TECHNISCHE
UNIVERSITÄT
WIEN

Diplomarbeit

AscAMG, Algebraic Multigrid with Alternative Strong Connections

Ausgeführt am Institut für

Analysis und Scientific Computing

der Technischen Universität Wien

unter der Anleitung von

Univ.Prof. Dipl.-Ing. Dr.techn. Joachim Schöberl

durch

Lukas Kogler BSc

Schallergasse 39, 1120 Wien

December 13, 2017

Kurzfassung

Diese Diplomarbeit stellt den AscAMG Vorkonditionierer vor, einen verteilten Algebraischen Mehrgitter-Vorkonditionierer für skalare, elliptische H1-Probleme, der zur Einbindung in Netgen/NGSolve entwickelt wurde. Die Methode ist benannt nach der alternativen Art und Weise wie starke Verbindungen charakterisiert werden. Die dabei zum Einsatz kommende Ersatzmatrix führt direkt zu einer neuen Variation der Methode der geglätteten Prolongation die in Aggregations-basierten Mehrgitterverfahren zum Einsatz kommt. Auf die Parallelisierung der Methode wird ganz besonders eingegangen und auch skalierbare parallele Glätter werden besprochen. Nachdem die Skalierbarkeit der Methode auf mindestens 1800 Prozessoren durch numerische Ergebnisse demonstriert wird, werden Schlüsse gezogen und eine Perspektive auf mögliche künftige Weiterentwicklungen der Methode gegeben.

Abstract

This thesis introduces the AscAMG preconditioner, a distributed Algebraic Multigrid Preconditioner for scalar, elliptic H1-problems, that has been developed for NGSolve. The method gets its' name from an alternative way to define strong connections that is based on a replacement matrix. This leads directly to a new variation of the smoothed prolongation method commonly found in aggregation based Multigrid solvers. The parallelization of the method is described in detail and scalable smoothers are found and discussed. After demonstrating the scalability of the method to at least 1800 cores with numerical results, conclusions are drawn and an outlook on possible future developments of the method is given.

Acknowledgements

First and foremost I want to thank Prof. Dr. Joachim Schöberl for supervising this thesis. Without his advice and patience this work would not have been possible.

Special thanks also go to Professor Jay Gopalakrishnan who granted me access to the COEUS cluster at Portland State University which was a huge help in developing, optimizing and benchmarking my code.

Another person who must be mentioned is my colleague, Matthias Hochsteger, who was always happy to help me with his expertise in C++ and anything computer-related in general. I hope he will never again have to hear me say that the SCALAPACK library was not linked correctly again.

And lastly, I want to sincerely thank my family for supporting me during the course of my studies.

Notation

The standard euclidian product in $\mathbb{R}^n$ will be denoted by $\langle\cdot,\cdot\rangle_2$ or, if the context is clear, by $\langle\cdot,\cdot\rangle$. The euclidian norm will be written as $\|\cdot\|_2$ or $\|\cdot\|$. We write $a \lesssim b$ for $a \leq Cb$ with some moderately sized constant C and $a \approx b$ for $a \lesssim b \wedge b \lesssim a$. We will conistently write vectors in $\mathbb{R}^n$ and matrices in $\mathbb{R}^{n\times n}$ bold. For matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$, we will write

$$\langle \mathbf{u}, \mathbf{v}\rangle_A := \langle \mathbf{A}\mathbf{u}, \mathbf{v}\rangle \tag{0.1}$$

$$\|\mathbf{u}\|_A^2 := \langle \mathbf{u}, \mathbf{u}\rangle_A \tag{0.2}$$

(0.1) is called the energy inner product. For SPD $\mathbf{A}$, the norm (0.2) is called the energy norm. For matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{n\times n}$, we write $\mathbf{A} \geq 0$ if $\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle \geq 0 \ \forall \mathbf{x} \in \mathbb{R}^n$ and $\mathbf{A} \geq \mathbf{B}$ if $\mathbf{A} - \mathbf{B} \geq 0$.

Contents

1 Introduction

Algebraic Multigrid Methods are a class of solvers and preconditioners for systems of linear equations that have been used in a wide variety of problems, among others in heat conduction, fuild dynamics and electromagnetics. While they are generally inferior to geometric multigrid methods where those can be applied, algebraic multigrid methods offer relatively "black-box" solvers that are robust against large jumps in coefficients and complex geometrical situations, neither of which is true for geometric multigrid algorithms. This work will introduce "AscAMG ", which stands for "Algebraic Multigrid with Alternative Strong Connections", an algebraic multigrid preconditioner for scalar, symmetric positive definite H^1 - problems. More precicely, we will consider the equation

$$-\mathrm{div}\,(\alpha\nabla u) + \beta u = f \tag{1.1}$$

with badly behaving coefficients α and β in 2 and 3 dimensions on arbitrarily complex domains Ω.

AscAMG is an extension to Netgen/NGSolve. Netgen/NGSolve is a multi purpuse C++ mesh-generation and Finite Element library that has been in development since the ninties (see also [5], [6]). AscAMG is itself implemented in C++ and has been shown to scale to 1800 cores.

Outline of the thesis

In chapters 2 and 3, we will very shortly review the Finite Element method and some numerical methods we can use to solve linear equations resulting from the Finite Element discretization of the problem (1.1) in question.

After that, we will discuss the challenges distributed computing environments pose for Finite Elements and very briefly review the Message Passing Interface (MPI), a standard for message passing in distributed systems. The distributed Finite Element Method as well as a particular viewpoint of distributed linear algebra that fits it very well will be presented. Closing out chapter 4,we will give some perspectives on how NGSolve handles MPI-parallelization.

After that, in chapter 5, motivation for multigrid methods will be provided, and in chapter 6 we will take a first look at multigrid methods in general, at their advantages and at the difference between geometric and algebraic methods.

In section 7, the algebric multigrid method will be introduced in a generic way and a well known result about a condition that implies its' convergence will be presented.

The AscAMG method itself will be introduced in detail in chapter 8. A proof for the methods' convergence based on the previous' chapter's abstract definition will be given. In particular, the parallelization of the method will be described in detail. In course of that, we will intordure a formalism that divides the degrees of freedom into equiva-

lence classes based on their parallelicity. This will be used to formulate our coarsening algorithm as well as the smoothers available in AscAMG with great ease. It will also provide a very simple characterization of a class of prolongation matrices that guarantee communication-free transfer between grid-levels.
Finally, the quality and scalability of the method will be demonstrated in section 9.

2 The Finite Element Method

In this chapter we will briefly review the most important facts about the finite element method for discretizing (1.1), as far as we will need it later on. As we will only be interested in the lowest order case and equation (1.1) is one of the most basic partial differntial equations there are, proofs for all lemmas and theorems in this chapter can be found in any good textbook on the Finite Element Method, for example [3]. Section 2.1 will give an overview over element matrices which will be used in chapter 4 to gain a better perspective on distributed linear algebra for the distributed Finite Element method.

Let us first enforce boundary conditions in (1.1). Let Γ_D and Γ_N be subsets of $\partial\Omega$ such that $\Gamma_D \cup \Gamma_N = \partial\Omega$. For $\alpha, \beta, f \in L^2(\Omega)$ and suitable $u_D, g \in L^2(\Gamma_N)$, enforcing boundary conditions in equation (1.1) leads to:

Find $u \in C^2(\overline{\Omega})$ such that:

$$-\operatorname{div}(\alpha \nabla u) + \beta u = f \quad \text{in } \Omega \tag{2.1}$$

$$\mathbf{tr}_{\Gamma_D} u = u_D \quad \text{on } \Gamma_D \tag{2.2}$$

$$\nabla u \cdot n = g \quad \text{on } \Gamma_N \tag{2.3}$$

Notation 2.1. For the remainder of this work, let $\Omega \subseteq \mathbb{R}^d$, where $d = 2$ or $d = 3$ be a Lipschitz domain. We will also limit ourselfs to the case where Ω is a polygon or polyhedron.

Next, we will work towards the weak formulation of (2.1).

Lemma 2.1: H^1-Sobolev Space

The space

$$H^1(\Omega) := \{u \in L^2(\Omega) : \nabla u \in L^2(\Omega)\}$$

equipped with the norm

$$\|u\|^2_{H^1(\Omega)} := \|u\|^2_{L^2(\Omega)} + \|\nabla u\|^2_{L^2(\Omega)}$$

is a hilbert space. Let $\Gamma \subseteq \partial\Omega$, with $|\Gamma| > 0$. Then there is a continuous trace operator $\boldsymbol{tr}_\Gamma : H^1(\Omega) \to L^2(\Gamma)$ that extends the restriction operator for continuous functions:

$$\forall u \in C(\overline{\Omega}) : \boldsymbol{tr}_\Gamma u = u_{|\Gamma} \quad \textit{on } \Gamma$$

The space

$$H^1_D(\Omega) := \boldsymbol{ker}\ \boldsymbol{tr}_\Gamma = \{u \in H^1(\Omega) : \boldsymbol{tr}_\Gamma\ u = 0\}$$

equipped with the H^1-norm is also a hilbert space. If $\Gamma = \partial\Omega$, we write $\boldsymbol{tr}$ instead of $\boldsymbol{tr}_\Gamma$ and $H^1_0(\Omega)$ instead of $H^1_D(\Omega)$.

The weak formulation of (2.1) reads:

Find $u \in H^1(\Omega)$ **such that:**

$$a(u,v) = f(v) \quad \forall v \in H^1_D(\Omega) \tag{2.4}$$

$$u = u_D \quad \text{on } \Gamma_D \tag{2.5}$$

With:

$$a(u,v) := \int_\Omega \alpha \nabla u \cdot \nabla v + \beta uv \; dx \tag{2.6}$$

$$f(v) := \int_\Omega fv \; dx + \int_{\Gamma_N} gv \; ds \tag{2.7}$$

Technically, we require $u_D \in H^{1/2}(\Gamma_D)$ and $g \in H^{-1/2}(\Gamma_D)$, with the fractional order Sobolev-Slobodeckij spaces $H^{1/2}(\Gamma_D)$ and $H^{-1/2}(\Gamma_D)$, however, we will simply assume that these functions are as smooth as we need them to be. The well known Lax-Milgram lemma guarantees the exinstence of a sultion for (2.4) and (2.5) in $H^1(\Omega)$, provided $\alpha \geq \alpha_0 > 0$, $\beta \geq 0$ and either $\beta \geq \beta_0 > 0$ or $|\Gamma_D| > 0$. We will assume this to be the case.

Lemma 2.2: Lax-Milgram

Let V be a Hilbert space, and let $a(\cdot,\cdot)$ be a continuous, elliptic (or coercive) bilinear form on V, that is

$$a(u,v) \leq C\|u\|_V\|v\|_V \quad \forall u,v \in V$$

$$a(u,u) \geq c\|u\|_V^2 \quad \forall u \in V$$

Let $f(\cdot)$ be a continuous linear form on V,

$$f(u) \leq D\|u\|_V \quad \forall u \in V$$

Then, the equation

$$a(u,v) = f(v) \quad \forall v \in V$$

has a unique solution $u \in V$. Furthermore,

$$\|u\|_V \leq \frac{D}{c}$$

For the Finite Element Method we replace the infinite-dimensional continuous spaces $H^1(\Omega)$ and $H^1_D(\Omega)$ by finite dimensional, discrete spaces V_h and $V_{D,h}$ and arrive at the discrete weak formulation:

Find $u_h \in V_h$ **such that:**

$$a(u_h,v_h) = f(v_h) \quad \forall v_h \in V_{0,h} \tag{2.8}$$

$$u_h = P_h u_D \quad \text{on } \Gamma_D \tag{2.9}$$

Here, P_h is the L^2-Projector onto $\mathbf{tr}_{\Gamma_D}(V_h)$, for simplicity we will from now on assume that $u_D \in \mathbf{tr}_{\Gamma_D}(V_h)$. The discrete spaces V_h, $V_{h,D}$ will be defined shortly.

Definition 2.1: Regular Triangulation

A regular triangulation $\mathcal{T}_h$ of Ω is a set of non degenerate, closed traingles or tetraheda such that

$$\overline{\Omega} = \bigcup_{T \in \mathcal{T}} T$$

For any $T_i, T_k \in \mathcal{T}_h$, $T_i \cap T_j$ must be either empty, a vertex or an edge. The mesh width h is the maximal diameter of any $T \in \mathcal{T}$. We assume that

$$\operatorname{diam}(T) \geq Ch \quad \forall T \in \mathcal{T}_h$$

with a moderately sized constant C. This property will also be called the shape regularity of $\mathcal{T}_h$.

Notation 2.2. We will write $\mathcal{V}(\mathcal{T}_h)$ for the set of all vertices and $\mathcal{E}(\mathcal{T}_h)$ for the set of all edges of a triangulation $\mathcal{T}_h$. The set of all its faces will be $\mathcal{F}(\mathcal{T}_h)$. A "node" can refer to either a vertex, an edge, a face or an element of $\mathcal{T}$. All k-dimensional nodes in a triangulation will be assigned a number in $\{1 \ldots n_k\}$, in no particular order, v_i will stand for the vertex with number i, e_i for the edge with number i etc. The edge connecting two vertices v_i and v_j will also be written as e_{v_i,v_j}, or, more briefly, e_{ij}. The two vertices of an edge e will also be written as v_{e_1} and v_{e_2}, in no particular order. We will also use the notation $\mathcal{T}_v := \{T \in \mathcal{T}_h : v \in \mathcal{V}_T\} \subseteq \mathcal{T}_h$.

In order to properly resolve the boundary conditions, we require Γ_N and Γ_D both to consist of a union of elements in $\mathcal{E}(\mathcal{T}_h)$ in two dimensions or $\mathcal{F}(\mathcal{T}_h)$ in three dimensions. We will assume this as given. We are now ready the define what a finite element is.

Definition 2.2: Finite Element

Let $\mathcal{T}_h$ be a regular triangulation of Ω. A finite element is a triple $(T, V_{h,T}, \Psi_T)$, where $T \in \mathcal{T}_h$, $V_{h,T}$ is a finite dimensional function space on T and Ψ_T is a base of its dual space $V_{h,T}^{'}$. The finite element's base functions are the dual basis vectors of Ψ_T. For each $\psi_i \in \Psi_T$ we write $\varphi_i \in \Phi_T$ for its dual basis function.

The global finite element space, which will take the place of V_h in (2.8), is just the product of these local spaces defined on each element in $\mathcal{T}_h$, with some additional enfored restrictions. We will only be concerned with the classical nodal, scalar, continuous, lowest order H^1 Finite Element space.

Definition 2.3: Lowest Order H^1 Finite Element Space

For a regular triangulation $\mathcal{T}_h$, and $T \in \mathcal{T}_h$ let $V_{h,T} = P^1(T)$. Let $\mathcal{V}_T$ be the set of vertices of T and for each $v \in \mathcal{V}_T$ let $\psi_v \in C(T)'$ be defined by $\psi_v(u) = u(v)$. Finally, let $\Psi_T := \{\psi_v : v \in \mathcal{V}_T\}$. The lowest order H^1 finite element is defined as $(T, P^1(T), \Psi_T)$. The lowest order H^1 Finite Element Space is defined as

$$V_h := \prod_{T \in \mathcal{T}} V_{h,T} \cap C^0(\Omega) \subseteq H^1(\Omega) \tag{2.10}$$

We also need the discrete space that corresponds to $H^1_D(\Omega)$, which is just the set of all V_h functions that vanish on Γ_D:

$$V_{h,D} := \{u_h \in V_h : u_h(v_i) = 0 \ \forall v_i \in \mathcal{V} \cap \Gamma_D\}$$

Again, we write $V_{h,0}$ instead of $V_{h,D}$ if $\Gamma_D = \partial\Omega$.

Note 2.1. In 2 dimensions, this gives us the standard "hat"-function basis.

Notation 2.3. We will write $\| \cdot \|_A$ for the A-norm, the norm on V_h induced by $a(\cdot, \cdot)$. We will write $u_h \perp_A v_h$ if $a(u_h, v_h) = 0$.

Lemma 2.3: Céa

Let V be a hilbert space and let V_h be a finite dimensional subspace of V. Let $a(\cdot, \cdot)$ be an elliptic and continuous bilinear form on V and let $f(\cdot)$ be a continuous linear form on V. Then, there exist unqiue solutions u and u_h of the problems

$$\begin{aligned} a(u, v) &= f(v) \quad \forall v \in V \\ a(u_h, v_h) &= f(v_h) \quad \forall v \in V_h \end{aligned}$$

With the ellipticity constant c and the continuity constant C of $a(\cdot, \cdot)$, they fulfill

$$\|u - u_h\| \leq \frac{C}{c} \inf_{v_h \in V_h} \|u - v_h\|$$

From now on, V_h will always be the lowest order H^1 Finite Element space defined on $\mathcal{T}_h$.

Lemma 2.4: Approximation Properties of V_h

On a regular triangulation, the lowest order nodal scalar H^1-finite element space fulfills

$$\begin{aligned} \inf_{v_h \in V_h} \|u - v_h\|_{H^1(\Omega)} &\lesssim h|u|_{H^2(\Omega)} \quad \forall u \in H^2(\Omega) \\ \inf_{v_h \in V_h} \|u - v_h\|_{L^2(\Omega)} &\lesssim h^2|u|_{H^2(\Omega)} \quad \forall u \in H^2(\Omega) \end{aligned}$$

We will now turn our attention towards the system of linear equation induced by (2.8). For that, we identify V_h with the coordinate space $\mathbb{R}^n$:

Lemma 2.5: Galerkin Isomorphism

The Galerkin isomorphism $G : \mathbb{R}^n \mapsto V_h$, *where* $n = \dim V_h = |\mathcal{V}|$

$$G\boldsymbol{u} = \sum_{i=1}^{n} \boldsymbol{u}_i \varphi_i =: u_h \tag{2.11}$$

fulfills the bounds

$$h^d \|\boldsymbol{u}\|_2^2 \lesssim \|u_h\|_{L^2}^2 \lesssim h^d \|\boldsymbol{u}\|_2^2$$

Notation 2.4. In prose, we will usually not distinguish between a finite element function u_h and its coordinate vector representation $G^{-1}u_h$ at all, however, where it is necessary to differentiate between the two, we will often omit G and write pairs with respect to G as u_h and $\mathbf{u}$, v_h and $\mathbf{v}$ etc.

Lemma 2.6: Finite Element Matrix

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ *and* $\mathbf{b} \in \mathbb{R}^n$, *with* $n = \dim V_h = |\mathcal{V}|$ *be defined by*

$$\mathbf{A}_{ij} = a(\varphi_i, \varphi_j)$$

$$\mathbf{b}_i = f(\varphi_i)$$

Then, the action of $a(\cdot, \cdot)$ *can be brougt from* V_h *to* $\mathbb{R}^n$ *with the galerkin isomorphism (2.11):*

$$\mathbf{v}^T \mathbf{A} \mathbf{u} = a(u_h, v_h)$$

Solving (2.8) is then, via the Galerkin isomorphism, equivalent to finding $\mathbf{u} \in \mathbb{R}^n$ *such that*

$$A\mathbf{u} = \mathbf{b} \tag{2.12}$$

Notation 2.5. We will write $\|\cdot\|_{\mathbf{A}}$ for the $\mathbf{A}$-norm, the norm on $\mathbb{R}^n$ induced by $\mathbf{A}$. We will write $\mathbf{u} \perp_{\mathbf{A}} \mathbf{v}$ if $\langle \mathbf{A}\mathbf{u}, \mathbf{v} \rangle = 0$.

Definition 2.4: Stiffness and Mass matrices

From now on, $\boldsymbol{M}$ *stands for the mass matrix, which is the finite element matrix for (2.8) and* $\alpha = 0, \beta \neq 0$, *or*

$$a(u, v) = \int_\Omega \beta u v dx$$

If $\beta = 1$, $\|\mathbf{u}\|_{\boldsymbol{M}} = \|u_h\|_{L^2(\Omega)}$.
The stiffness matrix $\boldsymbol{K}$ *will be the finite element matrix for (2.8) and* $\alpha \neq 0, \beta = 0$

$$a(u, v) = \int_\Omega \alpha \nabla u \cdot \nabla v dx$$

If $\alpha = 1$, $\|\mathbf{u}\|_K = |u_h|_{H^1(\Omega)} =: \|\nabla u_h\|_{L^2(\Omega)}$

2.1 Element Matrices

We will now take a look at element matrices, how to assemble the global matrix from these and how to perform the matrix vector multiplication with A while only using the element matrices. While this will not be a point of focus for the rest of this work, the perspective gained in this section will be useful for the later discussion of the parallelization of the Finite Element method in section 4.3.

Definition 2.5: Reference Element

The reference element $\hat{T} \subseteq \mathbb{R}^d$ is defined as the convec hull $\textbf{\textit{conv}}\{0, e_x, e_y\}$ in two and $\textbf{\textit{conv}}\{0, e_1, e_x, e_z\}$ in three dimensions.

Lemma 2.7: Element Transformation

For each $T \in \mathcal{T}_h$, there is a bijective linear element transformation $F_T : \hat{T} \to T$ such that:

$$|\det \nabla F_T| \approx h^d$$
$$|\nabla F_T| \approx h$$

Definition 2.6: Element Matrix

Let $T \in \mathcal{T}_h$ be an element with $\mathcal{V}_T = \{i_1, i_2, i_3\}$. Let $(T, V_T.\Psi_T)$ be the Finite Element corresponding to T and let $(\varphi_i)_{1 \le i \le d}$ be the local Finite Element basis functions as in definition 2.3, numbered such that $\varphi_j(v_{i_k}) = \delta_{jk}$. Note that the φ_i are restrictions of global basis functions to T.
The element matrix $A_T \in \mathbb{R}^{d \times d}$ associated with T and $a(\cdot, \cdot)$ is defined by

$$A_{T,jk} := a_{|T}(\varphi_j, \varphi_k) := \int_T \alpha \nabla \varphi_j \cdot \nabla \varphi_k + \beta \varphi_j \varphi_k \; dx \quad 1 \le j, k \le d \tag{2.13}$$

The element vector $b_T \in \mathbb{R}^d$ associated with T and $f(\cdot)$ is defined by

$$b_j := f_{|T}(\varphi_j) := \int_T f \varphi_j \; dx + \int_{\Gamma_N \cap \partial T} g \varphi_j \; ds \tag{2.14}$$

The element index-map is

$$\boldsymbol{m}_T : \begin{cases} \{1 \dots 3\} & \to \{0 \dots n-1\} \\ k & \mapsto i_k \end{cases}$$

The discrete embedding matrix $E_T \in \mathbb{R}^{n \times d}$ associated with T is defined by the element index map via

$$E_{T,ij} = \delta_{i, \boldsymbol{m}_T(j)}$$

Lemma 2.8: Assembling the Matrix

The global matrix A can be assembled from the element matrices and the embeddings by:

$$A = \sum_{T \in \mathcal{T}_h} E_T A_T E_T^T$$

Using $\mathcal{T}_v$ from Notation 2.2, this means that the components of $\mathbf{A}$ can be written as:

$$A_{ij} = \sum_{T \in \mathcal{T}_{v_i} \cap \mathcal{T}_{v_j}} A_{T, \boldsymbol{m}_T^{-1}(i) \boldsymbol{m}_T^{-1}(j)}$$

Note 2.2. We can express multiplication with $\mathbf{A}$ via the element matrices. For a vector $\mathbf{v} \in \mathbb{R}^n$ and an element $T \in \mathcal{T}_h$, $\mathbf{v}_T := \mathbf{E}_T \mathbf{v} \in \mathbb{R}^d$ is its component corresponding to the element's degrees of freedom. $G\mathbf{E}_T^T \mathbf{v} \in V_{h,T}$ is the restriction of v_h to T.

$$\mathbf{A}\mathbf{v} = \sum_{T \in \mathcal{T}_h} E_T A_T E_T^T \mathbf{v} = \sum_{T \in \mathcal{T}_h} E_T A_T \mathbf{v}_T$$

This means that $\mathbf{A}\mathbf{v}$ is the sum of local contributions $A_T \mathbf{v}_T$.

2.2 Numerical Aspects

The matrix graph of A is strongly related to the triangulation $\mathcal{T}_h$, $a_{ij} \neq 0$ if and only if there exists an edge connecting v_i and v_j, or in other words if $e_{ij} \in \mathcal{E}(\mathcal{T}_h)$. This means that, assuming shape regularity of $\mathcal{T}_h$, the number of non zero entries of A per row is limited by approximately 7 in two dimensions and 14 in three dimensions.

Lemma 2.9: Conditioning of the mass matrix

If $\beta = 1$, the mass matrix $\mathbf{M}$ fulfills the (sharp) spectral bounds

$$h^d \|\mathbf{u}\|_2^2 \lesssim \|\mathbf{u}\|_M^2 \lesssim h^d \|\mathbf{u}\|_2^2$$

This leads to the condition number

$$\kappa(\boldsymbol{M}) = \mathcal{O}(1)$$

Note 2.3. If $\beta \neq 1$ but $\max\{\beta(x),\ x \in \Omega\} < C$ and $\min\{\beta(x),\ x \in \Omega\} > c$ for positive constants, we at worst incur an additional factor C in the upper bound and c in the lower bound, leading to an additional worst case factor $\frac{C}{c}$ in the condition number.

Lemma 2.10: Conditioning of the stiffness matrix

If $\alpha = 1$, and $|\Gamma_D| > 0$, the stiffness matrix $\mathbf{K}$ fulfills the (sharp) spectral bounds

$$h^d \|\mathbf{u}\|_2^2 \leq \|\mathbf{u}\|_K^2 \leq h^{d-2} \|\mathbf{u}\|_2^2$$

This leads to the condition number

$$\kappa(\boldsymbol{K}) = \mathcal{O}(h^{-2})$$

Note 2.4. If $\alpha \neq 1$ but $\max\{\alpha(x),\ x \in \Omega\} < C$ and $\min\{\alpha(x),\ x \in \Omega\} > c$ for positive constants, we at worst incur an additional factor C in the upper bound and c in the lower bound, leading to an additional worst case factor $\frac{C}{c}$ in the condition number.

Note 2.5. From this we can see that condition-wise, the worst case in (2.8) is the pure poisson problem, where, $\beta = 0$, and α varies strongly. Adding any (positive) L^2-term will only improve the codition of the Finite Element matrix A.

3 Basic Iterative Methods

This chapter contains a short overview over some very basic iterative methods that will be referred to throughout the rest of this work. While the Conjugate Gradient Algorithm will only be mentioned shortly for the sake of completeness, stationary iterative methods will be elaborated upon in more detail, as these are the basic building blocks for all multigrid algorithms.

For now, we will take a step back and consider the equation $\mathbf{Ax} = \mathbf{b}$ for generic (sparse) SPD matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. When n is very large, solving such systems of equations in general requires iterative methods, as direct solution methods require too much memory and time to be feasible. Unfortunately, most iterative techniques also behave badly as $n \to \infty$ because their performance is dependent on $\kappa(\mathbf{A})$. In the case where $\mathbf{A}$ is a finite element matrix for the poisson problem, lemma 2.10 tells us that $\kappa(\mathbf{A}) \to \infty$ as $h \to 0$. One way out of the dilemma is to try and find a good Preconditioner $\mathbf{C}$ for $\mathbf{A}$, that is a matrix such that $\mathbf{C}^{-1}$ is a good approximation for $\mathbf{A}^{-1}$ that is also cheap to compute. Then the condition of $\mathbf{C}^{-1}\mathbf{A}$ is much better that that of $\mathbf{A}$ and one can apply the iterative method to a transformed system.

Definition 3.1: Preconditioner

A preconditioner for an SPD matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an SPD matrix $\mathbf{C} \in \mathbb{R}^{n \times n}$ such that

$$\kappa(\mathbf{C}^{-1}\mathbf{A}) < \kappa(\mathbf{A})$$

The **C**onjugate **G**radient Method (**CG**-Method) is probably the most famous method for solving generic SPD problems $\mathbf{Ax} = \mathbf{b}$. It is a member of the family of Krylov Space methods and features strict monotone convergence in the energy norm, however the rate of convergence is dependent on the condition of the matrix $\mathbf{A}$. The CG method is featured in many textbooks, for example, it is treated in great detail in [4].

Theorem 3.1: The Preonditioned Conjugate Gradient Method

The preconditioned Conjugate Gradient Method (algorithm 1) for solving the equation

$$\mathbf{Ax} = \mathbf{b}$$

for an SPD matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$, given an initial approximation $\mathbf{u}_0$ of $\mathbf{u}$ with help of an SPD preconditioner $\mathbf{C} \in \mathbb{R}^{n \times n}$ finds the exact solution after at most n steps. With the intermediate apprximate solution after k steps and the $\mathbf{e}_k \coloneqq \mathbf{u} - \mathbf{u}_k$, it fulfills the energy norm estimate

$$\|\mathbf{e}_k\|_{\mathbf{A}} \leq \left(\frac{1 - \sqrt{\kappa(\mathbf{C}^{-1}\mathbf{A})}}{1 + \sqrt{\kappa(\mathbf{C}^{-1}\mathbf{A})}} \right)^k \|\mathbf{e}_0\|_{\mathbf{A}}$$

Proof. See, for example [4] □

Note 3.1. PCG combines a very generic algorithm, the conjugate gradient method, that works for all SPD matrices $\mathbf{A}$ with a preconditioner $\mathbf{C}$, which is usually specifically tailored to the problem at hand.

