient Schwarz smoothers and matrix-free operator evaluation for high-order finite element methods

Our final report (Arndt, Fehn, et al., 2020) outlines the developed methods and implemented algorithms. Given the latest TOP500 list, society is close to but still waiting for Exascale[4].

The thesis has no particular focus on extensively demonstrating parallel computations on extremely large scales. We will analyze the parallel performance of our implementations at a smaller scale (using 2560 cores distributed over 64 compute nodes) with emphasis on the smoothing algorithms' computational efficiency. The thesis' contributions are instead designing cost-efficient algorithms and sound (early-staged) implementations, providing strong scaling. Finite element solvers rely on nearest-neighbor communication based on the mesh's domain decomposition: matrix-free operator evaluation communicates at element level and Schwarz smoothers at subdomain level. Since we use very small subdomains (at most a

[3]See http://www.sppexa.de/. Exascale computing refers to reaching 10^{18} floating-point operations per second.

[4]Fugaku remained the No. 1 system achieving an HPL benchmark score of 0.442 Exaflop/s according to https://www.top500.org/.

vertex patch), both communications are comparable. Thus, subspace corrections with high "sequential" efficiency are essential in reaching excellent node-level performance. To this end, we focus on developing algorithms that reduce computational complexities by one or more orders of magnitude (as explained above) rather than tweaking implementations to improve runtime by a factor of 2-3. In this regard, while high arithmetic intensity is essential for achieving peak performance, we prefer algorithms with higher numerical throughput (the ratio of degrees of freedom per runtime) even if they result in lower hardware performance indicators like memory throughput or arithmetic performance. In particular, tailoring implementations to modern computer architectures, our software automatically profits from the fast-paced advances of computer hardware.

1.2 Scientific Contribution and Outline

My work pursues two main goals. First, I design efficient algorithms for overlapping Schwarz smoothers. Second, I develop software in C++ to run these algorithms with an arithmetic intensity and computational efficiency comparable to matrix-free operators. A **quick reference** of my **scientific contribution** follows, highlighting the exclusive novelties in italic font:

(i) For high-order DG [and H^1-conforming] discretizations of the Poisson problem, I designed [and design] cost-efficient algorithms for additive and *multiplicative* tensor product Schwarz smoothers *on vertex patches for geometric multigrid methods* utilizing fast diagonalization. Moreover,

  (a) not only exact local solvers for Cartesian meshes, but also inexact local solvers for *more generic meshes* were introduced,

  (b) *memory-distributed parallel computations using SIMD capabilities are analyzed concerning strong scaling behavior and computational efficiency, comparing nonoverlapping and overlapping Schwarz smoothers,* and

  (c) mathematical efficiency of *restricted additive Schwarz smoothers on vertex patches* is shown for *two simple weighting strategies.*

(ii) For *high-order* C^0-IP *discretizations of the biharmonic Problem for two-dimensional Cartesian meshes*, this thesis presents a *novel design of efficient algorithms* for tensor product Schwarz smoothers on vertex patches for geometric multigrid methods *utilizing fast diagonalization.*

(iii) No tensor product smoothers existed for *high-order* $\boldsymbol{H}^{\mathrm{div}}$-IP *discretizations of the (simplified) Stokes Problem.* To this end, for two-dimensional Cartesian meshes,

(a) *I present methods that apply subspace approximations of stream functions, thus, developing a (local) pressure post-processing and local transfer operators between stream function and velocity-pressure coefficients,* and

(b) *design efficient algorithms for tensor product Schwarz smoothers on vertex patches for geometric multigrid methods using $\boldsymbol{H}^{div}$-conforming Raviart-Thomas elements*

The excellent mathematical efficiency and robustness of Schwarz smoothers on vertex patches for all three PDE problems were well-studied before my work (Kanschat and Mao, 2015; Kanschat and Sharma, 2014; Pavarino, 1993), but the smoothers' application (using naïve algorithms) was prohibitively expensive for high-order discretizations. I fill this gap with novel, cost-efficient algorithms given the quick reference's outline. The DG-related results of Item (i) and Item (i)(a) were published in (Witte et al., 2021), highlighted through the past tense. Our work (Kronbichler, Kormann, Fehn, et al., 2019) proposed an innovative finite element using a "Hermite-like" polynomial basis to improve data access of face integrals (i.e., obtaining a "thinner" finite element stencil) and proved again the versatility of Schwarz smoothers on vertex patches: the same mathematical efficiency as for standard finite elements is observed. In (Arndt, Fehn, et al., 2020, §5) we proposed a fast Schur complement technique for subspace corrections regarding standard DG discretizations of linear elasticity modeled by the Lamé-Navier equations. I drew motivation from the findings and started to develop the tensor product Schwarz methods using (non-standard) finite elements for Stokes flow (which constitutes the limit of an (almost) incompressible material).

Other fast Schwarz methods or matrix-free preconditioners have been developed during my studies. Stiller (2016, 2017) developed similar tensor product Schwarz methods for SIPG discretizations of the Laplacian but used smoothers on other subdomains for p-multigrid solvers. Furthermore, we refer to the works (Bastian, Müller, et al., 2018; Brubeck and Farrell, 2021; Pazner, 2020; Pazner and Kolev, 2021; Pazner and Persson, 2018) that were mentioned before. All methods (including my methods) mutually vary in applicability, and mathematical and computational efficiency. I address their merits in more detail in the course of this thesis. Highlighting contributions by other Ph.D. students, Niklas Fehn contributed to ExaDG's final report (Arndt, Fehn, et al., 2020) and to (Kronbichler, Kormann, Fehn, et al., 2019) as a Ph.D. student at the Technical University of Munich.

I implemented tensor product Schwarz smoothers with a strong focus on high performance. My academic software package `TPSS`[5] is built upon the general-purpose finite element library `deal.II`, in particular, making use of linear algebra and finite element classes as well as the multigrid and matrix-free framework therein. `TPSS` adds new features upon `deal.II`:

- cost-efficient additive and multiplicative (overlapping) tensor product Schwarz smoothers

[5]The software package, see https://github.com/jwitte08/TPSS, will not be maintained in the future. Note that it lacks any documentation beyond the README and inline documentation. Nevertheless, it may provide a basis for interested researchers that follow the author with the same passion for the thesis's topics.

  - on Cartesian meshes or generic meshes using automated hyper-rectangles for approximation (Witte et al., 2021)
  - with support for `deal.II`'s H^1- and L^2-conforming $\mathbb{Q}_k$-based finite elements (including the Hermite-like finite element (Kronbichler, Kormann, Fehn, et al., 2019)), and the development of a new (moment-based) Raviart-Thomas element making use of the anisotropic tensor structure,
  - with support for shared-memory task and/or memory-distributed parallelism (via MPI), using SIMD vectorization over subdomains, and
  - with user-defined and graph-based[6] coloring algorithms for additive and multiplicative methods

- restricted additive Schwarz smoothers[6]
- a highly-optimized tensor product matrix class enabling fast diagonalization and sum factorization
- a Kronecker product singular value decomposition utilizing SIMD capabilities
- an (approximative) Gaussian block elimination (Arndt, Fehn, et al., 2020, §5.3)

An extensive list of `TPSS`'s features is found in Remark 3.4.1. In Chapters 3 to 5, we elaborate which features have been used for numerical experiments.

Finally, let me **outline** the thesis' structure according to the quick reference's enumeration. Local DG discretizations of the Laplacian preserve the operator's separability on Cartesian meshes. Thus, I choose them as an entry point to study their computational complexities. In addition, I demonstrate that approximating generic-shaped subdomains through hyper-rectangles is still sufficient in obtaining efficient smoothers. The results for Item (i) and Item (i)(a) are summarized in Sections 3.1 and 3.4.1, and the smoothers' parallel performance (Item (i)(b)) is analyzed in Section 3.4.2, which was a missing piece of (Witte et al., 2021). Similar studies are found in Section 3.2 for H^1-conforming discretizations: in Section 3.4.2, we see that the respective MVS performs best in computational terms although lagging a bit behind in mathematical terms. Moreover, the smoothers for H^1-conforming elements gain more importance given hybrid multigrid methods (Fehn, Munch, et al., 2020). Motivated by the deficient performance of AVS with relaxation, I take an excursion (Item (i)(c)) on restricted additive Schwarz smoothers (RAVS). In Section 3.3, high mathematical efficiency for RAVS using two simple weighting strategies is demonstrated.

In Section 4.1.2, I design the efficient algorithms from Item (ii). The exact local solvers are not amenable to fast diagonalization. To this end, inexact local solvers are developed that are both mathematically sound and cost-efficient. In Sections 5.2 and 5.3, I develop a

[6] not fully supporting MPI

novel multilevel Schwarz method (Item (iii)) for (simplified) Stokes flow that goes beyond purely algorithmic development (Item (iii)(b)). Leveraging the equivalence of C^0-IP and $\boldsymbol{H}^{\mathrm{div}}$-IP discretizations, I make use of local stream function approximations from Section 4.1.2 for subspace correction: they are cheap to compute and mathematically robust. For their practical application, a pressure post-processing and transfer operators (Item (iii)(a)) are developed in Section 5.2.3.

Chapter 2

PRELIMINARIES

In this chapter, we introduce the fundamental concepts and prepare the mathematical tools that are utilized in subsequent chapters to develop and analyze tensor product Schwarz smoothers for various partial differential equations (PDE). The chapter starts with introducing tensor product methods that will constitute the heart of our tensor product Schwarz smoothers, reducing the computational complexity of algorithms by one or more orders of magnitude. The first section is subdivided into brief introductions of tensor products and spaces laying out the notational baseline of this monograph (Section 2.1), the sum factorization technique utilized to apply tensor-structured forward and inverse finite element operators in a highly-optimized way (Section 2.1.2), the fast diagonalization enabling fast inversion of matrices with specific tensor structure (Section 2.1.3), and the Kronecker product singular value decomposition (KSVD) computing the best rank-r order-2 tensor product approximation of any matrix (Section 2.1.4). While sum factorization and fast diagonalization constitute the kernel of every tensor product Schwarz smoother developed in this work, the KSVD enables inexact but efficient local stream function solvers in Chapters 4 and 5.

In Section 2.2 matrix-free operator evaluation is introduced: based on the standard tensor product finite elements and quadrature formulas (Section 2.2.1) we exemplify efficient operator application regarding finite element matrices on the basis of the Laplacian, in particular, the importance of sum factorization is stressed, including the evolution of matrix-free methods up to the state of the art in (prospective) Exascale computing. Setting the baseline of computational effort and memory intensity for state-of-the-art PDE solvers on modern hardware, matrix-free operator evaluation was a major motivation to start developing our tensor product Schwarz smoothers.

Multilevel Schwarz methods, i.e., geometric multigrid methods using Schwarz smoothers, are concisely introduced in Section 2.3. It starts with Section 2.3.1, summarizing the main components of geometric multigrid methods and presenting the V-cycle algorithm that is utilized in subsequent chapters. Afterward, Schwarz smoothers are elaborated in Section 2.3.2, being the multigrid component of major interest for this work. We put particular emphasis on

explaining why vertex patches are crucial for the biharmonic and Stokes problem, the coloring of additive and multiplicative smoothers to enable parallelization at all, and discussing the most central assumptions for the convergence analysis of Schwarz smoothers.

Note that all (sub-)sections in this chapter are condensed to its very minimum, discussing only the common concepts and establishing consistent notation for the application-related chapters. The fundamental concepts and methods used in this monograph, including finite element methods, are well-studied. Thus, we will refer to respective textbooks and literature for details in-place. Subsequent chapters will contribute by developing, analyzing, and implementing tensor product Schwarz smoothers to obtain efficient PDE solvers, going beyond the scope of these preliminaries. The intrinsic subtleties and technicalities, e.g., induced by the underlying PDE, are elaborated in the respective chapter

2.1 Tensor Products and Methods

The essential tensor product methods utilized in this monograph need a brief introduction of tensor algebra and calculus, in particular, (algebraic) tensor products of vectors and matrices in Section 2.1.1. We stick to the overview in (Hackbusch, 2012, 2014) and references therein. In most cases, it suffices to understand tensors as multidimensional arrays. We primarily use them in the context of coefficient spaces arising from numerical discretizations. The tensor product techniques *sum factorization*, *fast diagonalization*, and the *Kronecker product singular value decompositions* are motivated and presented in subsequent Sections 2.1.2 to 2.1.4.

2.1.1 Tensor Products and Spaces

Tensor notation, in particular multi-indexing, will ease the presentation of mathematical tools and algorithms exploiting tensor structure for arbitrary spatial dimensions. In this section, a brief overview of tensors and tensor products is given. We recommend reading (Hackbusch, 2012, §3) for a more detailed definition in the context of multilinear algebra.

While vectors have single-indexed entries v_i and matrices have double-indexed entries A_{ij} tensors carry D indices, where D defines the tensor *order*. We refer to the d-th index as *direction* or *dimension* d. The names originate from functions in D-dimensional spatial coordinates $\boldsymbol{x} = (x_1, \dots, x_D)$. For each dimension d, let n_d denote the finite number of sub-dimensions. The definition of scalar multiplication $\alpha\mathfrak{v}$ and addition $\mathfrak{v} + \mathfrak{w}$ of two tensors is straighforward; thus, the set of these tensors has the algebraic structure of a vector space over the field $\mathbb{R}$. This set of tensors is denoted $\mathbb{R}^{n_1 \times \dots \times n_D}$. Mathematically, we define the *tensor product* of vectors as follows.

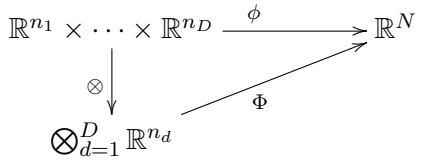


Fig. 2.1 Illustration of Definition 2.1.1 and Proposition 2.1.2 as commutative diagram.

Definition 2.1.1 (Tensor product)**.** Let $n_d < \infty$ for $d = 1, \ldots, D$. For vectors $v^{(d)} \in \mathbb{R}^{n_d}$, $d = 1, \ldots, D$, we define the tensor product

$$\mathfrak{v} := v^{(1)} \otimes v^{(2)} \otimes \ldots \otimes v^{(D)} \tag{2.1a}$$

entrywise by

$$\mathfrak{v}_{i_1,\ldots,i_D} = v^{(1)}_{i_1} v^{(2)}_{i_2} \cdots v^{(D)}_{i_D}, \tag{2.1b}$$

for all $i_d = 1, \ldots, n_d$ and $d = 1, \ldots, D$.

The tensor space is accordingly defined by

$$\bigotimes_{d=1}^{D} \mathbb{R}^{n_d} := \operatorname{span}\left\{ v^{(1)} \otimes \ldots \otimes v^{(D)} \mid v^{(d)} \in \mathbb{R}^{n_d},\ d = 1, \ldots, D \right\}, \tag{2.2}$$

that is the span over so-called *elementary tensors* $v^{(1)} \otimes \ldots \otimes v^{(D)}$. The "universality of the tensor product" characterizes the relation between vector spaces $\mathbb{R}^{n_d}, d = 1, \ldots, D$, and $\mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_D}$.

Proposition 2.1.2 (Universality of the tensor product)**.** *Let $n_d < \infty$ for $d = 1, \ldots, D$ and $N = \prod_{d=1}^{D} n_d$. Then, for any multilinear mapping $\phi\colon \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_D} \to \mathbb{R}^N$, there is a unique linear mapping $\Phi\colon \bigotimes_{d=1}^{D} \mathbb{R}^{n_d} \to \mathbb{R}^N$ such that*

$$\phi(v^{(1)}, \ldots, v^{(D)}) = \Phi(v^{(1)} \otimes \ldots \otimes v^{(D)}) \tag{2.3}$$

for all $v^{(d)} \in \mathbb{R}^{n_d}$.

Proof. See (Hackbusch, 2012, Proposition 3.22). □

Summarizing previous properties, we obtain a characterization of the (algebraic) tensor space.

Remark 2.1.3 (Characterization of the tensor space). Let $n_d < \infty$ for $d = 1, \ldots, D$. T is the (algebraic) tensor space $\bigotimes_{d=1}^{D} \mathbb{R}^{n_d}$ if there holds

1. span property:

$$T = \operatorname{span}\left\{ v^{(1)} \otimes \ldots \otimes v^{(D)} \mid v^{(d)} \in \mathbb{R}^{n_d} \right\},$$

2. multilinearity: for any $\alpha, \beta \in \mathbb{R}$, $v^{(d)}, w^{(d)} \in \mathbb{R}^{n_d}$, and any $d \in \{1, \dots, D\}$ it holds

$$v^{(1)} \otimes \cdots \otimes \left(\alpha v^{(d)} + \beta w^{(d)}\right) \otimes \cdots \otimes v^{(D)} \\ = \alpha v^{(1)} \otimes \cdots \otimes v^{(d)} \otimes \cdots \otimes v^{(D)} + \beta v^{(1)} \otimes \cdots \otimes w^{(d)} \otimes \cdots \otimes v^{(D)}, \quad (2.4)$$

3. inheritance: if for each direction $d = 1, \dots, D$ vectors $v_1^{(d)}, \dots, v_{n_d}^{(d)}$ are linearly independent, then $v_{i_1}^{(1)} \otimes \ldots \otimes v_{i_D}^{(D)}, i_d = 1, \dots, n_d$, are linearly independent in T.

The dimension of tensor space $\bigotimes_{d=1}^{D} \mathbb{R}^{n_d}$ is the product of sub-dimensions $n_1 n_2 \cdots n_D$. From the span property and taking all linear combinations of unit vectors in $\mathbb{R}^{n_d}$ there holds

$$\bigotimes_{d=1}^{D} \mathbb{R}^{n_d} = \mathbb{R}^{n_1 \times \cdots \times n_D}. \quad (2.5)$$

To be precise, "=" in (2.5) is well-defined as isomorphism between vector spaces that preserve the algebraic vector space properties. We use $\mathbb{R}^{n_1 \times \cdots \times n_D}$ in the context of multidimensional arrays or D-way arrays. In contrast, we use $\bigotimes_{d=1}^{D} \mathbb{R}^{n_d}$ when explicitly using elementary tensors or linear combinations of them. There exists an important difference between vector space isomorphisms and *tensor space isomorphisms*, as detailed in (Hackbusch, 2012, §3.2.5).

Definition 2.1.4 (Tensor space isomorphism)**.** A tensor space isomorphism is any bijection $\Phi = \bigotimes_{d=1}^{D} \phi_d$, that is the tensor product of vector space isomorphisms ϕ_d.

A tensor space isomorphism considers the number of directions (tensor order), the ordering of directions, and the dimension of each direction. For instance, the tensor spaces $\mathbb{R}^2 \otimes \mathbb{R}^6$, $\mathbb{R}^4 \otimes \mathbb{R}^3$ and $\mathbb{R}^2 \otimes \mathbb{R}^2 \otimes \mathbb{R}^3$ are isomorphic in the sense of vector spaces, all being vector space isomorphic to $\mathbb{R}^{12}$. However, they differ in tensor order, or the dimension of at least one tensor direction varies; thus, resulting in different tensor spaces in terms of characterizations from Remark 2.1.3. We shall remember that any tensor space is a vector space, but vector space isomorphisms between tensor spaces do not carry information on the tensor structure.

Definition 2.1.5 (Tensor rank)**.** For $n_d < \infty$ with $d = 1, \dots, D$, we define the set of linear combinations of elementary tensors

$$\mathcal{R}_r := \left\{ \sum_{\nu=1}^{r} v^{(1)} \otimes \ldots \otimes v^{(D)} \mid v^{(d)} \in \mathbb{R}^{n_d} \right\} \quad (2.6)$$

for any $r \in \mathbb{N}$. The *tensor rank* of $\mathfrak{v} \in \bigotimes_{d=1}^{D} \mathbb{R}^{n_d}$ is defined as

$$\operatorname{rank} \mathfrak{v} := \min \left\{ r \mid \mathfrak{v} \in \mathcal{R}_r \right\}. \quad (2.7)$$

The minimum in (2.7) is well-defined since subsets of $\mathbb{N}$ have a minimum. We refer to tensors with rank r as rank-r tensors and, in particular, to elementary tensors as rank-1

tensors. It is essential to notice that the tensor rank is (only) invariant under tensor space isomorphisms.

While in classical finite element codes tensor structures are sometimes ignored, the algorithms in this work heavily make use of them. The tensor structure plays a crucial role in the context of data structures: vectors, matrices, or multidimensional arrays are usually implemented as contiguous, linear storage in scientific code. To access an element by its multi-index the memory layout has to be known. For instance, row-wise consecutive elements in a matrix with row-major order are contiguous in memory (G. Karniadakis, G. E. Karniadakis, et al., 2003). In other words, the row index strides with the number of columns as step width through the linear storage and, thus, "runs faster" than the column index. This ordering is easily generalized to higher dimensions than two.

Definition 2.1.6 (Lexicographic ordering)**.** Let $n_d < \infty$ for $d = 1, \dots, D$ and $N = \prod_{d=1}^{D} n_d$. We define a bijective mapping $\varphi_{lxc}\colon \{1, \dots, n_1\} \times \cdots \times \{1, \dots, n_D\} \to \{1, \dots, N\}$ between index sets by means of the lexicographic ordering

$$\varphi_{lxc}(i_1, \dots, i_D) = 1 + \prod_{d=1}^{D} (i_d - 1)\underline{n}_d \tag{2.8}$$

for all $i_d = 1, \dots, n_d$ with "collapsed" sub-dimensions

$$\underline{n}_D = 1 \quad \text{and} \quad \underline{n}_d = \prod_{k=d+1}^{D} n_k,\ d = 1, \dots, D-1. \tag{2.9}$$

We will use the shorter notation $\boldsymbol{i}$ instead of $\varphi_{lxc}(i_1, \dots, i_D)$ for the unrolled multi-index $i_1, \dots, i_D$.

Taking the contiguous memory layout subject to lexicographic ordering into account, caching and efficient access patterns are achieved (Kronbichler and Kormann, 2012; Kronbichler, Kormann, Fehn, et al., 2019; Kronbichler and Kormann, 2019). We postpone this discussion for now: our implementations are elaborated later.

For `deal.II` users or interested readers of our article (Witte et al., 2021) we emphasize:

Remark 2.1.7 (Lexicographic ordering). In `deal.II` implementations, the x-coordinate "runs slowest" such that indices concerning the first dimension (for example i_1) stride with step width one. In (Witte et al., 2021) we used a multi-indexing and ordering close to `deal.II`'s source code,

1. index ranges start with zero ($i_d = 0$ or $\boldsymbol{i} = 0$)
2. multi-indexing is subject to *colexicographic ordering*

In this monograph, we follow a multi-indexing and ordering that is more natural in mathematical terms,

1. index ranges start with one ($i_d = 1$ or $\boldsymbol{i} = 1$)
2. multi-indexing is subject to *lexicographic ordering*

Taking the lexicographic ordering into account the *vectorization* $\operatorname{vect}\colon \mathbb{R}^{n_1 \times \cdots \times n_D} \to \mathbb{R}^N$ of a tensor $\mathfrak{v}$ is defined entrywise by

$$v_{\boldsymbol{i}} := \operatorname{vect}(\mathfrak{v})_{\varphi_{lxc}(i_1,\ldots,i_D)} := \mathfrak{v}_{i_1,\ldots,i_D}, \quad 1 \le i_d \le n_d. \tag{2.10}$$

The vectorization is a vector space isomorphism, in particular, isometric with respect to the Frobenius norm. The tensor structure is therefore not taken into account. In algorithms and data structures, vectorization is often called "flattening" of a multidimensional data array. The collapsed sub-dimension $\underline{n}_d$ defines the step width required when striding through the flattened array with all indices fixed except i_d.

The reverse operation is called *tensorization*: a vector is isomorphically transformed into a tensor by imposing a fitting tensor structure. Formally, the tensorization $\operatorname{tens}\colon \mathbb{R}^N \to \mathbb{R}^{n_1 \times \cdots \times n_D}$ reads

$$\mathfrak{v}_{i_1,\ldots,i_D} := \operatorname{tens}(v)_{i_1,\ldots,i_D} := v_{\varphi_{lxc}(i_1,\ldots,i_D)}, \quad 1 \le i_d \le n_d, \tag{2.11}$$

for sub-dimensions n_d satisfying $N = \prod_{d=1}^{D} n_d$. The tensorization, being the inverse vector space isomorphism of the vectorization, does not carry any information on the particular tensor structure, albeit called "tensorization". Therefore, for finite dimension $N < \infty$ each tensorization has a non-unique representation as linear combination of rank-1 tensors (with finite tensor rank r). As indicated by (2.10) and (2.11), we use from now on different fonts to abbreviate notation: $\mathfrak{v}$ denotes the tensorization of vector v and vice versa v denotes the vectorization of tensor $\mathfrak{v}$.

We define the tensor product of matrices, which is often called *Kronecker product*, in terms of tensors of linear mappings.

Definition 2.1.8 (Tensor product of matrices)**.** Let $m_d < \infty$ and $n_d < \infty$ for $d = 1, \ldots, D$, respectively. For linear mappings $A^{(d)}\colon \mathbb{R}^{n_d} \to \mathbb{R}^{m_d}$, $d = 1, \ldots, D$, we define the tensor product

$$\mathfrak{A} := \bigotimes_{d=1}^{D} A^{(d)}\colon \bigotimes_{d=1}^{D} \mathbb{R}^{n_d} \to \bigotimes_{d=1}^{D} \mathbb{R}^{m_d} \tag{2.12a}$$

by

$$\mathfrak{A}(v^{(1)} \otimes \ldots \otimes v^{(D)}) = A^{(1)}v^{(1)} \otimes \ldots \otimes A^{(D)}v^{(D)} \tag{2.12b}$$

for all $v^{(1)} \otimes \ldots \otimes v^{(D)} \in \bigotimes_{d=1}^{D} \mathbb{R}^{n_d}$.

Since $\bigotimes_{d=1}^{D} \mathbb{R}^{n_d}$ is spanned by elementary tensors $v^{(1)} \otimes \ldots \otimes v^{(D)}$ a tensor of linear mappings is uniquely determined by (2.12b). In the context of matrices, the tensor product

space is denoted

$$\bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d} := \operatorname{span}\left\{ \bigotimes_{d=1}^{D} A^{(d)} \mid A^{(d)} \in \mathbb{R}^{m_d \times n_d},\ d = 1, \dots, D \right\}, \tag{2.13}$$

which exemplifies the structure of a tensor product between matrices. We refer to this space as *Kronecker product space* and to its rank-1 elements as *Kronecker tensors*. Similar to vectors and tensors of vectors, it is readily proven that there exists a vector space isomorphism between the Kronecker product space $\bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d}$ and matrix space $\mathbb{R}^{M \times N}$ with factorizations $N = \prod_{d=1}^{D} n_d$ and $M = \prod_{d=1}^{D} m_d$. Using the lexicographic ordering (2.8) the so-called *Kronecker product* between two matrices is defined as follows.

Definition 2.1.9 (Kronecker product)**.** Let $A^{(d)} \in \mathbb{R}^{m_d \times n_d}$ for $d = 1, 2$, then, the Kronecker product $A^{(1)} \otimes A^{(2)}$ is a $\mathbb{R}^{(m_1 m_2) \times (n_1 n_2)}$-matrix such that

$$A^{(1)} \otimes A^{(2)} = \begin{bmatrix} A^{(1)}_{11} A^{(2)} & \dots & A^{(1)}_{1 n_1} A^{(2)} \\ \vdots & & \vdots \\ A^{(1)}_{m_1 1} A^{(2)} & \dots & A^{(1)}_{m_1 n_1} A^{(2)} \end{bmatrix}. \tag{2.14}$$

In other words, the Kronecker product between a $m_1 \times n_1$- and $m_2 \times n_2$-matrix is a $m_1 \times n_1$-block matrix with blocks in $\mathbb{R}^{m_2 \times n_2}$ using lexicographic ordering of rows and columns, respectively. For tensor orders higher than two a recursive block structure is obtained: the respective vector isomorphism $\Phi\colon \bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d} \to \mathbb{R}^{M \times N}$ is defined entrywise by

$$A_{\boldsymbol{ij}} := \Phi\Big(\bigotimes_{d=1}^{D} A^{(d)} \Big)_{\varphi_{lxc}(i_1, \dots, i_D), \varphi_{lxc}(j_1, \dots, j_D)} := \prod_{d=1}^{D} A^{(d)}_{i_d j_d} \tag{2.15}$$

for matrices $A^{(d)} \in \mathbb{R}^{m_d \times n_d}$, where $\boldsymbol{i}$ and $\boldsymbol{j}$ is the short notation for $\varphi_{lxc}(i_1, \dots, i_D)$ and $\varphi_{lxc}(j_1, \dots, j_D)$, respectively. Each tensor direction d of the order-D Kronecker product represents a "level" of the block structure. We refer to both, $\Phi(\bigotimes_{d=1}^{D} A^{(d)})$ and $\bigotimes_{d=1}^{D} A^{(d)}$, as order-$D$ *Kronecker product* and will omit writting $\Phi(\cdot)$ from now on but it becomes clear from context if $\bigotimes_{d=1}^{D} A^{(d)}$ is understood as element in $\mathbb{R}^{M \times N}$ or $\bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d}$, respectively. A multi-indexing subject to lexicographic ordering leads to Kronecker factors in (2.15) with increasing tensor dimension from left to right, for instance $A^{(1)} \otimes A^{(2)} \otimes A^{(3)}$. In contrast, using a colexicographic ordering as in (Witte et al., 2021), factors are unnaturally indexed from right to left, i.e., $A^{(3)} \otimes A^{(2)} \otimes A^{(1)}$.

The vector space isomorphism Φ is analogous to the vectorization $\operatorname{vect}(\cdot)$ which is introduced in (2.10). We emphasize that Φ is defined only for rank-1 tensors of matrices in (2.15). Nevertheless, due to its linearity and the span property of $\bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d}$, it is well-defined for any tensor. We deliberately do not introduce the term "matricization" here to avoid confusing the well-known tensor algebra concept of "matricization", also known as

"unfolding" or "flattening" (Kolda and Bader, 2009). Throughout this monograph, we will use Kronecker tensors, which means rank-1 tensors of matrices or linear combinations of them with low tensor rank; thus, in contrast to tensors of vectors, we will make no use of the space $\mathbb{R}^{(m_1 \times n_1) \times \cdots \times (m_D \times n_D)}$ which is vector isomorphic to $\bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d}$. As indicated in (2.12) and (2.15) we write in short A for the "vectorization" of Kronecker tensors $\mathfrak{A}$ and vice versa $\mathfrak{A}$ for the tensorization of A. Finally, we list some properties of the Kronecker product used in subsequent chapters.

Lemma 2.1.10 (Properties of Kronecker products)**.** *Given matrices A, B, C and D of appropriate sizes, in general, it holds*

$$(A \otimes B) \neq (B \otimes A), \tag{2.16a}$$

$$(A \otimes B)(C \otimes D) = AC \otimes BD, \tag{2.16b}$$

$$(A \otimes B)^{\mathsf{T}} = A^{\mathsf{T}} \otimes B^{\mathsf{T}}, \tag{2.16c}$$

$$(A \otimes B) \otimes C = A \otimes B \otimes C = A \otimes (B \otimes C) \tag{2.16d}$$

and, $A \otimes B$ is invertible if and only if A and B are invertible and it holds

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}. \tag{2.16e}$$

Proof. See (Van Loan, 2000) and references therein. □

In particular, from (2.16b) and (2.16c) it follows that $Q \otimes P$ is orthogonal if and only if Q and P are orthogonal. The generalizations of properties (2.16a) to (2.16c) are straightforward for D-ary Kronecker products. For instance, the transpose of Kronecker tensor $\mathfrak{A} = \bigotimes_{d=1}^{D} A^{(d)}$ is well-defined by successively using (2.16c),

$$\mathfrak{A}^{\mathsf{T}} = \bigotimes_{d=1}^{D} \left(A^{(d)}\right)^{\mathsf{T}}. \tag{2.17}$$

It is worth noticing that the transpose operation does not revert the ordering of matrix factors in the Kronecker product, as it does for the matrix-matrix multiplication. Kronecker products are extensively studied in (G. H. Golub and Van Loan, 2013; Graham, 1981; Van Loan, 2000).

2.1.2 Sum Factorization

Sum factorization denotes the fundamental concept in which a multi-indexed sum is efficiently refactored into sums carrying only a single index. Refactoring is possible whenever a certain degree of separation of variables is present. Sum factorization is crucial for efficient operator applications regarding both, the discretizations of "forward" and inverse differential operators.

Utilizing sum factorization for matrix-free operator evaluations dates back to Orszag (1980). Its background and usage in the context of matrix-free operators is detailed in Section 2.2.

Introducing the d-mode product between a matrix and a tensor allows us to concisely discuss sum factorization arising from a Kronecker product matrix multiplied with a tensor of vectors.

Definition 2.1.11 (d-mode product)**.** Let $n_\delta < \infty$ for $\delta = 1, \dots, D$ and $m_d < \infty$ for fixed direction $d \in \{1, \dots, D\}$. For a tensor $\mathfrak{v} \in \mathbb{R}^{n_1 \times \cdots \times n_D}$ and a matrix $A \in \mathbb{R}^{m_d \times n_d}$ we define the d-mode product

$$A \times_d \mathfrak{v} \in \mathbb{R}^{n_1 \times \cdots \times n_{d-1} \times m_d \times n_{d+1} \times \cdots \times n_D} \tag{2.18a}$$

entrywise by

$$(A \times_d \mathfrak{v})_{i_1,\dots,i_D} = \sum_{j=1}^{n_d} A_{i_d j} \, v_{i_1,\dots,i_{d-1},j,i_{d+1},\dots,i_D} \tag{2.18b}$$

for $i_d = 1, \dots, m_d$ and $i_k = 1, \dots, n_k$, $k \neq d$.

Given two dimensions d and e with $d \neq e$ consecutive mode products with a common tensor

$$A \times_d B \times_e \mathfrak{v} = A \times_d (B \times_e \mathfrak{v}) = B \times_e (A \times_d \mathfrak{v}) \tag{2.19}$$

commute for matrices A, B and tensor $\mathfrak{v}$ of appropriate size (De Lathauwer et al., 2000). While in mathematical terms, the order of mode products is irrelevant from an algorithmic viewpoint, an ordering close to or even matching the memory layout improves computational runtime (Kronbichler, Kormann, Fehn, et al., 2019; Kronbichler and Kormann, 2019).

For simplicity, we assume isotropic tensor directions meaning $n = n_1 = \cdots = n_D$ and $N = n^D$. Let $\mathfrak{v}$ denote a tensor in $\mathbb{R}^{n \times \cdots \times n}$ and $\mathfrak{A} = \bigotimes_{d=1}^{D} A^{(d)}$ a tensor of matrices in $\bigotimes_{d=1}^{D} \mathbb{R}^{n \times n}$ with vectorization $v \in \mathbb{R}^N$ and Kronecker product $A \in \mathbb{R}^{N \times N}$, respectively. The naïve matrix-vector multiplication Av requires $\mathcal{O}(n^{2D})$ floating point operations. Taking tensor structures into account, the matrix-vector multiplication of "matrix" $\mathfrak{A}$ with "vector" $\mathfrak{v}$ is equivalent to Av and expressed in terms of D consecutive d-mode products,

$$\mathfrak{A}\mathfrak{v} = A^{(1)} \times_1 \dots A^{(D-1)} \times_{D-1} A^{(D)} \times_D \mathfrak{v}. \tag{2.20a}$$

From successive mode products (2.20a) we obtain a natural factorization of D sum factors, each corresponding to the contraction over a single tensor direction, referred to as *sum*

factorization. The matrix-vector product (2.20a) is efficiently evaluated as follows

$$\begin{aligned}
(\mathfrak{A}\mathfrak{v})_{i_1,\dots,i_D} &= \sum_{j_1=1}^{n} A^{(1)}_{i_1 j_1} \cdots \sum_{j_{D-1}=1}^{n} A^{(D)}_{i_{D-1} j_{D-1}} \sum_{j_D=1}^{n} A^{(D)}_{i_D j_D} \mathfrak{v}_{j_1,\dots,j_D} \\
&= \sum_{j_1=1}^{n} A^{(1)}_{i_1 j_1} \cdots \sum_{j_{D-1}=1}^{n} A^{(D)}_{i_{D-1} j_{D-1}} \mathfrak{v}^{[1]}_{j_1,\dots,j_{D-1},i_D} \\
&\vdots \\
&= \sum_{j_1=1}^{n} A^{(1)}_{i_1 j_1} \mathfrak{v}^{[D-1]}_{j_1,i_2,\dots,i_D} \\
&= \mathfrak{v}^{[D]}_{i_1,\dots,i_D}
\end{aligned} \tag{2.20b}$$

Starting in (2.20b) with the first D-mode product, an intermediate tensor $\mathfrak{v}^{[1]}$ is computed. This involves D fixed indices $j_1, \dots, j_{D-1}$ and i_D as well as one contraction index j_D of the sum factor. Therefore, computing the D-mode product $\mathfrak{v}^{[1]}$ we require $\mathcal{O}(n^{D+1})$ floating-point operations. Proceeding similarly for the $D-1$ remaining sum factors, the matrix-vector multiplication $\mathfrak{A}\mathfrak{v}$ is evaluated at the cost of $\mathcal{O}(Dn^{D+1})$ arithmetic operations.

Similarly, a rank-r tensor product matrix $\mathfrak{A} = \sum_{\nu=1}^{r} \bigotimes_{d=1}^{D} A^{(d)}_{\nu}$ is multiplied with $\mathfrak{v}$ at the cost of $\mathcal{O}(rDn^{D+1})$ floating-point operations. Therefore, vectors can be efficiently applied to matrices with inherent low-rank tensorization. The same holds for the transpose of matrix A due to (2.17). Sum factorization is easily generalized to anisotropic tensors (including non-square matrices) and computational costs are given as above with n being the maximum of all sub-dimensions,

$$n = \max\{n_1, \dots, n_D, m_1, \dots, m_D\},$$

where n_d and m_d denote the number of rows and of columns for each direction d, respectively.

2.1.3 Fast Diagonalization

While sum factorization reduces the complexity of the "forward" problem (that is, matrix-vector products), we also want to compute and apply the inverse of a low-rank tensor product matrix at reduced computational costs by taking tensor structure into account. To this end, we discuss a technique called *fast diagonalization*, first introduced in (Lynch et al., 1964), that efficiently inverts matrices with a specific low-rank tensor representation.

Definition 2.1.12 (Separable Kronecker representation)**.** Let $n_1, \dots, n_D, m_1, \dots, m_D < \infty$. For rank-1 tensors $\mathfrak{A} = \bigotimes_{d=1}^{D} A^{(d)}$ and $\mathfrak{M} = \bigotimes_{d=1}^{D} M^{(d)}$ with $A^{(d)}$ and $M^{(d)}$ in $\mathbb{R}^{m_d \times n_d}$, the separable Kronecker representation reads

$$S := [\mathfrak{A} \boxtimes \mathfrak{M}] := \sum_{\nu=1}^{D} \mathfrak{B}_{\nu} \tag{2.21a}$$

with rank-1 tensors $\mathfrak{B}_\nu \in \bigotimes_{d=1}^{D} \mathbb{R}^{m_d \times n_d}$ defined by

$$B_\nu^{(d)} = \begin{cases} A^{(\nu)}, & \text{if } d = \nu \\ M^{(d)}, & \text{otherwise} \end{cases} \tag{2.21b}$$

for all $\nu = 1, \dots, D$.

In (2.21a) $S \in \mathbb{R}^{MxN}$ defines the "vectorization" of the sum of Kronecker tensors in the sense of the vector isomorphism (2.15). We continue to tacitly use roman letters for the vectorization of tensors, and Fraktur letters as abbreviation for rank-1 tensors of matrices, for instance, $\mathfrak{I}$ denotes the tensor of identity matrices $\bigotimes_{d=1}^{D} I^{(d)}$ in short from. We note that the separable Kronecker representation has tensor rank D. The definition (2.21a) is concise for any number of dimensions but seems cryptic at first sight; thus, let us clarify the "separable" tensor structure for three dimensions,

$$[\mathfrak{A} \boxtimes \mathfrak{I}] = A^{(1)} \otimes I^{(2)} \otimes I^{(3)} + I^{(1)} \otimes A^{(2)} \otimes I^{(3)} + I^{(1)} \otimes I^{(2)} \otimes A^{(3)}, \tag{2.22}$$

In analogy to separable differential operators, see (Hackbusch, 2012, §1.2.4.1), the "action" of rank-1 tensor $\mathfrak{A}$ is separated between tensor directions. For instance, the separable representation (2.22) is obtained for finite difference discretizations of the Laplacian, which is a separable differential operator (Lynch et al., 1964).

For simplicity, assume isotropic tensor structures $n = n_1 = \cdots = n_D = m_1 = \cdots = m_D$. Furthermore, assume matrices $A^{(d)}$ are symmetric and matrices $M^{(d)}$ are symmetric, positive definite for $d = 1, \dots, D$. Then, we have D well-defined generalized eigenvalue problems[1] such that

$$\begin{aligned} (Z^{(d)})^\mathsf{T} A^{(d)} Z^{(d)} &= \Lambda^{(d)}, \\ (Z^{(d)})^\mathsf{T} M^{(d)} Z^{(d)} &= I^{(d)}. \end{aligned} \tag{2.23}$$

The columns of matrix $Z^{(d)}$ are the juxtaposed generalized eigenvectors and the diagonal matrix $\Lambda^{(d)}$ carries the corresponding generalized eigenvalues, respectively, for each direction d. Using the Kronecker product properties in Lemma 2.1.10, the separable Kronecker representation (2.21) is transformed into diagonal form

$$\mathfrak{Z}^\mathsf{T} [\mathfrak{A} \boxtimes \mathfrak{M}] \mathfrak{Z} = [\mathfrak{L} \boxtimes \mathfrak{I}] \tag{2.24a}$$

with rank-1 tensors $\mathfrak{Z} = \bigotimes_{d=1}^{D} Z^{(d)}, \mathfrak{L} = \bigotimes_{d=1}^{D} \Lambda^{(d)}$ and $\mathfrak{I} = \bigotimes_{d=1}^{D} I^{(d)}$. The latter is the Kronecker tensor of identity matrices. Let Z and Λ denote the vectorization of $\mathfrak{Z}$ and $\mathfrak{L}$, respectively, then

$$Z^\mathsf{T} S Z = \Lambda \tag{2.24b}$$

[1] See (Saad, 2011).

is equivalent to (2.24a) in the sense of vector space isomorphism Φ introduced in (2.15). S was defined before in (2.21). It becomes apparent that the columns of Z are eigenvectors of S with corresponding eigenvalues given by the diagonal matrix Λ. Therefore, the inverse of S has the form

$$S^{-1} = Z\Lambda^{-1}Z^{\mathsf{T}}, \tag{2.25}$$

and is well-defined by the following lemma.

Lemma 2.1.13. *Let $n_1, \dots, n_D < \infty$ and $A^{(d)}$ and $M^{(d)}$ in $\mathbb{R}^{n_d \times n_d}$ be simultaneously diagonalizable matrices, i.e.,*

$$(Z^{(d)})^{-1} A^{(d)} Z^{(d)} = \Lambda^{(d)} \quad \text{and} \quad (Z^{(d)})^{-1} M^{(d)} Z^{(d)} = I^{(d)} \tag{2.26}$$

with identity matrices $I^{(d)}$ and generalized eigenvalues given by $\Lambda^{(d)} = \operatorname{diag}(\lambda_1^{(d)}, \dots, \lambda_{n_d}^{(d)})$ for $d = 1, \dots, D$. If

$$\sum_{d=1}^{D} \lambda_{i_d}^{(d)} \neq 0 \tag{2.27}$$

for all $1 \leq i_d \leq n_d$, then the Kronecker product S of separable Kronecker representation $[\mathfrak{A} \boxtimes \mathfrak{M}]$ is invertible.

Proof. Let A, M and Λ denote the vectorization of $\mathfrak{A}$, $\mathfrak{M}$ and $\left[\bigotimes_{d=1}^{D} \Lambda^{(d)} \boxtimes \bigotimes_{d=1}^{D} I^{(d)}\right]$, respectively. Following (2.23) to (2.25) we derive

$$S^{-1} = Z\Lambda^{-1}Z^{-1},$$

which is well-defined if $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_N)$ and Z are invertible, where $N = \prod_{d=1}^{D} n_d$. Using the Kronecker product properties in Lemma 2.1.10, Z is regular since all $(Z^{(d)})^{-1}$ are well-defined. From the special tensor structure $\left[\bigotimes_{d=1}^{D} \Lambda^{(d)} \boxtimes \bigotimes_{d=1}^{D} I^{(d)}\right]$ and (2.27), immediately follows

$$\lambda_{\boldsymbol{i}} = \sum_{d=1}^{D} \lambda_{i_d}^{(d)} \neq 0$$

for all $1 \leq i_d \leq n_d$, such that Λ is invertible. □

Naïve algorithms inverting a $N \times N$ matrix require $\mathcal{O}(N^3) = \mathcal{O}(n^{3D})$ arithmetic operations. The inverse S^{-1} in diagonal form (2.25) involves the computation of D generalized eigenvalue problems, each is computed at the cost of $\mathcal{O}(n^3)$ floating point operations. Computing the inverse of diagonal matrix Λ costs the same, namely $\mathcal{O}(n^3)$ operations. Taking the separable tensor structure (2.21) into account, the costs of computing the inverse of S are reduced from $\mathcal{O}(n^{3D})$ floating-point operations to $\mathcal{O}((D+1)n^3)$ which is why we call exploiting the Kronecker product form in (2.25) *fast diagonalization*: the exponential dependency on D is eliminated and so is the "curse of dimensionality".

Not only the computation of S^{-1} profits from the tensor structure also sum factorization (introduced in Section 2.1.2) is applicable when applying S^{-1} to a vector v since S^{-1} is a juxtaposition of Kronecker tensors and a diagonal matrix. Let $\mathfrak{v}$ denote the tensorization of v, then, $Zv = \mathfrak{Z}\mathfrak{v}$ and $Z^\mathsf{T} v = \mathfrak{Z}^\mathsf{T}\mathfrak{v}$ are evaluated each at the cost of $\mathcal{O}(Dn^{D+1})$ arithmetic operations by means of (2.20). The matrix-vector product with diagonal matrix Λ^{-1} requires as many operations as the inner product of two vectors, i.e., $\mathcal{O}(N) = \mathcal{O}(n^D)$ operations, being negligible. Using sum factorization reduces the total computational effort to $\mathcal{O}(Dn^{D+1})$ floating point operations compared to the $\mathcal{O}(n^{2D})$ operations for a naïve evaluation.

Remark 2.1.14 (Computational effort). Given the notation and definitions in (2.23) to (2.25),

1. inverting S costs $\mathcal{O}(Dn^3)$ arithmetic operations and
2. a matrix-vector product with $S^{-1}v$ costs $\mathcal{O}(Dn^{D+1})$,

if fast diagonalization is utilized.

Another benefit which is often forgotten is the significant decrease in memory, that is consumed to store the fast diagonalization efficiently.

Remark 2.1.15 (Memory intensity). The fast diagonalization (2.25) requires to store only Dn^2 and Dn values, arising from the D one-dimensional generalized eigenvectors $Z^{(d)}$ and eigenvalues $\Lambda^{(d)}$, respectively.

2.1.4 Kronecker Product Singular Value Decomposition

For the biharmonic and Stokes problem, we obtain discretization matrices that do not admit the low-rank tensor representations needed for the fast diagonalization technique of previous Section 2.1.3. To this end, we seek the best low-rank approximation with the specific tensor structure needed. Van Loan and N. Pitsianis (1993) studied this problem and referred to it as *nearest Kronecker product problem*. The mathematical tool to compute the best rank-r approximation of any matrix is the *Kronecker product singular value decomposition* (KSVD). We emphasize that the KSVD computes only order-2 tensor product approximations, i.e., it applies only to two-dimensional PDEs in the context of this work. Higher order techniques exist and are still an active field of research, for instance, the tensor train decomposition (Oseledets, 2011) or the multilinear SVD (De Lathauwer et al., 2000). However, none of them enables a fast and direct inversion in higher dimensions compared to fast diagonalization.

The section starts with the problem statement:

Definition 2.1.16 (Nearest Kronecker product problem)**.** Let $A \in R^{M\times N}$ with factorizations $M = m_1 m_2$ and $N = n_1 n_2$. The nearest rank-r Kronecker product problem reads: find matrices $B_\nu \in \mathbb{R}^{m_1\times n_1}$ and $C_\nu \in \mathbb{R}^{m_2\times n_2}$, $\nu = 1,\dots,r$ that minimize

$$\left\| A - \sum_{\nu=1}^{r} B_\nu \otimes C_\nu \right\|_F . \tag{2.28}$$

In theory, a similar minimization problem could be stated for higher tensor orders as well, for instance, seeking an approximation of order-3 Kronecker products that minimizes

$$\left\| A - \sum_{\nu=1}^{r} B_\nu \otimes C_\nu \otimes D_\nu \right\|_F .$$

However, in practice, we see next that the methodology to solve the nearest Kronecker product problem is restricted to tensor order two: a reshuffling of A's matrix entries enables us to postulate an equivalent minimization problem to (2.28) that can be solved utilizing a standard singular value decomposition. To the best of our knowledge, there is no generalization to dimensions higher than two.

We recall that the vectorization $\operatorname{vect}(\cdot)$ transforms any tensor into a column vector based on the lexicographic ordering from Section 2.1.1. In the case of matrices, $\operatorname{vect}(M^\mathsf{T})$ is a column vector obtained by stacking the columns of matrix M.

Definition 2.1.17 (Matrix reshuffling)**.** Given $A \in \mathbb{R}^{m \times n}$ with factorizations $m = m_1 m_2$ and $n = n_1 n_2$ its *matrix reshuffling* $\mathcal{R}(A) \in \mathbb{R}^{(m_1 n_1) \times (m_2 n_2)}$ is defined as

$$\mathcal{R}(A) = \begin{bmatrix} \operatorname{vect}(A_{11}^\mathsf{T})^\mathsf{T} \\ \operatorname{vect}(A_{21}^\mathsf{T})^\mathsf{T} \\ \vdots \\ \operatorname{vect}(A_{m_1 1}^\mathsf{T})^\mathsf{T} \\ \vdots \\ \vdots \\ \operatorname{vect}(A_{m_1 n_1}^\mathsf{T})^\mathsf{T} \end{bmatrix}, \tag{2.29}$$

where matrix blocks $A_{11}, \ldots, A_{m_1 n_1} \in \mathbb{R}^{m_2 \times n_2}$ are determined by the block structure of A,

$$A = \begin{bmatrix} A_{11} & \cdots & A_{1 n_1} \\ \vdots & \ddots & \vdots \\ A_{m_1 1} & \cdots & A_{m_1 n_1} \end{bmatrix}. \tag{2.30}$$

The matrix reshuffling is demonstrated in Figure 2.2: blocks of the 2×3-block matrix A are vectorized into rows, stacking the columns of each block and transposing afterward. Given this matrix reshuffling, the nearest Kronecker product problem (2.28) can be reformulated:

Lemma 2.1.18. *Let $A \in \mathbb{R}^{M \times N}$ with factorizations $M = m_1 m_2$ and $N = n_1 n_2$. For matrices $B_\nu \in \mathbb{R}^{m_1 \times n_1}$ and $C_\nu \in \mathbb{R}^{m_2 \times n_2}$, $\nu = 1, \ldots, r$, it holds*

$$\left\| A - \sum_{\nu=1}^{r} B_\nu \otimes C_\nu \right\|_F = \left\| \mathcal{R}(A) - \sum_{\nu=1}^{r} \operatorname{vect}(B_\nu^\mathsf{T}) \operatorname{vect}(C_\nu^\mathsf{T})^\mathsf{T} \right\|_F . \tag{2.31}$$

$$A = \left[\begin{array}{cc:cc:cc} 1 & 3 & 9 & 11 & 17 & 19 \\ 2 & 4 & 10 & 12 & 18 & 20 \\ \hdashline 5 & 7 & 13 & 15 & 21 & 23 \\ 6 & 8 & 14 & 16 & 22 & 24 \end{array}\right] \qquad \rightsquigarrow \qquad \mathcal{R}(A) = \left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ \hdashline 5 & 6 & 7 & 8 \\ \hdashline 9 & 10 & 11 & 12 \\ \hdashline 13 & 14 & 15 & 16 \\ \hdashline 17 & 18 & 19 & 20 \\ \hdashline 21 & 22 & 23 & 24 \end{array}\right]$$

Fig. 2.2 Matrix reshuffling of a 4×6-matrix sub-structured into 2×2-blocks.

r determines the maximal Kronecker rank of $\sum_{\nu=1}^{r} B_\nu \otimes C_\nu$.

Proof. See (Van Loan and N. Pitsianis, 1993). □

Lemma 2.1.18 enables us to seek the best approximation of $\mathcal{R}(A)$ subject to the Frobenius norm for the fixed (matrix) rank r. The best approximation is ensured using a singular value decomposition and the Eckart-Young theorem. The singular vectors and singular values after reshuffling solve the nearest Kronecker product problem:

Corollary 2.1.19 (Kronecker product singular value decomposition)**.** *Let $A \in \mathbb{R}^{M \times N}$ with factorizations $M = m_1 m_2$ as well as $N = n_1 n_2$ and $R = \min\{m_1 n_1, m_2 n_2\}$. Let the triplet of matrices $U \in \mathbb{R}^{(m_1 n_1) \times R}$, $V \in \mathbb{R}^{(m_2 n_2) \times R}$ and $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_R)$ denote the singular value decomposition of matrix reshuffling $\mathcal{R}(A)$, that is*

$$U^{\mathsf{T}} \mathcal{R}(A) V = \Sigma. \tag{2.32}$$

Then, the matrices B_ν and C_ν defined entrywise for all $\nu = 1, \ldots, r$ by

$$\begin{aligned} vect(B_\nu^{\mathsf{T}})_{\boldsymbol{i}} &= \sqrt{\sigma_\nu} U_{\boldsymbol{i}\nu}, \quad \boldsymbol{i} = 1, \ldots, m_1 n_1, \\ vect(C_\nu^{\mathsf{T}})_{\boldsymbol{i}} &= \sqrt{\sigma_\nu} V_{\boldsymbol{i}\nu}, \quad \boldsymbol{i} = 1, \ldots, m_2 n_2, \end{aligned} \tag{2.33}$$

minimize

$$\left\| A - \sum_{\nu=1}^{r} B_\nu \otimes C_\nu \right\|_F$$

for a fixed tensor rank $r \leq R$.

Proof. Use Lemma 2.1.18 and the best-approximation property of the singular value decomposition. □

We refer to the computation of B_ν and C_ν given by (2.33) as *Kronecker product singular value decomposition* (KSVD). To allow for large matrices (arising from high-order finite element discretizations) and exploiting tensor structure, we decide to use Golub-Kahan bidiagonalization (G. Golub and Kahan, 1965) (which relates to Lanczos tridiagonalization)

to efficiently compute good approximations of the largest r singular values, see Algorithm 1. The key motivation for Algorithm 1 is that the algorithm involves the matrix A only through matrix-vector products with its reshuffling $\mathcal{R}(A)$, and does not need access to single matrix entries. The orthogonalization in lines 6 and 10 of Algorithm 1 accelerates the convergence of $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_r$ towards the largest singular values of $\mathcal{R}(A)$ (G. H. Golub and Van Loan, 2013).

For our purposes, i.e., computing KSVDs for Kronecker ranks not larger than two or three, extra costs from orthogonalizing through a modified Gram-Schmidt process paid off. Throughout this work, we choose a threshold ϵ close to the given floating-point accuracy and $N_{Lanc} = r_A + 1$, which means N_{Lanc} is larger by one than the tensor rank of the input matrix A. Thus, the r largest singular values and singular vectors are computed up to the accuracy given by ϵ. A smaller number N_{Lanc} might be used to compute the rank-r KSVD with reduced computational complexity, but at least as large as r. However, the trade-off is less accurate approximations of the leading singular values and singular vectors. For more insights and subtleties to tweak Algorithm 1, we recommend (G. H. Golub, Luk, et al., 1981; G. H. Golub and Van Loan, 2013; G. Golub and Kahan, 1965; Simon and Zha, 2000; Van Loan and N. Pitsianis, 1993) to the interested reader.

Algorithm 1 The Kronecker product SVD computes $A \approx \sum_{\nu=1}^{r} B_\nu \otimes C_\nu$.

1: **procedure** $\mathrm{KSVD}_r(A)$
2: Choose initial p_1 such that $\beta_1 \leftarrow \|p_1\|_2 = 1$ and $u_1 \leftarrow 0$
3: **for** $k = 1$ to N_{Lanc} **do**
4: $v_k \leftarrow \beta_k^{-1} p_k$
5: $r_k \leftarrow \mathcal{R}(A) v_k - \beta_k u_k$
6: Orthogonalize $r_1, \ldots, r_k$
7: $\alpha_k \leftarrow \|r_k\|_2$
8: $u_{k+1} \leftarrow \alpha_k^{-1} r_k$
9: $p_{k+1} \leftarrow \mathcal{R}(A)^{\mathsf{T}} u_{k+1} - \alpha_k v_k$
10: Orthogonalize $p_1, \ldots, p_{k+1}$
11: $\beta_{k+1} \leftarrow \|p_{k+1}\|_2$
12: **if** $\beta_{k+1} < \epsilon$ **then**
13: **break**
14: **end if**
15: **end for** ▷ stopped after n iterations
16: $\widehat{U} \leftarrow [u_2 | \ldots | u_{n+1}]$
17: $\widehat{V} \leftarrow [v_1 | \ldots | v_n]$
18: Set bidiagonal matrix D with diagonal $(\alpha_1, \ldots, \alpha_n)$ and super-diagonal $(\beta_2, \ldots, \beta_n)$
19: Compute rank-r SVD $\tilde{U}^{\mathsf{T}} D \tilde{V} = \tilde{\Sigma}_r$ with singular values $\tilde{\Sigma}_r = \mathrm{diag}\,(\tilde{\sigma}_1, \ldots, \tilde{\sigma}_r)$
20: $U \leftarrow \widehat{U}\tilde{U}$
21: $V \leftarrow \widehat{V}\tilde{V}$
22: Set B_ν and C_ν as detailed in (2.33) for $\nu = 1, \ldots, r$
23: **return** $\{B_\nu, C_\nu\}_{\nu=1,\ldots,r}$
24: **end procedure**

Assuming the input matrix A of Algorithm 1 has already a low-rank tensor representation with rank r_A, the matrix-vector product with $\mathcal{R}(A)$ and $\mathcal{R}(A)^\mathsf{T}$ is computed at the cost of an inner product of vectors, respectively. For simplicity, consider a rank-1 matrix

$$A = A^{(1)} \otimes A^{(2)}$$

with $A^{(1)}, A^{(2)} \in \mathbb{R}^{n\times n}$. The respective matrix reshuffling reads

$$\mathcal{R}(A^{(1)} \otimes A^{(2)}) = \operatorname{vect}(A^{(1)}) \operatorname{vect}(A^{(2)})^\mathsf{T}. \tag{2.34}$$

Note that the right-hand side in (2.34) is the outer product between two (column) vectors, $\operatorname{vect}(A^{(1)})$ and $\operatorname{vect}(A^{(2)})$. Then, the computational costs of the matrix-vector product

$$\mathcal{R}(A^{(1)} \otimes A^{(2)})v = \left(\operatorname{vect}(A^{(2)})^\mathsf{T} \cdot v\right) \operatorname{vect}(A^{(1)}) \tag{2.35}$$

are dominated by the inner product, requiring only $\mathcal{O}(n^2)$ arithmetic operations in total. Given a rank-r_A matrix, the operations accumulate to $\mathcal{O}(r_A n^2)$. The computational effort of a modified Gram-Schmidt process (used for orthogonalization in lines 6 and 10) is also dominated by computing inner products; thus, we conclude:

Remark 2.1.20 (Reduced computational effort). Given an input matrix A with low-rank tensor representation

$$A = \sum_{\nu=1}^{r_A} A_\nu^{(1)} \otimes A_\nu^{(2)},$$

where $A_\nu^{(1)}$ and $A_\nu^{(2)} \in \mathbb{R}^{n\times n}$, then, computing the rank-r KSVD through Algorithm 1 up to floating-point accuracy requires only $\mathcal{O}(r_A r n^2)$ arithmetic operations. Therefore, the computational effort is linear in the number of unknowns, which is $2rn^2$.

For low ranks r, storing the rank-1 tensors of the rank-r KSVD requires much less memory than storing the vectorization of Kronecker products in the sense of (2.15). We emphasize that a single matrix entry is not accessible in this format. However, there is no need for this access: in this work, we will either use the elementary tensors directly for fast diagonalization or apply them to a vector. The latter is efficiently computed via sum factorization, see (2.20).

Remark 2.1.21 (Memory intensity). Storing the rank-r KSVD requires only $2rn^2$ values arising from the r elementary tensors, each consisting of two $n \times n$ matrices.

2.2 Matrix-free Operator Evaluation

Assuming a discretization matrix A, "matrix-free" simply refers to repeatedly computing the matrix-vector multiplication Av with interchanging vectors v without ever storing matrix entries for subsequent multiplications in (main) memory. In other words, the matrix entries

are re-computed on-the-fly. Conceptually, the memory intensity is significantly reduced but at the price of repetitive arithmetic operations. The great success of and ever-growing interest in matrix-free methods comes ultimately from leveraging a technique called sum-factorization that constitutes the method's heart by successively applying small parts of the discretization matrix to the vector efficiently. In the context of finite elements, these "parts" are defined by cells or facets.

In the context of integration-based evaluation, sum-factorization was first introduced in the spectral element community, originating from the work in (Orszag, 1980). In the 90s and 00s first matrix-free methods were systematically improved and implemented by the spectral element community. P. F. Fischer (1990) and P. F. Fischer and Patera (1991) developed methods for high-order spectral element discretizations of unsteady incompressible Navier-Stokes and Stokes equations in general three-dimensional domains, respectively, and implemented them on medium-grained and distributed-memory parallel computers. The work was continued later by tera-scale spectral element algorithms and implementations (Tufo and P. F. Fischer, 1999), efficient overlapping Schwarz methods (P. F. Fischer, Tufo, et al., 2000) and hybrid multigrid/Schwarz algorithms (Lottes and P. F. Fischer, 2005) on complex three-dimensional geometries solving incompressible flow problems. In the past 10 to 15 years their pioneering work has drawn attention to other researching communities, in particular, to those studying high-order finite element methods.

First, Brown (2010), Cantwell, S. J. Sherwin, et al. (2011), and Vos et al. (2010) discussed implementation strategies for non-linear solvers exploiting tensor product shape functions in the context of hexahedral spectral/hp elements. Then, Kronbichler and Kormann (2012) developed and discussed matrix-free methods in the context of continuous finite elements. Later, these methods were extended, generalized or adapted to high-order discontinuous Galerkin discretizations in (Kempf et al., 2020; Kronbichler and Allalen, 2018; Kronbichler and Kormann, 2019; Kronbichler, Kormann, Pasichnyk, et al., 2017; Müthing et al., 2017). We emphasize the recent activities of SPPEXA projects TerraNeo (Bauer et al., 2020), EXA-DUNE (Bastian, Altenbernd, et al., 2020), and ExaDG (Arndt, Fehn, et al., 2020), and the international collaboration (P. Fischer et al., 2020) striving towards Exascale computing and continuing the work of (Gmeiner et al., 2015; May et al., 2014; Rudi et al., 2015): matrix-free methods have proven to be the state-of-the-art implementation for numerical PDE solvers in the high-performance computing context, in particular, when large-scale, high-performance simulations are computed on the world's fastest supercomputers.

The broad interest comes also from the fact that modern hardware consists of multi-core architectures with steadily improving SIMD capabilities, increasing the computational performance at a higher pace than the memory bandwith. Consequently, many researchers actively developed *matrix-free methods* tailored to modern computer hardware, available in well-known academic software:

- `deal.II` (Arndt, Bangerth, Blais, et al., 2020; Arndt, Bangerth, Davydov, et al., 2021)
- DUNE (Bastian, Blatt, et al., 2021) and (Bastian, Altenbernd, et al., 2020; Bastian, Engwer, et al., 2016)
- MFEM (Anderson et al., 2021)
- NGSolve (Schöberl, 2014)
- Firedrake (McRae et al., 2016; Rathgeber et al., 2016)
- Nek5000 (P. F. Fischer, Lottes, and Kerkemeier, 2008)
- Nektar++ (Cantwell, Moxey, et al., 2015)

Throughout this work, we utilize the matrix-free framework of the general-purpose object-oriented finite element library `deal.II` for numerical implementations. The break-even point at which matrix-free implementations of finite element operators are not only competitive but superior to classical sparse matrix codes is already low on modern hardware: Kronbichler and Kormann (2012), Kronbichler and Kormann (2019), and May et al. (2014) proved that matrix-free methods are more efficient than matrix-based counterparts from finite element order three onwards, with a quickly growing gap for high-order elements. `deal.II`'s matrix-free framework features SIMD vectorizations over elements, optimized sum factorization kernels (computing (2.20) with highest performance), various looping strategies for integration (over cells and facets independently, or in a single sweep over elements involving face integrals), efficient tensor structured mappings for generic and complex geometries, and tailored finite elements like the Hermite-like variant in (Kronbichler, Kormann, Fehn, et al., 2019) optimizing data access patterns. For more details and recent advances of `deal.II`'s matrix-free framework, we refer to (Arndt, Bangerth, Davydov, et al., 2021; Arndt, Fehn, et al., 2020; Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019). In addition, we recommend other excellent works that demonstrate alternative concepts. For instance, promising SIMD strategies in (Müthing et al., 2017) with the issue of code sustainability and performance portability being mitigated by code generation in (Kempf et al., 2020).

In this section, we first introduce standard finite elements and quadrature rules with focus on their tensor structure. Afterward, we exemplify on basis of discretizing the Laplacian in which way tensor structure enables sum factorization. We explain how efficient operator application leads to significantly reduced computational costs and memory intensity. In particular, assuming isotropic polynomials with order n in D spatial coordinates, we highlight that matrix-free operator evaluation reduces the cost of naïve operator evaluation from $\mathcal{O}(n^D)$ arithmetic operations per unknown to $\mathcal{O}(Dn)$, see also (Arndt, Fehn, et al., 2020; Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019).

2.2.1 Tensor Product Finite Elements

We start with the definition of tensor product finite elements, defining the tensor product of polynomials first.

Definition 2.2.1 (Tensor product of polynomials)**.** Let $\mathcal{P}_k$ be the space of univariate polynomials of degree up to k and $k_1, \dots, k_D < \infty$. The tensor product of univariate polynomials is defined by

$$\hat{\phi}^{(1)} \otimes \hat{\phi}^{(2)} \cdots \otimes \hat{\phi}^{(D)}(\boldsymbol{x}) := \prod_{d=1}^{D} \hat{\phi}^{(d)}(x_d), \tag{2.36}$$

for $\hat{\phi}^{(d)} \in \mathcal{P}_{k_d}, d = 1, \dots, D$. The tensor product space of polynomials is defined as

$$\mathbb{Q}_{k_1,k_2,\dots,k_D} := \bigotimes_{d=1}^{D} \mathcal{P}_{k_d} := \operatorname{span}\left\{ \bigotimes_{d=1}^{D} \hat{\phi}^{(d)} \mid \hat{\phi}^{(d)} \in \mathcal{P}_{k_d},\ d = 1, \dots, D \right\}. \tag{2.37}$$

If $k = k_1 = \cdots = k_D$, tensors are called isotropic and the isotropic tensor product space is denoted

$$\mathbb{Q}_k := \mathbb{Q}_{k,k,\dots,k}. \tag{2.38}$$

Polynomials in $\mathbb{Q}_{k_1,\dots,k_D}$ are multivariate, meaning they have as many variables as tensor dimensions, here D. The elementary tensors of the form (2.36) are a product of univariate functions, and thus this tensor structure is often referred to as *separation of variables*. Note that $\mathbb{Q}_{k_1,\dots,k_D}$ is a Hilbert space with the inner product being induced from the multiplication of inner products for each $\mathcal{P}_{k_d}$. For more details on topological tensor products, in particular tensor products of Hilbert spaces, we recommend (Hackbusch, 2012, §4). For instance, tensor products for other classes of functions than polynomials are analogously defined, but one has to be careful with the definition of the tensor space (2.38): in the infinite-dimensional case, topological tensor spaces require closure with respect to some norm.

Lagrange element. We follow P. Ciarlet (1978) and introduce the *Lagrange element* as a triplet of a domain (referred to as cell), shape function space, and set of node functionals. In particular, using quadrilateral or hexahedral cells, we utilize their natural tensor structures. In this section, we write $\mathcal{P}_k([0,1])$ or $\mathbb{Q}_k([0,1]^D)$ to explicitly refer to univariate polynomials on the unit interval or tensor product polynomials on the unit hypercube, respectively. In subsequent sections, we will omit to specify the domain in favor of a concise notation.

Definition 2.2.2. The *Lagrange element* of degree k on the unit interval is the triplet $([0,1], \mathcal{P}_k([0,1]), \Sigma^{1D})$. The shape functions $\hat{\phi}_i \in \mathcal{P}_k([0,1])$ are defined as dual basis of $\hat{\mathcal{N}}_j^{1D} \in \Sigma^{1D}$, that is

$$\hat{\mathcal{N}}_j^{1D}(\hat{\phi}_i) = \delta_{ij}$$

for all $i, j = 1, \ldots, n_{dof}$. The node functionals are uniquely determined by the set of Gauss-Lobatto support points $\{\hat{y}_j\}$ in $[0, 1]$,

$$\hat{\mathcal{N}}_j^{1D}(\hat{\phi}) = \hat{\phi}(\hat{y}_j). \tag{2.39}$$

The number of degrees of freedom n_{dof} equals $(k + 1)$.

We emphasize that any choice of support points defining linearly independent node functionals is possible. For reasons of numerical stability, we choose Gauss-Lobatto support points which are superior to equidistant nodal points (G. Karniadakis and S. Sherwin, 2005). Using lexicographic ordering, we abbreviate multi-index notation $i_1, \ldots, i_D$ by $\boldsymbol{i}$. Therefore, we utilize the tensor product notation from previous sections to linear forms

$$\mathcal{N}_{\boldsymbol{i}} := \mathcal{N}_{i_1}^{(1)} \otimes \mathcal{N}_{i_2}^{(2)} \otimes \ldots \otimes \mathcal{N}_{i_D}^{(D)} := \prod_{d=1}^{D} \hat{\mathcal{N}}_{i_d}^{(d)}. \tag{2.40}$$

Here, (2.40) suffices to introduce tensor products of linear forms, which are elaborated in (Hackbusch, 2012). The Lagrange element on the unit hypercube is defined as isotropic tensor product.

Definition 2.2.3 (Lagrange element)**.** The *Lagrange element* of degree k on the unit hypercube $\hat{K} = [0, 1]^D$ is the triplet $(\hat{K}, \mathbb{Q}_k(\hat{K}), \Sigma)$. The node functionals $\hat{\mathcal{N}}_{\boldsymbol{j}} \in \Sigma$ are defined as tensor product of univariate node functionals from (2.39),

$$\hat{\mathcal{N}}_{\boldsymbol{j}}(\hat{\phi}^{(1)} \otimes \ldots \otimes \hat{\phi}^{(D)}) = \prod_{d=1}^{D} \hat{\mathcal{N}}_{j_d}^{1D}(\hat{\phi}^{(d)}). \tag{2.41}$$

The shape functions $\hat{\varphi}_{\boldsymbol{i}} \in \mathbb{Q}_k(\hat{K})$ are defined as dual basis of $\hat{\mathcal{N}}_{\boldsymbol{j}}$, that is

$$\hat{\mathcal{N}}_{\boldsymbol{j}}(\hat{\varphi}_{\boldsymbol{i}}) = \delta_{\boldsymbol{ij}}$$

for all $\boldsymbol{i}, \boldsymbol{j} = 1, \ldots, N_{dof}$. The number of degrees of freedom N_{dof} is $(k + 1)^D$.

We emphasize that basis node functionals $\hat{\mathcal{N}}_{\boldsymbol{j}}$ are only defined for rank-1 tensors in (2.41). However, due to linearity and span property (2.37) they are well-defined for any tensor product polynomial in $\mathbb{Q}_k$. A favorable characteristic is that a tensor product of basis elements is a basis element for its tensor product space. The Cartesian product of Gauss-Lobatto points on the unit interval defines a Gauss-Lobatto point on the unit cell $\hat{K}$,

$$\hat{\boldsymbol{y}}_{\boldsymbol{i}} = (\hat{y}_{i_1}, \ldots, \hat{y}_{i_D}).$$

Consequently, the basis of shape functions determined by Definition 2.2.3 reads

$$\hat{\varphi}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}) = \prod_{d=1}^{D} \hat{\phi}_{i_d}^{(d)}(x_d) \tag{2.42}$$

for all $1 \leq i_d \leq k+1, d = 1, \ldots, D$, and is as such a nodal basis in Gauss-Lobatto points on $\hat{K}$ by duality.

The shape function spaces $V(K)$ on actual mesh cells K are obtained by composition with the mapping $\boldsymbol{F}_K \colon \hat{K} \to K$ such that the basis of shape functions reads

$$\varphi_{K,\boldsymbol{i}}(\boldsymbol{x}) = \hat{\varphi}_{\boldsymbol{i}} \circ \boldsymbol{F}_K^{-1}(\boldsymbol{x}), \quad \boldsymbol{x} \in K, \tag{2.43}$$

for all $\boldsymbol{i} = 1, \ldots, N_{dof}$.

Similarly, we define elements for the same polynomial space but with moment-based degrees of freedom.

