Algebraic multigrid preconditioner for statically condensed systems arising from lowest-order hybrid discretizations

Daniele Antonio Di Pietro, Frank Hülsemann, Pierre Matalon, Paul Mycek, Ulrich Rüde

ALGEBRAIC MULTIGRID PRECONDITIONER FOR STATICALLY CONDENSED SYSTEMS ARISING FROM LOWEST-ORDER HYBRID DISCRETIZATIONS *

DANIELE A. DI PIETRO†, FRANK HÜLSEMANN‡, PIERRE MATALON†§¶‖, PAUL MYCEK¶, AND ULRICH RÜDE¶‖

Abstract. We address the numerical solution of linear systems arising from the hybrid discretizations of second-order elliptic partial differential equations (PDEs). Such discretizations hinge on a hybrid set of degrees of freedom (DoFs), respectively defined in cells and faces, which naturally gives rise to a global *hybrid* system of linear equations. Assuming that the cell unknowns are only locally coupled, they can be efficiently eliminated from the system, leaving only face unknowns in the resulting Schur complement, also called *statically condensed* matrix. We propose in this work an algebraic multigrid (AMG) preconditioner specifically targeting condensed systems corresponding to lowest order discretizations (piecewise constant). Like traditional AMG methods, we retrieve geometric information on the coupling of the DoFs from algebraic data. However, as the condensed matrix only gives information on the faces, we use the *uncondensed* version to reconstruct the connectivity graph between elements and faces. An aggregation-based coarsening strategy mimicking a geometric coarsening or semi-coarsening can then be set up to build coarse levels. Numerical experiments are performed on diffusion problems discretized by the Hybrid High-Order (HHO) method at the lowest order. Our approach uses a K-cycle to precondition an outer flexible Krylov method. The results demonstrate similar performances, in most cases, compared to a standard AMG method, and a notable improvement on anisotropic problems with Cartesian meshes.

Key words. Algebraic multigrid, hybrid methods, static condensation.

AMS subject classifications. 65N55, 65N22, 65F50, 65F08

1. Introduction.

Hybrid discretizations have been part of the landscape of numerical methods to solve Partial Differential Equations (PDEs) since the seventies. In his 1978 book [8, p. 421], P. G. Ciarlet states the following definition: "*we may define (...) as a* hybrid method *any finite element method based on a formulation where one unknown is a function, or some of its derivatives, on the set* Ω*, and the other unknown is the trace of some of its derivatives of the same function, or the trace of the function itself, along the boundaries of the set* K" (Ω representing the domain of study and K a mesh element). Although hybridization of finite element methods first appeared as an implementation trick [40], it was later proven [1] that the new unknowns at faces, introduced as Lagrange multipliers, held additional information on the exact solution, which could be exploited to improve the accuracy of the numerical approximation. A large number of finite element schemes have given rise to hybrid counterparts, starting with the mixed formulations of Raviart-Thomas (RT) [30] and Brezzi-Douglas-Marini (BDM) [6]. More recently, in the context of Discontinuous Galerkin (DG) methods, hybridization was also used to overcome its main drawback, namely, the large number of unknowns resulting from the lack of continuity at element interfaces. Indeed,

*Submitted to the editors June 28th, 2021.

Funding: ANR project Fast4HHO under contract ANR-17-CE23-0019.

†IMAG, Univ. Montpellier, CNRS, Montpellier, France (pierre.matalon@gmail.com)

‡EDF R&D, Paris-Saclay, France

§IRIT, Toulouse, France

¶CERFACS, Toulouse, France

‖FAU, Erlangen-Nürnberg, Germany

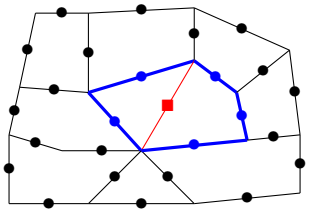
hybridization allows for the local elimination of cell-based unknowns from the global system, leaving the face unknowns as the only remaining ones in the resulting Schur complement, also called *statically condensed* or *trace* system. Examples of methods whose DoFs verify this structural property include, in particular, Hybridizable Discontinuous Galerkin (HDG), Compatible Discrete Operators (CDO) [3], Hybrid High-Order (HHO) methods [14, 13], Mimetic Finite Differences (MFD) [2], Mixed and Hybrid Finite Volumes (MHFV) [17, 19, 18]. For a more extensive introduction to hybrid methods and hybridization, we refer to the preface of [12] and the first pages of [10].

Algebraic multigrid (AMG) solvers [20, 33] are very popular for the solution of large linear systems arising from the discretization of elliptic equations on unstructured meshes. Unlike geometric multigrid methods, which require a hierarchy of meshes of different granularity, algebraic algorithms classically do not need more information than the linear system to solve. Discarding all geometric information as input parameter results in the most appreciated feature of these methods, that is, their usability in a black-box fashion.

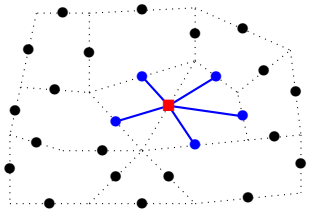
The availability of an easy-to-use, scalable linear solver is essential to help popularize novel discretization methods with the industrial actors, to whom it is crucial to efficiently solve problems of large size. Adopting a new discretization in an industrial context requires heavy preliminary testing, that can be facilitated if the software for the appropriate solver is already available on the market or if its development can easily be externalized. Being isolated from the mesh, which can be generated, stored, and transferred in numerous ways, AMG solvers ally interoperability and performance. Although novel hybrid methods like HHO have gained growing interest in recent years, thus pushing the development of ad-hoc geometric multigrid algorithms [9, 22, 39, 15] or other iterative methods [35, 23], we are not aware of any AMG specifically targeting condensed systems arising from such discretizations at this time.

Usual AMG solvers designed for low-order finite element or finite difference methods infer mesh information under the assumption that each row in the matrix corresponds to a DoF located at a *mesh node* or *element*. Thus, the connectivity graph of the mesh can be reconstructed algebraically, and coarsening strategies mimicking geometric algorithms can then be performed in order to build the coarse levels. Algebraic algorithms are commonly separated in two families according to how their coarsening strategies can be geometrically interpreted. In the first one, one defines the coarse unknowns as a subset of the fine ones. Geometrically, in an isotropic setting, it consists of selecting fine nodes to keep on the coarse mesh, in such a way that the domain is still uniformly covered while the number of nodes is significantly reduced. This approach has given rise to the so-called Classical AMG (also referred to as C/F AMG) [31, 32], of which BoomerAMG [21] can be mentioned as a popular implementation. The other family regroups aggregation-based methods [7, 4, 27, 25]. In such methods, unknowns are now respectively assimilated to node-defined DoFs (or DoFs within distinct elements), which can then be agglomerated to define a coarse mesh. Among the well-known representatives of aggregation-based AMG software packages, one can cite AGMG [28]. We refer to [34] for a numerical comparison of both approaches applied on a specific application of the Navier–Stokes equations. In the present work, we especially focus on aggregation-based methods. In our hybrid setting at the lowest order, the unknowns of the system are actually linked to faces, i.e. neither nodes nor elements. Consequently, at first glance it might seem peculiar, from a geometrical point of view, to apply the above approaches in this context.

Indeed, looking at the example stencil illustrated by Figure 1.1a, aggregation-based coarsening might (and sometimes actually does) aggregate the red DoF with the blue one located the further on its right. As their respective edges do not touch, it is difficult to perceive a geometrical sense in this aggregation. Nonetheless, numerical tests with AGMG show that the approach still works well, which can be geometrically justified by forgetting about the DoFs being actually face-defined and considering them as mere nodal values. See Figure 1.1b for an illustration of the algebraic stencil as perceived by standard AMG methods. That being said, one can legitimately wonder if a coarsening strategy making geometrical sense in light of the actual meaning of the DoFs as face-defined values could not yield even better results.



(a) Geometric, face-aware view of the stencil



(b) Algebraic view of the stencil (unaware of the faces), as perceived by a standard AMG method

Fig. 1.1: Stencil given by the statically condensed matrix at the lowest order. Solid points represent DoFs, which, in this context, are located at faces (edges here). The considered DoF is represented by a red square. Its stencil is highlighted by thick blue lines and blue DoFs.

The idea at the origin of the present work is the algebraic reconstruction of the mesh information based, not on the condensed matrix, but on the uncondensed one, which contains the connectivity graph between elements and faces. Note that it implies that this method requires more information than the sole system to solve. Parts of the uncondensed matrix must indeed be brought to the algorithm as additional information, which makes the method less "black-box", but still purely algebraic. Among similar approaches, one can cite AMGe [5]. Once the so-called algebraic mesh is retrieved, especially the neighbouring information between elements, an *element*-based aggregation method can be set up in order to mimic the behaviour of a geometric coarsening or semi-coarsening strategy. Although the construction of the coarse levels is mainly based on *plain* aggregation principles, the prolongation operator also uses techniques borrowed from *smoothed aggregation* methods [36, 37, 29].

AMG methods directly used as solvers may lack efficiency [38, p. 663][24]. Using them as preconditioners for a Krylov method is generally favored. Moreover, plain aggregation methods also suffer from slower convergence than Classical AMG in a V-cycle. To handle these issues, we adopt the choices made by AGMG [27]. Namely, we use the so-called K-cycle, which introduces Krylov acceleration into the multigrid recursive cycle. Secondly, one such cycle is used to precondition an outer Krylov method. As the K-cycle does not yield a constant preconditioner, the outer iteration is required to be *flexible*. More generally, the technical choices made in this work are borrowed from AGMG (pairwise aggregation, strong negative coupling criterion,

K-cycle...) in order to establish a proper comparison with a standard AMG solver that relies only on the condensed system.