Algorithm 1 The Preconditioned Conjugate Greadient Method

```
procedure PCG($\mathbf{A}$, $\mathbf{b}$, $\mathbf{x}_0$)
    Compute $\mathbf{r}_0 \coloneqq \mathbf{b} - \mathbf{A}\mathbf{x}_0$, $\mathbf{z}_0 = \mathbf{C}^{-1}\mathbf{r}_0$, $\mathbf{p}_0 \coloneqq \mathbf{z}_0$
    for $j = 0, 1 \ldots$ until convergence do
        $\alpha_j \coloneqq \langle \mathbf{r}_j, \mathbf{z}_j \rangle \,/\, \langle \mathbf{A}\mathbf{p_j}, \mathbf{p_j} \rangle$
        $\mathbf{x}_{j+1} \coloneqq \mathbf{x}_j + \alpha_j \mathbf{p}_j$
        $\mathbf{r}_{j+1} \coloneqq \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j$
        $\mathbf{z}_{j+1} \coloneqq \mathbf{M}^{-1}\mathbf{r}_{j+1}$
        $\beta_j \coloneqq \langle \mathbf{r}_{j+1}, \mathbf{z}_{j+1} \rangle \,/\, \langle \mathbf{r}_j, \mathbf{r}_j \rangle$
        $\mathbf{p}_{j+1} \coloneqq \mathbf{z}_{j+1} + \beta_j \mathbf{p}_j$
     return $x_j$
```

3.1 Stationary Linear Iterative Methods

In essence, **S**tationary **L**inear **I**terative **M**ethods (SLIMs) are affine linear mappings $\Phi : \mathbb{R}^n \to \mathbb{R}^n$. One starts with some initial guess $\mathbf{x}_0$ for the solution $\mathbf{x}^*$ and then iterates $\mathbf{x}_{k+1} = \Phi(\mathbf{x}_k)$. For consistency, the true solution $\mathbf{x}^*$ must be a fixed point, $\Phi(\mathbf{x}^*) = \mathbf{x}^*$.

Definition 3.2: Stationary Linear Iterative Methods

A stationary linear iterative method for the solution of $\mathbf{A}\mathbf{x} = \mathbf{b}$ *can be written as*

$$\mathbf{x}_{k+1} = \mathbf{M}\mathbf{x}_k + \mathbf{N}\mathbf{b} \tag{3.1}$$

with $\mathbf{M}$, $\mathbf{N} \in \mathbb{R}^{n \times n}$, $\mathbf{N}$ *invertible and* $\mathbf{I} = \mathbf{M} + \mathbf{N}\mathbf{A}$.

Note 3.2. Alternatively, one could also write (3.1) as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{N}\left(\mathbf{b} - \mathbf{A}\mathbf{x}_k\right) \tag{3.2}$$

This is just one step of the preconditioned Richardson-iteration. We see that $\mathbf{N}$ should be an approximation for $\mathbf{A}^{-1}$.

Note 3.3. Given an SPD preconditioner $\mathbf{C}$ for $\mathbf{A}$ such that $\mathbf{A} \leq C\mathbf{C}$ and $\mathbf{A} \geq c\mathbf{C}$, $\mathbf{N} \coloneqq \frac{1}{C}\mathbf{C}^{-1}$ defines a convergent SLIM for which $\|\mathbf{M}\| \leq \frac{C}{c} = 1 - \kappa(\mathbf{C}^{-1}\mathbf{A})^{-1}$.

Note 3.4. $\mathbf{M}$ is also called the error propagation matrix or simply the iteration matrix of the SLIM, because, with the error $\mathbf{e}_k \coloneqq \mathbf{x}^* - \mathbf{x}_k$, the error after the next iteration is

$$\mathbf{e}_{k+1} = \mathbf{M}\mathbf{e}_k$$

Lemma 3.1: Convergence Criterium for SLIMs

Let $\mathbf{W} \coloneqq \mathbf{N}^{-1}$, *then a sufficient condition for the convergence of SLIM is*

$$\mathbf{W} + \mathbf{W}^T - \mathbf{A} > 0 \tag{3.3}$$

Proof. By definition of the operator norm (w.r.t the euclidian norm), we have

$$\|\mathbf{e}_{k+1}\|_2 = \|\mathbf{M}\mathbf{e}_k\|_2 = \|(\mathbf{I}-\mathbf{N}\mathbf{A})\mathbf{e}_k\|_2 \leq \|\mathbf{I}-\mathbf{N}\mathbf{A}\|_2\|\mathbf{e}_k\|_2 \leq \|\mathbf{I}-\mathbf{N}\mathbf{A}\|_2^{k+1}\|\mathbf{e}_0\|_2$$

We have to show that $\|\mathbf{I}-\mathbf{N}\mathbf{A}\|_2 = \|\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}}\|_2 < 1$.

$$\begin{aligned}(\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}})^T(\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}}) &= \mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}^T\mathbf{A}^{\frac{1}{2}}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}}+\mathbf{A}^{\frac{1}{2}}\mathbf{N}^T\mathbf{A}\mathbf{N}\mathbf{A}^{\frac{1}{2}} = \\ &= \mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}^T\mathbf{N}^{-1}\mathbf{N}\mathbf{A}^{\frac{1}{2}}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}^T\mathbf{N}^{-T}\mathbf{N}\mathbf{A}^{\frac{1}{2}}+\mathbf{A}^{\frac{1}{2}}\mathbf{N}^T\mathbf{A}\mathbf{N}\mathbf{A}^{\frac{1}{2}} = \\ &= \mathbf{I}-(\mathbf{N}\mathbf{A}^{\frac{1}{2}})^T(\mathbf{W}+\mathbf{W}^T-\mathbf{A})(\mathbf{N}\mathbf{A}^{\frac{1}{2}}) = \\ &=: \mathbf{I}-\mathbf{B}\end{aligned}$$

Now, if $\mathbf{W}+\mathbf{W}^T-\mathbf{A} > 0$, $\mathbf{B}$ is SPD, $\mathbf{I}-\mathbf{B} < 1$ and

$$\begin{aligned}\|\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}}\mathbf{x}\|^2 &= \left\langle(\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}})\mathbf{x}, (\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}})\mathbf{x}\right\rangle = \\ &= \left\langle(\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}^T\mathbf{A}^{\frac{1}{2}})(\mathbf{I}-\mathbf{A}^{\frac{1}{2}}\mathbf{N}\mathbf{A}^{\frac{1}{2}})\mathbf{x}, \mathbf{x}\right\rangle \\ &= \langle(\mathbf{I}-\mathbf{B})\mathbf{x}, \mathbf{x}\rangle < \|\mathbf{x}\|_2^2\end{aligned}$$

And thus $\|\mathbf{I}-\mathbf{N}\mathbf{A}\|_2 < 1$. □

Notation 3.1. The matrices **M**, **N** and **W** will from now, unless explicitely stated otherwise, stand for the iteration matrix, the approximate inverse and it's inverse associated with a SLIM.

Definition 3.3: Dampened Jacobi Method

The dampened jacobi method with dampening parameter $\omega \in \mathbb{R}$ is given by definition 3.2 and

$$\mathbf{N} = \omega\mathbf{D}^{-1}$$

with the diagonal **D** *of* **A**.

Lemma 3.2: Spectral Bounds for Jacobi Preconditioning

The jacobi preconditioner for the stiffness matrix **K** *fulfills the (sharp) spectral bounds*

$$h^2\mathbf{D} \lesssim \mathbf{K} \lesssim \mathbf{D}$$

This leads to the condition number estimate $\kappa(\mathbf{D}^{-1}\mathbf{K}) = \mathcal{O}(h^{-2})$ with a constant that depends on α.
The jacobi preconditioner for the mass matrix **M** *fulfills the optimal spectral bounds*

$$c\mathbf{D} \lesssim \mathbf{M} \lesssim C\mathbf{D}$$

with condition number $\kappa(\mathbf{D}^{-1}\mathbf{M}) = \mathcal{O}(1)$ with a constant that depends on β.

Proof. If $0 < \alpha_0 < \alpha < \alpha_1$

$$\alpha_0 \min_{v\in\mathcal{V}(\mathcal{T}_h)} \|\varphi_v\|_{H^1(\Omega)}^2\|\mathbf{u}\|_2^2 \leq \langle\mathbf{D}\mathbf{u}, \mathbf{u}\rangle \leq \alpha_1 \max_{v\in\mathcal{V}(\mathcal{T}_h)} \|\varphi_v\|_{H^1(\Omega)}\|\mathbf{u}\|_2^2$$

Therefore, assuming shape regularity of the triangulation, $\mathbf{D} \approx h^{d-2}\mathbf{I}$. and if $0 < \beta_0 < \beta < \alpha$

$$\beta_0 \min_{v\in\mathcal{V}(\mathcal{T}_h)} \|\varphi_v\|_{L^2(\Omega)}^2\|\mathbf{u}\|_2^2 \leq \langle\mathbf{D}\mathbf{u}, \mathbf{u}\rangle \leq \beta_1 \max_{v\in\mathcal{V}(\mathcal{T}_h)} \|\varphi_v\|_{L^2(\Omega)}\|\mathbf{u}\|_2^2$$

Therefore, assuming shape regularity of the triangulation, $\mathbf{D} \approx h^d \mathbf{I}$. Now the claim is evident from lemmas 2.9 and 2.10. □

Note 3.5. According to lemma 3.1, the dampened jacobi method is convergent if $\frac{2}{\omega}\mathbf{D} > \mathbf{A}$.

Definition 3.4: Gauss-Seidel Method

The Gauss-Seidel method is given by definition 3.2 and

$$\mathbf{N} = (\mathbf{L} + \mathbf{D})^{-1}$$

where $\mathbf{L}$ *is the strictly lower triangular part of* $\mathbf{A}$*.*

Note 3.6. In contrast to the Jacobi method, the Gauss-Seidel method does not require a dampening parameter, however using one can improve the convergence rate altough finding a good parameter is not trivial.

Note 3.7. For any SLIM given by $\mathbf{N}$, $\mathbf{N}^T$ defines another, "transposed", SLIM, that converges iff. the original one does. The error propagation matrix of the transposed SLIM is $\mathbf{I} - \mathbf{N}^T\mathbf{A} = \mathbf{M}^*$, which is the $\langle \cdot, \cdot \rangle_A$-conjugate of $\mathbf{M}$. For example, the GS method becomes the backwards GS method with $\mathbf{N} = (\mathbf{U} + \mathbf{D})^{-1}$.

Definition 3.5: Symmetrized SLIM

Let a SLIM be given by N, then the symmetrized SLIM is given by

$$\widetilde{\mathbf{N}} \coloneqq \mathbf{N}^T(\mathbf{N}^{-T} + \mathbf{N}^{-1} - \mathbf{A})\mathbf{N}$$

or, equivalently, by

$$\widetilde{\mathbf{W}} \coloneqq \mathbf{W}(\mathbf{W}^T + \mathbf{W} - \mathbf{A})^{-1}\mathbf{W}^T$$

Note 3.8. One iteration of the symmetrized SLIM $\widetilde{\mathbf{N}}$ is equivalent to one iteration woth $\mathbf{N}$ followed by one iteration with $\mathbf{N}^T$. As $\widetilde{\mathbf{M}} = \mathbf{M}^*\mathbf{M}$ is symmetric wrt. $\langle \cdot, \cdot \rangle_{\mathbf{A}}$, $\rho(\widetilde{\mathbf{M}}) = \|\widetilde{\mathbf{M}}\|_A = \|\mathbf{M}\|_{\mathbf{A}}^2$.

Note 3.9. If we symmetrize GS with dampening parameter ω, we obtain the well known SSOR method (successive overrelaxation) with

$$\mathbf{N} = \omega(2 - \omega)\left(\mathbf{L} + \mathbf{D}\right)^{-1} \mathbf{D} \left(\mathbf{L} + \mathbf{U}\right)^{-1}$$

From lemma 3.2 we know that the contraction rate of the jacobi method is $1 - \mathcal{O}(h^2)$. We will now see that Gauss-Seidel has the same asymptotic behavior.

Lemma 3.3

If a SLIM fulfills, for some $\sigma_0, \sigma_1 > 0$

$$\left\langle \left(\mathbf{W} + \mathbf{W}^T - \mathbf{A}\right)\mathbf{x}, \mathbf{x}\right\rangle > \sigma_0 \left\langle \mathbf{D}\mathbf{x}, \mathbf{x}\right\rangle \tag{3.4}$$

$$\left\langle \mathbf{W}^T\mathbf{D}^{-1}\mathbf{W}\mathbf{x}, \mathbf{x}\right\rangle \le \sigma_1 \left\langle \mathbf{D}\mathbf{x}, \mathbf{x}\right\rangle \tag{3.5}$$

Then

$$\frac{\sigma_0}{4} \left\langle \mathbf{D}\mathbf{x}, \mathbf{x}\right\rangle \le \left\langle \widetilde{\mathbf{W}}\mathbf{x}, \mathbf{x}\right\rangle \le \frac{\sigma_1}{\sigma_0} \left\langle \mathbf{D}\mathbf{x}, \mathbf{x}\right\rangle \tag{3.6}$$

Proof. Let $\mathbf{B} \coloneqq \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$, now from (3.4) we get:

$$\begin{aligned}\|B\|^2 &= \sup_{\mathbf{x}} \frac{\left\langle \mathbf{D}^{-\frac{1}{2}}\mathbf{W}^T\mathbf{D}^{-1}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}\mathbf{x},\mathbf{x}\right\rangle}{\|\mathbf{x}\|^2} = \\ &= \sup_{\mathbf{x}} \frac{\left\langle \mathbf{W}^T\mathbf{D}^{-1}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}\mathbf{x},\mathbf{D}^{-\frac{1}{2}}\mathbf{x}\right\rangle}{\|\mathbf{x}\|^2} \leq \\ &\leq \sigma_1 \sup_{\mathbf{x}} \frac{\left\langle \mathbf{D}\mathbf{D}^{-\frac{1}{2}}\mathbf{x},\mathbf{D}^{-\frac{1}{2}}\mathbf{x}\right\rangle}{\|\mathbf{x}\|^2} = \sigma_1\end{aligned}$$

Now, using (3.5) we can show the upper bound for $\widetilde{\mathbf{W}}$ in (3.6):

$$\begin{aligned}\left\langle \widetilde{\mathbf{W}}\mathbf{x},\mathbf{x}\right\rangle &= \left\langle \left(\mathbf{W}+\mathbf{W}^T-\mathbf{A}\right)\mathbf{W}^T\mathbf{x},\mathbf{W}^T\mathbf{x}\right\rangle \leq \frac{1}{\sigma_0}\left\langle \mathbf{D}^{-1}\mathbf{W}^T\mathbf{x},\mathbf{W}^T\mathbf{x}\right\rangle = \\ &= \left\langle \mathbf{B}\mathbf{B}\mathbf{D}^{\frac{1}{2}}\mathbf{x},\mathbf{D}^{\frac{1}{2}}\mathbf{x}\right\rangle \leq \frac{\|B\|^2}{\sigma_0}\left\langle \mathbf{D}\mathbf{x},\mathbf{x}\right\rangle = \frac{\sigma_1}{\sigma_0}\left\langle \mathbf{D}\mathbf{x},\mathbf{x}\right\rangle\end{aligned}$$

For the other bound, (3.4) gives us:

$$2\left\langle \mathbf{W}\mathbf{x},\mathbf{x}\right\rangle = \left\langle \left(\mathbf{W}+\mathbf{W}^T\right)\mathbf{x},\mathbf{x}\right\rangle \geq \left\langle \left(\mathbf{W}+\mathbf{W}^T-\mathbf{A}\right)\mathbf{x},\mathbf{x}\right\rangle \geq \sigma_0\left\langle \mathbf{D}\mathbf{x},\mathbf{x}\right\rangle$$

Equivalently,

$$\left\langle \mathbf{B}\mathbf{x},\mathbf{x}\right\rangle \geq \frac{\sigma_0}{2}\left\langle \mathbf{x},\mathbf{x}\right\rangle$$

Now, we apply this inequality to $\mathbf{B}^{-1}\mathbf{x}$:

$$\|\mathbf{B}^{-1}\mathbf{x}\|^2 \leq \frac{2}{\sigma_0}\left\langle \mathbf{x},B^{-T}\mathbf{x}\right\rangle \leq \frac{2}{\sigma_0}\|\mathbf{B}^{-1}\mathbf{x}\|\|\mathbf{x}\| \quad \Rightarrow \quad \|B^{-1}\| \leq \frac{2}{\sigma_0}$$

From this we obtain

$$\begin{aligned}\left\langle \widetilde{\mathbf{W}}^{-1}\mathbf{x},\mathbf{x}\right\rangle &= \left\langle \mathbf{W}^T\left(\mathbf{W}+\mathbf{W}^{-T}-A\right)\mathbf{W}^{-1},\mathbf{x}\right\rangle = \\ &= \left\langle \left(\mathbf{W}^{-T}+\mathbf{W}^{-1}-\mathbf{W}^{-T}\mathbf{A}\mathbf{W}^{-1}\right)\mathbf{x},\mathbf{x}\right\rangle \leq \\ &\leq \left\langle \left(\mathbf{W}^{-T}+\mathbf{W}^{-1}\right)\mathbf{x},\mathbf{x}\right\rangle = 2\left\langle \mathbf{W}^{-1}\mathbf{x},\mathbf{x}\right\rangle = \\ &= 2\left\langle \mathbf{B}^{-1}\mathbf{D}^{-\frac{1}{2}}\mathbf{x},\mathbf{D}^{-\frac{1}{2}}\mathbf{x}\right\rangle \leq 2\|B^{-1}\|\left\langle \mathbf{D}^{-1}\mathbf{x},\mathbf{x}\right\rangle \leq \\ &\leq \frac{4}{\sigma_0}\left\langle \mathbf{D}^{-1}\mathbf{x},\mathbf{x}\right\rangle\end{aligned}$$

Or, equivalently

$$\frac{\sigma_0}{4}\left\langle \mathbf{D}\mathbf{x},\mathbf{x}\right\rangle \leq \left\langle \widetilde{\mathbf{W}}\mathbf{x},\mathbf{x}\right\rangle$$

□

Theorem 3.2: Asymptotic Behavior of Gauss-Seidel

The Gauss-Seidel method fulfills the conditions (3.4), (3.5) from theorem 3.3 with $\sigma_0 = 1$ and some moderately sized σ_1.

Proof. $\sigma_0 = 1$ is trivial because $\left(\mathbf{W}+\mathbf{W}^T-\mathbf{A}\right) = \mathbf{D}$. For the other condition we have to show

$$\left\langle \mathbf{W}^T\mathbf{D}^{-1}\mathbf{W}\mathbf{x},\mathbf{x}\right\rangle \leq \sigma_1\left\langle \mathbf{D}\mathbf{x},\mathbf{x}\right\rangle$$

This is equivalent to showing

$$\rho\left(\mathbf{D}^{-1}\mathbf{W}^T\mathbf{D}^{-1}\mathbf{W}\right) \leq \sigma_1$$

A stronger condition than this is

$$\|\mathbf{D}^{-1}\mathbf{W}\|\|\mathbf{D}^{-1}\mathbf{W}^T\| \leq \sigma$$

for some arbitraty matrix-norm $\|\cdot\|$. We choose $\|M\| \coloneqq \max_i \left\{\sum_j |m_{ij}|\right\}$. Assuming regularity of the trangulation, we can now bound $\|\mathbf{D}^{-1}\mathbf{W}\|$ from above by a moderately sized constant. Let m be the maximum number of entries in any row of $\mathbf{A}$. Then

$$\begin{aligned}\|\mathbf{D}^{-1}\mathbf{W}\| &= \max_i \left\{\frac{1}{a_{ii}}\sum_{j\leq i}|a_{ij}|\right\} \leq \max_i \left\{\frac{1}{a_{ii}}\sum_{j}|a_{ij}|\right\} \leq \\ &\leq m \max_{i,j}\left\{\frac{|a_{ij}|}{|a_{ii}|}\right\} \leq m \max_j \{a_{jj}\}\end{aligned}$$

□

Note 3.10. In the proof of theorem 3.2 we obtained $\sigma_1 = \mathcal{O}(m^2)$, which holds for all SPD matrices A but is very pessimistic in the case where A is the poisson matrix. In fact, because constant functions are in the kernel of the poisson-matrix (for a domain without dirichlet boundaries), we usually have

$$\frac{1}{a_{ii}}\sum_j |a_{ij}| = \mathcal{O}(1)$$

with the constant depending on the space dimension and the shape regularity of the triangulation.

Corrolary 3.1.1. *Forwards/Backwards/Symmetric Gauss-Seidel behave no better than Jacobi for $h \to 0$.*

Proof. Theorem 3.2 says that $\widetilde{\mathbf{W}} \approx \mathbf{D}$. This means that

$$h^2 \lesssim \frac{\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle}{\left\langle \widetilde{\mathbf{W}}\mathbf{x}, \mathbf{x}\right\rangle} = \frac{\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle}{\langle \mathbf{D}\mathbf{x}, \mathbf{x}\rangle}\frac{\langle \mathbf{D}\mathbf{x}, \mathbf{x}\rangle}{\left\langle \widetilde{\mathbf{W}}\mathbf{x}, \mathbf{x}\right\rangle}$$

where the lesser bound is sharp because of lemma 3.2! Thus the convergence rate of symmetric Gauss-Seidel is $1 - \mathcal{O}(h^2)$, just as that of Jacobi. With

$$\|\widetilde{\mathbf{M}}\| = \|\mathbf{M}^T\mathbf{M}\| \leq \|\mathbf{M}\|^2 \quad \Rightarrow \|\mathbf{M}\| \geq \sqrt{\|\widetilde{\mathbf{M}}\|} \geq \|\widetilde{\mathbf{M}}\|$$

it is clear that forwards/backwards Gauss-Seidel can do no better than symmetric GS.

□

4 Distributed Computing

In this chapter we will illustrate the most fundamental differences between smaller systems, ranging from personal computers to single machine servers, and fully distributed clusters and how a simplified programming model has to change to accomodate that. After that, we will show how, in NGSolve, this shift in perspectives is extended to the Finite Element method in a very natural way and how the linear algebra is implemented to match.

While systems of the first kind can, on the most basic level, be viewed as single, integrated machines consisting of a number of processing units that may act independently of one another but share all of the available memory, clusters typically consist of many such machines, in that context also called nodes, that are basically independent and only connected to the others by some kind of communication network.
Broadly speaking, accessing memory on the same node may involve latencies ranging from a few processor cycles, typically about 1 ns when accessing cached memory, to about 100 ns when accessing memory in DRAM. Making use of the communication system to get information from another node involves higher latencies that depend on the networks' exact makeup, the node-to-node distance and possible congestion, however we are typically on a µs scale.
Additionaly, the bandwidth of local memory access is in general much larger than the bandwidth of the communication network.
This discrepancy in both latency and throughput between accessing local memory and making use of the communication network to access memory on other nodes has to be reflected in our programming model if we want to write even remotely useful code for such systems.

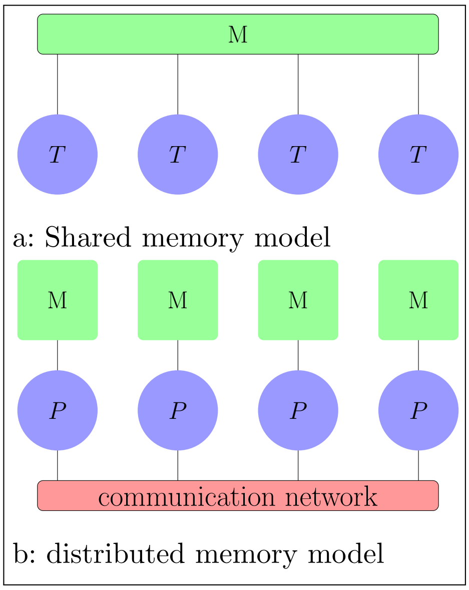
4.1 The Distributed Programming Model

The central distinction between a simple shared memory programming model and a simple distributed one is that the first is what one might call "task based" and the second "proc based". In this context a process, or in short proc, is a unit for scheduling and memory management, while a proc is unit for scheduling only.
A simple shared memory model, schematically depicted in figure 4.1a, assumes that there is a number of threads within a single process that can act independently of one another but can all access any part of the available memory at roughly the same speed. The rationale is that different threads can be assigned to different processors and can then work in parallel. We, as programmers, to some degree, depending on the framework used for threading, have to take care of synchronization between threads, but on the most basic level we do not need to care about which data is stored in which part of

the memory. Even when considering that in reality we do in fact have to consider non-uniform memory access if we want to write really efficient code, we can usually get away with parallelizing the sequential code on a relatively low level without changing many of the algorithms very much on a conceptual level.

In the context of Finite Elements, this may for example involve parallelizing for-loops over elements or nodes using a coloring of the underlying mesh.

On clusters on the other hand, we have to use more of a “top-down” approach to parallelization. The higher latency and lower throughput of the communication network as opposed to local memory access on each node forces us away from this kind of thread based approach towards one that is concerned with procs. Schematically, this new approach is illustrated in figure 4.1b. Each proc has its own private memory which can only be accessed by itself and is completely opaque to all other procs, the memory is now *distributed.* The only way for the procs to cooperate is by sending messages via the communication network.



a: Shared memory model

b: distributed memory model

Now we, as programmers, have to know exactly which data is stored in which part of the memory and we have to make any needed piece of data stored outside of any procs’ own memory available to it by explicitly making use of the network. In contrast to shared memory parallelization, it is now in most cases insufficient to parallelize sequential code on a low level, instead we now also have to adapt our algorithms on a high level in order to minimize use of the communication network. In scientific computing, the de facto standard way of interacting with the communication network is via an MPI-library.

4.2 MPI

The **M**essage-**P**assing-**I**nterface is a standard that specifies a number of functions that facilitate the passing of messages between different processes. This standard is implemented in a multitude of good libraries, for example OpenMPI, IntelMPI and MPICH. While MPI provides an interface that is platform independent, the libraries themselfs are very well optimized and can be tuned for any specific machine, which allows users to write efficient and portable code relatively easily.

We will now go over some of the terminology MPI introduces, as far as we will need it in the remainder of this work.

Notation 4.1. The number of processes at work will be written as n_p.

In MPI, a *communicator* is a context in which a group of processes can exchenge messages. This group can include all of the n_p procs or only a suset thereof. MPI provides a global communicator containing all procs called MPI_COMM_WORLD, and gives the user the ability to create new communicators containing any subset of all procs. Within each communicator, any participating proc is assigned a *rank*, a number in $\{0 \dots n_c - 1\}$, with n_c being the number of procs in the communicator, which is used as an identifier. This means that each proc can have different ranks in different communicators, however we will simply refer to any proc by its rank in MPI_COMM_WORLD and explicitly mention if a rank refers to some other sub-communicator.

Notation 4.2. We will consistently write $\mathcal{P}$ for the set of ranks $\{0 \dots n_p - 1\}$.

The most basic communication facility MPI provides is that of the matched Send/Recv operations. With a send operation, a proc can send a certain block of data to another one in a communicator. The target proc is specific by it's rank in the communicator in which the message is sent. This operation has to be matched by a corresponding Recv operation on the target proc. The receiver must know the origin, type of content and size of any message it wishes to receive. Data is passed to MPI-functions via a C-style pointer, the Send-operation provides a pointer to the memory where the data to be sent is stored and the Recv-operation provides a pointer to a buffer the received data is to be written into. There are multiple versions of these operations available, the ones that are relevant for us are the standard MPI_Send(..)/MPI_Recv(..) operations as well as their nonblocking counterparts MPI_Isend(..)/MPI_Irecv(..). MPI_Send(..) returns only when it is safe to modify the data it has been passed, and MPI_Recv(..) returns when the message has arrived and been written into the recv-buffer. MPI_Send(..) may only return when the matching receive-operation has been called and the message has actually been sent, it may, however, if the data is in the background copied to a seperate system buffer, return sooner than that. The nonblocking versions return immediately, without any guarantee about the state of the data in the send- and recv-buffers. The completion of a nonblocking send or receive can be waited for by calling MPI_Wait(..). MPI_Wait(..) returns exactly when MPI_Send(..) or MPI_Recv(..) would have, that is when it is safe to modify the send-buffer and the message has arrived in the recv-buffer respectively. Both send-variants come in different modes, for example buffered mode where outgoing messages are copied to a user provided buffer which lets the send-call return immediately after that or synchronous send that does not return until the matching receive has actually been posted. The advantage of nonblocking operations are, among others, that it is easier to avoid deadlocks, that multiple communications can happen at the same time and that it allows for overlapping of communication and computation if there is special hardware present that can work on communication in the background.
This only scratches the surface of MPI, which also provides sophisticated methods for collective communication, where a set of procs communicate as a group, onesided communication, where one proc provides a "window" to portions of its memory that other procs can look through and much more. We will only list some of them, and that mostly to establish notation for communication patterns. More details can be found in the MPI-standard [1].

- **MPI_Gather:** A collective operation that gathers data from all ranks in a communicator on one specified rank (the "root").
- **MPI_Scatter:** A collective operation where one specified rank in a communicator (again, the "root") sends *a potentially different* message to each other rank in the same communicator.

