SISSA
Scuola
Internazionale
Superiore di
Studi Avanzati

Mathematics Area - PhD course in
Mathematical Analysis, Modelling, and Applications

Non-matching and polytopic finite element techniques with applications to multilevel methods

Candidate:
Marco Feder

Advisors:
Prof. Andrea Cangiani
Prof. Luca Heltai

Academic Year 2023-2024

MA PER SEGUIR VIRTUTE E CANOSCENZA • SISSA •

To my family

"... not because they are easy, but because they are hard."

John Fitzgerald Kennedy - *Moon speech*

Abstract

The numerical approximation of partial differential equations (PDEs) involves the challenge of handling complex geometries and their discretization. In this thesis, we focus on different computational aspects that can be applied in a large variety of scientific and industrial contexts.

The first part of the thesis delves into the complexities of managing non-matching grids. Indeed, a common feature to many multi-physics problems is the need for the transfer of data or information between different meshes. We first consider the problem of computing coupling matrices, which require the integration of functions defined on different, arbitrarily overlapped, meshes. We discuss the relevant implementation details and provide a comparison between different unfitted methods. Additionally, we note that the transfer of discrete fields plays a crucial role in several other contexts, e.g. within multilevel methods. Motivated by the excellent properties of multilevel solvers and the performance gain given by matrix-free methodologies, we present a parallel and matrix-free implementation of the non-nested multigrid method. It allows for completely independent and distributed multigrid levels, thereby increasing flexibility on the choice of the hierarchy, while avoiding the explicit assembly of sparse matrices.

The second part is devoted to the task of implementing efficient agglomeration procedures, within the polytopic discontinuous Galerkin setting. We develop a novel and efficient approach to perform grid agglomeration using spatial data structures, and validate its robustness and performances also in the memory-distributed setting. Such a coarsening strategy is particularly appealing for multigrid methodologies, as it can deliver a hierarchy of nested grids out of a given geometry. We successfully exploited such versatility in the realistic setting of cardiac electrophysiology, by using our agglomeration procedure to build a multilevel preconditioner for a DG discretization of the monodomain problem. Finally, a preliminary investigation aimed at proving the convergence properties of our multilevel strategy is presented.

Many results presented in this thesis are also software contributions integrated into the C++ finite element library DEAL.II.

Table of contents

List of figures **xiii**

List of tables **xix**

0 Introduction **1**

I Non-matching techniques **7**

1 Non-matching methods **9**

1.1 Literature review 9

1.2 Model problem and notation 11

1.3 Non-matching discretizations 13

1.3.1 The method of Lagrange Multipliers 14

1.3.2 The cut-FEM method 17

1.4 Integration of coupling terms 19

1.4.1 Integration driven by the immersed mesh 20

1.4.2 Integration on mesh intersections 21

1.4.3 Integration through level set splitting 22

1.5 Numerical experiments 25

1.5.1 2D numerical tests 26

1.5.2 3D numerical tests 33

1.5.3 Computational times 37

1.6 Conclusions 38

2 Non-nested multigrid method **41**

2.1 Literature review 42

2.2 Non-nested multigrid method 43

2.2.1 Polynomial global coarsening 46

2.3 Implementation details 47

2.3.1 Distributed geometric search 48

2.3.2 Efficient evaluation on reference positions . . . 50
2.3.3 Domains with curved boundaries . . . 51
2.4 Numerical experiments . . . 52
2.4.1 Application to nested meshes . . . 54
2.4.2 L-shaped domains . . . 55
2.4.3 Applications to FEA . . . 57
2.5 Performance evaluation . . . 63
2.5.1 Serial and parallel runs . . . 64
2.5.2 Non-nested tests . . . 66
2.6 Conclusions . . . 71

II Polytopic Finite Elements **73**

3 Agglomeration of polytopic grids **75**
3.1 Literature review . . . 76
3.2 R-trees . . . 78
3.2.1 R-trees and finite element meshes . . . 80
3.3 Agglomeration based on R-trees . . . 81
3.4 Validation . . . 83
3.4.1 Validation on a set of grids . . . 83
3.4.2 Quality of resulting elements . . . 86
3.4.3 Performance validation . . . 91
3.5 Polytopic discontinuous Galerkin . . . 92
3.5.1 Notation and model problem . . . 92
3.5.2 Finite element spaces and trace operators . . . 92
3.5.3 Convergence tests . . . 95
3.5.4 Multigrid preconditioning . . . 98
3.6 Implementation details . . . 102
3.6.1 Data structures and interfaces . . . 102
3.6.2 Memory distributed implementation . . . 105
3.6.3 Building the sparse operator . . . 107
3.6.4 Benchmarks . . . 108
3.7 Conclusions . . . 111

4 Convergence analysis for multigrid **113**
4.1 Literature review . . . 113
4.2 Model problem . . . 114
4.2.1 Grid assumptions . . . 115
4.3 Inverse estimates . . . 118

4.3.1 R-tree agglomeration and p-coverability 120
4.3.2 A priori error bounds 121
4.4 Convergence analysis for the multigrid algorithm 123
4.4.1 Convergence of the two-level method 128
4.4.2 Convergence of the W-cycle algorithm 129
4.5 Conclusions 131

5 Application to cardiac electrophysiology 133
5.1 Literature review 133
5.2 Mathematical model 135
5.2.1 Monodomain problem 135
5.2.2 Ionic models 136
5.3 Space and time discretization 138
5.3.1 Semidiscrete formulation 138
5.4 Multilevel preconditioning based on polytopes 140
5.4.1 Inherited bilinear forms 140
5.5 Numerical experiments 142
5.5.1 Two-dimensional test case 142
5.5.2 Three-dimensional test case 144
5.5.3 Dependence on number of levels 147
5.6 Conclusions and perspectives 152

6 Conclusions and final remarks 155
6.1 Future perspectives 156

References 159

List of figures

1 Thesis structure: the first part focuses on computational aspects of non-matching methodologies, the second part on agglomeration techniques, which are applied to polytopic discontinuous Galerkin methods and multigrid preconditioning. 5

1.1 Model problem setting, with immersed domain ω, immersed boundary γ, and background domain Ω. 15

1.2 Squares: DoF for linear basis functions attached to some elements K_j of γ intersecting a background element $T_i \in \Omega_h$. Dots: DoF for a linear Lagrangian basis attached to cells T_i. Crosses: quadrature points corresponding to a Gaussian quadrature rule of order 3 on elements of γ. 21

1.3 Comparison between exact and non exact integration (zoomed-in on the interface in order to highlight the kinks) for a pathological case. (a) Spurious kinks around the interface, on the location of the immersed quadrature points. (b) Well-resolved solution around the interface. 21

1.4 Triangulation of the intersected region $\tilde{E}$ for a line immersed in 2D. The one-dimensional cell K is allowed to be positioned arbitrarily w.r.t. to the two-dimensional quadrilateral T. 23

1.5 Triangulation of the intersected region $\tilde{E}$ for a triangle cutting a square and the relative sub-tessellation. 24

1.6 Quadrature points distributed on O, S and I for a cell T cut by the curve γ. 24

1.7 The relative error for the discrete level set describing a disk of radius $R = 0.3$ as a function of the distance from the center $\boldsymbol{x_c} = (\frac{1}{2}, \frac{1}{2})$. 25

1.8 Zoom on pre-processed background grid Ω_h for the circle interface $\gamma = \gamma^1$ (left) and the flower-shaped interface $\gamma = \gamma^2$ (right). 27

1.9 L^2, H^1, and $H^{-\frac{1}{2}}$ errors versus the number of DoF for all schemes applied to γ^1 (left) and to γ^2 (right). 30

1.10 L^2, H^1, and $H^{-\frac{1}{2}}$ errors versus the number of DoF for all schemes applied to γ^1, with non-smooth solution. 33

1.11 L^2, H^1, and $H^{-\frac{1}{2}}$ error versus the number of DoF for all schemes applied to γ^3 with a smooth solution u (left) and with a non-smooth solution u (right). 37

1.12 Background mesh Ω_h and immersed mesh of the sphere interface γ_h for the three-dimensional case (left) and section of the contour plot for the approximate solution u_h in (1.29), $\gamma = \gamma^3$. 38
1.13 Breakdown of CPU times for the three schemes. In (bottom left) *Setup level set* indicates the time required to interpolate the discrete level set described in Algorithm 1 onto the finite element space. *Mesh classification* shows the time needed to partition the computational mesh and classify the cells in *cut, interior* or *outside* cells. Bottom right: overall CPU times to assemble the algebraic system for the smooth 3D test. For the cut-FEM method, we also show the CPU time obtained by using an analytical representation of the interface through an analytic level set function. 39
1.14 Work-precision diagrams for the two 3D tests. 40

2.1 Two overlapping cells coming from consecutive levels. Green dots: DoF associated to a $\mathcal{Q}^1$ element on the coarser cell K. Blue squares and red stars: DoF associated to a $\mathcal{Q}^1$ element on the finer cell T. Red stars correspond to the DoF $(\boldsymbol{p}_2, \boldsymbol{p}_4)$ falling inside $K \in \mathcal{T}_l$, while blue squares are the ones that are falling outside. When evaluating the contribution of this particular coarse cell to the DoF of the fine cell, each basis function related to such DoF $\boldsymbol{q}_i$, $i = 1, \ldots, 4$, will evaluate on $(\boldsymbol{p}_2, \boldsymbol{p}_4)$ only. 46
2.2 Schematic illustration of *hp*-multigrid scheme for a $\mathcal{Q}^3$ element. DoF corresponding to continuous Lagrangian elements are represented with white dots. Left: Classical nested setting. Right: Non-nested *hp* variant where the hierarchy of levels is *non-matching*. Notice how the `CoarseGridSolver` is set to polynomial multigrid (p-MG). 48
2.3 UML diagram of transfer operators available in the `MGTransferGlobalCoarsening` framework in DEAL.II. The new abstract class delegates the implementation of the intergrid transfers to the derived class `MGTwoLevelTransfer` (used in case of nested meshes) or to the new `MGTwoLevelTransferNonNested` in case of a non-nested multigrid method. Each two-level transfer object is specific to consecutive levels l and $l+1$. 49
2.4 Coupling between processors for two overlapped and distributed triangulations (each color represents a different MPI rank). (a) The two partitioned triangulations. The cube is displayed with a wireframe view in order to show the inner ball. (b) Clip view to highlight the coupling between ranks. Notice that grids are discretizing two different geometries only for the purpose of showing the issue. In practice, they will both discretize the same domain Ω. 50
2.5 (a) Gauss-Lobatto points for a quadrature rule of order $p = 4$ on a cell $T \in \mathcal{T}_{l+1}$. (b) Evaluation points seen from the coarser cell $K \in \mathcal{T}_l$ (red stars) do not have a tensor-product structure. 51

2.6 Illustration of the situation occurring when the hierarchy discretizes a two-dimensional domain Ω with curved boundaries. (a) Red dash-dotted lines: elements of a coarser triangulation $\mathcal{T}_l$. Green dashed lines: boundary for a finer triangulation $\mathcal{T}_{l+1}$. Black solid line: exact representation of the boundary $\partial\Omega$. (b) Pre-image of point $\boldsymbol{p}$ through F_K and its nearest point projection $\hat{\boldsymbol{p}} \in \overline{\hat{K}}$ defined according to (2.19). 53
2.7 Four of the levels employed for the L-shaped domain, refined near the re-entrant corner. 56
2.8 Consecutive levels employed for the Fichera test, refined near the re-entrant corner. . 56
2.9 Scatter plot with times required by piston test case with $l = 3$ while varying solver and polynomial degree p. 60
2.10 Left: CAD model used in the preprocessing procedure. Right: Two different levels. For the sake of visualization, the finer level is displayed in orange using a wireframe representation. The coarser level is represented as a volumetric mesh (edges in blue). Notice that each level has not been generated on top of the coarser one through a global or local refinement process. 61
2.11 Left: Magnitude of the displacement vector $\boldsymbol{u}$ for the piston test case. Right: Scaled view of the vector field u. 61
2.12 Magnitude (scaled) of the displacement $\boldsymbol{u}$ for the wrench test case. 62
2.13 Scatter plot with times required by wrench test case for $l = 3$ while varying solver and polynomial degree p. 63
2.14 Serial profiles of a V-cycle. Left: `Cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$. 65
2.15 Profile of a V-cycle with 12 processes with non-nested multigrid. Left: `Cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$. 66
2.16 Profile of a V-cycle with 12 processes with non-nested multigrid using a degree 5 Chebyshev smoother. Left: `Cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$. 67
2.17 Profile of a V-cycle with 12 processes with global coarsening variant. Left: `cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$. Notice how, compared to the previous Figure 2.15, the only major difference are bar charts associated to prolongation and restriction. 68
2.18 Breakdown of the computational time (in seconds) required for prolongation between levels with 12 processes. In blue: total time spent to prolongate a coarse finite element field from level l to $l + 1$. In red: time spent entirely on the evaluation at arbitrarily located reference points. 69
2.19 Exclusive time in a V-cycle with 12 processes for the `cube` example with $L = 7$ and $p = 4$. Right: global coarsening. Left: non-nested multigrid. 69
2.20 Clipped view of consecutive levels in the hierarchy for the piston test case 2.10 partitioned across 12 processors (each color represents a different MPI rank). It can be appreciated how a fixed MPI rank owns different regions when the mesh is refined. 70

3.1 Example of MBRs holding geometric data and their MBRs. 79
3.2 Corresponding R-tree data structure. 79
3.3 Left: 8×8 square grid obtained with 3 uniform refinements of $\Omega = [0,1]^2$. Right: Minimal bounding rectangles generated on top of the grid elements of Ω. 81
3.4 R-tree data structure for the 8×8 mesh example. For the sake of readability, only two internal nodes are shown. Each entry at the leaf level is one mesh element T. Notice how each node stores exactly 4 entries. 81
3.5 The set of meshes used in the numerical experiments. 84
3.6 Comparison between METIS and R-tree based agglomeration starting from the grid Ω_1 displayed in Figure 3.5a. Grids displayed in the same row always comprise the same number of elements. 85
3.7 Comparison between METIS and R-tree based agglomeration of the grid Ω_2 shown in Figure 3.5b. Grids displayed in the same row always comprise the same number of elements. 87
3.8 Comparison between METIS and R-tree agglomeration starting from the grid Ω_3 seen in Figure 3.5c. Grids displayed in the same row always comprise the same number of elements. 88
3.9 Min, Max and Average chart of metrics UF, CR, and BR for grids $\Omega_1, \Omega_2, \Omega_3$ and different agglomeration strategies. In the plots on the left, metrics related to the R-tree collapse to a single dot since all mesh elements have the same value for that particular metric. 90
3.10 Min, Max and Average chart of metrics UF, CR, and BR for grids $\Omega_1, \Omega_2, \Omega_3$ and different agglomeration strategies, after creation of one more level. In the plots on the left, metrics related to the R-tree collapse to a single dot since all mesh elements have the same value for that particular metric. 91
3.11 Wall clock time (in seconds) needed to build polytopic grids with R-tree and METIS. 93
3.12 Problem (3.8) with $d = 2$. Convergence under p-refinement for $p = 1,2,3,4,5$. . . . 96
3.13 Problem (3.8) with $d = 3$. Convergence under p-refinement for $p = 1,2,3,4$. 97
3.14 Left: sample agglomerates generated by the R-tree algorithm. Right: view of the solution. 98
3.15 Convergence under p-refinement for the piston test case for $p = 1,2,3,4$. Fixed polytopic grids made of 731 agglomerates. 98
3.16 Left: Clip of the piston geometry Ω_5 with colour plot of the solution computed with $p = 1$. In the foreground, highlighted in grey, a single element out of the 90 elements level grid. Right: detailed view of the highlighted element (wireframe) and its 8 sub-agglomerates belonging to the finer level. For better visualisation, the sub-agglomerates, identified by different colours, are displayed in either of the two plots. 101

3.17 A patch of pentagonal cells (thick black lines) is created after agglomeration of underlying triangles. Within each patch, the master cell taking care of the enumeration of DoF is highlighted with a red-dashed square. The new *global* enumeration of the three polygonal cells is $\{0,1,2\}$ and elements are indicated with K_0^A, K_1^A, K_2^A. 103
3.18 Left: view from process 0 of the locally owned (red), ghosted (yellow), and remote (blue) quadrilateral cells. Right: on the same grid, a patch of cells (orange) is created after agglomeration of some quadrilateral elements on process 0, while the dark green agglomerate has been created on process 1. The dashed squares flag the respective master cells. For instance, the three cells on top of K_1^A are fully remote cells for which process 0 has no information. 107
3.19 Strong scaling experiment with $p = 2$ on a *polytopic* mesh consisting of $N_A = 5\,000$ agglomerates built on top of a fine hexahedral mesh Ω_h of 262 144 elements. 110
3.20 Strong scaling experiment with $p = 2$ on a *polytopic* mesh consisting of $N_A = 70\,000$ agglomerates built on top of a fine hexahedral mesh Ω_h of 2 097 152 elements. . . . 111

4.1 Elements $K \in \mathcal{T}^{tr}$ are assumed to satisfy Assumption 4.2.1 (a) (left) and (b) (right); $\boldsymbol{\cdot}$ denotes a vertex. 116
4.2 Curved elements K, K'' with, respectively, 8 and 4 sub-elements satisfying Assumption 4.2.1. 116
4.3 One agglomerate, denoted by K, and its associated bounding box B_K with diameter H_K (indicated with the diagonal, red, dashed line). Right: zoom on the right boundary, showing the fine mesh edges. 117
4.4 A reference generalized prism $\hat{K}$ for $d = 2$. 118

5.1 Ω_h partitioned among 12 processors using PARMETIS. Each color maps to one MPI rank. 143
5.2 Snapshots of the transmembrane potential $\boldsymbol{U}$ after the external application of the current $\mathcal{I}_{app}(\boldsymbol{x},t)$ in Equation (5.13). 144
5.4 Detailed views of the hexahedral mesh Ω_h representing a left ventricle, with a zoom at the bottom of the ventricle to highlight (in red) the point where $\mathcal{I}_{app}(\boldsymbol{x},t)$ is applied. 144
5.3 Number of CG iterations per time step for Problem (5.1), varying polynomial degree p and the number of sweeps. 145
5.5 Number of CG iterations per time step for Problem (5.1) for the 3D ventricle test case with $p = 1$. 147
5.6 Number of CG iterations per time step for Problem (5.1) for the 3D ventricle test case with $p = 2$. 148
5.7 Snapshots of the transmembrane potential $\boldsymbol{U}$ at selected time steps after the external application of the current $\mathcal{I}_{app}(\boldsymbol{x},t)$ at one point. 149
5.8 Hexahedral mesh of an ellipsoid representing an idealized left ventricle. 150

5.9 Number of CG iterations per time step for Problem (5.9) for the idealized ventricle test case with $p = 1$ and different number of levels. 150
5.10 Number of CG iterations per time step for Problem (5.9) for the idealized ventricle test case with $p = 2$ and different number of levels. 151

List of tables

1.1 Rates in L^2 and H^1 for a smooth u and $H^{-\frac{1}{2}}$ rates for the Lagrange multiplier method. 28

1.2 L^2 and H^1 error rates for $\gamma = \gamma^1$ and a smooth solution with Nitsche. 29

1.3 L^2 and H^1 error rates for $\gamma = \gamma^1$ and a smooth solution with cut-FEM. 29

1.4 L^2 and H^1 error rates for $\gamma = \gamma^2$ and a smooth solution with Lagrange multiplier method. 30

1.5 L^2 and H^1 error rates for $\gamma = \gamma^2$ and a smooth solution with Nitsche 31

1.6 L^2 and H^1 error rates for $\gamma = \gamma^2$ and a smooth solution with cut-FEM 31

1.7 L^2-error and H^1-error for non smooth u in (1.26) and for the multiplier. 32

1.8 L^2-error, H^1-error for non smooth u in (1.26). 32

1.9 L^2-error, H^1-error for non smooth u in (1.26) with cut-FEM. 33

1.10 L^2 and H^1 error rates for $\gamma = \gamma^3$ and a smooth solution with Lagrange multiplier. . . 34

1.11 L^2 and H^1 error rates for $\gamma = \gamma^3$ and a smooth solution with Nitsche. 34

1.12 L^2 and H^1 error rates for $\gamma = \gamma^3$ and a smooth solution with cut-FEM 35

1.13 L^2 and H^1 error rates for $\gamma = \gamma^3$ and non-smooth solution u in (1.29) with Lagrange multiplier. 36

1.14 Rates for $\gamma = \gamma^3$ and a non-smooth solution u in (1.29) with Nitsche. 36

1.15 L^2 and H^1 error for $\gamma = \gamma^3$ and a non-smooth solution u in (1.29) with cut-FEM . . . 36

2.1 *#i*: number of iterations for non-nested multigrid in two space dimensions. *#i_n*: number of iterations for the nested version. *#i_a*: number of iterations required by AMG. 54

2.2 *#i*: number of iterations for non-nested multigrid in three space dimensions. *#i_n*: number of iterations for the nested version. *#i_a*: number of iterations required by AMG. 55

2.3 Number of required iterations and DoF per level to solve the system with the L-shaped domain. 55

2.4 Number of required iterations and DoF per level to solve the system with the L-shaped domain and a coarser structured level with 192 elements as first level for each polynomial degree. 56

2.5 Number of required iterations and DoF per level to solve the system with the Fichera's corner test. 57

2.6 In each row we report the polynomial space and the solvers applied to the FEA problem (2.20). Legend: AMG (Algebraic multigrid), NN (Non-nested multigrid), PMG (Polynomial multigrid). . . . 58
2.7 Number of iterations and time to solution for algebraic multigrid (AMG), polynomial multigrid (PMG), non-nested Multigrid (NN) applied to the piston test case with different polynomial degrees from $p = 1$ to $p = 4$. AMG times are shown for $\mathcal{Q}^1$ and $\mathcal{Q}^2$ elements only. . . . 60
2.8 Total number of DoF for polynomial degree p and mesh-size for the piston test. . . . 61
2.9 Total number of DoF for polynomial degree p and mesh-size for the wrench test. . . 62
2.10 Number of iterations and time to solution for algebraic multigrid (AMG), polynomial multigrid (PMG), non-nested Multigrid (NN) applied to the wrench test case with different polynomial degrees from $p = 1$ to $p = 4$. AMG times are shown for $\mathcal{Q}^1$ and $\mathcal{Q}^2$ elements only. . . . 63
2.11 Number of iterations and Time to solution for Global Coarsening (GC) and Non-Nested Multigrid (NN) applied to the `cube` test case with polynomial degree $p = 4$. . 64
2.12 Number of iterations and Time to solution for Global Coarsening (GC) and Non-Nested Multigrid (NN) applied to the `ball` test case with polynomial degree $p = 4$. . 64
2.13 Multigrid statistics for the 2D L-shaped test case for different number of levels (wl: serial/parallel workload, wl-eff: parallel workload efficiency, v-eff: vertical communication efficiency, v-eff-METIS: vertical communication efficiency employing METIS.). . . . 68
2.14 Multigrid statistics for the 3D Fichera test case for different number of levels (wl: serial/parallel workload, wl-eff: parallel workload efficiency, v-eff: vertical communication efficiency, v-eff-METIS: vertical communication efficiency employing METIS.). . . . 69
2.15 Multigrid statistics for the piston test case for different number of levels (wl: serial/-parallel workload, wl-eff: parallel workload efficiency, v-eff: vertical communication efficiency, v-eff-METIS: vertical communication efficiency employing METIS.). . . 70

3.3 3D piston model. Total number of DoF in function of the polynomial degree p. . . . 98
3.4 Iteration counts for conjugate-gradient with multigrid preconditioning (CG+MG) on the grids $\Omega_1, \Omega_2, \Omega_3$. Multigrid levels are obtained through the R-tree procedure. The number of required iterations by plain conjugate-gradient (CG) is reported in the last column of each table. . . . 100
3.5 Iteration counts for conjugate-gradient with multigrid preconditioning (CG+MG) on the grids $\Omega_1, \Omega_2, \Omega_3$ with $l = 3$ levels and $m = 5$ smoothing steps. Multigrid levels are obtained through the R-tree procedure. The number of required iterations by plain conjugate-gradient (CG) is reported in the last column. . . . 101

3.6 Iteration counts for conjugate-gradient with multigrid preconditioning (CG+MG) on the 3D grids Ω_4, Ω_5 with polynomial degree $p = 1,2,3$ and $l = 3$ levels. Multigrid levels are obtained through the R-tree procedure. The number of required iterations by plain conjugate-gradient (CG) is reported in the last column of each table. 102

5.1 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^{3}$ on top of the original mesh $\Omega_0 \equiv \mathcal{T}_0$, partitioning the original mesh across **128** processors. The number of agglomerates per level is shown in second column, while the ratio between the cardinality of consecutive grids is reported in the last column. 146

5.2 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^{3}$ on top of the original mesh $\Omega_0 \equiv \mathcal{T}_0$, partitioning the original mesh across **256** processors. The number of agglomerates per level is shown in second column, while the ratio between the cardinality of consecutive grids is reported in the last column. 147

5.3 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^{3}$ on top of the original mesh $\Omega_0 \equiv \mathcal{T}_0$, partitioning the original mesh across **1024** processors. The number of agglomerates per level is shown in second column, while the ratio between the cardinality of consecutive grids is reported in the last column. 147

5.4 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^{4}$ on top of the mesh representing the left ventricle, partitioning the original mesh across **12** processors. 150

5.5 FitzHugh-Nagumo parameters used in the 2D numerical test. 153

5.6 Parameters for Bueno-Orovio model used in the 3D numerical test. In this case $\chi_m \equiv C_m = 1$ as we are using formulation (5.1). 153

Chapter 0

Introduction

Several mathematical and computational approaches have been presented during the years in order to tackle the solution of multi-physics problems on complex domains, such as in fluid-structure interaction problems. Often, the discretization of these systems of PDEs naturally leads to the use of multiple grids. We can classify these methods with respect to the approach used to handle the interaction of one domain (for instance, the solid domain) into the other domain (the fluid domain) or vice versa. In this context, methods are historically divided into two big categories: *non-boundary-fitted methods* and *unfitted methods*. The so-called Arbitrary Lagrangian Eulerian formulation (ALE) [115, 80, 119] is by far the most popular scheme of the first type, where different grids for independent problems are joined at the (shared) interface between domains. Although ideal in that kinematic constraints are satisfied by construction, it may easily become computationally inefficient during the evolution of the system in presence of moving interfaces. Indeed, the computational mesh may become severely distorted, requiring a costly remeshing procedure. Non-boundary-fitted methods were hence developed as an alternative approach to avoid such computational drawbacks. Non-boundary-fitted methods can embed the possibly complicated domain of interest in a geometrically simple background grid (usually a structured Cartesian mesh) providing a great amount of flexibility at the geometrical discretization step, often at cost of a reduction in accuracy near the interface between the two grids. Among these, we mention the X-FEM [36, 144], the immersed boundary method [152], and fictitious domain methods [99, 100].

The great freedom shared by this class of schemes comes at the price of handling non-matching grids which arbitrarily overlap at interfaces, where the meshes need to be coupled and exchange information [127, 139]. The interaction between different geometries is usually represented by a coupling matrix which describes how the physical fields defined on the two domains interact. In some cases (cf. [42]), the accurate computation of coupling terms is crucial in order not to destroy the convergence properties of the numerical scheme. In Chapter 1, we present several implementation details related to the exact integration of coupling terms, discussing different numerical integration strategies and comparing their influence on different unfitted methods. As a by-product, we developed dimension-independent routines that can be used to assemble interface terms and coupling matrices across heterogeneous dimensions in a robust way, which we have included in the finite element library DEAL.II [21].

In practice, the implementation of such schemes in a fully distributed setting in which different grids are independently partitioned among processors brings several challenges due to lack of predefined communication patterns between different geometries. After having identified such sources of complexity, we note that for practical applications it is crucial to provide efficient solvers and preconditioners. Multigrid methods are among the most competitive solvers for elliptic problems [97]. In general, the choice of a multigrid approach depends on the way the mesh is generated and on the underlying finite-element space. For globally refined meshes, geometric multigrid is a natural choice, as the levels of the resulting mesh hierarchy may be used as multigrid levels. In the context of high-order finite elements, levels can also be obtained by reducing the polynomial order of the shape functions of the elements, while keeping the mesh fixed, as done in polynomial multigrid [29, 148]. Hybrid multigrid solvers such as *hp*-multigrid, where both the mesh-size h and the polynomial degree p are allowed to vary among levels, are nowadays common and exploit the robustness of both approaches. It is common to assemble the system matrix related to the discretized system and to pass this matrix to an iterative solver. However, from a High-Performance-Computing perspective, the assembly step is not cheap and the matrix might become locally dense, e.g., in the context of high-order methods, leading to high costs of the matrix application due to limited bandwidth on modern CPU-based hardware. For these reasons, matrix-free algorithms are a means to accelerate FEM computations, by applying the effect of the system matrix without assembling. Matrix-free implementations of multigrid schemes have proven extremely efficient on modern computing systems [133, 145, 131], achieving significant gain against standard matrix-based approaches as the polynomial degree is greater than one.

Possibly, the biggest drawback of classical geometric multigrid schemes is that for *complex* geometries and moderately low order elements, it becomes impractical to create a nested hierarchy of grids. Indeed, one may be given with grids generated externally after using CAD (computer-assisted-design) solid modelers. Or, a fine and unstructured mesh may be given as is. In such instances, multigrid levels are not directly available and other multilevel strategies must be employed. The simplest and most used way to overcome this issue, especially for linear elements, is the Algebraic Multigrid method [167] which circumvents the explicit construction of a hierarchy by working only at the algebraic level. In this scenario, a *non-nested* multigrid method [47, 2, 38] can reduce the burden of hierarchy generation by allowing levels to be generated independently, requiring in turn ad-hoc definitions of suitable intergrid transfer operators. These operators realize the coupling between arbitrarily overlapped (and memory-distributed) levels. It becomes then apparent how the computational kernels identified in Chapter 1 are really close to the tasks that we have to face if we want to provide a reliable realization of such a multigrid technique. The realization of a parallel and matrix-free non-nested multigrid scheme is the subject of Chapter 2. We avoid to explicitly construct the sparse and large transfer matrices and rely on efficient algorithms for pointwise evaluation of finite element fields at arbitrarily located points [37], as well as on distributed geometric search algorithms. Our implementation has been thoroughly tested on a variety of configurations by changing polynomial degrees, geometries, and number of levels, confirming its practical validity. The associated software has been added in the DEAL.II library [21]. In this way it has been possible to integrate our implementation with other core components of the multigrid pipeline such as the pre- and post-smoothing phases, highly-optimized matrix-free kernels for operator evaluators developed within such library during the years [130].

An attractive approach to perform coarsening of very fine grids can be offered by *agglomeration* techniques, which consist in merging elements of the original mesh to obtain coarser grids. Indeed, through agglomeration we can obtain a coarser approximation of the underlying differential problem, resulting in a huge reduction in terms of number of unknowns. This strategy can be naturally performed in the context of numerical methods that allow general polygons and polyhedra as grid elements. Among numerical schemes of this type, we mention the Virtual Element Method (VEM) [16], the Polytopic Discontinuous Galerkin [15, 30, 59], and the Hybrid High-Order method (HHO) [75, 78]. In this thesis, we adopt a Polytopic Discontinuous Galerkin (PolyDG) framework. Early approaches on agglomerated multigrid can be found, although in different contexts, in works [141, 174] and in [63]. The process of element agglomeration is still an active area of research. Standard approaches rely on graph partitioners such as METIS [125], which take into account only the connectivity of the graph associated to the computational grid, rather than any geometrical information. Recently, Machine-Learning based strategies have been developed in [20, 19] to automatically perform agglomeration of meshes eventually composed of heterogeneous media. In this thesis, a novel approach to obtain a balanced sequence of *agglomerated levels* that can be employed in a multigrid framework is the subject of Chapter 3. The new strategy is based on efficient spatial data structures such as R-trees [105] that can be extracted from the geometrical and topological representation of the computational grid at hand. We show how the R-tree-based agglomeration delivers automatically a *nested* hierarchy of agglomerated levels which can be employed within a geometric multigrid method, which we named R3MG (R-tree MultiGrid) [87]. We also describe and validate our memory-distributed implementation with a series of benchmarks, confirming its robustness. This thus appears as an alternative approach to the conforming setting of Chapter 2, fully relying on polytopic shapes to obtain coarser levels out of a fine grid.

Leveraging on the versatility and the efficiency of R3MG, we apply our multilevel methodology to a real context such as cardiac electrophysiology. In particular, we consider the monodomain problem [146], where discontinuous Galerkin methods could be beneficial in order to resolve the physics of the system thanks to their simple applicability to complex geometries and higher-order discretizations [69, 90, 106]. More specifically, we consider a discontinuous Galerkin discretization of the monodomain problem as done in [15, 118]. We show how, starting from a realistic hexahedral mesh and its associated standard DG discretization, we can exploit polytopic shapes obtained through the agglomeration strategy outlined in Chapter 3 to build a parallel multigrid preconditioner. We confirm the practical validity and robustness of this approach through several experiments varying models, polynomial degrees, and number of levels employed. Finally, building on the analysis done in [9, 10], and on estimates and bound developed in the general setting [61], we show in Chapter 4 some preliminary results aimed at proving the convergence of our multilevel scheme by adapting such analysis to the properties of our agglomerates.

The flexibility allowed by polytopic methods makes them potentially extremely powerful computational tools. Their exploitation for the construction of efficient multigrid solvers, pioneered in [30, 10] and pursued here with the R3MG algorithm, constitutes one of the prominent examples of their potential. On the other hand, their success depends on the availability of efficient implementations tackling the computational overhead involved, for instance, in the design of appropriate basis, quadrature rules, and solvers. For instance, R-tree agglomeration is shown here to produce, in a scalable manner, agglomerates which are tightly close to their bounding boxes, thus yielding polygonal and polyhedral meshes for which the design of optimal basis is trivial. One of the main objectives of this thesis is to propose a number of such efficient techniques, and we trust that their impact goes well beyond the results presented here.

Thesis outline

This thesis is organized into two parts. The first part covers the presentation of some computational aspects related to non-matching methods and interface problems and the application of such procedures to develop a non-nested and matrix-free multigrid solver. The second part involves polytopic methods. A new and efficient agglomeration technique based on spatial indices is presented and applied within the context of polytopic discontinuous Galerkin methods. Building on this infrastructure, we develop a novel multigrid strategy that exploits the flexibility of polytopes to build coarser levels that can be employed in a multilevel solver, and investigate its effectiveness by applying it to non-trivial geometries. The outline of the thesis is summarized in Figure 1.

- Part I. **Non-matching methods**

- chapter 1 **Interface problems**. This chapter refers to the paper *A comparison of non-matching techniques for the finite element approximation of interface problems* [43] in collaboration with Prof. Daniele Boffi (KAUST), Prof. Lucia Gastaldi (University of Brescia), Prof. Luca Heltai (SISSA), and Prof. Andrea Cangiani (SISSA). Part of the work involved the development of generic computational geometry routines which have been added to the DEAL.II library and such contributions are part of publication [22].

- chapter 2 **Matrix-free implementation of the non-nested multigrid method**. This chapter refers to a work done in collaboration with Prof. Martin Kronbichler (University of Augsburg, Ruhr-Universität Bochum), Dr. Peter Münch (University of Augsburg, Uppsala University), and Prof. Luca Heltai (SISSA). This implementation has been added to the DEAL.II library and is part of publication [23].

- Part II. **Polytopic finite elements**

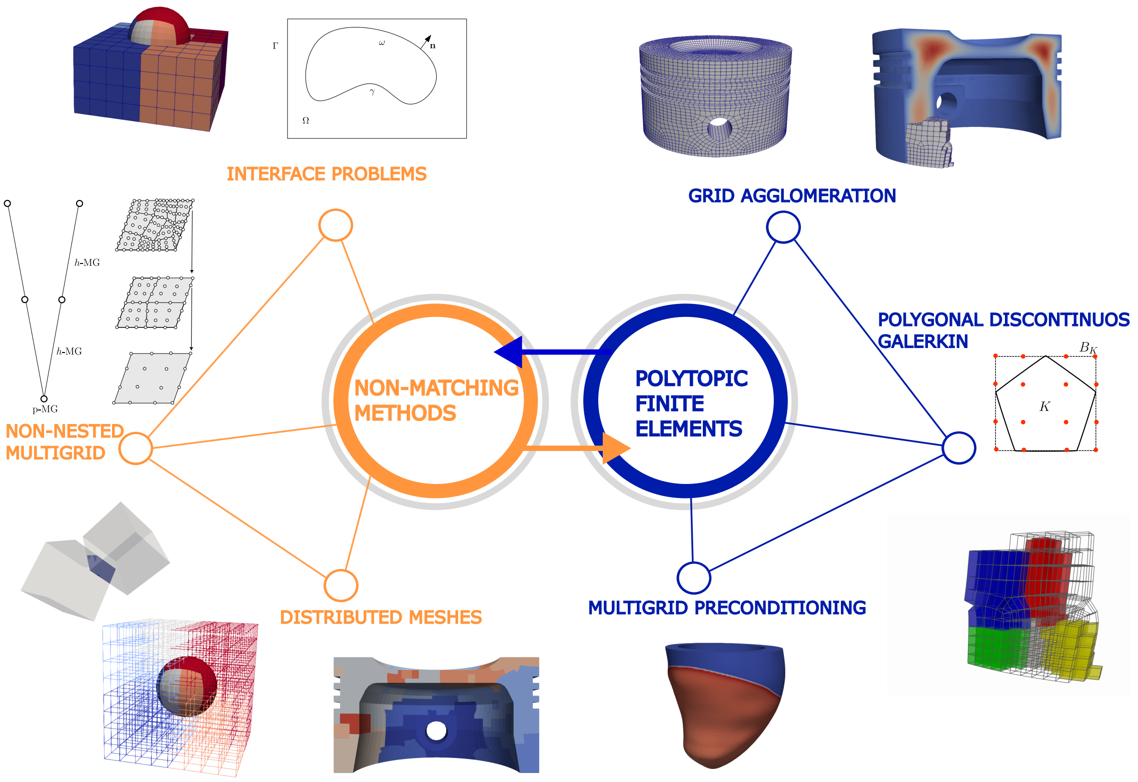


Fig. 1 Thesis structure: the first part focuses on computational aspects of non-matching methodologies, the second part on agglomeration techniques, which are applied to polytopic discontinuous Galerkin methods and multigrid preconditioning.

– chapter 3 **Agglomeration of grids**. This chapter refers to the preprint *R-tree based agglomeration of polytopic grids with applications to multilevel methods* [87] in collaboration with Prof. Andrea Cangiani (SISSA) and Prof. Luca Heltai (SISSA, University of Pisa).

– chapter 4 **Convergence Analysis for multigrid**. This chapter refers to preliminary results about the convergence analysis of the multigrid method based on the agglomeration strategy developed in Chapter 3.

– chapter 5 **Application of multilevel solver to cardiac electrophysiology**. This chapter refers to the application and the investigation of a multilevel methodology, based on the results in Chapter 3, to cardiac electrophysiology. This part is based on an ongoing collaboration with Dr. Pasquale Claudio Africa (SISSA).

Part I

Non-matching techniques

Chapter 1

Non-matching methods

We perform a systematic comparison of various numerical schemes for the approximation of interface problems. We consider unfitted approaches in view of their application to possibly moving configurations. Particular attention is paid to the implementation aspects and to the analysis of the computational costs related to the different phases of the simulations.

Contents

1.1 Literature review **9**

1.2 Model problem and notation **11**

1.3 Non-matching discretizations **13**

1.3.1 The method of Lagrange Multipliers 14

1.3.2 The cut-FEM method 17

1.4 Integration of coupling terms **19**

1.4.1 Integration driven by the immersed mesh 20

1.4.2 Integration on mesh intersections 21

1.4.3 Integration through level set splitting 22

1.5 Numerical experiments **25**

1.5.1 2D numerical tests 26

1.5.2 3D numerical tests 33

1.5.3 Computational times 37

1.6 Conclusions **38**

1.1 Literature review

The efficient numerical solution of partial differential equations modeling the interaction of physical phenomena across interfaces with complex, possibly moving, shapes is of great importance in many scientific fields. We refer, for instance, to fluid-structure interaction, or crack propagation, just to mention two relevant examples. A crucial issue is the handling of computational grids. In this respect, we can classify computational methods for interface problems into two families: boundary fitted methods and boundary unfitted methods. For time dependent problems, the former are typically handled using the Arbitrary Eulerian Lagrangian formulation ([80], [116]), where meshes are deformed in a conforming way with respect to movements of the physical domains. In this case, the imposition of interface conditions is usually easy to implement. However, an accurate description of both meshes is required, and the allowed movements are restricted by the topological structure of the initial state. When topology may change, or when the grid undorgoes severe deformations, these methods require remeshing. An operation which is computationally very expensive in time dependent scenarios and three dimensional settings. Conversely, unfitted approaches are based on describing the physical domains as embedded into a constant background mesh. As it does not require remeshing, this approach is extremely flexible, but it requires sophisticated methods to represent interfaces. Among unfitted approaches, we mention the Immersed Boundary Method [152, 40], the Cut Finite Element Method [53], and the Extended Finite Element Method [143]. Another popular choice is the Fictitious Domain Method with Lagrange multipliers, proposed by Glowinski, Pan and Periaux in [100] for a Dirichlet problem, analyzed in [98], and then extended to particulate flows in [99].

The use of a Lagrange multiplier to deal with Dirichlet boundary conditions was introduced in the seminal work by Babuška [25]. This is formulated in terms of a symmetric saddle point problem where the condition at the interface is enforced through a Lagrange multiplier. The main drawback of this method is that it suffers from a loss of accuracy at interfaces, even if it is known (cf. [111]) that this detrimental effect on the convergence properties of the approximate solution is a local phenomenon, restricted to a small neighbourhood of the interface. From the computational standpoint, the Fictitious Domain Method poses the additional challenge of assembling coupling terms involving basis functions living on different meshes. In this context, one can distribute quadrature points on the immersed mesh and let it drive the integration process, or one may compute a composite quadrature rule by identifying exact intersections (polytopes, in general) between the two meshes. This approach has been presented in different papers and frameworks: for instance, in [128], a high performance library has been developed to perform such tasks, and in [42] it has been shown how composite rules on interfaces turn out to be necessary to recover optimal rates for a fluid-structure interaction problem where both the solid and the fluid meshes are two dimensional objects. In the cut-FEM framework, we mention [140] for an efficient implementation of Nitsche's method on three dimensional overlapping meshes.

We consider a two or three dimensional domain with an immersed interface of co-dimension one and we study the numerical approximation of the solution of an elliptic PDE whose solution is prescribed along the interface. Although in our case the domain and the interface are fixed, we are discussing the presented numerical schemes in view of their application to more general settings. We perform a systematic comparison between three different unfitted approaches, analyzing them in terms of accuracy, computational cost, and implementation effort. In particular, we perform a comparative analysis in terms of accuracy and CPU times for the Lagrange multiplier method, Nitsche's penalization method, and cut-FEM for different test cases, and discuss the benefit of computing accurate quadrature rules on mesh intersections.

After introducing the model Poisson problem posed on a domain with an internal boundary, we review the Lagrange multiplier method, the Nitsche penalization method, and the cut-FEM method for its solution. We discuss the central issue of the numerical integration of the coupling terms, and present a numerical comparison of the three methods, focusing on the validation of the implementation, on how coupling terms affect the accuracy of numerical solutions, and on computational times.

1.2 Model problem and notation

Let ω be a closed and bounded domain of $\mathbb{R}^d$, $d = 2,3$, with Lipschitz continuous boundary $\gamma := \partial\omega$, and $\Omega \subset \mathbb{R}^d$ a Lipschitz domain such that $\omega \Subset \Omega$; see Figure 1.1 for a prototypical configuration. We consider the model problem

$$\begin{cases} -\Delta u &= f \qquad \text{in } \Omega \setminus \gamma, \\ u &= g \qquad \text{on } \gamma, \\ u &= 0 \qquad \text{on } \Gamma := \partial\Omega, \end{cases} \tag{1.1}$$

for given data $f \in L^2(\Omega)$ and $g \in H^{\frac{1}{2}}(\gamma)$. Throughout this chapter we refer to Ω as the *background* domain, while we refer to ω as the *immersed* domain, and γ as the *immersed boundary*. The rationale behind this setting is that it allows to solve problems in a complex and possibly time dependent domain ω, by embedding the problem in a simpler background domain Ω – typically a box – and imposing some constraints on the immersed boundary γ. For the sake of simplicity, we consider the case in which the immersed domain is *entirely* contained in the background domain, but more general configurations may be considered.

As ambient spaces for (1.1), we consider $V(\Omega) := H^1_0(\Omega) = \{v \in H^1(\Omega) : v_{|\Gamma} = 0\}$ and $Q(\gamma) := H^{-\frac{1}{2}}(\gamma)$. Given a domain $D \subset \mathbb{R}^d$ and a real number $s \geq 0$, we denote by $\|\cdot\|_{s,D}$ the standard norm of $H^s(D)$. In particular, $\|\cdot\|_{0,D}$ stands for the L^2-norm stemming from the standard L^2-inner product $(\cdot,\cdot)_D$ on D. Finally, with $\langle\cdot,\cdot\rangle_\gamma$ we denote the standard duality pairing between $Q(\gamma)$ and its dual $Q'(\gamma) = H^{\frac{1}{2}}(\gamma)$. Across the immersed boundary γ, we define the *jump* operator as

$$\begin{aligned} [\![v]\!]_{|\gamma} &= v^+ - v^-, \\ [\![\boldsymbol{\tau}]\!]_{|\gamma} &= \boldsymbol{\tau}^+ - \boldsymbol{\tau}^-, \end{aligned}$$

for smooth enough scalar- and vector- valued functions v and $\boldsymbol{\tau}$. Here, $v^\pm$ and $\boldsymbol{\tau}^\pm$ are external and internal traces defined according to the direction of the outward normal $\boldsymbol{n}$ to ω at γ.

Problem (1.1) can be written as a constrained minimization problem by introducing the Lagrangian $\mathcal{L} : V(\Omega) \times Q(\gamma) \to \mathbb{R}$ defined as

$$\mathcal{L}(v,q) := \frac{1}{2}(\nabla v, \nabla v)_\Omega - (f,v)_\Omega + \langle q, v-g\rangle_\gamma. \tag{1.2}$$

Looking for stationary points of $\mathcal{L}$ gives the following saddle point problem of finding a pair $(u,\lambda) \in V(\Omega) \times Q(\gamma)$ such that

$$\begin{aligned}
(\nabla u, \nabla v)_\Omega + \langle \lambda, v\rangle_\gamma &= (f,v)_\Omega \qquad \forall v \in V(\Omega), && (1.3)\\
\langle q, u\rangle_\gamma &= \langle q, g\rangle_\gamma \qquad \forall q \in Q(\gamma), && (1.4)
\end{aligned}$$

Below in Theorem 1.2.1 we show that this problem admits a unique solution. Starting from (1.1) and integrating by parts, one can easily show that setting $\lambda = -[\![\nabla u \cdot \boldsymbol{n}]\!]_{|\gamma}$, the pair $(u,\lambda) \in V(\Omega) \times Q(\gamma)$ is the solution of (1.3)-(1.4). Conversely, with proper choices for $v \in V(\Omega)$ in (1.3) one gets that $-\Delta u = f$ in $\Omega \setminus \gamma$, while (1.4) implies $u = g$ on γ, so that (1.3)-(1.4) are equivalent to (1.1) with $-\lambda$ equal to the jump of the normal derivative of u on the interface γ. Moreover, if the datum g is sufficiently smooth, say $g \in H^s(\gamma)$ for $s > 1$, we can further take $\lambda \in L^2(\gamma)$ and use in practice $Q(\gamma) = L^2(\gamma)$.

In the following theorem we sketch the proof of existence and uniqueness of the solution of (1.3)-(1.4).

Theorem 1.2.1. *Given $f \in L^2(\Omega)$ and $g \in H^{1/2}(\gamma)$, there exists a unique solution of Problem* (1.3)-(1.4) *satisfying the following stability estimate*

$$\|u\|_{1,\Omega} + \|\lambda\|_{-1/2,\gamma} \leq C(\|f\|_{0,\Omega} + \|g\|_{1/2,\gamma}).$$

Proof. Problem (1.3)-(1.4) is a saddle point problem, hence to show existence and uniqueness of the solution we need to check the *ellipticity on the kernel* and the *inf-sup condition* [41]. The kernel $\mathbb{K} = \{v \in V(\Omega) : \langle q, v\rangle = 0 \; \forall q \in Q(\gamma)\}$, can be identified with the subset of functions in $V(\Omega)$ with vanishing trace along γ. Thanks to Poincaré inequality we have that there exists $\alpha_0 > 0$ such that

$$(\nabla u, \nabla u)_\Omega \geq \alpha_0 \|u\|^2_{1,\Omega}.$$

The inf-sup condition can be verified using the definition of the norm in $Q(\gamma)$ and the fact that, by the trace theorem, for each $w \in H^{1/2}(\gamma)$ there exists at least an element $v \in V(\Omega)$ such that $v = w$ on γ, with $\|v\|_{1,\Omega} \leq C_1 \|w\|_{1/2,\gamma}$. Hence we get the inf-sup condition

$$\inf_{q \in Q(\gamma)} \sup_{v \in V(\Omega)} \frac{\langle q, v\rangle}{\|q\|_{-1/2,\gamma} \|v\|_{1,\Omega}} \geq \beta_0,$$

with $\beta_0 = 1/C_1$. $\square$

A detailed analysis for the Dirichlet problem, where the Lagrange multiplier is used to impose the boundary condition, can be found in the pioneering work by Glowinski, Pan and Periaux [100].

1.3 Non-matching discretizations

We assume that both Ω and ω are Lipschitz domains and we discretize the problem introducing computational meshes for the domain Ω and for the immersed boundary γ which are *unfitted* with respect to each other in that they are constructed independently. The computational meshes Ω_h of Ω and γ_h of γ consist of disjoint elements such that $\Omega = \bigcup_{T\in\Omega_h} T$ and $\gamma = \bigcup_{K\in\gamma_h} K$. When $d = 2$, Ω_h will be a triangular or quadrilateral mesh and γ_h a mesh composed by straight line segments. For $d = 3$, Ω_h will be a tetrahedral or hexahedral mesh and γ_h a surface mesh whose elements are triangles or planar quadrilaterals embedded in the three dimensional space. We denote by h_Ω and h_γ the mesh sizes of Ω_h and γ_h, respectively. For simplicity, we ignore geometrical errors in the discretizations of Ω, and γ, and we assume that the mesh sizes h_Ω and h_γ are small enough so that the geometrical error is negligible with respect to the discretization error (see Sect. 1.4.3 for a quantitative estimate of the geometrical error in our numerical experiments).

We consider discretizations of $V(\Omega)$ based on standard Lagrange finite elements, namely

$$V_h(\Omega) := \{v \in H^1_0(\Omega) : v|_T \in \mathcal{R}^p(T), \forall T \in \Omega_h\} \qquad p \geq 1, \tag{1.5}$$

with $\mathcal{R}^p(T) := \mathcal{P}^p(T)$ or $\mathcal{Q}^p(T)$, the spaces of polynomials of total degree up to p or of degree p separately in each variable, respectively, depending on whether T is a simplex or a quadrilateral/hexahedron. For the cut-FEM method, the corresponding spaces will be constructed separately in each subdomain; this results in a doubling of the degrees of freedom on the elements cut by the interface (see Sect. 1.3.2).

For the Lagrange multiplier space $Q(\gamma)$ we employ the space of piecewise-polynomial functions

$$Q_h(\gamma) := \{q \in L^2(\gamma) : q|_K \in \mathcal{R}^p(K), \forall K \in \gamma_h\} \qquad p \geq 0, \tag{1.6}$$

which we equip with the mesh dependent norm

$$\|q\|_{-\frac{1}{2},\gamma} := \|h^{-\frac{1}{2}} q\|_{0,\gamma} \qquad \forall q \in Q_h(\gamma), \tag{1.7}$$

where h is the piecewise constant function given by $h|_K = h_K$ the diameter of K for each $K \in \gamma_h$. The definition and the notation are justified by the fact that, on a quasi-uniform meshes, norm (1.7) is equivalent to the norm in $H^{-1/2}(\gamma)$.

1.3.1 The method of Lagrange Multipliers

The discrete counterpart of (1.3)-(1.4) is to find a pair $(u_h,\lambda_h)\in V_h\times Q_h$ such that

$$\begin{aligned}(\nabla u_h,\nabla v_h)_\Omega+\langle\lambda_h,v_h\rangle_\gamma &= (f,v_h)_\Omega \qquad \forall v_h\in V_h, && (1.8)\\ \langle q_h,u_h\rangle_\gamma &= \langle q_h,g\rangle_\gamma \qquad \forall q_h\in Q_h. && (1.9)\end{aligned}$$

The next theorem states existence, uniqueness, and stability of the discrete solution together with optimal error estimates.

Theorem 1.3.1. *Assume that the mesh γ_h is quasi uniform and that there exists a constant $\mathrm{C}_r>1$ independent of h_Ω and h_γ such that $h_\Omega/h_\gamma\le\mathrm{C}_r$. Then, there exists a unique solution $(u_h,\lambda_h)\in V_h\times Q_h$ of Problem (1.8)-(1.9). Moreover, it holds*

$$\|u-u_h\|_{1,\Omega}+\|\lambda-\lambda_h\|_{-1/2,\gamma}\le \mathrm{C}\inf_{\substack{v_h\in V_h\\ \mu_h\in Q_h}}\left(\|u-v_h\|_{1,\Omega}+\|\lambda-\mu_h\|_{-1/2,\gamma}\right), \qquad (1.10)$$

with $\mathrm{C}>0$ a constant independent of the mesh sizes h_Ω and h_γ.

Proof. The existence, uniqueness, and stability of the discrete solution can be obtained by showing that there exist positive constants α and β, independent of h_Ω and h_γ, such that the ellipticity on the kernel and inf-sup condition hold true at the discrete level [41]. Since $Q_h\subset L^2(\gamma)$, we have that the discrete kernel $\mathcal{K}_h=\{v_h\in V_h:\langle q_h,v_h\rangle=0\forall q_h\in Q_h\}$ contains element with $\int_\gamma v_h ds=0$. Hence, Poincaré inequality (see [48, (5.3.3)]), which can be formulated as

$$\|v_h\|_{1,\Omega}\le \mathrm{C}_\Omega\left(\left|\int_\gamma v_h ds\right|+\|\nabla v_h\|_{0,\Omega}\right)=\mathrm{C}_\Omega\|\nabla v_h\|_{0,\Omega},$$

implies that

$$(\nabla v_h,\nabla v_h)_\Omega\ge\alpha\|v_h\|^2_{1,\Omega}\quad \forall v_h\in\mathcal{K}_h.$$

The discrete inf-sup condition

$$\inf_{q_h\in Q_h}\sup_{v_h\in V_h}\frac{\langle q_h,v_h\rangle}{\|q_h\|_{-1/2,\gamma}\|v_h\|_{1,\Omega}}\ge\beta,$$

is more involved and makes use of the continuous inf-sup, together with Clément interpolation, trace theorem, and inverse inequality. The interested reader can find the main arguments of this proof in [39, sect. 5].

Thanks to the above conditions, the theory on the approximation of saddle point problems gives both existence and uniqueness of the solution of Problem (1.8)-(1.9) satisfying the a priori estimate

$$\|u_h\|_{1,\Omega}+\|\lambda_h\|_{-1/2,\gamma}\le \mathrm{C}(\|f\|_{0,\Omega}+\|g\|_{1/2,\gamma}),$$

and the error estimate (1.10). □

Given basis functions $\{v_i\}_{i=1}^N$ and $\{q_i\}_{i=1}^M$ such that $V_h := \text{span}\{v_i\}_{i=1}^N$ and $Q_h := \text{span}\{q_i\}_{i=1}^M$, we have that (1.8), (1.9) can be written as the following algebraic problem

$$\begin{pmatrix} A & C^\top \\ C & 0 \end{pmatrix} \begin{pmatrix} U \\ \lambda \end{pmatrix} = \begin{pmatrix} F \\ G \end{pmatrix} \tag{1.11}$$

where

$$\begin{aligned} A_{ij} &= (\nabla v_j, \nabla v_i)_\Omega \qquad i,j = 1,\dots,N \\ C_{\alpha j} &= \langle q_\alpha, v_j \rangle_\gamma \qquad j = 1,\dots,N, \alpha = 1,\dots,M \\ F_i &= (f, v_i)_\Omega \qquad i = 1,\dots,N \\ G_\alpha &= \langle q_\alpha, g \rangle_\gamma \qquad \alpha = 1,\dots,M. \end{aligned}$$

To solve the block linear system (1.11) we use Krylov subspace iterative methods applied to the Schur complement system:

$$\lambda = S^{-1}(CA^{-1}F - G), \tag{1.12}$$

$$U = A^{-1}(F - C^\top \lambda), \tag{1.13}$$

where $S := CA^{-1}C^\top$, and we use $CAC^\top + M$ as preconditioner for S, where M is the immersed boundary mass matrix with entries $(M)_{ij} = \langle q_j, q_i \rangle_\gamma$.

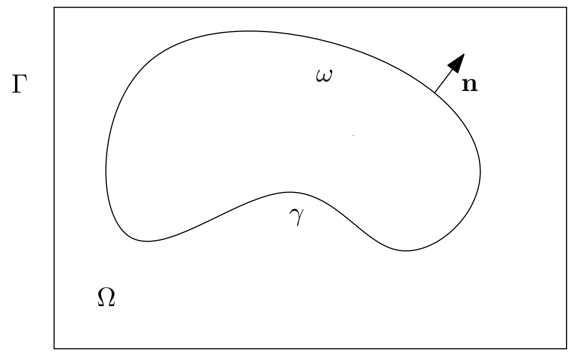


Fig. 1.1 Model problem setting, with immersed domain ω, immersed boundary γ, and background domain Ω.

We next show how the Lagrange multiplier formulation is directly linked to a penalization approach used to impose the Dirichlet condition $u = g$ on the internal curve γ, by locally eliminating the multiplier in the same spirit of the work by Stenberg [165].

Instead of enforcing the constraint on γ with a multiplier, it is possible to impose it weakly through a penalization approach following the so-called method of Nitsche. Here we show that the enforcement of boundary conditions via Nitsche's method can be derived from a *stabilized* Lagrange multiplier method by adding a consistent term that penalizes the distance between the discrete multiplier λ_h and the normal derivative [165]. With this in mind, we penalize the *jump* of the normal derivative along the internal curve γ to impose the constraint $u = g$. The consistency here follows from the observation that at the continuous level the multiplier is the jump of the normal derivative on the interface.

We define $h(\boldsymbol{x})$ as the piecewise constant function describing the mesh-size of γ and we choose the discrete spaces V_h and Q_h as in the Lagrange multiplier case.

The addition of the normal gradient penalization term to the Lagrange multiplier formulation (1.8)-(1.9) leads to the problem of seeking a pair $(u_h, \lambda_h) \in V_h \times Q_h$ such that

$$\begin{aligned}(\nabla u_h, \nabla v_h)_\Omega + \langle \lambda_h, v_h\rangle_\gamma + \frac{1}{\beta}\langle [\![\nabla v_h \cdot \boldsymbol{n}]\!], h(\lambda_h + [\![\nabla u_h \cdot \boldsymbol{n}]\!])\rangle_\gamma &= (f, v_h)_\Omega \qquad \forall v_h \in V_h,\\ \langle q_h, u_h\rangle_\gamma - \frac{1}{\beta}\langle q_h, h(\lambda_h + [\![\nabla u_h \cdot \boldsymbol{n}]\!])\rangle_\gamma &= \langle q_h, g\rangle_\gamma \qquad \forall q_h \in Q_h,\end{aligned}$$

where β is a positive penalty parameter. In the discrete setting, $\langle q, v\rangle_\gamma$ is identified with the scalar product in $L^2(\gamma)$ for $q \in Q_h$ and $v \in V_h$ and we use the notation $\langle q, v\rangle_\gamma = \sum_{K\in\gamma_h}(q, v)_K$. The second equation gives

$$\langle q_h, u_h - \frac{1}{\beta}h(\lambda_h + [\![\nabla u_h \cdot \boldsymbol{n}]\!]) - g\rangle_\gamma = 0 \qquad \forall q_h \in Q_h,$$

and, introducing the L^2-projection on Q_h as $\Pi_h : L^2(\gamma) \to Q_h$, we can eliminate the multiplier locally on each element $K \in \gamma_h$:

$$\lambda_h|_K = -\left(\Pi_h [\![\nabla u_h \cdot \boldsymbol{n}]\!]\right)_K + \beta h_K^{-1}\left(\Pi_h(u_h - g)\right)_K.$$

We observe that it is possible to formally refine γ_h to γ_h' such that the element boundaries of Ω_h coincide with some element boundaries of the immersed grid γ_h'. If we now choose a space Q_h' which contains piece wise polynomials of degree compatible with that of the elements in V_h, we can avoid the projection operator altogether, and are allowed to write

$$\lambda_h' = -[\![\nabla u_h \cdot \boldsymbol{n}]\!] + \beta h^{-1}(u_h - g).$$

Inserting this back into the first equation we get the variational problem (in which no multiplier is involved) of finding $u_h \in V_h$ such that

$$\begin{aligned}(\nabla u_h, \nabla v_h)_\Omega - \langle [\![\nabla u_h \cdot \boldsymbol{n}]\!], v_h\rangle_\gamma - \langle [\![\nabla v_h \cdot \boldsymbol{n}]\!], u_h\rangle_\gamma + \beta\langle h^{-1}u_h, v_h\rangle_\gamma \\ = (f, v_h)_\Omega - \langle [\![\nabla v_h \cdot \boldsymbol{n}]\!], g\rangle_\gamma + \beta\langle h^{-1}g, v_h\rangle_\gamma,\end{aligned} \tag{1.14}$$

holds for every $v_h \in V_h$.

Equation (1.14) represents the Nitsche method [147] applied to Problem (1.1). Owing to the non-matching nature of our discretization, the immersed and background mesh facets are not expected to be aligned in general. If indeed for all facets (elements) K of γ and facets F of Ω_h we have

$\mathcal{H}^{d-1}(K \cap F) = 0$, with $\mathcal{H}^{d-1}$ denoting the Hausdorff measure in $d-1$-dimension, then the variational problem can be simplified since all the jump terms vanish.

1.3.2 The cut-FEM method

When using the cut-FEM discretization approach, one changes the perspective of the original variational problem, which is no longer solved on a single space defined globally on Ω: instead, one solves two separate problems on the two domains ω and $\Omega \setminus \omega$. This approach gives additional flexibility on the type of problems that can be solved. For example, problems with more general transmission conditions across γ where the solution u is allowed to jump. Moreover, separating the problem transforms the cut-FEM method into a boundary-fitted approach, where the approximation space is changed to resolve the interface.

The usual approach to impose constraints on γ in the cut-FEM method is to use Nitsche's method applied on the two subdomains separately:

$$\begin{cases} \Omega^1 := \omega, \\ \Omega^2 := \Omega \setminus \bar{\omega}. \end{cases}$$

In this context, it is necessary to take special care of those elements of Ω_h that are cut by γ. First, we introduce the corresponding computational meshes Ω_h^i given by

$$\Omega_h^i := \{T \in \Omega_h : T \cap \Omega_i \neq \emptyset\} \qquad i = 1,2,$$

and notice that both meshes share the set of cells intersected by the curve γ, namely

$$\tau := \{T \in \Omega_h : T \cap \gamma \neq \emptyset\}, \qquad \tau^i := \{\tilde{T}^i := T \cap \Omega^i,\ T \in \tau\},$$

where we distinguish between entire cells that intersect γ (these are in the set τ) and cut cells (with arbitrary polytopic shape, which are in the set τ^i), i.e., $(T \in \tau) = (\tilde{T}^1 \in \tau^1) \cup (\tilde{T}^2 \in \tau^2) \cup (\gamma \cap T)$ with $\tilde{T}^1 \cap \tilde{T}^2 = \emptyset$.