Definition 2.2.4. The *Legendre element* of degree k on the unit interval is the triplet $([0,1], \mathcal{P}_k([0,1]), \Sigma_{Leg}^{1D})$. The shape functions $\hat{\phi}_i \in \mathcal{P}_k([0,1])$ are defined as dual basis of $\hat{\mathcal{N}}_j^{1D} \in \Sigma_{Leg}^{1D}$, that is

$$\hat{\mathcal{N}}_j^{1D}(\hat{\phi}_i) = \delta_{ij}$$

for all $i, j = 1, \ldots, n_{dof}$. The node functionals are uniquely determined by L^2-orthogonality,

$$\hat{\mathcal{N}}_j^{1D}(\hat{\phi}) = \int_0^1 \hat{\phi}\hat{\phi}_j \, d\hat{x}, \tag{2.44}$$

involving a normalization condition for $i = j$. The number of degrees of freedom n_{dof} equals $(k+1)$.

We refer to this finite element as *Legendre element* since the shape function basis

$$\left\{\hat{\phi}_1, \ldots, \hat{\phi}_{n_{dof}}\right\}$$

consists of the well-known Legendre polynomials, which are usually defined on $[-1,1]$ and with different normalization condition. In literature, we frequently find the normalization condition

$$\hat{\phi}_i(1) = 1 \tag{2.45}$$

instead of using

$$\hat{\mathcal{N}}_i^{1D}(\hat{\phi}_i) = \int_0^1 \hat{\phi}_i^2 \, d\hat{x}$$

as part of (2.44). Both conditions have their own merits.

The definition is a construction principle that reflects the hierarchical character of Legendre elements: starting with degree zero, $\hat{\phi}_1 \equiv 1$ is uniquely determined by normalization. Then,

from orthogonality to $\hat{\phi}_1$ and normalization $\hat{\phi}_2(x) = x$ is obtained, and so on for higher degrees. We still refer to $\hat{\mathcal{N}}_j^{1D}$ as node functionals, although strictly speaking, these functionals are moments. The Legendre element on the unit hypercube is simply the isotropic tensor product of univariate Legendre elements.

Definition 2.2.5 (Legendre element)**.** The *Legendre element* of degree k on the unit hypercube $\hat{K} = [0,1]^D$ is the triplet $(\hat{K}, \mathbb{Q}_k(\hat{K}), \Sigma_{Leg})$. The node functionals $\hat{\mathcal{N}}_{\boldsymbol{j}} \in \Sigma_{Leg}$ are defined as tensor product of univariate node functionals from (2.44),

$$\hat{\mathcal{N}}_{\boldsymbol{j}}(\hat{\phi}^{(1)} \otimes \ldots \otimes \hat{\phi}^{(D)}) = \prod_{d=1}^{D} \hat{\mathcal{N}}_{j_d}^{1D}(\hat{\phi}^{(d)}). \tag{2.46}$$

The shape functions $\hat{\varphi}_{\boldsymbol{i}} \in \mathbb{Q}_k(\hat{K})$ are defined as dual basis of $\hat{\mathcal{N}}_{\boldsymbol{j}}$, i.e.,

$$\hat{\mathcal{N}}_{\boldsymbol{j}}(\hat{\varphi}_{\boldsymbol{i}}) = \delta_{\boldsymbol{ij}}$$

for all $\boldsymbol{i}, \boldsymbol{j} = 1, \ldots, N_{dof}$. The number of degrees of freedom N_{dof} is $(k+1)^D$.

The conformity of finite elements is determined by its degree of inter-element continuity. To this end, it is crucial that degrees of freedom are closely related to geometrical entities, like vertices, edges, faces, to impose some sort of continuity between elements, since they are smooth in the interior of each element. For instance, Lagrange elements have degrees of freedom associated with vertices, edges, and faces. Thus, it is possible to impose global C^0 regularity. Whereas the moment-based functionals (2.46) are solely associated with the cell which makes Legendre elements piecewise discontinuous approximations by nature. This topic is, among others, comprehensively studied in (Arnold, Boffi, and Bonizzoni, 2014; Arnold, Falk, and Winther, 2006, 2010) given the periodic table of finite elements[2].

Quadrature. A quadrature formula on the unit cell $\hat{K}$ is defined as D-fold tensor product of one-dimensional quadrature formulas on the unit interval. Let the set of pairs $\{(\hat{x}_{q_d}, w_{q_d})\}_{q=1,\ldots,n_{quad}}$ characterize a one-dimensional quadrature rule. Then, abscissas and weights of a D-dimensional quadrature formula are defined by

$$\hat{\boldsymbol{x}}_{\boldsymbol{q}} = (\hat{x}_{q_1}, \ldots, \hat{x}_{q_D}) \quad \text{and} \quad w_{\boldsymbol{q}} = \prod_{d=1}^{D} w_{q_d}, \tag{2.47}$$

for all $1 \leq q_d \leq n_{quad}, d = 1, \ldots, D$. Choosing the same one-dimensional rule for each spatial dimension implies isotropy such that $\boldsymbol{q} \in \{1, \ldots, N_{quad}\}$, where N_{quad} equals n_{quad}^D, denoting the number of D-dimensional quadrature pairs.

Let V_h denote a generic finite element space associated with mesh $\mathcal{T}_h$ of characteristic size h such that $u_h|_K \in V(K)$ for any $u_h \in V_h$ and $K \in \mathcal{T}_h$; "generic" in the sense that we do not

[2]See https://www-users.cse.umn.edu/~arnold/femtable/background.html.

specify inter-element constraints. For instance, in case of discontinuous Galerkin methods for elliptic problems

$$V_h = \left\{ v \in L^2(\Omega) \mid v|_K \in V(K) \quad \forall K \in \mathcal{T}_h \right\} = \bigoplus_{K \in \mathcal{T}_h} V(K),$$

which means no inter-element continuity is imposed (Arnold, Brezzi, et al., 2002). The global basis

$$\{\varphi_i\}_{i=1,\dots,N_h}$$

defines by duality the coefficient space $\mathbb{R}^{N_h}$ of the same dimension as V_h equipped with the Euclidean inner product. In computations, this is the inner product used to compute norms, such that we will identify V_h with the coefficient space and do not distinguish them in notation: to be precise any finite element function $u_h \in V_h$ reads

$$u_h(\boldsymbol{x}) = \sum_{i=1}^{N_h} u_{h;i} \varphi_i(\boldsymbol{x}),$$

where u_h on the right-hand side is a coefficient vector in $\mathbb{R}^{N_h}$. N_h denotes the number of global degrees of freedom on the mesh $\mathcal{T}_h$.

2.2.2 Efficient Operator Application of the Laplacian

To demonstrate "matrix-free operator evaluation" or "efficient operator application", respectively, we assume applying a vector of coefficients u_h to an abstract finite element matrix A_h. Due to the domain decomposition given by a mesh $\mathcal{T}_h$, the matrix-vector product reads

$$A_h u_h = \sum_{K \in \mathcal{T}_h} \Pi_{h;K}^{\mathsf{T}} A_{h;K} \Pi_{h;K} u_h = \sum_{K \in \mathcal{T}_h} \Pi_{h;K}^{\mathsf{T}} A_{h;K} u_K, \tag{2.48}$$

where $A_{h;K}$ represents the local contribution of A_h for each cell K. The transfer from global degrees of freedom to local degrees of freedom is denoted by $\Pi_{h;K}$, i.e., the restriction operator to the local coefficient space $\mathbb{R}^{N_{dof}}$. The transpose of the restriction prolongates local coefficient vectors into the global coefficient space $\mathbb{R}^{N_h}$. From now on, we suppress the mesh index h if it occurs along with cell index K. For example, we write in short A_K and Π_K instead of $A_{h;K}$ and $\Pi_{h;K}$, respectively.

In classical finite element codes, the discretization matrix A_h is assembled from local matrices A_K and stored in a sparse matrix format. However, on modern computing architectures, in particular large-scale computer clusters, transferring data in main memory has become the major limiting factor. In addition, most iterative methods utilize (discretization) matrices only through matrix-vector products, not needing access to specific matrix entries. Matrix-free methods leverage this fact, efficiently computing the numerical integration on-

the-fly in each cell K, i.e., computing $A_K u_K$ without storing local matrices A_K or even the sparse matrix A_h. To be competitive with or superior to classical sparse matrix formats (in the sense of runtime), exploiting the tensor structure of finite elements and quadrature rules is crucial. In that case, we refer to $A_h u_h$ as *efficient operator application* or *matrix-free operator evaluation* to stress the integration-based application or evaluation. Consequently, A_h is implemented as linear operator rather than as matrix.

The computational efficiency of matrix-free methods was claimed in this section's introduction, but it was not shown how it is realized. The efficiency comes from exploiting tensor structure through *sum factorization*. Let us exemplify the benefits and implementation of sum factorization on the basis of the Laplacian. The respective finite element discretization for each cell K is given by

$$\begin{aligned}(A_K)_{i,j} &= \int_K \boldsymbol{\nabla}\varphi_{K;j}(\boldsymbol{x}) \cdot \boldsymbol{\nabla}\varphi_{K;i}(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} \\ &= \int_{\hat{K}} \left(\hat{J}_K^{-\mathsf{T}}(\hat{\boldsymbol{x}})\hat{\boldsymbol{\nabla}}\hat{\varphi}_i(\hat{\boldsymbol{x}})\right) \cdot \left(\hat{J}_K^{-\mathsf{T}}(\hat{\boldsymbol{x}})\hat{\boldsymbol{\nabla}}\hat{\varphi}_j(\hat{\boldsymbol{x}})\right) \det\left(\hat{J}_K(\hat{\boldsymbol{x}})\right) d\hat{\boldsymbol{x}}\end{aligned} \tag{2.49}$$

for $i, j = 1, \ldots, N_{dof}$. A simple change of variables from real to unit coordinates results in the identity (2.49): the gradient of shape functions in real space expressed as gradient of unit shape functions is the result of the multi-dimensional chain rule and (2.43), involving the Jacobian, which is defined by

$$(\hat{J}_K)_{ij} = \hat{\partial}_j \boldsymbol{F}_{K;i}.$$

Using the Gauss quadrature from (2.47) for integration, a cell-local operator application reads

$$(A_K u_K)_{\boldsymbol{i}} = \sum_{\boldsymbol{q}=1}^{N_{quad}} \det\left(\hat{J}_K(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\right) w_{\boldsymbol{q}} \left(\hat{J}_K^{-\mathsf{T}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\hat{\boldsymbol{\nabla}}\hat{\varphi}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\right) \cdot \left[\hat{J}_K^{-\mathsf{T}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}}) \sum_{\boldsymbol{j}=1}^{N_{dof}} \hat{\boldsymbol{\nabla}}\hat{\varphi}_{\boldsymbol{j}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}}) u_{K;\boldsymbol{j}}\right] \tag{2.50}$$

for $\boldsymbol{i} = 1, \ldots, N_{dof}$. Assuming tensor product finite elements, the number of quadrature points N_{quad} and number of degrees of freedom N_{dof} is proportional to $(n_{dof})^D$, respectively, such that a naïve evaluation of (2.50) costs $\mathcal{O}((n_{dof})^{2D})$ arithmetic operations.

The computational complexity is reduced by factorizing both sums in (2.50), utilizing tensor structure of finite elements and of quadrature rules, respectivley. Using isotropic tensor product shape functions, the first vector component of $\hat{\boldsymbol{\nabla}}\hat{\varphi}_{\boldsymbol{j}}$ decomposes into

$$\partial_1 \hat{\varphi}_{\boldsymbol{j}}(\hat{\boldsymbol{x}}) = \hat{\phi}'_{j_1}(x_1)\hat{\phi}_{j_2}(x_2)\ldots\hat{\phi}_{j_D}(x_D). \tag{2.51}$$

We introduce matrices Φ and Φ' carrying the univariate shape functions and gradients, respectively, each evaluated in each quadrature point,

$$\Phi_{i,q} = \phi_i(x_q) \quad \text{and} \quad \Phi'_{i,q} = \phi'_i(x_q) \tag{2.52}$$

for $i = 1, \dots, n_{dof}$ and $q = 1, \dots, n_{quad}$. Let $\mathfrak{u}$ be the tensorization of u_K, the interpolation of $\partial_1 u_h$ on the reference cell $\hat{K}$ can be written as D consecutive mode products,

$$\begin{aligned}\sum_{\boldsymbol{j}=1}^{N_{dof}} \partial_1 \hat{\varphi}_{\boldsymbol{j}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}}) u_{K;\boldsymbol{j}} &= \sum_{i_1=1}^{n_{dof}} \hat{\phi}'_{i_1}(\hat{x}_{q_1}) \sum_{i_2=1}^{n_{dof}} \hat{\phi}_{i_2}(\hat{x}_{q_2}) \cdots \sum_{i_D=1}^{n_{dof}} \hat{\phi}_{i_D}(\hat{x}_{q_D}) \mathfrak{u}_{i_1,\dots,i_D} \\ &= \Big(\Phi' \times_1 \Phi \times_2 \dots \Phi \times_D \mathfrak{u}\Big)_{q_1,\dots,q_D} \\ &=: \hat{\mathfrak{u}}^{\partial_1}_{q_1,\dots,q_D}\end{aligned} \tag{2.53}$$

It is computed at the cost of a single sum factorization, see Section 2.1.2 for details. We have similar sum factorizations for remaining components $\partial_2 u_h, \dots, \partial_D u_h$. The $\mathbb{R}^D$-valued interpolation tensor of $\boldsymbol{\nabla} u_h$ on element K reads

$$\mathfrak{u}^{\boldsymbol{\nabla}}_{q_1,\dots,q_D} := \begin{pmatrix} \mathfrak{u}^{\partial_1}_{q_1,\dots,q_D} \\ \vdots \\ \mathfrak{u}^{\partial_D}_{q_1,\dots,q_D} \end{pmatrix} \tag{2.54a}$$

with

$$\mathfrak{u}^{\partial_d}_{q_1,\dots,q_D} = \sum_{r=1}^{D} \Big(\hat{J}_K^{-\mathsf{T}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\Big)_{d,r} \hat{\mathfrak{u}}^{\partial_r}_{q_1,\dots,q_D} \tag{2.54b}$$

for all $1 \le q_d \le n_{quad}$. In other words, $\mathfrak{u}^{\boldsymbol{\nabla}}$ is a D-dimensional array where each element $\mathfrak{u}^{\partial_d}_{q_1,\dots,q_D}$ is a real-valued (column) vector in $\mathbb{R}^D$. Note that $\mathfrak{u}^{\boldsymbol{\nabla}}$ represents the term in square brackets in (2.50).

The right-hand side of (2.50) without this term describes testing against the gradients of shape functions $\varphi_{K;1}, \dots, \varphi_{K;N_{dof}}$. The sum over quadrature indices is again factorizable, but a change of variables from real to unit space needs to applied first. To this end, we define the intermediate $\mathbb{R}^D$-valued tensor

$$\hat{\mathfrak{v}}_{q_1,\dots,q_D} := \Big(\hat{\mathfrak{v}}^1_{q_1,\dots,q_D}, \dots, \hat{\mathfrak{v}}^D_{q_1,\dots,q_D}\Big) \tag{2.55a}$$

by

$$\hat{\mathfrak{v}}^d_{q_1,\dots,q_D} = \det\Big(\hat{J}_K(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\Big) w_{\boldsymbol{q}} \sum_{r=1}^{D} \mathfrak{u}^{\partial_r}_{q_1,\dots,q_D} \Big(\hat{J}_K^{-\mathsf{T}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\Big)_{r,d} \tag{2.55b}$$

for all $1 \le q_d \le n_{quad}$. $w_{\boldsymbol{q}}$ refers to the quadrature weight introduced in (2.47). The number of univariate quadrature points n_{quad} is (almost) identical to n_{dof}. Finally, the local operator application from (2.50) reads

$$(A_K u_K)_{\boldsymbol{i}} = \sum_{d=1}^{D} \mathfrak{v}^{\partial_d}_{i_1,\dots,i_D} := \sum_{\boldsymbol{q}=1}^{N_{quad}} \hat{\mathfrak{v}}_{q_1,\dots,q_D} \hat{\boldsymbol{\nabla}} \hat{\varphi}_{\boldsymbol{i}}(\hat{\boldsymbol{x}}_{\boldsymbol{q}}) \tag{2.56a}$$

The right-hand side of (2.56a) is evaluated using a sum factorization analogous to (2.53), but here over quadrature indices instead of degrees of freedom,

$$\mathfrak{v}^{\partial_1} = (\Phi')^\mathsf{T} \times_1 \Phi^\mathsf{T} \times_2 \ldots \Phi^\mathsf{T} \times_D \hat{\mathfrak{v}}^1 \tag{2.56b}$$

Similar factorizations are obtained for $\mathfrak{v}^{\partial_2}, \ldots, \mathfrak{v}^{\partial_D}$. Consequently, the local matrix-vector product $A_K u_K$ is efficiently evaluated through successively computing the tensors in (2.53) to (2.56). In total $\mathcal{O}(D^2(n_{dof})^{D+1})$ arithmetic operations are required: the D sum factorizations in (2.53) and (2.56b) with $\mathcal{O}(D^2(n_{dof})^{D+1})$ operations dominate the change of variables in (2.54) and (2.55) with $\mathcal{O}(D^2(n_{dof})^{D})$ operations, respectively. We emphasize that the successive steps (2.53) to (2.56) for matrix-free operator evaluation $A_K u_K$ are in prinicple the same for other differential operators than the Laplacian. In particular, integrals over facets which are inherent in discontinuous Galerkin discretizations are treated similarly in theory. Moreover, a space-dependent coefficient, for instance, a diffusion coefficient in (2.49), enters the transformation of the interpolation from real to unit space in (2.55) naturally. For most tensor product elements, the number of univariate degrees of freedom n_{dof} is almost identical to a polynomial degree k, which is used in Remark 2.2.6. The section is concluded with a remark on the computational effort.

Remark 2.2.6 (Computational effort). Instead of assembling A_h and applying a vector u_h afterward, i.e., computing $A_h u_h$, the underlying integration and application given u_h are folded into a single operation, following the successive steps (2.48), (2.50) and (2.53) to (2.56). Then, due to making use of sum factorization the arithmetic work per degree of freedom is linear in k instead of k^D. In that case, we refer to $A_h u_h$ as *efficient operator application* or *matrix-free operator evaluation*.

2.3 Multilevel Schwarz Methods

The basic principle of multigrid methods traces back to Fedorenko and Bachwalow. In the late 1970s, Brandt and Hackbusch independently developed multigrid methods that could be applied in practice. The original work on Schwarz methods goes back to Schwarz (1870). The review article (Xu, 1992) draws the link between the multigrid and Schwarz theory. In view of this, we refer to a geometric multigrid method using Schwarz smoothers as *multilevel Schwarz methods*.

2.3.1 Geometric Multigrid Method

A geometric multigrid method is a careful balance of components such as discretizations, smoothers, and transfer operators concerning a common partial differential equation, determining the method's success. If all components are chosen well and implemented correctly, the arithmetic complexity is linear in the number of unknowns. A detailed convergence and

complexity analysis is found in (Bramble, 2019; Hackbusch, 1985; Rannacher, 2017). This monograph relies on V-cycle multigrid preconditioners with Schwarz smoothers. To this end, we introduce their components in the finite element context next, in particular, the components that are common to all problems in subsequent chapters.

Hierarchy. We subdivide the physical domain Ω into meshes $\mathcal{T}_\ell$ for levels $\ell = 1, \ldots, L$, where the finest level L is the actual discretization level on which we want to solve the finite element problem. The intermediate levels $\ell < L$ form the hierarchy for the geometric multigrid method. Each mesh consists of a collection of quadrilateral/hexahedral cells K, which are obtained by a mapping $\boldsymbol{F}_K$ from the reference cell $\hat{K} = [0,1]^D$. The relation of these meshes is defined by induction as follows: starting from a coarse mesh $\mathcal{T}_1$ consisting of few cells at most, we generate a hierarchy of meshes $\mathcal{T}_\ell$ for levels $\ell = 1, \ldots, L$ by recursively splitting each cell in $\mathcal{T}_\ell$ with respect to its midpoint into 2^D children in $\mathcal{T}_{\ell+1}$. These meshes are nested in the fashion that every cell of $\mathcal{T}_\ell$ is equal to the union of its 2^D children in $\mathcal{T}_{l+1}$ as well as conforming in the sense that either any edge/face of the cell is at the domain's boundary or an entire edge/face of an adjacent cell. Simplifying notation, we use the term *facet* to denote an edge in two- and a face in three dimensions. In geometry, a facet is a polytope of exactly one dimension less than the structure itself. Throughout this work, nested spaces with hierarchic ordering are used,

$$V_1 \subset \ldots \subset V_\ell \subset \ldots \subset V_L, \tag{2.57}$$

where V_ℓ denotes a generic finite element space for level ℓ. We note that adaptive mesh refinement is not trivially compatible with the Schwarz smoothers presented later; thus, we limit ourselves to a uniform refinement policy as detailed before. The generic structure of a single V-cycle multigrid step is summarized in Algorithm 2.

Transfers. If not stated otherwise we choose generic transfer operators between the nested spaces: the prolongation $I_\ell^\uparrow \colon V_\ell \to V_{\ell+1}$ is the natural embedding from V_ℓ into $V_{\ell+1}$. The restriction $I_{\ell+1}^\downarrow \colon V_{\ell+1} \to V_\ell$ is defined as the adjoint of the prolongation preserving symmetry when transferring between meshes. To be precise, the adjoint is defined with respect to the Euclidean inner product in the coefficient spaces, thus, in matrix form $I_{\ell+1}^\downarrow$ is the transpose of $I_\ell^\uparrow$.

Smoothers. The discretization operators A_ℓ and, in particular, smoothers S_ℓ^{pre} and S_ℓ^{post} for each level are the focal point of this monograph: these depend on the application at hand, and will be developed and analyzed in subsequent chapters.

Coarse-grid solver. Assuming a coarse mesh $\mathcal{T}_1$ consisting of few cells at most, we simply use a direct method as coarse-grid solver A_1^{-1}. We postpone the explicit definition of the direct method.

Algorithm 2 V-cycle of geometric multigrid method

procedure $\mathrm{MG}_\ell(x_\ell, b_\ell)$
  if $\ell = 1$ **then**
    $x_1 \leftarrow A_1^{-1} b_1$ ▷ coarse-grid solver
  end if
  for $k = 1$ to m_{pre} **do**
    $x_\ell \leftarrow S_\ell^{pre}(x_\ell, b_\ell)$ ▷ pre-smoothing
  end for
  $b_{\ell-1} \leftarrow I_{\ell-1}^{\downarrow}\left(b_\ell - A_\ell x_\ell\right)$ ▷ restriction
  $e_{\ell-1} \leftarrow \mathrm{MG}_{\ell-1}(0, b_{\ell-1})$ ▷ recursion
  $x_\ell \leftarrow x_\ell + I_{\ell-1}^{\uparrow} e_{\ell-1}$ ▷ prolongation
  for $k = 1$ to m_{post} **do**
    $x_\ell \leftarrow S_\ell^{post}(x_\ell, b_\ell)$ ▷ post-smoothing
  end for
  return x_ℓ
end procedure

2.3.2 Schwarz Smoothers

We use the terms *Schwarz smoothers* or *domain decomposition smoothers* in the context of geometric multigrid methods and many very small subdomains, on which we solve the differential equation either exactly or sometimes approximately. Because of Xu's review in (Xu, 1992), which unifies the theory of iterative methods, domain decomposition methods, and multigrid methods, we refer to the multigrid method from Section 2.3.1 using Schwarz smoothers as *multilevel Schwarz methods*. We have a particular interest in solving the many subspace problems cost-efficiently by exploiting tensor structure. In this context, we refer to them as *tensor product Schwarz smoother*. We will develop and discuss different tensor product Schwarz smoothers for various applications. A unified theory for second-order elliptic problems using conforming finite element methods was developed by Dryja and O. B. Widlund (1990) and O. Widlund and Dryja (1990). Two-level additive Schwarz methods, overlapping and nonoverlapping, for discontinuous Galerkin discretizations were analyzed in (Feng and Karakashian, 2001). Feng and Karakashian (2005) extended their theory to fourth-order elliptic problems, namely the biharmonic equation. In addition, a theory for cell-based smoothers for the symmetric interior penalty method was found in (Kanschat, 2008), which has been generalized to singularly perturbed reaction-diffusion problems in (Lucero Lorca and Kanschat, 2018): Lucero and Kanschat have also generalized the convergence analysis from (Dryja and Krzyżanowski, 2016) to quadrilateral and hexahedral meshes. Further examples from literature are $\boldsymbol{H}^{\mathrm{div}}(\Omega)$ and $\boldsymbol{H}^{\mathrm{curl}}(\Omega)$ smoothers, introduced by Arnold, Falk, and Winther (1997) and Arnold, Falk, and Winther (2000). This group of methods has been successfully generalized to Stokes (Kanschat and Mao, 2015) and later Darcy-Stokes-Brinkman (Kanschat, Lazarov, et al., 2017) problems. This collection of literature covers

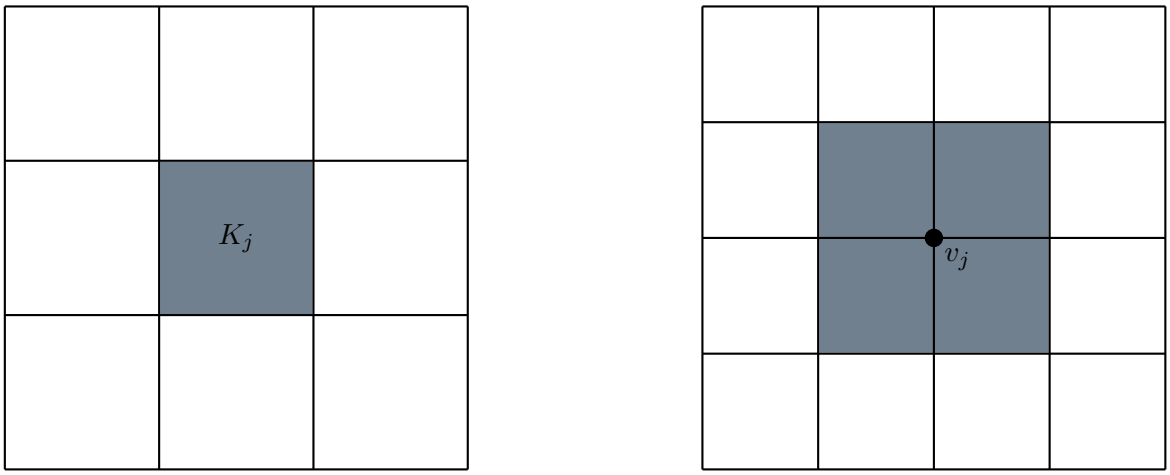


Fig. 2.3 Cell-based subdomain (*left*) and vertex patch (*right*).

Schwarz smoothers that will be discussed throughout this monograph - regarding discontinuous and conforming elements, respectively, for the Poisson problem (Chapter 3), the biharmonic problem (Chapter 4) and the Stokes problem (Chapter 5). The Schwarz smoothers referenced before have in common that local solvers either on cells or on patches of cells around a vertex were used.

1. **cell-based** smoothers: each subdomain of the spatial decomposition on level ℓ consists of a single cell of the mesh $\mathcal{T}_\ell$ as depicted in Figure 2.3. After enumerating the cells in $\mathcal{T}_\ell$ as K_j with $j = 1, \dots, J_\ell$, the subspaces $V_{\ell;j} \subset V_\ell$ consist of functions with support in the cell K_j. Details on subspaces are deferred to sections with a finite element discretization specified. For instance, the spatial decomposition is nonoverlapping for discontinuous finite elements, consequently, V_ℓ is the direct sum of subspaces $V_{\ell;j}$.

2. **vertex patch** smoothers: each subdomain Ω_j is a patch of cells sharing the vertex v_j (after enumeration) as shown in Figure 2.3. We refer to the local mesh $\mathcal{T}_{\ell;j} \subset \mathcal{T}_\ell$ as *vertex patch*. The subspaces $V_{\ell;j} \subset V_\ell$ consist of functions with support in subdomain Ω_j for $j = 1, \dots, J_\ell$, where J_ℓ is the number of interior vertices in $\mathcal{T}_\ell$. The explicit definition of subspaces, including local boundary conditions, is problem-dependent. As typically 2^D cells share a vertex and a cell has 2^D vertices, the spatial decomposition is overlapping and the sum of the subspaces is not direct.

In both cases, we define the local projections $P_{\ell;j} \colon V_\ell \to V_{\ell;j}$ by

$$a_\ell(P_{\ell;j}u_\ell, v) = a_\ell(u_\ell, v) \quad \forall v \in V_{\ell;j}. \tag{2.58}$$

We denote the operator $A_{\ell;j}$ that is associated with the restriction of bilinear form a_ℓ on subspaces $V_{\ell;j} \times V_{\ell;j}$. In that case, we refer to them as exact local solvers. From now on, we

suppress the level index ℓ in expressions if it occurs together with the subdomain subscript, for instance, writing V_j in short for $V_{\ell;j}$. We define the additive Schwarz operator for level ℓ as

$$P_{ad;\ell} := \omega \sum_{j=1}^{J_\ell} P_j = \omega \sum_{j=1}^{J_\ell} R_j^\mathsf{T} A_j^{-1} R_j A_\ell, \tag{2.59}$$

where $\omega \in \mathbb{R}_{>0}$ is a relaxation parameter. $R_j \colon V_\ell \to V_j$ denotes the restriction operator onto subspace V_j and $R_j^\mathsf{T} \colon V_j \to V_\ell$ the prolongation into V_ℓ, the latter being the natural embedding. These transfers are non-trivial in the coefficient space, and as indicated by the notation the restriction R_j is defined as adjoint of R_j^T with respect to the Euclidean inner product. Although both operators are trivial in a functional context, we keep them to emphasize the typical structure of Schwarz methods. The right-hand side of (2.59) highlights the structure as a product of the additive Schwarz preconditioner

$$A_{ad;\ell}^{-1} = \sum_{j=1}^{J_\ell} R_j^\mathsf{T} A_j^{-1} R_j$$

and level matrix A_ℓ. In contrast, the multiplicative Schwarz operator in its standard form consists of a "long" product of single subspace corrections,

$$P_{mu;\ell} := I - \Big(I - \omega P_{J_\ell}\Big) \dots \Big(I - \omega P_2\Big)\Big(I - \omega P_1\Big), \tag{2.60}$$

where I denotes the identity operator.

Throughout this work, the computational performance of mathematical methods plays a significant role. To this end, we have parallel execution by vectorization, multi-threading, and MPI-parallelization on distributed systems in mind. Neither the standard form of the multiplicative Schwarz operator nor the additive operator with overlapping vertex patches is suited for such parallelism. While the additive Schwarz operator on vertex patches suffers from race conditions, both the cell-based and the vertex patch multiplicative operator are inherently sequential. Therefore, we use "coloring" of subdomains in order to recover potential parallelism. Coloring depends on the discretization of local solvers and the domain decomposition such that a particular description is postponed to sections with a finite element method defined. In general, coloring refers to splitting the index set $\mathcal{J} = \{1, \dots, J_\ell\}$ of subdomains into disjoint subsets $\mathcal{J}_c$ with color index $c = 1, \dots, N_{col}$, such that the operations within each subset can be performed in parallel overcoming the mathematical and technical constraints mentioned before.

The latter includes data races that occur due to simultaneously reading and writing: two memory operations conflict if they access the same location in memory and at least one of them is a write operation (Adve et al., 1991). Here, data races might occur when two subdomains share a degree of freedom and both local solvers are executed in parallel. Then,

we might write simultaneously into the same array element during the prolongation of local coefficients.

Remark 2.3.1 (Coloring for additive Schwarz). A coloring scheme avoiding data races is defined as follows: two subspaces V_j and V_i have the same color if

$$R_i R_j^{\mathsf{T}} = 0, \tag{2.61}$$

this means V_j and V_i are orthogonal with respect to the inner product of V_ℓ.

The colored version of the additive Schwarz operator reads

$$P_{ad;\ell} := \omega \sum_{c=1}^{N_{col}} \sum_{j \in \mathcal{J}_c} P_j. \tag{2.62}$$

The colored operator is mathematically equivalent to (2.59) and thus corresponds to the assertion that there is only a technical challenge.

The multiplicative Schwarz operator, in its standard form (2.60), applies subspace corrections sequentially. Therefore, a coloring should not just avoid race conditions but should recover parallelism at all. To this end, we note that

$$\begin{aligned}(I - P_i)(I - P_j) &= (I - R_i^{\mathsf{T}} A_i^{-1} R_i A_\ell)(I - R_j^{\mathsf{T}} A_j^{-1} R_j A_\ell) \\ &= I - P_i - P_j + R_i^{\mathsf{T}} A_i^{-1} (R_i A_\ell R_j^{\mathsf{T}}) A_j^{-1} R_j A_\ell.\end{aligned} \tag{2.63}$$

The parenthesis in the last term evaluates to zero if and only if V_i is A_ℓ-orthogonal to V_j.

Remark 2.3.2 (Coloring for multiplicative algorithms). A coloring scheme for the multiplicative Schwarz method is designed as follows: two subspaces V_j and V_i have the same color if

$$R_i A_\ell R_j^{\mathsf{T}} = 0. \tag{2.64}$$

We emphasize that in computations A_ℓ-orthogonality is usually a stronger assumption than the orthogonality stated in (2.61); thus, a coloring scheme satisfying A_ℓ-orthogonality most likely prevents data races as well. The colored multiplicative Schwarz operator is defined as

$$P_{mu;\ell} := I - \left(I - \omega \sum_{j \in \mathcal{J}_{N_{col}}} P_j\right) \dots \left(I - \omega \sum_{j \in \mathcal{J}_1} P_j\right). \tag{2.65}$$

It may differ from the sequential operator (2.60) due to a reordering of factors.

The additive Schwarz operator factors as $P_{ad;\ell} = A_{ad;\ell}^{-1} A_\ell$. Consequently, the smoothing step $S_{ad;\ell}$ is encoded as shown in Algorithm 3. Here, we use a "for" loop for sequential operations, while "Σ" indicates a concurrent summation. The colored multiplicative Schwarz operator describes a preconditioned operator as well, but there exists no straightforward splitting $A_{mu;\ell}^{-1} A_\ell$ due to successive application of the factors in (2.65). However, in com-

Algorithm 3 Additive Schwarz smoothing step

procedure $S_{ad;\ell}(x_\ell, b_\ell)$
  $r_\ell \leftarrow b_\ell - A_\ell x_\ell$ ▷ residual
  for $c = 1$ to N_{col} **do**
    $x_\ell \leftarrow x_\ell + \omega \sum_{j \in \mathcal{J}_c} R_{\ell;j}^\mathsf{T} A_{\ell;j}^{-1} R_{\ell;j} r_\ell$ ▷ subspace correction
  end for
  return x_ℓ
end procedure

putations the application of $P_{mu;\ell} A_\ell^{-1}$ to a vector $b_\ell^{(0)}$ is recursive over colors (without ever computing A_ℓ^{-1}): given initial residual $r_\ell^{(0)} = b_\ell$ and vector $x_\ell^{(0)} \equiv 0$ intermediate residuals $r_\ell^{(c)} = b_\ell - A_\ell x_\ell^{(c)}$ are recursively defined by intermediate subspace corrections

$$x^{(c)} = x^{(c-1)} + \omega \sum_{j \in \mathcal{J}_c} R_{\ell;j}^\mathsf{T} A_{\ell;j}^{-1} R_{\ell;j} r_\ell^{(c-1)} \tag{2.66}$$

for each color $c = 1, \ldots, N_{col}$. Using

$$A_\ell^{-1} r_\ell^{(c)} = \left(I - \omega \sum_{j \in \mathcal{J}_c} P_j \right) A_\ell^{-1} r_\ell^{(c-1)}$$

and the form in (2.65), the multiplicative smoother $S_{mu;\ell}$ can be implemented as shown in Algorithm 4, similar to the additive algorithm with the single change that computing residuals happens inside the loop over all colors. Both methods are implemented as a

Algorithm 4 Multiplicative Schwarz smoothing step

procedure $S_{mu;\ell}(x_\ell, b_\ell)$
  for $c = 1$ to N_{col} **do**
    $r_\ell \leftarrow b_\ell - A_\ell x_\ell$ ▷ residual
    $x_\ell \leftarrow x_\ell + \omega \sum_{j \in \mathcal{J}_c} R_{\ell;j}^\mathsf{T} A_{\ell;j}^{-1} R_{\ell;j} r_\ell$ ▷ subspace correction
  end for
  return x_ℓ
end procedure

"short" product (in the sense of a sequence of operations) over colors with parallel, additive smoothers for each color. Since parallelization is only viable within each color, a small number of colors with many subdomains per color is desirable. That is particularly important for the multiplicative algorithm when each residual is computed using a complete operator application A_ℓ, i.e., looping over all cells.

We do not need to define local projections P_j by means of restrictions of the "global" bilinear form a_ℓ onto subspaces V_j as in (2.58). Given symmetric, coercive local bilinear

forms $a_j \colon V_j \times V_j \to \mathbb{R}$, local solvers P_j are defined by

$$a_j(P_j u_\ell, v) = a_\ell(u_\ell, R_j^\mathsf{T} v) \quad \forall v \in V_j. \tag{2.67}$$

Then, we refer to local bilinear forms differing from the restriction of the bilinear form a_ℓ onto subspaces as *inexact local solvers*. If $V_j \subset V_\ell$ the prolongation R_j^T is trivial in continuous space.

We follow Toselli and O. Widlund (2005) to briefly discuss the convergence of Schwarz operators $P_{ad;\ell}$ and $P_{mu;\ell}$. Let $\|\cdot\|_{a_\ell}$ denote the norm induced by $a_\ell(\cdot,\cdot)$ and $\operatorname{cond} P$ the condition number of operator P with respect to this norm. In order to prove abstract bounds for $\operatorname{cond} P_{ad;\ell}$ and $\|I - P_{mu;\ell}\|_{a_\ell}$, respectively, three assumptions are made.

Assumption 2.1 (Stable decomposition). There exists a constant C_{sd} such that any $v \in V_\ell$ admits a decomposition

$$v = \sum_{j=0}^{J} R_j^\mathsf{T} v_j, \quad v_j \in V_j, \tag{2.68}$$

that satisfies

$$\sum_{j=0}^{J} a_j(v_j, v_j) \leq C_{sd}^2 \, a_\ell(v, v). \tag{2.69}$$

The first assumption ensures a stable splitting of the global problem into many local problems. In order to be satisfied, a valid decomposition of V_ℓ by the family of subspaces V_j is needed. In view of (2.67), local problems do not necessarily have to be restrictions of the global problem. We note that $j = 0$ refers to a coarse problem; thus, in our multilevel context, the problem for the next level $\ell - 1$. Using coarse subspace corrections has proven to be crucial to obtain robust two-level and multilevel methods, i.e., methods that have uniform bounds for C_{sd} (almost) independent of the size of the problem (for instance, characterized by the mesh size h_ℓ or the polynomial degree k). We are particularly interested in finding cost-efficient local solvers that admit uniform bounds of C_{sd} close to one.

Assumption 2.2 (Strengthened Cauchy-Schwarz inequalities). It exist constants $0 \leq \mathcal{E}_{ij} \leq 1$ for $i, j = 1, \ldots, J$ such that

$$\left| a_\ell(R_i^\mathsf{T} u_i, R_j^\mathsf{T} v_j) \right| \leq \mathcal{E}_{ij} a_\ell(R_i^\mathsf{T} u_i, R_i^\mathsf{T} u_i)^{1/2} a_\ell(R_j^\mathsf{T} v_j, R_j^\mathsf{T} v_j)^{1/2} \tag{2.70}$$

for all $u_i \in V_i$ and $v_j \in V_j$. The spectral radius of matrix $\mathcal{E}$ is denoted $\rho(\mathcal{E})$.

We note that local bilinear forms and the coarse space are not involved, thus, Assumption 2.2 is solely a condition on the family of subspaces $V_j, j = 1, \ldots, J$. Assuming a_ℓ is symmetric, positive definite, the assumption is trivially satisfied utilizing the Cauchy-Schwarz inequality ($\mathcal{E}_{ij} = 1$ for all i, j). However, this results in $\rho(\mathcal{E}) = J$ which is unacceptable since the spectral radius of $\mathcal{E}$ bounds the error propagation of $P_{ad;\ell}$ and $P_{mu;\ell}$ from above. The matrix $\mathcal{E}$ depends on the A_ℓ-orthogonality of subspaces. Consequently, when discussing a

coloring for the multiplicative Schwarz operator in subsequent sections, we automatically find a bound for $\rho(\mathcal{E})$. In general, Gershgorin's circle theorem proves that $\rho(\mathcal{E})$ does not exceed the number of adjacent subdomains (Toselli and O. Widlund, 2005), where *adjacent* depends on the finite element discretization at hand.

Assumption 2.3 (Local stability). There exists a positive constant C_{ls} such that

$$a(R_j^{\mathsf{T}} v_j, R_j^{\mathsf{T}} v_j) \leq C_{ls}\, a_j(v_j, v_j) \qquad \forall v_j \in \operatorname{Im} P_j \subset V_j \tag{2.71}$$

for all $j = 0, \ldots, J_\ell$.

This assumption ensures that local/coarse bilinear forms are coercive and provides an upper bound for $\|P_j\|_{a_\ell}$. For exact local solvers, the constant C_{ls} is trivially one. The local stability constant C_{ls} is anti-proportional to the relaxation factor ω of local problems. A well-chosen relaxation will be a necessity to obtain robust solvers, in particular for inexact local solvers. However, scaling local forms by ω effects the constant C_{sd} in contrast to C_{ls}; thus, choosing a favorable scaling is generally not straightforward.

We conclude with abstract bounds for the additive and multiplicative Schwarz operator.

Theorem 2.3.3. *Let Assumptions 2.1 to 2.3 be satisfied.*

1. *Then, the additive Schwarz operator satisfies*

$$\operatorname{cond} P_{ad;\ell} \leq C_{sd}^2 C_{ls}(\rho(\mathcal{E}) + 1). \tag{2.72}$$

2. *Suppose that $C_{ls} \in (0, 2)$, then, the multiplicative Schwarz operator satisfies*

$$\|I - P_{mu;\ell}\|_{a_\ell} \leq \frac{2 - C_{ls}}{(2\hat{C}^2\, \rho(\mathcal{E})^2 + 1) C_{sd}^2} < 1, \tag{2.73}$$

where $\hat{C} = \max\{1, C_{ls}\}$.

Proof. See textbook (Toselli and O. Widlund, 2005). □

Further details exceeding the abstract multilevel Schwarz theory are postponed to subsequent chapters. The abstract convergence theory will ease the discussion on mathematical efficieny. In particular, in the context of inexact local solvers we will revisit Assumptions 2.1 to 2.3.

Chapter 3

POISSON PROBLEM

In this chapter, we discuss multilevel Schwarz methods for the model problem of Poisson's equation

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega, \end{aligned} \tag{3.1}$$

where Ω is a polygonal domain in $\mathbb{R}^D$ with $D = 2, 3$. f and g are given functions in $L^2(\Omega)$ and $H^{1/2+\epsilon}(\partial\Omega)$, respectively.

Solving model problem (3.1) is essential to compute several kinds of diffusion problems: we emphasize Darcy's law describing fluid flow through porous media, Fourier's law describing heat conduction, or in general Fick's law stating the flux of a diffusion process. In addition, geometric multigrid methods for the Poisson problem may be one step towards efficient incompressible flow solvers (Fehn, Wall, et al., 2017; Shahbazi et al., 2007): splitting methods have proven to be among the most efficient solvers in the computational fluid dynamics (CFD) community: the incompressible Navier-Stokes equations are decomposed into a Poisson equation for the pressure and a (convection-)diffusion equation for the velocity. To this end, engineers may demand more than 10^{10} spatial unknowns (resp. degrees of freedom) per time step to resolve physical scales realistically. Consequently, high-performance implementations are indispensable for the CFD community, solving model problem (3.1) at large scales on supercomputers.

In view of cost-efficient "forward" finite element operators for this chapter, we follow Kronbichler and others which have developed robust and accurate discretization schemes, fast matrix-free operator evaluations and scalable geometric multigrid preconditioners over recent years (Clevenger et al., 2020; Fehn, Munch, et al., 2020; Fehn, Wall, et al., 2017, 2018b,c; Krank et al., 2017). We make use of highly-optimized implementations in the general-purpose finite element library `deal.II`. The state of the art is summarized in our project's final report (Arndt, Fehn, et al., 2020). Moreover, the essential aspects of matrix-free operator evaluation for the Laplacian were elaborated in Section 2.2.2.

To obtain fast numerical solvers, matrix-free preconditioners and smoothers are as important as matrix-free operator evaluation. The listed publications above have in common that they apply multigrid methods utilizing only mathematically "simple" smoothers, for instance, diagonal preconditioners or nonoverlapping Schwarz methods. In that case, the multilevel solver's convergence degrades with increasing polynomial degree. For instance, the results in (Maitre and Pourquier, 1996) imply that using simple diagonal-based smoothers the number of iterations is proportional to k^{D-1} for a (preconditioned) conjugate gradient solver (PCG). Note that Krylov subspace solvers are usually utilized to accelerate multilevel methods. To this end, we focus in this chapter on designing cost-efficient algorithms for overlapping Schwarz smoothers, leveraging the separability of the Laplacian and the tensor structure of finite elements, quadrature, and Cartesian meshes. To be precise, we apply Schwarz smoothers on vertex patches, that result in multilevel solvers with uniform convergence regarding the mesh size, polynomial degree, and the penalty parameter for DG methods (Antonietti, Sarti, et al., 2017; Pavarino, 1993). However, they are prohibitively expensive when computing them through standard algorithms, in particular, for high-order finite elements. To this end, exploiting tensor structure, we design algorithms that are as computationally efficient as matrix-free operator evaluation.

The tensor structure of local PDE problems is utilized through fast diagonalization (Lynch et al., 1964), see Section 2.1.3 for an introduction. There exist other works concerning tensor product Schwarz smoothers based on fast diagonalization: we refer to the element-centered overlapping Schwarz methods in the context of spectral elements (P. F. Fischer and Lottes, 2005; P. F. Fischer, Tufo, et al., 2000; Lottes and P. F. Fischer, 2005) and similarly element-centered and facet-centered overlapping Schwarz methods for high-order DG methods (Stiller, 2016, 2017), that are the most-related to our approach from a methodological perspective. Nevertheless, we demonstrated in (Witte et al., 2021) and demonstrate here as first the computational efficiency of tensor product Schwarz methods on *vertex patches*, in particular, *multiplicative* methods utilizing coloring. We identified many benefits of this specific class of Schwarz methods: (i) their mathematical robustness with respect to mesh size, polynomial degree and penalty parameters providing scalability of PDE problems (in particular, using multiplicative smoothers results in almost direct solvers), (ii) vertex patches being a union of elements provide a simple memory layout, fast data access, and MPI communication similar to forward operators, and (iii) their versatility, i.e., being applicable to many PDE problems (see Chapters 4 and 5).

We recommend reading (Bastian, Müller, et al., 2018; Brubeck and Farrell, 2021; Pazner, 2020; Pazner and Kolev, 2021; Pazner and Persson, 2018) for other state-of-the-art matrix-free preconditioners that differ (partly) to our algorithms in its methodology. In particular, the methods from (Bastian, Müller, et al., 2018; Pazner and Persson, 2018) are applicable to generic PDE problems on generic meshes. In (Witte et al., 2021), we have demonstrated that our Schwarz methods may be used on non-Cartesian meshes to some extent. Pazner and

Persson (2018) utilize also fast diagonalization for subspace corrections but they obtain low-rank tensor products at the algebraic level, computing a low-rank KSVD (see Section 2.1.4) of (arbitrary) local finite element matrices. While this methodology is optimal in two spatial dimensions it becomes suboptimal in three. Bastian, Müller, et al. (2018) make use of local Krylov subspace methods, solving local PDE problems according to a pre-defined accuracy or number of iterations. Consequently, we may combine the concepts to solve other PDE problems on more generic meshes efficiently, i.e., utilizing our low-rank (inexact) subspace corrections as preconditioners for local Krylov solvers. In (Pazner, 2020; Pazner and Kolev, 2021) matrix-free subspace correction preconditioners were developed based on a space splitting of high-order SIPG discretizations (Antonietti, Sarti, et al., 2017).

The **outline** of this chapter is as follows. In Section 3.1 and Section 3.2, multilevel tensor product Schwarz methods using subspace corrections on a single cell or vertex patch for the SIPG discretization and the conforming FEM, respectively, are developed: deriving a low-rank tensor product representation which is used for fast diagonalization of local solvers and discussing the mathematical efficiency of smoothers in terms of the iterative solver's convergence steps. Since standard additive smoothers on vertex patches suffer from small relaxation factors, two simple variants of restricted additive Schwarz methods are studied in Section 3.3. As addressed in Section 1.1, mathematical efficiency by itself does not guarantee computational efficiency. To this end, we conclude the chapter with a study of (parallel) performance in Section 3.4.2: we compare the computational efficiency of our tensor product Schwarz smoothers and elaborate our C++ implementation. We discuss the "sequential" efficiency in terms of arithmetic complexity, the strong scaling behavior and the computational efficiency in terms of numerical throughput (measured in DoFs per second).

3.1 Symmetric Interior Penalty Method

The model problem is discretized through the symmetric interior penalty method (SIPG) following (Arnold, 1982; Arnold, Brezzi, et al., 2002). In contrast to conforming finite element methods, the inter-element continuity is not imposed by means of the functional space but through penalty terms entering the bilinear form. Given the mesh $\mathcal{T}_h$ with characteristic size h, we introduce the broken Sobolev space

$$H^1(\mathcal{T}_h) := \left\{ v \in L^2(\Omega) \mid v|_K \in H^1(K) \quad \forall K \in \mathcal{T}_h \right\} \not\subseteq H^1(\Omega) \tag{3.2}$$

containing functions on Ω whose restriction to cell $K \in \mathcal{T}_h$ is in $H^1(K)$.

To this end, we subdivide the physical domain Ω into many quadrilateral/hexahedral cells K, which are obtained by a mapping $\boldsymbol{F}_K$ from the reference cell $\hat{K} = [0,1]^D$. Unifying the concept of an edge in two and a face in three dimensions, the expression *facet* is introduced. As shape function spaces $V(K)$ for actual mesh cells, we choose standard Lagrange elements

based on tensor product polynomials $\mathbb{Q}_k$, see Definition 2.2.3. The integer k characterizes the polynomial degree for each coordinate direction. The finite element space on mesh $\mathcal{T}_h$ is then defined by

$$V_h := \left\{ v \in L^2(\Omega) \mid v|_K \in V(K) \quad \forall K \in \mathcal{T}_h \right\} = \bigoplus_{K \in \mathcal{T}_h} V(K). \tag{3.3}$$

The indexing of the basis $\{\varphi_{K;i}\}$ follows the structure as a direct sum. This basis defines by duality the coefficient space $\mathbb{R}^{N_h}$ of the same dimension as V_h equipped with the Euclidean inner product. In computations, this is the inner product used to compute norms, such that we will identify V_h with the coefficient space and do not distinguish them in notation. N_h defines the number of degrees of freedom subject to the mesh $\mathcal{T}_h$. We refer to functions in V_h as discontinuous tensor product elements, or in short discontinuous $\mathbb{Q}_k$-elements.

Let $\mathcal{E}_h^\circ$ be the set of all interior facets (i.e., "faces" of spatial dimension $D-1$) between any two cells K^+ and K^-. Then, we refer to traces of functions $v \in H^1(\mathcal{T}_h)$ on $e \in \mathcal{E}_h^\circ$ taken from cell $K^\pm$ as $v^\pm$. For a facet at the physical boundary, denoted by $e \in \mathcal{E}_h^\partial$, there is only a trace from the interior, and thus we define average and jump operators as follows.

Definition 3.1.1 (Average and jump operator)**.** Given $v \in H^1(\mathcal{T}_h)$ the average operator on interior facets $e \in \mathcal{E}_h^\circ$ is defined by

$$\{v\}(\boldsymbol{x}) = \frac{v^+(\boldsymbol{x}) + v^-(\boldsymbol{x})}{2} \qquad \boldsymbol{x} \in e, \tag{3.4}$$

and on boundary facets $e \in \mathcal{E}_h^\partial$

$$\{v\}(\boldsymbol{x}) = v(\boldsymbol{x}) \qquad \boldsymbol{x} \in e. \tag{3.5}$$

The jump operators are defined as

$$[\![v]\!](\boldsymbol{x}) = v^+(\boldsymbol{x}) + v^-(\boldsymbol{x}) \qquad \boldsymbol{x} \in e \in \mathcal{E}_h^\circ, \qquad [\![v]\!](\boldsymbol{x}) = v(\boldsymbol{x}) \qquad \boldsymbol{x} \in e \in \mathcal{E}_h^\partial. \tag{3.6}$$

Same definitions hold for vector-valued functions, for example $v\boldsymbol{n}$ with $\boldsymbol{n}$ being the outward pointing normal of cell K at facet e. We introduce the interior penalty bilinear form

$$\begin{aligned} a_{ip;h}(u,v) := &\int_{\mathcal{T}_h} \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v \,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\mathcal{E}_h} \left(\gamma_e \, [\![u\boldsymbol{n}]\!] \cdot [\![v\boldsymbol{n}]\!] - \{\boldsymbol{\nabla} u\} \cdot [\![v\boldsymbol{n}]\!] - [\![u\boldsymbol{n}]\!] \cdot \{\boldsymbol{\nabla} v\} \right) \mathrm{d}\sigma(\boldsymbol{x}). \end{aligned} \tag{3.7}$$

Here, the integrals over a set of cells $\mathcal{T}_h$ (set of facets $\mathcal{E}_h$) is an abuse of notation, denoting the sum of individual integrals over corresponding cells (facets). From left to right, we refer to the four integrals in (3.7) as the *bulk*, *penalty*, *consistency* and *adjoint consistency* term. We still have to define the penalty parameter γ_e, which penalizes the jumps of the solution

and yields stability of the bilinear form (Kanschat, 2003, §2.2.8). It is of the form

$$\gamma_e = \frac{\gamma_{0,e}}{2}\left(\frac{1}{h^+} + \frac{1}{h^-}\right) \quad \text{on } \mathcal{E}_h^\circ, \tag{3.8}$$

where $h^\pm$ is the (average) length of cell $K^\pm$ orthogonal to common facet e. For facet $e \in \mathcal{E}_h^\partial$ at the physical boundary, we let $h^+ = h^- = h$, where h is the length of the corresponding cell orthogonal to e. Through the average operator a factor $1/2$ enters the *consistency* and the *adjoint consistency* term on interior facets such that half of the penalty factor γ_e suffices. Consequently, we set $\gamma_{0,e} = k(k+1)$ on interior facets $e \in \mathcal{E}_h^\circ$ and $\gamma_{0,e} = 2k(k+1)$ on boundary facets $e \in \mathcal{E}_h^\partial$. The pre-factor $\gamma_{0,e}$ has to be increased by an additional factor for non-Cartesian cells to preserve the discretization's stability. Finally, we state the symmetric interior penalty method of model problem (3.1): find $u_h \in V_h$ such that

$$a_{ip;h}(u_h, v) = F(v) \qquad \forall v \in V_h, \tag{3.9}$$

where

$$F(v) := \int_\Omega fv \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} (\gamma_e\, gv - g\boldsymbol{n}\cdot\boldsymbol{\nabla} v)\,\mathrm{d}\sigma(\boldsymbol{x}), \qquad v \in H^1(\mathcal{T}_h). \tag{3.10}$$

Remark 3.1.2. The interior penalty method (3.9) is consistent and adjoint consistent. Let $u \in H^1(\mathcal{T}_h) \cap H^{3/2+\epsilon}(\Omega)$ solve the weak formulation

$$a_{ip;h}(u, v) = F(v) \quad \forall v \in H^1(\mathcal{T}_h) \cap H^{3/2+\epsilon}(\Omega). \tag{3.11}$$

Then, the rate of convergence of the finite element error $u - u_h$ is optimal subject to the energy norm and L^2-norm, respectively, (Arnold, Brezzi, et al., 2002).

3.1.1 Mathematical Efficiency of Schwarz Smoothers

Next, results for multilevel Schwarz methods on Cartesian meshes for a simple domain $\Omega = [0,1]^D$ are presented, in two and three spatial dimensions. The coarse mesh $\mathcal{T}_1$ is the decomposition of the hypercube into 2^d congruent cells, thus, it consists of a single vertex patch. A hierarchy of Cartesian meshes $\mathcal{T}_\ell$ for levels $\ell = 1, \ldots, L$ is defined by induction as outlined in Section 2.3.1. Consequently, the characteristic mesh sizes satisfy $2h_\ell = h_{\ell-1}$ for $\ell = 2, \ldots, L$. The finest level L is the actual discretization level on which we want to solve the finite element problem. The symmetric interior penalty method with homogeneous Dirichlet conditions is used for each discretization level $\ell = 1, \ldots, L$, which leads to nested finite element spaces,

$$V_1 \subset \ldots \subset V_\ell \subset \ldots \subset V_L. \tag{3.12}$$

Spaces $V_\ell := V_{h_l}$ and bilinear forms $a_\ell(\cdot,\cdot) := a_{ip;h_\ell}(\cdot,\cdot)$ are determined by (3.3) and (3.7), replacing the generic mesh width h by h_ℓ.

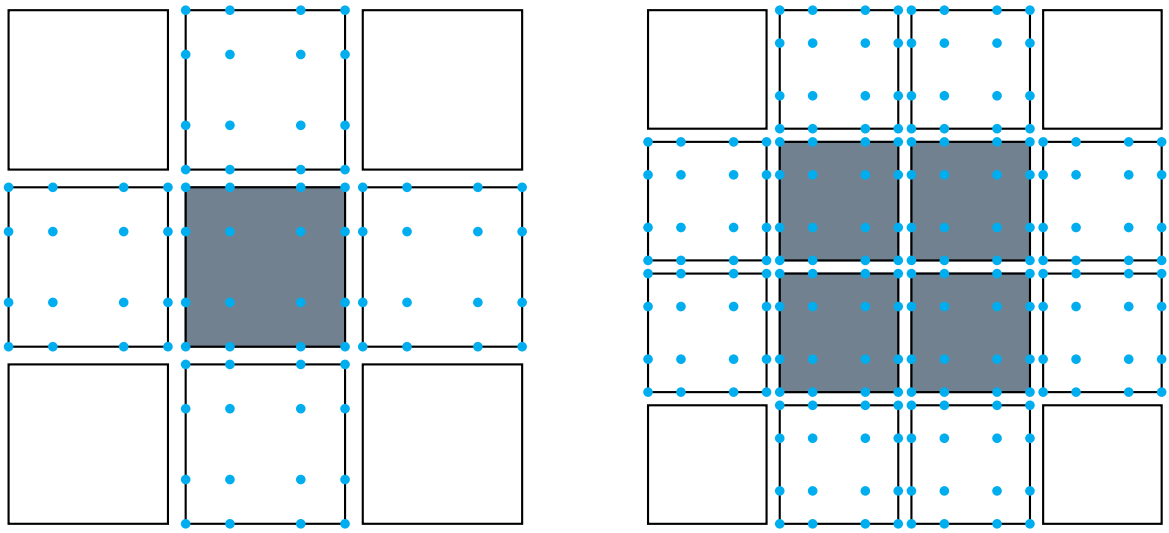


Fig. 3.1 The cell stencil (*left*) and vertex patch stencil (*right*) for SIPG given bi-cubic tensor product elements. The degrees of freedom are represented by Gauss-Lobatto support points (●). The shaded area highlights the subdomain Ω_j, thus, the shape functions associated with enclosed nodal points build a basis for the subspace $V_{\ell;j}$. Gaps between cells illustrate discontinuities.

Following the notation from Section 2.3.2, $\mathcal{T}_{\ell;j} \subset \mathcal{T}_\ell$ denotes the subset of cells that defines the subdomain Ω_j. $\mathcal{T}_j$ either contains a single cell for *cell-based* smoothers or all cells attached to an interior vertex for *vertex patch* smoothers. We tacitly suppressed the level index ℓ before when a subdomain index j was present and continue so hereafter. The discontinuous Galerkin subspace V_j consists of all functions in V_ℓ with support in the subdomain Ω_j,

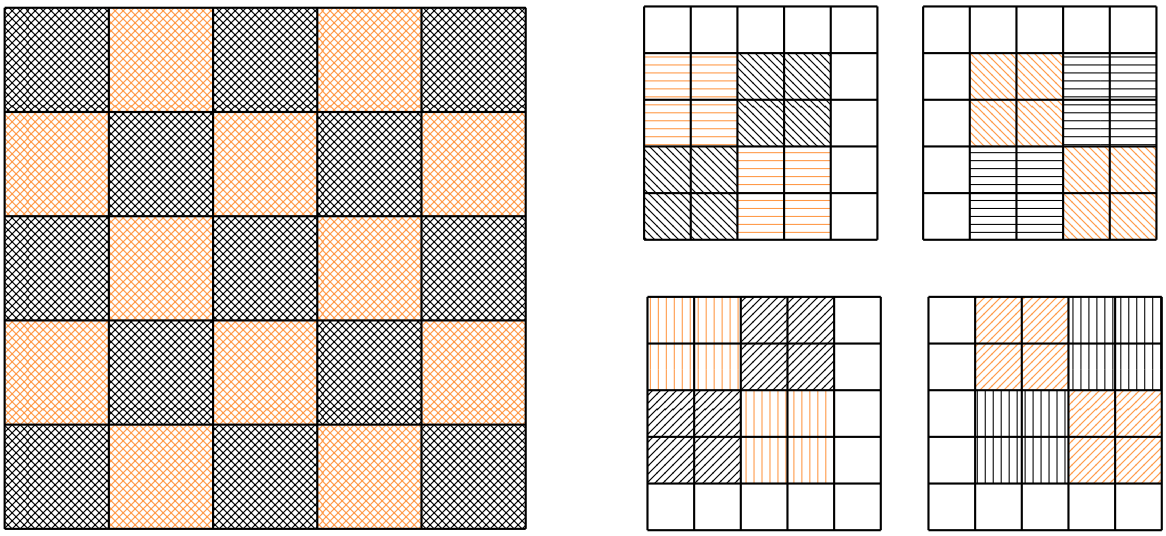
$$V_j := \bigoplus_{K \in \mathcal{T}_j} V(K). \tag{3.13}$$

The shape function space $V(K)$ was defined in (2.43). We assume exact local solvers, i.e., the local bilinear form is the restriction of the multilevel form onto $V_j \times V_j$,

$$a_j(u, v) = a_\ell(u, v) \quad \text{for all } u, v \in V_j.$$

In this way, "homogeneous" boundary conditions are weakly imposed on $\partial\Omega_j$. In Section 3.3 we elaborate what we mean by "homogeneous" and see that actual boundary conditions are inherent in the right-hand side of the local problem. Transfer operators between subspaces and the level space are standard and were defined in Section 2.3.2.

Coloring. Given a regular mesh $\mathcal{T}_\ell$, subspaces V_j and discretization matrix A_ℓ, it is straightforward to determine an optimal coloring scheme for the (multiplicative) Schwarz algorithms introduced in Section 2.3.2 - *optimal* in the sense of a minimal number of colors. Recalling the Remark 2.3.2, two subdomains Ω_i and Ω_j may have the same color if their



(a) standard "red-black" (for cells)

(b) "red-black parqueting" (for vertex patches)

Fig. 3.2 Optimal coloring for multiplicative Schwarz algorithms given DG elements.

corresponding subspaces are A_ℓ-orthogonal, i.e.,

$$R_i A_\ell R_j^\mathsf{T} = 0.$$

Note that subdomain indices i and j are interchangeable since A_ℓ is symmetric. Having the coefficient space $\mathbb{R}^{N_\ell}$ associated with V_ℓ in mind, we refer to coefficients as degrees of freedom. The action of $A_\ell R_j^\mathsf{T}$ to a local vector is visualized in Figure 3.1 in terms of discontinuous Galerkin stencils[1]. For standard discontinuous $\mathbb{Q}_k$-elements, the SIPG cell stencil consists of all degrees of freedom associated with its cell and with neighboring cells that share a joint facet. The coupling with adjacent cells is due to the face integrals in (3.7). The SIPG vertex patch stencil is simply the composition of DG cell stencils. The degrees of freedom in the shaded area determine the local coefficients that the restriction R_j maps to. We emphasize that two subspaces belonging to subdomains that mutually share only a common vertex or in three dimensions even a joint edge are A_ℓ-orthogonal. Consequently, the "red-black" coloring illustrated in Figure 3.2a is optimal for multiplicative cell-based smoothers (MCS) in two and three dimensions. Avoiding overlap demands an additional "parqueting" of regular vertex patches for each of its 2^D cells. Each parquet is colored red and black; thus, 2^{D+1} colors are obtained for multiplicative Schwarz smoothers (MVS), see Figure 3.2b. In the case of more general meshes, a graph coloring algorithm could be used. For instance, in (Witte et al., 2021) we compared the red-black coloring from Figure 3.2 to a DSATUR graph coloring algorithm.

[1]Finite element stencils are understood in the sense of (Arndt, Fehn, et al., 2020, §1).

To enable (shared memory) parallelism for additive Schwarz algorithms, the constraint in Remark 2.3.1, i.e.,

$$R_i R_j^\mathsf{T} = 0,$$

has to be satisfied. Consequently, only overlapping cells must be avoided such that an uncolored version of the parqueting in Figure 3.2b suffices for additive vertex patch smoothers (AVS). The additive cell-based smoother (ACS) has no such overlap; thus, no coloring is needed.

Relaxation. The classical Jacobi method might be damped by the factor

$$\omega_{sym} = \frac{2}{2 - \lambda_{min} - \lambda_{max}}, \tag{3.14}$$

where λ_{min} and λ_{max} denote the minimal and maximal eigenvalue of the corresponding error propagation matrix. The ω_{sym}-damped Jacobi method is optimal when used as iterative solver: ω_{sym} symmetrizes the spectrum of the error propagation matrix around zero, thus, the method converges fastest for errors with mid-range frequencies and equally "bad" for high and low frequencies (Meister, 2015). The same applies to block-Jacobi methods with no overlap, including the additive cell-based smoother (ACS). The spectrum of the error propagation matrix $I - A_{ad;\ell}^{-1} A_\ell$ associated with the additive smoothing step in Algorithm 3 is symmetric around zero for a damping parameter $\omega = 1$. It is well-known that good multigrid smoothers should efficiently smooth error modes with high to mid-high frequencies. Using a parameter ω below one shifts the spectrum of $I - A_{ad;\ell}^{-1} A_\ell$ to our needs: experiments have shown that $\omega = 0.7$ is reasonable.

An additive vertex patch smoothing step is a block-Jacobi step as well but with "overlapping blocks". For regular meshes, i.e., each vertex patch consists of 2^D cells, most degrees of freedom belong to 2^D subspaces and are smoothed this many times. The undamped block-Jacobi method would not converge, but damping with 2^{-D} recovers convergence (Meister, 2015). Not all degrees of freedom are smoothed 2^D times such that the spectrum of the error propagation matrix is not expected to be symmetric around zero. One option to obtain a good multigrid smoother is to compute the symmetrizing damping factor ω_{sym} first for the already 2^{-D}-damped method and scale ω_{sym} afterward by 0.7. Simple numerical experiments have shown that both effects, symmetrizing and shifting by 0.7, nearly cancel each other out. Therefore, we simply use a damping parameter $\omega = 2^{-D}$ for the additive vertex patch smoother (AVS).

We use a conjugate gradient method (CG) to accelerate the multigrid V-cycle (Algorithm 2). The stopping criterion is the relative reduction of the initial residual by $\epsilon_{rel} = 10^{-8}$. On each level of the V-cycle iteration a single pre- and post-smoothing step is performed. On the coarse mesh $\mathcal{T}_1$ we solve exactly by computing the inverse SVD of the coarse-grid discretization matrix A_1. The CG method is well-defined for symmetric, positive definite

Table 3.1 Fractional iterations ν_{frac} for additive cell-based smoother (ACS). CG solver with relative accuracy 10^{-8} preconditioned by multigrid with **ACS**. Entries "—" not computed, showing results with $10^5 - 10^9$ DoFs on level L.

Level L	Convergence steps ν_{frac}							
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$
5	—	—	—	—	—	—	22.7	25.6
6	—	—	14.5	16.5	16.8	18.7	22.6	25.4
7	12.2	14.5	14.4	16.5	16.8	18.7	22.6	25.4
8	12.2	14.5	14.4	16.5	16.8	18.7	22.6	25.4
9	12.2	14.5	14.3	16.4	16.8	18.7	22.6	25.4
10	12.2	14.5	14.3	16.4	16.8	18.7	22.6	25.4
11	12.1	14.4	14.3	16.4	16.8	18.7	22.6	—
12	11.9	14.4	14.3	16.4	16.8	—	—	—
13	11.9	—	—	—	—	—	—	—
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$
2	—	—	—	—	—	—	24.9	28.5
3	—	—	—	20.1	19.9	21.9	26.2	29.5
4	15.5	17.1	16.9	20.0	20.2	21.9	26.4	29.4
5	15.9	17.2	17.1	20.2	20.2	22.3	26.6	29.5
6	16.2	17.2	17.1	20.2	20.2	22.3	26.7	—
7	16.2	17.1	17.1	20.2	20.2	—	—	—
8	16.2	—	—	—	—	—	—	—

matrices. The (colored) multiplicative Schwarz smoothers are non-symmetric and thus is the preconditioned system. To this end, the symmetrized multiplicative Schwarz operator in (Toselli and O. Widlund, 2005) first traverses forwards through all colors and then backward starting from the penultimate color. Numerical experiments have shown that this nearly doubled the number of subspace corrections and residual computations in Algorithm 4 (only subspace corrections of the last color are not applied twice but once) without (significantly) decreasing the number of solver iterations. However, choosing the same multiplicative Schwarz smoother for pre- and post-smoothing, except that during pre-smoothing colors are traversed forward, and during post-smoothing backward, the multigrid V-cycle is symmetrized. The CG method becomes applicable without increasing the computational effort.

Due to their efficiency, multiplicative vertex patch smoothers require three or less iterations such that a fractional number of iterations ν_{frac} for more accurate assessment of their performance is considered. If the relative tolerance ϵ_{rel} is achieved after n solver iterations, we compute

$$\nu_{frac} = \frac{\log(\epsilon_{rel})}{\log(\rho_{avg})} \tag{3.15}$$

Table 3.2 Fractional iterations ν_{frac} for multiplicative cell-based smoother (MCS). CG solver with relative accuracy 10^{-8} preconditioned by multigrid with **MCS**. "Colors" refers to coloring on mesh level L. Entries "—" not computed, showing results with $10^5 - 10^9$ DoFs on level L.

Level L	Convergence steps ν_{frac}								Colors
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$	
5	—	—	—	—	—	—	16.4	17.8	2
6	—	—	9.9	12.4	12.3	13.7	16.3	17.7	2
7	8.6	10.4	9.9	12.4	12.3	13.7	16.2	17.7	2
8	8.6	10.3	9.9	12.4	12.3	13.7	16.3	17.7	2
9	8.6	10.3	9.9	12.4	12.3	13.7	16.3	17.7	2
10	8.5	10.3	9.9	12.4	12.3	13.7	16.3	17.7	2
11	8.5	10.3	9.9	12.4	12.3	13.7	16.3	—	2
12	8.5	10.3	9.9	12.4	12.3	—	—	—	2
13	8.5	—	—	—	—	—	—	—	2
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$	
2	—	—	—	—	—	—	18.7	20.5	2
3	—	—	—	14.4	14.5	16.6	19.4	20.8	2
4	10.2	12.2	12.1	14.5	14.5	16.7	19.5	20.9	2
5	10.2	12.3	12.1	14.5	14.5	16.7	19.5	20.9	2
6	10.2	12.4	12.1	14.5	14.5	16.7	19.4	—	2
7	10.2	12.4	12.1	14.5	14.5	—	—	—	2
8	10.2	—	—	—	—	—	—	—	2

in terms of the average residual reduction

$$\rho_{avg} = (\rho_n/\rho_0)^{1/n},$$

where ρ_n is the Euclidean norm of the residual after n steps.

The right-hand side f of our model problem (3.1) is manufactured such that the exact solution u is given by a superposition of normalized multivariate Gaussian bell curves,

$$u(\boldsymbol{x}) = \frac{1}{(\sqrt{2\pi}\sigma)^D} \sum_{i=1}^{3} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2}{\sigma^2}\right), \tag{3.16}$$

with width $\sigma = 1/3$ and sources at $\boldsymbol{x}_1 = (0,0,0)$, $\boldsymbol{x}_2 = (0.25, 0.85, 0.85)$ and $\boldsymbol{x}_3 = (0.6, 0.4, 0.4)$. In two dimensions, the source points are projected onto the xy-plane at $z = 0$.

The numerical results for additive and multiplicative cell-based smoothers are summarized in Tables 3.1 and 3.2, where for each polynomial degree k fractional convergence steps ν_{frac} for discretizations on mesh level L are shown with 10^5 to 10^9 in two and three dimensions. First, we observe that all iteration counts are independent of the mesh level. Consequently,

Table 3.3 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS). CG solver with relative accuracy 10^{-8} preconditioned by multigrid with **AVS**. Entries "—" not computed, showing results with $10^5 - 10^9$ DoFs on level L.

Level L	Convergence steps ν_{frac}							
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$
5	—	—	—	—	—	—	24.0	25.7
6	—	—	18.4	20.5	21.0	22.5	24.6	26.5
7	16.9	18.3	18.6	20.0	20.7	22.1	25.3	26.9
8	17.2	18.3	18.7	20.0	20.0	21.6	24.8	26.8
9	17.3	18.3	18.7	19.8	20.0	21.3	24.1	26.3
10	17.3	18.2	18.7	19.7	20.1	21.1	23.9	26.6
11	17.2	18.1	18.6	19.5	20.2	20.9	23.7	—
12	17.2	17.8	18.4	19.4	20.2	—	—	—
13	17.2	—	—	—	—	—	—	—
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$
2	—	—	—	—	—	—	21.3	20.9
3	—	—	—	28.5	28.5	28.9	29.5	29.6
4	27.7	29.5	30.6	32.8	32.9	33.8	36.7	37.5
5	28.7	31.8	33.4	35.6	35.8	38.1	41.9	44.5
6	29.4	32.2	34.2	37.1	38.5	40.8	45.9	—
7	29.5	32.1	33.6	37.2	39.2	—	—	—
8	29.2	—	—	—	—	—	—	—

uniform convergence with respect to the mesh size is confirmed. As discussed before, the additive cell-based smoother (ACS) is damped by $\omega = 0.7$. Table 3.1 shows a slight growth of the number of iterations with respect to the polynomial degree. It takes about twice as many steps as the multiplicative version (MCS). Given that MCS with red-black coloring in (2.65) needs two applications of the operator A_ℓ in each smoothing step, both cell-based smoothers compare at similar levels.