The rest of this work is organized as follows. Section 2 lists the features we assume for the underlying discretization to fit our method. Section 3 describes the construction of our algebraic multigrid algorithm. In Section 4, we apply our method to the lowest order HHO discretization of homogeneous and heterogeneous diffusion problems in 2D and 3D. The outer solver is a Flexible Conjugate Gradient, preconditioned with our algebraic multigrid in conjunction with the K-cycle: compared to a standard aggregation-based AMG, we report equivalent performances in CPU time, an enhanced robustness to anisotropy on Cartesian meshes, and a similar quasi-optimal asymptotic behaviour. Finally, we discuss limitations and future work in the concluding Section 5.

2. Assumptions. We consider a scalar elliptic PDE over a domain discretized by a polytopal mesh. For simplicity, we suppose Dirichlet boundary conditions. We assume that the PDE is discretized by a lowest-order hybrid discretization method DoFs corresponding to one scalar value per cell and per face. Throughout this work, the subscript T (resp. F) will consistently refer to the cell-based (resp. face-based) quantities. We also assume that the global *uncondensed* linear system arising from the hybrid discretization at hand is symmetric positive definite, of the form

$$\begin{pmatrix} A_{TT} & A_{TF} \\ A_{TF}^\top & A_{FF} \end{pmatrix} \begin{pmatrix} x_T \\ x_F \end{pmatrix} = \begin{pmatrix} b_T \\ b_F \end{pmatrix}, \tag{2.1}$$

from which Dirichlet boundary unknowns have been eliminated, and where A_{TT} represents the coupling among cell-DoFs, A_{TF} between cell- and face-DoFs, and A_{FF} among face-DoFs. Assuming the discretization is such that the cell unknowns are only locally coupled, A_{TT} is diagonal, and thus inexpensive to invert. The statically condensed system resulting from the local elimination of the cell unknowns is

$$\widetilde{A} x_F = \widetilde{b}, \qquad \widetilde{A} := A_{FF} - A_{TF}^\top A_{TT}^{-1} A_{TF}, \quad \widetilde{b} := b_F - A_{TF}^\top A_{TT}^{-1} b_T. \tag{2.2}$$

As a Schur complement, $\widetilde{A}$ is also symmetric positive definite.

3. Algebraic multigrid. We propose to construct an algebraic multigrid method to solve the condensed system (2.2) by using the coupling information given in the uncondensed matrix (2.1). We base our multigrid algorithm on ingredients classically used in aggregation-based AMG. AGMG [27] will serve as a reference for specific technical choices such as the pairwise aggregation, the strong negative coupling criterion, the Krylov acceleration in the multigrid cycle. We also take inspiration from the good results of the geometric multigrid algorithm [16] for the adaptation of the coarsening strategy to the hybrid setting, as well as for the multigrid prolongation operator.

3.1. Construction of the algebraic mesh. It is straightforward to algebraically reconstruct the geometric relationships using the connectivity graph given by A_{TF}. Rows of A_{TF} correspond to elements, while columns correspond to faces. Adopting the notation $[1, n] := \{1, \dots, n\}$ for all $n \in \mathbb{N}_+^*$, we then define the set of element indices $T := [1, n_T]$ (resp. the set of face indices $F := [1, n_F]$) where n_T (resp. n_F) is the number of rows (resp. columns) of A_{TF}. For each $i \in T$, the locations of the non-zero coefficients in the i-th row of A_{TF} correspond to the associated face indices, which we collect in the set $F_i \subset F$. Reciprocally, for all $k \in F$, we collect in $T_k \subset T$ the element indices that contain in their boundary the face of index k. In

A_{TF}, two different rows having a non-zero entry in the same column correspond to neighbouring elements. Their interface is given by the faces algebraically defined by the indices of such columns. Formally, for all $(i,j) \in T^2$, i and j are neighbours if $F_i \cap F_j \neq \emptyset$. Moreover, for all $i \in T$ and $k \in F_i$, we denote by σ_{ik} the neighbour of i relative to the face k. Algorithm 3.1 summarizes the process.

Algorithm 3.1 `BuildMesh`

Input: A_{TF}

Output: Mesh defined as the dataset $M := (T, F, (F_i)_{i\in T}, (T_k)_{k\in F}, (\sigma_{ik})_{(i,k)\in T\times F})$

1: $n_T := \text{rows}(A_{TF})$; $T := [1, n_T]$
2: $n_F := \text{cols}(A_{TF})$; $F := [1, n_F]$
3: **for** $i \in T$ **do** $F_i := \{k \in F \mid (A_{TF})_{ik} \neq 0\}$ **end for**
4: **for** $k \in F$ **do** $T_k := \{i \in T \mid (A_{TF})_{ik} \neq 0\}$ **end for**
5: **for** $i \neq j \in T$ **do**
6: **if** $F_i \cap F_j \neq \emptyset$ **then**
7: $\forall k \in F_i \cap F_j$, set $\sigma_{ik} := j$ and $\sigma_{jk} := i$
8: **end if**
9: **end for**

3.2. Mesh coarsening by element aggregation and face collapsing. Now that we have built the algebraic mesh, that is, a list of elements, a list of faces, as well as the links between them and subsequently the neighbouring relationships, we are able to algebraically reproduce a geometric *element-based* aggregation strategy. The framework of the present contribution does not restrict the aggregation method, as long as the required information for choosing the aggregates can be retrieved from the uncondensed system. That is why the way the elements are agglomerated will remain abstract in the general algorithm. As such, Algorithm 3.2, which describes the global process of element aggregation, refers to the abstract function `BuildAggregate` (at step 6). `BuildAggregate` takes an element $i \in T$ as an argument and returns a list of elements (including i) chosen to form an aggregate. The simplest aggregation method, corresponding to clustering i with all its unaggregated neighbours, would be enough to put our algorithm to the test. However, it would only rely on the element connectivity graph, i.e. on the location of the non-zero coefficients in the block A_{TF}, regardless of their values. In order to manage anisotropic problems and give an example of how semi-coarsening can be performed in our hybrid setting, we give in subsection 3.3 a hybrid counterpart of the node-defined pairwise aggregation based on the *strong negative coupling* criterion, as it is formulated in the early version of AGMG described by [27]. We denote by $(G_{T,i})_{i\in[1,n_{T,c}]}$ the produced aggregates, with $n_{T,c}$ defining the number of aggregates.

In a multigrid method that applies to trace systems, as the smoother operates on the face unknowns, the efficient reduction of the low-frequency components of the error relies on accessing coarse representations of the face-defined functions. This implies that *faces* must be coarsened between levels (see [15, §4.4.3]), which is a new constraint imposed to any suited coarsening strategy. Consequently, we combine the element aggregation with an additional step of *face aggregation*, also called *face collapsing*. In particular, we reproduce the technique devised in [16], which consists of merging into single faces the interfaces between aggregates.

During the element aggregation process, the fine faces are split into two disjoint subsets $\mathring{F} \cup \widehat{F} = F$ according to their situation with respect to the aggregates. $\mathring{F}$

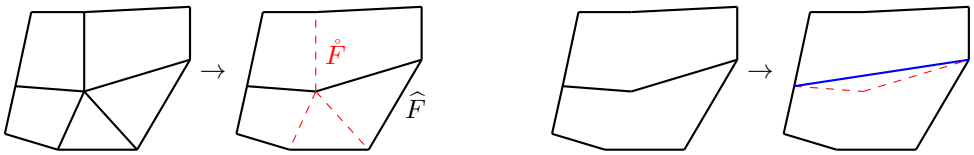
Algorithm 3.2 `ElementAggregation`

Input: Mesh, output of Algorithm 3.1

Output: Aggregation information $\mathcal{G}_T$, defined as the collection of data:
$(G_{T,i})_{i\in[1,n_{T,c}]}$: element aggregates
$(g_i)_{i\in[1,n_T]}$: association of the element i to the aggregate g_i it belongs to
$(\mathring{F}_i)_{i\in[1,n_{T,c}]}$: fine faces interior to the aggregates
$(\widehat{F}_i)_{i\in[1,n_{T,c}]}$: fine faces at the boundary of the aggregates

`Todo` $:= T$ // remaining non-aggregated elements
$n := 0$ // aggregate index
while `Todo` $\neq \emptyset$ **do**
  Select $i \in$ `Todo`
  $n := n + 1$
  $G_{T,n} :=$ `BuildAggregate`$(i,$ `Todo`$)$ // see Algorithm 3.4 for a possible algo.
  for $j \in G_{T,n}$ **do** // save for each fine element the aggregate it is in
    $g_j := n$
  end for
  $\mathring{F}_n := \{k \in \bigcup_{i\in G_{T,n}} F_i \mid \exists i \neq j \in G_{T,n} \text{ s.t. } k \in F_i \cap F_j\}$ // interior faces
  $\widehat{F}_n := \left(\bigcup_{i\in G_{T,n}} F_i\right) \setminus \mathring{F}_n$ // boundary faces
  `Todo` $:=$ `Todo` $\setminus G_{T,n}$
end while
$n_{T,c} := n$

regroups the faces interior to an aggregate, i.e. the faces shared by two elements aggregated together. Geometrically speaking, those faces are "removed" to give rise to the aggregates. The remaining faces, which compose the aggregates' boundaries, are collected in $\widehat{F}$. We also denote their local counterparts, with respect to each aggregate, by $(\mathring{F}_i)_{i\in[1,n_{T,c}]}$ and $(\widehat{F}_i)_{i\in[1,n_{T,c}]}$. See Figure 3.1a for a geometric illustration.