When defining a finite element space on these elements, one uses the same definition of the original finite element space defined on entire elements $T \in \tau$, which is then duplicated and restricted to the corresponding domain, i.e., one introduces on Ω_h^i, $i = 1,2$ the discrete spaces

$$V_h^i := V_h(\Omega_h^i)|_{\Omega^i} = \{v_h|_{\Omega^i},\ v_h \in V_h(\Omega_h^i)\}.$$

Then, applying twice Nitsche's method requires to find $u_h^i \in V_h^i$ such that

$$a_h^i(u_h^i, v_h^i) = l_h^i(v_h^i) \qquad \forall v_h^i \in V_h^i \qquad i = 1,2,$$

with

$$a_h^i(u_h^i, v_h^i) = (\nabla u_h^i, \nabla v_h^i)_{\Omega_h^i \cap \Omega^i} - \langle [\![\nabla u_h^i \cdot \boldsymbol{n}]\!], v_h^i \rangle_\gamma - \langle u_h^i, [\![\nabla v_h^i \cdot \boldsymbol{n}]\!] \rangle_\gamma + \frac{\beta_1}{h} \langle u_h^i, v_h^i \rangle_\gamma, \tag{1.15}$$

and

$$l_h^i(v_h^i) = (f, v_h^i)_{\Omega_h^i \cap \Omega^i} + \left\langle g, \frac{\beta_1}{h} v_h^i - [\![\nabla v_h^i \cdot \boldsymbol{n}]\!] \right\rangle_\gamma . \tag{1.16}$$

This formulation is known to suffer from the so called *small-cut* problem deriving from the fact that the size of the cuts $T \cap \Omega^i$ cannot be controlled and hence can be arbitrarily small. This may result in a loss of coercivity for the bilinear forms a_h^i when the size of a cut cell goes to zero. As shown in [53], the formulation can be stabilized by adding to the bilinear form the following penalty term acting on the interior or exterior faces of the intersected cells, depending on the domain Ω_h^i:

$$\mathcal{G}_h^i := \{F = \overline{T}_+ \cap \overline{T}_- : T_+ \in \tau, T_- \in \Omega_h^i\} \qquad i = 1, 2, \tag{1.17}$$

$$j_h^i(u, v) := \beta_2 \sum_{F \in \mathcal{G}_h^i} \langle h_F [\![\nabla u \cdot \boldsymbol{n}]\!], [\![\nabla v \cdot \boldsymbol{n}]\!] \rangle_F, \tag{1.18}$$

where β_2 is a positive penalty parameter and h_F the size of $F \in \mathcal{G}_h^i$. With such definition at hand we set

$$V_h := V_h^1 + V_h^2, \text{ with elements } v_h = \begin{cases} v_h^1 & \text{in } \Omega^1 \\ v_h^2 & \text{in } \Omega^2 \end{cases} \tag{1.19}$$

$$a_h(u_h, v_h) := \sum_{i=1}^{2} (a_h^i(u_h^i, v_h^i) + j_h^i(u_h^i, v_h^i)), \tag{1.20}$$

$$l_h(v_h) := \sum_{i=1}^{2} l_h^i(v_h^i), \tag{1.21}$$

and the method reads: find $u_h \in V_h$ such that

$$a_h(u_h, v_h) = l_h(v_h) \qquad \forall v_h \in V_h.$$

The penalization term in (1.18), usually called *ghost-penalty*, is not the only choice that allows to recover optimal rates. Another practical alternative, developed earlier in [108], consists in associating "bad elements", i.e. elements with small cuts, with suitable neighboring elements having sufficiently large intersections with the domain in order to extend the polynomial approximation. This technique is usually referred to as *agglomeration* and it has been applied to several unfitted methods in recent years [122, 54].

1.4 Integration of coupling terms

In all three methods, some terms need to be integrated over the non-matching interface γ. For example, in the Lagrange multiplier method, we need to compute $\langle \lambda_h, v_h \rangle_\gamma$ where $v_h \in V_h$ and $\lambda_h \in Q_h$, while in the Nitsche's interface penalization method and in the cut-FEM method, one needs to integrate terms of the kind $\langle \beta h^{-1} u_h, v_h \rangle_\gamma$, where both u_h and v_h belong to V_h, but the integral is taken over γ.

We start by focusing our attention on the term $\langle \lambda_h, v_h \rangle_\gamma$ where $v_h \in V_h$ and $\lambda_h \in Q_h$. This is delicate to assemble as it is the product of trial and test functions living on *different* meshes and it encodes the interaction between the two grids. Let $K \in \gamma_h$ and F_K be the map $F_K : \hat{K} \to K$ from the reference immersed cell $\hat{K}$ to the physical cell K, $JF_K(\boldsymbol{x})$ the determinant of its Jacobian and assume to have a quadrature rule $\{\hat{\boldsymbol{x}}_q, w_q\}_{q=1}^{N_q}$ on $\hat{K}$. Since the discrete functions are piecewise polynomials, we can use the scalar product in $L^2(\gamma)$ instead of the duality pairing, then standard finite element assembly reads:

$$\langle \lambda_h, v_h \rangle_\gamma = \sum_{K \in \gamma_h} \langle \lambda_h, v_h \rangle_K. \tag{1.22}$$

In the forthcoming subsections we discuss three strategies to compute this integral. Identical considerations apply in order to compute the term $\beta \langle h^{-1} u_h, v_h \rangle_\gamma$ in (1.14).

In general, such integrals are always computed using quadrature formulas. What changes is the algorithm that is used to compute these formulas, and the resulting accuracy. Independently on the strategy that is used to compute the quadrature formulas, all algorithms require the efficient identification of pairs of potentially overlapping cells, or to identify background cells where quadrature points may fall. This task can be performed efficiently by using queries to R-trees of axis-aligned bounding boxes of cells for both the background and immersed mesh.

An R-tree is a data structure commonly used for spatial indexing of multi-dimensional data that relies on organizing objects (e.g., points, lines, polygons, or bounding boxes) in a hierarchical manner based on their spatial extents, such that objects that are close to each other in space are likely to be located near each other in the tree.

In an R-tree, each node corresponds to a rectangular region that encloses a group of objects, and the root node encloses all the objects. Each non-leaf node in the tree has a fixed number of child nodes, and each leaf node contains a fixed number of objects. R-trees support efficient spatial queries such as range queries, nearest neighbor queries, and spatial joins by quickly pruning parts of the tree that do not satisfy the query constraints.

In particular, we build two R-tree data structures to hold the bounding boxes of every cell of both the background and the immersed meshes. Spatial queries are performed traversing the R-tree structure generated by the `Boost.Geometry` library [44]. The construction of an R-tree with M objects has a computational cost that is proportional to $O(M \log(M))$, while the cost of a single query is $O(\log(M))$.

1.4.1 Integration driven by the immersed mesh

Applying straightforwardly a given quadrature rule defined over γ to equation (1.22) gives:

$$\langle \lambda_h, v_h \rangle_\gamma \approx \sum_{K \in \gamma_h} \sum_{q=0}^{N_q} \lambda_h\big(F_K(\hat{\boldsymbol{x}}_q)\big) v_h\big(F_K(\hat{\boldsymbol{x}}_q)\big) JF_K(\hat{\boldsymbol{x}}_q) w_q, \tag{1.23}$$

with respect to some reference quadrature points $\hat{\boldsymbol{x}}_q$ and weights w_q, letting the immersed domain drive the integration. In this case the computational complexity stems from the evaluation of the terms $v_h(F_K(\hat{\boldsymbol{x}}_q))$, since the position within the background mesh Ω_h of the quadrature point $F_K(\hat{\boldsymbol{x}}_q)$ is not known *a-priori* (see Figure 1.2). A possible algorithm for the evaluation of $v_h(F_K(\hat{\boldsymbol{x}}_q))$ can be summarized as follows:

- Compute the physical point $\boldsymbol{y} = F_K(\hat{\boldsymbol{x}}_q)$;
- Find the cell $T \in \Omega_h$ s.t. $\boldsymbol{y} \in T$;
- Given the shape function $\hat{v}_h(\hat{\boldsymbol{x}})$ in the reference element $\hat{T}$, compute $\hat{v_h}\big(G_T^{-1}(\boldsymbol{y})\big)$, where $G_T : \hat{T} \to T$ denotes the reference map associated to $T \in \Omega_h$ for the background domain.

The first part of the computation takes linear time in the number of cells of the immersed mesh γ_h, while the second part requires a computational cost that scales logarithmically with the number of cells of the background grid *for each quadrature point*. Finally, the evaluation of the inverse mapping G_T^{-1} requires Newton-like methods for general unstructured meshes or higher order mappings (i.e., whenever F_K is non-affine). Overall, implementations based on R-tree traversal will result in a computational cost that scales at least as $O((M+N)\log(N))$, where M is the total number of quadrature points (proportional to the number of cells of the immersed mesh), and N is the number of cells of the background mesh.

The main drawbacks of this approach are twofold: i) on one side it does not yield the required accuracy even if quadrature rules with the appropriate order are used, due to the piecewise polynomial nature of the integrands, that may have discontinuities within the element K, and ii) since the couplings between the two grids is based solely on a collection of quadrature points on the immersed surface, it may happen that two elements overlap, but no quadrature points fall within the intersection of the two elements.

This is illustrated in a pathological case in Figure 1.3 (top), where we show a zoom-in of a solution computed with an insufficient number of quadrature points, and an overly refined background grid. In this case the quadrature rule behaves like a collection of Dirac delta distributions, and since the resolution of the background grid is much finer than the resolution of the immersed grid, one can recognize in the computed solution the superposition of many small fundamental solutions, that, in the two-dimensional case, behave like many logarithmic functions centered at the quadrature points of γ_h.

These issues with the quadrature driven approach may hinder the convergence properties of the methods, as shown in [42] for a 2D-2D problem, and require a careful equilibrium between the

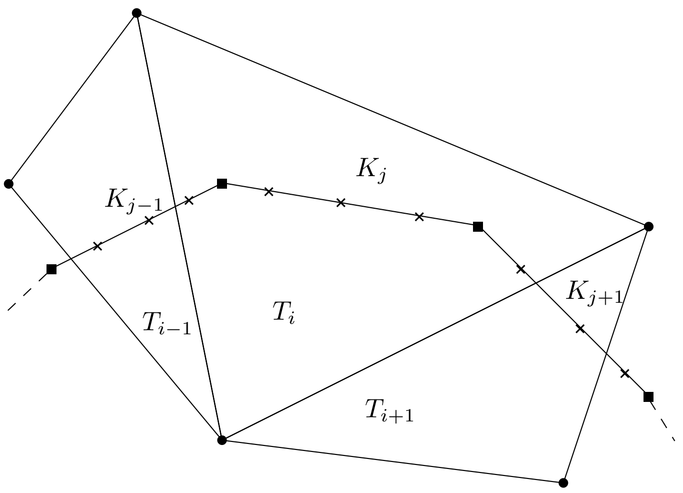
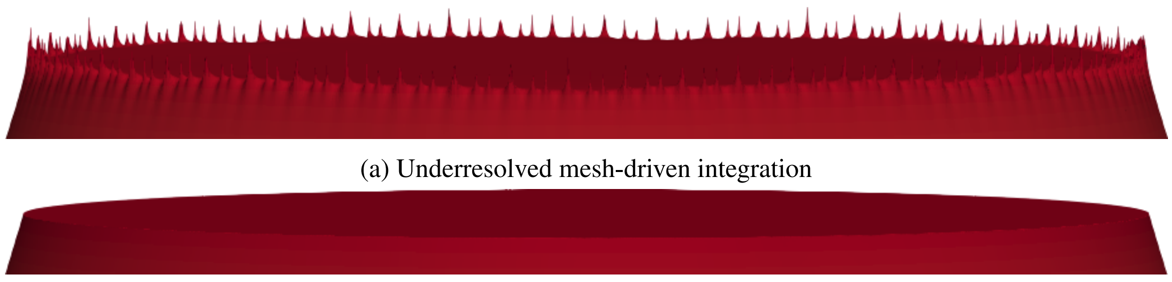


Fig. 1.2 Squares: DoF for linear basis functions attached to some elements K_j of γ intersecting a background element $T_i \in \Omega_h$. Dots: DoF for a linear Lagrangian basis attached to cells T_i. Crosses: quadrature points corresponding to a Gaussian quadrature rule of order 3 on elements of γ.



(a) Underresolved mesh-driven integration

(b) Integration using mesh intersection

Fig. 1.3 Comparison between exact and non exact integration (zoomed-in on the interface in order to highlight the kinks) for a pathological case. (a) Spurious kinks around the interface, on the location of the immersed quadrature points. (b) Well-resolved solution around the interface.

resolution of the immersed grid γ_h, the choice of the quadrature formula on K, and the resolution of the background grid Ω_h. Alternatively, one can follow a different approach that is based on the identification of the intersections between the two grids, and on the use of a quadrature rules defined on the intersection of the two elements, to remove the artifacts discussed above (at the cost of computing the intersection between two non-matching grids), as shown in Figure 1.3 (bottom).

1.4.2 Integration on mesh intersections

An accurate computation of the interface terms may be performed by taking into account the intersection between the two grids. First, the non-empty intersections $\tilde{E} := T \cap K \neq \emptyset$ between any $T \in \Omega_h$ and $K \in \gamma_h$ are identified and the intersection is computed accordingly. Then, given that the restriction of v_h to $\tilde{E}$ is smooth, a suitable quadrature formula can be applied in $\tilde{E}$. Since $\tilde{E}$ is a polygon in general, we use a sub-tessellation technique, consisting in splitting $\tilde{E}$ into sub-elements $S \in S_{\tilde{E}}$ such that $\tilde{E} = \bigcup_{S \in S_{\tilde{E}}} S$ and we use standard Gaussian quadrature rules on each of these sub-elements. See Figures 1.4 and 1.5 for an example in two and three dimensions, respectively.

In conclusion, interface terms such as $\langle \lambda_h, v_h \rangle_\gamma$ are assembled by summing the contribution of each intersection $\tilde{E}$ computed by appropriate quadrature:

$$\langle \lambda_h, v_h \rangle_{\tilde{E}} = \sum_{S \in S_{\tilde{E}}} \langle \lambda_h, v_h \rangle_S \approx \sum_{S \in S_{\tilde{E}}} \sum_{q=1}^{N_q} \lambda_h(F_S(\hat{x}_q)) v_h(F_S(\hat{\mathbf{x}}_q)) JF_S(\hat{\mathbf{x}}_q) w_q, \tag{1.24}$$

where $F_S \colon \hat{K} \to S_S$ is the mapping from the reference element $\hat{K}$ to S and $\{\hat{\mathbf{x}}_q, w_q\}_{q=1}^{N_q}$ a suitable quadrature rule defined on $\hat{K}$.

The efficient identification of pairs of potentially overlapping cells is performed using queries to R-trees of axis-aligned bounding boxes of cells for both the background and immersed mesh. While this allows to record the indices of the entries to be allocated during the assembly procedure and the relative mesh iterators, the actual computation of the *geometric* intersection $\tilde{E}$ between two elements $T \subset \Omega_h$ and $K \subset \gamma$ is computed using the free function `CGAL::intersection()`. The resulting polytope is sub-tessellated into simplices, i.e., $\tilde{E} = \cup_{i=0}^{N_s} S_i$, even though other techniques for numerical quadrature on polygons may be employed [59].

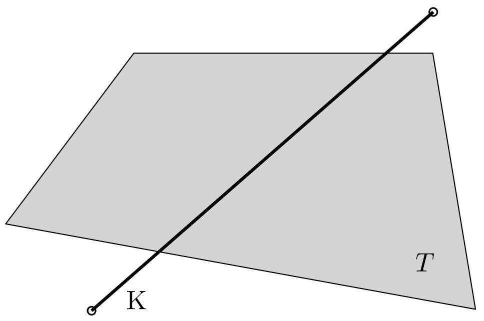
In Figures 1.4 and 1.5 we show this procedure graphically for cells T and K and provide an example for the corresponding sub-tessellations of the intersection $\tilde{E}$ in two and three dimensions. The resulting sub-tesselations are used to construct the quadrature rules $\{Q_i\}_i$ in (1.24).

The overall complexity of the algorithm is $O((N+M)\log(N))$, where N and M are the numbers of cells in the background and immersed mesh, respectively. Notice, however, that the complexity of the algorithm should be multiplied by the complexity of the `CGAL::intersection()` function, which is $O(nm)$ for the intersection of two polygons with n and m vertices, respectively. In practice, this is not a problem since the number of vertices of the polygons is typically small, and the complexity of the algorithm is dominated by the complexity of the R-tree queries, but these costs are non-negligible for large grids, and many overlapping cells (see, e.g., the discussion in [42]).

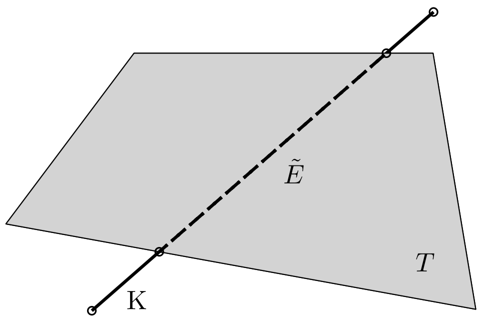
1.4.3 Integration through level set splitting

If a level set description of the immersed domain is available, this may be used to generate quadrature formulas without explicitly computing the geometric intersection (see [53] for some implementation details). It has also been shown in [162] how to generate quadrature rules on different regions of a cut element using Ψ, identified as

$$O = \{(x,y) \in T : \Psi > 0\},$$
$$S = \{(x,y) \in T : \Psi = 0\},$$
$$I = \{(x,y) \in T : \Psi < 0\},$$



(a) Background cell T and foreground cell K.



(b) The intersection $\tilde{E}$ is computed and triangulated. In this case, resulting in a single one-dimensional simplex (a segment).

Fig. 1.4 Triangulation of the intersected region $\tilde{E}$ for a line immersed in 2D. The one-dimensional cell K is allowed to be positioned arbitrarily w.r.t. to the two-dimensional quadrilateral T.

where $\Psi : \mathbb{R}^d \to \mathbb{R}$ is the level set function determining the immersed domain. A typical configuration including quadrature points for each entity is shown in Figure 1.6.

In practical computations, however, the interface γ is not available in terms of a simple analytical description of the level set function. Should one still wish to use quadrature formulas based on a level set, a discrete (possibly approximated) level set function Ψ_h that is zero on γ_h must be provided when γ_h is a triangulated surface. Such level set would also allow a robust partitioning of the background computational mesh into cells that are completely inside ω, cells cut by γ, and cells that are completely inside $\Omega \setminus \omega$.

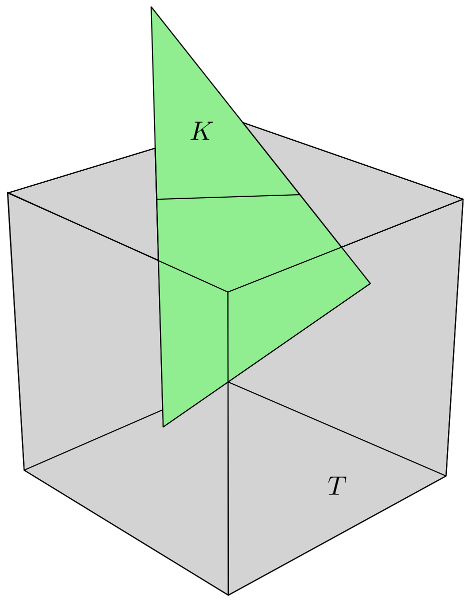
We propose a simple implementation of a discrete level set function Ψ_h constructed from a triangulated interface γ_h. Point classification (i.e., detecting if a point is inside or outside ω) is performed using a query to the `CGAL` library, to detect if a point is inside or outside the coarsest simplicial mesh bounded by γ_h (denoted by $\mathcal{I}_h$).

We then define a *discrete* level set function $\Psi_h(\boldsymbol{x})$ on top of γ_h as follows:

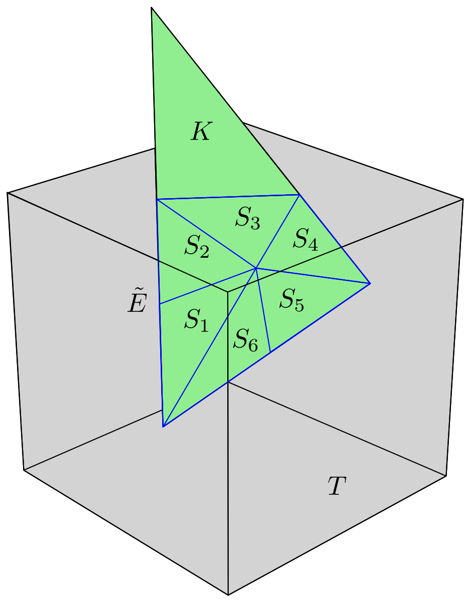
$$\Psi_h(\boldsymbol{p}) = \begin{cases} -d(\boldsymbol{p},\gamma) & \text{if } \boldsymbol{p} \in \mathcal{I}_h, \\ d(\boldsymbol{p},\gamma) & \text{if } \boldsymbol{p} \notin \mathcal{I}_h, \end{cases}$$

where $d(\boldsymbol{p},\gamma) := \min_{\boldsymbol{y}\in\gamma} d(\boldsymbol{p},\boldsymbol{y})$ is constructed by first finding the closest elements of γ to $\boldsymbol{p}$ using efficient R-tree data structures indexing the cells of the immersed triangulation, and then computing the distance between $\boldsymbol{p}$ and those elements.

This procedure is summarized in Algorithm 1. We validate our approach with a manufactured case by choosing randomly distributed points $\{p_i\}_i$ in the interval $[-1,1]^2$ and computing the relative error $E_r(\boldsymbol{p}) := \frac{|\Psi(\boldsymbol{p})-\Psi_h(\boldsymbol{p})|}{|\Psi(\boldsymbol{p})|}$ for a level set describing a circle of radius $R = 0.3$, and a corresponding approximated grid γ_h.

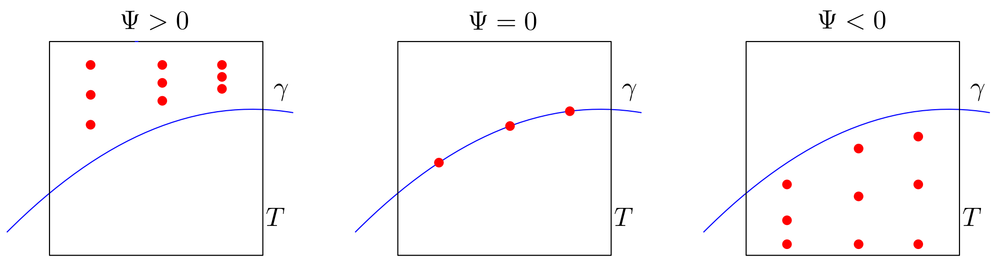


(a) In grey the background cell, in green the immersed one.



(b) The intersection $\tilde{E}$ is computed and triangulated.

Fig. 1.5 Triangulation of the intersected region $\tilde{E}$ for a triangle cutting a square and the relative sub-tessellation.



(a) (b) (c)

Fig. 1.6 Quadrature points distributed on O, S and I for a cell T cut by the curve γ.

In Figure 1.7 we report the relative error committed by replacing the exact level set with the discrete one (induced by replacing the exact curve with a triangulated one). The initial discretization of γ is chosen so that the error is below 10^{-6} everywhere, and we can safely neglect it when computing convergence rates of the error computed w.r.t. exact solutions known on the analytical level set.

As long as such geometrical error is not dominating we can observe optimal rates in the numerical experiments for cut-FEM. At the same time, this setting allows for a fair comparison between all the schemes. We expect that other practical implementations would require similar tasks and hence that the observed computational cost is representative.

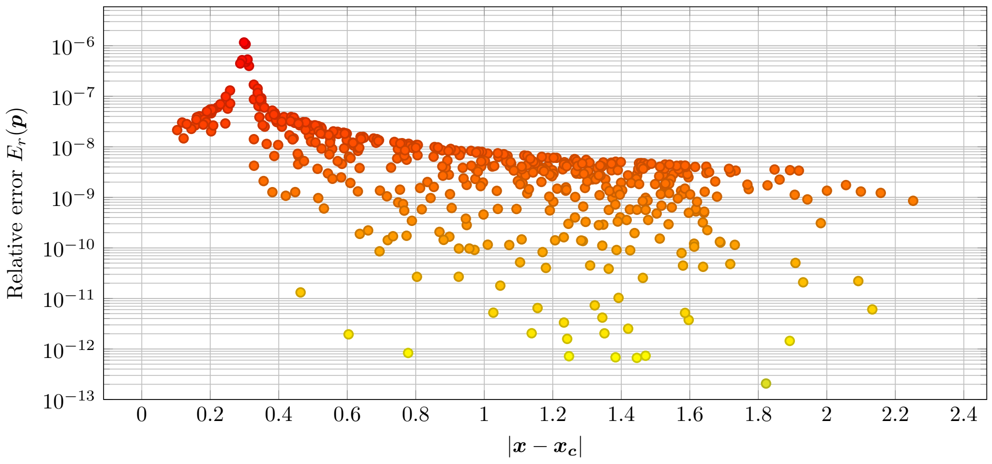


Fig. 1.7 The relative error for the discrete level set describing a disk of radius $R = 0.3$ as a function of the distance from the center $\boldsymbol{x_c} = (\frac{1}{2}, \frac{1}{2})$.

Algorithm 1: Evaluation of the discrete level set (1.4.3) for a given mesh γ.

```
  Input  : $\gamma$ polygonal surface mesh,
           $\mathcal{T}_h$ coarse triangulation for the interior of $\gamma$,
           $\boldsymbol{p} \in \Omega$.
  Output : $d(\boldsymbol{p}, \gamma)$.
1 if $\boldsymbol{p} \in \mathcal{T}_h$ then
2     $s \leftarrow -1$
3 else
4     $s \leftarrow +1$
5 Find$\{K_i\} \in \gamma_h$ nearest to $\boldsymbol{p}$.
6 for $K \in \{K_i\}_i$ do
7     $d_i \leftarrow d(\boldsymbol{p}, K)$;
8 Return $s \cdot \min_i d_i$
```

1.5 Numerical experiments

Our implementation is based on the C++ finite element library `deal.II` [22, 21], providing a dimension independent user interface. The implementation of the Lagrange multiplier and of the Nitsche's interface penalization methods are adapted from the tutorial programs `step-60` and `step-70` of the deal.II library, respectively, while the cut-FEM algorithm is adapted from the tutorial program `step-85`, developed in [166].

As a result of this chapter, we added support and wrappers for the C++ library `CGAL` ([172], [83]) into the `deal.II` library [22], in order to perform most of the computational geometry related tasks.

Thanks to the so called *exact computation paradigm* provided by `CGAL`, which relies on computing with numbers of arbitrary precision, our intersection routines are guaranteed to be robust.

We assume that the background mesh Ω_h is a d-dimensional triangulation and the immersed mesh γ_h is $(d-1)$-dimensional with $d = 2,3$. We validate our implementations with several experiments varying mesh configurations, algorithms, and boundary conditions. The source code used to reproduce the numerical experiments is available on GitHub [1].

The tests are designed to analyze the performance of the methods presented in Section 1.3 in different settings, varying the complexity of the interface and the smoothness of the exact solution in both two and three dimensions. All tests are performed using background meshes made of quadrilaterals or hexahedra and immersed boundary meshes made of segments and quadrilaterals. For the Lagrange multiplier and Nitsche's interface penalization methods, we perform an initial pre-processing of the background grid Ω by applying a localized refinement around the interface (where most of the error is concentrated), so that the resulting number of degrees of freedom for the variable u_h is roughly the same for all methods. Sample grids resulting from this process are shown in Figure 1.8, where the interface has been resolved from Ω_h. We then proceed by computing errors and convergence rates against a manufactured solution under simultaneous refinement of both the background and immersed mesh.

Classical $\mathcal{Q}^1$ Lagrangian elements are used for the background space while piecewise constant elements are used to discretize the Lagrange multiplier. For the Nitsche penalization method (1.14), we set the penalty parameter as $\beta = 10$. Errors in the H^1- and L^2-norm are reported for the main variable while for the Lagrange multiplier we use the discrete norm in (1.7) which, as already observed, is equivalent to the $H^{-1/2}(\gamma)$ norm on a quasi-uniform mesh. In the case of the Lagrange multiplier method, we report the sum of the number of Degrees of Freedom (DoF) for u_h and λ_h to underline the fact that a larger system must be solved, while rates are computed against the number of DoF of each unknown.

For the (symmetric) penalized methods, the resulting linear systems are solved using a preconditioned conjugate gradient method, with an algebraic multigrid preconditioner based on the `Trilinos ML` implementation [113], while for the Lagrange multiplier we exploit the preconditioner described in Section 1.3.1 with the same preconditioned conjugate gradient method for inner solves of the stiffness matrices, and flexible GMRES as the outer solver. In the case of the Lagrange multiplier method, we list the total number of inner iterations required to invert the Schur complement.

1.5.1 2D numerical tests

In order to make the comparison between the three methods as close to real use cases as possible, we do not exploit any *a-priori* knowledge of analytical level set descriptions of the exact interfaces. Indeed, using this information one could expect faster computations of the intersections, and an overall

[1]https://github.com/fdrmrc/non_matching_test_suite.git

reduction of the computational costs of the assembly routines. Instead, we fix the same discretization of the interface as input data of the computational problem for all three methods.

In particular, we consider as computational domain $\Omega = [-1,1]^2$ with immersed domains of different shapes originating from an unfitted discretization of two different curves:

- *circle interface;* we let $\gamma^1 := \partial B_R(\boldsymbol{c})$,
- *flower-shaped interface;* we let
$\gamma^2 := \left\{(x,y) : \sqrt{x^2+y^2} - r\left(1-2\frac{y^2}{x^2+y^2}\right)\left(1-16\frac{x^2y^2}{(x^2+y^2)^2}\right) - R = 0\right\}$,

where the first is a circle of radius R centered at $\boldsymbol{c}$ and the second is a flower-like interface. We choose as parameter values $\boldsymbol{c} = (\frac{1}{2},\frac{1}{2})$, $R = 0.3$ and $r = 0.1$. The initial discretization of the immersed domains are chosen so that the geometrical error is negligible w.r.t. the discretization errors (see Figure 1.8). In the case of the circle interface γ^1, the discrete interface is generated using a built-in mesh generators from `deal.II` while the flower-shaped interface γ^2 is imported from an external file. When required, we implement a discrete level set function Ψ_h as described in Algorithm 1. Figure 1.8 shows a representation of the two interfaces.

In the first two tests, we set up the problem using the method of manufactured solutions, imposing the data f in Ω and the boundary conditions on Γ and γ according to the exact solution

$$u(x,y) := \sin(2\pi x)\sin(2\pi y). \tag{1.25}$$

Hence, the right hand side of (1.1) is $f = 8\pi^2 u(x,y)$ and the data on the outer boundary Γ and on γ are computed accordingly. Notice that the smoothness of the solution implies $\lambda \equiv 0$ in this case, meaning that the Dirichlet condition does not constraint the solution u at the interface.

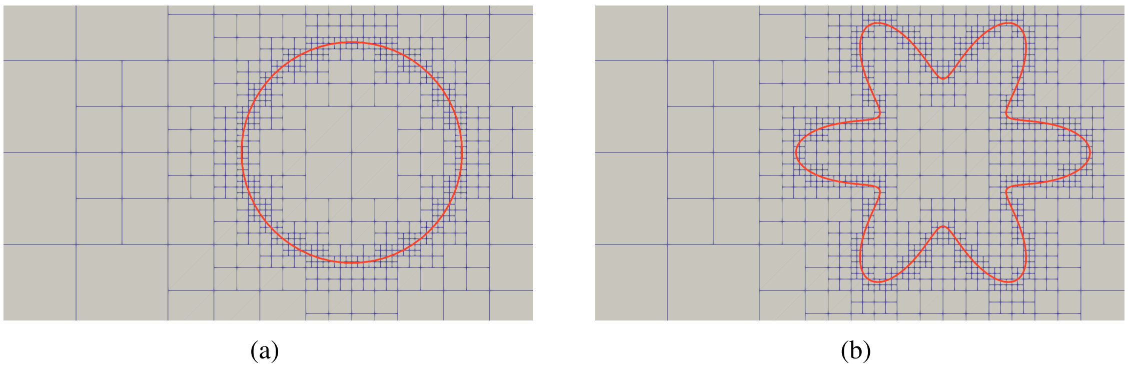

(a) (b)

Fig. 1.8 Zoom on pre-processed background grid Ω_h for the circle interface $\gamma = \gamma^1$ (left) and the flower-shaped interface $\gamma = \gamma^2$ (right).

This test is meant to assess the basic correctness of the implementation of the three methods, and corresponds to a case in which the interface is truly not an interface. Instead, when we impose an arbitrary value for the solution at the interface, we expect the solution u to be only in $H^{\frac{3}{2}-\varepsilon}(\Omega)$ for any $\varepsilon > 0$, even thought its local regularity on the two subdomains ω and $\Omega \setminus \omega$ may be higher. This is due to the fact that the gradient of the solution is not a continuous function across the interface, and therefore the solution cannot be in $H^2(\Omega)$. In this case, we cannot expect non-matching methods that does not resolve the interface *exactly*, such as the Lagrange multiplier and the Nitsche's interface penalization method, to be able to recover the optimal rate of convergence.

In the tables below we also report the number of iterations required in the solution phase in the column 'Iter.'. We observe for all experiments a similar number of iterations for the three methods (which are independent on the number of degrees of freedom, indicating a good choice of preconditioner for all three methods) even though the solution of the linear system stemming from the Lagrange multiplier method is generally more expensive compared to the other two methods, owing to the higher computational complexity of the preconditoner for the saddle-point problem. The balance in the computational cost of the three different methods is discussed in details in Section 1.5.3.

Test 1: smooth solution over circular interface

We report in Tables 1.1, 1.2, and 1.3 the errors and computed rates for the Lagrange multiplier, Nitsche's interface penalization, and cut-FEM method, respectively. In each case, the background variable converges linearly and quadratically in the H^1- and L^2-norm, respectively. As for the Lagrange multiplier, we observe a convergence rate close to two instead of the theoretical rate of one, most likely due to the very special exact solution that the multiplier converges to (i.e., the zero function). For a direct comparison, we also report in Figure 1.9 (left) the convergence history of all three methods against the number of DoF. These results clearly indicate that for smooth problems with relatively simple interfaces the three methods perform similarly.

Results for $\gamma = \gamma^1$ and smooth solution with Lagrange multiplier							
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	$\|\lambda-\lambda_h\|_{-\frac{1}{2},\gamma}$	$H^{-\frac{1}{2}}(\gamma)$ *rate*	Iter.
389+32	5.579e-02	-	1.879e+00	-	7.402e-02	-	7
1721+64	1.393e-02	1.87	9.391e-01	0.93	1.042e-02	2.83	7
7217+128	3.479e-03	1.94	4.690e-01	0.97	2.805e-03	1.89	9
29537+256	8.691e-04	1.97	2.343e-01	0.98	6.716e-04	2.06	11
119489+512	2.172e-04	1.98	1.171e-01	0.99	2.078e-04	1.69	11

Table 1.1 Rates in L^2 and H^1 for a smooth u and $H^{-\frac{1}{2}}$ rates for the Lagrange multiplier method.

Results for $\gamma = \gamma^1$ and smooth solution with Nitsche					
DoF number	$\|\|u-u_h\|\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|\|u-u_h\|\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
389	5.597e-02	-	1.879e+00	-	1
1721	1.396e-02	1.87	9.391e-01	0.93	11
7217	3.487e-03	1.94	4.690e-01	0.97	11
29537	8.712e-04	1.97	2.343e-01	0.98	12
119489	2.177e-04	1.98	1.171e-01	0.99	12

Table 1.2 L^2 and H^1 error rates for $\gamma = \gamma^1$ and a smooth solution with Nitsche.

Results for $\gamma = \gamma^1$ and smooth solution with cut-FEM					
DoF number	$\|\|u-u_h\|\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|\|u-u_h\|\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
329	8.7560e-02	-	2.1335e+00	-	2
1161	2.2064e-02	1.99	1.0582e+00	1.01	2
4377	5.4875e-03	2.01	5.2351e-01	1.02	15
16953	1.3554e-03	2.02	2.5863e-01	1.02	17
66665	3.3152e-04	2.03	1.2855e-01	1.01	18

Table 1.3 L^2 and H^1 error rates for $\gamma = \gamma^1$ and a smooth solution with cut-FEM.

Test 2: smooth solution over flower-shaped interface

We report in Tables (1.4), (1.5) and (1.6) the error with the computed rates of convergence for the three schemes. Again we observe the theoretical rates of convergence for all three methods. This time, however, the direct comparison of the errors shown in Figure 1.9 (right) indicates that the cut-FEM approach has an advantage over the other two methods. This may be due to the worst approximation properties of the non-matching methods based on Lagrange multipliers and Nitsche's interface penalization in the presence of high curvature sections of the immersed boundary, as can be seen by comparing the meshes shown in Figure 1.8. Despite the fact that the reached accuracy is the same for the Lagrange multiplier and We further note that the Nitsche's interface penalization method requires less iterations than the Lagrange multiplier method.

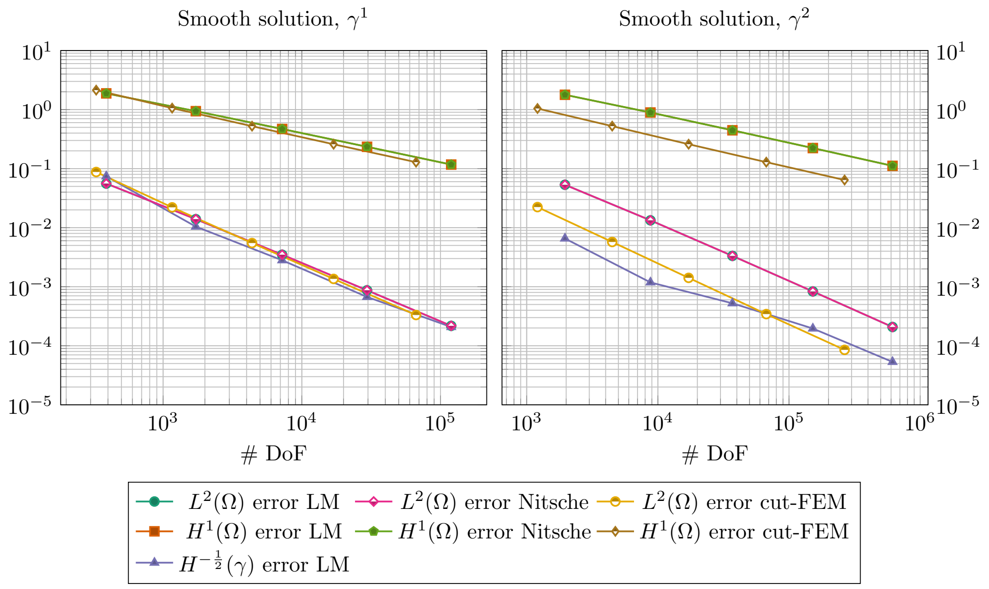


Fig. 1.9 L^2, H^1, and $H^{-\frac{1}{2}}$ errors versus the number of DoF for all schemes applied to γ^1 (left) and to γ^2 (right).

Results for $\gamma = \gamma^2$ and smooth solution with Lagrange multiplier							
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	$\|\lambda-\lambda_h\|_{-\frac{1}{2},\gamma}$	$H^{-\frac{1}{2}}(\gamma)$ *rate*	Iter.
1958+256	5.327e-02	-	1.781e+00	-	6.551e-03	-	17
8783+512	1.331e-02	1.85	8.916e-01	0.92	1.183e-03	2.47	20
37037+1024	3.326e-03	1.93	4.458e-01	0.96	5.237e-04	1.18	33
151961+2048	8.312e-04	1.96	2.228e-01	0.98	1.959e-04	1.42	49
615473+4096	2.078e-04	1.98	1.114e-01	0.99	5.327e-05	1.88	53

Table 1.4 L^2 and H^1 error rates for $\gamma = \gamma^2$ and a smooth solution with Lagrange multiplier method.

Results for $\gamma = \gamma^2$ and smooth solution with Nitsche					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
1958	5.327e-02	-	1.781e+00	-	13
8783	1.331e-02	1.85	8.916e-01	0.92	12
37037	3.326e-03	1.93	4.458e-01	0.96	13
151961	8.312e-04	1.96	2.228e-01	0.98	12
615473	2.078e-04	1.98	1.114e-01	0.99	14

Table 1.5 L^2 and H^1 error rates for $\gamma = \gamma^2$ and a smooth solution with Nitsche

Results for $\gamma = \gamma^2$ and smooth solution with cut-FEM					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
1209	2.2306e-02	-	1.0422e+00	-	2
4487	5.7207e-03	1.96	5.2424e-01	0.99	17
17163	1.4123e-03	2.02	2.5901e-01	1.02	17
67089	3.4377e-04	2.04	1.2865e-01	1.01	19
265249	8.5223e-05	2.01	6.4118e-02	1.00	20

Table 1.6 L^2 and H^1 error rates for $\gamma = \gamma^2$ and a smooth solution with cut-FEM

Test 3: non-smooth solution over circular interface

In this test, we fix once more $\gamma = \gamma^1$ and we define an exact solution with a non-zero jump of the normal gradient $\nabla u \cdot \boldsymbol{n}$ across γ taken from [111], namely

$$u(x,y) = \begin{cases} -\ln(R) & \text{if } |r| \leq R \\ -\ln(r) & \text{if } |r| > R, \end{cases} \tag{1.26}$$

where $r := \boldsymbol{x} - \boldsymbol{c}$, implying as right hand side $f = 0$. The Lagrange multiplier associated to this solution is $\lambda(\boldsymbol{x}) \equiv \lambda = -\frac{1}{R}$, as u solves the following classical interface problem:

$$\begin{cases} -\Delta u & = 0 \quad \text{in } \Omega \setminus \gamma, \\ u & = -\ln(r) \quad \text{on } \Gamma, \\ [\![\nabla u \cdot \boldsymbol{n}]\!] & = \frac{1}{R} \quad \text{on } \gamma, \\ [\![u]\!] & = 0 \quad \text{on } \gamma. \end{cases} \tag{1.27}$$

Since the global regularity of the solution is $H^{\frac{3}{2}-\varepsilon}(\Omega)$ for any $\varepsilon > 0$, theoretically we would expect the convergence rates of the $L^2(\Omega)$, $H^1(\Omega)$, and $H^{-\frac{1}{2}}(\gamma)$ norms of the errors to be 1, 0.5, and 0.5 for the Lagrange multiplier method, the same for the variable u_h in the Nitsche's interface penalization method, and the optimal convergence rates observed in the smooth case in the case of the cut-FEM method. These are shown in Tables 1.7, 1.8 and 1.9 and plotted in Figure 1.10.

The results show a clear advantage in terms of convergence rates and absolute values of the errors for the cut-FEM method. In all cases (both smooth and non-smooth), the Nitsche's interface penalization method and the Lagrange multiplier method give essentially the same errors. A simple way to improve the situation for the latter two methods would be to use a weighted norm during the computation of the error, which was proven to be localized at the interface in [111]. This would allow to reduce the overall error, but it would still result in a solution that does not capture correctly the jump of the normal gradient across the interface, which is the main source of error.

Results for $\gamma = \gamma^1$ and non-smooth solution with Lagrange multipliers							
DoF number	$\lVert u - u_h \rVert_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\lVert u - u_h \rVert_{1,\Omega}$	$H^1(\Omega)$ *rate*	$\lVert \lambda - \lambda_h \rVert_{-\frac{1}{2},\gamma}$	$H^{-\frac{1}{2}}(\gamma)$ *rate*	Iter.
389+32	1.413e-02	-	3.656e-01	-	3.354e-02	-	2
1721+64	5.698e-03	1.22	2.254e-01	0.65	1.112e-02	1.59	2
7217+128	3.248e-03	0.78	1.621e-01	0.46	6.273e-03	0.83	2
29537+256	1.796e-03	0.84	1.141e-01	0.50	4.094e-03	0.62	2
119489+512	1.099e-03	0.70	8.015e-02	0.51	2.845e-03	0.52	2

Table 1.7 L^2-error and H^1-error for non smooth u in (1.26) and for the multiplier.

Results for non-smooth u with Nitsche					
DoF number	$\lVert u - u_h \rVert_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\lVert u - u_h \rVert_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
389	9.216e-03	-	3.667e-01	-	1
1721	3.324e-03	1.37	2.286e-01	0.64	11
7217	1.936e-03	0.75	1.640e-01	0.46	11
29537	9.957e-04	0.94	1.151e-01	0.50	12
119489	5.016e-04	0.98	8.070e-02	0.51	13

Table 1.8 L^2-error, H^1-error for non smooth u in (1.26).

Results for non-smooth u with cut-FEM					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
329	3.1670e-02	-	5.6660e-01	-	2
1161	1.5998e-03	4.31	1.2352e-01	2.20	2
4377	3.6791e-04	2.12	5.8327e-02	1.08	16
16953	8.5282e-05	2.11	2.7307e-02	1.09	18
66665	2.1914e-05	1.96	1.3726e-02	0.99	19

Table 1.9 L^2-error, H^1-error for non smooth u in (1.26) with cut-FEM.

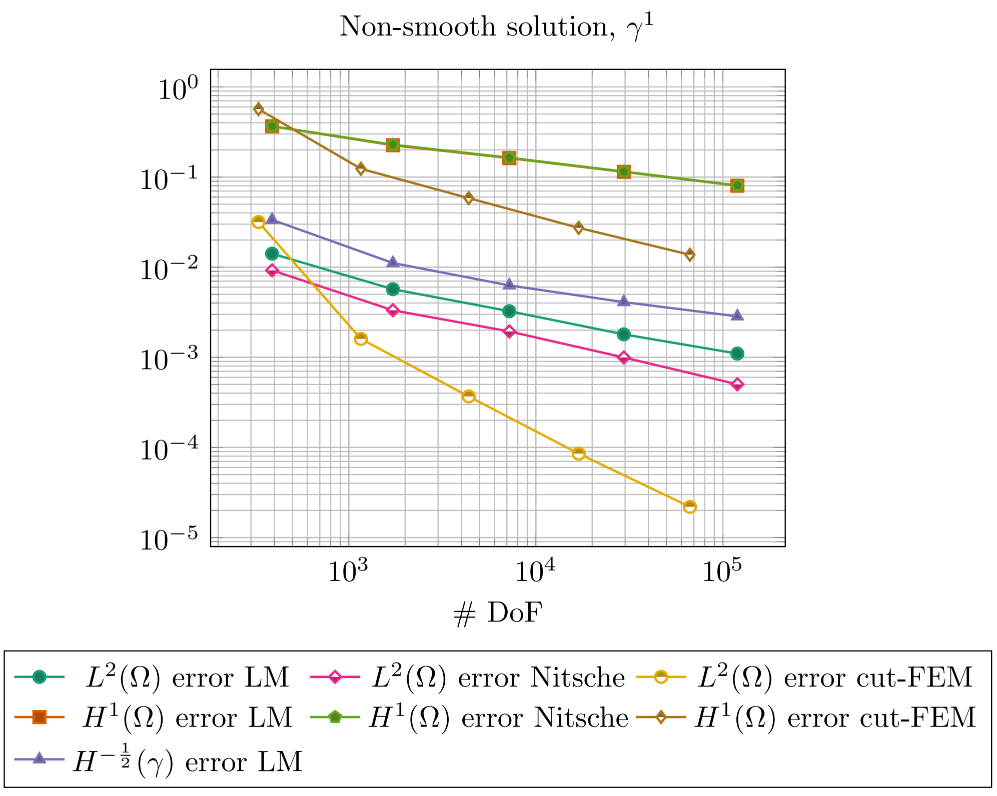


Fig. 1.10 L^2, H^1, and $H^{-\frac{1}{2}}$ errors versus the number of DoF for all schemes applied to γ^1, with non-smooth solution.

1.5.2 3D numerical tests

We mimic the tests reported above for the two-dimensional setting also in the three-dimensional case, but we restrict our analysis to the case of a sphere immersed in a box. We fix $\Omega = [-1,1]^3$

and consider as immersed interface a sufficiently fine discretization of the sphere $\gamma^3 := \partial B_R(\boldsymbol{c})$ with $\boldsymbol{c} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, $R = 0.3$.

In this setting, we consider two test cases with smooth and non-smooth solution.

Test 1: smooth solution over spherical interface

We proceed analogously to the previous section by computing convergence rates when the solution u is smooth and defined as

$$u(x,y,z) := \sin(2\pi x)\sin(2\pi y)\sin(2\pi z), \tag{1.28}$$

which corresponds to the right hand side $f = 12\pi^2 u(x,y,z)$. We report in Tables 1.10, 1.11 and 1.12 the error rates for the three schemes, while in Figures 1.11 (left) we plot errors against the number of DoF.

The convergence rates are as expected and the results are in line with the two-dimensional case. The Lagrange multiplier method and the interface penalization method give again very close computational errors for u_h. As in the smooth two-dimensional case, this test should only be considered as a validation of the code and of the error computation, since the interface does not have any effect on the computational solutions.

Results for $\gamma = \gamma^3$ and smooth solution with Lagrange multiplier							
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	$\|\lambda-\lambda_h\|_{-\frac{1}{2},\gamma}$	$H^{-\frac{1}{2}}(\gamma)$ rate	Iter.
4127+96	6.416e-02	-	2.432e+00	-	0.1027	-	9
37031+384	1.590e-02	1.91	1.212e+00	0.95	0.0287	1.84	14
313007+1536	3.963e-03	1.95	6.051e-01	0.98	0.0069	2.06	19
2572511+6144	9.996e-04	1.96	3.024e-01	0.99	0.0019	1.89	14

Table 1.10 L^2 and H^1 error rates for $\gamma = \gamma^3$ and a smooth solution with Lagrange multiplier.

Results for $\gamma = \gamma^3$ and smooth solution with Nitsche					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
4127	6.411e-02	-	2.432e+00	-	17
37031	1.588e-02	2.01	1.212e+00	1.00	14
313007	3.959e-03	2.00	6.050e-01	1.00	14
2572511	9.887e-04	2.00	3.023e-01	1.00	14

Table 1.11 L^2 and H^1 error rates for $\gamma = \gamma^3$ and a smooth solution with Nitsche.

Results for $\gamma = \gamma^3$ and smooth solution with cut-FEM					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
5163	7.0847e-02	-	2.4835e+00	-	24
36781	1.7743e-02	2.12	1.2479e+00	1.05	21
278157	4.5124e-03	2.03	6.2396e-01	1.03	22
2160541	1.1302e-03	2.03	3.1154e-01	1.02	22

Table 1.12 L^2 and H^1 error rates for $\gamma = \gamma^3$ and a smooth solution with cut-FEM

Test 2: non-smooth solution over spherical interface

We consider the test case in [111]:

$$u(x,y) = \begin{cases} \frac{1}{R} & \text{if } |r| \le R, \\ \frac{1}{|r|} & \text{if } |r| > R, \end{cases} \tag{1.29}$$

where $r := \boldsymbol{x} - \boldsymbol{c}$, $f = 0$, and $R = 0.3$. Analogously to the two dimensional case, the multiplier associated to this problem is $\lambda(\boldsymbol{x}) = \lambda = -\frac{1}{R^2}$, since u solves the following problem:

$$\begin{cases} -\Delta u & = 0 \quad \text{in } \Omega \setminus \gamma, \\ u & = \frac{1}{|r|} \quad \text{on } \Gamma, \\ [\![\nabla u \cdot \boldsymbol{n}]\!] & = \frac{1}{R^2} \quad \text{on } \gamma, \\ [\![u]\!] & = 0 \quad \text{on } \gamma. \end{cases} \tag{1.30}$$

We report in Tables 1.13, 1.14, 1.15 the error rates for the three schemes and in Figure 1.11 (right) the error decay in $L^2(\Omega)$ and $H^1(\Omega)$ for u_h and the decay in $H^{-\frac{1}{2}}(\gamma)$ for λ_h. A contour plot of the discrete solution u_h obtained with Nitsche's penalization method is shown in Figure 1.12.

The difference between the three methods is less evident in terms of absolute values of the errors when compared to the two dimensional case: while it is still clear that the convergence rates of cut-FEM are higher compared to the other two methods, the difference between the three methods is smaller. In particular, the Nitsche's interface penalization method seems to perform better than the Lagrange multiplier method when considering the L^2 norm.

Results for $\gamma = \gamma^3$ and non-smooth solution with Lagrange multiplier							
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	$\|\lambda-\lambda_h\|_{-\frac{1}{2},\gamma}$	$H^{-\frac{1}{2}}(\gamma)$ rate	Iter.
4127+96	3.907e-02	-	1.251e+00	-	0.4552	-	2
37031+384	2.041e-02	0.89	8.119e-01	0.59	0.2598	0.81	3
313007+1536	1.095e-02	0.88	5.764e-01	0.48	0.1806	0.52	3
2572511+6144	6.214e-03	0.81	4.253e-01	0.43	0.1216	0.57	3

Table 1.13 L^2 and H^1 error rates for $\gamma = \gamma^3$ and non-smooth solution u in (1.29) with Lagrange multiplier.

Results for $\gamma = \gamma^3$ and non-smooth solution with Nitsche					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
4127	8.688e-02	-	1.562e+00	-	19
37031	2.157e-02	2.01	8.885e-01	0.81	16
313007	4.893e-03	2.14	5.724e-01	0.63	16
2572511	1.887e-03	1.37	3.954e-01	0.53	16

Table 1.14 Rates for $\gamma = \gamma^3$ and a non-smooth solution u in (1.29) with Nitsche.

Results for $\gamma = \gamma^3$ and non-smooth solution with cut-FEM					
DoF number	$\|u-u_h\|_{0,\Omega}$	$L^2(\Omega)$ *rate*	$\|u-u_h\|_{1,\Omega}$	$H^1(\Omega)$ *rate*	Iter.
5163	6.3609e-02	-	1.3046e+00	-	25
36781	6.9076e-03	3.39	4.5097e-01	1.62	22
278157	1.5076e-03	2.26	2.2098e-01	1.06	24
2160541	3.3802e-04	2.19	9.3590e-02	1.26	24

Table 1.15 L^2 and H^1 error for $\gamma = \gamma^3$ and a non-smooth solution u in (1.29) with cut-FEM

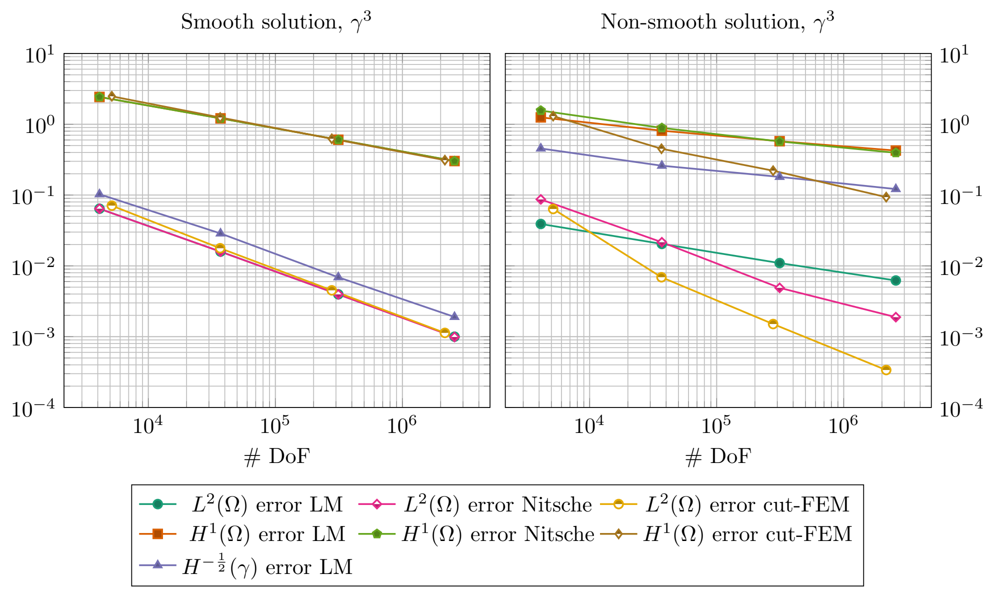


Fig. 1.11 L^2, H^1, and $H^{-\frac{1}{2}}$ error versus the number of DoF for all schemes applied to γ^3 with a smooth solution u (left) and with a non-smooth solution u (right).

1.5.3 Computational times

In order to better understand the performance of the three methods, we consider a breakdown of the computational costs into work precision diagrams. These provide a more fair measure of efficiency as they take into account the computational cost required to reach a given accuracy.

We report hereafter a breakdown of the computational times needed by our implementations of the three proposed methods. All computations were carried out on a 2.60GHz Intel Xeon processor. For each benchmark we report the average time required to solve 10 times the 3D smooth Problem 1.5.2. We recall that a smooth solution implies a vanishing Lagrange multiplier λ, hence the solution u is not constrained. We compute separately the required CPU times (in seconds) for the main tasks that each scheme has to perform. On a quasi-uniform mesh, the number N of background cells in Ω_h scales with $\mathcal{O}(h_\Omega^{-3})$, and we expect the assembly of the stiffness matrix to scale linearly in the number of cells. On the other hand, the number of facets in γ_h scales with $\mathcal{O}(h_\gamma^{-2})$; in our experiments, the ratio h_Ω/h_γ is kept fixed, therefore we expect the assembly of the coupling terms $\langle \lambda, v\rangle_\gamma$ and $\langle u, v\rangle_\gamma$ to scale with $\mathcal{O}(N^{\frac{2}{3}})$.

This is indeed what we observe in the experiments as shown in CPU breakdown plots of Figure 1.13 for each method.

The three schemes have comparable computational times. In particular, the Nitsche interface penalization method exhibits lower global assembly times compared to the others. However, it is well known that the number of iterations required to solve the algebraic problem is influenced by the choice of the penalty parameter β in (1.14), which determines simultaneously also the accuracy of the numerical solution u_h. This can be better seen in work-precision diagrams, where we compare the

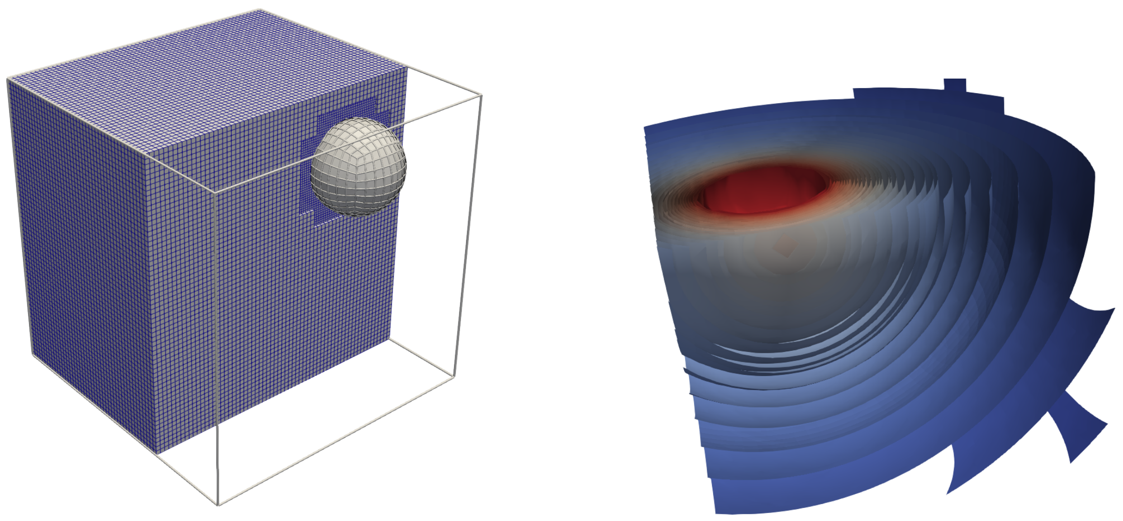


Fig. 1.12 Background mesh Ω_h and immersed mesh of the sphere interface γ_h for the three-dimensional case (left) and section of the contour plot for the approximate solution u_h in (1.29), $\gamma = \gamma^3$.

CPU times to solve each refinement cycle versus the L^2 error for both test problems (1.28) and (1.29) in Figure 1.14. In the smooth scenario, results for the Lagrange multiplier and Nitsche's interface penalization methods are almost overlapping both in terms of time and accuracy, while cut-FEM shows larger computational times. The situation is different in the non-smooth case where cut-FEM better captures the discontinuity at the interface and thus gives more accurate results, with a larger cost in terms of time for low degrees of freedom count, and with smaller cost for large degrees of freedom count, owing to the better convergence rate of the method. These results indicate that the additional implementation complexity eventually pays back for non-smooth solutions. This would be even more the case with higher order elements, as cut-FEM would keep optimal rates while the other methods would not be optimal.

Based on the higher efficiency of the Lagrange multiplier and Nitsche penalization method in the smooth case, we believe that these methods can be made competitive also in the non-smooth case by an improved local refinement strategy [110].

1.6 Conclusions

The numerical solution of partial differential equations modeling the interaction of physical phenomena across interfaces with complex, possibly moving, shapes is of great importance in many scientific fields. Boundary unfitted methods offer a valid alternative to remeshing and Arbitrary Lagrangian Eulerian formulations, but require care in the representation of the coupling terms between the interface and the bulk equations.

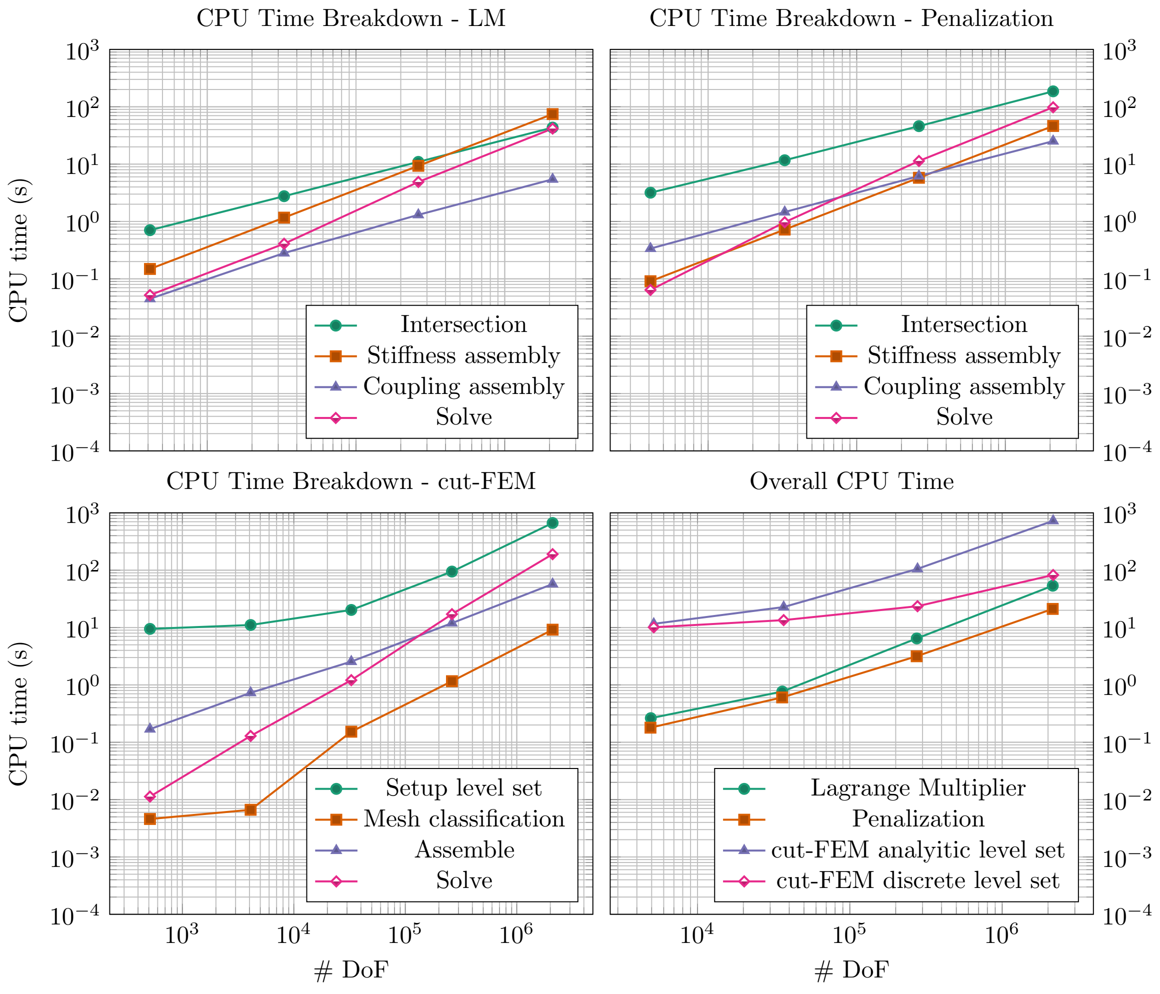


Fig. 1.13 Breakdown of CPU times for the three schemes. In (bottom left) *Setup level set* indicates the time required to interpolate the discrete level set described in Algorithm 1 onto the finite element space. *Mesh classification* shows the time needed to partition the computational mesh and classify the cells in *cut, interior* or *outside* cells. Bottom right: overall CPU times to assemble the algebraic system for the smooth 3D test. For the cut-FEM method, we also show the CPU time obtained by using an analytical representation of the interface through an analytic level set function.