- **MPI_Bcast:** A collective operation where one specified rank in a communicator (again, the "root") sends *the same* message to each other rank in the same communicator.
- **MPI_Reduce:** A collective operation that combines data from all ranks in a communicator and makes the result available on one "root" rank. The combination is done via an MPI_Op, which can be MPI_SUM, MPI_MAX, etc.
- **MPI_Allreduce:** The same as MPI_Reduce, but afterwards the combined data is available on all ranks in the communicator. Can be expressed as an MPI_Reduce followed by an MPI_Bcast operation.

Note 4.1. All of these are also available in nonblocking variations.

There are many more, we have only singled these out so that it is clear what we mean by a "gather-operation", a "reduce-operation" and so forth. These terms will only be used to describe communication patterns, in order to make clear which data has to go where in a communication step. The actual implementation is often realized differently, it might, for example, instead of an MPI_Allreduce, consist of a series of nonblocking send/recv operations. Also note that all of the above collective operations require all of the procs of a communicator to participate, which is not the situation we are usually in in Finite Elements. As we will see in the next section, each proc is typically most of the time only interested in exchanging messages with a few "neighbouring" procs, not all others.

4.3 The Distributed Finite Element Method

We will now define distributed meshes and Finite Element spaces and show how NGSolve adapts to the changes to the programming model and makes use of MPI when running on a cluster.

Definition 4.1: Mesh Partition

Let $\mathcal{T}_h$ be a regular partition of Ω, as in definition 2.1. Let $(\Omega_i)_{i\in J}$ be a finite partition of Ω, that is all (finitely many) Ω_i are open and

$$\bigcup_{i\in J} \overline{\Omega}_i = \overline{\Omega}$$

$$\Omega_i \cap \Omega_j = \emptyset \quad \forall i \neq j$$

We also require the partition to respect the triangulation, that is:

$$\forall T_h \in \mathcal{T}_h : \exists! i : T_h \subseteq \overline{\Omega}_i$$

The corresponding partition $(\mathcal{T}_i)_{i\in I}$ of the triangulation $\mathcal{T}_h$ is defined by

$$\mathcal{T}_i := \left\{T \in \mathcal{T}_h : T \subseteq \overline{\Omega}_i\right\}$$

Notation 4.3. As for the global partition $\mathcal{T}_h$, we will write $\mathcal{V}(\mathcal{T}_i)$ for the set of all vertices, $\mathcal{E}(\mathcal{T}_i)$ for the set of all edges and $\mathcal{F}(\mathcal{T}_i)$ for the set of all faces in $\mathcal{T}_i$.

Note 4.2. This definition does not allow for an overlap between different subdomains. Thus, different sub-meshes do not share cells but only vertices, edges and in three dimensions also faces. In other words, sub-meshes do not share any d-dimensional nodes.

It is evident how the concept of partitioning a mesh fits into the distributed programming model: Each proc gets assigned one of the subdomains and with it it's submesh. This is very standard approach is also the one taken in NGSolve. In Finite Elements, most operations are local in nature, for example a matrix vector multiplication will need access to degrees of freedoms sitting in all neighbouring nodes of a particular vertex in order to compute the value of the result there. When assembling the bilinear form matrices, we only need to access one element at a time. Data exchange most of the time only has to happen for operations that concern degrees of freedom that sit either on an interface or directly next to one. This naturally decouples the entire Finite Element Method into a row of local Finite Element Methods that have to synchronize and work in concert with each other wherever boundary nodes are concerned. We will now formally introduce distributed Finite Element spaces and talk about distributed linear algebra after that.

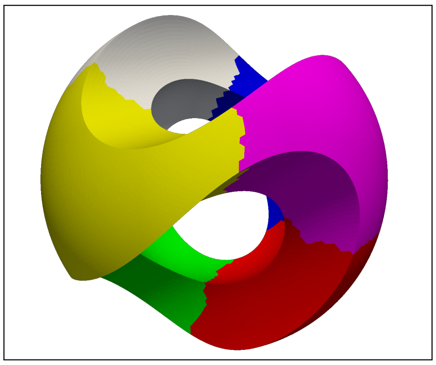


Figure 4.2: Partition of a mesh onto 6 procs.

Note 4.3. In practice, Netgen/NGSolve does not keep track of a particular conistent global enumeration, instead all nodes are only numbered locally, however the order of the numbering is kept consistent between different procs. Occasionaly some subset is ordered globally when needed.

Note 4.4. Altough in definition 4.1, the partition of Ω defines the partition of the mesh, in practice, this is handeled the other way around and the mesh is partitioned which then induces the partition of Ω. NGSolve uses the software library METIS for this purpose.

As we will see now, the partition of a mesh naturally induces a decomposition of the global Finite Element space into smaller, local Finite Element spaces. Although we are only concerned with the lowest order H^1 space in this work, we will show how distributed Finite Elelment Spaces can be defined in a general way.

Definition 4.2: Distributed Fintie Element Space

Given a Finite Element space V_h, defined by Finite Elements $(T, V_{h,T}, h_T, \Psi_T)$ and a function space X that imposes additional regularity, such that $V_h = \prod_{T\in\mathcal{T}_h} V_h, T \cap X$, and a partition of the triangulation as in definition 4.1, the local (sub-) Finite Element spaces are defined as

$$V_{h,i} := \left\{ u_{h|\Omega_i} : u_h \in V_h \right\}$$

With the restrictions $X_i := \left\{ f_{|\Omega_i} : f \in X \right\}$ of X, the subspaces can also be written as

$$V_{h,i} = \prod_{T\in\mathcal{T}_i} V_{h,T} \cap X_i$$

The (global) Finite Element space V_h then admits the representation

$$V_h = \prod_{i\in I} V_{h,i} \cap X$$

For each bilinear form $a(\cdot,\cdot)$ and linear form $f(\cdot)$ on V_h we can introduce their local restrictions

$$\begin{aligned} a_i(u,v) &:= a(u,v) && \forall u, v \in V_{h,i} \\ f_i(v) &:= f(v) && \forall v \in V_{h,i} \end{aligned}$$

Their matrix and vector representations will be written as $\mathbf{A}^i \in \mathbb{R}^{n_i\times n_i}$ and $\mathbf{b}^i \in \mathbb{R}^{n_i}$, with $n_i = \dim V_{h,i} = |\mathcal{V}(\mathcal{T}_i)|$.

Definition 4.3: Distributed lowest order H^1 Finite Element space

The local subspaces of the lowest order H^1 Finite Element space from definition 2.3 are

$$V_{h,i} = \prod_{T\in\mathcal{T}_i} V_{h,T} \cap C^0(\overline{\Omega}_i) \subseteq H^1(\Omega_i) \tag{4.1}$$

The space itself can also be written as

$$V_h = \prod_{i\in I} V_{h,i} \cap C^0(\overline{\Omega}) \subseteq H^1(\Omega) \tag{4.2}$$

In light of the equivalent representations (2.10) and (4.2) of V_h, the natural idea here is to view each subdomain as a kind of “makro Finite Element”. Now we can think back to theorem 2.8 and note 2.2, where we have shown how to implement the multiplication with $\mathbf{A}$ via the element matrices and the embeddings, without assembling $\mathbf{A}$ itself. Extending this from elements to makro elements, a.k.a subdomains, we now have an idea of how to realize multiplication with $\mathbf{A}$ only using the local contributions $\mathbf{A}^i$.
The next section will generalize and elaborate on this approach in more detail.

4.4 Distributed Linear Algebra

Now that we know how the gloal Finite Element space cna be represented by local ones and that the global bilinear form is just a sum of local contributions, represented by local matrices, we already have a good idea how to implement distributed lineara algebra. We will formalize this now.

4.4.1 Distributed Matrices and Vectors

Definition 4.4: Parallel Vector

Let $\mathbf{v} \in \mathbb{R}^n$ *be some global vector. For* $k \in \mathcal{P}$, *let* $I_k = \{l_0, l_1, \dots l_{n_k-1}\} \subseteq \{0, 1 \dots n-1\}$ *such that*

$$\bigcup_{k \in \mathcal{P}} I_k = \{0 \dots n-1\}$$

Let the embedding matrices $\mathbf{E}^k \in \mathbb{R}^{n \times n_k}$ *for* $k \in \mathcal{P}$ *be defined by*

$$\mathbf{E}^k_{ij} = \begin{cases} 1 & \text{if } i = l_j \\ 0 & \text{else} \end{cases}$$

A parallel vector is then a tuple $(\mathbf{v}^k)_{k \in \mathcal{P}}$ *with* $\mathbf{v}^k \in \mathbb{R}^{n_k}$.
We say that the global vector $\mathbf{v}$ *is represented by the parallel vector if* $(\mathbf{v}^k)_{k \in \mathcal{P}}$ *either*

1. $\mathbf{v} = \sum_{k \in \mathcal{P}} \mathbf{E}^k \mathbf{v}^k$, *in which case* $(\mathbf{v}^k)_{k \in \mathcal{P}}$ *is called* ***distributed.***
2. $\mathbf{v}^k = \mathbf{E}^{k,T} \mathbf{v}\ \forall k \in \mathcal{P}$, *in which case* $(\mathbf{v}^k)_{k \in \mathcal{P}}$ *case is called* ***cumulated.***

If $\mathbf{v}$ *is represented by* $(\mathbf{v}^k)_{k \in \mathcal{P}}$ *we will also write* $\mathbf{v} \cong (\mathbf{v}^k)_{k \in \mathcal{P}}$.
If the sets I_k *are pairwise disjoint,* $\mathbf{v}$ *has a unique, distributed and cumulated representation. If the sets* I_k *do overlap, vectors* v *in the global space* $\mathbb{R}^n$ *do not have a unique parallel representation.*

Note 4.5. A parallel vector representing a global vector is also said to have a parallel status, which can either be distributed or cumulated.

Definition 4.5: Parallel Matrix

Let $\mathbf{A}^{n \times n}$ *some global matrix, and* I_k *and* $\mathbf{E}^k$ *for* $k \in \mathcal{P}$ *given as in definition 4.4. A parallel matrix is a tuple* $(\mathbf{A}^k)_{k \in \mathcal{P}}$ *with* $\mathbf{A}^k \in \mathbb{R}^{n_k \times n_k}$. *We say that* $\mathbf{A}$ *is represented by* $(\mathbf{A}^k)_{k \in \mathcal{P}}$ *if*

$$\mathbf{A} = \sum_{k \in \mathcal{P}} \mathbf{E}^k \mathbf{A}^k \mathbf{E}^{k,T}$$

In that case, we will also write $\mathbf{A} \cong (\mathbf{A}^i)_i$.

Note 4.6. We only define one “valid” parallel status for parallel matrices, which is the analogue of the distributed parallel status for parallel vectors, as this is the one that arises naturally when assembling the local bilinear form contributions on each subdomain. In chapter 8, we will also see an example for a matrix analogon of a cumulated vector. In contrast to that, we need both distributed and cumulated statuses for parallel vectors, as we will see in the next sessions.

Global vectors always have a local representation, as long as $\bigcup_k I_k = \{0 \dots n-1\}$. For a global matrix $\mathbf{A}$ to have a local representation, we also need

$$\mathbf{A}_{ij} \neq 0 \Rightarrow \exists k \in P : \{i, j\} \subseteq I_k$$

Note 4.7. In many other software libraries, for example PETSc and and the hypre package, global matrices are simply distributed row-wise. This is, of course, also a practical approach and in fact, from a linear algebra point of view, simpler in many ways, however it implies a bigger distance between the linear algebra and Finite Element perspectives. As a sidenote, the NGSolve way of partitioning with overlapping rows also allows us to make do without a global enumeration, just so long as dofs are ordered consistently on all procs that share it.

4.4.2 Distributed Linear Algebra Operations

A global vector can have multiple representations by parallel vectors, and while the cumulated representation is unique, there can be multiple distributed representations. From a distributed representation, we can always get the cumulated one and from the cumulated one we can get any distributed one.

Lemma 4.1: Cumulating and Distributing Parallel Vectors

For a distributed parallel vector $\mathbf{v} \cong \mathbf{v} \cong (\mathbf{v}^k)_{k \in \mathcal{P}}$*, there is a unique cumulated parallel vector* $(\mathbf{w}^k)_{k \in \mathcal{P}} \cong \mathbf{v}$*. It is given by*

$$\mathbf{w}^k = \mathbf{E}^{k,T} \sum_{j \in \mathcal{P}} \mathbf{E}^j \mathbf{v}^k$$

For a cumulated parallel vector $\mathbf{v} \cong (\mathbf{v}^k)_{k \in \mathcal{P}}$*, there is no unique distributed representation, but a possible one is defined by*

$$\mathbf{w}^k = \mathbf{v}^k - \sum_{j=0}^{k-1} \mathbf{E}^{k,T} \mathbf{E}^j \mathbf{v}^j = \tag{4.3}$$

$$= \mathbf{E}^{k,T} (\mathbf{I} - \sum_{j=0}^{k-1} \mathbf{E}^j \mathbf{E}^{j,T}) \mathbf{v} \tag{4.4}$$

Proof. Directily follows form the definition of parallel vectors. □

Note 4.8. Distributing a parallel vector means just zeroing out all entries in the local vectors corresponding to degrees of freedom that are shared with a proc that has lower rank. This means the full value $\mathbf{v}_i$ is stored on the proc P^k with $k = \min\{j : i \in I_j\}$, that is, the proc that shares it with the lowest rank. This is an inherently local operation, no data has to be exchanged.

Cumulating a distributed vector requires communication. Every proc has to exchange a message with all other procs it shares a degree of freedom (DOF) with, containing the values of its' local vector for all shared DOFs.

Now that we are able to switch between different representations of global vectors at will, we can perform parallel linear algebra operations.

Lemma 4.2: Vector Scalar product

The scalar product between a distributed parallel vector $\mathbf{v} \cong (\mathbf{v}^k)_{k\in\mathcal{P}}$ *and a cumulated one* $\mathbf{w} \cong (\mathbf{w}^k)_{k\in\mathcal{P}}$ *is given by*

$$\langle \mathbf{v}, \mathbf{v} \rangle = \sum_{k\in\mathcal{P}} \left\langle \mathbf{v}^k, \mathbf{w}^k \right\rangle$$

Lemma 4.3: Matrix Vector Multiplication

For a cumulated parallel vector $\mathbf{v} \cong (\mathbf{v}^k)_{k\in\mathcal{P}}$ *and a parallel matrix* $\mathbf{A} \cong (\mathbf{A}^k)_{k\in\mathcal{P}}$*,* $(\mathbf{A}^k\mathbf{v}^k)$ *is a distributed representation of* $\mathbf{Ab}$*.*

$$\mathbf{Ab} = \sum_{k\in\mathcal{P}} \mathbf{E}^k\mathbf{A}^k\mathbf{b}^k$$

Proof. Follows directly from the definitions of parallel matrices and vectors. □

Note 4.9. Therefore, when we want to multiply a parallel matrix with a parallel vector, we first cumulate the vector, then multiply locally and end up with a distributed vector.

Parallel matrix-matrix multiplication is more difficult. Even when both matrices **A** and **B** can be represented locally, this is not necessarily true for their product because matrix-matrix multiplication can introduce nonzero entries

$$(\mathbf{AB})_{ij} \neq 0 \wedge \nexists k \in \mathcal{P} : \ \{i, j\} \subseteq I_k$$

This problem will return later on in chapter 8.

4.5 MPI-Parallelization in Netgen/NGSolve

We will now outline the key aspects of the distributed parallel implementation of NGSolve. For now, this helps with understanding what exactly is involved in each of the steps of a parallel matrix-vector multiplication and it will also build a base for understanding the difficulties we ran into during the development of AscAMG .

Note 4.10. One of the peculiarities of Netgen/NGsolve is that the global “master proc”, the proc with rank 0, not to be confused with the “master proc” of a particular DOF or node, never gets assigned a submesh, which means that it is idle most of the time. On occasion, however, it allows us to shift some computational work to the master without impacting the performance of other, concurrent computations. This has historically grown.

The two key classes in NGSolve that together manage to encapsulate much of the gritty details of MPI from the higher level components are the **ParallelDofs** and the **ParallelVector** classes.
The class **ParallelDofs**, locally on each proc mainly consists of a table, where the k'th row stores the list of other processors that shere the k'th local DOF. These are also

called the DistantProcs of the k'th DOF. **ParallelDofs** provides methods to access this information in different ways, the most important ones are:

- **GetDistantProcs():** Returns the list of all procs that share any DOF with the caller.
- **GetDistantProcs(int k):** Returns the list of procs the dof k is shared with.
- **GetExchangeDofs(int p):** Returns the list of DOFs that are shared with proc p.
- **IsMasterDof(int k):** Returns true if the proc is not shared with any proc with a lower rank, and false in any other case. Informally, it answers the question “Am I the master of this DOF?”.
- **GetMasterProc(int dof):** Returns the lowest rank of any proc that shares this dof (which is the master proc's).

It also provides the method **EnumerateGlobally**, with which one can find a global enumeration of some subset of DOFs. This is used occasionally, however keep in mind that there is no overarching global enumeration of DOFs in NGSolve.

It also features three methods that perform communication on given data:

- **ReduceDofData(FlatArray<T> data, MPI_Op op):** Input is an array with data for each local dof and an operation to perform on the data (for example MPI_SUM, MPI_MAX, etc…). Computes an array a, where the for each masterdof k, $a[k] = op(v_1, v_2, \dots)$, where $v_1 = \text{data}[k]$ and $v_2 \dots v_m$ are the values for the DOF the array has on the DOFs DistantProcs. For non-master DOFs k, $\text{data}[k] = a[k]$. Overwrites data with a. In particular, it returns an inconsistent array! It “reduces” information to the master of each dof. This can be thought of an MPI_Reduce for each DOF, where only the procs that share it participate.
- **ScatterDofData(FlatArray<T> data):** Input is an array with data for each local dof. Modifies data such that each entry has the same value across all procs, the one it initially had on the master proc. In particular, does not modify values of master DOFs. It “scatters” information from the master of each DOF to the others. This can be thought of an MPI_Bcast for each DOF, where only the procs that share it participate.
- **AllReduceDofData(FlatArray<T> data, MPI_Op op):** Performs ReduceDofData followed ScatterDofData. For each dof k it performs “op” on the set of values it has on all procs that share it and makes that information available to all of them. This is basically a “Cumulate”-operation, it is however only a moderately efficient implementation of it.

As iterative algorighms require us to do matrix-vector multiplications and evaluate scalar products in each iteration step, we have to pay dearly for any inefficiencies in this regard. In section 4.4.2, we have seen that these operations can be realized by local matrix-vector multiplications and scalar products combined with Cumulate- and Distribute- operations. The **ParallelVector** class of NGSolve is a wrapper around the other, sequential vector classes that provides exactly this functionality, implemented very efficiently.

A **ParallelVector** has a PARALLEL_STATUS, a variable that can take any of the values CUMULATED, DISTRIBUTED or NOT_PARALLEL. The most important methods **ParallelVector** provides above a sequential vector are:

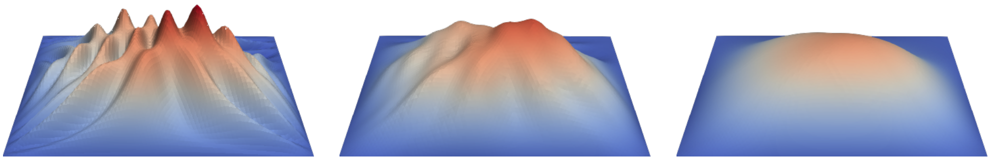
- **GetParallelStatus():** Returns the parallel status of the vector.

- **SetParallelStatus(PARALLEL_STATUS stat):** Sets the parallel status of the vector. This is useful occasionally, for example when zeroing out an entire vector or writing information of which we know that it is already cumulated into it.
- **Cumulate():** If the status is DISTRIBUTED, cumulates the vector. Otherwise, does nothing.
- **Distribute():** If the status is CUMULATED, distributes the vector by zeroing all entries of non master dofs. Otherwise, does nothing.

5 Error Smoothing

In this section, we will make observations about the behavior of certain **S**tationary **L**inear **I**therative **M**ethods (SLIMs) that will ultimately motivate the introduction of multigrid algorithms.

As we have seen in section 2.2, both Jacobi and Gauss-Seidel give very poor spectral bounds as $h \to 0$ and are insufficient as both preconditioners and stand alone solvers on their own, however this does not mean that *all* error components are eliminated equally slowly. In fact, error components that belong to eigenvectors of the error propagation matrix $\mathbf{M} = \mathbf{I} - \mathbf{NA}$ with small eigenvalues (which are made up of eigenvectors of $\mathbf{A}$ with large eigenvalues) are eliminated rapidly. In the case of the pure poisson problem (equation (2.1) with $\alpha = 1, \beta = 0$), the $\mathbf{A}$-norm is just the H^1-seminorm (of the V_h-funcion identified with the vector via the Galerkin isomorphism) and eigenvectors with large eigenvalues are highly oscillating functions. Conversely, eigenvectors with small eigenvalues, which are exactly the ones Jacobi and Gauss-Seidel can not eliminate effectively, are very smooth functions.
Figure 5.1 shows the evolution of the error when performing Gauss-Seidel on equation (2.1) with $\Omega = [0,1]^2$, $\alpha = 1$, homogenous dirichlet boundary conditions and $f = 1$. We see that the error looses it's oscillating components very rapidly, while its smooth components can not be treated effectively by Gauss-Seidel. This should not come as a big surprise, since one GS iteration essentially consists of a series of local updates for each degree of freedom which can only ever take into account the neighbouring DOFs. This phenomenon is called "error smoothing", and SLIMs that behave in this way are also called "smoothers".



a: initial error b: error after iteration 5 c: error after iteration 50

Figure 5.1: Error of GS applied to $-\Delta u = 1 \quad \text{in} \quad$ with homogeneous dirichlet condition.

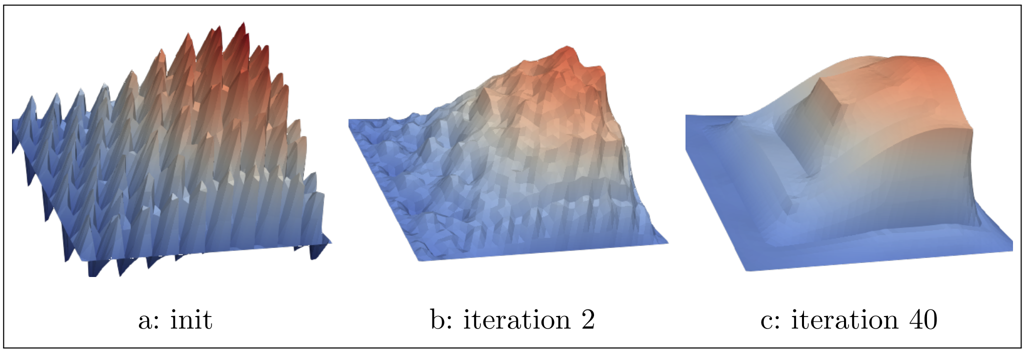


Figure 5.3: Error of GS applied to $-\nabla \cdot (\alpha \nabla u) = 1$ with homogeneous dirichlet condition and α from figure 5.2.

5.1 Algebraically Smooth Error

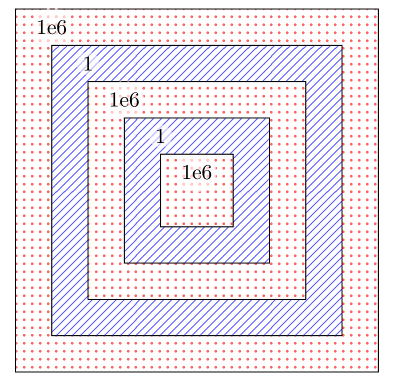


Figure 5.2: α for figure 5.3

So far, geometrically smooth errors are exactly those that are not effectively eliminated by a cheap linear iteration like J or GS, however this is not always the case. Figure 5.3 shows the same sitation as figure (5.1), however with a different coefficient α, as in figure 5.2. The coefficient is very large on one part of the domain (Ω_1) and moderately sized on the rest of the domain ($\Omega_2 = \Omega \setminus \Omega_1$). As can be seen, the error is now very smooth in Ω_1 and varies strongly in Ω_2. The reason for this behavior is that $\|\cdot\|_{\mathbf{A}}$ now weighs variation in Ω_1 much more more strongly than variation in Ω_2:

$$\|\mathbf{u}\|_{\mathbf{A}}^2 = 10^6 \cdot |u_h|_{H^1(\Omega_1)}^2 + |u_h|_{H^1(\setminus\Omega_2)}^2$$

This kind of error is called purely "algebraically smooth", which means nothing else than that it cannot be effectively eliminated by a smoother. As we have seen now, algebraic smoothness does not necessarily translate to geometric smoothness, although it does so in many cases.

In [7], a SLIM is said to satisfy the *soothing property* if for some $\sigma > 0$

$$\|\mathbf{Me}\|_{\mathbf{A}}^2 \leq \|\mathbf{e}\|_{\mathbf{A}}^2 - \sigma \|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2 \tag{5.1}$$

Here, the $\mathbf{D}^{-1}$-norm is used in order to retain the scaling with respect to $\mathbf{D}$ between the two terms on the right hand side. This condition says that errors for which $\|\mathbf{e}\|_{\mathbf{A}}^2$ and $\|\mathbf{Ae}\|_{\mathbf{D}^{-1}}^2$ are comparably large will be eliminated efficiently. For these terms to be comparable, $\mathbf{e}$ has to be mostly a linear combination of eigenvectors of $\mathbf{A}$ with large eigenvalues. We can also see that errors that are mostly made up of components with small eigenvalues will be reduced only very inefficiently. For a SLIM it is equivalent to satisfy the smoothing property and to satisfy

$$\sigma \mathbf{W}^T \mathbf{D}^{-1} \mathbf{W} \leq \mathbf{W} + \mathbf{W}^T - \mathbf{A} \tag{5.2}$$

for some $\sigma > 0$. This condition can also be obtained by a combination of conditions (3.4) and (3.5) from theorem 3.3.

6 Multigrid Methods

This chapter will introduce the basic multigrid-approach with its two main variations, geometric and algebraic multigrid and hightlight the differences between the two. We will look at algebraic multigrid in more detail in section 7.

6.1 The Two-Grid Algorithm

They key to the multigrid idea are the observations made in section 5, that is that error components that are not handled efficiently by a smoother can be characterized by their (geometric or algebraic) smoothness. In the simplest case, the approach is to combine a computationally cheap smoother, which takes care of high frequency error components with a so called *coarse grid correction* that takes care of the low frequency error components.

Assume that, in addition to the finite element space V_h, we also have access to V_H, which is itself a lowest order H^1-finite element space defined on a triangulation $\mathcal{T}_H$ of Ω with mesh size $H > h$. The coarse triangulation should be such that $V_h \subset V_H$. We will write $\mathbf{A}_h, \mathbf{A}_H$ to distinguish the finite element matrices of V_h and V_H and n_h, n_H for the dimensions of V_h and V_H. The prolongation matrix $\mathbf{P} \in \mathbb{R}^{n_H \times n_h}$ is the linear operator that is defined by

$$\mathbf{P}G_H^{-1}u_H = G_h^{-1}u_H \quad \forall u_H \in V_H$$

with the galerkin isomorphisms G_h, G_H. It is just the embedding of V_H into V_h brought to the coordinate space. Because

$$\left\langle \mathbf{A}_h G_h^{-1}u_H, G_h^{-1}v_H \right\rangle = a(u_H, v_H) = \left\langle \mathbf{A}_H G_H^{-1}u_H, G_H^{-1}v_H \right\rangle \quad \forall u_H, v_H \in V_H$$

the coarse finite element matrix can be expressed as:

$$\mathbf{A}_H = \mathbf{P}^T \mathbf{A}_h \mathbf{P}$$

At this point, remember that for ease of notation we use u_h and $\mathbf{u}$ for $u_h \in V_h$ and $\mathbf{u} \in \mathbb{R}^{n_h}$ such that $G_h u_h = \mathbf{u}$.

Applying a couple of smoothing steps to equation

$$\mathbf{A}_h \mathbf{u} = \mathbf{b}$$

gives us an approximate solution u_h to

$$a(u_h, v_h) = f(v_h) \quad \forall v_h \in V_h$$

such that with the true solution $\hat{u}_h$, the error $e_h = \hat{u}_h - u_h$ consists mainly of geometrically smooth components and can be approximated well by some $e_H \in V_H$. Therefore updating

$$u_h \to u_h + e_H$$

should yield a good approximation for $u_h + e_h = \hat{u}_h$.