(a) Element aggregation (b) Face collapsing

Fig. 3.1: Aggregation process with face collapsing. In (a), elements are aggregated, yielding two aggregates. "Removed" edges, represented in dashed red lines, are collected in $\mathring{F}$, while the remaining ones are collected in $\widehat{F}$. Then, in (b), the interface between the two neighbouring aggregates, here made of two edges (in dashed red lines), is collapsed into a single one (solid blue line). The other edges yield *singleton* face aggregates.

Neighbouring relationships between element aggregates can be directly deduced from $\widehat{F}$. We can then collapse the interfaces between aggregates into one single face

without altering the coarse adjacency graph. Note that each interface, whether it is made of multiple faces or only one, gives rise to one face aggregate, so singleton aggregates are produced. Figure 3.1b gives a geometric interpretation of the face collapsing, and Algorithm 3.3 formalizes the process.

Algorithm 3.3 `FaceCollapsing`

Input: Fine mesh, output of Algorithm 3.1
Aggregation information, output of Algorithm 3.2

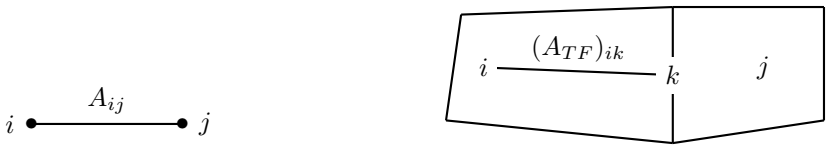
Output: Face collapsing information $\mathcal{G}_F$, defined as the collection of data:
$(G_{F,k})_{k\in[1,n_{F,c}]}$: face aggregates
$(H_i)_{i\in[1,n_{T,c}]}$: collapsed faces defining the new boundaries of the aggregates

1: `Todo` $:= \widehat{F}$ // fine faces to process
2: $m := 0$ // face aggregate index
3: **while** `Todo` $\neq \emptyset$ **do**
4: Select $k \in$ `Todo`
5: $m := m + 1$
6: Let $G := \bigcup_{i\in T_k} g_i$ // the aggregates the face k is at the interface of
7: $G_{F,m} := \bigcap_{n\in G} \widehat{F}_n$ // fine faces (including k) composing that interface
8: **for** $n \in G$ **do**
9: $H_n := H_n \cup \{m\}$ // in the coarse mesh, m is now a face of the aggregate n
10: **end for**
11: `Todo` $:=$ `Todo` $\setminus G_{F,m}$
12: **end while**
13: $n_{F,c} := m$

Now that aggregates have been made for elements and faces, they can be numbered and become the coarse elements and faces, thus defining a coarse mesh.

3.3. Pairwise aggregation by strong negative coupling. This strategy allows one to aggregate pairs of neighbouring elements in the direction of strong anisotropy, and gives an implementation of the abstract method `BuildAggregate` at step 6 of Algorithm 3.2. In standard AMG, the usual rule of *negative coupling* governs the choice of the aggregated neighbours. Here, this rule is adapted to hybrid unknowns. For each element, it allows one to evaluate, for all of its neighbours, a numerical criterion indicating their strength of connection. Only those which have a strong enough connection and are not already aggregated are considered for aggregation. Among them, the strongest one is chosen, and leads to a pair aggregate. However, if none of the strong neighbours are available, i.e. they have all already been previously aggregated, then the element stays alone in a so-called *singleton* aggregate.

Before introducing our hybrid criterion for the strong negative relationship, let us recall the *node-defined* criterion used by standard AMG methods. As multiple variations of this criterion exist, we follow the example of AGMG in the version of [27]. Given the stiffness matrix A and an algebraic node i associated to the i–th row of A, the coupling coefficient modelling the connection of j to i is provided by the matrix entry A_{ij} (see Figure 3.2a). We say that j is negatively coupled (or connected) to i if $A_{ij} < 0$, and the strength of connection is defined by the modulus of that coefficient. The strongest connection then corresponds to $\overline{c}_i := \max_{j|A_{ij}<0} |A_{ij}|$. Given a weak/strong connection threshold $0 < \beta \leq 1$ (typically set to 0.25), the set of nodes strongly connected to i is $\{j \mid A_{ij} < 0 \text{ and } |A_{ij}| \geq \beta\, \overline{c}_i\}$.



(a) Coupling of nodes i and j in standard AMG.

(b) Coupling of elements i and j via their common face k in the hybrid setting.

Fig. 3.2: Coupling values in standard and hybrid settings.

In our case, in hybrid form, elements are coupled through A_{TF}. Specifically, given an element of index $i \in T$ and its neighbour of index j, the coupling coefficient is provided through their common face of index k by the matrix coefficient $(A_{TF})_{ik}$; see Figure 3.2b. We then introduce the following definition for the negative coupling criterion: j is negatively coupled to i via k if $(A_{TF})_{ik} < 0$. Now, for the purpose of managing heterogeneous problems by preventing aggregation across large jumps in the diffusion coefficient, we remark that this sole value is not enough to detect a discontinuity between i and j. Indeed, $(A_{TF})_{ik}$ only bears information local to i. We also notice that, in the heterogeneous isotropic diffusion case, the coefficient $(A_{TF})_{ik}$ is scaled by the actual diffusion coefficient of the element of index i. We then introduce the heterogeneity ratio between the elements of indices i and j connected by the face of index k as

$$\rho_{ij} := \max\left(\frac{(A_{TF})_{ik}}{(A_{TF})_{jk}}, \frac{(A_{TF})_{jk}}{(A_{TF})_{ik}}\right) \geq 1. \tag{3.1}$$

Meaning to penalize aggregation across jumps, instead of simply defining the coupling strength by $|(A_{TF})_{ik}|$, we define it as

$$c_{ik} := |(A_{TF})_{ik}|/\rho_{i\sigma_{ik}},$$

where we recall that σ_{ik} refers to the element index j that shares the face index k with i. According to this criterion, the remaining definitions are straightforward. The strongest connection to i is given by

$$\overline{c}_i := \max_{k \in F_i, (A_{TF})_{ik} < 0} c_{ik},$$

and the set of faces strongly connected to i by

$$\underline{F}_i := \{k \in F_i \mid (A_{TF})_{ik} < 0 \text{ and } c_{ik} \geq \beta\, \overline{c}_i\}. \tag{3.2}$$

Finally, the strong neighbours of i may be retrieved in the set $\{\sigma_{ik}, k \in \underline{F}_i\}$. The corresponding implementation of the abstract function `BuildAggregate` is given by Algorithm 3.4.

Notice that the number of singleton aggregates can significantly vary depending on the order following which the elements are aggregated. So, to minimize the number of singleton aggregates, the elements are beforehand parsed and attributed a priority value in order to favor those that have the fewest strong neighbours. In particular, we follow the priority numbering algorithm described in [11] and process elements by order of priority at step 4 of Algorithm 3.2.

Algorithm 3.4 `BuildAggregate`

Input: $i \in T$: element to aggregate
$\widetilde{T} \subset T$: non-aggregated elements
Output: G: aggregate

1: // Collect faces strongly connected to i through which neighbours are still available
2: $\widetilde{F}_i := \{k \in \underline{F}_i \mid \sigma_{ik} \in \widetilde{T}\}$ // cf. (3.2) for the definition of $\underline{F}_i$
3: **if** $\widetilde{F}_i \neq \emptyset$ **then**
4: $\quad k := \arg\max_{\ell \in \widetilde{F}_i} c_{i\ell}$ // face with the strongest coupling
5: $\quad G := \{i, \sigma_{ik}\}$ // aggregation of i and its neighbour relative to k
6: **else**
7: $\quad G := \{i\}$ // i forms a singleton aggregate
8: **end if**

3.4. Cell- and face-defined auxiliary prolongation operators. Given $T := [1, n_T]$ (resp. $F := [1, n_F]$) the fine element (resp. face) indices in the algebraic mesh, we denote by $T_c := [1, n_{T,c}]$ (resp. $F_c := [1, n_{F,c}]$) the coarse elements (resp. faces) constructed by the aggregation process of subsection 3.2. We start by defining an auxiliary cell-defined prolongation matrix Q_T (of size $n_T \times n_{T,c}$) in the manner of plain aggregation:

$$\forall i \in T,\ \forall j \in T_c, \quad (Q_T)_{ij} := \begin{cases} 1 & \text{if } i \in G_{T,j} \\ 0 & \text{otherwise.} \end{cases} \tag{3.3a}$$

This highly sparse prolongation operator (exactly 1 non-zero per row) transfers the unknown values respectively assigned to the coarse elements onto the fine elements they aggregate. Without smoothed aggregation techniques, all fine elements of the same aggregate receive the same value. Regarding the faces, we define the auxiliary prolongation matrix Q_F (of size $n_F \times n_{F,c}$) such that for $k \in F$,

(i) if $k \in \widehat{F}$, i.e. k belongs to a face aggregate,

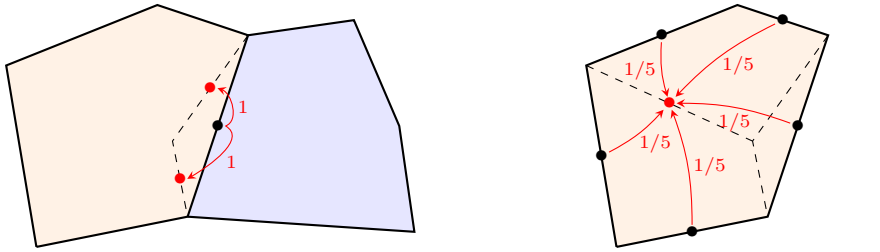
$$\forall \ell \in F_c, \quad (Q_F)_{k\ell} := \begin{cases} 1 & \text{if } k \in G_{F,\ell} \\ 0 & \text{otherwise;} \end{cases} \tag{3.3b}$$

(ii) if $k \in \mathring{F}$, let m be the coarse element embedding k (i.e. $k \in \mathring{F}_m$)) and H_m its set of (potentially) collapsed faces; then,

$$\forall \ell \in F_c, \quad (Q_F)_{k\ell} := \begin{cases} 1/\operatorname{card}(H_m) & \text{if } \ell \in H_m \\ 0 & \text{otherwise,} \end{cases} \tag{3.3c}$$

where $\operatorname{card}(\cdot)$ returns the number of elements. To summarize, an aggregated face takes the value of the corresponding coarse aggregate (just like the elements), and a "removed" face, embedded in a coarse element, takes the average value of that coarse element's faces; see Figure 3.3.