In this chapter, we performed a comparative analysis of three non-matching methods, namely the Lagrange multiplier method (or fictitious domain method), the Nitsche's interface penalization method, and the cut-FEM method, in terms of accuracy, computational cost, and implementation effort.

We presented the major algorithms used to integrate coupling terms on non-matching interfaces, discussed the benefit of computing accurate quadrature rules on mesh intersections, and concluded our analysis with a set of numerical experiments in two and three dimensions.

Our results show that accurate quadrature rules can significantly improve the accuracy of the numerical methods, and that there are cases in which simpler methods, like the Nitsche's interface

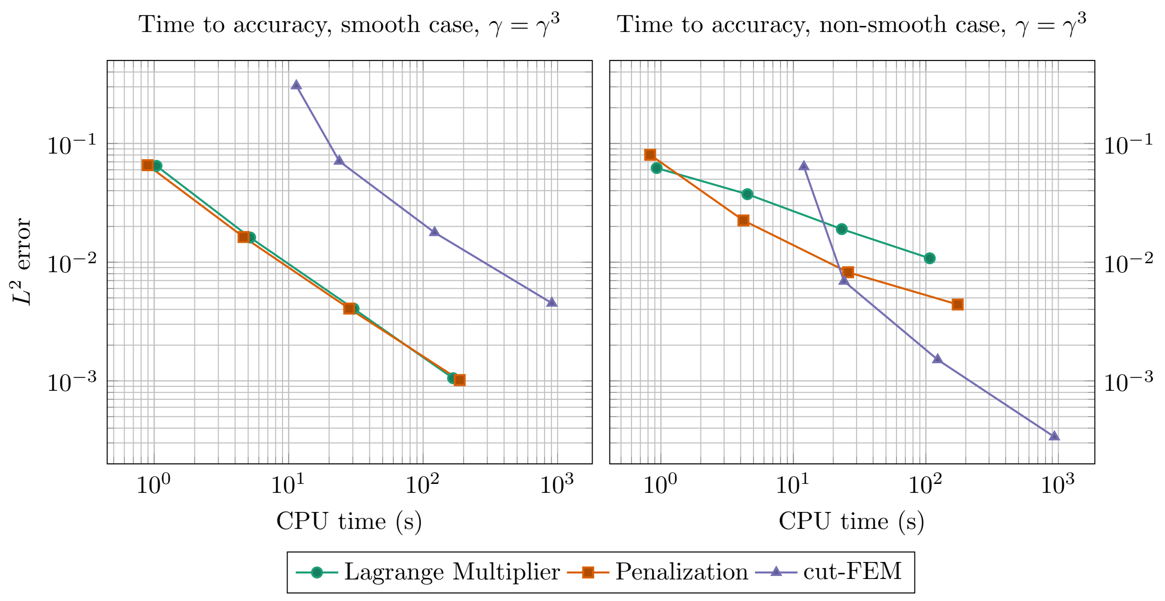


Fig. 1.14 Work-precision diagrams for the two 3D tests.

penalization method, are competitive in terms of accuracy per computational effort. In general, the additional implementation burden of the cut-FEM method is justified by the higher accuracy that it achieves, especially in three dimensions and for the solution of non-smooth problems. Some computational strategies presented in this chapter will be applied in a memory-distributed setting in the next chapter, where non-matching grids will be emplyed in a multilevel context.

Chapter 2

Non-nested multigrid method

Traditionally, the geometric multigrid method is used with nested levels. However, the construction of a suitable hierarchy for very fine and unstructured grids is, in general, highly non-trivial. In this scenario, the non-nested multigrid method could be exploited in order to handle the burden of hierarchy generation, allowing some flexibility on the choice of the levels. We present a parallel, matrix-free, implementation of the non-nested multigrid method for continuous Lagrange finite elements. Each level may consist of independently partitioned triangulations. Our implementation has been added to the multigrid framework of the C++ finite-element library DEAL.II, paving the way to *hp*-multigrid with non-matching levels. Several 2D and 3D numerical experiments with different polynomial degrees and non-trivial geometries show the robustness and performance of proposed implementation.

Contents

2.1 Literature review **42**
2.2 Non-nested multigrid method **43**
2.2.1 Polynomial global coarsening 46
2.3 Implementation details **47**
2.3.1 Distributed geometric search 48
2.3.2 Efficient evaluation on reference positions 50
2.3.3 Domains with curved boundaries 51
2.4 Numerical experiments **52**
2.4.1 Application to nested meshes 54
2.4.2 L-shaped domains 55
2.4.3 Applications to FEA 57
2.5 Performance evaluation **63**
2.5.1 Serial and parallel runs 64
2.5.2 Non-nested tests 66

2.6 Conclusions 71

2.1 Literature review

Scientific and industrial applications often require the solution of differential problems of the form

$$\mathcal{A}u = f \tag{2.1}$$

where $\mathcal{A}$ and f are a given elliptic operator and source term, respectively, and u the solution field of interest. Discretizations relying on the finite element method (FEM) applied to complicated geometries often require large meshes with many unknowns and adaptive mesh refinement in order to capture the behavior of the solution in some regions or to resolve the geometry of the domain. In order to solve the resulting large and sparse linear systems of equations, fast and robust solvers and preconditioners are developed. Among various possibilities, multigrid methods constitute one of the most efficient techniques for elliptic and parabolic problems [175]. Multigrid methods come in various flavors, and the choice of variant strongly depends on the underlying mesh and finite element space. Globally refined meshes generated out of modestly-sized coarse grids are ideal candidates for the geometric multigrid method, which uses each mesh as a level for the multigrid algorithm. In this context, the construction of coarse meshes out of a finer one is a simple task and can be exploited by element agglomeration when the fine grid is structured. On the other hand, for very fine and unstructured meshes on complicated geometries, becomes highly non-trivial the generation of a suitable hierarchy of levels needed by the multigrid algorithm, representing a subject of active research. This is often the case, for instance, for complex industrial CAD (computer-assisted-design) models. In such cases, one may fall back to algebraic multigrid methods (AMG) [167, 70, 71, 95] or to *non-nested* multigrid variants [47, 38]. Multigrid methods for non-nested hierarchies have recently gained renewed attention in the context of polygonal-based methods, where the presence of polyhedra naturally induces non-nestedness between levels and hence an ad-hoc definition of the transfer operators has to be carried out, see e.g. [7, 76] and references therein. In this context, due to the lack of automatic and high-quality coarsening strategies for complicated tetrahedral or hexahedral meshes, it is typical to build each level as an independent remeshing of the problem geometry by changing the mesh size inside a chosen mesh generator.

From the implementation standpoint, the main difficulty in the design of a geometric multigrid method for non-matching levels lies in the implementation of the intergrid operators, used to transfer discrete functions from one level to another. In [79], different options have been thoroughly compared both in terms of accuracy and computational cost for grids of moderate size. The tasks required in the implementation of such transfer operators, as for instance a spatial search procedure to identify which elements of the two relevant meshes are intersecting, are shared by a large class of non-matching finite element techniques such as the ones introduced in Chapter 1. We refer, for instance, to the fictitious domain method, the immersed boundary method, X-FEM, and the distributed Lagrange multiplier scheme [100, 152, 99] as well as to particle-like methodologies [101, 121]. All of them require the computation of coupling operators which represent the interaction between two non-matching meshes over which suitable finite element spaces and physics are defined. The inaccurate computation of the entries of the matrices representing at the algebraic level these operators may in some cases hinder the convergence properties of the method [109, 42, 134].

Despite the different contexts and applications of the works mentioned so far, a shared feature is the explicit storage of the coupling operator as a sparse and generally rectangular matrix that is later used in matrix-vector products. In contrast, we propose a *completely matrix-free* implementation of a geometric multigrid solver and of its transfer operator, applicable for low and moderately high polynomial degrees. For this purpose, we have extended the multigrid infrastructure of DEAL.II [65, 145], which has been developed for nested meshes and optimized in the last decade. We perform comparison with geometric multigrid as well as with AMG and polynomial multigrid, indicating good performance and robust behavior of the new implementation. To the best our author's knowledge, this is the first matrix-free implementation of the multigrid method for non-nested levels. The algorithms described have been developed for continuous Lagrangian finite elements and pointwise interpolation as transfer operator. All of this infrastructure has been added within the C++ finite-element library DEAL.II [21], and are available in the 9.5 release [23]. Note, however, that the main building blocks are generic and could be inserted in other finite element frameworks which support distributed memory parallelism through the MPI standard [103], matrix-free evaluation of weak forms on generic points in reference coordinates as well as efficient geometric search capabilities. The rest of this chapter is organized as follows. In Section 2.2, the non-nested multigrid method and its relevant properties are recalled. Section 2.3 focuses on implementation details related to the matrix-free transfer operator between two arbitrarily overlapping and distributed grids. Section 2.4 presents a series of 2D and 3D numerical experiments that validate the described implementation and then their performance is tested on some examples from classical Finite Element Analysis. Conclusions are drawn in Section 3.7.

2.2 Non-nested multigrid method

The main goal of this section is to describe the prolongation and restriction operators used to transfer between two consecutive grids. We refer the reader to [107] for an extensive presentation of the multigrid method and to [47] for the convergence analysis of multigrid methods with non-nested spaces and non-inherited bilinear forms applied to elliptic problems. For the sake of clarity we consider the classical form of a variational problem: *find $u \in V$ such that*

$$a(u,v) = b(v) \qquad \forall v \in V, \tag{2.2}$$

where V is a suitable functional space defined on a domain $\Omega \subset \mathbb{R}^d$, $d = 2,3$ with Lipschitz continuous boundary. $a(\cdot,\cdot)$ and $b(\cdot)$ are the bilinear and linear forms of the problem, respectively. Let $\{\mathcal{T}_l\}_{l=1,\dots,L}$ denote a family of non-nested and shape-regular partitions of Ω, each one characterized by disjoint open elements K with diameter h_K, such that $\overline{\Omega} = \bigcup_{K\in\mathcal{T}_l} \overline{K}$. The subscript $l \in \{1,\dots,L\}$ is used to index multigrid levels and we indicate with $h_l = \max_{K\in\mathcal{T}_l} h_K$ the mesh size of the l-th level. In our notation, level L consists of the finest mesh, whereas $l = 1$ represents the coarsest level in the hierarchy. The coarser level $\mathcal{T}_l$ is independent of $\mathcal{T}_{l+1}$, with the only refinement constraint that there exists a constant $C > 0$ independent of the discretization parameters such that

$$Ch_l \leq h_{l+1} \leq h_l \qquad l \in \{1,\dots,L-1\}. \tag{2.3}$$

All grids consist of tensor product elements (quadrilaterals or hexahedra) and to each one of the levels we associate a set of degrees of freedom (DoF) $\mathcal{D}^l$ and a conforming Lagrangian finite element space

$$V_l = \{v \in V : v_{|K} \in \mathcal{Q}^p(K), \forall K \in \mathcal{T}_l\} \subset V,$$

with $\mathcal{Q}^p(K)$ the space of polynomials of degree p in each variable. In particular, the fact that $\mathcal{T}_l \not\subset \mathcal{T}_{l+1}$ induces the sequence of finite element spaces to be non-nested: $V_l \not\subset V_{l+1}$. The approximation of (3.1) by finite elements and multigrid techniques requires solving several variational problems of the form: find $u_l \in V_l$ such that

$$a(u_l, v) = b(v) \qquad \forall v \in V_l. \tag{2.4}$$

By introducing discrete operators $A_l : V_l \to V_l$ and $f_l : V_l \to \mathbb{R}$ defined by

$$\left(A_l w, v\right)_{L^2(\Omega)} = a(w, v) \qquad \forall w, v \in V_l, \tag{2.5}$$

and

$$\left(f_l, v\right)_{L^2(\Omega)} = b(v) \qquad \forall v \in V_l, \tag{2.6}$$

problem (2.4) can be written as follows: *find* $u_l \in V_l$ such that

$$A_l u_l = f_l. \tag{2.7}$$

Let $C^0(\Omega)$ be the space of continuous functions defined on Ω. The standard nodal interpolant for Lagrangian finite elements $I_l : C^0(\Omega) \to V_l$ is defined as

$$u \mapsto I_l u := \sum_{i\in\mathcal{D}^l} u(x_i)\varphi_l^i. \tag{2.8}$$

The Lagrange interpolant can be employed to derive a transfer operator $\mathcal{P}_{l-1}^l$ from the coarse to the fine grid. In (2.8), φ_l^i denotes the basis function associated to the i-th degree of freedom in the mesh $\mathcal{T}_l$ located at the support point x_i. The residual associated to any finite element function $\tilde{u}_l \in V_l$ is defined as

$$r_l(v) := b(v) - a(\tilde{u}_l, v) = \sum_{i\in\mathcal{D}^l} \left[b(\varphi_l^i) - a(\tilde{u}_l, \varphi_l^i)\right] v(x_i), \tag{2.9}$$

for all $v \in V_l$, $l \in \{1, \dots, L\}$. It is natural to associate a functional $\tilde{r}_l \in V_{l-1}^{'}$ defined by

$$\tilde{r}_l : V_{l-1} \to \mathbb{R}, \qquad \tilde{r}_l(v_{l-1}) := r_l(I_l v_{l-1}). \tag{2.10}$$

It is not difficult to verify that $\tilde{r}_l$ is a linear and bounded functional, the latter thanks to the coercivity assumption on a. For a given $v_{l-1} \in V_{l-1}$, based on employing (2.8), (2.9) (2.10), we have:

$$\begin{aligned} r_l(I_l v_{l-1}) &= b(I_l v_{l-1}) - a(\tilde{u}_l, I_l v_{l-1}) \\ &= \sum_{i \in \mathcal{D}^l} \left[b(\varphi_l^i) - a(\tilde{u}_l, \varphi_l^i) \right] v_{l-1}(x_i) \\ &= \sum_{i \in \mathcal{D}^l} \left(\left[b(\varphi_l^i) - a(\tilde{u}_l, \varphi_l^i) \right] \sum_{x_j \in \mathcal{D}_{l-1}} v_{l-1}(j) \varphi_{l-1}^j(x_i) \right) \\ &= \sum_{j \in \mathcal{D}^{l-1}} \left(\sum_{i \in \mathcal{D}_l} \left[b(\varphi_l^i) - a(\tilde{u}_l, \varphi_l^i) \right] \varphi_{l-1}^j(x_i) \right) v_{l-1}(x_j) \\ &= \sum_{j \in \mathcal{D}^{l-1}} \left(\sum_{i \in \mathcal{D}_l} r_l(\varphi_l^i) \varphi_{l-1}^j(x_i) \right) v_{l-1}(x_j). \end{aligned} \tag{2.11}$$

This implies that the residual vector r_l can be transferred from $\mathcal{T}_l$ to $\mathcal{T}_{l-1}$ with the matrix-vector product

$$r_{l-1} = \left(\mathcal{P}_{l-1}^l\right)^T r_l, \tag{2.12}$$

where

$$\mathcal{P}_{l-1}^l = \begin{bmatrix} \varphi_{l-1}^1(x_1) & \varphi_{l-1}^2(x_1) & \varphi_{l-1}^3(x_1) & \dots & \varphi_{l-1}^{|\mathcal{D}_{l-1}|}(x_1) \\ \varphi_{l-1}^1(x_2) & \varphi_{l-1}^2(x_2) & \varphi_{l-1}^3(x_2) & \dots & \varphi_{l-1}^{|\mathcal{D}_{l-1}|}(x_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \varphi_{l-1}^1(x_{|\mathcal{D}_l|}) & \varphi_{l-1}^2(x_{|\mathcal{D}_l|}) & \varphi_{l-1}^3(x_{|\mathcal{D}_l|}) & \dots & \varphi_{l-1}^{|\mathcal{D}_{l-1}|}(x_{|\mathcal{D}_l|}) \end{bmatrix}.$$

The generic entry of matrix $\mathcal{P}_{l-1}^l$ thus being given by:

$$\left(\mathcal{P}_{l-1}^l\right)_{ij} := \varphi_{l-1}^j(x_i), \tag{2.13}$$

for all integers i, j indexing DoF in $\mathcal{D}^{l-1}$ and $\mathcal{D}^l$, respectively. This corresponds to the evaluation of the j-th basis function located in the coarser triangulation $\mathcal{T}_{l-1}$ on the i-th DoF attached to $\mathcal{T}_l$ and leads in general to a large and sparse rectangular matrix. The restriction operator is naturally defined from (2.12) as the transpose of the prolongation operator defined via (2.13):

$$\mathcal{R}_l^{l-1} := \left(\mathcal{P}_{l-1}^l\right)^T. \tag{2.14}$$

Using nodal Lagrangian basis, if a support point $x_i \notin \operatorname{supp} \varphi_{l-1}^j$, then the corresponding entry evaluates to 0. Clearly, if parent-child relations between consecutive levels are *not* directly available, then ad-hoc search strategies must be performed in order to identity the coarse-grid element $T \in \mathcal{T}_{l-1}$ s.t. $x_i \in T$ for

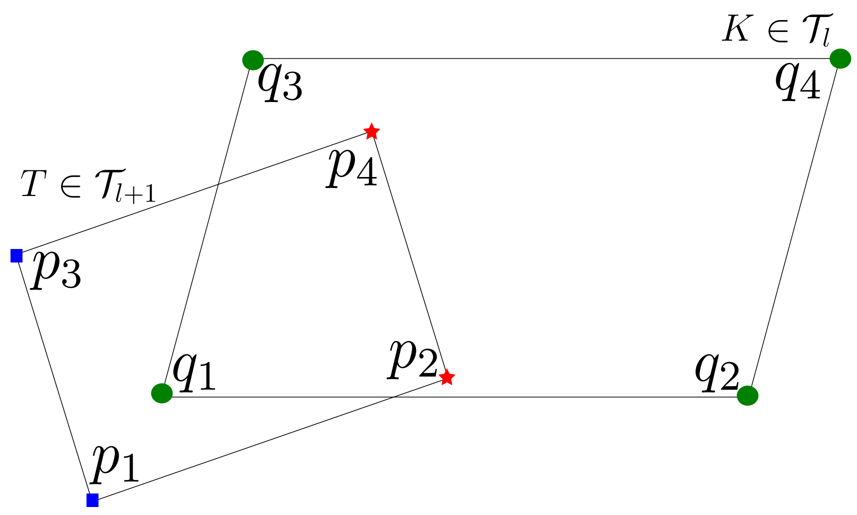


Fig. 2.1 Two overlapping cells coming from consecutive levels. Green dots: DoF associated to a $\mathcal{Q}^1$ element on the coarser cell K. Blue squares and red stars: DoF associated to a $\mathcal{Q}^1$ element on the finer cell T. Red stars correspond to the DoF $(\boldsymbol{p}_2, \boldsymbol{p}_4)$ falling inside $K \in \mathcal{T}_l$, while blue squares are the ones that are falling outside. When evaluating the contribution of this particular coarse cell to the DoF of the fine cell, each basis function related to such DoF $\boldsymbol{q}_i$, $i = 1,\dots,4$, will evaluate on $(\boldsymbol{p}_2, \boldsymbol{p}_4)$ only.

each finer point x_i. This has the effect that the sparsity pattern of $\mathcal{P}^l_{l-1}$ is more involved with respect to the nested case, where only the knowledge of neighbors is needed, and can hence be pre-allocated just by using the connectivity of the mesh. A prototypical scenario is shown in Figure 2.1, where some DoF on the finer cells fall outside the coarser cell. A more in-depth description of the information retrieval between two meshes and of the necessary data structures is given in Section 2.3. Given an initial guess $u_0 \in V_L$ and the numbers of pre-smoothing and post-smoothing steps $(m_1, m_2) \in \mathbb{N} \times \mathbb{N}$, the non-nested multigrid V-cycle algorithm for the approximation of u_L is implemented in a recursive fashion, as summarized in Algorithm 2. Concrete choices for `PreSmoother`, `CoarseGridSolver` and `PostSmoother` usually depend on the application at hand, discussed in Section 2.4. In the present work, we use the multigrid method as a preconditioner within conjugate-gradient solvers, as it is often more robust [173]. As a critical remark, if the true geometry is *complex*, it may not be possible to mesh it with a coarser mesh while preserving the relevant geometrical features. Therefore, the coarsest level may indeed not be really *coarse*, affecting the choice of the `CoarseGridSolver` component. This is particularly evident as soon as the finite element degree becomes moderately high, where plain conjugate-gradient as coarse solver soon becomes the bottleneck in terms of the total time to solution. We will describe a possible way to overcome this issue in Section 2.3.3.

2.2.1 Polynomial global coarsening

With polynomial global coarsening [163, 51, 148], the mesh size h is kept constant on every level of the hierarchy, but the polynomial degree p of the finite element space is reduced incrementally from one level to the other. Among various possibilities, here we choose to decrease the polynomial degree by one from one level to the next, i.e., the degree on the $p^{(c)}$ on the coarse mesh is defined as $p^{(c)} = p^{(f)} - 1$, being $p^{(f)}$ the degree defined on the finer mesh. The main benefit in our framework stems from the observation that whenever the coarsest level has a large number of DoF due to high polynomial degrees p for which CG is suffering as a coarse-grid solver, we can change to conjugate-gradient method preconditioned by polynomial global coarsening. As we assume to work with linear elements at the coarsest level of the polynomial hierarchy, we can switch to AMG which is very competitive in that scenario [130]. Denoting with $\mathcal{T}_1$ the coarsest level over which we apply polynomial multigrid starting with degree p, then the sequence of finite element spaces $\{V_1^l\}_{l=1,\dots,p}$ generated by the polynomial hierarchy is indeed *nested* in the classical sense, as each space is defined on the same triangulation $\mathcal{T}_1$. In particular, highly optimized matrix-free kernels for polynomial transfer with nested levels can be exploited [89, 145]. An illustration of this multigrid methodology and a comparison with the classical hp-multigrid is shown in Figure 2.2. The outcome of this procedure can be regarded as an hp-multigrid scheme where the "h-" component may be non-nested.

Algorithm 2: One iteration of non-nested multigrid V-cycle on level $l \geq 2$ to solve $A_l x = f_l$

```
   Data: $x_0$ initial guess
1  if $l = 1$ then
2      $\delta_1 \leftarrow$ CoarseGridSolver$(A_1, f_1)$;
3  else
4      $x_l \leftarrow$ PreSmoother$(A_l, 0, f_l, m_1)$;
5      $r_l \leftarrow f_l - A_l x_l$;
6      $r_{l-1} \leftarrow$ Restrictor$(r_l)$;
7      $\delta_{l-1} \leftarrow$ V − cycle$(A_{l-1}, r_{l-1}, 0, m_1, m_2)$;
8      $x_l \leftarrow x_l +$ Prolongator$(\delta_{l-1})$;
9      $x_l \leftarrow$ PostSmoother$(A_l, 0, f_l, m_2)$;
```

2.3 Implementation details

Since each level $l \in \{1, \dots, L\}$ covers the whole computational domain as in geometric global coarsening algorithms [32, 123, 31], it is natural to implement the present framework in the already available global coarsening infrastructure present in DEAL.II. The global coarsening infrastructure (`MGTransferGlobalCoarsening`), extensively described in [145] for nested grids, delegates the actual transfer operations to a two-level implementation class (`MGTwoLevelTransfer`). In order to allow a non-nested transfer within the same interface, a new abstract base class (`MGTwoLevelTransferBase`) has been introduced. A UML diagram is displayed in Figure (2.3).

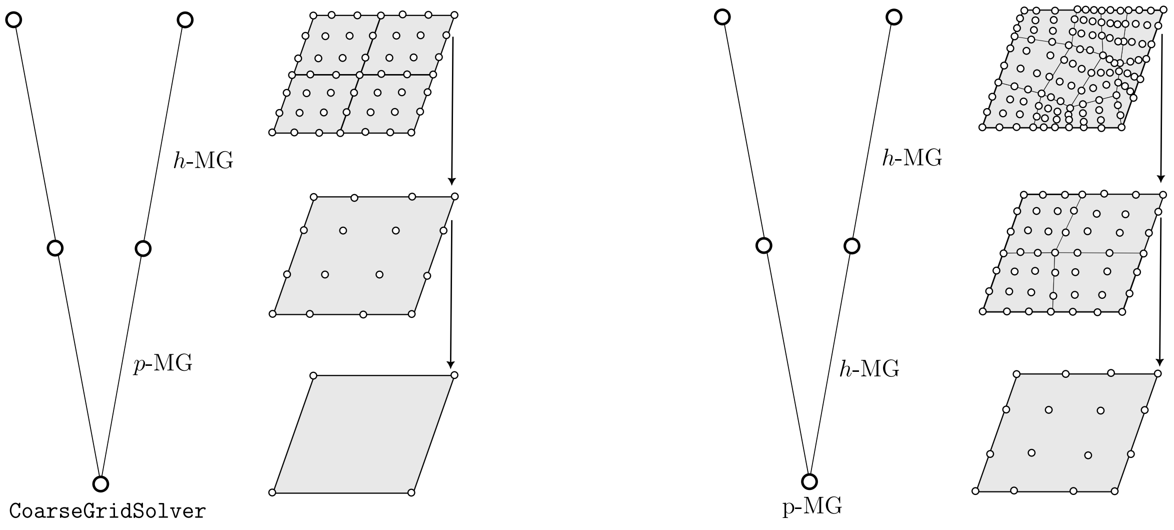


Fig. 2.2 Schematic illustration of *hp*-multigrid scheme for a $\mathcal{Q}^3$ element. DoF corresponding to continuous Lagrangian elements are represented with white dots. Left: Classical nested setting. Right: Non-nested *hp* variant where the hierarchy of levels is *non-matching*. Notice how the `CoarseGridSolver` is set to polynomial multigrid (p-MG).

Among the several components required by a multigrid framework, we detail the two following main ingredients. First, due to the overlapping nature of the two grids, a *geometric search* must be carried out to obtain the mutual relationships between two consecutive levels l and $l+1$, i.e., which cells on level l own the DoF of level $l+1$ and the associated reference positions. In a distributed setting, where all triangulations are partitioned independently (see Figure (2.4)), the relevant owner processes need to be determined by appropriate communication patterns. Second, a matrix-free transfer operator has to evaluate the solution field on arbitrary located points in the reference cell $[0,1]^d$ using the underlying polynomial basis. The same setting is well-established in distributed implementations of immersed methods, and we refer to [127] for an extensive description of this task and details concerning load balancing.

2.3.1 Distributed geometric search

The evaluation of a discrete finite element field on a given list of physical points requires to determine the cell K owning every point $\boldsymbol{x}$ along with its reference coordinates $\hat{\boldsymbol{x}}$. As described before, the major difficulty in the distributed case is that K and $\boldsymbol{x}$ might belong to different processes. Consider a communicator consisting of p processes, a list $\{\boldsymbol{x}_i\}_i$ of points locally owned by process $q \neq p$, and the task to *find cells and corresponding ranks owning each* $\boldsymbol{x}_i$. A first coarse search is used to determine, in a cheap way, the candidate ranks owning each entity. A possible option uses a global communication step where each process shares a rough description of the local portion of the domain it owns with other participants, allowing the global mesh description to be available to each process. Using the MPI standard, this can be achieved via an `MPI_Allgather` with vectors of Axis-Aligned

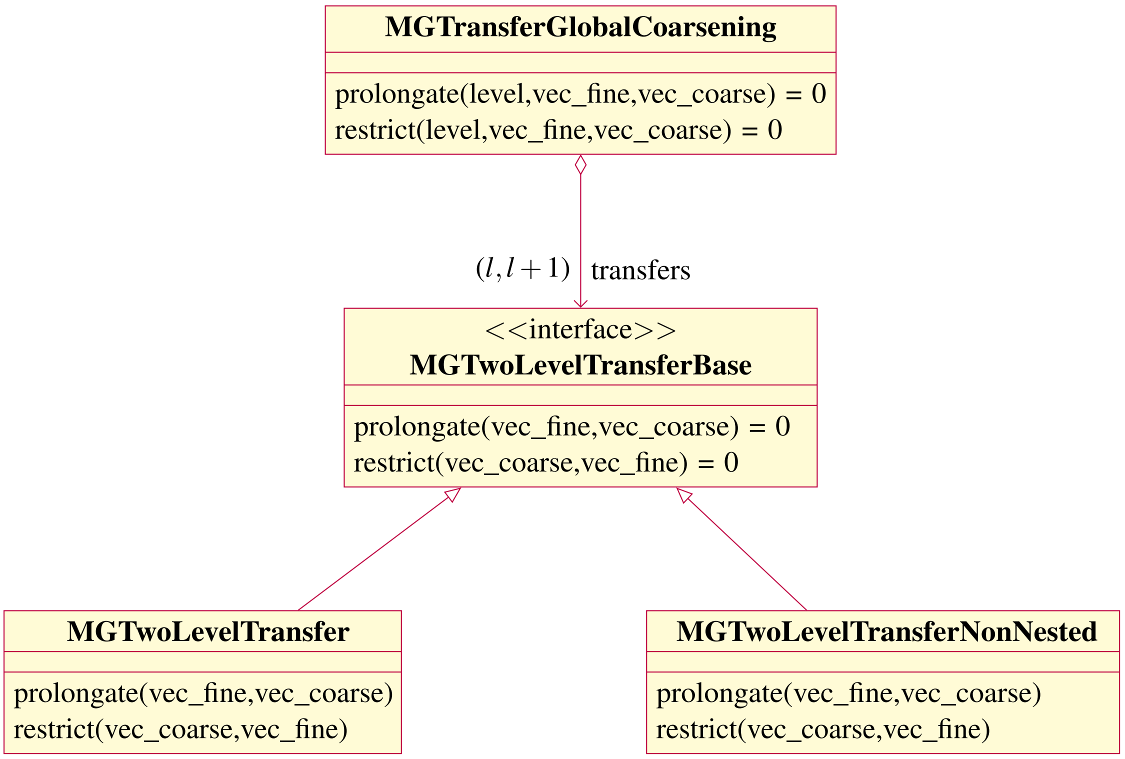
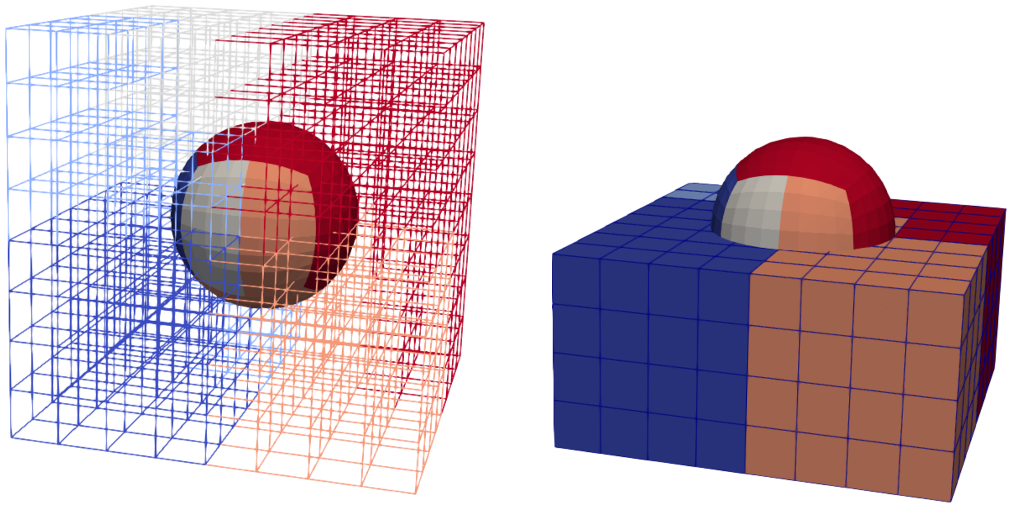

MGTransferGlobalCoarsening
prolongate(level,vec_fine,vec_coarse) = 0
restrict(level,vec_fine,vec_coarse) = 0
$(l,l+1)$
transfers
<<interface>>
MGTwoLevelTransferBase
prolongate(vec_fine,vec_coarse) = 0
restrict(vec_coarse,vec_fine) = 0
MGTwoLevelTransfer
prolongate(vec_fine,vec_coarse)
restrict(vec_coarse,vec_fine)
MGTwoLevelTransferNonNested
prolongate(vec_fine,vec_coarse)
restrict(vec_coarse,vec_fine)

Fig. 2.3 UML diagram of transfer operators available in the `MGTransferGlobalCoarsening` framework in DEAL.II. The new abstract class delegates the implementation of the intergrid transfers to the derived class `MGTwoLevelTransfer` (used in case of nested meshes) or to the new `MGTwoLevelTransferNonNested` in case of a non-nested multigrid method. Each two-level transfer object is specific to consecutive levels l and $l+1$.

Bounding Boxes (AABB) local to each rank. Thanks to a local tree data structure, for $\boldsymbol{x} \in \{\boldsymbol{x}_i\}_i$ one can find possible owning cells and ranks. Once candidates have been determined, a finer search is carried out by each process, checking whether it owns the points or not. The sequence of requests and answers can be realized efficiently by using consensus-based algorithms for dynamic sparse communications [117]. In order to avoid the quadratic complexity of `MPI_Allgather` in the number of ranks, we optionally use a distributed tree search provided by the ARBORX library [135] during the coarse search: each process builds the local tree out of a local vector of AABB. Then, roots of all the local trees are used to create a second tree used for querying possible ownership of entities. Alternatively, other techniques that allow to determine ownership of possibly remote points based on forest-of-trees approaches are possible [55]. Search strategies of this kind are general and can be performed with other *geometric entities* for which suitable simple predicates can be queried. Once all the owners have been determined, reference positions $\{\hat{\boldsymbol{x}}_i\}_i$ associated to each $\{\boldsymbol{x}_i\}_i$ are computed by inversion of the transformation map from real to unit cell. For affine mappings, the inversion can be done explicitly, otherwise Newton-like methods are used.



(a) Ball inside the cube (b) Clip view

Fig. 2.4 Coupling between processors for two overlapped and distributed triangulations (each color represents a different MPI rank). (a) The two partitioned triangulations. The cube is displayed with a wireframe view in order to show the inner ball. (b) Clip view to highlight the coupling between ranks. Notice that grids are discretizing two different geometries only for the purpose of showing the issue. In practice, they will both discretize the same domain Ω.

2.3.2 Efficient evaluation on reference positions

The cell K and reference positions of each point are determined in a setup phase of the multigrid transfer operator. For the actual transfer operation, the remaining work is to evaluate the finite element solution at each point:

$$u_K(\boldsymbol{x}) = \sum_{1\leq i\leq N_K} \varphi_i\left(\hat{\boldsymbol{x}}_K\right) u_{K,i}, \tag{2.15}$$

where N_K is the number of DoF on cell K and φ_i the basis functions in reference coordinates with $\hat{\boldsymbol{x}}_K$ the reference position in cell K for the point $\boldsymbol{x}$. For tensor-product shape functions in $\mathbb{R}^d$, the evaluation of a basis function φ_i at point $\hat{\boldsymbol{x}}$ can be written as follows:

$$\varphi_i(\hat{\boldsymbol{x}}) = \prod_{j=1,\ldots,d} \widehat{\varphi}_{i_j}\left(\hat{\boldsymbol{x}}^j\right), \tag{2.16}$$

being $\varphi_i(\cdot)$ the one-dimensional shape function in direction i and $\hat{\boldsymbol{x}}^i$ the $i-$th coordinate of point $\hat{\boldsymbol{x}}$. Combining Equations (2.15) and (2.16), we get

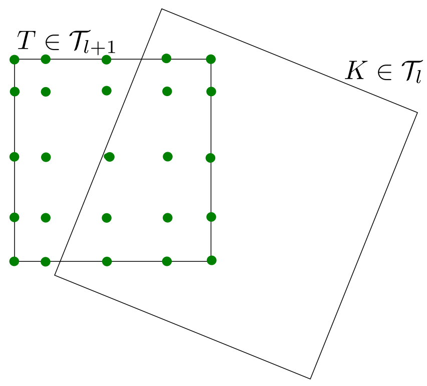
$$u_K(\boldsymbol{x}) = \sum_{1\leq k\leq N_{\text{DoF}}^{\text{1D}}} \widehat{\varphi}_k(\hat{\boldsymbol{x}}^3) \sum_{1\leq j\leq N_{\text{DoF}}^{\text{1D}}} \widehat{\varphi}_j(\hat{\boldsymbol{x}}^2) \sum_{1\leq i\leq N_{\text{DoF}}^{\text{1D}}} \widehat{\varphi}_i(\hat{\boldsymbol{x}}^1) u_{K,ijk}, \tag{2.17}$$

using multi-indices (i,j,k) that are related to the index i in (2.15) in a bijective way. Following the algorithm presented in [131] for a single point with optimizations from [37], the one-dimensional basis functions at $\hat{\boldsymbol{x}}^j$, $j = 1,2,3$, are tabulated. With the tabulated arrays $N_{\boldsymbol{x}}^1, N_{\boldsymbol{x}}^2, N_{\boldsymbol{x}}^3$, one per direction with size $N_{\text{DoF}}^{\text{1D}}$ each, equation (2.17) can be written in tensor-product notation as

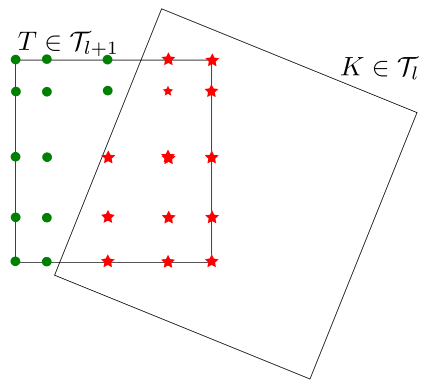
$$u_K(\boldsymbol{x}) = N_{\boldsymbol{z}}^3 \left(I_z \otimes N_{\boldsymbol{x}}^2\right) \left(I_z \otimes I_y \otimes N_{\boldsymbol{x}}^1\right) u_K. \tag{2.18}$$

The resulting computational complexity per point is $\mathcal{O}\left(\left(N_{\text{DoF}}^{\text{1D}}\right)^d\right)$. Note that if the evaluation points within a cell have a tensor-product structure, classical sum-factorization techniques [149, 131] combining evaluation steps for all points on a cell yield a final complexity of $\mathcal{O}\left(dN_{\text{DoF}}^{\text{1D}}\right)$ per degree of freedom. For nested transfers, we use this strategy.

To summarize the computational properties, the present non-nested algorithm has a similar $\mathcal{O}\left(\left(N_{\text{DoF}}^{\text{1D}}\right)^d\right)$ complexity as matrix-based variants for the transfer. The crucial difference to previous methods is the fact that the identified computational complexity of unstructured matrix-free evaluation proposed in the present contribution happens on cached data, with the different arithmetic intensity suggesting an advantage of at least one order of magnitude for contemporary hardware [130, 37].



(a) Gauss-Lobatto points for degree $p = 4$.



(b) Finer Gauss-Lobatto points seen from coarser cell K.

Fig. 2.5 (a) Gauss-Lobatto points for a quadrature rule of order $p = 4$ on a cell $T \in \mathcal{T}_{l+1}$. (b) Evaluation points seen from the coarser cell $K \in \mathcal{T}_l$ (red stars) do not have a tensor-product structure.

2.3.3 Domains with curved boundaries

For simple geometrical shapes such as shells and cylinders, the placement of new vertices upon mesh refinement can be done through polar or cylindrical coordinates. In practice, more complex and realistic shapes are modeled by means of CAD tools, which are internally exploiting B-splines and NURBS to represent most surfaces of interest with high accuracy. In consequence, the external generation by meshing software of a sequence of refined grids that discretize an input complex geometry Ω gives rise to non-conformity on the boundaries of consecutive levels if $\partial\Omega$ is curved.

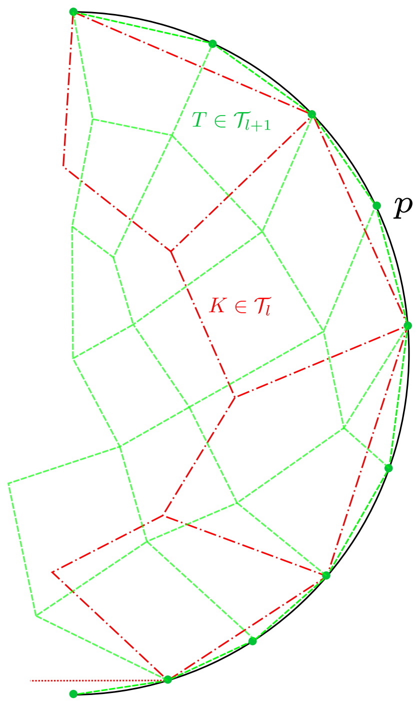
Furthermore, this approach uses the geometrical information only at the pre-processing stage of meshing, not at later stages of the pipeline where it may be required to use high order representations of mappings from real to unit cells. Indeed, even if each level (mesh) has boundary vertices *placed correctly* on $\partial\Omega$, support points defining the shape functions on each boundary face will not lie on the manifold $\partial\Omega$. The non-conformity implies that some entries of the matrix $\mathcal{P}^{l}_{l-1}$, which should be non-zero, will vanish on points on the finer grid if not contained in the mesh involving coarser shape functions, resulting in an inaccurate intergrid transfer. For Dirichlet boundary conditions on a portion $\Gamma^D \subset \partial\Omega$, this does not constitute a problem as DoF on Γ^D are already constrained. Conversely, for Neumann boundaries Γ^N it is necessary to include all unconstrained DoF on Γ^N in order to provide an accurate transfer operator. This situation is illustrated in Figure 2.6. Point $\boldsymbol{p}$ lies on a boundary face of $\mathcal{T}_{l+1}$ and falls outside the coarser level $\mathcal{T}_l$. During the geometric search procedure, by expanding local bounding boxes by a large enough tolerance, $\boldsymbol{p}$ is associated to the red element $K \in \mathcal{T}_l$. Denoting by $\boldsymbol{F}_K : \hat{K} \to K$ the unit-to-real map to cell K, it holds that $\boldsymbol{F}_K^{-1}(\boldsymbol{p}) \notin \overline{\hat{K}}$. Hence, shape functions are evaluated at point $\hat{\boldsymbol{p}} \in \overline{\hat{K}}$ nearest to $\boldsymbol{F}_K^{-1}(\boldsymbol{p})$:

$$\hat{\boldsymbol{p}} := \underset{\hat{\boldsymbol{x}} \in \overline{\hat{K}}}{\operatorname{argmin}}\, d\left(\hat{\boldsymbol{x}}, \boldsymbol{F}_K^{-1}(\boldsymbol{p})\right), \tag{2.19}$$

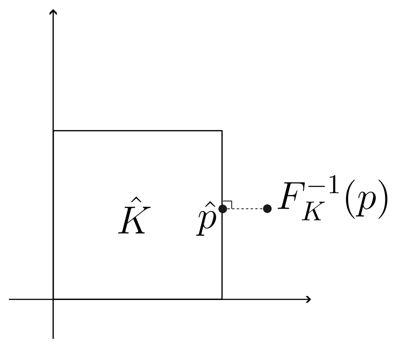
with $d(\cdot,\cdot)$ the Euclidean distance on $\mathbb{R}^d$. Another approach, more robust and requiring fewer heuristics, is to propagate the geometrical information stored inside the CAD into the finite element computations by using state-of-the-art libraries such as OPENCASCADE [161] to interrogate the CAD model during the refinement process and the distribution of evaluation points on boundary faces [112].

2.4 Numerical experiments

This section presents several numerical experiments to verify the robustness of our algorithm with respect to 2D and 3D geometries and polynomial degrees. Some sanity checks aimed to check basic properties of the algorithm in well-known simple cases are shown first before considering actual non-matching levels. Each test solves the Poisson equation with homogeneous Dirichlet boundary conditions and constant right-hand side $f = 1$. Continuous Lagrangian finite elements defined on quadrilaterals or hexahedra with polynomial degree p ranging from 1 to 4 are employed. As quadrature formula, the Gauss-Legendre rule with $(p+1)^d$ points per cell is considered, with $d = 2,3$. As discussed in Section 2.3, each level consists of a sequence of (possibly) distributed and unstructured triangulations meshed independently. In 2D tests, unstructured levels are meshed by using the mesh generator GMSH [96]. Concerning 3D tests, the commercial software COREFORM CUBIT [1] has been adopted in order to generate high quality unstructured hexahedral meshes. Finally, we conclude the section with applications of the proposed methodology using non-trivial geometries stemming from classical Finite Element Analysis (FEA): a complicated geometry is first read from a CAD model and then meshed by an external software. We remark that in practice, a CAD file must



(a) Non-conformity at the boundaries of different levels.



(b) Position of $F_K^{-1}(p)$ with respect to reference cell $\hat{K}$.

Fig. 2.6 Illustration of the situation occurring when the hierarchy discretizes a two-dimensional domain Ω with curved boundaries. (a) Red dash-dotted lines: elements of a coarser triangulation $\mathcal{T}_l$. Green dashed lines: boundary for a finer triangulation $\mathcal{T}_{l+1}$. Black solid line: exact representation of the boundary $\partial\Omega$. (b) Pre-image of point $\boldsymbol{p}$ through F_K and its nearest point projection $\hat{\boldsymbol{p}} \in \overline{\hat{K}}$ defined according to (2.19).

first be repaired in order to be meshed. Removing the so-called small features from a CAD file is necessary in order to mesh the model at hand and in general is a non-trivial operation. After this procedure is completed, specific mesh-related parameters have to be carefully tuned, depending on the physical situation at hand. The grids generated through this procedure are inherently *non-nested*. The iterative solver is configured in the following way, taken from [145], with the exception that in the present chapter a *non-nested* multigrid preconditioner is employed in place of a nested one:

- The conjugate-gradient solver is run until a reduction of the l_2-norm of the unpreconditioned residual by 10^4 is reached. Such a loose tolerance is typical in many multigrid applications, for instance in time-dependent problems in computational fluid dynamics, for which good initial guesses are usually available by employing extrapolation or projection.

- The conjugate-gradient solver is preconditioned by a single V-cycle of the *non-nested* multigrid method.

l	$p=1$				$p=2$				$p=3$				$p=4$			
	#*i*	#*i_n*	#*i_a*	#DoF	#*i*	#*i_n*	#*i_a*	#DoF	#*i*	#*i_n*	#*i_a*	DoF	#*i*	#*i_n*	#*i_a*	#DoF
2	3	3	1	9	3	3	3	25	3	3	4	49	3	3	6	81
3	3	3	3	25	3	3	5	81	3	3	8	169	3	3	12	289
4	3	3	4	81	3	3	7	289	3	3	10	625	3	3	14	1 089
5	3	3	6	289	3	3	8	1 089	3	3	14	2 401	3	3	20	4 225

Table 2.1 #*i*: number of iterations for non-nested multigrid in two space dimensions. #*i_n*: number of iterations for the nested version. #*i_a*: number of iterations required by AMG.

- Operations in the V-cycle are run with single precision floating-point numbers, while conjugate-gradient is run in double precision, which increases the computational throughput while maintaining acceptable accuracy [131, 129].
- A point Jacobi smoother employed within a Chebyshev iteration of degree 3 is used on every level, using eigenvalue estimates computed with 12 iterations of the Lanczos iteration.
- Two V-cycles of the TRILINOS ML implementation of AMG [95] are used, in double precision, as coarse-grid solver. The used parameters are the same as the ones in Appendix C of [145].

In every test, we report the number of DoF attached to each level for different polynomial degrees. All the experiments have been carried out on compute nodes with Intel Xeon Skylake, 2.7 GHz, and the AVX-512 ISA extension so that 8 doubles or 16 floats can be processed per instruction.

The wall-clock times (measured in seconds) shown in the upcoming Tables consist of the actual solve time needed by the conjugate-gradient iteration. All results shown in this chapter, along with the necessary meshes and CAD files, are available on a maintained GitHub repository https://github.com/peterrum/dealii-multigrid.

2.4.1 Application to nested meshes

The present test is only meant to assess the consistency of the algorithm. A sequence of structured and globally refined nested meshes discretizing $[-1,1]^2$ is considered. Since $V_{l-1} \subset V_l$, the transfer operator $\mathcal{P}_{l-1}^{l}$ coincides with the classical injection from V_{l-1} to V_l. As a matter of fact, the numerical results obtained both with the nested and the non-nested implementation are identical, as reported in Table 2.1. The same sanity check is repeated for a sequence of nested levels discretizing the unit cube $[-1,1]^3$. Results are reported in Table 2.2.

l	$p=1$				$p=2$				$p=3$				$p=4$			
	#i	*#i_n*	*#i_a*	#DoF	*#i*	*#i_n*	*#i_a*	#DoF	*#i*	*#i_n*	*#i_a*	DoF	*#i*	*#i_n*	*#i_a*	#DoF
2	3	3	1	27	3	3	3	125	3	3	10	343	3	3	20	729
3	3	3	4	125	3	3	10	729	3	3	20	2 197	3	3	33	4 913
4	3	3	5	729	3	3	12	4 913	3	3	31	15 625	3	3	47	35 937
5	3	3	6	4 913	3	3	16	35 937	3	3	73	117 649	3	3	100	274 625

Table 2.2 *#i*: number of iterations for non-nested multigrid in three space dimensions. *#i_n*: number of iterations for the nested version. *#i_a*: number of iterations required by AMG.

2.4.2 L-shaped domains

2D unstructured L-shaped domains

A sequence of unstructured meshes of an L-shaped domain is constructed using GMSH [96]. The right-hand side and boundary conditions force a singular behavior at the re-entrant corner. For this reason, several refinements have been applied near the corner during the generation of the levels. The number of iterations required by our multigrid preconditioner and the number of DoF on the finest level are reported in Table 2.3. For every level and polynomial degree, an almost constant number of iterations required by the solver is observed. A set of the first four levels is shown in Figure 2.7. To highlight the flexibility of the present framework, we repeat the same test with the exception that the coarsest level is a classical structured L-shaped domain with 192 cells, leaving all the other levels unchanged. The number of required iterations follows the one in the fully unstructured case (see the first row of Table 2.4).

l	$p=1$		$p=2$		$p=3$		$p=4$	
	#i	#DoF	*#i*	#DoF	*#i*	#DoF	*#i*	#DoF
2	4	1 691	6	6 613	9	14 767	12	26 153
3	4	2 988	6	11 757	9	26 308	12	46 641
4	4	5 055	6	19 965	9	44 731	12	79 353
5	4	8 735	6	34 605	9	77 611	12	137 753
6	4	14 675	6	58 261	9	130 759	11	232 169
7	4	24 308	6	96 661	9	217 060	12	385 505

Table 2.3 Number of required iterations and DoF per level to solve the system with the L-shaped domain.

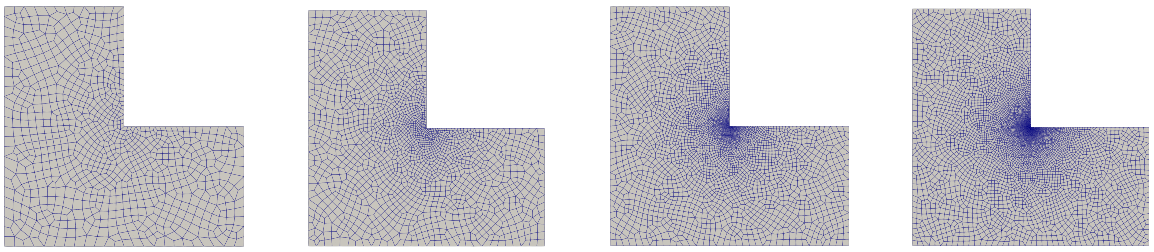


Fig. 2.7 Four of the levels employed for the L-shaped domain, refined near the re-entrant corner.

l	$p=1$		$p=2$		$p=3$		$p=4$	
	#i	#DoF	#i	#DoF	#i	#DoF	#i	#DoF
2	5	1 691	6	6 613	7	14 767	11	26 153
3	4	2 988	5	11 757	7	26 308	11	46 641
4	4	5 055	5	19 965	6	44 731	11	79 353
5	4	8 735	5	34 605	6	77 611	11	137 753
6	4	14 675	5	58 261	5	130 759	10	232 169
7	4	24 308	5	96 661	6	217 060	11	385 505

Table 2.4 Number of required iterations and DoF per level to solve the system with the L-shaped domain and a coarser structured level with 192 elements as first level for each polynomial degree.

Fichera's corner

Fichera's corner is the prototype of a domain where edge and corner singularities interact and represents the three-dimensional extension of the two-dimensional L-shaped test shown in 2.4.2. Levels are generated with COREFORM CUBIT starting from a coarse, structured three-dimensional L-shaped domain, refining around the vertex located at the re-entrant corner by increasing the `element_depth` parameter (how many elements away from the specified vertex are refined) from 2 to 5. Three consecutive levels of the hierarchy are shown in Figure 2.8. Similarly to the two-dimensional case, we observe iterations counts which are in practice independent of the number of levels employed.

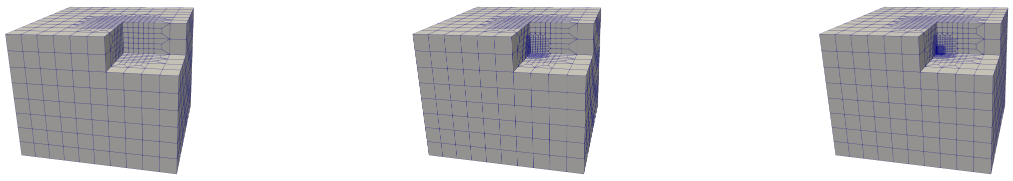


Fig. 2.8 Consecutive levels employed for the Fichera test, refined near the re-entrant corner.

l	$p=1$		$p=2$		$p=3$		$p=4$	
	#i	#DoF	#i	#DoF	#i	#DoF	#i	#DoF
2	4	3 983	5	30 105	7	99 775	9	234 401
3	4	11 579	4	90 171	7	301 789	9	712 445
4	3	27 859	4	219 283	6	736 381	8	1 741 261
5	3	57 719	4	456 513	6	1 535 311	8	3 633 041
6	4	219 283	4	1 741 261	7	5 862 799	9	13 880 761

Table 2.5 Number of required iterations and DoF per level to solve the system with the Fichera's corner test.

2.4.3 Applications to FEA

On a given linear elastic body $\Omega \subset \mathbb{R}^3$, with boundary $\partial\Omega$ partitioned in $\partial\Omega_D$ and $\partial\Omega_N$, we solve, under the assumption of infinitesimal strains, the linear elasticity equation for compressible materials. The governing equations for the unknown displacement $\boldsymbol{u} : \Omega \to \mathbb{R}^3$ are([104]):

$$\begin{cases} -\nabla\cdot\sigma(\boldsymbol{u}) & = \boldsymbol{f} \qquad \text{in } \Omega, \\ \sigma(\boldsymbol{u}) & = \lambda\,\mathrm{tr}(\varepsilon(\boldsymbol{u}))I + 2\mu\varepsilon(\boldsymbol{u}), \\ \varepsilon(\boldsymbol{u}) & = \frac{1}{2}\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^\top\right), \\ \boldsymbol{u} & = \boldsymbol{0} \qquad \text{on } \partial\Omega_D, \\ \sigma(\boldsymbol{u})\cdot\boldsymbol{n} & = \boldsymbol{g} \qquad \text{on } \partial\Omega_N, \end{cases} \tag{2.20}$$

where λ, μ are the Lamé coefficients for the material, $\boldsymbol{\sigma}$ is the stress tensor, I the identity tensor, $\varepsilon(\boldsymbol{u})$ the linearized strain rate tensor and $\boldsymbol{f} : \Omega \to \mathbb{R}^3$ is the body force exerted per unit volume. The boundary $\partial\Omega_D$ is clamped, whereas the normal load $\boldsymbol{g}$ is imposed on $\partial\Omega_N$. A natural space for the kinematically admissible displacement field u and related test functions is

$$V_D := \{\boldsymbol{v} \in [H^1(\Omega)]^3 : \boldsymbol{v}|_{\partial\Omega_D} = \boldsymbol{0}\},$$

the set of vector-valued H^1 functions in Ω with zero trace (displacement) on the Dirichlet portion of the boundary $\partial\Omega_D$. The variational formulation of (2.20) requires to find $\boldsymbol{u} \in V_D$ such that

$$a(\boldsymbol{u},\boldsymbol{v}) = b(\boldsymbol{v}) \qquad \forall\boldsymbol{v} \in V_D,$$

where $a(\boldsymbol{u},\boldsymbol{v}) = \big(\sigma(\boldsymbol{u}),\varepsilon(\boldsymbol{v})\big)_\Omega$ and $b(\boldsymbol{v}) = \big(f,\boldsymbol{v}\big)_\Omega + \langle\boldsymbol{g},\boldsymbol{v}\rangle_{\partial\Omega_N}$. Well-posedness for this problem follows from Korn inequalities and the Lax-Milgram lemma [82]. The material considered in the forthcoming examples is plain steel with Young's modulus $E = 205$ GPa and Poisson's ratio $\nu = 0.3$.

Given E and ν, Lamè constants can be computed as follows:

$$\lambda = \frac{E\nu}{(1-2\nu)(1+\nu)}, \tag{2.21}$$

$$\mu = \frac{E}{2(1+\nu)}. \tag{2.22}$$

It is well known that low-order elements ($p = 1,2$) applied to (2.20) with mixed boundary conditions may suffer from the so-called *locking* phenomenon when $\lambda \to \frac{1}{2}$, meaning that the material is nearly incompressible. Among the possible remedies, a simple and generally accepted rule in engineering is that using higher order conforming elements can reduce the potential for locking (even though there are counterexamples, cf. [5]). On the other hand, matrix-vector multiplications for higher order matrix-based discretization rapidly become quite expensive as the bandwidth of the matrices increases [132, 72]. Other strategies that can prevent locking consist in mixed or augmented formulations [41], but higher order polynomials are rarely employed in practice, usually owing to some low regularity arguments. The forthcoming examples consider domains arising from applications in structural analysis that are widely studied in the literature. The geometry is described through `.step` files, a standardized ISO file format used in CAD design. Models have been first repaired and later meshed using COREFORM CUBIT. On both the considered domains, we solve the elasticity equations (2.20) varying the polynomial degree p, the number of levels in the hierarchy, and the number of processes. We show in Table 2.6 the chosen solvers depending on the degree of the finite element space. For $\mathcal{Q}^1$ elements we compare with AMG, a setting very competitive for linear elements. For higher orders, we compare with polynomial multigrid (PMG) preconditioning, using CG preconditioned by AMG as coarse grid solver since the coarsest mesh of the polynomial hierarchy is made by linear elements.

Solvers for problem (2.20)			
Element \ Solver	AMG	NN	PMG
$\mathcal{Q}^1$	✓	✓	✗
$\mathcal{Q}^2$	✓	✓	✓
$\mathcal{Q}^3$	✗	✓	✓
$\mathcal{Q}^4$	✗	✓	✓

Table 2.6 In each row we report the polynomial space and the solvers applied to the FEA problem (2.20). Legend: AMG (Algebraic multigrid), NN (Non-nested multigrid), PMG (Polynomial multigrid).

In order to successfully apply AMG to vector-valued problems it is necessary to provide the nullspace of the weak form $\tilde{a}(\cdot,\cdot)$ associated to the problem with free boundary conditions on $\partial\Omega$, defined as:

$$\ker(\tilde{a}) := \{\boldsymbol{v} \in H^1(\Omega) : \tilde{a}(\boldsymbol{v},\boldsymbol{v}) = 0\}. \tag{2.23}$$

In case of the three-dimensional linear elasticity problem with strain tensor ε and it holds [26]:

$$\ker(\tilde{a}) = \ker(\varepsilon) = \operatorname{span}\{\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3, \boldsymbol{r}_4, \boldsymbol{r}_5, \boldsymbol{r}_6\}, \tag{2.24}$$

with

$$\boldsymbol{t}_1 = \boldsymbol{e}_1, \boldsymbol{t}_2 = \boldsymbol{e}_2, \boldsymbol{t}_3 = \boldsymbol{e}_3, \boldsymbol{r}_4 = \begin{bmatrix} 0 \\ z \\ -y \end{bmatrix}, \boldsymbol{r}_5 = \begin{bmatrix} -z \\ 0 \\ x \end{bmatrix}, \boldsymbol{r}_6 = \begin{bmatrix} y \\ -x \\ 0 \end{bmatrix}. \tag{2.25}$$

The first three elements correspond to translation modes and can be represented by the canonical basis $\{\boldsymbol{e}_i\}_i$ of $\mathbb{R}^3$, while $\{\boldsymbol{r}_i(\boldsymbol{x})\}_i$ are the rotational modes describing rotations around coordinate axes. It should be noted that, from a user's perspective, the human time required for preparing and exporting all levels might certainly be non-negligible in practice even for experts in mesh generation. On the other hand, classical AMG implementations such as the one used in this thesis usually require only the setup of a list of parameters related to the problem at hand.

Elasticity examples: piston

The domain depicted in Figure 2.10(a) describes a flat head piston. We set $\boldsymbol{f} = \boldsymbol{0}\ \mathrm{N/m^3}$, neglecting external body forces, and we prescribe zero displacement on the surfaces of pin hole and a normal load $\boldsymbol{g} = -10^5 \boldsymbol{n}$ Pa on the head, being $\boldsymbol{n}$ the outward unit normal. The rest of the boundary is traction-free. Table 2.8 lists the approximate mesh size per level h_l set inside the mesh generator and the resulting number of DoF obtained using continuous Lagrangian finite elements for each $l \in \{1, \dots, 4\}$ while varying the polynomial degree p from 1 to 4. Each generated grid constitutes a level used in the V-cycle. Figure 2.10(b) shows a zoom to highlight the non-nestedness of two resulting levels obtained by the aforementioned process. A visualization of the magnitude of the displacement $\boldsymbol{u}$ is given in Figure 2.11. Table 2.7 shows time to solution and number of iterations for polynomial degrees k ranging from 1 to 4 and different number of levels, with 72 processes. Some general observations can be made. First, the number of iterations for the non-nested multigrid method is roughly constant for every degree p, indicating a correct implementation of the proposed methodology. Moreover, for a fixed number of levels l, we observe similar patterns in the time to solution. Figure 2.9 shows the times to solution for the combination of solvers reported in Table 2.6 when $l = 3$. For $\mathcal{Q}^1$ elements, AMG shows higher number of iterations compared to the non-nested multigrid, but with overall better time to solution. When employing $\mathcal{Q}^2$ elements, AMG shows times that are always higher than the ones observed with the non-nested multigrid, while PMG is clearly comparable to the non-nested approach. Moving to the $\mathcal{Q}^3$ case, we note again almost overlapping results between PMG and the non-nested procedure. The gap between non-nested and PMG starts increasing for higher order elements such as $\mathcal{Q}^4$. This is expected as the prolongation and restriction operators for polynomial global coarsening

72 processes																		
	$p=1$				$p=2$						$p=3$				$p=4$			
l	AMG		NN		AMG		NN		PMG		NN		PMG		NN		PMG	
	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
2	16	0.0263	6	0.2364	22	0.3971	8	0.2204	8	0.1854	9	0.4266	12	0.3093	11	0.9177	9	0.2888
3	17	0.0340	6	0.2298	22	0.8033	7	0.3040	8	0.2682	8	0.5202	11	0.4221	11	1.3653	10	0.5512
4	12	0.0519	6	0.2323	20	1.0428	7	0.3100	8	0.4258	9	0.7887	12	0.7198	11	1.9711	10	0.8504

Table 2.7 Number of iterations and time to solution for algebraic multigrid (AMG), polynomial multigrid (PMG), non-nested Multigrid (NN) applied to the piston test case with different polynomial degrees from $p = 1$ to $p = 4$. AMG times are shown for $\mathcal{Q}^1$ and $\mathcal{Q}^2$ elements only.

are employing matrix-free kernels based on sum-factorization algorithms for classical nodal FEM spaces, which generally have favorable complexity for higher-order elements.