The choice for e_H that minimizes $\|\hat{u}_h - (u_h + e_H)\|_A = \|e_h - e_H\|_A$, is just the A-orthogonal projection of e_h onto V_H, which is the solution to

$$a(e_H, v_H) = a(e_h, v_H) = a(\hat{u}_h, v_H) - a(u_h, v_H) = f(v_H) - a(u_h, v_H) \quad \forall v_H \in V_H$$

With $\tilde{u}_h \coloneqq u_h + e_H$ and $\tilde{e}_h \coloneqq \hat{u}_h - \tilde{u}_h$ we have $\tilde{e}_h = (I - P_A)e_h \perp_A V_H$, where P_A is the A-orthogonal projector onto V_H. Because

$$\begin{aligned} f(v_H) - a(u_h, v_H) &= \left\langle \mathbf{b} - \mathbf{A}_h\mathbf{u}, G_h^{-1} v_H \right\rangle = \left\langle \mathbf{b} - \mathbf{A}_h\mathbf{u}, \mathbf{P} G_H^{-1} v_H \right\rangle = \\ &= \left\langle \mathbf{P}^T \left(\mathbf{b} - \mathbf{A}_h\mathbf{u}\right), G_H^{-1} v_H \right\rangle \end{aligned}$$

the vector representation of e_H (w.r.t. the V_H-basis) is $\mathbf{e} = \mathbf{A}_H^{-1}\mathbf{P}^T(\mathbf{b} - \mathbf{A}_h\mathbf{u})$, and with

$$G_h^{-1}(u_h + e_H) = \mathbf{u} + G_h^{-1} e_H = \mathbf{u} + \mathbf{P}\mathbf{e}$$

the coarse grid correction in the coordinate space (of V_h) reads as

$$\begin{aligned} \mathbf{r} &= \mathbf{b} - \mathbf{A}_h\mathbf{u} \\ \tilde{\mathbf{u}} &= \mathbf{u} + \mathbf{P}\mathbf{A_H}^{-1}\mathbf{P}^T\mathbf{r} \end{aligned}$$

Given a starting vector $\mathbf{u}^0$, the standard **T**wo-**G**rid (TG) algorithm (algorithm 2) now consists of the following three steps

1. One iteration of the SLIM, "Pre-Coarsening"
   $$\mathbf{u}^1 = \mathbf{M}\mathbf{u}^0 + \mathbf{N}\mathbf{b}$$
2. Coarse Grid Correction
   $$\mathbf{u}^2 = \mathbf{u}^1 + \mathbf{P}\mathbf{A_H}^{-1}\mathbf{P}^T(\mathbf{b} - \mathbf{A}_h\mathbf{u}^1)$$
3. One iteration of the transposed SLIM, "Post-Coarsening"
   $$\mathbf{u}^3 = \mathbf{M}^T\mathbf{u}^2 + \mathbf{N}^T\mathbf{b}$$

Algorithm 2 Two-Grid Algorithm.

```
procedure TG($A, b, x_0$)
    $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{N}(\mathbf{b} - \mathbf{A}\mathbf{x}_0)$        ▷ (Pre)-Smooth
    $\mathbf{r}_H = \mathbf{P}^T(\mathbf{b} - \mathbf{A}\mathbf{x}_1)$        ▷ Restrict residuum
    $\mathbf{x}_H = \mathbf{A}_H^{-1}\mathbf{r}_H$        ▷ Coarse level solve
    $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{P}\mathbf{x}_H$        ▷ Coarse grid correction
    $\mathbf{x}_3 = \mathbf{x}_2 + \mathbf{N}^T(\mathbf{b} - \mathbf{A}\mathbf{x}_2)$        ▷ (Post)-Smooth
    return $x_3$
```

Remembering that with the error propagation matrix of the SLIM, $\mathbf{M}$, the error propagation matrix of the transposed SLIM is its $\langle\cdot,\cdot\rangle_\mathbf{A}$-adjoint $\mathbf{M}^*$, the error propagation matrix of the TG algorithm can be written as:

$$\mathbf{M}_{TG} = \mathbf{M}^* \left(\mathbf{I} - \mathbf{P}\mathbf{A}_H^{-1}\mathbf{P}^T\mathbf{A}\right)\mathbf{M} \tag{6.1}$$

As $(\mathbf{I} - \mathbf{P}\mathbf{A}_H^{-1}\mathbf{P}^T\mathbf{A})$ is the $\mathbf{A}$-orthogonal projector onto **ran** $\mathbf{P}$, $\mathbf{M}_{TG}$ is self adjoint wrt. $\langle\cdot,\cdot\rangle_\mathbf{A}$. We can also write the two-grid algorithm as one step of a Richardson-iteration with preconditioner

$$\mathbf{B}_{TG}^{-1} = \widetilde{\mathbf{W}} + \left(\mathbf{I} - \mathbf{N}^{-T}\mathbf{A}\right)\mathbf{P}\mathbf{A}_H^{-1}\mathbf{P}^T\left(\mathbf{I} - \mathbf{N}^{-1}\mathbf{A}\right) \tag{6.2}$$

Note that $\mathbf{B}_{TG}^{-1}$ is a symmetric matrix, and we can therefore use it in the PCG (algorithm 1). This would not be the case if we used only one smoothing step either before or after the coarse grid correction.

Note 6.1. If $\mathbf{x}_0 = 0$ in algorithm 2, it returns $\mathbf{x}_3 = \mathbf{B}_{TG}^{-1}\mathbf{b}$.

6.2 The Multi-Grid Algorithm

In principle, we could replace the exact inverse $\mathbf{A}_H^{-1}$ in algorithm 2 with $\mathbf{C}^{-1}$ where $\mathbf{C}$ is a good preconditioner for $\mathbf{A}_H$. For example, we could use another two-grid preconditioner, only now we start from the coarse level. In that case what we end up with is called a three-level method. Doing this recursively, until one arrives at a small enough space where direct solution is feasible, leads to the V-cycle multigrid-method (algorithm 3). The ingredients are now a nested series of finite element spaces

$$V_h = V_0 \supset V_1 \supset V_2 \ldots \supset V_L$$

of dimension n_l with finite element matrices $\mathbf{A}_l$ and SLIMs defined by $\mathbf{N}_l$ for every level $l \in \{0 \ldots L\}$ as well as prolongation matrices $\mathbf{P}_l \in \mathbb{R}^{n_l \times n_{l+1}}$ for every level but the last.

Algorithm 3 Multi-Grid V-cycle Algorithm.

```
procedure MG($b, x_0, l$)
    if $l = L$ then
        return $\mathbf{A}_L^{-1}\mathbf{b}$                               ▷ Coarsest level solve
    else
        $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{N}_l(\mathbf{b} - \mathbf{A}_l\mathbf{x}_0)$          ▷ (Pre)-Smooth
        $\mathbf{r}_c = \mathbf{P}_l^T(\mathbf{b} - \mathbf{A}_l\mathbf{x}_1)$                ▷ Restrict residuum
        $\mathbf{x}_c = \mathrm{MG}(\mathbf{r}_c, 0, l+1)$                      ▷ Recurse
        $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{P}_l\mathbf{x}_c$                        ▷ Coarse level correction
        $\mathbf{x}_3 = \mathbf{x}_2 + \mathbf{N}_l^T(\mathbf{b} - \mathbf{A}_l\mathbf{x}_2)$          ▷ (Post)-Smooth
        return $\mathbf{x}_3$
```

The error propagation matrix takes the form

$$\mathbf{M}_V = \mathbf{M}_0^*\mathbf{P}_0\mathbf{M}_1^*\mathbf{P}_1 \ldots \mathbf{M}_{L-1}^* \left(\mathbf{I} - \mathbf{P}_{L-1}\mathbf{A}_L^{-1}\mathbf{P}_{L-1}^T\mathbf{A}_{L-1}\right) \mathbf{M}_{L-1} \ldots \mathbf{P}_1^T\mathbf{M}_1\mathbf{P}_0^T\mathbf{M}_0 \quad (6.3)$$

Note 6.2. There are other variations of multigrid cycles that are mainly used when employing multigrid on its own instead of as accelerators for a krylov space method like CG. For a W-cycle, instead of going back down a level after the coarse grid correction, we instead update the residuum and goe up a level once more. That way, we smooth 2^l times on level l. In a **F**ull **M**ulti**G**rid (FMG)-cycle, one starts with an exact solution on the coarsest level L, then prolongates that solution to the next coarsest level $L-1$ and calls a V-cycle starting on level $L-1$ to get a new solution, which is then prolongated to level $L-2$ and so on. As we are mainly interested in AMG in combination with CG, we will restrict ourselfs to V-cycles.

6.3 Advantages and Weakpoints of the Geometric Multigrid Method

Assuming we choose $h_{l+1} \approx 2h_l$ we get $n_{l+1} \approx 2^{-d}n_l$. As typically, one iteration of a SLIM requires $\mathcal{O}(n)$ operations, by the convergence of the geometric series, the V-cycle multigrid algorithm also has linear operator complexity. For the simple case where α and β are somewhat smooth in equation 2.1, one can show that $\kappa(\mathbf{B}_V^{-1}\mathbf{A}) = \mathcal{O}(1)$. A very nice proof for this that looks at multigrid algorithms in the context of general subspace correction methods is given in [11].

If applicable, geometric multigrid is a preconditioner with linear operator complexity and ideal spectral bounds!

This is the best possible behavior we could ever hope for. Unfortunately, there are some restrictions to the applicability of this simple kind of geometric multigrid approach. First of all, we need the spaces V_l to be nested, $V_0 \subset V_1 \subset \ldots \subset V_L$. For that to be true, each triangulation has to be a refinement of the next coarser one. This means that we have to be able to triangulate the domain Ω with with a very coarse h_L, which poses a problem for complex Ω.

Let us also remember what we discovered in section 5, specifically that algebraic smoothness and geometric smoothness are not at all the same thing if, for example, α varies strongly or even has large jumps. The entire idea of multigrid was based on the fact that error components that can not be efficiently adressed by the smoother can be approximated well on the coarse grid, that is that algebraically smooth errors are also geometrically smooth!

There are two basic approaches to deal with these problems. On the one hand, one can try to fit the multigrid method to the problem and the geometry at hand, which usually involves more sophisticated smoothers. The other is to stick to cheap, simple smoothers like GS or Chebyshev-smoothers but to completely throw the idea that the coarse spaces have to be finite element spaces defined coarse meshes overboard. Instead, one constructs the coarse spaces such that they contain the algebraically smooth vectors. Methods based on this idea are called algebraic multigrid methods and will be discussed starting with chapter 7.

7 Algebraic Multigrid

In section 5 we observed that in many cases, algebraically smooth vectors are also geometrically smooth and in section 6 we combined this with the fact that finite element spaces defined on some coarser triangulation are on the one hand of much lower dimension than the original space and on the other hand consist of just such functions to arrive at geometric multigrid methods. Now we will travel down a different path and try to **construct** coarse spaces, again of decreasing dimension, such that they contain algebraically smooth vectors. After fixing some ideas on how to go about doing this, section 7.1 will show conditions for the convergence of the algebraic two-grid algorithm and we will discuss the difficulty of proofing convergence of the algebraic multigrid method in section 7.2.

In principle, in order for the two-grid algorithm (algorithm 2) to be defined, we need a smoother defined by a matrix $\mathbf{N}$ as well as the prolongation matrix $\mathbf{P}$ and the coarse system matrix. We will now, as there is no longer an actual coarse "h" involved, write $\mathbf{A}_c$ for the coarse matrix. As already mentioned, we want to construct the coarse space such that it contains algebraically smooth vectors, which means that we fit the coarse space to the given smoother and we do not need to be concerned with choosing $\mathbf{N}$ right now. Since we think of the prolongation matrix as the embedding of the, to be constructed, coarse space, which we will now call V_c into V_h, we can simply define the coarse system matrix as

$$\mathbf{A}_c \coloneqq \mathbf{P}^T\mathbf{A}\mathbf{P} \tag{7.1}$$

This preserves the property that the coarse grid correction is an $\mathbf{A}$-orthogonal projection.

We see that our task is to, given a smoother $\mathbf{N}$*, construct* $\mathbf{P}$ *such that the two grid method converges.*

As the columns of $\mathbf{P}$ are just the coordinates of the V_c basis wrt. the V_h basis, this is equivalent to constructing the coarse space V_c.
The extension of this approach from a two-grid method to a multigrid method is not quite as straightforward as it was for geometric multigrid. In principle one could use information about the triangulation or the exact shape of the base functions when constructing $\mathbf{P}_0$, the prolongation on the finest level, however, as there is no coarse mesh present in the background, we usually cannot rely on this kind of information on any coarse level. Algebraic multigrid methods therefore construct the prolongation matrices only on basis of $\mathbf{A}$, that is on basis of the matrix graph and the coefficients of $\mathbf{A}$.
Because the coarse grid correction now per construction takes care of the algebraically smooth error components, we are free to choose a smoother mostly based on its computational cost. The standard choices here are properly dampened Jacobi, some variant of Gauss-Seidel and polynomial Chebyshev smoothers.

When constructing **P**, we have two conflicting goals in mind; on the one hand we want the coarse space to be big enough to approximate all algebraically smooth vectors as well as possible and on the other hand we want to keep the operator complexity of the entire multigrid-cycle as small as possible, which means that we want the coarse spaces to be as small as possible and we want to keep $\mathbf{A}_c$ as sparse as we possibly can.

The rest of this chapter will present some established results on sufficient conditions posed to **P** and **N** that imply convergence of the two grid method method. We will finish the chapter with some remarks on the difficulties of analyzing the algebraic multigrid method. The actual construction of the prolongation matrix **P** and the exact choice of the smoother will be discussed in chapter 8.

7.1 Analysis of the two grid method

In this section we will show conditions for **N** and **P** that guarantee convergence of the two-grid method or, equivalently, spectral bounds for the two-grid preconditioner while sticking to the way things are presented in the relevant section of [10].
We will need the following lemma for the proof of the main theorem.

Lemma 7.1

*Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be SPD and let V be an m-dimensional subspace of $\mathbb{R}^n$ with an orthonormal basis $\{q_1 \dots q_m\}$ of V and an orthonormal basis $\{p_1, p_2 \dots p_{n-m}\}$ of $V^\perp$. Let $\mathbf{Q} := (q_1, q_2 \dots q_m) \in \mathbb{R}^{n\times m}$ and $\mathbf{P} := (p_1, p_2 \dots p_{m-n}) \in \mathbb{R}^{n\times(n-m)}$
Then*

$$\left\langle (\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})^{-1}\mathbf{x}_q, \mathbf{x}_q \right\rangle = \inf_{x_p\in\mathbb{R}^{n-m}} \langle \mathbf{A}(\mathbf{Q}x_q + \mathbf{P}x_p), (\mathbf{Q}x_q + \mathbf{P}x_p)\rangle \quad \forall x_q \in \mathbb{R}^m \tag{7.2}$$

Proof. Let us first show an identity for the matrix $\mathbf{S} := (\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})^{-1}$:

$$\begin{aligned}\mathbf{Q}\mathbf{x}_q = \mathbf{A}\mathbf{A}^{-1}\mathbf{Q}\mathbf{x}_q &= \mathbf{A}(\mathbf{Q}(\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})\mathbf{x}_q + \mathbf{P}(\mathbf{P}^T\mathbf{A}^{-1}\mathbf{Q})\mathbf{x}_q) = \\ &= \mathbf{Q}[(\mathbf{Q}^T\mathbf{A}\mathbf{Q})(\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})x + (\mathbf{Q}^T\mathbf{A}\mathbf{P})(\mathbf{P}^T\mathbf{A}^{-1}\mathbf{Q})x] + \\ &+ \mathbf{P}[(\mathbf{P}^T\mathbf{A}\mathbf{Q})(\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})x + (\mathbf{P}^T\mathbf{A}\mathbf{P})(\mathbf{P}^T\mathbf{A}^{-1}\mathbf{Q})x]\end{aligned}$$

As $\mathbf{Q}\mathbf{x}_q \perp \mathbf{ran}\ \mathbf{P}$, the second term above, $\mathbf{P}[\dots]$, must be 0 and the first one, $\mathbf{Q}[\dots]$, must be $\mathbf{Q}\mathbf{x}_q$. From this we get

$$(\mathbf{P}^T\mathbf{A}^{-1}\mathbf{Q}) = -(\mathbf{P}^T\mathbf{A}\mathbf{P})^{-1}(\mathbf{P}^T\mathbf{A}\mathbf{Q})(\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})$$

Inserting this in the first expression yields

$$\mathbf{I} = [(\mathbf{Q}^T\mathbf{A}\mathbf{Q}) - (\mathbf{Q}^T\mathbf{A}\mathbf{P})(\mathbf{P}^T\mathbf{A}\mathbf{P})^{-1}(\mathbf{P}^T\mathbf{A}\mathbf{Q})](\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})$$

As both matrices are symmetric, we have an identity for the matrix in question

$$\mathbf{S} = [(\mathbf{Q}^T\mathbf{A}\mathbf{Q}) - (\mathbf{Q}^T\mathbf{A}\mathbf{P})(\mathbf{P}^T\mathbf{A}\mathbf{P})^{-1}(\mathbf{P}^T\mathbf{A}\mathbf{Q})]$$

Now, using the ontained identity and some elementary manipulations,

$$\begin{aligned}\mathbf{A} &= (\mathbf{Q}\mathbf{Q}^T + \mathbf{P}\mathbf{P}^T)\mathbf{A}(\mathbf{Q}\mathbf{Q}^T + \mathbf{P}\mathbf{P}^T) = \\ &= \mathbf{Q}(\mathbf{Q}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T + \mathbf{Q}(\mathbf{Q}^T\mathbf{A}\mathbf{P})\mathbf{P}^T + \mathbf{P}(\mathbf{P}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T + \mathbf{P}(\mathbf{P}^T\mathbf{A}\mathbf{P})\mathbf{P}^T \\ &= \mathbf{Q}(\mathbf{Q}^T\mathbf{A}^{-1}\mathbf{Q})^{-1}\mathbf{Q}^T + \mathbf{Q}(\mathbf{Q}^T\mathbf{A}\mathbf{P})\mathbf{P}^T + \mathbf{P}(\mathbf{P}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T + \mathbf{P}(\mathbf{P}^T\mathbf{A}\mathbf{P})\mathbf{P}^T + \\ &+ \mathbf{Q}(\mathbf{Q}^T\mathbf{A}\mathbf{P})(\mathbf{P}^T\mathbf{A}\mathbf{P})^{-1}(\mathbf{P}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T = \\ &= \mathbf{Q}\mathbf{S}\mathbf{Q}^T + [(\mathbf{P}(\mathbf{P}^T\mathbf{A}\mathbf{P}) + \mathbf{Q}(\mathbf{Q}^T\mathbf{A}\mathbf{P})](\mathbf{P}^T\mathbf{A}\mathbf{P})^{-1}[\mathbf{P}^T\mathbf{A}\mathbf{P})\mathbf{P}^T + (\mathbf{P}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T]\end{aligned}$$

This means that $\mathbf{Q}\mathbf{S}\mathbf{Q}^T \leq \mathbf{A}$, which shows (7.2) with "$\leq$" instead of "$=$" and because

$$\left\langle \mathbf{S}\mathbf{Q}^T\mathbf{x}, \mathbf{Q}^T\mathbf{x}\right\rangle = \langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle \quad \Leftrightarrow \quad (\mathbf{P}^T\mathbf{A}\mathbf{P})\mathbf{P}^T + (\mathbf{P}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T)\mathbf{x} = 0$$

is the case iff.

$$\mathbf{P}^T\mathbf{x} = (\mathbf{P}^T\mathbf{A}\mathbf{P})^{-1}(\mathbf{P}^T\mathbf{A}\mathbf{Q})\mathbf{Q}^T\mathbf{x}$$

the inf is attained for all $\mathbf{Q}^T\mathbf{x} \in \mathbb{R}^m$.

□

Let us remember the representation of the two-grid preconditioner

$$\mathbf{B}_{TG}^{-1} = \widetilde{\mathbf{W}} + \left(\mathbf{I} - \mathbf{N}^{-T}\mathbf{A}\right)\mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^T\left(\mathbf{I} - \mathbf{N}^{-1}\mathbf{A}\right) \tag{7.3}$$

ant the two-grid error propagation is

$$\mathbf{M}_{TG} = \left(\mathbf{I} - \mathbf{B}_{TG}^{-1}\mathbf{A}\right) = \mathbf{M}^*\left(\mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^T\mathbf{A}\right)\mathbf{M} \tag{7.4}$$

The following well known result, the proof of wich we have adapted from [10], is the basis for the construction of the prolongation matrix.

Theorem 7.1: Convergence of the Two-Grid Method

Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ be SPD, let $\mathbf{N} \in \mathbb{R}^{n\times n}$ define a SLIM such as in definition 3.2 with $\widetilde{\mathbf{N}}$ as in definition 3.5 and let $\mathbf{P} \in \mathbb{R}^{n\times m}$ with $m < n$ be a prolongation matrix. The spectral bounds for preconditioning by performing one step of the the 2-grid algorithm (alg. 2) with starting value $\mathbf{x}_0 = 0$ are given by

$$\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle \leq \langle \mathbf{B}_{TG}\mathbf{x}, \mathbf{x}\rangle \leq K_{TG}\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle \tag{7.5}$$

where K_{TG} is a constant such that

$$\inf_{\mathbf{x}_c\in\mathbb{R}^m} \|\mathbf{x} - \mathbf{P}\mathbf{x}_c\|_{\widetilde{\mathbf{W}}} \leq K_{TG}\|\mathbf{x}\|_{\mathbf{A}} \quad \forall \mathbf{x} \in \mathbb{R}^n \tag{7.6}$$

For the error propagation matrix of the two-grid algorithm $\mathbf{M}_{TG}$ we have

$$\|\mathbf{M}_{TG}\|_{\mathbf{A}} \leq 1 - \frac{1}{K_{TG}} \tag{7.7}$$

Proof. begin by showing (7.7) with K_{TG} given by (7.6), which implies the upper bound in (7.5).

Using (7.4) and the fact that $\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}} := \mathbf{I} - \mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^T\mathbf{A}$ is the $\mathbf{A}$-orthogonal projector onto **ran** $\mathbf{P}^{\perp_{\mathbf{A}}}$, we see that

$$\mathbf{M}_{\mathrm{TG}} = \mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{M} = \mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})^2\mathbf{M} = [(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{M}]^* \left[(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\,\mathbf{M}\right]$$

$$\Rightarrow \|\mathbf{M}_{\mathrm{TG}}\|_{\mathbf{A}} = \|(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{M}\|_{\mathbf{A}}^2 = \|\mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\|_{\mathbf{A}}^2$$

Now, with

$$\begin{aligned}(\mathbf{M}^*)^T\mathbf{A}\mathbf{M} = (\mathbf{I} - \mathbf{A}\mathbf{N})\mathbf{A}(\mathbf{I} - \mathbf{N}^T\mathbf{A}) &= \\ &= \mathbf{A}^{\frac{1}{2}}\left[\mathbf{I} - \mathbf{A}^{\frac{1}{2}}\mathbf{N}(\mathbf{N}^{-T} + \mathbf{N}^{-1} - \mathbf{A})\mathbf{N}^{\mathbf{T}}\mathbf{A}^{\frac{1}{2}}\right]\mathbf{A}^{\frac{1}{2}} = \\ &= \mathbf{A}^{\frac{1}{2}}\left[\mathbf{I} - \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}}\right]\mathbf{A}^{\frac{1}{2}}\end{aligned}$$

we get:

$$\begin{aligned}\|\mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\|_{\mathbf{A}}^2 &= \sup_{\mathbf{x}} \frac{\|\mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}\|_{\mathbf{A}}}{\|\mathbf{x}\|_{\mathbf{A}}} = \sup_{\mathbf{x}} \frac{\langle \mathbf{A}\mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}, \mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}\rangle}{\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle} = \\ &= \sup_{\mathbf{x}} \frac{\left\langle (\mathbf{M}^*)^T\mathbf{A}\mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}, (\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}\right\rangle}{\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle} = \\ &= \sup_{\mathbf{x}} \frac{\left\langle (\mathbf{I} - \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}})\mathbf{A}^{\frac{1}{2}}(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}, \mathbf{A}^{\frac{1}{2}}(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{x}\right\rangle}{\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle} = \\ &= \sup_{\mathbf{x}} \frac{\left\langle (\mathbf{I} - \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}})\mathbf{A}^{\frac{1}{2}}(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{A}^{-\frac{1}{2}}\mathbf{x}, \mathbf{A}^{\frac{1}{2}}(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{A}^{-\frac{1}{2}}\mathbf{x}\right\rangle}{\langle \mathbf{x}, \mathbf{x}\rangle}\end{aligned}$$

As $(\mathbf{I} - \overline{\boldsymbol{\pi}}_{\mathbf{A}}) := \mathbf{A}^{\frac{1}{2}}(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\mathbf{A}^{-\frac{1}{2}} = \mathbf{I} - \mathbf{A}^{\frac{1}{2}}\mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^T\mathbf{A}^{\frac{1}{2}}$ is an orthogonal projector

$$\begin{aligned}\|\mathbf{M}^*(\mathbf{I} - \boldsymbol{\pi}_{\mathbf{A}})\|_{\mathbf{A}}^2 &= \sup_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I} - \overline{\boldsymbol{\pi}}_{\mathbf{A}})} \left(1 - \frac{\left\langle \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}}\mathbf{x}, \mathbf{x}\right\rangle}{\langle \mathbf{x}, \mathbf{x}\rangle}\right) = \\ &= 1 - \inf_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I} - \overline{\boldsymbol{\pi}}_{\mathbf{A}})} \frac{\left\langle \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}}\mathbf{x}, \mathbf{x}\right\rangle}{\langle \mathbf{x}, \mathbf{x}\rangle} = \\ &= 1 - \left[\sup_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I} - \overline{\boldsymbol{\pi}}_{\mathbf{A}})} \frac{\langle \mathbf{x}, \mathbf{x}\rangle}{\left\langle \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}}\mathbf{x}, \mathbf{x}\right\rangle}\right]^{-1}\end{aligned}$$

We will now show that

$$\sup_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I} - \overline{\boldsymbol{\pi}}_{\mathbf{A}})} \frac{\langle \mathbf{x}, \mathbf{x}\rangle}{\left\langle \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}}\mathbf{x}, \mathbf{x}\right\rangle} \leq K_{\mathrm{TG}}$$

For that, let $W := \mathbf{ran}\ (\mathbf{I} - \overline{\boldsymbol{\pi}}_{\mathbf{A}})$, $m := \mathbf{dim}\ W$ and let $\mathbf{Q} \in \mathbb{R}^{n\times m}$ such that $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_m$ and $\mathbf{Q}\mathbf{Q}^T$ is the orthogonal projector onto W. Finally, with $\mathbf{R} \in \mathbb{R}^{n\times(n-m)}$ such that $\mathbf{R}^T\mathbf{R} = \mathbf{I}_{n-m}$ and such that $\mathbf{R}\mathbf{R}^T$ is the orthogonal projector onto $W^\perp$, we can use theorem 7.1 to see

$$\sup_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})} \frac{\langle \mathbf{x},\mathbf{x}\rangle}{\left\langle \mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}}\mathbf{x},\mathbf{x}\right\rangle} = \sup_{\mathbf{x}_c\in\mathbb{R}^m} \frac{\langle \mathbf{x}_c,\mathbf{x}_c\rangle}{\left\langle \mathbf{Q}^T(\mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}})\mathbf{Q}\mathbf{x}_c,\mathbf{x}_c\right\rangle} =$$

$$= \sup_{\mathbf{x}_c\in\mathbb{R}^m} \frac{\left\langle [\mathbf{Q}^T(\mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}})\mathbf{Q}]^{-1}\mathbf{x}_c,\mathbf{x}_c\right\rangle}{\langle \mathbf{x}_c,\mathbf{x}_c\rangle} = \sup_{\mathbf{x}_c\in\mathbb{R}^m} \frac{\left\langle [\mathbf{Q}^T(\mathbf{A}^{\frac{1}{2}}\widetilde{\mathbf{N}}\mathbf{A}^{\frac{1}{2}})\mathbf{Q}]^{-1}\mathbf{x}_c,\mathbf{x}_c\right\rangle}{\langle \mathbf{x}_c,\mathbf{x}_c\rangle} \overset{7.1}{=}$$

$$= \sup_{\mathbf{x}_c\in\mathbb{R}^m} \frac{\inf_{\mathbf{x}_r\in\mathbb{R}^{n-m}} \left\langle (\mathbf{A}^{-\frac{1}{2}}\widetilde{\mathbf{N}}^{-1}\mathbf{A}^{-\frac{1}{2}})(\mathbf{Q}\mathbf{x}_c+\mathbf{R}\mathbf{x}_r),(\mathbf{Q}\mathbf{x}_c+\mathbf{R}\mathbf{x}_r)\right\rangle}{\langle \mathbf{x}_c,\mathbf{x}_c\rangle} =$$

$$= \sup_{\mathbf{x}\in\mathbf{ran}\ \mathbf{A}^{\frac{1}{2}}(\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{A}^{-\frac{1}{2}}} \ \inf_{\mathbf{y}\in\mathbf{ran}\ \mathbf{A}^{\frac{1}{2}}\boldsymbol{\pi}_\mathbf{A}\mathbf{A}^{-\frac{1}{2}}} \frac{\left\langle (\mathbf{A}^{-\frac{1}{2}}\widetilde{\mathbf{W}}\mathbf{A}^{-\frac{1}{2}})(\mathbf{x}+\mathbf{y}),(\mathbf{x}+\mathbf{y})\right\rangle}{\langle \mathbf{x},\mathbf{x}\rangle} =$$

$$= \sup_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})} \ \inf_{\mathbf{y}\in\mathbf{ran}\ \boldsymbol{\pi}_\mathbf{A}} \frac{\left\langle \widetilde{\mathbf{W}}(\mathbf{x}+\mathbf{y}),(\mathbf{x}+\mathbf{y})\right\rangle}{\langle \mathbf{A}\mathbf{x},\mathbf{x}\rangle} =$$

$$= \sup_{\mathbf{x}} \ \inf_{\mathbf{y}\in\mathbf{ran}\ \boldsymbol{\pi}_\mathbf{A}} \frac{\left\langle \widetilde{\mathbf{W}}(\mathbf{x}+\mathbf{y}),(\mathbf{x}+\mathbf{y})\right\rangle}{\langle \mathbf{A}(\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x},(\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x}\rangle} =: K_1$$