3.5. Multilevel hierarchy. As the method described in subsection 3.2 does not necessarily yield an aggressive enough coarsening [27], and also in order to build more levels for the multigrid hierarchy, we want to repeat the coarsening process, thus



(a) Aggregated faces ($\in \widehat{F}$) (b) Interior/removed face ($\in \mathring{F}$)

Fig. 3.3: Operator Q_F. The fine red DoFs are set by the coarse black ones.

defining the so-called *multiple* coarsening. To do so, one has to define a coarse version of the uncondensed matrix (2.1) to allow recursive execution.

Given the initial blocks A_{TT}, A_{TF} and A_{FF} of the fine uncondensed matrix, we use the auxiliary prolongation operators introduced in subsection 3.4 to define coarse counterparts in a Galerkin fashion:

$$\begin{pmatrix} A_{TT,c} & A_{TF,c} \\ A_{TF,c}^\top & A_{FF,c} \end{pmatrix} := \begin{pmatrix} Q_T & \\ & Q_F \end{pmatrix}^\top \begin{pmatrix} A_{TT} & A_{TF} \\ A_{TF}^\top & A_{FF} \end{pmatrix} \begin{pmatrix} Q_T & \\ & Q_F \end{pmatrix}. \tag{3.4}$$

Note that in practice, only the blocks used in the algorithm must be assembled. In this work, we only need A_{TF} (for the coarsening strategy) and A_{TT} (used in the multigrid prolongation operator P_F further described in subsection 3.6).

Algorithm 3.5 describes one step of coarsening. In addition to building the coarse blocks (step 5), the coarsening process also constructs the operator P_F that will be used as a prolongation operator in the multigrid algorithm (step 6). Indeed, although Q_F could be employed for that purpose, we choose to explore another, more efficient approach (the construction of the operator P_F is described in subsection 3.6 below). Furthermore, the coarse operator for the lower level of the multigrid algorithm is defined as the Galerkin operator, constructed from P_F and the condensed matrix $\widetilde{A}$, initialized at the finest level by the Schur complement (2.2) (step 7). Finally, to be more consistent with this coarse operator, we recompute the coarse blocks following formula (3.4) in which Q_F is replaced with P_F, i.e.

$$\begin{pmatrix} A_{TT,c} & A_{TF,c} \\ A_{TF,c}^\top & A_{FF,c} \end{pmatrix} := \begin{pmatrix} Q_T & \\ & P_F \end{pmatrix}^\top \begin{pmatrix} A_{TT} & A_{TF} \\ A_{TF}^\top & A_{FF} \end{pmatrix} \begin{pmatrix} Q_T & \\ & P_F \end{pmatrix}. \tag{3.5}$$

Again, only the blocks actually needed, here $A_{TF,c}$, are recomputed; see step 8.

While Algorithm 3.5 performs one step of coarsening, Algorithm 3.6 handles the recursion until a targeted coarsening factor is reached. The end of this multiple coarsening process defines one new multigrid level, and the two-level prolongation operator is defined by successively chaining the prolongation operators coming out of each coarsening (step 7).

3.6. Multigrid prolongation operator. Although Q_F could also be used as a prolongation operator for the multigrid algorithm, we choose to explore another approach, which happens to give better results. Thus, we would like to emphasize that Q_F is only employed to build the coarse blocks during the setup phase, while

Algorithm 3.5 `Coarsening`

Input: A_{TT}, A_{TF}, $\widetilde{A}$
Output: $A_{TT,c}$, $A_{TF,c}$, $\widetilde{A}_c$, P_F

$M := \texttt{BuildMesh}(A_{TF})$ // Algorithm 3.1
$\mathcal{G}_T := \texttt{ElementAggregation}(M)$ // Algorithm 3.2
$\mathcal{G}_F := \texttt{FaceCollapsing}(M, \mathcal{G}_T)$ // Algorithm 3.3
Compute Q_T and Q_F by (3.3)
$A_{TT,c} := Q_T^\top A_{TT} Q_T$; $A_{TF,c} := Q_T^\top A_{TF} Q_F$ // cf. (3.4)
Compute P_F by (3.8)
$\widetilde{A}_c := P_F^\top \widetilde{A} P_F$
$A_{TF,c} := Q_T^\top A_{TF} P_F$ // cf. (3.5)

Algorithm 3.6 `MultipleCoarsening`

Input: A_{TT}, A_{TF}, $\widetilde{A}$, `targetCF`
Output: $A_{TT,c}$, $A_{TF,c}$, $\widetilde{A}_c$, P_F

$A_{TT,\text{aux}} := A_{TT}$; $A_{TF,\text{aux}} := A_{TF}$
$\widetilde{A}_{\text{aux}} := \widetilde{A}$
$P_F := I$
`cf` $:= 0$ // coarsening factor
while `cf` $<$ `targetCF` **do**
  $[A_{TT,c}, A_{TF,c}, \widetilde{A}_c, P_{F,\text{aux}}] := \texttt{Coarsening}(A_{TT,\text{aux}}, A_{TF,\text{aux}}, \widetilde{A}_{\text{aux}})$ // Algorithm 3.5
  $P_F := P_F P_{F,\text{aux}}$
  $A_{TT,\text{aux}} := A_{TT,c}$; $A_{TF,\text{aux}} := A_{TF,c}$
  $\widetilde{A}_{\text{aux}} := \widetilde{A}_c$
  `cf` $:= \text{cols}(A_{TF})/\text{cols}(A_{TF,c})$
end while

P_F, described in this section, defines the prolongation operator used in the multigrid iterations. It is meant to be an algebraic counterpart of the geometric prolongation operator defined in [15], which relies on the decondensation of the cell unknowns.

First, we introduce a preliminary prolongation operator denoted by $P_F^{(0)}$. For all $k \in F$, its k-th row $(P_F^{(0)})_k$ is defined as

$$(P_F^{(0)})_k := \begin{cases} (Q_F)_k & \text{if } k \in \widehat{F} \\ (\Pi_c^f \Theta_c)_k & \text{if } k \in \mathring{F}. \end{cases} \tag{3.6}$$

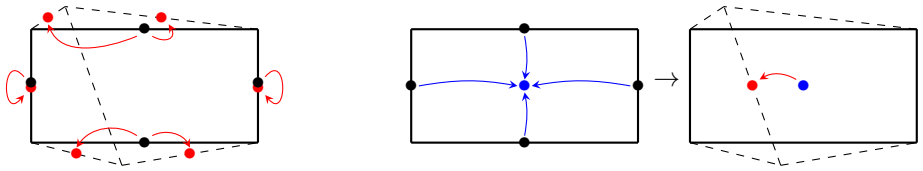
In this definition, $\Theta_c \in \mathbb{R}^{n_{T,c} \times n_{F,c}}$ locally computes the value on the coarse cells from their respective coarse faces, while $\Pi_c^f \in \mathbb{R}^{n_F \times n_{T,c}}$ transfers the value associated the coarse cells to their respective interior fine faces. We define

$$\Theta_c := -A_{TT,c}^{-1} A_{TF,c}, \tag{3.7}$$

which reverses the static condensation by solving for the cell unknowns local problems on the coarse cells given values on the faces. Next, for any face $k \in F$,

$$\forall n \in T_c, \qquad (\Pi_c^f)_{kn} := \begin{cases} 1 & \text{if } k \in \mathring{F}_n \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3.4 illustrates $P_F^{(0)}$.



(a) Q_F for $\widehat{F}$ (b) $\Pi_c^f \Theta_c$ for $\mathring{F}$

Fig. 3.4: Preliminary prolongation operator $P_F^{(0)}$. In these figures, we consider two fine elements (dashed lines) aggregated into one (solid lines). The DoFs on the coarse faces are represented by black dots, on the fine faces by red dots, on the coarse element by a blue dot.

Second, we remark that the stencil, in $P_F^{(0)}$, of the DoFs associated with removed fine faces (i.e. $k \in \mathring{F}$) is local to coarse elements. Given that the stencil in $\widetilde{A}$ is also local to coarse elements for those unknowns, one sweep of Jacobi smoothing can be applied to them without enlarging the prolongation stencil. This allows one to boost the convergence with virtually no additional computational cost. Note that a second smoothing iteration would enlarge the stencil outside of coarse elements, which we do not want. Setting the damping factor to $\omega := 2/3$, the smoothing matrix is defined by $J := I - \omega \widetilde{D}^{-1} \widetilde{A}$, where I is the identity matrix and $\widetilde{D}$ the diagonal part of $\widetilde{A}$. The final multigrid prolongation operator P is then defined row-wise for all rows $k \in F$ as

$$(P_F)_k := \begin{cases} (Q_F)_k & \text{if } k \in \widehat{F} \\ (JP_F^{(0)})_k & \text{if } k \in \mathring{F}. \end{cases} \tag{3.8}$$

To conclude about the formulation of P_F, we want to point out its mixed construction with respect to aggregation-based methods: plain aggregation is used for $\widehat{F}$, while $\mathring{F}$ benefits from smoothed prolongation.