Similarly, we compare both vertex patch smoothers in Tables 3.3 and 3.4. We previously discussed the relaxation parameter $\omega = 2^{-D}$ for the additive method (AVS) and see that challenges pointed out there are confirmed by the numerical results here: only in two dimensions uniform convergence with respect to the mesh size is observed. A subtle analysis of optimal damping parameters might recover uniform convergence in three dimensions. AVS is inferior to ACS due to a lower damping factor, requiring more iterations while simultaneously having additional computational costs due to solving "larger" local problems. For the model problem given, we recommend the additive cell-based smoother. However, in Section 3.3 we introduce restricted additive Schwarz smoothers for vertex patches which are even on par with the multiplicative vertex patch smoothers (MVS).

Table 3.4 Fractional iterations ν_{frac} for multiplicative vertex patch smoother (MVS). CG solver with relative accuracy 10^{-8} preconditioned by multigrid with **MVS**. "Colors" refers to coloring on mesh level L. Entries "—" not computed, showing results with $10^5 - 10^9$ DoFs on level L.

Level L	Convergence steps ν_{frac}								Colors
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$	
5	—	—	—	—	—	—	2.6	2.4	8
6	—	—	3.3	3.3	2.9	2.9	2.6	2.4	8
7	3.6	3.6	3.3	3.3	2.9	2.9	2.6	2.4	8
8	3.6	3.7	3.3	3.3	2.9	2.9	2.6	2.4	8
9	3.6	3.7	3.3	3.3	2.9	2.9	2.6	2.4	8
10	3.6	3.7	3.3	3.3	2.9	2.9	2.6	2.4	8
11	3.6	3.7	3.3	3.3	2.9	2.9	2.6	—	8
12	3.6	3.7	3.3	3.3	2.9	—	—	—	8
13	3.6	—	—	—	—	—	—	—	8
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$	
2	—	—	—	—	—	—	2.4	2.2	16
3	—	—	—	3.1	2.9	2.8	2.6	2.4	16
4	3.4	3.4	3.2	3.2	2.8	2.8	2.6	2.4	16
5	3.4	3.4	3.2	3.2	2.8	2.8	2.6	2.4	16
6	3.4	3.4	3.2	3.2	2.8	2.8	2.6	—	16
7	3.4	3.4	3.2	3.2	2.8	—	—	—	16
8	3.4	—	—	—	—	—	—	—	16

The latter is the only smoother analyzed in this section that uniformly converges with respect to both the mesh size and the polynomial degree k. Table 3.4 shows that we almost obtain a direct solver, in particular, for high-order discretizations. In (Witte et al., 2021), we compared the optimal red-black coloring depicted in Figure 3.2b to a graph coloring that resulted in a few more colors. Both coloring schemes yielded iteration counts close to two with a slight advantage for the red-black coloring. This result confirmed that reordering local solvers in the multiplicative Schwarz operator (2.65) affects the smoother mathematically on the one hand. On the other hand, for the given model problem (3.1) and a change in order due to different coloring, the small difference in the fractional number of solver iterations is not significant. If the mesh is regular, we recommend using the optimal coloring to reduce the computational effort. However, generic graph coloring schemes are feasible as well.

We conclude that the multiplicative vertex patch smoother is mathematically sound and may lead to fast numerical solvers if we manage to implement it cost-efficiently. Its computational effort for a single smoothing step is relatively high compared to other smoothers. Consequently, we must compare the computational effort of the complete solver to decide on the optimal version. Numerical experiments and complexity analysis of naïve smoother

algorithms in (Witte et al., 2021) show that inverting each local discretization matrix by direct methods requires a tremendous computational effort, clearly dominating the effort of the (iterative) finite element solver. Besides, storing each local matrix counteracts the benefits of matrix-free algorithms that efficiently apply level matrices, particularly the significantly reduced memory footprint. To this end, we will exploit tensor structure, which solves both challenges equally well.

3.1.2 Tensor Product Schwarz Smoothers

The Laplacian is a separable differential operator on a quadrilateral or hexahedral cell, but it remains to argue that the discontinuous Galerkin formulation on this type of cells or even on a vertex patch of these cells is as well. Indeed, this holds for Cartesian meshes only, where each cell K is a Cartesian product of intervals,

$$K = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_D, b_D]. \tag{3.17}$$

Let h_d denote the length of the interval in dimension d, the Cartesian mapping for the cell K is denoted

$$\boldsymbol{F}_K = (\boldsymbol{F}_{K;1}, \ldots, \boldsymbol{F}_{K;D})^{\mathsf{T}},$$

and determined by

$$\boldsymbol{F}_{K;d}(\hat{x}) = a_d + h_d \hat{x} \tag{3.18}$$

for all $\hat{x} \in [0, 1]$. The Jacobian of $\boldsymbol{F}_K$ is the constant, diagonal matrix $\operatorname{diag}(h_1, \ldots, h_D)$. Using isotropic tensor product elements from Definition 2.2.3, the bulk integral in (3.7) for a single cell K factorizes into

$$\int_K \boldsymbol{\nabla}\varphi_{K;\boldsymbol{i}} \cdot \boldsymbol{\nabla}\varphi_{K;\boldsymbol{j}} \,\mathrm{d}\boldsymbol{x} = \sum_{d=1}^{D} \left(\int_0^1 \frac{1}{h_d} \hat{\phi}'_{i_d} \hat{\phi}'_{j_d} \,d\hat{x}_d \prod_{\delta=1,\delta\neq d}^{D} \int_0^1 \hat{\phi}_{i_\delta} \hat{\phi}_{j_\delta} h_\delta \,d\hat{x}_\delta \right). \tag{3.19}$$

The Cartesian mapping preserves the separation of unit coordinates in real space in (3.19). Taking also the (exact) tensor product quadrature from Section 2.2 into account, we define univariate mass and bulk matrices, respectively,

$$M_{ij}^{(d)} = \sum_{q=1}^{n_{quad}} \hat{\phi}_i(\hat{x}_q) \hat{\phi}_j(\hat{x}_q) h_d w_q \quad \text{and} \quad L_{ij}^{(d)} = \sum_{q=1}^{n_{quad}} \frac{1}{h_d} \hat{\phi}'_i(\hat{x}_q) \hat{\phi}'_j(\hat{x}_q) w_q \tag{3.20}$$

for $i, j = 1, \ldots, n_{dof}$. From the separable structure in (3.19) immediately follows that the bulk integral for a single cell K has a separable Kronecker representation

$$\left[\bigotimes_{d=1}^{D} L^{(d)} \boxtimes \bigotimes_{d=1}^{D} M^{(d)} \right]. \tag{3.21}$$

For details on this representation and its notation, we refer to Section 2.1.3. We want to prove similar Kronecker representations for face integrals of the interior penalty formulation. For simplicity, assume the two facets e_0 and e_1 of cell K having normals aligned with the first coordinate that read

$$e_p = \{y_p\} \times [a_2, b_2] \times \cdots \times [a_D, b_D] \tag{3.22}$$

with endpoints $y_0 = a_1$ and $y_1 = b_1$, respectively. Again a change of variables decomposes the penalty integral in (3.7) on these facets into

$$\int_{e_p} \gamma_{e_p} \varphi_{K;\boldsymbol{i}} \boldsymbol{n} \cdot \varphi_{K;\boldsymbol{j}} \boldsymbol{n} = \gamma_{e_p} \hat{\phi}_{i_1}(p) \hat{\phi}_{j_1}(p) \prod_{d=2}^{D} \int_0^1 \hat{\phi}_{i_d} \hat{\phi}_{j_d} h_d \, d\hat{x}_d. \tag{3.23}$$

Similar decompositions hold for the two facets in each remaining dimension. We deliberately omit to index a facet by its normal direction for readability because it is apparent from the context. We note that the cell-based local solver A_j only involves shape functions in subspace V_j, defined in (3.13), such that jump and average contributions involving adjacent cells are not present. Let the univariate point mass matrices $M_p^{(d)}$ at unit endpoints $p = 0$ and $p = 1$ be determined by

$$\left(M_p^{(d)}\right)_{ij} = \hat{\phi}_i(p) \hat{\phi}_j(p) \tag{3.24}$$

for $i, j = 1, \ldots, n_{dof}$, then, the sum of penalty terms (3.23) over all facets of K admits a separable form

$$\left[\bigotimes_{d=1}^{D} \left(\gamma_{e_0} M_0^{(d)} + \gamma_{e_1} M_1^{(d)} \right) \boxtimes \bigotimes_{d=1}^{D} M^{(d)} \right]. \tag{3.25}$$

It is an easy exercise to derive separable Kronecker representations for the consistency and adjoint consistency integrals in (3.7) as well. For both, it suffices to define matrices $G_p^{(d)}$ by

$$\left(G_p^{(d)}\right)_{ij} = (-1)^{p+1} \frac{1}{h_d} \hat{\phi}_i(p) \hat{\phi}_j'(p) \tag{3.26}$$

for $i, j = 1, \ldots, n_{dof}$. Note that $G_p^{(d)}$ represents the normal gradient and the alternating sign $(-1)^{p+1}$ identifies the normal direction. Finally, for each dimension, the penalty, consistency, and adjoint consistency integrals add up in this order to the matrix

$$N^{(d)} = \sum_{p \in \{0,1\}} \left(\gamma_{e_p} M_p^{(d)} - \eta_{e_p} G_p^{(d)} - \eta_{e_p} \left(G_p^{(d)}\right)^{\mathsf{T}} \right) \tag{3.27}$$

with average factor $\eta_e = 1/2$ for interior facets $e \in \mathcal{E}_h^\circ$, and $\eta_e = 1$ for facets $e \in \mathcal{E}_h^\partial$ at the physical boundary. We refer to $N^{(d)}$ as *univariate Nitsche matrices* since the face integrals inherent in the SIPG bilinear form are traced back to Nitsche's work (Nitsche, 1971). The

cell-based local solver A_j has the separable Kronecker form

$$\left[\bigotimes_{d=1}^{D}\left(L^{(d)}+N^{(d)}\right) \boxtimes \bigotimes_{d=1}^{D} M^{(d)}\right]. \tag{3.28}$$

Consequently, inverting A_j becomes cost-efficient when utilizing the fast diagonalization technique, as discussed in Section 2.1.3. Besides, the memory footprint is significantly reduced as explained in Remark 2.1.15.

Still assuming Cartesian meshes, vertex patches Ω_j have the same structure as in (3.17) but each interval is the disjoint union of two subintervals,

$$\bigtimes_{d=1}^{D}\left([a_d^+, b_d^+] \dot\cup [a_d^-, b_d^-]\right).$$

For corresponding local solvers, we additionally have to consider integrals on interfaces, involving the jump and/or average operator of discontinuous shape functions. The mass, the bulk, and Nitsche contributions are determined by (3.20) and (3.27) for traces of shape functions with support either exclusively in K^+ or K^-; thus, we denote them $M_+^{(d)}$, $L_+^{(d)}$, and $N_+^{(d)}$ or $M_-^{(d)}$, $L_-^{(d)}$, and $N_-^{(d)}$, respectively.

For simplicity, assume two adjacent cells

$$K^+ = [a_1^+, b_1^+] \times [a_2^+, b_2^+] \times \cdots \times [a_D^+, b_D^+] \quad \text{and} \quad K^- = [a_1^-, b_1^-] \times [a_2^+, b_2^+] \times \cdots \times [a_D^+, b_D^+],$$

with interface

$$e = \{b_1^+\} \times [a_2^+, b_2^+] \times \cdots \times [a_D^+, b_D^+],$$

that means $b_1^+ = a_1^-$. The normals $\boldsymbol{n}^+ = -\boldsymbol{n}^-$ are aligned with the first coordinate. The penalty integral on this interface with test functions from K^+ and ansatz functions from K^- reads

$$\int_e \gamma_e \, \varphi_{K;\boldsymbol{i}}^+ \boldsymbol{n}^+ \cdot \varphi_{K;\boldsymbol{j}}^- \boldsymbol{n}^- = -\gamma_e \hat\phi_{i_1}(1)\hat\phi_{j_1}(0) \prod_{d=2}^{D} \int_0^1 \hat\phi_{i_d}\hat\phi_{j_d} h_d^+ \, d\hat{x}_d. \tag{3.29}$$

The same applies to test functions from K^- and ansatz functions from K^+, interchanging $-$ and $+$ in (3.29). We define the univariate point mass matrices $M_{+-}^{(d)}$ and $M_{-+}^{(d)}$ by its entries

$$\left(M_{+-}^{(d)}\right)_{ij} = \hat\phi_i^+(1)\hat\phi_j^-(0) \quad \text{and} \quad \left(M_{-+}^{(d)}\right)_{ij} = \hat\phi_i^-(0)\hat\phi_j^+(1) \tag{3.30}$$

for $i, j = 1, \ldots, n_{dof}$. The separable structure of (3.29) leads to the rank-1 Kronecker tensor

$$-\gamma_e M_{+-}^{(1)} \otimes M_+^{(2)} \otimes \ldots \otimes M_+^{(D)}. \tag{3.31}$$

Similar rank-1 tensors are obtained for all remaining interfaces of vertex patch Ω_j. Furthermore, separable structures for the consistency and adjoint consistency terms on interfaces are derived analogously to (3.29) to (3.31). Then, interface matrices $G_{+-}^{(d)}$ and $G_{-+}^{(d)}$ for the normal gradient are given by

$$\left(G_{+-}^{(d)}\right)_{ij} = \frac{1}{h_d^-}\hat{\phi}_i(1)\hat{\phi}_j'(0) \quad \text{and} \quad \left(G_{-+}^{(d)}\right)_{ij} = -\frac{1}{h_d^+}\hat{\phi}_i(0)\hat{\phi}_j'(1) \tag{3.32}$$

for $i, j = 1, \dots, n_{dof}$. For each dimension d, the univariate Nitsche matrix for a generic interface e reads

$$N_{e;*}^{(d)} = \gamma_e M_*^{(d)} - \frac{1}{2}G_*^{(d)} - \frac{1}{2}\left(G_*^{(d)}\right)^\intercal \tag{3.33}$$

with placeholder $*$ for $+-$ and $-+$, respectively. Then, the local finite element matrix A_j for vertex patch Ω_j has the separable Kronecker representation

$$\left[\bigotimes_{d=1}^{D}\left(\boldsymbol{L}^{(d)} + \boldsymbol{N}^{(d)}\right) \boxtimes \bigotimes_{d=1}^{D}\boldsymbol{M}^{(d)}\right] \tag{3.34}$$

with the mass, the bulk, and Nitsche matrices defined by

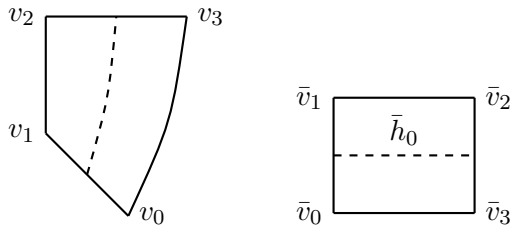
$$\boldsymbol{M}^{(d)} = \begin{bmatrix} M_+^{(d)} & 0 \\ 0 & M_-^{(d)} \end{bmatrix}, \quad \boldsymbol{L}^{(d)} = \begin{bmatrix} L_+^{(d)} & 0 \\ 0 & L_-^{(d)} \end{bmatrix} \quad \text{, and} \quad \boldsymbol{N}^{(d)} = \begin{bmatrix} N_+^{(d)} & N_{-+}^{(d)} \\ N_{+-}^{(d)} & N_-^{(d)} \end{bmatrix}. \tag{3.35}$$

for each dimension d. Therefore, the fast diagonalization is applicable again such that the computational effort of inverting and storing local matrices is significantly reduced.

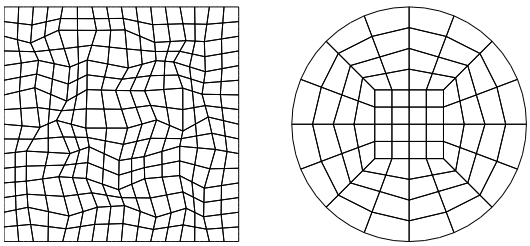
Non-Cartesian Meshes

The separable Kronecker representation of the local discretization matrix, recalling (3.28) for a single cell or (3.34) for a vertex patch, is intrinsically related to the Cartesian mapping of cells. In other words, the separability of the Laplacian is not preserved for arbitrary meshes. To this end, we have developed inexact local solvers in (Witte et al., 2021) that still admit fast diagonalization: the actual subdomain was replaced by a surrogate subdomain with a Cartesian structure. Averaging arc lengths of parallel edges as illustrated in Figure 3.3a lead to an approximating hyper-rectangle, where *parallel* is understood in a topological sense. Since the Laplacian is invariant under translation and rotation, the position and orientation of the surrogate geometry did not matter.

The numerical experiments from (Witte et al., 2021) for the non-Cartesian meshes, depicted in Figure 3.3b, showed that the number of CG iterations are still independent of the mesh size for these types of inexact local solvers. Non-standard relaxation parameters ω were used (which compensated the inexactness of the local solvers) such that only a few more iterations were needed compared to exact local solvers, which are lacking tensor structure



(a) surrogate hyper-rectangle (*right*) of the actual subdomain (*left*)



(b) non-Cartesian meshes from (Witte et al., 2021)

Fig. 3.3 Surrogate hyper-rectangle and non-Cartesian meshes from (Witte et al., 2021)

and, thus, are costly to compute. Although requiring more iterations, these inexact tensor product smoothers outperformed their exact counterparts in terms of computational effort, with a quickly growing gap for increasing polynomial degrees (Kronbichler, Kormann, Fehn, et al., 2019; Witte et al., 2021).

To avoid substantial duplication, we refer the interested reader to (Witte et al., 2021, §4) or (Arndt, Fehn, et al., 2020, §5.2) for more details. Therein, we elaborate the design of inexact local solvers based on surrogate hyper-rectangles, discuss challenges arising from Schwarz theory (surrounding the topic of relaxation), and demonstrate superior computational efficiency for the model problems illustrated in Figure 3.3b over standard smoothing algorithms.

3.2 Conforming Finite Element Method

Compared to the symmetric interior penalty method, we may take a step back in the finite element's complexity and the methods' chronology. H^1-conforming finite element methods for elliptic partial differential equations are well studied and covered in many popular textbooks. We refer to (Braess, 2013; P. Ciarlet, 1978) as the two representatives of our choice. However, designing cost-efficient algorithms of tensor product Schwarz smoothers for conforming finite elements is also a natural step forward, in particular, in view of Chapters 4 and 5: the multilevel Schwarz methods for the biharmonic and Stokes problem will also make use of H^1-conforming finite elements.

More importantly, we will see that the (multiplicative) Schwarz smoother on vertex patches has some significant algorithmic benefits. We demonstrate that a "thinner" finite element stencil[2] for local solvers on vertex patches (implying also less colors for MVS) reduces the computational complexity for a single smoothing step compared to DG elements. The final comparison of Schwarz smoothers for conforming and discontinuous elements in terms of computational efficiency follows afterward in Section 3.4. In addition, the cost-efficient Schwarz smoothers gain more importance given the hybrid multigrid methods in (Fehn, Munch,

[2]Finite element stencils are understood in the sense of (Arndt, Fehn, et al., 2020, §1).

et al., 2020). These multigrid methods utilize not only all kinds of geometric, polynomial, and algebraic coarsening, but also a beneficial transfer from discontinuous to conforming finite elements. In this regard, the tensor product Schwarz smoothers from Sections 3.1.2 and 3.2.2 are both of interest.

We define the ansatz space

$$V^g = \left\{ v \in H^1(\Omega) \mid v|_{\partial\Omega} = g \right\}, \tag{3.36}$$

imposing Dirichlet boundary conditions in a strong sense, where $v|_{\partial\Omega}$ is well-defined by the trace operator. The Dirichlet problem for Poisson's equation (3.1) in weak form reads: find $u \in V$ such that

$$a(u, v) = F(v) \qquad \forall v \in H_0^1(\Omega), \tag{3.37}$$

with bilinear form

$$a(u, v) := \int_{\mathcal{T}_h} \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v \, \mathrm{d}\boldsymbol{x}, \tag{3.38}$$

and right-hand side operator

$$F(v) := \int_{\Omega} f v \, \mathrm{d}\boldsymbol{x}. \tag{3.39}$$

Similar to the SIPG method, we choose local shape function spaces $V(K)$ that are defined by duality to nodal Gauss-Lobatto points, see Definitions 2.2.2 and 2.2.3. The difference here is the additional inter-element continuity: the conforming finite element space V_h is defined by

$$V_h = \left\{ v \in C^0(\bar{\Omega}) \mid v|_K \in V(K) \quad \forall K \in \mathcal{T}_h \right\} \cap H_0^1(\Omega). \tag{3.40}$$

A conforming finite element space V_h^g with nonzero boundary data is obtained when replacing $H_0^1(\Omega)$ by V^g. The finite element method for the model problem reads: find $u_h \in V_h^g$ such that

$$a(u_h, v) = F(v) \qquad \forall v \in V_h. \tag{3.41}$$

Well-posedness of the weak as well as finite element method and a rigorous error analysis is studied in (Braess, 2013; P. Ciarlet, 1978), for instance.

3.2.1 Mathematical Efficiency of Schwarz Smoothers

The same numerical setup as in Section 3.1.1 is used, in particular, the way a hierarchy of nested meshes $\mathcal{T}_\ell$ for $\ell = 1, \ldots, L$ is obtained. We only show results for vertex patches here. The cell-based Schwarz smoothers for conforming finite elements are inferior from an algorithmic perspective. A closer look at local finite element stencils will reveal why.

What was discussed before for a mesh $\mathcal{T}_h$ with generic size h readily translates to each level mesh $\mathcal{T}_\ell$. For each discretization level finite element spaces $V_\ell := V_{h_\ell}$ are defined by (3.40) substituting h by h_ℓ. As before, let $\mathcal{T}_j \subset \mathcal{T}_\ell$ denote the vertex patch, that is the local mesh

of the subdomain

$$\Omega_j = \bigcup_{K \in \mathcal{T}_j} K.$$

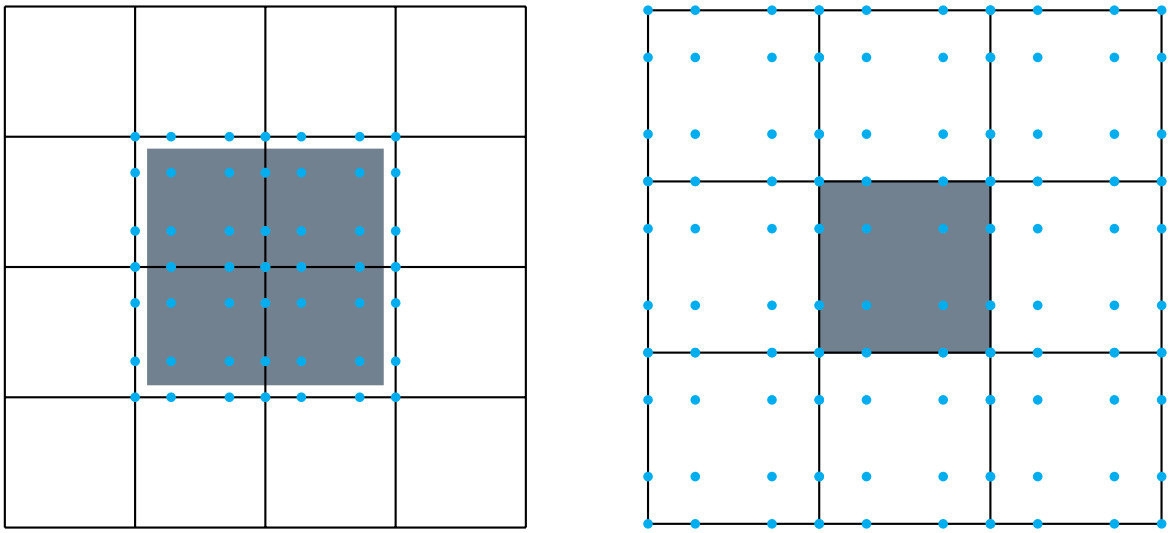
The local finite element space V_j consists of all functions in V_ℓ with vanishing trace on $\partial\Omega_j$,

$$V_j = \left\{ v \in C^0(\bar{\Omega}) \mid v|_K \in V(K) \; \forall K \in \mathcal{T}_j \quad \text{and} \quad v|_{\partial\Omega_j} = 0 \right\}. \tag{3.42}$$

Note that local functions in V_j are simply extended by zero outside the local subdomain Ω_j. Bilinear forms for each level and each subdomain are simply the restrictions of $a(\cdot,\cdot)$ to each finite element space V_ℓ and each subspace V_j, respectively. Consequently, Dirichlet problems with homogeneous boundary conditions are solved on each subdomain.

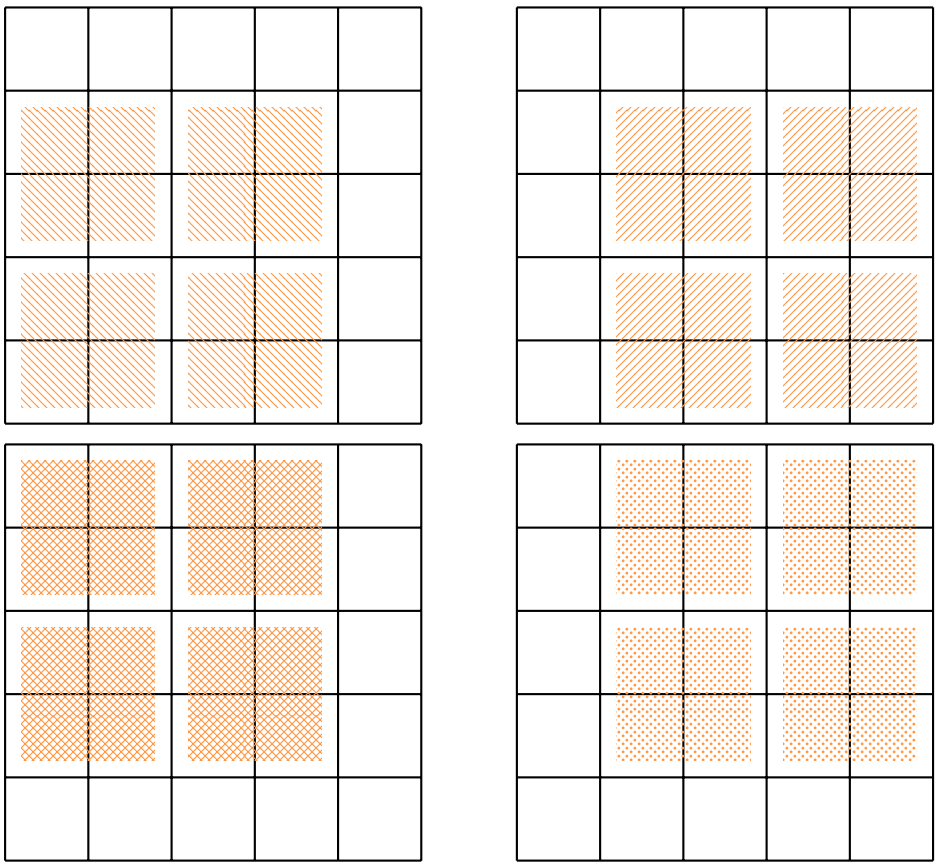
The local finite element stencil is illustrated in Figure 3.4a. The face integrals which were essential for the shape of the SIPG stencils in Figure 3.1 are, on the one hand, not present. On the other hand, the inter-element continuity introduces a different coupling between degrees of freedom. Fortunately, degrees of freedom associated with the subdomain's boundary are constrained to zero, such that the stencil does not spread out to adjacent cells, i.e., the closest neighbors of the vertex patch. Cell-based Schwarz smoothers should smooth each degree of freedom. To this end, the local finite element space associated with a single cell K is spanned by basis shape functions with support in K, including basis functions associated with degrees of freedom at ∂K. This minor difference has a major effect, illustrated by the stencil in Figure 3.4b. While the ratio of the number of the subspace's degrees of freedom to those for its stencil is about $1/3^D$ for cell-based subdomains, it is almost one for vertex patches. In (P. F. Fischer and Lottes, 2005; P. F. Fischer, Tufo, et al., 2000; Lottes and P. F. Fischer, 2005; Stiller, 2016, 2017) element-centered patches with a minimal overlap were studied that have intermediate sizes compared to the cell-based and vertex patches. However, we decided for the generous overlap of vertex patches because it simplifies data structures and communication for memory-distributed parallelism. Moreover, exploiting tensor structures to break the curse of dimensionality, the more generous overlap is computationally negligible. Thus, most important will be the localization of stencils to the vertex patches' union of cells.

In general, finite element stencils with very good data locality (Chang and Grady, 2019) can improve parallel performance, particularly when using distributed memory parallelism. To this end, we developed in (Kronbichler, Kormann, Fehn, et al., 2019) a so-called *Hermite-like* finite element based on discontinuous tensor product polynomials $\mathbb{Q}_k$ to obtain also more localized SIPG stencils than those of standard Lagrange elements from Figure 3.1. The number of iterations for the same solver as in Section 3.1.1 with Hermite-like finite elements were almost identical to the numbers in Tables 3.1 to 3.4, using tensor product Schwarz smoothers similar to Section 3.1.2. At the same time, the performance was increased. It underlines again the versatility of Schwarz smoothers on vertex patches.



(a) vertex patch stencil

(b) cell stencil



(c) "parqueting" (for vertex patches)

Fig. 3.4 The vertex patch stencil (*top-left*), cell stencil (*top-right*), and a coloring for MVS (*bottom*) for H^1-conforming discretizations given bi-cubic Lagrange elements are illustrated. Degrees of freedom are represented by Gauss-Lobatto support points (●). The shaded areas highlight the subdomain Ω_j, a vertex patch (*top-left*) and a single cell (*top-right*), thus, the shape functions associated with included nodal points form a basis for the subspace $V_{\ell;j}$. Hatched and dotted areas (*bottom*) show the parquet-like coloring satisfying A_ℓ-orthogonality.

Coloring. The vertex patch stencil visualizes the action of $A_\ell R_j^\mathsf{T}$ to local coefficients: Figure 3.4a shows that the action is limited to degrees of freedom associated with the cells that constitute the vertex patch. Since functions in V_i have vanishing trace at $\partial\Omega_i$, A_ℓ-orthogonality (2.64) is not satisfied if two subdomains Ω_j and Ω_i overlap by a joint cell. To this end, a parqueting of regular vertex patches as illustrated in Figure 3.4c is the optimal coloring for multiplicative Schwarz smoothers. In contrast to the symmetric interior penalty method, an additional red-black coloring for each parquet is not required (compared to Figure 3.2b) such that only half of the colors are needed here. Accordingly, the number of residual computations in a single multiplicative smoothing step (Algorithm 4) is halved. To enable shared memory parallelism for additive Schwarz smoother, the very same coloring is optimal since two subdomains without any joint cell satisfy (2.61).

Relaxation. A single additive smoothing step, see Algorithm 3, on vertex patches is a damped block-Jacobi method with overlap. Damping with the reciprocal number of cells (that is, with 2^{-D}) is natural and guarantees convergence when used as iterative solver (Meister, 2015). Computing ω_{sym} as given by (3.14) the optimal iterative solver is determined. Using the rule of thumb that a good iterative solver relaxed by 0.7 is a good smoother, damping with $\omega = 0.7\omega_{sym}$ is a reasonable choice. From numerical experiments for discretizations factors with a few thousand unknowns, relaxation factors slightly larger than 2^{-D} proved to be optimal. However, the improvement in the number of solver iterations was at most one, such that numerical results for additive Schwarz smoothers damped by 2^{-D} are shown below, regardless of the polynomial degree k and the mesh size. Further studies of relaxation parameters are left to the interested reader.

The fractional iteration counts ν_{frac} for the same solver setup used for the multilevel symmetric interior penalty method in Section 3.1.1 are discussed now. Numerical results are shown for relaxed additive Schwarz smoothers (AVS) in Table 3.5 and for multiplicative Schwarz smoothers (MVS) in Table 3.6.

First of all, both smoothers lead to robust solvers with a few iteration steps needed, thus being readily scalable to discretizations with very many unknowns. A solver is robust if it uniformly converges regarding one or several parameters of interest: here we observe that iteration counts are independent of the most refined mesh level L and the polynomial degree k. This is satisfied even for additive Schwarz smoothers in contrast to the SIPG method in Section 3.1.1.

The symmetric interior penalty method has by definition more degrees of freedom for the identical multilevel hierarchy of meshes than the conforming finite element method. Easing the comparison, we consider only the discretizations on the most refined level L with the number of degrees of freedom in a given range, here 10^5 to 10^9 in two and three dimensions. Most noticeable is the outstanding performance of the relaxed additive smoothing (AVS) here in contrast to its SIPG counterpart, compare Table 3.5 to Table 3.3. We recall that both additive vertex patch smoothers are relaxed by the same factor $\omega = 2^{-D}$, without

Table 3.5 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) regarding **conforming FEM**. CG solver with relative accuracy 10^{-8} preconditioned by multigrid with **AVS**. Entries "—" not computed, showing results with $10^5 - 10^9$ DoFs on level L.

Level L	Convergence steps ν_{frac}							
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$
5	—	—	—	—	—	—	10.9	10.8
6	—	—	—	11.2	11.3	11.2	10.8	10.7
7	—	11.6	11.7	11.1	11.2	11.1	10.8	10.6
8	12.2	11.6	11.6	11.1	11.2	11.1	10.7	10.6
9	12.1	11.6	11.5	11.0	11.1	11.1	10.7	10.5
10	11.9	11.6	11.3	10.9	11.1	11.1	10.7	10.5
11	11.8	11.6	11.2	10.9	11.1	11.1	10.7	10.5
12	11.7	11.6	11.0	10.9	11.0	11.0	—	—
13	11.7	11.6	—	—	—	—	—	—
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$
2	—	—	—	—	—	—	—	13.7
3	—	—	—	—	15.9	15.3	14.8	14.7
4	—	16.8	16.8	15.5	16.1	15.1	14.7	14.5
5	20.0	16.8	16.8	15.3	16.1	14.9	14.5	14.2
6	20.0	16.6	16.7	15.1	15.9	14.8	14.3	13.9
7	19.9	16.4	16.6	15.0	15.8	14.8	—	—
8	19.8	16.3	—	—	—	—	—	—

tuning either of them. The multilevel solvers using AVS for the SIPG method lack uniform convergence concerning the mesh level and the polynomial degree, with steadily growing iteration counts from 30 to 45 iterations regarding polynomial degrees from 2 to 15 in three dimensions. The Table 3.5 does not only confirm the solver's robustness but at least more than a 30% decrease compared to the numbers before. With 14 versus 45 iterations (i.e., 68%) for the highest polynomial degree shown, additive vertex patch smoothers for the conforming finite element method prove superior, particularly for high-orders.

Let us compare the results for both methods presented in this section, the additive and multiplicative smoother. On the hand hand, the multilevel solver using AVS requires four to five times more iterations than using MVS in three dimensions (three to four times more in two dimensions). On the other hand, MVS needs eight colors (four colors in two dimensions) compared to a single for AVS (if no shared-memory parallelism is utilized). Assuming that each residual computation in Algorithms 3 and 4 has the same computational costs regardless of the underlying color, a multiplicative smoothing step costs almost eight times (or four times in two dimensions) more than a single additive smoothing step; neglecting the computational effort of computing subspace corrections which is the same for each kind of smoothing step. In theory, AVS is slightly favored in computational terms. In Section 3.4.2, we will see that using time-to-solution as metric for computational efficiency, a solver using MVS will outperform one using AVS.

A comparison of fractional iterations between MVS for the SIPG method as well as for the conforming finite element method slightly favors the former in total numbers, with 3.6 versus 4.5 in two dimensions and with 3.4 to 5.0 in three dimensions for quadratic polynomials, respectively. Both gaps in iteration counts are closed with increasing polynomial degree. Keeping in mind that MVS for the SIPG method has twice as many colors given an optimal coloring, it will be interesting to compare the computational efficiency of both multiplicative smoothers in Section 3.4.2.

We conclude that the additive vertex patch smoother for the conforming finite element method performs remarkably well. Both multiplicative Schwarz smoothers, the one presented here and the one from Section 3.1.1, are sound and superior when it comes to mathematical efficiency. However, each multiplicative smoothing step consists of many sub-iterations, each with a computational effort similar to a single additive smoothing step. The amount of sub-iterations depends on the number of colors. Therefore, if each of these three candidates can be computed efficiently, it remains to show which is the best in computational metrics. To this end, we introduce tensor product Schwarz smoothers for H^1-conforming finite elements in the next section. Then, a study of (parallel) performance follows in Section 3.4.

Table 3.6 Fractional iterations ν_{frac} for multiplicative vertex patch smoother (MVS) regarding **conforming FEM**. CG solver with relative accuracy 10^{-8} preconditioned by multigrid with **MVS**. "Colors" refers to coloring on mesh level L. Entries "—" not computed, showing results with $10^5 - 10^9$ DoFs on level L.

Level L	Convergence steps ν_{frac}								Colors
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$	
5	—	—	—	—	—	—	2.8	2.6	4
6	—	—	—	3.8	3.4	3.4	2.9	2.6	4
7	—	4.4	3.8	3.8	3.4	3.4	2.9	2.6	4
8	4.5	4.4	3.8	3.8	3.4	3.4	2.9	2.6	4
9	4.5	4.4	3.8	3.8	3.4	3.4	2.9	2.6	4
10	4.5	4.4	3.8	3.8	3.4	3.4	2.9	2.6	4
11	4.5	4.4	3.8	3.8	3.4	3.4	2.9	2.6	4
12	4.5	4.4	3.8	3.8	3.4	3.4	—	—	4
13	4.5	4.4	—	—	—	—	—	—	4
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=11$	$k=15$	
2	—	—	—	—	—	—	—	2.4	8
3	—	—	—	—	3.3	3.3	2.8	2.6	8
4	—	4.4	3.8	3.7	3.4	3.4	2.9	2.6	8
5	5.1	4.4	3.8	3.7	3.4	3.4	2.9	2.7	8
6	5.1	4.4	3.8	3.7	3.4	3.4	2.9	2.7	8
7	5.0	4.4	3.8	3.7	3.4	3.4	—	—	8
8	5.0	4.4	—	—	—	—	—	—	8

3.2.2 Tensor Product Schwarz Smoothers

Using standard methods, local solvers for vertex patches are very memory-intense and inverting local discretization matrices through direct methods is infeasible, in particular, for high-order finite element methods. In analogy to the tensor product Schwarz smoothers from Section 3.2.2, fast diagonalization of local matrices amenable to a separable Kronecker representation overcomes these challenges.

Proving the separable Kronecker representation is even simpler than for the symmetric interior penalty method since face integrals are missing. Taking homogeneous Dirichlet boundary conditions and inter-element continuity into account will be straightforward. Assuming a Cartesian mesh $\mathcal{T}_\ell$, a vertex patch Ω_j is the Cartesian product of intervals, each subdivided into two subintervals,

$$\Omega_j = \bigtimes_{d=1}^{D} [a_d^+, b_d^+] \cup [a_d^-, b_d^-]. \tag{3.43}$$

The Cartesian mapping $\boldsymbol{F}_K \colon \hat{K} \to K$ for cells $K \in \mathcal{T}_\ell$ was defined in (3.18). The local bilinear form is given as restriction of $a_\ell(\cdot,\cdot)$ to $V_j \times V_j$, only consisting of the bulk integral on the vertex patch Ω_j. For each $K \in \mathcal{T}_\ell$ a set of shape functions $\varphi_{\boldsymbol{i}} \in V_\ell$ that form a local shape function basis for $V(K)$,

$$\varphi_{K;\boldsymbol{i}} := \varphi_{\boldsymbol{i}}|_K = \hat{\varphi}_{\boldsymbol{i}} \circ \boldsymbol{F}_K^{-1}, \tag{3.44}$$

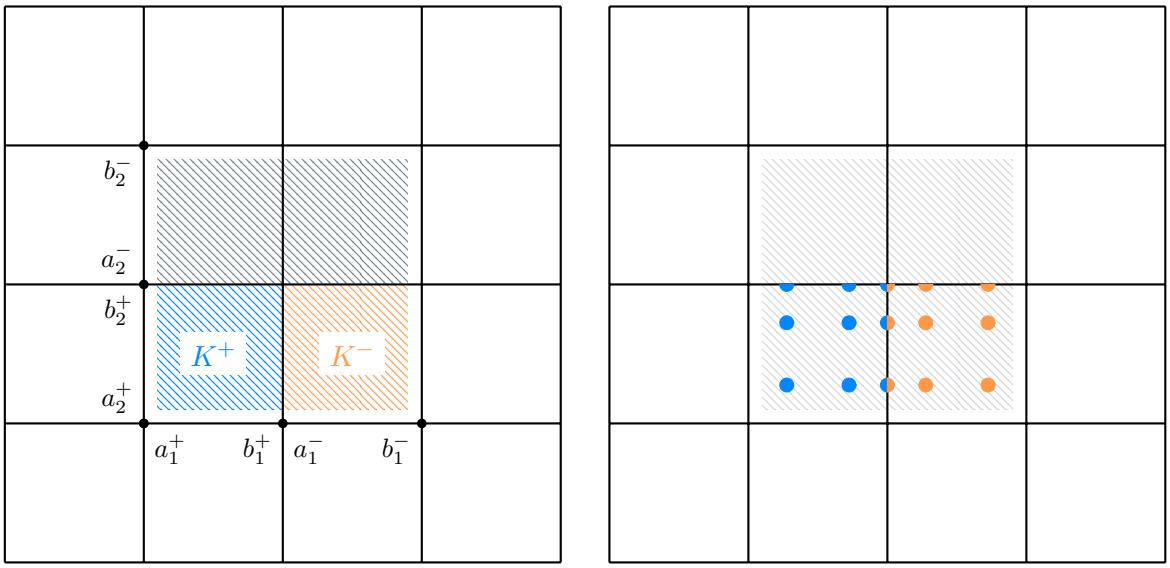
The Cartesian mapping preserves the separation of unit coordinates in real space such that the separation of variables for unit shape functions $\hat{\varphi}_{\boldsymbol{i}}$, being introduced in Definition 2.2.3, is preserved as well for shape functions $\varphi_{K;\boldsymbol{i}}$. Assuming any two basis functions φ and ψ in V_j, we denote by $\mathcal{K}$ the set of cells on which both functions have non-trivial support. As in (3.19), the bulk integral for a generic cell K factorizes into

$$\int_K \boldsymbol{\nabla}\varphi \cdot \boldsymbol{\nabla}\psi \,\mathrm{d}\boldsymbol{x} = \sum_{d=1}^{D} \left(\int_0^1 \frac{1}{h_d} \hat{\phi}'_{i_d} \hat{\phi}'_{j_d} \,d\hat{x}_d \prod_{\delta=1,\delta\neq d}^{D} \int_0^1 \hat{\phi}_{i_\delta} \hat{\phi}_{j_\delta} h_\delta \,d\hat{x}_\delta \right), \tag{3.45}$$

with $\psi|_K = \varphi_{K;\boldsymbol{i}}$ and $\varphi|_K = \varphi_{K;\boldsymbol{j}}$. For the local problem on the vertex patch Ω_j, we have

$$a_\ell(\varphi_{\boldsymbol{i}}, \varphi_{\boldsymbol{j}}) = \sum_{K\in\mathcal{K}} \int_K \boldsymbol{\nabla}\varphi \cdot \boldsymbol{\nabla}\psi \,\mathrm{d}\boldsymbol{x}$$

where each cell integral on the right-hand side admits the factorization in (3.45). It is apparent that the local discretization $\boldsymbol{A}_j$ corresponding to section 3.2.2 has a separable Kronecker representation, but we do not have univariate mass and bulk matrices as simple as in (3.35), where the indexing of the shape function basis followed the structure of a direct sum (3.3). The direct sum implied block-structured matrices. Deriving the explicit separable Kronecker



(a) vertex patch (b) local nodal points

Fig. 3.5 Visualization of a Cartesian vertex patch Ω_j (shaded area) with two adjacent cells K^+ and K^- (*left*) and corresponding shape functions in V_j (*right*) characterized by Gauss-Lobatto support points for bi-cubic Lagrange elements. The Cartesian product structure is given by intervals $[a_d^\pm, b_d^\pm]$ for each spatial dimension d. Semi- and quarter circles identify (by coloring) shape functions with support on two or four cells, respectively.

representation of $\boldsymbol{A}_j$, becomes a tedious juggling with indices, in particular, indices $\boldsymbol{i}$ and $\boldsymbol{j}$ in (3.45) implicitly depend on the cell K.

We keep it simple by focusing on a specific Cartesian vertex patch Ω_j for two spatial dimensions, illustrated in Figure 3.5. To this end, two adjacent cells

$$K^+ = [a_1^+, b_1^+] \times [a_2^+, b_2^+] \quad \text{and} \quad K^- = [a_1^-, b_1^-] \times [a_2^+, b_2^+]$$

are introduced with $b_1^+ = a_1^-$. The interface e reads

$$e = \{b_1^+\} \times [a_2^+, b_2^+]. \tag{3.46}$$

Local shape functions are visualized by their nodal points in Figure 3.5b. Shape functions with exclusive support either in K^+ or in K^- have colored circles (●) or (●), respectively. Bicolored circles characterize basis functions exclusively belonging to the interface e. Likewise, the bicolored semicircle illustrates the single shape function associated with the interior vertex, with support in four cells. Due to the Cartesian product in (3.43) any nodal point is uniquely assigned either to (a_1^+, b_1^+), to (a_1^-, b_1^-) or to the "midpoint" $\{b_1^+\}$. Following the indexing of univariate Lagrange elements from Definition 2.2.2, if on cell K^+ the last shape function $\hat{\phi}_{n_{dof}}$ belongs to e, on cell K^- it is the first $\hat{\phi}_1$. The same applies to any other dimension and generalizes to Cartesian vertex patches with more than two spatial dimensions.

Using the tensor product Gaussian quadrature from Section 2.2 the univariate mass matrix with respect to $[a_d^+, b_d^+] \cup [a_d^-, b_d^-]$ reads

$$\boldsymbol{M}^{(d)} = \begin{bmatrix} M_{++}^{(d)} & M_{+e}^{(d)} & 0 \\ (M_{+e}^{(d)})^\mathsf{T} & M_{ee}^{(d)} & (M_{-e}^{(d)})^\mathsf{T} \\ 0 & M_{-e}^{(d)} & M_{--}^{(d)} \end{bmatrix} \tag{3.47}$$

with

$$\begin{aligned} \left(M_{\pm\pm}^{(d)}\right)_{ij} &= \sum_{q=1}^{n_{quad}} \left(\hat{\phi}_{i+1}(\hat{x}_q)\hat{\phi}_{j+1}(\hat{x}_q)h_d^{\pm}w_q\right), \\ \left(M_{ee}^{(d)}\right)_{11} &= \sum_{q=1}^{n_{quad}} \left(\hat{\phi}_{n_{dof}}(\hat{x}_q)\hat{\phi}_{n_{dof}}(\hat{x}_q)h_d^{+}w_q + \hat{\phi}_1(\hat{x}_q)\hat{\phi}_1(\hat{x}_q)h_d^{-}w_q\right), \\ \left(M_{+e}^{(d)}\right)_{i1} &= \sum_{q=1}^{n_{quad}} \left(\hat{\phi}_{i+1}(\hat{x}_q)\hat{\phi}_{n_{dof}}(\hat{x}_q)h_d^{+}w_q\right), \\ \left(M_{-e}^{(d)}\right)_{i1} &= \sum_{q=1}^{n_{quad}} \left(\hat{\phi}_{i+1}(\hat{x}_q)\hat{\phi}_1(\hat{x}_q)h_d^{-}w_q\right), \end{aligned} \tag{3.48}$$

for all $i = 1, \ldots, n_{dof}-2$ and $j = 1, \ldots, n_{dof}-2$. Here, $h_d^{\pm}$ denotes the length of interval $[a_d^{\pm}, b_d^{\pm}]$. The subscript e indicates the joint vertex $b^+ = a^-$ between both intervals. Consequently, the block $M_{ee}^{(d)}$ has a single entry. We emphasize that the index $i+1$ in definitions of $M_{\pm\pm}^{(d)}, M_{+e}^{(d)}$ and $M_{-e}^{(d)}$ skips $i = 1$ and leaves out $i = n_{dof}$, thus neglecting shape functions associated with $\partial\Omega_j$. Neglecting shape functions at the boundary implicitly imposes homogeneous Dirichlet boundary conditions for the local finite element problem. Similarly, univariate bulk matrices $\boldsymbol{L}^{(d)}$ are defined,

$$\boldsymbol{L}^{(d)} = \begin{bmatrix} L_{++}^{(d)} & L_{+e}^{(d)} & 0 \\ (L_{+e}^{(d)})^\mathsf{T} & L_{ee}^{(d)} & (L_{-e}^{(d)})^\mathsf{T} \\ 0 & L_{-e}^{(d)} & L_{--}^{(d)} \end{bmatrix} \tag{3.49}$$

with

$$\begin{aligned} \left(L_{\pm\pm}^{(d)}\right)_{ij} &= \sum_{q=1}^{n_{quad}} \left(\frac{1}{h_d^{\pm}}\hat{\phi}'_{i+1}(\hat{x}_q)\hat{\phi}'_{j+1}(\hat{x}_q)w_q\right), \\ \left(L_{ee}^{(d)}\right)_{11} &= \sum_{q=1}^{n_{quad}} \left(\frac{1}{h_d^{+}}\hat{\phi}'_{n_{dof}}(\hat{x}_q)\hat{\phi}'_{n_{dof}}(\hat{x}_q)w_q + \frac{1}{h_d^{-}}\hat{\phi}'_1(\hat{x}_q)\hat{\phi}'_1(\hat{x}_q)w_q\right), \\ \left(L_{+e}^{(d)}\right)_{i1} &= \sum_{q=1}^{n_{quad}} \left(\frac{1}{h_d^{+}}\hat{\phi}'_{i+1}(\hat{x}_q)\hat{\phi}'_{n_{dof}}(\hat{x}_q)w_q\right), \\ \left(L_{-e}^{(d)}\right)_{i1} &= \sum_{q=1}^{n_{quad}} \left(\frac{1}{h_d^{-}}\hat{\phi}'_{i+1}(\hat{x}_q)\hat{\phi}'_1(\hat{x}_q)w_q\right), \end{aligned} \tag{3.50}$$

The specific ordering of blocks in (3.47) and (3.49) is crucial such that the lexicographic order from Definition 2.1.6 for the shape function basis of subspace V_j is obtained. Consequently, the separable Kronecker representation of the local discretization matrix $\boldsymbol{A}_j$ reads

$$\boldsymbol{A}_j = \left[\bigotimes_{d=1}^{D} \boldsymbol{L}^{(d)} \boxtimes \bigotimes_{d=1}^{D} \boldsymbol{M}^{(d)} \right], \tag{3.51}$$

This tensor representation enables fast diagonalization, which overcomes the memory-intensity of local solvers and significantly reduces the arithmetic complexity, while still computing exact inverses. We refer to Section 2.1.3 a theoretical comparison of using fast diagonalization instead of direct methods for inverting matrics. In practice, we highlight these benefits in Section 3.4, where we compare to the efficient operator application from Section 2.2 and study the (parallel) performance of our algorithms.

3.3 Restricted Additive Schwarz Methods

In Section 3.1.1, we observed that additive vertex patch smoothers for the SIPG method led to slow non-uniform solver convergence, most likely due to small relaxation factors. On the contrary, the same smoothers for the H^1-conforming multilevel method performed much better. To this end, we present a variant of additive vertex patch methods that do not need relaxation at all.

Restricted additive Schwarz methods (RAS) were accidentally found and first discussed in (Cai and Sarkis, 1999). The authors showed that a RAS preconditioner leads to fewer iterations than the standard additive Schwarz preconditioner (AS), both accelerated by GMRES, for a finite difference scheme with a five-point stencil concerning Poisson and convection-diffusion problems, respectively. Efstathiou and Gander (2003) addressed the question: *Why is RAS a convergent or even faster converging iterative solver than AS?* We note that in both publications the (standard) additive Schwarz methods (AS) was used without relaxation.

We focus on restricted additive Schwarz smoothers for vertex patches, answering the question: *Are RAS smoothers superior in terms of the solver's convergence steps when compared to their AS counterparts using a simple relaxation factor?* The latter smoothers were presented in Sections 3.1.1 and 3.2.1. Stiller, 2016, 2017 studied restricted additive Schwarz smoothers using nonuniformly weighting functions for the SIPG method from Section 3.1, but not for smoothers on vertex patches. We compare our results later.

First, for each subdomain Ω_j a weighting operator $W_{\ell;j}$ is introduced which satisfies a *partition of unity.*

Assumption 3.1 (Partition of unity). The restriction operators $R_{\ell;j}$ and weighting operators $W_{\ell;j}$ define a *partition of unity* if

$$\sum_{j=1}^{J_\ell} R_{\ell;j}^{\mathsf{T}} W_{\ell;j} R_{\ell;j} = \mathrm{id}, \tag{3.52}$$

where id is the respective identity operator for the finite element space V_ℓ.

The standard additive Schwarz smoothers were introduced in Section 2.3.2. We refer there for the corresponding notation used here. For brevity, we suppress the level index ℓ. In computations, we identify the finite element subspace V_j with the coefficient space $\mathbb{R}^{N_j}$ and do not distinguish them in notation. Similarly, W_j defines a square matrix $\mathbb{R}^{N_j \times N_j}$ with respect to the coefficient space. In Algorithm 5, the restricted additive Schwarz smoother (RAS) is defined (not suppressing the level index for the sake of completeness). Assuming W_j is simply the identity matrix for each subdomain, RAS is identical to the standard additive Schwarz smoother without relaxation, i.e., Algorithm 3 with $\omega = 1$. In general, given two

Algorithm 5 Restricted Additive Schwarz smoothing step

procedure $S_{ad;\ell}(x_\ell, b_\ell)$
  $r_\ell \leftarrow b_\ell - A_\ell x_\ell$ ▷ update residual
  for $c = 1$ to N_{col} **do**
    $x_\ell \leftarrow x_\ell + \sum_{j \in \mathcal{J}_c} R_{\ell;j}^{\mathsf{T}} W_{\ell;j} A_{\ell;j}^{-1} R_{\ell;j} r_\ell$ ▷ apply local solvers
  end for
  return x_ℓ
end procedure

overlapping subdomains Ω_j and Ω_i, to satisfy Assumption 3.1 it is impossible that $W_j = \mathrm{id}$ as well as $W_i = \mathrm{id}$.

The *transmission conditions*[3] play an important role for restricted Schwarz methods, see (St-Cyr et al., 2007; Efstathiou and Gander, 2003; Gander, 2006). They define the coupling of local problems in overlapping regions of adjacent subdomains. A study of them helps to define a partition of unity at the discrete level.

We start with discussing transmission conditions for the standard H^1-conforming method, which was introduced in Section 3.2. Given the vertex patch Ω_j, we assume that global degrees of freedom are ordered such that the discretization matrix on level ℓ has the form

$$A_\ell = \begin{bmatrix} A_j & B_{j\diamond} \\ B_{\diamond j} & A_\diamond \end{bmatrix}, \tag{3.53}$$

where the generic subscript $\diamond$ indexes the coefficient space that complements the coefficient space associated with V_j. The complementation is then the coefficient space dual to V_ℓ.

[3] See (Toselli and O. Widlund, 2005, §1) for a definition in the context of domain decomposition methods.

Assuming V_j consists only of functions with support in Ω_j, see Section 2.3.2, the local matrix A_j discretizes the model problem (3.1) on Ω_j with homogeneous Dirichlet boundary. Let us have a closer look at the respective local subspace correction

$$x_\ell + R_j^\mathsf{T} W_j A_j^{-1} R_j r_\ell$$

for the previous iterate x_ℓ and its corresponding residual $r_\ell = A_\ell x_\ell - b_\ell$. We refer to $A_j^{-1} R_j r_\ell$ as local solution update in view of

$$A_j^{-1} R_j r_\ell = R_j x_\ell + A_j^{-1}(R_j b_\ell - B_{j\diamond} x_\diamond), \tag{3.54}$$

where $x_\diamond$ complements local coefficients to x_ℓ following the structure in (3.53),

$$x_\ell = \begin{pmatrix} x_j \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ x_\diamond \end{pmatrix},$$

where x_j is the restriction of x_ℓ according to R_j. The previous local solution $R_j x_\ell$ is updated by Δx_j, which is the solution of the local finite element problem at matrix level

$$A_j \Delta x_j = R_j b_\ell - B_{j\diamond} x_\diamond. \tag{3.55}$$

Following Efstathiou and Gander (2003), strong Dirichlet transmission conditions in discrete form enter through $B_{j\diamond} x_\diamond$. Boundary conditions on $\partial\Omega_j \cap \partial\Omega$ were lifted already at the global level; thus, they implicitly enter the equation through $R_j b_\ell$. By *global* we refer here to the continuous and/or discrete problem on mesh level ℓ. The boundary conditions and transmission conditions for local problems are both implicitly enforced by the residual. For strong Dirichlet transmission conditions, only the degrees of freedom associated with $\partial\Omega_j$ are affected by adjacent cells; thus, a very localized finite element stencil is obtained, which was elaborated in Section 3.2.1.

For the symmetric interior penalty method, the local problems have in the absence of strong transmission conditions normal fluxes at interfaces that arise from penalty terms. Assuming standard tensor product elements, the fluxes couple to all degrees of freedom of adjacent cells, which leads to a broader finite element stencil (see Figure 3.1) than for conforming finite elements. We start again at the algebraic level with the specific block structure (3.53). First, let e be a facet at the subdomain's boundary $\partial\Omega_j$ that is not at the physical boundary $\partial\Omega$. The outward-pointing normal at $\partial\Omega_j$ is denoted $\boldsymbol{n}_j$. We introduce $\boldsymbol{n}_\diamond = -\boldsymbol{n}_j$. Let $u_\ell \in V_\ell$ be the finite element function associated with the coefficient vector x_ℓ. Similarly, the complementing function $u_\diamond = u_\ell|_{\Omega\setminus\Omega_j}$ is defined by duality to $x_\diamond$. The transmission condition at interface e, enforced by $(-B_{j\diamond} x_\diamond)$ in (3.55) at the discrete level, is

refactored at the continuous level into

$$\begin{aligned} &-\int_e \left(\gamma_e\, u_\diamond \boldsymbol{n}_\diamond \cdot v\boldsymbol{n}_j - \frac{1}{2}\boldsymbol{\nabla} u_\diamond \cdot v\boldsymbol{n}_j - \frac{1}{2} u_\diamond \boldsymbol{n}_\diamond \cdot \boldsymbol{\nabla} v\right) \mathrm{d}\sigma(\boldsymbol{x}) \\ &= \frac{1}{2}\left[\int_e \left(2\gamma_e\, u_\diamond \cdot v - u_\diamond \boldsymbol{\nabla} v \cdot \boldsymbol{n}_j\right) \mathrm{d}\sigma(\boldsymbol{x})\right] + \frac{1}{2}\left[\int_e \boldsymbol{\nabla} u_\diamond \cdot \boldsymbol{n}_j v \,\mathrm{d}\sigma(\boldsymbol{x})\right] \end{aligned} \tag{3.56}$$

for any $v \in V_j$; derived from the penalty, the consistency, and the adjoint consistency terms in (3.7). The penalty factor γ_e at interior facets is scaled by $1/2$ compared to the identical penalty at a physical boundary $\partial\Omega$; thus, $2\gamma_e$ is the natural penalty when defining the SIPG method on subdomain Ω_j, see the discussion following (3.8). The first bracket in (3.56) defines the typical boundary conditions for the SIPG method (see (3.10), replacing the boundary data g by $u_\diamond$), weakly enforcing Dirichlet conditions. Here, they weakly impose Dirichlet transmission conditions weighted by $1/2$. The second bracket imposes Neumann-type transmission conditions with prescribed normal flux $\boldsymbol{\nabla} u_\diamond \cdot \boldsymbol{n}_j$, also weighted by $1/2$. For facets e at the physical boundary, the respective boundary conditions enter through $R_j b_\ell$ the local discrete problem (3.55), where b_ℓ is defined by duality to the right-hand side operator F_ℓ of the SIPG method in (3.10). Consequently, we have weak Dirichlet conditions at the physical boundary and equally weighted Dirichlet and Neumann transmission conditions. Schwarz smoothers with weak Dirichlet transmission conditions at the boundary $\partial\Omega_j$ were studied by Antonietti and Ayuso (2007, 2008), avoiding equally weighted Dirichlet and Neumann conditions. However, local solvers A_j^{-1} in (3.54) are then inexact.

3.3.1 Mathematical Efficiency of Restricted Additive Schwarz Smoothers

Next, we study two variants of weighting matrices W_j that satisfy the partition of unity assumption. Finding a nonoverlapping domain decomposition at the algebraic level is the first option. To this end, each degree of freedom is associated with one and only one vertex patch Ω_j such that Assumption 3.1 is satisfied. We refer to this variant as *boolean* restricted additive Schwarz method (bRAS) since all entries in W_j are either 0 or 1. In literature, this method is referred to as the original restrictive Schwarz method (Cai and Sarkis, 1999). Note that we postpone discussing the algorithm which determines a nonoverlapping domain decomposition from an original decomposition of overlapping vertex patches to Section A.1.

The second variant to obtain a partition of unity at the algebraic level is counting the number of subdomains each degree of freedom corresponds to and then using the reciprocal count as the weight for each matrix W_j. We refer to the method with this kind of weighting matrix as the *weighted* restricted additive Schwarz (wRAS). Both variants were suggested in the original work (Cai and Sarkis, 1999) but only the former was studied in numerical experiments.

The numerical setup of comparing the solver's fractional convergence steps is almost identical to previous sections, particularly the multilevel SIPG method in Section 3.1 and the

Table 3.7 Fractional iterations ν_{frac} for (standard) additive vertex patch smoother (**AVS**), for either H^1-conforming or discontinuous Lagrange elements using $\mathbb{Q}_k$. **GMRES** solver with relative accuracy 10^{-8} preconditioned by multigrid. Entries "—" not computed due to restricting experiments to $10^5 - 10^8$ unknowns on level L.

Level L	Convergence steps ν_{frac}							
	H^1-conforming				SIPG			
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=2$	$k=3$	$k=4$	$k=5$
6	—	—	—	9.5	—	—	16.9	18.3
7	—	9.6	9.7	9.3	15.6	16.3	16.8	17.6
8	10.2	9.5	9.5	9.2	15.7	16.3	16.7	17.4
9	9.9	9.3	9.4	9.0	15.9	16.0	16.9	17.2
10	9.8	9.2	9.3	8.9	15.7	16.0	16.5	16.7
11	9.7	9.1	8.9	—	15.5	15.9	—	—
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=2$	$k=3$	$k=4$	$k=5$
3	—	—	—	—	—	—	—	26.5
4	—	14.2	14.4	13.3	25.3	27.7	29.4	31.2
5	16.9	13.7	13.9	12.9	26.1	27.9	31.1	32.7
6	16.7	13.5	13.7	12.6	26.2	27.7	31.1	33.6
7	16.5	12.9	—	—	25.5	—	—	—

multilevel H^1-conforming method in Section 3.2. In general, the restricted additive Schwarz smoothing step (Algorithm 5) is intrinsically non-symmetric due to applying weighting matrices only before prolongation and not after restriction. To this end, a preconditioned GMRES solver is utilized for acceleration instead of a conjugate gradient method. For reasons of comparability, we compute fractional convergence steps for standard additive Schwarz smoothers on vertex patches (AVS) again, but this time utilizing a GMRES solver, see Table 3.7. AVS is relaxed by the factor 0.25 in two spatial dimensions and by 0.125 in three, as elaborated in previous sections. Compared to the convergence steps for the CG solver in Tables 3.3 and 3.5, the absolute numbers do not differ significantly. The dependency of the solver's convergence on the mesh size h_L and the polynomial degree k is as expected identical for both, CG and GMRES. For conforming finite elements, the solver's convergence is uniform with respect to both h_L and k. On the contrary, for the SIPG method, convergence steps slightly increase with an increasing polynomial degree for both dimensions and slightly grow with respect to the mesh size in three dimensions. Furthermore, the number of iterations is again higher for the SIPG method compared to the conforming finite element method: for $k = 5$, the fractional number of steps ν_{frac} is about 85% higher in two dimensions and even 160% higher in three dimensions.

Let us continue with the restricted additive Schwarz smoothers (RAVS) on vertex patches, first for H^1-conforming finite elements. Compared to the convergence steps in Table 3.7, the

Table 3.8 Fractional iterations ν_{frac} for both restricted additive vertex patch smoothers, the *weighted* as well as *boolean* **RAVS**, for H^1**-conforming** Lagrange elements. GMRES solver with relative accuracy 10^{-8} preconditioned by multigrid. Entries "—" not computed due to restricting experiments to $10^5 - 10^8$ unknowns on level L.

Level L	Convergence steps ν_{frac}							
	boolean RAVS				weighted RAVS			
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=2$	$k=3$	$k=4$	$k=5$
6	—	—	—	3.9	—	—	—	5.2
7	—	4.6	3.8	3.9	—	4.8	5.0	5.2
8	5.2	4.5	3.7	3.9	4.6	4.8	5.0	5.1
9	5.2	4.5	3.7	3.8	4.5	4.7	4.9	5.1
10	4.9	4.4	3.6	3.7	4.5	4.7	4.9	5.0
11	4.9	4.3	3.6	—	4.5	4.6	4.8	—
12	4.8	—	—	—	4.4	—	—	—
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=2$	$k=3$	$k=4$	$k=5$
3	—	—	—	—	—	—	—	—
4	—	5.1	4.5	4.2	—	5.3	5.1	5.0
5	6.2	4.9	4.5	4.2	5.9	5.2	4.8	4.9
6	5.9	4.8	4.4	3.9	5.8	5.2	4.7	4.8
7	5.8	4.8	—	—	5.7	5.1	—	—

numbers in Table 3.8 for each RAVS variant are significantly lower, with slight advantages for the boolean RAVS. For $k = 5$ and boolean RAVS, the fractional iteration steps are 55% less in 2D than with the standard AVS, even 70% less in 3D. With the weighted RAVS, we normally need one step more than with the boolean RAVS. Nevertheless, with each variant the GMRES solver converges uniformly concerning both the mesh size and the polynomial degree. It is noteworthy that there is almost no difference in iteration counts for two and three spatial dimensions. Recalling the results of the multiplicative vertex patch smoother from Table 3.6 (there, using a CG solver), boolean RAVS and MVS compare at similar levels. However, RAVS requires one instead of eight residual computations per smoothing step in three dimensions, one instead of four computations in two dimensions. The additional cost of applying weighting matrices W_j should be negligible.

Table 3.9 shows the behavior using restricted additive Schwarz smoothers for the SIPG method. Comparing both RAVS variants against each other, convergence steps differ only by one or two for quadratic polynomials in two or three dimensions, respectively, with the boolean RAVS having the edge over the weighted RAVS. However, the gap in steps continues to grow for higher polynomial degrees. Recalling the results of the standard AVS for the SIPG method from Table 3.7, RAVS is superior because the GMRES solver converges uniformly with respect to the mesh size and the polynomial degree, needing only a few steps in total.

Table 3.9 Fractional iterations ν_{frac} for both restricted additive vertex patch smoothers, the *weighted* as well as *boolean* **RAVS**, for **SIPG** discretizations using discontinuous $\mathbb{Q}_k$-elements. GMRES solver with relative accuracy 10^{-8} preconditioned by multigrid. Entries "—" not computed due to restricting experiments to $10^5 - 10^8$ unknowns on level L.

Level L	Convergence steps ν_{frac}							
	boolean RAVS				weighted RAVS			
2D	$k=2$	$k=3$	$k=4$	$k=5$	$k=2$	$k=3$	$k=4$	$k=5$
6	—	—	4.2	4.1	—	—	7.1	7.5
7	5.4	4.6	4.2	3.9	6.2	6.8	6.9	7.3
8	5.3	4.6	4.1	3.9	5.9	6.6	6.7	7.2
9	5.2	4.5	4.1	3.8	5.7	6.5	6.6	6.9
10	5.2	4.5	3.9	3.8	5.6	6.3	6.4	6.8
11	5.1	4.4	—	—	5.5	6.2	—	—
3D	$k=2$	$k=3$	$k=4$	$k=5$	$k=2$	$k=3$	$k=4$	$k=5$
3	—	—	—	3.9	—	—	—	9.2
4	5.2	4.0	4.1	3.9	7.1	8.2	8.2	9.2
5	5.1	3.9	4.1	3.9	6.9	7.8	7.8	8.7
6	5.1	3.9	4.0	3.8	6.7	7.6	7.6	8.5
7	5.0	—	—	—	6.5	—	—	—

For $k = 5$ and boolean RAVS, only $1/5$ of the fractional steps ν_{frac} are needed compared to the standard AVS in two dimensions, only $1/8$ of the steps in three dimensions. Comparing to the numbers in Table 3.4, only one convergence step more is needed using boolean RAVS instead of MVS. The former requires only a single residual computation per smoothing step, instead of the eight computations of MVS in two dimensions and even 16 in three dimensions.

Stiller (2016) studied restricted additive Schwarz smoothers on element-centered subdomains with minimal overlap and face patches (i.e., the union of cells with joint face) studied for the SIPG method. Nonuniformly weighting functions were used to impose a partition of unity. We can not compare numerical results directly because Stiller (2016) made use of polynomial multigrid while we are utilizing geometric multigrid. However, comparing restricted and standard additive Schwarz smoothers in each work the same tendencies are observed, although different multilevel methods and partion of unities are used.

To conclude, we introduced restricted additive Schwarz smoothers that lead to robust solvers, needing only a few iterations to converge. Thus, avoiding relaxation by a partition of unity improves the solver's convergence significantly. Given the lower computing costs in theory, the boolean RAVS may lead to faster numerical solvers than MVS in practice. However, CG solvers do not consume as much memory as GMRES methods and require less communication of data. We would need to compare time-to-solution to decide on the best computational efficiency. However, implementing RAVS given memory-distributed parallelism

was out of the thesis' scope; thus, we do not consider RAVS for the performance analysis in Section 3.4.

Outlook on Optimized Schwarz Methods

In particular, the significant improvement for the SIPG method using restricted additive Schwarz smoothers instead of standard smoothers with relaxation surprised us. Assuming non-standard transmission conditions (for instance, conditions involving partial derivatives effectively), there exists the theory on *optimized Schwarz methods*. Replacing standard Dirichlet transmission conditions with more general Robin-type conditions in (St-Cyr et al., 2007), improved the solver's convergence significantly for a positive definite Helmholtz problem. The interested reader can also find therein references to optimized Schwarz methods for the indefinite Helmholtz problem and the convection-diffusion problem. We have seen before that the inferred transmission conditions for the SIPG method are also not the standard Dirichlet conditions imposed at the physical boundary: the transmission conditions are of equally weighted Dirichlet-type (imposed weakly) and Neumann-type. On the one hand, we may have obtained an optimized Schwarz method.

On the other hand, we recall the discussion on local finite element stencils, that determine the size of the overlap or at least the part of the overlap where we need to transmit values from neighbors. Because of the theory in (St-Cyr et al., 2007) this overlap has to be "small enough". It is questionable that we satisfy this minimum overlap assumption using standard Lagrange elements based on $\mathbb{Q}_k$-polynomials. Consequently, elaborating on this topic may be beneficial for a deeper understanding but was also beyond the scope of this work.

3.4 Performance of Tensor Product Schwarz Smoothers

Before showing computational efficiency in terms of time-to-solution, and analyzing strong and weak scaling of tensor product Schwarz smoother implementations, we quantify the arithmetic complexity of their one-time setup and application. In theory, the key technique is fast diagonalization (see Section 2.1.3) significantly reducing the memory intensity as well as arithmetic operations needed to invert and apply local matrices, compared to standard implementations of Schwarz smoothers. By *standard* we refer to inverting and applying local matrices without exploiting tensor structure. We will show that tensor product Schwarz smoothers (if well implemented) keep up with the efficient operator application of A_ℓ (see Section 2.2) in terms of both computational complexity and memory footprint. In particular, for high-order finite elements, standard implementations of Schwarz smoothers on vertex patches are prohibitively expensive.

3.4.1 Analysis of Computational Complexity

High-performance iterative solvers for large-scale finite element problems rely on nearest-neighbor communication based on the domain decomposition into mesh cells. The matrix-free operators communicate at element level (looping over elements), while subspace corrections inherent in Schwarz smoothers communicate at subdomain level. Only very small subdomains (at most a vertex patch) such that both communications are comparable. Subspace corrections with a high "sequential" efficiency are essential in achieving optimal node-level performance. Therefore, counting floating-point operations is a sound hardware-independent metric to measure the computational efficiency of our implementations, despite the ever-growing complexity of modern multi-core architectures.

We assume the same numerical setting on the unit cube $\Omega = [0,1]^3$ as in previous sections. Here with the most-refined level $L = 3$, such that the mesh $\mathcal{T}_L$ consists of 4096 cells. The number of vertex patches J_L equals 3375. At the heart of the efficient operator application is the sum factorization (2.20b), which requires $\mathcal{O}(D(n_{dof})^{D+1})$ arithmetic operations for each cell, elaborated in Section 2.2 for the model problem (3.1). In our case $n_{dof} = k+1$, thus, we simply use the polynomial degree k as the characteristic size when discussing the arithmetic complexity. In view of Remark 2.1.14, the fast diagonalization of the local matrix A_j has a one-time cost of $\mathcal{O}(Dk^3)$ operations. The corresponding matrix-vector product $A_j^{-1}x_j$ requires $\mathcal{O}(Dk^{D+1})$ arithmetic operations, again benefitting from a sum factorization. A single additive smoothing step $S_{ad}(x_L, b_L)$, see Algorithm 3, consists of two parts, the residual update $b_L - A_L x_L$ and a sum of subspace corrections, that includes applying inverses A_j^{-1} to local coefficients for each subdomain.