Now, because $\|\mathbf{x}\|^2_\mathbf{A} = \|(\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x}\|^2_\mathbf{A} + \|\boldsymbol{\pi}_\mathbf{A}\mathbf{x}\|^2_\mathbf{A}$, it is obvious that

$$K_1 \le \sup_{\mathbf{x}} \ \inf_{\mathbf{y}\in\mathbf{ran}\ \boldsymbol{\pi}_\mathbf{A}} \frac{\left\langle \widetilde{\mathbf{W}}(\mathbf{x}+\mathbf{y}),(\mathbf{x}+\mathbf{y})\right\rangle}{\langle \mathbf{A}\mathbf{x},\mathbf{x}\rangle} =: K_2$$

On the other hand, by writing $\mathbf{y} \to \boldsymbol{\pi}_\mathbf{A}\mathbf{x}+\mathbf{y}$ in K_1 and then taking the sup over a bigger set we see that

$$K_1 = \sup_{\mathbf{x}} \ \inf_{\mathbf{y}\in\mathbf{ran}\ \boldsymbol{\pi}_\mathbf{A}} \frac{\left\langle \widetilde{\mathbf{W}}((\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x}+\mathbf{y}),((\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x}+\mathbf{y})\right\rangle}{\langle \mathbf{A}(\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x},(\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{x}\rangle} =$$

$$= \sup_{\mathbf{x}\in\mathbf{ran}\ (\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})} \ \inf_{\mathbf{y}\in\mathbf{ran}\ \boldsymbol{\pi}_\mathbf{A}} \frac{\left\langle \widetilde{\mathbf{W}}(\mathbf{x}+\mathbf{y}),(\mathbf{x}+\mathbf{y})\right\rangle}{\langle \mathbf{A}\mathbf{x},\mathbf{x}\rangle} \le K_2$$

Thus $K_1 = K_2$ and with K_{TG} as in (7.6), because $\mathbf{ran}\ \boldsymbol{\pi}_\mathbf{A} = \mathbf{ran}\ \mathbf{P}\mathbf{A}_c^{-1}\mathbf{P}^T\mathbf{A} \subseteq \mathbf{ran}\ \mathbf{P}$

$$K_1 = \sup_{\mathbf{x}} \ \inf_{\mathbf{x}_c\in\mathbb{R}^m} \frac{\|\mathbf{x}+\mathbf{P}\mathbf{x}_c\|^2_{\widetilde{\mathbf{W}}}}{\|\mathbf{x}\|^2_\mathbf{A}} \le K_{\text{TG}}$$

Therefore we have shown (7.7) and the upper bound in (7.5). Lastly, the lower bound follows from (7.4):

$$\langle \mathbf{A}\mathbf{x},\mathbf{x}\rangle \le \langle \mathbf{B}_{\text{TG}}\mathbf{x},\mathbf{x}\rangle \quad \Leftrightarrow \quad \left\langle \mathbf{B}_{\text{TG}}^{-1}\mathbf{A}\mathbf{x},\mathbf{x}\right\rangle \le \|\mathbf{x}\|^2 \quad \Leftrightarrow$$
$$0 \le \left\langle (\mathbf{I}-\mathbf{B}_{\text{TG}}^{-1}\mathbf{A})\mathbf{x},\mathbf{x}\right\rangle = \langle \mathbf{M}_{\text{TG}}\mathbf{x},\mathbf{x}\rangle = \langle (\mathbf{I}-\boldsymbol{\pi}_\mathbf{A})\mathbf{M}\mathbf{x},\mathbf{M}\mathbf{x}\rangle$$

As $\overline{\boldsymbol{\pi}}_\mathbf{A} = \mathbf{A}^{\frac{1}{2}}\boldsymbol{\pi}_\mathbf{A}\mathbf{A}^{-\frac{1}{2}}$ is an orthogonal projector, $\forall\mathbf{x}$ we can show this easily by

$$\langle \boldsymbol{\pi}_\mathbf{A}\mathbf{x},\mathbf{x}\rangle = \left\langle \overline{\boldsymbol{\pi}}_\mathbf{A}\mathbf{A}^{\frac{1}{2}}\mathbf{x},\mathbf{A}^{-\frac{1}{2}}\mathbf{x}\right\rangle \le \left\langle \mathbf{A}^{\frac{1}{2}}\mathbf{x},\mathbf{A}^{-\frac{1}{2}}\mathbf{x}\right\rangle = \langle \mathbf{x},\mathbf{x}\rangle$$

□

Corrolary 7.1.1. *Preconditioning by the Two-grid algebraic multigrid method yields spectral bounds*

$$\kappa(\mathbf{B}_{TG}^{-1}\mathbf{A}) \leq K_{TG}$$

If the prolongation $\mathbf{P}$ fulfills equation (7.6), it is also said to fulfill the *weak approximation property.* Unfortunately, while condition (7.6) is very compact, it is not very usable in practice. For example, in order to show convergence of a two-grid method using forward/backward GS smoothing, we would have to show this condition using the very ugly expression $\widetilde{\mathbf{W}} = (\mathbf{L}+\mathbf{D})\mathbf{D}^{-1}(\mathbf{U}+\mathbf{D})$ which does not have a very "nice" representation. However, in section 3.1, using theorem 3.3, we showed that GS is no substantial improvement over (properly dampened) jacobi and that $\widetilde{\mathbf{W}} \approx \mathbf{D}$. Using this fact, instead of (7.6), we can show the, much simpler, modified condition below.

Definition 7.1: Weak Approximation Property

$$\inf_{\mathbf{x}_c \in \mathbb{R}^m} \|\mathbf{x} - \mathbf{P}\mathbf{x}_c\|_{\mathbf{D}} \leq K_{TG}\|\mathbf{x}\|_{\mathbf{A}} \quad \forall \mathbf{x} \in \mathbb{R}^n \tag{7.8}$$

We have to pay the constants incurred when going from $\widetilde{\mathbf{W}}$ to $\mathbf{D}$ and back in the estimates. As all common smoothers fulfill $\widetilde{\mathbf{W}} \approx \mathbf{D}$, (they fulfill the smoothing property (5.2)), we will restrict ourselfs to this case.

7.2 Analysis of the Multi Grid Method

When applicable, geometric multigrid methods fulfill spectral bounds that are independent of the number of levels. Proofs for this rest on the fact that when the coarse spaces are nested finite element spaces defined on coarse grids, each $u \in V_0$ has a regular decomposition that is a set of $u_i \in V_i$ such that $u = \sum_i u_i$ and $\|u\|_{H^1}^2 \approx \sum_i \|u_i\|_{H^1}^2$ with constants independent of the number of levels. This is achieved by choosing $u_i \coloneqq (Q_i - Q_{i+1})u$ with well understood interpolation operators $Q_i : C(\Omega) \to V_i$.

Something similar can not be done for generic algebraic multigrid methods. The exact makeup of the coarse spaces is not known a priori and the interpolation operators are difficult to pin down and are also often much weaker than those in the geometric case. Generally speaking, AMG methods are always constructed such that the weak approximation property (7.8) holds on all levels, but this alone is much too weak for level-independent bounds. It is in fact very difficult to proof level-independence of algebraic multigrid methods and we will not touch upon this topic any more. The good news is that in praxis ALlebraic Multigrid behaves nicely.

In [9], the spectral bounds of a variant of algebraic multigrid known as smoothed aggregation are proven to be at worst $\mathcal{O}(L^3)$.

8 Algebraic MultiGrid with Alternative Strong Connections

In this chapter the AscAMG method will be described in detail. Section 8.1 will define a tentative prolongation operator, show an energy based criterium for the prolongation that implies two grid convergence and finally discuss a way to improve on this tentative prolongation by a smoothing step.

The parallelization and implementation of these ideas will be the topic of section 8.2, where, in section 8.2.4, we will also discuss distributed smoothers.

AscAMG is implemented in a *C++* library that acts as an extension to Netgen/NGSolve. It is integrated into it's NGSPy python-interface. It uses and expands on the MPI-parallelization of NGSolve. The parallel matrices and vectors provided by NGSolve, as discussed in chapter 4, are used extensively, however the parallel smoothers described in seciton 8.2.4 had to be implemented from scratch, as the only parallel smoother currently provided by NGSolve is Jacobi.

8.1 The Prolongation

Definition 8.1: "Piecewise" Prolongation

*Let n be the dimension of some fine space (*including *dirichlet DOFs) and let $D \subseteq \{0 \dots n-1\}$ be the set of dirichlet dofs. The* "piecewise" prolongation *matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n_c}$ defined by a partition $\mathcal{C} = \{C_i, i = 0 \dots n_c - 1\}$ such that*

$$C_i \cap C_j = \emptyset \quad \forall i \neq j$$

$$\bigcup_{i=0}^{n_c-1} C_i = \{0 \dots n-1\} \setminus D$$

is given as:

$$P_{ij} = \begin{cases} 1 & \textit{if } i \in C_j \\ 0 & \textit{else} \end{cases}$$

Note 8.1. This is the classical tentative prolongation operator that also plays a role in aggregation based AMG ([8]).

Note 8.2. Note that the $(C_i)_i$ are only a partition of the fine degree dofs **excluding** dirichlet dofs, this means that all dofs on the coarse level are "free".

Each subset C_i defines one coarse grid variable, and by making these sets large, one could achieve very rapidly decreasing number of degrees of freedom on coarse levels, and thus excellent operator complexity, at the cost of the rate of convergence.

In ascAMG, we only **pair up** dofs istead of using bigger sets. This will have far-reaching ramifications. In section 8.2, we will discuss how we can still have good operator complexity in spite of this fact.

Definition 8.2: AscAMG Piecewise Prolongation

*Let n be the dimension of the fine space (*including *dirichlet-dofs) and let $D \subseteq \{0 \dots n-1\}$ be the set of dirichlet dofs. The ascAMG piecewise prolongation matrix $\mathbf{P} \in \mathbb{R}^{n \times n_c}$ defined by a partition $\mathcal{C} = \{C_i : i = 0 \dots n_c - 1\} \cup \{D_c\}$ of the fine degrees of freedom such that*

$$\begin{aligned}
&(i) \quad C \cap \widetilde{C} = \emptyset \quad \forall C \neq \widetilde{C} \in \mathcal{C} \\
&(ii) \quad \bigcup_{C \in \mathcal{C}} C = \{0 \dots n-1\} \\
&(iii) \; D \subseteq D_c \\
&(iv) \; |C_i| \leq 2 \quad \forall i \in \{0 \dots n_c - 1\}
\end{aligned}$$

is given as:

$$P_{ij} = \begin{cases} 1 & \text{if } i \in C_j \\ 0 & \text{else} \end{cases} \tag{8.1}$$

Note 8.3. The difference between definitions 8.1 and 8.2 is that the set D_c in the latter is allowed to be a true superset of D. This allows us to exclude additional DOFs besides the dirichlet DOFs from the coarse level. Doing this poses an additional loss of information from one level to the next, however there are are two considerations that motivate us to allow this all the same. Firstly, as we have seen in section 3.1, in the case where the mass term dominates the stiffness term in the system matrix, the GS method is sufficient on its own, therefore there is no need to introduce any coarse spaces at all in those parts of the domain that feature a large l2-term. In fact, not introducing coarse spaces in those areas of the domain decreases the operator complexity of the multigrid cycle while hardly impacting the quality of the method. Secondly, and related to this, often times dirichlet conditions are not imposed in an essential but rather in a weak way by adding a large l2-term on those degrees of freedom where one wishes to impose dirichlet conditions. If we can automatically identify those DOFs where the l2-term dominates and not include these in the coarse spaces, we are able to deal with these situations as well.

Note 8.4. Note that $|C_i| \leq 2$, therefore, unless we exclude many DOFs from the coarse level by including tem in D_c, the dimension of the coarse space will not be mucch smaller than $\frac{n}{2}$. If many single DOFs are include on the coarse level, it might even be much larger.

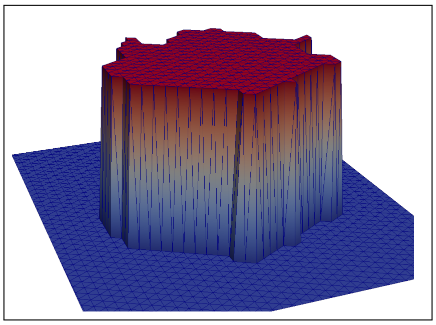


Figure 8.1: A base function on level 9 using piecwise prolongation.

This kind of piecewise prolongation leads to stepwise functions on coarse levels (figure 8.1). In the case of constant coefficients it is clear that interpolation with coarse level basis functions has much worse properties than interpolation with hat basis function coming from a coarse finite element space. The stepwise interpolation can give a good L^2-approximation, but because of the never diminishing steepness of the base functions they cannot give a good H^1-approximation. It is clear that the resulting multigrid algorithm does not feature level-independent bounds. In section 8.1.3, we will try to solve this problem.
The process of finding a good partition $\mathcal{C}$ is known as **"coarsening"**. We will elaborate upon our coarsening algorithm in more detail in section 8.2.1. In general, the coarsening algorithm has to construct the partition in such a way that the resulting prolongation matrix fulfills the weak approximation property (7.8). In our specific case, when we include the set $\{i, j\}$ in the partition, we effectively connect the dofs i and j, of the space **span** $\{\varphi_i, \varphi_j\}$ only **span** $\{\varphi_i + \varphi_j\}$ is retained on the coarse level, while **span** $\{\varphi_i - \varphi_j\}$ is discarded. Thus it obviously only makes sense to connect i and j if they share an "algebraic edge" (that is, an edge in the matrix graph of $\mathbf{A}$, or, as we will cal it, the "algebraic mesh"). The set of edges of the algebraic mesh will for the remainder of this section be called $\mathcal{E}(\hat{\mathbf{A}})$, and will be formalized and extended in section 8.2. The concept of an algebraic edge will be formalized and expanded in section 8.2.1. We call this process "collapsing" of the connecting edge e_{ij}, as the coarse algebraic mesh emerges from the fine one after merging the vertices on either end of any edge e_{kl} with $\{k, l\} \in \mathcal{C}$.

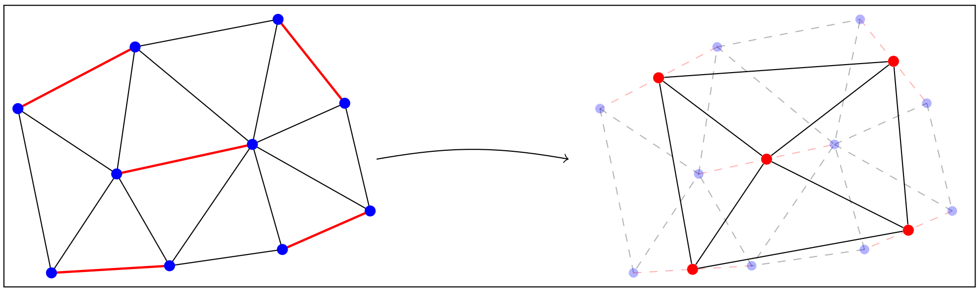


Figure 8.2: Red edges are collapsed. Coarse DOFs are "located" at the midpoints of the collapsed edges.

We know from chapter 5 that algebraically smooth error varies slowly inside parts of the domain with a comparatively large coefficient α and can vary more strongly in domains with comparatively small α, especially in areas where two "strong" domains are connected by a "weak" domain. Therefore we want to allow collapsing of an edge within areas of relatively constant α and we absolutely do not want to collapse edges that connect two strong domains but that themselfs "lie" in a weak region. We must do all of this on the basis of the matrix $\mathbf{A}$ itself, without knowledge of the actual coefficients or the mesh underneath. The criterium for an edge's eligibility for collapsing is based on the construction of a replacement matrix that is spectrally equivalent to $\mathbf{A}$ but has an even simpler structure.

8.1.1 Replacement Matrix

We will now show that we can find a replacement matrix $\hat{\mathbf{A}}$ for $\mathbf{A}$ *on the finest level.*

Theorem 8.1: Replacement Matrix

Let V_h of dimension n be the lowest order H^1-Finite Element space.

$$\alpha_{ij} := \begin{cases} \sum_{T\in\mathcal{T}_h;\ e_{ij}\in T} \textbf{\textit{tr}}\ S_{T,ij} & , e_{ij}\in\mathcal{E}(\mathcal{T}) \\ 0 & , else \end{cases} \quad i,j = 0\dots n-1$$

Here, $S_{T,ij}$ is the schur complement of the element matrix for the element T with respect to the dofs i and j.
Then, the replacement stiffness matrix matrix $\hat{\mathbf{K}}\in\mathbb{R}^{n\times n}$ defined by

$$\hat{\mathbf{K}}_{ij} = \begin{cases} \sum_{l=0}^{n-1}\alpha_{il} & i = j \\ -\alpha_{ij} & i\neq j \end{cases}$$

and the original stiffness matrix $\boldsymbol{K}$ induce equivalent norms.

$$\|\mathbf{u}\|^2_{\mathbf{K}} \approx \|\mathbf{u}\|^2_{\hat{\mathbf{K}}} = \sum_{i,j=0}^{n-1}\alpha_{ij}(\mathbf{u}_i - \mathbf{u}_j)^2 \tag{8.2}$$

Define $\beta_j := \mathbf{D}_{M,j}$, where $\mathbf{D}_M$ is the diagonal of the mass matrix. Then $\mathbf{D}_M \approx M$. A replacement matrix for $\boldsymbol{A}$ is given by $\hat{\mathbf{A}} := \hat{\mathbf{K}} + \mathbf{D}_M$, or, in other words:

$$\|\mathbf{u}\|^2_{\mathbf{A}} \approx \|\mathbf{u}\|^2_{\hat{\mathbf{A}}} = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\alpha_{ij}(\mathbf{u}_i - \mathbf{u}_j)^2 + \sum_{i=0}^{n-1}\beta_i\mathbf{u}_i^2 \tag{8.3}$$

All constants in these equivalences only depend on the spatial dimension and the shape regularity of the triangulation (and are, in particular, h-independent).

Proof. We have to show, with $|u_h|_{L^2_\beta(\Omega)} \coloneqq \int_\Omega \beta|u_h|^2 dx$, that

$$\left|\sum_{i=0}^{n-1}\mathbf{u}_i\varphi_i\right|^2_{L^2_\beta(\Omega)} \approx \sum_{i=0}^{n-1}|\mathbf{u}_i\varphi_i|^2_{L^2_\beta(\Omega)}$$

Let $T\in\mathcal{T}$ be any element in $\mathcal{T}$, $\hat{T}$ be the reference element, $\hat{\varphi}_i = \varphi_i\circ\phi$ be the hat basis functions on the reference element with the usual transformation $\phi : \hat{T}\to T$ and $F \coloneqq \phi'$. We will show equivalence in the L^2_β norms restricted to $\mathcal{T}$. Summation over elements then gives us the result. For $u_h\in V_h$ and and $T\in\mathcal{T}$

$$\left|\sum_{i=0}^{n-1}\mathbf{u}_i\varphi_i\right|^2_{L^2_\beta(T)} = \int_{\hat{T}}\hat{\beta}|\mathbf{det}\ F|\left|\sum_{i:v_i\in T}\mathbf{u}_i\hat{\varphi}_i\right|^2 dx$$

and

$$\sum_{i=0}^{n-1} |\mathbf{u}_i\varphi_i|^2_{L^2_\beta(T)} = \sum_{i:v_i\in T} \int_{\hat{T}} \hat{\beta}|\mathbf{det}\ F|\,|\mathbf{u}_i\hat{\varphi}_i|^2\,dx$$

These are both norms on $\mathbb{R}^3$ and thus equivalent.

We will now show the eqivalence of the stiffness and replacement-stiffness matrices in two dimensions. The proof works in the same in three dimensions, but it is a bit more compact in two.
First we will show that for all $T\in\mathcal{T}_h$ and $u_h\in V_h$

$$|u_{h|T}|^2_{H^1_\alpha(T)} \coloneqq \int_T \alpha|\nabla u_h|^2 dx \approx \sum_{e_{ij}\in T} (\mathbf{u}_i-\mathbf{u}_j)^2|\mathcal{H}^{ij}(\varphi_i-\varphi_j)|_{H^1_\alpha(T)} \tag{8.4}$$

Here, $\mathcal{H}^{ij}$ is the harmonic extension from the edge e_{ij} to the entire element T, defined by

$$\mathcal{H}^{ij}(\mathbf{u}_i\varphi_i+\mathbf{u}_j\varphi_j) = \operatorname*{arg\,min}_{\lambda\in\mathbb{R}}\left\{|\mathbf{u}_i\varphi_i+\mathbf{u}_j\varphi_j+\lambda\varphi_k|_{H^1_\alpha}(T)\right\}$$

where k is the index of the DOF at the third vertex of T. On the one hand, we have

$$\mathbf{u}_{h|T} = \frac{1}{2}(\mathbf{u}_i-\mathbf{u}_j)(\varphi_i-\varphi_j) + \left(\mathbf{u}_k - \frac{1}{2}(\mathbf{u}_i+\mathbf{u}_j)\right)\varphi_k$$

and thus

$$|u_{h|T}|^2_{H^1_\alpha(T)} \geq (\mathbf{u}_i-\mathbf{u}_j)^2|\mathcal{H}^{ij}(\varphi_i-\varphi_j)|_{H^1_\alpha(T)}$$

On the other hand, with $\bar{\mathbf{u}} \coloneqq \frac{1}{3}(\mathbf{u}_i+\mathbf{u}_j+\mathbf{u}_k)$ we have

$$u_{h,|T} = \bar{\mathbf{u}} + \frac{1}{3}(\mathbf{u}_i-\mathbf{u}_j)(\varphi_i-\varphi_j) + \frac{1}{3}(\mathbf{u}_j-\mathbf{u}_k)(\varphi_j-\varphi_k) + \frac{1}{3}(\mathbf{u}_k-\mathbf{u}_i)\,|\,(\varphi_k-\varphi_i)$$

and thus

$$|u_h|^2_{H^1_\alpha(T)} = \left|\sum_{e_{ij}\in T}(\mathbf{u}_i-\mathbf{u}_j)^2(\varphi_i-\varphi_j)\right|^2_{H^1_\alpha(T)} \lesssim \sum_{e_{ij}\in T}(\mathbf{u}_i-\mathbf{u}_j)^2\,|\,(\varphi_i-\varphi_j)\,|^2_{H^1_\alpha(T)}$$

We need to show

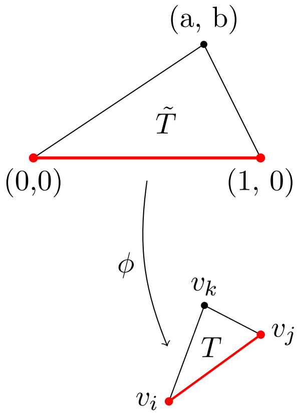
$$|\varphi_i-\varphi_j|^2_{H^1_\alpha(T)} \lesssim |\mathcal{H}(\varphi_i-\varphi_j)\,|^2_{H^1_\alpha(T)}$$

We would like to show the other bound by transforming to the reference element $\hat{T}$, however this turns out not to work easily. Let again $\hat{T}$ be the reference element, $\hat{\varphi}_i = \varphi_i\circ\phi$ be the hat basis functions on the reference element with the usual transformation $\phi:\hat{T}\to T$ and $F\coloneqq\phi'$. transformation, with $F\coloneqq\phi'$. With $\varphi(\phi(x))=\hat{\varphi}(x)$, the chain rule gives $\nabla\varphi = F^T\hat{\nabla}\hat{\varphi}$ and transformation of one term from the right hand side of (8.4) to the reference element gives:

$$\begin{aligned}|\mathcal{H}(\varphi_i-\varphi_j)\,|^2_{H^1_\alpha(T)} &= \inf_{\lambda\in\mathbb{R}}\int_T \alpha|\nabla(\varphi_i-\varphi_j+\lambda\varphi_k)|^2dx\\ &= \inf_{\lambda\in\mathbb{R}}\int_{\hat{T}}\alpha|\mathbf{det}\ F|\left\langle F^T\hat{\nabla}(\hat{\varphi}_i-\hat{\varphi}_j+\lambda\hat{\varphi}_k), F^T\hat{\nabla}(\hat{\varphi}_i-\hat{\varphi}_j+\lambda\hat{\varphi}_k)\right\rangle dx\end{aligned}$$

We see that taking the infimum and doing the transformation only commute if $FF^T = \gamma I$ for some constant $\gamma \in \mathbb{R}$, or in other words: if the element transformation is a (scaled) euclidian motion.

For this reason, instead of transforming back to the reference element $\hat{T}$, we transform to some "stretched reference element" $\tilde{T} \coloneqq \mathbf{cov}\left\{(0,0)^T, (1,0)^T, (a,b)^T\right\}$ (with $a, b > 0$). The transformation ϕ can now be chosen to take the form $\phi(x) = h_e Qx + c$ with an orthogonal matrix Q and some vector $c \in \mathbb{R}^2$ and the length of the edge e_{ij}, h_e. The transformation maps e_{ij} to the edge $\mathbf{cov}\left\{(0,0)^T, (1,0)^T\right\}$. In a shape regular triangulatuon we can assume a and b to be bounded away from 0 and bounded from above by moderately sized constants.



The base functions on $\tilde{T}$ are:

$$\begin{aligned}
\varphi_1 &= 1 - x + \frac{a-1}{b}y & \nabla\varphi_1 &= \left(-1, \frac{a-1}{b}\right)\\
\varphi_2 &= x - \frac{a}{b}y & \nabla\varphi_2 &= \left(1, -\frac{a}{b}\right)\\
\varphi_3 &= \frac{1}{b}y & \nabla\varphi_3 &= \left(0, \frac{1}{b}\right)
\end{aligned}$$

Now, the transformation and the infimum commute and we can compute

$$\begin{aligned}
|\mathcal{H}\left(\varphi_i - \varphi_j\right)|^2_{H^1(T)} &= \inf_\lambda \int_{\tilde{T}} h_e |\nabla\left(\varphi_1 - \varphi_2 + \lambda\varphi_3\right)|^2 dx\\
&= \inf_\lambda \int_{\tilde{T}} h_e \left|\left(-2, \frac{2a-1+\lambda}{b}\right)^T\right|^2\\
&= |\tilde{T}| h_e \inf_\lambda 4 + \left(\frac{2a-1+\lambda}{b}\right)^2\\
&= 4|\tilde{T}|h_e\\
|\left(\varphi_i - \varphi_j\right)|^2_{H^1(T)} &= |\tilde{T}|h_e 4 + \left(\frac{2a-1}{b}\right)^2
\end{aligned}$$

The shape regularity of the triangulation $\mathcal{T}_h$, guarantees that the term $\frac{2a-1}{b}$ is bouned from above and below. The second inequality for (8.4) now follows from applying this to each edge (in particular, transforming three times, with each edge mapped to the $(0,0) - (1,0)$ edge once). Now (8.2) follows from summing over all elements and the simple argument that by definition of the schur complement and the harmonic extension $\mathcal{H}$

$$|\mathcal{H}(\phi_i - \phi_j)|^2_{H^1_\alpha(T)} = |(1,-1)^T|^2_{S_{T,ij}} = \mathbf{tr}\ S_{T,ij}$$

□

Note 8.5. The replacement-matrix from theorem 8.1 is an $M-$matrix.

Note 8.6. The α_{ij} and the β_i can be computed on the element matrix level.