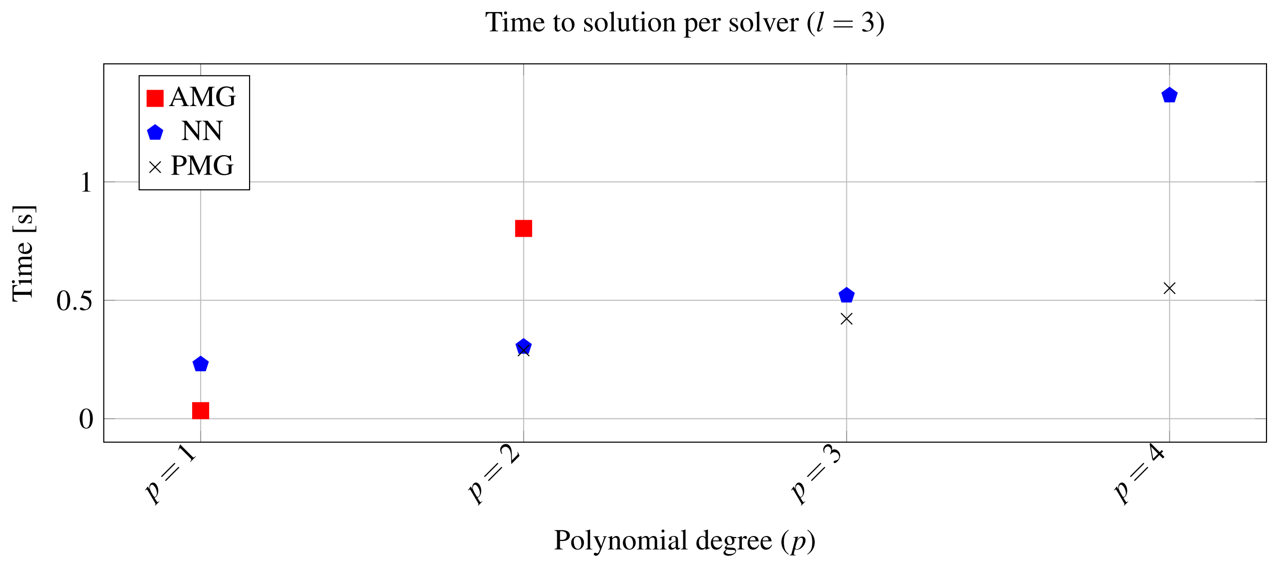
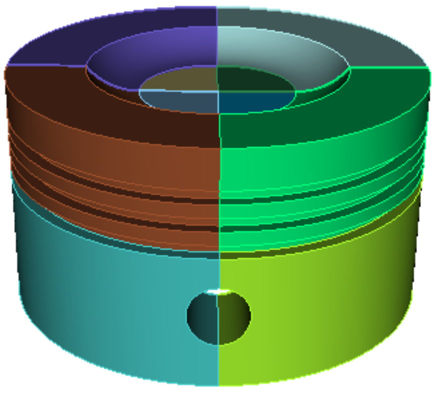


Fig. 2.9 Scatter plot with times required by piston test case with $l = 3$ while varying solver and polynomial degree p.

DoF per mesh size and polynomial degree p				
h_l \ p	1	2	3	4
0.25	25 005	171 765	548 367	1 262 901
0.19	50 814	360 843	1 167 090	2 706 561
0.15	110 904	804 399	2 622 462	6 107 073
0.123	165 069	1 208 187	3 952 203	9 219 969

Table 2.8 Total number of DoF for polynomial degree p and mesh-size for the piston test.


(a) CAD model for the piston.

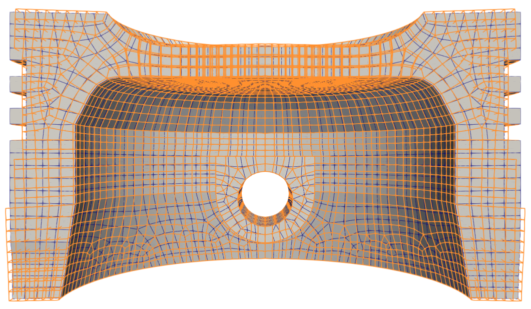
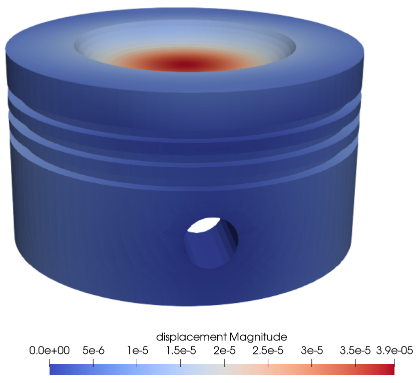

(b) Two non-nested levels for the piston (clipped, reference configuration).

Fig. 2.10 Left: CAD model used in the preprocessing procedure. Right: Two different levels. For the sake of visualization, the finer level is displayed in orange using a wireframe representation. The coarser level is represented as a volumetric mesh (edges in blue). Notice that each level has not been generated on top of the coarser one through a global or local refinement process.



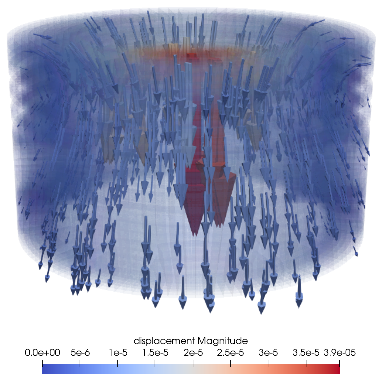


Fig. 2.11 Left: Magnitude of the displacement vector $\boldsymbol{u}$ for the piston test case. Right: Scaled view of the vector field u.

DoF per mesh size and polynomial degree p				
h_l \ p	1	2	3	4
1.8	25 425	171 138	542 691	1 245 636
1.5	48 366	336 822	1 082 052	2 500 740
1.1	108 492	777 894	2 526 372	5 872 092
0.95	175 020	1 275 312	4 166 334	9 713 544

Table 2.9 Total number of DoF for polynomial degree p and mesh-size for the wrench test.

Elasticity examples: wrench

As a second example we consider the static structural analysis of a wrench, a classical benchmark problem in FEA. Zero displacement is imposed on one head, while a pressure of $\boldsymbol{g} = -10^5\boldsymbol{n}$ Pa is applied on one top of the other head, acting in onward direction. The rest of the boundary is traction-free. In Table 2.10 we show time to solution along with the number of required iterations by the outer solver, while Table 2.9 shows the number of degrees of freedom per each level l. The magnitude of the displacement field $\boldsymbol{u}$, along with a graphical representation of the displacement vector field is displayed in Figure 2.12. The same observations made for the piston test readily apply also for this test. In particular, Figure 2.13 shows the same behavior in terms of time to solution and polynomial degrees as the one exhibited by the piston test case in Figure 2.9. In this case, we notice a high number of iterations for AMG with $\mathcal{Q}^1$ and $\mathcal{Q}^2$ elements, while geometric approaches keep low and constant iteration counts for every polynomial degree p, resulting in overall better performances.

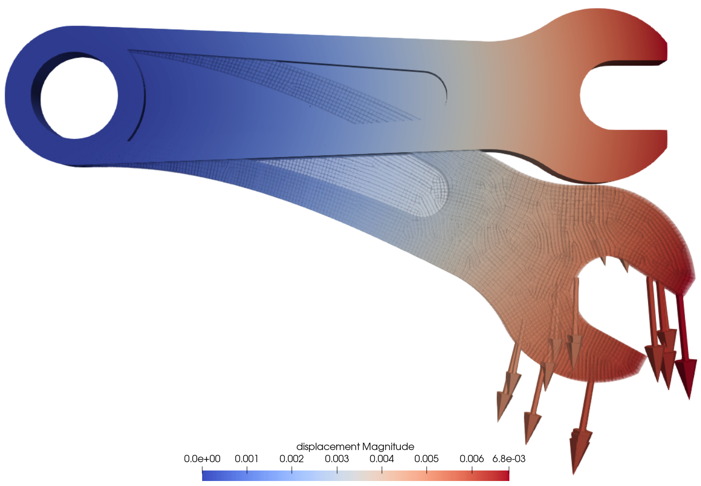


Fig. 2.12 Magnitude (scaled) of the displacement $\boldsymbol{u}$ for the wrench test case.

	72 processes																	
	$p=1$				$p=2$						$p=3$				$p=4$			
l	AMG		NN		AMG		NN		PMG		NN		PMG		NN		PMG	
	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
2	35	0.0444	7	0.1406	36	0.5099	9	0.2047	7	0.1374	13	0.5155	11	0.2070	16	1.1557	9	0.2451
3	35	0.0640	8	0.1748	44	1.5859	10	0.2923	7	0.4400	14	0.7704	10	0.6029	16	1.7310	8	0.6973
4	37	0.1212	8	0.1875	47	2.7108	10	0.3669	7	0.6863	13	1.0150	10	1.0249	17	2.8615	8	1.1035

Table 2.10 Number of iterations and time to solution for algebraic multigrid (AMG), polynomial multigrid (PMG), non-nested Multigrid (NN) applied to the wrench test case with different polynomial degrees from $p = 1$ to $p = 4$. AMG times are shown for $\mathcal{Q}^1$ and $\mathcal{Q}^2$ elements only.

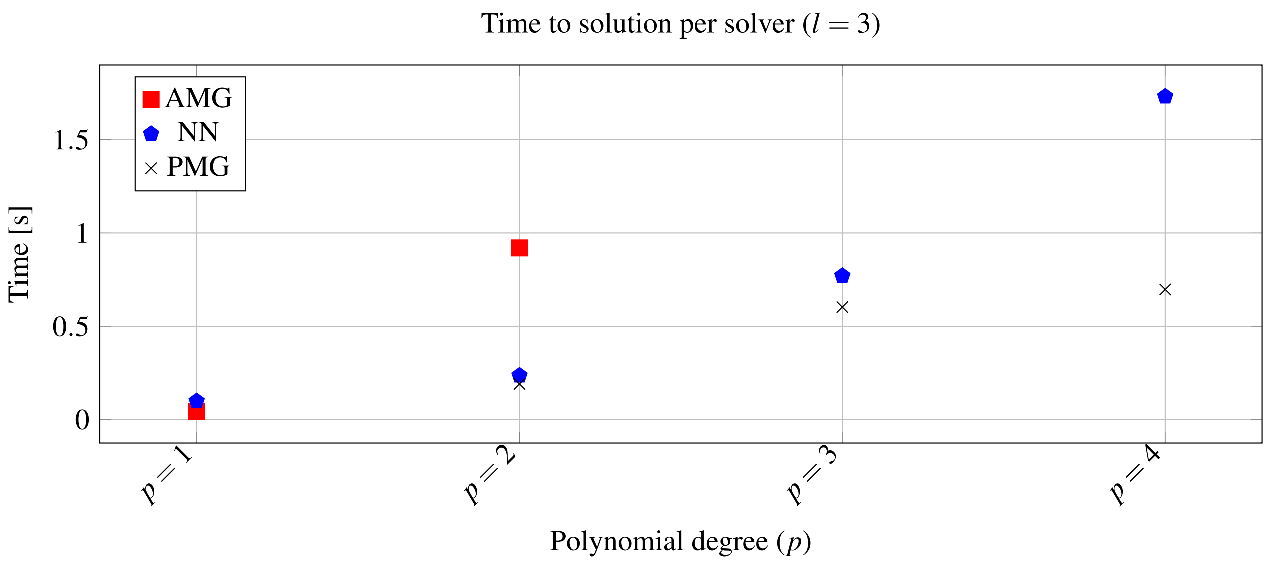


Fig. 2.13 Scatter plot with times required by wrench test case for $l = 3$ while varying solver and polynomial degree p.

2.5 Performance evaluation

We investigate the performance of our implementation by means of the 3D Poisson problem detailed in Section 2.4. In the next tests, we compare the time to solution and the breakdown of times spent on each multigrid level by global coarsening and non-nested multigrid for different geometries for which both methods can be applied. As a crucial remark, we stress that we are measuring against the highly optimized base-line of matrix-free methods described in [145]. The hierarchies considered are:

- **cube**: refine globally l times the cube $[-1,1]^3$,
- **ball**: refine globally l times $\mathcal{B}_1(\mathbf{0})$, the ball of radius 1 centered at the origin.

	1 process				12 processes			
l	GC		NN		GC		NN	
	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
4	4	2.3×10^{-2}	4	4.4×10^{-2}	4	1.4×10^{-2}	4	2.0×10^{-2}
5	4	2.0×10^{-1}	4	3.8×10^{-1}	4	3.5×10^{-2}	4	6.7×10^{-2}
6	4	1.6e+0	4	3.2e+0	4	2.0×10^{-1}	4	3.8×10^{-1}
7	4	1.2e+1	4	2.5e+1	4	1.5e+0	4	2.9e+0

Table 2.11 Number of iterations and Time to solution for Global Coarsening (GC) and Non-Nested Multigrid (NN) applied to the `cube` test case with polynomial degree $p = 4$.

	1 process				12 processes			
l	GC		NN		GC		NN	
	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]	$\#i$	t[s]
4	5	3.3e-1	6	6.4e-1	5	6.0e-2	6	1.1e-1
5	6	3.2e+0	6	5.3e+0	6	4.3e-1	6	6.5e-1
6	6	2.5e+1	6	4.3e+1	6	3.4e+0	6	4.9e+0

Table 2.12 Number of iterations and Time to solution for Global Coarsening (GC) and Non-Nested Multigrid (NN) applied to the `ball` test case with polynomial degree $p = 4$.

2.5.1 Serial and parallel runs

The nestedness of levels for these tests implies that the transfer operator $\mathcal{P}_{l-1}^{l}$ for the multigrid method described in 2.2 coincides with the canonical injection. While the number of iterations is the same for both variants, the transfers $\mathcal{P}_{l-1}^{l}$ and $\mathcal{R}_{l}^{l-1}$ are expected to be much more expensive if compared to classical transfer operators tailored for nested hierarchies. This is confirmed by Figure 2.14 for a serial run with the `cube` and `ball` examples, where a factor of roughly 20 for transfers is observed between the two multigrid variants, whereas the other ingredients are in practice identical. As shown in Tables 2.11 and 2.12, this results in times to solution that are slower by a factor of 2 in favor of global coarsening in the serial case. It is classical in multigrid literature to assume that pre-/post-smoothing steps consume most of the run time, while transfers between levels are in general not a bottleneck. Figures 2.15 and 2.17 show a profile of the V-cycle for a parallel run with 12 processes, where we show the minimum and maximum time (in seconds) spent on each component of the algorithm. The figures illustrate that, for the non-nested approach, transfers are more expensive than the classical transfer with sum factorization used by global coarsening with a run time comparable to the application of the smoother.

To better understand what is contributing to these higher values, a further analysis of the cost of $\mathcal{P}_{l-1}^{l}$ has to be carried out. Besides the total time spent to interpolate the coarse finite element field δ^{l-1} from $\mathcal{T}_{l-1}$ on $\mathcal{T}_l$, we also consider the time spent internally in evaluating δ^l at arbitrarily located reference points determined after the search procedure explained in Sec. 2.3.1. Even if in these particular instances levels are nested and hence points are not truly arbitrary, this is transparent to the algorithmic realization of the method and allows to have an indication about how much time is spent in sending messages compared to the effective evaluation cost. Figure 2.18 shows that evaluation is the most expensive component of prolongation kernels, taking roughly between 60% and 70% of the total time for the present setup and geometries, whereas the remaining part is related to communication. In Figure 2.19 we show the minimum and maximum time spent on each level for the `ball` test case for a parallel run with 12 processes for both approaches. With a similar distribution between levels, the non-nested variant exhibits higher computational times overall because of the cost of the intergrid transfers. The figure also confirms that the good workbalance for the time to solution typical of the global coarsening algorithms is seamlessly inherited by the non-nested approach. Increasing the degree of the Chebyshev smoother in order to reduce the impact of the level transfer by possibly reducing the total number of conjugate gradient iterations, as shown in Figure 2.16, is shown to not improve the overall time to solution.

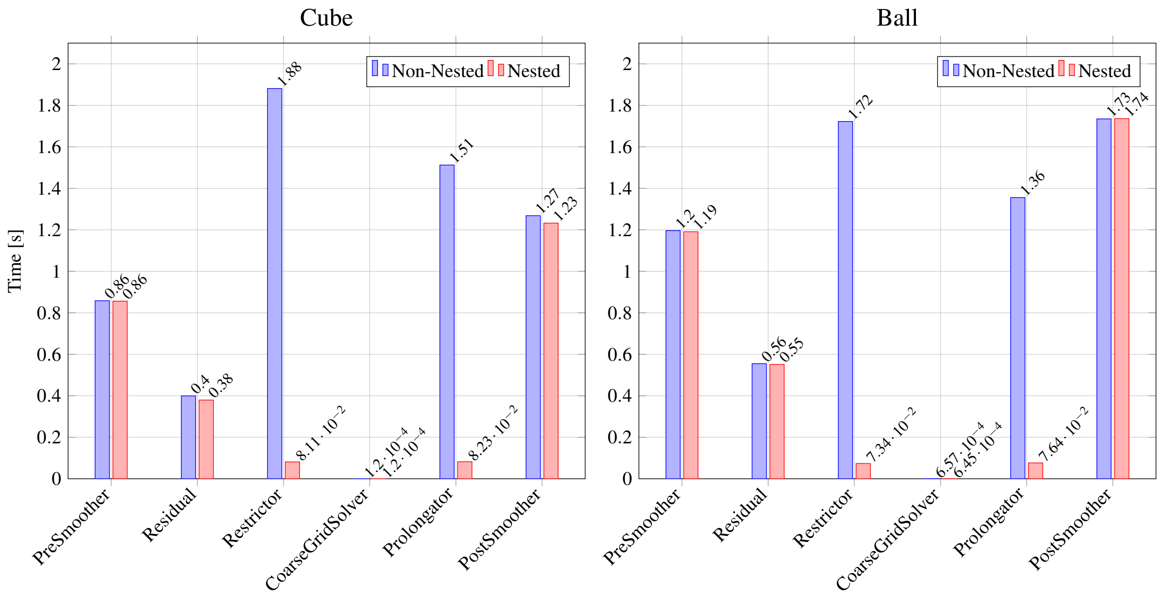


Fig. 2.14 Serial profiles of a V-cycle. Left: `Cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$.

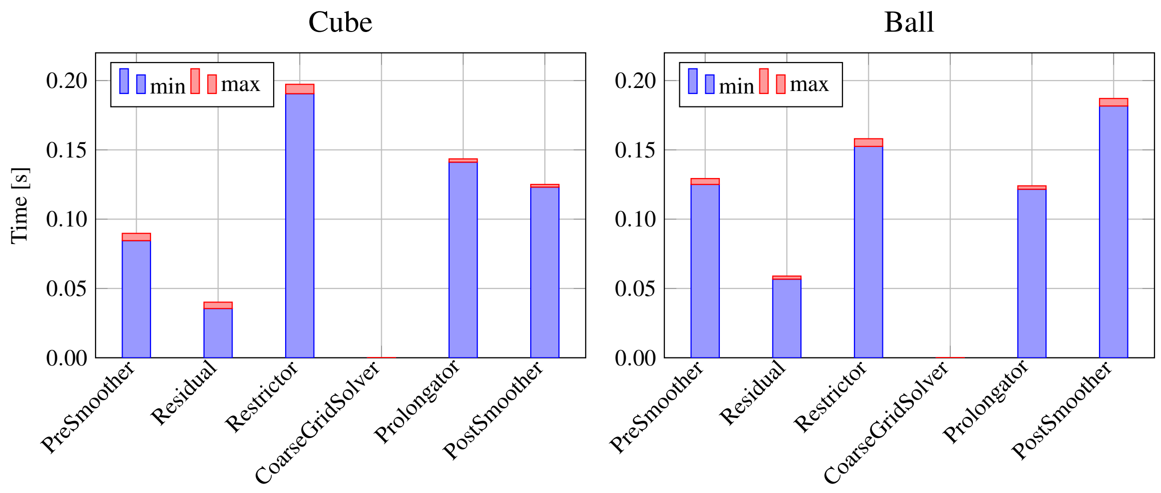


Fig. 2.15 Profile of a V-cycle with 12 processes with non-nested multigrid. Left: `Cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$.

2.5.2 Non-nested tests

In this subsection, we analyze the two non-nested tests presented in Section 2.4.2 more in depth. An analysis of the different components required by the non-nested multigrid method is presented. We adapt the setup and tests described in [145, 65]. Almost every metric adopted therein and in classical multigrid literature can be immediately reused, except for the *vertical communication efficiency*, i.e., the share of fine cells that are owned by the same process that owns their parent coarse cell. That metric quantifies the amount of data that has to be exchanged by the participating processes during the level transfer. The lack of exact overlapping between distributed levels slightly complicates standard metrics that are derived solely on geometrical information, usually considered in standard multigrid literature. In order to extend this definition to our context, we replace the concept of children cells with the one of *owned points*.

Definition 1 (Vertical communication efficiency for non-matching levels)**.** *Given two arbitrarily partitioned triangulations discretizing* Ω*, the* vertical communication efficiency *is the share of* owned points *on the finer grid that have the same owning process as their corresponding owners on the coarse grid.*

Moreover, we consider other more classical metrics taken from [145, 65]:

- **Serial workload**: Sum of the number of cells on all levels $\mathcal{W}_s := \sum_l \mathcal{C}_l$, being $\mathcal{C}_l$ the number of cells on the l-th level.

- **Parallel workload**: Sum of the maximum number of cells owned by any process on every level of the hierarchy: $\mathcal{W}_p := \sum_l \max_p \mathcal{C}_l^p$, where $\mathcal{C}_l^p$ is the number of cells owned by process p on level l. Load imbalances imply that in general one has $\mathcal{W}_p \neq \frac{\mathcal{W}_s}{p}$, meaning that the work is not properly distributed on the levels. The **parallel workload efficiency** is hence defined as $\frac{\mathcal{W}_s}{\mathcal{W}_p \cdot p}$.

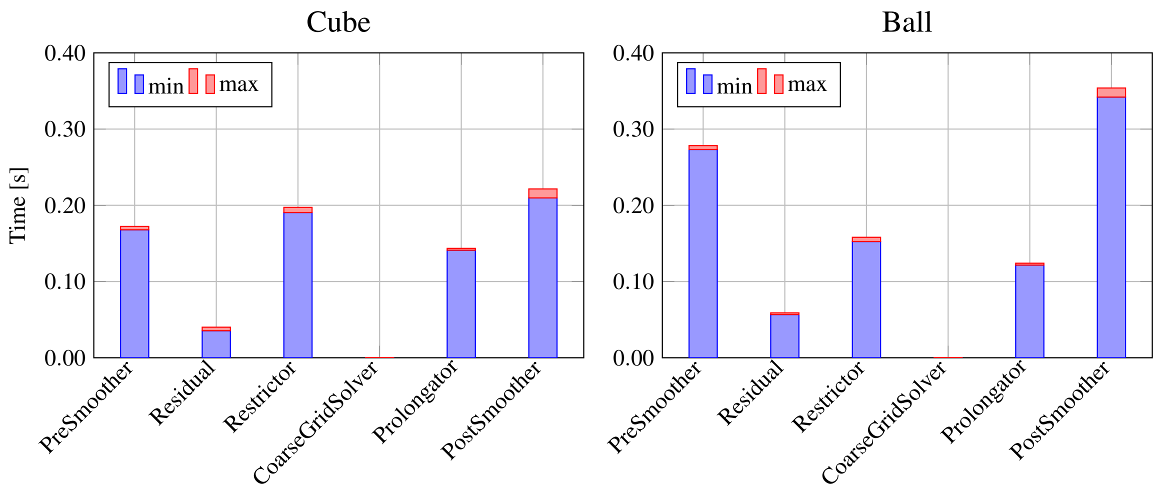


Fig. 2.16 Profile of a V-cycle with 12 processes with non-nested multigrid using a degree 5 Chebyshev smoother. Left: `Cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$.

- **Vertical efficiency** according to Definition 1: This metric gives a good indication on how much data has to be exchanged in order to perform intergrid transfers. A large value means that consecutive partitions of the levels are well overlapped, implying a small volume of communication. Within this chapter, all meshes are partitioned independently by employing space-filling curves through the P4EST library [55]. Optionally, graph partitioners such as METIS can be employed.

In the global coarsening case the parallel workload is, by construction, well distributed among the participating processes as each level is partitioned during the construction of the hierarchy. On the other hand, this affects the value of vertical efficiency which is generally low (less than 20%). Indeed, a high load balance and a high overlap of parallel partitions are mutually orthogonal requirements. Overall, this implies a higher computational pressure on the data exchange phase needed for prolongation and restriction operations. In this context, where levels do not share the same coarse mesh, the drop in vertical efficiency is even more pronounced compared to global coarsening. Tables 2.13 and 2.14 confirm this expected behavior for the levels displayed in Figures 2.7 and 2.8. In particular, the good parallel workload efficiency of global coarsening is automatically inherited, at the cost of a poor vertical efficiency for both geometries. Table 2.15 reports the findings for the piston test case in 2.10 partitioned with 12 processors. The statements made for the previous geometries still hold also for the piston hierarchy. In this case the low value of the vertical efficiency is even more prominent. The vertical efficiency obtained employing METIS as partitioning tool is shown in the last column of Tables 2.13,2.14, and 2.15. As it can be appreciated from Figure 2.20, the overlap of the parallel partitions for two consecutive levels is quite low with both partitioners, although METIS gives slightly better results. Our findings match with the ones reported in [145], so making definite statements is not possible as they would depend on the number of processes and are, in practice,

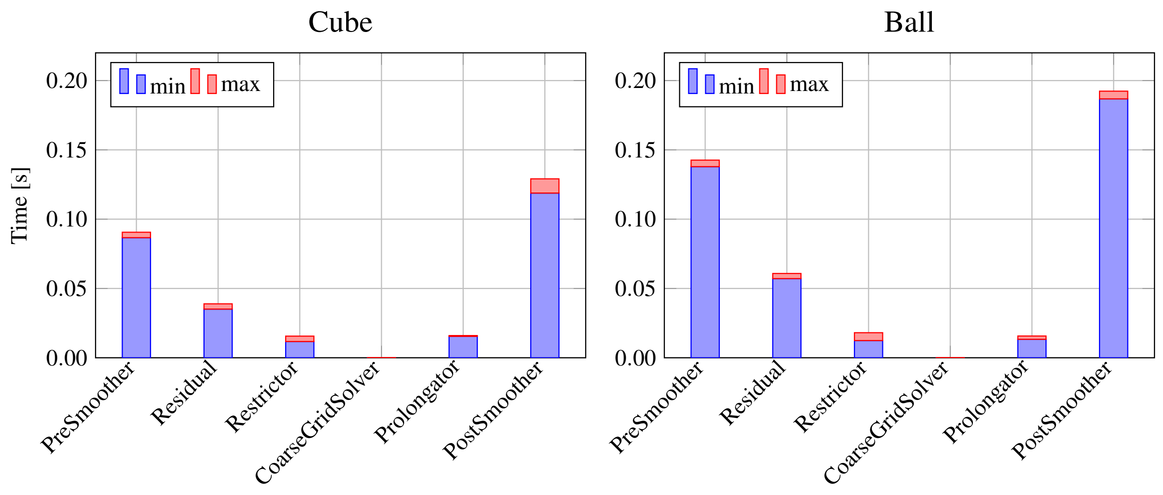


Fig. 2.17 Profile of a V-cycle with 12 processes with global coarsening variant. Left: `cube` example with $L = 7$ and $p = 4$. Right: `ball` example with $L = 6$ and $p = 4$. Notice how, compared to the previous Figure 2.15, the only major difference are bar charts associated to prolongation and restriction.

problem-specific. A possible workaround that aims to reduce the load imbalance between levels could consist in matching the partitioning of some levels of the hierarchy.

1 process		12 processes			
l	wl	wl	wl-eff	v-eff	v-eff METIS
3	5.0e+3	4.4e+2	99%	10%	6%
4	1.0e+4	8.5e+2	99%	7%	5%
5	1.8e+4	1.5e+3	99%	6%	8%
6	3.3e+4	2.7e+3	99%	15%	35%

Table 2.13 Multigrid statistics for the 2D L-shaped test case for different number of levels (wl: serial/parallel workload, wl-eff: parallel workload efficiency, v-eff: vertical communication efficiency, v-eff-METIS: vertical communication efficiency employing METIS.).

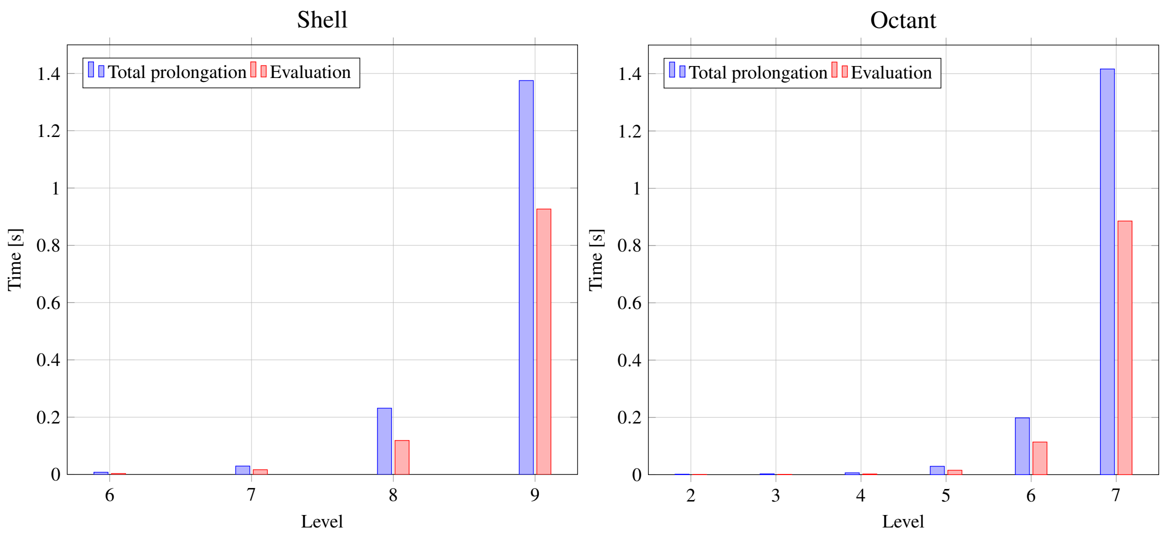


Fig. 2.18 Breakdown of the computational time (in seconds) required for prolongation between levels with 12 processes. In blue: total time spent to prolongate a coarse finite element field from level l to $l+1$. In red: time spent entirely on the evaluation at arbitrarily located reference points.

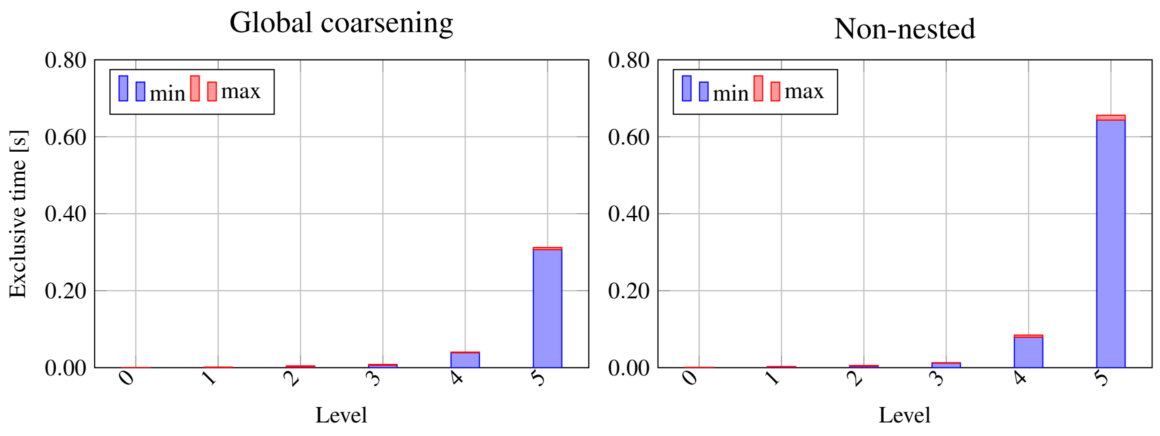


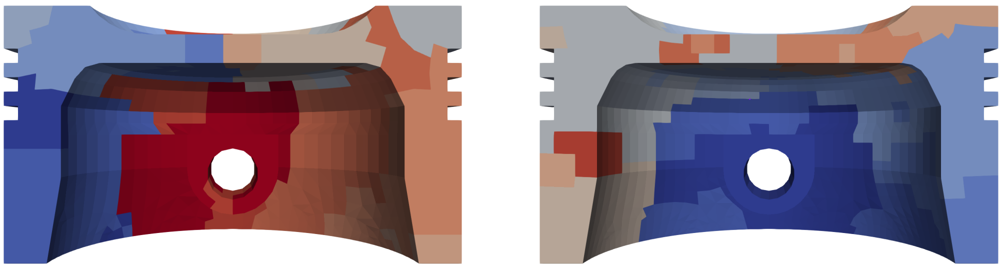
Fig. 2.19 Exclusive time in a V-cycle with 12 processes for the cube example with $L=7$ and $p=4$. Right: global coarsening. Left: non-nested multigrid.

	1 process	12 processes			
l	wl	wl	wl-eff	v-eff	v-eff METIS
2	1.5e+4	1.2e+3	99%	1%	10%
3	4.2e+4	3.5e+3	99%	2%	11%
4	9.9e+4	8.2e+3	99%	5%	7%

Table 2.14 Multigrid statistics for the 3D Fichera test case for different number of levels (wl: serial/parallel workload, wl-eff: parallel workload efficiency, v-eff: vertical communication efficiency, v-eff-METIS: vertical communication efficiency employing METIS.).

	12 processes			
l	wl	wl-eff	v-eff	v-eff METIS
2	1.5e+3	99%	1%	1%
3	4.1e+3	99%	2%	9%
4	7.9e+3	99%	5%	9%

Table 2.15 Multigrid statistics for the piston test case for different number of levels (wl: serial/parallel workload, wl-eff: parallel workload efficiency, v-eff: vertical communication efficiency, v-eff-METIS: vertical communication efficiency employing METIS.).



(a) Coarser level. (b) Finer level.

Fig. 2.20 Clipped view of consecutive levels in the hierarchy for the piston test case 2.10 partitioned across 12 processors (each color represents a different MPI rank). It can be appreciated how a fixed MPI rank owns different regions when the mesh is refined.

2.6 Conclusions

In this chapter, we have presented a matrix-free and memory-distributed implementation of the *non-nested* multigrid method which gives large flexibility for the levels that can be employed, allowing them to be arbitrarily overlapping and distributed among processors. We have shown its robustness through an extensive set of examples by varying grids, equations, polynomial degrees, and the number of levels. We confirmed its reliability also in the case of complicated three-dimensional geometries. Building on top of the DEAL.II finite element library, it was possible to integrate our implementation with highly-optimized and state-of-the-art matrix-free evaluation kernels. Finally, we have carried out a breakdown of the different components of our pipeline, showing the computational cost associated to the different phases and pointing to possible future improvements.

Part II

Polytopic Finite Elements

Chapter 3

Agglomeration of polytopic grids

We present a novel approach to perform agglomeration of polygonal and polyhedral grids based on spatial indices. Agglomeration strategies are a key ingredient in polytopic methods for PDEs as they are used to generate (hierarchies of) computational grids from an initial grid. Spatial indices are specialized data structures that significantly accelerate queries involving spatial relationships in arbitrary space dimensions. We show how the construction of the R-tree spatial indexes database of an arbitrary fine grid offers a natural and efficient agglomeration strategy with the following characteristics: i) the process is fully automated, robust, and dimension-independent, ii) it automatically produces a balanced and nested hierarchy of agglomerates, and iii) the shape of the agglomerates is tightly close to the respective axis aligned bounding boxes. Moreover, the R-tree approach provides a full hierarchy of nested agglomerates which permits fast query and allows for efficient geometric multigrid methods to be applied also to those cases where a hierarchy of grids is not present at construction time. We present several examples based on polygonal discontinuous Galerkin methods, confirming the effectiveness of our approach in the context of challenging three-dimensional geometries, including for the design of geometric multigrid preconditioners.

Contents

3.1 Literature review **76**
3.2 R-trees **78**
3.2.1 R-trees and finite element meshes 80
3.3 Agglomeration based on R-trees **81**
3.4 Validation **83**
3.4.1 Validation on a set of grids 83
3.4.2 Quality of resulting elements 86
3.4.3 Performance validation 91
3.5 Polytopic discontinuous Galerkin **92**
3.5.1 Notation and model problem 92

3.5.2 Finite element spaces and trace operators 92
3.5.3 Convergence tests 95
3.5.4 Multigrid preconditioning 98
3.6 Implementation details 102
3.6.1 Data structures and interfaces 102
3.6.2 Memory distributed implementation 105
3.6.3 Building the sparse operator 107
3.6.4 Benchmarks 108
3.7 Conclusions 111

3.1 Literature review

The Finite Element Method (FEM) is well known to provide a flexible discretization framework thanks to the use of unstructured meshes. Tetrahedral, hexahedral, or prismatic elements are typically used to mesh general domains. Mesh generation is a crucial step in the FEM and often a bottleneck in the resolution of complex real-world models.

To overcome these limitations, a natural possibility is to use methods that leverage general polygons and polyhedra as mesh elements. These methods, known as polytopic methods, have witnessed tremendous development in the last two decades. We refer, for instance, to polygonal FEM [168, 93], Mimetic Finite Differences [34, 120, 50], Virtual Element Method (VEM) [35, 16], Polygonal Discontinuous Galerkin [60, 30, 15], Hybridized Discontinuous Galerkin (HDG) [66, 68, 67], and Hybrid High-Order method (HHO) [77, 78].

Regardless of the underlying method, generating hierarchies of computational grids is a key step in the efficient numerical solution of partial differential equations. For simple geometries and traditional FEMs, the generation of these hierarchies is often straightforward, and it is performed in a bottom-up approach, by refinement of an initially coarse grid. For complex geometries, however, the construction of a hierarchy of grids may be challenging or impractical. One may only have access to non-nested sequences, which requires the construction of challenging transfer operators with a possibly high computational cost due to the non-matching nature of consecutive levels. For instance, this may occur when CAD (computer-aided design) models are meshed with external software and one is given a very fine geometry for which no hierarchical information is available.

The use of polytopic methods is attractive in this respect since coarse grids can be simply generated by merging polygonal and polyhedral elements [63, 14, 30, 150]. However, providing automated and good-quality agglomeration strategies for polygonal and polyhedral elements remains an open problem and challenging task. It is indeed crucial, but challenging, to preserve the original mesh quality, as any deterioration could potentially affect the overall performance of the method in terms of stability and accuracy. Typically, general graph-partitioning tool such as METIS [124] are used, which are not designed to retain the properties of the underlying fine mesh.

During the last few years, significant progress has been made, for instance by exploiting Machine Learning (ML) algorithms. ML provides a framework to automatically extract information from data to enhance and accelerate numerical methods for scientific computing [156, 157]. Their use in handling polygonal meshes has been recently introduced by Antonietti and co-authors in [20, 58, 17].

In this chapter, we propose an alternative approach to produce efficient and high-quality agglomerates based on R-trees [105, 138], a spatial indexing data structures. R-trees excel at organizing spatial data using bounding boxes, particularly in contexts where performing spatial queries for large sets of geometric objects in a fast way is required. Such geometric predicates are ubiquitous within the efficient implementations of non-matching finite element techniques, where two different geometries, usually representing different physical components, overlap arbitrarily. This is the case for the immersed methods [127, 43] studied in Chapter 1 as well as in particle methods [121]. By leveraging the R-trees structure, we develop an efficient and automatic mesh agglomeration algorithm specifically tailored for polygonal and polyhedral grids. This approach is fully automated, robust, and dimension-independent. Moreover, it preserves mesh quality while significantly reducing the computational cost associated with the agglomeration process.

To validate R-tree as an agglomeration methodology, we assess the quality metrics of the polytopic elements produced from a set of representative fine grids. We compare our findings with METIS, a standard solver for graph partitioning, which is designed to process only the graph information regarding the mesh, and is often employed as an agglomeration algorithm. Our results show that the R-trees approach produces meshes which are either similar or superior in quality to those produced by METIS using a fraction of the computational time. In particular, R-tree based agglomeration preserves structured meshes, a property not shared by METIS. Thus, for instance, when starting from a square grid, the repeated application of R-tree agglomeration produces a sequence of nested square grids. Moreover, even in the case of unstructured meshes, subsequent agglomerates tend to align to the corresponding bounding boxes, thus producing logically rectangular grids.

In the context of polytopic methods, multilevel solvers generally rely on non-nested hierarchies due to the presence of general shapes which naturally induces non-nestedness between levels. As a consequence, ad-hoc definitions of the transfer operators have to be employed, see e.g. [7, 76]. On the contrary, by exploiting the tree structure and appropriate parent-child relationships, it becomes possible to automatically extract *nested* sequences of agglomerated meshes, which can then be directly used within multigrid solvers. We propose R-tree based MultiGrid (R3MG) preconditioning with discontinuous Galerkin methods. We test R3MG on a second-order elliptic model problem discretized using the polytopic *hp*-version of Interior Penalty discontinuous Galerkin method analyzed in [59–61]. Multigrid methods for DG discretizations have been developed and studied, for instance, in [10, 7, 9, 123]. Exploiting the nested sequences of polytopic meshes produced by R-tree agglomeration, we construct multigrid preconditioners and report on the number of iterations required by the preconditioned conjugate gradient iterative solver. Our results show good convergence properties of both the two- and three-level preconditioned iterative solver in two- and three-dimensions. The experiments are carried out using a newly developed C++ library based on the well-established

DEAL.II Finite Element Library [21]. Our implementation allows for the use of polytopic discontinuous Galerkin methods on agglomerated grids both in 2D and 3D, in parallel, and with different agglomeration strategies.

The chapter is organized as follows. In Section 3.2 we recall the R-tree data structure and the algorithmic realization of our approach, which is then used to build hierarchies of agglomerated grids in Section 3.3. We validate the R-tree based agglomeration strategy in Section 3.4, while in Section 3.5 we introduce the polytopic discontinuous Galerkin method for the discretization of second-order elliptic problems on general meshes and showcase the application of R-tree based agglomeration in the solution of benchmark PDEs as well as its application to multigrid methodology. In Section 3.6 we will discuss relevant implementation details and data structure. Finally, Section 3.7 summarizes our conclusions and points to further research directions.

3.2 R-trees

We briefly list the basic properties of the R-tree data structure proposed by Guttman in the seminal paper [105] and discuss its variations following [138]. R-trees are hierarchical data structures used for the dynamic organization of collections of d-dimensional geometric objects, representing them by their *minimum bounding – axis aligned – d-dimensional rectangle*, also denoted by MBR. In this context, dynamic means that no global reorganization is required upon insertion or deletion of new elements of the tree. We will use the terms *MBR* or *bounding box* interchangeably throughout the work. An internal node of an R-tree consists of the MBR that bounds all its children. In particular, each internal node stores two pieces of data: a way of identifying a child node and the MBR of all entries within this child node. The actual data is stored in the *leaves* of the tree, i.e., the terminal nodes of the tree data strcuture. We summarize these aspects in the following definition of R-tree.

Definition 2. *R-tree of order (m,M). An R-tree of order (m,M) has the following characteristics:*

- *Each* leaf *node is a container that can host between a minimum of $m \leq \frac{M}{2}$ and a maximum of M entries (except for the root node, which is allowed to contain fewer elements if the number of objects to classify is less than m). Each entry is of type (MBR, id) where id is the object's identifier and MBR is the minimum bounding rectangle that covers the object.*

- *Each* internal *node is a container that can store again between a minimum of $m \leq \frac{M}{2}$ and a maximum of M entries. Each entry is of the form (MBR, p) where p is a pointer to a child node (that can be either an* internal *or a* leaf *node) and MBR is the minimum bounding rectangle that covers all the MBRs contained in this child.*

- *All leaves of the R-tree are at the same level.*

- *The minimum allowed number of entries in the root node is 2 unless it is a leaf node, in which case it may contain zero or one entry (and this is the only exception to the rule that all leaves must contain a minimum of m entries).*

An example of the minimum bounding rectangles of some geometric objects (not shown) is given in Figure 3.1, while the associated R-tree of order $(4,2)$ is shown in Figure 3.2. In this example, the leaf level stores the minimum bounding rectangles D,E,F,G,H,I,J,K,L,M,N. Conversely, the internal node comprises the three MBRs A,B,C. Notice the empty box in the internal node and the rightmost leaf node, meaning another entry could be stored there since the order is $(4,2)$.

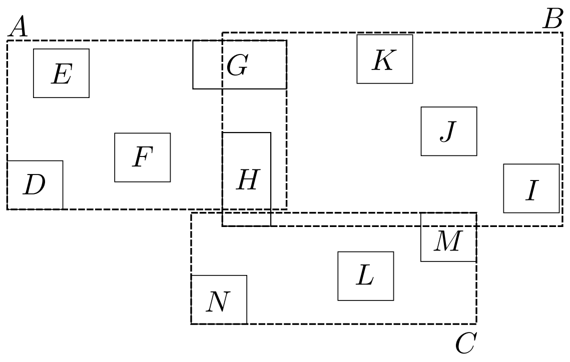


Fig. 3.1 Example of MBRs holding geometric data and their MBRs.

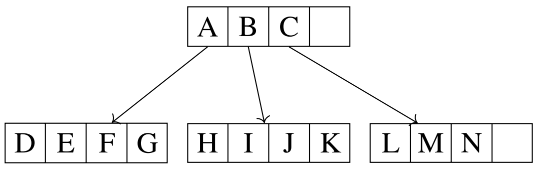


Fig. 3.2 Corresponding R-tree data structure.

The original R-tree is based solely on the minimization of the measure of each MBR. Several variants have been proposed, aimed at either improving performance or flexibility, generally depending on the domain of application.

In view of optimizing quality and handling of grid generation, it is desirable to minimize the *overlap* between MBRs. Indeed, the larger the overlap, the larger the number of paths to be processed during queries. Moreover, the smaller the overlap the closer the agglomerated element will conform to the corresponding bounding boxes, thus making the resulting agglomerated grid qualitatively close to rectangular. Among the several available variants, we adopt the R*-tree data structure designed in [33]. The criteria that R*-trees aim to achieve are the following:

- *Minimization of the area covered by each MBR.* This criterion aims at minimizing the dead space (area covered by MBRs but not by the enclosed elements) to reduce the number of paths pursued during query processing.
- *Minimization of overlap between MBRs.* The larger the overlap, the larger the expected number of paths followed for a query. As such, this criterion has the same objective as the previous one.
- *Minimization of MBRs perimeters.* Shaping more square bounding boxes results in reduced query time as this suffers in the presence of large overlaps and/or heterogeneous shapes. Moreover, since square objects are packed more easily, the corresponding MBRs at upper levels are expected to be smaller.
- *Maximization of storage utilization.* Increasing the storage utilization per node will generally reduce the cost of queries since the height of the tree will be low. This holds especially true for

larger queries, where a significant portion of the entries satisfies the query. Conversely, when storage utilization is low, more nodes tend to be invoked during query processing.

It should be noted that such requirements can easily be orthogonal. As an example, keeping the area and the overlap between MBRs low could imply a lower number of entries packed within each node of the tree, leading to higher storage usage. Therefore, the R*-tree does some heuristics to find the best possible compromise of these criteria. We refer to the original paper [33] for the relevant algorithmic details.

In our implementation, we rely on the BOOST.GEOMETRY module supplied by the Boost C++ Libraries [44] for the construction and manipulation of R*-trees. BOOST is a generic C++ library providing concepts, geometry types, and general algorithms for solving, among others, problems in computational geometry. Its kernels are designed to be agnostic with respect to the number of space dimensions, coordinate systems, and types. Moreover, BOOST provides the capability to perform spatial queries with polygonal shapes, therefore we envision its usage also when the underlying grid is already polytopic, such as with Voronoi tessellations.

In the remainder of the thesis, we will not make any distinction between R-trees and R*-trees, as we always employ the latter.

3.2.1 R-trees and finite element meshes

Here we describe the construction of the R-tree data structure associated with a given finite element mesh. Let a domain $\Omega \subset \mathbb{R}^d$, $d \geq 1$ be given, which we identify with its partition into non-overlapping mesh elements $T \in \Omega$ covering Ω. The construction of the R-tree data structure for Ω is initialised with the construction of the minimal bounding rectangle for each mesh element. We denote by $\{\mathrm{MBR}(T_i)\}_{i=1}^N$, with N the cardinality of Ω, the resulting collection of MBRs. To give a simple yet practical example, consider the discretization of the unit square $[0,1]^2$ with the 8×8 square grid in Figure 3.3 (left). In this particular instance, for every $T \in \Omega$, it holds that $\mathrm{MBR}(T) \equiv T$. Once the container with all MBRs is stored, the R-tree is ready to be built. The generated hierarchical structure is depicted in Figure 3.3 (right). The root node, which constitutes the first level of the R-tree, has four entries with MBRs A, B, C, D (dashed lines). Each one of the entries points to an internal node, which in turn is composed of other four MBRs. For instance, looking at the entry associated with A, it can be seen that it has as a child the internal node with entries E, F, G and H, each one composed of other four elements, which are leaves of the R-tree (and coincide with the mesh elements). The same pattern applies to the other three blocks of the mesh. The resulting R-tree is a $(4,2)$ R-tree, according to Definition 2. A schematic view of the tree hierarchy is shown in Figure 3.4.

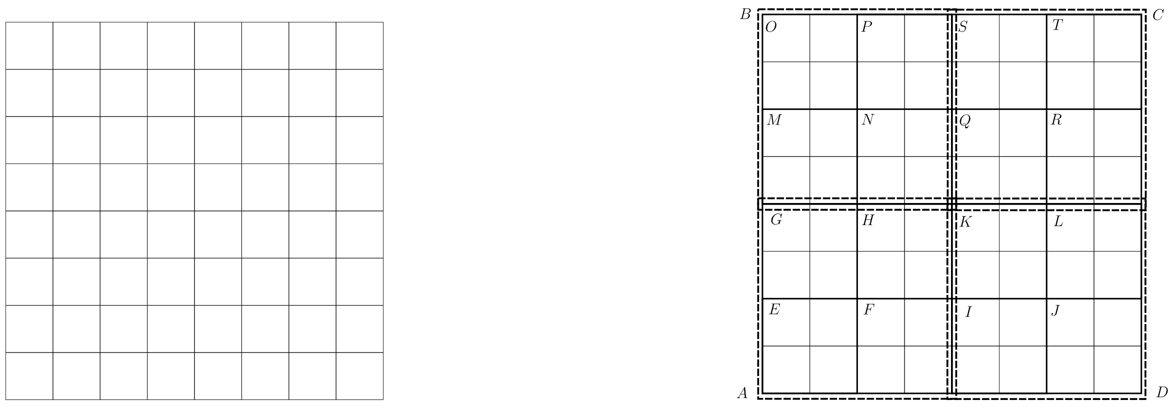


Fig. 3.3 Left: 8×8 square grid obtained with 3 uniform refinements of $\Omega = [0,1]^2$. Right: Minimal bounding rectangles generated on top of the grid elements of Ω.

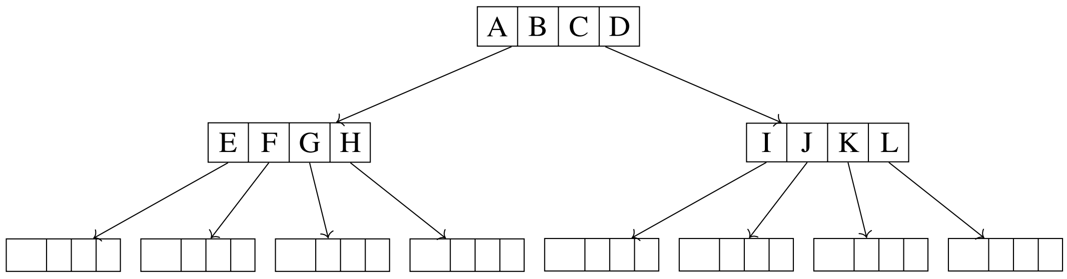


Fig. 3.4 R-tree data structure for the 8×8 mesh example. For the sake of readability, only two internal nodes are shown. Each entry at the leaf level is one mesh element T. Notice how each node stores exactly 4 entries.

3.3 Agglomeration based on R-trees

Here we describe the construction of an agglomerated mesh starting from a given finite element mesh and the associated R-tree data structure. Let the mesh Ω of cardinality N be given and assume the associated R-tree of order (m, M) has been constructed. We denote with L the total number of levels of the R-tree and with N_l the set of nodes in level l, for $l \in \{1, \dots, L\}$. Our agglomeration strategy depends on an input parameter $l \in \{1, \dots, L\}$ which describes the level to be employed to generate the final agglomerates. The basic idea consists of looping through the nodes N_l and, for each node $n \in N_l$, descending recursively its children until leaf nodes are reached. These leaves share the same ancestor node n (on level l) and thus are *agglomerated*. We store such elements in a vector $v[n]$, indexed by the node n. This procedure is outlined in Algorithm 3.

After the recursive visit of the children of a node n, a sequence of at most M elements T_n^j can be stored in $v[n]$ and flagged appropriately for agglomeration. Since the R-tree data structure provides a spatial partition of mesh elements, each of which is uniquely associated with a node, the traversal of

Algorithm 3: Creation of agglomerates.

```
   Data: R-tree $R$ of order $(M,m)$
   Data: $l \in \{1,\dots,L\}$ target level.
   Result: $v$ vector s.t. $v[n]$ stores leafs associated to node $n$
1  Function ComputeAgglomerates($R,l$):
2      for node $n$ in $N_l$ do
3          $v[n] \leftarrow$ ExtractLeafs($l,n$);
4      return $v$;
```

Algorithm 4: Recursive extraction of leafs from a node n on level l.

```
   Data: $l \in \{1,\dots,L\}$ target level
   Data: $n \in N_l$
   Result: vector $v[n]$ containing leafs which share the ancestor node $n$
1  Function ExtractLeafs($l,n$):
2      if $l = 1$ then
3          $v[n] \leftarrow \{T_n^1,\dots,T_n^M\}$;
4      else
5          ExtractLeafs($l-1,n$)
```

all nodes on a level l provides a partition of mesh elements into agglomerates. The overall procedure can be summarized as follows:

1. Compute $\{\mathrm{MBR}(T_i)\}_i$ for $i = 1,\dots,N$,
2. Build the R-tree data structure using $\{\mathrm{MBR}(T_i)\}_i$ as described in Section 3.2,
3. Choose one level $l \in \{1,\dots,L\}$ and apply Algorithm 3,
4. For each node n, flag together elements of $v[n]$.

We point out that elements in $v[n]$ are usually mesh-like iterators, i.e. lightweight objects such as pointers that uniquely identify elements of Ω. As we will show in Section 3.4.3, the overall construction of the R-tree and the actual identification of agglomerates turns out to be quite efficient also for large 3D meshes. Moreover, having in mind multilevel methods, we notice that Algorithm 3 can be employed to generate sequences of *nested* agglomerated meshes. As a consequence, such meshes can be used as a hierarchy in a multigrid algorithm, allowing the usage of simpler and much cheaper intergrid transfer operators, when compared to the non-nested case. Building efficient intergrid transfer operators for non-matching meshes is far from trivial even for simple geometries and their construction constitutes a severe bottleneck in terms of computational efforts, becoming critical in 3D [7, 76]. We notice, however, that in the context of Lagrangian Finite Elements on standard hexahedral or quadrilateral grids, a completely parallel and matrix-free implementation of the non-nested geometric multigrid method has been recently addressed in [88] and is part of Chapter 2.

3.4 Validation

To show the effectiveness of the R-tree based grid agglomeration strategy, we perform several experiments with different grid types and compare it with METIS [124], a standard graph partitioning algorithm often used for grid agglomeration.

More in detail, we use the multilevel k-way partitioning algorithm implemented in METIS to perform a partition of the graph associated with the given grid. Each partition corresponds to one agglomerated element and the cardinality of the resulting grid is thus fixed a priori as the number of partitions of the graph. We recall that, instead, the input parameter in R-tree-based agglomeration is the level of the tree used to extract the agglomerates, as described in Algorithm 3. Therefore, we will write `extraction_level` and `n_partitions` for the parameters required by the R-tree and METIS strategy, respectively. The value of parameter M in 2 is set to 2^d, being d the space dimension, while m is set to $\frac{M}{2} = 2^{d-1}$.

In view of a fair comparison, we always employ METIS setting `n_partitions` as the number of elements produced by the R-tree-based algorithm with a given `extraction_level`. In this way, the grids we compare, although effectively different, are guaranteed to be made of the same number of elements.

A crucial part of this chapter has been the development of the C++ library POLYDEAL, using the well-established Finite Element library DEAL.II [23, 21] as a third-party library. POLYDEAL provides the building blocks for solving partial differential equations with polytopic discontinuous Galerkin methods. It is distributed and builds on the Message Passing Interface (MPI) communication model [142]. Providing an implementation within an existent finite element library has several advantages. Most importantly, we seamlessly inherit many robust features readily available and well-tested. Among them, we mention here P4EST [57] for mesh partitioning across several processors, and TRILINOS [113] or PETSC [27] as parallel linear algebra libraries which provide a large variety of solvers. The library can be compiled by following the instructions available at the maintained GitHub repository [86]. The results presented in this section, as well as the grids used in all the examples, are also made available in the same repository.

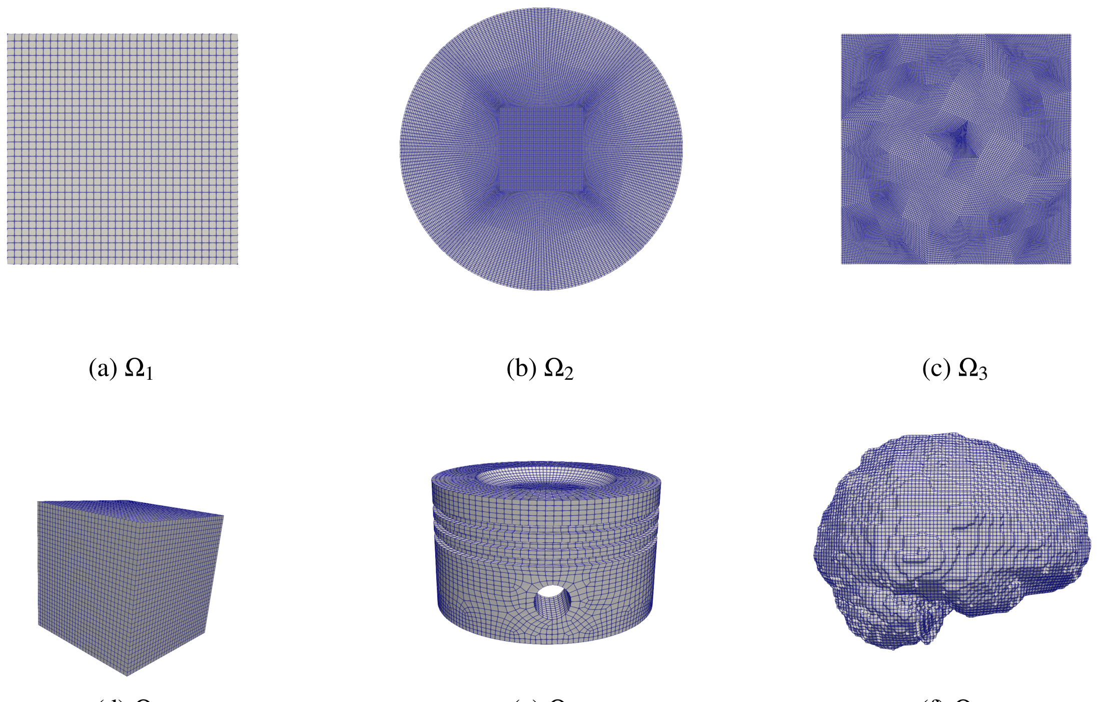
3.4.1 Validation on a set of grids

To validate our methodology, we consider the following set of grids, sampled in Figure 3.5:

- Ω_1 structured partition of $[0,1]^2$,
- Ω_2 structured partition of $\mathcal{B}_1(\mathbf{0})$,
- Ω_3 unstructured partition of $[0,1]^2$,
- Ω_4 structured partition of $[0,1]^3$,
- Ω_5: CAD-modelled mesh of a piston,

- Ω_6: mesh of a human brain.

We expect the circular domain and grid Ω_2 to provide a less favourable case, for which higher overlap is to be expected due to the fixed, axis-aligned, orientation of the bounding boxes. The grid Ω_5 has been generated after repairing and meshing the associated CAD 3D model with the commercial mesh generator CUBIT [1], while Ω_6 is a brain model created from magnetic resonance imaging (MRI) data which has been preprocessed, segmented, and meshed in [102]. These two latter grids are challenging and provide good three-dimensional test cases for assessing the performance of our algorithm.



(a) Ω_1 (b) Ω_2 (c) Ω_3

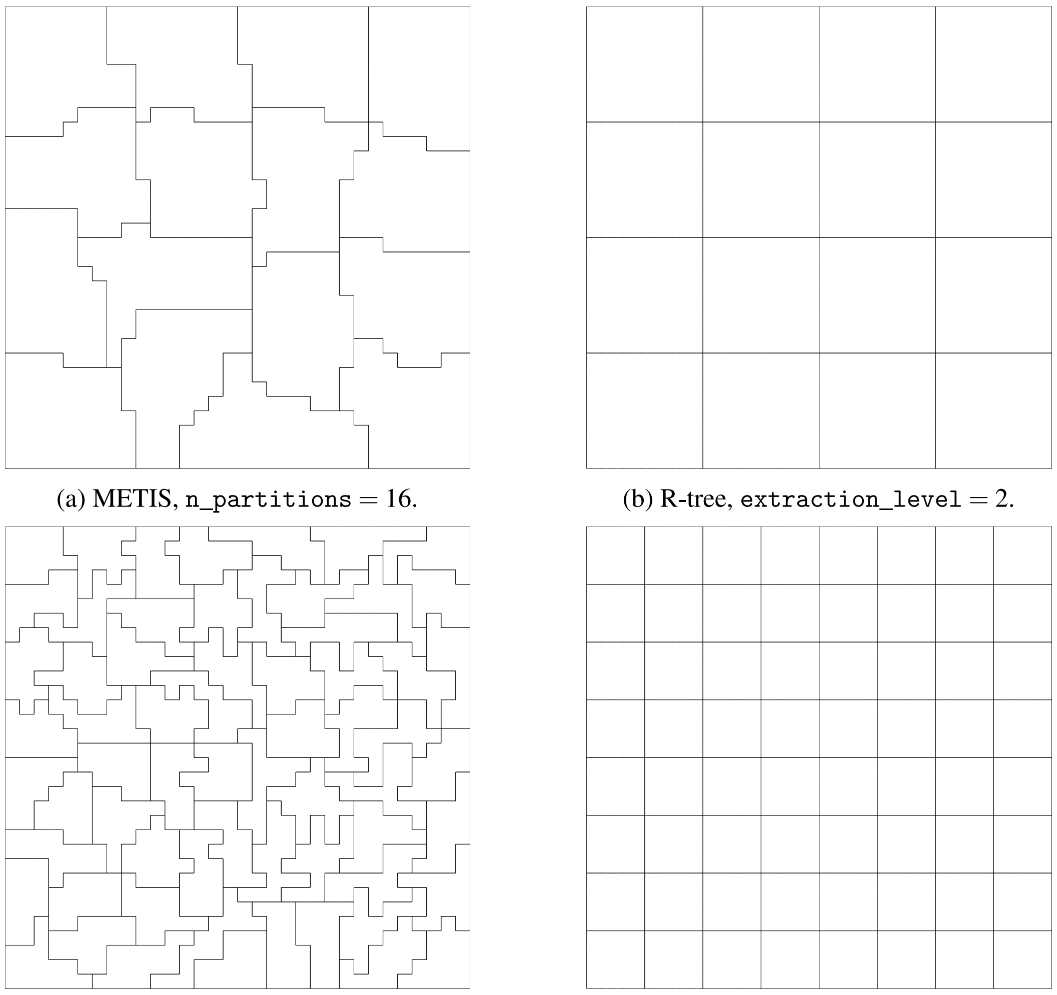
(d) Ω_4 (e) Ω_5 (f) Ω_6

Fig. 3.5 The set of meshes used in the numerical experiments.