3.7. Multigrid method. A hierarchy of L levels is built by multiple coarsening following subsection 3.5, and numbered from 1 (the coarsest) to L (the finest). At each level ℓ, the prolongation operator is given by (3.8), which we simply denote by P_ℓ instead of $P_{F,\ell}$. The other multigrid ingredients are chosen as per the variational framework: namely, the restriction is set to $P_\ell^\top$, and the coarse operator $\widetilde{A}_{\ell-1}$ to the Galerkin construction, initialized by the condensed matrix (2.2) as the finest operator $\widetilde{A}_L$ (i.e. $\widetilde{A}_{\ell-1} := P_\ell^\top \widetilde{A}_\ell P_\ell, \quad \forall \ell = 2, \dots, L$). The other parameters of the method (smoothers, cycle, coarsening factor, weak/strong coupling threshold, coarse grid solver) are left to the user's discretion; our choices are detailed in subsection 4.1.

Prolongation operators arising from plain aggregation are known to yield poor approximation properties of the coarse grid correction. However, it is known [24] that this loss of approximation, leading to bad convergence of the V-cycle, can be compensated by the use of the K-cycle ([27, Algorithm 3.2]), and by preconditioning a Krylov method. The variable number of Krylov iterations in the K-cycle makes it a variable preconditioner, which implies that a flexible version of the Krylov method has to be used. As the arising system is symmetric positive-definite, a flexible conjugate

gradient (FCG) is chosen. In particular, we use the so-called FCG(1) [26], also referred to as IPCG.

4. Numerical tests.

4.1. Experimental setup. Letting Ω be a bounded polytopal domain of $\mathbb{R}^d$, $d \in \{2, 3\}$, we consider the diffusion problem

$$\begin{cases} -\nabla \cdot (\mathbf{K}\nabla u) = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

where $f \in L^2(\Omega)$ is a given source term and $\mathbf{K}\colon \Omega \to \mathbb{R}^{d\times d}$ is the diffusion tensor field, which is assumed to be real, symmetric, uniformly elliptic. This problem is discretized by the HHO method [12] at the lowest order, which matches the structural requirements of Section 2. The homogeneous Dirichlet boundary condition is handled by elimination. The multigrid preconditioner performs one sweep of Gauss–Seidel in lexicographic order as pre-smoothing and one sweep of Gauss–Seidel in anti-lexicographic order as post-smoothing. We refer to this cycle as the K(1,1)-cycle. FCG(1) is used for the outer iteration as well as for the inner iteration of the K-cycle, meaning that the FCG, as outer solver, is preconditioned by the K(1,1)-cycle of our multigrid method. The preconditioner being symmetric positive definite, convergence of the outer FCG is ensured. To build each coarse level, multiple pairwise aggregations with the weak/strong coupling threshold $\beta = 0.25$ are performed (subsection 3.3), enforcing a coarsening factor ≥ 3.8. Coarse levels are built until the operator matrix has less than 1000 rows, where the system is solved by a direct solver. Iterations stop when the backward error, defined by the residual normalized by the right-hand-side, reaches a value lower than 10^{-8}. In the following results, note that the number of iterations refers to the outer solver, i.e. FCG.

4.2. Methodology. The main goal of the following numerical experiments is to establish a comparison between the solver developed in this work and the equivalent one made in the way of standard AMG. We will refer to them as Uncondensed AMG (U-AMG) and Condensed AMG (C-AMG), respectively. The former uses the uncondensed matrix to devise an *element-based* coarsening strategy, while the latter is directly working on the condensed system by implementing a *node-defined* coarsening strategy. The pairwise aggregation of C-AMG is performed according to the *nodewise* strong coupling relationship described in the introductory paragraphs of subsection 3.3, and we note that the multiple pairwise aggregation reduces in this case to the double pairwise aggregation. Its prolongation operator follows plain aggregation, i.e. is built similarly to the operator Q_T in (3.3a). The rest of the method shall be parametrized identically to U-AMG (same Krylov method, smoothers, cycle, etc.).

Comparing overall performances of two iterative methods is a difficult exercise. The convergence rate or number of iterations, alone, is not sufficient to establish a fair comparison, because the actual time to solution also depends on the iteration cost. Combining both criteria is usually made in terms of computational work or CPU time. The plain aggregation prolongation matrices, which contain only ones, are therefore applied to vectors without any theoretical flop, although their practical application still consumes non-negligible CPU time. As a consequence, we find the computational work not to be a good indicator in that case. As our U-AMG and C-AMG implementations both benefit from identical software components and optimizations, we adopt the CPU time (in sequential execution) as overall performance

criterion. Additionally, classical data used to assess convergence and cost of multigrid methods shall also be given. In particular, we introduce the operator complexity $\mathcal{C}_{op}$ and grid complexity $\mathcal{C}_{gd}$ defined by

$$\mathcal{C}_{op} := \sum_{\ell=1}^{L} \frac{\mathrm{nnz}(\widetilde{A}_\ell)}{\mathrm{nnz}(\widetilde{A}_L)}, \qquad \mathcal{C}_{gd} := \sum_{\ell=1}^{L} \frac{\mathrm{rows}(\widetilde{A}_\ell)}{\mathrm{rows}(\widetilde{A}_L)}.$$

These indicators give insight into the memory requirement and the computational cost of multigrid solvers.

Given the chosen parameters, namely FCG Krylov method, Gauss-Seidel smoothers, K(1,1)-cycle, etc., C-AMG corresponds almost exactly to the algorithm implemented by AGMG in the version of [27]. One minor difference is that our algorithm omits the special treatment of strongly diagonal-dominant rows, made to manage Dirichlet boundary conditions enforced by penalization. Furthermore, the current release of the software AGMG implements a quality control over the aggregates described in [25], which may significantly improve its overall performance, especially in anisotropic cases, where the "shape" of the coarse elements plays an essential role in the convergence rate. Such a quality control preventing the formation of "bad" aggregates is omitted in our C-AMG and U-AMG algorithms. Additionally, differences in the implementation prevents a fair comparison, in terms of execution time, with the fully optimized AGMG, for which better results can reasonably be expected. The term *implementation* here refers to any factor, besides the algorithm itself, that can influence the CPU time. Typically, it includes the software technologies employed (programming language, third-party libraries, compiling options, etc.) as well as the efficiency of the coding itself. For those reasons, results obtained with the current release of AGMG shall be included for information, more as a reference to a state-of-the-art solver than as direct comparative data.

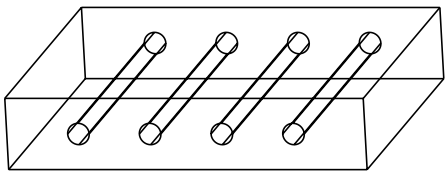
4.3. Numerical results.

4.3.1. Speed and robustness. Table 4.1 describes the test cases studied. Simple and complex geometries are used, discretized by Cartesian or unstructured simplicial meshes. Tests with anisotropic and heterogeneous tensors are performed. They all gather between 3 and 6 million face unknowns, ensuring at least 6 multigrid levels. Although all but the heterogeneous one are 3D problems, we point out that the results are consistent in 2D. The test results are displayed in Table 4.2. They include the following data: operator complexity ($\mathcal{C}_{op}$); grid complexity ($\mathcal{C}_{gd}$); number of multigrid levels (L); number of iterations to reach the convergence criterion (it); asymptotic convergence rate (ϱ), defined as the geometric mean of the residual convergence ratios for the last five iterations; solve CPU time in seconds, excluding setup (t). Figure 4.2 summarizes in a comparative chart the solve CPU times of the solvers. As explained in subsection 4.2, this figure shall concentrate most of the comments in this section.

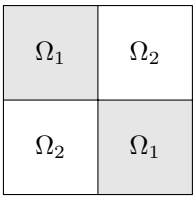
Let us first examine the dependency on the mesh. On a structured Cartesian mesh (`Cube-cart`), we remark that U-AMG is significantly faster than C-AMG (-25%). However, on the same geometry, this time with an unstructured tetrahedral mesh (`Cube-tet`), we get equivalent solve time. Finally, on a tetrahedral mesh describing a complex geometry (`Complex-tet`), the advantage of U-AMG fades out: U-AMG becomes slightly slower than C-AMG ($+6\%$). Next, on a heterogeneous problem with large coefficient jump (`Heterog1e8`), we see that both methods perform equivalently. Finally, tackling anisotropic problems, U-AMG is considerably faster than C-AMG on a Cartesian mesh (`Cube-cart-aniso100`), whereas they show comparable perfor-

Test case	Geometry	Mesh	Tensor	Elements	Unknowns
`Cube-cart`	Cube	Cartesian	Isotropic, homogeneous	2,097,152	6,242,304
`Cube-tet`	Cube	Unstruct. tetrahedral	Isotropic, homogeneous	1,224,179	2,418,910
`Complex-tet`	Figure 4.1a	Unstruct. tetrahedral	Isotropic, homogeneous	3,319,309	6,532,291
`Heterog1e8`	Square	Unstruct. triangular	Isotropic, heterogeneous according to Figure 4.1b	2,431,032	3,644,496
`Cube-cart-aniso100`	Cube	Cartesian	Anisotropic in the x direction, coefficient 100	2,097,152	6,242,304
`Cube-tet-aniso20`	Cube	Unstruct. tetrahedral	Anisotropic in the x direction, coefficient 20	1,224,179	2,418,910

Table 4.1: Description of the test cases.