The total number[4] of floating-point operations (FLOP) is counted through the performance monitoring tool `likwid-perfctr` (Treibig et al., 2010) on an Intel Xeon Gold 6142M processor. Normalizing the number of floating-point operations N_{FLOP} by a respective number of subunits N_{unit} and an expected arithmetic complexity k^{order}, the leading factor of the (asymptotic) workload per subunit is obtained by

$$C_{load} = \frac{N_{FLOP}}{N_{unit} \times k^{\text{order}}}. \tag{3.57}$$

For the additive vertex patch smoother (AVS), the respective subunit is a vertex patch, $N_{unit} = J_L$. For additive cell-based smoothers (ACS) and computations of the residual, respectively, the subunit is a cell, $N_{unit} = N_{cell}$.

Table 3.10 shows that the leading factor C_{load} levels off with increasing polynomial degree, which confirms the asymptotic computational complexity (*last column*) we expect. The slight decrease of C_{load} for high-order polynomials is explained by a closer look at the SIPG

[4] We accumulate the retired floating-point instruction counters (`FP_ARITH_INST_RETIRED_*_DOUBLE` in `likwid` notation with '*' classifying scalars and different vector lengths), where vector instructions are weighted with their respective vector length.

Table 3.10 The (asymptotic) **workload** per cell (residual and ACS) or per vertex patch (AVS) of additive Schwarz smoothers in three dimensions given by the leading factor C_{load}, see (3.57). The complexity of the one-time setup is independent of the dimension, dominated by D one-dimensional eigenvalue solvers. The factor C_{load} of applying the local inverse is independent in k because inverses are assembled once. Adapted from (Witte et al., 2021).

Routine	Workload C_{load}							Complexity
Degree k:	7	11	15	19	23	27	31	
$b_L - A_L x_L$	32	25	21	19	18	18	17	$\times k^{D+1}$
ACS: $S_{ad}(x_L, b_L)$	45	38	34	32	31	30	29	$\times k^{D+1}$
setup of S_{ad}	96	77	67	63	57	54	52	$\times k^3$
$A_{L;j}^{-1} R_j x_L$	12	12	12	12	12	12	12	$\times k^{D+1}$
AVS: $S_{ad}(x_L, b_L)$	236	227	222	219	218	217	216	$\times k^{D+1}$
setup of S_{ad}	506	419	369	393	365	354	341	$\times k^3$
$A_{L;j}^{-1} R_j x_L$	195	195	195	195	195	195	195	$\times k^{D+1}$

form (3.7): the integration cost of lower-order face terms become less prominent when increasing the polynomial degree. We emphasize that the one-time setup of S_{ad}, consisting of integration and fast inversion[5], is independent of the spatial dimension D. Highlighting the paramount importance of fast diagonalization, we compared a standard and tensor product implementation of ACS in terms of FLOP counts in (Witte et al., 2021): for tri-cubic polynomials the setup cost of standard smoothers are more than 4500 times higher, for $\mathbb{Q}_7$ even more than 250000 times higher. Therein, local matrices A_j were inverted through a singular value decomposition[6] for the standard implementation. The absurdly large difference in computational effort reflects the theoretical complexity of $\mathcal{O}(Dk^3)$ for a fast diagonalization and $\mathcal{O}(k^{3D})$ for a standard SVD. Moreover, it was noticed that the setup cost for each tensor product smoother, either ACS or AVS, are lower than the cost of one (four) smoothing steps $S_{ad}(x_L, b_L)$ for ACS (for AVS). Consequently, the one-time setup cost of our tensor product Schwarz smoothers are negligible compared to iteratively solving the PDE problem. In addition, the cost of subspace corrections are also significantly reduced when utilizing sum factorizations to apply A_j^{-1} to local coefficients as detailed in Section 2.1.3. We confirmed this in (Witte et al., 2021) by counting FLOPs: the floating-point operation counts reflected well the theoretical computational complexities of $\mathcal{O}(Dk^{D+1})$ (tensor product) versus $\mathcal{O}(k^{2D})$ (standard). A benefit of tensor product smoothers which might be overlooked is the reduced consumption of memory: only $\mathcal{O}(Dk^2)$ floating-point numbers need to be stored for each inverse A_j, namely the D univariate eigendecompositions, in contrast to $\mathcal{O}(k^{2D})$ floating-point numbers for a standard SVD. The reduction in memory aligns perfectly with matrix-free operator evaluation and is crucial for the feasibility of Schwarz smoothers on vertex patches

[5]The D one-dimensional generalized eigenvalue problems are solved by means of the `LAPACK` routine `DSYGV`.
[6]The singular value decomposition is computed by the `LAPACK` routine `XGESDD`.

for high-order finite elements. Historically, matrix-free methods were primarily used to fit large-scale computations in memory at all (Bergen et al., 2005). Nowadays, on state-of-the-art computing architectures, these methods provide excellent (parallel) performance. A very high computational intensity paired with outstanding numerical throughput is achieved, such that classical finite element implementations utilizing sparse-matrix formats are outperformed for cubic or high-order finite elements (Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019; May et al., 2014).

3.4.2 Computational Efficiency and Parallel Performance

Counting floating-point operations confirmed (theoretical) computational complexities. It is a simple hardware-independent metric gaining first insight about computational efficiency. However, the metric is not sufficient to evaluate parallel performance on large, complex clusters of compute nodes. Our implementations of tensor product Schwarz smoothers make use of different types of parallel programming: SIMD vectorization at instruction level[7] as well as shared-memory parallelism using Intel TBB (Threading Building Blocks) and distributed-memory parallelism utilizing Open MPI (Message Passing Interface implementation) at task level, respectively. We note that our implementation SIMD-vectorizes over subdomains similar to the vectorization strategy of `deal.II`'s matrix-free operators, which vectorize over mesh cells, see (Arndt, Fehn, et al., 2020; Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019). Nevertheless, we do neither analyze in detail the effect of SIMD vectorization nor the effect of processing tasks in parallel in shared memory. We focus on the MPI scaling analysis here.

In most recent years, matrix-free routines in `deal.II` were continuously improved, in particular, driven by the German Exascale project ExaDG. Their excellence performance was studied in (Arndt, Fehn, et al., 2020; Kronbichler and Kormann, 2012; Kronbichler and Kormann, 2019) and references therein. The efficient operator application in `deal.II` has been improved to a point such that respective matrix-vector products were computed so efficiently that they no longer dominated the conjugate gradient solver. See our final project report (Arndt, Fehn, et al., 2020, §3) for more details - for instance, a benchmark problem[8] introduced by the US Exascale initiative "Center for Efficient Exascale Discretization" (CEED) was studied to confirm high performance. In (Fehn, Munch, et al., 2020) and (Clevenger et al., 2020), latest advances of `deal.II`'s geometric multigrid implementations were discussed. The latter publication provided strong scaling experiments for the h-multigrid V-cycle based on uniform mesh refinement.

[7] The instruction set architecture (ISA) extension is user-specified: for instance, using the AVX 256-bit extension on x86 architectures with 64-bit floating point data type leads to processing four floating-point numbers in parallel. Each floating-point number fitting into the extension possesses a so-called *SIMD lane*.

[8] The benchmark problem BP5 (P. Fischer et al., 2020) involves a conforming finite element discretization of the Laplacian operator in 3D, using a CG solver with efficient operator application and diagonal preconditioner.

First, we emphasize that an technical analysis and performance tweaks as found in (Arndt, Fehn, et al., 2020; Fehn, Munch, et al., 2020) exceed the scope of this thesis. Nevertheless, our implementations are far beyond a proof of concept. They are carefully designed with parallel computing in mind but have not reached a maturity level comparable to `deal.II`'s matrix-free framework. Second, we want to be open[9] to the interested reader and new contributors. Consequently, we make use of standard implementations[10] of the conjugate gradient solver, geometric multigrid preconditioner, and matrix-free operator evaluation. Furthermore, we try to discuss our methods on a high level of computing abstraction. We elaborate on prospective optimizations at the end of this section. Concerning our numerical experiments, note that we refer to implementations according to the release version 9.2 of `deal.II` (Arndt, Bangerth, Blais, et al., 2020).

During the studies for this thesis, efficient implementations for tensor product Schwarz smoothers (TPSS) were developed with the following features:

Remark 3.4.1 (`TPSS`). The software package `TPSS` for tensor product Schwarz smoothers provides the following features:

- support for continuous and discontinuous tensor product elements, including (moment-based) Raviart-Thomas elements, see Definition 5.2.5
- additive and multiplicative Schwarz smoothers on cells or vertex patches for uniformly refined meshes (including non-Cartesian cells, see Section 3.1.2 and (Witte et al., 2021))
- restricted additive Schwarz smoothers[11]
- support for shared memory task parallelism and/or MPI implementations, particularly supporting `deal.II`'s vector class `LinearAlgebra::distributed::Vector` that provides linear algebra functionality with distributed storage
- SIMD vectorization across subdomains
- user-defined coloring and graph coloring[11] to avoid data races and/or enable parallelism for multiplicative smoothers
- a tensor product matrix class `TensorProductMatrixSymmetricSum`[12] that enables fast diagonalization (Section 2.1.3) and a matrix-vector product using sum factorization (Section 2.1.2), respectively
- a Kronecker product singular value decomposition (Section 2.1.4) supporting SIMD vectorization

[9] https://github.com/jwitte08/TPSS
[10] See `deal.II`'s tutorial program *step-37* for details.
[11] Not (fully) compatible with MPI implementations

- an (approximative) Gaussian block elimination following the ideas in (Arndt, Fehn, et al., 2020, §5.3)

In analogy to `deal.II`'s matrix-free framework, we use flat data arrays with contiguous memory layout following the lexicographic ordering from Definition 2.1.6. It enables fast and direct access to data as well as facilitates SIMD vectorization. At the very heart of our implementation of tensor product Schwarz smoothers is the matrix class `TensorProductMatrixSymmetricSum`[12]. Given a matrix with separable Kronecker representation (2.21), the matrix is inverted by fast diagonalization: the generalized eigenvalues and eigenvectors are computed by the `LAPACK` function `xSYGVX`. The matrix and its inverse are not stored explicitly but only in terms of univariate matrices, univariate generalized eigenvectors and eigenvalues, see (2.21) and (2.25), respectively. Consequently, it is not possible to access a single entry of the matrix or its inverse. However, it is only the matrix-vector product regarding the inverse matrix that is needed for subspace corrections (the same applies to the matrix-vector product with the matrix itself), efficiently computed via sum factorization. From a technical perspective, the d-mode products involved in the sum factorization (2.20) are computed by the highly optimized kernel `EvaluatorTensorProduct`. The same kernel that is used to compute sum factorizations for `deal.II`'s matrix-free operator evaluation. Thus, algorithmic components on all levels, numerical smoothers, preconditioners, and solvers, profit when improving this kernel.

Computational resources. All computations were conducted on the bwUniCluster 2.0[13]. Each compute node consists of a 2×20 core Intel Xeon Gold 6230 Cascade Lake and has main memory with 96 GB RAM. Intel processors are operated at 2.1 GHz and run with two-way hyperthreading[14], that makes 80 threads per node. We use the AVX-512 FMA units for SIMD vectorization, which means 8 SIMD lanes with 64-bit floating-point numbers (the default double-precision on the x86 architecture). The C++ implementation is compiled with OpenMPI 4.1 and GCC 10.2, using optimization level 3 with loop unrolling enabled. We assign an MPI rank to each physical core. The main memory of each compute node is evenly distributed among the 40 MPI ranks.

Time measurement. To avoid noisy wall time measurements of algorithmic components that require not more than 10^{-2} seconds, we run each component in a loop. Then, the whole loop's wall time is recorded and divided afterwards by the number of iterations. The number chosen guarantees that a single execution of the loop takes at least 0.1 seconds. The wall-clock time is measured independently for each MPI rank and communicated after executing the algorithmic component in a loop. We take the maximum wall time among all MPI ranks as runtime into account. Furthermore, each component-in-a-loop measurement is repeated

[12] There exists a more general class `TensorProductMatrix` (not merged into `deal.II` yet) that extends functionality to more tensor product representations than (2.21), used in Chapters 4 and 5

[13] The author acknowledges support by the state of Baden-Württemberg through bwHPC.

[14] Focusing on the MPI implementation, we disable the usage of (hyper-)threading, although we may use Intel TBB for the assembly of local matrices or applying subspace corrections.

$20 - 50$ times, depending on the algorithm: for instance, measuring the time-to-solution, which means running the preconditioned CG solver until a relative residual reduction of 10^{-8} is reached, was repeated only 20 times. While measurements for the efficient operator application or a single smoothing step were repeated 50 times. The wall-clock time shown in figures and tables is the median of these repetitions.

Multilevel Schwarz methods. For the subsequent numerical experiments, the identical multilevel hierarchy of uniformly refined meshes on the domain $[0, 1]^3$ as in Section 3.1.1 (SIPG) and Section 3.2.1 (H^1-conforming) is used. Note that L denotes the most-refined mesh level. Furthermore, the same optimal coloring and relaxation factors are used. As before, a single pre- and post-smoothing step respectively is applied with an order of subspace corrections that symmetrizes the multigrid V-cycle preconditioner. This enables using a CG method for acceleration.

Strong and weak scaling. We start with results for the SIPG discretization of the Laplacian. In Figure 3.6 and Figure 3.7 runtimes for computing the matrix-free operator evaluation $A_L x_L$, its residual (delineated in blue) and a single tensor product smoothing step (ACS, MCS, or MVS) are shown for polynomial degree $k = 3$ and $k = 7$, respectively. The greyish-dotted lines are references for perfect (strong) scaling. Weak scaling is seen by horizontally matching line markers (that means diamonds, triangles, and others) with a factor of eight in the number of cores.

As expected, the more degrees of freedom are processed per compute node, the better the scaling of algorithmic components. In particular, both cell-based smoothing steps, the additive (ACS) in Algorithm 3 and multiplicative (MCS) in Algorithm 4, have almost identical weak and strong scaling compared to the efficient operator application, for both polynomial degrees given. Recalling both algorithms, the reasoning lies in the nature of cell-based tensor product methods, which loop for the residual update and subspace corrections over cells (or a subset of cells with the same color). The local work is either the integration-based computation and application of $A_{L;j}$, or its inverse: since both rely on sum factorization, both are computed equally efficient, see Sections 2.1.3 and 2.2. The multiplicative smoothing step requires two colors such that the residual is computed twice. Consequently, it is expected that MCS requires more time than ACS. We will elaborate on absolute time measurements below.

For tri-cubic elements in Figure 3.6, the scaling behavior of the vertex patch smoother (MVS) compares at similar levels, with a larger and growing gap to the perfect scaling line when the workload for each CPU decreases. Nevertheless, the scaling does not deteriorate that much, except for discretizations with two million degrees of freedom. The gaps start to become more significant in Figure 3.7 ($k = 7$): the difference between more subdomains with less work ($k = 3$) and fewer subdomains with more work ($k = 7$) for the same amount of unknowns is apparent, but only for MVS. Pointing to scaling results also for very high polynomial degree $k = 15$ in Figure A.3, we see that this trend is continued for elements

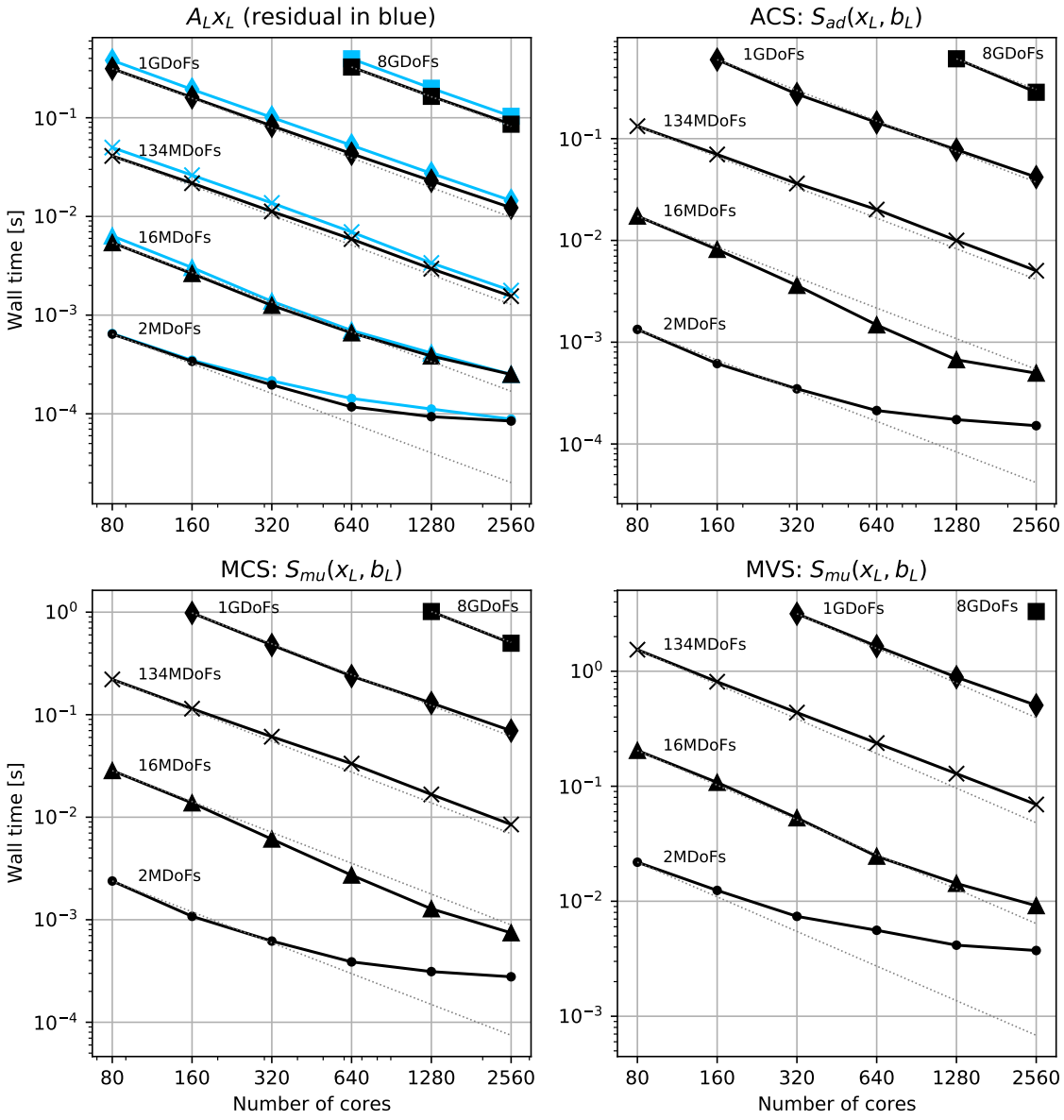


Fig. 3.6 MPI scaling analysis of one smoothing step for ACS (*top-right*), MCS (*bottom-left*), and MVS (*bottom-right*) given **SIPG** discretizations using **tri-cubic** Lagrange elements. Smoothing is compared against the matrix-free operator evaluation (*top-left*) and its residual $b_L - A_L x_L$ delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

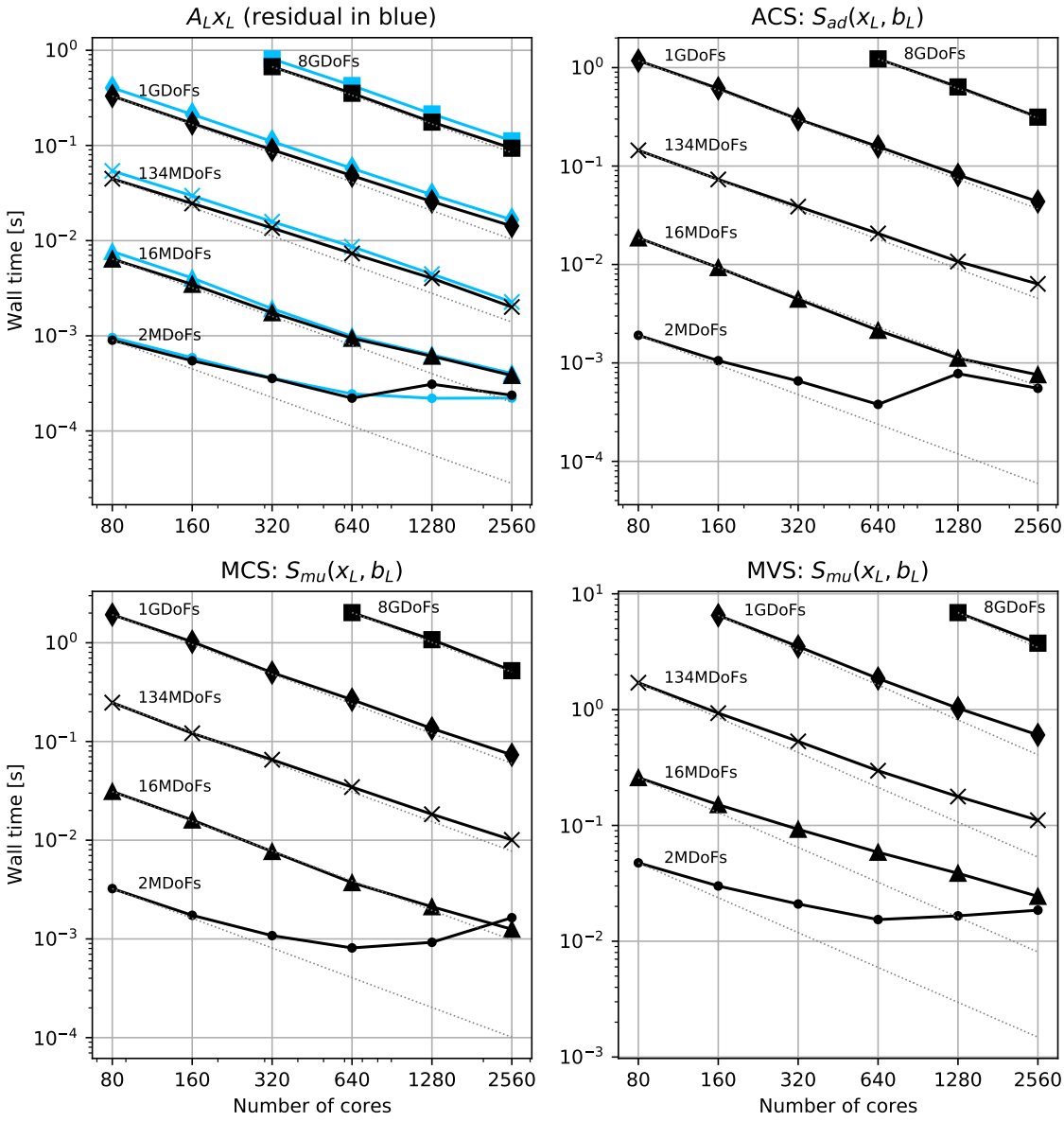


Fig. 3.7 MPI scaling analysis of one smoothing step for ACS (*top-right*), MCS (*bottom-left*), and MVS (*bottom-right*) given **SIPG** discretizations using $\mathbb{Q}_7$**-elements**. Smoothing is compared against the matrix-free operator evaluation (*top-left*) and its residual $b_L - A_L x_L$ delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

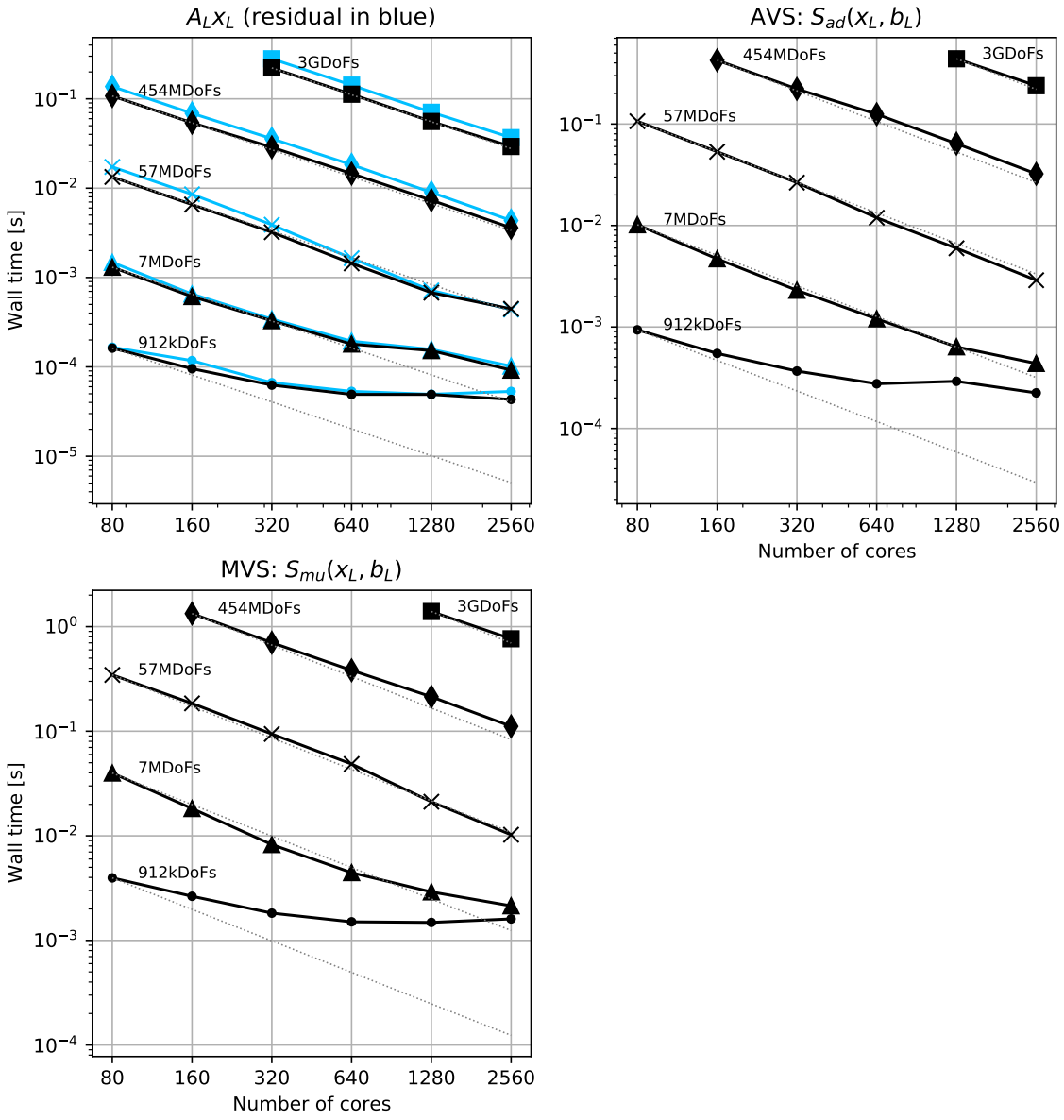


Fig. 3.8 MPI scaling analysis of one smoothing step for AVS (*top-right*) and MVS (*bottom-left*) given H^1-**conforming** discretizations using **tri-cubic** Lagrange elements. Smoothing is compared against the matrix-free operator evaluation (*top-left*) and its residual $b_L - A_L x_L$ delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

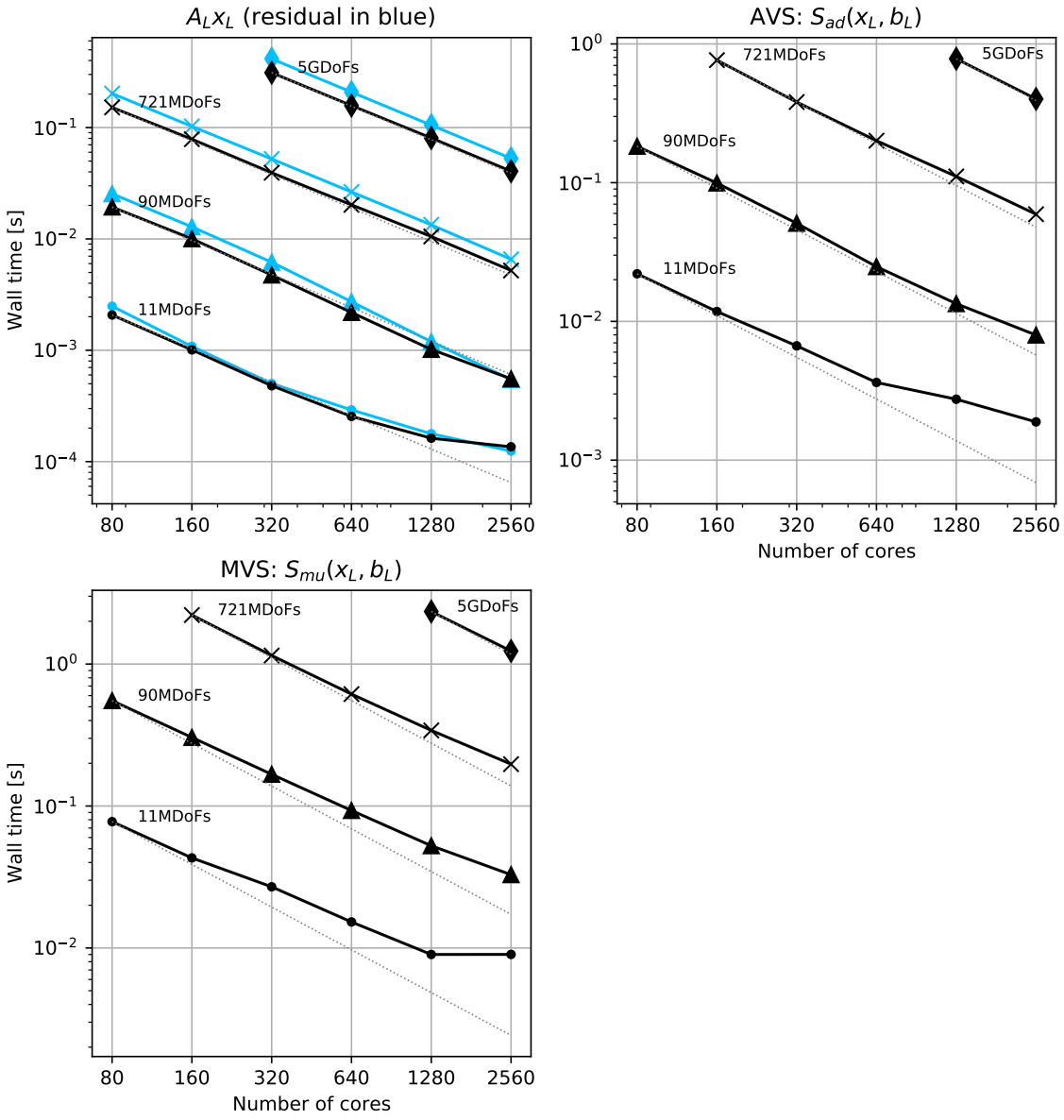


Fig. 3.9 MPI scaling analysis of one smoothing step for AVS (*top-right*) and MVS (*bottom-left*) given H^1**-conforming** discretizations using $\mathbb{Q}_7$**-elements**. Smoothing is compared against the matrix-free operator evaluation (*top-left*) and its residual $b_L - A_L x_L$ delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

with very high-order. For MVS, subspace corrections are computed as a loop over vertex patches: fast diagonalization makes applying inverses very efficient, but each inverse has twice the size in each dimension compared to cell-based smoothers. The increase of workload per subdomain by a factor of 16 was observed before in Table 3.10, and we also observe it here by looking at runtimes. Another effect leading to more runtime is computing 16 times the residual per smoothing step due to a red-black coloring with parqueting, see Figure 3.2b.

What is the reasoning for the quickly growing gap? To this end, we compared runtimes between MPI ranks and found significant variance in measurements without stating any times explicitly. We found some reasons that led to these imbalances.

First, the current way we distribute vertex patches across MPI ranks is as follows.

1. The MPI process owning[15] the most cells of the vertex patch takes ownership
2. If there exist two or more MPI processes owning the maximum number of cells of the subdomain, the process with the lowest rank (an integer in C++) takes ownership

This distribution leads to imbalanced workloads between MPI ranks. In addition, to avoid branching through conditional statements, our implementation groups subdomains either at the physical boundary or in the domain's interior, further aggravating the imbalances. The respective subspace corrections are then processed in two successive loops. Since we SIMD-vectorize over subdomains, having fewer (physical) subdomains than SIMD lanes we use a padding of leftover lanes with dummies. The grouping mentioned before leads to more dummies than necessary. Aggravating imbalances further, we also have this effect of grouping through coloring. Looking at the colored multiplicative smoothing step given by Algorithm 4, each color has only a subset of subspace corrections, basically another group. It explains why, in particular MVS, which requires 16 colors, shows growing gaps to the perfect scaling line for high-order elements.

For discretizations with few subdomains but large CPU workload per subdomain, for instance Figure 3.7 ($k = 7$) or Figure A.3 ($k = 15$), these imbalances result in a loss of scaling capability. A smarter distribution of vertex patches across MPI boundaries is easy to implement. Vectorizing over evaluation points, like quadrature points and/or degrees of freedom, or grouping shape function values and gradients together may leverage imbalances. The latter was implemented with great success by the EXA-DUNE project, see (Bastian, Altenbernd, et al., 2020). Nevertheless, in cases where each core is sufficiently loaded with work, strong and weak scaling can be observed for all smoothing steps presented. Therefore, the scaling results are promising.

For the H^1-conforming multilevel method runtimes for both vertex patch smoothers AVS and MVS are shown in Figures 3.8 and 3.9 for $k = 3$ and $k = 7$, respectively. We

[15] We refer to (Bangerth et al., 2011): algorithms and data structures for meshes stored in distributed memory across many processors are studied, and what is meant by owning and ownership is elaborated.

achieve strong scaling for tri-cubic elements. For $\mathbb{Q}_7$-polynomials, AVS shows the same scaling behavior as the efficient operator application. At the same time, a slightly growing gap is observed for MVS, particularly when the CPU workload is low. In Figure A.4 for $k = 15$, the imbalances discussed before lead to largely growing gaps for MVS and smaller growing gaps for AVS. In general, the vertex patch smoothers for the conforming finite element method scale better than for SIPG discretizations. The coloring explains this observation: MVS for conforming finite elements does need only a parqueting, i.e., eight colors per smoothing step instead of 16 colors for parqueting with red-black coloring. In this way, twice as many local inverses per color are distributed across the same number of MPI ranks which mitigates workload imbalances. The mitigation becomes even more prominent when compared to AVS, where a single color is needed. It explains why AVS scales much better and nearly as good as cell-based smoothers. As motivated in the introduction of this chapter, `deal.II`'s matrix-free operator evaluation achieves excellent performance for state-of-the-art benchmarks from CEED[16]. Therefore, we consider the efficient operator application as a baseline or benchmark for our tensor product Schwarz smoothers. Achieving runtimes close to this baseline speaks in favor of our methods.

The pure wall-clock time of algorithmic components T_{meth} is compared best in numbers. For convenience, see the results in Table 3.11 where we gather runtimes for both finite element methods, limiting ourselves to 1 billion DoFs for SIPG and 454 million or 721 million DoFs for conforming elements with $k = 3$ or $k = 7$, respectively. We use the time to compute the residual T_{res} as a baseline since computing residuals is the integral part of Schwarz smoothers besides subspace corrections. Smoothers are compared in units of the residual computation by displaying T_{meth}/T_{res}. The (standard) matrix-free operator evaluation needs only $75 - 85\%$ of the residual's time. In the last column, we state the number of residuals N_{res} that are computed per smoothing step.

Starting with ACS_{DG}, subspace corrections require twice the time of computing the residual. For MCS_{DG} it is close to three times, indicating an overhead due to coloring. For MVS_{DG}, subspace corrections need $15 - 20$ times T_{res}. On the one hand, applying a local inverse needs around 16 times more arithmetic operations than cell-based smoothers. On the other hand, we have slightly fewer vertex patches than cells. Therefore, MVS_{DG} compares well. MVS for conforming finite elements compares at similar levels as MVS_{DG}, while AVS is even better, needing only $5 - 8$ times the amount of T_{res}. The difference between AVS and MVS indicates an imbalance again due to coloring. Taking into account that our implementations have not reached such a maturity level as `deal.II`'s matrix-free operator evulation and having identified some imbalances, the results are promising.

Next, we have a look at time-to-solution, i.e., the wall-clock time the CG solver needs to reach a relative residual reduction of 10^{-8}. From Sections 3.1.1 and 3.2.1 the open question

[16] https://ceed.exascaleproject.org/

Table 3.11 Analysis of one smoothing step and matrix-free operator evaluation $A_L x_L$ in **units of residual computations**: the measured wall time T_{meth} for each method, i.e., either the matrix-vector product or application of the smoother, is divided by the time T_{res} to compute the respective residual. Smoothers indexed by DG refer to the SIPG discretization in 3D with 1 GDoFs for polynomial degrees $k = 3$ and $k = 7$. Unindexed smoothers refer to the conforming FEM with 454 MDoFs for $k = 3$, 721 MDoFs for $k = 7$. N_{res} states the number of residual computations within a single (colored) smoothing step. Runtime measurements are the same as in Figures 3.6 to 3.9. Entries "—" not computed due to exceeding main memory limits.

Method	T_{meth}/T_{res}						N_{res}
N_{cores}:	80	160	320	640	1280	2560	$\mathbb{Q}_3$
$A_L^{DG} x_L$	0.82	0.81	0.83	0.82	0.83	0.84	—
ACS_{DG}	—	3.08	2.71	2.74	2.84	2.89	1
MCS_{DG}	—	5.06	4.73	4.52	4.74	4.82	2
MVS_{DG}	—	—	31.33	31.41	32.42	34.94	16
$A_L x_L$	0.78	0.78	0.79	0.79	0.81	0.85	—
AVS	—	6.16	6.19	6.82	7.13	7.42	1
MVS	—	19.34	19.63	20.87	23.78	25.62	8
N_{cores}:	80	160	320	640	1280	2560	$\mathbb{Q}_7$
$A_L^{DG} x_L$	0.81	0.80	0.81	0.83	0.84	0.85	—
ACS_{DG}	2.91	2.89	2.71	2.74	2.67	2.62	1
MCS_{DG}	4.77	4.82	4.52	4.62	4.45	4.41	2
MVS_{DG}	—	30.74	31.81	32.35	33.81	36.42	16
$A_L x_L$	0.74	0.76	0.75	0.76	0.79	0.80	—
AVS	—	7.45	7.29	7.59	8.28	9.01	1
MVS	—	21.59	21.97	23.25	25.45	30.02	8

will be answered: *which tensor product Schwarz smoother leads to the numerical solver with the highest computational efficiency*? MPI scaling results are shown in Figure 3.10 for SIPG discretizations, in Figure 3.11 for the conforming FEM. The strong and weak scaling is not perfect but promising: overcoming the imbalances identified before it should scale better. In particular, down to one second of runtime, solvers scale already well. The previous trends continue: the cell-based smoothers almost provide perfect scaling, while the vertex patch smoothers for conforming FEM scale better than MVS for SIPG discretizations.

Besides the time-to-solution also the time to compute the smoothers is delineated in blue. Noticing the discussion on arithmetic complexities in Section 3.4.1, we have identified that mainly the tremendous computational effort of computing local inverses makes patch-based Schwarz smoothers infeasible if computed naïvely. For our tensor product Schwarz smoothers, it completely changes due to fast diagonalization. The assembly of univariate discretization matrices and the one-dimensional generalized eigenvalue problems are solved so fast that only a fraction of the time-to-solution is spent setting up the tensor product Schwarz smoothers. Similar scaling results are shown in Figures A.5 and A.6 for polynomial degree $k = 7$, respectively.

We see in Figures 3.10 and 3.11 that the time-to-solution for different smoothers compares at similar levels. To answer the previous question on the computational efficiency in detail, we consider the *numerical throughput* as metric, i.e., the number of million DoFs processed per second. Thus, the higher the numerical throughput, the better. Algorithms are preferred that increase this metric, even if it may lead to lower hardware performance indicators, traditionally measured in FLOPs per second (arithmetic performance) and/or Gigabytes per second (memory throughput). The reasoning is that algorithms needing many arithmetic operations per DoF may scale well but are not necessarily very efficient. Maximizing the computational intensity is one side of the coin. However, the ultimate goal should be to reduce computational complexity per unknown while keeping the computational intensity sufficiently high.

Comparing number of DoFs per node against the numerical throughput in Figure 3.12, we confirm many conclusions from previous scaling results. The numerical throughput is high given a sufficiently high workload, such as around 10^6 DoFs per node for tri-cubic elements, see Figure 3.12a. For this range of DoFs per node, the numerical throughput starts to level off, particularly for conforming finite elements. The cell-based tensor product smoothers, ACS_{DG} and MCS_{DG} (slightly) outperform the vertex patch smoothers. They have the best tradeoff between computational complexity per smoothing step and the number of CG iterations needed. Furthermore, they do scale best. Nevertheless, by exploiting tensor structure, the vertex patch smoothers also achieve high numerical throughputs, with MVS being the "winner", having only eight residual computations plus subspace corrections and needing only five CG steps to converge. Although AVS does scale better than MVS, it is inferior in numerical throughput.

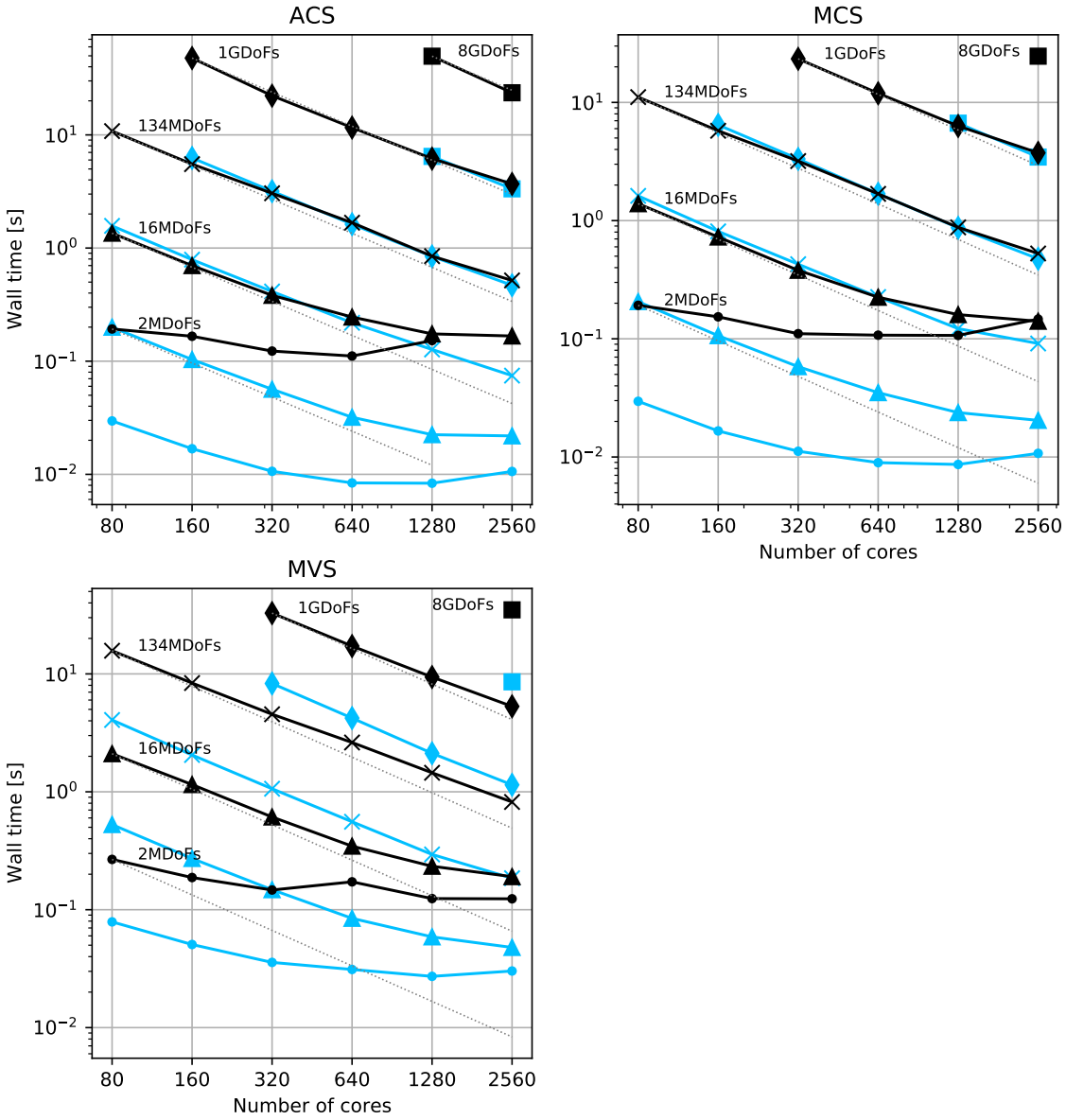


Fig. 3.10 MPI scaling analysis of the numerical solver, i.e., measuring **time-to-solution** (*black lines*), for **SIPG** discretizations using **tri-cubic** elements. The CG solver accelerating the multilevel method with ACS (*top-left*), MCS (*top-right*), or MVS (*bottom-left*), respectively, solves with relative accuracy 10^{-8}. The respective time to compute pre- and post-smoothers before solving is delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

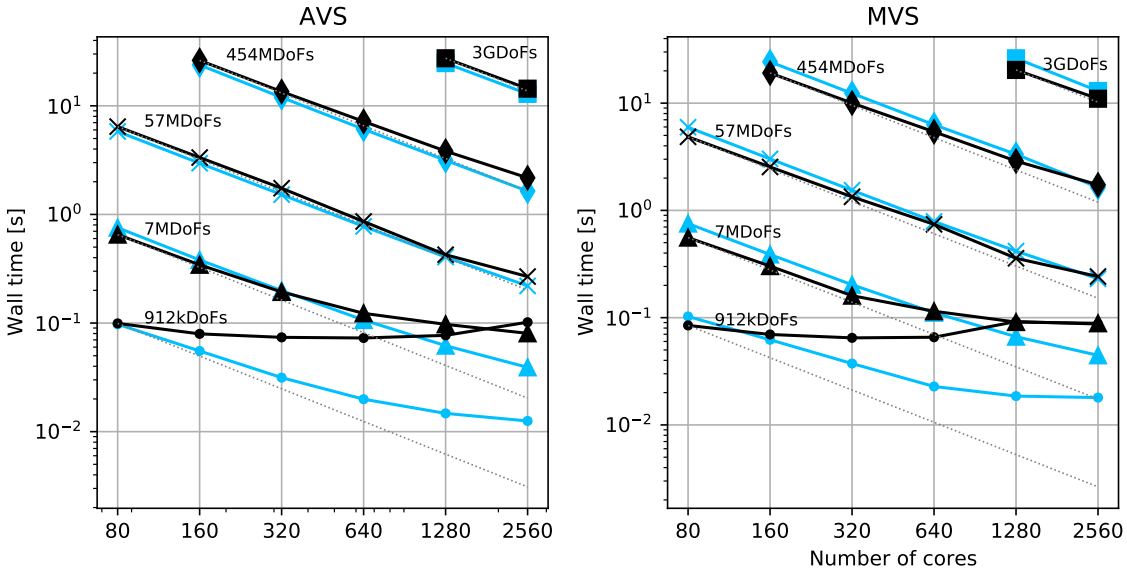
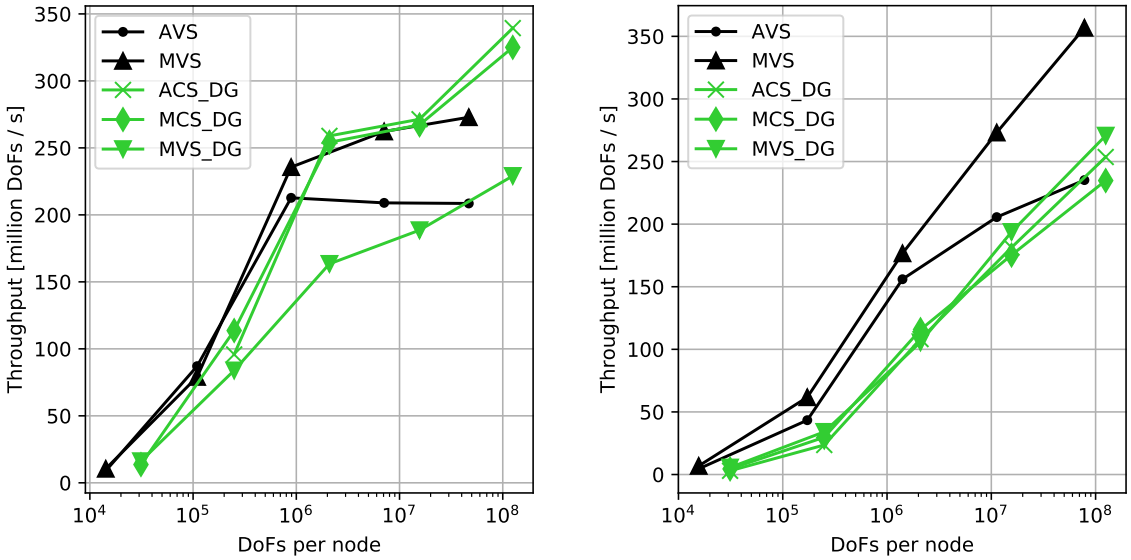


Fig. 3.11 MPI scaling analysis of the numerical solver, i.e., measuring **time-to-solution** (*black lines*), for H^1**-conforming** discretizations using **tri-cubic** elements. The CG solver accelerating the multilevel method with AVS (*top-left*) or MVS (*top-right*), respectively, solves with relative accuracy 10^{-8}. The respective time to compute pre- and post-smoothers before solving is delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

Table 3.12 The underlying number of CG iterations according to the time-to-solution in Figure 3.12 (there shown in terms of numerical throughput). Any smoother leads to solvers with uniform convergence independent of the mesh size; thus, the iteration count is the same for varying DoFs per node.

CG iterations									
AVS		MVS		MVS_{DG}		MCS_{DG}		ACS_{DG}	
$k=3$	$k=7$	$k=3$	$k=7$	$k=3$	$k=7$	$k=3$	$k=7$	$k=3$	$k=7$
16	15	5	4	4	3	13	17	18	23



(a) for $\mathbb{Q}_3$-elements

(b) for $\mathbb{Q}_7$-elements

Fig. 3.12 Analysis of the **computational efficiency** for tensor product Schwarz smoothers in terms of numerical throughput, i.e., measuring million DoFs per second, for SIPG discretizations (*with suffix* _DG) and the conforming FEM (*without suffix*). Underlying times are the times-to-solution from Figures 3.10 and 3.11 using 2560 MPI cores (= 64 compute nodes with 40 cores each).

Switching to PDE solvers using $\mathbb{Q}_7$-polynomials and looking at the evolution of CG iteration steps from Table 3.12, Figure 3.12b shows that vertex patch smoothers become superior to cell-based smoothers. Although cell-based smoothers scale by far the best and as good as for tri-cubic elements, they suffer from requiring more and more iteration steps to solve for high-order elements. In contrast, both multiplicative vertex patch smoothers significantly improve their throughput given sufficient workload per compute node. Sufficient DoFs per node mitigate their technical shortcomings. Both profit from requiring one CG iteration less, which translates to 20% and 25% of total steps less for MVS and MVS_{DG}, respectively. AVS improves only slightly due to requiring 6% of total steps less. MVS has the edge over MVS_{DG} due to needing half the colors while its CG solver needs only a single convergence step more. Compared to PDE solvers in (Arndt, Fehn, et al., 2020), we achieve good numerical throughput, albeit the known shortcomings of our implementation. For high-order elements, the superior mathematical efficiency of multiplicative vertex patch smoothers seems to be a game-changer.

3.5 Conclusion

Let us conclude the performance analysis and thereby this chapter as we have finally addressed all questions raised. For low-order finite elements, the Poisson problem (3.1) leads to a linear system of equations that can be tackled by simple multigrid smoothers, for instance, cell-based Schwarz methods or a Chebyshev smoother[17] with inner pointwise preconditioner, see (Arndt, Fehn, et al., 2020; Fehn, Munch, et al., 2020; Kronbichler, Kormann, Fehn, et al., 2019). With increasing polynomial degree, the outer solver's iteration steps do not deteriorate for these kinds of smoothers significantly. Therefore, if implemented well it is very challenging to "beat" them in numerical throughput. However, we have shown that already for $k = 7$ using tensor product smoothers on vertex patches led to higher throughputs than using cell-based smoothers if the workload was sufficiently balanced between MPI ranks. Recalling the arithmetic complexities from Section 3.4.1, Schwarz smoothers that invert local matrices by standard direct methods and store them in standard matrix formats completely dominate numerical solvers in runtime due to tremendous one-time setup costs and an infeasibly high memory intensity, particularly for high-order finite elements. Exploiting tensor structure overcomes both problems. Storing only what is needed for the Kronecker product presentation of local matrices and their inverses as well as inverting and applying inverses using fast diagonalization and sum factorization, respectively, Schwarz smoothers on vertex patches for high-order finite elements become not only feasible. Their computational efficiency improves to the point where subspace corrections compare similarly to state-of-the-art matrix-free operator evaluation used to compute residuals.

[17] See (Varga, 2009).

Furthermore, using coloring enables parallel computing of multiplicative Schwarz smoothers at all. If we fix the identified shortcomings of our implementation, we expect to improve towards ideal scaling, and outstanding numerical throughput for fewer workloads per compute node. For sufficiently many DoFs per node, we have already observed good (parallel) performance and computational efficiency. Tackling the load imbalances between MPI ranks exceeded the scope of this thesis, albeit some of them may be easily improved.

For the monograph's second half, we focus on developing tensor product Schwarz smoothers on vertex patches for more (challenging) finite element problems. First, for the biharmonic problem in Chapter 4, i.e., methods have to be extended to second-order derivatives. Second, for the Stokes problem in Chapter 5 (also closely related to linear elasticity in the limit of (nearly) incompressible material), we extend our concepts to a vector-valued partial differential equation. For both, cost-efficient numerical solvers rely on well-designed Schwarz smoothers on vertex patches. Simple pointwise or nonoverlapping block smoothers do not suffice from a mathematical perspective. We already note that we will not perform any performance analysis in the following chapters. From a technical perspective, we require only minor adjustments. In this regard, what has been observed here should be almost identical there. Therefore, our tensor-based implementations for the Laplace problem are prototypical for cost-efficient numerical solvers in subsequent chapters.

Chapter 4

BIHARMONIC PROBLEM

In this chapter, we discuss a method for the model problem of the biharmonic equation

$$\begin{aligned} -\Delta^2 u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega, \\ \boldsymbol{\nabla} u \cdot \boldsymbol{n} &= j \quad \text{on } \partial\Omega, \end{aligned} \tag{4.1}$$

where Ω is a polygonal domain in $\mathbb{R}^D$ with $D = 2$. The numerical results shown in this chapter are limited to two spatial dimensions. However, we will point out which of our tensor product Schwarz methods are (easily) extensible to dimensions higher than two and use the generic number of dimensions D when needed.

Several applications from thin bending theory, e.g., bending Poisson-Kirchhoff plates or Bernoulli-Euler beams, rely on the biharmonic equation and the strain gradient theory of Toupin-Mindlin as surveyed by (Engel et al., 2002) and references therein. By "thin", we mean that no transverse shear deformation exists. For more extensive studies, we refer the reader to textbooks on structural mechanics, for example (P. G. Ciarlet, 1988, 1997, 2000), or in the context of finite elements to (Braess, 2013; Hughes, 2012). The volumetric load f and boundary functions g and j are given, where their regularity is defined when needed. In the case of modeling thin structures, boundary conditions as specified in (4.1) are referred to as *clamped* boundary conditions: the thin structure, determined by the displacement u, is clamped at a certain height g with a specific angle j against the horizontal. For the remainder of this chapter, we refer to the boundary conditions as "clamped", albeit not committing ourselves to thin structure applications.

In the context of computational structural mechanics, many finite element methods have been developed over the past decades for the biharmonic equation (4.1). The focal point of many methods was to answer the question: how do we impose the essential C^1 inter-element continuity? In the early days, straightforward answers were classical C^1 elements like the Bogner-Fox-Schmit element (Bogner et al., 1965), elements of the TUBA family, (Argyris

et al., 1968) or the macro-element approach (J. J. Douglas et al., 1979). For C^1-continuous elements or, shortly, C^1-elements, both the function value and its first-order derivatives are continuous across cell interfaces. These methods have in common that they are very challenging to implement, particularly at the time they were developed.

Then, researchers began focusing on mixed formulations by introducing rotations as an auxiliary independent field beside the primary displacement field. Independent C^0-interpolations for displacement and rotation were sufficient to avoid common challenges of C^1-elements. The benefits gained came at the cost of additional unknowns, sophisticated formulations, and shear-locking difficulties. The latter were tackled by inf-sup considerations (Babuška, 1971; Brezzi, 1974), the reduced-integration approach (Malkus and Hughes, 1978) or nonconforming elements (Arnold and Falk, 1989). Nowadays, these methods are based on a rich mathematical foundation, and the interested reader may find more details in textbooks (Boffi et al., 2013; Brezzi and Fortin, 1991). Besides, nonconforming rotation-free methods have been developed that ignore the C^1-continuity requirements at all (Oñate and Cervera, 1993; Oñate and Zárate, 2000; Phaal and Calladine, 1992a,b), or impose them weakly by penalization (Babuška and Zlámal, 1973).

Finally, arising out of the great success of discontinuous Galerkin methods the C^0 *interior penalty method* (C^0IP) by (Brenner, 2011; Brenner and Sung, 2005) has been developed, originating from (Engel et al., 2002). In the original work, the method was referred to as the continuous/discontinuous Galerkin method. It is rotation-free and nonconforming as the methods mentioned before. However, the C^0IP method is superior to the penalization method (Babuška and Zlámal, 1973) due to symmetric interior penalties that lead to optimal convergence in energy and L^2 norm, see the results below. The symmetric penalties are traced back to Arnold (Arnold, 1982) for second-order elliptic partial differential equations, or even the original work from Nitsche (Nitsche, 1971). In the context of second-order elliptic problems, the differences of both penalization ideas were discussed in (Arnold, Brezzi, et al., 2002). In contrast to the SIPG method, C^0 Lagrange elements are used and interior penalty terms of first-order derivatives will enter the variational formulation, weakly imposing C^1-continuity.

We have found some studies on two-level and multilevel Schwarz methods for C^0 interior penalty discretizations of the model problem (4.1). In (Brenner and Wang, 2005) two-level additive Schwarz preconditioners with a generic overlap at the scale of fine-level elements were analyzed to prove that the preconditioned system's condition number is bounded by $\mathcal{O}(1 + H^3/\delta^3)$. H is the coarse-level mesh size and the overlap δ is a multiple of the fine-level mesh size h. Similarly, in (Feng and Karakashian, 2005), theory on two-level non-overlapping additive and multiplicative Schwarz smoothers was provided, showing also that the condition is bound by $\mathcal{O}(H^3/h^3)$. The same bounds were demonstrated for isogeometric discretizations in (Cho et al., 2018). Kanschat and Sharma (2014) followed Kanschat and Mao (2015) to study the (algebraic) identity of C^0IP and $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty methods

for biharmonic and Stokes problems, respectively, in two spatial dimensions. They proved that using multilevel Schwarz methods on vertex patches results in mathematically efficient numerical solvers with uniform convergence.

These works mainly focused on mathematical efficiency and did not address the challenge of gaining computational efficiency. We have already motivated and seen that overlapping Schwarz methods are prohibitively expensive in computational terms, in particular, for high-order discretizations. We emphasize that overlap is essential to obtain numerical solvers with uniform convergence. To this end, we extend the work in (Kanschat and Mao, 2015; Kanschat and Sharma, 2014), designing innovative algorithms to compute Schwarz smoothers on vertex patches cost-efficiently. In view of Chapter 3, we propose tensor product Schwarz smoothers such that the residuals are efficiently computed by matrix-free operator evaluation and the subspace corrections by fast diagonalization. Although the Bilaplacian is a separable differential operator, we will see that local discretization matrices do not (directly) admit a separable Kronecker representation (see Definition 2.1.12). However, we will utilize the Kronecker product singular value decomposition (KSVD) to obtain low-rank tensor product approximations that are amenable to fast diagonalization.

The chapter starts with a brief introduction of the C^0 interior penalty method regarding the Biharmonic model problem (4.1). In Section 4.1.1 numerical results underline the mathematical efficiency of multilevel Schwarz methods on vertex patches. Then, our main scientific contribution in obtaining fast and robust numerical robust solvers follows in Section 4.1.2: first, the low-rank tensor representation of local C^0IP discretization matrices is derived. Afterward, inexact local solvers amenable to fast diagonalization are designed that provide us with both, high mathematical and computational efficiency. Finally, we conclude the chapter by discussing computational costs and relate (prospective) implementations to the observed (parallel) performance for the Poisson problem.

4.1 C^0 Interior Penalty Method

We want to develop multilevel Schwarz smoothers by extending the concepts from Chapter 3. To this end, the main advantages of the C^0 interior penalty method are that it is well-defined for arbitrary finite element orders. Second, we can rely on standard tensor product structures again, which were introduced in Section 2.2. Consequently, the respective multilevel method is easy to implement, even for high-order elements, unlike the C^1-finite element methods mentioned before. If necessary, C^1-finite element approximations can be obtained from the interior penalty solutions via the post-processing procedure in (Brenner and Sung, 2005). We will see that not much of our implementation for the Poisson problem needs to be adapted to obtain efficient PDE solvers. For sufficiently smooth solutions of the model problem, the C^0 interior penalty method converges (quasi)-optimally in the energy as well as L^2 norm (Brenner, 2011) (convergence in L^2 norm is suboptimal only for quadratic polynomials).

Furthermore, in Chapter 5 we will make use of the relation between stream function solutions of the C^0 interior penalty method and Stokes flow solutions of a corresponding interior penalty method in two dimensions. Thus, the smoothers developed here will be used later to solve the Stokes problem efficiently.

The natural (weak) solution space for the biharmonic equation with clamped boundary conditions reads

$$V_{g,j} = \left\{ v \in H^2(\Omega) \mid \operatorname{Tr} v = g \text{ and } \operatorname{Tr} \boldsymbol{\nabla} v \cdot \boldsymbol{n} = j \text{ on } \partial\Omega \right\}. \tag{4.2}$$

Assuming homogeneous boundary conditions, i.e., $g \equiv 0$ and $j \equiv 0$ above, we use the abbreviated notation V instead of explicitly writing $V_{0,0}$ for the solution space. Then, V coincides with the definition of Sobolev space $H_0^2(\Omega)$ in (Adams and Fournier, 2003). The C^0-continuity and vanishing trace at the physical boundary already imply that the tangential derivative vanishes as well at $\partial\Omega$. Any $v \in V_{g,j}$ is decomposable into $v = v_0 + \tilde{u}$, where $v_0 \in V$ is the homogeneous part and $\tilde{u} \in V_{g,j}$ a particular lifting of the boundary conditions in (4.1). Thus, we assume from now on homogeneous clamped conditions, which simplifies notation without losing generality. Then, the weak problem reads: find $u \in V$ such that

$$a(u, v) = F(v) \quad \forall v \in V, \tag{4.3}$$

where

$$a(u, v) = \int_\Omega \boldsymbol{\nabla}^2 u : \boldsymbol{\nabla}^2 v \, \mathrm{d}\boldsymbol{x}, \tag{4.4a}$$

and

$$F(v) = \int_\Omega f v \, \mathrm{d}\boldsymbol{x}. \tag{4.4b}$$

The second-order tensor $\boldsymbol{\nabla}^2 v$ is the Hessian matrix of a scalar field v and the double dot product $A : B$ denotes the full contraction of second-order tensors A and B.

Given mesh the $\mathcal{T}_h$ with characteristic size h, we define the piecewise Sobolev space with differential index $r > 0$,

$$H^r(\mathcal{T}_h) := \left\{ v \in L^2(\Omega) \mid v|_K \in H^r(K) \quad \forall K \in \mathcal{T}_h \right\} \not\subseteq H^r(\Omega). \tag{4.5}$$

We choose standard C^0-continuous Lagrange elements based on $\mathbb{Q}_k$-polynomials, denoted as the finite element space V_h,

$$V_h = \left\{ v \in C^0(\bar{\Omega}) \mid v|_K \in \mathbb{Q}_k \quad \forall K \in \mathcal{T}_h \right\} \cap H_0^1(\Omega) \subset H^2(\mathcal{T}_h). \tag{4.6}$$

From the definition, it becomes apparent that V_h is nonconforming, this means V_h is not a subspace of $H_0^2(\Omega)$.

We adopt the notation from the symmetric interior penalty method in Section 3.1: let $\mathcal{E}_h^\circ$ and $\mathcal{E}_h^\partial$ denote the set of interior and boundary facets, respectively, and v^+ or v^- the trace of v on the interface $K^+ \cap K^-$ seen from cell K^+ or K^-, respectively. We choose $\boldsymbol{n}$ to be the outward pointing normal on boundary facets. On interior facets, it denotes the normal pointing from K^+ to K^-, i.e., $\boldsymbol{n} = \boldsymbol{n}^+$. We follow Definition 3.1.1 in defining average and jump operator for higher-order derivatives,

$$\begin{aligned} \left[\!\left[\frac{\partial v}{\partial \boldsymbol{n}} \right]\!\right](\boldsymbol{x}) &= \frac{\partial v^+}{\partial \boldsymbol{n}^+}(\boldsymbol{x}) + \frac{\partial v^-}{\partial \boldsymbol{n}^-}(\boldsymbol{x}) \qquad \boldsymbol{x} \in e \in \mathcal{E}_h^\circ, \\ \left[\!\left[\frac{\partial v}{\partial \boldsymbol{n}} \right]\!\right](\boldsymbol{x}) &= \frac{\partial v}{\partial \boldsymbol{n}}(\boldsymbol{x}) \qquad \boldsymbol{x} \in e \in \mathcal{E}_h^\partial, \end{aligned} \tag{4.7}$$

and

$$\begin{aligned} \left\{ \frac{\partial^2 v}{\partial \boldsymbol{n}^2} \right\}(\boldsymbol{x}) &= \frac{1}{2}\left(\frac{\partial^2 v^+}{\partial \boldsymbol{n}^2}(\boldsymbol{x}) + \frac{\partial^2 v^-}{\partial \boldsymbol{n}^2}(\boldsymbol{x}) \right) \qquad \boldsymbol{x} \in e \in \mathcal{E}_h^\circ, \\ \left\{ \frac{\partial^2 v}{\partial \boldsymbol{n}^2} \right\}(\boldsymbol{x}) &= \frac{\partial^2 v}{\partial \boldsymbol{n}^2}(\boldsymbol{x}) \qquad \boldsymbol{x} \in e \in \mathcal{E}_h^\partial. \end{aligned} \tag{4.8}$$

We emphasize that superscripts $+$ and $-$ for the normal vector were omitted due to the square and $\boldsymbol{n}^- = -\boldsymbol{n}$.

By multiplying (4.1) with a test function, integration by parts and stabilizing by Nitsche terms, the C^0 interior penalty formulation reads: find $u_h \in V_h$ such that

$$a_{c0ip;h}(u_h, v) = F_h(v) \quad \forall v \in V_h \tag{4.9}$$

with the bilinear form

$$\begin{aligned} a_{c0ip;h}(u, v) := &\int_{\mathcal{T}_h} \boldsymbol{\nabla}^2 u : \boldsymbol{\nabla}^2 v \,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\mathcal{E}_h} \left(\gamma_e \left[\!\left[\frac{\partial u}{\partial \boldsymbol{n}} \right]\!\right] \left[\!\left[\frac{\partial v}{\partial \boldsymbol{n}} \right]\!\right] - \left\{ \frac{\partial^2 u}{\partial \boldsymbol{n}^2} \right\} \left[\!\left[\frac{\partial v}{\partial \boldsymbol{n}} \right]\!\right] - \left[\!\left[\frac{\partial u}{\partial \boldsymbol{n}} \right]\!\right] \left\{ \frac{\partial^2 v}{\partial \boldsymbol{n}^2} \right\} \right) \mathrm{d}\sigma(\boldsymbol{x}), \end{aligned} \tag{4.10}$$

and the right-hand side operator

$$F_h(v) := \int_{\mathcal{T}_h} f v \,\mathrm{d}\boldsymbol{x}. \tag{4.11}$$

Due to the average of second-order derivatives at edges, the bilinear form is well-defined for functions in $H^{5/2+\epsilon}(\mathcal{T}_h)$. The integrals over a set of cells $\mathcal{T}_h$ or set of facets $\mathcal{E}_h$ is an abuse of notation, abbreviating the sum of individual integrals over each cell or facet, respectively. We refer to the cell integral of bilinear form (4.10) as *bulk* term and to the three face integrals, from left to right, as *penalty*, *consistency* and *adjoint consistency* term. The bilinear form was derived as follows: the (discrete) ansatz and test functions vanish at $\partial\Omega$ because the

tangential derivatives are continuous between elements, given homogeneous clamped boundary conditions (actually $g \equiv 0$ suffices) and the C^0-continuity in Ω. To this end, the first-order normal derivatives are penalized, which imposes C^1-continuity in the limit $h \to 0$. In the case of a heterogeneous boundary function j in (4.1), the right-hand side operator extends to

$$F_h(v) = \int_{\mathcal{T}_h} fv \,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{E}_h^\partial} \left(\gamma_e \, j \frac{\partial v}{\partial \boldsymbol{n}} - j \left\{ \frac{\partial^2 v}{\partial \boldsymbol{n}^2} \right\} \right) \mathrm{d}\sigma(\boldsymbol{x}). \tag{4.12}$$

We choose here the same interior penalty parameter as before for the SIPG method of the Poisson problem. The penalty factor was defined in (3.8). The numerical experiments below show that this penalty factor leads to a stable discretization.

Remark 4.1.1 (Elliptic regularity). Given $F \in H^{-2+\alpha}(\Omega)$, then, for some $\alpha \in (1/2, 1]$ the weak solution u in (4.3) is $H^{2+\alpha}$-regular and satisfies the stability estimate

$$\|u\|_{2+\alpha,\Omega} \leq C_\Omega \|F\|_{-2+\alpha,\Omega}\,, \tag{4.13}$$

see (Brenner and Sung, 2005) and references therein. We refer to α as the *index of elliptic regularity*.

The index of elliptic regularity depends on the interior angles of $\partial\Omega$'s vertices. For a convex polygonal domain $\alpha = 1$ is possible. Later, we consider the unit square as a domain, for which α is one.

Remark 4.1.2 (Well-posedness). The C^0 interior penalty method (4.9) is consistent and well-posed (Brenner and Sung, 2005), and adjoint consistent (Engel et al., 2002).

We define a mesh-dependent seminorm in terms of

$$|v|_h^2 = \sum_{K \in \mathcal{T}_h} |v|_{2,K}^2 + \sum_{e \in \mathcal{E}_h} \frac{\gamma_{0,e}}{h_e} \left\| \left[\!\left[\frac{\partial v}{\partial \boldsymbol{n}} \right]\!\right] \right\|_e^2, \tag{4.14}$$

where $h_e = \operatorname{diam} e$ and $\gamma_{0,e}$ is the interior penalty constant in (3.8). By a standard inverse estimate (P. Ciarlet, 1978) this seminorm is equivalent to the energy norm induced by $a_{c0ip;h}(\cdot,\cdot)$ on the finite element space V_h .

Remark 4.1.3 (Energy error). Let $f \in H^s(\Omega), s \in \mathbb{N}_0$, and the weak solution u in (4.3) belong to $H^{2+\alpha}(\Omega)$ with (elliptic) regularity index $\alpha \in (1/2, s+2]$. Then, the interior penalty solution u_h in (4.9) satisfies the energy error estimate

$$|u - u_h|_h \leq C h^{\min\{\alpha, k-1\}}, \tag{4.15}$$

see (Brenner, 2011).

When u is sufficiently smooth, that means $\alpha \geq k-1$, higher-order elements lead to an optimal convergence order $(k-1)$ of the discretization error measured in the energy norm.

Table 4.1 The energy error $|u - u_h|_h$ and its order of convergence p for the biharmonic problem with clamped boundary on $\Omega = [0,1]^2$. Finite element solutions u_h for varying polynomial degree are computed on a Cartesian mesh $\mathcal{T}_h$ subject to uniform refinement. Entries "—" are not computed.

2D	$\mathbb{Q}_2$		$\mathbb{Q}_3$		$\mathbb{Q}_4$	
h	$\lvert u-u_h\rvert_h$	p	$\lvert u-u_h\rvert_h$	p	$\lvert u-u_h\rvert_h$	p
$1/4$	$8.1 \times 10^{+0}$	—	$2.2 \times 10^{+0}$	—	4.0×10^{-1}	—
$1/8$	$3.7 \times 10^{+0}$	1.20	5.9×10^{-1}	2.03	5.2×10^{-2}	3.06
$1/16$	$1.8 \times 10^{+0}$	1.10	1.4×10^{-1}	2.04	6.2×10^{-3}	3.15
$1/32$	9.0×10^{-1}	1.04	3.7×10^{-2}	2.03	7.5×10^{-4}	3.07
$1/64$	4.4×10^{-1}	1.02	9.3×10^{-3}	2.02	9.3×10^{-5}	3.03
$1/128$	2.2×10^{-1}	1.01	2.3×10^{-3}	2.01	1.4×10^{-5}	2.72
$1/256$	1.1×10^{-1}	1.00	5.8×10^{-4}	2.01	—	—

In what follows, assume u to be the superposition of Gaussian bell curves from (3.16). Then, u is smooth and the volumetric load f and boundary functions g and j are determined for the biharmonic model problem (4.1) with domain $\Omega = [0,1]^2$. We emphasize that the Dirichlet boundary condition is imposed by using a particular lifting $u_{h;g}$ of the boundary function g as described before. Then, the respective C^0 interior penalty method is given by (4.9) when additionally subtracting $a_{c0ip;h}(u_{h;g}, v)$ from the right-hand side $F_h(v)$. In that context we speak of lifting the boundary data. The operator F_h is defined in (4.12) which weakly imposes the boundary condition $\frac{\partial u}{\partial \boldsymbol{n}} = j$. The solution $u_{h;0} \in V_h$ obtained is the "homogeneous" solution that needs to be added to the particular lifting computing the discrete solution $u_h = u_{h;0} + u_{h;g}$. For standard C^0-continuous Lagrange elements (4.6) based on $\mathbb{Q}_k$-polynomials with degree $k = 2, 3$ or 4 optimal convergence of the energy error is confirmed in Table 4.1. The discrete solution is computed for a sequence of uniformly refined Cartesian meshes $\mathcal{T}_h$, obtained by dividing the mesh width h in half. What is denoted as order of convergence p in Table 4.1 is the logarithm of the error reduction,

$$p = \log_2 \frac{\|u - u_h\|}{\|u - u_{h/2}\|}, \tag{4.16}$$

here defined for any arbitrary norm or seminorm $\|\cdot\|$. If the weak solution u of the model problem is not smooth enough the convergence rate is restricted to $\mathcal{O}(h^\alpha)$.