Note 8.7. Given a prolongation operator $\mathbf{P} \in \mathbb{R}^{n \times n_C}$, we can define the coarse replacement matrix $\hat{\mathbf{A}}_C \coloneqq \mathbf{P}^T \hat{\mathbf{A}} \mathbf{P}$ which is equivalent to the coarse matrix $\mathbf{A}_C = \mathbf{P}^T \mathbf{A} \mathbf{P}$ with the same bounds as $\mathbf{A}$ and $\hat{\mathbf{A}}$.

$$\hat{\mathbf{A}}_C \approx \mathbf{A}_C$$

If we have no L^2-term in the original equation all $\beta_j = 0$ and the equation $\hat{\mathbf{A}}\mathbf{u} = \mathbf{b}$ can interpreted as describing a resistor network where each vertex v_i is a node, each $e_{ij} \in \mathcal{E}(\hat{\mathbf{A}})$ represents a a conductor of resistance $1/\alpha_{ij}$ that connects v_i and v_j, $\mathbf{u}_i$ is the value of the electric potential u at v_i and $\mathbf{b}_i$ gives the (constant) electric current flowing into/out of the network at v_i. Assuming our dirichlet-data is $u_D = 0$, $\mathbf{u}_i = 0$ for all nodes $v_i \in \partial\Omega$; this just means that these nodes are grounded. As any other case with $u_D \neq 0$ can, by taking any function w that fulfills the dirichlet conditions and considerin the equation for $\tilde{u} \coloneqq u - w$ which has 0 boundary values by construction, be reduced to one where $u_D = 0$, it is the only case we need to concern ourself with. Each line of the equation corresponding to a non-dirichlet DOF i is then just Kirchhoff's first law applied at v_i:

$$\sum_j \alpha_{ij}(\mathbf{u}_i - \mathbf{u}_j) = \mathbf{b}_i$$

Given a piecewise prolongation as in definition 8.2, we can find a similar interpretation for $\hat{\mathbf{A}}_C$. Let $i \neq j$, then, with the unit vectors $\mathbf{e}_i \in \mathbb{R}^{n_c}$ we have

$$\hat{\mathbf{A}}_{C,ij} = \left\langle \hat{\mathbf{A}}\mathbf{P}\mathbf{e}_i, \mathbf{P}\mathbf{e}_j \right\rangle = -\sum_{k \in C_i} \sum_{l \in C_j} \alpha_{kl} \eqqcolon \alpha_{ij}^C$$

The off-diagonal entries of $\hat{\mathbf{A}}_{C,ij}$ are the sums of all off-diagonal entries of $\hat{\mathbf{A}}$ corresponding to edges that connect a vertex in C_i to one in C_j. This sum corresponds to the inverse of the collective resistance of the resistors along all of those edges connected in parallel. Unlike at the fine level, the diagonal entries of the coarse replacement matrix , in addition to the (negative) sum of the odd-diagonal entries of the row, also have additional values corresponding to edges that connect any node in C_i to any node in D (if there are any).

$$\hat{\mathbf{A}}_{C,ii} = \sum_j \hat{\mathbf{A}}_{C,ij} + \sum_{k \in C_i} \sum_{j \in D_C} \alpha_{kj} \eqqcolon \sum_j \alpha_{ij}^C + \beta_i^C$$

We see that the coarse replacement matrix has the form of a replacement matrix in the sense of theorem 8.1, however with an l2-term and induces the quadratic form

$$\|\mathbf{u}\|_{\hat{\mathbf{A}}_C}^2 = \sum_{i,j} \alpha_{ij}^C (\mathbf{u}_i - \mathbf{u}_j)^2 + \sum_i \beta_i^C \mathbf{u}_i^2$$

Let us therefore go back to the fine level and interpret the problem for a nonzero β. We can write the induced quadratic form 8.3 as

$$\|\mathbf{u}\|_{\hat{\mathbf{A}}}^2 = \sum_{i,j} (\mathbf{u}_i - \mathbf{u}_j)^2 \alpha_{ij} + \sum_i \mathbf{u}_i^2 \beta_i = \sum_{i,j} (\mathbf{u}_i - \mathbf{u}_j)^2 \alpha_{ij} + \sum_i (\mathbf{u}_i - 0)^2 \beta_i$$

With $\mathbf{b} \coloneqq (\beta_0, \beta_1, \dots \beta_{n_c - 1})^T$ and $b \coloneqq \sum_i \beta_i$, we can now define an extension $\widetilde{\mathbf{A}}$ of $\hat{\mathbf{A}}$

by

$$\widetilde{\mathbf{A}} = \begin{pmatrix} b & -\mathbf{b}^T \\ -\mathbf{b} & \hat{\mathbf{A}} \end{pmatrix} \in \mathbb{R}^{n+1\times n+1}$$

$\widetilde{\mathbf{A}}$ now has the form of a replacement matrix without l2-term, for which we already have an interpretation, the β_i now stand for additional connections between v_i and the newly intriduced node v_{-1} (we will use indices starting with -1 for the extended replacement matrix). The equation $\hat{\mathbf{A}}\mathbf{u} = \mathbf{b}$ is equivalent to

$$\widetilde{\mathbf{A}} \begin{pmatrix} 0 \\ \mathbf{u} \end{pmatrix} = \begin{pmatrix} * \\ \mathbf{b} \end{pmatrix}$$

Solving $\hat{\mathbf{A}}\mathbf{u} = \mathbf{b}$ is equivalent to solving an equation with $\widetilde{\mathbf{A}}$ where we prescribe 0 as a dirichlet condition for v_{-1}. The β_i act as grounding conductors. The connections between vertices on the dirichlet-boundary and vertices not on the dirichlet boundary act in the same way. We can extend $\hat{\mathbf{A}}_C$ in the same way to get an extended coarse replacement matrix $\widetilde{\mathbf{A}}_C \in \mathbb{R}^{n_c+1\times n_c+1}$. After extending the prolongation matrix $\mathbf{P} \in \mathbb{R}^{n\times n_c}$ to $\widetilde{\mathbf{P}} \in \mathbb{R}^{n+1\times n_c+1}$ by defining

$$\widetilde{\mathbf{P}}_{ij} = \begin{cases} \mathbf{P}_{ij} & \text{if } i,j \geq 0 \\ 1 & \text{if } j = -1 \text{ and } i \in D_{\mathcal{C}} \cup \{-1\} \\ 0 & \text{else} \end{cases}$$

we see that $\widetilde{\mathbf{A}}_C = \widetilde{\mathbf{P}}^T \widetilde{\mathbf{A}} \widetilde{\mathbf{P}}$. All fine dirichlet-DOFs, the (fine) fictitious DOF -1 as well as all other dofs in $D_{\mathcal{C}}$ have been mapped to the coarse dirichlet dof -1. All in all, as mentioned above, for the interpretation of the coarse matrix this means that coarse dofs i and j are connected by a resistor that corresponds to all the resistors connecting a dof in C_i with one in C_j connected in parallel. The resistances along connections between dofs i and j with $\{i,j\} =: C_k \in \mathcal{C}$, that is α_{mn} for $m,n \in C_k$ do not factor into any entry of $\widetilde{\mathbf{A}}_C$ - in the coarse network these connections play the role of a perfect conductor. The connections between some DOF in $D_{\mathcal{C}}$ and some DOF not in $D_{\mathcal{C}}$ become grounding conductors on the coarse level. Having some $i \in D_{\mathcal{C}} \setminus D$ is therefore equivalent to collapsing the edge connecting nodes i and -1, for this reason we call putting some free DOF i into $D_{\mathcal{C}}$ “collapsing the vertex v_i”, in analogy to “collapsing an edge e_{ij}” when $\{i,j\} \in \mathcal{C} \setminus \{D_{\mathcal{C}}\}$. An edge e_{ij} with $\{i,j\} \in \mathcal{C} \setminus \{D_{\mathcal{C}}\}$ will be called a collapsed edge, and a vertex v_i with $i \in D_{\mathcal{C}}$ will be called a collapsed vertex.

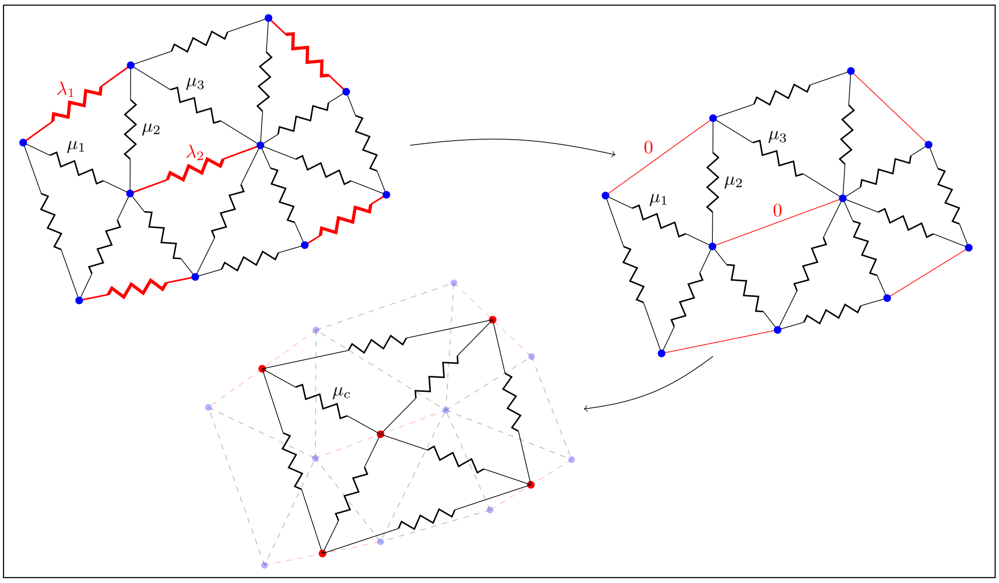


Figure 8.3: The coarse resistor network for the piecewise prolongation. Resistors along collapsed edges are replaced by perfect conductors, the resistances λ_i along these connections are 0. Those along the other labelled edges are put in parallel, $\mu_c = (\sum_i \mu_i^{-1})^{-1}$.

These ideas will be used in section 8.1.3 as motivation for an improvement to the piecewise prolongation operator.

8.1.2 Two Grid Convergence

We will now show how to construct $\mathcal{C}$ such that the resulting piecewise prolongation operator $\mathbf{P}$ fulfills the weak approximation property (7.8) and thus establish two-grid convergence.

Theorem 8.2: Weak Approximation Property for AscAMG

Let $\mathbf{P}$ be a piecewise prolongation and $\mathcal{C}$ be the partition as in definition 8.2. Define the vertex strengths $s_i := \hat{A}_{ii}$, the edge-collapse-weights $w_{ij} := \frac{\alpha_{ij}}{s_i+s_j}$ and the vertex-collapse-weights $w_i := \frac{\beta_i}{s_i}$.
If, for some $1 > \sigma > 0$

$$w_{ij} > \sigma \qquad \forall \{i,j\} \in \mathcal{C} \tag{8.5}$$

$$w_i > \sigma \qquad \forall i \in D_c \setminus D \tag{8.6}$$

Then

$$\inf_{\mathbf{x}_c \in \mathbb{R}^m} \|\mathbf{x} - \mathbf{P}\mathbf{x}_c\|^2_{\hat{\mathbf{D}}} \leq \sigma^{-1} \|\mathbf{x}\|^2_{\hat{\mathbf{A}}} \quad \forall \mathbf{x} \in \mathbb{R}^n \tag{8.7}$$

That is, $\mathbf{P}$ fulfulls the coarse approximation property 7.8 with resprect to $\hat{\mathbf{A}}$ and its diagonal $\widehat{\mathbf{D}}$ and therefore for $\mathbf{A}$ and its diagonal $\mathbf{D}$.

Proof. For any $\mathbf{x} \in \mathbb{R}^n$ we define the coarse grid interpolant $\mathbf{x}_C$ by taking the average for each collapsed edge and keeping $\mathbf{u}_i$ the same for all i such that $\{i\} \in \mathcal{C}$.

$$\mathbf{x}_{C,i} = \begin{cases} \frac{\mathbf{u}_k + \mathbf{u}_l}{2} & \text{if } C_i = \{k, j\} \\ \mathbf{u}_k & \text{if } C_i = \{k\} \end{cases}$$

With this we have

$$\mathbf{P}\mathbf{x}_C = \sum_{\{i,j\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \frac{1}{2}(\mathbf{u}_i + \mathbf{u}_j)(\mathbf{e}_i + \mathbf{e}_j) + \sum_{\{i\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \mathbf{u}_i \mathbf{e}_i$$

And

$$\begin{aligned} \mathbf{x} - \mathbf{P}\mathbf{x}_C &= \sum_{\{i,j\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \mathbf{u}_i \mathbf{e}_i + \mathbf{u}_j \mathbf{e}_j - \frac{1}{2}(\mathbf{u}_i + \mathbf{u}_j)(\mathbf{e}_i + \mathbf{e}_j) + \sum_{i \in D_\mathcal{C} \setminus D} \mathbf{u}_i \mathbf{e}_i = \\ &= \sum_{\{i,j\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \frac{1}{2}(\mathbf{u}_i - \mathbf{u}_j)\mathbf{e}_i + \frac{1}{2}(\mathbf{u}_j - \mathbf{u}_i)\mathbf{e}_j + \sum_{i \in D_\mathcal{C} \setminus D} \mathbf{u}_i \mathbf{e}_i \end{aligned}$$

Thus

$$\begin{aligned} \|\mathbf{x} - \mathbf{P}\mathbf{x}_C\|^2_{\hat{\mathbf{D}}} &= \sum_{\{i,j\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \frac{1}{4}(\mathbf{u}_i - \mathbf{u}_j)^2 \hat{\mathbf{A}}_{ii} + \frac{1}{4}(\mathbf{u}_j - \mathbf{u}_i)^2 \hat{\mathbf{A}}_{jj} + \sum_{i \in D_\mathcal{C} \setminus D} \mathbf{u}_i^2 \hat{\mathbf{A}}_{ii} = \\ &= \sum_{\{i,j\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \frac{1}{4}(\mathbf{u}_i - \mathbf{u}_j)^2 \left(\hat{\mathbf{A}}_{ii} + \hat{\mathbf{A}}_{jj}\right) + \frac{1}{\sigma} \sum_{i \in D_\mathcal{C} \setminus D} \mathbf{u}_i^2 \beta_i \leq \\ &\leq \frac{1}{\sigma} \sum_{\{i,j\} \in \mathcal{C} \setminus \{D_\mathcal{C}\}} \frac{1}{4}(\mathbf{u}_i - \mathbf{u}_j)^2 \alpha_{ij} + \frac{1}{\sigma} \sum_i \mathbf{u}_i^2 \beta_i \leq \\ &\leq \frac{1}{\sigma} \sum_{i,j} \frac{1}{4}(\mathbf{u}_i - \mathbf{u}_j)^2 \alpha_{ij} + \frac{1}{\sigma} \sum_i \mathbf{u}_i^2 \beta_i \leq \\ &\leq \frac{1}{\sigma} \|\mathbf{x}\|^2_{\hat{\mathbf{A}}} \end{aligned}$$

Therefore $\mathbf{P}$ fulfills the weak approximation property for $\hat{\mathbf{A}}$ and its diagonal $\widehat{\mathbf{D}}$ and because $\mathbf{A} \approx \hat{\mathbf{A}}$ also for $\mathbf{D} \approx \widehat{\mathbf{D}}$. □

8.1.3 Smoothed Prolongation

As mentioned before, because the "steepness" of coarse base function never diminishes, coarse spaces constructed by peicewise prolongation feature poor H^1-approximation properties. For the method this means that it's quality gets worse as the number of levels increases. The term "smoothed prolongation" refers to a modification of the piecewise prolongation that is supposed to smooth out the coarse base functions to deal with this problem. In principle, given the error prolongation matrix $\mathbf{M}$ of our smoother of choice, we want the coarse space be able to approximate all algebraically smooth vectors, that is vectors such that $\mathbf{M}\mathbf{x} \approx \mathbf{x}$ or at least not to contain nonsmooth vectors, which are those with $\mathbf{M}\mathbf{x} \approx 0$. We could now define the smoothed prolongation as $\mathbf{P}_s \coloneqq \mathbf{M}\mathbf{P}$. The extra smoothing step is supposed remove nonsmooth components in the range of $\mathbf{P}$. The problem with this approach is that when using this smoothed prolongation to build the coarse matrix $\mathbf{A}_C = \mathbf{P}_s^T \mathbf{A} \mathbf{P}_s$, we potentially introduce a lot of nonzero entries which we would not have if we used the simple $\mathbf{P}$. In many cases, $\mathbf{M}$ is not even sparse, for example $\mathbf{M} = \mathbf{I} - (\mathbf{L} + \mathbf{D})^{-1}\mathbf{A}$ for Gauss-Seidel. Even if we wanted to compute $\mathbf{M}$ in that case, which is unfeasible in and of itself for large problems, the number of nonzeros in $\mathbf{A}_C$, which determines the operator complexity of the multigrid algorithm, would be unacceptably large and even if we would be willing to live with that, this problem would get worse and worse on coarser levels and after a few levels we would have a dense system matrix. We already observed in chapter 3 that jacobi and Gauss-Seidel actually behave similarly, therefore the usual approach is to use the iteration matrix of dampened jacobi instead of $\mathbf{M}$ to smooth the prolongation and define $\mathbf{P}_s \coloneqq (\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{A})\mathbf{P}$. This decreases the problem we face somewhat:

$$\mathbf{A}_C = \mathbf{P}^T \underbrace{(\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{A})^T\mathbf{A}(\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{A})}_{\coloneqq\mathbf{A}_2}\mathbf{P}$$

$\mathbf{A}_C$ is now the coarse matrix of $\mathbf{A}_2$, built with the original piecewise prolongation. $\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{A}$ has the same sparsity pattern as $\mathbf{A}$, therefore the matrix graph of $\mathbf{A}_2$ has an edge for each path of length 3 in the matrix graph of $\mathbf{A}$. In traditional smoothed aggregation methods, the aggregates $C \in \mathcal{C}$ are made so large that the corresponding piecewise prolongation cancels out many of these additional connections.

In our case, we have $|C| \leq 2 \; \forall C \in \mathcal{C} \setminus \{D_c\}$ therefore the piecewise prolongation can not eliminate many of the additional connections and the matrix graph of $\mathbf{A}_C$ is again a much denser matrix that $\mathbf{A}$, however, it is still much sparser than it would be if we had used $\mathbf{M}_{\text{GS}}$. Over many levels, however, these additional entries still compound and pose an increasingly big problem. What follows is a modification of the smoothed prolongation based on the replacement matrix $\hat{\mathbf{A}}$.

8.1.4 A better coarse system

We have seen how to interpret the coarse replacement matrix, constructed from a piecewise prolongation operator $\mathbf{P}$, via a coarse resistor network which corresponds for the fine one, with resistors along each collapsed edge replaced by perfect conductors and resistors along edges connecting the same agglomerates connected in parallel (figure 8.3). We have also seen how to extend the replacement matrix in order to reduce the inerpretation for the case where some $\beta_i \neq 0$ to one where all $\beta_i = 0$, which is therefore the only case we need to consider in this section.

Notation 8.1. For the next section, it will be convenient to be able to map fine DOFs to coarse ones. A prolongation matrix $\mathbf{P}$ defines a fine-to-coarse index map

$$c : \begin{cases} \{0 \ldots n-1\} \mapsto \{0 \ldots n_c - 1\} \\ i \to j \quad \text{such that } i \in C_j \end{cases}$$

which we will use for the remainder of this section only. It will be generalized later.

Note 8.8. The corresponding coarse-to-fine index map, although mapping indices to sets, is just $i \to C_i$.

This coarse netowork is simply not a very good approximation to the fine one, as all resistors along collapsed edges are simply removed. We can find a better fitting coarse network that features the exact same nodes and connections as the original coarse one but has different resistances. First, we replace each resistor in the fine grid that runs along a collapsed edge by two, weaker, resistors, connected in serial, each with halve of the original's resistance, as seen in figure 8.4 on the right. Let us call this simply the extended network (which is a different extended network than the one before, where we interpreted the l2-terms!). Then, for the coarse network, we call the node on the edge e_{ij} with $\{i, j\} = C_k \in \mathcal{C} \setminus \{D_c\}$ node k, remove nodes in vertices of collapsed edges and call the node on vertex v_i with $\{i\} = C_j \in \mathcal{C} \setminus \{D_c\}$ node j. As before, we establish a connection between nodes i and j in the coarse network exactly if there is some edge $e_{kl} \in \mathcal{E}(\hat{\mathbf{A}})$ with $c(k) = i$ and $c(l) = j$. This leaves us with the exact same nodes and connections between nodes we had in the old coarse network constructed from the piecewise prolongation matrix. We want the resistances on the coarse network to be chosen such that the extended nework is approximated as well as possible by the coarse one. We are not interested in how exactly the optimal choice looks, however it is obvious that it somehow has to incorporate resistances along the split collapsed edges, in contrast to before, where we simply replaced these with perfect conductors.

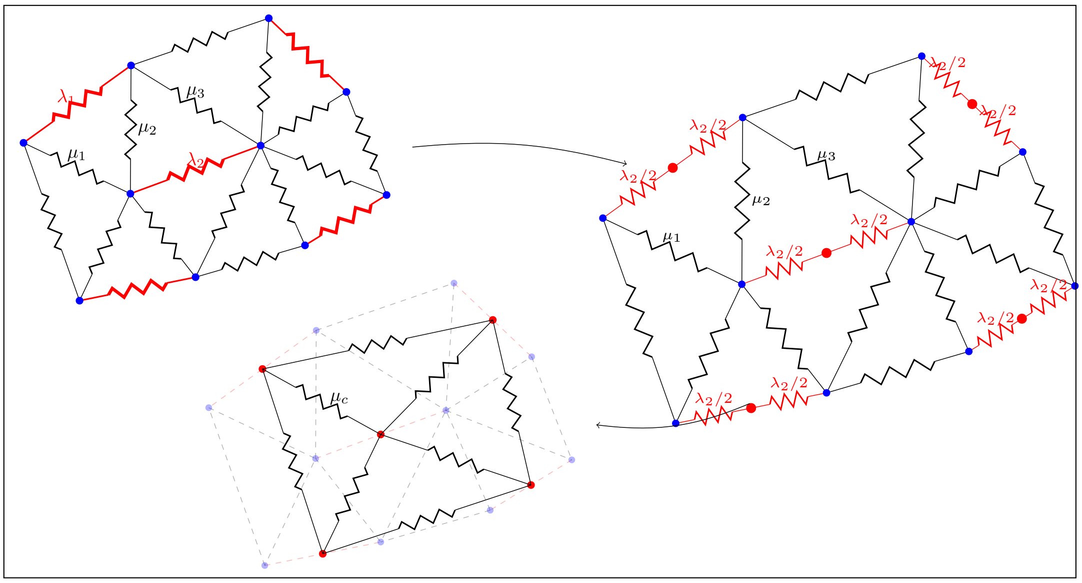
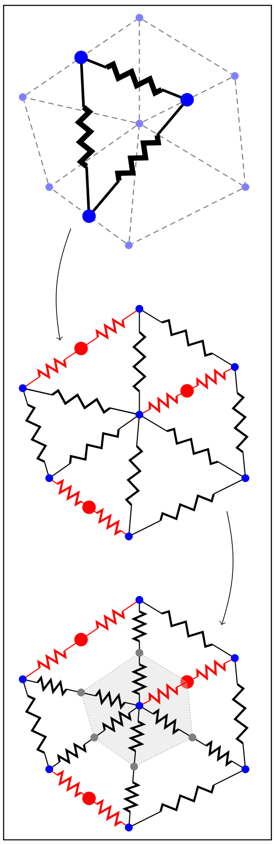


Figure 8.4: A coarse resistor netwerk that better approximates the original one. Resistors along collapsed edges are split into two, each of them connecting a newly introduced node at the midpoint of the edge with a vertex, each accounting for half the resistance of the original one. One possible, not necessarily ideal, choice for the coarse resistance would be to first connect resistors along edges that are not collapsed in parallel where appropriate and then connect them in serial with the newly introduced ones for $\mu_c = (2\lambda_1^{-1} + 2\lambda_2^{-1} + \sum_i \mu_i^{-1})^{-1}$.

What we are interested in is to build a good prolongation matrix $\mathbf{P}_s$ that takes a vector $\mathbf{u}^c$ of nodal values of an electric potential u on the coarse network and gives us a good approximation of its nodal values on the fine network. That is, given $\mathbf{u}^c$ as dirichlet data in the coarse nodes of the extended network, we want to find the nodal values of u in the rest of the extended network (of course, with no current exiting or entering the extended network in nodes not present in the coarse network).

These nodal values restricted to the nodes present in the fine network are then $\mathbf{P}_s\mathbf{u}^c$.



Finding the exact nodal values would require solving a laplacian-like equation on the entire extended network and is not feasible. We will therefore, as we did with the collapsed edges before, now also introduce new nodes on edges e_{ij} that are not collapsed, split their resistors in two and then prescribe dirichlet-values $\frac{1}{2}(u_{c(i)}+u_{c(j)})$ there - this is not exact, but these nodes should lie somewhere in the middle between the nodes $c(i)$ and $c(j)$ and there is no current entering the network in between so it should be a reasonable approximation. Now, with the neighbor set $N_i=\{j : e_{ij}\in\mathcal{E}(\hat{\mathbf{A}})\}$, Kirchhoff's law for each node i with $\{i,j\}\in\mathcal{C}\setminus\{D_c\}$ is

$$2\alpha_{ij}(\mathbf{u}_i-\mathbf{u}^c_{c(i)})+\sum_{\substack{l\in N_i\\ c(l)\neq c(i)}}2\alpha_{il}\left(\mathbf{u}_i-\frac{1}{2}(\mathbf{u}^c_{c(i)}+\mathbf{u}^c_{c(l)})\right)=0$$

Equivalently, as $\alpha_{kl}=-\hat{A}_{kl}$, $\sum_{j\in N_i}\alpha_{ij}=\hat{\mathbf{A}}_{ii}$, $c(j)=c(i)$ and $\mathbf{u}^c_{c(i)}=(\mathbf{P}\mathbf{u}^c)_i$, which is clear by the definition of the replacement matrix, the fine-to-coarse index map and the piecewise prolongation respectively,

$$\mathbf{u}_i=\frac{\hat{\mathbf{A}}_{ii}^{-1}}{2}\left(\hat{\mathbf{A}}_{ii}\mathbf{u}^c_{c(i)}+\sum_{l\in N_i}\alpha_{il}\mathbf{u}^c_{c(l)}\right)=\frac{1}{2}\mathbf{u}^c_{c(i)}-\frac{\hat{\mathbf{A}}_{ii}^{-1}}{2}\sum_{l\in N_i}\hat{\mathbf{A}}_{il}\mathbf{u}^c_{c(l)}=$$

$$=\left((\mathbf{I}-\frac{1}{2}\widehat{\mathbf{D}}^{-1}\hat{\mathbf{A}})\mathbf{P}\mathbf{u}^c\right)_i$$

So $\mathbf{P}_s=(\mathbf{I}-\frac{1}{2}\widehat{\mathbf{D}}^{-1}\hat{\mathbf{A}})\mathbf{P}_s$, which is just the smoothed prolongation we already know, except that here, we are using the replacement matrix $\hat{\mathbf{A}}$ instead of $\mathbf{A}$ to smooth the prolongation matrix!
All in all, these considerations may have convinced us that it is reasonable to use the replacement matrix to smooth the prolongation, however, as the matrix graphs of $\hat{\mathbf{A}}$ and $\mathbf{A}$ are the same, this does not solve the problem we had, which was that the additional nonzeros on coarser and coarser levels introduced by smoothing the prolongations keep compounding at an unacceptable rate because the matrices $(\mathbf{I}-\frac{1}{2}\widehat{\mathbf{D}}^{-1}\hat{\mathbf{A}})$ themselfs keep on getting denser and denser.

The next observation is that, while we have no choice but to define the coarse matrix as $\mathbf{A}_C=\mathbf{P}_s^T\mathbf{A}\mathbf{P}_s$ for the coarse grid correction to be an $\mathbf{A}$-orthogonal projection, we are free to modify $\hat{\mathbf{A}}_C$, as long as it stays equivalent to $\mathbf{A}_C$. For the concrete choice $\hat{\mathbf{A}}_C\coloneqq\mathbf{P}^T\hat{\mathbf{A}}\mathbf{P}$, we maintain equivalence of $\mathbf{A}_C$ and $\hat{\mathbf{A}}_C$ and, because as we know the coarse level matrix does not decrease in sparsity under $\mathbf{P}$, $\hat{\mathbf{A}}_C$ is just as sparse as $\hat{\mathbf{A}}$. This means that, while the smoothed prolongation keeps on increasing the nonzero elements per row as levels get coarser, the rate of increase is under control!

Definition 8.3: AscAMG Smoothed Prolongation

Given $\mathbf{A}$*,* $\hat{\mathbf{A}}$*, a piecewise prolongation* $\mathbf{P}$*, and a dampening parameter* $\omega \in (0, 1)$*, which will usually be* $\frac{1}{2}$*, the smoothed prolongation is defined as*

$$\mathbf{P}_s \coloneqq (\mathbf{I} - \omega \widehat{\mathbf{D}}^{-1} \hat{\mathbf{A}})\mathbf{P}$$

The coarse level replacement matrix is defined as

$$\hat{\mathbf{A}}_C \coloneqq \mathbf{P}^T \hat{\mathbf{A}} \mathbf{P}$$

Note 8.9. As $\hat{\mathbf{A}}$ is an M-matrix, the modification of the coarse replacement matrix introduces only a moderate additional constant in $\mathbf{A}_C \approx \hat{\mathbf{A}}_C$.

Even with these modifications of the smoothed prolongation and the coarse system, the resulting operator complexity of the V-cycle MG algorithm is still too large. More optimizations to this method, in the form of alternating prolongation types and the hierarchic prolongation will be introduced in section 8.2.3

8.2 Parallelization and Implementation

In this chapter we will discuss the implementation of AscAMG and present some of the optimization that has been done to achieve the performance and scalability that will be shown in chapter 9. After outlining some of the problems we have run into during developing AscAMG , section 8.2.1 will introduce the terminology we use to formulate the subsequently presented coarsening algorihm. Afterwards, in section 8.2.2 we will discuss problems very small coarse spaces pose in large computations. Section 8.2.3 contains the promised solutions to the problems we still have with the smoothed prolongation and finally, in section 8.2.4 we will touch upon the topic of the exact choice of smoothers that are available in AscAMG , which is a topic that has been completely neglected so far.