(a) Geometry of test case `Complex-tet`: 3D plate with cylindrical holes.



(b) Heterogeneity pattern of test case `Heterog1e8`: for $i = 1, 2$, $\mathbf{K}_{|\Omega_i} := \kappa_i I$, with $\kappa_1/\kappa_2 = 10^8$.

Fig. 4.1: Supplementary figures for test cases `Complex-tet` (a) and `Heterog1e8` (b).

mance on an unstructured one (`Cube-tet-aniso20`). This set of tests demonstrates that U-AMG is favored by Cartesian meshes. To justify this result, we begin by recalling that the remaining unknowns of the condensed system are located on the faces. Indeed, viewed as nodes located at the center of the faces, these DoFs are *not* displayed, relative to each other, in a Cartesian way. See the node locations in Figure 4.3a: geometrically speaking, compared to the usual 2D Cartesian grid of element width h, the nodes form a set of rows evenly spaced by $h/2$, and where every other row has been shifted by $h/2$, giving the impression that the nodes are diagonally aligned. A fortiori, the Cartesian structure is partially lost in the sense that only one Cartesian direction is present in the stencil of each node (see solid red and dashed blue stencils in Figure 4.3a). The problem for C-AMG becomes visible on an anisotropic setting, where the anisotropy follows —for instance— the x-axis. Although one wants the aggregation process to produce horizontal aggregates, the shapes actually formed are

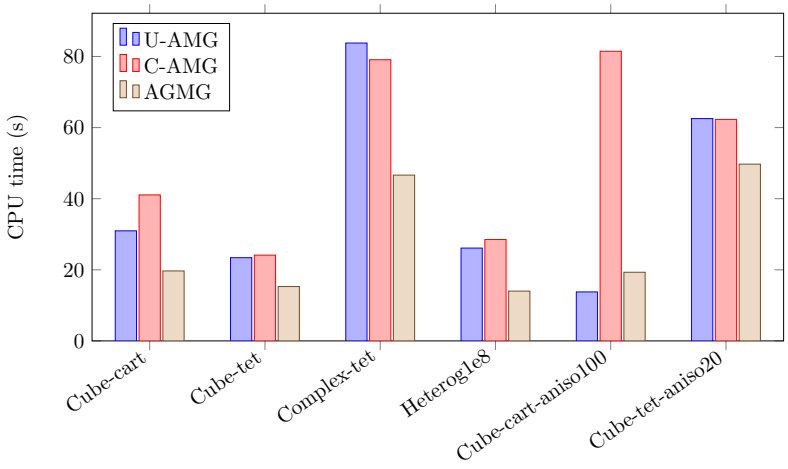


Fig. 4.2: Solver comparison in CPU time.

more diverse, and can even be vertical. Figure 4.3b illustrates the aggregates obtained by the double pairwise aggregation in this case: while desired horizontal aggregates are represented in solid red, one can also see vertical aggregates in dashed blue, as well as "waves" in dotted green. Referring to the solid red stencil of Figure 4.3a, we notice that nodes located on vertical grid lines have horizontal stencils, which allows them to be aggregated horizontally and form red aggregates. Similarly, nodes located on horizontal grid lines have inherently *vertical* stencils (in dashed blue). Specifically, their stencils do not contain any node to aggregate with in the horizontal direction in order to comply with the anisotropy. Nodes on the same grid line are indeed not part of the stencil. Consequently, due to the values of coefficients and the game of aggregation priorities, other shapes are formed instead: vertical aggregates in dashed blue or, better (because closer to horizontal), waves in dotted green. On the other hand, the reconstruction of the actual elements performed by U-AMG yields entities with fully Cartesian stencils, allowing the desired semi-coarsening; see Figure 4.3c. This explains why U-AMG performs so much better than C-AMG on the `Cube-cart-aniso100` test case. One can also add that in this case, the larger the anisotropy ratio, the better U-AMG performs compared to C-AMG. Whereas if the mesh is unstructured, they both perform equivalently whatever the anisotropy ratio. Note that this advantage is not limited to anisotropy directions that follow one of the axes; this profitable behaviour is also observed for orthotropic diffusion, namely, when the elements line up in the anisotropy direction. They can be rectangles in 2D and hexahedra in 3D, but also, more loosely, polytopes having two opposite faces orthogonal to the direction of anisotropy. However, if the mesh is fully unstructured, aggregating nodes probably offers more, or at least equivalent flexibility to follow the direction of anisotropy than aggregating elements. Hence the results obtained on the `Cube-tet-aniso20` test case, where U-AMG loses its superiority.

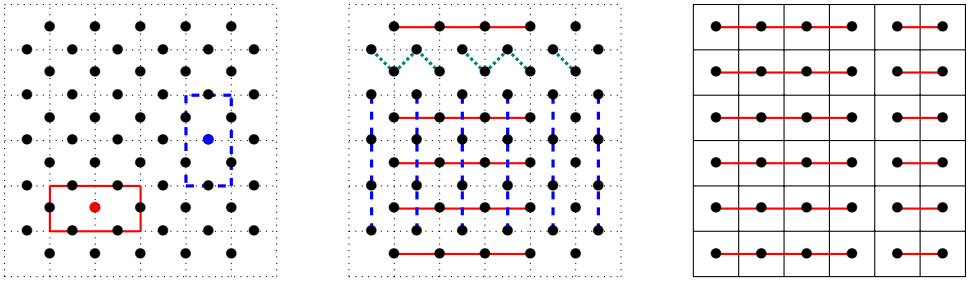
An interpretation of the results of AGMG can be attempted in light of these observations. As stated in subsection 4.2, AGMG implements a complex quality control preventing bad aggregates from being formed, which we have not carried out in C-AMG. In particular, we think that aggregates such as the blue ones in Figure 4.3b

`Cube-cart`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG	1.33	1.30	7	19	0.38	31.0
C-AMG	1.51	1.34	8	15	0.25	41.1
AGMG	2.03	1.64	9	24		19.7
`Cube-tet`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG	1.78	1.22	6	31	0.51	23.4
C-AMG	1.51	1.34	7	27	0.49	24.1
AGMG	1.98	1.79	7	28		15.3
`Complex-tet`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG	1.76	1.22	7	31	0.51	83.8
C-AMG	1.51	1.34	8	27	0.46	79.1
AGMG	1.96	1.78	7	27		46.6
`Heterog1e8`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG	1.55	1.27	7	27	0.42	26.1
C-AMG	1.42	1.34	7	23	0.38	28.5
AGMG	1.52	1.40	7	20		18.8
`Cube-cart-aniso100`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG	1.32	1.33	7	10	0.15	13.8
C-AMG	1.95	1.33	8	30	0.54	81.5
AGMG	1.70	1.51	7	23		19.3
`Cube-tet-aniso20`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG	1.82	1.23	6	77	0.80	62.5
C-AMG	1.60	1.43	8	75	0.78	62.3
AGMG	2.97	2.78	6	55		49.7

Table 4.2: Test results.

(namely, those orthogonal to the direction of anisotropy) do not occur in AGMG thanks to that quality control, thus explaining the large performance gap between C-AMG and AGMG on the `Cube-cart-aniso100` test case. We can also suppose that, when the problem is isotropic and the mesh unstructured, there are not many bad aggregates to prevent. In that case, we can then admit that the difference in CPU time between C-AMG and AGMG results from other aspects of the implementation. Looking at the results of the test cases `Cube-tet`, `Complex-tet` and `Heterog1e8`, we can attribute 35 to 50% of the CPU time consumed by C-AMG to an implementation overhead. As U-AMG benefits from the same implementation, this proportion gives a hint on how to compare U-AMG to AGMG. Specifically, we remark that even in spite of this overhead, U-AMG still performs better than AGMG on the test case `Cube-cart-aniso100`. This indicates that the new algorithm can lead to an improved efficiency for such cases.

4.3.2. Asymptotic behaviour. Figure 4.4 presents, for the test case `Cube-tet` and for each solver, the number of iterations required to achieve convergence according to the number of unknowns in the system. We remark that U-AMG scales the same



(a) Face DoFs and stencils. (b) Node aggregation. (c) Element aggregation.

Fig. 4.3: (a) Location of the face DoFs on a Cartesian grid. (b) and (c): result of the nodewise and elementwise double pairwise aggregations, according to an anisotropic problem following the x-axis.

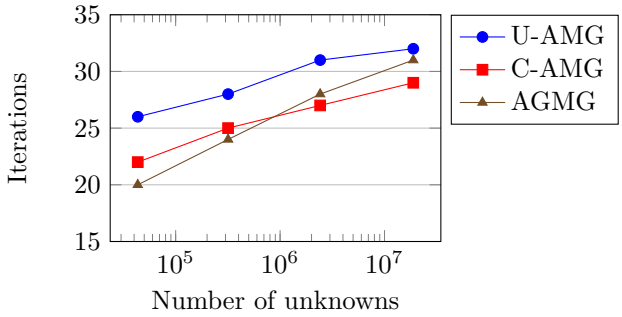


Fig. 4.4: Asymptotic behaviour with respect to the number of unknowns.

way as C-AMG, and slightly better than AGMG. This means that the new algorithm offers equivalent robustness to the meshsize as the existing method, and shares its algorithmic quasi-optimality.