Test 1: Structured square

We fix as underlying mesh Ω_1 the 32×32 structured grid of squares. We report in Figure 3.6 the grids obtained by agglomeration of Ω_1 using either METIS or the R-tree. With the extraction of the R-tree second level, we obtain the 4×4 square mesh reported in Figure 3.6b. The target number of mesh elements required by METIS is hence set to 16. The resulting agglomerated elements are jagged, cf. Figure 3.6a. This is to be expected: METIS only processes the information coming from the graph topology of the mesh, hence the agglomerates are not supposed to preserve in any way the initial geometry, despite the fact that each polygon is made by 16 sub-elements (as it is also the case for the R-tree).

We repeat the procedure by setting `extraction_level` $= 3$. The R-tree naturally produces the 8×8 Cartesian mesh, exactly the subdivision one would get by globally refining the coarser mesh in Figure 3.6b. We thus set 64 as the number of target elements for METIS. Results are shown in the bottom plots in Figure 3.6c. We observe that METIS produces polygons with even more jagged shapes, with many skinny and elongated elements,a particularly poor result considering that the underlying mesh is Cartesian.


(a) METIS, `n_partitions` $= 16$.

(b) R-tree, `extraction_level` $= 2$.

(c) METIS, `n_partitions` $= 64$.

(d) R-tree, `extraction_level` $= 3$.

Fig. 3.6 Comparison between METIS and R-tree based agglomeration starting from the grid Ω_1 displayed in Figure 3.5a. Grids displayed in the same row always comprise the same number of elements.

Test 2: structured ball

We consider next as Ω_2 the structured partition of a circle shown in Figure 3.5b. We remark that this grid is much finer than Ω_1 as it is made of 20,480 elements. We start with extracting the third level of the R-tree, which gives 20 agglomerates and accordingly set the target number of elements for METIS as `n_partitions` $= 20$. We observe from Figure 3.7b that the R-tree agglomerates conform to the rectangular shape of the respective bounding boxes, even though in this case the underlying quadrilateral mesh is not axis-aligned. METIS partitions, on the other hand, produce elements with general shapes. Extracting the next level (i.e. setting `extraction_level` $= 4$) of the hierarchy is equivalent to partitioning each agglomerate of `extraction_level` $= 3$ into balanced sub-agglomerates, as shown in Figure 3.7d, similarly to what happens with the square case shown previously. On the contrary, the two corresponding grids produced by METIS and shown in Figure 3.7a and Figure 3.7c are completely unrelated.

Test 3: unstructured square

The same procedure is repeated with the grid Ω_3, composed of 93,184 non-uniform quadrilaterals. The main difference between the two examples above relies on the fact that this grid is fully unstructured. With the R-tree, we extract first level 4, which gives 91 elements. Employing METIS with `n_partitions` $= 91$ as input value, we obtain the grid in Figure 3.8a. Here it is even more evident how the R-tree approach inherently gives rectangular-like shapes, in contrast to METIS. This is confirmed by the grids corresponding to `extraction_level` $= 5$ and shown in Figures 3.8c and 3.8d.

3.4.2 Quality of resulting elements

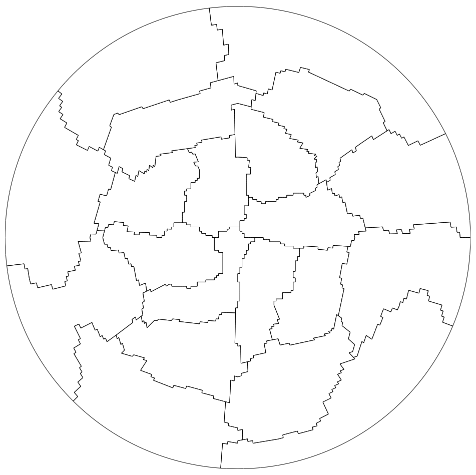
To assess the quality of the proposed methodology, we follow the same approach used in [20] and compute some of the quality metrics devised in [24], which are reported hereafter for completeness:

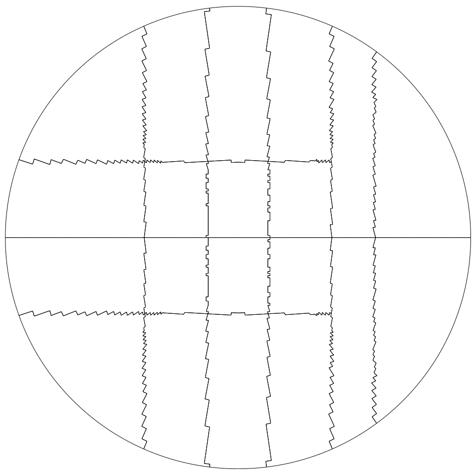
- *Uniformity Factor*. Ratio between the diameter of an element K and the mesh size h defined as the maximum overall diameter of the polygons in the mesh:

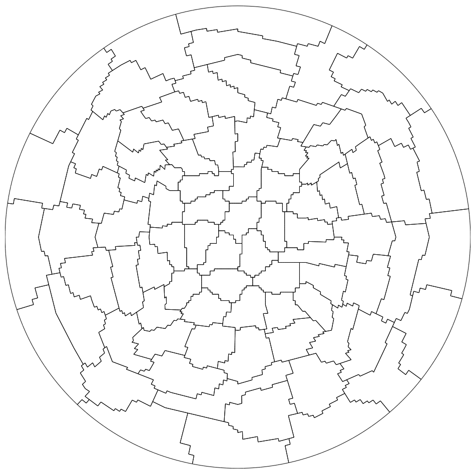
$$\mathrm{UF}(K) = \frac{\mathrm{diam}(K)}{h}.$$

- *Circle Ratio*. Ratio between the radius of the inscribed circle and the radius of the circumscribed circle of an element K:

$$\mathrm{CR}(K) = \frac{\max_{\{B(r)\subset K\}} r}{\min_{\{K\subset B(r)\}} r}.$$


(a) METIS, `n_partitions` $= 20$.


(b) R-tree, `extraction_level` $= 3$.


(c) METIS, `n_partitions` $= 80$.

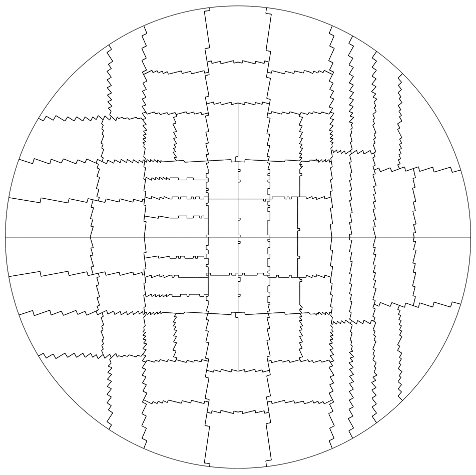

(d) R-tree, `extraction_level` $= 4$.

Fig. 3.7 Comparison between METIS and R-tree based agglomeration of the grid Ω_2 shown in Figure 3.5b. Grids displayed in the same row always comprise the same number of elements.

- *Box Ratio*. Ratio between the measure of an element K and the measure of its bounding box:

$$\mathrm{BR}(K) = \frac{|K|}{|\mathrm{MBR}(K)|}.$$

In addition to the previous metrics, we consider also the following:

- *Overlap Factor*. Ratio between the sum of the measures of all the bounding boxes in the polytopic mesh and the global measure of the domain:

$$\mathrm{OF}(\Omega) = \frac{\sum_{i=1}^{N} |\mathrm{MBR}(K_i)|}{|\Omega|},$$

  where N is the cardinality of the mesh Ω.

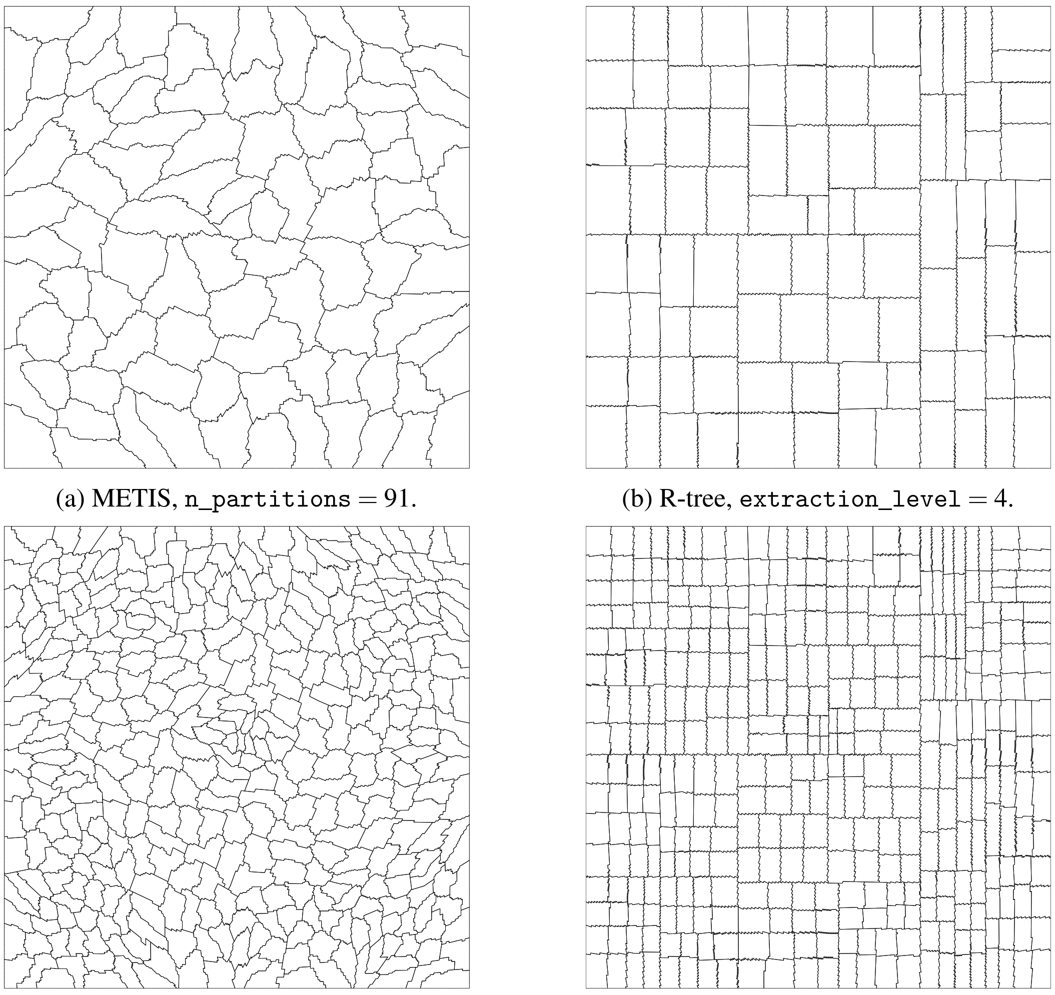

(a) METIS, `n_partitions` $= 91$. (b) R-tree, `extraction_level` $= 4$. (c) METIS, `n_partitions` $= 364$. (d) R-tree, `extraction_level` $= 5$.

Fig. 3.8 Comparison between METIS and R-tree agglomeration starting from the grid Ω_3 seen in Figure 3.5c. Grids displayed in the same row always comprise the same number of elements.

The first three metrics take values in $[0,1]$, while OF takes values greater or equal to 1. In particular, higher average values of UF imply that elements of the grid have comparable sizes, while higher averages of CR suggest that elements are close to circular shapes. To estimate this particular metric, we compute the radius of the inscribed circle of general polygons using the C++ library CGAL [83] and approximate the radius of the inscribed circle with $\frac{\text{diam}(P)}{2}$. The metric BR takes into account how much an agglomerate K and its bounding box are coinciding. Therefore, values close to 1 indicate that the bounding box is tightly close to the agglomerate.

The metric $\text{OF}(\Omega)$ is a global version of the metric BR: it estimates the sum of the overlaps between all bounding boxes relative to the measure of Ω. In addition, it indicates how tightly the computational domain can be covered by boxes. For instance, we obtain the optimal $\text{BR} = 1$ for the axis-aligned grid of Figure 3.5a, cf. Table 3.1.

In the table, we report the average values of UF, CR, and BR for each geometry when we perform the first round of agglomeration of the grids displayed in Figure 3.5 (top), namely the structure square, structure ball, and unstructured square grids. The last two columns of Table 3.1 show the global overlap factor OF(Ω). The R-tree approach produces elements with shapes close to the associated Cartesian bounding box, resulting in values of OF(Ω) close to 1 even in the case of underlying grids whose elements are *not* axis-aligned. With reference to the grid Ω_3, which is fully unstructured, we are nevertheless able to obtain an almost optimal value for the overlap factor. In the case of Ω_2, we have a slightly larger value of 1.2 due to the curved geometry. In this particular case, the contributions coming from agglomerates on the boundary exit from the domain due to the axis-aligned nature of the bounding boxes.

Regarding the procedure employing METIS, we note that this is not meant to conserve any geometric information. This fact can be visually deduced by examining the Figures in Section 3.4.1. The results reported in Table 3.1 show that the R-tree grids are superior to the corresponding METIS grids in all four metrics. In Figures 3.9 we show, for each grid and agglomeration strategy, the minimum, maximum, and average value of each metric. The metrics obtained after performing a second round of agglomeration are reported in Table 3.2 and Figure 3.10. We observe that in both cases the metrics related to the R-tree approach are better than with METIS. Among these metrics, BR is showing values always close to 1 with the R-tree and high gaps compared to METIS, confirming the fact that shapes are preserved while increasing the levels. Furthermore, the overlap factor does not deteriorate using the R-tree, meaning that the global percentage of overlap of the bounding boxes is very low and that the resulting polygonal grid is very close to a global refinement of the coarser agglomerates.

Grid	# polygons	UF		CR		BR		OF	
		R-tree	**METIS**	**R-tree**	**METIS**	**R-tree**	**METIS**	**R-tree**	**METIS**
Ω_1	16	1.0	0.8441	0.7071	0.5054	1.0	0.7713	1.00	1.32
Ω_2	20	0.7317	0.6193	0.4432	0.3690	0.8541	0.5485	1.23	1.97
Ω_3	91	0.7622	0.7500	0.7622	0.4070	0.9235	0.5965	1.08	1.71

Table 3.1 Average values for the Uniformity Factor (UF), Circle Ratio (CR), and Box Ratio (BR), and global Overlap Factor (OF) for the *first* level of agglomerated meshes with the two different agglomeration strategies.

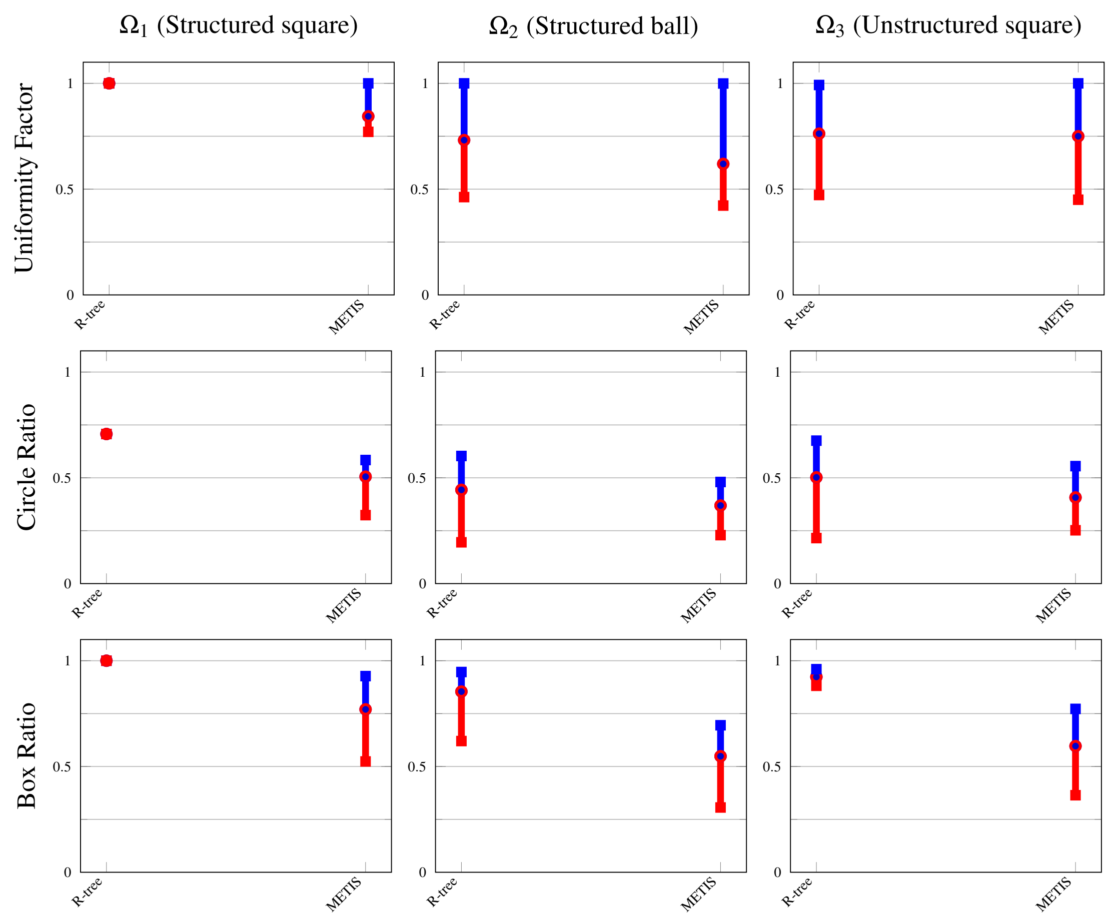


Fig. 3.9 Min, Max and Average chart of metrics UF, CR, and BR for grids $\Omega_1, \Omega_2, \Omega_3$ and different agglomeration strategies. In the plots on the left, metrics related to the R-tree collapse to a single dot since all mesh elements have the same value for that particular metric.

Grid	# polygons	UF		CR		BR		OF	
		R-tree	**METIS**	**R-tree**	**METIS**	**R-tree**	**METIS**	**R-tree**	**METIS**
Ω_1	64	1.0	0.6382	0.7071	0.3368	1.0	0.5605	1.00	2.00
Ω_2	80	0.599658	0.5457	0.4462	0.4002	0.8356	0.5890	1.23	1.77
Ω_3	364	0.698383	0.6111	0.4799	0.3970	0.8630	0.5815	1.15	1.75

Table 3.2 Average values for the Uniformity Factor (UF), Circle Ratio (CR), Box Ratio (BR), and global Overlap Factor (OF) for the *second* level of agglomerated meshes with the two different agglomeration strategies.

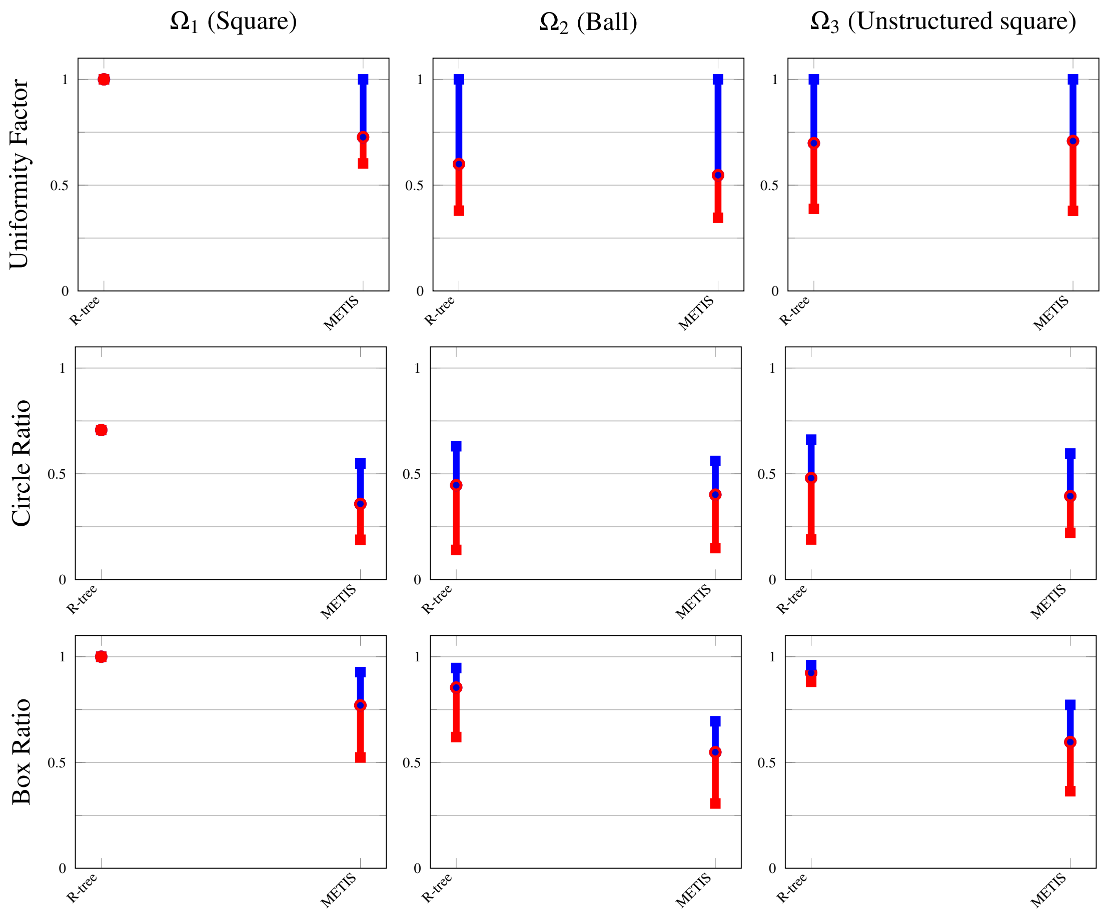


Fig. 3.10 Min, Max and Average chart of metrics UF, CR, and BR for grids $\Omega_1, \Omega_2, \Omega_3$ and different agglomeration strategies, after creation of one more level. In the plots on the left, metrics related to the R-tree collapse to a single dot since all mesh elements have the same value for that particular metric.

3.4.3 Performance validation

To further assess our method, a breakdown of the computing times required by the agglomeration process is tracked both in 2D and in 3D test cases. The grid used for the 2D case is Ω_3 (the unstructured square $[0,1]^2$), while for 3D meshes we consider a globally refined version of the piston grid in Figure 3.5e, referred to as $\Omega_{5,\text{ref}}$, and the brain mesh Ω_6. The elapsed time we are interested in measuring is the time to build the target polytopic mesh. This means that different components must be measured depending on the agglomeration strategy. In the case of METIS, we measure the wall-clock time (in seconds) associated with the call to the METIS function `METIS_PartGraphKway()`. For the R-tree-based strategy, we time cumulatively the following phases:

- Build the R-tree;
- Visit the hierarchy and store data structures needed to generate agglomerates;

- Flag elements of the underlying triangulation Ω.

We perform 10 runs for each agglomeration strategy and average the recorded wall-clock times. All the experiments have been performed on a 2.60GHz Intel Xeon processor. We observe from Figure 3.11 that the R-tree based approach keeps constant timings which are independent of the extraction level. On the other hand, a graph partitioner requires increasing computational times which in all cases are orders of magnitude larger than those required by the R-tree-based strategy. Moreover, the parallel extension of our approach in the case of distributed grids is conceptually straightforward. In an MPI-distributed framework, each process stores only a local part of the computational domain. And, within each locally owned partition, agglomeration can be performed locally. Since the original fine grid is already distributed, it must be noted that the ghost polytopic elements do not live on the layer of standard ghost cells as is usual with classical FEMs implementations, but are owned by a different partition. Our algorithmic realization carefully avoids calls to potentially expensive collective MPI routines and performs the exchange of such information in a setup phase, cf. Section 3.6.

3.5 Polytopic discontinuous Galerkin

3.5.1 Notation and model problem

Let Ω be a bounded, simply connected, and open polygonal/polyhedral domain in $\mathbb{R}^d$, $d = 2,3$. The boundary $\partial\Omega$ of Ω is split into two disjoint parts, Γ_{D} and Γ_{N} with $|\Gamma_{\mathrm{D}}| \neq 0$. We consider the linear elliptic problem: find $u \in H^1(\Omega)$, such that

$$\begin{cases} -\nabla\cdot(a\nabla u) = f & \text{in } \Omega, \\ u = g_{\mathrm{D}} & \text{on } \Gamma_{\mathrm{D}}, \\ a\nabla u\cdot \mathrm{n} = g_{\mathrm{N}} & \text{on } \Gamma_{\mathrm{N}}, \end{cases} \tag{3.1}$$

with data $f \in L^2(\Omega)$, $g_{\mathrm{D}} \in H^{1/2}(\Gamma_{\mathrm{D}})$, $g_{\mathrm{N}} \in L^2(\Gamma_{\mathrm{N}})$, and a positive definite diffusion tensor $a \in \left[L^\infty(\Omega)\right]^{d\times d}$. Setting $H^1_{\mathrm{D}} := \{v \in H^1(\Omega) : v = 0 \text{ on } \Gamma_{\mathrm{D}}\}$, the weak formulation of (3.1) reads: find $u \in H^1(\Omega)$, with $u = g_{\mathrm{D}}$ on Γ_{D} such that

$$\int_\Omega a\nabla u\cdot\nabla v\,\mathrm{d}\mathbf{x} = \int_\Omega f v\,\mathrm{d}\mathbf{x} + \int_{\Gamma_{\mathrm{N}}} g_{\mathrm{N}} v\,\mathrm{d}s, \tag{3.2}$$

for all $v \in H^1_{\mathrm{D}}(\Omega)$. The well-posedness of the weak problem (3.2) is guaranteed by the Lax-Milgram Lemma.

3.5.2 Finite element spaces and trace operators

We consider meshes consisting of general polygonal (for $d = 2$) or polyhedral (for $d = 3$) mutually disjoint open elements $K \in \mathcal{T}$, henceforth termed collectively as *polytopic*, with $\cup_{K\in\mathcal{T}}\bar{K} = \bar{\Omega}$.

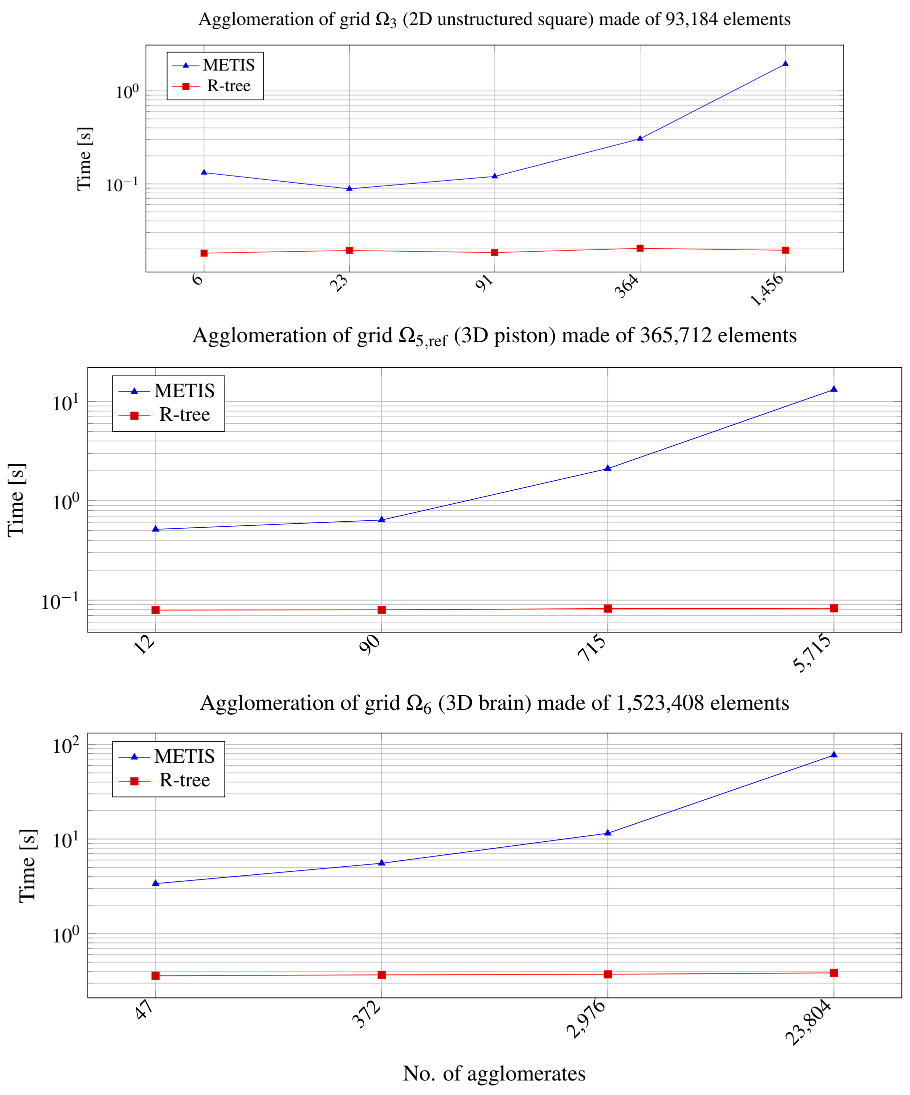


Fig. 3.11 Wall clock time (in seconds) needed to build polytopic grids with R-tree and METIS.

Given $h_K := \text{diam}(K)$, the diameter of $K \in \mathcal{T}$, we define the mesh-function $\mathrm{h} : \cup_{K\in\mathcal{T}} K \to \mathbb{R}_+$ by $\mathrm{h}|_K = h_K$, $K \in \mathcal{T}$. Further, we let $\Gamma := \cup_{K\in\mathcal{T}} \partial K$ denote the mesh skeleton and set $\Gamma_{\text{int}} := \Gamma \backslash \partial\Omega$. The mesh skeleton Γ is decomposed into $(d-1)$–dimensional simplices F denoting the mesh *faces*, shared by at most two elements. These are distinct from elemental *interfaces*, which are defined as the simply connected components of the intersection between the boundary of an element and either a neighbouring element or $\partial\Omega$. As such, an interface between two elements may consist of more than one face, separated by hanging nodes/edges shared by those two elements only.

Over $\mathcal{T}$ we introduce the discontinuous *finite element space* defined by

$$V_h := \{u \in L_2(\Omega) : u|_K \in \mathcal{Q}_p(K), K \in \mathcal{T}\},$$

for some $p \in \mathbb{N}$, with $\mathcal{Q}_p(K)$ denoting the space of tensor-product polynomials of degree p on K. Note that in original works on polytopic methods [59–61], the space of total degree p was used instead, with the flexibility of variable polynomial degrees. Here we keep a unique polynomial degree and exploit once again the bounding boxes to construct V_h: we use readily available tensor-product basis for polynomials defined on the bounding boxes and then consider their restriction over the physical element K; see Section 3.5.3 for more details. We observe though that all developments described below remain valid with the appropriate adjustments in the context considered in the original works.

Let K_i and K_j be two adjacent elements of $\mathcal{T}$ sharing a face $F \subset \partial K_i \cap \partial K_j \subset \Gamma_{\text{int}}$. The outward unit normal vectors on F of ∂K_i and ∂K_j are denoted by $\boldsymbol{n}_{K_i}$ and $\boldsymbol{n}_{K_j}$, respectively. For a function element-wise continuous function $v\colon \Omega \to \mathbb{R}$, we define its *average* and the *jump* across F by

$$[\![v]\!]|_F := v|_{F\cap K_i}\boldsymbol{n}_{K_i} + v|_{F\cap K_j}\boldsymbol{n}_{K_j}, \qquad \{\!\{v\}\!\}|_F := \frac{1}{2}(v|_{F\cap K_i} + v|_{F\cap K_j}). \tag{3.3}$$

Similarly, for a vector-valued function $\boldsymbol{w}$, piecewise smooth on $\mathcal{T}$, we define

$$[\![\boldsymbol{w}]\!]|_F := \boldsymbol{w}|_{F\cap K_i}\cdot\boldsymbol{n}_{K_i} + \boldsymbol{w}|_{F\cap K_j}\cdot\boldsymbol{n}_{K_j}, \qquad \{\!\{\boldsymbol{w}\}\!\}|_F := \frac{1}{2}(\boldsymbol{w}|_{F\cap K_i} + \boldsymbol{w}|_{F\cap K_j}). \tag{3.4}$$

On a boundary face $F \subset \Gamma_{\text{D}}$, with $F \subset \partial K_l$, $K_l \in \mathcal{T}$, we set $\{\!\{v\}\!\} := v$, $[\![v]\!] := v\boldsymbol{n}$, and $[\![\boldsymbol{w}]\!] := \boldsymbol{w}\cdot\boldsymbol{n}$, where $\boldsymbol{n}$ denotes the outward unit normal to the boundary $\partial\Omega$.[1].

For $v \in V_h$ we denote by $\nabla_h v$ the element-wise gradient; namely, $(\nabla_h v)|_K := \nabla(v|_K)$ for all $K \in \mathcal{T}$. Then, the symmetric interior penalty discontinuous Galerkin method reads: find $u_h \in V_h$ such that

$$B(u_h, v_h) = l(v_h), \qquad \forall v_h \in V_h, \tag{3.5}$$

with

$$B(u_h,v_h) = \int_\Omega a\nabla_h u_h\cdot\nabla_h v_h\,\mathrm{d}\mathbf{x} - \int_\Gamma \Big(\{\!\{a\nabla u_h\}\!\}\cdot[\![v_h]\!] + \{\!\{a\nabla v_h\}\!\}\cdot[\![u_h]\!]\Big)\,\mathrm{d}s + \int_\Gamma \sigma[\![u_h]\!]\cdot[\![v_h]\!]\,\mathrm{d}s, \tag{3.6}$$

and

$$l(v_h) = \int_\Omega f v\,\mathrm{d}\mathbf{x} + \int_{\Gamma_\mathrm{D}} g_\mathrm{D}(\sigma v_h - a\nabla v_h\cdot \mathrm{n})\,\mathrm{d}s + \int_{\Gamma_\mathrm{N}} g_\mathrm{N} v\,\mathrm{d}s, \tag{3.7}$$

where $\sigma : \Gamma \to \mathbb{R}$ is the so-called penalization function, which we fix with (3.9) below. The stability of the discontinuous Galerkin method is linked to the correct choice of σ. Furthermore, it can strongly affect convergence properties when high local variation of geometry or diffusion coefficients occur, as detailed in [81]. A rigorous analysis showing that the method can be made stable and optimally convergent in the hp-setting even on the rough grids considered herein is presented in [59–61].

[1]Due to the different context, we use a different notation for jumps and averages with respect to Chapter 1.

3.5.3 Convergence tests

In the following examples, we consider the Poisson model problem

$$\begin{cases} -\Delta u &= f \quad \text{in } \Omega, \\ u &= g \quad \text{on } \partial\Omega. \end{cases} \tag{3.8}$$

For $d = 2$, we consider Ω as either the unit square or the unit circle, while for $d = 3$ we take Ω as the unit cube. In all cases, the Dirichlet boundary conditions g is forced by a smooth manufactured analytical solution $u(\boldsymbol{x}) = \Pi_{i=1}^{d} \sin(\pi x_i)$. We discretize it using the polytopic interior penalty discontinuous Galerkin method described in Section 3.5, setting $a = I_{d\times d}$.

For any agglomerated polytopic element $K \in \mathcal{T}$, we define on its bounding box B_K the standard polynomial space $\mathcal{Q}^p(B_K)$ spanned by tensor-product Lagrange polynomials of degree p in each variable, denoted with $\{\phi_i\}_{i=1}^{N_p}$, with local dimension $N_p = (p+1)^d$. Since $\overline{K} \subset \overline{B_K}$, the basis on K may be defined by restricting each basis function to K. Standard Gauß-Lobatto quadrature rules of order $2p+1$ are defined on the already available sub-tessellation $\{\tau_K\}$ of K part of the underlying grid. Several other choices are possible, and we refer to the textbook [59] for a discussion about relevant implementation details. We stress that in all comparisons presented below the only difference lies in the choice of the agglomeration strategy, while the number of degrees of freedom is identical as the number of total partitions is the same. For each domain, we investigate convergence under p refinement in both the L^2-norm and the H^1-seminorm. The penalization function in (4.2) is fixed as

$$\sigma(\boldsymbol{x}) = C_\sigma \begin{cases} \frac{p^2}{h_{K_I}} & \text{on } F \in \Gamma \cap \partial\Omega, \\ \frac{p^2}{\min\{h_{K_i}, h_{K_j}\}} & \text{on } F \in \Gamma_{\text{int}}, \end{cases} \tag{3.9}$$

where in the forthcoming experiments we have set $C_\sigma = 10$. As discussed in Section 3.5, the choice of the penalization function σ can dictate the stability of the DG method. Our choice above is classical in view of the good quality of the agglomerates, but we refer to [61, 81] where this parameter plays a crucial role. Moreover, as it will be clear from the experiments, the convergence property for the present example yield almost always comparable results. We first consider the case $d = 2$. For Ω the unit square, we fix as underlying grid either the square or an unstructured quadrilateral mesh made of 93,184 elements. Identifying the mesh with the domain, we shall refer to these two cases as $\Omega = \Omega_1$ (structured square), and $\Omega = \Omega_3$ (unstructured square). Finally, for Ω the unit circle, we let $\Omega = \Omega_2$ represent the domain and its structured mesh made of 20,480 elements. These three meshes are visible in Figures 3.5a, 3.5b, and 3.5c.

In Figure 3.12 we display errors in function of the polynomial degree p for fixed grids obtained by agglomeration of the two-dimensional grids Ω_1, Ω_2 and Ω_3. We repeat the same procedure starting from the unit cube grid Ω_4 consisting of 32,768 hexahedra, which is then agglomerated to generate coarse polytopic grids $\mathcal{T}$ obtained with either METIS or R-tree and comprising only 72 polytopes. The respective findings are reported in Figure 3.13. In all cases, we observe the exponential convergence predicted by the theory [59–61]. METIS and R-tree results are always very close to each other, with the exclusion of the structure square case the perfectly square mesh produced by the R-tree yields marginally superior results. In any case our results confirm the robustness of the polygonal discontinuous Galerkin method over very general polytopic grids.

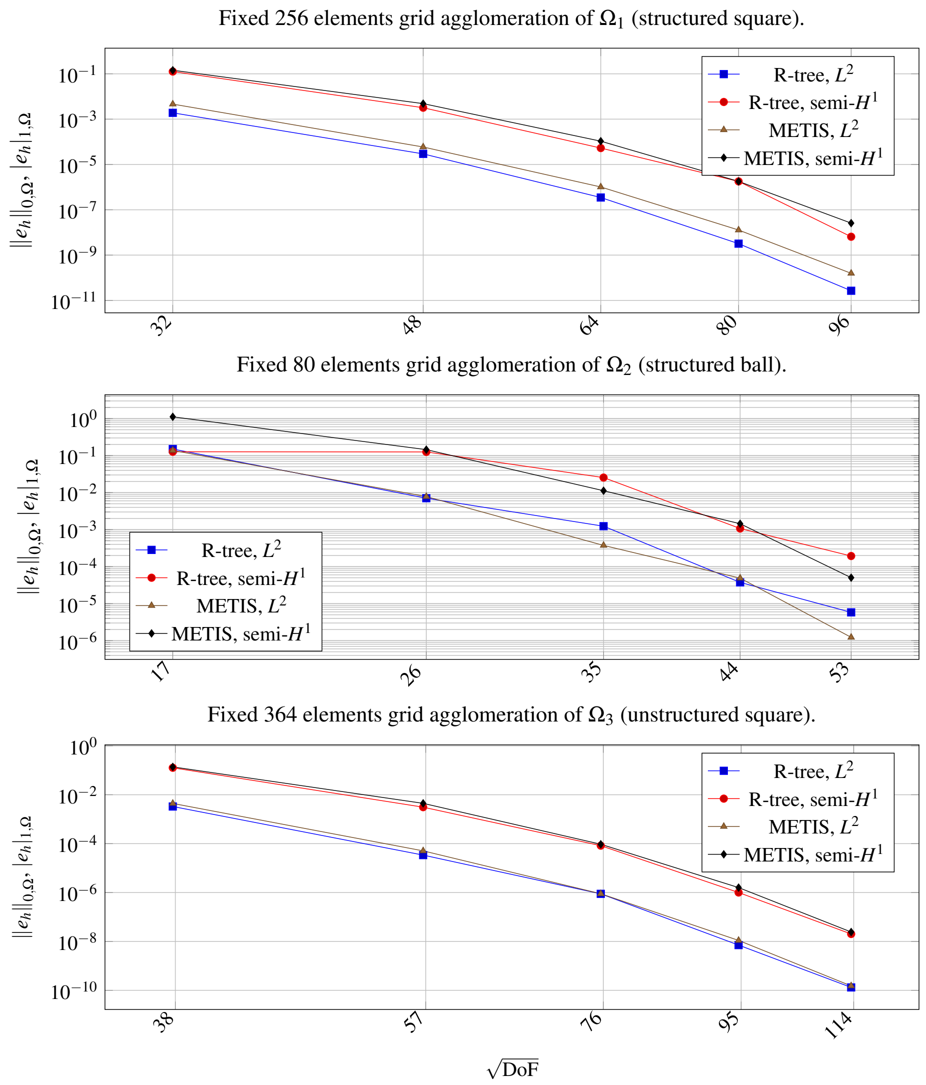


Fig. 3.12 Problem (3.8) with $d = 2$. Convergence under p-refinement for $p = 1,2,3,4,5$.

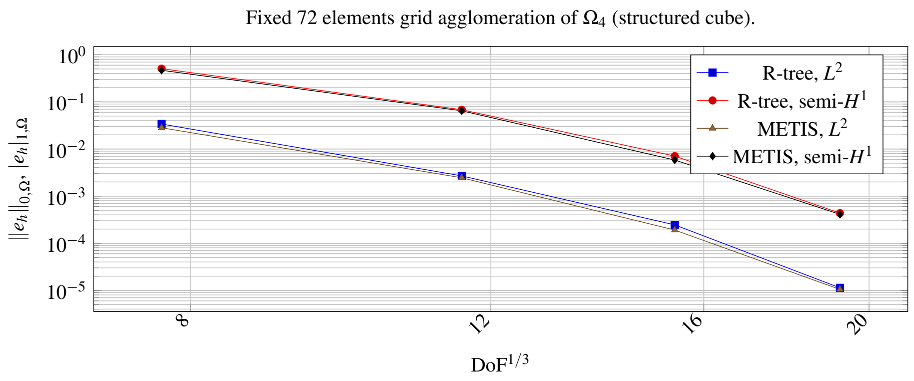


Fig. 3.13 Problem (3.8) with $d = 3$. Convergence under p-refinement for $p = 1,2,3,4$.

3D complex geometry

We finally compare the two strategies on a three-dimensional example with a *complex* geometry. We refer to complex geometries as geometries that can be meshed only with a non-negligible number of coarse cells. As an example, we employ the grid Ω_5, consisting of 45,714 hexahedra, displayed in Figure 3.5e, which once again we are identifying with the domain Ω of the problem. Such mesh comes from a real three-dimensional CAD model which has been first repaired and later meshed with hexahedra through the commercial software CUBIT. We adopt, on this fixed mesh, both the METIS and R-tree agglomeration strategies, and investigate again convergence under refinement of the polynomial degree p, using as manufactured solution $u(\boldsymbol{x}) = \Pi_{i=1}^{3} \sin(\pi \frac{x_i}{10})$. Given the simple form of the chosen solution, we expect the polytopic DG method to still produce exponential convergence under p-refinement even on the complex geometry considered here. Even if agglomeration strategies strongly decrease the number of DoF, sparse direct solvers quickly become prohibitive for non-trivial three-dimensional geometries with moderately high degrees, such as $p \geq 3$, and modest mesh sizes. We report in Table 3.3 the number of DoF in function of the polynomial degree required by the discontinuous Galerkin method with the original hexahedral model and the agglomerated version. The level of agglomeration is fixed so as to keep the number of DoF within a regime where sparse direct solvers are still effective. This allows us to solve the resulting linear system with the MUMPS solver by the TRILINOS library [113]. After generating the R-tree data structure as outlined in Section 3.2, we extract its second level, resulting in a polytopic mesh with only 731 elements, thereby reducing the size of the original problem by a factor of 64. Following the procedure of the previous experiments, we employ METIS by setting 731 as the target number of partitions. A view of some elements of the coarse mesh $\mathcal{T}$ generated using Algorithm 3, as well as a view of the solution u interpolated onto the finer mesh, are shown in Figure 3.14. The convergence history is reported in Figure 3.15; it can be appreciated how both approaches yield comparable results in terms of accuracy also for the present configuration, thus verifying once more in practice the robustness of the polytopic DG method, with METIS giving slightly more accurate results in the L^2-norm.

	Number of DoF	
p	Number of DoF (original mesh)	Number of DoF (agglomerated mesh)
1	365,712	25,425
2	1,234,278	48,366
3	2,925,696	108,492
4	5,714,250	175,020

Table 3.3 3D piston model. Total number of DoF in function of the polynomial degree p.

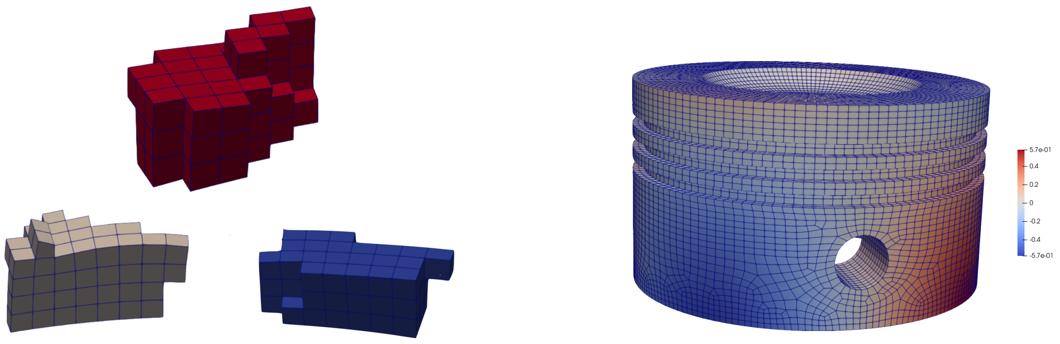


Fig. 3.14 Left: sample agglomerates generated by the R-tree algorithm. Right: view of the solution.

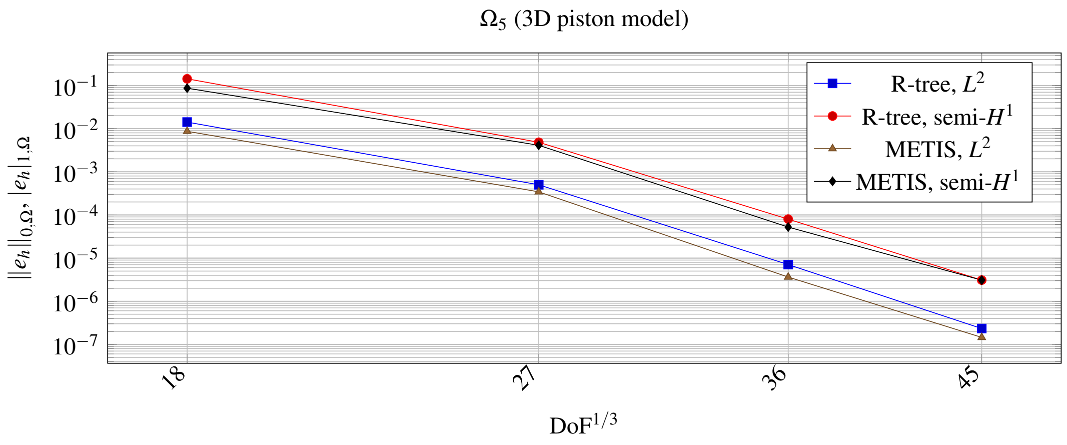


Fig. 3.15 Convergence under p-refinement for the piston test case for $p = 1,2,3,4$. Fixed polytopic grids made of 731 agglomerates.

3.5.4 Multigrid preconditioning

We finally introduce R-tree based MultiGrid (R3MG) preconditioning. The R-tree agglomeration algorithm naturally produces *nested* hierarchies of agglomerated grids. We exploit these grids and the flexibility of the discontinuous Galerkin framework to construct multigrid preconditioners. For an analysis of multigrid solvers applied to polygonal discontinuous Galerkin methods we refer to [7, 10].

We consider again the model problem (3.1) with $a = I_{d\times d}$ and manufactured solution $u = \sin(\pi x)\sin(\pi y)$ on the set of 2D grids shown at the beginning of section 3.4. We use multigrid as preconditioner for the conjugate-gradient solver (CG) [114] with one multigrid cycle per iteration as this is known to be usually more robust than using multigrid as a solver. We stress that a sequence of *nested* agglomerated grids $\{\mathcal{T}_l\}_{l=1}^M$ can be directly generated thanks to the structure of the tree. Denoting with V_h^l the finite-dimensional discontinuous space defined on $\mathcal{T}_l$, the sequence of nested grids induces a nested sequence of spaces $V_h^1 \subset V_h^2 \subset \ldots \subset V_h^M$. Thanks to this property, the intergrid transfer operators (*restriction* and *prolongation*) are more easily defined and cheaper to compute compared to a non-nested version. Indeed, the latter requires the computation of expensive L^2 projections over arbitrarily intersecting meshes.

The prolongation operator between the spaces V_{l-1} and V_l is denoted by $\mathcal{P}_{l-1}^l$ and consists of the natural injection operator, $\mathcal{P}_{l-1}^l : V_{l-1} \hookrightarrow V_l$, while as restriction operator we choose $\mathcal{R}_l^{l-1} := \left(\mathcal{P}_{l-1}^l\right)^T$. For each domain, the experiments are configured in the following way:

- The conjugate-gradient solver is run with `abstol` $= 10^{-12}$ and `reltol` $= 10^{-9}$;
- The conjugate-gradient solver is preconditioned by a single V-cycle of multigrid;
- As pre- and post-smoothers, m steps of a Chebyshev smoother of degree 3 is employed, using eigenvalue estimates computed with 20 iterations of the Lanczos iteration.
- As coarse grid solver, a direct solver is used.

The two finest levels used in each experiment correspond to the grids shown in the right columns of Figures 3.6, 3.7, and 3.8. We report the iteration counts when varying the number of levels l, the number of smoothing steps m, and the polynomial degree p. To show the effectiveness of the multigrid preconditioner, we also report the number of iterations needed by the conjugate-gradient method without preconditioning.

In Table 3.4a we report the iteration counts when varying the number of levels l and fixing the polynomial degree and the number of smoothing steps by $(p,m) = (1,2)$. We observe a roughly constant number of iterations when MG is employed as a preconditioner. The iterations with plain CG are significantly higher when no preconditioner is used. This is expected since CG is applied to the finest level of the hierarchy.

Table 3.4b shows, instead, the results obtained by varying the number of smoothing steps m from 3 to 5, again with $(p,l) = (1,3)$. As expected, the iteration counts decrease when increasing m. For completeness, we display again in the last column the plain CG iterations.

In Table 3.5 we investigate the iteration counts varying the polynomial degree p from 1 to 3, with $(l,m) = (3,5)$. In this scenario, the benefit of a preconditioner is even more evident as the number of iterations required by plain CG becomes soon order of magnitudes larger compared to the preconditioned version.

Finally, we test our approach on the three-dimensional grids Ω_4 and Ω_5, varying the number of smoothing steps m and employing again a Chebyshev smoother of degree 3. In the case of grid Ω_4, we consider the same model problem as in Section 3.5.3, having as exact solution $u(\boldsymbol{x}) = \Pi_{i=1}^{d} \sin(\pi x_i)$. In this case, the first level and second level comprise 64 and 512 agglomerated elements, respectively. For Ω_5, we solve again problem (3.8), but this time with $f = 10$ and $g = 0$, producing a more challenging setting for which the exact solution and its regularity properties are not available. For this mesh, the hierarchy of levels consists of the fine mesh, and two coarser agglomerated grids of 90 and 715 polytopes, respectively. We display in Figure 3.16 a coarse three-dimensional polytopic element and its sub-agglomerates for the piston mesh Ω_5, giving a pictorial representation of the capability of generating coarse elements even for complex geometries. We remark that the tolerances we have chosen can be considered quite strict. Indeed, many real multigrid applications with complicated geometries are generally configured with much looser tolerances, especially in time-dependent problems. As in the previous examples, we perform experiments with polynomial degrees up to 3.

The iteration counts reported in Tables 3.6a, 3.6b, and 3.6c confirms once again how increasing the number of smoothing steps m leads to lower number of iterations for every instance of the problem. Compared to plain CG, we observe that for a non-trivial geometry such as Ω_5 we get a consistently lower number of iterations for every polynomial degree p. With $p = 3$, in particular, plain CG does not achieve convergence within 10^5 iterations. This is to be expected given the moderately high polynomial degree, the complicated three-dimensional geometry, and the absence of a preconditioner.

(a) Polynomial degree $p = 1$ and $m = 2$ smoothing steps.

Iteration counts ($p = 1, m = 2$)			
Grid	l	CG+MG	CG
Ω_1	2	10	46
	3	11	
Ω_2	2	15	183
	3	16	
Ω_3	2	18	343
	3	18	

(b) Polynomial degree $p = 1$ and $l = 3$ levels.

Iteration counts ($p = 1, l = 3$)			
Grid	m	CG+MG	CG
Ω_1	3	9	46
	5	7	
Ω_2	3	13	183
	5	10	
Ω_3	3	15	343
	5	11	

Table 3.4 Iteration counts for conjugate-gradient with multigrid preconditioning (CG+MG) on the grids $\Omega_1, \Omega_2, \Omega_3$. Multigrid levels are obtained through the R-tree procedure. The number of required iterations by plain conjugate-gradient (CG) is reported in the last column of each table.

Iteration counts ($l = 3$, $m = 5$)			
Grid	p	CG+MG	CG
Ω_1	1	10	46
	2	10	105
	3	11	149
Ω_2	1	13	183
	2	14	639
	3	32	2,537
Ω_3	1	16	343
	2	16	914
	3	23	2,166

Table 3.5 Iteration counts for conjugate-gradient with multigrid preconditioning (CG+MG) on the grids $\Omega_1, \Omega_2, \Omega_3$ with $l = 3$ levels and $m = 5$ smoothing steps. Multigrid levels are obtained through the R-tree procedure. The number of required iterations by plain conjugate-gradient (CG) is reported in the last column.

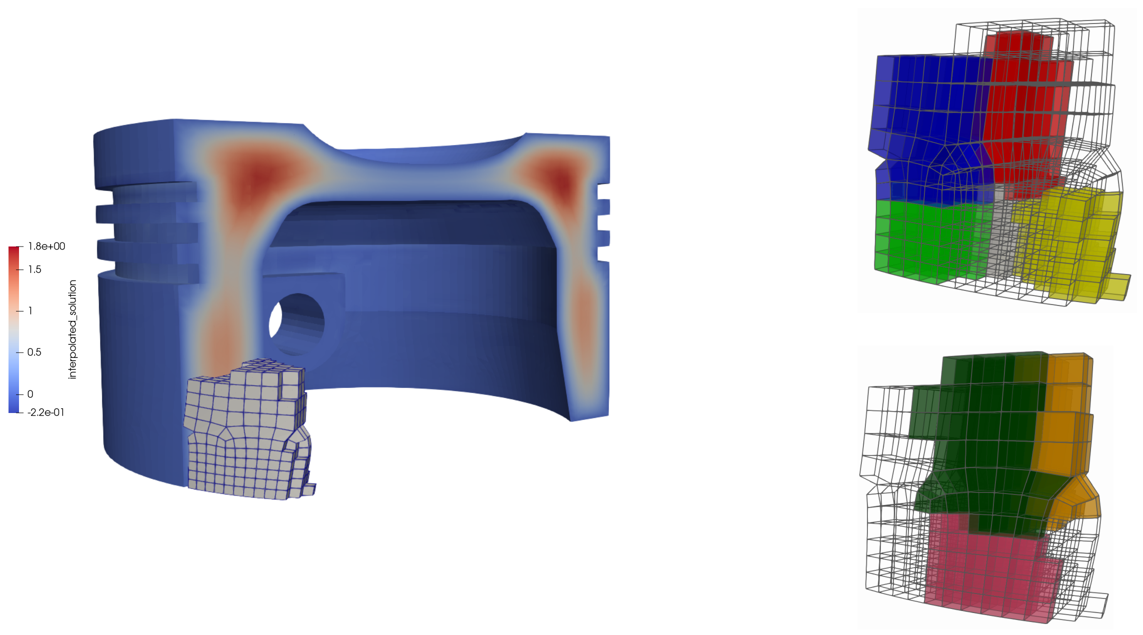


Fig. 3.16 Left: Clip of the piston geometry Ω_5 with colour plot of the solution computed with $p = 1$. In the foreground, highlighted in grey, a single element out of the 90 elements level grid. Right: detailed view of the highlighted element (wireframe) and its 8 sub-agglomerates belonging to the finer level. For better visualisation, the sub-agglomerates, identified by different colours, are displayed in either of the two plots.

(a) Polynomial degree $p = 1$ and $l = 3$ levels.

Iteration counts ($p = 1, l = 3$)			
Grid	m	CG+MG	CG
Ω_4	3	11	86
	5	8	
Ω_5	3	7	1,546
	5	6	

(b) Polynomial degree $p = 2$ and $l = 3$ levels.

Iteration counts ($p = 2, l = 3$)			
Grid	m	CG+MG	CG
Ω_4	3	10	194
	5	8	
Ω_5	3	14	26,869
	5	11	

(c) Polynomial degree $p = 3$ and $l = 3$ levels.

Iteration counts ($p = 3, l = 3$)			
Grid	m	CG+MG	CG
Ω_4	3	10	238
	5	7	
Ω_5	3	18	$> 10^5$
	5	15	

Table 3.6 Iteration counts for conjugate-gradient with multigrid preconditioning (CG+MG) on the 3D grids Ω_4, Ω_5 with polynomial degree $p = 1,2,3$ and $l = 3$ levels. Multigrid levels are obtained through the R-tree procedure. The number of required iterations by plain conjugate-gradient (CG) is reported in the last column of each table.

3.6 Implementation details

In this section, some relevant implementation choices and details will be discussed. As mentioned at the beginning of Section 3.4, building on top of a consolidated Finite Element library as DEAL.II has several advantages such as inheriting robust and optimized implementations of extensively used computational kernels, as well as a set of external library related to mesh partitioning, distributed linear algebra, and adaptive mesh refinement, to name a few. Nevertheless, the fact that all the popular and widely used Finite Element libraries are tailored to simplex or tensor-product elements poses several challenges if one wants to embed *polytopic* shapes (generated by agglomeration) in such frameworks. Our approach is generic and can be applied in a *non-intrusive* fashion to any mature Finite Element framework that supports MPI-based parallelism, advanced mesh partitioning strategies, and parallel linear algebra capabilities. The whole codebase is validated and maintained through modern CI/CD pipelines (Continuous integration and deployment), where a large number of unit tests is run at each committed change.

3.6.1 Data structures and interfaces

Given an input mesh $\Omega_h = \cup_i^N T_i$ composed by N elements, the agglomeration strategy outlined in Section 3.3 induces a different partition of Ω_h into N_A elements (agglomerates) defined as

$$\Omega_h = \cup_{i=1}^{N_A} K_i^A,$$

where each K_i^A represent an agglomerate of, say, $M_i \in \mathbb{N}$ standard-shaped cells coming from the original underlying triangulation Ω_h, that is

$$K_i^A = \cup_{j=1}^{M_i} T_j.$$

Notice that this approach *does not* depend on the partitioning strategy. The agglomeration procedure identifies each agglomerate K_i^A as a vector of mesh-like iterators (essentially pointers) which can easily be stored in a container, such as an STL vector of iterators.

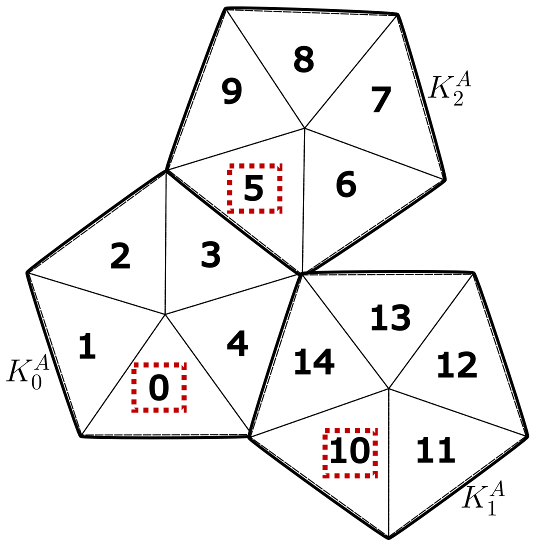


Fig. 3.17 A patch of pentagonal cells (thick black lines) is created after agglomeration of underlying triangles. Within each patch, the master cell taking care of the enumeration of DoF is highlighted with a red-dashed square. The new *global* enumeration of the three polygonal cells is $\{0,1,2\}$ and elements are indicated with K_0^A,K_1^A,K_2^A.

Since each K_i^A is a new element over which we need to define a set of DoF, we enumerate the indices of the DoF only to one of the original T_i, $i = 1,\dots,M$, which is identified as a *master cell*. The rest of the cells are called *slave cells*, and no local indices are associated to them. Such cells are in practice needed only for what concerns the geometrical and topological information of the elements, and for the generation of the quadrature rules. Hence, there is a bijective correspondence between the concept of *polytope* and *master cell*. For efficiency reasons, it is appropriate to translate the global index of a master cell (which pertains to the original triangulation Ω_h) to a unique index related to the polytopic entity. In particular, it is necessary to provide a contiguous indexing for polytopes from 0 to $N_A - 1$. For a given agglomeration strategy, each agglomerate is handed out to a routine `DefineAgglomerate()` which does the following operations:

- Identify one master cell,
- Compute bounding box $B_{K_i^A}$ of the agglomerate K_i^A,
- Index the present polytope,

which are summarized in Algorithm 5. The actual basis functions are defined on the bounding box of the agglomerate $B_{K_i^A}$ as explained at the beginning of Section 3.4. The construction of $B_{K_i^A}$

Algorithm 5: Insertion of an agglomerate given a patch of cells v.

```
   Data: Cells to be agglomerated $v = \{K_0, \dots, K_{N-1}\}$
1  Function DefineAgglomerate($v$):
2      master_cell ← $K_0$ ;                              /* Attach DoF to master */
3      slaves ← $\{K_1, \dots, K_{N-1}\}$;
4      ++n_agglomerates;
5      ComputeBoundingBox(v) ;   /* Compute bounding box of the agglomerate */
```

is particularly cheap and is performed by using all the vertices of the present agglomerate. Upon insertion of each agglomerate, we perform a translation from the global index of the master cell to a contiguous index in terms of polytopic cells. The effect of this can be visualized in Figure 3.17. The underlying triangulation Ω_h is simplicial, and the original cell index is indicated in black at the center of each triangle. After agglomeration, we obtain three pentagonal patches each identified by a master cell, which is highlighted with a red-dashed square in the figure. The *global* index of the new polytope is then defined by incrementing a counter (named `n_agglomerates` in Algorithm 5) based on the current number of polytopes, rather than using the global index of the underlying cell. With reference to the same figure, this means the polytope consisting of cells with global indices $\{10, 11, 12, 13, 14\}$ is indexed by 1 instead of 10. Such a contiguous enumeration allows for fast traversal of associative containers such as STL vectors, instead of resorting to associative containers. In this way, it is possible to store the bounding boxes over which the basis functions are defined contiguously in a vector indexed by the polytope's index, something convenient especially in the assembly phase where we need frequent access to the bounding boxes in order to map points from real to reference space $\hat{K}$. After the definition of all agglomerates, the connectivity of the resulting grid is carried out. Then, for each polytope we can query the number of faces (which in this setting cannot be defined a-priori), its neighbors, as well as the DoF associated to a neighboring element. We build our library on top of a dimension-independent programming paradigm already present in the design of DEAL.II [21] and exploiting modern C++ techniques such as template metaprogramming. To foster the overall readability of application codes and the user experience, we provide design patterns such as *iterators* and *accessors* patterns [94] which enable the iteration over mesh-like containers without exposing the user to underlying data structures by adding only an additional layer of abstraction. As a matter of fact, an application code is for all intents and purposes quite close to a classical user code in DEAL.II as it can be appreciated by comparing Listings 3.1 and 3.2. The main difference of the two snippets is the usage of the new class `AgglomerationHandler`, which implements the same interface as the original `DoFHandler` class of DEAL.II and is responsible to distribute DoF on a given mesh, along with mesh traversal.

```
// Assume dof_handler object has been set up
for (const auto &cell : dof_handler.active_cell_iterators())
{
  if (cell->is_locally_owned())
```

```
    {
      // work with cell
      const unsigned int n_faces = cell->n_faces();
      for (unsigned int f = 0; f < n_faces; ++f)
      {
        const auto &neighbor = cell->neighbor(f);
        neighbor->get_dof_indices(local_dof_indices); // fill DoF vector
      }
    }
}

```

Listing 3.1 Standard loop over local cells in DEAL.II.

```
// Assume agglomeration_handler has been set up after agglomeration
for (const auto &polytope : agglomeration_handler.polytope_iterators())
{
  if (polytope->is_locally_owned())
    {
      // work with polytope on the current processor
      const unsigned int n_faces = polytope->n_faces();
      for (unsigned int f = 0; f < n_faces; ++f)
      {
        const auto &neighbor = polytope->neighbor(f);
        neighbor->get_dof_indices(local_dof_indices); // fill DoF vector
      }
    }
}

```

Listing 3.2 Loop over local *polytopes*.