For our smooth solution, however, we expect from the theory in (Engel et al., 2002) optimal L^2-error convergence of order $(k+1)$ for polynomial degrees equal to or larger than three: adjoint consistency of the C^0 interior penalty method is essential here. Table 4.2 confirms our expectations. For quadratic tensor product polynomials, we obtain the suboptimal convergence order of $p = 2$. When analyzing the iterative solvers' convergence in terms of iterations in the next section, we will observe further deficiencies for quadratic polynomials

Table 4.2 The L^2-error $\|u - u_h\|_\Omega$ and its convergence order p for the biharmonic problem with clamped boundary on $\Omega = [0,1]^2$. Finite element solutions u_h for varying polynomial degree are computed on a Cartesian mesh $\mathcal{T}_h$ subject to uniform refinement. Entries "—" are not computed.

2D	$\mathbb{Q}_2$		$\mathbb{Q}_3$		$\mathbb{Q}_4$	
h	$\|u - u_h\|_\Omega$	p	$\|u - u_h\|_\Omega$	p	$\|u - u_h\|_\Omega$	p
$1/4$	3.1×10^{-2}	—	3.3×10^{-3}	—	2.0×10^{-4}	—
$1/8$	8.5×10^{-3}	2.03	1.7×10^{-4}	4.50	9.7×10^{-6}	4.59
$1/16$	2.2×10^{-3}	2.01	1.0×10^{-5}	4.22	3.4×10^{-7}	4.92
$1/32$	5.8×10^{-4}	2.00	6.3×10^{-7}	4.09	1.1×10^{-8}	4.98
$1/64$	1.4×10^{-4}	2.00	3.9×10^{-8}	4.04	1.6×10^{-8}	−0.56
$1/128$	3.7×10^{-5}	2.00	1.9×10^{-8}	1.01	2.2×10^{-7}	−3.77
$1/256$	9.3×10^{-6}	1.99	2.1×10^{-7}	−3.46	—	—

compared to higher-order polynomials. Due to limitations of floating-point arithmetic and an ill-conditioned discretization matrix ($\sim h^{-4}$), the errors do not decrease beyond 10^{-8} in Table 4.2.

4.1.1 Mathematical Efficiency of Schwarz Smoothers

Similar to the Poisson problem in Chapter 3, we first show the efficiency of Schwarz smoothers in terms of the solver's convergence steps, without taking any tensor structure into account. Later, we will derive low-rank tensor approximations of local solvers which enable sum factorization or fast diagonalization, see Section 2.1.2 or Section 2.1.3 respectively.

We begin with a coarse mesh $\mathcal{T}_1$ subdividing the unit square $\Omega = [0,1]^2$ into four congruent cells. We follow the induction in Section 2.3.1 to define a hierarchy of uniform Cartesian meshes $\mathcal{T}_\ell, l = 1, \dots, L$. The finite element spaces V_ℓ, defined by substituting $\mathcal{T}_h$ with $\mathcal{T}_\ell$ in (4.6), are nested. Multilevel bilinear forms are defined by

$$a_\ell(\cdot,\cdot) := a_{c0ip;h_\ell}(\cdot,\cdot).$$

Following Section 2.3.1, we use standard transfer operators between levels. It remains to define smoothers for the multigrid V-cycle step given by Algorithm 2. To this end, we define finite element subspaces $V_{\ell;j}$ consisting of all functions in V_ℓ with vanishing trace at the subdomain's boundary,

$$V_{\ell;j} = \left\{ v \in C^0(\bar{\Omega}) \mid v|_K \in \mathbb{Q}_k \; \forall K \in \mathcal{T}_j \quad \text{and} \quad v|_{\partial\Omega_j} = 0 \right\}.$$

We note that these are the same subspaces used for the Poisson problem, but we apply them here with one order higher due to the second-order derivatives in (4.9). Thus, the lowest-order

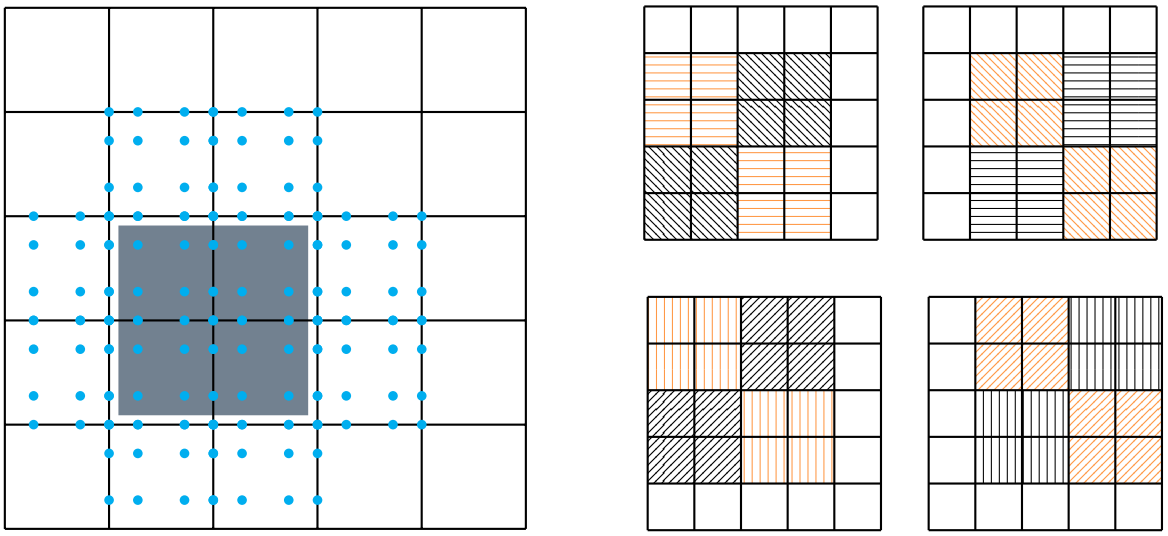
element consists of quadratic polynomials. The local problem is simply the restriction of the C^0 interior penalty method restricted to the subspace $V_{\ell;j}$. In that case, we obtain exact local solvers in the sense of (2.58). We emphasize that besides the strongly imposed boundary conditions $u_j|_{\partial\Omega_j} = 0$, boundary conditions for $\frac{\partial u_j}{\partial \boldsymbol{n}}$ are weakly imposed with different weights at the physical boundary $\partial\Omega_j \cap \partial\Omega$ and at interior parts of the subdomain's boundary $\partial\Omega_j \cap \Omega$, respectively. Section 3.3 provides the details of what is meant by "different weights".

We only consider smoothers on vertex patches for the biharmonic problem. Using smoothers on a single cell leads to either slow convergence or no convergence at all for high-order finite elements. In addition to the mathematical inferiority, we would have the same technical disadvantage as discussed in Section 3.2.1 for the Poisson problem. The disadvantage refers to a wide stencil of C^0-continuous finite elements, which results in even more colors than vertex patches.

The numerical setup is almost unchanged compared to the previous section and Section 3.1.1. We continue with the reference solution u defined in (3.16), which is a superposition of Gaussian bell curves. We use standard C^0-continuous Lagrange elements with Gauss-Lobatto nodal points, see Definition 2.2.3. However, we perform two pre- and post-smoothing steps (instead of one), respectively, for the multigrid V-cycle from Algorithm 2. We will develop inexact local solvers that enable exploiting tensor structure. We observed a significant decrease in the outermost solver's iterations when applying two smoothing steps (the gain was negligible for three or more). Thus, to achieve a fair comparison, we already use two smoothing steps for our exact local solvers here. Apart from this deviation, the sequence of uniformly refined Cartesian meshes, using a conjugate gradient solver until a relative residual reduction of $\epsilon_{rel} = 10^{-8}$ is reached and traversing colors backward when post smoothing continues to be the same. It remains to discuss the coloring and relaxation.

Coloring. In Section 3.1.1 the coloring was specified by face integrals inherent in the SIPG form, whereas in Section 3.2.1 the C^0-continuity of conforming finite elements was decisive. For the C^0-IP method, we have to consider both. Identifying by duality the nodal shape function basis of V_j with the local coefficient space $\mathbb{R}^{N_j}$, the stencil in Figure 4.1a shows the action of $A_\ell R_j^\mathsf{T}$ to those local coefficients. The stencil spreads out to adjacent cells as depicted due to face integrals inherent in the C^0 interior penalty form (4.10). The shape function basis of V_j is illustrated by nodal points, which are included in the shaded area, highlighting the restriction R_j of coefficients onto $\mathbb{R}^{N_j}$. Degrees of freedom associated with $\partial\Omega_j$ are excluded such that the homogeneous Dirichlet boundary conditions $u_j = 0$ are strongly imposed, part of homogeneous clamped conditions. Furthermore, the stencil has on purpose no nodal points at the physical boundary $\partial\Omega$ of the mesh since the same homogeneous conditions are strongly imposed at the global level. By global, we refer to the generic level ℓ. Therefore, two subspaces V_j and V_i are A_ℓ-orthogonal, i.e., satisfying

$$R_i A_\ell R_j^\mathsf{T} = 0,$$



(a) vertex patch stencil

(b) "red-black parqueting" (coloring)

Fig. 4.1 Illustration of the vertex patch stencil (*left*) given H^1-conforming $\mathbb{Q}_3$-elements, i.e., presenting the action of $A_\ell R_j^\mathsf{T}$, and optimal coloring for MVS (*right*) for C^0-IP discretizations. Gauss-Lobatto support points (●) represent degrees of freedom. The shaded area highlights the subdomain Ω_j; thus, shape functions associated with included nodal points form a basis for the subspace $V_{\ell;j}$. Hatched and dotted areas on the right-hand depict a multiple red-black coloring satisfying A_ℓ-orthogonality, same as Figure 3.2b for SIPG discretizations.

if their corresponding subdomains do not share a common facet. The parqueting with red-black coloring in Figure 4.1b is optimal for regular vertex patches, optimal in the sense of a minimal number of colors.

Relaxation. The natural relaxation parameter for additive vertex patch smoothers is $1/4$ due to possible overlap of cells. Following the discussion in Section 3.1.1, numerical experiments with a few thousand degrees of freedom and polynomial degrees less than four have shown that factors slightly larger than $1/4$ are slightly better, needing one or two solver iterations less. However, we also found a slight dependency of the polynomial degree regarding optimal factors. For convenience, we decided to use the natural relaxation parameter $1/4$ for the numerical results shown below.

In Table 4.3 solver iterations for both the additive and multiplicative vertex patch smoother are compared. Both smoothers lead to robust solvers with a few iterations needed: the number of solver iterations is independent of the discretization level L and the polynomial degree k (except for $k = 2$). For the Poisson problem, the number of iterations decreases with increasing finite element order for the multiplicative method. Iteration counts remain on similar levels for the additive variant when increasing the polynomial degree. Similar to the discretization errors discussed in Tables 4.1 and 4.2, biquadratic finite elements seem to be deficient. Without the constrained degrees of freedom to strongly impose or weakly impose boundary conditions, i.e., g or j, respectively, only a single degree of freedom locally

Table 4.3 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **exact** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^4 to 10^7 DoFs.

Level L	Convergence steps ν_{frac}						Colors
AVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	10.9	1
6	—	—	10.3	10.1	10.4	10.9	1
7	21.9	11.7	9.9	9.5	10.2	10.5	1
8	22.8	11.7	9.8	9.5	9.8	10.3	1
9	23.3	11.6	9.8	9.5	9.8	—	1
10	23.9	11.5	—	—	—	—	1
MVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	2.3	8
6	—	—	2.9	2.6	2.5	2.3	8
7	8.8	4.4	2.9	2.6	2.5	1.9	8
8	8.9	4.4	2.9	2.5	2.4	1.9	8
9	9.2	4.4	2.9	2.5	2.4	—	8
10	9.3	4.4	—	—	—	—	8

remains unconstrained. Thus, we believe AVS and MVS nearly act like "point-wise" Jacobi and Gauss-Seidel smoothers.

4.1.2 Tensor Product Schwarz Smoothers

We derive the low-rank tensor representation of local C^0 interior penalty matrices. Unfortunately, we do not obtain the separable Kronecker representation from Definition 2.1.12 that enables fast diagonalization. In two spatial dimensions, we use, among other options, the Kronecker product singular value decomposition (KSVD), which computes the best rank-2 approximation from the low-rank tensor representation of local matrices. The rank-2 approximation can then be fast diagonalized. Derivations will be presented for two dimensions but are easily extended to higher dimensions. However, we note that the KSVD is only applicable to order-2 tensor products and, to the best of our knowledge, there is no higher-dimensional generalization.

The Bilaplace operator

$$\Delta^2 = (\partial_{11} + \partial_{22})^2 = \partial_{1111} + 2\partial_{11}\partial_{22} + \partial_{2222}$$

is not separable on its own, but it is the square of a separable differential operator, obviously the Laplacian operator. From (Lynch et al., 1964) it is known that the finite difference

matrix $\hat{L}$ of the Laplacian preserves the tensor product structure of the differential operator one-to-one,

$$\hat{L} = \hat{L}^{(1)} \otimes I^{(2)} + I^{(1)} \otimes \hat{L}^{(2)}. \tag{4.17}$$

$\hat{L}^{(d)}$ denotes the univariate discretization matrix corresponding to ∂_{dd} and $I^{(d)}$ the identity matrix for $d = 1, 2$. Let the unitary matrix $\hat{Q}^{(d)}$ and diagonal matrix $\hat{\Lambda}^{(d)}$ define a diagonalization of $\hat{L}^{(d)}$, that means

$$(\hat{Q}^{(d)})^{\mathsf{T}} \hat{L}^{(d)} \hat{Q}^{(d)} = \hat{\Lambda}^{(d)},$$

for $d = 1, 2$. Then, the finite difference matrix $\hat{B}$ of the Bilaplacian is diagonalizable due to the diagonalization of $\hat{L}$ in (4.17),

$$\hat{B} = \hat{L}^2 = \hat{Q}^{(1)} \otimes \hat{Q}^{(2)} \left(\hat{\Lambda}^{(1)} \otimes I^{(2)} + I^{(1)} \otimes \hat{\Lambda}^{(2)} \right)^2 \left(\hat{Q}^{(1)} \otimes \hat{Q}^{(2)} \right)^{\mathsf{T}}. \tag{4.18}$$

Thus, a fast diagonalization, see Section 2.1.3, can be used to efficiently compute the inverse of $\hat{B}$, i.e., reducing computational costs significantly. The diagonalization of the finite difference method is studied in (Lynch et al., 1964): Lynch, Rice, and Thomas show that in general finite difference discretizations of the product of a separable differential operator are fast diagonalizable.

Unfortunately, in the context of finite elements this does not even hold for Cartesian meshes as seen in the following. Each Cartesian cell $K \in \mathcal{T}_h$ is the Cartesian product of intervals,

$$K = [a_1, b_1] \times [a_2, b_2].$$

The length of each interval is denoted by $h_d = b_d - a_d, d = 1, 2$. Evaluating the double contraction of second-order tensors, the bulk term of (4.10) becomes

$$\int_K \boldsymbol{\nabla}^2 u : \boldsymbol{\nabla}^2 v \, \mathrm{d}\boldsymbol{x} = \int_K \left(\partial_{11} u \partial_{11} v + 2 \partial_{12} u \partial_{12} v + \partial_{22} u \partial_{22} v \right) \mathrm{d}\boldsymbol{x}. \tag{4.19}$$

The bulk term was obtained by integration by parts of the Bilaplacian, which does not preserve the separable tensor structure. Moreover, we derive next that the finite element discretization of the Bilaplacian is not the product of a separable Kronecker representation. Nevertheless, it still admits a low-rank tensor decomposition. We adapt the notation from Sections 2.2, 3.1.2 and 3.2.2 and recall that Cartesian meshes preserve the separation of unit coordinates in real space. The mapping $\boldsymbol{F}_K \colon \hat{K} \to K$ from reference to real space was defined in (3.18). Besides, by definition (4.6) of V_h there exists for each $K \in \mathcal{T}_h$ a set of shape functions $\varphi_{\boldsymbol{i}} \in V_h$ such that their restrictions define a local shape function basis,

$$\varphi_{K;\boldsymbol{i}} := \varphi_{\boldsymbol{i}}|_K = \hat{\varphi}_{\boldsymbol{i}} \circ \boldsymbol{F}_K^{-1}, \tag{4.20}$$

for $\boldsymbol{i} = 1, \dots, N_{dof}$. The space spanned by shape functions $\varphi_{K;\boldsymbol{i}}$ is denoted $V(K)$. The unit shape functions $\hat{\varphi}_{\boldsymbol{i}}$, being introduced in Definition 2.2.3, are subject to a separation of variables,

$$\hat{\varphi}_{\boldsymbol{i}}(x_1, x_2) = \phi^{(1)}_{i_1}(x_1)\phi^{(2)}_{i_2}(x_2), \tag{4.21}$$

and lexicographic order, that means $\boldsymbol{i}$ denotes the unrolled multi-index i_1, i_2 as detailed in Definition 2.1.6. We derive a low-rank tensor representation for a single cell K first. In this way, it is easier to introduce some concepts, keeping the notational effort low. We emphasize that we do not use any cell-based Schwarz smoother. Afterwards, we derive the tensor represenation is derived for a Cartesian vertex patch Ω_j.

To this end, we assume boundary and transmission conditions $u = 0$ at the boundary ∂K of a generic cell K. The homogeneous Dirichlet condition is implicitly imposed by omitting all shape functions from $V(K)$ that do not vanish at ∂K. In view of Definition 2.2.3, any basis function $\varphi_{K;\boldsymbol{i}}$ needs to be omitted, where at least for one dimension d either $i_d = 1$ or $i_d = n_{dof}$ holds. The space which is spanned by remaining basis functions $\varphi_{K;\boldsymbol{i}}$ is denoted $V^\circ(K) \subset V(K)$.

The first- and second-order derivatives of $\boldsymbol{F}_K$ read

$$\hat{\partial}_i \boldsymbol{F}_{K;k}(\hat{\boldsymbol{x}}) = \delta_{ik} h_k \quad \text{and} \quad \hat{\partial}_{ij} \boldsymbol{F}_{K;k}(\hat{\boldsymbol{x}}) = 0, \quad \hat{\boldsymbol{x}} \in \hat{K}, \tag{4.22}$$

for $i, j = 1, \dots, D$. For a scalar field $f\colon \mathbb{R}^n \to \mathbb{R}$ and vector field $\boldsymbol{g}\colon \mathbb{R}^m \to \mathbb{R}^n$ second-order derivatives of its composition $(f \circ \boldsymbol{g})$ are determined by means of the chain and product rule,

$$\hat{\partial}_{ij}(f \circ \boldsymbol{g})(\hat{\boldsymbol{x}}) = \sum_{k,l=1}^{n} \partial_{kl} f(\boldsymbol{x}) \hat{\partial}_i \boldsymbol{g}_k(\hat{\boldsymbol{x}}) \hat{\partial}_j \boldsymbol{g}_l(\hat{\boldsymbol{x}}) + \sum_{k=1}^{n} \partial_k f(\boldsymbol{x}) \hat{\partial}_{ij} \boldsymbol{g}_k(\hat{\boldsymbol{x}}), \quad \boldsymbol{x} = \boldsymbol{g}(\hat{\boldsymbol{x}}), \tag{4.23}$$

for $i, j = 1, \dots, m$. With $\boldsymbol{F}_K$ substituting $\boldsymbol{g}$ and some shape function φ_K substituting f we obtain

$$\hat{\partial}_{ij}(\varphi_K \circ \boldsymbol{F}_K)(\hat{\boldsymbol{x}}) = h_i h_j \partial_{ij} \varphi_K(\boldsymbol{x}), \quad \boldsymbol{x} = \boldsymbol{F}_K(\hat{\boldsymbol{x}}), \quad \hat{\boldsymbol{x}} \in \hat{K}. \tag{4.24}$$

The bulk integral evaluated for tensor product shape functions factorizes into

$$\begin{aligned} \int_K \boldsymbol{\nabla}^2 \varphi_{K;\boldsymbol{j}} : \boldsymbol{\nabla}^2 \varphi_{K;\boldsymbol{i}} \,\mathrm{d}\boldsymbol{x} = &\int_0^1 \frac{1}{h_1^3} \hat{\phi}''_{j_1} \hat{\phi}''_{i_1} \,d\hat{x}_1 \int_0^1 \hat{\phi}_{j_2} \hat{\phi}_{i_2} h_2 \,d\hat{x}_2 \\ &+ 2 \int_0^1 \frac{1}{h_1} \hat{\phi}'_{j_1} \hat{\phi}'_{i_1} \,d\hat{x}_1 \int_0^1 \frac{1}{h_2} \hat{\phi}'_{j_2} \hat{\phi}'_{i_2} \,d\hat{x}_2 \\ &+ \int_0^1 \hat{\phi}_{j_1} \hat{\phi}_{i_1} h_1 \,d\hat{x}_1 \int_0^1 \frac{1}{h_2^3} \hat{\phi}''_{j_2} \hat{\phi}''_{i_2} \,d\hat{x}_2, \end{aligned} \tag{4.25}$$

following the ordering of summands in (4.19). We recall from (3.20) the definition of univariate mass matrices and discretizations of the Laplacian for each interval $[a_d, b_d]$,

$$\begin{aligned} M_{ij}^{(d)} &= \sum_{q=1}^{n_{quad}} \hat{\phi}_{i+1}(\hat{x}_q)\hat{\phi}_{j+1}(\hat{x}_q)h_d w_q \\ L_{ij}^{(d)} &= \sum_{q=1}^{n_{quad}} \frac{1}{h_d}\hat{\phi}'_{i+1}(\hat{x}_q)\hat{\phi}'_{j+1}(\hat{x}_q)w_q, \end{aligned} \tag{4.26}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. $(\hat{x}_q, \hat{w}_q)$ denote pairs of quadrature support points and weights on the unit interval, for $q = 1, \ldots, n_{quad}$. We emphasize that index $i + 1$ (resp. $j + 1$) is used on purpose in (4.26) omitting shape functions associated with ∂K. Similarly, for each dimension d, we obtain one-dimensional discretizations of the Bilaplacian by

$$B_{ij}^{(d)} = \sum_{q=1}^{n_{quad}} \frac{1}{h_d^3}\hat{\phi}''_{i+1}(\hat{x}_q)\hat{\phi}''_{j+1}(\hat{x}_q)w_q, \tag{4.27}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. Then, the discretization matrix of the bulk term (4.25) is a rank-3 tensor product of matrices,

$$B^{(1)} \otimes M^{(2)} + 2L^{(1)} \otimes L^{(2)} + M^{(1)} \otimes B^{(2)}. \tag{4.28}$$

Still assuming the auxiliary local problem on a generic cell K with homogeneous Dirichlet condition $u = 0$ at ∂K, it remains to derive low-rank representations for the face integrals of the C^0 interior penalty formulation (4.10), namely the consistency, adjoint consistency and penalty terms. We begin by discussing the latter. Let e_0 and e_1 denote the facets of cell K which are aligned with the first coordinate such that

$$e_p = y_p \times [a_2, b_2] \tag{4.29}$$

for $p = 0, 1$, with endpoints $y_0 = a_1$ and $y_1 = b_1$. The normal vector reads $\boldsymbol{n} = (\pm 1, 0)^\mathsf{T}$ with negative sign at e_0 and positive at e_1. For a basis function $\varphi_{K;\boldsymbol{i}} \in V^\circ(K)$ the jump of normal derivatives at facet e_p simplifies to

$$\left[\!\left[\frac{\partial \varphi_{K;\boldsymbol{i}}}{\partial \boldsymbol{n}} \right]\!\right] = \frac{\partial \varphi_{K;\boldsymbol{i}}}{\partial \boldsymbol{n}}.$$

Consequently, the penalty term reads

$$\int_{e_p} \gamma_{e_p} \frac{\partial \varphi_{K;\boldsymbol{i}}}{\partial \boldsymbol{n}} \frac{\partial \varphi_{K;\boldsymbol{j}}}{\partial \boldsymbol{n}} \,\mathrm{d}\boldsymbol{x} = \gamma_{e_p} \frac{1}{h_1^2}\hat{\phi}'_{i_1}(p)\hat{\phi}'_{j_1}(p) \int_0^1 \hat{\phi}_{i_2}\hat{\phi}_{j_2} h_2 \, d\hat{x}_2, \tag{4.30}$$

for $p = 0, 1$. To this end, we define discretization matrices $G_p^{(d)}$ for normal derivatives in a single unit coordinate, either $p = 0$ or $p = 1$,

$$\left(G_p^{(d)}\right)_{ij} = \frac{1}{h_d^2}\hat{\phi}_i'(p)\hat{\phi}_j'(p), \tag{4.31}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. The discretization matrix of the penalty term in (4.30) has a separable Kronecker representation,

$$\left(\gamma_{e_0} G_0^{(1)} + \gamma_{e_1} G_1^{(1)}\right) \otimes M^{(2)} + M^{(1)} \otimes \left(\gamma_{e_0} G_0^{(2)} + \gamma_{e_1} G_1^{(2)}\right). \tag{4.32}$$

For the same set of shape functions, the average of second-order normal derivatives simplifies to

$$\left\{\frac{\partial^2 \varphi_{K;\boldsymbol{i}}}{\partial \boldsymbol{n}^2}\right\} = \eta_e \frac{\partial^2 \varphi_{K;\boldsymbol{i}}}{\partial \boldsymbol{n}^2},$$

where $\eta_e = 1$ for any facet e at the physical boundary and $1/2$ otherwise. For facets e_p from (4.29), the consistency term in (4.10) factorizes into

$$-\int_{e_p} \eta_{e_p} \frac{\partial \varphi_{K;\boldsymbol{i}}}{\partial \boldsymbol{n}} \frac{\partial^2 \varphi_{K;\boldsymbol{j}}}{\partial \boldsymbol{n}^2} \,\mathrm{d}\boldsymbol{x} = \pm \frac{\eta_{e_p}}{h_1^3} \hat{\phi}_{i_1}'(p)\hat{\phi}_{j_1}''(p) \int_0^1 \hat{\phi}_{i_2}\hat{\phi}_{j_2} h_2 \, d\hat{x}_2, \tag{4.33}$$

with positive sign for $p = 0$ and negative for $p = 1$. We define univariate matrices for higher-order normal derivatives in a single coordinate, $p = 0$ or $p = 1$, as

$$\left(H_p^{(d)}\right)_{ij} = \frac{1}{h_d^3}\hat{\phi}_i'(p)\hat{\phi}_j''(p), \tag{4.34}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. The discretization matrix of consistency terms admits a separable Kronecker representation as well,

$$\left(\eta_{e_0} H_0^{(1)} - \eta_{e_1} H_1^{(1)}\right) \otimes M^{(2)} + M^{(1)} \otimes \left(\eta_{e_0} H_0^{(2)} - \eta_{e_1} H_1^{(2)}\right). \tag{4.35}$$

Using the same finite elements for both ansatz and test functions, the tensor representation of the adjoint consistency terms is simply the transpose of (4.35). We sum up the matrices from before,

$$N_p^{(d)} = \gamma_{e_p} G_p^{(d)} \pm \eta_{e_p} H_p^{(d)} \pm \eta_{e_p} \left(H_p^{(d)}\right)^{\mathsf{T}}, \tag{4.36}$$

for $d = 1, 2$, with positive sign for $p = 0$ and negative for $p = 1$. We refer to $N_p^{(d)}$ as Nitsche matrix. Finally, the discretization matrix of the C^0 interior penalty formulation on a single cell K (excluding shape functions associated with ∂K) has a rank-3 tensor representation

$$\underline{B}^{(1)} \otimes M^{(2)} + 2L^{(1)} \otimes L^{(2)} + M^{(1)} \otimes \underline{B}^{(2)}, \tag{4.37}$$

where

$$\underline{B}^{(d)} = B^{(d)} + N_0^{(d)} + N_1^{(d)}, \tag{4.38}$$

for $d = 1, 2$. What has been observed for the SIPG discretization of the Poisson problem is also confirmed here: contributions of Nitsche terms preserve the tensor structure of the bulk term on a single cell subject to Cartesian mapping. Consequently, $\underline{B}^{(d)}$ is the one-dimensional C^0 interior penalty discretization on interval $[a_d, b_d]$. In other words, Nitsche terms do not contribute to the Kronecker product $L^{(1)} \otimes L^{(2)}$ in (4.37).

We are interested in tensor product Schwarz smoothers on vertex patches. To this end, we extend the concepts for the auxiliary local problem to one on Cartesian vertex patches. First, we recall that the conforming finite element method for the Poisson problem had the same local space V_j. In particular, here and there, homogeneous Dirichlet conditions are imposed locally. Here, we additionally have homogeneous boundary conditions for normal derivatives, $\frac{\partial u}{\partial \boldsymbol{n}} = 0$, which are imposed weakly. Therefore, we simply extend what has been done for the Poisson problem in Section 3.2.2 to what needs to be done for the bulk integral here. Due to the inter-element continuity of shape functions dealing with cell-local and patch-local indices for shape functions leads to an overwhelming amount of sub- and superscripts. To this end, we try to keep it simple and consider a generic but specific Cartesian vertex patch

$$\Omega_j = [a_1^+, b_1^+] \cup [a_1^-, b_1^-] \times [a_2^+, b_2^+] \cup [a_2^-, b_2^-]$$

illustrated in Figure 3.5. We add to the notation from Section 3.2.2 that $\boldsymbol{n}^+ = (1, 0)^\mathsf{T} = -\boldsymbol{n}^-$ denote respective outward pointing normal vectors at

$$e = \{b_1^+\} \times [a_2^+, b_2^+], \tag{4.39}$$

the edge between cells K^+ and K^-. Given any two basis functions φ and ψ in V_j we define $\mathcal{K}$ as the set of cells for which both functions have non-trivial support. The bulk integral on vertex patch Ω_j reads

$$\int_{\Omega_j} \boldsymbol{\nabla}^2\varphi : \boldsymbol{\nabla}^2\psi \,\mathrm{d}\boldsymbol{x} = \sum_{K\in\mathcal{K}} \int_K \boldsymbol{\nabla}^2\varphi_{K;\boldsymbol{j}} : \boldsymbol{\nabla}^2\varphi_{K;\boldsymbol{i}} \,\mathrm{d}\boldsymbol{x},$$

where each integral over K admits a separation of variables as in (4.25). There are two points to notice. First, index $\boldsymbol{i}$ depends on K. Second, it follows the indexing in Definition 2.2.3: if $K = K^+$ in view of Figure 3.5, i.e., that $i_1 \neq 1$ and $i_2 \neq 1$ such that $\varphi \in V_j$. Both points apply to index $\boldsymbol{j}$ as well. Univariate mass matrices and discretizations of the Laplacian, namely $\boldsymbol{M}^{(d)}$ and $\boldsymbol{L}^{(d)}$, are defined for each interval $[a_d^+, b_d^+] \cup [a_d^-, b_d^-]$ in (3.47) to (3.50).

Similiarly, we define one-dimensional discretizations of the Bilaplacian,

$$\boldsymbol{B}^{(d)} = \begin{bmatrix} B_{++}^{(d)} & B_{+e}^{(d)} & 0 \\ (B_{+e}^{(d)})^\mathsf{T} & B_{ee}^{(d)} & (B_{-e}^{(d)})^\mathsf{T} \\ 0 & B_{-e}^{(d)} & B_{--}^{(d)} \end{bmatrix} \tag{4.40}$$

with

$$\left(B_{\pm\pm}^{(d)}\right)_{ij} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{\pm})^3} \hat{\phi}''_{i+1}(\hat{x}_q)\hat{\phi}''_{j+1}(\hat{x}_q)w_q \right), \tag{4.41a}$$

$$\left(B_{ee}^{(d)}\right)_{11} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{+})^3} \hat{\phi}''_{n_{dof}}(\hat{x}_q)\hat{\phi}''_{n_{dof}}(\hat{x}_q)w_q + \frac{1}{(h_d^{-})^3} \hat{\phi}''_{1}(\hat{x}_q)\hat{\phi}''_{1}(\hat{x}_q)w_q \right), \tag{4.41b}$$

$$\left(B_{+e}^{(d)}\right)_{i1} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{+})^3} \hat{\phi}''_{i+1}(\hat{x}_q)\hat{\phi}''_{n_{dof}}(\hat{x}_q)w_q \right), \tag{4.41c}$$

$$\left(B_{-e}^{(d)}\right)_{i1} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{-})^3} \hat{\phi}''_{i+1}(\hat{x}_q)\hat{\phi}''_{1}(\hat{x}_q)w_q \right), \tag{4.41d}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. As before in (3.48) and (3.50), indices $i + 1$ and $j + 1$ are used such that all shape functions associated with $\partial\Omega_j$ are omitted. The matrices $B_{\pm\pm}^{(d)}$ in (4.41a) are identical to $B^{(d)}$ from (4.27) for the respective interval, that is, substituting $[a_d, b_d]$ with $[a_d^+, b_d^+]$ or $[a_d^-, b_d^-]$. The shape functions associated with nodal points in the interior of $[a_d^+, b_d^+]$ or $[a_d^-, b_d^-]$, respectively, are continued by zero in the other interval, such that matrices $B_{\pm\mp}^{(d)}$ are zero in (4.40). The discretization of the bulk integral for a Cartesian vertex patch has the same rank-3 tensor representation as in (4.28),

$$\boldsymbol{B}^{(1)} \otimes \boldsymbol{M}^{(2)} + 2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)} + \boldsymbol{M}^{(1)} \otimes \boldsymbol{B}^{(2)}. \tag{4.42}$$

We emphasize that the ordering of blocks in the definitions of $\boldsymbol{B}^{(d)}, \boldsymbol{M}^{(d)}$ and $\boldsymbol{L}^{(d)}$ is crucial to obtain Kronecker products subject to the lexicographic ordering from Definition 2.1.6.

It remains to derive a low-rank tensor representation also for the face integrals, present in the C^0 interior penalty bilinear form (4.10). For generic ansatz and test functions φ and ψ in V_j, both having only support on a single cell, a representation was found in (4.36). For simplicity, consider in this paragraph two adjacent cells K^+ and K^- with interface e defined in (4.39). Given two basis functions φ_e and ψ_e in V_j, each is defined by duality to a nodal

point at e, visualized as bi-colored circle in Figure 3.5. Then, the penalty term reads

$$\int_e \gamma_e \left[\!\left[\frac{\partial \varphi_e}{\partial \boldsymbol{n}}\right]\!\right] \left[\!\left[\frac{\partial \psi_e}{\partial \boldsymbol{n}}\right]\!\right] \mathrm{d}\boldsymbol{x} = \gamma_e \left(\frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}'_{n_{dof}}(1)}{h_1^+ h_1^+} - \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}'_1(0)}{h_1^+ h_1^-} - \frac{\hat{\phi}'_1(0)\hat{\phi}'_{n_{dof}}(1)}{h_1^- h_1^+} + \frac{\hat{\phi}'_1(0)\hat{\phi}'_1(0)}{h_1^- h_1^-} \right) \int_0^1 \hat{\phi}_{j_2}\hat{\phi}_{i_2} h_2^+ \, d\hat{x}_2. \quad (4.43)$$

We applied that $\boldsymbol{F}_{K^+}(1) = b_1^+ = a_1^- = \boldsymbol{F}_{K^-}(0)$. Furthermore, index i_1 is fixed for each cell given, namely $\psi_e|_{K^+}(b_1^+, y) = \hat{\phi}_1(0)\hat{\phi}_{i_2}(\hat{y})$ and $\psi_e|_{K^-}(a_1^-, y) = \hat{\phi}_{n_{dof}}(1)\hat{\phi}_{i_2}(\hat{y})$. Since shape functions associated with $\partial\Omega_j$ are omitted index i_2 can not equal one. Defining in analogy to (4.31) the univariate matrix of normal derivatives evaluated at the single point e,

$$\left(G^{(1)}_{ee;e}\right)_{11} = \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}'_{n_{dof}}(1)}{h_1^+ h_1^+} - \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}'_1(0)}{h_1^+ h_1^-} - \frac{\hat{\phi}'_1(0)\hat{\phi}'_{n_{dof}}(1)}{h_1^- h_1^+} + \frac{\hat{\phi}'_1(0)\hat{\phi}'_1(0)}{h_1^- h_1^-}, \quad (4.44)$$

a Kronecker product of univariate discretization matrices for the exemplary penalty integral (4.43) exists,

$$\left(\gamma_e G^{(1)}_{ee;e} \otimes M^{(2)}_{++}\right)_{i_2 j_2}, \quad (4.45)$$

for $i_2 = 1, \ldots, n_{dof} - 2$ and $j_2 = 1, \ldots, n_{dof} - 2$. The number of subscripts in (4.44) starts to become overwhelming: the first two subscripts e refer to the test and ansatz basis function being associated with interface e, the third refers to the point evaluation at $e = b_1^+ = a_1^-$. Deriving matrices for shape functions associated with the interior of $[a_1^+, b_1^+]$ or interior of $[a_1^-, b_1^-]$ follows the same principle.

As in Section 3.2.2, we distinguish between sets of univariate shape functions associated with the interior of $[a_d^+, b_d^+]$, the interior of $[a_d^-, b_d^-]$ and "interface" e, identified by symbols $+$, $-$ and e. Discretization matrices of the Laplacian evaluated at the one-dimensional "interface" e between intervals $[a_d^+, b_d^+]$ and $[a_d^-, b_d^-]$ for the spatial dimension d are defined by

$$\left(G^{(d)}_{ee;e}\right)_{11} = \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}'_{n_{dof}}(1)}{h_d^+ h_d^+} - \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}'_1(0)}{h_d^+ h_d^-} - \frac{\hat{\phi}'_1(0)\hat{\phi}'_{n_{dof}}(1)}{h_d^- h_d^+} + \frac{\hat{\phi}'_1(0)\hat{\phi}'_1(0)}{h_d^- h_d^-}, \quad (4.46a)$$

$$\left(G^{(d)}_{+e;e}\right)_{i1} = \frac{\hat{\phi}'_{i+1}(1)\hat{\phi}'_{n_{dof}}(1)}{h_d^+ h_d^+} - \frac{\hat{\phi}'_{i+1}(1)\hat{\phi}'_1(0)}{h_d^+ h_d^-}, \quad (4.46b)$$

$$\left(G^{(d)}_{-e;e}\right)_{i1} = \frac{\hat{\phi}'_{i+1}(0)\hat{\phi}'_1(0)}{h_d^- h_d^-} - \frac{\hat{\phi}'_{i+1}(0)\hat{\phi}'_{n_{dof}}(1)}{h_d^- h_d^+}, \quad (4.46c)$$

$$\left(G^{(d)}_{+-;e}\right)_{ij} = -\frac{\hat{\phi}'_{i+1}(1)\hat{\phi}'_{j+1}(0)}{h_d^+ h_d^-}, \quad (4.46d)$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. The univariate shape function associated with the one-dimensional "interface" e has not a single-valued trace of the first-order derivative, i.e., $\hat{\phi}'_{n_{dof}}(1)$ seen from $[a_d^+, b_d^+]$ or $\hat{\phi}'_1(0)$ seen from $[a_d^-, b_d^-]$, already being transformed to unit space. Shape functions defined by duality to nodal functions in the interior of the interval $[a_d^+, b_d^+]$ have only a nonzero trace "on their side", i.e., $\hat{\phi}'_{i+1}(1)$ for $i = 1, \ldots, n_{dof}-2$. The same applies to shape functions associated with the interval $[a_d^-, b_d^-]$. Consequently, four summands arise from the jumps of ansatz and test functions in (4.46a), and two summands in (4.46b) and (4.46c), respectively. Due to discontinuities, there exist also nonzero contributions $G^{(d)}_{+-;e}$.

Besides the interface $e = b_d^+ = a_d^-$, we have to consider contributions at the left- and right-hand boundary of the merged interval, that is $e_0^+ = a_1^+$ and $e_1^- = b_1^-$. These contributions were derived in (4.31) for the auxiliary local problem on a single cell K. Replacing the generic interval $[a_d, b_d]$ by $[a_d^+, b_d^+]$ or $[a_d^-, b_d^-]$, respectively, we obtain

$$\frac{1}{(h_d^{\pm})^2} \hat{\phi}'_i(p) \hat{\phi}'_j(p), \tag{4.47}$$

for admissible indices i, j and unit coordinate p identifying the evaluation at $e_p^{\pm}$. Combining the point-evaluations at e_0^+, e and e_1^-, the univariate discretization matrices of the penalty term reads

$$\left(P^{(d)}_{ee}\right)_{11} = \frac{\gamma_{e_0^+}}{(h_d^+)^2} \hat{\phi}'_{n_{dof}}(0) \hat{\phi}'_{n_{dof}}(0) + \gamma_e \left(G^{(d)}_{ee;e}\right)_{11} + \frac{\gamma_{e_1^-}}{(h_d^-)^2} \hat{\phi}'_1(1) \hat{\phi}'_1(1), \tag{4.48a}$$

$$\left(P^{(d)}_{+e}\right)_{i1} = \frac{\gamma_{e_0^+}}{(h_d^+)^2} \hat{\phi}'_{i+1}(0) \hat{\phi}'_{n_{dof}}(0) + \gamma_e \left(G^{(d)}_{+e;e}\right)_{i1}, \tag{4.48b}$$

$$\left(P^{(d)}_{-e}\right)_{i1} = \gamma_e \left(G^{(d)}_{-e;e}\right)_{i1} + \frac{\gamma_{e_1^-}}{(h_d^-)^2} \hat{\phi}'_{i+1}(1) \hat{\phi}'_1(1), \tag{4.48c}$$

$$\left(P^{(d)}_{+-}\right)_{ij} = \gamma_e \left(G^{(d)}_{+-;e}\right)_{ij}, \tag{4.48d}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. The derivation of univariate discretizations for the consistency term follows the same concepts, in particular, in view of (4.34). Skipping a detailed derivation, the point-evaluation matrices at e_0^+, e and e_1^- concerning univariate consistency terms are defined by

$$\left(C^{(d)}_{ee}\right)_{11} = +\frac{\eta_{e_0^+}}{(h_d^+)^3} \hat{\phi}'_{n_{dof}}(0) \hat{\phi}''_{n_{dof}}(0) - \frac{1}{2} \left(H^{(d)}_{ee;e}\right)_{11} - \frac{\eta_{e_1^-}}{(h_d^-)^3} \hat{\phi}'_1(1) \hat{\phi}''_1(1), \tag{4.49a}$$

$$\left(C^{(d)}_{+e}\right)_{i1} = +\frac{\eta_{e_0^+}}{(h_d^+)^3} \hat{\phi}'_{i+1}(0) \hat{\phi}''_{n_{dof}}(0) - \frac{1}{2} \left(H^{(d)}_{+e;e}\right)_{i1}, \tag{4.49b}$$

$$\left(C^{(d)}_{-e}\right)_{i1} = -\frac{1}{2} \left(H^{(d)}_{-e;e}\right)_{i1} + \frac{\eta_{e_1^-}}{(h_d^-)^3} \hat{\phi}'_{i+1}(1) \hat{\phi}''_1(1), \tag{4.49c}$$

$$\left(C^{(d)}_{+-}\right)_{ij} = -\frac{1}{2} \left(H^{(d)}_{+-;e}\right)_{ij}, \tag{4.49d}$$

with matrices for higher-order derivatives at the one-dimensional "interface" e

$$\left(H_{ee}^{(d)}\right)_{11} = \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}''_{n_{dof}}(1)}{h_d^+(h_d^+)^2} + \frac{\hat{\phi}'_{n_{dof}}(1)\hat{\phi}''_d(0)}{h_d^+(h_d^-)^2} - \frac{\hat{\phi}'_d(0)\hat{\phi}''_{n_{dof}}(1)}{h_d^-(h_d^+)^2} - \frac{\hat{\phi}'_d(0)\hat{\phi}''_d(0)}{h_d^-(h_d^-)^2}, \tag{4.50a}$$

$$\left(H_{+e}^{(d)}\right)_{i1} = \frac{\hat{\phi}'_{i+1}(1)\hat{\phi}''_{n_{dof}}(1)}{h_d^+(h_d^+)^2} + \frac{\hat{\phi}'_{i+1}(1)\hat{\phi}''_d(0)}{h_d^+(h_d^-)^2}, \tag{4.50b}$$

$$\left(H_{-e}^{(d)}\right)_{i1} = -\frac{\hat{\phi}'_{i+1}(0)\hat{\phi}''_d(0)}{h_d^-(h_d^-)^2} - \frac{\hat{\phi}'_{i+1}(0)\hat{\phi}''_{n_{dof}}(1)}{h_d^-(h_d^+)^2}, \tag{4.50c}$$

$$\left(H_{+-}^{(d)}\right)_{ij} = \frac{\hat{\phi}'_{i+1}(1)\hat{\phi}''_{j+1}(0)}{h_d^+(h_d^-)^2}, \tag{4.50d}$$

for $i = 1, \ldots, n_{dof} - 2$ and $j = 1, \ldots, n_{dof} - 2$. η_e is one for facets e at the physical boundary $\partial\Omega$ and $1/2$ otherwise, which explains the factor of $1/2$ in (4.49) for contributions at the joint point e. Combining point-evaluations at e_0^+, e and e_1^- for the penalty, consistency and adjoint-consistency terms, the respective univariate Nitsche matrix

$$\boldsymbol{N}^{(d)} = \begin{bmatrix} N_{++}^{(d)} & N_{+e}^{(d)} & N_{+-}^{(d)} \\ (N_{+e}^{(d)})^\mathsf{T} & N_{ee}^{(d)} & (N_{-e}^{(d)})^\mathsf{T} \\ (N_{+-}^{(d)})^\mathsf{T} & N_{-e}^{(d)} & N_{--}^{(d)} \end{bmatrix} \tag{4.51}$$

is defined by

$$N_{\pm\pm}^{(d)} = N_{\pm\pm;0}^{(d)} \pm N_{\pm\pm;1}^{(d)}, \tag{4.52a}$$

$$N_{ee}^{(d)} = P_{ee}^{(d)} + C_{ee}^{(d)} + (C_{ee}^{(d)})^\mathsf{T}, \tag{4.52b}$$

$$N_{\pm e}^{(d)} = P_{\pm e}^{(d)} + C_{\pm e}^{(d)} + (C_{\pm e}^{(d)})^\mathsf{T}, \tag{4.52c}$$

$$N_{+-}^{(d)} = P_{+-}^{(d)} + C_{+-}^{(d)} + (C_{+-}^{(d)})^\mathsf{T}. \tag{4.52d}$$

Replacing the generic interval $[a_d, b_d]$ in (4.36) by $[a_d^+, b_d^+]$ or $[a_d^-, b_d^-]$, respectively, the matrices $N_{\pm\pm;p}^{(d)}$ in (4.52a) are defined, either at endpoints e_p^+ or at e_p^-, $p = 0, 1$. The C^0 interior penalty discretization matrix for a Cartesian vertex patch admits a rank-3 tensor representation,

$$\underline{\boldsymbol{B}}^{(1)} \otimes \boldsymbol{M}^{(2)} + 2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)} + \boldsymbol{M}^{(1)} \otimes \underline{\boldsymbol{B}}^{(2)}, \tag{4.53}$$

with

$$\underline{\boldsymbol{B}}^{(d)} = \boldsymbol{B}^{(d)} + \boldsymbol{N}^{(d)}. \tag{4.54}$$

We notice that (4.54) defines the one-dimensional C^0 interior penalty discretization for each merged interval $[a_d^+, b_d^+] \cup [a_d^-, b_d^-]$, $d = 1, 2$.

The local C^0 interior penalty problem is not amenable to fast diagonalization because (4.53) is not identical to the separable Kronecker representation (2.21) needed. However, omitting

the elementary tensor $\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)}$ in (4.53) a separable Kronecker product is obtained. We refer to this local solver as *Bila* since it involves only univariate mass matrices and discretizations of the one-dimensional C^0 interior penalty formulation of the Bilaplacian. Another way fast diagonalization may be exploited is to compute a best rank-r Kronecker product approximation using a Kronecker singular value decomposition (KSVD). We note that this option is restricted to two spatial dimensions. The KSVD with its best approximation property was introduced in Section 2.1.4. In what follows, we analyze the number of solver iterations with respect to these local solvers for the same solver settings given in Section 4.1.1.

Bila. If the term $\boldsymbol{L}_j^{(1)} \otimes \boldsymbol{L}_j^{(2)}$ which involves mixed-partial derivatives arising from the bulk term is omitted, local solvers are inexact in the sense of (2.67). However, they admit then the separable Kronecker representation

$$\tilde{A}_j = \underline{\boldsymbol{B}}_j^{(1)} \otimes \boldsymbol{M}_j^{(2)} + \boldsymbol{M}_j^{(1)} \otimes \underline{\boldsymbol{B}}_j^{(2)}, \tag{4.55}$$

that enables fast diagonalization. The matrices $\underline{\boldsymbol{B}}_j^{(d)}$ and $\boldsymbol{M}_j^{(d)}$ are symmetric, positive semi-definite or symmetric, positive definite, respectively, for $d = 1, 2$. Consequently, generalized eigenvalue problems are well-defined, such that a fast diagonalization is feasible. From now on, We use the subscript j to explicitly refer to the local solver for vertex patch Ω_j. The positive definiteness of $\tilde{A}_j$ ensures that the Assumption 2.3 of local stability is satisfied.

Comparing the number of solver iterations in Table 4.4 to those for exact local solvers in Table 4.3, we observe that additive vertex patch smoothers using the inexact method are slightly better concerning this metric. This observation is unexpected, but supposedly more inaccurate local solvers may lead to "better" smoothing. In addition, the differences are minimal. For low polynomial degrees k, the multiplicative vertex patch methods compare at similar levels. Still, with a growing gap in favor of exact local solvers. Using inexact local solvers, on the one hand, the number of iterations grows slowly with increasing polynomial degrees. On the other hand, a descending trend of the number of iterations is observed using exact local solvers. Nevertheless, the gap in iteration counts is moderate, and the solver's convergence speed is excellent for both kinds of local solvers. The computational cost and memory intensity of these inexact but fast diagonalizable local solvers (which we refer to as Bila local solvers) is significantly reduced, discussed in detail in Section 4.2. Therefore, Schwarz smoothers with Bila solvers are expected to be superior in computational efficiency.

The Kronecker product singular value decomposition (KSVD), which was introduced in Section 2.1.4, computes a best rank-r approximation $\tilde{A}_j$ of some matrix A_j,

$$A_j \approx \tilde{A}_j = \sum_{i=1}^{r} \tilde{A}_{j;i}^{(1)} \otimes \tilde{A}_{j;i}^{(2)}, \tag{4.56}$$

Table 4.4 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **Bila** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^4 to 10^7 DoFs.

Level L	Convergence steps ν_{frac}						Colors
AVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	10.9	1
6	—	—	9.2	9.9	10.3	10.8	1
7	19.2	10.4	9.0	9.3	9.8	10.3	1
8	19.6	10.3	9.0	9.2	9.5	9.9	1
9	19.9	10.3	9.0	9.1	9.4	—	1
10	20.1	10.2	—	—	—	—	1
MVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	5.8	8
6	—	—	4.2	4.5	5.2	5.8	8
7	9.2	4.8	4.2	4.5	5.2	5.7	8
8	9.4	4.8	4.2	4.5	5.1	5.7	8
9	9.5	4.8	4.2	4.4	4.9	—	8
10	9.5	4.8	—	—	—	—	8

for a fixed tensor rank r. Using Algorithm 1 to compute the KSVD for a matrix A_j admitting a low-rank tensor representation, the computational costs are significantly reduced, see Remark 2.1.20. We recall that the exact local solvers have the rank-3 representation (4.57).

It is noteworthy that a rank-3 KSVD approximation $\tilde{A}_j$ of the local C^0 interior penalty discretization matrix

$$A_j = \underline{\boldsymbol{B}}_j^{(1)} \otimes \boldsymbol{M}_j^{(2)} + 2\boldsymbol{L}_j^{(1)} \otimes \boldsymbol{L}_j^{(2)} + \boldsymbol{M}_j^{(1)} \otimes \underline{\boldsymbol{B}}_j^{(2)}, \tag{4.57}$$

is identical as "matrix". However, the three elementary tensors in (4.56) differ in general from those in (4.57) due to the best approximation property of the KSVD. Let us state some structural properties of the KSVD that are relevant in our context, found in the original works (N. P. Pitsianis, 1997; Van Loan and N. Pitsianis, 1993):

1. If A is symmetric, each of the r elementary tensors of the respective rank-r KSVD is symmetric.

2. If A is symmetric, positive definite, then its rank-1 KSVD is symmetric and positive definite, so are both matrices $\tilde{A}_1^{(1)}$ and $\tilde{A}_1^{(2)}$.

We emphasize that positive definiteness is only preserved for the rank-1 KSVD and not for higher ranks.

We will refer to local solvers based on a Kronecker product singular value decomposition with appropriate but fixed rank r as $\text{KSVD}_{1\text{i}}$, where the subscript 1 means using the best rank-1 approximation,

$$\tilde{A}_{j;1}^{(1)} \otimes \tilde{A}_{j;1}^{(2)},$$

and the subscript i using a second rank-1 tensor from (4.56),

$$\tilde{A}_{j;i}^{(1)} \otimes \tilde{A}_{j;i}^{(2)}$$

with $1 < i \leq r$. The second elementary tensor is optional and is omitted for the first option discussed.

KSVD_1. The first approach is to approximate local problems only by means of a rank-1 KSVD

$$\tilde{A}_{j;1}^{(1)} \otimes \tilde{A}_{j;1}^{(2)}. \tag{4.58}$$

Since local matrices A_j are symmetric and positive definite, eigenvalue decompositions are well-defined,

$$(Q^{(d)})^{\mathsf{T}} \tilde{A}_j^{(d)} Q^{(d)} = \Lambda^{(d)} \tag{4.59}$$

for $d = 1, 2$. The diagonal matrix $\Lambda^{(d)}$ consists of eigenvalues and the columns of $Q^{(d)}$ are corresponding eigenvectors. Given these eigendecompositions, a fast diagonalization is obtained,

$$(Q^{(1)} \otimes Q^{(2)})^{\mathsf{T}} \, \tilde{A}_{j;1}^{(1)} \otimes \tilde{A}_{j;1}^{(2)} \, (Q^{(1)} \otimes Q^{(2)}) = \Lambda^{(1)} \otimes \Lambda^{(2)}, \tag{4.60}$$

which is even simpler than (2.24), but enables the same fast inversions as in Section 2.1.3. Compared to previous exact or inexact local solvers, the smoothing quality obtained by inexact local solvers based on the rank-1 KSVD is inferior, shown by the numerical results in Table 4.5. Looking at bi-cubic elements, the iteration counts are for both smoothers, additive and multiplicative, more than a factor of two higher; with a growing gap for higher-order elements, for instance, more than a factor of seven for the polynomial degree $k = 7$. However, we notice that these rank-1 local solvers smooth well enough to obtain a numerical solver whose convergence is uniform with respect to the mesh size.

Table 4.4 has shown that "separable" rank-2 local solvers (referred to as Bila local solvers) smooth well. Thus, using the best rank-2 approximation of the local matrix A_j instead, we expect to perform equally well. However, we will encounter some difficulties that make it infeasible to apply the exact inverse of the rank-2 KSVD. Let

$$\tilde{A}_j = \tilde{A}_{j;1}^{(1)} \otimes \tilde{A}_{j;1}^{(2)} + \tilde{A}_{j;2}^{(1)} \otimes \tilde{A}_{j;2}^{(2)} \tag{4.61}$$

denote the rank-2 KSVD of the local matrix A_j. We note that due to the best approximation property $\tilde{A}_{j;1}^{(1)}$ and $\tilde{A}_{j;1}^{(2)}$ are identical in (4.58) and (4.61). In other words, we simply add a

Table 4.5 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with $\mathbf{KSVD_1}$ local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^4 to 10^7 DoFs.

Level L	Convergence steps ν_{frac}						Colors
AVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	63.5	1
6	—	—	27.3	27.3	41.6	66.9	1
7	31.4	26.3	27.4	27.4	41.5	70.3	1
8	34.7	26.5	26.7	26.7	41.3	73.4	1
9	37.4	26.8	26.5	26.5	40.6	—	1
10	39.9	26.8	—	—	—	—	1
MVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	19.6	8
6	—	—	11.5	14.4	16.3	19.4	8
7	10.0	11.5	11.5	14.3	16.3	18.9	8
8	10.1	11.6	11.5	14.3	16.2	18.8	8
9	10.1	11.6	11.5	13.9	15.9	—	8
10	10.1	11.7	—	—	—	—	8

second elementary tensor,

$$\tilde{A}^{(1)}_{j;2} \otimes \tilde{A}^{(2)}_{j;2},$$

to the local problems of the rank-1 approximation in (4.58). Since local matrices A_j are symmetric so are the Kronecker factors $\tilde{A}^{(d)}_{j;i}$ from (4.61), for $i = 1, 2$ and $d = 1, 2$. In addition, $\tilde{A}^{(1)}_{j;1}$ and $\tilde{A}^{(2)}_{j;1}$ are positive definite. Consequently, there generalized eigenvalue decompositions are well-defined,

$$\begin{aligned} (Z^{(d)})^\mathsf{T} \tilde{A}^{(d)}_{j;2} Z^{(d)} &= \Lambda^{(d)}, \\ (Z^{(d)})^\mathsf{T} \tilde{A}^{(d)}_{j;1} Z^{(d)} &= I^{(d)}, \end{aligned} \tag{4.62}$$

with real eigenvalues $\Lambda^{(d)}$, generalized eigenvectors $Z^{(d)}$ and identity matrices $I^{(d)}$ of appropriate size, for $d = 1, 2$. In that case, $\tilde{A}_j$ diagonalizes into

$$(Z^{(1)} \otimes Z^{(2)})^\mathsf{T} \; \tilde{A}_j \; (Z^{(1)} \otimes Z^{(2)}) = I^{(1)} \otimes I^{(2)} + \Lambda^{(1)} \otimes \Lambda^{(2)}, \tag{4.63}$$

where the sum of Kronecker products on the right-hand side is a diagonal matrix. Therefore, the rank-2 approximation $\tilde{A}_j$ can be inverted cost-efficiently, similar but not identical to the fast diagonalization in (2.25). Nevertheless, the computational complexities remain the same as discussed in Remark 2.1.14.

Table 4.6 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with $\mathbf{KSVD_{13}}$ local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^4 to 10^7 DoFs.

Level L	Convergence steps ν_{frac}						Colors
AVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	63.7	1
6	—	—	27.3	32.7	41.6	67.8	1
7	31.3	26.3	27.4	32.4	41.5	71.4	1
8	34.7	26.5	26.9	32.5	41.3	74.7	1
9	37.5	26.9	26.6	31.8	40.7	—	1
10	40.1	26.9	—	—	—	—	1
MVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	19.5	8
6	—	—	11.5	14.4	16.4	19.3	8
7	10.1	11.6	11.5	14.3	16.3	18.9	8
8	10.1	11.6	11.5	14.2	16.3	18.8	8
9	10.1	11.7	11.5	13.9	16.2	—	8
10	10.2	11.7	—	—	—	—	8

Although the local matrix A_j is positive definite, it is not guaranteed that $\tilde{A}_j$ is, which violates a necessary assumption of the Schwarz theory; to be precise, the Assumption 2.3 of local stability is not satisfied. Numerical experiments have shown this is not only posing a problem in theory: in practice, most rank-2 KSVDs based on the C^0 interior penalty formulation of local biharmonic problems were indefinite, in particular for higher-order finite elements and small mesh size h, which frequently resulted in diverging CG solvers.

$\mathbf{KSVD_{13}}$. One approach to overcome this problem is computing the rank-3 KSVD and using the sum of the *first* and *third* elementary tensor. We refer to this approximation as KSVD_{13}. Since A_j has a rank-3 tensor representation and is positive definite, it is straightforward to prove positive definiteness of such approximations. Numerical results in Tables 4.5 and 4.6 reveal that adding the third tensor to the best rank-1 approximation from (4.58) does not improve smoothing at all. The numbers in Table 4.5 refer to using the KSVD_1 local solvers, i.e., using the inverse of the rank-1 KSVD for subspace corrections. We recall that the elementary tensors of the KSVD approximation are naturally ordered by their approximation quality. On the one hand, we obtain cost-efficient and feasible smoothers for KSVD_{13} local solvers. On the other hand, they lead to numerical solvers with inferior convergence. Therefore, we do not recommend applying these local solvers.

$\mathbf{KSVD_{12}}(\alpha)$ Instead, we recommend a second option that is considering the best rank-2 approximation $\tilde{A}_j$ in (4.61), but scaling $\tilde{A}^{(1)}_{j;2} \otimes \tilde{A}^{(2)}_{j;2}$ by a positive factor α less than one such

Table 4.7 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with $\mathbf{KSVD_{12}}(\alpha)$ local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^4 to 10^7 DoFs.

Level L	Convergence steps ν_{frac}						Colors
AVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	17.8	1
6	—	—	13.6	14.6	16.3	17.0	1
7	21.8	11.1	13.3	14.6	16.3	16.8	1
8	22.5	11.1	12.9	14.3	15.9	16.7	1
9	22.9	11.0	12.8	13.8	15.8	—	1
10	23.5	10.9	—	—	—	—	1
MVS	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
5	—	—	—	—	—	4.9	8
6	—	—	4.1	4.3	4.6	4.8	8
7	8.8	4.4	4.1	4.3	4.6	4.8	8
8	8.9	4.4	4.1	4.3	4.5	4.8	8
9	9.3	4.4	4.1	4.3	4.5	—	8
10	9.3	4.4	—	—	—	—	8

that a positive definite approximation is obtained. The approximation reads

$$\tilde{A}_{\alpha;j} = \tilde{A}^{(1)}_{j;1} \otimes \tilde{A}^{(2)}_{j;1} + \alpha \tilde{A}^{(1)}_{j;2} \otimes \tilde{A}^{(2)}_{j;2}, \tag{4.64}$$

which is better than the rank-1 KSVD but worse than the rank-2 KSVD with respect to the approximation quality measured in the Frobenius norm. The closer α is to one, the better its approximation. The additional error concerning the Frobenius norm compared to the rank-2 KSVD is given by $\alpha\tilde{\sigma}_2$, with singular value $\tilde{\sigma}_2$ being computed in Algorithm 1. After computing the rank-2 KSVD and the generalized eigenvalue problems (4.62), the factor α is easily determined and a diagonalization of $\tilde{A}_{\alpha;j}$ is given by

$$(Z^{(1)} \otimes Z^{(2)})^{\mathsf{T}}\, \tilde{A}_{\alpha,j}\, (Z^{(1)} \otimes Z^{(2)}) = I^{(1)} \otimes I^{(2)} + \alpha \Lambda^{(1)} \otimes \Lambda^{(2)}, \tag{4.65}$$

without any extra computational costs. Let λ_{min} denote the minimal eigenvalue with respect to $\Lambda^{(1)} \otimes \Lambda^{(2)}$, then, $\mu_{min} = 1 + \lambda_{min}$ characterizes the minimal eigenvalue of $\tilde{A}_j$ in (4.61). If μ_{min} is negative, $\tilde{A}_{\alpha,j}$ is positive definite for any $\alpha < \alpha_{max}$ with $\alpha_{max} = {}^{-1}/_{\lambda_{min}}$. In computations, we use $\alpha = \alpha_{max} - \epsilon$ utilizing a small positive ϵ that sufficiently bounds the smallest eigenvalue of $\tilde{A}_{\alpha,j}$ away from zero. Then, the Assumption 2.3 of local stability is satisfied.

Comparing the numerical results in Table 4.7 to the Tables 4.3 and 4.4, we observe that the inexact local solvers given by (4.64) smooth well. For additive vertex patches, the iteration counts grow slightly but steadily with increasing polynomial degrees compared to exact and Bila local solvers. For multiplicative vertex patch smoothers, numerical solvers with uniform convergence are obtained, also independent of the polynomial degree: in particular, requiring slightly more iteration steps than for exact local solvers, but slightly less than for Bila local solvers. We believe that the relaxation factor $\omega = 0.25$ is sub-optimal for our $\mathrm{KSVD}_{12}(\alpha)$ local solvers because MVS smoothers that do not require relaxation perform better and compare at similar levels as with exact local solvers. However, this claim needs to be proved in theory and in practice, exceeding the scope of this work.

4.2 Conclusion

An extensive (parallel) performance analysis of the tensor product Schwarz smoothers from Section 4.1.2 also exceeds the thesis's scope. However, we will elaborate next why almost the same computational efficiency as for the Poisson problem, see Section 3.4, can be expected. Thus, a performance analysis would not add new major findings.

4.2.1 Computational Efficiency

At the time when the numerical experiments in Section 4.1.2 were conducted, `deal.II`'s matrix-free framework did not provide the functionality to compute integration-based evaluations involving second-order derivatives of shape functions, not even for standard Lagrange elements. However, in theory, not much needs to be changed to obtain an efficient operator application for the Bilaplacian, compared to the efficient operator application of the Laplace operator in Section 2.2. Computing the cell-based operator application $A_K u_K$ for the biharmonic model problem follows the same structure as equation (2.50) and subsequent equations. In analogy to the $\mathbb{R}^D$-valued interpolation of gradients $\mathfrak{u}^{\boldsymbol{\nabla}}$ from (2.54), we have to compute $\mathbb{R}^{D\times D}$-valued interpolation of Hessians here,

$$\mathfrak{u}^{\boldsymbol{\nabla}^2}_{q_1,\dots,q_D} := \begin{bmatrix} \mathfrak{u}^{\partial_{11}}_{q_1,\dots,q_D} & \dots & \mathfrak{u}^{\partial_{1D}}_{q_1,\dots,q_D} \\ \vdots & & \\ \mathfrak{u}^{\partial_{D1}}_{q_1,\dots,q_D} & \dots & \mathfrak{u}^{\partial_{DD}}_{q_1,\dots,q_D} \end{bmatrix}$$

Note that $\mathfrak{u}^{\boldsymbol{\nabla}^2}$ is a multi-dimensional array of order D, where each entry is the second-order derivative of a corresponding ansatz function u evaluated in the quadrature point $(x_{q_1},\dots,x_{q_D})$. The interpolation tensor of each second-order derivative $\mathfrak{u}^{\partial_{ij}}_{q_1,\dots,q_D}$ (simultaneously evaluated in all quadrature points) is efficiently computed using a sum factorization similar to (2.53). The same applies to the second sum factorization in (2.56), testing against second-order derivatives of test functions. An analogous rationale holds for efficient operator

Table 4.8 Comparing computational complexities of standard and tensor-based algorithms for Schwarz smoothers on vertex patches. "Standard" refers to using inverse SVDs for subspace corrections. We compare against the two tensor product Schwarz smoothers from Section 4.1.2 with the equally-best mathematical efficiency, i.e, using either Bila or $\text{KSVD}_{12}(\alpha)$ local solvers. The last column (*Ext.*) refers to a method's extensibility to dimensions higher than two.

Method	Summary	Comp. complex.		Memory	Ext.
		setup	apply		
Exact	A_j^{-1} by inverse SVD	$\mathcal{O}(n^{3D})$	$\mathcal{O}(n^{2D})$	n^{2D}	No limit
Bila	Omit $L \otimes L$ from A_j	—	—	—	No limit
	A_j^{-1} by fast diagonalization	$\mathcal{O}(Dn^3)$	$\mathcal{O}(Dn^{D+1})$	Dn^2	No limit
$\text{KSVD}_{12}(\alpha)$	Rank-2 KSVD of A_j	$\mathcal{O}(6n^2)$	—	$4n^2$	Only in 2D
	A_j^{-1} by fast diagonalization	$\mathcal{O}(Dn^3)$	$\mathcal{O}(Dn^{D+1})$	Dn^2	No limit

evaluations concerning the face integrals of the C^0 interior penalty method. Due to the symmetry of second-order derivatives, further arithmetic operations can be saved. The change from unit to real space (and vice versa) needs to be adapted. However, this is particularly easy for Cartesian coordinate transformations, for instance, see (4.24) for second-order derivatives of shape functions. Implementing efficient matrix-free operator evaluation is the only missing piece in our software for multilevel Schwarz methods. Then, forward operators and especially the residuals of Schwarz smoothers are computed cost-efficiently.

Computing subspace corrections is already highly-optimized, using the implementations of the software package `TPSS` for tensor product matrices as well as their fast diagonalization and KSVDs. If the input matrix admits already a low-rank tensor representation, our implementation of the KSVD given by Algorithm 1 (based on the Lanczos tridiagonalization) makes use of it such that the arithmetic complexity and memory intensity are reduced, see Remarks 2.1.20 and 2.1.21, respectively. The "small-sized" singular value decomposition in Algorithm 1 is computed via the `LAPACK` routine `XGESDD`. For similar but different fast diagonalizations in (4.60) and (4.65), we have extended our tensor product matrix class `TensorProductMatrixSymmetricSum` in `deal.II` to the more general class `TensorProductMatrix` in `TPSS`. For more details on `TPSS`'s functionality, we refer to Remark 3.4.1 and Section 1.2.

Besides the simple modifications for matrix-free operator evaluation, we rely on the same implementations of cost-efficient tensor product Schwarz smoothers compared to the Poisson problem, see Remark 3.4.1. In Table 4.8 the two tensor product Schwarz smoothers with the best mathematical efficiency, i.e., using either Bila or $\text{KSVD}_{12}(\alpha)$ local solvers, are summarized and compared to smoothers with exact local solvers (not amenable to fast inversion) in terms of arithmetic operations and memory consumption. Although we assumed two spatial dimensions throughout this chapter, the derivation of the low-rank tensor presentation of (local) C^0 interior penalty discretizations and some tensor techniques are

readily extensible to three dimensions. Thus, we explicitly state the number of dimensions D in Table 4.8 when comparing computational complexity (abbreviated by *Comp. complex.*) and memory intensity in Table 4.8. In particular, the one-time cost of inverting local matrices is drastically reduced: $\mathcal{O}(Dn^3)$ for fast diagonalized but inexact local solvers versus $\mathcal{O}(Dn^6)$ for exact local solvers based on standard inversion (i.e., utilizing standard SVDs for inversion), already in two dimensions. We note that $n = 2k - 1$, the number of degrees of freedom per dimension per vertex patch. An efficient operator application requires $\mathcal{O}(Dn^{D+1})$ arithmetic operations. Therefore, using any tensor product Schwarz smoother, either with Bila or $\text{KSVD}_{12}(\alpha)$ local solvers, we can expect a similarly high computational efficiency of numerical solvers as already being observed in Section 3.4.2 for the Poisson problem. For the tensor product Schwarz smoothers summarized in Table 4.8, we observed a comparable mathematical efficiency of respective numerical solvers in Tables 4.3, 4.4 and 4.7.

4.2.2 Outlook

In the future, we want to extend the matrix-free operator evaluation in `deal.II` to second-order derivatives first, obtaining high-performance computations for large-scale biharmonic problems. We recommend the tensor product Schwarz smoothers using $\text{KSVD}_{12}(\alpha)$ local solvers for two spatial dimensions: the best-approximation property of the KSVD may handle other "sources of inexactness", for instance, using non-Cartesian meshes or non-trivial coefficients depending on spatial coordinates. However, it is also appealing to extend our smoothers with Bila local solvers to three dimensions, and examine if they compare equally well to exact local solvers as in two dimensions. In the next chapter, we will make use of the efficient local solvers developed here to develop tensor product Schwarz smoothers for the Stokes problem there.

Chapter 5

STOKES PROBLEM

In this chapter, we discuss a method for the model problem of *Stokes equations*

$$-2\nabla \cdot \boldsymbol{\epsilon}(\boldsymbol{u}) + \boldsymbol{\nabla} p = \boldsymbol{f} \qquad \text{in } \Omega, \tag{5.1a}$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{5.1b}$$

$$\boldsymbol{u} = \boldsymbol{g} \qquad \text{on } \partial\Omega, \tag{5.1c}$$

where Ω is a polygonal domain in $\mathbb{R}^D$ with $D = 2$. Although restricting ourselves to two spatial dimensions, we use a generic dimension index D since many concepts can easily be extended to three dimensions. The vector fields $\boldsymbol{f}$ and $\boldsymbol{g}$ are given and sufficiently regular. The *symmetric gradient* of a vector field $\boldsymbol{v}\colon \mathbb{R}^D \to \mathbb{R}^D$ is defined by

$$\boldsymbol{\epsilon}(\boldsymbol{v}) = \frac{\boldsymbol{\nabla v} + (\boldsymbol{\nabla v})^{\mathsf{T}}}{2} \tag{5.2}$$

with

$$\boldsymbol{\nabla v} = \begin{bmatrix} \partial_1 \boldsymbol{v}_1 & \dots & \partial_D \boldsymbol{v}_1 \\ \vdots & & \vdots \\ \partial_1 \boldsymbol{v}_D & \dots & \partial_D \boldsymbol{v}_D \end{bmatrix}. \tag{5.3}$$

The *divergence* of a tensor-field $\boldsymbol{\sigma}\colon \mathbb{R}^D \to \mathbb{R}^{D\times D}$ is given by

$$\nabla \cdot \boldsymbol{\sigma} = \begin{pmatrix} \sum_{j=1}^{D} \partial_j \boldsymbol{\sigma}_{1j} \\ \vdots \\ \sum_{j=1}^{D} \partial_j \boldsymbol{\sigma}_{Dj} \end{pmatrix}. \tag{5.4}$$

The Stokes equations (5.1), determining the equations of motion for Stokes flow, are the limit of the steady-state Navier-Stokes equations for very low Reynolds numbers. Assuming viscous forces dominate over inertial forces, the latter are eliminated from the momentum balance in Navier-Stokes equations which leads to (5.1a). The conservation of mass is modeled

by (5.1b), assuming a steady state and a fluid density which is constantly one. Stokes flow is also referred to as creeping flow since it models flow where the viscosities are large or fluid velocities are assumed to be slow. There exist many use cases, ranging from movement of microorganisms over viscous polymers to the flow of magma.