4.3.3. Convergence/cost trade-off. We remark from Table 4.2 that the number of iterations required by U-AMG to reach convergence is generally higher than for C-AMG. We would like to discuss in this section the link between convergence rate and aggressiveness of coarsening.

The so-called multiple coarsening performed by U-AMG and C-AMG recursively coarsens until a desired coarsening factor (relative to the number of unknowns, i.e. the number of faces) is achieved. Note that for C-AMG, the number of required steps of coarsening is always 2, whereras for U-AMG, it needs to be higher to build the first levels, and decreases as the levels grow coarser. The fact that unknowns are face unknowns, again, explains this phenomenon. Indeed, one step of coarsening corresponds to aggregating elements pairwise and collapsing faces between aggregates. Consequently, the efficient reduction of unknowns heavily relies on opportunities to collapse faces. Now, starting from a simplicial mesh, i.e. polytopes with minimal number of faces, the possibilities of collapsing faces is limited, and so is the size of the subsequent face aggregates. The situation starts to improve as the levels grow

coarser because the elements then have a larger number of faces, which benefits the face collapsing process.

The downside of enforcing a coarsening factor, thus triggering multiple steps of coarsening, is that element aggregates can be large between two levels, which deteriorates the accuracy of the prolongation operator, and therefore that of the coarse grid correction. On the other hand, by fixing the number of coarsening steps performed between each level, we expect a better accuracy, but costlier iterations. Besides the larger number of levels built due to the less aggressive coarsening, we emphasize that between the highest levels, where the coarsening factor is low, the sparsity of the operator is barely improved, which implies similar smoothing costs at those levels. We compare the multigrid results of both strategies in Table 4.3. As expected, the fixed double coarsening strategy induces a better convergence rate than the multiple coarsening, with a number of iterations that is now lower than both C-AMG and AGMG. However, the operator and grid complexities have increased. While the grid complexity is still reasonable, in the sense that it is equivalent to that of AGMG, the operator complexity is significantly larger than with the adaptive multiple coarsening strategy, which reflects the high cost of smoothing and memory storage. All in all, the solver converges in more CPU time, hence our choice of the multiple coarsening method. Nonetheless, the double coarsening is yet not to be discarded. Finding ways to sparsen the coarse operators in order to optimize the trade-off between convergence rate and operator complexity is another research path.

`Cube-tet`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
U-AMG (multiple coarsening)	1.78	1.22	6	31	0.51	23.4
U-AMG (double coarsening)	2.88	1.72	8	25	0.46	25.5
C-AMG	1.51	1.34	7	27	0.49	24.1
AGMG	1.98	1.79	7	28		15.3

Table 4.3: Comparative solver results.

4.4. Alternative algorithms. In order to justify our algorithmic choices, we present supplementary numerical results using alternative prolongation operators. In particular, we want to compare the results of our method with those obtained using Q_F as a prolongation operator (cf. subsection 3.6). Indeed, since Q_F is used to build coarse levels in the setup phase, re-using it as the prolongation operator in the multigrid iterations comes as a more straightforward solution than constructing a new operator. Second, in order to evaluate the effect of the partial smoothing (cf. J in (3.8)), we also consider the multigrid method without this enhancement. Namely, it corresponds to using $P_F^{(0)}$ (cf. (3.6)) as a prolongation operator instead of P_F, and to introduce the operator Q_F^{smooth} as the counterpart of Q_F, enhanced with the same partial smoothing. Let us first consider the results obtained on the `Cube-tet` test case, given in the top half of Table 4.4. While plain Q_F provides a faster solver than $P_F^{(0)}$, the addition of the partial smoothing makes the final P_F and Q_F^{smooth} give equivalent results. In particular, the addition of one Jacobi sweep significantly improves the convergence rate of $P_F^{(0)}$, resulting in a non-negligible reduction of the CPU time, whereas no notable improvement is observed with Q_F. Although the results given by P_F and Q_F^{smooth} on isotropic test cases do not present much difference, the better

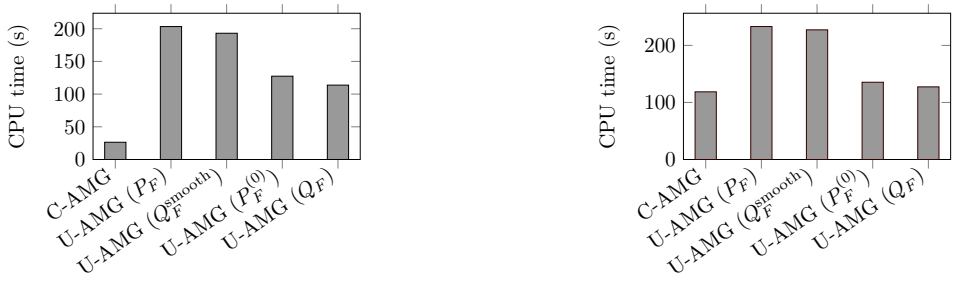
robustness of P_F manifests itself on the anisotropic test case `Cube-cart-aniso100`. Indeed, with or without additional smoothing, the method based on P_F gives significantly better results than that based on Q_F. This difference can be explained by the simplicity of Q_F. Clearly, assigning the mere average value of the local boundary faces to the DoFs on the local interior faces does not take the anisotropic coefficient into account. On the other hand, the decondensation of the cell unknowns performed by P_F through formula (3.7) successfully does so.

`Cube-tet`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
P_F	1.78	1.22	6	31	0.51	23.4
Q_F^{smooth}	1.79	1.22	6	30	0.51	23.4
$P_F^{(0)}$	1.80	1.22	6	32	0.56	28.0
Q_F	1.80	1.22	6	31	0.51	24.1
`Cube-cart-aniso100`	$\mathcal{C}_{op}$	$\mathcal{C}_{gd}$	L	it	ϱ	t
P_F	1.32	1.33	7	10	0.15	13.8
Q_F^{smooth}	1.32	1.32	7	15	0.36	22.1
$P_F^{(0)}$	1.32	1.32	7	19	0.47	28.8
Q_F	1.31	1.30	7	27	0.56	39.3

Table 4.4: Results with alternative prolongation operators.

4.5. Setup cost. Figure 4.5 compares the setup costs, measured in CPU time, of C-AMG and U-AMG. The alternative algorithms of the preceding section are also included. Firstly, in Figure 4.5a, one can observe the large overhead of U-AMG's setup relative to C-AMG's in the unstructured, isotropic case. As discussed in subsection 4.3.3, this is mainly due to the need for multiple coarsening steps to efficiently reduce the coarsening ratio in the number of unknowns, while only two are performed in C-AMG. Besides, the reconstruction of the algebraic mesh at each coarsening step is inherently more involved in U-AMG than in C-AMG. The partial smoothing operation included in the prolongation also has a significant setup cost (see U-AMG (P_F) vs. U-AMG ($P_F^{(0)}$), and U-AMG (Q_F^{smooth}) vs. U-AMG (Q_F)). In our implementation, the smoothing matrix J is explicitly assembled to construct the matrix P_F (cf. (3.8)), which is then applied as a matrix-vector product during the solving step. A more efficient implementation could be to dynamically smooth the data after applying the prolongation operator $P_F^{(0)}$, without explicitly computing J. Secondly, in Figure 4.5b, we observe that the gap between C-AMG and U-AMG has significantly reduced. The reason is two-fold: first, the Cartesian mesh benefits the coarsening process of U-AMG, which, in this case, is only applied twice between levels. In the meantime, the anisotropy slightly slows down that of C-AMG, which creates more singleton aggregates. In this configuration, and without the partial smoothing, U-AMG and C-AMG have comparable setup costs.

5. Conclusion. The solver developed in this work proposes an alternative AMG approach for the solution of linear systems arising from lowest-order hybrid discretizations. Although not entirely "black-box" (because it requires parts of the uncondensed system), it remains purely algebraic. Compared to the equivalent aggregation-based AMG constructed in the standard way (i.e. by viewing system unknowns as nodes),



(a) `Cube-tet` (b) `Cube-cart-aniso100`

Fig. 4.5: CPU time of the setup phase by algorithm.

it shows similar performance in most cases, while being more robust with respect to orthotropic anisotropy. Consequently, it can offer substantial added value for solving problems comprising both isotropic and anisotropic regions, like, e.g., Darcy flows. The solver, in this case, allies the flexibility of AMG to handle unstructured meshes on isotropic regions while exploiting the special element shapes on anisotropic ones. The cost of this improvement is mainly payed during the setup phase: (i) more memory storage may be required because of the use of the uncondensed matrix; but the blocks needed by the setup may be kept in storage anyway, because they are also needed to recover the cell unknowns after solving the condensed system. (ii) The more involved coarsening strategy and more complex integrid operators also imply a costlier setup.

Hybrid discretizations achieve their full potential in high order of approximation. Yet, this solver only applies to the lowest order. Even for more classical, non-hybrid discretizations, purely algebraic solvers for higher orders are still an open problem. In aggregation-based methods, the difficulty lies in the transfer of high order components from the coarse unknowns to the fine ones they aggregate. In this context, the elementwise view of the aggregation process is certainly easier to work with and geometrically interpret than a face aggregation view.

REFERENCES

[1] Arnold, D. N. and Brezzi, F., *Mixed and nonconforming finite element methods : implementation, postprocessing and error estimates*, ESAIM: M2AN, 19 (1985), pp. 7–32, https://doi.org/10.1051/m2an/1985190100071.