3.6.2 Memory distributed implementation

Adapting the original DEAL.II infrastructure to polytopes to a MPI-based parallelism is not immediately possible due to the level of freedom that elements now have. In order to inherit crucial properties achieved for standard shapes such as load balancing, we employ classical mesh partitioners such as PARMETIS, P4EST, and SCOTCH [125, 56, 64] for the original input triangulation Ω_h. An extensive description about the algorithmic details and data structures used in DEAL.II to handle the parallel distribution of mesh data, adaptive mesh refinement (AMR), and related data structures is described in [28]. To present our approach, we need to introduce some notation. Let $\mathbb{T}$ denote the set of cells existing in the distributed (original) mesh consisting of standard shapes. Moreover, let $\mathbb{T}^p_{loc} \subset \mathbb{T}$ be the subset of cells *owned* by processor p. By construction, this induces a partition of the elements for which it holds that

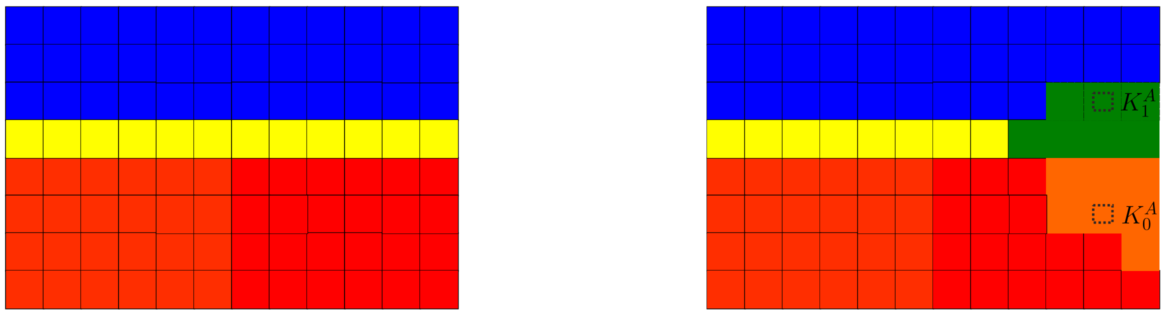
$$\bigcup_p \mathbb{T}^p_{loc} = \mathbb{T}, \qquad \mathbb{T}^q_{loc} \cap \mathbb{T}^p_{loc} = \{\emptyset\},$$

for all $q \neq p$. Finally, we indicate with $\mathbb{T}^p_{ghost}$ the set of ghost cells that processor p knows about; we have that $\mathbb{T}^p_{loc} \cap \mathbb{T}^p_{ghost} = \{\emptyset\}$ and we make the assumption that each ghost cell $T \in \mathbb{T}^p_{ghost}$ has at least one neighbor in $\mathbb{T}^p_{loc}$ via faces, lines, or vertices. Out of the above partitioning of the original triangulation, we perform the agglomeration procedure *within* each processor p by agglomerating the cells in the set $\mathbb{T}^p_{loc}$, resulting in a new *agglomerated* partition where we indicate all the agglomerated elements in the distributed mesh with $\mathbb{K}$. Hence, we can define the following sets as before:

- $\mathbb{K}^p_{loc} \subset \mathbb{K}$, the sets of *polytopes locally owned* by processor p,
- $\mathbb{K}^p_{ghost} \subset \mathbb{K}$, the sets of *ghost polytopes* that process p knows about,

with the property that $\mathbb{K}^p_{loc} \cap \mathbb{K}^p_{ghost} = \{\emptyset\}$. Furthermore, we define $\mathcal{I} = [0, N_{dof})$ as the complete index set for the N_{dof} degrees of freedom of the problem. We denote with $\mathcal{I}^p_{loc}$ the (contiguous) set of degrees of freedom *locally owned* by processor p. They are all defined on polytopes in $\mathbb{K}^p_{loc}$ and owned exclusively by such elements as no conformity is required by a DG method. We have therefore that $N = \#\mathcal{I}^p_{loc}$, $\bigcup_p \mathcal{I}^p_{loc} = \mathcal{I}$ and $\mathcal{I}^p_{loc} \cap \mathcal{I}^q_{loc} = \{\emptyset\}$ for $p \neq q$. This approach naturally inherits load balancing properties of the chosen mesh partitioner. However, due to the fact that the ghost layer is made by classical elements, we do not immediately have a clear way to define a *ghost polytope* since the master cell of a neighboring polytope belonging to another processor could not (and, in most of the cases, will not) live on the layer of ghost cells. However, the identification of a ghosted neighbor is necessary in order to compute flux terms appearing in the weak form (4.2) on faces f lying at processor's boundaries. This situation is illustrated in Figure 3.18, where the set $\mathbb{K}^0_{loc}$ (the cells owned by process 0) is shown in red, $\mathbb{K}^0_{ghost}$ in yellow, and the *remote* cells (the ones not in $\mathbb{K}^0_{loc}$ and $\mathbb{K}^0_{ghost}$), are shown in blue. Given a patch of cells $K^A_0 \in \mathbb{K}^0_{loc}$, the figure shows how the master cell of the neighboring polytope K^A_1 is fully remote. The simplest approach to build the discrete operator is to exchange with all the participating processors the metadata needed to assemble the matrix by using a many-to-many (and blocking) exchange by calling `MPI_Allgather()`, which however has quadratic complexity in the number of ranks. To avoid this, we determine the communication pattern when building the connectivity of the agglomerated grid. During this setup phase, when we are on polygons at boundaries of the working processor, we identify the rank of the neighboring processor which is meant to receive the data by querying it to ghost cells, and we prepare metadata such as ghosted bounding boxes, indices of the ghosted degrees of freedom, and jacobians at quadrature points to be sent. In this way, we can initiate a some-to-some communication pattern, leading to a ghost exchange in a non-blocking manner by using calls to standard MPI non-blocking routines `MPI_Isend()` and `MPI_Irecv()`.

With reference to Snippet 3.2, the variable `neighbor` in Line 10 can identify both an element in $\mathbb{K}^p_{loc}$ or in $\mathbb{K}^p_{ghost}$. For what concerns the handling of parallel linear algebra, no additional change is required as we rely on TRILINOS and PETSC, offering a large variety of solvers including Krylov-space methods and commonly used preconditioners available through the packages HYPRE [84] and ML [95].



(a) Original (idealized) partition among processors.

(b) A pair of *topologically* neighboring agglomerations living on different partitions.

Fig. 3.18 Left: view from process 0 of the locally owned (red), ghosted (yellow), and remote (blue) quadrilateral cells. Right: on the same grid, a patch of cells (orange) is created after agglomeration of some quadrilateral elements on process 0, while the dark green agglomerate has been created on process 1. The dashed squares flag the respective master cells. For instance, the three cells on top of K_1^A are fully remote cells for which process 0 has no information.

One of the advantages of the application of agglomeration techniques within multilevel frameworks is that the *explicit* construction of the sequence of grids is not required. As noted in Section 3.6.2, since the hierarchy is generated on top of a unique (distributed) grid, and agglomerates are conveniently generated within each local region, the coherence between different partitions at different levels of the hierarchy is *by construction* optimal. With reference to the performance metrics presented in Section 2.5 of Chapter 2, this implies very high values for the *vertical efficiency* metric.

3.6.3 Building the sparse operator

Using a Discontinuous Galerkin method, and in particular the formulation (4.2), one has to integrate the flux terms over the faces between neighboring cells. This implies that the DoF on each element also couple to the DoF on other cells connected to the current one by a common face. These couplings must be added to the sparsity pattern of the problem, which is stored through a compressed row storage (CSR) format and is used as a basis to build (also in the memory-distributed case) the resulting sparse matrix. The elements of a sparsity pattern, corresponding to the places where resulting sparse matrix objects can store nonzero entries, are stored row-by-row. The generation of the sparsity pattern associated to a DG discretization (as well as the one for continuous Lagrangian elements) in DEAL.II is a well-established core component of the library. To adapt this to our context, we mimic the classical approach: each processor loops over its own polytopes $\mathbb{K}_{loc}^p$ and simulates which elements of the matrix would be written if we were assembling the system matrix of the problem from local contributions. Due to the generality of our framework and the fact that the coupling now happens between neighboring *polytopes*, we manually add the face couplings through the mesh iterators described above. We notice that each processor will only store matrix rows indexed by $\mathcal{I}_{loc}^p$. Since the index sets are a partition of the global index space, we automatically have a non-overlapping distribution of rows among the available processors. The algorithmic realization of this procedure is sketched in Algorithm 6. In practice, we also exploit the symmetry of the operator and avoid visiting the same faces multiple times, adding redundant information to the same entry. It is worth mentioning that in the distributed case, where each processor is owning only a portion of the triangulation, the neighboring polytope can be ghosted, as it is the case with classical elements. Finally, we need to exchange entries with other processors to make sure that each processor knows all the elements of the rows of a matrix it stores and that may eventually be written to. This is achieved by calling the `distribute_sparsity_pattern()` function in Line 9, which takes care of setting up the communication pattern between the participating ranks. It is crucial to remark that since hanging nodes are not present in a DG setting, we do not have to take care of *constraints* during the setup of the sparsity pattern as well as the system matrix.

Algorithm 6: Creation of sparsity pattern for a polygonal DG discretization.

```
    Data: Reference to an empty sparsity pattern $S$
 1  Function MakeAgglomerateSparsityPattern($S$):
 2      locally_owned_rows ← extract_local_dofs();
 3      for $K$ in LocallyOwnedPolytopes do
 4          dof_indices ← get_dof_indices($K$) ;                /* Get DoF of $K$ */
 5          add_entries_local_to_global(dof_indices,S) ; /* Volumetric terms */
 6          for $f$ in $faces$ do
 7              $K_f$ ← neighbor($K,f$);
 8              dof_indices_neigh ← get_dof_indices($K_f$);
 9              add_entries_local_to_global(dof_indices_neigh,S)
10      distribute_sparsity_pattern(S,locally_owned_rows,communicator);
```

3.6.4 Benchmarks

In the following, we will present some test cases that are intended to verify the scalability of the algorithms and data structures discussed above. The numerical simulations presented here have been performed on the GALILEO100 supercomputer at CINECA center (10 nodes with 2xCPU Intel CascadeLake 8260, 2.4 GHz,384GB RAM)[2]. The scalability of the DEAL.II core has been already demonstrated in [28]. The forthcoming tests are part of POLYDEAL, a C++ library developed and maintained by the author of this thesis, and have been compiled with the `intel − oneapi − mpi/2021.10.0` compiler and `−O3` optimization level. For the sake of showing the scalability of the algorithms only, we will test the three-dimensional Poisson problem $-\Delta u = f$, on the unit cube $[0,1]^3$. The right-hand-side f and Dirichlet boundary conditions are chosen so that the exact solution is $u(\boldsymbol{x}) = \Pi_{i=1}^{3} \sin(\pi x_i)$. Albeit the geometry is very simple, we stress that after mesh partitioning among processors (here

[2]For the technical specifications, see https://www.hpc.cineca.it/systems/hardware/galileo100/. Retrieved on August 8, 2024.

through the graph partitioner PARMETIS), the locally owned regions are *not* preserving the original geometric structure of the original mesh. The equation is discretized as in Section 3.5.3, using polynomial degree $p = 2$, and solving the resulting linear systems with the conjugate gradient method preconditioned with an incomplete LU factorization provided by TRILINOS, with absolute tolerance 10^{-8}. We note that, thanks to the great reduction in terms of elements provided by the agglomeration strategy, we are able to exploit a direct solver. We measure the wall clock time for the following dominant parts of the program:

- *Init matrix:* Exchange between processors which non-locally owned matrix entries they will write to in order to populate the necessary sparsity pattern for the global matrix. Copy intermediate data structures used to collect these entries into a more compact one and allocate memory for the system matrix.
- *Sparsity Pattern:* Determine the locations of non-zero matrix entries as described in 3.6.3.
- *Assembly:* Assembling the contributions by looping over locally owned polytopes $\mathbb{K}^p_{loc}$ to the global system matrix and right hand side vector. This includes the transfer of matrix and vector elements locally computed but stored on other processors.
- *Linear Solver:* Solving the linear system with the conjugate gradient method preconditioned by ILU factorization.

Test 1: $N_A = $ **5 000**

We globally refine 6 times the unit cube Ω, resulting in a mesh Ω_h consisting of $N = 2^{6\times3} =$ 262 144 hexahedra. Such mesh is distributed among processors, and cells among each partition are agglomerated in such a way that the global number of polytopes is $N_A = 5\,000$, gaining a reduction factor of about 52 compared to the size of the original matrix. We increase the number of processors progressively from 10 to 256 and report the wall-clock times in Figure 3.19. We observe almost perfect scalability for all components. In contrast to what is seen in all scalability results for FEM codes, for which the solution phase is usually largely dominant, the polytopic DG pipeline is dominated by the assembly phase. This is due to the fact the *fine* underlying mesh has to be traversed, while the great reduction in the number of unknowns clearly simplifies the solver part.

Test 2: $N_A = $ **70 000**

We globally refine 7 times the unit cube Ω, resulting in a mesh Ω_h consisting of $N = 2^{7\times3} = 2\,097\,152$ hexahedra. We agglomerate cells among each partition in such a way that the global number of polytopes is $N_A = 70\,000$, gaining a reduction of a factor of about 40. As before, we increase the number of processors from 10 to 256 and plot the result in Figure 3.20. We observe the same pattern of Figure 3.20, with the complexity shifting from the linear solver to the assembly phase. Since we are

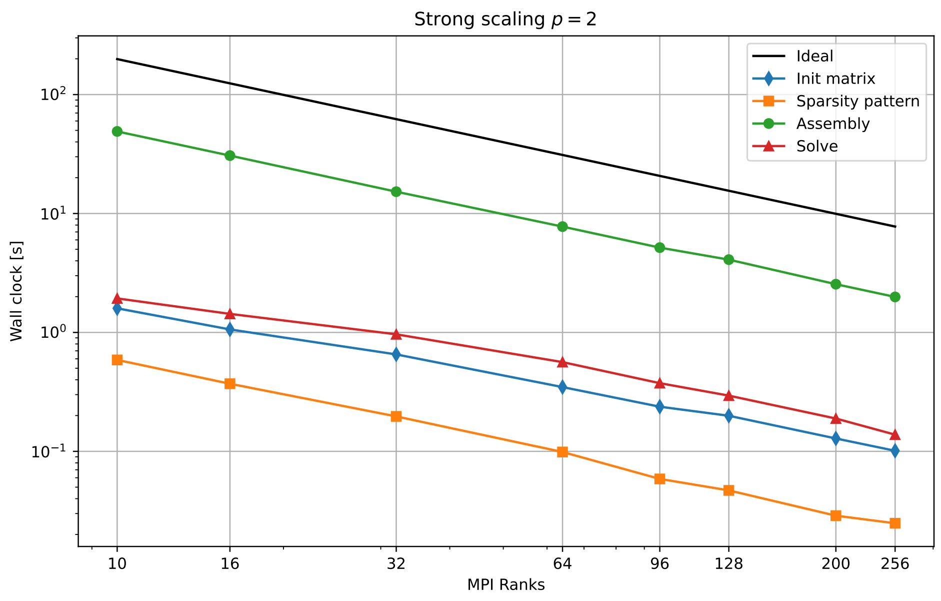


Fig. 3.19 Strong scaling experiment with $p = 2$ on a *polytopic* mesh consisting of $N_A = 5\,000$ agglomerates built on top of a fine hexahedral mesh Ω_h of 262 144 elements.

solving a three-dimensional problem with a non-trivial number of agglomerates and degree $p = 2$, the solution phase timing are relatively higher compared to the previous case, but still below the assembly phase timings as before.

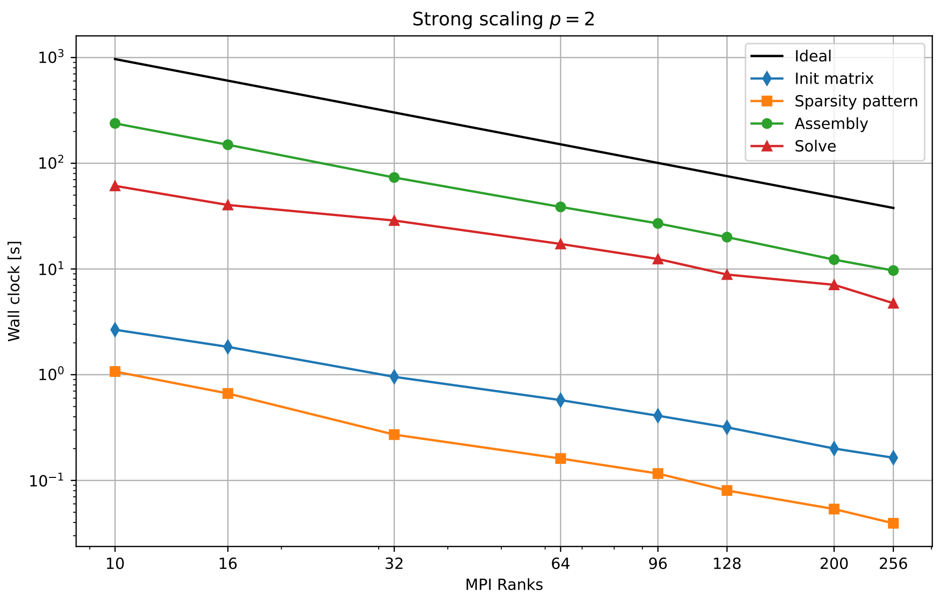


Fig. 3.20 Strong scaling experiment with $p = 2$ on a *polytopic* mesh consisting of $N_A = 70\,000$ agglomerates built on top of a fine hexahedral mesh Ω_h of 2 097 152 elements.

3.7 Conclusions

We presented and validated a novel method to generate polytopic grids based on a spatial structure that can be easily built by agglomeration of an underlying fine mesh. The method relies on the so-called R-tree data structure and generates sequences of nested agglomerated grids in a quite fast and scalable way compared to standard approaches such as METIS. We showed its robustness and effectiveness on a sequence of two- and three-dimensional benchmarks and non-trivial geometries. The numerical experiments for R-tree based Geometric Multigrid (R3MG) show good convergence properties typical of multilevel solvers, also in cases where a hierarchy of grids was not available to start with. The time to assemble the linear system is the most computationally intensive operation. However, this cost can be dramatically reduced, for instance, by employing the so-called quadrature free algorithm as shown in [11] and more recently in [155], which allows the computation of volume integrals without resorting to the sub-tessellation of the background mesh Ω. A proof of the R3MG convergence is given in the next chapter. Also, we consider in Chapter 5 an alternative solution that exploits so-called *inherited* approaches to build coarser operators, aiming to more complex geometries and time-dependent problems.

Chapter 4

Convergence analysis for multigrid

In the previous chapter, a new way to generate a hierarchy of agglomerated grids was presented and validated on classical second order elliptic problems. This chapter contains some preliminary results about the convergence of our multilevel strategy. To do so, we build on the analysis performed by *Antonietti et. al* in [10], where the convergence properties of two-level and W-cycle multigrid solvers for system of equations arising from the *hp*-version Symmetric Interior Penalty Discontinuous Galerkin discretizations were established. In particular, we adapt such analysis to the properties of the agglomerates obtained with the R-tree approach by relying on estimates and error bounds developed in the general setting of almost arbitrarily shaped elements in [61].

Contents

4.1 Literature review . . . 113

4.2 Model problem . . . 114

4.2.1 Grid assumptions . . . 115

4.3 Inverse estimates . . . 118

4.3.1 R-tree agglomeration and p-coverability . . . 120

4.3.2 A priori error bounds . . . 121

4.4 Convergence analysis for the multigrid algorithm . . . 123

4.4.1 Convergence of the two-level method . . . 128

4.4.2 Convergence of the W-cycle algorithm . . . 129

4.5 Conclusions . . . 131

4.1 Literature review

Recently, there has been significant interest in developing multilevel schemes the system of equations stemming from the discretization of differential problems by polytopic methods. Multilevel developments for HHO discretizations can be found in [73, 74, 76], while for VEM we refer to [18, 12, 154].

The many advantages offered by Discontinuous Galerkin methods in handling general meshes, hanging nodes, and the limited neighboring communication through numerical fluxes, perfectly matches many desirable characteristics needed for the efficient development of fast multilevel solvers. The possibility to perform h-coarsening by agglomeration [20, 19, 87] leads to high flexibility in the definition of the coarse meshes. Indeed, starting from a fine grid, a coarse mesh can be generated clustering together a number of mesh elements. Such versatility allows a natural definition of the associated subspaces since no inter-element continuity is required. This property overcomes the usual difficulties encountered in the construction of agglomeration multigrid schemes in conforming frameworks, where a proper definition of the conforming subspaces and ad-hoc intergrid operators must be constructed. In the DG setting, we refer to [46, 150, 10, 62]. In [7], a non-nested multigrid scheme for DG spaces defined on overlapping and independent polygonal grids was developed. In [10], extending the theoretical analysis developed in [9], the convergence of a two-level scheme and W-cycle multigrid method on meshes consisting of polytopic elements was analyzed, exploiting trace and inverse inequalities for polytopic elements developed in [59]. More recently, new estimates for *essentially arbitrarily-shaped* elements were provided in [61], generalizing what initially proved in [59], and allowing a larger degree of generality for the shapes of agglomerated elements. In this regard, the agglomeration strategy developed in the previous chapter fits seamlessly into the framework and allows the convergence analysis of the h-multigrid method presented, which was validated through a series of numerical experiments varying polynomial degree, geometry, and number of smoothing steps in Section 3.5.3. In this chapter, we closely follow the structure of [10], generalizing it to the case when polytopic elements are stemming from our agglomeration routine, hence removing the dependence on the ratio between the size of the fine grid and the size of the agglomerates.

4.2 Model problem

Let $\Omega \subset \mathbb{R}^d$, $d = 2,3$, be a convex polytopic domain with Lipschitz boundary. We consider the Poisson problem with homogeneous Dirichlet boundary conditions: find $u \in V$ such that

$$-\nabla \cdot (a \nabla u) = f \quad \text{in } \Omega, \tag{4.1}$$

where $V := H^2(\Omega) \cap H_0^1(\Omega)$, $a = \{a_{ij}\}_{i,j=1}^{d}$ symmetric with $a_{ij} \in L_\infty(\Omega)$, and $f \in L^2(\Omega)$ a given source term. We close the problem by prescribing homogeneous Dirichlet boundary conditions on $\partial\Omega$.

We denote with $\mathcal{T}$ a shape-regular and quasi-uniform subdivision of the domain Ω and consider a sequence of partitions $\{\mathcal{T}_j\}_{j=1}^{J}$ of Ω, with $\mathcal{T}_J \equiv \mathcal{T}$, into disjoint polytopic elements K (with possibly curved Lipschitz boundaries) such that $\bar{\Omega} = \bigcup_{K \in \mathcal{T}_j} \bar{K}$. The mesh skeleton $\Gamma := \cup_{K \in \mathcal{T}} \partial K$ is subdivided into its internal part $\Gamma^{\text{int}} := \Gamma \setminus \partial\Omega$ and its boundary part $\partial\Omega$. Furthermore, we indicate the mesh size of $\mathcal{T}_j$ with $h_j = \max_{K \in \mathcal{T}_j} h_K$. As in previous chapters, on each partition $\mathcal{T}_j$, $j \in \{1, \dots, J\}$ we associate a Discontinuous Galerkin space V_j defined as

$$V_j = \left\{ v \in L^2(\Omega) \colon v_{h|K} \in \mathcal{P}_p(K), K \in \mathcal{T}_j \right\},$$

with $\mathcal{P}_p(K)$ denoting the space of polynomials of degree p in each element. The classical h-multigrid framework is obtained by keeping fixed the polynomial degree p on a sequence of *nested* and quasi-uniform partitions $\{\mathcal{T}\}_{j=1}^J$. We require that the coarse level $\mathcal{T}_{j-1}$ can be obtained by agglomeration from the finer level $\mathcal{T}_j$, $j = \{2,\dots,J\}$, so that the resulting sequence of nested finite element spaces is nested, e.g. $V_1 \subset V_2 \subset \dots \subset V_J$.
On each level j, $j \in \{1,\dots,J\}$, we have the following weak for problem 4.1: find $u \in V_j$ such that

$$\mathcal{A}_j(u,v) = l(v), \qquad \forall v \in V_j, \tag{4.2}$$

where $\mathcal{A}_j(u,v) \colon V_j \times V_j \to \mathbb{R}$ defined as

$$\mathcal{A}_j(u,v) = \int_\Omega a\nabla u \cdot \nabla v \, d\mathbf{x} - \int_\Gamma \Big(\{\{a\nabla u\}\} \cdot [\![v]\!] + \{\{a\nabla v\}\} \cdot [\![u]\!] \Big) \, ds + \int_\Gamma \sigma [\![u]\!] \cdot [\![v]\!] \, ds, \tag{4.3}$$

and

$$l(v) = \int_\Omega f v \, d\mathbf{x}, \tag{4.4}$$

with jump $[\![\cdot]\!]$ and average $\{\{\cdot\}\}$ operators defined as in Section 3.5.2. The precise definition of the penalization function σ will be given in next sections. In particular, we are interested in solving Problem 4.1 on the finest level J with a geometric multigrid method. For a given level j, we define the DG norm for a $v \in V_j$ as follows

$$\|v\|_{DG,j}^2 := \left\|\sqrt{a}\nabla v\right\|_{\mathcal{T}}^2 + \left\|\sqrt{\sigma}[\![v]\!]\right\|_{\Gamma}^2. \tag{4.5}$$

4.2.1 Grid assumptions

In this section we present some assumptions and definition that will be needed throughout the analysis. We start by quoting the general framework developed in [61] which provides trace and inverse inequalities for general shaped elements, cf. [61, Assumption 4.1, 4.3].

Assumption 4.2.1. For each element $K \in \mathcal{T}$, we assume that K is a Lipschitz domain, and that we can subdivide ∂K into mutually exclusive subsets $\{F_i\}_{i=1}^{n_K}$ satisfying the following property: there exist respective sub-elements $K_{F_i} \equiv K_{F_i}(\mathbf{x}_i^0) \subset K$ with d planar faces meeting at one vertex $\mathbf{x}_i^0 \in K$, with $F_i \subset \partial K_{F_i}$, such that, for $i = 1,\dots,n_K$,

(a) K_{F_i} is star-shaped with respect to $\mathbf{x}_i^0$. We refer to Figure 4.1(left) for an illustration for $d = 2$;

(b) $\mathbf{m}_i(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) > 0$ for $\mathbf{m}_i(\mathbf{x}) := \mathbf{x} - \mathbf{x}_i^0$, $\mathbf{x} \in K_{F_i}$, and $\mathbf{n}(\mathbf{x})$ the respective unit outward normal vector to F_i at $\mathbf{x} \in F_i$. (We refer to Figure 4.1(right) for an illustration for $d = 2$.)

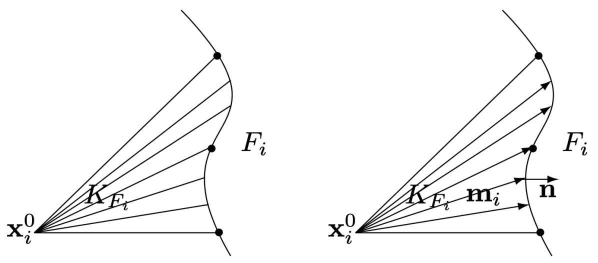


Fig. 4.1 Elements $K \in \mathcal{T}^{tr}$ are assumed to satisfy Assumption 4.2.1 (a) (left) and (b) (right); $\cdot$ denotes a vertex.

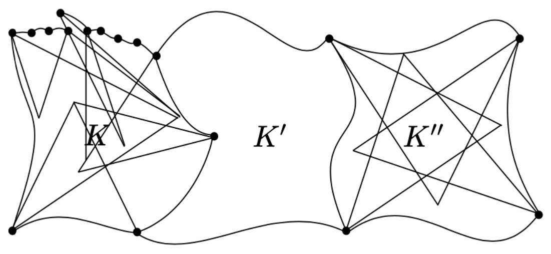


Fig. 4.2 Curved elements K, K'' with, respectively, 8 and 4 sub-elements satisfying Assumption 4.2.1.

Remark 4.2.1. As noted in [61], the sub-domains $\{F_i\}_{i=1}^{n_K}$ are *not* required to coincide with the faces of the element K: n_K are not required to be uniformly bounded. This provides a very general setting. In the case of polytopic meshes obtained by agglomeration of an underlying standard mesh, there exists a global constant $c_{sh} > 0$

$$\mathbf{m}_i(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) \geq c_{sh} h_{K_{F_i}}; \tag{4.6}$$

Assumption 4.2.2. We assume that the boundary ∂K of each element $K \in \mathcal{T}$ is the union of a finite (but arbitrarily large) number of closed C^1 surfaces.

Assumption 4.2.1 is sufficient for the proof of the trace estimate, while requiring Assumptions 4.2.1 and 4.2.2 is sufficient in view of deriving optimal a priori error bounds.

Assumption 4.2.3. With reference to Figure 4.3, we indicate with H_K the diameter of the bounding box B_K for an agglomerate K element constructed by patching together fine elements. Hinging on the agglomeration algorithm developed in Chapter 3, we assume that the R-tree based strategy implies

$$|K| \approx H_K^d. \tag{4.7}$$

Finally, we require a bounded variation in terms of mesh sizes between subsequent levels.

Assumption 4.2.4. Let h_j, $j \in \{1,\ldots,J\}$, denote the mesh-size of the j-th partition. For any $j \in \{1,\ldots,J\}$, the partitions $\{\mathcal{T}_j\}_{j=1}^J$ satisfy

$$h_{j-1} \lesssim h_j \leq h_{j-1}. \tag{4.8}$$

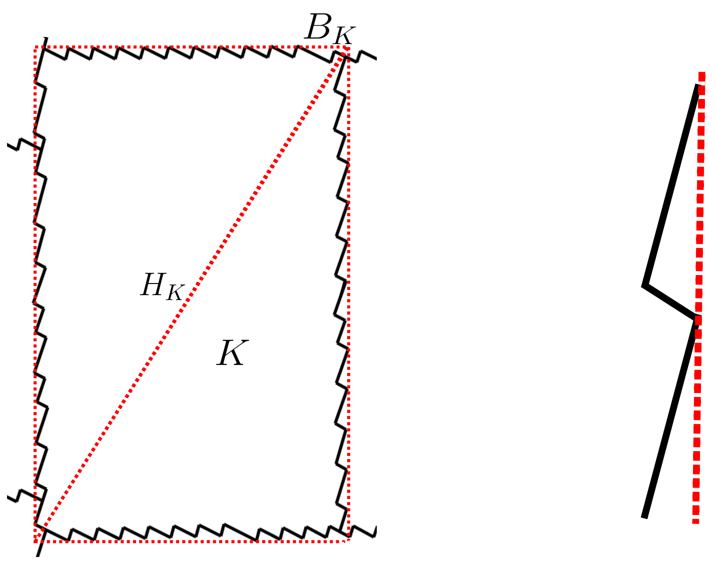


Fig. 4.3 One agglomerate, denoted by K, and its associated bounding box B_K with diameter H_K (indicated with the diagonal, red, dashed line). Right: zoom on the right boundary, showing the fine mesh edges.

Remark 4.2.2. The last two assumptions are not restrictive in our framework, thanks to the geometric properties that the R*-tree data structure tries to fulfill when performing agglomeration on top of the fine mesh $\mathcal{T}_J$, as underpinned by numerical evidences and the quality metrics reported experimentally in previous chapter. Moreover, the hierarchical subdivision of the space done by the tree and the fact that each agglomerate is split into 2^d sub-agglomerates, perfectly fits the definition of bounded variation requirement. Such a property can already be appreciated in Figure 3.5e and will be showed to hold in practice for other real geometries in next chapter.

Remark 4.2.3. We note that in case of domains with complex (curvilinear) boundaries, Assumption 4.2.3 may not hold since the axis-aligned bounding box associated to a boundary agglomerate may not be able to fit the profile of the underlying mesh. This will be more and more the case as more levels of agglomeration are performed. From this viewpoint, the relation (4.7) represents a cheap criterion to evaluate the applicability of the theory.

The next assumption is needed for the stability of the intergrid transfer operators used within the multigrid method, which we quote from [10].

Assumption 4.2.5. For any $F \in \mathcal{F}_j \cap \mathcal{F}_{j-1}$, $j \in \{2\dots,J\}$, let $K_j^{\pm}$ and $K_{j-1}^{\pm}$ the neighboring elements sharing face F in $\mathcal{T}_j$ and $\mathcal{T}_{j-1}$, respectively. We assume that there exists a $\Theta > 0$ such that

$$1 < \frac{h_{K_{j-1}^{\pm}}}{h_{K_j^{\pm}}} \leq \Theta \qquad \forall F \in \mathcal{F}_j \cap \mathcal{F}_{j-1}. \tag{4.9}$$

Remark 4.2.4. Motivated by the fact that with the R-tree procedure agglomerates from belonging to subsequent levels have comparable ratios, we consider the above assumption not restrictive and satisfied by construction in our setting.

4.3 Inverse estimates

Next, we introduce a some definitions needed to state precise trace and inverse estimates for very general polytopic elements, cf. [61, Section 4].

Definition 3. *Let $\hat{F}^0 := [0,1]^{d-1} \subset \mathbb{R}^d$ and $\phi : \hat{F}^0 \to \mathbb{R}$ a Lipschitz continuous scalar function. A* reference generalized prism *is a domain $\hat{K} \equiv \hat{K}_\phi \subset \mathbb{R}^d$ given by*

$$\hat{K} \equiv \hat{K}_\phi := \{\mathbf{x} \in \mathbb{R}^d : 0 \le x_i \le 1, i = 1, \dots, d-1, 0 \le x_d \le \phi(x_1, \dots, x_{d-1})\},$$

with the properties: 1) $[0,1]^d \subset \hat{K}$, and 2) the straight line connecting any pair $(\mathbf{x}, \mathbf{y}) \in \hat{F}^0 \times \hat{F}$ lies fully in $\hat{K}$.

$$\hat{F} := \{\mathbf{x} \in \mathbb{R}^d : 0 \le x_i \le 1, i = 1, \dots, d-1, x_d = \phi(x_1, \dots, x_{d-1})\}.$$

Also, we set $\hat{\rho} := \sup\{\rho \ge 1 : \hat{F}^0 \times [0,\rho] \subset \hat{K}\}$ and $\hat{r} := \lfloor \max_{\mathbf{x} \in \hat{F}^0} \phi(\mathbf{x}) \rfloor + 1$. □

We refer to Figure 4.4 for an illustration.

Remark 4.3.1. The 'height' $\hat{r}$ is a measure of anisotropy of the reference generalized prism. Note that we can take $\hat{\rho} = 1$ without essential loss of generality. Indeed, if $\hat{\rho} > 1$, the change of variables $x_d \to x_d/\hat{\rho}$ implies a modification of the Lipschitz function ϕ, reducing its Lipschitz constant. Star-shapedness with respect to $\hat{F}^0$ is also ensured (cf. [61]).

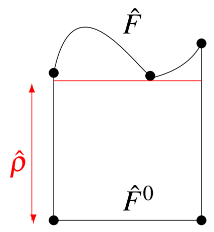


Fig. 4.4 A reference generalized prism $\hat{K}$ for $d = 2$.

Definition 4. *Given a mesh $\mathcal{T}$, we define a* covering *$\mathcal{T}^\sharp = \{\mathcal{K}\}$ of $\mathcal{T}$ to be a set of open shape-regular d–simplices $\mathcal{K}$, such that for each $K \in \mathcal{T}$, there exists a $\mathcal{K} \in \mathcal{T}^\sharp$ with $K \subset \mathcal{K}$. For a given $\mathcal{T}^\sharp$, we define the* covering domain *$\bar{\Omega}_\sharp := \cup_{\mathcal{K} \in \mathcal{T}^\sharp} \bar{\mathcal{K}}$.*

Assumption 4.3.1. For a given mesh $\mathcal{T}$, we postulate the existence of a covering $\mathcal{T}^\sharp$, and of a (global) constant $\mathcal{O}_\Omega \in \mathbb{N}$, independent of the mesh parameters, such that

$$\max_{K \in \mathcal{T}} \mathrm{card}\Big\{K' \in \mathcal{T} : K' \cap \mathcal{K} \neq \emptyset,\ \mathcal{K} \in \mathcal{T}^\sharp \ \text{ such that } \ K \subset \mathcal{K}\Big\} \le \mathcal{O}_\Omega.$$

For such $\mathcal{T}^\sharp$, we further assume that $h_{\mathcal{K}} := \operatorname{diam}(\mathcal{K}) \leq C_{\text{diam}} h_K$, for all pairs $K \in \mathcal{T}$, $\mathcal{K} \in \mathcal{T}^\sharp$, with $K \subset \mathcal{K}$, for a (global) constant $C_{\text{diam}} > 0$, uniformly with respect to the mesh size h_K.

The next definition, introduced in [61, Definition 4.18], relaxes an earlier coverability assumption (cf. [59, Definition 10]) which assumed the ability to properly cover general-shaped elements by unions of simplices of similar size.

Definition 5. *An element $K \in \mathcal{T}$ is said to be p-*coverable *with respect to $p \in \mathbb{N}$, if there exists a set of $m_K \in \mathbb{N}$ generalized prisms $\hat{K}_j$ and corresponding affine maps Φ_j, such that the mapped generalized prisms $\overline{K}_j := \Phi_j(\hat{K}_j)$, $j = 1, \ldots, m_K$, form a, possibly overlapping, covering of K with the additional properties*

$$\operatorname{dist}(\partial\overline{\mathrm{K}}_{\mathrm{j}}, \mathrm{K}) \leq \mathfrak{h}_{\overline{K}_j}(8p)^{-2} \tag{4.10}$$

and

$$|\overline{K}_j| \geq c_{as}|K|, \tag{4.11}$$

for all $j = 1, \ldots, m_K$, where $\mathfrak{h}_{\overline{K}_j} := \sup_{\mathbf{x}\in\hat{F}^0} |\Phi_j(\ell_{\mathbf{x},j})|$ and c_{as} is a positive constant, independent of K and of $\mathcal{T}$, with $\operatorname{dist}(\partial\overline{K}_j, K) := \sup_{\mathbf{x}\in\partial\overline{K}} \inf_{\mathbf{y}\in K} |\mathbf{x}-\mathbf{y}|$ the one-sided Hausdorff distance of $\partial\overline{K}_j$ from K, and $\ell_{\mathbf{x},j} := \hat{K}_j \cap \{\mathbf{x} + \alpha \mathrm{e}_d : \alpha \in \mathbb{R}\}$.

We quote hereafter the trace and inverse estimates on general elements, and we refer the reader to [61, Lemmata 4.21, 4.23] for their detailed proofs.

Lemma 4.3.1. *Let $K \in \mathcal{T}$ Lipschitz satisfying Assumption 4.2.1. Then, for each $v \in \mathcal{P}_p(K)$, we have the inverse inequality*

$$\|v\|_{F_i}^2 \leq \mathcal{C}_{\text{INV}}(p, K, F_i) \frac{(p+1)(p+d)|F_i|}{|K|} \|v\|_K^2, \tag{4.12}$$

with

$$\mathcal{C}_{\text{INV}}(p, K, F_i) := \left\{ \begin{array}{ll} \min\left\{\mathcal{C}_{\text{reg}}(K, F_i), 2c_{as}^{-1} 32^d p^{2(d-1)}\right\}, & K \ p\text{-coverable}, \\ \mathcal{C}_{\text{reg}}(K, F_i), & \text{otherwise}, \end{array} \right\} \tag{4.13}$$

with $\mathcal{C}_{\text{reg}}(K, F_i) := |K| / \big(|F_i| \sup_{\mathbf{x}_i^0 \in K} \min_{\mathbf{x}\in F_i} (\mathbf{m}_i \cdot \mathbf{n})\big)$, and $c_{as} > 0$ as in Def. 5.

Lemma 4.3.2. *Let $K \in \mathcal{T}$ satisfy Assumptions 4.2.1 and 4.2.2. Then, for each $v \in \mathcal{P}_p(K)$, the inverse estimate*

$$\|\nabla v\|_K^2 \leq \mathcal{C}_{\text{INV}}^B(p, K) \frac{p^4}{\rho_K^2} \|v\|_K^2, \tag{4.14}$$

holds, with ρ_ω denoting the radius of the largest inscribed circle of a domain $\omega \subset \mathbb{R}^d$, and

$$\mathcal{C}_{\text{INV}}^B(p, K) := \left\{ \begin{array}{ll} \min\left\{\rho_{\text{cov}}(K), \rho_{\text{p-cov}}(p, K)\right\}, & \text{if } K \ p\text{-coverable} \\ \rho_{\text{cov}}(K), & \text{otherwise}, \end{array} \right. \tag{4.15}$$

with

$$\rho_{\mathrm{cov}}(K) := \sum_{j=1}^{m_K} C_{\mathrm{inv}}^{B}(d,\hat{r}_j)\left(\frac{\hat{r}_j \rho_K}{\rho_{\overline{K}_j}}\right)^2, \tag{4.16}$$

and

$$\rho_{\mathrm{p-cov}}(p,K) := c_{as}^{-1}(32)^d (p-1)^{2d} \max_{1\le \ell \le m_K} C_{\mathrm{inv}}^{B}(d,\hat{r}_\ell)\left(\frac{\hat{r}_\ell \rho_K}{\rho_{\overline{K}_\ell}}\right)^2, \tag{4.17}$$

for $\overline{K}_\ell$, $\ell = 1,\dots,m_K$, cover of K consisting of affinely mapped generalised prisms.

Remark 4.3.2. We note that the key attribute of $\mathcal{C}_{\mathrm{INV}}(p,K,F_i)$ from Lemma 4.3.1 is that it remains bounded for degenerating $|F_i|$, allowing for the estimate (4.12) to remain finite as $|F_i| \to 0$.

4.3.1 R-tree agglomeration and p-coverability

The inherent geometrical nature of the agglomeration algorithm presented in Chapter 3 nicely fits in this framework and allows to explicitly characterize the notion of p-coverability. Recall that starting from the fine mesh $\mathcal{T}_J$, each subsequent mesh level $\mathcal{T}_j$, $j \in \{1,\dots,J\}$, is obtained by successive agglomeration dictated by the R-tree. Let $\kappa \in \mathcal{T}_J$ denote the fine mesh element and c_{sh} the shape regularity constant of $\mathcal{T}_J$, such that $\rho_\kappa \ge c_{sh} h_\kappa$. We shall also assume that each face of each $\kappa \in \mathcal{T}_J$ is shape-regular with respect to the same constant c_{sh}. Now, recall that each $K \in \mathcal{T}_j$, $j \in \{1,\dots,J-1\}$, is the union of all $\kappa \in \mathcal{T}_J$ such that $\kappa \in B_K$. Hence, each K is tightly close to its bounding box B_K and, with the notation of Assumption 4.2.3, it follows we can rewrite the p-coverability condition (4.10) as

$$\max_{\kappa \in \mathcal{T}_J, \kappa \subset K} h_\kappa \le \frac{H_K}{c_{sh}}(8p)^{-2}, \tag{4.18}$$

We now specialize the constant $\mathcal{C}_{\mathrm{INV}}(p,K,F_i)$ in Lemma 4.3.1 by exploiting the new characterization of p-coverability. To do so, we consider each case in (4.13) separately. If the agglomerate $K \in \mathcal{T}_j$, is not p-coverable, i.e. (4.18) does not hold, then we have

$$\mathcal{C}_{\mathrm{INV}}(p,K,F_i) = \mathcal{C}_{\mathrm{reg}}(K,F_i) = |K|/\big(|F_i| \sup_{\mathbf{x}_i^0 \in K} \min_{\mathbf{x}\in F_i}(\mathbf{m}_i \cdot \mathbf{n})\big) \le c \sup_{\kappa \in \mathcal{T}_J, \partial\kappa \cap F_i \neq \emptyset} \frac{H_K^d}{h_\kappa^{d-1} h_\kappa} \le c_1 p^{2d}.$$

Conversely, if the agglomerate satisfies (4.18), we have, using Assumption 4.2.1,

$$\mathcal{C}_{\mathrm{INV}}(p,K,F_i) = \min\left\{\mathcal{C}_{\mathrm{reg}}(K,F_i), c p^{2(d-1)}\right\} = \min\left\{c_1 p^{2d}, c_2 p^{2(d-1)}\right\}$$

where both constants c_1, c_2 are independent of h, H, and p, but depend on the shape regularity of the agglomerates. We assume from now on that the agglomeration procedure based on the R-tree implies the above Assumptions, and hence such characterization is guaranteed.

4.3.2 A priori error bounds

f In the following, we recall error bounds for the DG and $L^2(\Omega)$ norms for the interior penalty DG method adapting from [61]. To this aim, we first need to introduce a precise form of the penalization term σ, which we quote from [61, Lemma 5.2].

Lemma 4.3.3. *Let $\mathcal{T}$ be a mesh satisfying Assumption 4.2.1. We define the discontinuity-penalization function $\sigma : \Gamma \to \mathbb{R}$ for every interface $F \in \Gamma$, $F = \partial K \cap \partial K'$, by*

$$\sigma|_F := 2 \max_{\mathcal{K} \in \{K,K'\}} \left\{ \max_{i \in I_F^{\mathcal{K}}} \{\mathcal{C}_{\mathrm{INV}}(p_{\mathcal{K}}, \mathcal{K}, F_i^{\mathcal{K}})|F_i^{\mathcal{K}}|\} \frac{\bar{a}_K (p+1)(p+d)}{|\mathcal{K}|} \right\}; \tag{4.19}$$

when $F \subset \partial\Omega$ we set $K = K'$.

With this definition of σ we can present the next theorem, adapted from [61].

Theorem 4.3.1. *Let $\mathcal{T} = \{K\}$ be a subdivision of $\Omega \subset \mathbb{R}^d$, consisting of general curved elements satisfying Assumptions 4.2.1, 4.2.2 and 4.3.1. Let also $\mathcal{T}^\sharp = \{\mathcal{K}\}$ an associated covering of $\mathcal{T}$ consisting of shape-regular simplices as per Definition 4. Assume that $u \in H^1(\Omega)$, the exact solution to (4.1), is such that $u|_K \in H^{l_K}(K)$, $l_K > 1 + d/2$, for each $K \in \mathcal{T}$. Let $u_h \in V_h$, with $p \geq 1$, $K \in \mathcal{T}$, be the solution of (4.2), with σ as in (4.19). Then, we have*

$$\|u - u_h\|_{DG}^2 \leq C \sum_{K \in \mathcal{T}} \frac{h_K^{2s_K}}{p^{2l_K}} \left(\mathcal{G}_K(F, \mathcal{C}_{\mathrm{INV}}, \mathcal{C}_{ap}, p)\right) \|\mathfrak{E}u\|_{H^{l_K}(\mathcal{K})}^2,$$

with $s_K = \min\{p+1, l_K\}$, and

$$\begin{aligned}
\mathcal{G}_K(F, \mathcal{C}_{\mathrm{INV}}, \mathcal{C}_{ap}, p_K) = {}& \bar{a}_K^2 p^3 h_K^{-d-2} \sum_{F \subset \partial K \cap \Gamma} \sigma^{-1} \max_{i \in I_F^K} \{\mathcal{C}_{ap}(p, K, F_i^K)|F_i^K|\} \\
&+ \bar{a}_K^2 p^4 h_K^{-2} |K|^{-1} \sum_{F \subset \partial K \cap \Gamma} \sigma^{-1} \max_{i \in I_F^K} \{\mathcal{C}_{\mathrm{INV}}(p, K, F_i^K)|F_i^K|\} \\
&+ h_K^{-d} p \sum_{F \subset \partial K \cap \Gamma} \sigma \max_{i \in I_F^K} \{\mathcal{C}_{ap}(p, K, F_i^K)|F_i^K|\},
\end{aligned}$$

and $C > 0$ a constant which does not depend on discretization parameters. Here $\mathfrak{E} : H^s(\Omega) \mapsto H^s(\mathbb{R}^d)$, $s \in \mathbb{N}_0$, is the classical linear extension operator presented by Stein [164].

For mesh sequences obtained by agglomeration such that condition (4.18) holds, the expression $\mathcal{G}_K(F, \mathcal{C}_{\mathrm{INV}}, \mathcal{C}_{ap}, p_K)$ can be bounded as follows

$$\mathcal{G}_K(F, \mathcal{C}_{\mathrm{INV}}, \mathcal{C}_{ap}, p_K) \leq \tilde{C} C(p) h_K^{-2} \qquad K \in \mathcal{T}, \tag{4.20}$$

where $\tilde{C}$ is a positive constant and does not depend on the discretization parameters. Thus, in this case we are able to derive the following bounds which are the basis of the forthcoming multigrid analysis.

Theorem 4.3.2. *Assume that Assumptions 4.2.1, 4.2.2, 4.2.3, and 4.3.1 hold. Let $u_j \in V_j$, $j \in \{1,\dots,J\}$, be the DG solution of problem (4.2) with penalty function δ as in (4.19). Assume that the exact solution u is such that $u \in H^l(\Omega)$, $l > 1+\frac{d}{2}$. Then the following bounds hold*

$$\left\|u-u_j\right\|_{DG} \le G^j \sqrt{C(p)}\frac{h_j^{s-1}}{p^{l-\frac{3}{2}}}\|u\|_{H^l(\Omega)}, \tag{4.21}$$

$$\left\|u-u_j\right\|_{L^2(\Omega)} \le C^j_{L^2} \sqrt{C(p)}\frac{h_j^{s}}{p^{l-1}}\|u\|_{H^l(\Omega)}, \tag{4.22}$$

where $s=\min\{p+1,l\}$ and the constants G^j and $C^j_{L^2}$ are independent of the discretization parameters.

Proof. The first bound follows immediately by using $\mathcal{G}_K$ as in (4.20). The second bound is proven by a duality argument. Let $j\in\{1,\dots,J\}$ be fixed and let $w\in V$ be the only function satisfying

$$\mathcal{A}_j(v,w)=\int_\Omega (u-u_j)v \qquad \forall v\in V.$$

For any $w_I \in V_j$, exploiting Galerkin orthogonality we have

$$\left\|u-u_j\right\|^2_{L^2(\Omega)} = \mathcal{A}_j(u-u_j, w-w_I).$$

Since $V_j \not\subset V$, we can not directly control the last term with the DG-norm. To this aim, we add and subtract $u_I \in V_j$ and write

$$\left\|u-u_j\right\|^2_{L^2(\Omega)} = \mathcal{A}_j(u-u_I+u_I-u_j, w-w_I).$$

We highlight the steps to bound the problematic term containing the product of jumps and averages, namely

$$\int_\Gamma \Big(\{\{a\nabla(u-u_I)\}\}\cdot[\![w-w_I]\!]\Big)\,\mathrm{d}s + \int_\Gamma \Big(\{\{a\nabla(u_I-u_j)\}\}\cdot[\![w-w_I]\!]\Big)\,\mathrm{d} = \mathcal{I}_1+\mathcal{I}_2$$

As homogeneous Dirichlet boundary conditions are considered, by multiplying and dividing by $\sqrt{\sigma}$ we have

$$\mathcal{I}_1 \le \frac{1}{2}\sum_{F\in\Gamma^{\text{int}}}\int_F \sigma^{-\frac{1}{2}}\Big(\nabla(u-u_I)^+ + \nabla(u-u_I)^-\Big)[\![\sigma^{\frac{1}{2}}(w-w_I)]\!],$$

and upon application of trace inverse inequality and the penalization function σ defined in (4.19), we have, for any $F\in\Gamma$, with $F\subset\partial K$, for a $K\in\mathcal{T}$,

$$\left\|\sigma^{-\frac{1}{2}}\nabla(u-u_I)\right\|_F \le C\Big(h_j^{-1}\left\|\sigma^{-\frac{1}{2}}\nabla(u-u_I)\right\|_{L^2(K)} + h_j|\sigma^{\frac{1}{2}}\nabla(u-u_I)|_{1,K}\Big) \tag{4.23}$$

$$\le C\Big(C_\sigma(p)p^{-2}\|\nabla(u-u_I)\| + h_j^2 C_\sigma(p)p^{-2}|\nabla(u-u_I)|_{1,K}\Big), \tag{4.24}$$

where $C_\sigma(p)$ contains the polynomial dependence of the penalization function in 4.19. Choosing now, on each element $K \in \mathcal{T}$, $u_I = \Pi_j u$, where $\Pi_j \colon H^l(\mathcal{K}) \to \mathcal{P}^p(\mathcal{K})$ is the operator defined in [61, Theorem 4.31], we obtain upon using Assumption 4.3.1

$$\left\|\sigma^{-\frac{1}{2}}\nabla(u-u_I)\right\|_F \leq C_\sigma(p)\left(\frac{h_j^{2(s-1)}}{p^{2l-2+2}} + \frac{h_j^{2(s-1)}}{p^{2l-4+2}}\right)\|u\|^2_{H^l(\mathcal{K})}\,. \tag{4.25}$$

From this, after summing over all elements we conclude

$$\mathcal{I}_1 \leq \sqrt{C_\sigma(p)}\frac{h_j^{s-1}}{p^{l-1}}\|w-w_I\|_{DG,j}\|u\|_{H^l(\Omega)}\,.$$

For $\mathcal{I}_2$, we divide and multiply again by $\sqrt{\sigma}$, use the trace-inverse multiplicative inequality in [61, Lemma 4.21], and exploit the L^2-stability of Π_j, obtaining

$$\mathcal{I}_2 \leq \sqrt{C_\sigma(p)}|u-u_j|_{1,\Omega}\|w-w_I\|_{DG,j}\,.$$

Choosing again $w_I = \Pi_j w$, and employing elliptic regularity we have

$$\|w-w_I\|_{DG} \leq C^j\frac{h_j}{p^{\frac{1}{2}}}\|w\|_{H^2(\Omega)} \lesssim C^j\frac{h_j}{p^{\frac{1}{2}}}\left\|u-u_j\right\|_{L^2(\Omega)},$$

where C^j is independent of the discretization parameters. All in all, we have

$$\mathcal{I}_1+\mathcal{I}_2 \leq \sqrt{C_\sigma(p)}\left(\frac{h_j^{s-1}}{p^{l-1}} + |u-u_j|_{1,\Omega}\right)\frac{h_j}{p^{\frac{1}{2}}}\left\|u-u_j\right\|_{L^2(\Omega)}\,.$$

From this, by using standard arguments one obtains

$$\left\|u-u_j\right\|_{L^2(\Omega)} \leq \sqrt{C_\sigma(p)}\left(\frac{h_j^{s-1}}{p^{l-1}} + \frac{h^{s-1}}{p^{l-\frac{3}{2}}}\right)\frac{h}{p^{\frac{1}{2}}}\|u\|_{H^l(\Omega)} \leq C(p)\frac{h_j^s}{p^{l-1}}\|u\|_{H^l(\Omega)},$$

where $C(p)$ contains the dependence on $C_\sigma(p)$.

□

4.4 Convergence analysis for the multigrid algorithm

We closely follow the framework developed in [10, 9], cf. also [49]. The two main ingredients we need to specify are the *intergrid transfers* operators and the smoothing iterations. Moreover, assume that mesh levels are obtained by agglomeration such that p-coverability holds under condition (4.18). The *prolongation* operator is denoted by $\mathcal{P}_{j-1}^j \colon V_{j-1} \to V_j$, for $j \in \{2,\dots,J\}$, while the restriction operator $\mathcal{R}_j^{j-1} \colon V_j \to V_{j-1}$ is its the adjoint with respect to the $L^2(\Omega)$-inner product $(\cdot,\cdot)$ and is thus defined by

$$\left(\mathcal{P}_{j-1}^{j}v,w\right)=\left(v,\mathcal{R}_{j}^{j-1}w\right)\qquad\forall v\in V_{j-1},w\in V_{j}.\tag{4.26}$$

The smoother on level j is defined to be a Richardson iteration, whose iteration matrix is

$$B_j=\Lambda_j I_j,$$

where I_j is the identity and $\Lambda_j\in\mathbb{R}$ is a bound for the spectral radius of the operator $A_j\colon V_j\to V_j$ defined by

$$(A_j u,v)=\mathcal{A}_j(u,v)\qquad\forall u,v\in V_j,\quad j\in\{1,\dots,J\}.\tag{4.27}$$

Next, we define the operator $P_j^{j-1}\colon V_j\to V_{j-1}$ as

$$\mathcal{A}_{j-1}\left(P_j^{j-1}v,w\right)=\mathcal{A}_j\left(v,\mathcal{P}_{j-1}^{j}w\right)\qquad\forall v\in V_j,w\in V_{j-1}.\tag{4.28}$$

For any j, $j\in\{1,\dots,J\}$, we define the following norms based on $\mathcal{A}_j$

$$|||v|||_{s,j}:=\sqrt{\left(A_j^s v,v\right)_j}\qquad\forall s\in\mathbb{N}\cup\{0\},v\in V_j.$$

In particular, it holds

$$|||v|||_{1,j}^2=\left(A_j v,v\right)_j=\mathcal{A}_j(v,v),\qquad |||v|||_{0,j}=(v,v)_j=\|v\|_{L^2(\Omega)}^2.$$

In the following theorem we prove an upper bound on the largest eigenvalue of $\mathcal{A}_j$ by employing the general estimates of [61], adapted in Section 4.3.2 to our agglomerates.

Theorem 4.4.1. *Assume that Assumptions 4.2.1, 4.2.2, 4.2.3 and 4.2.4 hold. For any $v\in V_j$, $j\in\{1,\dots,J\}$, we have that*

$$\mathcal{A}_j(u,u)\le C_{\mathtt{eig}}^{j}(p)\left(\frac{p^4}{\rho_j^2}+\frac{(p+1)^2(p+d)^2}{\rho_j^2}\right)\|v\|_{0,\Omega}^2,\tag{4.29}$$

where $C_{\mathtt{eig}}^{j}(p):=\max\{C(p),\mathcal{C}_{\mathrm{INV}}^{j}(p)\}$, with $\mathcal{C}_{\mathrm{INV}}^{B,j}(p):=\max_{K\in\mathcal{T}_j}\mathcal{C}_{\mathrm{INV}}^{B}(p,K)$.

Proof. We estimate separately the two contributions involved in the definition of the DG norm. The first addendum involving the H^1-seminorm is bounded by using Lemma 4.3.2 and Assumption 4.2.4:

$$\sum_{K\in\mathcal{T}_j}|u|_{1,K}^2\le\sum_{K\in\mathcal{T}_j}\mathcal{C}_{\mathrm{INV}}^{B}(p,K)\frac{p^4}{\rho_K^2}\|u\|_{0,K}^2\le\max_{K\in\mathcal{T}_j}\mathcal{C}_{\mathrm{INV}}^{B}(p,K)\sum_{K\in\mathcal{T}_j}\frac{p^4}{\rho_K^2}\|u\|_{0,K}^2\le\mathcal{C}_{\mathrm{INV}}^{B,j}\frac{p^4}{\rho_j^2}\|u\|_{0,\Omega}^2,$$

being $\mathcal{C}_{\mathrm{INV}}^{B,j}(p)=\max_{K\in\mathcal{T}_j}\mathcal{C}_{\mathrm{INV}}^{B}(p,K)$. Jump terms across facets $F\in\mathcal{F}_j$ are bounded thanks to the inverse inequality from Lemma 4.3.1, and the element-wise definition of the penalty σ_K:

$$\sum_{F\in\mathcal{F}_j}\left\|\sigma_j^{\frac{1}{2}}[\![u]\!]\right\|_{0,F}^2 \leq \sum_{K\in\mathcal{T}_j}\left\|\sigma_j^{\frac{1}{2}}[\![u]\!]\right\|_{0,\partial K}^2 \leq \sum_{K\in\mathcal{T}_j}\sum_{F\in\partial K}\mathcal{C}_{\mathrm{INV}}(K,F,p)\frac{(p+1)^2(p+d)^2|F|}{\rho_K^2|K|}\|u\|_{0,K}^2.$$

Using the inverse estimate (4.12) and the characterization of C_{INV} implies

$$\sum_{F\in\mathcal{F}_j}\left\|\sigma_j^{\frac{1}{2}}[\![u]\!]\right\|_{0,F}^2 \leq C(p)\frac{(p+1)^2(p+d)^2}{\rho_j^2}\|u\|_{0,\Omega}^2.$$

The statement follows by summing the above bounds and collecting the constants. □

Remark 4.4.1. In the special case for which we do not need to resort to the p-coverability setting for all $K\in\mathcal{T}_j$, $j\in\{1,\dots,J\}$, the constants C_{INV}^B and C_{INV} involved in Lemmata 4.3.1 and 4.3.2 do not depend on the polynomial degree p. Therefore, C_{eig} is independent of p. In particular, we retrieve in Theorem 4.4.1 the same behavior of $\mathcal{O}\left(\frac{p^4}{h_K^2}\right)$ as found in [10, Theorem 7].

We consider the two-level cycle sketched in Algorithm 7 to solve the problem (posed on the fine level) $A_J u_J = f_J$ with $f_J\in V_J$.

Algorithm 7: Two-level method to solve $A_J x = f_J$

```
   Data: $x_0$ initial guess, $m_1, m_2$ smoothing steps parameters.
1  for $i = 1,\dots,m_1$ do
2      $x^{(i)} \leftarrow x^{(i-1)} + B_J^{-1}(f_J - A_J x^{(i-1)})$;
3  $r_{J-1} \leftarrow \texttt{Restrictor}(f_J - A_J x^{(m_1)})$;
4  $e_{J-1} \leftarrow \texttt{CoarseGridSolver}(A_{J-1}, r_{J-1})$;
5  $x^{(m_1+1)} \leftarrow x^{m_1} + \texttt{Prolongator}(e_{J-1})$;
6  for $i = m_1+2,\dots,m_1+m_2+1$ do
7      $x^{(i)} \leftarrow x^{(i-1)} + B_J^{-1}(f_J - A_J x^{(i-1)})$;
```

In order to perform a convergence analysis of the two-level method, we need to estimate the error propagation operator associated to the multigrid scheme, defined by

$$\mathbb{E}_{m_1,m_2}^{\texttt{2lvl}} := G_J^{m_2}\left(\mathrm{Id}_J - \mathcal{P}_{J-1}^J P_J^{J-1}\right)G_J^{m_1}, \tag{4.30}$$

with $G_J = \mathrm{Id}_J - B_J^{-1}A_J$. To this aim, we follow the classical approach of studying separately the *smoothing property* and the *approximation property* associated to the two-level operator. By Theorem 4.4.1, we can bound Λ_j, $j\in\{1,\dots,J\}$ as follows:

$$\Lambda_j \leq C_{\texttt{eig}}^j(p)\left(\frac{p^4}{\rho_j^2}+\frac{(p+1)^2(p+d)^2}{\rho_j^2}\right), \tag{4.31}$$

with $C_{\texttt{eig}}^j(p)$ as in Theorem 4.4.1.