When talking about the problems we ran into during develompent of AscAMG , going through them in the chronological order in which we ran into them makes some sense, however we will choose a more orderly and logically more coherent order.

The first thing we needed was the coarsening algorithm, which is not trivial to efficiently implement because we are essentially partitioning a graph which is distributed over many procs with each only having access to the subgraph coming from its own subdomain. We also had to consider that we would run into problems when allowing collapsing of edges for which there are some procs that only "see" one of its vertices. We therefore had to either completely forbid the collapsing of these edges, which turned out to be too restrictive to be efficient, to continuously redistribute the algebraic mesh - and the parallel matrices with them, which we did not even attempt, or to find a kind of middle ground between these two approaches. We ended up having to forbid some edges in order to limit proc interfaces on coarse levels. The chronologically last major problem we ran into was that when doing larger computations, on very coarse levels, which had only very few DOFs per core left, the restrictions to the collapseable edges we had put in place turned out to halt further coarsening almost completely. Additionally, as DOFs/core decrease on the coarse levels, communication overhead plays a bigger and bigger role as opposed to actual computing. The solution to this was to redistribute the entire problem to fewer cores at certain breakpoints.

We also needed to implement an efficient distributed smoother. While jacobi is easily implemented even in a distributed setting, it also needs dampening with some parameter we have to choose appropriately. Gauss-Seidel on the other hand is incredibly difficult to implement efficiently in a distributed setting. Ultimately it turned out to be too difficult to make run efficiently on more than about 1000 cores. The solution to this was using the so-called ℓ1-smoother. In section 8.2.4, this will be elaborated upon.

After having dealt with that we ran into problems with trying to implement a distributed smoothed prolongation, the smoothing itself introduced entries in the coarse matrices that brought processors into contact with each other that were not in contact before at all, which would have increased the necessary communication for each smoothing step and also necessitated making major adjustments to the distribution of data and the communication structure on each level which we were not willing to put up with and implement respectively. Our solution to both this problem and the answer to the still unsolved question of how to limit operator complexity despite prolongation smoothing from the previous chapter will be given in section 8.2.3.

First, however, we will get some general notation out of the way. Some notation form previous chapters will be slightly modified or replaced throughout this section. Some notation from previous chapters will also be repeated here because we have opted to keep as much of the notation necessary for reading the chapter in one place.

Notation 8.2. We will write n_p for the number of procs and each proc will be identified by its rank, a number in $0 \dots n_p - 1$. The set of all procs will be $\mathcal{P} = \{0, 1, \dots n_p - 1\}$. For the proc with rank k we will write P^k.

Notation 8.3. The global number of DOFs (on the fine level) will be written as N, the local number of DOFs on some generic proc will be n and the number of local DOFs on P^k will be n^k.

Notation 8.4. The set of global DOFs will be $\mathcal{N} \coloneqq \{0 \dots N-1\}$, the set of DOFs local to some generic proc will be $\mathcal{N}_\ell \coloneqq \{0 \dots n-1\}$ and the set of DOFs local on P^k to $\mathcal{N}_\ell^k$.

Notation 8.5. The local-to-global DOF map $\mathbf{g}^k$ maps local dof-numbers on proc P^k to global ones. Its inverse is the global-to-local DOF map $\boldsymbol{\ell}^k = (\mathbf{g}^k)^{-1}$.

Notation 8.6. For each local DOF k, its proc-rank-set, or, more informally, its proc-set will be $I_k^{\mathcal{P}} \coloneqq \big\{l : P^l \text{ shares DOF } k\big\} \subseteq \mathcal{P}$. The proc-set of a global dof $\mathbf{g}^k(j)$ is, consistently defined, $I_{g^k(j)}^{\mathcal{P}} = I_j^{\mathcal{P},k}$. On proc P^j, for each local DOF k, its distant-proc-rank-set or dist-proc-set will be $I_k^{\mathcal{P}_j} \coloneqq I_k^{\mathcal{P}} \setminus \{j\}$, that is, the set of ranks of all other procs that share the k-th local DOF.

Notation 8.7. We will use the notation from chapter 4 for distributed matrices and write $\mathbf{A} \cong (\mathbf{A}^k)$. On the finest level, each $\mathbf{A}^k$ is the Finite elemetnt Matrix assembled on the subdomain Ω_k and therefore posesses a replacement matrix $\hat{\mathbf{A}}^k$. We will write the global replacement matrix as $\hat{\mathbf{A}} \cong (\hat{\mathbf{A}}^k)$.

Notation 8.8. A partition $\mathcal{C}$ of $\mathcal{N}$ induces a partition $\mathcal{C}^k$ of $\mathcal{N}_\ell^k$ by

$$\mathcal{C}^l \coloneqq \Big\{C_i \coloneqq \boldsymbol{\ell}^k(C_{\mathbf{g}^k(i)} \cap \mathbf{g}^k(\mathcal{N}_\ell^k)) : C \in \mathcal{C} \setminus \{D_\mathcal{C}\},\ C \cap \mathbf{g}^k(\mathcal{N}_\ell^k) \neq \emptyset\Big\} \cup \{D_\mathcal{C}^k\}$$

where we write $D_\mathcal{C}^k \coloneqq \boldsymbol{\ell}^k(D_\mathcal{C} \cap \mathbf{g}^k(\mathcal{N}_\ell^k))$ and $D^k \coloneqq \boldsymbol{\ell}^k(D \cap \mathbf{g}^k(\mathcal{N}_\ell^k))$. In particular, the numbering of the C_i of the local and global partitions are conistent. The piecewise prolongation based on this local partition is called $\mathbf{P}^k$. They induce the coarse replacement matrix by $\hat{\mathbf{A}}_C \cong (\mathbf{P}^T \hat{\mathbf{A}}^k \mathbf{P}^k)_k$.
The coarse DOF-set will be $\mathcal{N}^c \coloneqq \{0 \dots |\mathcal{C} \setminus \{D_\mathcal{C}\}| - 1\}$ $\mathcal{C}$ also induces a (global) coarse-to-fine DOF-map $\mathbf{f}$

$$\mathbf{f} : \begin{cases} \mathcal{N}^f \mapsto 2^{\mathcal{N}} \\ k \to C_k \end{cases}$$

and a (global) fine-to-coarse DOF-map $\mathbf{crs}$

$$\mathbf{crs} : \begin{cases} \mathcal{N} \mapsto \mathcal{N}^c \\ k \to j \quad \text{such that } k \in C_j \in \mathcal{C} \end{cases}$$

Note 8.10. The local piecewise prolongations $\mathbf{P}^k$ do not induce the global matrix $\mathbf{P}$, because they have full values $\mathbf{P}_{ij}^k = 1$ for $i \in C_j \in \mathcal{C}^k$ and therefore

$$(\sum_i \mathbf{E}_i \mathbf{P}^i \mathbf{E}_i^T)_{kl} = \begin{cases} |I_k^{\mathcal{P}}| & \text{if } k \in C_l \in \mathcal{C} \\ 0 & \text{else} \end{cases}$$

$(\mathbf{P}^k)_k$ is a matrix-analogon to a cumulated vector as, with the E^i as in chapter 4,

$$\mathbf{P}^i = E^{i,T}\mathbf{P}E^i$$

However, keep in mind that we have not formally defined non-rectangular parallel matrices. If $(\mathbf{v}^k)_k$ is a distributed/cumulated parallel vector, then $(\mathbf{P}^k\mathbf{v}^k)_k$ is again a distributed/cumulated parallel vector and represents $\mathbf{Pv}$. We can do multiplication with the global matrix $\mathbf{P}$ locally, without worrying about or modifying the parallel status of a parallel vector!

Note 8.11. In order to compute $\mathbf{P}$, we are not actually interested in computing a global $_C$, we only need the local partitions ^k_C on all procs but do we need them to be consistent!

8.2.1 Coarsening

In order to compactly formulate the coarsening algorithm as well as most considerations in subsequent sections, we need a little more notation that replaces the mesh with a purely algebraic construct.

Definition 8.4: Weighted Vertices and Edges

A weighted vertex is a tuple

$$v_k \coloneqq (k, x, y) \in \mathbb{N} \times \mathbb{R}^+ \times \mathbb{R}^+$$

A weighted edge is a tuple

$$e_{kl} \coloneqq (\{k, l\}, x, y) \in \mathbb{N}^2 \times \mathbb{R}^+ \times \mathbb{R}^+$$

With the usual projection operators π_1, π_2, π_3, the vertex-to-index map for weighted vertices is defined by $\mathbf{i} \coloneqq \pi_1$. For weighted edges, we define the edge-to-vertex maps as $\mathbf{i}_1 \coloneqq \pi_1 \circ \pi_1$ and $\mathbf{i}_2 \coloneqq \pi_2 \circ \pi_1$. The weight-functions $\mathbf{w} \coloneqq \pi_2$ and collapse-weight-functions $\mathbf{cw} \coloneqq \pi_3$ are defined for both weighted edges and vertices.
We call a set A of weighted edges or a set B of weighted vertices regular if π_1 is injective on A or B respectively. That just means, there are no "double" edges or vertices that only differ in weights.
For two such sets, if $\mathbf{i}_1(A) \cup \mathbf{i}_2(A) \subseteq \mathbf{i}(B)$, the edge-to-vertex maps $\mathbf{v}_1, \mathbf{v}_2$ are defined by:

$$\mathbf{v}_k \coloneqq \begin{cases} A \mapsto B \\ e \to (\mathbf{i}^{-1} \circ \mathbf{i}_k)(e) \end{cases}$$

Definition 8.5: (Global) Algebraic Mesh

With the global system and replacement matrices $\mathbf{A}$ *and* $\hat{\mathbf{A}}$*, the (global) edge weights* α_{ij}*, vertex weights* β_i*, vertex-collapse-weights* $\mathbf{w}_i$*, and edge collapse weights* w_{ij}*, we define the (global) algebraic mesh as*

$$\mathcal{M} \coloneqq (\mathcal{V}, \mathcal{E}) \tag{8.8}$$

Where $\mathcal{V}$ *is the set of (weighted) vertices*

$$\mathcal{V} \coloneqq \{(k, \beta_k, w_k) : k \in \mathcal{N}\} \tag{8.9}$$

And $\mathcal{E}$ *is the set of (weighted) edges*

$$\mathcal{E} \coloneqq \Big\{(\{i,j\}, \alpha_{ij}, w_{ij}) : i,j \in \mathcal{N} \wedge \hat{\mathbf{A}}_{ij} \neq 0\Big\} \tag{8.10}$$

Definition 8.6: (Local) Algebraic Mesh

For each proc P^k*, with its' local system and replacement matrices* $\mathbf{A}^k$ *and* $\hat{\mathbf{A}}^k$*, the (global) edge weights* α_{ij}*, vertex weights* β_i*, vertex-collapse-weights* $\mathbf{w}_i$*, and edge collapse weights* w_{ij}*, we define the (local) algebraic mesh as*

$$\mathcal{M}^k \coloneqq (\mathcal{V}^k, \mathcal{E}^k) \tag{8.11}$$

Where $\mathcal{V}^k$ *is the set of (weighted) vertices*

$$\mathcal{V}^k \coloneqq \Big\{(l, \beta_{\mathbf{g}(l)}, w_{\mathbf{g}(l)}) : l \in \mathcal{N}_\ell^k\Big\} \tag{8.12}$$

nd $\mathcal{E}^k$ *is the set of (weighted) edges*

$$\mathcal{E}^k \coloneqq \Big\{(\{i,j\}, \alpha_{\mathbf{g}(i)\mathbf{g}(j)}, w_{\mathbf{g}(k)\mathbf{g}(j)}) : i,j \in \mathcal{N}_\ell^k \wedge \hat{\mathbf{A}}_{\mathbf{g}(i)\mathbf{g}(j)} \neq 0\Big\} \tag{8.13}$$

Note 8.12. Note that the local algebraic meshes are defined with the global edge- and vertex-weights and vertex strengths! In praxis, these have to be computed from the local ones.

Note 8.13. In addition to the edge-collapse-weights w_{ij} and the vertex-collapse-weights w_i, which are all that is needed in the coarsening algorithms, the algebraic mesh also holds the edge-weights α_{ij} and the vertex-weights β_i. The reason for that is that with this additional data, it holds all the information $\hat{\mathbf{A}}$ does, therefore we never need to explicitly assemble $\hat{\mathbf{A}}$ at all. The coarse algebraic mesh can also relatively easily computed from the fine algebraic mesh, without ever assembling the coarse replacement matrix $\hat{\mathbf{A}}_C$.

Note 8.14. The sets $\mathcal{E}$ and $\mathcal{V}$ are, by definition, regular and the global edge-to-vertex maps $\mathbf{v}_1, \mathbf{v}_2 : \mathcal{E} \mapsto \mathcal{V}$ are defined. The same goes for the local sets $\mathcal{E}^k$ and $\mathcal{V}^k$ and the local edge-to-vertex maps $\mathbf{v}_1^k, \mathbf{v}_2^k$.

Definition 8.7: Vertex- and Edge-Maps

For each proc P^k, the local-to-global vertex-map $\mathbf{g}^k$ is

$$\mathbf{g}\begin{cases}\mathcal{V}^k \mapsto \mathcal{V}\\ v \to (\mathbf{i}_1^{-1} \circ \mathbf{g}^k \circ \mathbf{i}_1)(v)\end{cases}$$

For each proc P^k, the global-to-local vertex-map $\boldsymbol{\ell}^k$ is

$$\mathbf{g}\begin{cases}\mathbf{g}^k(\mathcal{V}^k) \mapsto \mathcal{V}^k\\ v \to (\mathbf{i}_1^{-1} \circ \boldsymbol{\ell}^k \circ \mathbf{i}_1)(v)\end{cases}$$

For each proc P^k, the local-to-global edge-map $\mathbf{g}^k$ is

$$\mathbf{g}\begin{cases}\mathcal{E}^k \mapsto \mathcal{E}\\ e \to (\mathbf{i}_1^{-1} \circ (\mathbf{g}^k, \mathbf{g}^k) \circ \mathbf{i}_1)(e)\end{cases}$$

For each proc P^k, the global-to-local edge-map $\boldsymbol{\ell}^k$ is

$$\mathbf{g}\begin{cases}\mathbf{g}^k(\mathcal{E}^k) \mapsto \mathcal{E}^k\\ e \to (\mathbf{i}_1^{-1} \circ (\boldsymbol{\ell}^k, \boldsymbol{\ell}^k) \circ \mathbf{i}_1)(e)\end{cases}$$

After this exhaustive introduction of notation, we will now present a compact formalism to describe the parallelism of DOFs in a distributed setting which will subsequently be used to discuss our coarsening algorithm as well as an optimization for very large problems on many cores we have implemented.

DOF EQCS

Definition 8.8: DOF-EQCs

The proc-sets $I_k^{\mathcal{P}}$ define an equivalence relation on $\mathcal{N}$ by

$$j \sim k :\Leftrightarrow I_j^{\mathcal{P}} = I_k^{\mathcal{P}}$$

We define a partial order on $\mathcal{N}$ by

$$j \lhd k :\Leftrightarrow I_j^{\Omega} \subseteq I_k^{\Omega}$$

*This induces a partial order on $\mathcal{N}/_\sim$ by $[j] \lhd [k] :\Leftrightarrow j \lhd k$, which we call the **eqc-hierarchy**. For the equivalence class of some vertex a we write $[a]^v$ instead of the more common $[a]_\sim$. For any $J \subseteq \mathcal{P}$, we will write its DOF equivalence class, or simply its DOF eqc, as $[\cdot]_J^v \coloneqq \{k \in \mathcal{N} : I_k^{\mathcal{P}} = J\}$.*
The "master" of an EQC $[\cdot]_J^v$ is P^k with $k = \min\{j : j \in J\}$.
An equivalence class $[j]^v \in \mathcal{N}/_\sim$ is called a:

- ***V-eqc:*** *"vol-eqc", if $|I_j^{\mathcal{P}}| = 1$*
- ***F-eqc:*** *"face-eqc", if $|I_j^{\mathcal{P}}| = 2$*
- ***W-eqc:*** *"wire-eqc", if $|I_j^{\mathcal{P}}| \geq 3$*

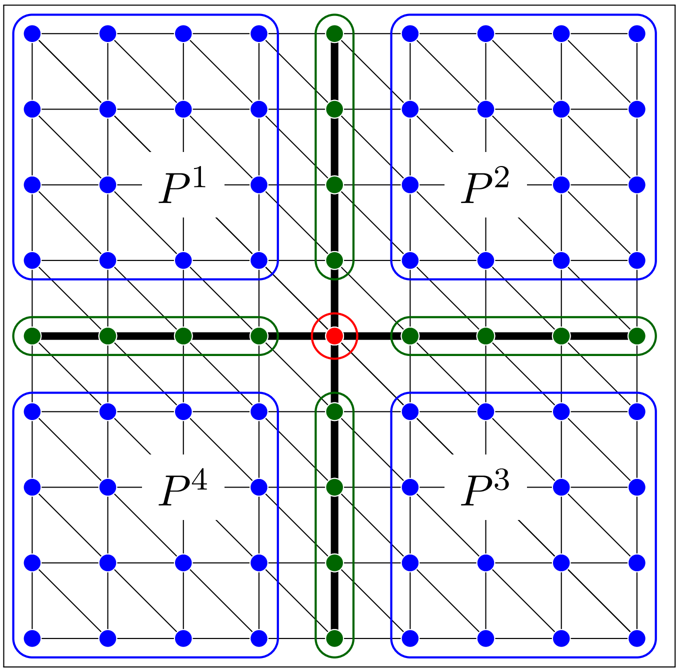


Figure 8.5: DOF-EQCs for a simple case of four procs. The bold lines are proc-interfaces. Blue:vol-eqcs, Green:face-eqcs, Red:wire-eqcs

Note 8.15. The terms vol-, face-, and wire-eqc come from the fact that if the DOFs come from a finite element space defined on a mesh, in 3d, vol-eqcs usually lie in the 3-dimensional interiors of subdomains , face-eqcs lie on usually 2-dimensional interfaces between two subdomains and wire-eqcs lie on the intersections of the subdomain-interfaces, which are usually 1- or 0-dimensional. With algebraic multigrid, on coarse levels, this is not true anymore, however it does help with visualizing the coarsening and prolongation algorithms.

Note 8.16. In general, perforing operations on dofs in v-eqcs can be done locally to each processor and requires no communication and the communication necessary for operations on f-eqcs is only pair-wise and is also quite easy to implement and relatively cheap (as long es the number of neighbouts for each proc is bounded), operations on w-eqcs can require extensive (potentially even global!) communication which often difficult to implement efficiently.

For formulating the coarsening algorithm we will also need to define equivalence classes on the set of edges

Definition 8.9: Edge-EQCS

For an edge $e \in \mathcal{E}$, we define its proc-set $I^{\mathcal{P}}_e$ as

$$I^{\mathcal{P}}_e \coloneqq I^{\mathcal{P}}_{\mathbf{i}_1(e)\mathbf{i}_2(e)} \coloneqq I^{\mathcal{P}}_{\mathbf{i}_1(e)} \cap I^{\mathcal{P}}_{\mathbf{i}_2(e)}$$

Again, this induces an equivalence relation on $\mathcal{E}$ by

$$e \sim \tilde{e} :\Leftrightarrow I^{\mathcal{P}}_e = I^{\mathcal{P}}_{\tilde{e}}$$

The partial orders on $\mathcal{E}$ and $\mathcal{E}/_\sim$ are defined in the same way as for DOFs, and will again be called the eqc-hierarchy.
For the equivalence class of some edge a we write $[a]^e$ instead of the more common $[a]_\sim$. For any $J \subseteq \mathcal{P}$, we will write its edge equivalence class, or simply its edge-eqc, as $[\cdot]^e_J \coloneqq \left\{e \in \mathcal{E} : I^{\mathcal{P}}_e = J\right\}$. These classes in $\mathcal{E}/_\sim$ will be called vol-, face-, and wire- eqcs in the same way that DOF-eqcs are. The "master" of an EQC $[\cdot]^e_J$ is P^k with $k = \min\{j : j \in J\}$.
We will call an edge an in-eqc edge if $I^{\mathcal{P}}_e = I^{\mathcal{P}}_{\mathbf{i}_1(e)} = I^{\mathcal{P}}_{\mathbf{i}_2(e)}$ and a cross-eqc edge if $I^{\mathcal{P}}_{\mathbf{i}_1(e)} \neq I^{\mathcal{P}}_{i_2(e)}$.

Note 8.17. The case $I^{\mathcal{P}}_e = \emptyset$ will be excluded, it cannot occur on the finest level and the coarsening algorithm and parallel prolongation will be built in such a way that it will also not occur on any coarser levels.

Note 8.18. In figure 8.5, exactly those edges are in-eqc that do not cross any differently colored vertices.

The Coarsening Algorithm

To reiterate and summarize what we know from section 8.1, it is the job of the coarsening algorithm to construct a partition $\mathcal{C}$ of $\mathcal{N}$ as needed for building a piecewise prolongation (definition 8.2) such that conditions (8.5) and (8.6) are fulfilled, that is for $\sigma \in (0,1)$ (we will usually choose $\sigma = 0.1$)

$$\begin{aligned} w_{ij} &> \sigma \qquad \forall \{i,j\} \in \mathcal{C} \\ w_i &> \sigma \qquad \forall i \in D_{\mathcal{C}} \setminus D \end{aligned}$$

We will first give an algorithm that works sequentially. Algorithm 4 is very simple, as we are very restrictive with which agglomerates $C_i \in \mathcal{C}$ we allow (that is, only single DOFs or paris of DOFs). We also do not need to concern ourselfs as much with any criterium for the agglomerates that limits the nonzero entries introduced by smoothing the prolongation (see section 8.1.3).
Algorithm 4 can be given an initial partial partition $\mathcal{C}_0$ to start from which nothing will be removed, it will only be added to. It also takes a boolean parameter MAKE_COMPLETE, which allows for an incomplet partition that does not contain any single DOFs (it consists only of collapsed edges and vertices). Both of these options will be used later, for now we will call it with $\mathcal{C}_0 = \emptyset$ and MAKE_COMPLETE=TRUE.

Algorithm 4 Coarsen Algebraic Mesh sequentially. Input:

- $\mathcal{M}$ algebraic mesh
- $D \dots$ set ofdirichlet dofs.
- $\mathcal{C}_0 \dots$ initial partition, can be empty. If it not empty it must contain the set $D_c \supseteq D$. Will only be added to, no elements will be removed or modified.
- MAKE_COMPLETE… boolean value, if true, returns a complete partition of the DOFs. Otherwise, only collapses vertices and edges and leaves left over DOFs unassigned.

Output:

- $\mathcal{C} \dots$ (local) partition.

```
procedure CAM_SEQ($\mathcal{M}$, $D$, $\mathcal{C}_0$, MAKE_COMPLETE)
    if $\mathcal{C}_0 = \emptyset$ then
        Set $D_c \coloneqq D$
        Set $\mathcal{C} \coloneqq \{D_c\}$
        Set $\mathcal{U} \coloneqq D$
    else
        Set $\mathcal{U} \coloneqq \bigcup_{C \in \mathcal{C}_0} C$
    for $e \in \mathcal{E}$, in descending order of their collapse-weight $\mathbf{cw}(e)$ do
        if $\mathbf{cw}(e) > \sigma$ and $\mathbf{i}_1(e) \notin \mathcal{U}$ and $\mathbf{i}_2(e) \notin \mathcal{U}$ then
            $\mathcal{C} \rightarrow \mathcal{C} \cup \{\{\mathbf{i}_1(e), \mathbf{i}_2(e)\}\}, \quad \mathcal{U} \rightarrow \mathcal{U} \cup \{\mathbf{i}_1(e), \mathbf{i}_2(e)\}$
    for $v \in \mathcal{V}$, in descending order of their collapse-weight $\mathbf{cw}(v)$ do
        if $\mathbf{cw}(v) > \sigma$ and $\mathbf{i}(v) \notin \mathcal{U}$ then
            $\mathcal{D}_c \rightarrow \mathcal{D}_c \cup \{\mathbf{i}(v)\}, \quad \mathcal{U} \rightarrow \mathcal{U} \cup \{\mathbf{i}(v)\}$
        else if MAKE_COMPLETE then
            $\mathcal{C} \rightarrow \mathcal{C} \cup \{\{\mathbf{i}(v)\}\}, \quad \mathcal{U} \rightarrow \mathcal{U} \cup \{\mathbf{i}(v)\}$
    return $\mathcal{C}$
```

Note 8.19. Algorithm 4 detects weakly enforced dirichlet boundary conditions and collapses all concerned DOFs. This can be seen easily: When we weakly enforce a dirichlet condition at DOF j, we add an l2-term with a coefficient that is larger than α by a couple of orders of magnitude. This means that β_i is larger than all α_{ij} by a multiple orders of magnitude, which means $w_{ij} << 1$ and no edge connected to i can be collapsed. We also have $w_i \approx 1$ and therefore the DOF is collapsed.

A first, simple approach to parallelizing algorithm 4 is grounded in the observation that for each P^k, its purely local part of the algebraic mesh, consisting of the purely local vertices and the edges that connect purely local DOFs to each other, is not seen from the outside by any other proc. We can just apply algorithm 4 to each of these local sub-meshes sequentially. In fact, there is even more we can do purely locally, without any communication. For that, let us define more rigorously what a "block of an algebraic mesh" is:

Definition 8.10: Algebraic Mesh Block

For $J \subseteq \mathcal{P}$, the "J-Block of $\mathcal{M}$", is defined as $\mathcal{M}_J \coloneqq (\mathcal{V}_J, \mathcal{E}_J)$, with the eqc-vertex set

$$\mathcal{V}_J \coloneqq \{v \in \mathcal{V} : \mathbf{i}(v) \in [\cdot]^v_J\} \subseteq \mathcal{V}$$

and the in-eqc-edge set

$$\mathcal{E}_J \coloneqq \{e \in \mathcal{E} \cap [\cdot]^v_J : [\mathbf{v}_1(e)]^v = [\mathbf{v}_2(e)]^v\} \subseteq \mathcal{E}$$

For each proc P^k, the local J-Block is defined as $\mathcal{M}^k_J \coloneqq (\mathcal{V}^k_J, \mathcal{E}^k_J)$, with the local eqc-vertex set

$$\mathcal{V}^k_J \coloneqq \boldsymbol{\ell}^k \left(\mathcal{V}_J \cap \mathbf{g}^k(\mathcal{V}^k)\right)$$

and the local in-eqc-edge set

$$\mathcal{E}^k_J \coloneqq \boldsymbol{\ell}^k \left(\mathcal{E}_J \cap \mathbf{g}^k(\mathcal{E}^k)\right)$$

For each J we also define the global cross-eqc-edge set

$$\mathcal{G}_J \coloneqq \{e \in \mathcal{E} \cap [\cdot]^v_J : [\mathbf{v}_1(e)]^v \neq [\mathbf{v}_2(e)]^v\} \subseteq \mathcal{E}$$

For each proc P^k and proc-set J, the local cross-eqc-edge set is

$$\mathcal{G}^k_J \coloneqq \boldsymbol{\ell}^k \left(\mathcal{G}_J \cap \mathbf{g}^k(\mathcal{G}^k)\right)$$

For obious reasons, we call $\mathcal{M}_J$ a "vol-block" if $|J| = 1$, a "face-block" if $|J| = 2$ and a "wire-block" if $|J| = 3$.

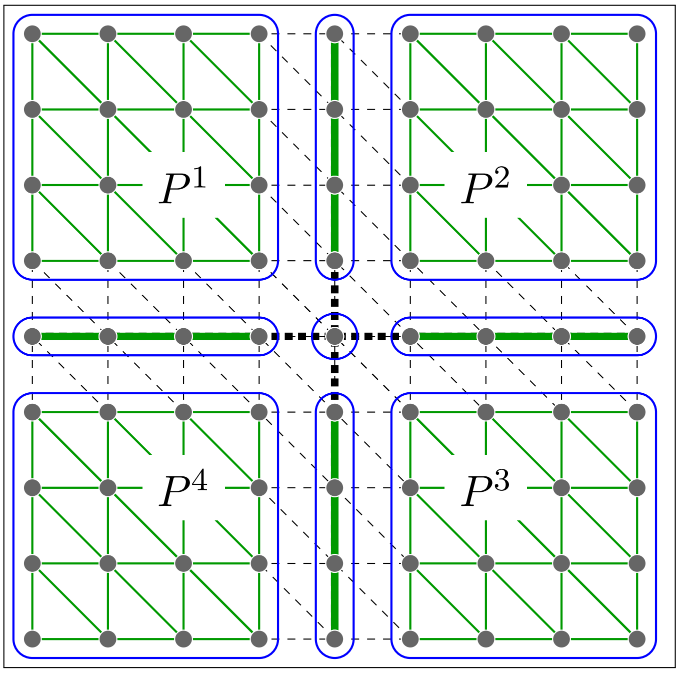
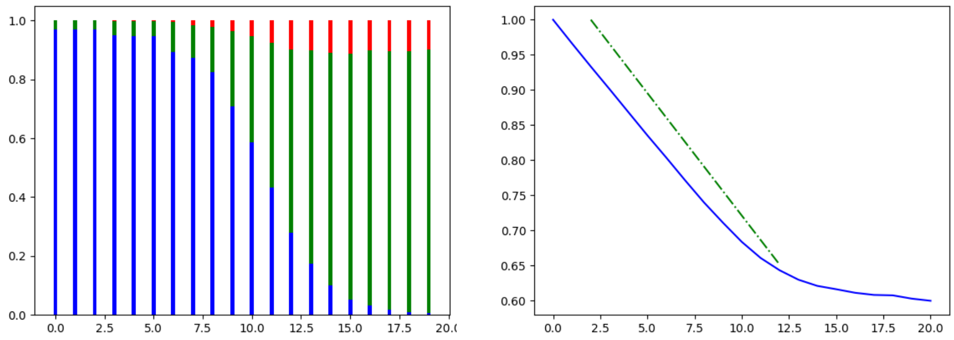


Figure 8.6: The same case as figure 8.5. All green edges are in-eqc and are therefore viable for collapse by algorithm 5. Cross-eqc edges are dashed to indicate that the are not seen by algorithm 5.