Nevertheless, the biggest motivation comes from the fact that it models a special case of incompressible flow of Newtonian fluids. Developing efficient and robust numerical solvers for the model problem (5.1) is a major step towards fast solvers for (incompressible) flow. Fluid dynamics is ubiquituous, thus, there is no need to mention all its real-world applications here. The computational fluid dynamics (CFD) community demands robust and efficient numerical solvers that solve flow simulations with millions of time steps and trillions of space-dependent unknowns in a reasonably low amount of elapsed time.

Among many others, we refer to a series of works (Fehn, Kronbichler, et al., 2019; Fehn, Wall, et al., 2017, 2018a,b,c; Piatkowski and Bastian, 2019; Piatkowski, Müthing, et al., 2018) for fast and robust numerical solvers given stable DG discretizations of (incompressible) flow problems. An overview of these works, in particular, how they relate mutually and to other state-of-the-art numerical solvers, is given in (Arndt, Fehn, et al., 2020; Bastian, Altenbernd, et al., 2020). In the series of literature mentioned, consistent stabilization strategies are developed to weakly enforce inter-element mass conservation and the divergence-free constraint. In (Piatkowski and Bastian, 2019) a pressure-robust formulation with $\boldsymbol{H}^{\mathrm{div}}$-reconstruction was used. For details on pressure-robustness we refer to (Linke, 2014; Linke et al., 2016). The other works cited before apply stabilized L^2-conforming discretizations, which were compared to exactly divergence-free $\boldsymbol{H}^{\mathrm{div}}$-conforming discretizations in (Fehn, Kronbichler, et al., 2019).

We focus on the latter, following (Kanschat and Mao, 2015; Kanschat and Sharma, 2014). In particular, we utilize the (exact) algebraic relation of C^0 and $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty methods for biharmonic and Stokes problems, respectively, presented therein. The multilevel Schwarz methods in (Kanschat and Mao, 2015; Kanschat and Sharma, 2014) show high mathematical efficiency. More importantly, they lead to numerical solvers with uniform convergence and, thus, are scalable in size. However, the overlapping Schwarz smoothers on vertex patches are known to be prohibitively expensive, in particular, for high-order finite elements.

In this chapter, we will address this challenge, designing cost-efficient algorithms for Schwarz smoothers. Simultaneously to our research, other fast and efficient multilevel Schwarz methods utilizing matrix-free operators and preconditioners were developed for $\boldsymbol{H}^{\mathrm{div}}$-conforming discretizations concerning. We refer to Barker and Kolev (2020) who designed state-of-the-art preconditioners for high-order $\boldsymbol{H}^{\mathrm{curl}}$-conforming discretizations of the Maxwell operator based on the auxiliary space framework. $\boldsymbol{H}^{\mathrm{curl}}$-conforming methods naturally relate to $\boldsymbol{H}^{\mathrm{div}}$-conforming methods in view of finite element exterior calculus (Arnold, Falk, and Winther, 2000, 2006; Hiptmair, 2002). In (Brubeck and Farrell, 2021), fast and

robust multilevel additive Schwarz methods on vertex patches are presented for a mixed formulation of linear elasticity (being equivalent to the Stokes problem in the limit of a (nearly) incompressible material) based on the FEM-SEM equivalence studied in (Pazner, 2020). Their efficient algorithms are even applicable to unstructured vertex-patches and may handle mixed derivatives. However, they do not achieve optimal computational complexity in general.

We put in this chapter emphasis on designing algorithms that optimally compute Schwarz smoothers on structured vertex patches, presenting our main scientific contribution. To this end, we make use of the (algebraic) equality of C^0 and $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty method from (Kanschat and Mao, 2015; Kanschat and Sharma, 2014) for subspace corrections. It enables us to utilize the tensor product Schwarz methods from Chapter 4 for cost-efficient smoothing. In addition, we will see that a (local) reconstruction of the pressure is needed. Thus, we adopt the concepts in (Caussignac, 1987) to the model problem (5.1) and extend the pressure post-processing to numerically stable Raviart-Thomas elements.

The chapter starts with introducing the Stokes equations, in particular, the well-posedness of its weak formulation and basic notations. In Section 5.2, the $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty method, the known relation to stream functions, and our strategy for a pressure post-processing is elaborated for the simplified Stokes problem. The high mathematical efficiency of Schwarz smoothers on vertex patches and respective numerical solvers with uniform convergence are demonstrated by numerical experiments. Our main scientific contribution, namely computing tensor product Schwarz smoothers via cost-efficient algorithms, is presented in Section 5.2.5. Briefly discussing necessary modifications given the (actual) Stokes equations, similar mathematical efficiency and cost-efficient algorithms for tensor product Schwarz smoothers are demonstrated in Section 5.3. The chapter concludes with discussing computational complexity, relating to the performance analysis in Section 3.4 for the Poisson problem, and giving an outlook on highly-optimized (prospective) implementations.

5.1 The Stokes Equations

We adopt the nomenclature from the computational fluid dynamics (CFD) community. We refer to $\boldsymbol{u}$ and p in (5.1) as velocity and pressure field, respectively. Vector-valued or, more generally, tensor-valued differential operators, functions and associated function spaces are highlighted in bold font. A vector field $\boldsymbol{v}$ is called *divergence-free*, if there holds

$$\nabla \cdot \boldsymbol{v} = 0. \tag{5.5}$$

Flow described by a divergence-free function is called *incompressible*, thus, refering to (5.1b) as incompressibility condition. The natural velocity space for Dirichlet boundary conditions (5.1c)

is

$$\boldsymbol{V} = \left\{ \boldsymbol{v} \in \boldsymbol{H}^1(\Omega) \mid \operatorname{Tr} \boldsymbol{v} = \boldsymbol{g} \right\}, \tag{5.6}$$

and, in particular, $\boldsymbol{V} = \boldsymbol{H}_0^1(\Omega)$ if homogeneous boundary conditions are considered. In the context of flow problems, homogeneous Dirichlet conditions are called *no-slip* conditions. If we assume for some velocity $\boldsymbol{v} \in \boldsymbol{V}$ that $\boldsymbol{v} \cdot \boldsymbol{n} = 0$ on the whole boundary $\partial\Omega$, then, by integration of parts we obtain

$$\int_\Omega \nabla \cdot \boldsymbol{v} p \,\mathrm{d}\boldsymbol{x} = -\int_\Omega \boldsymbol{v} \cdot \boldsymbol{\nabla} p \,\mathrm{d}\boldsymbol{x} + \int_{\partial\Omega} \boldsymbol{v} \cdot \boldsymbol{n} p \,\mathrm{d}\sigma(\boldsymbol{x}) = -\int_\Omega \boldsymbol{v} \cdot \boldsymbol{\nabla} p \,\mathrm{d}\boldsymbol{x}. \tag{5.7}$$

Consequently, if the velocity solution $\boldsymbol{u}$ in (5.1) satisfies

$$\boldsymbol{u} \cdot \boldsymbol{n} = 0 \text{ on } \partial\Omega, \tag{5.8}$$

testing the first equation (5.1a) by velocities in $\boldsymbol{H}_0^1(\Omega)$ reveals that the pressure solution p is only determined up to a constant. For simplicity, we assume no-slip conditions for the remainder of this chapter. In case of nonzero boundary data $\boldsymbol{g}$ a lifting of any particular function endowed with this boundary condition results in simply solving the Stokes equations with no-slip conditions and a modified right-hand side. The natural pressure space is defined as

$$Q = L_0^2(\Omega) := \left\{ q \in L^2(\Omega) \mid \int_\Omega q \,\mathrm{d}\boldsymbol{x} = 0 \right\}, \tag{5.9}$$

the space of mean value free functions in $L^2(\Omega)$. The weak formulation of the model problem reads: Find the velocity-pressure pair $(\boldsymbol{u}, p) \in \boldsymbol{V} \times Q = \boldsymbol{H}_0^1(\Omega) \times L_0^2(\Omega)$ such that

$$a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) = F(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}, \tag{5.10a}$$

$$b(\boldsymbol{u}, q) = 0 \qquad \forall q \in Q, \tag{5.10b}$$

with bilinear forms

$$a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega 2\boldsymbol{\epsilon}(\boldsymbol{u}) : \boldsymbol{\epsilon}(\boldsymbol{v}) \,\mathrm{d}\boldsymbol{x}, \tag{5.11a}$$

$$b(\boldsymbol{v}, q) = -\int_\Omega \nabla \cdot \boldsymbol{v} q \,\mathrm{d}\boldsymbol{x}, \tag{5.11b}$$

and the right-hand side operator

$$F(\boldsymbol{v}) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x}. \tag{5.12}$$

We refer to (5.10) as *saddle point system* and the associated bilinear form on the product space $\boldsymbol{V} \times Q$ reads

$$\mathfrak{a}((\boldsymbol{u}, p), (\boldsymbol{v}, q)) := a(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) + b(\boldsymbol{u}, q). \tag{5.13}$$

Instead of postulating conditions on $\mathfrak{a}(\cdot,\cdot)$ for well-posedness of the saddle point problem, the well-posedness is guaranteed by assumptions on $a(\cdot,\cdot)$ and $b(\cdot,\cdot)$, respectively, see (Brezzi, 1974). Following the abstract theory on saddle point problems in (Brezzi and Fortin, 1991, Chapter 2), crucial for existence and uniqueness of solutions in (5.10) is that the divergence operator $\nabla\cdot\colon V \to Q'$ (i.e. the operator associated with $b(\cdot,\cdot)$ from (5.11b)) satisfies the *inf-sup condition*,

$$\exists \beta > 0 \quad \text{such that} \quad \inf_{q\in Q} \sup_{\boldsymbol{v}\in \boldsymbol{V}} \frac{b(\boldsymbol{v},q)}{\|\boldsymbol{v}\|_{1,\Omega}\|q\|_{\Omega}} > \beta, \tag{5.14}$$

and the bilinear form $a(\cdot,\cdot)$ is coercive on $\mathrm{Ker}(\nabla\cdot)$. The kernel of $\nabla\cdot$ can be written as

$$\mathrm{Ker}(\nabla\cdot) = \left\{\boldsymbol{v}\in\boldsymbol{V} \mid b(\boldsymbol{v},q) = 0 \quad \forall q\in Q\right\}. \tag{5.15}$$

The orthogonal complement of $\mathrm{Ker}(\nabla\cdot)$ as a subspace of $\boldsymbol{V}$ is denoted by $\mathrm{Ker}(\nabla\cdot)^{\perp}$.

Lemma 5.1.1. *Let Ω be connected. Then, the map $\nabla\cdot\colon \mathrm{Ker}(\nabla\cdot)^{\perp} \to Q'$ is an isomorphism. There exists a positive constant β such that*

$$\forall q\in Q \quad \exists \boldsymbol{v}\in\mathrm{Ker}(\nabla\cdot) \quad \textit{with} \quad \nabla\cdot\boldsymbol{v} = Iq \;\; \textit{and} \;\; \beta\|\boldsymbol{v}\|_{1,\Omega} \le \|q\|_{\Omega}, \tag{5.16}$$

where I denotes the L^2 Riesz isomorphism.

Proof. See (Girault and Raviart, 1986) and, in particular, (Ladyženskaja, 1969) for the estimate in (5.16). □

Remark 5.1.2 (Well-posedness). The inf-sup condition (5.14) follows from Lemma 5.1.1 and from a Korn inequality in (Duvaut and Lions, 1976) that $a(\cdot,\cdot)$ is coercive on $\boldsymbol{V}$. Then, the weak formulation (5.10) of the Stokes problem (5.1) with no-slip conditions is well-posed (Brezzi and Fortin, 1991, Chapter 2).

Assuming a vanishing trace on (a part of) $\partial\Omega$, a simplified Korn inequality can be found in (Braess, 2013). An equivalent statement to Lemma 5.1.1 for $\boldsymbol{\nabla}\colon L^2(\Omega) \to V'$, the adjoint operator of the divergence, was proved even earlier by Nečas (Nečas, 1967). The equivalence between both statements and the inf-sup condition relies on implications of the closed range theorem (Yosida, 1965).

5.2 The Simplified Stokes Equations

We analyze and develop tensor-structured Schwarz smoothers for the *simplified Stokes equations*,

$$-\boldsymbol{\Delta}\,\boldsymbol{u} + \boldsymbol{\nabla} p = \boldsymbol{f} \qquad \text{in } \Omega, \tag{5.17a}$$

$$\nabla \cdot \boldsymbol{u} = 0 \qquad \text{in } \Omega, \tag{5.17b}$$

$$\boldsymbol{u} = 0 \qquad \text{on } \partial\Omega, \tag{5.17c}$$

The *Laplacian* of a vector field $\boldsymbol{v}\colon \mathbb{R}^D \to \mathbb{R}^D$ is the Laplacian applied to each vector component,

$$\boldsymbol{\Delta}\,\boldsymbol{v} = \begin{pmatrix} \Delta\,\boldsymbol{v}_1 \\ \vdots \\ \Delta\,\boldsymbol{v}_D \end{pmatrix}. \tag{5.18}$$

For any divergence-free velocity $\boldsymbol{u} \in \boldsymbol{H}_0^1(\Omega) \cap \boldsymbol{H}^2(\Omega)$ there holds

$$2(\boldsymbol{\epsilon}(\boldsymbol{u}), \boldsymbol{\epsilon}(\boldsymbol{v}))_\Omega = (\boldsymbol{\nabla}\boldsymbol{u}, \boldsymbol{\nabla}\boldsymbol{v})_\Omega \qquad \forall \boldsymbol{v} \in \boldsymbol{H}_0^1(\Omega) \cap \boldsymbol{H}^2(\Omega), \tag{5.19}$$

which justifies the simplification if no-slip boundary conditions are present. Given nonzero boundary data $\boldsymbol{g}$, the simplified Stokes equations become a physically inexact model for (slow) flow problems. Nevertheless, from a methodological perspective these equations still retain their appeal for preconditioning and smoothing. The solution of the simplified Stokes equations approximates well the solution of the physically correct model problem (5.1).

Replacing in (5.11a) symmetric gradients, we define the so-called *simplified bilinear form*

$$\widetilde{a}(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega \boldsymbol{\nabla}\boldsymbol{u} : \boldsymbol{\nabla}\boldsymbol{v}\,\mathrm{d}\boldsymbol{x}. \tag{5.20}$$

The weak formulation of the simplified model problem with no-slip boundary conditions reads: Find $(\boldsymbol{u}, p) \in \boldsymbol{H}_0^1(\Omega) \times L_0^2(\Omega)$ such that

$$\widetilde{a}(\boldsymbol{u}, \boldsymbol{v}) + b(\boldsymbol{v}, p) = F(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}, \tag{5.21a}$$

$$b(\boldsymbol{u}, q) \qquad\quad = 0 \qquad \forall q \in Q, \tag{5.21b}$$

with $b(\cdot,\cdot)$ and F previously defined in (5.11b) and (5.12), respectively. The well-posedness of the simplified formulation is already implied by Remark 5.1.2.

5.2.1 $\boldsymbol{H}^{\mathrm{div}}$ Interior Penalty Method

In this section we discuss multilevel Schwarz methods exploiting tensor structure for the simplified model problem (5.17). First results for multigrid methods in $\boldsymbol{H}^{\mathrm{div}}$ and in $\boldsymbol{H}^{\mathrm{curl}}$ are

traced back to Hiptmair (Hiptmair, 1998; Hiptmair, 1997). The theory therein was unified and simplified in (Hiptmair and Toselli, 2000): it relies on the overlapping Schwarz methods defined in (Smith et al., 2004). The multilevel Schwarz methods we use are more related to the theoretical studies by Arnold, Falk and Winther. For example, see the $\boldsymbol{H}^{\mathrm{div}}$-conforming multilevel methods in two spatial dimensions (Arnold, Falk, and Winther, 1997), which were generalized to three dimensions and $\boldsymbol{H}^{\mathrm{curl}}$ (Arnold, Falk, and Winther, 2000). In both publications Schwarz methods for boundary value problems associated with the differential operator

$$\boldsymbol{\Lambda} = \mathrm{id} - \boldsymbol{grad}\,\mathrm{div}$$

were developed and analyzed. Moreover, it was shown that local problems on vertex patches are necessary to obtain robust multigrid solvers and that, for example, simple cell-based or face-based smoothers are not sufficient. The same concepts were extended to a $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty method for the simplified Stokes equations in (Kanschat and Mao, 2015), using the equivalence between mixed formulations and singularly peturbed, divergence-dominated elliptic forms as introduced in (Schöberl, 1999a,b). In all studies a (discrete) Helmholtz decomposition relying on exact sequences of (discrete) spaces was the essential mathematical tool. The decomposition leverages the special properties of the divergence operator. More recently, this concept was studied in the broader context of finite element exterior calculus, presenting a unified tool for many finite element applications (Arnold, Boffi, and Bonizzoni, 2014; Arnold, Falk, and Winther, 2006, 2010; Bonizzoni and Kanschat, 2021; Hiptmair, 2002). We will make use of this theory when exploiting the identity between the $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty method here and the C^0 interior penalty method from Section 4.1, see (Kanschat and Sharma, 2014). Therefore, we solve the latter problem locally on each vertex patch obtaining a local stream function φ_j such that $\boldsymbol{\nabla} \times \varphi_j$ is the divergence-free velocity solving the corresponding $\boldsymbol{H}^{\mathrm{div}}$ interior penalty formulation locally. The multilevel finite element discretizations will be given by a hierarchy of $\boldsymbol{H}^{\mathrm{div}}$ interior penalty methods, thus, local pressure solutions are reconstructed from corresponding stream function solutions. To this end, we adopt a similar two-step post-process as in (Caussignac, 1987). To our best knowledge such Schwarz smoothers have not been applied before. Furthermore, we will discuss inexact local solvers for stream functions following Section 4.1.2, which significantly reduce computational costs and memory-intensity.

We begin with defining the divergence-conforming Sobolev space

$$\boldsymbol{H}^{\mathrm{div}}(\Omega) = \left\{ \boldsymbol{v} \in \boldsymbol{L}^2(\Omega) \mid \nabla \cdot \boldsymbol{v} \in \boldsymbol{L}^2(\Omega) \right\}, \tag{5.22}$$

endowed with the inner product

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle_{\mathrm{div},\Omega} = \int_\Omega \boldsymbol{u} \cdot \boldsymbol{v}\,\mathrm{d}\boldsymbol{x} + \int_\Omega \nabla \cdot \boldsymbol{u} \nabla \cdot \boldsymbol{v}\,\mathrm{d}\boldsymbol{x}, \tag{5.23}$$

which induces the norm $\|\cdot\|_{\mathrm{div},\Omega}$. The next lemma defines the leading principle when constructing $\boldsymbol{H}^{\mathrm{div}}$-conforming finite elements.

Lemma 5.2.1. *Let $\mathcal{T}_h$ be a mesh of domain Ω with $\mathcal{E}_h^\circ$ being the set of interior facets, the following statements are equivalent:*

(i) $\boldsymbol{v} \in \boldsymbol{H}^{div}(\Omega)$

(ii) $\boldsymbol{v}|_K \in \boldsymbol{H}^{div}(K)$ *for all* $K \in \mathcal{T}_h$ *and* $[\![\boldsymbol{v} \cdot \boldsymbol{n}]\!] = 0$ *on each* $e \in \mathcal{E}_h^\circ$

Proof. See (Raviart and Thomas, 1977) and references therein. □

Consequently, $\boldsymbol{H}^{\mathrm{div}}$-conforming finite elements have continuous normal components between elements. Furthermore, from a hierarchy of nested meshes nested $\boldsymbol{H}^{\mathrm{div}}$-conforming approximation spaces are naturally obtained. The normal component also plays a major role when imposing essential boundary conditions:

Lemma 5.2.2. *The (normal) trace operator* $\mathrm{Tr}_{\boldsymbol{n}} \colon \boldsymbol{C}^\infty(\overline{\Omega}) \to C^\infty(\overline{\partial\Omega})$, $\boldsymbol{v} \mapsto \boldsymbol{v} \cdot \boldsymbol{n}$, *can be extended to a continuous, linear mapping*

$$\mathrm{Tr}_{\boldsymbol{n}} \colon \boldsymbol{H}^{div}(\Omega) \to H^{-1/2}(\partial\Omega), \tag{5.24}$$

where $H^{-1/2}(\partial\Omega)$ is the dual of $H^{1/2}(\partial\Omega)$. Furthermore, the trace operator in (5.24) is surjective.

Proof. See (Girault and Raviart, 1986). □

This lemma justifies the characterization of $\boldsymbol{H}_0^{\mathrm{div}}(\Omega) := \overline{\boldsymbol{C}_0^\infty(\Omega)}^{\|\cdot\|_{\mathrm{div},\Omega}}$ as the equivalent of $\boldsymbol{H}^{\mathrm{div}}(\Omega)$ with vanishing trace of the normal component, that is

$$\boldsymbol{H}_0^{\mathrm{div}}(\Omega) = \left\{ \boldsymbol{v} \in \boldsymbol{H}^{\mathrm{div}}(\Omega) \mid \boldsymbol{v} \cdot \boldsymbol{n}|_{\partial\Omega} = 0 \right\}, \tag{5.25}$$

where $\boldsymbol{v} \cdot \boldsymbol{n}|_{\partial\Omega}$ is an abuse of notation for $\mathrm{Tr}_{\boldsymbol{n}} \boldsymbol{v}$.

As velocity approximation space $\boldsymbol{V}_h$ we choose $\boldsymbol{H}^{\mathrm{div}}$-conforming finite elements, in particular, $\boldsymbol{V}_h \subset \boldsymbol{H}_0^{\mathrm{div}}(\Omega)$ since assuming no-slip conditions. Two prominent options are the Raviart-Thomas elements (RT) on quadrilateral meshes (Raviart and Thomas, 1977) (generalized by Nédélec to hexahedra (Nédélec, 1980)) or Brezzi-Douglas-Marini elements (BDM) on quadrilateral meshes (Brezzi, J. Douglas, and Marini, 1985) (generalized to hexahedra in (Brezzi, J. Douglas, Durán, et al., 1987; Nédélec, 1986)). The matching pressure approximations Q_h are discontinuous $\mathbb{Q}_k$-finite elements for RT and discontinuous $\mathbb{P}_k$-finite elements for BDM to obtain stable discretizations (Boffi et al., 2013), respectively. We choose

Raviart-Thomas elements due to the natural tensor structure of its matching pressure space,

$$\boldsymbol{V}_h = \left\{ \boldsymbol{v} \in \boldsymbol{H}^{\mathrm{div}}(\Omega) \mid \boldsymbol{v}|_K \in \mathrm{RT}_k \ \forall K \in \mathcal{T}_h, \quad \boldsymbol{v} \cdot \boldsymbol{n}|_{\partial\Omega} = 0 \right\}, \tag{5.26a}$$

$$Q_h = \left\{ q \in L^2(\Omega) \mid q|_K \in \mathbb{Q}_k \ \forall K \in \mathcal{T}_h, \quad \int_\Omega q \,\mathrm{d}\boldsymbol{x} = 0 \right\}. \tag{5.26b}$$

A detailed definition of Raviart-Thomas polynomials RT_k and the corresponding Raviart-Thomas finite element is postponed to Section 5.2.3, where we make excessive use of the definition of node functionals. In contrast to original works (Nédélec, 1980; Raviart and Thomas, 1977), we will explicitly highlight the anisotropic tensor product structure, that enables efficient operator application.

The attentive reader has noticed that $\boldsymbol{V}_h$ is a nonconforming approximation space of $\boldsymbol{V}$ (which consists of $\boldsymbol{H}^1(\Omega)$-regular velocities). To this end, interior penalties (Arnold, 1982) are used again obtaining consistency (and adjoint consistency).

Remark 5.2.3. The jump $[\![\boldsymbol{\sigma}]\!]$ and average $\{\boldsymbol{\sigma}\}$ of any tensor-field $\boldsymbol{\sigma}\colon \mathbb{R}^D \to \mathbb{R}^{D\times D}$ are component-wise defined as for scalar fields in Definition 3.1.1, i.e, interchanging v there with $\boldsymbol{\sigma}_{ij}$ here.

Assuming no-slip boundary conditions, i.e., $\boldsymbol{g} = 0$ on $\partial\Omega$, the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method for simplified Stokes flow reads: Find the velocity-pressure pair $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times Q_h$ such that

$$\widetilde{a}_{ip;h}(\boldsymbol{u}_h, \boldsymbol{v}) + b_h(\boldsymbol{v}, p_h) = \widetilde{F}_h(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h, \tag{5.27a}$$

$$b_h(\boldsymbol{u}_h, q) \qquad\qquad = 0 \qquad \forall q \in Q_h, \tag{5.27b}$$

where the bilinear forms are defined by

$$\begin{aligned} \widetilde{a}_{ip;h}(\boldsymbol{u}, \boldsymbol{v}) &= \int_{\mathcal{T}_h} \boldsymbol{\nabla}\boldsymbol{u} : \boldsymbol{\nabla}\boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \\ &\quad + \int_{\mathcal{E}_h} \left(\gamma_e \, [\![\boldsymbol{u} \otimes \boldsymbol{n}]\!] : [\![\boldsymbol{v} \otimes \boldsymbol{n}]\!] - \{\boldsymbol{\nabla}\boldsymbol{u}\} : [\![\boldsymbol{v} \otimes \boldsymbol{n}]\!] - [\![\boldsymbol{u} \otimes \boldsymbol{n}]\!] : \{\boldsymbol{\nabla}\boldsymbol{v}\} \right) \mathrm{d}\sigma(\boldsymbol{x}), \end{aligned} \tag{5.28a}$$

$$b_h(\boldsymbol{v}, q) = -\int_{\mathcal{T}_h} \nabla \cdot \boldsymbol{v} q \,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{E}_h} [\![\boldsymbol{v} \cdot \boldsymbol{n}]\!] \{q\} \,\mathrm{d}\boldsymbol{x}, \tag{5.28b}$$

and the right-hand side operator by

$$\widetilde{F}_h(\boldsymbol{v}) = \int_{\mathcal{T}_h} \boldsymbol{f} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \tag{5.29}$$

For vector fields $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{L}^2(\Omega)$ the identity

$$\int_{\mathcal{E}_h} \gamma_e \, [\![\boldsymbol{u} \otimes \boldsymbol{n}]\!] : [\![\boldsymbol{v} \otimes \boldsymbol{n}]\!] \,\mathrm{d}\sigma(\boldsymbol{x}) = \int_{\mathcal{E}_h} \gamma_e \, [\![\boldsymbol{u}]\!] \cdot [\![\boldsymbol{v}]\!] \,\mathrm{d}\sigma(\boldsymbol{x}) \tag{5.30}$$

simplifies the penalty term in (5.28a). The method's well-posedness and a priori estimates were studied in (Cockburn et al., 2006; Montlaur et al., 2008). In particular, the bilinear form $\widetilde{a}_{ip;h}(\cdot,\cdot)$ is continuous and coercive on $\boldsymbol{V}$, see (Hansbo and Larson, 2002). For Cartesian meshes of the simple domain $\Omega = [0,1]^2$, numerical experiments have shown that the same penalty factor as for the Poisson problem provides a stable discretization, see Section 3.1 for reference. Consequently, we obtain optimal error convergence in the energy as well as $\boldsymbol{L}^2(\Omega)$ norm.

Given the local normal vector $\boldsymbol{n}$, any vector field $\boldsymbol{v}\colon \mathbb{R}^D \to \mathbb{R}^D$ is decomposable into $\boldsymbol{v} = \boldsymbol{v}^\perp + \boldsymbol{v}^\parallel$, i.e., its normal vector field

$$\boldsymbol{v}^\perp = (\boldsymbol{v}\cdot\boldsymbol{n})\,\boldsymbol{n}, \tag{5.31}$$

and tangential vector field

$$\boldsymbol{v}^\parallel = \boldsymbol{v} - \boldsymbol{v}^\perp. \tag{5.32}$$

It would be more accurate to refer to (5.27) as symmetric interior penalty method since interior penalties are used concerning the complete vector field $\boldsymbol{u}$. However, from Lemma 5.2.1 it is known that only the tangential vector field $\boldsymbol{v}^\parallel$ for any $\boldsymbol{v} \in \boldsymbol{V}_h$ is discontinuous along interfaces. Thus, interior penalties for the tangential vector field suffice to handle the non-conformity of velocity approximations in $\boldsymbol{V}_h$ for the (simplified) Stokes problem. No-slip boundary conditions are implicitly and weakly imposed by the right-hand side operator (5.29). Note that $\boldsymbol{u}\cdot\boldsymbol{n} = 0$ is also strongly imposed by the definition of $\boldsymbol{V}_h$.

Given nonzero boundary data $\boldsymbol{g}$ we have to be more subtle when enforcing the boundary conditions for the normal and tangential vector field, see (Brezzi and Fortin, 1991; Cockburn et al., 2006). The boundary condition regarding the normal component may only be strongly imposed through the approximation space: usually, the degrees of freedom associated with the physical boundary are constrained subject to the $L^2(\partial\Omega)$-projection of $\boldsymbol{g}\cdot\boldsymbol{n}$. Furthermore, the tangential component $\boldsymbol{g}^\parallel$ may only be weakly imposed through interior penalties. Considering interior penalties only for the tangential vector fields, we guarantee both quite naturally. Then, the generic $\boldsymbol{H}^{\mathrm{div}}$ interior penalty formulation reads: Find the velocity-pressure pair $(\boldsymbol{u}_h, p_h) \in \boldsymbol{V}_h \times Q_h$ such that

$$\widetilde{a}_{div;h}(\boldsymbol{u}_h,\boldsymbol{v}) + b(\boldsymbol{v},p_h) = \widetilde{F}_h(\boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h, \tag{5.33a}$$

$$b(\boldsymbol{u}_h,q) = 0 \qquad \forall q \in Q_h, \tag{5.33b}$$

where the bilinear form regarding the Laplacian is defined by

$$\begin{aligned}\widetilde{a}_{div;h}(\boldsymbol{u},\boldsymbol{v}) = &\int_{\mathcal{T}_h} \boldsymbol{\nabla}\boldsymbol{u} : \boldsymbol{\nabla}\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\mathcal{E}_h} \left(\gamma_e \left[\!\left[\boldsymbol{u}^{\parallel}\right]\!\right] \cdot \left[\!\left[\boldsymbol{v}^{\parallel}\right]\!\right] - \left\{\boldsymbol{\nabla}\boldsymbol{u}^{\parallel}\right\} : \left[\!\left[\boldsymbol{v}^{\parallel} \otimes \boldsymbol{n}\right]\!\right] - \left[\!\left[\boldsymbol{u}^{\parallel} \otimes \boldsymbol{n}\right]\!\right] : \left\{\boldsymbol{\nabla}\boldsymbol{v}^{\parallel}\right\}\right) \mathrm{d}\sigma(\boldsymbol{x}),\end{aligned} \tag{5.34}$$

and the right-hand side operator by

$$\widetilde{F}_h(\boldsymbol{v}) = \int_{\mathcal{T}_h} \boldsymbol{f}\cdot\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{E}_h^{\partial}} \left(\gamma_e\, \boldsymbol{g}^{\parallel}\cdot\boldsymbol{v}^{\parallel} - \boldsymbol{g}^{\parallel}\otimes\boldsymbol{n} : \boldsymbol{\nabla}\boldsymbol{v}^{\parallel}\right) \mathrm{d}\sigma(\boldsymbol{x}). \tag{5.35}$$

The bilinear form $b(\cdot,\cdot)$ is defined in (5.11b). Given no-slip boundary conditions both formulations (5.27) and (5.33) are equal. In particular, $\widetilde{a}_{div;h}(\cdot,\cdot)$ and $\widetilde{a}_{ip;h}(\cdot,\cdot)$ are equal on $\boldsymbol{V}_h \subset \boldsymbol{H}^{\mathrm{div}}(\Omega)$. In practice, we prefer computing $\widetilde{a}_{ip;h}(\boldsymbol{u}_h,\boldsymbol{v})$ such that, among others, the "sophisticated" gradient $\boldsymbol{\nabla}\boldsymbol{u}^{\parallel}$ is avoided. If $\boldsymbol{\nabla}\boldsymbol{v}^{\parallel}$ needs to be computed (for example, due to nonzero boundary data), the identity

$$\boldsymbol{\nabla}\boldsymbol{v}^{\parallel}\,\boldsymbol{n} = \boldsymbol{\nabla}\boldsymbol{v}\,\boldsymbol{n} - (\boldsymbol{\nabla}\boldsymbol{v} : \boldsymbol{n}\otimes\boldsymbol{n})\boldsymbol{n}$$

is leveraged for Cartesian meshes.

5.2.2 Stream Function Formulation

Instead of solving the simplified Stokes problem (5.17) in its mixed formulation (5.21), it is also possible to solve the problem directly on the subspace of divergence-free functions, thus, enforcing the incompressibility condition as part of the solution space and not as part of the variational formulation (5.21b). The subspace of divergence-free velocities is defined by

$$\boldsymbol{V}^0 = \mathrm{Ker}(\nabla\cdot) = \left\{\boldsymbol{v}\in\boldsymbol{V} \mid \nabla\cdot\boldsymbol{v} = 0 \text{ a.e.}\right\}. \tag{5.36}$$

The weak form of the so-called *reduced problem* for the simplified Stokes equations reads: Find $\boldsymbol{u}\in\boldsymbol{V}^0$ such that

$$\widetilde{a}(\boldsymbol{u},\boldsymbol{v}) = F(\boldsymbol{v}) \qquad \forall\boldsymbol{v}\in\boldsymbol{V}^0, \tag{5.37}$$

where the bilinear form $\widetilde{a}(\cdot,\cdot)$ and linear operator F are defined in (5.12) and (5.20), respectively. The pressure is eliminated from (5.10a) due to the velocity subspace $\boldsymbol{V}^0$. Then, the challenge consists in constructing finite element approximations that satisfy the incompressibility condition in the strong sense given by (5.36).

In two spatial dimensions, there exists a curl operator for scalar as well as vector fields. The *scalar curl* of a vector field $\boldsymbol{v}$ is defined by

$$\boldsymbol{\nabla}\times\boldsymbol{v} = \partial_1\boldsymbol{v}_2 - \partial_2\boldsymbol{v}_1, \tag{5.38}$$

and the *vector curl* of a scalar field ψ by

$$\boldsymbol{\nabla} \times \psi = \begin{pmatrix} \partial_2 \psi \\ -\partial_1 \psi \end{pmatrix} \tag{5.39}$$

The latter is the gradient of ψ rotated clockwise by 90°. Similarly, the unit tangential vector $\boldsymbol{t}$ is obtained by rotating the unit normal vector $\boldsymbol{n}$ counterclockwise by 90°,

$$\boldsymbol{t} = \begin{pmatrix} -n_2 \\ n_1 \end{pmatrix}$$

In (Falk and Neilan, 2013) two *Stokes complexes* were studied, one being the *Hilbert cochain complex* defined by

$$\mathbb{R} \xrightarrow{\subset} H^2(\Omega) \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{H}^1(\Omega) \xrightarrow{\nabla\cdot} L^2(\Omega) \longrightarrow 0. \tag{5.40}$$

The attribute *complex* means that the composition of two consecutive mappings in (5.40) is zero, for instance, from vector calculus it is well-known that the divergence of a rotational vector field $\boldsymbol{\nabla} \times \psi$ is always zero. The Stokes complex is *exact* if Ω is simply connected (Girault and Raviart, 1986), i.e., the image of each map is the kernel of the succeeding linear mapping. From exactness of the cochain (5.40) follows that if $\boldsymbol{v} \in H^1(\Omega)$ with $\nabla \cdot \boldsymbol{v} = 0$, then $\boldsymbol{v} = \boldsymbol{\nabla} \times \psi$ for some stream function $\psi \in H^2(\Omega)$. The stream function space Ψ matching the velocity space $\boldsymbol{V}$ with no-slip boundary conditions reads

$$\Psi = \left\{ \psi \in H^2(\Omega) \mid \psi|_{\partial\Omega} = 0 \quad \text{and} \quad (\boldsymbol{\nabla}\psi \cdot \boldsymbol{n})|_{\partial\Omega} = 0 \right\} \tag{5.41}$$

In Section 4.1 the boundary conditions inherent in Ψ were called clamped boundary conditions. The expression *matching spaces* refers to the exactness of any (co)chain complex. The Stokes complex with boundary conditions reads

$$0 \xrightarrow{\subset} \Psi \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{V} \xrightarrow{\nabla\cdot} Q \longrightarrow 0. \tag{5.42}$$

Q is the space of mean value free pressure functions matching the velocities in $\boldsymbol{V}$ as discussed earlier. The Hodge decomposition, implied by sequence (5.42), characterizes the divergence-free velocities by

$$\boldsymbol{V}^0 = \boldsymbol{\nabla} \times \Psi, \tag{5.43}$$

if the physical domain Ω is simply connected. The finite element method introduced next will be applied ultimately on each vertex patch, thus, assuming Ω to be simply connected domain from now on is reasonable.

In Section 4.1 we discussed how challenging the implementation of $H^2(\Omega)$-conforming finite elements is. To this end, we follow (Kanschat and Sharma, 2014) in using $\boldsymbol{H}^{\mathrm{div}}(\Omega)$-conforming velocity approximations and $H^1(\Omega)$-conforming stream functions. Kanschat and Sharma have shown the equivalence of the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method and the C^0 interior penalty method from Section 4.1 for these finite elements. We emphasize that this allows the use of tensor product finite elements from preceeding sections. Let us elaborate on the equivalence of both methods, which facilitates introducing relevant notations. If Ω is simply connected, the L^2 de Rham complex,

$$\mathbb{R} \xrightarrow{\subset} H^1(\Omega) \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{H}^{\mathrm{div}}(\Omega) \xrightarrow{\nabla\cdot} L^2(\Omega) \longrightarrow 0, \tag{5.44}$$

is exact (Arnold, Falk, and Winther, 2006): if $\nabla\cdot\boldsymbol{v} = 0$ for any $\boldsymbol{v} \in \boldsymbol{H}^{\mathrm{div}}(\Omega)$, then $\boldsymbol{v} = \boldsymbol{\nabla}\times\psi$ for some $\psi \in H^1(\Omega)$. Compared to the exact Stokes complex (5.40), we are lacking $H^2(\Omega)$- and $\boldsymbol{H}^1(\Omega)$-regularity for stream functions and velocities, respectively. Consistency to the Stokes problem as well as the biharmonic problem is retained by means of interior penalties, respectively, as explained earlier. The exactness of the L^2 de Rham complex implies again a Hodge decomposition which justifies in analogy to (5.43) nonconforming divergence-free velocities as the vector curl of nonconforming stream functions in $H^1(\Omega)$.

Given a quadrilateral mesh $\mathcal{T}_h$, we define the approximation space of stream functions

$$\Psi_h = \left\{\psi \in H^1(\Omega) \mid \psi|_K \in \mathbb{Q}_{k+1}\ \forall K \in \mathcal{T}_h, \quad \psi|_{\partial\Omega} = 0\right\}, \tag{5.45}$$

which matches the velocity and pressure finite element space, $\boldsymbol{V}_h$ and Q_h, respectively, in the sense that an exact Hilbert cochain subcomplex of (5.44) is obtained,

$$0 \xrightarrow{\subset} \Psi_h \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{V}_h \xrightarrow{\nabla\cdot} Q_h \longrightarrow 0. \tag{5.46}$$

The interested reader finds more details in (Arnold, Boffi, and Bonizzoni, 2014). Those parts of the boundary condition missing in Ψ_h and $\boldsymbol{V}_h$ to obtain clamped and no-slip conditions, respectively, are weakly enforced by the C^0 interior penalty and $\boldsymbol{H}^{\mathrm{div}}$ interior penalty formulations, (4.9) and (5.33), respectively. The exact sequence of discrete spaces (and the fact that $\mathrm{Ker}(\nabla\cdot) \subset L^2(\Omega)$ is closed) implies a (discrete) Hodge decomposition.

Lemma 5.2.4 (Discrete Hodge decomposition)**.** *Let $\Omega \subset \mathbb{R}^2$ be simply connected. The space $\boldsymbol{V}_h$ admits a L^2-orthogonal decomposition with respect to the (discrete) divergence operator $\nabla\cdot\colon \boldsymbol{V}_h \to Q_h'$ into*

$$\boldsymbol{V}_h =: \boldsymbol{V}_h^0 \oplus \boldsymbol{V}_h^{\perp} \tag{5.47}$$

with characterization

$$\boldsymbol{V}_h^0 = \boldsymbol{\nabla}\times\Psi = \mathrm{Ker}(\nabla\cdot), \tag{5.48}$$

and its orthogonal complement

$$\boldsymbol{V}_h^{\perp} = \{\boldsymbol{v}_h \in \boldsymbol{V}_h \mid \exists q_h \in Q_h : \quad (\boldsymbol{w}, \boldsymbol{v}_h)_\Omega = -(\nabla \cdot \boldsymbol{w}, q_h)_\Omega \ \forall \boldsymbol{w} \in \boldsymbol{V}_h\}. \tag{5.49}$$

Proof. For a complete proof see (Girault and Raviart, 1986). We note that characterization (5.48) follows from the exactness of the discrete L^2 de Rham subcomplex (5.46) and (5.49) is implied by the closed range theorem. □

Given characterization (5.48) of divergence-free velocities, the *reduced* $\boldsymbol{H}^{\mathrm{div}}$ *interior penalty method* for simplified Stokes flow reads: Find a solution $\phi_h \in \Psi_h$ such that

$$\widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi_h, \boldsymbol{\nabla} \times \psi) = F_h(\boldsymbol{\nabla} \times \psi) \qquad \forall \psi \in \Psi_h. \tag{5.50}$$

The bilinear form $\widetilde{a}_{div;h}(\cdot,\cdot)$ and the right-hand side operator $\widetilde{F}_h$ are defined in (5.34) and (5.35), respectively, with $\boldsymbol{g} \equiv 0$. In (Kanschat and Sharma, 2014) the equivalence

$$a_{c0ip;h}(\phi, \psi) \equiv \widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi, \boldsymbol{\nabla} \times \psi) \tag{5.51}$$

for any stream functions ϕ and ψ was shown. There is a single missing piece before constructing local solvers subject to the stream function formalism: the local pressure functions need to be computed as well.

5.2.3 Post-processing Pressure

We want to develop Schwarz smoothers which solve the reduced model problem (5.37) locally by utilizing the C^0 interior penalty method (5.50). For each level ℓ of our geometric multigrid preconditioner the $\boldsymbol{H}^{\mathrm{div}}$-conforming interior penalty method (5.33) is used, defining the finite element discretization matrix A_ℓ from Algorithm 2. Therefore, subspace corrections need to be added to both, the velocity solution $\boldsymbol{u}_\ell$ and pressure solution p_ℓ. To this end, given the local velocity solution $\boldsymbol{u}_{\ell;j} = \boldsymbol{\nabla} \times \phi_{\ell;j}$ the local pressure $p_{\ell;j}$ needs to be computed as well. For the assumptions of the previous section, Kanschat and Sharma (2014) have shown that the velocity and pressure solution of the simplified Stokes equations can be computed decoupled: while the adjoint of the divergence operator $\nabla^*_{a_h}$, introduced in (Kanschat and Sharma, 2014, Corollary 3.3.), is intuitive in its theoretical nature, its numerical computation is not straightforward. To this end, we generalize the concepts of (Caussignac, 1987) to reconstruct the pressure $p_{\ell;j}$ from the stream function solution $\phi_{\ell;j}$. Caussignac solved a stream function-vorticity problem to compute the velocity-pressure pair of a simplified Stokes flow. The pressure approximate was obtained by post-processing the known vorticity approximate. Caussignac's post-processing relies on defining Raviart-Thomas elements subject to monomial bases. We extend the concept to compute discrete pressure solutions from stream function

approximates. In addition, for reasons of numerical stability[1], we generalize the pressure post-processing to Raviart-Thomas elements with arbitrary ansatz polynomials, for both the shape function basis and the generation of node functionals.

We elaborate on Raviart-Thomas elements to construct a basis for the L^2-orthogonal complement $\boldsymbol{V}_h^\perp$ from Lemma 5.2.4, that is essential for the pressure post-processing. The space of anisotropic tensor product polynomials $\mathbb{Q}_{k_1,k_2,\ldots,k_D}$ was defined in in Definition 2.2.1. From previous chapters, we are familiar with the isotropic tensor product space $\mathbb{Q}_k$, which abbreviates the special case $\mathbb{Q}_{k,\ldots,k}$ in notation. We simplify notation of the product space $\mathbb{Q}_k \times \mathbb{Q}_k$ by writing $[\mathbb{Q}_k]^2$ and, similarly, $\bigtimes_{n=1}^N \mathbb{Q}_k$ by writing $[\mathbb{Q}_k]^N$ for N vector components.

Definition 5.2.5. The *Raviart-Thomas element* of degree k on the unit hypercube $\hat{K} = [0,1]^D$ is the triplet $(\hat{K}, \mathrm{RT}_k, \Sigma_{RT})$. The shape function space is given by

$$\mathrm{RT}_k = \mathbb{Q}_{k+1,\ldots,k} \times \cdots \times \mathbb{Q}_{k,\ldots,k+1} \tag{5.52}$$

The set of node functionals is

$$\Sigma_{RT} = \left\{\hat{\mathcal{N}}_{\boldsymbol{j}}^\circ\right\}_{\boldsymbol{j}=1,\ldots,N_{dof}^\circ} \cup \left\{\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^\partial\right\}_{j_1=1,\ldots,2^D;\boldsymbol{j}_2=1,\ldots,N_{dof}^e} \tag{5.53}$$

with

$$\hat{\mathcal{N}}_{\boldsymbol{j}}^\circ(\hat{\boldsymbol{v}}) = \int_{\hat{K}} \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{w}}_{\boldsymbol{j}}\, d\hat{\boldsymbol{x}}, \qquad \hat{\boldsymbol{w}}_{\boldsymbol{j}} \in \mathbb{Q}_{k-1,k,\ldots,k} \times \cdots \times \mathbb{Q}_{k,\ldots,k,k-1}, \tag{5.54a}$$

$$\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^\partial(\hat{\boldsymbol{v}}) = \int_{\hat{e}_{j_1}} \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{n}} \hat{q}_{\boldsymbol{j}_2}\, \mathrm{d}\sigma(\hat{\boldsymbol{x}}), \qquad \hat{q}_{\boldsymbol{j}_2} \in \mathbb{Q}_k(\hat{e}_{j_1}), \quad \hat{e}_{j_1} \subset \partial\hat{K}. \tag{5.54b}$$

The number of interior degrees of freedom N_{dof}° equals $D(k+1)k^{D-1}$ and the number of degrees of freedom per facet N_{dof}^e is $(k+1)^{D-1}$. The unit normal $\hat{\boldsymbol{n}}$ in (5.54b) points always in positive direction.

We will use the notation RT_k interchangeably for both, the polynomial space (5.52) and as short form for Raviart-Thomas elements of degree k. The compact notation of the tensor product space (5.52) refers to

$$\mathbb{Q}_{k+1,k} \times \mathbb{Q}_{k,k+1}$$

in two dimensions, and

$$\mathbb{Q}_{k+1,k,k} \times \mathbb{Q}_{k,k+1,k} \times \mathbb{Q}_{k,k,k+1}$$

in three dimensions.

Lemma 5.2.6. *For Raviart-Thomas elements of degree k there holds*

$$\nabla \cdot \mathrm{RT}_k = \mathbb{Q}_k, \tag{5.55}$$

[1]Numerically stable Legendre and Lagrange polynomials are used in computations.

and for each $e \subset \partial\hat{K}$ *and each* $\boldsymbol{v} \in \mathrm{RT}_k$ *there holds*

$$\boldsymbol{v} \cdot \boldsymbol{n}_e \in \mathbb{Q}_k. \tag{5.56}$$

Proof. See (Boffi et al., 2013). □

Assuming no-slip conditions, the incompressibility condition (5.1b) and (5.55) imply that weakly divergence-free velocities in $\boldsymbol{V}_h$ are also pointwise divergence-free, see (Cockburn et al., 2006). In view of Lemma 5.2.1, Raviart-Thomas elements are $\boldsymbol{H}^{\mathrm{div}}$-conforming if the degrees of freedom (5.54b) associated with interfaces are single-valued. Consequently, their normal component is continuous along interfaces.

For arbitrarily shaped cells a divergence-preserving mapping is needed. The *Piola transform* maps a vector field $\hat{\boldsymbol{v}}$ under the mapping $\boldsymbol{F}_K \colon \hat{K} \to K$ to a vector field

$$\boldsymbol{v} = \det(\hat{J}_K)^{-1} \hat{J}_K \hat{\boldsymbol{v}}, \tag{5.57}$$

where $\hat{J}_K$ denotes the Jacobian matrix of $\boldsymbol{F}_K$. On affine meshes the divergence is preserved pointwise up to a factor under the Piola transform ,

$$\nabla \cdot \boldsymbol{v}(\boldsymbol{x}) = \det(\hat{J}_K)^{-1} \hat{\nabla} \cdot \hat{\boldsymbol{v}}(\hat{\boldsymbol{x}}), \qquad \boldsymbol{x} = \boldsymbol{F}_K(\hat{\boldsymbol{x}}), \quad \hat{\boldsymbol{x}} \in \hat{K}. \tag{5.58}$$

As a consequence there holds:

Lemma 5.2.7. *Let* q *and* $\hat{q}$ *be scalar fields related by* $q \circ \boldsymbol{F}_K = \hat{q}$*, and* $\boldsymbol{v}$ *the Piola transform of* $\hat{\boldsymbol{v}}$ *under* $\boldsymbol{F}_K$*, then, there holds*

$$\int_K \boldsymbol{v} \cdot \boldsymbol{\nabla} q \,\mathrm{d}\boldsymbol{x} = \int_{\hat{K}} \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{\nabla}} \hat{q} \, d\hat{\boldsymbol{x}}, \tag{5.59a}$$

$$\int_K \nabla \cdot \boldsymbol{v} q \,\mathrm{d}\boldsymbol{x} = \int_{\hat{K}} \hat{\nabla} \cdot \hat{\boldsymbol{v}} \hat{q} \, d\hat{\boldsymbol{x}}, \tag{5.59b}$$

$$\int_{\partial K} \boldsymbol{v} \cdot \boldsymbol{n} q \,\mathrm{d}\boldsymbol{x} = \int_{\partial \hat{K}} \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{n}} \hat{q} \, d\hat{\boldsymbol{x}}. \tag{5.59c}$$

The Lemma 5.2.7 implies that the definition of $\boldsymbol{H}^{\mathrm{div}}$-conforming finite elements, in particular, Raviart-Thomas elements, is sufficient on the unit hypercube $\hat{K}$ as the conformity is preserved under Piola transforms for any cell $K \in \mathcal{T}_h$. There are some caveats for non-affine meshes, where $\det(\hat{J}_K)$ in (5.58) is not constant, such that the approximation quality may suffer (Arnold, Boffi, and Falk, 2005).

We conclude by highlighting the (anisotropic) tensor structure of Raviart-Thomas elements, for simplicity only in two spatial dimensions. The unit square $\hat{K} = [0,1]^2$ provides a natural separation of coordinates. Basis scalar fields and components of vector fields used in Definition 5.2.5 possess a separation of variables. Basis function indices $\boldsymbol{j}$ or $\boldsymbol{j}_2$ in (5.54a) and (5.54b) are subject to the lexicographic ordering from Definition 2.1.6. To be precise,

the lexicographic ordering holds for each vector component of $\hat{\boldsymbol{w}}_{\boldsymbol{j}}$ only and varies depending on its anisotropic structure. To this end, a primitive basis of $\mathbb{Q}_{k-1,k} \times \mathbb{Q}_{k,k-1}$ is assumed, i.e., basis functions associated with the first component are enumerated first,

$$\hat{\boldsymbol{w}}_{\boldsymbol{j}} = \begin{pmatrix} \hat{w}_{d;\boldsymbol{j}} \\ 0 \end{pmatrix}, \quad \boldsymbol{j} = 1, \dots, N^{\circ}_{dof}/2, \tag{5.60}$$

and those associated with the second component afterward,

$$\hat{\boldsymbol{w}}_{\boldsymbol{j}+N^{\circ}_{dof}/2} = \begin{pmatrix} 0 \\ \hat{w}_{d;\boldsymbol{j}} \end{pmatrix}, \quad \boldsymbol{j} = 1, \dots, N^{\circ}_{dof}/2, \tag{5.61}$$

where

$$\left\{ \hat{w}_{d;1}, \dots, \hat{w}_{d;N^{\circ}_{dof}/2} \right\}$$

is a basis of $\mathbb{Q}_{k-1,k}$ for $d = 1$ and a basis of $\mathbb{Q}_{k,k-1}$ for $d = 2$, respectively. We define a first set of moment-based functionals on the unit interval,

$$\hat{\mathcal{N}}^{1D}_{j}(\hat{\phi}) = \int_0^1 \hat{\phi}\, \hat{q}_j \, d\hat{x}, \quad \hat{q}_j \in \mathcal{P}_k([0,1]), \quad j = 1, \dots, k+1, \tag{5.62}$$

which reminds us of the Legendre element from Definition 2.2.5. We define another set of degrees of freedom on the unit interval consisting of both, node value functionals and moments,

$$\hat{\mathcal{M}}^{1D}_{1}(\hat{\phi}) = \hat{\phi}(0), \tag{5.63a}$$

$$\hat{\mathcal{M}}^{1D}_{k+2}(\hat{\phi}) = \hat{\phi}(1), \tag{5.63b}$$

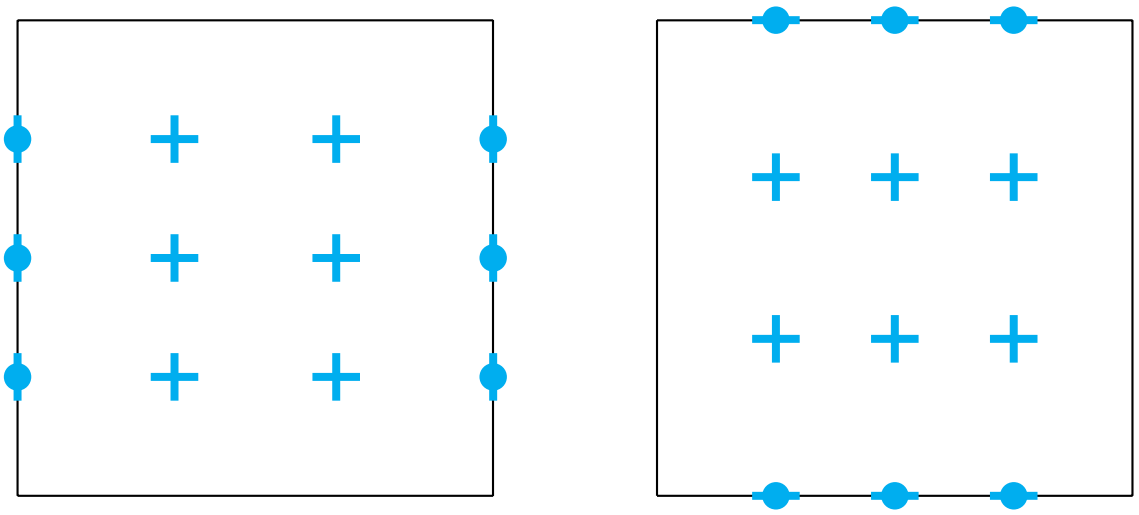
$$\hat{\mathcal{M}}^{1D}_{j}(\hat{\phi}) = \int_0^1 \hat{\phi}\, \hat{q}_{j-1} \, d\hat{x}, \quad \hat{q}_{j-1} \in \mathcal{P}_{k-1}([0,1]), \quad j = 2, \dots, k+1. \tag{5.63c}$$

Let us assume $\hat{e}_{j_1} = \{0\} \times [0,1]$ for $j_1 = 1$ such that $\hat{\boldsymbol{n}} = (1,0)^{\mathsf{T}}$: the Raviart-Thomas functionals associated with $\hat{e}_1$ decompose into

$$\hat{\mathcal{N}}^{\partial}_{0,j_2}(\hat{\boldsymbol{v}}) = \hat{v}^{(1)}_1(0) \int_0^1 \hat{v}^{(2)}_1 \hat{q}_{j_2} \, d\hat{x}_2 = \hat{\mathcal{M}}^{1D}_1 \otimes \hat{\mathcal{N}}^{1D}_{j_2}(\hat{v}^{(1)}_1 \otimes \hat{v}^{(2)}_1), \tag{5.64}$$

for $\hat{q}_{j_2} \in \mathbb{Q}_k(\hat{e}_1) = \mathcal{P}_k([0,1]), j_2 = 1, \dots, k+1$. For the opposite edge $\hat{e}_{j_1} = \{1\} \times [0,1]$ only $\hat{\mathcal{M}}^{1D}_1$ needs to be replaced by $\hat{\mathcal{M}}^{1D}_{k+2}$. Similarly, Raviart-Thomas node functionals with nonzero first component of $\hat{\boldsymbol{w}}_{\boldsymbol{j}}$ decompose into

$$\hat{\mathcal{N}}^{\circ}_{\boldsymbol{j}}(\hat{\boldsymbol{v}}) = \int_0^1 \hat{v}^{(1)}_1 \hat{w}^{(1)}_{1;j_1} \, d\hat{x}_1 \int_0^1 \hat{v}^{(2)}_1 \hat{w}^{(2)}_{1;j_2} \, d\hat{x}_2 = \hat{\mathcal{M}}^{1D}_{j_1+1} \otimes \hat{\mathcal{N}}^{1D}_{j_2}(\hat{v}^{(1)}_1 \otimes \hat{v}^{(2)}_1), \tag{5.65}$$



(a) $\hat{\mathcal{M}}_{j_1}^{1D} \otimes \hat{\mathcal{N}}_{j_2}^{1D}$ (first component) (b) $\hat{\mathcal{N}}_{j_1}^{1D} \otimes \hat{\mathcal{M}}_{j_2}^{1D}$ (second component)

Fig. 5.1 Illustrating degrees of freedom of the RT_2 element in two spatial dimensions. The tensor product of node functionals on the left- or right-hand side determine basis functions with nonzero first (i.e., $\hat{\boldsymbol{v}}_1$) or nonzero second component (i.e., $\hat{\boldsymbol{v}}_2$), respectively. One-dimensional moment-based functionals are illustrated through ▬ (the long edge of ▬ highlights the integration variable (in this case x_1)), node value functionals are illustrated by ●.

for $\hat{w}_{1;\boldsymbol{j}} \in \mathbb{Q}_{k-1,k}, j_1 = 1, \dots, k, j_2 = 1, \dots, k+1$ with index notation $\boldsymbol{j} = (j_1, j_2)$ in short form. The first vector component of primitive basis functions in RT_k is uniquely determined by the tensor product of functionals. In analogy, decompositions for functionals associated with nonzero second component of $\hat{\boldsymbol{w}}_{\boldsymbol{j}}$ and edges aligned with the second coordinate uniquely determine the remaining basis functions in RT_k. The degrees of freedom are schematically visualized in Figure 5.1. If we follow the enumeration of functionals $\hat{\mathcal{M}}_j^{1D}$ a lexicographic ordering for each vector component is obtained. Therefore, efficient operator applications using Raviart-Thomas elements are possible, see Section 5.4 for more details.

Equipped with the knowledge on the structure of Raviart-Thomas elements, we can tackle the original problem of this section: computing the pressure solution $p_h \in Q_h$ from the stream function $\phi_h \in \Psi_h$ that solves the reduced $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method (5.50). Testing (5.33) with the curl of discrete stream functions, the pressure solution p_h is eliminated. To this end, we seek the pressure solution p_h such that

$$-\int_\Omega \nabla \cdot \boldsymbol{v} p_h \,\mathrm{d}\boldsymbol{x} = F_h(\boldsymbol{v}) - \widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi_h, \boldsymbol{v}), \tag{5.66}$$

for all $\boldsymbol{v} \in \boldsymbol{V}_h$ with $\nabla \cdot \boldsymbol{v} \neq 0$. In view of the discrete Hodge decomposition from Lemma 5.2.4, the test space contains the L^2-orthogonal velocities in $\boldsymbol{V}_h^\perp$, i.e., orthogonal with respect to the kernel of the divergence operator). Introducing the discrete gradient $\boldsymbol{\nabla}_h \colon Q_h \to \boldsymbol{V}_h$ which

is implicitly defined in (5.49) as adjoint of the divergence,

$$(\boldsymbol{\nabla}_h q_h, \boldsymbol{v})_\Omega = -(q_h, \nabla \cdot \boldsymbol{v})_\Omega, \quad \forall \boldsymbol{v} \in \boldsymbol{V}_h, \tag{5.67}$$

the discrete Hodge decomposition from Lemma 5.2.4 reads

$$\boldsymbol{V}_h = \boldsymbol{\nabla}_h Q_h \oplus \boldsymbol{\nabla} \times \Psi_h, \tag{5.68}$$

described in detail in (Arnold, Falk, and Winther, 1997; Arnold, Falk, and Winther, 2000; Brezzi, Fortin, and Stenberg, 1991). In the context of finite elements, the duality of the adjoint gradient is characterized by the definition of node functionals. To this end, we will use a polynomial basis of $\boldsymbol{\nabla}\mathbb{Q}_k$ generating a subset of those Raviart-Thomas node functionals in (5.54a) and (5.54b) to construct a shape function basis of $\boldsymbol{V}_h^\perp$.

For brevity, the notation $\mathbb{Q}_k$ is used interchangeably for $\mathbb{Q}_k(\mathbb{R}^D), \mathbb{Q}_k(\hat{K})$ and $\mathbb{Q}_k(K)$, that is, the space of tensor product polynomials of degree up to k on $\mathbb{R}^D$, on the unit cell $\hat{K}$ or on any cell K, respectively. The same applies to other polynomial spaces, for instance, RT_k. The tensor product polynomial space of degree $k \geq 0$ admits a decomposition

$$\mathbb{Q}_k = \mathbb{Q}_0 \oplus \mathbb{Q}_k^\perp, \tag{5.69}$$

where $\mathbb{Q}_k^\perp$ is well-defined as L^2-orthogonal complement of $\mathbb{Q}_0 \subset \mathbb{Q}_k$. Thus, any pressure $p_h \in Q_h$ is decomposed into

$$p_h = \sum_{K \in \mathcal{T}_h} p_K^0 + p_K^\perp, \tag{5.70}$$

with both $p_K^0 \in \mathbb{Q}_0(K)$ and $p_K^\perp \in \mathbb{Q}_k^\perp(K)$ being extended by zero to the whole domain Ω. In computations, we make use of Legendre polynomials which inherently admit the decomposition (5.69), see Definition 2.2.5.

First, we define node functionals on the unit cell $\hat{K} = [0,1]^2$,

$$\hat{\mathcal{N}}_{\boldsymbol{j}}^{\perp\circ}(\hat{\boldsymbol{v}}) = \int_{\hat{K}} \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{\nabla}} \hat{p}_{\boldsymbol{j}}^\perp \, d\hat{\boldsymbol{x}}, \quad \hat{p}_{\boldsymbol{j}}^\perp \in \mathbb{Q}_k^\perp, \quad \boldsymbol{j} = 1, \ldots, N_{dof}^{\perp\circ}, \tag{5.71a}$$

$$\hat{\mathcal{N}}_j^{\perp\partial}(\hat{\boldsymbol{v}}) = \int_{\hat{K}} \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{n}} \, \mathrm{d}\sigma(\hat{\boldsymbol{x}}), \quad \hat{e}_j \subset \partial\hat{K}. \quad j = 1, \ldots, 4, \tag{5.71b}$$

We note that the unit normal vector $\hat{\boldsymbol{n}}$ points either in positive x- or y-direction, thus, omitting the facet index j. Furthermore, $N_{dof}^{\perp\circ} = (k+1)^2 - 1$ and $\hat{p}_{\boldsymbol{j}}^\perp$ denotes a basis function of $\mathbb{Q}_k^\perp$. Since

$$\boldsymbol{\nabla}\mathbb{Q}_k^\perp \subset \mathbb{Q}_{k-1,k} \times \mathbb{Q}_{k,k-1},$$

the span of node functionals $\hat{\mathcal{N}}_{\boldsymbol{j}}^{\perp\circ}$ is a subspace of the span of Raviart-Thomas functionals $\hat{\mathcal{N}}_{\boldsymbol{j}}^\circ$. Similarly, the span of $\hat{\mathcal{N}}_j^{\perp\partial}$ is a subspace of the span of $\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^\partial$. Note that we implicitly used a constant-valued test polynomial equal to one in (5.71b), coinciding with $\hat{q}_{\boldsymbol{j}_2} \equiv 1$ in (5.54b).

From here on, only technical arguments are missing for constructing a shape function basis of $\boldsymbol{V}_h^{\perp}$, which is postponed to Section B.1, consequently.

Integrating by parts and applying boundary conditions from $\boldsymbol{V}_h$, we obtain

$$-\int_{\Omega} \nabla \cdot \boldsymbol{v} p_h \,\mathrm{d}\boldsymbol{x} = -\int_{\mathcal{E}_h^{\circ}} \boldsymbol{v} \cdot [\![p_h \boldsymbol{n}]\!] \,\mathrm{d}\sigma(\boldsymbol{x}) + \int_{\mathcal{T}_h} \boldsymbol{v} \cdot \boldsymbol{\nabla} p_h \,\mathrm{d}\boldsymbol{x} \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_h^{\perp}. \tag{5.72}$$

In view of Lemma 5.2.1, $\boldsymbol{v} \cdot [\![p_h \boldsymbol{n}]\!]$ is well-defined because the normal component of $\boldsymbol{H}^{\mathrm{div}}$-conforming vector fields is continuous along interfaces. We notice that piecewise constant pressure functions affect only integrals over interfaces. Moreover, the right-hand side of (5.72) consists of the two kinds of moment-based functionals (5.71). It is straightforward to derive a two-step post-processing similar: first, the piecewise non-constant contributions $p_K^{\perp}$ are computed and afterward the piecewise constants p_K^0.

Lemma 5.2.8. *Let $p_h \in Q_h$ admit decomposition (5.70) with*

$$p_K^{\perp} = \sum_{\boldsymbol{i}=1}^{N_{dof}^{\perp\circ}} \alpha_{\boldsymbol{i}}^K (\hat{p}_{\boldsymbol{i}}^{\perp} \circ \boldsymbol{F}_K^{-1}).$$

and solve equation (5.66). Let $\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}$ be basis functions of $\boldsymbol{V}_h^{\perp}$ as defined in Lemma B.1.2. Then, it holds

$$\alpha_{\boldsymbol{i}}^K = F_h(\boldsymbol{v}_{K;\boldsymbol{i}}^{\circ}) - \widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi_h, \boldsymbol{v}_{K;\boldsymbol{i}}^{\circ}), \tag{5.73}$$

for any $\boldsymbol{i} = 1, \ldots, N_{dof}^{\perp\circ}$, such that p_h is determined up to piecewise constants.

Lemma 5.2.9. *Let $p_h \in Q_h$ admit decomposition (5.70) and solve equation (5.66). Let $\boldsymbol{v}_e^{\perp\partial}$ be basis functions of $\boldsymbol{V}_h^{\perp}$ as defined in Lemma B.1.2. Then, it holds*

$$p_{K^+}^0 - p_{K^-}^0 = -F_h(\boldsymbol{v}_e^{\perp\partial}) + \widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi_h, \boldsymbol{v}_e^{\perp\partial}) - \int_e \boldsymbol{v}_e^{\perp\partial} \cdot (p_{K^+}^{\perp} \boldsymbol{n}^+ + p_{K^-}^{\perp} \boldsymbol{n}^-) \,\mathrm{d}\sigma(\boldsymbol{x}), \tag{5.74}$$

for any interface $e \in \mathcal{E}_h^{\circ}$ with $e = K^+ \cap K^-$.

From Lemma 5.2.8 and Lemma 5.2.9 the algorithmic procedure for computing the discrete pressure solution p_h of (5.66) in essentially two steps becomes apparent:

Remark 5.2.10 (Post-processing pressure). Given the stream function solution $\phi_h \in \Psi_h$ of (5.50), the pressure solution $p_h \in Q_h$ of (5.66), admitting the decomposition (5.70) based on Legendre polynomials, is computed by following steps:

(S1) For each cell $K \in \mathcal{T}_h$ compute $p_K^{\perp}$ as described in Lemma 5.2.8.

(S2) Choose a single cell $K_0 \in \mathcal{T}_h$ to determine the subset of interfaces $\tilde{\mathcal{E}}_h^{\circ}$ as described in Remark B.1.1, and set $p_{K_0}^0 = 0$. Solve the system of equations given by (5.74), with as many equations as interfaces in $\tilde{\mathcal{E}}_h^{\circ}$, which determines p_K^0 for each cell $K \in \mathcal{T}_h$.

(S3) Reset $p^0_{K_0}$ such that the mean value condition $\int_\Omega p_h \,\mathrm{d}\boldsymbol{x} = 0$ is satisfied.

Utilizing piecewise Legendre polynomials, we emphasize that a single constant p^0_K may be used to satisfy the mean value condition of Q_h. Thus, computing the remaining constants p^0_K in Step (S2) requires only as many interfaces as in $\tilde{\mathcal{E}}^\circ_h \subset \mathcal{E}^\circ_h$. The proofs of Lemmas 5.2.8 and 5.2.9 are found in Section B.1.

5.2.4 Mathematical Efficiency of Schwarz Smoothers

We compare now two classes of algorithms computing the Schwarz smoothers on vertex patches from (Kanschat and Mao, 2015) in different ways. They are used to compute geometric multigrid preconditioners for the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method (5.33). In view of Sections 5.2.2 and 5.2.3, we emphasize that both classes of algorithms lead to mathematically equivalent Schwarz smoothers.

HdivSC Schwarz smoothing steps computing exact local solvers based on local $\boldsymbol{H}^{\mathrm{div}}$ interior penalty discretizations

SFSC Schwarz smoothing steps based on exact local solvers that compute local velocities through the stream function formulation and post-process local pressure solutions

The first class of algorithms, called (HdivSC), relies on **S**ubspace **C**orrections based on local $\boldsymbol{H}^{\mathrm{div}}$ interior penalty discretizations, that were studied first in (Kanschat and Mao, 2015). The main difference of the second class, referred to as (SFSC), is computing local **S**tream **F**unction solutions based on an equivalent C^0 interior penalty method. Post-processing the local stream functions leads to local velocity-pressure **S**ubspace **C**orrections. The (algebraic) identity of the DG discretizations was studied in (Kanschat and Sharma, 2014), but to the best of our knowledge the class of smoothing algorithms (SFSC) has not been studied before. In this section, we focus on the mathematical efficiency of Schwarz smoothers for simplified Stokes flow (5.17). Cost-efficient algorithms will be discussed in Section 5.2.5. Conducting numerical experiments, we want to confirm that our implementations of both classes of algorithms result in identical results in mathematical terms. Both classes will lead to numerical solvers with uniform convergence concerning the mesh size and polynomial degree, needing only very few iterations to converge, in particular, for high-order finite elements. We stress that the multilevel forward operators A_ℓ are subject to $\boldsymbol{H}^{\mathrm{div}}$ interior penalty discretizations, i.e., stream functions are computed only locally when using (SFSC). The details will follow.

Let the domain Ω again be the unit square $[0,1]^2$. We follow Section 3.1.1 in defining a hierarchy of uniform Cartesian meshes $\mathcal{T}_\ell, \ell = 1,\dots,L$, starting from a single vertex patch $\mathcal{T}_1$ consisting of four congruent cells. The $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method with no-slip boundaries is solved on each level. To this end, nested approximation spaces $\boldsymbol{V}_\ell := \boldsymbol{V}_{h_\ell}$

and $Q_\ell := Q_{h_\ell}$ are defined in (5.26) by substituting $\mathcal{T}_h$ with $\mathcal{T}_\ell$. Accordingly, bilinear forms $a_\ell(\cdot,\cdot) := \widetilde{a}_{div;h_\ell}(\cdot,\cdot)$ are defined, i.e., replacing h by h_ℓ in (5.34). The divergence-related bilinear form $b(\cdot,\cdot)$ from (5.11b) is independent of the mesh width and, thus, simply restricted to the finite element spaces $\boldsymbol{V}_\ell$ and Q_ℓ. We use standard level transfers, which were introduced in Section 2.3.1.