Lemma 4.4.1 (Smoothing property). *Let Assumptions 4.2.1, 4.2.2, 4.2.3 and 4.2.4 hold. Then, for any $v \in V_j$, $j \in \{1,\dots,J\}$, we have*

$$\left|\left|\left| G_j^m v \right|\right|\right|_{s,j} \le \left[C_{\texttt{eig}}^j(p) \left(\frac{p^4}{\rho_j^2} + \frac{(p+1)^2(p+d)^2}{\rho_j^2} \right) \right]^{\frac{s-t}{2}} (1+m)^{\frac{t-s}{2}} |||v|||_{t,j}^2, \tag{4.32}$$

for $0 \le t < s \le 2$, $m \in \mathbb{N} \setminus \{0\}$.

Proof. By considering the eigenvalue problem associated to the level operator A_j, namely

$$A_l \varphi_i^j = \lambda_i \varphi_i^j, \tag{4.33}$$

with operator eigenvalues $0 < \lambda_1 \le \lambda_2 \le \dots \le \lambda_n$ and associated eigenbasis $\{\varphi_i^j\}_{i=1}^{n_j}$ of V_j, we can write any $v \in V_j$ as

$$v = \sum_{i=1}^{n_l} v_i \varphi_i^j, \qquad v_i \in \mathbb{R}. \tag{4.34}$$

Based on the previous identity, G_j^m can be written as:

$$G_j^m = \left(I_j - \frac{1}{\Lambda_j} A_j \right)^m v = \sum_{i=1}^{n_j} \left(1 - \frac{\lambda_i}{\Lambda_j} \right)^m v_i \varphi_i^j.$$

Combining the above identity and estimate (4.31) gives the thesis:

$$\begin{aligned} |||G_j^m v|||_{s,j}^2 &= h_l^d \sum_{i=1}^{n_j} \left(1 - \frac{\lambda_i}{\Lambda_j} \right)^{2m} v_i^2 \lambda_i^s = \Lambda_j^{s-t} \left\{ h_j^d \sum_{i=1}^{n_j} \left(1 - \frac{\lambda_i}{\Lambda_j} \right)^{2m} \frac{\lambda_i^{s-t}}{\Lambda_j^{s-t}} v_i^2 \lambda_i^t \right\} \\ &\le \Lambda_j^{s-t} \max_{x \in [0,1]} \{ x^{s-t} (1-x)^{2m} \} |||v|||_{t,j}^2 \le \left[C_{\texttt{eig}}^j(p) \left(\frac{p^4}{\rho_j^2} + \frac{(p+1)^2(p+d)^2}{\rho_j^2} \right) \right]^{s-t} (1+m)^{t-s} |||v|||_{t,j}^2. \end{aligned}$$

□

Lemma 4.4.2 (Approximation property). *Let Assumptions 4.2.1, 4.2.2, 4.2.3, and 4.3.1 hold. Then for any $v \in V_j$, $j \in \{1,\dots,J\}$, we have*

$$\left|\left|\left| \left(Id_j - \mathcal{P}_{j-1}^j P_j^{j-1} \right) v \right|\right|\right|_{0,j} \lesssim \sqrt{C(p)} \left(C_{L^2}^j + C_{L^2}^{j-1} \right) \frac{h_j^2}{p} |||v|||_{2,j}. \tag{4.35}$$

Proof. For a given $v \in V_j$, we have by definition of the norm

$$\left|\left|\left| \left(\mathrm{Id}_j - \mathcal{P}_{j-1}^j P_j^{j-1} \right) v \right|\right|\right|_{0,j} = \sup_{0 \ne \phi \in L^2(\Omega)} \frac{\int_\Omega \phi (I_k - \mathcal{P}_{j-1}^j P_j^{j-1}) v}{\|\phi\|_{L^2(\Omega)}}. \tag{4.36}$$

We consider the following problem

$$\int_\Omega \nabla\eta\cdot\nabla v\mathrm{dx} = \int_\Omega \phi\cdot\nabla v\mathrm{dx} \qquad \forall v\in V,$$

for $\phi\in L^2(\Omega)$, and write $\eta_j\in V_j$ and $\eta_{j-1}\in V_{j-1}$ for the DG solution of

$$\mathcal{A}_j(\eta_j,v) = \int_\Omega \phi v \qquad \forall v\in V_j, \tag{4.37}$$

$$\mathcal{A}_{j-1}(\eta_{j-1},v) = \int_\Omega \phi v \qquad \forall v\in V_{j-1}. \tag{4.38}$$

Using now Theorem 4.3.2 we have

$$\|\eta-\eta_j\|_{L^2(\Omega)} \lesssim C^j_{L^2}\sqrt{C(p)}\frac{h^2_{j-1}}{p}\|\phi\|_{L^2(\Omega)}, \tag{4.39}$$

$$\|\eta-\eta_{j-1}\|_{L^2(\Omega)} \lesssim C^{j-1}_{L^2}\sqrt{C(p)}\frac{h^2_{j-1}}{p}\|\phi\|_{L^2(\Omega)}, \tag{4.40}$$

By the definition of P_j^{j-1} we obtain the following chain of equalities for a $w\in V_{j-1}$

$$\mathcal{A}_{j-1}(P_j^{j-1}\eta_j,w) = \mathcal{A}_j(\eta_j,\mathcal{P}^j_{j-1}w) = \mathcal{A}_j(\eta_j,w) = \int_\Omega \phi w\mathrm{dx} = \mathcal{A}_{\mathrm{j-1}}(\eta_{\mathrm{j-1}},\mathrm{w}),$$

which implies

$$\eta_{j-1} = P_j^{j-1}\eta_j. \tag{4.41}$$

Using (4.37), the error estimates (4.39), the equality (4.41), and the definition of P_j^{j-1} we get

$$\begin{aligned}\int_\Omega \phi(\mathrm{Id}_j-\mathcal{P}^j_{j-1}P_j^{j-1})v\mathrm{dx} &= \mathcal{A}_j(\eta_j,v)-\mathcal{A}_j(\eta_j,\mathcal{P}^j_{j-1}P_j^{j-1}v)\\ &= \mathcal{A}_j(\eta_j,v)-\mathcal{A}_{j-1}(P_j^{j-1}\eta_j,P_j^{j-1}v)\\ &= \mathcal{A}_j(\eta_j,v)-\mathcal{A}_{j-1}(\eta_{j-1},P_j^{j-1}v)\\ &= \mathcal{A}_j(\eta_j-\mathcal{P}^j_{j-1}\eta_{j-1},v)\\ &\lesssim |||\eta_j-\eta_{j-1}|||_{0,j}|||v|||_{2,j}\\ &\lesssim \left(\|\eta_j-\eta\|_{L^2(\Omega)}+\|\eta_{j-1}-\eta\|_{L^2(\Omega)}\right)|||v|||_{2,j}\\ &\lesssim \sqrt{C(p)}\left(C^j_{L^2}+C^{j-1}_{L^2}\right)\frac{h^2_j}{p}|||\phi|||_{L^2(\Omega)}|||v|||_{2,j},\end{aligned} \tag{4.42}$$

where the first inequality exploits the generalized Cauchy-Schwarz inequality for elements of V_j developed in [10, Lemma 4.1] (cf. also [48, Lemma 6.2.10]). Replacing (4.42) in (4.36) yields the result. □

4.4.1 Convergence of the two-level method

Lemmata 4.4.1 and 4.4.2 allow the convergence analysis of a two-level method with the error propagation operator defined in (4.30).

Theorem 4.4.2. *Assume that Assumptions 4.2.1, 4.2.2, 4.2.4, 4.2.3, and 4.3.1 hold. There exists a positive constant $C_{\mathtt{2lvl}}$, independent of the mesh size and the polynomial degree such that*

$$\left|\left|\left|\mathbb{E}^{\mathtt{2lvl}}_{m_1,m_2} v\right|\right|\right|_{1,J} \leq C_{\mathtt{2lvl}} \Sigma_K |||v|||_{1,J} \tag{4.43}$$

for any $v \in V_J$, where

$$\Sigma_j := C_{J,J-1} C^J_{\mathtt{eig}}(p) \frac{\sqrt{C(p)}}{p} \left(p^4 + (p+1)^2 (p+d)^2\right) (1+m_2)^{-\frac{1}{2}} (1+m_1)^{-\frac{1}{2}}, \tag{4.44}$$

and $C_{J,J-1}$ does not depend on the discretization parameters, but on the shape-regularity the agglomerated grids.

Proof. By using the definition of $\mathbb{E}^{\mathtt{2lvl}}_{m_1,m_2}$, the approximation property, and the smoothing property twice

$$\begin{aligned}
&\left|\left|\left|\mathbb{E}^{\mathtt{2lvl}}_{m_1,m_2} v\right|\right|\right|_{1,J} \\
&\quad = \left|\left|\left| G^{m_2}_J \left(\mathrm{Id}_J - \mathcal{P}^J_{J-1} P^{J-1}_J\right) G^{m_1}_J v \right|\right|\right|_{1,J} \\
&\quad \leq \left[C^J_{\mathtt{eig}}(p) \left(\frac{p^4}{\rho_J^2} + \frac{(p+1)^2 (p+d)^2}{\rho_J^2} \right) \right]^{\frac{1}{2}} (1+m_2)^{-\frac{1}{2}} \left|\left|\left| \left(I_j - R^J_{J-1} P^{J-1}_j\right) G^J_{m_1} v \right|\right|\right|_{0,J} \\
&\quad \lesssim \left[C^J_{\mathtt{eig}}(p) \left(\frac{p^4}{\rho_J^2} + \frac{(p+1)^2 (p+d)^2}{\rho_J^2} \right) \right]^{\frac{1}{2}} (1+m_2)^{-\frac{1}{2}} \sqrt{C(p)} (C^J_{L^2} + C^{J-1}_{L^2}) \frac{h^2_{J-1}}{p} |||G^J_{m_1} v|||_{2,J} \\
&\quad \lesssim C^J_{\mathtt{eig}}(p) \frac{\sqrt{C(p)}}{p} (C^J_{L^2} + C^{J-1}_{L^2}) \left(\frac{p^4}{\rho_j^2} + \frac{(p+1)^2 (p+d)^2}{\rho_J^2} \right) (1+m_2)^{-\frac{1}{2}} (1+m_1)^{-\frac{1}{2}} h^2_{J-1} |||v|||_{1,J}.
\end{aligned}$$

The thesis follows by setting

$$C_{J,J-1} := \left(C^J_{L^2} + C^{J-1}_{L^2}\right),$$

and using the bounded variation assumption 4.2.4. □

From (4.44) we can immediately derive a condition on the number of smoothing steps needed for the convergence of the two-level method.

Corollary 4.4.2.1. Let $\xi > 0$ be a constant such that $\xi > C_{\mathtt{2lvl}}$. Then, provided that

$$(1+m_2)^{\frac{1}{2}} (1+m_1)^{\frac{1}{2}} > \xi C_{J,J-1} C^J_{\mathtt{eig}}(p) \frac{\sqrt{C(p)}}{p} \left(p^4 + (p+1)^2 (p+d)^2\right),$$

the two-level multigrid method converges uniformly.

As expected, we observe that the number of smoothing steps needed to achieve uniform convergence increases with the polynomial degree p. In particular, we observe in this general case a bound of type $(m_1+1)^{\frac{1}{2}}(m_2+1)^{\frac{1}{2}} \geq \mathcal{O}\left(C^J_{\mathtt{eig}}(p)\sqrt{C(p)}p^3\right)$, with additional contribution of p compared to $\mathcal{O}(p^{2+\mu})$, $\mu = 1$, observed in [10, Theorem 8]. As noted in Remark 4.4.1 above, in the case for which the involved constants do not depend on p, we obtain indeed the expected behavior of $\mathcal{O}(p^3)$.

4.4.2 Convergence of the W-cycle algorithm

To prove the convergence of the W-cycle multigrid algorithm, summarized in Algorithm 8, the stability for operators $\mathcal{P}^j_{j-1}$ and P^{j-1}_j is required. In turn, this is based on the following inequality expressing the equivalence between DG norms on subsequent grid levels (cf. [9, Lemma 4.6]):

$$\|v\|_{DG,j} \leq C_{\text{equiv}}\|v\|_{DG,j-1} \qquad \forall v \in V_{j-1}, j \in \{2,\ldots,J\}.$$

Such equivalence can be derived using Assumption 4.2.5, as observed in [10, Lemma 7].

Lemma 4.4.3. *Under Assumption 4.2.5, there exists a constant $C_{\text{stab}} \geq 1$ which does not depend on the mesh size, the polynomial degree, and the multigrid level j, $j \in \{2,\ldots,J\}$, such that*

$$\begin{aligned} \left|\left|\left|\mathcal{P}^j_{j-1}v\right|\right|\right|_{1,j} &\leq C_{\text{stab}}|||v|||_{1,j-1} \qquad \forall v \in V_{j-1}, \\ \left|\left|\left|P^{j-1}_j v\right|\right|\right|_{1,j-1} &\leq C_{\text{stab}}|||v|||_{1,j} \qquad \forall v \in V_j. \end{aligned} \tag{4.45}$$

Algorithm 8: One iteration of W-cycle to solve $A_J x = f_J$

```
  Data: $x_0$ initial guess, $m_1, m_2$ smoothing steps parameters.
1 if $j = 1$ then
2     $\delta_1 \leftarrow \texttt{CoarseGridSolver}(A_1, f_1)$;
3 else
4     for $i = 1,\ldots,m_1$ do
5         $x^{(i)} \leftarrow x^{(i-1)} + B_j^{-1}(f_j - A_j x^{(i-1)})$;
6     $r_{j-1} \leftarrow \texttt{Restrictor}(f_j - A_j x_j)$;
7     $\bar{\delta}_{j-1} \leftarrow \texttt{W-cycle}(A_{j-1}, r_{j-1}, 0, m_1, m_2)$;
8     $\delta_{j-1} \leftarrow \texttt{W-cycle}(A_{j-1}, r_{j-1}, \bar{\delta}_{j-1}, m_1, m_2)$;
9     $x^{m_1+1} \leftarrow x^{m_1} + \texttt{Prolongator}(\delta_{j-1})$;
10    for $i = m_1+2,\ldots,m_1+m_2+1$ do
11        $x^{(i)} \leftarrow x^{(i-1)} + B_j^{-1}(f_j - A_j x^{(i-1)})$;
```

For a W-cycle, the error propagation operator can be defined recursively on top of the two-level error propagation (4.30) as follows

$$\mathbb{E}_{j,m_1,m_2}v := G_J^{m_2}\left(\text{Id}_j - \mathcal{P}^j_{j-1}(\text{Id}_j - \mathbb{E}^2_{j,m_1,m_2})P^{j-1}_j\right)G_J^{m_1}v, \qquad j \in \{2,\ldots,J\}, \tag{4.46}$$

with $\mathbb{E}_{1,m_1,m_2}v = 0$. Similarly to the two-level case, we define $G_j = \mathrm{Id}_j - B_j^{-1}A_j$ and restriction and prolongation operators analogously.

Theorem 4.4.3. *Assume that Assumptions 4.2.1, 4.2.2, 4.2.3, 4.2.4, and 4.2.5 hold. Let $C_{2\mathrm{lvl}}$ and C_{stab} be defined as in Theorem 4.4.2 and Lemma 4.4.3, respectively. Moreover, let $C_{j,j-1}$ be as in Theorem 4.13 but on a generic level j. Then, there exists a constant $\hat{C} > C_{2\mathrm{lvl}}$, independent of the mesh size and the polynomial degree on level j, $j \in \{1,\dots,J\}$, such that if the number of pre- and post- smoothing steps satisfy*

$$(1+m_1)^{1/2}(1+m_2)^{1/2} \geq \begin{cases} \tilde{C}_j(p)C_{j,j-1}\frac{\left(C_{\mathrm{stab}}\right)^2\hat{C}^2}{\hat{C}-C_{2\mathrm{lvl}}} & \textit{if} \quad \frac{C_{j-1,j-2}}{C_{j,j-1}} \leq 1, \\ \tilde{C}_j(p)\frac{\left(C_{j-1,j-2}\right)^2}{C_{j,j-1}}\frac{\left(C_{\mathrm{stab}}\right)^2\hat{C}^2}{\hat{C}-C_{2\mathrm{lvl}}} & \textit{otherwise}. \end{cases} \tag{4.47}$$

then

$$\left|\left|\left|\mathbb{E}_{j,m_1,m_2}v\right|\right|\right|_{1,j} \leq \hat{C}\Sigma_j |||v|||_{1,j} \qquad v \in V_j, \tag{4.48}$$

with

$$\Sigma_j := C_{j,j-1}\frac{\tilde{C}_j(p)}{(1+m_1)^{1/2}(1+m_2)^{1/2}}, \tag{4.49}$$

and $\tilde{C}_j(p) := C^j_{\mathtt{eig}}(p)\frac{\sqrt{C(p)}}{p}$.

Proof. The proof is essentially identical to similar results showed in [10, 9] and is done by induction. For the base case $j = 1$, the statement holds. For $j > 1$, by induction, we assume that (4.48) holds for $j-1$. The definition of the error propagator $\mathbb{E}_{j,m_1,m_2}$ as in (4.46) implies the following bound

$$\left|\left|\left|\mathbb{E}_{j,m_1,m_2}v\right|\right|\right|_{1,j} \leq \left|\left|\left|G_J^{m_2}\left(\mathrm{Id}_j - \mathcal{P}^j_{j-1}P_j^{j-1}\right)G_J^{m_1}v\right|\right|\right|_{1,j} + \left|\left|\left|G_J^{m_2}\mathcal{P}^j_{j-1}\mathbb{E}^2_{j,m_1,m_2}P_j^{j-1}G_J^{m_1}v\right|\right|\right|_{1,j}.$$

The expression in the first term is precisely the error operator associated to a two-level method between levels $j-1$ and j and is bounded by applying the two-level estimate in Theorem 4.4.2. The second addendum can be bounded by applying to a generic level j the smoothing property twice, the stability estimate in (4.4.3), and the induction hypothesis. It follows that

$$\left|\left|\left|G_J^{m_2}\mathcal{P}^j_{j-1}\mathbb{E}^2_{j-1,m_1,m_2}P_j^{j-1}G_J^{m_1}v\right|\right|\right|_{1,j} \leq C^2_{\mathrm{stab}}\hat{C}^2\Sigma^2_{j-1}|||v|||_{1,j}. \tag{4.50}$$

Therefore, we obtain

$$\left|\left|\left|\mathbb{E}_{j,m_1,m_2}v\right|\right|\right|_{1,j} \leq \left(C_{2\mathrm{lvl}}\Sigma_j + C^2_{\mathrm{stab}}\hat{C}^2\Sigma^2_{j-1}\right)|||v|||_{1,j} \tag{4.51}$$

After setting $\tilde{C}_j(p) := C^j_{\mathtt{eig}}(p)\frac{\sqrt{C(p)}}{p}$, the error operators between consecutive levels Σ_{j-1} and Σ_j are linked by the following equality

$$\Sigma_{j-1} = \Sigma_j \tilde{C}_j(p) \left(\frac{C_{j-1,j-2}}{C_{j,j-1}} \right) \leq \Sigma_j \left(\frac{C_{j-1,j-2}}{C_{j,j-1}} \right). \tag{4.52}$$

Using this relation in (4.51) it follows that

$$C_{2\mathrm{lvl}}\Sigma_j + C_{\mathrm{stab}}^2 \hat{C}^2 \Sigma_{j-1}^2 \leq \left(C_{2\mathrm{lvl}} + C_{\mathrm{stab}}^2 \hat{C}^2 \frac{\tilde{C}_j(p)}{(1+m_1)^{1/2}(1+m_2)^{1/2}} \frac{\left(C_{j-1,j-2}\right)^2}{C^{j-1,j}} \right) \Sigma_j.$$

Then, if smoothing steps m_1 and m_2 are such that

$$(1+m_1)^{1/2}(1+m_2)^{1/2} \geq \tilde{C}_j(p) \frac{\left(C_{j-1,j-2}\right)^2}{C_{j-1,j}} \frac{C_{\mathrm{stab}}\hat{C}^2}{\hat{C} - C_{2\mathrm{lvl}}},$$

it holds

$$C_{2\mathrm{lvl}}\Sigma_j + C_{\mathrm{stab}}^2 \hat{C}^2 \Sigma_{j-1}^2 \leq \hat{C}\Sigma_j.$$

Moreover, if $C_{j-1,j-2} \leq C_{j,j-1}$, the condition on m_1 and m_2 can be written as

$$(1+m_1)^{1/2}(1+m_2)^{1/2} \geq \tilde{C}_j(p) C_{j,j-1} \frac{C_{\mathrm{stab}}^2\hat{C}^2}{\hat{C} - C_{2\mathrm{lvl}}}. \tag{4.53}$$

□

4.5 Conclusions

In this chapter we have shown preliminary results on the convergence of two-level and multigrid methods for the solution of the system of equations associated to the interior penalty DG scheme, following the analysis in [10] and adapting it to the case in which the agglomerates are produced using the algorithms in Chapter 3. We confirmed that also in our case the two-level and the W-cycle multigrid schemes converge uniformly with respect to the discretization parameters (and, in the W-cycle case, the number of levels in the hierarchy), if the number of smoothing steps (depending on parameters m_1 and m_2) is sufficiently large.

Chapter 5

Application to cardiac electrophysiology

In this chapter, we exploit the agglomeration strategy developed in Chapter 3 in a polytopic discontinuous Galerkin framework to build a multigrid preconditioner for the monodomain problem, a classical and well established model in cardiac electrophysiology. We present several results which validate our approach both in two and three dimensions, varying ionic models, geometries, and polynomial degrees.

Contents

5.1 Literature review **133**

5.2 Mathematical model **135**

5.2.1 Monodomain problem 135

5.2.2 Ionic models 136

5.3 Space and time discretization **138**

5.3.1 Semidiscrete formulation 138

5.4 Multilevel preconditioning based on polytopes **140**

5.4.1 Inherited bilinear forms 140

5.5 Numerical experiments **142**

5.5.1 Two-dimensional test case 142

5.5.2 Three-dimensional test case 144

5.5.3 Dependence on number of levels 147

5.6 Conclusions and perspectives **152**

5.1 Literature review

Computational modeling of the heart has been proposed and actively pursued as a tool for accelerating cardiovascular research. One of the main challenges of the computational modeling of the heart is the high computational cost, in particular when moving towards whole heart modeling and coupling different physics and scales. The development of methods that reduce the computing time while keeping numerical accuracy becomes essential for speeding-up fundamental research and, ultimately, for translation of modeling into clinical practice. Cardiac electrophysiology simulations are classically based on monodomain or bidomain reaction-diffusion equations for the transmembrane electrical potential. In order to take into account the electrochemical reactions that occur at a cellular level, these systems are coupled through a reaction term to a system of Ordinary Differential Equations (ODEs), modelling the inward and outward flow of ionic currents across the cell membrane [92]. During the years, a large variety of models have been developed, ranging from reduced models with only one or few unknowns such as the Rogers-McCulloch ionic model [160], or the FitzHugh and Nagumo model [91], up to the Bueno-Orovio and Ten Tusscher-Panfilov ionic models [170, 171, 52]. The electric impulse is originated in the sinoatrial node and then is transmitted to the ventricle myocardium, causing a fast depolarization of the tissue. Due to the quick upstroke of the action potential, which is caused by voltage-dependent sodium channels, a numerically robust calculation of the propagation of the wave across the tissue is well known to be computationally challenging. The rapid increase of the transmembrane potential in one cell over a few milliseconds results also in a steep wave front in space, requiring high resolution in both temporal and spatial discretizations. The classical approaches to solve the monodomain or bidomain problems usually exploit linear elements on very fine computational grids, to be able to capture the features of the solution. Higher order Finite Elements and Discontinuous Galerkin (DG) approaches have recently received more attention [118, 4, 45]. The discontinuous setting allows a straightforward implementation of high-order discretizations and lends itself well for hp-adaptivity, assuming that a good a-posteriori error estimator is available. The development of efficient preconditioners for these models is an incredibly vast and active area of research. Among them, we mention Balancing Domain Decomposition with Constraints (BDDC) [136, 137] and Finite Element Tearing and Interconnecting (FETI) methods [85, 126]. In this chapter we exploit the flexibility of the polytopic Discontinuous Galerkin framework introduced in previous chapters, and in particular the multilevel strategy developed in Chapter 3, to precondition the monodomain problem with polynomial degrees $p \geq 1$. We present a sequence of numerical experiments varying ionic models and testing from simple two-dimensional geometries up to fine three-dimensional meshes of true human ventricles. Grids of real-life models are generally very fine due to geometrical complexity. Classical geometric multigrid approaches require the construction of a hierarchy of grids that in the present context is most often obtained by uniformly refining the input mesh. As discussed at the beginning of Chapter 2, in case of very fine and unstructured geometries, building such hierarchies is non-trivial. During the last years, several multilevel strategies have been explored for a large variety of numerical methods posed on arbitrarily shaped elements such as in Virtual Element methods [18, 12, 154] and Hybrid High-Order methods [74, 73]. Motivated by a Discontinuous Galerkin setting, we take here a different perspective and exploit the flexibility given by polytopes to perform a coarsening of the original fine, *given*, mesh and employ such coarsened levels in a multigrid framework to precondition a DG discretization [46, 141] using the strategy developed in Chapter 3.

This chapter is organized as follows. In Section 5.2 we introduce the electrophysiology and monodomain model, and address its spatial and time discretization in Section 5.3. We discuss the details of our preconditioning and coarsening strategy in Section 5.4. Finally, we present numerical results in Section 5.5 and point to future developments in Section 5.6.

5.2 Mathematical model

The heart wall consists of three distinct layers: the internal thin endocardium, the external thin epicardium and the thick muscular cardiac tissue known as the myocardium. The latter is predominantly composed of cardiomyocytes, which are specialized, striated excitable muscle cells responsible for the essential cardiac function. When these cardiomyocytes are stimulated by an electrical impulse, a change in the electro-chemical balance of the cell membrane results in a series of biochemical reactions that determine a large variation of the transmembrane potential, namely the voltage differential between the intra and extracellular spaces of the cell, which implies a depolarization and subsequent slow repolarization mechanism. This process is triggered and controlled by the opening and closing of voltage-gated ion channels that make the cell membrane permeable to specific ionic species, like sodium, potassium, and calcium. The transmembrane potential changes as a result of the ionic fluxes, which, in turn, are driven by the voltage difference itself.

5.2.1 Monodomain problem

The *monodomain model* in the computational domain Ω over the time interval $(0,T]$ reads [92, 69]

$$\begin{cases} \frac{\partial u}{\partial t}(t) + \mathcal{I}_{ion}(u(t),\boldsymbol{w}(t)) - \nabla\cdot(\mathbb{D}\nabla u) = \mathcal{I}_{app}(\boldsymbol{x},t), & \text{in } \in \Omega\times(0,T], \\ \frac{\partial \boldsymbol{w}(t)}{\partial t} = \boldsymbol{H}(u(t),\boldsymbol{w}(t)), & \text{in } \in \Omega\times(0,T], \\ \mathbb{D}\nabla u\cdot\boldsymbol{n} = 0 & \text{on } \partial\Omega\times(0,T], \\ u(0) = u_0, \boldsymbol{w}(0) = \boldsymbol{w}_0, \end{cases} \tag{5.1}$$

with unknowns $u\colon \Omega\times[0,T]\to\mathbb{R}$ and $\boldsymbol{w}\colon \Omega\times[0,T]\to\mathbb{R}^M$, where M is the number of ionic variables. The permeability tensor $\mathbb{D}$ is defined as

$$\mathbb{D} = \sigma_l \boldsymbol{f}_0\otimes\boldsymbol{f}_0 + \sigma_t\boldsymbol{s}_0\otimes\boldsymbol{s}_0 + \sigma_n\boldsymbol{n}_0\otimes\boldsymbol{n}_0,$$

where the vector fields $\boldsymbol{f}_0, \boldsymbol{s}_0$, and $\boldsymbol{n}_0$ express respectively the longitudinal, transversal and normal conductivities [153]. The action potential is triggered by an external applied stimulus $\mathcal{I}_{app}(\boldsymbol{x},t)$, which mimics the presence of a (natural or artificial) pacemaker, while the $\mathcal{I}_{ion}$ term is responsible for describing the electric current generated by the flux of ionic species across the cell membrane.

Homogeneous Neumann boundary conditions are prescribed on the whole boundary $\partial\Omega$ to impose the condition of electrically isolated domain, with $\boldsymbol{n}$ denoting the outward normal unit vector to the boundary. When $d = 2$, we will test the model on a unit square $[0,1]^2$, while with $d = 3$ we will consider a realistic mesh of a human ventricle. In order to compare different models, we note that the monodomain equation can also be written as

$$\chi_m \left(C_m \frac{\partial u}{\partial t} + \widehat{\mathcal{I}}_{ion}(u, \boldsymbol{w}) \right) - \nabla \cdot (\widehat{\mathbb{D}} \nabla u) = \chi_m \widehat{\mathcal{I}}_{app}(\boldsymbol{x}, t),$$

with C_m the membrane cell capacitance, and χ_m the membrane surface-to-volume ratio. Such equation can immediately be written as (5.1) by using the relations $\mathcal{I}_{ion} = \frac{\widehat{\mathcal{I}}_{ion}}{C_m}$, $\mathcal{I}_{app} = \frac{\widehat{\mathcal{I}}_{app}}{C_m}$, and $\mathbb{D} = \frac{\widehat{\mathbb{D}}}{\chi_m C_m}$.

5.2.2 Ionic models

Many ionic models can be recast as the following set of ODEs

$$\begin{cases} \frac{\partial \boldsymbol{w}(t)}{\partial t} = \boldsymbol{H}(u(t), \boldsymbol{w}(t)) & t \in (0, T], \\ u(0) = u_0, \boldsymbol{w}(0) = \boldsymbol{w}_0, \end{cases} \tag{5.2}$$

where the unknowns are the transmembrane potential $u = u(t)$ and the vector $\boldsymbol{w} = (w_1, \ldots, w_M)$ of the M ionic variables. The dynamic of the ionic variables is governed by the functions $\boldsymbol{H}$ and $\mathcal{I}_{ion}$ which couple the gating variables with the evolution of the action potential. In this chapter, we will consider two phenomenological ionic models: FitzHugh-Nagumo [91] and Bueno-Orovio [52].

FitzHugh-Nagumo

The FitzHugh-Nagumo model has the following form

$$\begin{cases} \frac{\partial w(t)}{\partial t} = \varepsilon(u - \Gamma w) & t \in (0, T], \\ u(0) = u_0, w(0) = w_0, \end{cases} \tag{5.3}$$

The $\mathcal{I}_{ion}$ term associated to this model has the following expression

$$\mathcal{I}_{ion} = ku(u-a)(u-1) + w,$$

where $(k, a, \varepsilon, \Gamma)$ are known parameters used to tune the ionic model.

Bueno-Orovio

The Bueno-Orovio minimal model has 3 gating variables $\boldsymbol{w} = (w_1, w_2, .w_3)$ and is described by the following system of ODEs

$$\begin{cases} \frac{\partial w_0(t)}{\partial t} = [(b_0(u) - a_0(u))w_0 + a_0(u)w_0^\infty(u)] & t \in (0,T], \\ \frac{\partial w_1(t)}{\partial t} = [(b_1(u) - a_1(u))w_1 + a_1(u)w_1^\infty(u)] & t \in (0,T], \\ \frac{\partial w_2(t)}{\partial t} = [(b_2(u) - a_2(u))w_2 + a_2(u)w_2^\infty(u)] & t \in (0,T], \\ u(0) = u_0, \boldsymbol{w}(0) = \boldsymbol{w}_0, \end{cases} \tag{5.4}$$

where

- $a_0(u) = \frac{1-H_{V_1}(u)}{H_{V_1}^-(u)(\tau_1'' - \tau_1') + \tau_1'}$,
- $a_1(u) = \frac{1-H_{V_2}(u)}{H_{V_2}^-(u)(\tau_2'' - \tau_2') + \tau_2'}$,
- $a_2(u) = \frac{1}{H_{V_2}(u)(\tau_3'' - \tau_3') + \tau_3'}$,
- $b_0(u) = -\frac{H_{V_1}(u)}{\tau_1^+}$,
- $b_1(u) = -\frac{H_{V_2}(u)}{\tau_2^+}$,
- $b_2(u) = 0$,
- $w_0^\infty(u) = 1 - H_{V_1^-}(u)$,
- $w_1^\infty(u) = H_{V_o}(u)\left(w_\infty^* - 1\frac{u}{\tau_2^\infty}\right) + 1 - \frac{u}{\tau_2^\infty}$,
- $w_2^\infty(u) = H_{V_3^-}^{K_3}(u)$.

Here, the function $H_{z_0}^\varepsilon(z) = \frac{1+\tanh(\varepsilon(z-z_0))}{2}$ is a smooth approximation of the Heaviside function depending on the constant parameter $\varepsilon \in \mathbb{R}^+$. When ε is omitted, it corresponds to the classical Heaviside function. With this model, the ionic term in (5.1) is given by

$$\mathcal{I}_{ion}(u, \boldsymbol{w}) = \sum_{q \in \{fi, so, si\}} I^q(u, \boldsymbol{w}), \tag{5.5}$$

where

- $I^{fi} = -\frac{H_{V_1}(u)(u-V_1)(\hat{V}-u)}{\tau_{fi}} w_0$,
- $I^{so} = \frac{1-H_{V_2}(u)(u-V_o)}{H_{V_o}(u)(\tau_o'' - \tau_o') + \tau_o'} + \frac{H_{V_2(u)}}{H_{V_{so}}(u)(\tau_{so}'' - \tau_{so}') + \tau_{so}'}$,
- $I^{si} = -\frac{H_{V_2}}{\tau_{si}} w_1 w_2$.

We can hence introduce here the monodomain equation, which will be coupled with one of the presented ionic models. Albeit relatively simple, the Bueno-Orovio model has the characteristic of capturing the main features of the electrophysiology in healthy myocardial tissues.

5.3 Space and time discretization

In this section we describe the spatial and temporal discretization of system (5.1). We adopt a classical discontinuous Galerkin approach. We consider a shape-regular mesh Ω_h which satisfies standard assumptions of regularity and quasi-uniformity and parametrized by the mesh size h. Let us recall the usual function space V_h^p of globally discontinuous polynomials of degree p in each coordinate direction, namely

$$V_h^p = \{v_h \in L^2(\Omega) \colon v_{h|K} \in \mathcal{Q}_p(K), K \in \Omega_h\},$$

with $\mathcal{Q}_p(K)$ denoting the space of tensor-product polynomials on element K of degree $p \geq 1$. The application of the *Symmetric Interior Penalty Discontinuous Galerkin* [159] method involves the definition of the following bilinear forms, with $a(u_h, v_h)$ adapted from (4.3)

$$\begin{aligned} a(u_h,v_h) &= \int_\Omega \mathbb{D}\nabla_h u_h \cdot \nabla_h v_h \,\mathrm{d}\mathbf{x} - \int_\Gamma \Big(\{\{\mathbb{D}\nabla u_h\}\} \cdot [\![v_h]\!] + \{\{\mathbb{D}\nabla v_h\}\} \cdot [\![u_h]\!]\Big)\,\mathrm{d}s + \int_\Gamma \sigma [\![u_h]\!] \cdot [\![v_h]\!]\,\mathrm{d}s, \\ m(u_h,v_h) &= \int_\Omega u_h v_h \,\mathrm{d}\mathbf{x}. \end{aligned}$$

The penalty parameter is defined edge-wise as $\sigma(\boldsymbol{x}) = \alpha \boldsymbol{n}^T \mathbb{D} \boldsymbol{n} > 0$, with $\alpha = \frac{Cp^2}{h}$. In our numerical experiments, we set $C = 10$.

5.3.1 Semidiscrete formulation

The semi-discrete formulation associated to problem (5.1) requires, for any $t \in (0,T]$, to find $u_h(t) \in V_h^p$ and $w_h(t) \in V_h^p$ such that

$$\begin{cases} \left(\frac{\partial u_h}{\partial t}, v_h\right) + a(u_h,v_h) + \Big(\mathcal{I}_{ion}(u_h,\boldsymbol{w_h}), v_h\Big) = \Big(\mathcal{I}_{app}, v_h\Big), \\ \left(\frac{\partial w_h}{\partial t}, v_h\right) = \boldsymbol{H}(u_h, w_h), \\ u_h(0) = u_{h,0}, \\ \boldsymbol{w_h}(0) = \boldsymbol{w}_{h,0}, \end{cases} \tag{5.6}$$

holds for every $v_h \in V_h^p$. Here, $u_{h,0}$ and $\boldsymbol{w}_{h,0}$ are given initial data in V_h^p and $(\cdot,\cdot)$ is the L^2-inner product on Ω. The right-hand side of the equation involves only the external current term since homogeneous Neumann boundary conditions are imposed. Given a basis $\{\phi_i\}_i^N$ of V_h^p, with $N = \dim(V_h^p)$, we write $\boldsymbol{U}_h(t)$ and $\boldsymbol{W}_h(t)$ for the vectors holding the expansion coefficients of $u_h(t), \boldsymbol{w}_h(t)$ with respect to

$\{\phi_i\}_i^N$ for each time instant $t \in (0,T]$. We denote with A and M the stiffness and mass matrix obtained after numerical integration of the bilinear forms $a(\cdot,\cdot)$ and $m(\cdot,\cdot)$. Finally, let I_{ion}^h, I_{app}^h be the finite element interpolants of $\mathcal{I}_{ion}$ and $\mathcal{I}_{app}$, respectively. With this notation, the algebraic formulation of (5.6) reads

$$\begin{cases} M\dot{\boldsymbol{U}}_h + A\boldsymbol{U}_h + MI_{ion}^h(\boldsymbol{U}_h, \boldsymbol{W}_h) = MI_{app}^h, \\ \boldsymbol{U}_h(0) = \boldsymbol{U}_0, \\ \boldsymbol{W}_h(0) = \boldsymbol{W}_0, \end{cases} \tag{5.7}$$

and has to be coupled with the system of ODEs for the ionic model at hand, cf. (5.3) and (5.4) above. To obtain a fully discrete system, we split the interval $(0,T]$ into N_T uniform subintervals with a time step $\Delta t = t_{n+1} - t_n$, with $t_n = n\Delta t$ for $n = 0,\dots,N_T - 1$. We use the subscript n to indicate the approximation of $\boldsymbol{U}_h(t)$ and $\boldsymbol{W}_h(t)$ at time t_n. We adopt a semi-implicit backward Euler approach where the diffusion term is discretized implicitly in time, while the reaction term is evaluated explicitly [176, 151, 45]. This results in the following scheme

$$M\frac{\boldsymbol{U}_{n+1} - \boldsymbol{U}_n}{\Delta t} + A\boldsymbol{U}_{n+1} + MI_{ion}^h(\boldsymbol{U}_n, \boldsymbol{W}_{n+1}) = MI_{app,n+1}^h, \tag{5.8}$$

which yields an explicit formula for $\boldsymbol{U}_{n+1}$

$$\left(\frac{M}{\Delta t} + A\right)\boldsymbol{U}_{n+1} = MI_{app,n+1}^h - MI_{ion}^h(\boldsymbol{U}_n, \boldsymbol{W}_{n+1}) + \frac{M}{\Delta t}\boldsymbol{U}_n. \tag{5.9}$$

Therefore, given the solution $(\boldsymbol{U}_n, \boldsymbol{W}_n)$ at time t_n, the solution at time t_{n+1} is computed as follows:

- For $i = 1,\dots,N$: solve the ionic model system for $\boldsymbol{W}_{n+1}$.
- Plug the computed $\boldsymbol{W}_{n+1}$ in (5.9) and solve the resulting linear system for $\boldsymbol{U}_{n+1}$.

Note that this scheme allows crucial gains in computational efficiency: the problem for $\boldsymbol{U}_{n+1}$ is linear and hence the system matrix $\frac{M}{\Delta t} + A$ and relative preconditioner are assembled only once at the beginning of the computation. In our implementation, matrices A and M are both assembled consistently, without lumping the mass matrix M which is clearly an option since in a DG discretization it is block diagonal. As for the evaluation of the ionic term $I_{ion}^h(\boldsymbol{U}_n, \boldsymbol{W}_{n+1})$ in (5.7), we opt for the so-called Ionic Current Interpolation (ICI) method [158]. We first compute the values of $\boldsymbol{U}_n$ and $\boldsymbol{W}_{n+1}$ at the degrees of freedom, and then we interpolate them at quadrature points while assembling local contributions for the right-hand side vector. This procedure is cheaper and requires less memory than solving at each time step the system of ODEs and computing I_{ion}^h at quadrature nodes (a procedure also known as State Variable Interpolation). The system matrix is symmetric and positive definite, hence the natural choice is to solve the resulting system with the preconditioned conjugate gradient method. The focus of this chapter will be the investigation of a multilevel preconditioner which exploits a polytopic coarsening of the given mesh.

5.4 Multilevel preconditioning based on polytopes

In this section, we present our multilevel preconditioner based on polytopic elements, used to enable convergence acceleration. The classical geometric multigrid (GMG) begins with a coarse (possibly unstructured) mesh which is then repeatedly subdivided in finer meshes to obtain a hierarchy of levels for which intergrid transfers are simple and fast. However, GMG does not allow for the usage of a *predetermined* fine grid. A way to alleviate this strong bond between mesh generation and hierarchies is to employ a non-nested approach, where coarse and fine meshes are generated independently. A novel procedure to avoid the computational burden stemming from this flexibility was the object of Chapter 2. A popular and effective way to obviate the need for coarsening mesh levels is to operate at the discrete level as done in Algebraic Multigrid approaches (AMG). Algebraic Multigrid methodologies are widely employed as method of choice for large and sparse systems arising from a large class of PDEs, particularly effective for those arising from scalar elliptic problems [70, 71]. In finite element the context, AMGs are often used for matrices stemming from low order Lagrangian elements ($p = 1, 2$). The fact that they only rely on information extracted from the system matrix is a clear advantage in terms of usability. With a Discontinuous Galerkin setting, however, the redundancy of the degrees of freedom associated to the same grid point is known to hinder the creation of aggregates. In [6], a purely algebraic (i.e. without the need to access mesh points) AMG method tailored for high-order DG schemes has been developed and tested in two dimensions. Here, we assume to be given the target, fine, mesh Ω_h over which we need to solve the problem, along with the associated discretized operator. Instead of uniformly refining Ω_h or using only the matrix, we take a hybrid route and use the built-in flexibility given by polytopic shapes, together with the strategy developed in Chapter 3, to build a hierarchy of *coarser* operators that can be employed as level matrices in a multilevel method. We note that, given the sequence of grids is *nested* by construction, the transfer operators between consecutive levels are cheap as previously discussed in this thesis. Our multilevel preconditioner will then be built at the algebraic level exploiting the geometrical information encoded in the agglomeration routine, complemented by such convenient transfer operators. Therefore, we propose an approach somewhat dual in that it can be interpreted both as algebraic and geometric.

5.4.1 Inherited bilinear forms

Following the agglomeration procedure in Section 3.3, a sequence of L nested levels $\{\mathcal{T}_l\}_{l=0}^{L-1}$ and Finite Element spaces $\{V_l\}_{l=0}^{L-1}$, where $V_0 \subset V_1 \subset \ldots \subset V_{L-1}$, is generated, with $\mathcal{T}_0 \equiv \Omega_h$ and $V_0 \equiv V_h^p$. There are essentially two ways to build a sequence of coarser operators: the *inherited* approach, where the discrete operators are recursively built on top of the fine one, and the *non-inherited* one, where bilinear forms are explicitly assembled on each agglomerated grid $\mathcal{T}_l$ of the hierarchy. Since the number of resulting levels L may not be known *a-priori*, we index the sequence of operators in the same way as it is customary in AMG methods, where to a higher index corresponds a coarser level, and denote with $\mathcal{P}_{l+1}^{l}$ the prolongation from the *coarser* level $l+1$ to the *finer* level l. As usual, the restriction operator $\mathcal{R}_l^{l+1}$ is defined as the adjoint of the prolongation. On the input mesh $\Omega_h \equiv \mathcal{T}_0$, we consider the bilinear form $\mathcal{A}_0 : V_0 \times V_0 \to \mathbb{R}$ defined by

$$\mathcal{A}_0(u,v) := \frac{m(u,v)}{\Delta t} + a(u,v) \qquad \forall u,v \in V_0, \tag{5.10}$$

and its associated matrix representation $A_0 := \left(\frac{M}{\Delta t} + A\right)$ introduced in Section 5.3.

Instead of assembling each bilinear form on each sublevel, we considered *inherited* bilinear forms [9] which are obtained iteratively from the restriction of the original bilinear form $\mathcal{A}_0$ as follows

$$\mathcal{A}_{l+1}(u,v) := \mathcal{A}_l\left(\mathcal{P}_{l+1}^l u, \mathcal{P}_{l+1}^l v\right) \qquad \forall u,v \in V_{l+1}, \tag{5.11}$$

with associated matrices

$$A_{l+1} := \left(P_{l+1}^l\right)^T A_l P_{l+1}^l. \tag{5.12}$$

This technique is one of the basic ingredients of AMG methods and is often referred to as *Galerkin projection* or *Galerkin triple product*.

Algorithm 9 summarizes the overall non-inherited procedure. From the implementation standpoint, we observe that each prolongation matrix $P_{l+1}^l \in \mathbb{R}^{n_l \times n_{l+1}}$ is a distributed matrix whose parallel layout is easily defined thanks to nestedness of the partitions and the fact that in a DG discretization DoF are non-overlapping. Moreover, the simple block structure of the prolongation matrices allows avoiding explicit assembly: restriction and prolongation of vectors can indeed be performed in a completely matrix-free fashion by using the local action of each block element-wise. The strategy of avoiding to discretize the operators A_l ($l \geq 1$) explicitly for each level is quite attractive, as looping over all the polytopes for each level l (see snippet 3.2), albeit showed to scale in Section 3.6, becomes increasingly expensive when agglomerated elements are composed of many sub-elements, particularly for high polynomial degrees, due to large number of quadrature points. In [13], it was showed that in case of homogeneous coefficients such cost can be dramatically reduced using a so-called quadrature-free approach. This exploits the generalized Stokes theorem together with the Euler's homogeneous function theorem in order to reduce the integration over a polytope only to boundary evaluations. In both quadrature approaches, the original mesh Ω_h has to be used to assemble all operators, with a huge gain in favor of the quadrature-free approach if it can be employed. Furthermore, we stress that the *inherited* technique is easy to be added in existing Finite Element frameworks, in that it does not require assembling explicitly flux terms over ghost polytopic elements. Indeed, with this approach the polytopic shapes obtained by agglomeration are used just as a tool to build a sequence of operators $\{\mathcal{P}_l\}_l$ which are in turn used as in (5.12) to create coarse level matrices, as observed originally in [46]. On the other hand, it was proved in [9], for classical h-multigrid with Interior Penalty DG discretization of the Laplace equation, that only the *non-inherited* multigrid provides uniform convergence with respect to the number of levels (at the cost of assembling the bilinear forms over agglomerated elements). Following the approach [46], this dependence on the number of levels for the inherited approach can be removed by using a properly rescaled Galerkin projection of the stabilization terms in the DG discretization, originally proposed in [8] in the context of two level Schwarz methods. However, our numerical simulations indicate that for the present problem the number of iterations does not seem to be affected by the number of levels, cf. Section 5.5 for details.

Algorithm 9: Construction of level operators $\{A_l\}_l$.

```
Data: Mesh $\Omega_h$.
Result: Sequence of level operators $\{A_l\}_l$.
1 Function BuildLevelOperators():
2     read_mesh($\Omega_h$);
3     $A_0 \leftarrow$ assemble_system();
4     build_agglomerated_hierarchy($\Omega_h$);
5     for $l$ in $0,\dots,L-1$ do
6         $P^l_{l+1} \leftarrow$ compute_prolongation($l$) ;          /* Canonical injection */
7         $A_{l+1} \leftarrow \left(P^l_{l+1}\right)^T A_l P^l_{l+1}$;
8     return $\{A_l\}_l$;
```

5.5 Numerical experiments

In this section we investigate the capabilities of the polytopic preconditioner in different scenarios by varying dimensions, geometry, and ionic models. We compare our scheme with the TRILINOS ML implementation of AMG [95] used as a preconditioner to the conjugate-gradient solver. In all the tests, the stopping criterion for the preconditioned conjugate-gradient is based on the absolute residual with tolerance 1×10^{-14}. We consider the isotropic case for the diffusion tensor, i.e. $\mathbb{D} = \Sigma I_{d\times d}$, $d = 2,3$, as it is known that for this particular case the number of iterations does not depend strongly on the anisotropy of $\mathbb{D}$ since there is always a dominant direction.

5.5.1 Two-dimensional test case

We start with the following test, taken from the recent work [45], aiming at assessing the validity of our implementation with a two-dimensional problem which exhibits both the depolarization and repolarization phases. We consider Problem (5.1) on the domain $\Omega = [0,1]^2 \subset \mathbb{R}^2$ and as ionic model the FitzHugh-Nagumo model outlined in (5.3), with the same parameters used in [45], which we report in Table 5.5 for completeness. The initial unit square is refined 6 times, resulting in a mesh Ω_h of $2^{2\cdot 6} = 4096$ quadrilaterals. Running all the experiments in parallel by splitting the workload through a mesh partitioner implies that the resulting partitions loose the original structured nature of the grid, as shown in Figure 5.1. Hence, the agglomeration procedure is not producing the sequence one would obtain by coarsening a fine square in a structured way. As such, this test is representative of the general situation.

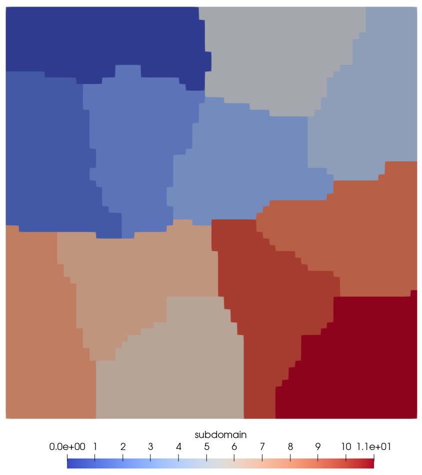
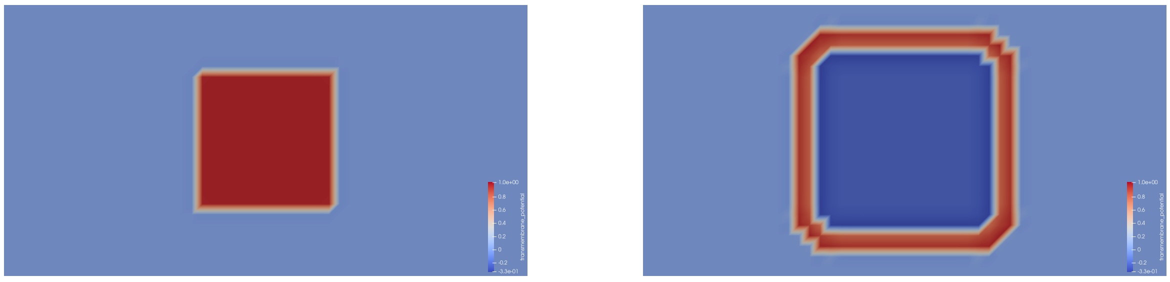


Fig. 5.1 Ω_h partitioned among 12 processors using PARMETIS. Each color maps to one MPI rank.

The applied current is

$$\mathcal{I}_{app}(\boldsymbol{x},t) = 2 \times 10^6 \mathbb{1}_{[0.4,0.6]}(x)\mathbb{1}_{[0.4,0.6]}(y)\mathbb{1}_{[0,1\times 10^{-3}]}(t), \tag{5.13}$$

where $\mathbb{1}_{[a,b]}(\cdot)$ is the indicator function over the interval $[a,b]$. Such definition corresponds to a temporary and localized electric shock localized in the square $[0.4,0.6]^2$. We solve the problem until time $T = 3 \times 10^{-3}$s, with step size $\Delta t = 1 \times 10^{-4}$s and both the potential u and the gating variable $\boldsymbol{w} \equiv w$ set at rest. We show the activation and the subsequent propagation of the transmembrane potential u in Figure 5.2, with polynomial degree $p = 2$ and standard nodal Lagrangian basis. As can be seen from the figure, the wavefront has been captured by the DG discretization, which achieves the maximum value of 1 on the while the internal part undergoes repolarization. We test the multilevel framework detailed in Section 5.4 and Algorithm 9. In particular, we precondition a conjugate-gradient by a single V-cycle of multigrid, using $m = (1,3)$ pre- and post-smoothings sweeps of a Chebyshev smoother of degree 5. The needed eigenvalue estimates are computed with 20 iterations of the Lanczos iteration. This is compared with AMG, with parameters tuned for best results, varying the number of sweeps to be performed by the Chebyshev smoother from 1 to 3. We report in Figure 5.3 the number of outer iterations required by conjugate gradient for each time step, varying polynomial degree p from 1 to 5. In all instances, the multilevel strategy achieves lower iterations counts compared to the AMG preconditioner. Lower iteration counts is achieved by our approach also in the low-order case $p = 1$, which is usually the most favourable for AMG. Increasing the polynomial order p shows a more pronounced gain for the polytopic approach, displaying a clear pattern for which the number of iterations is essentially constant after the initial propagation of the electric potential u, while the iterations' path for AMG is more prone to jitter.



(a) $t = 0.04$s (b) $t = 0.16$s

Fig. 5.2 Snapshots of the transmembrane potential $\boldsymbol{U}$ after the external application of the current $\mathcal{I}_{app}(\boldsymbol{x},t)$ in Equation (5.13).

5.5.2 Three-dimensional test case

The following experiment employs a CAD-modeled mesh Ω_h representing the realistic left ventricle made of 374 022 hexahedra and displayed in Figure 5.4, available at the repository associated to the work [3]. On this mesh and with time-step $\Delta t = 1 \times 10^{-4}$s, we solve until $T = 0.4$s the problem (5.1) using the Bueno-Orovio ionic model in (5.4), with the parameters reported in Table 5.6, until . The external stimulus $\mathcal{I}_{app}(\boldsymbol{x},t)$ is localized at one point of the ventricle for 3ms with a magnitude of $300\mathrm{V\,s}^{-1}$. The location of the point where the impulse is applied is shown in Figure 5.4 (right). As initial data we take $u = -84$mV for the transmembrane potential while all gating variables are set at rest. Some snapshots of the transmembrane potential u_h at selected time steps (with $p = 2$) are shown in Figure 5.7. Note that the steep wavefront has been captured.

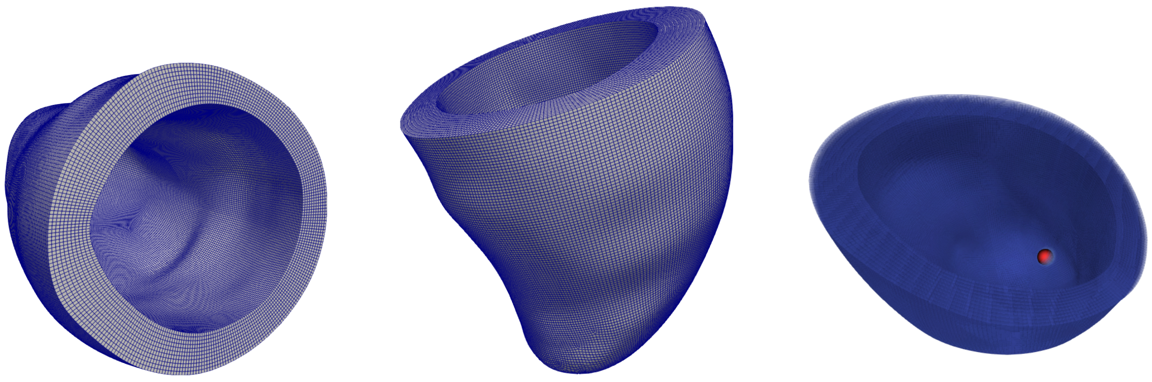
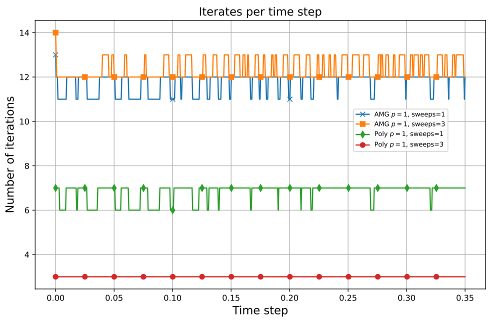
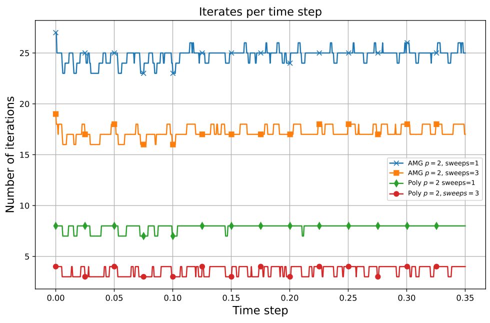


Fig. 5.4 Detailed views of the hexahedral mesh Ω_h representing a left ventricle, with a zoom at the bottom of the ventricle to highlight (in red) the point where $\mathcal{I}_{app}(\boldsymbol{x},t)$ is applied.

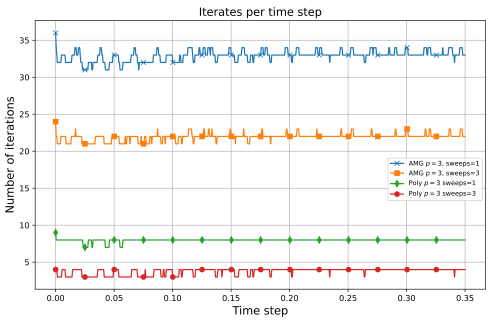
The multigrid hierarchy has been generated using the adapted parallel version of our Algorithm 3. The number of mesh elements (agglomerates) at each level $l \in \{0,\ldots,L-1\}$ is shown in Table 5.2 when the full ventricle geometry is distributed with MPI among 256 processors. With the given number of processors, the coarsest level ($l = 3$) consists of 3 agglomerates per process. It is remarkable that the decrease of the global number of elements between consecutive levels has a ratio very close



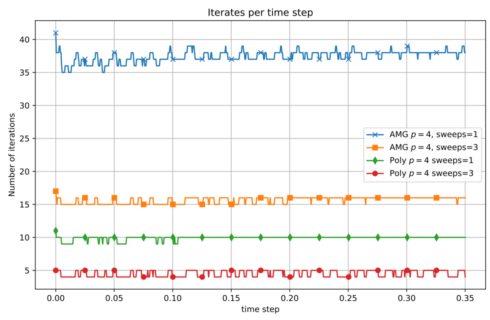
(a) $p = 1$



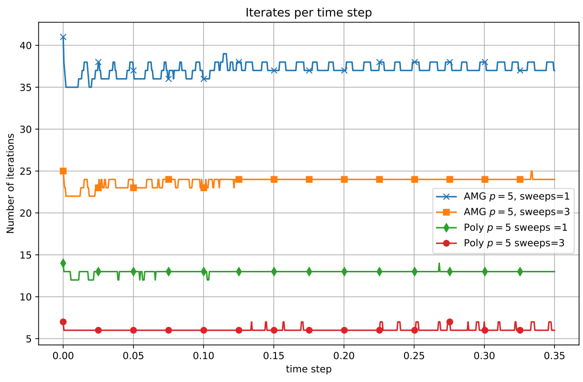
(b) $p = 2$



(c) $p = 3$



(d) $p = 4$



(e) $p = 5$

Fig. 5.3 Number of CG iterations per time step for Problem (5.1), varying polynomial degree p and the number of sweeps.

Hierarchy for ventricle mesh (**128** processors)		
Level l	Card($\mathcal{T}_l$)	Card($\mathcal{T}_{l-1}$)/Card($\mathcal{T}_l$)
$l = 0$	374 022	–
$l = 1$	46 809	~ 7.9
$l = 2$	5 908	~ 7.9
$l = 3$	768	~ 7.7

Table 5.1 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^{3}$ on top of the original mesh $\Omega_0 \equiv \mathcal{T}_0$, partitioning the original mesh across **128** processors. The number of agglomerates per level is shown in second column, while the ratio between the cardinality of consecutive grids is reported in the last column.

to 8, which is the value one would obtain by halving the mesh step size of a uniform hexahedral grid. In three-dimensional problems, a sufficient number of levels L is needed to guarantee that the grid on the coarse level is *coarse enough* and, consequently, that the coarse solver is not destroying the performance of the whole V-cycle. In Tables 5.1 and 5.3 we report the number of agglomerates per level by shifting the number of processors from 128 to 1024. We observe again very good balancing in terms of workload per process and ratios between the cardinality of consecutive grids. This is possible thanks to the combination of the graph partitioner PARMETIS, applied to the initial grid Ω_0, and the balance achieved by the agglomeration routine. In case of 1024 processes, the hierarchy is formed by 3 levels, with the coarsest one comprising precisely 6 agglomerates per process. In order to coarsen even further the coarse grid, it is possible to remove processes on the coarse level in a controlled way and to switch to subcommunicators, as explained in [169] in the field of high order multigrid for continuous FEM.

We report the iterations' path per time step with polynomial degrees $p = 1$ and 2 in Figures 5.5 and 5.6, respectively. For all degrees and approaches, the number of CG iterations have a maximum during the initial propagation of the potential, a successive drop around $t = 0.17$s, when the solution does not exhibit abrupt variations, and finally a slight increase at the end of the simulation where the repolarization phase starts taking place. The observed patterns follow what was observed in the two-dimensional case, where the polytopic approach achieves lower iterations counts than AMG for all the polynomial degrees. As expected, we verify that increasing the number of smoothing steps from 1 to 3 (with fixed degree p) leads to lower iteration counts across the whole time interval $[0, T]$. Conversely, fixing the number of smoothing steps and increasing the polynomial degree p we observe a slight increase in terms of iteration counts for all approaches.

Hierarchy for ventricle mesh (**256** processors)		
Level l	Card$(\mathcal{T}_l)$	Card$(\mathcal{T}_{l-1})/$Card$(\mathcal{T}_l)$
$l=0$	374 022	–
$l=1$	46 871	~ 7.9
$l=2$	5 987	~ 7.8
$l=3$	768	~ 7.8

Table 5.2 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^3$ on top of the original mesh $\Omega_0 \equiv \mathcal{T}_0$, partitioning the original mesh across **256** processors. The number of agglomerates per level is shown in second column, while the ratio between the cardinality of consecutive grids is reported in the last column.

Hierarchy for ventricle mesh (**1024** processors)		
Level l	Card$(\mathcal{T}_l)$	Card$(\mathcal{T}_{l-1})/$Card$(\mathcal{T}_l)$
$l=0$	374 022	–
$l=1$	47 160	~ 7.9
$l=2$	6 144	~ 7.7

Table 5.3 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^3$ on top of the original mesh $\Omega_0 \equiv \mathcal{T}_0$, partitioning the original mesh across **1024** processors. The number of agglomerates per level is shown in second column, while the ratio between the cardinality of consecutive grids is reported in the last column.