[2] L. Beirão da Veiga, K. Lipnikov, and G. Manzini, *The Mimetic Finite Difference Method for Elliptic Problems*, Springer International Publishing, Cham, 2014, https://doi.org/10.1007/978-3-319-02663-3.

[3] Bonelle, Jérôme and Ern, Alexandre, *Analysis of compatible discrete operator schemes for elliptic problems on polyhedral meshes*, ESAIM: M2AN, 48 (2014), pp. 553–581, https://doi.org/10.1051/m2an/2013104.

[4] D. Braess, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55 (1995), pp. 379–393, https://doi.org/10.1007/BF02238488.

[5] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, *Algebraic Multigrid Based on Element Interpolation (AMGe)*, SIAM Journal on Scientific Computing, 22 (2001), pp. 1570–1592, https://doi.org/10.1137/S1064827598344303.

[6] F. Brezzi, J. Douglas, and L. D. Marini, *Two families of mixed finite elements for second*

order elliptic problems, Numerische Mathematik, 47 (1985), pp. 217–235, https://doi.org/10.1007/BF01389710.

[7] V. E. Bulgakov, *Multi-level iterative technique and aggregation concept with semi-analytical preconditioning for solving boundary-value problems*, Communications in Numerical Methods in Engineering, 9 (1993), pp. 649–657, https://doi.org/10.1002/cnm.1640090804.

[8] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems.*, Elsevier, Burlington, 1978, http://www.123library.org/book_details/?id=41072. OCLC: 476223224.

[9] B. Cockburn, O. Dubois, J. Gopalakrishnan, and S. Tan, *Multigrid for an HDG method*, IMA Journal of Numerical Analysis, 34 (2014), pp. 1386–1425.

[10] B. Cockburn, J. Gopalakrishnan, and R. Lazarov, *Unified Hybridization of Discontinuous Galerkin, Mixed, and Continuous Galerkin Methods for Second Order Elliptic Problems*, SIAM Journal on Numerical Analysis, 47 (2009), pp. 1319–1365, https://doi.org/10.1137/070706616.

[11] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th national conference, ACM '69, New York, NY, USA, Aug. 1969, Association for Computing Machinery, pp. 157–172, https://doi.org/10.1145/800195.805928.

[12] D. A. Di Pietro and J. Droniou, *The Hybrid High-Order method for polytopal meshes*, no. 19 in Modeling, Simulation and Application, Springer International Publishing, 2020, https://doi.org/10.1007/978-3-030-37203-3.

[13] D. A. Di Pietro and A. Ern, *A hybrid high-order locking-free method for linear elasticity on general meshes*, Comput. Meth. Appl. Mech. Engrg., 283 (2015), pp. 1–21, https://doi.org/10.1016/j.cma.2014.09.009.

[14] D. A. Di Pietro, A. Ern, and S. Lemaire, *An arbitrary-order and compact-stencil discretization of diffusion on general meshes based on local reconstruction operators*, Comput. Meth. Appl. Math., 14 (2014), pp. 461–472, https://doi.org/10.1515/cmam-2014-0018. Open access (editor's choice).

[15] D. A. Di Pietro, F. Hülsemann, P. Matalon, P. Mycek, U. Rüde, and D. Ruiz, *An h-multigrid method for Hybrid High-Order discretizations*, SIAM Journal on Scientific Computing, (2021), https://doi.org/10.1137/20M1342471.

[16] 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 HHO discretizations and non-nested multigrid methods*, International Journal for Numerical Methods in Engineering, (2021), https://doi.org/10.1002/nme.6803.

[17] J. Droniou and R. Eymard, *A mixed finite volume scheme for anisotropic diffusion problems on any grid*, Numerische Mathematik, 105 (2006), pp. 35–71, https://doi.org/10.1007/s00211-006-0034-1.

[18] J. Droniou, R. Eymard, T. Gallouët, and R. Herbin, *A unified approach to mimetic finite difference, hybrid finite volume and mixed finite volume methods*, Mathematical Models and Methods in Applied Sciences, 20 (2010), pp. 265–295, https://doi.org/10.1142/S0218202510004222.

[19] R. Eymard, T. Gallouët, and R. Herbin, *Discretization of heterogeneous and anisotropic diffusion problems on general nonconforming meshes SUSHI: a scheme using stabilization and hybrid interfaces*, IMA Journal of Numerical Analysis, 30 (2010), pp. 1009–1043, https://doi.org/10.1093/imanum/drn084.

[20] R. D. Falgout, *An Introduction to Algebraic Multigrid*, Computing in Science & Engineering, 8 (2006), pp. 24–33, https://doi.org/10.1109/MCSE.2006.105.

[21] V. E. Henson and U. M. Yang, *Boomeramg: A parallel algebraic multigrid solver and preconditioner*, Appl. Numer. Math., 41 (2002), p. 155–177, https://doi.org/10.1016/S0168-9274(01)00115-5.

[22] M. Kronbichler and W. Wall, *A Performance Comparison of Continuous and Discontinuous Galerkin Methods with Fast Multigrid Solvers*, SIAM Journal on Scientific Computing, 40 (2018), pp. A3423–A3448, https://doi.org/10.1137/16M110455X.

[23] S. Muralikrishnan, T. Bui-Thanh, and J. N. Shadid, *A multilevel approach for trace system in HDG discretizations*, Journal of Computational Physics, 407 (2020), p. 109240, https://doi.org/10.1016/j.jcp.2020.109240.

[24] A. C. Muresan and Y. Notay, *Analysis of Aggregation-Based Multigrid*, SIAM Journal on Scientific Computing, 30 (2008), pp. 1082–1103, https://doi.org/10.1137/060678397.

[25] A. Napov and Y. Notay, *An Algebraic Multigrid Method with Guaranteed Convergence Rate*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1079–A1109, https://doi.org/10.1137/100818509.

[26] Y. Notay, *Flexible Conjugate Gradients*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1444–1460, https://doi.org/10.1137/S1064827599362314.

[27] Y. Notay, *An aggregation-based algebraic multigrid method*, Electronic Transactions on Numerical Analysis, 37 (2010), pp. 123–146.

[28] Y. Notay and A. Napov, *A massively parallel solver for discrete Poisson-like problems*, Journal of Computational Physics, 281 (2015), pp. 237–250, https://doi.org/10.1016/j.jcp.2014.10.043.

[29] L. N. Olson and J. B. Schroder, *Smoothed aggregation multigrid solvers for high-order discontinuous Galerkin methods for elliptic problems*, Journal of Computational Physics, 230 (2011), pp. 6959–6976.

[30] P. A. Raviart and J. M. Thomas, *A mixed finite element method for 2-nd order elliptic problems*, in Mathematical Aspects of Finite Element Methods, I. Galligani and E. Magenes, eds., Berlin, Heidelberg, 1977, Springer Berlin Heidelberg, pp. 292–315.

[31] J. Ruge and K. Stüben, *Efficient solution of finite difference and finite element equations by algebraic multigrid (AMG)*, GMD, 1984.

[32] J. W. Ruge and K. Stüben, *4. Algebraic Multigrid*, in Multigrid Methods, Frontiers in Applied Mathematics, Society for Industrial and Applied Mathematics, Jan. 1987, pp. 73–130, https://doi.org/10.1137/1.9781611971057.ch4.

[33] K. Stüben, *A review of algebraic multigrid*, in Numerical Analysis: Historical Developments in the 20th Century, C. Brezinski and L. Wuytack, eds., Elsevier, Amsterdam, Jan. 2001, pp. 331–359, https://doi.org/10.1016/B978-0-444-50617-7.50015-X.

[34] S. J. Thomas, S. Ananthan, S. Yellapantula, J. J. Hu, M. Lawson, and M. A. Sprague, *A Comparison of Classical and Aggregation-Based Algebraic Multigrid Preconditioners for High-Fidelity Simulation of Wind Turbine Incompressible Flows*, SIAM Journal on Scientific Computing, 41 (2019), pp. S196–S219, https://doi.org/10.1137/18M1179018. Publisher: Society for Industrial and Applied Mathematics.

[35] X. Tu and B. Wang, *A BDDC algorithm for second-order elliptic problems with hybridizable discontinuous Galerkin discretizations*, Electronic Transactions on Numerical Analysis, 45 (2016).

[36] P. Vaněk, *Acceleration of convergence of a two-level algorithm by smoothing transfer operators*, Applications of Mathematics, 37 (1992), pp. 265–274, https://doi.org/10.21136/AM.1992.104509.

[37] P. Vaněk, *Fast multigrid solver*, Applications of Mathematics, 40 (1995), pp. 1–20, https://doi.org/10.21136/AM.1995.134274.

[38] R. Wienands and C. W. Oosterlee, *On Three-Grid Fourier Analysis for Multigrid*, SIAM Journal on Scientific Computing, 23 (2001), pp. 651–671, https://doi.org/10.1137/S106482750037367X.

[39] T. Wildey, S. Muralikrishnan, and T. Bui-Thanh, *Unified Geometric Multigrid Algorithm for Hybridized High-Order Finite Element Methods*, SIAM Journal on Scientific Computing, 41 (2019), pp. S172–S195, https://doi.org/10.1137/18M1193505.

[40] O. C. Zienkiewicz, *Displacement and equilibrium models in the finite element method by B. Fraeijs de Veubeke, Chapter 9, Pages 145–197 of Stress Analysis, Edited by O. C. Zienkiewicz and G. S. Holister, Published by John Wiley & Sons, 1965*, International Journal for Numerical Methods in Engineering, 52 (2001), pp. 287–342, https://doi.org/10.1002/nme.339.