Defining local solvers based on the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method is analogous to the definitions of local interior penalty solvers for the Poisson or biharmonic problem. Let $\mathcal{T}_{\ell;j} \subset \mathcal{T}_\ell$ denote the vertex patch of subdomain Ω_j as introduced in Section 2.3.2. From now on, we suppress the level index ℓ whenever a subdomain index j is present simultaneously. For instance, we write $\mathcal{T}_j$ instead of $\mathcal{T}_{\ell;j}$. For each vertex patch the simplified Stokes problem with no-slip boundary conditions is solved locally by means of the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method. The local velocity and pressure subspaces consist of those piecewise polynomials in $\boldsymbol{V}_\ell$ and Q_ℓ, respectively, which have only support on the respective subdomain Ω_j,

$$\boldsymbol{V}_{\ell;j} = \left\{ \boldsymbol{v} \in \boldsymbol{H}^{\mathrm{div}}(\Omega) \mid \boldsymbol{v}|_K \in \mathrm{RT}_k \ \forall K \in \mathcal{T}_{\ell;j}, \quad \boldsymbol{v}\cdot\boldsymbol{n}|_{\partial\Omega_j} = 0 \right\}, \tag{5.75a}$$

$$Q_{\ell;j} = \left\{ q \in L^2(\Omega) \mid q|_K \in \mathbb{Q}_k \ \forall K \in \mathcal{T}_{\ell;j}, \quad \int_{\Omega_j} q \,\mathrm{d}\boldsymbol{x} = 0 \right\}. \tag{5.75b}$$

We continue defining the subspace corrections for the first class of algorithms (HdivSC). The local bilinear form $\widetilde{a}_{div;j}(\boldsymbol{u},\boldsymbol{v})$ is defined by replacing $\mathcal{T}_h$ with $\mathcal{T}_j$ in (5.34) for all $\boldsymbol{u},\boldsymbol{v} \in \boldsymbol{V}_j$. The divergence bilinear form $b(\cdot,\cdot)$ from (5.11b) is simply restricted to $\boldsymbol{V}_j$ and Q_j. Then, the local solver is given by the weak formulation that reads: Find a solution $(\boldsymbol{u}_j, p_j) \in \boldsymbol{V}_j \times Q_j$ such that

$$\widetilde{a}_{div;j}(\boldsymbol{u}_j,\boldsymbol{v}) + b(\boldsymbol{v},p_j) = \int_{\Omega_j} \boldsymbol{f}\cdot\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_j, \tag{5.76a}$$

$$b(\boldsymbol{u}_j, q) = 0 \qquad \forall q \in Q_j. \tag{5.76b}$$

We emphasize that the right-hand side in (5.76a) implicitly imposes $\boldsymbol{u}_j^{\parallel} = 0$ on $\partial\Omega_j$. However, since $\widetilde{a}_{div;j}(\cdot,\cdot)$ is simply the restriction of $\widetilde{a}_{div;h_\ell}(\cdot,\cdot)$ to local approximation spaces, $\boldsymbol{u}_j^{\parallel} = 0$ is weakly imposed with different penalty factors for local facets at the physical boundary $\partial\Omega \cap \partial\Omega_j$ and facets in the domain's interior $\Omega \cap \partial\Omega_j$, respectively. In particular, at $\Omega \cap \partial\Omega_j$ we obtain half a Dirichlet-type and half a Neumann-type condition regarding the tangential component $\boldsymbol{u}_j^{\parallel}$, which was elaborated in Section 3.3 for the Poisson problem. Consequently, there are nearly but not exactly no-slip boundary conditions imposed for each local problem (5.76). The condition $\boldsymbol{u}_j \cdot \boldsymbol{n} = 0$ on $\partial\Omega_j$ is always imposed strongly through $\boldsymbol{V}_{\ell;j}$ on every part of the subdomain's boundary $\partial\Omega_j$. We use standard transfer operators between $\boldsymbol{V}_j \times Q_j$ and $\boldsymbol{V}_\ell \times Q_\ell$, see Section 2.3.2 for details. To be executed in parallel, the multiplicative

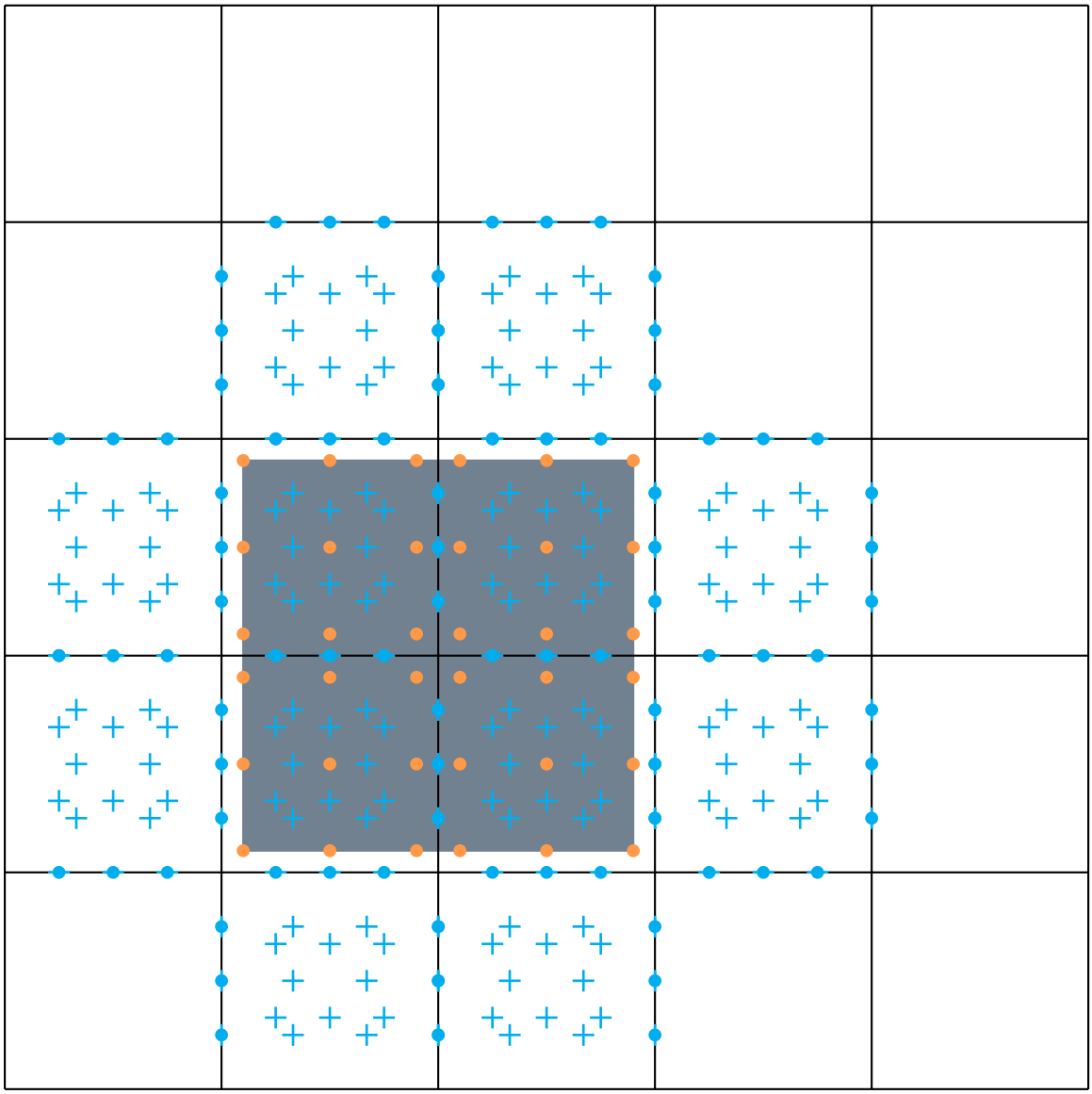


Fig. 5.2 The vertex patch stencil for $\boldsymbol{H}^{\mathrm{div}}$-IP discretizations schematically shows the action of $A_\ell R_j^{\mathsf{T}}$ given velocity-pressure ansatz polynomials in $\mathrm{RT}_2 \times \mathbb{Q}_2$. Degrees of freedom for Raviart-Thomas elements are visualized in cyan (see Figure 5.1 for details). Degrees of freedom for Legendre elements are highlighted in orange. Degrees of freedom inside the shaded area illustrate the local basis functions of $\boldsymbol{V}_j \times Q_j$.

smoothing Algorithm 4 requires a coloring satisfying A_ℓ-orthogonality, i.e.,

$$R_i A_\ell R_j^\mathsf{T} = 0,$$

where A_ℓ is the discretization matrix associated with $\widetilde{a}_{div;h_\ell}(\cdot,\cdot)$. The action of $A_\ell R_j^\mathsf{T}$ is illustrated by the vertex patch stencil in Figure 5.2. The vertex patch $\mathcal{T}_j$ consists of all shaded cells. The coefficients of adjacent cells are coupled through the inter-element continuity of Raviart-Thomas elements and face integrals inherent in the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method, where the latter dominates the stencil width. The shaded area marks local degrees of freedom and, thus, those coefficients R_j restricts to. We emphasize that velocity degrees of freedom associated with $\partial\Omega_j$ and $\partial\Omega$ (the boundary of the 5×5 grid) are excluded to strongly impose boundary conditions of $\boldsymbol{V}_j$ and $\boldsymbol{V}_\ell$, respectively. Two subdomains Ω_j and Ω_i may have the same color if they do not share any joint facet. Consequently, the optimal coloring is given by the red-black parqueting depicted in Figure 4.1b.

The second class of algorithms (SFSC) numerically solves the reduced $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method (5.50) on each vertex patch via the stream function formulation from Section 5.2.2. The local pressure solution is then post-processed from the stream function, as detailed in Section 5.2.3. To this end, a biharmonic problem (4.1) is solved for each vertex patch $\mathcal{T}_j$. We define the local subspace of stream function approximations similar to (5.75),

$$\Psi_{\ell;j} = \left\{\psi \in H^1(\Omega) \mid \psi|_K \in \mathbb{Q}_{k+1}\ \forall K \in \mathcal{T}_{\ell;j}, \quad \psi|_{\partial\Omega_j} = 0\right\}. \tag{5.77}$$

The stream functions in Ψ_j have support only on Ω_j and a vanishing trace on $\partial\Omega_j$. Each subdomain Ω_j is simply connected such that the exact sequence of subspaces

$$0 \xrightarrow{\subset} \Psi_j \xrightarrow{\boldsymbol{\nabla}\times} \boldsymbol{V}_j \xrightarrow{\nabla\cdot} Q_j \longrightarrow 0 \tag{5.78}$$

implies a discrete Hodge decomposition

$$\boldsymbol{V}_j = \boldsymbol{\nabla}\times\Psi_j \oplus \boldsymbol{V}_j^\perp \tag{5.79}$$

subject to the $\boldsymbol{L}^2(\Omega_j)$-inner product. Then, there holds

$$a_{c0ip;j}(\phi,\psi) \equiv \widetilde{a}_{div;j}(\boldsymbol{\nabla}\times\phi, \boldsymbol{\nabla}\times\psi) \quad \text{for all } \phi,\psi \in \psi_j, \tag{5.80}$$

where the bilinear form $a_{c0ip;j}(\phi,\psi)$ is defined by substituting $\mathcal{T}_h$ with $\mathcal{T}_j$ in (4.10) for all $\phi,\psi\in\Psi_j$. We emphasize that in (Kanschat and Sharma, 2014) the equivalence (5.51) was shown for "global" finite element problems, where the no-slip boundary condition matches the clamped boundary condition in the sense of (5.42). Using restrictions of global bilinear forms $a_{c0ip;h}(\cdot,\cdot)$ and $\widetilde{a}_{div;h}(\cdot,\cdot)$ with respective subspaces, we have seen in the previous discussion on (HdivSC) and in Section 4.1.1 that we obtain nearly but not exactly no-slip and clamped

boundary conditions, respectively. Due to different penalty factors at the physical boundary $\partial\Omega \cap \partial\Omega_j$ and the interfaces on $\Omega \cap \partial\Omega_j$, equally weighted Neumann- and Dirichlet-type conditions are imposed for the latter. Nevertheless, continuing the concepts from (Kanschat and Sharma, 2014), it is straightforward to show equivalence also for this pair of boundary conditions.

The arithmetically dominant part of (SFSC) is solving the local velocity approximation with respect to the equivalence (5.80): Find a stream function $\phi_j \in \Psi_j$ such that

$$a_{c0ip;j}(\phi_j, \psi) = \int_{\Omega_j} \boldsymbol{f} \cdot \boldsymbol{\nabla} \times \psi \,\mathrm{d}\boldsymbol{x} \qquad \forall \psi \in \Psi_j. \tag{5.81}$$

Then, the divergence-free velocity solution $\boldsymbol{u}_j$ of (5.76) is determined by $\boldsymbol{\nabla} \times \phi_j$. The second step is post-processing the local pressure solution p_j given the stream function ϕ_j. Let $\tilde{\mathcal{E}}_j^\circ$ denote the subset of interfaces from Remark B.1.1, here concerning the vertex patch $\mathcal{T}_j$ instead of $\mathcal{T}_h$. In analogy to Lemma B.1.2, we obtain a local basis for $\boldsymbol{V}_j^\perp$ that is used to compute p_j from ϕ_j by utilizing the pseudo-algorithm explained in Remark 5.2.10: Find a pressure $p_j \in Q_j$ such that

$$-\int_{\Omega_j} \nabla \cdot \boldsymbol{v} p_j \,\mathrm{d}\boldsymbol{x} = \int_{\Omega_j} \boldsymbol{f} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} - \widetilde{a}_{div;j}(\boldsymbol{\nabla} \times \phi_j, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_j^\perp. \tag{5.82}$$

Let $\tilde{\boldsymbol{f}}_j$ denote the $\boldsymbol{L}^2(\Omega_j)$-projection of the local restriction $\boldsymbol{f}|_{\Omega_j}$. Then, $\boldsymbol{f}_j$ is implicitly decomposed regarding the discrete Hodge decomposition,

$$\boldsymbol{f}_j = \tilde{\boldsymbol{f}}_j^0 + \tilde{\boldsymbol{f}}_j^\perp.$$

Testing in (5.81) against $\boldsymbol{\nabla} \times \psi$ for any $\psi \in \Psi_j$, the solution ϕ_j only depends on the divergence-free part $\tilde{\boldsymbol{f}}_j^0 \in \boldsymbol{\nabla} \times \Psi_j$. Similarly, the solution p_j of (5.82) only depends on the orthogonal part $\tilde{\boldsymbol{f}}_j^\perp \in \boldsymbol{V}_j^\perp$. In computations, we explicitly need to distinguish the coefficients associated with $\tilde{\boldsymbol{f}}_j^0$ and $\tilde{\boldsymbol{f}}_j^\perp$, respectively. The coefficient vector $\tilde{\boldsymbol{f}}_j \in \mathbb{R}^{N_j^{\boldsymbol{v}}}$, which is defined by duality to the $\boldsymbol{L}^2(\Omega_j)$-projection $\tilde{\boldsymbol{f}}_j$ given the local shape function basis of $\boldsymbol{V}_j$, is passed to the local solver algorithm. We used the notation $\tilde{\boldsymbol{f}}_j$ for both the coefficient vector and the $\boldsymbol{L}^2(\Omega_j)$-projection and continue so hereafter, underlining the duality. $N_j^{\boldsymbol{v}}$ and N_j^ψ denote the dimension of $\boldsymbol{V}_j$ and Ψ_j, namely the number of local velocity and stream function degrees of freedom, respectively.

Let $\hat{\psi}_1, \ldots, \hat{\psi}_{N_{dof}^\psi} \in \mathbb{Q}_{k+1}$ denote the unit shape function basis of the Lagrange element from Definition 2.2.3. The canonical interpolation defined by Raviart-Thomas node functionals (5.54) prolongates $\hat{\boldsymbol{\nabla}} \times \hat{\psi}_i$ into RT_k on the unit cell. Taking the matching essential boundary conditions of $\boldsymbol{V}_j$ and Ψ_j into account, the cell-wise canonical interpolation under Piola transformation defines naturally a prolongation from $\boldsymbol{\nabla} \times \Psi_j$ into $\boldsymbol{V}_j$. In the context of

coefficients, the prolongation operator is determined by the matrix

$$(R_j^{\psi})^{\mathsf{T}} \in \mathbb{R}^{N_j^{\boldsymbol{v}} \times N_j^{\psi}},$$

that prolongates the coefficient vector of any stream function in Ψ_j into a coefficient vector dual to a velocity approximation in $\boldsymbol{V}_j$. We continue the notation from Section 2.3.2, defining the prolongation as the transposed restriction operator R_j^{ψ} of (dual) coefficients. Consequently, it holds

$$\tilde{\boldsymbol{f}}_j^0 = R_j^{\psi} \tilde{\boldsymbol{f}}_j.$$

Then, the local stream function solver (5.81) reads in matrix form

$$\boldsymbol{A}_j^{\psi} \phi_j = R_j^{\psi} \tilde{\boldsymbol{f}}_j, \tag{5.83}$$

where $\boldsymbol{A}_j^{\psi}$ denotes the discretization matrix associated with $a_{c0ip;j}(\cdot,\cdot)$.

In analogy, we obtain restriction matrices $R_j^{\perp\circ}$ and $R_j^{\perp\partial}$, defined by duality to the local basis functions

$$\boldsymbol{v}_{K;\boldsymbol{i}}^{\circ} \in \boldsymbol{V}_j^{\perp}, \qquad K \in \mathcal{T}_j, \quad \boldsymbol{i} = 1, \ldots, N_{dof}^{\perp\circ},$$

and

$$\boldsymbol{v}_e^{\partial} \in \boldsymbol{V}_j^{\perp}, \quad e \in \tilde{\mathcal{E}}_j^{\circ},$$

respectively. Then, following the pseudo-algorithm given by Remark 5.2.10, we make use of the restricted vectors $R_j^{\perp\circ}\tilde{\boldsymbol{f}}_j$ in Step (S1) and $R_j^{\perp\partial}\tilde{\boldsymbol{f}}_j$ in Step (S2), respectively: to be precise, $(R_j^{\perp\circ}\tilde{\boldsymbol{f}}_j)_{K;\boldsymbol{i}}$ and $(R_j^{\perp\partial}\tilde{\boldsymbol{f}}_j)_e$ provide the values $F_h(\boldsymbol{v}_{K;\boldsymbol{i}}^{\circ})$ and $F_h(\boldsymbol{v}_e^{\perp\partial})$ in computations of (5.73) and (5.74), respectively. Following the algorithmic steps from (5.83) until here, we successively compute the local velocity approximation $\boldsymbol{v}_j = \boldsymbol{\nabla} \times \phi_j$ and the local pressure solution p_j.

The same standard restriction R_j and prolongation R_j^{T} operators are used here, transferring coefficients between $\boldsymbol{V}_j \times Q_j$ and $\boldsymbol{V}_\ell \times Q_\ell$ as for the (HdivSC) class of Schwarz smoothers. Therefore, the same coloring is optimal, namely the red-black parqueting from Figure 4.1b.

Finally, we compare both classes of algorithms numerically, in particular, we expect the same number of iteration steps, only with slight deviations due to round-off errors. The right-hand side $\boldsymbol{f}$ in the simplified Stokes equations (5.17) is manufactured from reference solutions $\boldsymbol{v} = \boldsymbol{\nabla} \times \varphi$ and p,

$$\varphi(x,y) = \phi(x)\phi(y), \tag{5.84a}$$

$$p(x,y) = \cos(2\pi x)\cos(2\pi y), \tag{5.84b}$$

with

$$\phi(x) = \frac{x^2(x-1)^2}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{\sigma^2}\right). \tag{5.85}$$

Table 5.1 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **exact $\boldsymbol{H}^{\mathrm{div}}$-IP** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed, only levels L with 5×10^3 to 5×10^6 DoFs.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	11.0	11.9	11.9	1
4	—	11.3	10.5	11.2	11.7	12.0	11.8	1
5	21.7	11.5	10.2	10.1	11.3	11.3	11.0	1
6	22.4	11.7	9.4	9.2	9.7	10.5	10.5	1
7	23.7	12.1	9.4	9.5	8.9	9.5	10.0	1
8	25.4	12.3	9.5	9.3	—	—	—	1
9	26.0	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	3.8	3.6	3.6	8
4	—	5.9	4.6	4.1	4.2	3.9	3.9	8
5	11.2	6.0	4.8	4.4	4.3	4.1	4.0	8
6	12.4	6.5	5.0	4.6	4.3	4.1	4.0	8
7	13.2	6.9	5.0	4.6	4.3	4.1	4.0	8
8	13.5	7.1	5.2	4.6	—	—	—	8
9	14.0	—	—	—	—	—	—	8

Table 5.2 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **exact C^0-IP stream function** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed, only levels L with 5×10^3 to 5×10^6 DoFs. The convergence steps should be (and are) identical to Table 5.1 except for round-off errors.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	11.0	11.9	11.9	1
4	—	11.3	10.5	11.2	11.7	12.0	11.8	1
5	21.7	11.5	10.2	10.1	11.3	11.3	11.0	1
6	22.4	11.7	9.4	9.2	9.7	10.6	10.5	1
7	23.7	12.1	9.4	9.5	8.9	9.6	10.0	1
8	25.4	12.3	9.5	9.4	—	—	—	1
9	26.0	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	3.8	3.6	3.6	8
4	—	5.9	4.6	4.1	4.2	3.9	3.9	8
5	11.2	6.0	4.8	4.4	4.3	4.1	4.0	8
6	12.4	6.5	5.0	4.6	4.3	4.1	4.0	8
7	13.2	6.9	5.1	4.6	4.3	4.1	4.0	8
8	13.5	7.7	5.1	4.6	—	—	—	8
9	14.1	—	—	—	—	—	—	8

The vector curl of a scalar field is always divergence-free. Thus, $\boldsymbol{v}$ satisfies the incompressibility condition. The numerator $x^2(x-1)^2$ exists to satisfy no-slip boundary conditions. Furthermore, the mean value condition is satisfied by definition.

For additive smoothers (AVS), local solutions $(\boldsymbol{u}_j, p_j)$ are relaxed by a factor $\omega = 1/4$. A conjugate gradient method preconditioned by the multigrid V-cycle (Algorithm 2) solves the multilevel $\boldsymbol{H}^{\text{div}}$ interior penalty method until the Euclidean norm of the initial residual is reduced by 10^{-8}. A transposed order of colors in post-smoothing compared to pre-smoothing steps symmetrizes the V-cycle as explained in Section 3.1.1. Numerical experiments have shown, that two pre- and post-smoothing steps each are beneficial for the tensor product Schwarz smoothers presented in the next section. For the sake of comparability, we also use two smoothing steps here.

The coarse-grid solver A_1^{-1} belonging to the bilinear form $\widetilde{a}_{div,h_1}(\cdot,\cdot)$ is the inverse SVD of A_1. To be precise, we compute the pseudo-inverse neglecting the smallest singular value, which is zero since the mean value of the pressure is not unique. Consequently, the mean value is implicitly fixed. Similarly, we solve each local $\boldsymbol{H}^{\text{div}}$ interior penalty method (5.76) by means of a truncated SVD. Local stream functions ϕ_j in (5.83) are computed by SVDs, too. In contrast to the standard $\boldsymbol{H}^{\text{div}}$-IP local solver, Step (S3) in Remark 5.2.10 takes explicitly care of the mean value condition in Q_j.

Fractional iteration steps ν_{frac}[2] regarding the Schwarz smoothers computed by (HdivSC), i.e., using standard $\boldsymbol{H}^{\text{div}}$-IP local solvers, are listed in Table 5.1. The experiments concerning Schwarz smoothers computed by the second class of algorithms (SFSC), i.e., those using the local stream function solvers, are summarized in Table 5.2. We observe that except for rounding errors identical numbers of iterations are shown, which verifies the local equivalence (5.80) and confirms our post-processing algorithm of pressure approximates numerically. Both, the additive (AVS) as well as multiplicative Schwarz smoothers on vertex patches (MVS), lead to uniform convergence of numerical solvers with very few iterations needed (except for bi-linear elements): the iteration count is independent of the polynomial degree k and the mesh-refinement level L. The multiplicative smoother is mathematically superior because it does not need relaxation. Using MVS, even the number of iterations decreases with increasing polynomial degree. In mathematical terms, we have developed an efficient numerical solver.

5.2.5 Tensor Product Schwarz Smoothers

In computational terms, we face again the problem of inverting local discretization matrices cost-efficiently. Solving the local problems by means of SVDs is prohibitively expensive, in particular, for high-order finite elements. Both classes of Schwarz smoothers introduced

[2]The definition of ν_{frac} is given by (3.15).

in Section 5.2.4 do not admit a direct fast inversion of local matrices, although the matrices or, to be precise, the blocks of the matrices have low-rank tensor structures.

Let us have a closer look to the Schwarz smoothers obtained by (HdivSC). Skipping detailed derivations as in previous chapters, the local matrix for the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method on the subdomain Ω_j reads

$$\boldsymbol{A}_j = \begin{bmatrix} \boldsymbol{A}_{div;j} & \boldsymbol{B}_j^{\mathsf{T}} \\ \boldsymbol{B}_j & 0 \end{bmatrix} \tag{5.86}$$

with

$$\boldsymbol{A}_{div;j} = \begin{bmatrix} \boldsymbol{L}_{ip}^{(1)} \otimes \boldsymbol{M}^{(2)} + \boldsymbol{M}^{(1)} \otimes \boldsymbol{L}_{ip}^{(2)} & 0 \\ 0 & \boldsymbol{L}_{ip}^{(1)} \otimes \boldsymbol{M}^{(2)} + \boldsymbol{M}^{(1)} \otimes \boldsymbol{L}_{ip}^{(2)} \end{bmatrix} \tag{5.87}$$

and

$$\boldsymbol{B}_j = \begin{bmatrix} \boldsymbol{D}_{q\boldsymbol{v}}^{(1)} \otimes \boldsymbol{M}_{q\boldsymbol{v}}^{(2)} & \boldsymbol{M}_{q\boldsymbol{v}}^{(1)} \otimes \boldsymbol{D}_{q\boldsymbol{v}}^{(2)} \end{bmatrix}. \tag{5.88}$$

The univariate discretization matrix $\boldsymbol{D}_{q\boldsymbol{v}}^{(d)}$ relates to the d'th partial derivative arising from the divergence bilinear form $b(\cdot,\cdot)$, restricted to the d'th interval of the Cartesian product defining vertex patch Ω_j. Similarly, $\boldsymbol{M}_{q\boldsymbol{v}}^{(d)}$ denotes the univariate mass matrix with velocity ansatz and pressure test functions. In view of $\boldsymbol{L}_{ip}^{(d)}$, we emphasize that in one dimension the bilinear forms $\widetilde{a}_{div;h}(\cdot,\cdot)$ and $\widetilde{a}_{ip;h}(\cdot,\cdot)$ from (5.28) and (5.34) are identical. In other words, the $\boldsymbol{H}^{\mathrm{div}}$ and symmetric interior penalty method are identical on a subdivision of intervals. Furthermore,

$$\boldsymbol{L}_{ip}^{(d)} = \boldsymbol{L}^{(d)} + \boldsymbol{N}^{(d)}$$

is the univariate SIPG discretization of the Laplacian, here applied to the d'th vector component of the local velocity field, $d = 1, 2$. The bulk matrix $\boldsymbol{L}^{(d)}$, Nitsche boundary contributions $\boldsymbol{N}^{(d)}$ and the mass matrix $\boldsymbol{M}^{(d)}$ were defined in (3.35).

Since each diagonal block of $\boldsymbol{A}_{div;j}$ admits the separable Kronecker representation (2.21), $\boldsymbol{A}_{div;j}$ can be computed efficiently via fast diagonalization. However, no method was known to us that computed the local matrix $\boldsymbol{A}_j$ cost-efficiently and at the same time results in robust numerical solvers. We developed in (Arndt, Fehn, et al., 2020) an approximate Gaussian block elimination of local matrices that leverages the separable Kronecker representation of $\boldsymbol{A}_{div;j}$ and the rank-1 tensor structure of $\boldsymbol{B}_j$. However, here and there for the similar but slightly different local matrices concerning linear elasticity (in the limit of an incompressible material), numerical experiments have shown that we do not obtain robust numerical solvers, in particular, not for high-order finite elements.

Our Gaussian block elimination approach in (Arndt, Fehn, et al., 2020) computed an approximate velocity-pressure pair locally given the algebraic problem, i.e., given local discretization matrices. The structure of underlying differential operators was neglected, which is not the case for the Schwarz smoothers utilizing (SFSC). Using local stream function

solvers, computing the local velocity and pressure solution is decoupled into (5.81) and (5.82), respectively. Moreover, it enables us to compute only an approximate stream function $\tilde{\phi}_j$ of (5.81) which then leads to an approximate but still divergence-free velocity solution $\tilde{\boldsymbol{u}}_j = \boldsymbol{\nabla} \times \tilde{\phi}_j$ by definition, regardless of the quality of the stream function approximation. A local pressure approximation $\tilde{p}_j$, which is mean value free on Ω_j, can be computed from any known $\tilde{\phi}_j$ following (5.82). Consequently, any local stream function approximation in Ψ_j results in an approximate velocity-pressure pair in $(\boldsymbol{\nabla} \times \Psi_j) \times Q_j$. The studies in (Arnold, Falk, and Winther, 1997; Arnold, Falk, and Winther, 2000; Kanschat and Mao, 2015) have shown that satisfying the discrete Hodge decomposition (5.79) is crucial for constructing efficient $\boldsymbol{H}^{\text{div}}$-conforming Schwarz smoothers. Then, we are able to obtain robust numerical solvers.

We have shown in Section 4.1.2 that the finite element matrix $\boldsymbol{A}_j^{\psi}$ has a rank-3 tensor representation for each Cartesian vertex patch,

$$\boldsymbol{A}_j^{\psi} = \underline{\boldsymbol{B}}^{(1)} \otimes \boldsymbol{M}^{(2)} + 2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)} + \boldsymbol{M}^{(1)} \otimes \underline{\boldsymbol{B}}^{(2)}. \tag{5.89}$$

To this end, we utilize the approximations of $\boldsymbol{A}_j^{\psi}$ from Chapter 4, which admit fast inversions based on the tensor product techniques from Section 2.1. Before comparing numerical experiments, we emphasize that exact local solvers (computing stream function solutions) implies that the class of algorithms (SFSC) is computed through the standard implementations given in Section 5.2.4. "Standard" refers to computing local inverses via singular value decompositions. In contrast to that, we will introduce next a series of subspace corrections that may be cost-efficiently computed via fast diagonalization.

The same numerical setup as in Section 5.2.4 is used for the numerical results discussed next. The first experiment shall highlight that not necessarily any low-rank stream function approximation $\tilde{\phi}_j$ automatically results in "good" velocity-pressure solutions $(\tilde{\boldsymbol{u}}_j, \tilde{p}_j)$. In other words, satisfying only the incompressibility and mean value free condition is not sufficient for "good" smoothing. We still require subspace corrections that approximate the $\boldsymbol{H}^{\text{div}}$ interior penalty discretizations of local Stokes problem well enough.

To this end, we start with the simple approximation (4.58) utilizing the rank-1 KSVD of $\boldsymbol{A}_j^{\psi}$ which leads to $KSVD_1$ local solvers concerning the C^0-IP discretization. In Table 5.3 fractional convergence steps are shown for the Schwarz smoothers based on (SFSC), but computing local stream function approximations via inexact $KSVD_1$ instead of exact local solvers (see results in Table 5.2). Similarly to Section 4.1.2 (there comparing results from Tables 4.3 and 4.5, respectively), we observe that using $KSVD_1$ local solvers is also inferior here. Although the numerical solver converges uniformly with respect to the discretization level L, iteration steps quickly grow with increasing polynomial degree k. Comparing absolute numbers, significantly more iteration steps are needed in contrast to using exact stream function solvers: from $k = 2$ to $k = 7$ the gap grows from 2.5 times more iterations to over ten times more

Table 5.3 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **$\mathbf{KSVD_1}$ stream function** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed, only levels L with 5×10^3 to 5×10^6 DoFs.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	43.3	57.2	>100	1
4	—	26.0	30.8	37.0	44.9	64.2	>100	1
5	27.2	26.7	30.8	35.8	47.9	65.4	>100	1
6	30.2	27.5	31.5	35.5	47.2	68.2	>100	1
7	33.9	28.0	31.1	38.0	46.6	69.1	>100	1
8	38.6	29.0	31.6	36.9	—	—	—	1
9	41.6	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	14.9	16.7	18.7	8
4	—	13.8	11.9	16.0	15.9	16.7	18.6	8
5	15.2	14.6	12.5	16.3	15.7	17.5	20.2	8
6	15.9	15.6	12.6	16.4	16.2	17.9	20.6	8
7	15.9	16.2	12.7	16.4	16.0	18.3	20.9	8
8	15.6	16.8	12.8	16.4	—	—	—	8
9	16.7	—	—	—	—	—	—	8

Table 5.4 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **Bila stream function** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed, only levels L with 5×10^3 to 5×10^6 DoFs.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	12.8	13.3	13.7	1
4	—	10.4	12.1	12.8	12.5	12.6	13.8	1
5	19.0	10.6	11.4	11.6	11.9	12.1	11.5	1
6	20.2	10.6	10.3	10.4	10.8	11.4	11.0	1
7	20.9	10.7	9.7	10.1	10.0	10.4	10.5	1
8	22.5	10.7	10.1	10.2	—	—	—	1
9	22.5	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	6.4	6.8	7.7	8
4	—	5.9	5.7	5.8	6.6	7.3	7.9	8
5	11.0	6.0	5.9	5.9	6.6	6.9	7.7	8
6	11.9	6.7	6.0	5.9	6.6	6.5	7.1	8
7	12.9	7.1	6.0	5.9	5.9	5.8	6.7	8
8	13.4	7.5	6.0	5.7	—	—	—	8
9	13.7	—	—	—	—	—	—	8

for AVS, from two times more iterations to five times more for MVS. We emphasize that especially for MVS the growing gap is not only affected by increasing convergence steps for KSVD_1 local solvers but also by decreasing numbers for exact local solvers. The numerical solver with MVS converges uniformly regarding the mesh refinement, and almost uniformly regarding the polynomial degree. We conclude that the additive smoother used in Table 5.3 is deficient. MVS is feasible but mathematically inferior compared to its counterpart using exact stream function solvers. The quality of $KSVD_1$-related stream function approximations is not sufficient to obtain numerical solvers with uniform convergence.

In Section 4.1.2 we have developed two more inexact local solvers with significantly better smoothing quality than KSVD_1 local solvers. Both variants utilize better approximations of $\boldsymbol{A}_j^{\psi}$ and still enable fast inversion. The first variant omitted the elementary tensor involving mixed partial derivatives to obtain an approximate but separable rank-2 tensor representation (4.55) of $\boldsymbol{A}_j^{\psi}$. We referred to this method as *Bila local solver* because only the univariate C^0-IP discretization matrices concerning the "one-dimensional" *Bila*place operator are considered. Table 5.4 shows that the numerical solver using either the additive or the multiplicative vertex patch smoother with inexact Bila local solvers converges uniformly concerning both the level of mesh refinement and the polynomial degree. In addition, we obtain a robust solver needing almost as few iterations as the iterative solver with Schwarz

smoothers using exact stream function solvers. In particular, AVS compares at similar levels for both kinds of local solvers. For MVS we observe that convergence steps do not decrease with increasing polynomial degree in Table 5.4, leading to a slightly growing gap compared to the numbers in Table 5.2. To conclude, we have developed a tensor product Schwarz smoother that performs well in mathematical and computational terms. The topic of computational efficiency is postponed to Section 5.4.

A final remark before we continue with the next local solver, we also performed numerical experiments for the complementary part of (4.53), i.e., only using the mixed partial derivatives which admit a rank-1 tensor approximation (comparing to KSVD_1 local solvers)

$$\boldsymbol{A}_j^{\psi} \approx 2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)}.$$

The obtained smoothing quality is deficient. For both smoothers, AVS and MVS, the respective CG solver needs more than 200 iterations regardless of the polynomial degree, thus, no numerical results are displayed here. In view of Section 4.1.2, the inferior smoothing quality does not surprise because all face integrals from the interior penalty method are omitted. The Schwarz smoothers based on KSVD_1 local solvers offer significantly better results, which underlines the best approximation property of the rank-1 KSVD of $\boldsymbol{A}_j^{\psi}$. In addition, it shows again that not any low-rank stream approximation $\tilde{\phi}_j$ leads automatically to "good" subspace corrections.

The second option is computing the best rank-2 approximation of $\boldsymbol{A}_j^{\psi}$ by means of a KSVD. The approximation is rescaled given a factor α which guarantees positive definiteness locally, see the discussion in Section 4.1.2 for details. We referred to this method as $\mathrm{KSVD}_{12}(\alpha)$ local solvers. The convergence steps in Table 5.5 show that the multiplicative smoother results in a robust solver, while the additive smoother results in slowly growing numbers with increasing polynomial degree. The same tendencies were observed for solving the global biharmonic problem in Chapter 4. Here and there we suspect that the relaxation factor $\omega = 1/4$ is sub-optimal because smoothing with MVS using the same local solvers is nearly as good as smoothing with MVS using exact local solvers, see Table 5.2 for comparison. Furthermore, convergence steps are almost identical for polynomial degree $k = 1$ to $k = 4$ for MVS. For polynomial degree $k = 5$ to $k = 7$ a gap appears and grows slowly such that 1.4 times more fractional steps are needed with $\mathrm{KSVD}_{12}(\alpha)$ local solvers for the highest polynomial degree. However the inexact local solvers are amenable to fast inversion, thus, theoretically superior in computational terms. Comparing to Bila local solvers (which have also a tensor rank of two), the iteration steps for AVS are more compared to Table 5.5, but on the contrary MVS smooths better using $\mathrm{KSVD}_{12}(\alpha)$ local solvers.

We conclude that Schwarz smoothers using either Bila or $\mathrm{KSVD}_{12}(\alpha)$ local solvers achieve similarly high mathematical efficiency than those using exact local solvers. The overall conclusion, i.e., discussing also computational efficiency, follows in Section 5.4.

Table 5.5 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with $\mathbf{KSVD_{12}}(\alpha)$ **stream function** local solvers, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed, only levels L with 5×10^3 to 5×10^6 DoFs.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	16.8	17.5	18.6	1
4	—	11.1	13.6	14.9	16.7	17.7	18.1	1
5	20.4	11.1	13.8	14.2	16.4	17.8	17.8	1
6	22.7	11.2	13.1	14.4	16.1	16.3	17.9	1
7	23.7	11.3	13.1	13.6	15.4	16.4	17.2	1
8	24.6	11.4	13.3	13.8	—	—	—	1
9	25.8	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	5.3	5.6	5.8	8
4	—	6.1	4.9	5.2	5.5	5.7	5.8	8
5	11.3	6.4	4.8	5.1	5.4	5.6	5.7	8
6	12.5	6.8	4.8	5.1	5.4	5.6	5.6	8
7	13.3	7.0	4.8	5.1	5.3	5.6	5.6	8
8	13.7	7.5	4.8	5.1	—	—	—	8
9	14.4	—	—	—	—	—	—	8

5.3 Multilevel Methods for the Stokes Equations

Finally, we return to the initial model problem (5.1), the (non-simplified) *Stokes problem*. In comparison to the weak formulation (5.21) for simplified Stokes equations, only the bilinear form and right-hand side operator need to be changed, replacing the Laplacian with the symmetric gradient in (5.10). We follow (Girault, Kanschat, et al., 2014; Kanschat and Mao, 2015; Kanschat and Rivière, 2010; Kanschat and Sharma, 2014) in stating a multilevel $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method for the Stokes problem (5.1). Then, we will develop (tensor product) Schwarz smoothers utilizing (inexact) local solvers based on a stream function formulation and pressure post-processing similar to (5.81) and (5.82).

5.3.1 Mathematical Efficiency of Schwarz Smoothers

We consider the same numerical setup as in Section 5.2.4 on the simple domain $\Omega = [0,1]^2$, using a hierarchy of Cartesian meshes $\mathcal{T}_\ell, \ell = 1, \dots, L$. Therefore, the finite element spaces $\boldsymbol{V}_\ell$ and Q_ℓ are the same as before, i.e., imposing parts of no-slip boundary and mean value conditions, respectively. Then, the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty method for (non-simplified) Stokes flow reads: Find the velocity-pressure pair $(\boldsymbol{u}_\ell, p_\ell) \in \boldsymbol{V}_\ell \times Q_\ell$ such that

$$a_{div;h_\ell}(\boldsymbol{u}_\ell, \boldsymbol{v}) + b(\boldsymbol{v}, p_\ell) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_\ell, \tag{5.90a}$$

$$b(\boldsymbol{u}_\ell, q) = 0 \qquad \forall q \in Q_\ell, \tag{5.90b}$$

with

$$\begin{aligned} a_{div;h_\ell}(\boldsymbol{u}, \boldsymbol{v}) = &\int_{\mathcal{T}_\ell} 2\boldsymbol{\epsilon}(\boldsymbol{u}) : \boldsymbol{\epsilon}(\boldsymbol{v}) \,\mathrm{d}\boldsymbol{x} \\ &+ \int_{\mathcal{E}_\ell} 2 \left(\gamma_e \left[\!\left[\boldsymbol{u}^{\parallel} \right]\!\right] \cdot \left[\!\left[\boldsymbol{v}^{\parallel} \right]\!\right] - \left\{ \boldsymbol{\epsilon}(\boldsymbol{u}^{\parallel}) \right\} : \left[\!\left[\boldsymbol{n} \otimes \boldsymbol{v}^{\parallel} \right]\!\right] - \left[\!\left[\boldsymbol{n} \otimes \boldsymbol{u}^{\parallel} \right]\!\right] : \left\{ \boldsymbol{\epsilon}(\boldsymbol{v}^{\parallel}) \right\} \right) \mathrm{d}\sigma(\boldsymbol{x}). \end{aligned} \tag{5.91}$$

The divergence-related bilinear form $b(\cdot,\cdot)$ was defined in (5.11b). The right-hand side in (5.90a) implicitly imposes the complementary part of no-slip boundary conditions through penalties. We use standard transfer operators between multigrid levels, see Section 2.3.1 for details.

For Schwarz methods, we use the same finite element subspaces $\boldsymbol{V}_j$ and Q_j defined in (5.75). Let $\mathcal{T}_j$ denote the vertex patch regarding subdomain Ω_j, see Section 2.3.2 for details. The local bilinear form $a_{div;j}(\cdot,\cdot)$ is the restriction of $a_{div;h_\ell}(\cdot,\cdot)$ onto $\boldsymbol{V}_j \times \boldsymbol{V}_j$, i.e., replacing $\mathcal{T}_\ell$ and $\mathcal{E}_\ell$ through local sets $\mathcal{T}_j$ and $\mathcal{E}_j$ in (5.91). The local solver reads: Find

Table 5.6 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with **exact** local solvers *for Stokes flow*, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^3 to 5×10^6 DoFs.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	10.5	11.1	11.4	1
4	—	11.7	10.7	10.7	11.2	11.7	11.4	1
5	28.4	11.6	9.7	9.9	10.6	10.8	10.5	1
6	30.7	11.9	9.3	9.0	9.4	9.9	10.7	1
7	31.3	12.1	9.2	9.5	8.8	9.4	9.8	1
8	32.4	12.3	9.5	9.5	—	—	—	1
9	31.2	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	3.9	3.7	3.7	8
4	—	6.0	4.9	4.2	4.3	4.0	4.0	8
5	16.9	6.4	5.0	4.5	4.4	4.2	4.1	8
6	19.4	6.7	5.3	4.7	4.5	4.2	4.1	8
7	20.8	7.4	5.3	4.4	4.3	3.8	3.8	8
8	22.5	6.4	4.9	4.3	—	—	—	8
9	20.3	—	—	—	—	—	—	8

$\boldsymbol{u}_j \in \boldsymbol{V}_j$ and $p_j \in Q_j$ such that

$$a_{div;j}(\boldsymbol{u}_j, \boldsymbol{v}) + b(\boldsymbol{v}, p_j) = \int_{\Omega_j} \boldsymbol{f} \cdot \boldsymbol{v} \, \mathrm{d}\boldsymbol{x} \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_j, \tag{5.92a}$$

$$b(\boldsymbol{u}_j, q) = 0 \qquad \forall q \in Q_j. \tag{5.92b}$$

Due to restricting the bilinear form $a_{div;h_\ell}(\cdot,\cdot)$ onto subspaces, penalty factors associated with the physical boundary $\partial\Omega_j \cap \partial\Omega$ differ by a factor two to those associated with the "interior" boundary $\partial\Omega_j \cap \Omega$. Thus, we impose almost but not exactly no-slip boundary conditions locally, for details we refer to the discussion below equation (5.76).

We use the same numerical setup as in Section 5.2.4 for our experiments: manufacturing the right-hand side data $\boldsymbol{f}$ from reference solutions (5.84), using a CG solver preconditioned by V-cycle multigrid (Algorithm 2) and utilizing Schwarz smoothers with two pre- and post-smoothing steps each. The additive Schwarz smoother is relaxed by $\omega = 1/4$ and the coloring used for the multiplicative smoother is illustrated in Figure 4.1b.

Table 5.6 shows that we obtain numerical solvers with uniform convergence for both Schwarz methods, the additive (AVS) and multiplicative vertex patch smoother (MVS). The absolute numbers of iteration steps are as expected small, with MVS needing only half of the

steps than AVS, and are comparable with those in Table 5.1 for the simplified Stokes flow. Mathematically, both Schwarz smoothers perform well and are numerically efficient.

5.3.2 Tensor Product Schwarz Smoothers

Albeit their high mathematical efficiency, the computational efficiency of previous Schwarz methods using exact[3] local solvers is low since the inversion of local discretization matrices are costly and their memory consumption is high compared to matrix-free operators. The same held true concerning the smoothers in Table 5.1 for simplified Stokes flow.

To this end, we develop tensor product Schwarz smoothers as in Section 5.2.5. Using velocity and pressure subspaces $\boldsymbol{V}_j$ and Q_j, we again make use of the exact sequence of subspaces (5.78), thus, we utilize the stream function subspaces Ψ_j defined in (5.77). However, using symmetric gradients of velocity fields for Stokes flow, the local $\boldsymbol{H}^{\mathrm{div}}$ and C^0 interior penalty bilinear forms are not equal compared to (5.80),

$$a_{div;j}(\boldsymbol{\nabla} \times \phi, \boldsymbol{\nabla} \times \psi) \neq a_{c0ip;j}(\phi, \psi) \qquad \forall \phi, \psi \in \psi_j. \tag{5.93}$$

Therefore, we use the $\boldsymbol{H}^{\mathrm{div}}$ interior penalty bilinear form to define a local solver for the velocity solution $\boldsymbol{u}_j$: Find the stream function $\phi_j \in \Psi_j$ such that

$$a_{div;j}(\boldsymbol{\nabla} \times \phi_j, \boldsymbol{\nabla} \times \psi) = \int_{\Omega_j} \boldsymbol{f} \cdot \boldsymbol{\nabla} \times \psi \,\mathrm{d}\boldsymbol{x} \qquad \forall \psi \in \Psi_j, \tag{5.94}$$

$\boldsymbol{u}_j = \boldsymbol{\nabla} \times \phi_j$. For reasons of disctinction, we refer to (5.94) as *modified C^0 interior penalty method*. Due to the transpose of vector field gradients $(\boldsymbol{\nabla}\boldsymbol{v})^{\mathsf{T}}$ inherent in the symmetric gradient $\boldsymbol{\epsilon}(\boldsymbol{v})$, additional terms enter the left-hand side in (5.93) being not included in $a_{c0ip;j}(\cdot,\cdot)$. In this regard, we do not explicitly derive a modified C^0-IP bilinear form extending $a_{c0ip;j}(\cdot,\cdot)$, but simply use the $\boldsymbol{H}^{\mathrm{div}}$-IP bilinear form to define the modified C^0 interior penalty method. For such explicit derivation, we refer the interested reader to (Kanschat and Sharma, 2014, §4.2).

We need to adapt (5.82) slightly for the post-processing of the local pressure solution: Find a pressure $p_j \in Q_j$ such that

$$-\int_{\Omega_j} \nabla \cdot \boldsymbol{v} p_j \,\mathrm{d}\boldsymbol{x} = \int_{\Omega_j} \boldsymbol{f} \cdot \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} - a_{div;j}(\boldsymbol{\nabla} \times \phi_j, \boldsymbol{v}) \qquad \forall \boldsymbol{v} \in \boldsymbol{V}_j^{\perp}. \tag{5.95}$$

The construction of a local shape function basis for $\boldsymbol{V}_j^{\perp}$ is unchanged and, thus, also defined in Section B.1.

The computational cost of post-processing local pressure solutions is negligible compared to solving the modified C^0 interior penalty problems (5.94). In addition, a local pressure p_j

[3] Exactness implies that standard inversion techniques, e.g., the SVD, must be used in computations.

is computed from the local stream function ϕ_j as before, utilizing the pseudo-algorithm given by Remark 5.2.10 and only replacing in Lemmas 5.2.8 and 5.2.9 the simplified bilinear form $\widetilde{a}_{div;j}(\cdot,\cdot)$ by $a_{div;j}(\cdot,\cdot)$. Computational subletities were discussed below (5.82) for simplified Stokes flow that can be taken over one to one.

In contrast to (5.89), the local discretization matrix $\boldsymbol{A}_j^{\psi}$ associated with the stream function problem (5.94) for non-simplified Stokes flow has a larger tensor rank than three due to the transpose vector field gradients arising from symmetric gradients. For instance, it holds

$$\int_{\mathcal{T}_j} 2\boldsymbol{\epsilon}(\boldsymbol{u}) : \boldsymbol{\epsilon}(\boldsymbol{v})\,\mathrm{d}\boldsymbol{x} = \int_{\mathcal{T}_j} \boldsymbol{\nabla}\boldsymbol{u} : \boldsymbol{\nabla}\boldsymbol{v}\,\mathrm{d}\boldsymbol{x} + \int_{\mathcal{T}_j} \boldsymbol{\nabla}\boldsymbol{u} : \boldsymbol{\nabla}\boldsymbol{v}^{\mathsf{T}}\,\mathrm{d}\boldsymbol{x}. \tag{5.96}$$

The first integral on the right-hand side is what we obtained for simplified Stokes flow, which decomposed into a rank-3 tensor product representation for respective local ansatz and test functions, i.e., substituting $\boldsymbol{u} = \boldsymbol{\nabla} \times \phi_j$ and $\boldsymbol{v} = \boldsymbol{\nabla} \times \psi_i$, respectively,

$$\boldsymbol{B}^{(1)} \otimes \boldsymbol{M}^{(2)} + 2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)} + \boldsymbol{M}^{(1)} \otimes \boldsymbol{B}^{(2)}.$$

See Section 4.1.2 for definitions and notation. In analogy, we obtain a similar rank-3 tensor product representation for the rightmost integral in (5.96),

$$2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)} - \boldsymbol{H}^{(1)} \otimes (\boldsymbol{H}^{(2)})^{\mathsf{T}} - (\boldsymbol{H}^{(1)})^{\mathsf{T}} \otimes \boldsymbol{H}^{(2)}. \tag{5.97}$$

The derivation of this tensor presentation and definition of $\boldsymbol{H}$ is postponed to Section B.2. Proceeding analogously for the interior penalty terms, the local discretization matrix associated with (5.94) admits a rank-5 tensor representation in total,

$$\boldsymbol{A}_j^{\psi} = \underline{\boldsymbol{B}}^{(1)} \otimes \boldsymbol{M}^{(2)} + \boldsymbol{M}^{(1)} \otimes \underline{\boldsymbol{B}}^{(2)} + 4\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)} - \underline{\boldsymbol{H}}^{(1)} \otimes (\underline{\boldsymbol{H}}^{(2)})^{\mathsf{T}} - (\underline{\boldsymbol{H}}^{(1)})^{\mathsf{T}} \otimes \underline{\boldsymbol{H}}^{(2)}. \tag{5.98}$$

For the definition of $\underline{\boldsymbol{H}}$ we refer the interested reader to Section B.2. All remaining matrices were derived and defined in Section 4.1.2.

We use the same experimental setup as before to show the mathematical efficiency of tensor product Schwarz smoothers for Stokes flow. First, we tested three inexact local stream function solvers for the second class of algorithms (SFSC). We simply omit some of the elementary tensors in (5.98) such that tensor product approximations of $\boldsymbol{A}_j^{\psi}$ are obtained that enable fast diagonalization,

$$\boldsymbol{A}_j^{\psi} \approx \underline{\boldsymbol{B}}^{(1)} \otimes \boldsymbol{M}^{(2)} + \boldsymbol{M}^{(1)} \otimes \underline{\boldsymbol{B}}^{(2)}, \tag{5.99a}$$

$$\boldsymbol{A}_j^{\psi} \approx -\underline{\boldsymbol{H}}^{(1)} \otimes (\underline{\boldsymbol{H}}^{(2)})^{\mathsf{T}} - (\underline{\boldsymbol{H}}^{(1)})^{\mathsf{T}} \otimes \underline{\boldsymbol{H}}^{(2)}, \tag{5.99b}$$

$$\boldsymbol{A}_j^{\psi} \approx 4\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)}. \tag{5.99c}$$

Table 5.7 Fractional iterations ν_{frac} for additive vertex patch smoother (AVS) or multiplicative vertex patch smoother (MVS) with $\mathbf{KSVD_{12}}(\alpha)$ **stream function** local solvers for *modified C^0-IP discretizations*, respectively. CG solver with relative accuracy 10^{-8} preconditioned by multigrid. "Colors" refers to coloring on mesh level L. Entries "—" not computed only levels L with 5×10^3 to 5×10^6 DoFs.

Level L	Convergence steps ν_{frac}							Colors
AVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	19.6	20.7	22.6	1
4	—	13.6	16.5	18.3	19.4	21.5	21.6	1
5	27.7	14.2	16.4	17.7	19.6	20.4	22.4	1
6	29.6	14.5	16.8	17.2	18.3	19.6	21.3	1
7	30.6	14.6	15.8	17.2	18.6	19.0	21.3	1
8	31.2	14.8	15.8	16.6	—	—	—	1
9	31.5	—	—	—	—	—	—	1
MVS	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	
3	—	—	—	—	6.5	7.1	7.2	8
4	—	6.5	5.9	6.7	6.9	7.5	8.1	8
5	16.9	6.9	6.2	6.5	6.7	7.2	7.7	8
6	18.7	7.0	6.2	6.5	6.7	7.1	7.7	8
7	19.5	7.1	6.1	6.5	6.7	7.1	7.7	8
8	20.7	7.2	6.2	6.5	—	—	—	8
9	21.1	—	—	—	—	—	—	8

However, none of the respective local stream function solvers was sufficient in obtaining uniform solver convergence. In particular, none of them did converge with less than 200 conjugate gradient iteration steps for higher-order finite elements. Consequently, we do not present any fractional iteration steps here. Note that (5.99a) defines the so-called Bila local solvers used in Sections 4.1.2 and 5.2.5.

Second, we apply $\mathrm{KSVD}_{12}(\alpha)$ local solvers: computing the rank-2 KSVD of $\boldsymbol{A}_j^{\psi}$, which is then re-scaled by factor α to guarantee positive definiteness and efficiently inverted through fast diagonalization. For more details see Sections 4.1.2 and 5.2.5.

Table 5.7 shows the respective fractional convergence steps for both additive (AVS) and multiplicative Schwarz smoothers (MVS). Skipping the deficient linear case, we obtain uniform solver convergence regarding both the mesh level L and the polynomial degree k for MVS. For AVS we observe a similar steady increase of iterations with increasing polynomial degree as in Table 5.5 for simplified Stokes flow. In view of Table 5.6, MVS with $\mathrm{KSVD}_{12}(\alpha)$ local solvers compares well to MVS with exact local solvers, not needing more than a factor of two times the iterations for any polynomial degree: we require 1.1 times the iterations for $k = 2$, with a steadily increasing gap to 2.0 times the iterations for $k = 7$.

We conclude that we have designed algorithms for tensor product Schwarz smoothers that are both mathematically and computationally efficient. The few more CG solver steps are easily compensated by a much lower computational cost and memory consumption. The computational complexity compares at similar levels to the matrix-free operators of $\boldsymbol{H}^{\text{div}}$ interior penalty methods. More details follow in the next section.

5.4 Conclusion

It remains to discuss the computational efficiency of our tensor product Schwarz smoothers. We show below that an extensive analysis of (parallel) performance is not required and why similar performance results as in Section 3.4 for the Laplace operator can be expected here.

5.4.1 Computational Efficiency

At the time when the numerical experiments in Sections 5.2.5 and 5.3.2 were conducted, `deal.II`'s matrix-free framework did not support efficient operator application for Raviart-Thomas elements. However, given the anisotropic tensor structure of Raviart-Thomas elements (elaborated in Section 5.2.3) not much needs to be modified theoretically to obtain matrix-free operators for $\boldsymbol{H}^{\text{div}}$-IP discretizations compared to those for SIPG discretizations of the Laplacian. The matrix-free operator evaluation for the Laplace operator was detailed in Section 2.2.

Computing the cell-based operator application $A_K \boldsymbol{u}_K$ for the Stokes problem follows the same concepts as before. Similar to (2.50), we want to efficiently evaluate, among other terms,

$$\int_K \boldsymbol{\nabla}\, \boldsymbol{u}_h|_K : \boldsymbol{\nabla}\boldsymbol{\varphi}_{K;\boldsymbol{i}}\, \mathrm{d}\boldsymbol{x} \tag{5.100}$$

for all $\boldsymbol{i} = 1, \ldots, N_{dof}$, utilizing numerical integration. To this end, in analogy to the $\mathbb{R}^D$-valued interpolation $\mathrm{u}^{\boldsymbol{\nabla}}$ from (2.54) for the gradient $\boldsymbol{\nabla} u$, we compute $\mathbb{R}^{D\times D}$-valued interpolations of vector field gradients,

$$\mathbf{u}^{\boldsymbol{\nabla}}_{q_1,\ldots,q_D} := \begin{bmatrix} \mathrm{u}^{\partial_1}_{1;q_1,\ldots,q_D} & \cdots & \mathrm{u}^{\partial_D}_{1;q_1,\ldots,q_D} \\ \vdots & & \\ \mathrm{u}^{\partial_1}_{D;q_1,\ldots,q_D} & \cdots & \mathrm{u}^{\partial_D}_{D;q_1,\ldots,q_D} \end{bmatrix}$$

We emphasize that $\mathbf{u}^{\boldsymbol{\nabla}}$ is a multi-dimensional array of order D where each entry is the evaluation of the ansatz vector field gradient $\boldsymbol{\nabla}\boldsymbol{u}_h$ in a quadrature point $\boldsymbol{x}_{\boldsymbol{q}} = (x_{q_1}, \ldots, x_{q_D})$, respectively. Following the sum factorization in (2.53) concerning the (scalar) Laplace operator, the underlying array $\hat{\mathbf{u}}^{\boldsymbol{\nabla}}$ for the vector field gradient in unit space is cost-efficiently computed. Compared to (2.54b), the contributions from the cell transformation change due

to using the Piola transform (5.57) instead of a standard transform such that

$$\mathbf{u}^{\partial_d}_{c;q_1,\ldots,q_D} = \left|\det \hat{J}_K(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\right|^{-1} \sum_{r,s=1}^{D} \left(\hat{J}_K^{-1}(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\right)_{c,r} \hat{\mathbf{u}}^{\partial_s}_{r;q_1,\ldots,q_D} \left(\hat{J}_K(\hat{\boldsymbol{x}}_{\boldsymbol{q}})\right)_{s,d}.$$

$\hat{J}_K$ denotes the Jacobian of the cell mapping $\boldsymbol{F}_K$. Computing the array $\hat{\mathbf{u}}^{\boldsymbol{\nabla}}$ requires D^2 sweeps of the sum factorization (2.53) and, thus, $\mathcal{O}(D^4(n_{dof})^{D+1})$ arithmetic operations[4] in total. Applying the Piola transform costs only $\mathcal{O}(D^3(n_{dof})^D)$ operations. Consequently, computing $\mathbf{u}^{\boldsymbol{\nabla}}$ is dominated by the sum factorizations associated with the unit gradient $\hat{\mathbf{u}}^{\boldsymbol{\nabla}}$. Testing against the cell-local shape functions $\boldsymbol{\varphi}_{K;\boldsymbol{i}}$ utilizes again the application of the Piola transform and D^2 sweeps of sum factorizations (see (2.56) for further details). Consequently, computing (5.100) requires $\mathcal{O}(D^4(n_{dof})^{D+1})$ arithmetic operations asymptotically. More importantly, the computational complexity here has the same order of magnitude as needed for the Laplacian. Beside (5.100), other integrals in $a_{div;h}(\boldsymbol{u}_h, \boldsymbol{\varphi}_{K;\boldsymbol{i}})$ associated with cell K are evaluated similarly, including also the face integrals inherent in interior penalty methods.

What seems to be easily achieved in theory, requires some amount of work in practice. For instance, we implemented highly-optimized moment-based Raviart-Thomas elements in `TPSS`, leveraging the anisotropic tensor structure entirely. However, due to the anisotropic tensor structure of Raviart-Thomas elements, `deal.II`'s innermost tensor product evaluation kernels need some carefully-designed modifications, that exceeded the scope of the thesis. We implemented only prototypes of anisotropic tensor product kernels in `TPSS`, using them to handle anisotropic tensor product matrices, see the C++ class `TensorProductMatrix`. However, satisfying the requirements to be part of `deal.II` (for example, the applicability to generic meshes) exceeded this work's scope. We note that these topics are recently an active field of development in `deal.II`.

Using matrix-free operator evaluations for $a_{div;h}(\boldsymbol{\nabla} \times \tilde{\phi}_j, \cdot)$, also the local pressure coefficients in (5.73) and (5.74) can be computed very cost-efficiently. Thus, post-processing pressure solutions locally by means of the algorithm stated in Remark 5.2.10 aligns well with matrix-free operator evaluations for $\boldsymbol{H}^{\mathrm{div}}$ interior penalty discretizations. Therefore, the cost of computing pressure approximates $\tilde{p}_j$ via the pseudo-algorithm in Remark 5.2.10 are negligible compared to solving the C^0 interior penalty problems (5.83).

Taking all of this into account, the numerical solvers introduced in this chapter have computational complexities comparable to the Poisson problem. We argued in Section 4.2 that the same (parallel) performance as for the Laplacian can be expected for the Bilaplacian. In particular, we discussed our highly-optimized implementations of Bila and $\mathrm{KSVD}_{12}(\alpha)$ local solvers in Section 4.2, see Table 4.8 for an overview on computational complexities. In Sections 5.2.5 and 5.3.2, we demonstrated that the Schwarz smoothers on vertex patches

[4] Without loss of generality, we assume the same number of univariate quadrature points and basis functions, i.e., $n_{quad} = n_{dof}$.

utilizing either Bila or $\mathrm{KSVD}_{12}(\alpha)$ local solvers achieve (almost) an identical mathematical efficiency than the smoothers using exact local solvers. The latter is prohibitively expensive due to computing inverses at the cost of $\mathcal{O}((n_{dof})^{3D})$ arithmetic operations, thus, clearly lacking computational efficiency. On the contrary, we can expect for multilevel solvers based on our tensor product Schwarz smoothers a computational efficiency as high as demonstrated for the Laplace operator in Section 3.4.

We conclude that both tensor product smoothers qualify as mathematically sound and cost-efficient smoothers for simplified Stokes flow, but only the smoother using $\mathrm{KSVD}_{12}(\alpha)$ local solvers proves viable for (actual) Stokes flow. Constructing subspace corrections that preserve the exact sequence of subspaces (5.78) and have low-rank tensor structure, is essential to obtain fast and robust numerial solvers in this chapter.

Reference List

Adams, R. A. and Fournier, J. J. F. (2003). *Sobolev spaces*. Second. Pure and Applied Mathematics. Amsterdam Boston: Academic Press. ISBN: 9780120441433.

Adve, S. V., Hill, M. D., Miller, B. P., and Netzer, R. H. B. (1991). "Detecting data races on weak memory systems". In: *ACM SIGARCH Computer Architecture News* 19.3, pp. 234–243. DOI: 10.1145/115953.115976.

Anderson, R., Andrej, J., Barker, A., Bramwell, J., Camier, J.-S., Dobrev, J. C. V., Dudouit, Y., Fisher, A., Kolev, T., Pazner, W., Stowell, M., Tomov, V., Akkerman, I., Dahm, J., Medina, D., and Zampini, S. (2021). "MFEM: A Modular Finite Element Methods Library". In: *Computers & Mathematics with Applications* 81, pp. 42–74. DOI: 10.1016/j.camwa.2020.06.009.

Antonietti, P. F., Sarti, M., Verani, M., and Zikatanov, L. T. (2017). "A Uniform Additive Schwarz Preconditioner for High-Order Discontinuous Galerkin Approximations of Elliptic Problems". In: *Journal of Scientific Computing* 70.2, pp. 608–630. DOI: 10.1007/s10915-016-0259-9.

Antonietti, P. F. and Ayuso, B. (2007). "Schwarz domain decomposition preconditioners for discontinuous Galerkin approximations of elliptic problems: non-overlapping case". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 41.1, pp. 21–54. DOI: 10.1051/m2an:2007006.

— (2008). "Multiplicative Schwarz Methods for Discontinuous Galerkin Approximations of Elliptic Problems". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 42.3, pp. 443–469. DOI: 10.1051/m2an:2008012.

Argyris, J. H., Fried, I., and Scharpf, D. W. (1968). "The TUBA Family of Plate Elements for the Matrix Displacement Method". In: *The Aeronautical Journal (1968)* 72.692, pp. 701–709. DOI: 10.1017/S000192400008489X.

Arndt, D., Bangerth, W., Blais, B., Clevenger, T. C., Fehling, M., Grayver, A. V., Heister, T., Heltai, L., Kronbichler, M., Maier, M., et al. (2020). "The deal.II library, version 9.2". In: *Journal of Numerical Mathematics* 28.3, pp. 131–146.

Arndt, D., Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Pelteret, J.-P., Turcksin, B., and Wells, D. (2021). "The deal.II finite element library: Design, features, and insights". In: *Computers & Mathematics with Applications* 81. Development and Application of Open-source Software for Problems with Numerical PDEs, pp. 407–422. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2020.02.022.

Arndt, D., Fehn, N., Kanschat, G., Kormann, K., Kronbichler, M., Munch, P., Wall, W. A., and Witte, J. (2020). "ExaDG: High-Order Discontinuous Galerkin for the Exa-Scale". In: *Software for Exascale Computing - SPPEXA 2016-2019*. Ed. by H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel. Springer International Publishing, pp. 189–224. ISBN: 978-3-030-47956-5.

Arnold, D. N. (1982). "An Interior Penalty Finite Element Method with Discontinuous Elements". In: *SIAM Journal on Numerical Analysis* 19.4, pp. 742–760. DOI: 10.1137/0719052.

Arnold, D. N., Boffi, D., and Bonizzoni, F. (2014). "Finite element differential forms on curvilinear cubic meshes and their approximation properties". In: *Numerische Mathematik* 129.1, pp. 1–20. DOI: 10.1007/s00211-014-0631-3.

Arnold, D. N., Boffi, D., and Falk, R. S. (2005). "Quadrilateral H(div) Finite Elements". In: *SIAM Journal on Numerical Analysis* 42.6, pp. 2429–2451. DOI: 10.1137/s0036142903431924.

Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D. (2002). "Unified analysis of discontinuous Galerkin methods for elliptic problems". In: *SIAM Journal on Numerical Analysis* 39.5, pp. 1749–1779.

Arnold, D. N. and Falk, R. S. (1989). "A uniformly accurate finite element method for the Reissner–Mindlin plate". In: *SIAM Journal on Numerical Analysis* 26.6, pp. 1276–1290.

Arnold, D. N., Falk, R. S., and Winther, R. (1997). "Preconditioning in $H(\mathrm{div})$ and applications". In: *Mathematics of Computation* 66.219, pp. 957–984. ISSN: 0025-5718. DOI: 10.1090/S0025-5718-97-00826-0.

Arnold, D. N., Falk, R. S., and Winther, R. (2000). "Multigrid in $H(\mathrm{div})$ and $H(\mathrm{curl})$". In: *Numer. Math.* 85.2, pp. 197–217. DOI: 10.1007/PL00005386.

— (2006). "Finite element exterior calculus, homological techniques, and applications". In: *Acta Numerica* 15, pp. 1–155. DOI: 10.1017/s0962492906210018.

— (2010). "Finite element exterior calculus: from Hodge theory to numerical stability". In: *Bulletin of the American Mathematical Society* 47.2, pp. 281–354. DOI: 10.1090/s0273-0979-10-01278-4.

Babuška, I. (1971). "Error-bounds for finite element method". In: *Numerische Mathematik* 16.4, pp. 322–333.

Babuška, I. and Zlámal, M. (1973). "Nonconforming elements in the finite element method with penalty". In: *SIAM Journal on Numerical Analysis* 10.5, pp. 863–875.

Bangerth, W., Burstedde, C., Heister, T., and Kronbichler, M. (2011). "Algorithms and data structures for massively parallel generic adaptive finite element codes". In: *ACM Trans. Math. Softw.* 38, pp. 141–28.

Barker, A. T. and Kolev, T. (2020). "Matrix-free preconditioning for high-order H(curl) discretizations". In: *Numerical Linear Algebra with Applications* 28.2. DOI: 10.1002/nla.2348.

Bastian, P., Müller, E. H., Müthing, S., and Piatkowski, M. (2018). "Matrix-free multigrid block-preconditioners for higher order Discontinuous Galerkin discretisations". In: arXiv preprint 1805.11930.

Bastian, P., Altenbernd, M., Dreier, N.-A., Engwer, C., Fahlke, J., Fritze, R., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Shegunov, N., and Turek, S. (2020). "Exa-Dune—Flexible PDE Solvers, Numerical Methods and Applications". In: *Software for Exascale Computing - SPPEXA 2016-2019*. Ed. by H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel. Springer International Publishing, pp. 225–269. DOI: 10.1007/978-3-030-47956-5_9.

Bastian, P., Blatt, M., Dedner, A., Dreier, N.-A., Engwer, C., Fritze, R., Gräser, C., Grüninger, C., Kempf, D., Klöfkorn, R., Ohlberger, M., and Sander, O. (2021). "The Dune framework: Basic concepts and recent developments". In: *Computers & Mathematics with Applications* 81. Development and Application of Open-source Software for Problems with Numerical PDEs, pp. 75–112. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2020.06.007.

Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., and Turek, S. (2016). "Hardware-Based Efficiency Advances in the EXA-DUNE Project". In: Springer International Publishing, pp. 3–23. DOI: 10.1007/978-3-319-40528-5_1.

Bauer, S., Bunge, H.-P., Drzisga, D., Ghelichkhan, S., Huber, M., Kohl, N., Mohr, M., Rüde, U., Thönnes, D., and Wohlmuth, B. (2020). "TerraNeo—Mantle Convection Beyond a Trillion Degrees of Freedom". In: *Software for Exascale Computing - SPPEXA 2016-2019*.

Ed. by H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, and W. E. Nagel. Springer International Publishing, pp. 569–610. ISBN: 978-3-030-47956-5.

Bergen, B., Hülsemann, F., and Rüde, U. (2005). "Is 1.7×10^{10} Unknowns the Largest Finite Element System that Can Be Solved Today?" In: *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pp. 5–5. DOI: 10.1109/SC.2005.38.

Boffi, D., Brezzi, F., and Fortin, M. (2013). *Mixed Finite Element Methods and Applications*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-36519-5.

Bogner, F. K., Fox, R. L., and Schmit, L. A. (1965). "The generation of interelement-compatible stiffness and mass matrices by the use of interpolation formulas". In: *Proc. Conf. Matrix Meth. Struct. Mech.* Wright-Patterson AFB, pp. 397–443.

Bonizzoni, F. and Kanschat, G. (2021). "H^1-conforming finite element cochain complexes and commuting quasi-interpolation operators on Cartesian meshes". In: *Calcolo* 58.2. DOI: 10.1007/s10092-021-00409-6.

Braess, D. (2013). *Finite Elemente. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. 5. überarbeitete Auflage. Masterclass. Berlin ; Heidelberg: Springer Spektrum, XVI, 369 Seiten. ISBN: 978-3-642-34796-2.

Bramble, J. H. (Jan. 2019). *Multigrid methods*. Chapman and Hall/CRC. DOI: 10.1201/9780203746332.

Brenner, S. C. (2011). "C^0 Interior Penalty Methods". In: *Lecture Notes in Computational Science and Engineering*. Springer Berlin Heidelberg, pp. 79–147. DOI: 10.1007/978-3-642-23914-4_2.

Brenner, S. C. and Sung, L.-Y. (2005). "C^0 Interior Penalty Methods for Fourth Order Elliptic Boundary Value Problems on Polygonal Domains". In: *Journal of scientific computing* 22-23.1-3, pp. 83–118. ISSN: 1573-7691 and 0885-7474 and 1573-7691.

Brenner, S. C. and Wang, K. (2005). "Two-level additive Schwarz preconditioners for C^0 interior penalty methods". In: *Numerische Mathematik* 102.2, pp. 231–255. DOI: 10.1007/s00211-005-0641-2.

Brezzi, F. (1974). "On the existence, uniqueness and approximation of saddle-point problems arising from Lagrangian multipliers". In: *Publications mathématiques et informatique de Rennes* S4, pp. 1–26.

Brezzi, F., Douglas, J., Durán, R., and Fortin, M. (1987). "Mixed finite elements for second order elliptic problems in three variables". In: *Numerische Mathematik* 51.2, pp. 237–250.

Brezzi, F., Douglas, J., and Marini, L. D. (1985). "Two families of mixed finite elements for second order elliptic problems". In: *Numerische Mathematik* 47.2, pp. 217–235.

Brezzi, F. and Fortin, M. (1991). *Mixed and Hybrid Finite Element Methods*. Ed. by F. Brezzi and M. Fortin. Springer series in computational mathematics. Springer New York. DOI: 10.1007/978-1-4612-3172-1.

Brezzi, F., Fortin, M., and Stenberg, R. (1991). "Error analysis of mixed-interpolated elements for Reissner-Mindlin plates". In: *Mathematical Models and Methods in Applied Sciences* 1.02, pp. 125–151.

Brown, J. (2010). "Efficient nonlinear solvers for nodal high-order finite elements in 3D". In: *Journal of Scientific Computing* 45.1, pp. 48–63.

Brubeck, P. D. and Farrell, P. E. (2021). "A scalable and robust vertex-star relaxation for high-order FEM". In: arXiv: 2107.14758.

Cai, X.-C. and Sarkis, M. (Jan. 1999). "A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems". In: *SIAM Journal on Scientific Computing* 21.2, pp. 792–797. DOI: 10.1137/s10648275973267 8x.