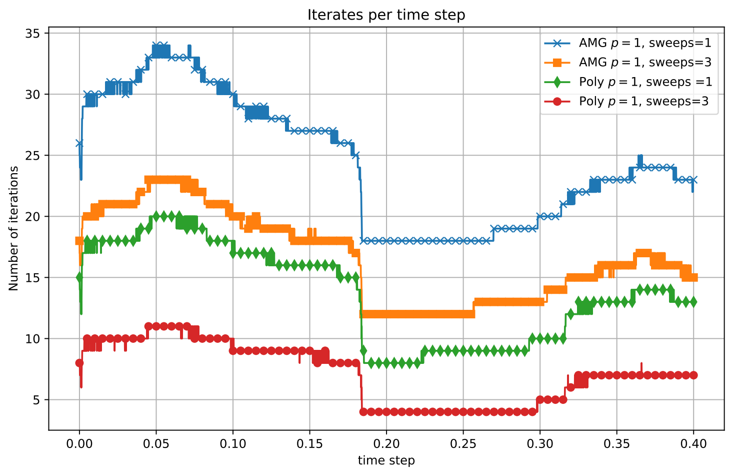


Fig. 5.5 Number of CG iterations per time step for Problem (5.1) for the 3D ventricle test case with $p=1$.

5.5.3 Dependence on number of levels

As discussed in Section 5.4, a drawback of building the coarse operators through an inherited approach is that the number of iterations depends on the number of employed multigrid levels L, as proven in [10]. In [46, Section 6], this behavior was indeed observed while solving a series of two-dimensional

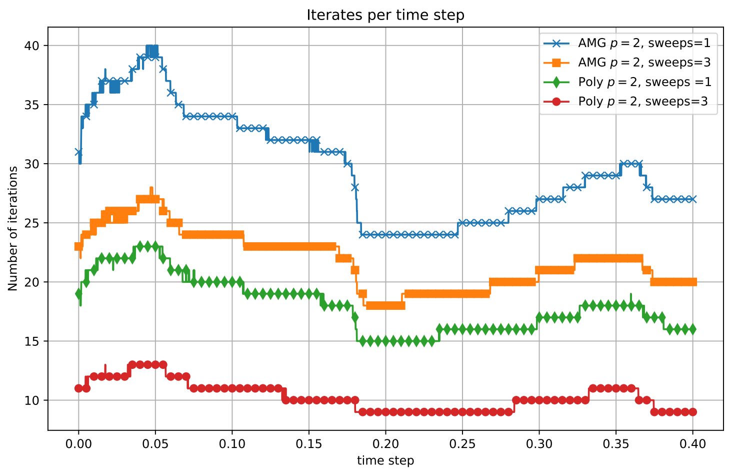


Fig. 5.6 Number of CG iterations per time step for Problem (5.1) for the 3D ventricle test case with $p = 2$.

Poisson problems with smooth solutions using FGMRES preconditioned with h-multigrid, and later fixed by adjusting the amount of stabilization in each level through a suitable choice of the intergrid transfers. Here, we investigate this dependence for the linear system of equations arising from (5.7). In order to provide a meaningful test, we consider as grid Ω_h the idealized left ventricle showed in Figure 5.8, consisting of 50 988 hexahedral elements. Using 12 processors we obtain the 5 levels indicated in Table 5.4. We solve the monodomain problem with the same data and tolerances as in previous test, but changing the number of levels and the polynomial degree p in order to verify if the number of iterations is affected by the number of levels. We report our findings in Figures 5.9, 5.10. The number of required CG iterations is in all cases constant both in time and in the number of levels L present in the hierarchy. We note only an increase of 1 iteration when the number of levels is 2. As mentioned before, however, for large three-dimensional geometries the number of levels must be chosen large enough because of the higher pressure on the coarse grid solver.

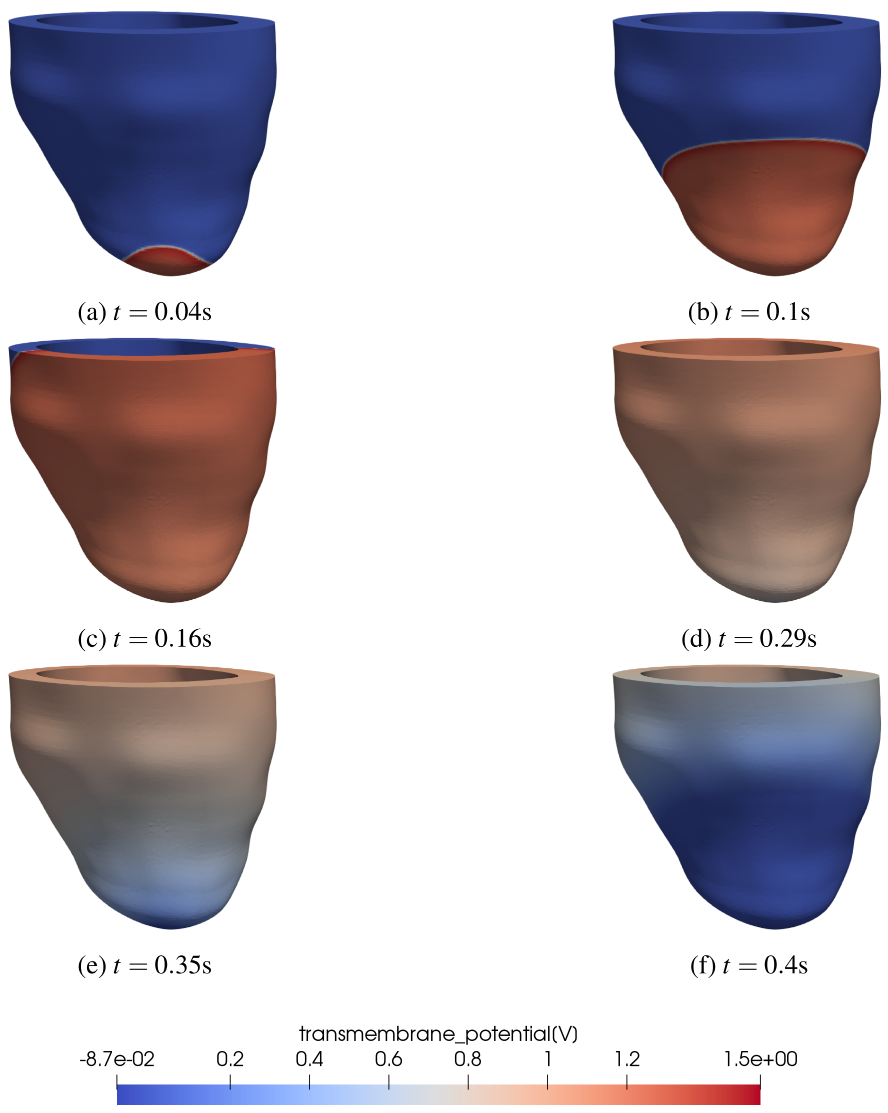


Fig. 5.7 Snapshots of the transmembrane potential $\boldsymbol{U}$ at selected time steps after the external application of the current $\mathcal{I}_{app}(\boldsymbol{x},t)$ at one point.

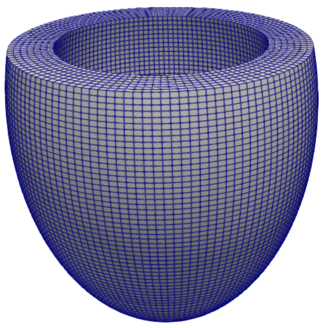


Fig. 5.8 Hexahedral mesh of an ellipsoid representing an idealized left ventricle.

Idealized ventricle (**12** processors)	
Level l	Card$(\mathcal{T}_l)$
$l = 0$	50 988
$l = 1$	6 377
$l = 2$	803
$l = 3$	108
$l = 4$	24

Table 5.4 Coarsened hierarchy $\{\mathcal{T}_l\}_{l=0}^{4}$ on top of the mesh representing the left ventricle, partitioning the original mesh across **12** processors.

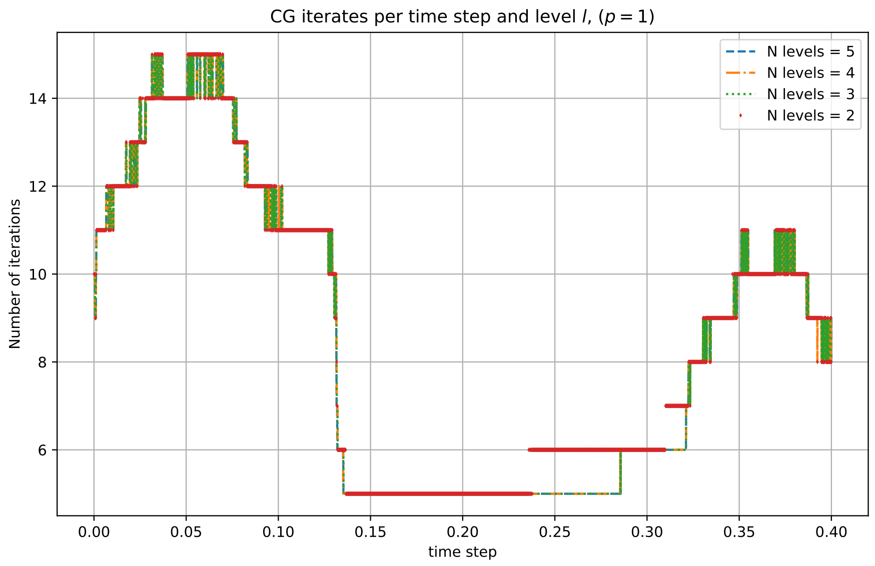


Fig. 5.9 Number of CG iterations per time step for Problem (5.9) for the idealized ventricle test case with $p = 1$ and different number of levels.

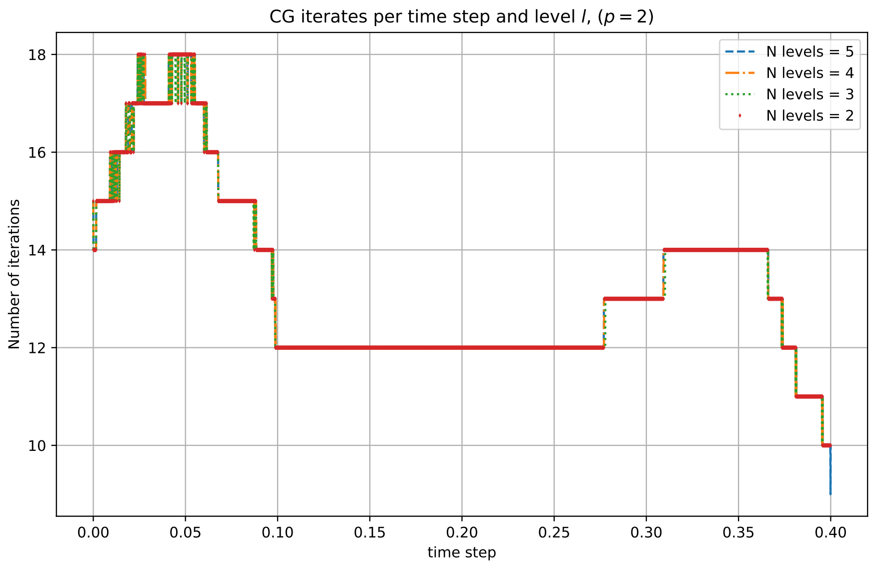


Fig. 5.10 Number of CG iterations per time step for Problem (5.9) for the idealized ventricle test case with $p = 2$ and different number of levels.

5.6 Conclusions and perspectives

In this chapter, we have demonstrated the high flexibility of a multilevel preconditioner in a polytopic Discontinuous Galerkin framework by applying it to the monodomain problem in cardiac electrophysiology. By exploiting the coarsening strategy introduced in Chapter 3, we described how to generate a sequence of coarse operators in a completely distributed fashion, and showed its effectiveness through a sequence of two- and three-dimensional tests, including real geometries and data. This approach fully exploits the power of DG methods in terms of their ability to handle general shapes. Future work could include the investigation of the stabilization strategy of [46] and its influence on the iteration counts. A matrix-free implementation of the action of coarser operators is currently under investigation and is subject of future work. In particular, we aim to operate on batches of cells and perform cell operations for each batch in a vectorization-over-cells fashion to increase the throughput, by exploiting SIMD capabilities of modern processors. The author of this thesis thanks the Italian supercomputing center CINECA for the computational resources made available through the polyDEAL-HP10C9NFGS ISCRA-C project.

Parameter	Value
$C_m[\mathrm{m}^{-1}]$	1×10^{-2}
$\chi_m[\mathrm{F\,m}^{-2}]$	1
$\Sigma[\mathrm{m}^2\,\mathrm{s}^{-2}]$	1.2×10^{-1}
$\mathbb{D}[\mathrm{m}^2\,\mathrm{s}^{-2}]$	$\Sigma I_{2\times 2}$
κ	19.5
ε	40
Γ	0.1
a	1.3×10^{-2}

Table 5.5 FitzHugh-Nagumo parameters used in the 2D numerical test.

Parameter	Value
$\Sigma[\mathrm{m}^2\,\mathrm{s}^{-2}]$	1×10^{-4}
$\tau_o^{'}[\mathrm{s}^{-1}]$	6×10^{-3}
$\tau_o^{''}[\mathrm{s}^{-1}]$	6×10^{-3}
$\tau_{so}^{''}[\mathrm{s}^{-1}]$	4.3×10^{-2}
$\tau_{so}^{''}[\mathrm{s}^{-1}]$	2×10^{-4}
$\tau_{si}[\mathrm{s}^{-1}]$	2.8723×10^{-3}
$\tau_{fi}[\mathrm{s}^{-1}]$	1.1×10^{-4}
$\tau_1^{+}[\mathrm{s}^{-1}]$	1.4506×10^{-3}
$\tau_2^{+}[\mathrm{s}^{-1}]$	2.8×10^{-1}
$\tau_2^{\infty}[\mathrm{s}^{-1}]$	7×10^{-2}
$\tau_1^{'}[\mathrm{s}^{-1}]$	6×10^{-2}
$\tau_1^{''}[\mathrm{s}^{-1}]$	1.15
$\tau_2^{''}[\mathrm{s}^{-1}]$	7×10^{-2}
$\tau_2^{''}[\mathrm{s}^{-1}]$	2×10^{-2}
$\tau_3^{'}[\mathrm{s}^{-1}]$	2.7342×10^{-3}
$\tau_3^{''}$	3×10^{-3}
$w_\infty^\star$	9.4×10^{-1}
k_2	6.50×10^{1}
k_3	2.0994
k_{so}	2.0
V_1	3×10^{-1}
V_{1m}	1.5×10^{-2}
V_2	1.5×10^{-2}
V_{2m}	3×10^{-2}
V_3	9.087×10^{-1}
$\hat{V}$	1.58
V_o	6×10^{-3}
V_{so}	6.5×10^{-1}

Table 5.6 Parameters for Bueno-Orovio model used in the 3D numerical test. In this case $\chi_m \equiv C_m = 1$ as we are using formulation (5.1).

Chapter 6

Conclusions and final remarks

In this thesis, we have presented and validated a series of computational techniques associated to the handling of complex and non-matching meshes in a distributed-memory setting, as well as their application to the context of multilevel solvers. Notably, we have designed and implemented an efficient agglomeration strategy in the context of polytopic finite elements, and in particular within the polytopic Discontinuous Galerkin framework. Our approach leverages spatial data structures and allows constructing hierarchy of grids that can be employed in multilevel contexts starting from of an original fine mesh. Here we summarize the main results presented in this manuscript.

- Handling the coupling of grids stemming from different domains has been successfully performed in a dimension-independent way. This is relevant in FSI problems and within non-matching finite element approaches. We have investigated the effect of exact quadrature rules on mesh intersections as a tool to preserve accuracy. We validated our implementation by testing several non-matching techniques for elliptic interface problems and discussed relevant implementation challenges and computational costs [23]. Our software contribution was integrated into a well-established software framework [22].

- In Chapter 2, we have developed a novel implementation of the non-nested multigrid method [47, 7, 38] in a memory-distributed setting where levels can be generated and partitioned independently in case of continuous Lagrangian elements. Our implementation is matrix-free, which is known to be a crucial feature when the polynomial degree p increases. We provide a detailed explanation of the algorithmic realization of our scheme and perform benchmarks in order to assess computational costs. Tests on non-trivial geometries confirm the robustness and reliability of our implementation, as well its robustness in terms of iteration counts. The resulting software contribution has also been integrated within the DEAL.II library in [23] and is part of [88].

- In Chapter 3, we have shown a novel agglomeration algorithm to generate a sequence of polytopic and nested grids starting from an initial fine mesh. The theoretical framework where we have applied our algorithm is the polytopic discontinuous Galerkin method. The key strategy was the usage of spatial data structures such as R-trees [105] and their application within a (polytopic) finite element context. We have demonstrated, through an extensive set of geometries and numerical experiments on Poisson problems, that our algorithm produces high-quality agglomerated meshes in a robust and dimension-independent fashion. One of the crucial points of such technique is that the sequence of grids can be used as a hierarchy in a multilevel framework. This chapter is based on our work [87]. On top of this, we discuss and present several implementation details. Notably, we highlight the design choices of our memory-distributed implementation and show, through a series of tests, how our data structures (which build on top of the DEAL.II library design), allow obtaining a scalable solver also in presence of polytopic shapes.

- In Chapter 4, we established convergence for two-level and multigrid methods for the general polytopal Discontinuous Galerkin discretizations analysed in [61]. Exploiting the theoretical framework developed in [10], we prove that our multilevel strategy based on R-trees is convergent.

- By extending the results presented in Chapters 3 and 4 we have developed a multigrid preconditioner for the monodomain problem, one of the central models in cardiac electrophysiology. After describing the implementation pipeline for the particular model at hand, we validate our multilevel preconditioner in both two- and three-dimensional geometries. The approach has been validated by covering different ionic models, geometries, and polynomial degrees.

6.1 Future perspectives

During the investigation of the various methodologies presented in this thesis, we have identified several research lines that should be further studied and developed.

The computational infrastructure developed in Chapter 2, within a non-nested multigrid context, perfectly fits with the topics presented in Chapter 1. Exploiting such algorithms in the context of non-matching methods where different domains are coupling in a distributed and arbitrarily manner will boost their efficiency and applicability to even more challenging scenarios.

With regard to the agglomeration procedure outlined in Chapter 3, we foresee several possible research directions. From a more mathematical standpoint, the assumption on the delivered shapes allows carrying out a more detailed analysis such as the one presented in Chapter 4. In addition to that, the fact that agglomerates are geometrically similar to their enclosing box is appealing in that suitable basis functions can be devised. From a more practical side, the application to even more complex shapes such as meshes with different material properties as well as grids comprising complex features must be taken into account. Preliminary tests in this direction show promising results for grids arising from biomedical applications. Enlarging the viewpoint beyond polytopic methods, and inspired by the results in Chapter 5 where agglomeration stands as a basis to multilevel methodologies, we think that multigrid strategies exploiting mesh coarsening can be extremely versatile and appealing in contexts where the generation of hierarchy is practically not feasible.

Finally, from a more HPC-oriented perspective, various paths should be explored and are currently under investigation. In relation to the performance results in Chapter 3, it is clear that the computational bottleneck (especially for higher-orders) is associated to the assembly phase. This motivates the investigation of suitable matrix-free approaches also in this context, possibly combining vectorization strategies such as the ones exploited for classical elements [131], as well as taking advantage of hardware accelerators such as GPUs, which have emerged in the last decade as a game changer in the context of high-order methods.

References

[1] Coreform Cubit 2023.4.0 [Computer software]. Orem, UT: Coreform LLC. Retrieved from http://coreform.com.

[2] Mark F. Adams. Evaluation of three unstructured multigrid methods on 3d finite element problems in solid mechanics. *International Journal for Numerical Methods in Engineering*, 55, 2000. URL https://api.semanticscholar.org/CorpusID:13969890.

[3] Pasquale Claudio Africa, Roberto Piersanti, Francesco Regazzoni, Michele Bucelli, Matteo Salvador, Marco Fedele, Stefano Pagani, Luca Dede', and Alfio Quarteroni. lifex-ep: a robust and efficient software for cardiac electrophysiology simulations. *BMC bioinformatics*, 24(1): 389, 2023.

[4] P.C. Africa, M. Salvador, P. Gervasio, L. Dede', and A. Quarteroni. A matrix-free high-order solver for the numerical solution of cardiac electrophysiology. *Journal of Computational Physics*, 478:111984, 2023. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2023.111984. URL https://www.sciencedirect.com/science/article/pii/S0021999123000797.

[5] Mark Ainsworth and Charles Parker. Unlocking the secrets of locking: Finite element analysis in planar linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 395: 115034, 2022. ISSN 0045-7825. doi:https://doi.org/10.1016/j.cma.2022.115034. URL https://www.sciencedirect.com/science/article/pii/S0045782522002560.

[6] Paola F. Antonietti and Laura Melas. Algebraic multigrid schemes for high-order nodal discontinuous galerkin methods. *SIAM Journal on Scientific Computing*, 42(2):A1147–A1173, 2020. doi:10.1137/18M1204383. URL https://doi.org/10.1137/18M1204383.

[7] Paola F. Antonietti and Giorgio Pennesi. V-cycle multigrid algorithms for discontinuous Galerkin methods on non-nested polytopic meshes. *Journal of Scientific Computing*, 78: 625–652, 2019. doi:10.1007/s10915-018-0783-x.

[8] Paola F. Antonietti, Blanca Ayuso de Dios, Susanne C. Brenner, and Li-Yeng Sung. Schwarz methods for a preconditioned wopsip method for elliptic problems. 2012. URL https://api.semanticscholar.org/CorpusID:17493272.

[9] Paola F. Antonietti, Marco Sarti, and Marco Verani. Multigrid algorithms for hp-discontinuous galerkin discretizations of elliptic problems. *SIAM Journal on Numerical Analysis*, 53(1):598–618, 2015. doi:10.1137/130947015. URL https://doi.org/10.1137/130947015.

[10] Paola F. Antonietti, Paul Houston, Marco Sarti, and Marco Verani. Multigrid algorithms for *hp*-version interior penalty discontinuous galerkin methods on polygonal and polyhedral meshes. *Calcolo*, 54:1169–1198, 2017. doi:https://doi.org/10.1007/s10092-017-0223-6.

[11] Paola F. Antonietti, Paul Houston, and Giorgio Pennesi. Fast numerical integration on polytopic meshes with applications to discontinuous galerkin finite element methods. *Journal of Scientific Computing*, 77, 12 2018. doi:10.1007/s10915-018-0802-y.

[12] Paola F. Antonietti, Lorenzo Mascotto, and Marco Verani. A multigrid algorithm for the p-version of the virtual element method. *ESAIM: M2AN*, 52(1):337–364, 2018. doi:10.1051/m2an/2018007. URL https://doi.org/10.1051/m2an/2018007.

[13] Paola F. Antonietti, Paul Houston, and Giorgio Pennesi. Fast numerical integration on polytopic meshes with applications to discontinuous galerkin finite element methods. *Journal of Scientific Computing*, 77:1339–1370, 2019. doi:https://doi.org/10.1007/s10915-018-0802-y.

[14] Paola F. Antonietti, Paul Houston, Giorgio Pennesi, and Endre Süli. An agglomeration-based massively parallel non-overlapping additive schwarz preconditioner for high-order discontinuous galerkin methods on polytopic grids. *Mathematics of Computation*, 89(325): 2047–2083, 2020.

[15] Paola F. Antonietti, Chiara Facciolà, Paul Houston, Ilario Mazzieri, Giorgio Pennesi, and Marco Verani. *High–order Discontinuous Galerkin Methods on Polyhedral Grids for Geophysical Applications: Seismic Wave Propagation and Fractured Reservoir Simulations*, pages 159–225. 06 2021. ISBN 978-3-030-69362-6. doi:10.1007/978-3-030-69363-3_5.

[16] Paola F. Antonietti, Lourenco Beirõ da Veiga, and Gianmarco Manzini. *The Virtual Element Method and its Applications*. SIMAI Springer Series, Springer International Publishing, 2022.

[17] Paola F. Antonietti, F. Dassi, and E. Manuzzi. Machine learning based refinement strategies for polyhedral grids with applications to virtual element and polyhedral discontinuous galerkin methods. *Journal of Computational Physics*, 469:111531, 2022. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2022.111531. URL https://www.sciencedirect.com/science/article/pii/S0021999122005939.

[18] Paola F. Antonietti, Stefano Berrone, Martina Busetto, and Marco Verani. Agglomeration-based geometric multigrid schemes for the virtual element method. *SIAM Journal on Numerical Analysis*, 61(1):223–249, 2023. doi:10.1137/21M1466864. URL https://doi.org/10.1137/21M1466864.

[19] Paola F. Antonietti, Mattia Corti, and Gabriele Martinelli. Polytopal mesh agglomeration via geometrical deep learning for three-dimensional heterogeneous domains. *arXiv preprint arXiv:2406.10587*, 2024.

[20] Paola F. Antonietti, Nicola Farenga, Enrico Manuzzi, Gabriele Martinelli, and Luca Saverio. Agglomeration of polygonal grids using graph neural networks with applications to multigrid solvers. *Computers & Mathematics with Applications*, 154:45–57, 2024.

[21] Daniel Arndt, Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Jean-Paul Pelteret, Bruno Turcksin, and David Wells. The deal.II finite element library: Design, features, and insights. *Computers & Mathematics with Applications*, 81:407–422, January 2021.

[22] Daniel Arndt, Wolfgang Bangerth, Marco Feder, Marc Fehling, Rene Gassmöller, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Simon Sticko, Bruno Turcksin, and David Wells. The `deal.II` library, version 9.4. *Journal of Numerical Mathematics*, 2022. doi:10.1515/jnma-2022-0054.

[23] Daniel Arndt, Wolfgang Bangerth, Maximilian Bergbauer, Marco Feder, Marc Fehling, Johannes Heinz, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Bruno Turcksin, David Wells, and Stefano Zampini. The `deal.II` library, version 9.5. *Journal of Numerical Mathematics*, 31(3):231–246, 2023. doi:10.1515/jnma-2023-0089.

[24] Marco Attene, Silvia Biasotti, Silvia Bertoluzza, Daniela Cabiddu, Marco Livesu, Giuseppe Patanè, Micol Pennacchio, Daniele Prada, and Michela Spagnuolo. Benchmarking the geometrical robustness of a virtual element poisson solver. *Mathematics and Computers in Simulation*, 190:1392–1414, 2021.

[25] Ivo Babuška. The finite element method with Lagrangian multipliers. *Numerische Mathematik*, 20(3):179–192, 1973.

[26] A. H. Baker, Tz. V. Kolev, and U. M. Yang. Improving algebraic multigrid interpolation operators for linear elasticity problems. *Numerical Linear Algebra with Applications*, 17(2-3): 495–517, 2010. doi:https://doi.org/10.1002/nla.688. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.688.

[27] Satish Balay, William Gropp, Lois Curfman McInnes, and Barry F Smith. Petsc, the portable, extensible toolkit for scientific computation. *Argonne National Laboratory*, 2(17), 1998.

[28] Wolfgang Bangerth, Carsten Burstedde, Timo Heister, and Martin Kronbichler. Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Transactions on Mathematical Software (TOMS)*, 38:1 – 28, 2011. URL https://api.semanticscholar.org/CorpusID:1158172.

[29] F. Bassi, A. Ghidoni, S. Rebay, and P. Tesini. High-order accurate p-multigrid discontinuous galerkin solution of the euler equations. *International Journal for Numerical Methods in Fluids*, 60(8):847–865, 2009. doi:https://doi.org/10.1002/fld.1917. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.1917.

[30] F. Bassi, L. Botti, A. Colombo, D.A. Di Pietro, and P. Tesini. On the flexibility of agglomeration based physical space discontinuous galerkin discretizations. *Journal of Computational Physics*, 231(1):45–65, 2012. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2011.08.018. URL https://www.sciencedirect.com/science/article/pii/S0021999111005055.

[31] P. Bastian and C. Wieners. Multigrid methods on adaptively refined grids. *Computing in Science & Engineering*, 8(6):44–54, 2006. doi:10.1109/MCSE.2006.116.

[32] R. Becker and M. Braack. Multigrid techniques for finite elements on locally refined meshes. *Numerical Linear Algebra with Applications*, 7(6):363–379, 2000. doi:https://doi.org/10.1002/1099-1506(200009)7:6<363::AID-NLA202>3.0.CO;2-V.

[33] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *ACM SIGMOD Conference*, 1990. URL https://api.semanticscholar.org/CorpusID:11567855.

[34] L. Beirão da Veiga, K. Lipnikov, and G. Manzini. *The mimetic finite difference method for elliptic problems*, volume 11 of *MS&A. Modeling, Simulation and Applications*. Springer, Cham, 2014. ISBN 978-3-319-02662-6; 978-3-319-02663-3. doi:10.1007/978-3-319-02663-3. URL http://dx.doi.org/10.1007/978-3-319-02663-3.

[35] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L.D. Marini, and A. Russo. Basic principles of virtual element methods. *Math. Models Methods Appl. Sci.*, 23(1):199–214, 2013. ISSN 0218-2025.

[36] T. Belytschko and T. Black. Elastic crack growth in finite elements with minimal remeshing. *International Journal for Numerical Methods in Engineering*, 45(5):601–620, 1999. doi:https://doi.org/10.1002/(SICI)1097-0207(19990620)45:5<601::AID-NME598>3.0.CO;2-S. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0207%2819990620%2945%3A5%3C601%3A%3AAID-NME598%3E3.0.CO%3B2-S.

[37] Maximilian Bergbauer, Peter Munch, Wolfgang A. Wall, and Martin Kronbichler. High-performance matrix-free unfitted finite element operator evaluation, 2024. arXiv preprint 2404.07911.

[38] Marco L. Bittencourt, Craig C. Douglas, and Raúl A. Feijóo. Nonnested multigrid methods for linear problems. *Numerical Methods for Partial Differential Equations*, 17(4):313–331, 2001. doi:https://doi.org/10.1002/num.1013. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/num.1013.

[39] Daniele Boffi and Lucia Gastaldi. A fictitious domain approach with Lagrange multiplier for fluid-structure interactions. *Numer. Math.*, 135(3):711–732, 2017. ISSN 0029-599X. doi:10.1007/s00211-016-0814-1. URL https://doi.org/10.1007/s00211-016-0814-1.

[40] Daniele Boffi, Lucia Gastaldi, Luca Heltai, and Charles S. Peskin. On the hyper-elastic formulation of the immersed boundary method. *Computer Methods in Applied Mechanics and Engineering*, 197(25-28):2210–2231, 04 2008. ISSN 0374-2830. doi:10.1016/j.cma.2007.09.015.

[41] Daniele Boffi, Franco Brezzi, and Michel Fortin. *Mixed finite element methods and applications*, volume 44 of *Springer Series in Computational Mathematics*. Springer, Heidelberg, 2013.

[42] Daniele Boffi, Fabio Credali, and Lucia Gastaldi. On the interface matrix for fluid-structure interaction problems with fictitious domain approach. *Comput. Methods Appl. Mech. Engrg.*, 401(part B):Paper No. 115650, 23, 2022. ISSN 0045-7825. doi:10.1016/j.cma.2022.115650. URL https://doi.org/10.1016/j.cma.2022.115650.

[43] Daniele Boffi, Andrea Cangiani, Marco Feder, Lucia Gastaldi, and Luca Heltai. A comparison of non-matching techniques for the finite element approximation of interface problems. *Comput. Math. Appl.*, 151:101–115, 2023. ISSN 0898-1221,1873-7668. doi:10.1016/j.camwa.2023.09.017. URL https://doi.org/10.1016/j.camwa.2023.09.017.

[44] Boost. Boost C++ Libraries. http://www.boost.org/, 2015.

[45] Federica Botta, Matteo Calafà, Pasquale C. Africa, Christian Vergara, and Paola F. Antonietti. High-order discontinuous galerkin methods for the monodomain and bidomain models, 2024. URL https://arxiv.org/abs/2406.03045.

[46] L. Botti, A. Colombo, and F. Bassi. h-multigrid agglomeration based solution strategies for discontinuous galerkin discretizations of incompressible flow problems. *Journal of Computational Physics*, 347:382–415, 2017. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2017.07.002. URL https://www.sciencedirect.com/science/article/pii/S0021999117305041.

[47] James H. Bramble, Joseph E. Pasciak, and Jinchao Xu. The analysis of multigrid algorithms with nonnested spaces or noninherited quadratic forms. *Mathematics of Computation*, 56(193): 1–34, 1991. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2008527.

[48] Susanne C. Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*, volume 15 of *Texts in Applied Mathematics*. Springer, 2008. URL http://dx.doi.org/10.1007/978-0-387-75934-0.

[49] Susanne C. Brenner and Jie Zhao. Convergence of multigrid algorithms for interior penalty methods. *Applied Numerical Analysis & Computational Mathematics*, 2(1):3–18, 2005. doi:https://doi.org/10.1002/anac.200410019. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/anac.200410019.

[50] Franco Brezzi, Konstantin Lipnikov, and Valeria Simoncini. A family of mimetic finite difference methods on polygonal and polyhedral meshes. *Mathematical Models and Methods in Applied Sciences*, 15, 04 2005. doi:10.1142/S0218202505000832.

[51] Jed Brown. Efficient nonlinear solvers for nodal high-order finite elements in 3D. *Journal of Scientific Computing*, 45:48–63, 2010.

[52] Alfonso Bueno-Orovio, Elizabeth M. Cherry, and Flavio H. Fenton. Minimal model for human ventricular action potentials in tissue. *Journal of Theoretical Biology*, 253(3):544–560, 2008. ISSN 0022-5193. doi:https://doi.org/10.1016/j.jtbi.2008.03.029. URL https://www.sciencedirect.com/science/article/pii/S0022519308001690.

[53] Erik Burman, Susanne Claus, Peter Hansbo, Mats G. Larson, and André Massing. Cutfem: Discretizing geometry and partial differential equations. *International Journal for Numerical Methods in Engineering*, 104(7):472–501, 2015. doi:https://doi.org/10.1002/nme.4823.

[54] Erik Burman, Matteo Cicuttin, Guillaume Delay, and Alexandre Ern. An unfitted hybrid high-order method with cell agglomeration for elliptic interface problems. *SIAM Journal on Scientific Computing*, 43(2):A859–A882, 2021. doi:10.1137/19M1285901. URL https://doi.org/10.1137/19M1285901.

[55] Carsten Burstedde. Parallel tree algorithms for amr and non-standard data access. *ACM Trans. Math. Softw.*, 46(4), nov 2020. ISSN 0098-3500. doi:10.1145/3401990. URL https://doi.org/10.1145/3401990.

[56] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011. doi:10.1137/100791634.

[57] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011. doi:10.1137/100791634.

[58] Matteo Caldana, Paola F. Antonietti, and Luca Dede'. A deep learning algorithm to accelerate algebraic multigrid methods in finite element solvers of 3d elliptic pdes, 2023.

[59] Andrea Cangiani, Emmanuil Georgoulis, and Paul Houston. *hp*-version discontinuous Galerkin methods on polygonal and polyhedral meshes. *Mathematical Models and Methods in Applied Sciences*, 24, 05 2014. doi:10.1142/S0218202514500146.

[60] Andrea Cangiani, Zhaonan Dong, Emmanuil H. Georgoulis, and Paul Houston. *hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes*. SpringerBriefs in Mathematics. Springer, Cham, 2017. ISBN 978-3-319-67671-5; 978-3-319-67673-9.

[61] Andrea Cangiani, Zhaonan Dong, and Emmanuil H. Georgoulis. *hp*-version discontinuous Galerkin methods on essentially arbitrarily-shaped elements. *Math. Comp.*, 91(333):1–35, 2021. ISSN 0025-5718,1088-6842. doi:10.1090/mcom/3667. URL https://doi.org/10.1090/mcom/3667.

[62] Tony F. Chan, Susie Go, and Ludmil Zikatanov. *Multilevel elliptic solvers on unstructured grids*, pages 488–511. doi:10.1142/9789812812957_0027. URL https://www.worldscientific.com/doi/abs/10.1142/9789812812957_0027.

[63] Tony F. Chan, Jinchao Xu, and Ludmil T. Zikatanov. An agglomeration multigrid method for unstructured grids. *Contemporary mathematics*, 1998. URL https://api.semanticscholar.org/CorpusID:11554160.

[64] C. Chevalier and F. Pellegrini. Pt-scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6):318–331, 2008. ISSN 0167-8191. doi:https://doi.org/10.1016/j.parco.2007.12.001. URL https://www.sciencedirect.com/science/article/pii/S0167819107001342.

[65] Thomas C. Clevenger, Timo Heister, Guido Kanschat, and Martin Kronbichler. A flexible, parallel, adaptive geometric multigrid method for FEM. *ACM Trans. Math. Softw.*, 47(1), 2020. ISSN 0098-3500. doi:10.1145/3425193. URL https://doi.org/10.1145/3425193.

[66] B. Cockburn, J. Gopalakrishnan, and R. Lazarov. Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems. *SIAM J. Numer. Anal.*, 47(2):1319–1365, 2009. ISSN 0036-1429. doi:10.1137/070706616. URL http://dx.doi.org/10.1137/070706616.

[67] Bernardo Cockburn, Bo Dong, and Johnny Guzmán. A superconvergent ldg-hybridizable galerkin method for second-order elliptic problems. *Mathematics of Computation*, 77(264): 1887–1916, 2008. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/40234595.

[68] Bernardo Cockburn, Jayadeep Gopalakrishnan, and Francisco-Javier Sayas. A projection-based error analysis of hdg methods. *Mathematics of Computation*, 79:1351–1367, 07 2010. doi:10.1090/S0025-5718-10-02334-3.

[69] P. Colli Franzone and Luca F. Pavarino. A parallel solver for reaction-diffusion systems in computational electrocardiology. *Mathematical Models and Methods in Applied Sciences*, 14(06):883–911, 2004. doi:10.1142/S0218202504003489. URL https://doi.org/10.1142/S0218202504003489.

[70] Pasqua D'Ambra, Fabio Durastante, and Salvatore Filippone. Amg preconditioners for linear solvers towards extreme scale. *SIAM Journal on Scientific Computing*, 43(5):S679–S703, 2021. doi:10.1137/20M134914X. URL https://doi.org/10.1137/20M134914X.

[71] Pasqua D'Ambra, Fabio Durastante, S M Ferdous, Salvatore Filippone, Mahantesh Halappanavar, and Alex Pothen. Amg preconditioners based on parallel hybrid coarsening and multi-objective graph matching. In *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 59–67, 2023. doi:10.1109/PDP59025.2023.00017.

[72] Denis Davydov, Jean-Paul Pelteret, Daniel Arndt, Martin Kronbichler, and Paul Steinmann. A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid. *International Journal for Numerical Methods in Engineering*, 121(13):2874—2895, 2020. doi:10.1002/nme.6336.

[73] D. A. Di Pietro, F. Hülsemann, P. Matalon, P. Mycek, U. Rüde, and D. Ruiz. Towards robust, fast solutions of elliptic equations on complex domains through hybrid high-order discretizations and non-nested multigrid methods. *International Journal for Numerical Methods in Engineering*, 122(22):6576–6595, 2021. doi:10.1002/nme.6803.

[74] D. A. Di Pietro, P. Matalon, P. Mycek, and U. Rüde. High-order multigrid strategies for hho discretizations of elliptic equations. *Numerical Linear Algebra with Applications*, 30(1):e2456, 2023. doi:10.1002/nla.2456.

[75] Daniele Di Pietro and Jerome Droniou. The hybrid high-order method for polytopal meshes, design, analysis, and applications. *Modeling, Simulation and Applications*, 01 2020. doi:10.1007/978-3-030-37203-3.

[76] Daniele Di Pietro, Frank Hülsemann, Pierre Matalon, Paul Mycek, Ulrich Rüde, and Daniel Ruiz. Towards robust, fast solutions of elliptic equations on complex domains through HHO discretizations and non-nested multigrid methods. *International Journal for Numerical Methods in Engineering*, pages 6576–6595, 07 2021. doi:10.1002/nme.6803.

[77] Daniele A. Di Pietro and Alexandre Ern. A hybrid high-order locking-free method for linear elasticity on general meshes. *Comput. Methods Appl. Mech. Engrg.*, 283:1–21, 2015. ISSN 0045-7825. doi:10.1016/j.cma.2014.09.009. URL https://doi.org/10.1016/j.cma.2014.09.009.

[78] Daniele A. Di Pietro and Alexandre Ern. Hybrid high-order methods for variable-diffusion problems on general meshes. *Comptes Rendus Mathematique*, 353(1):31–34, 2015. ISSN 1631-073X. doi:https://doi.org/10.1016/j.crma.2014.10.013. URL https://www.sciencedirect.com/science/article/pii/S1631073X1400257X.

[79] Thomas Dickopf and Rolf Krause. *A Study of Prolongation Operators Between Non-nested Meshes*, volume 78, pages 343–350. 01 2011. ISBN 978-3-642-11303-1. doi:10.1007/978-3-642-11304-8_39.

[80] J. Donea, S. Giuliani, and J. Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33(1-3):689–723, September 1982. doi:10.1016/0045-7825(82)90128-1.

[81] Zhaonan Dong and Emmanuil H Georgoulis. Robust interior penalty discontinuous galerkin methods. *Journal of Scientific Computing*, 92(2):57, 2022.

[82] Alexandre Ern and Jean-Luc Guermond. *Theory and practice of finite elements*. 2004. URL https://api.semanticscholar.org/CorpusID:117037950.

[83] Andreas Fabri, Geert-Jan Giezeman, Lutz Kettner, Stefan Schirra, and Sven Schönherr. The CGAL kernel: A basis for geometric computation. *Lecture Notes in Computer Science*, pages 191–202, 1996.

[84] Robert D. Falgout, Jim E. Jones, and Ulrike Meier Yang. The design and implementation of hypre, a library of parallel high performance preconditioners. In Are Magnus Bruaset and Aslak Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, pages 267–294, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-31619-0.

[85] Charbel Farhat, Jan Mandel, and Francois Xavier Roux. Optimal convergence properties of the feti domain decomposition method. *Computer Methods in Applied Mechanics and Engineering*, 115(3):365–385, 1994. ISSN 0045-7825. doi:https://doi.org/10.1016/0045-7825(94)90068-X. URL https://www.sciencedirect.com/science/article/pii/004578259490068X.

[86] Marco Feder, Luca Heltai, and Andrea Cangiani. polydeal.

[87] Marco Feder, Andrea Cangiani, and Luca Heltai. R3mg: R-tree based agglomeration of polytopal grids with applications to multilevel methods. *arXiv e-prints*, pages arXiv–2404, 2024.

[88] Marco Feder, Luca Heltai, Peter Munch, and Martin Kronbichler. A matrix-free implementation of the non-nested multigrid method. *In preparation*, 2024.

[89] N. Fehn, P. Munch, W. A. Wall, and Ma. Kronbichler. Hybrid multigrid methods for high-order discontinuous Galerkin discretizations. *J. Comput. Phys.*, 415:109538, 2020. doi:10.1016/j.jcp.2020.109538.

[90] Gerald Fischer, Bernhard Tilg, Robert Modre, G.J.M. Huiskamp, J. Fetzer, W. Rucker, and Paul Wach. A bidomain model based bem-fem coupling formulation for anisotropic cardiac tissue. *Annals of Biomedical Engineering*, 28(10):1229–1243, 2000. doi:10.1114/1.1318927.

[91] Richard FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical Journal*, 1(6):445–466, 1961. ISSN 0006-3495. doi:https://doi.org/10.1016/S0006-3495(61)86902-6. URL https://www.sciencedirect.com/science/article/pii/S0006349561869026.

[92] Piero Colli Franzone, Luca F. Pavarino, and Simone Scacchi. Mathematical cardiac electrophysiology/ piero colli franzone, luca f. pavarino, simone scacchi. 2014. URL https://api.semanticscholar.org/CorpusID:123850677.

[93] T.-P. Fries and T. Belytschko. The extended/generalized finite element method: an overview of the method and its applications. *Internat. J. Numer. Methods Engrg.*, 84(3):253–304, 2010. ISSN 0029-5981.

[94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., USA, 1995. ISBN 0201633612.

[95] Michael Gee, C Siefert, J Hu, R. Tuminaro, and M Sala. Ml 5.0 smoothed aggregation user's guide. 01 2006.

[96] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. doi:https://doi.org/10.1002/nme.2579. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2579.

[97] Amir Gholami, Dhairya Malhotra, Hari Sundar, and George Biros. Fft, fmm, or multigrid? a comparative study of state-of-the-art poisson solvers for uniform and nonuniform grids in the unit cube. *SIAM Journal on Scientific Computing*, 38(3):C280–C306, 2016. doi:10.1137/15M1010798. URL https://doi.org/10.1137/15M1010798.

[98] V. Girault and R. Glowinski. Error analysis of a fictitious domain method applied to a Dirichlet problem. *Japan J. Indust. Appl. Math.*, 12(3):487–514, 1995. ISSN 0916-7005. doi:10.1007/BF03167240. URL https://doi.org/10.1007/BF03167240.

[99] R. Glowinski, T.-W. Pan, T.I. Hesla, and D.D. Joseph. A distributed Lagrange multiplier/fictitious domain method for particulate flows. *International Journal of Multiphase Flow*, 25(5): 755–794, 1999. ISSN 0301-9322. doi:https://doi.org/10.1016/S0301-9322(98)00048-2.

[100] Roland Glowinski, Tsorng-Whay Pan, and Jacques Périaux. A fictitious domain method for Dirichlet problem and applications. *Comput. Methods Appl. Mech. Engrg.*, 111(3-4):283–303, 1994. ISSN 0045-7825. doi:10.1016/0045-7825(94)90135-X. URL https://doi.org/10.1016/0045-7825(94)90135-X.

[101] Shahab Golshan, Peter Munch, Rene Gassmoller, Martin Kronbichler, and Bruno Blais. Lethe-dem : An open-source parallel discrete element solver with load balancing. 2021.

[102] Emma Griffiths, Jan Hinrichsen, Nina Reiter, and Silvia Budday. On the importance of using region-dependent material parameters for full-scale human brain simulations. *European Journal of Mechanics - A/Solids*, 99:104910, 2023. ISSN 0997-7538. doi:https://doi.org/10.1016/j.euromechsol.2023.104910. URL https://www.sciencedirect.com/science/article/pii/S0997753823000025.

[103] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996. ISSN 0167-8191. doi:https://doi.org/10.1016/0167-8191(96)00024-5. URL https://www.sciencedirect.com/science/article/pii/0167819196000245.

[104] Morton Gurtin and W. Drugan. An introduction to continuum mechanics. *Journal of Applied Mechanics*, 51:949, 12 1984. doi:10.1115/1.3167763.

[105] Antonin Guttman. R trees: A dynamic index structure for spatial searching. volume 14, pages 47–57, 01 1984. doi:10.1145/971697.602266.

[106] S. Göktepe and E. Kuhl. Computational modeling of cardiac electrophysiology: A novel finite element approach. *International Journal for Numerical Methods in Engineering*, 79(2): 156–178, 2009. doi:https://doi.org/10.1002/nme.2571. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2571.

[107] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*, volume 4. 01 1985. ISBN 3-540-12761-5. doi:10.1007/978-3-662-02427-0.

[108] Jaroslav Haslinger and Yves Renard. A new fictitious domain approach inspired by the extended finite element method. *SIAM Journal on Numerical Analysis*, 47(2):1474–1499, 2009. doi:10.1137/070704435. URL https://doi.org/10.1137/070704435.

[109] Johannes Heinz, Peter Munch, and Manfred Kaltenbacher. High-order non-conforming discontinuous galerkin methods for the acoustic conservation equations. *International Journal for Numerical Methods in Engineering*, 124(9):2034–2049, 2023. doi:https://doi.org/10.1002/nme.7199. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.7199.

[110] Luca Heltai and Wenyu Lei. Adaptive finite element approximations for elliptic problems using regularized forcing data. *SIAM Journal on Numerical Analysis*, 61(2):431–456, March 2023.

[111] Luca Heltai and Nella Rotundo. Error estimates in weighted sobolev norms for finite element immersed interface methods. *Computers & Mathematics with Applications*, 78(11):3586–3604, 2019. ISSN 0898-1221. doi:https://doi.org/10.1016/j.camwa.2019.05.029. URL https://www.sciencedirect.com/science/article/pii/S0898122119302925.

[112] Luca Heltai, Wolfgang Bangerth, Martin Kronbichler, and Andrea Mola. Propagating geometry information to finite element computations. *ACM Trans. Math. Softw.*, 47(4), sep 2021. ISSN 0098-3500. doi:10.1145/3468428. URL https://doi.org/10.1145/3468428.

[113] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397—423, sep 2005. ISSN 0098-3500. doi:10.1145/1089014.1089021. URL https://doi.org/10.1145/1089014.1089021.

[114] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. 49:409–436 (1953), 1952. ISSN 0091-0635 (print), 2376-5305 (electronic).

[115] C.W Hirt, A.A Amsden, and J.L Cook. An arbitrary lagrangian-eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14(3):227–253, 1974. ISSN 0021-9991. doi:https://doi.org/10.1016/0021-9991(74)90051-5. URL https://www.sciencedirect.com/science/article/pii/0021999174900515.

[116] Cyril W. Hirt, Anthony A. Amsden, and J. L. Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics*, 135:203–216, 1997.

[117] Torsten Hoefler, Christian Siebert, and Andrew Lumsdaine. Scalable communication protocols for dynamic sparse data exchange. page 159–168, 2010. doi:10.1145/1693453.1693476.

[118] Julia M. Hoermann, Cristóbal Bertoglio, Martin Kronbichler, Martin R. Pfaller, Radomir Chabiniok, and Wolfgang A. Wall. An adaptive hybridizable discontinuous galerkin approach for cardiac electrophysiology. *International Journal for Numerical Methods in Biomedical Engineering*, 34(5):e2959, 2018. doi:https://doi.org/10.1002/cnm.2959. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.2959. e2959 cnm.2959.

[119] Thomas J.R. Hughes, Wing Kam Liu, and Thomas K. Zimmermann. Lagrangian-eulerian finite element formulation for incompressible viscous flows. *Computer Methods in Applied Mechanics and Engineering*, 29(3):329–349, 1981. ISSN 0045-7825. doi:https://doi.org/10.1016/0045-7825(81)90049-9. URL https://www.sciencedirect.com/science/article/pii/0045782581900499.

[120] James Hyman, Mikhail Shashkov, and Stanly Steinberg. The numerical solution of diffusion problems in strongly heterogeneous non-isotropic materials. *Journal of Computational Physics*, 132(1):130–148, 1997. ISSN 0021-9991. doi:https://doi.org/10.1006/jcph.1996.5633. URL https://www.sciencedirect.com/science/article/pii/S0021999196956338.

[121] Jeanne Joachim, Carole-Anne Daunais, Valérie Bibeau, Luca Heltai, and Bruno Blais. A parallel and adaptative nitsche immersed boundary method to simulate viscous mixing. *Journal of Computational Physics*, 488:112189, 2023. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2023.112189. URL https://www.sciencedirect.com/science/article/pii/S002199912300284X.

[122] August Johansson and Mats Larson. A high order discontinuous galerkin nitsche method for elliptic problems with fictitious boundary. *Numerische Mathematik*, 123, 04 2013. doi:10.1007/s00211-012-0497-1.

[123] Guido Kanschat. Multilevel methods for discontinuous Galerkin FEM on locally refined meshes. *Computers & Structures*, 82(28):2437–2445, 2004. ISSN 0045-7949. doi:https://doi.org/10.1016/j.compstruc.2004.04.015. URL https://www.sciencedirect.com/science/article/pii/S0045794904002603. Preconditioning methods: algorithms, applications and software environments.

[124] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998. doi:10.1137/S1064827595287997.

[125] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. 01 1997.

[126] Axel Klawonn, Olof B. Widlund, and Maksymilian Dryja. Dual-primal feti methods for three-dimensional elliptic problems with heterogeneous coefficients. *SIAM Journal on Numerical Analysis*, 40(1):159–179, 2002. doi:10.1137/S0036142901388081.

[127] Rolf Krause and Patrick Zulian. A parallel approach to the variational transfer of discrete fields between arbitrarily distributed unstructured finite element meshes. *SIAM Journal on Scientific Computing*, 38(3):C307–C333, 2016. doi:10.1137/15M1008361. URL https://doi.org/10.1137/15M1008361.

[128] Rolf Krause and Patrick Zulian. A parallel approach to the variational transfer of discrete fields between arbitrarily distributed unstructured finite element meshes. *SIAM Journal on Scientific Computing*, 38:C307–C333, 01 2016. doi:10.1137/15M1008361.

[129] M. Kronbichler and K. Ljungkvist. Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Transactions on Parallel Computing*, 6(1):2:1–32, 2019. doi:10.1145/3322813.

[130] M. Kronbichler and W. A. Wall. A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. *SIAM J. Sci. Comput.*, 40(5):A3423–A3448, 2018. doi:10.1137/16M110455X.

[131] Martin Kronbichler and Katharina Kormann. A generic interface for parallel cell-based finite element operator application. *Computers & Fluids*, 63:135–147, 2012. ISSN 0045-7930. doi:https://doi.org/10.1016/j.compfluid.2012.04.012. URL https://www.sciencedirect.com/science/article/pii/S0045793012001429.

[132] Martin Kronbichler and Per-Olof Persson. *Efficient High-Order Discretizations for Computational Fluid Dynamics*. 01 2021. ISBN 978-3-030-60609-1. doi:10.1007/978-3-030-60610-7.

[133] Martin Kronbichler and Wolfgang A. Wall. A performance comparison of continuous and discontinuous galerkin methods with fast multigrid solvers. *SIAM Journal on Scientific Computing*, 40(5):A3423–A3448, 2018. doi:10.1137/16M110455X. URL https://doi.org/10.1137/16M110455X.

[134] Edward Laughton, Gavin Tabor, and David Moxey. A comparison of interpolation techniques for non-conformal high-order discontinuous Galerkin methods. *Computer Methods in Applied Mechanics and Engineering*, 381:113820, 2021. doi:10.1016/j.cma.2021.113820.

[135] D. Lebrun-Grandié, A. Prokopenko, B. Turcksin, and S. R. Slattery. ArborX: A performance portable geometric search library. *ACM Trans. Math. Softw.*, 47(1), dec 2020. ISSN 0098-3500. doi:10.1145/3412558. URL https://doi.org/10.1145/3412558.

[136] Jan Mandel and Clark R. Dohrmann. Convergence of a balancing domain decomposition by constraints and energy minimization. *Numerical Linear Algebra with Applications*, 10(7): 639–659, 2003. doi:https://doi.org/10.1002/nla.341. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.341.

[137] Jan Mandel, Clark R. Dohrmann, and Radek Tezaur. An algebraic theory for primal and dual substructuring methods by constraints. *Applied Numerical Mathematics*, 54(2):167–193, 2005. ISSN 0168-9274. doi:https://doi.org/10.1016/j.apnum.2004.09.022. URL https://www.sciencedirect.com/science/article/pii/S0168927404001795. 6th IMACS.

[138] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos Papadopoulos, and Yannis Theodoridis. *R-Trees: Theory and Applications*. 01 2005. ISBN 978-1-85233-977-7. doi:10.1007/978-1-84628-293-5.

[139] André Massing, Mats G Larson, and Anders Logg. Efficient implementation of finite element methods on non-matching and overlapping meshes in 3d. *arXiv preprint arXiv:1210.7076*, 2012.

[140] André Massing, Mats G. Larson, and Anders Logg. Efficient implementation of finite element methods on nonmatching and overlapping meshes in three dimensions. *SIAM Journal on Scientific Computing*, 35(1):C23–C47, 2013. doi:10.1137/11085949X. URL https://doi.org/10.1137/11085949X.

[141] D.J. Mavriplis and V. Venkatakrishnan. *A 3D agglomeration multigrid solver for the Reynolds-averaged Navier-Stokes equations on unstructured meshes*. doi:10.2514/6.1995-345. URL https://arc.aiaa.org/doi/abs/10.2514/6.1995-345.

[142] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.0*, June 2021. URL https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf.

[143] Nicolas Moës, John Dolbow, and Ted Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1):131–150, 1999. doi:https://doi.org/10.1002/(SICI)1097-0207(19990910)46:1<131::AID-NME726>3.0.CO;2-J.

[144] Nicolas Moës, John Dolbow, and Ted Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46(1): 131–150, 1999. doi:https://doi.org/10.1002/(SICI)1097-0207(19990910)46:1<131::AID-NME726>3.0.CO;2-J. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0207%2819990910%2946%3A1%3C131%3A%3AAID-NME726%3E3.0.CO%3B2-J.

[145] Peter Munch, Timo Heister, Laura Prieto Saavedra, and Martin Kronbichler. Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. *ACM Trans. Parallel Comput.*, 10(1), 2023. ISSN 2329-4949. doi:10.1145/3580314. URL https://doi.org/10.1145/3580314.

[146] Steven Niederer, Eric Kerfoot, Alan Benson, Miguel Bernabeu, Olivier Bernus, Chris Bradley, Elizabeth Cherry, Richard Clayton, Flavio Fenton, Alan Garny, Elvio Heidenreich, Sander Land, Mary Maleckar, Pras Pathmanathan, Gernot Plank, Jos Rodrguez, Ishani Roy, Frank Sachse, Gunnar Seemann, and Nic Smith. Verification of cardiac tissue electrophysiology simulators using an n-version benchmark. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 369:4331–51, 11 2011. doi:10.1098/rsta.2011.0139.

[147] J. Nitsche. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abh. Math. Sem. Univ. Hamburg*, 36:9–15, 1971. doi:10.1007/BF02995904. URL https://doi.org/10.1007/BF02995904.

[148] B. O'Malley, J. Kópházi, R.P. Smedley-Stevenson, and M.D. Eaton. P-multigrid expansion of hybrid multilevel solvers for discontinuous Galerkin finite element discrete ordinate (DG-FEM-SN) diffusion synthetic acceleration (DSA) of radiation transport algorithms. *Progress in Nuclear Energy*, 98:177–186, 2017. ISSN 0149-1970. doi:https://doi.org/10.1016/j.pnucene.2017.03.014. URL https://www.sciencedirect.com/science/article/pii/S0149197017300598.

[149] Steven A. Orszag. Spectral methods for problems in complex geometries. *J. Comput. Phys.*, 37:70–92, 1980. doi:10.1016/0021-9991(80)90005-4.

[150] Y. Pan and P.-O. Persson. Agglomeration-based geometric multigrid solvers for compact discontinuous galerkin discretizations on unstructured meshes. *Journal of Computational Physics*, 449:110775, 2022. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2021.110775. URL https://www.sciencedirect.com/science/article/pii/S0021999121006707.

[151] P. Pathmanathan, M.O. Bernabeu, S.A. Niederer, D.J. Gavaghan, and D. Kay. Computational modelling of cardiac electrophysiology: explanation of the variability of results from different numerical solvers. *International Journal for Numerical Methods in Biomedical Engineering*, 28(8):890–903, 2012. doi:https://doi.org/10.1002/cnm.2467. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.2467.

[152] Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002. doi:10.1017/S0962492902000077.

[153] Roberto Piersanti, Pasquale C. Africa, Marco Fedele, Christian Vergara, Luca Dedè, Antonio F. Corno, and Alfio Quarteroni. Modeling cardiac muscle fibers in ventricular and atrial electrophysiology simulations. *Computer Methods in Applied Mechanics and Engineering*, 373:113468, 2021. ISSN 0045-7825. doi:https://doi.org/10.1016/j.cma.2020.113468. URL https://www.sciencedirect.com/science/article/pii/S0045782520306538.

[154] Daniele Prada and Micol Pennacchio. Algebraic multigrid methods for virtual element discretizations: A numerical study. *arXiv: Numerical Analysis*, 2018. URL https://api.semanticscholar.org/CorpusID:119714424.

[155] Thomas Radley, Paul Houston, and Matthew Hubbard. Quadrature-free polytopic discontinuous galerkin methods for transport problems. *Mathematics in Engineering*, 6:192–220, 03 2024. doi:10.3934/mine.2024009.

[156] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[157] Deep Ray and Jan S. Hesthaven. An artificial neural network as a troubled-cell indicator. *Journal of Computational Physics*, 367:166–191, 2018. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2018.04.029. URL https://www.sciencedirect.com/science/article/pii/S0021999118302547.

[158] F. Regazzoni, M. Salvador, P.C. Africa, M. Fedele, L. Dedè, and A. Quarteroni. A cardiac electromechanical model coupled with a lumped-parameter model for closed-loop blood circulation. *Journal of Computational Physics*, 457:111083, 2022. ISSN 0021-9991. doi:https://doi.org/10.1016/j.jcp.2022.111083. URL https://www.sciencedirect.com/science/article/pii/S0021999122001450.

[159] Beatrice Riviere. *Discontinuous Galerkin Methods For Solving Elliptic And Parabolic Equations: Theory and Implementation*, volume 35. 01 2008. doi:10.1137/1.9780898717440.

[160] J.M. Rogers and A.D. McCulloch. A collocation-galerkin finite element model of cardiac action potential propagation. *IEEE Transactions on Biomedical Engineering*, 41(8):743–757, 1994. doi:10.1109/10.310090.

[161] Open Cascade S.A.S. Opencascade technology, 2010. http://www.opencascade.org.

[162] R. I. Saye. High-order quadrature methods for implicitly defined surfaces and volumes in hyperrectangles. *SIAM Journal on Scientific Computing*, 37(2):A993–A1019, 2015. doi:10.1137/140966290.

[163] Georg Stadler and George Biros. Comparison of multigrid algorithms for high-order continuous finite element discretizations. *Numerical Linear Algebra with Applications*, 22, 02 2014. doi:10.1002/nla.1979.

[164] Elias M. Stein. *Singular Integrals and Differentiability Properties of Functions (PMS-30), Volume 30*. Princeton University Press, Princeton, 1971. ISBN 9781400883882. doi:doi:10.1515/9781400883882. URL https://doi.org/10.1515/9781400883882.

[165] Rolf Stenberg. On some techniques for approximating boundary conditions in the finite element method. *Journal of Computational and Applied Mathematics*, 63(1):139–148, 1995. ISSN 0377-0427. doi:https://doi.org/10.1016/0377-0427(95)00057-7. URL https://www.sciencedirect.com/science/article/pii/0377042795000577. Proceedings of the International Symposium on Mathematical Modelling and Computational Methods Modelling 94.

[166] Simon Sticko and Gunilla Kreiss. A stabilized nitsche cut element method for the wave equation. *Computer Methods in Applied Mechanics and Engineering*, 309:364–387, September 2016. doi:10.1016/j.cma.2016.06.001. URL https://doi.org/10.1016/j.cma.2016.06.001.

[167] Klaus Stuben. Algebraic multigrid (amg): Experiences and comparisons. *Appl. Math. Comput.*, 13(3-4):419–451, jan 1983. ISSN 0096-3003. doi:10.1016/0096-3003(83)90023-1. URL https://doi.org/10.1016/0096-3003(83)90023-1.

[168] N. Sukumar and A. Tabarraei. Conforming polygonal finite elements. *Internat. J. Numer. Methods Engrg.*, 61(12):2045–2066, 2004. ISSN 0029-5981.

[169] Hari Sundar, Georg Stadler, and George Biros. Comparison of multigrid algorithms for high-order continuous finite element discretizations. *Numerical Linear Algebra with Applications*, 22(4):664–680, 2015. doi:https://doi.org/10.1002/nla.1979.

[170] K. H. W. J. ten Tusscher and A. V. Panfilov. Alternans and spiral breakup in a human ventricular tissue model. *American Journal of Physiology-Heart and Circulatory Physiology*, 291(3): H1088–H1100, 2006. doi:10.1152/ajpheart.00109.2006.

[171] K. H. W. J. ten Tusscher, D. Noble, P. J. Noble, and A. V. Panfilov. A model for human ventricular tissue. *American Journal of Physiology-Heart and Circulatory Physiology*, 286(4): H1573–H1589, 2004. doi:10.1152/ajpheart.00794.2003.

[172] The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5 edition, 2022. URL https://doc.cgal.org/5.5/Manual/packages.html.

[173] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Elsevier Academic Press, London, 2001.

[174] V. Venkatakrishnan and D. J. Mavriplis. Agglomeration multigrid for the three-dimensional euler equations. *AIAA Journal*, 33(4):633–640, 1995. doi:10.2514/3.12625. URL https://doi.org/10.2514/3.12625.

[175] P. Wesseling and Cornelis Oosterlee. Geometric multigrid with application to computational fluid dynamics. *Journal of Computational and Applied Mathematics*, 128:311–334, 03 2001. doi:10.1016/S0377-0427(00)00517-3.

[176] J.P. Whiteley. An efficient numerical technique for the solution of the monodomain and bidomain equations. *IEEE Transactions on Biomedical Engineering*, 53(11):2139–2147, 2006. doi:10.1109/TBME.2006.879425.