Cantwell, C. D., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., De Grazia, D., Yakovlev, S., Lombard, J.-E., Ekelschot, D., Jordi, B., Xu, H., Mohamied, Y., Eskilsson, C., Nelson, B., Vos, P., Biotto, C., Kirby, R. M., and Sherwin, S. J. (2015). "Nektar++:

An open-source spectral/hp element framework". In: *Computer Physics Communications* 192, pp. 205–219. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2015.02.008.

Cantwell, C. D., Sherwin, S. J., Kirby, R. M., and Kelly, P. H. J. (2011). "From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements". In: *Computers & Fluids* 43.1, pp. 23–28.

Caussignac, P. (1987). "Computation of pressure from the finite element vorticity stream-function approximation of the Stokes problem". In: *Communications in Applied Numerical Methods* 3.4, pp. 287–295. DOI: 10.1002/cnm.1630030409.

Chang, W. L. and Grady, N. (2019). *NIST Big Data Interoperability Framework:* tech. rep. DOI: 10.6028/nist.sp.1500-1r2.

Cho, D., Pavarino, L. F., and Scacchi, S. (2018). "Isogeometric Schwarz preconditioners for the biharmonic problem". In: *ETNA - Electronic Transactions on Numerical Analysis* 49, pp. 81–102. DOI: 10.1553/etna_vol49s81.

Ciarlet, P. (1978). *The finite element method for elliptic problems.* Amsterdam New York New York: North-Holland Pub. Co. Sole distributors for the U.S.A. and Canada, Elsevier North-Holland. ISBN: 0444850287.

Ciarlet, P. G. (1988). *Mathematical Elasticity: Volume I: Three-Dimensional Elasticity.* North-Holland. ISBN: 978-0-444-70259-3.

— (1997). *Mathematical Elasticity: Volume II: Theory of Plates.* Studies in mathematics and its applications. Amsterdam: North-Holland, LXI, 497 S. ISBN: 978-0-444-82570-4.

— (2000). *Mathematical Elasticity: Volume III: Theory of Shells.* Studies in mathematics and its applications. Amsterdam: North-Holland, LX, 599 S. ISBN: 978-0-444-82891-0.

Clevenger, T. C., Heister, T., Kanschat, G., and Kronbichler, M. (2020). "A Flexible, Parallel, Adaptive Geometric Multigrid Method for FEM". In: *ACM Trans. Math. Softw.* 47.1. ISSN: 0098-3500. DOI: 10.1145/3425193.

Cockburn, B., Kanschat, G., and Schötzau, D. (2006). "A Note on Discontinuous Galerkin Divergence-free Solutions of the Navier–Stokes Equations". In: *Journal of Scientific Computing* 31.1-2, pp. 61–73. DOI: 10.1007/s10915-006-9107-7.

Couzy, W. and Deville, M. O. (1994). "Spectral-element preconditioners for the Uzawa pressure operator applied to incompressible flows". In: *Journal of Scientific Computing* 9.2, pp. 107–122.

— (1995). "A fast Schur complement method for the spectral element discretization of the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 116.1, pp. 135–142.

St-Cyr, A., Gander, M. J., and Thomas, S. J. (2007). "Optimized Multiplicative, Additive, and Restricted Additive Schwarz Preconditioning". In: *SIAM Journal on Scientific Computing* 29.6, pp. 2402–2425. DOI: 10.1137/060652610.

De Lathauwer, L., De Moor, B., and Vandewalle, J. (2000). "A Multilinear Singular Value Decomposition". In: *SIAM Journal on Matrix Analysis and Applications* 21.4, pp. 1253–1278.

Douglas, J. J., Dupont, T., Percell, P., and Scott, R. (1979). "A family of C^1 finite elements with optimal approximation properties for various Galerkin methods for 2^{nd} and 4^{th} order problems". In: *R.A.I.R.O. Analyse numérique* 13.3, pp. 227–255. ISSN: 0399-0516.

Dryja, M. and Widlund, O. B. (1990). "Some Domain Decomposition Algorithms for Elliptic Problems". In: *Iterative Methods for Large Linear Systems.* Academic Press Professional, Inc., pp. 273–291. ISBN: 0124074758.

Dryja, M. and Krzyżanowski, P. (2016). "A massively parallel nonoverlapping additive Schwarz method for discontinuous Galerkin discretization of elliptic problems". In: *Numerische Mathematik* 132.2, pp. 347–367. DOI: 10.1007/s00211-015-0718-5.

Duvaut, G. and Lions, J.-L. (1976). *Inequalities in mechanics and physics*. Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen. Springer. ISBN: 978-0-387-07327-9.

Efstathiou, E. and Gander, M. J. (2003). "Why Restricted Additive Schwarz Converges Faster than Additive Schwarz". In: *BIT Numerical Mathematics* 43.5, pp. 945–959. DOI: 10.1023/b:bitn.0000014563.33622.1d.

Engel, G., Garikipati, K., Hughes, T. J. R., Larson, M. G., Mazzei, L., and Taylor, R. L. (2002). "Continuous/discontinuous finite element approximations of fourth-order elliptic problems in structural and continuum mechanics with applications to thin beams and plates, and strain gradient elasticity". In: *Computer methods in applied mechanics and engineering* 191.34, pp. 3669–3750.

Falk, R. S. and Neilan, M. (Jan. 2013). "Stokes Complexes and the Construction of Stable Finite Elements with Pointwise Mass Conservation". In: *SIAM Journal on Numerical Analysis* 51.2, pp. 1308–1326. DOI: 10.1137/120888132.

Fehn, N., Kronbichler, M., Lehrenfeld, C., Lube, G., and Schroeder, P. W. (2019). "High-order DG solvers for underresolved turbulent incompressible flows: A comparison of L^2 and H(div) methods". In: *International Journal for Numerical Methods in Fluids* 91.11, pp. 533–556. DOI: 10.1002/fld.4763.

Fehn, N., Munch, P., Wall, W. A., and Kronbichler, M. (2020). "Hybrid multigrid methods for high-order discontinuous Galerkin discretizations". In: *Journal of Computational Physics* 415, p. 109538. DOI: 10.1016/j.jcp.2020.109538.

Fehn, N., Wall, W. A., and Kronbichler, M. (2017). "On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations". In: *Journal of Computational Physics* 351, pp. 392–421. DOI: 10.1016/j.jcp.2017.09.031.

— (2018a). "A matrix-free high-order discontinuous Galerkin compressible Navier-Stokes solver: A performance comparison of compressible and incompressible formulations for turbulent incompressible flows". In: *International Journal for Numerical Methods in Fluids* 89.3, pp. 71–102. DOI: 10.1002/fld.4683.

— (2018b). "Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows". In: *International Journal for Numerical Methods in Fluids* 88.1, pp. 32–54. DOI: 10.1002/fld.4511.

— (2018c). "Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows". In: *Journal of Computational Physics* 372, pp. 667–693. DOI: 10.1016/j.jcp.2018.06.037.

Feng, X. and Karakashian, O. A. (2001). "Two-level additive Schwarz methods for a discontinuous Galerkin approximation of second order elliptic problems". In: *SIAM Journal on Numerical Analysis* 39.4, pp. 1343–1365.

— (2005). "Two-level non-overlapping Schwarz preconditioners for a discontinuous Galerkin approximation of the biharmonic equation". In: *Journal of Scientific Computing* 22.1-3, pp. 289–314.

Fischer, P. F. (1990). "Analysis and application of a parallel spectral element method for the solution of the Navier-Stokes equations". In: *Computer Methods in Applied Mechanics and Engineering* 80.1-3, pp. 483–491.

Fischer, P. F. and Lottes, J. W. (2005). "Hybrid Schwarz-multigrid methods for the spectral element method: Extensions to Navier-Stokes". In: *Domain Decomposition Methods in Science and Engineering*. Springer, pp. 35–49.

Fischer, P. F., Lottes, J. W., and Kerkemeier, S. G. (2008). *nek5000 Web page*. http://nek5000.mcs.anl.gov.

Fischer, P. F. and Patera, A. T. (1991). "Parallel spectral element solution of the Stokes problem". In: *Journal of Computational Physics* 92.2, pp. 380–421.

Fischer, P. F., Tufo, H. M., and Miller, N. I. (2000). "An overlapping Schwarz method for spectral element simulation of three-dimensional incompressible flows". In: *Parallel Solution of Partial Differential Equations*. Springer, pp. 159–180.

Fischer, P., Min, M., Rathnayake, T., Dutta, S., Kolev, T., Dobrev, V., Camier, J.-S., Kronbichler, M., Warburton, T., Świrydowicz, K., and Brown, J. (2020). "Scalability of high-performance PDE solvers". In: *The International Journal of High Performance Computing Applications* 34.5, pp. 562–586. DOI: 10.1177/1094342020915762.

Gander, M. J. (2006). "Optimized Schwarz Methods". In: *SIAM Journal on Numerical Analysis* 44.2, pp. 699–731. DOI: 10.1137/s0036142903425409.

Girault, V., Kanschat, G., and Rivière, B. (2014). "Error analysis for a monolithic discretization of coupled Darcy and Stokes problems". In: *Journal of Numerical Mathematics* 22.2. DOI: 10.1515/jnma-2014-0005.

Girault, V. and Raviart, P.-A. (1986). *Finite Element Methods for Navier-Stokes Equations. theory and algorithms*. Springer series in computational mathematics. Springer Berlin Heidelberg, X, 374 S. DOI: 10.1007/978-3-642-61623-5.

Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., and Wohlmuth, B. (2015). "Performance and scalability of hierarchical hybrid multigrid solvers for Stokes systems". In: *SIAM Journal on Scientific Computing* 37.2, pp. C143–C168.

Golub, G. H., Luk, F. T., and Overton, M. L. (1981). "A Block Lanczos Method for Computing the Singular Values and Corresponding Singular Vectors of a Matrix". In: *ACM Transactions on Mathematical Software* 7.2, pp. 149–169. DOI: 10.1145/355945.355946.

Golub, G. H. and Van Loan, C. F. (2013). *Matrix Computations*. J. Hopkins Uni. Press. ISBN: 1421407949.

Golub, G. and Kahan, W. (1965). "Calculating the singular values and pseudo-inverse of a matrix". In: *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis* 2.2, pp. 205–224.

Graham, A. (1981). *Kronecker Products and Matrix Calculus: With Applications*. Ellis Horwood Ltd.

Hackbusch, W. (1985). *Multi-grid methods and applications*. Berlin New York: Springer-Verlag. ISBN: 9780387127613.

Hackbusch, W. (2012). *Tensor spaces and numerical tensor calculus*. Vol. 42. Springer Science & Business Media.

— (2014). "Numerical tensor calculus". In: *Acta Numerica* 23, pp. 651–742. DOI: 10.1017/S0962492914000087.

Hansbo, P. and Larson, M. G. (2002). "Discontinuous Galerkin methods for incompressible and nearly incompressible elasticity by Nitsche's method". In: *Computer Methods in Applied Mechanics and Engineering* 191.17-18, pp. 1895–1908. DOI: 10.1016/s0045-7825(01)00358-9.

Hiptmair, R. (1998). "Multigrid Method for Maxwell's Equations". In: *SIAM Journal on Numerical Analysis* 36.1, pp. 204–225. DOI: 10.1137/s0036142997326203.

— (2002). "Finite elements in computational electromagnetism". In: *Acta Numerica* 11, pp. 237–339. DOI: 10.1017/s0962492902000041.

Hiptmair, R. (1997). "Multigrid method for H(div) in three dimensions". In: *Electron. Trans. Numer. Anal* 6.1, pp. 133–152.

Hiptmair, R. and Toselli, A. (2000). "Overlapping and Multilevel Schwarz Methods for Vector Valued Elliptic Problems in Three Dimensions". In: *Parallel Solution of Partial Differential Equations*. Ed. by P. Bjørstad and M. Luskin. New York, NY: Springer New York, pp. 181–208. ISBN: 978-1-4612-1176-1.

Hughes, T. J. R. (2012). *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

Kanschat, G. (2003). "Discontinuous Galerkin Finite Element Methods for Advection-Diffusion Problems". Habilitationsschrift. Universität Heidelberg.

— (2008). "Robust smoothers for high order discontinuous Galerkin discretizations of advection-diffusion problems". In: *Journal of Computational and Applied Mathematics* 218, pp. 53–60. DOI: 10.1016/j.cam.2007.04.032.

Kanschat, G., Lazarov, R., and Mao, Y. (2017). "Geometric Multigrid for Darcy and Brinkman models of flows in highly heterogeneous porous media: A numerical study". In: *Journal of Computational and Applied Mathematics* 310, pp. 174–185. DOI: 10.1016/j.cam.2016.05.016.

Kanschat, G. and Mao, Y. (2015). "Multigrid methods for $\mathbf{H}^{\mathrm{div}}$-conforming discontinuous Galerkin methods for the Stokes equations". In: *J. Numer. Math.* 23.1, pp. 51–66. DOI: 10.1515/jnma-2015-0005.

Kanschat, G. and Rivière, B. (2010). "A strongly conservative finite element method for the coupling of Stokes and Darcy flow". In: *Journal of Computational Physics* 229.17, pp. 5933–5943. DOI: 10.1016/j.jcp.2010.04.021.

Kanschat, G. and Sharma, N. (2014). "Divergence-Conforming Discontinuous Galerkin Methods and C^0 Interior Penalty Methods". In: *SIAM Journal on Numerical Analysis* 52.4, pp. 1822–1842. DOI: 10.1137/120902975.

Karniadakis, G. and Sherwin, S. (2005). *Spectral/hp element methods for computational fluid dynamics*. Second. Oxford University Press, pp. 1–650. ISBN: 9780198528692.

Karniadakis, G., Karniadakis, G. E., and Kirby II, R. M. (2003). *Parallel scientific computing in C++ and MPI: a seamless approach to parallel algorithms and their implementation*. Vol. 1. Cambridge University Press.

Kempf, D., Heß, R., Müthing, S., and Bastian, P. (2020). "Automatic Code Generation for High-Performance Discontinuous Galerkin Methods on Modern Architectures". In: *ACM Trans. Math. Softw.* 47.1. ISSN: 0098-3500. DOI: 10.1145/3424144.

Kolda, T. G. and Bader, B. W. (2009). "Tensor decompositions and applications". In: *SIAM review* 51.3, pp. 455–500.

Krank, B., Fehn, N., Wall, W. A., and Kronbichler, M. (2017). "A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow". In: *Journal of Computational Physics* 348, pp. 634–659. DOI: 10.1016/j.jcp.2017.07.039.

Kronbichler, M. and Kormann, K. (2012). "A generic interface for parallel cell-based finite element operator application". In: *Computers & Fluids* 63, pp. 135–147.

Kronbichler, M., Kormann, K., Fehn, N., Munch, P., and Witte, J. (2019). *A Hermite-like basis for faster matrix-free evaluation of interior penalty discontinuous Galerkin operators*. Tech. rep. arXiv:1907.08492.

Kronbichler, M. and Allalen, M. (2018). "Efficient High-Order Discontinuous Galerkin Finite Elements with Matrix-Free Implementations". In: *Progress in IS*. Springer International Publishing, pp. 89–110. DOI: 10.1007/978-3-319-99654-7_7.

Kronbichler, M. and Kormann, K. (2019). "Fast matrix-free evaluation of discontinuous Galerkin finite element operators". In: *ACM Transactions on Mathematical Software (TOMS)* 45.3, p. 29.

Kronbichler, M., Kormann, K., Pasichnyk, I., and Allalen, M. (2017). "Fast Matrix-Free Discontinuous Galerkin Kernels on Modern Computer Architectures". In: *High Performance Computing*. Ed. by J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes. Springer International Publishing, pp. 237–255. ISBN: 978-3-319-58667-0.

Ladyženskaja, O. A. (1969). *The Mathematical theory of viscous incompressible flow*. Second. Mathematics and its applications. New York: Gordon & Breach, XVIII, 224 S.

Linke, A. (2014). "On the role of the Helmholtz decomposition in mixed methods for incompressible flows and a new variational crime". In: *Computer Methods in Applied Mechanics and Engineering* 268, pp. 782–800. DOI: 10.1016/j.cma.2013.10.011.

Linke, A., Matthies, G., and Tobiska, L. (2016). "Robust Arbitrary Order Mixed Finite Element Methods for the Incompressible Stokes Equations with pressure independent velocity errors". In: *ESAIM: Mathematical Modelling and Numerical Analysis* 50.1, pp. 289–309. DOI: 10.1051/m2an/2015044.

Lottes, J. W. and Fischer, P. F. (2005). "Hybrid multigrid/Schwarz algorithms for the spectral element method". In: *Journal of Scientific Computing* 24.1, pp. 45–78.

Lucero Lorca, J. P. and Kanschat, G. (2018). "Multilevel Schwarz preconditioners for singularly perturbed symmetric reaction-diffusion systems". In: submitted.

Lynch, R. E., Rice, J. R., and Thomas, D. H. (1964). "Direct solution of partial difference equations by tensor product methods". In: *Numerische Mathematik* 6.1, pp. 185–199.

Maitre, J.-F. and Pourquier, O. (1996). "Condition number and diagonal preconditioning: comparison of the p -version and the spectral element methods". In: *Numerische Mathematik* 74.1, pp. 69–84. DOI: 10.1007/s002110050208.

Malkus, D. S. and Hughes, T. J. R. (1978). "Mixed finite element methods—reduced and selective integration techniques: a unification of concepts". In: *Computer Methods in Applied Mechanics and Engineering* 15.1, pp. 63–81.

May, D. A., Brown, J., and Le Pourhiet, L. (2014). "pTatin3D: High-performance methods for long-term lithospheric dynamics". In: *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, pp. 274–284.

McRae, A. T. T., Bercea, G.-T., Mitchell, L., Ham, D. A., and Cotter, C. J. (2016). "Automated Generation and Symbolic Manipulation of Tensor Product Finite Elements". In: *SIAM Journal on Scientific Computing* 38.5, S25–S47. DOI: 10.1137/15m1021167.

Meister, A. (2015). *Numerik linearer Gleichungssysteme*. Vol. 5. Springer.

Montlaur, A., Fernandez-Mendez, S., and Huerta, A. (2008). "Discontinuous Galerkin methods for the Stokes equations using divergence-free approximations". In: *International journal for numerical methods in fluids* 57.9, pp. 1071–1092.

Müthing, S., Piatkowski, M., and Bastian, P. (2017). *High-performance implementation of matrix-free high-order discontinuous Galerkin methods*. Tech. rep. arXiv:1711.10885.

Nečas, J. (1967). *Les méthodes directes en théorie des équations elliptiques*. Masson, 351 S.

Nédélec, J. C. (1980). "Mixed finite elements in $\mathbb{R}^3$". In: *Numerische Mathematik* 35.3, pp. 315–341. DOI: 10.1007/bf01396415.

— (1986). "A new family of mixed finite elements in $\mathbb{R}^3$". In: *Numerische Mathematik* 50.1, pp. 57–81. DOI: 10.1007/bf01389668.

Nitsche, J. (1971). "Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind". In: *Abhandlungen aus dem mathematischen Seminar der Universität Hamburg*. Vol. 36. 1. Springer, pp. 9–15.

Oñate, E. and Cervera, M. (1993). "Derivation of thin plate bending elements with one degree of freedom per node: a simple three node triangle". In: *Engineering Computations: Int J for Computer-Aided Engineering* 10.6, pp. 543–561.

Oñate, E. and Zárate, F. (2000). "Rotation-free triangular plate and shell elements". In: *International Journal for Numerical Methods in Engineering* 47.1-3, pp. 557–603.

Orszag, S. A. (1980). "Spectral methods for problems in complex geometries". In: *Journal of Computational Physics* 37.1, pp. 70–92.

Oseledets, I. V. (2011). "Tensor-train decomposition". In: *SIAM Journal on Scientific Computing* 33.5, pp. 2295–2317.

Pavarino, L. F. (1993). "Additive Schwarz methods for the p-version finite element method". In: *Numerische Mathematik* 66.1, pp. 493–515. DOI: 10.1007/bf01385709.

Pazner, W. (2020). "Efficient Low-Order Refined Preconditioners for High-Order Matrix-Free Continuous and Discontinuous Galerkin Methods". In: *SIAM Journal on Scientific Computing* 42.5, A3055–A3083. DOI: 10.1137/19M1282052.

Pazner, W. and Kolev, T. (2021). "Uniform Subspace Correction Preconditioners for Discontinuous Galerkin Methods with hp-Refinement". In: *Communications on Applied Mathematics and Computation*. DOI: 10.1007/s42967-021-00136-3.

Pazner, W. and Persson, P.-O. (2018). "Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods". In: *Journal of Computational Physics* 354, pp. 344–369. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2017.10.030.

Phaal, R. and Calladine, C. R. (1992a). "A simple class of finite elements for plate and shell problems. I: Elements for beams and thin flat plates". In: *International Journal for Numerical Methods in Engineering* 35.5, pp. 955–977.

— (1992b). "A simple class of finite elements for plate and shell problems. II: An element for thin shells, with only translational degrees of freedom". In: *International Journal for Numerical Methods in Engineering* 35.5, pp. 979–996.

Piatkowski, M. and Bastian, P. (2019). "A high-order discontinuous Galerkin pressure robust splitting scheme for incompressible flows". In: *arXiv preprint arXiv:1912.10242*.

Piatkowski, M., Müthing, S., and Bastian, P. (2018). "A stable and high-order accurate discontinuous Galerkin based splitting method for the incompressible Navier–Stokes equations". In: *Journal of Computational Physics* 356, pp. 220–239.

Pitsianis, N. P. (1997). "The Kronecker Product in Approximation and Fast Transform Generation". PhD thesis. Cornell University.

Rannacher, R. (2017). "Numerik 2". In: DOI: 10.17885/HEIUP.281.370.

Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., Mcrae, A. T. T., Bercea, G.-T., Markall, G. R., and Kelly, P. H. J. (2016). "Firedrake: Automating the Finite Element Method by Composing Abstractions". In: *ACM Trans. Math. Softw.* 43.3. ISSN: 0098-3500. DOI: 10.1145/2998441.

Raviart, P.-A. and Thomas, J.-M. (1977). "A mixed finite element method for 2-nd order elliptic problems". In: *Mathematical aspects of finite element methods*. Springer, pp. 292–315.

Rudi, J., Malossi, A. C. I., Isaac, T., Stadler, G., Gurnis, M., Staar, P. W. J., Ineichen, Y., Bekas, C., Curioni, A., and Ghattas, O. (2015). "An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle". In: *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–12.

Saad, Y. (2011). *Numerical Methods for Large Eigenvalue Problems: Revised Edition*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics. ISBN: 9781611970722.

Schöberl, J. (1999a). "Multigrid methods for a parameter dependent problem in primal variables". In: *Numerische Mathematik* 84.1, pp. 97–119. DOI: 10.1007/s002110050465.

— (1999b). "Robust Multigrid Methods for Parameter Dependent Problems". PhD thesis. Johannes Kepler Universität.

— (2014). "C++11 implementation of finite elements in NGSolve". In: *Institute for Analysis and Scientific Computing, Vienna University of Technology* 30.

Schwarz, H. A. (1870). "Ueber einen Grenzübergang durch alternirendes Verfahren". In:

Shahbazi, K., Fischer, P. F., and Ethier, C. R. (2007). "A high-order discontinuous Galerkin method for the unsteady incompressible Navier–Stokes equations". In: *Journal of Computational Physics* 222.1, pp. 391–407. DOI: 10.1016/j.jcp.2006.07.029.

Simon, H. D. and Zha, H. (2000). "Low-Rank Matrix Approximation Using the Lanczos Bidiagonalization Process with Applications". In: *SIAM Journal on Scientific Computing* 21.6, pp. 2257–2274. DOI: 10.1137/s1064827597327309.

Smith, B., Bjorstad, P., and Gropp, W. D. (2004). *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press. ISBN: 9780521602860.

Stiller, J. (2016). "Robust multigrid for high-order discontinuous Galerkin methods: A fast Poisson solver suitable for high-aspect ratio Cartesian grids". In: *Journal of Computational Physics* 327, pp. 317–336.

— (2017). "Robust Multigrid for Cartesian Interior Penalty DG Formulations of the Poisson Equation in 3D". In: *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2016*. Ed. by M. L. Bittencourt, N. A. Dumont, and J. S. Hesthaven. Springer International Publishing, pp. 189–201. ISBN: 978-3-319-65870-4.

Toselli, A. and Widlund, O. (2005). *Domain decomposition methods—algorithms and theory*. Vol. 34. Springer Series in Computational Mathematics. Berlin: Springer-Verlag, pp. xvi+450.

Treibig, J., Hager, G., and Wellein, G. (2010). "LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments". In: *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*. San Diego CA.

Tufo, H. M. and Fischer, P. F. (1999). "Terascale spectral element algorithms and implementations". In: *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, 68–es.

Van Loan, C. F. (2000). "The ubiquitous Kronecker product". In: *Journal of computational and applied mathematics* 123.1-2, pp. 85–100.

Van Loan, C. F. and Pitsianis, N. (1993). "Approximation with Kronecker products". In: *Linear algebra for large scale and real-time applications*. Springer, pp. 293–314.

Varga, R. S. (2009). *Matrix iterative analysis*. Second. Berlin: Springer.

Vos, P. E. J., Sherwin, S. J., and Kirby, R. M. (2010). "From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance for low-and high-order discretisations". In: *Journal of Computational Physics* 229.13, pp. 5161–5181.

Widlund, O. and Dryja, M. (1990). "Towards a unified theory of domain decomposition algorithms for elliptic problems". In: *Proceedings of the Third International Symposium on Domain Decomposition Methods for Partial Differential, Houston, Texas, March 20-22, 1989*. SIAM.

Witte, J., Arndt, D., and Kanschat, G. (2021). "Fast Tensor Product Schwarz Smoothers for High-Order Discontinuous Galerkin Methods". In: *Computational Methods in Applied Mathematics* 21.3, pp. 709–728. DOI: 10.1515/cmam-2020-0078.

Xu, J. (1992). "Iterative Methods by Space Decomposition and Subspace Correction". In: *SIAM Review* 34.4, pp. 581–613. DOI: 10.1137/1034116.

Yosida, K. (1965). *Functional Analysis*. Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-25762-3.

Appendix A

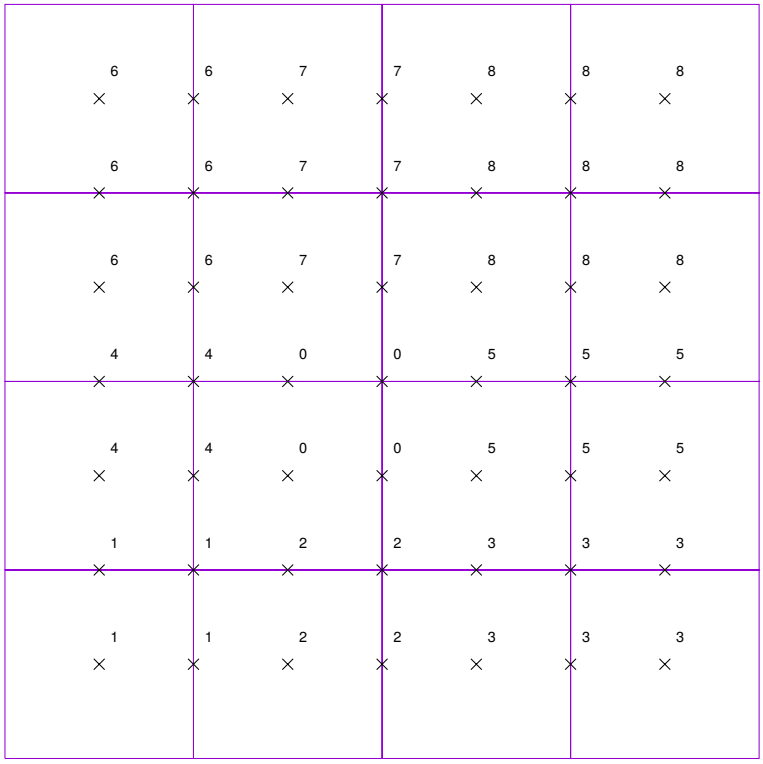
POISSON PROBLEM

A.1 Restricted Additive Schwarz Method

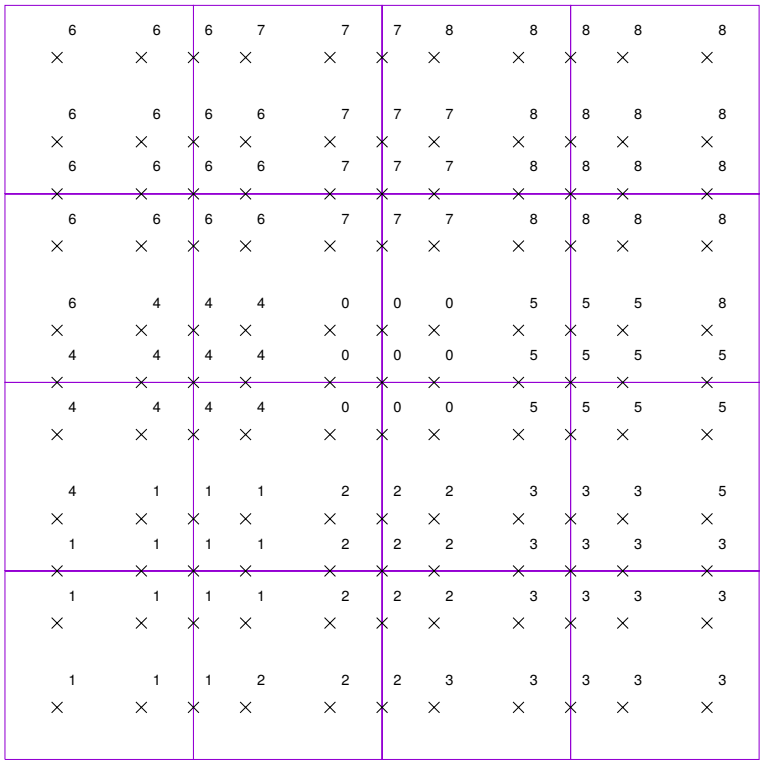
In Section 3.3.1 we postponed describing how to obtain a partition of unity for boolean restricted additive Schwarz (bRAS). To this end, we use illustrations Figure A.1 and Figure A.2 for H^1-conforming and discontinuous finite elements with quadratic and cubic polynomial degrees in two dimensions: a nodal shape function basis is visualized through Gauss-Lobatto support points. We recall that local basis functions on a vertex patch were already illustrated in Figures 3.1 and 3.4a.

We want to uniquely assign each degree of freedom (illustrated as a cross $\times$) to a single "nonoverlapping vertex patch". For conforming finite elements, we start with assigning a unique number to each degree of freedom associated with an interior vertex. This enumeration identifies our nonoverlapping subdomains, i.e., those subdomains with a unique association of degrees of freedom. The assignment of remaining degrees of freedom is done by a closest distance measure with respect to the interior vertices. The Chebyshev distance is used as metric for the grid of support points. If a support point has minimal Chebyshev distance to more than one interior vertex, the nonoverlapping vertex patch with the higher numbering takes "ownership". Examples are shown in Figure A.1.

For DG elements, we proceed similarly. However, four degrees of freedom per interior vertex and two per interior edge are obtained in two dimensions. The degrees of freedom belonging to interior vertices are enumerated first. Then, we apply the same decision rule based on the Chebyshev distance, where we additionally take into account to which element a degree of freedom belongs. The results are depicted in Figure A.2.

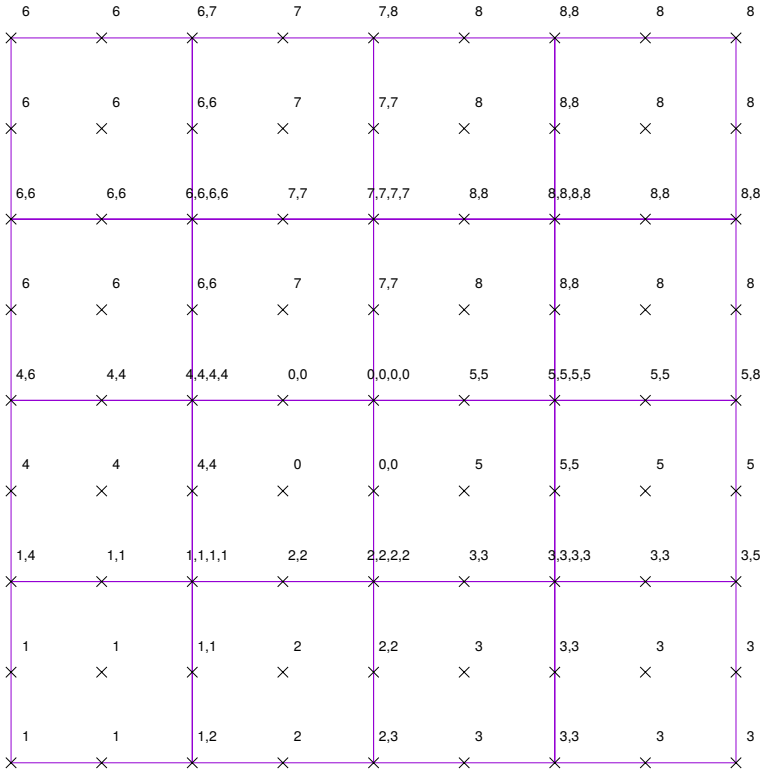


(a) $k = 2$

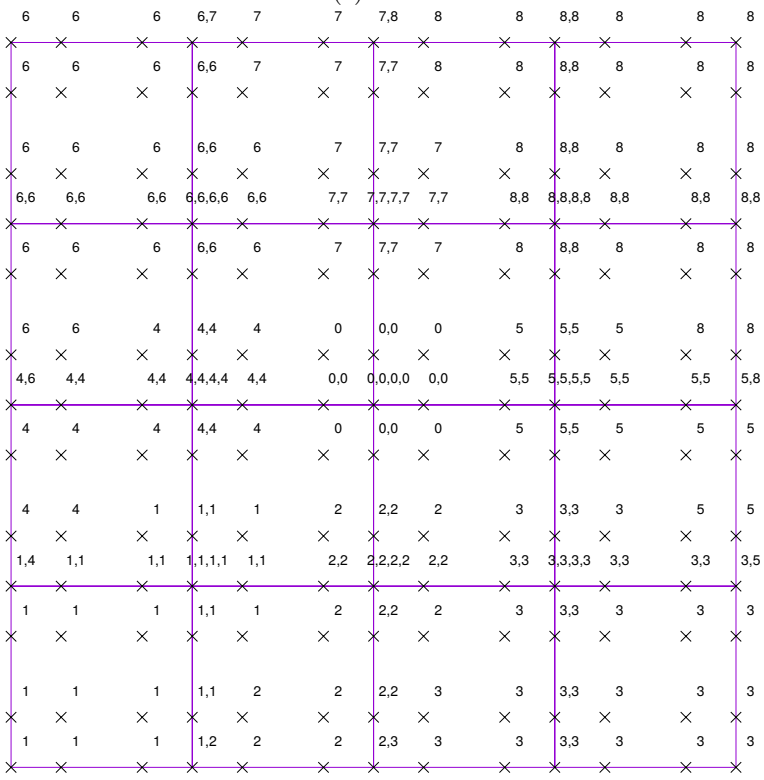


(b) $k = 3$

Fig. A.1 Boolean **partition of unity** for H^1**-conforming FEM** methods $\mathbb{Q}_k$-polynomials. Degrees of freedom are represented as nodal values ($\times$) and its owning nonoverlapping subdomain is given by superscripts $0, \ldots, 8$. Degrees of freedom at the boundary of the physical domain are neglected due to strongly imposed Dirichlet conditions.



(a) $k = 2$



(b) $k = 3$

Fig. A.2 Boolean **partition of unity** for **SIPG** discretizations with discontinuous $\mathbb{Q}_k$-polynomials. Degrees of freedom are represented as nodal values ($\times$) and its owning subdomain through superscripts $0, \ldots, 8$. Nodal values on edges and vertices are associated with one degree of freedom per mesh cell attached. It is apparent to which cell each comma-seperated patch superscript belongs.

A.2 Parallel Performance

In this section, we show some more strong and weak scaling results Figures A.3 and A.4 for a very high polynomial degree $k = 15$. These results append those from Section 3.4.2. In addition, we display time-to-solution results for polynomial degree $k = 7$ in Figures A.5 and A.6.

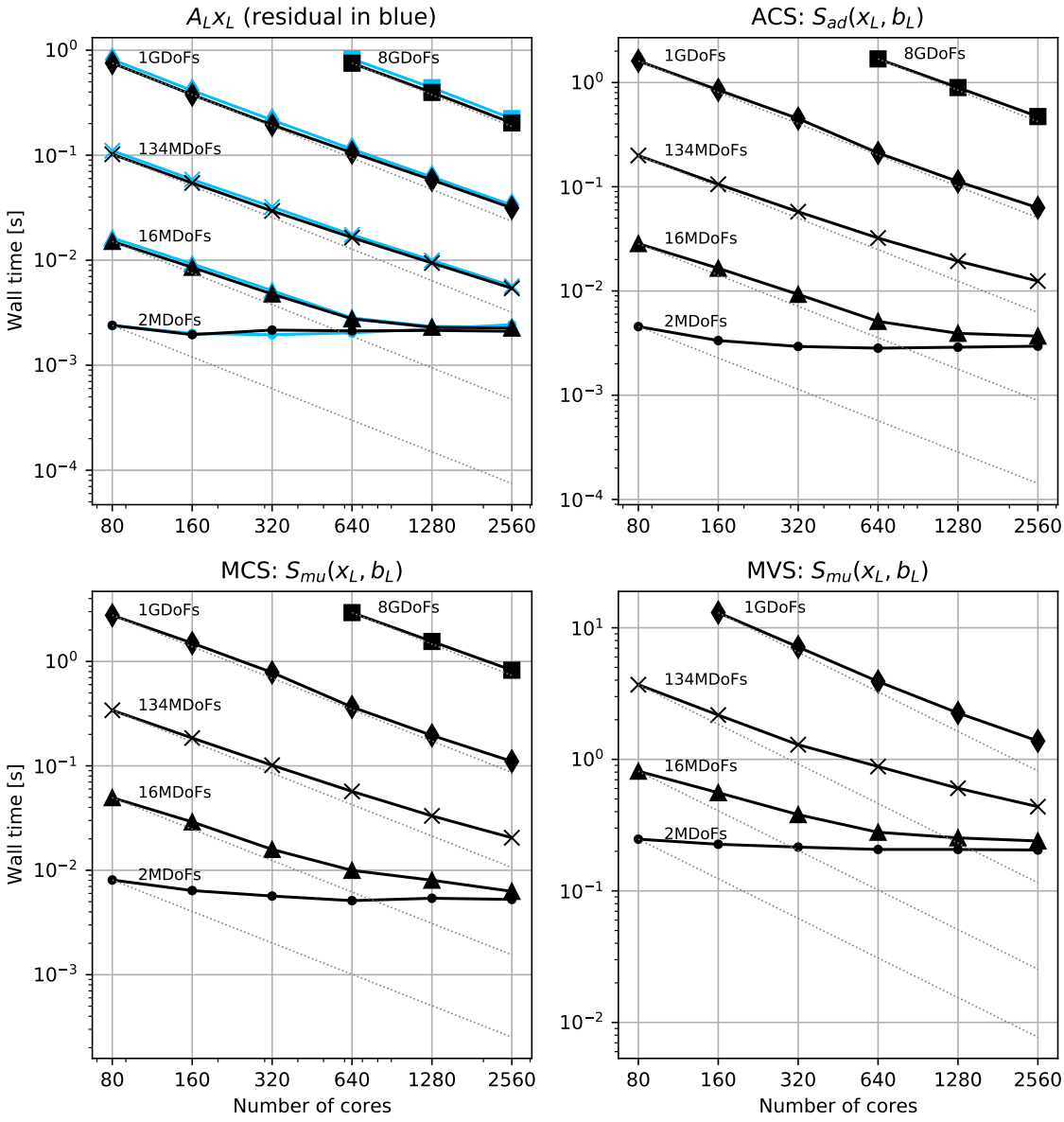


Fig. A.3 MPI scaling analysis of one smoothing step for ACS (*top-right*), MCS (*bottom-left*), and MVS (*bottom-right*) given **SIPG** discretizations using $\mathbb{Q}_{15}$**-elements**. Smoothing is compared against the matrix-free operator evaluation (*top-left*) and its residual $b_L - A_L x_L$ delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

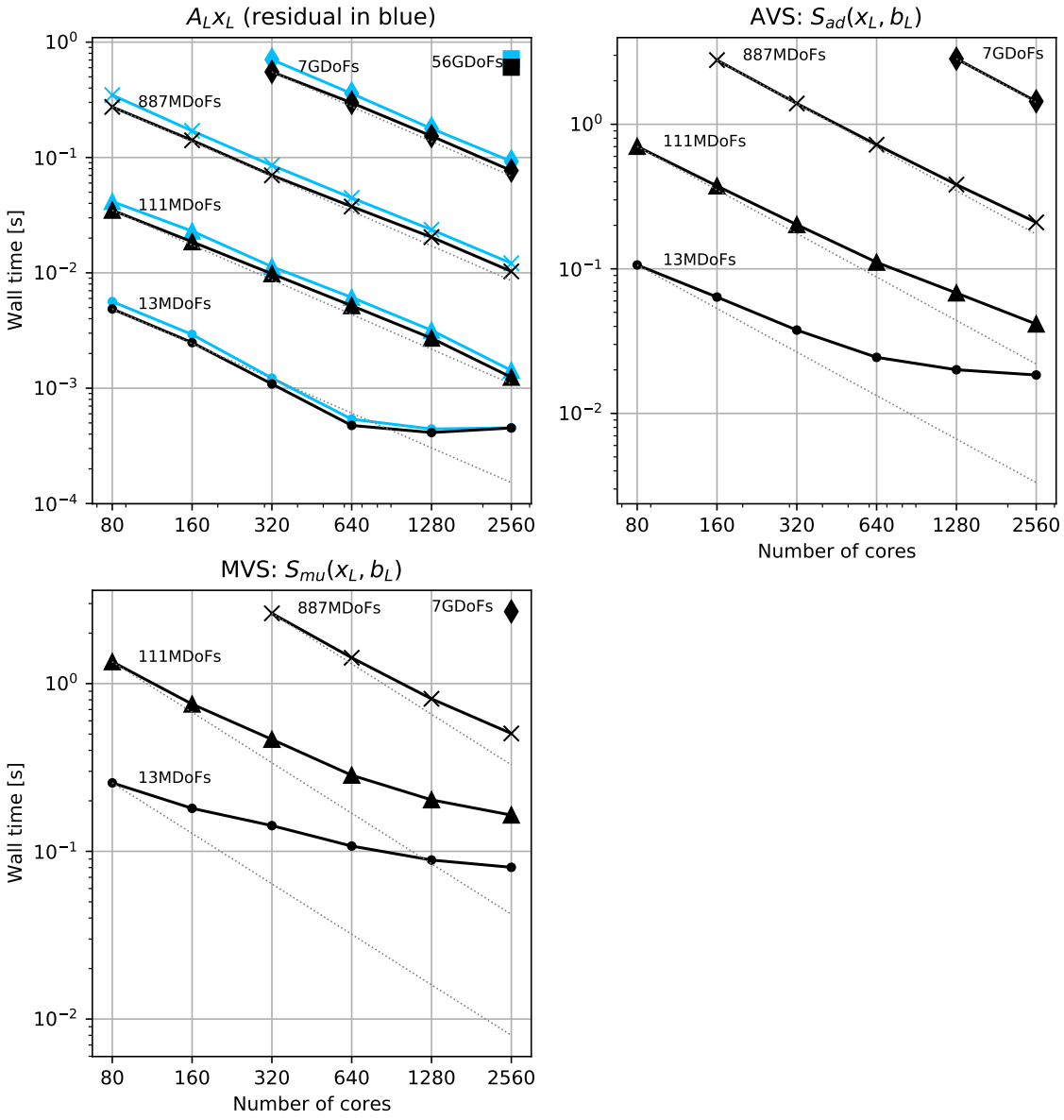


Fig. A.4 MPI scaling analysis of one smoothing step for AVS (*top-right*) and MVS (*bottom-left*) given H^1-**conforming** discretizations using $\mathbb{Q}_{15}$-**elements**. Smoothing is compared against the matrix-free operator evaluation (*top-left*) and its residual $b_L - A_L x_L$ delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

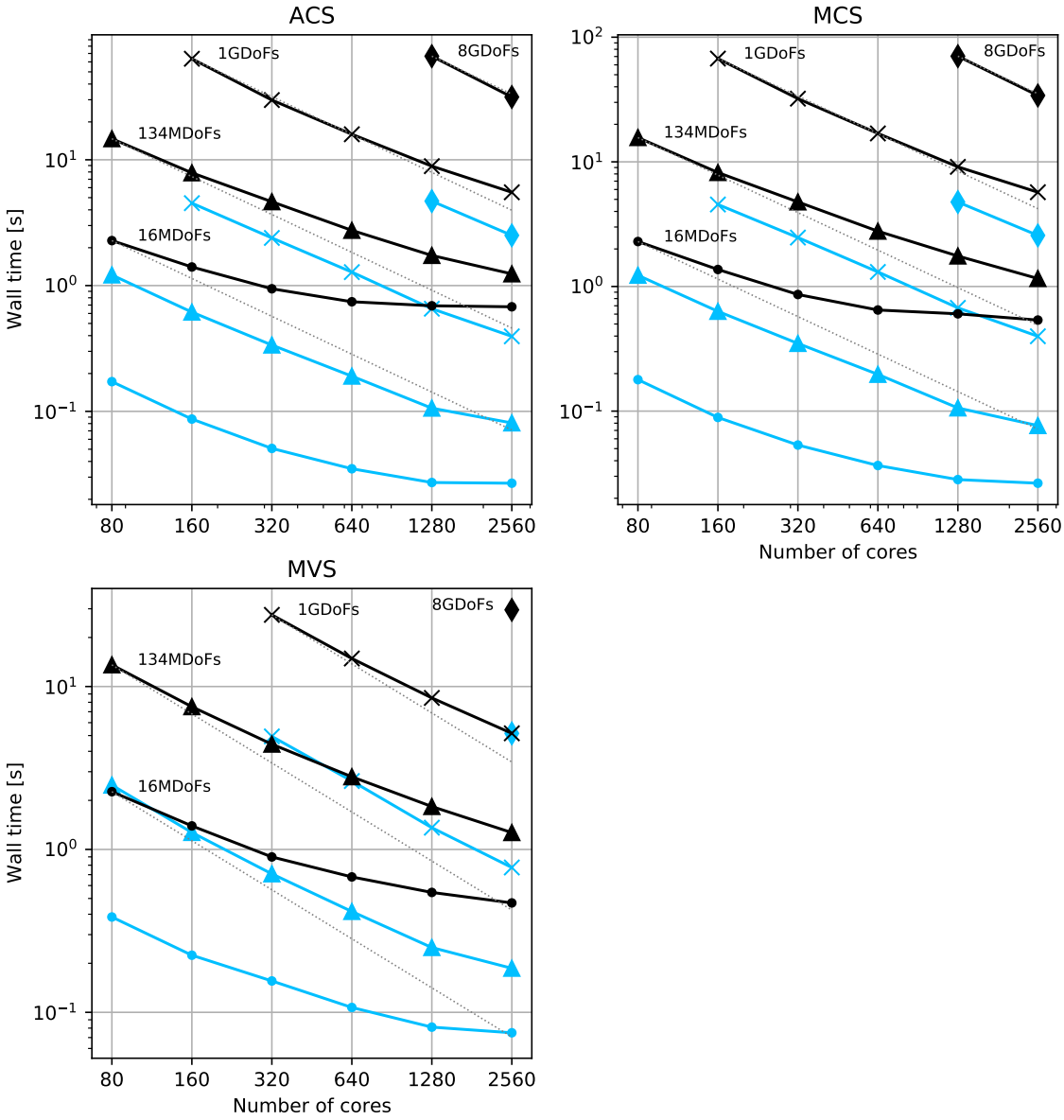


Fig. A.5 MPI scaling analysis of the numerical solver, i.e., measuring **time-to-solution** (*black lines*), for **SIPG** discretizations using $\mathbb{Q}_7$**-elements**. The CG solver accelerating the multilevel method with ACS (*top-left*), MCS (*top-right*), or MVS (*bottom-left*), respectively, solves with relative accuracy 10^{-8}. The respective time to compute pre- and post-smoothers before solving is delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

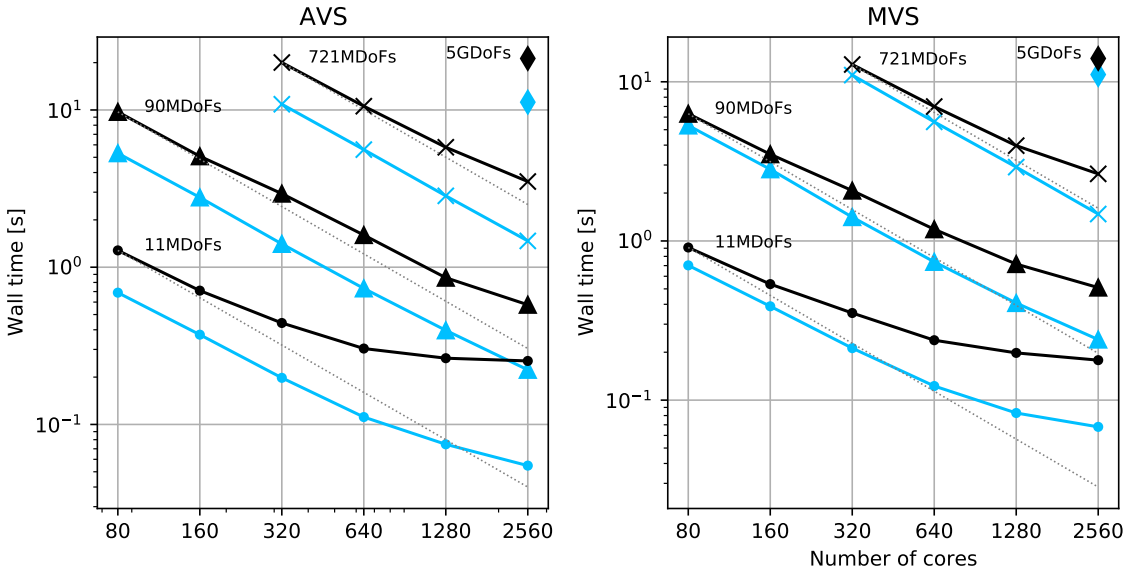


Fig. A.6 MPI scaling analysis of the numerical solver, i.e., measuring **time-to-solution** (*black lines*), for H^1**-conforming** discretizations using $\mathbb{Q}_7$**-elements**. The CG solver accelerating the multilevel method with AVS (*top-left*) or MVS (*top-right*), respectively, solves with relative accuracy 10^{-8}. The respective time to compute pre- and post-smoothers before solving is delineated in *blue*. Perfect strong scaling is seen along *greyish-dotted lines*.

Appendix B

STOKES PROBLEM

B.1 Post-processing Pressure

This section extends Section 5.2.3 with a rigorous construction of $\boldsymbol{V}_h^{\perp}$'s basis functions. For some notations and definitions we refer to Section 5.2.3. Without loss of generality, we assume that the $N_{dof}^{\perp\circ}$ first Raviart-Thomas node functionals $\hat{\mathcal{N}}_{\boldsymbol{j}}^{\circ}$ from (5.54a) are generated through polynomials $\hat{\boldsymbol{w}}_{\boldsymbol{j}} = \hat{p}_{\boldsymbol{j}}^{\perp} \in \boldsymbol{\nabla}\mathbb{Q}_k^{\perp}$, and the four node functionals $\hat{\mathcal{N}}_{j_1,1}^{\partial}$ from (5.54a) through $\hat{q}_1 \equiv 1$. In other words, we assume

$$\hat{\mathcal{N}}_{\boldsymbol{j}}^{\circ} = \hat{\mathcal{N}}_{\boldsymbol{j}}^{\perp\circ}, \quad \boldsymbol{j} = 1, \ldots, N_{dof}^{\perp\circ}, \tag{B.1a}$$

$$\hat{\mathcal{N}}_{j_1,1}^{\partial} = \hat{\mathcal{N}}_{j_1}^{\perp\partial}, \quad j_1 = 1, \ldots, 4, \tag{B.1b}$$

for the functionals from (5.54) and (5.71), respectively. The remaining Raviart-Thomas node functionals

$$\hat{\mathcal{N}}_{\boldsymbol{j}}^{\circ}, \quad \boldsymbol{j} = N_{dof}^{\perp\circ} + 1, \ldots, N_{dof}^{\circ}, \tag{B.2a}$$

$$\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^{\partial}, \quad j_1 = 1, \ldots, 4, \ \boldsymbol{j}_2 = 2, \ldots, k+1, \tag{B.2b}$$

are generated by (arbitrary) polynomials

$$\hat{\boldsymbol{w}}_{\boldsymbol{j}}, \quad \boldsymbol{j} = N_{dof}^{\perp\circ} + 1, \ldots, N_{dof}^{\circ}, \tag{B.3a}$$

$$\hat{q}_{\boldsymbol{j}_2}, \quad \boldsymbol{j}_2 = 2, \ldots, k+1, \tag{B.3b}$$

extending $\hat{p}_{\boldsymbol{j}}^{\perp}$, $\boldsymbol{j} = 1, \ldots, N_{dof}^{\perp\circ}$, and $\hat{q}_1 \equiv 1$ to form a basis of $\mathbb{Q}_{k-1,k} \times \mathbb{Q}_{k,k-1}$ and $\mathbb{Q}_k$, respectively.

We are interested in the unit shape functions $\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}$ dual to $\hat{p}_{\boldsymbol{j}}^{\perp}$,

$$\hat{\mathcal{N}}_{\boldsymbol{j}}^{\circ}(\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}) = \delta_{\boldsymbol{ij}}, \quad \boldsymbol{j} = 1, \dots, N_{dof}^{\circ}, \tag{B.4a}$$

$$\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^{\partial}(\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}) = 0, \quad j_1 = 1, \dots, 4,\ \boldsymbol{j}_2 = 1, \dots, k+1. \tag{B.4b}$$

for $\boldsymbol{i} = 1, \dots, N_{dof}^{\perp\circ}$, and unit shape functions $\hat{\boldsymbol{v}}_i^{\perp\partial}$ dual to $\hat{q}_1 \equiv 1$ and unit facet $\hat{e}_i$,

$$\hat{\mathcal{N}}_{\boldsymbol{j}}^{\circ}(\hat{\boldsymbol{v}}_i^{\perp\partial}) = 0, \quad \boldsymbol{j} = 1, \dots, N_{dof}^{\circ}, \tag{B.5a}$$

$$\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^{\partial}(\hat{\boldsymbol{v}}_i^{\perp\partial}) = \delta_{ij_1}\,\delta_{1\boldsymbol{j}_2}, \quad j_1 = 1, \dots, 4,\ \boldsymbol{j}_2 = 1, \dots, k+1. \tag{B.5b}$$

for $i = 1, \dots, 4$.

For a fixed cell $K \in \mathcal{T}_h$, global shape functions $\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}$ are defined by the Piola transform of $\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}$ onto cell K,

$$\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}(\boldsymbol{x}) = \left(\det(\hat{J}_K)^{-1}\hat{J}_K\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}\right)(\hat{\boldsymbol{x}}), \qquad \boldsymbol{x} = \boldsymbol{F}_K(\hat{\boldsymbol{x}}), \quad \hat{\boldsymbol{x}} \in \hat{K}. \tag{B.6}$$

Shape functions are extended by zero to $\Omega \setminus K$ such that $\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ} \in \boldsymbol{V}_h$. For a fixed interface $e \in \mathcal{E}_h^{\circ}$ with $e = K^+ \cap K^-$, there exist unit face indices i^+ and i^- such that $\hat{e}_{i^+}$ is mapped to e under $\boldsymbol{F}_{K^+}$ and $\hat{e}_{i^-}$ to e under $\boldsymbol{F}_{K^-}$, respectively. A corresponding global shape function $\boldsymbol{v}_e^{\perp\partial}$ is defined by each Piola transform of $\hat{\boldsymbol{v}}_i^{\perp\partial}$,

$$\boldsymbol{v}_e^{\perp\partial}|_{K^\pm}(\boldsymbol{x}) = \left(\det(\hat{J}_{K^\pm})^{-1}\hat{J}_{K^\pm}\hat{\boldsymbol{v}}_{i^\pm}^{\perp\partial}\right)(\hat{\boldsymbol{x}}), \qquad \boldsymbol{x} = \boldsymbol{F}_{K^\pm}(\hat{\boldsymbol{x}}), \quad \hat{\boldsymbol{x}} \in \hat{K}. \tag{B.7}$$

The shape function is extended by zero to $\Omega \setminus (K^+ \cup K^-)$ such that $\boldsymbol{v}_e^{\perp\partial} \in \boldsymbol{V}_h$.

Before stating the lemma that presents a shape function basis of $\boldsymbol{V}_h^{\perp}$, we define a subset of interfaces $\tilde{\mathcal{E}}_h^{\circ} \subset \mathcal{E}_h^{\circ}$ by recursion:

Remark B.1.1. We choose a cell $K_0 \in \mathcal{T}_h$ and define a set of cells $\mathcal{K}_h = \{K_0\}$. The set of interfaces $\tilde{\mathcal{E}}_h^{\circ}$ is recursively defined by

(S1) For each $K \in \mathcal{K}_h$: If $\tilde{K} \in \mathcal{T}_h$ is adjacent to K with interface $e = K \cap \tilde{K}$ and $\tilde{K} \notin \mathcal{K}_h$ add the cell $\tilde{K}$ to $\mathcal{K}_h$ and the interface e to $\tilde{\mathcal{E}}_h^{\circ}$, respectively.

(S2) If $\mathcal{K}_h \neq \mathcal{T}_h$ return to Step (S1), otherwise stop.

We note that computing $\tilde{\mathcal{E}}_h^{\circ}$ boils down to computing a graph problem which may be solved by a more efficient (parallel) algorithm than the simple (pseudo-)algorithm stated in Remark B.1.1, especially for many interfaces. However, in this monograph we are only interested in finding $\tilde{\mathcal{E}}_h^{\circ}$ for vertex patches. Consequently, the pseudo-algorithm in Remark B.1.1 suffices us.

Lemma B.1.2. *Let $\tilde{\mathcal{E}}_h^\circ$ be as defined in Remark B.1.1, and $\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}$ and $\boldsymbol{v}_e^{\perp\partial}$ as defined in (B.6) and (B.7), respectively. Then, the shape functions*

$$\begin{aligned} &\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}, && K \in \mathcal{T}_h,\ \boldsymbol{i} = 1, \dots, N_{dof}^{\perp\circ}, \\ &\boldsymbol{v}_e^{\perp\partial}, && e \in \tilde{\mathcal{E}}_h^\circ, \end{aligned}$$

form a basis of $\boldsymbol{V}_h^\perp$.

Proof. Integrating the characteristic equation (5.49) of $\boldsymbol{V}_h^\perp$ by parts,

$$(\boldsymbol{w}, \boldsymbol{v}_h)_\Omega = -(\nabla \cdot \boldsymbol{w}, q_h)_\Omega = -\int_{\mathcal{E}_h^\circ} \boldsymbol{w} \cdot [\![q_h \boldsymbol{n}]\!] \,\mathrm{d}\sigma(\boldsymbol{x}) + \int_{\mathcal{T}_h} \boldsymbol{w} \cdot \boldsymbol{\nabla} q_h \,\mathrm{d}\boldsymbol{x},$$

it directly follows that $\boldsymbol{v}_{K;i}^{\perp\circ}$ and $\boldsymbol{v}_e^{\perp\partial}$ are elements in $\boldsymbol{V}_h^\perp$, for any $K \in \mathcal{T}_h$, $\boldsymbol{i} = 1, \dots, N_{dof}^{\perp\circ}$, and $e \in \tilde{\mathcal{E}}_h^\circ$, respectively. The linear independence follows from the same arguments that proved the unisolvence of the Raviart-Thomas element, see the textbook (Boffi et al., 2013) or original works by Nédélec (1980) and Raviart and Thomas (1977).

Moreover, it holds $\nabla \cdot \mathrm{RT}_k = \mathbb{Q}_k$, see for instance (Boffi et al., 2013). Therefore, choosing $N_{dof}^{\perp\circ} = (\dim \mathbb{Q}_k - 1)$ basis shape functions $\boldsymbol{v}_{K;i}^\circ$ for each cell K, there is exactly one degree of freedom per cell left to define a basis for $\boldsymbol{V}_h^\perp$. The remaining shape functions $\boldsymbol{v}_e^{\perp\partial}$ for each facet $e \in \tilde{\mathcal{E}}_h^\circ$ add up to $(N_{cell} - 1)$, where N_{cell} denotes the number of cells in $\mathcal{T}_h$. While it seems we are lacking a single shape function, the identity $\boldsymbol{V}_h^\perp = \boldsymbol{\nabla}_h Q_h$ known from the Hodge decomposition (5.68) reveals that $(N_{cell} - 1)$ basis functions are sufficient: the mean value condition in Q_h leads to one constraint degree of freedom for $\boldsymbol{V}_h^\perp$. Therefore, counting dimensions concludes this proof. We note that by involving no basis shape functions $\boldsymbol{v}_e^{\perp\partial}$ for facets $e \in \mathcal{E}_h^\partial$, the boundary conditions inherited from $\boldsymbol{V}_h$ are implicitly imposed. □

We emphasize that we do not need to use the generating polynomials from (B.1). This assumption was made to simplify the derivation of Lemma B.1.2. In computations, we make use of ansatz polynomials in Definition 5.2.5 (Raviart-Thomas elements) that provide us with numerical stability, in particular, for high polynomial degrees. Nevertheless, the restriction from Raviart-Thomas shape functions to those shape functions generated by the subset of node functionals (5.71) on the unit cell is readily computed. To this end, let

$$\begin{aligned} &\hat{\boldsymbol{v}}_{\boldsymbol{i}}^\circ, && \boldsymbol{i} = 1, \dots, N_{dof}^\circ, \\ &\hat{\boldsymbol{v}}_{i_1,\boldsymbol{i}_2}^\partial, && i_1 = 1, \dots, 4,\ \boldsymbol{i}_2 = 1, \dots, N_{dof}^e, \end{aligned}$$

denote the dual basis to the Raviart-Thomas node functionals (5.54). For simplicity, we simplify the notation to $\hat{\boldsymbol{v}}_i$ utilizing a contiguous index i: thus, enumerating first the functions $\hat{\boldsymbol{v}}_{\boldsymbol{i}}^\circ$, and then the functions $\hat{\boldsymbol{v}}_{i_1,\boldsymbol{i}_2}^\partial$, where i_1 strides faster than $\boldsymbol{i}_2$. We proceed similarly with the Raviart-Thomas node functionals, $\hat{\mathcal{N}}_j^\perp$, the basis functions from (B.4) and (B.5), $\hat{\boldsymbol{v}}_i^\perp$, and

their associated node functionals from (5.71), $\hat{\mathcal{N}}_j^\perp$. We seek a representation

$$\hat{\boldsymbol{v}}_i^\perp = \sum_{l=1}^{N_{dof}} \alpha_{il} \hat{\boldsymbol{v}}_l. \tag{B.10}$$

Due to the duality of $\hat{\boldsymbol{v}}_i^\perp$ to $\hat{\mathcal{N}}_j^\perp$, there holds

$$\hat{\mathcal{N}}_j^\perp(\sum_{l=1}^{N_{dof}} \alpha_{il} \hat{\boldsymbol{v}}_l) = \delta_{ij}. \tag{B.11}$$

Given rectangular matrices $A = (\alpha_{ij})_{ij}$ and $N = (\hat{\mathcal{N}}_i^\perp(\hat{\boldsymbol{v}}_j))_{ij}$, from (B.11) it follows

$$A = N^{-\mathsf{T}}. \tag{B.12}$$

The pseudo-inverse of N^T is computed via singular value decomposition.

In Section 5.2.3 we skipped the proofs of Lemma 5.2.8 and Lemma 5.2.9, amending them now.

Proof of Lemma 5.2.8. We recall that $p_h \in Q_h$ admits the decomposition (5.70),

$$p_h = \sum_{K \in \mathcal{T}_h} p_K^0 + p_K^\perp,$$

with

$$p_K^\perp = \sum_{\boldsymbol{j}=1}^{N_{dof}^{\perp\circ}} \alpha_{\boldsymbol{j}}^K (\hat{p}_{\boldsymbol{j}}^\perp \circ \boldsymbol{F}_K^{-1}).$$

Testing against $\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}$ in (5.66) and using (5.72), we obtain

$$-\sum_{e \subset \partial K} \int_e \boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ} \cdot p_h \boldsymbol{n} \,\mathrm{d}\sigma(\boldsymbol{x}) + \int_K \boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ} \cdot \boldsymbol{\nabla} p_h \,\mathrm{d}\boldsymbol{x} = F_h(\boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}) - \widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi_h, \boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ}) \tag{B.13}$$

for $\boldsymbol{i} = 1, \ldots, N_{dof}^{\perp\circ}$. The integrals over facets vanish by definition (see (B.4) and (B.6)) of shape functions, for any cell $K \in \mathcal{T}_h$, any facet $e \subset \partial K$ and any local index $\boldsymbol{i}$. To be precise, due to the Piola transform's preservation property (5.59c) it holds

$$\int_e \boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ} \cdot p_h \boldsymbol{n} \,\mathrm{d}\sigma(\boldsymbol{x}) = \int_{\hat{e}} \hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ} \cdot \hat{p} \hat{\boldsymbol{n}} \,\mathrm{d}\sigma(\hat{\boldsymbol{x}}) = 0, \tag{B.14a}$$

where $\hat{p} = p_h \circ \boldsymbol{F}_K$. The integral over the unit facet is a linear combination of the functionals $\hat{\mathcal{N}}_{j_1,\boldsymbol{j}_2}^\partial$ evaluated in $\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}$ and, consequently, equals zero. Using similar arguments, in particular,

preservation property (5.59a), it holds

$$\int_K \boldsymbol{v}_{K;\boldsymbol{i}}^{\perp\circ} \cdot \boldsymbol{\nabla} p_h \,\mathrm{d}\boldsymbol{x} = \sum_{\boldsymbol{j}=1}^{N_{dof}^{\perp\circ}} \alpha_{\boldsymbol{j}}^K \int_{\hat{K}} \hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ} \cdot \hat{\boldsymbol{\nabla}} \hat{p}_{\boldsymbol{j}}^{\perp} \, d\hat{\boldsymbol{x}} = \alpha_{\boldsymbol{i}}^K$$

The last equation follows from $\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\circ}$ being dual to $\hat{\mathcal{N}}_{\boldsymbol{j}}^{\perp\circ}$. Then, from (B.13) immediately follows (5.73). □

Proof of Lemma 5.2.9. The proof follows the same structure as the previous previous proof, thus keeping it concise. Without loss of generality we choose K^+ such that the unit normal $\hat{\boldsymbol{n}}$ mapped to $\boldsymbol{n}^+$ points in positive direction. Inserting $\boldsymbol{v}_e^{\partial}$ into (5.66) and (5.72), it holds

$$\begin{aligned} &F_h(\boldsymbol{v}_e^{\perp\partial}) - \widetilde{a}_{div;h}(\boldsymbol{\nabla} \times \phi_h, \boldsymbol{v}_e^{\perp\partial}) \\ &\quad = -\int_e \boldsymbol{v}_e^{\perp\partial} \cdot [\![p_h \boldsymbol{n}]\!] \,\mathrm{d}\sigma(\boldsymbol{x}) + \int_{K^+} \boldsymbol{v}_e^{\perp\partial} \cdot \boldsymbol{\nabla} p_h \,\mathrm{d}\boldsymbol{x} + \int_{K^-} \boldsymbol{v}_e^{\perp\partial} \cdot \boldsymbol{\nabla} p_h \,\mathrm{d}\boldsymbol{x} \\ &\quad = -(p_{K^+}^0 - p_{K^-}^0) \int_e \boldsymbol{v}_e^{\perp\partial} \cdot \boldsymbol{n}^+ \,\mathrm{d}\sigma(\boldsymbol{x}) - \int_e \boldsymbol{v}_e^{\perp\partial} \cdot (p_{K^+}^{\perp} \boldsymbol{n}^+ + p_{K^-}^{\perp} \boldsymbol{n}^-) \,\mathrm{d}\sigma(\boldsymbol{x}), \end{aligned} \tag{B.15}$$

The integrals over K^+ and K^- vanish because by definition it holds $\hat{\mathcal{N}}_{\boldsymbol{j}}^{\circ}(\hat{\boldsymbol{v}}_{\boldsymbol{i}}^{\perp\partial}) = 0$. The definition of $\boldsymbol{v}_e^{\perp\partial}$ implies $\int_e \boldsymbol{v}_e^{\perp\partial} \cdot \boldsymbol{n}^+ \,\mathrm{d}\sigma(\boldsymbol{x}) = 1$, thus, (5.74) immediately follows from (B.15). □

B.2 Modified Stream Function Formulation

It remains to derive the low-rank tensor representation (5.97) given the local integral

$$\int_{\mathcal{T}_j} \boldsymbol{\nabla} \boldsymbol{u}^{\mathsf{T}} : \boldsymbol{\nabla} \boldsymbol{v} \,\mathrm{d}\boldsymbol{x} = \sum_{c=1}^{D} \int_{\mathcal{T}_j} \partial_c \boldsymbol{u}_c \partial_c \boldsymbol{v}_c \,\mathrm{d}\boldsymbol{x} + \sum_{c,d=1;c\neq d}^{D} \int_{\mathcal{T}_j} \partial_c \boldsymbol{u}_d \partial_d \boldsymbol{v}_c \,\mathrm{d}\boldsymbol{x} \tag{B.16}$$

for the generic vertex patch $\mathcal{T}_j$. In two dimensions, the vector curl of a scalar field ψ was defined in (5.39). Substituting $\boldsymbol{u}$ and $\boldsymbol{v}$ by $\boldsymbol{\nabla} \times \phi$ and $\boldsymbol{\nabla} \times \psi$ in the right-hand side of (B.16), we obtain

$$\sum_{c=1}^{2} \int_{\mathcal{T}_j} \partial_c \boldsymbol{u}_c \partial_c \boldsymbol{v}_c \,\mathrm{d}\boldsymbol{x} = \int_{\mathcal{T}_j} \partial_{12}\phi \partial_{12}\psi + \partial_{21}\phi \partial_{21}\psi \,\mathrm{d}\boldsymbol{x} \tag{B.17}$$

and

$$\sum_{c,d=1;c\neq d}^{2} \int_{\mathcal{T}_j} \partial_c \boldsymbol{u}_d \partial_d \boldsymbol{v}_c \,\mathrm{d}\boldsymbol{x} = \int_{\mathcal{T}_j} -\partial_{11}\phi \partial_{22}\psi - \partial_{22}\phi \partial_{11}\psi \,\mathrm{d}\boldsymbol{x}. \tag{B.18}$$

The first integral is known from Section 4.1.2, resulting in the low-rank representation

$$2\boldsymbol{L}^{(1)} \otimes \boldsymbol{L}^{(2)}. \tag{B.19}$$

In analogy to (4.40) and (4.41), we define the one-dimensional discretization matrix

$$\boldsymbol{H}^{(d)} = \begin{bmatrix} H_{++}^{(d)} & H_{+e}^{(d)} & 0 \\ (H_{+e}^{(d)})^{\mathsf{T}} & H_{ee}^{(d)} & (H_{-e}^{(d)})^{\mathsf{T}} \\ 0 & H_{-e}^{(d)} & H_{--}^{(d)} \end{bmatrix} \tag{B.20}$$

with

$$\left(H_{\pm\pm}^{(d)}\right)_{ij} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{\pm})} \hat{\phi}_{i+1}''(\hat{x}_q) \hat{\phi}_{j+1}(\hat{x}_q) w_q \right), \tag{B.21a}$$

$$\left(H_{ee}^{(d)}\right)_{11} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{+})} \hat{\phi}_{n_{dof}}''(\hat{x}_q) \hat{\phi}_{n_{dof}}(\hat{x}_q) w_q + \frac{1}{(h_d^{-})} \hat{\phi}_1''(\hat{x}_q) \hat{\phi}_1(\hat{x}_q) w_q \right), \tag{B.21b}$$

$$\left(H_{+e}^{(d)}\right)_{i1} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{+})} \hat{\phi}_{i+1}''(\hat{x}_q) \hat{\phi}_{n_{dof}}(\hat{x}_q) w_q \right), \tag{B.21c}$$

$$\left(H_{-e}^{(d)}\right)_{i1} = \sum_{q=1}^{n_{quad}} \left(\frac{1}{(h_d^{-})} \hat{\phi}_{i+1}''(\hat{x}_q) \hat{\phi}_1(\hat{x}_q) w_q \right), \tag{B.21d}$$

for $i = 1, \ldots, n_{dof}-2$ and $j = 1, \ldots, n_{dof}-2$. Then, (B.18) admits the low-rank representation

$$- \boldsymbol{H}^{(1)} \otimes (\boldsymbol{H}^{(2)})^{\mathsf{T}} - (\boldsymbol{H}^{(1)})^{\mathsf{T}} \otimes \boldsymbol{H}^{(2)}, \tag{B.22}$$

Combining (B.19) and (B.22), we finally derive the rank-5 tensor product representation (5.97). Proceeding similarly for the face integrals

$$\int_{\mathcal{E}_\ell} \left(- \left\{ \boldsymbol{\nabla} \boldsymbol{u}^{\mathsf{T}} \right\} : [\![\boldsymbol{n} \otimes \boldsymbol{v}]\!] - [\![\boldsymbol{n} \otimes \boldsymbol{u}]\!] : \left\{ \boldsymbol{\nabla} \boldsymbol{v}^{\mathsf{T}} \right\} \right) \mathrm{d}\sigma(\boldsymbol{x}),$$

we may define matrices $\underline{\boldsymbol{H}}$ in analogy to $\underline{\boldsymbol{B}}$ from Section 4.1.2.