In addition to applying algorithm 4 to each $\mathcal{M}_{\{k\}}$ locally, we can actually apply it to all blocks $\mathcal{M}_J$ locally on each proc $P^j \in J$. The collapsing of each block is independent and the results for collapsing each block must be consistent across all procs as algorithm 4 is deterministic.

Algorithm 5 does just that, is fairly easy to implement and also looks pretty good at first glance. In fact, it does work very well for the first couple of levels, however after that things go awry quickly as seen in figure 8.7.



a: x-axis: levels. y-axis: Fraction of DOFs in vol-eqcs (blue), face-eqcs (green) and wire-eqcs (red).

b: x-axis: levels. y-axis: Relative logarithmic number of DOFs compared to level 0. Dashed: $2^{-l} \cdot$ (inital NDOF)

Figure 8.7: Behavior of algorithm 5 for the 3d-problem poisson problem, $\alpha = 1, \beta = 0$ on the unit cube with approximately 10^8 DOFs on only 60 cores.

The problem that occurs here is caused by the fact that the support of coarse base functions corresponding to nodes in vol-eqcs tend to have 3-dimensional support, those in face-eqcs tend to have 2-dimensional and those in wire-eqcs 1-dimensional support. The overlap of two coarse level basis functions with d-dimensional support tends to be (d-1)-dimensional, and as the edge-weight for the corresponding coarse algebraic edge is given by the sum over all finest level edge weights that lead through the common support, it scales like l^{d}, with the number of levels l. The overlap of a (d+k)-dimensional and a d-dimensional base function support overlap still tends to be d-dimensional, and the corresponding edge-weight scales like l^{d}.

Summarized, vol-vol- and vol-face edge-weights scale like l^2, vol-wire, face-face and face-wire edge-weights like l and wire-wire weights stay constant. After enough levels, all face-face edge weights are outscaled by surrounding vol-face edge weights and all wire-wire edge weights are outscaled by surrounding face-wire edge weights until they cannot fulfill condition (8.5), which, when $\beta = 0$ just says that an edge that is admissible for collapse has to represent a certain fraction of the total edge weights coming together in either vertex. Of coarse, now the vol-face and face-wire edges are very strong and admissible for collapse, but algorithm 5 never even considers these (cross-eqc) edges. We will now see what we can do about that.

Algorithm 5 Coarsen Algebraic Mesh EQC-wise on each proc. Input:

- $\mathcal{M}\dots$ the (local) algebraic mesh
- $D\dots$ (local) set of dirichlet dofs.
- $J^{\mathcal{P}}\dots$ set of all proc-sets $J \subseteq \mathcal{P}$ that contain the calling proc with non-empty $\mathcal{V}_J^k$
- $\mathcal{C}_0\dots$ initial (local) partition, can be empty. If it not empty it must contain the set $D_{\mathcal{C}} \supseteq D$. Will only be added to, no elements will be removed or modified.
- MAKE_COMPLETE... boolean value, if true, returns a complete partition of the DOFs. Otherwise, only collapses vertices and edges and leaves left over DOFs unassigned.

Output:

- $\mathcal{C}\dots$ (local) partition

```
procedure CAM_EQC_WISE($\mathcal{M}$, $D$, $J^{\mathcal{P}}$, $\mathcal{C}_0$, MAKE_COMPLETE)
    if $\mathcal{C}_0 = \emptyset$ then
        Set $D_{\mathcal{C}} \coloneqq D$
        Set $\mathcal{C} \coloneqq \{D_{\mathcal{C}}\}$
        Set $\mathcal{U} \coloneqq D$
    else
        Set $\mathcal{U} \coloneqq \bigcup_{C \in \mathcal{C}_0} C$
    for $J \in J^{\mathcal{P}}$ do
        Set $\mathcal{D} \coloneqq$ CAM_SEQ($\mathcal{M}_J, D, \emptyset, TRUE$)
        $\mathcal{C} \to \mathcal{C} \cup \mathcal{D}$
    return $\mathcal{C}$
```

Note 8.20. We assume that each proc P^k with $k \in J$ has access to everything in $\mathcal{M}_J$. This is not actually induced by the parallelization of the Finite Element space "out of the box": Starting out from the finest level, each rank in J knows about all DOFs with vertices in $\mathcal{V}_J$ and the global vertex-weights and vertex-collapse-weights are easy to compute - this is just one parallel vector Cumulate-operation each. However, there can be edges $e \in \mathcal{E}_J$ that , even though are in-eqc, are not known to all P^k in J. To see this, consider, for example, the 2-dimensional case of a triangle where two of its edges lie on a subdomain interface between Ω_i and Ω_j and the third lies within Ω_i. P^i will know about the corresponding algebraic edge as $\mathbf{A}^i$, which was assembled on Ω_i has a corresponding entry, but $\mathbf{A}^j$ does not and therefore P^j does not know about this algebraic edge. This is the reason that we use $\hat{\mathbf{A}}$ istead of $\hat{\mathbf{A}}^k$ in the definition of $\mathcal{E}_J^k$. It is, however a relatively simple matter to synchronize this information once in the beginning and to then make sure to keep it consistent whenever constructing a coarser level. Computing the global edge- and edge-collapse weights is also easy and only has to be done once.

There are two possible ways out of this dilemma. One is to redistribute the algebraic mesh, and with it the system matrix such that cross-eqc edges become in-eqc edges. The other is to try and find an algorithm that is also capable of collapsing cross-eqc edges. AscAMG went the latter way.

The first thing we have to think about is which cross-eqc edges we will even allow for collapse in the first place. If we do not put any restrictions in place, we will end up bringing procs into contact with each other that were not in contact in the first place. This means that whenever we do have to do MPI-communication on those coarse levels, as we have to every time we do cumulate a parallel vector, which happens all the time, we do not only have to send more data but we also have to send the data to more other procs which results in an increase in message size and number. Therefore, we want to forbid at least all edges that would result in creating additional proc-interfaces.

On the other hand, at the very least, we want to allow collapsing of all edges e that connect DOFs "upwards" or "downwards" in the eqc-hierarchy, that is $e \in \mathcal{E}$ such that $[\mathbf{i}_l(e)]^v \lhd [\mathbf{i}_{1-l}(e)]^v$. In that case, the proc-set of one of the edge's vertices is a superset of the other one's and collapsing the edge clearly introduces a coarse level DOF in the eqc of the "larger" one. This just removes one DOF from the "smaller" eqc and leaves one in the "larger" one. In absolute numbers, this does not even increase the size of any proc-interfaces. In the general case, collapsing an edge e introduces a coarse dof in the eqc $[\cdot]^v_{I^{\mathcal{P}}_{\mathbf{i}_1(e)} \cup I^{\mathcal{P}}_{\mathbf{i}_2(e)}}$, in contrast to the edges own eqc which is $[\cdot]^e_{I^{\mathcal{P}}_{\mathbf{i}_1(e)} \cap I^{\mathcal{P}}_{\mathbf{i}_2(e)}}$. This can potentially introduce a coarse DOF in an equivalence class which was empty beforehand, but as long as it does not create a new proc-interface we will put up with that.

Definition 8.11: Admissible Cross-EQC-Edges

The set of allowed EQC-identifiers is the set of all proc-sets where there is at least a larger one that already has a DOF in its EQC. It is called $H^{\mathcal{P}}$ to indicate its relation to the hierarhcy of the relevant EQCs.

$$H^{\mathcal{P}} \coloneqq \{J \subseteq \mathcal{P} : \exists I \subseteq \mathcal{P},\ [\cdot]^v_I \neq \emptyset,\ J \subseteq I\} \tag{8.14}$$

An edge is called algebraically admissible for collapse if it fulfills condition (8.5) and topologically admissible if $I^{\mathcal{P}}_{\mathbf{i}_1(e)} \cup I^{\mathcal{P}}_{\mathbf{i}_2(e)} \in H^{\mathcal{P}}$. An algebraically and topologically admissible edge is called absolutely admissible.

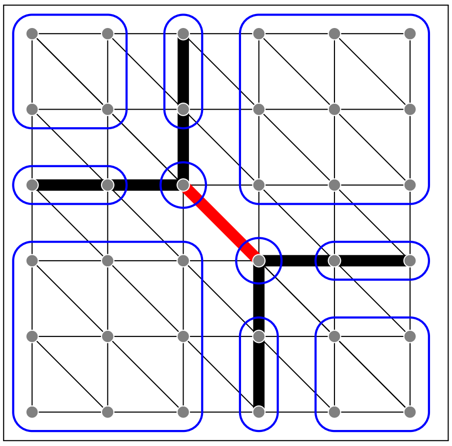


Figure 8.8: An example for a topologically not admissible edge (red). Proc-interfaces are again bold.

Now we know which edges are admissible in the first place and we only need an algorithm that is capable of following through.

Our distributed coarsening algorithm (algorithm 6) consists of 6 steps:

- **Pre-Coarsening:** We do an inital coarsening-step with the eqc-wise coarsening algorithm 5 from before, however we set MAKE_COMPLETE=FALSE, so we do only edge- and vertex-collapses. This step can be done locally.
- **Vertex-Marking:** On each proc we iterate through all local absolutely admissible cross-eqc edges and mark each of it's vertices with the edges' collapse-weight.
- **Vertex-Reduction:** We gather all vertex-markings on the master of each eqc. The master of the eqc assigns each vertex to the proc that has marked it with the highest weight. Gathering the vertex-markings requires communication.
- **Cross-Coarsening:** On each proc we iterate through all local absolutely admissible cross-eqc edges and, if both got assigned to the local proc, collapse them. No communication required.
- **Restoring Consistency:** Next, for all collapsed cross edges e we have to communicate the fact that the edge was collapsed to all ranks in $[e]^e$ (the other procs in $[e]^e$ so far only know which proc the two vertices got assigned to, not that the connecting edge is actually collapsed). All procs in $(I^{\mathcal{P}}_{\mathbf{i}_1(e)} \setminus I^{\mathcal{P}}_{\mathbf{i}_2(e)}) \cup (I^{\mathcal{P}}_{\mathbf{i}_2(e)} \setminus I^{\mathcal{P}}_{\mathbf{i}_1(e)})$ do not know of the existance of e, they only have access to one of e's vertices and all they need to be notified of is that the coarse DOF belongs into the EQC $[\cdot]^v_{I^{\mathcal{P}}_{\mathbf{i}_2(e)} \cap I^{\mathcal{P}}_{\mathbf{i}_1(e)}}$. This requires some relatively fancy communication.
- **Post-Coarsening:** Do a final caorsening-step with the eqc-wise coarsening algorithm 5, this time with MAKE_COMPLETE=TRUE to accept all otherwise unassigned dofs on the coarse level.

In algorithm 6, all of the required bookkeeping and communication is only hinted at. M is used as whatever object holds the information on the edge markings $M(i,j)$ holds the value P^j has marked the global dof j with, Gather M does not gathering of all information to some root-proc but gathering the entire "row" $M(i,\cdot)$ on the master of the EQC $[i]^v$ for all DOFs $i \in \mathcal{N}$. Broadcasting M means making the value $M(i, \text{master}/i)$ available to all procs in $I^{\mathcal{P}}_i$.

```
procedure CAM_DISTRIBUTED($\mathcal{M}$, $D$, $J^{\mathcal{P}}$)
    Set $D_{\mathcal{C}} \coloneqq D$
    Set $\mathcal{C} \coloneqq \{D_{\mathcal{C}}\}$
    Set $\mathcal{U} \coloneqq D$
    Set $M \coloneqq 0$
    Set $K \coloneqq$ rank of the local proc
    Set $\mathcal{D} \coloneqq$ CAM_SEQ($\mathcal{M}_J, D, \emptyset, FALSE$)
    $\mathcal{C} \to \mathcal{C} \cup \mathcal{D}$
    for $J \in J^{\mathcal{P}}$ do
        for $e \in \mathcal{G}^k_J$ do
            $M(\mathbf{g}^k(\mathbf{i}_1(e)), K) = \mathbf{cw}(e)$
            $M(\mathbf{g}^k(\mathbf{i}_2(e)), K) = \mathbf{cw}(e)$
    Gather $M$ to master
    for $J \in J^{\mathcal{P}}$ do
```

```
15:         if $K = \min(J)$ then
16:             for $e \in \mathcal{G}^k_J$ do
17:                 for $l \in \{1, 2\}$ do
18:                     $q = \arg\max_{p \in J}\{M(\mathbf{i}_l(e), p)\}$
19:                     $M(\mathbf{i}_l(e), q) = -1$
20:     Broadcast $M$
21:     for $J \in J^{\mathcal{P}}$ do
22:         for $e \in \mathcal{G}^k_J$ do
23:             if $M(\mathbf{i}_1(e), K) = -1$ and $M(\mathbf{i}_2(e), K) = -1$ then
24:                 $\mathcal{C} = \mathcal{C} \cup \{\mathbf{i}_1(e), \mathbf{i}_2(e)\}$
25:     Restore Consistency
26:     Set $\mathcal{D} :=$ CAM_SEQ$(\mathcal{M}_J, D, \emptyset, TRUE)$
27:     $\mathcal{C} \to \mathcal{C} \cup \mathcal{D}$
28:     return $\mathcal{C}$
```

Algorithm 6: Distributed Coarsening Algorithm. Step 5 is only hinted at in line 39. Input:

- $\mathcal{M}$ ... the global algebraic mesh
- D ... (local) set of dirichlet dofs.
- $J^{\mathcal{P}}$ ... set of all proc-sets $J \subseteq \mathcal{P}$ that contain the calling proc with non-empty $\mathcal{V}^k_J$

Output:

- $\mathcal{C}$ ... (local but globally consistent) partition, that is, a partition of $\mathcal{N}^k_\ell$ where the C_i are indexed consistently across all procs (such that they induce a global partition).

Note 8.21. In praxis, algorithm 6 returns, besides ${}_{C}$, also some additional information about collapsed cross-edges wherever the local proc sees one of its vertices. This is basically the information mentioned above, under the point "Restoring Consistency", and is needed to be able to properly build the ParallelDof-object (which holds all of the parallelism-information used by standard NGSolve lienar algebra classes, see chapter 4) and the algebraic mesh on the coarse level. In theory, however, we can ignore this and can recover the global partition from the local ones by

$$C_i := \bigcup_{\substack{p \in \mathcal{P} \\ i \in \mathbf{g}^p(\mathcal{N}^p_\ell)}} C_{\boldsymbol{\ell}^p(i)} \quad i = 1 \dots N$$

Here, we have to take the union of the local C_l because collapsed cross-edges with only one vertex shared by a particular proc $\mathcal{P}^k$ are represented as a singleton in $\mathcal{C}^k$. With

$$D_C := \bigcup_{p \in \mathcal{P}} \mathbf{g}^p(D^p_C)$$

we have

$$\mathcal{C} = \{C_i : i = 1 \dots N\} \cup \{D_C\}$$

What can in general be said about coarsening for AscAMG is that the fact that we restrict ourselfs to agglomerates of maximum size 2 made it a lot easier to both come up with and implement a reasonably efficient distributed coarsening algorithm. In this place we get back a bit of what we loose by not going for larger agglomerates. For how relatively straightforward the algorithm is, it performs pretty well, although there is certainly room for improvement. In particular, it is probably the least optimized part of AscAMG , as we are on the one hand only coarsening once on each level and as it was, on the other hand, one of the messier and more complicated things to implement. In contrast, the smoothing operations are performed many times and have therefore been optimized much more heavily and most of the other linear algebra, like constructing the coarse matrices once the prolongations are in place, already come very well optimized out of NGSolve.

Ultimately however, we still face the issue that allowing cross-eqc collapses, as these always introduce coarse nodes "upwards" in the eqc-hierarchy, lead to an increase in the fraction of shared DOFs versus local DOFs on each level. Also, the restrictions we had to put in place to counter uncontrolled growth of proc interfaces will eventually, after many levels, prevent us from maintaining a constant decrease by a factor 2 from level to level we would like to have. The problem is much, much less pronounced than what we saw in figure 8.7, and does usually not become debilitating until the coarse spaces reach a dimension in the range of under 100 DOFs per proc, but it does still occur. It also becomes more and more of an issue the further we try to scale up in terms of number of procs.

In the next section we will outline what has been done to combat both of these problems.

8.2.2 Contracting

On very caorse levels, as the number of DOFs per proc becomes small, communication overhead starts playing a bigger and bigger role compared to computation cost. This is especially true when we are doing sparse matrix vector operations which have linear operator complexity. To reiterate the conclusion from the last section, this is exacerbated by the tendency of the caorsening algorithm to make DOFS on caorser levels more and more "global" and by the fact that the restrictions to topologically admissible algebraic edges become more noticeable on coarse levels.

The solution to all of these problems is as straightforward to formulate as messy to implement. We can simply redistribute the entire problem *to fewer procs* after reaching certain breakpoints.This might seem a bit unintuitive on first glance, as we are essentially giving awas computing power, however, if done correctly, what we loose is more than made up by the decreased communication costs.

Redistributing to fewer procs everytime makes some additional edges topologically admissible. Such a redistribution can be wonderfully formalized with the notation developed in section 8.2.1 and yields an unorthodox application of algorithm 4.

Definition 8.12: Contraction Map

We define the weighted-proc-vertex-set as

$$\mathcal{V}^p := \{(p,0,0) : p \in \mathcal{P}\}$$

With proc-edge-collapse-weights

$$w^p_{ij} := \frac{|I^{\mathcal{P}}_{\{i,j\}}|}{|I^{\mathcal{P}}_{\{i\}}| + |I^{\mathcal{P}}_{\{i,j\}}| + |I^{\mathcal{P}}_{\{j\}}|}$$

and the weighted-proc-edge-set as

$$\mathcal{E}^p := \{(\{k,l\}, w^p_{kl}, 0) : w^p_{kl} \neq 0\}$$

The (global) algebraic proc-mesh is

$$\mathcal{M}^p := (\mathcal{V}^p, \mathcal{E}^p)$$

The partition $\mathcal{C}^p := CAM\_SEQ(\mathcal{M}^p, \emptyset, \emptyset, TRUE) \setminus \{\emptyset\}$ *induces a "Proc-Contraction-Map" that takes ranks and delivers coarse ranks*

$$\mathbf{g}^p : \begin{cases} \mathcal{P} \mapsto \mathcal{P}^c := \{0 \ldots |\mathcal{C}^p| - 1\} \\ i \to j \quad \text{for } i \in C_j \in \mathcal{C} \end{cases}$$

$\mathbf{g}^p$ *can also be extended to mapping vertex-EQCs to corresponding coarse EQCs that define a coarse equivalence relation* $\sim_2$ *on* $\mathcal{N}$*.*

$$\mathbf{g}^p : \begin{cases} \mathcal{N}/_{\sim} \mapsto \mathcal{N}/_{\sim_2} \\ [\cdot]^v_J \to [\cdot]^v_{\mathbf{g}^p(J)} \end{cases}$$

Redistributing, or, as we call it, contracting the problem is now just:

- Constructing the contraction map
- Collecting the local algebraic meshes $\mathcal{M}^i$ as well as the local matrices $\mathbf{A}^i$, $\hat{\mathbf{A}}^i$ and $\mathcal{P}^i$ on proc $\mathbf{g}^P(i)$.
- On the master, combine the collected algebraic meshes, matrices, etc. into new, "coarse" objects.
- The master ranks $k \in \mathcal{P}^c$ continue, while all others are done with the setup.

For DOFs, if $k \in [\cdot]^v_J$, then per construction $k \in [\cdot]^c_{\mathbf{g}^p(J)}$, we say DOF-EQCs are invariant under contraction. The DOF-proc sets on the "coarse" space (that is, in this context, the same space but distributed to fewer procs) become smaller.
When collapsing P^k with P^j, all of the dofs in the face-eqc $[\cdot]^v_{I^{\mathcal{P}}_{\{i,j\}}}$, which are shared between these two exclusively, become new, local dofs in the coarse vol-eqc $[\cdot]^v_{\{\mathbf{g}^p(i)\}}$. Besides that, any other eqc $[\cdot]_J$ with $\{i,j\} \subseteq J$ loses at least one proc, so it becomes "less parallel", some wire-eqcs can even become face-eqcs. This is the reason why the proc-edge-collapse-weights are defined the way they are.

Note 8.22. After a contraction, the problem has been redistributed to fewer cores, and on all the following levels we have to take into account that only certain procs are active anymore. On the surface, this would require the coarsening algorithm and the smoothers to constantly be aware of which procs are active and which are not. In practice, however, MPI does all of that work for us. We can simply construct a new communicator that consists of all the active procs and no others. Therefore neither the coarsening algorithm nor the smoothers need to be aware of contractions at all. We do however have to take care of gathering vector entries when going up in the v-cycle and scattering them when going down.

Note 8.23. The Edge-EQCs are not invariant under the contraction map, they have to be rebuilt after the vertices have been mapped. To see this, consider an edge between vertices with proc-sets $\{1,2,3\}$ and $\{2,3,4\}$, where $\mathbf{g}^p$ that maps $1 \to 1; 2 \to 2; 3 \to 3; 4 \to 1$. The fine edge is in $[\cdot]^e_{\{2,3\}}$, but the coarse one is in $[\cdot]^e_{\{1,2,3\}}$

Note 8.24. One thing to keep in mind here is that contracting is a narrow road to walk. Excessive, or simply too early contracting can hinder more than help. Whenever we contract, we approximately double the DOFs that are local to each proc and therefore increase the local operator complexity of the multigrid cycle. The total operator complexity remains unchanged, but a part of the work is shared between fewer procs and in the end what we really care about is wall time.
Additionaly, while we only have to move data around once when contracting the algebraic mesh and the matrices, during the solution phase we have to move data everytime we cross a “contraction level” on the way up or down in the v-cycle. The communication necessary for that is however only responsible for collecting the partial vectors of all procs at the master proc of each $C \in \mathcal{C}^p$ when going up or distributing the data from the master to the other procs when going down, which is all in all about equivalent to one parallel vector cumulate operation.

Note 8.25. We have not yet experimented with allowing bigger proc-agglomerates in the contraction-map. It might or might not be more efficient, one should keep in mind that contracting procs by a factor k increases the local DOF-number on the coarse level by the same factor.

The exact algorithm that decides on which levels and under which conditions we should do a contraction and when we should not do one is still up to debate, currently each proc says it wants to contract if one of the following three conditions holds:

- If there is a very attractive partner for collapse
$$\max_{j \in \mathcal{P}} w^p_{ij} > 0.2$$
- If the local NDOF is very small:
$$|\mathcal{N}^k_\ell| < 100$$
The thought behind this condition is that for so few DOFs, any communication at all is more expensive than computation.
- If the local vol-eqc DOFs make up too small a fraction of all local ones:
$$\frac{|[\cdot]^v_{\{k\}}|}{\mathcal{N}^k_\ell} \leq 0.2$$
The rationale here is that if there are too few local vol-eqc DOFs, that is if almost all DOFs are shared with other procs anyways, what is then point of this proc even running anymore. Again, the small amount of computation done on the local vol-DOFs is far outweighed by the costly computation on all the other dofs.

On each level, if more than a third of all procs want to do a collapse, we do one, otherwise we do not.

This relatively simple approach seems to work well enough for now, although it does not take into account the distance between procs in the communication network of the cluster. In principle it would probably be preferrable to choose the partition in such a way that the minimum number of nodes are occupied, but this is a feature that AscAMG currently simply does not support.

Note 8.26. For me personally, implementing the contracting as well as the mappings needed for the multigrid-cycle that map vectors to contracted vectors and the other way around was probably the least rewarding part of the entire project. While the DOF-EQC formalism describes the contraction very nicely, implementation was a lot of work for the speedup it resulted in on the medium size systems we had access to. However as figure 8.7 shows quite convincingly, it is a crucial component for further scalability.

8.2.3 Distributed Smoothed Prolongation and Optimizing Operator Complexity

We will now give the answer to the question how good V-cycle operator complexity can be achieved despite the small agglomerates we use for building the piecewise prolongation. Besides the idea of using different prolongations to transport the system matrix $\mathbf{A}$ and the replacement matrix $\hat{\mathbf{A}}$ (and with it the algebraic mesh) to the coarse level, there are two more components to our strategy.
We will first show how to properly define and construct a distributed parallel version of the smoothed prolongation from definition 8.3.

Hierarchic Prolongation

While we have seen that the local piecewise prolongations $\mathbf{P}^k$ form a kind of cumulated parallel vector and can be used to transport cumulated or distributed parallel vectors up or down between levels without communicating at all, it is not really clear how to compute local $\mathbf{P}^k_s$ that could represent $\mathbf{P}_s$. Even worse, it is usually impossible to compute local $\mathbf{A}^k_C$ that represent the coarse system without doing some serious redistribution of the problem. This fact becomes clear when we remember what we observed in section 8.1; The matrix $\mathbf{A}_2 \coloneqq (\mathbf{I} - \omega\hat{\mathbf{D}}^{-1}\hat{\mathbf{A}})^T\mathbf{A}(\mathbf{I} - \omega\hat{\mathbf{D}}^{-1}\hat{\mathbf{A}})$ has nonzero entries corresponding to each path of length three or less in the matrix graph of $\mathbf{A}$. In particular, there are connections between DOFs $i \in [\cdot]^v_I$ and $j \in [\cdot]^v_J$ where $I \cap J = \emptyset$. This means that for two procs $P^k \in I$ and $\mathcal{P}^j \in J$, there is an an entry of $\mathbf{A}_2$ that connects a DOF of P^k, which P^l does not know exists, to a dof of P^l, which P^k does not know about. In other words, in order to be able to write this entry in some local component of a prallel matrix, we have to eiter put i into $\mathcal{N}^l_\ell$ or j into $\mathcal{N}^k_\ell$, which either way makes the proc-set of one of the DOFs larger, or in other words, makes its' equivalence class "larger" in the eqc-hierarchy. There is no representation of the global matrix $\mathbf{A}_2$ by local components without changing the $\mathcal{N}_\ell$. This means that the coarse matrix $\mathbf{A}_C = \mathbf{P}^T\mathbf{A}_2\mathbf{P}$ does not have a representation by local components that fit the dof-sets of the coarse algebraic mesh. In addition to all of this, $\mathbf{P}^T\mathbf{A}_2\mathbf{P}$ can have entries that connect procs that were not in direct contact on the fine level. We took great pains to avoid such things happening when constructing the coarsening algorithm, and the same considerations that made us do it there also hold up here.

Note 8.27. For the system matrix $\mathbf{A}$ and the coarse system matrix $\mathbf{A}_C$, we use the standard parallel sparse matrices coming from NGSolve. As we know by now, these matrices store the information on the parallel structure of the DOFs in ParallelDofs-objects. As long as the distant procs on the coarse level are the same as, or a subset of, the distant procs on the fine level, it is quite easy to construct the appropriate ParallelDofs for the coarse matrix, however if the coarse ones are a true superset of the fine ones, this becomes messy as well.

Notation 8.9. To differentiate between EQCS on the fine and coarse level, we will write ${}^f[\cdot]$ and ${}^c[\cdot]$ respectively. To differentiate between local-to-global maps we will write ${}^f\mathbf{g}^k$ and ${}^c\mathbf{g}^k$, the global-to-local maps will be ${}^f\boldsymbol{\ell}^k$ and ${}^c\boldsymbol{\ell}^k$. The DOF-Sets will be ${}^f\mathcal{N}_\ell{}^k$ and ${}^c\mathcal{N}_\ell{}^k$, etc. We can extend the definition of $\lhd$ to accept a coarse and a fine, or a fine and a coarse EQC in the canonical way.

We will now show a condition for $\mathbf{P}_s$ that guarantees the existence of a local representation for the global coarse system matrix.

Theorem 8.3: Hierarchic Prolongation

Given a partition $\mathcal{C}$ with $|\mathcal{C}| = n_c$ that induces a piecewise prolongation, if some $\mathbf{P} \in \mathbb{R}^{n \times n_c}$ fulfills

$$\mathbf{P}_{ij} \neq 0 \quad \text{for } i \in C_j \in \mathcal{C} \text{ only if } {}^f[i]^v \lhd {}^c[j]^v \tag{8.15}$$

and there exist local matrices $\mathbf{P}^k$ on all procs such that

$$\mathbf{P}^k_{ij} = \mathbf{P}_{\mathbf{g}^k(i)\mathbf{g}^k(j)} \quad \forall k \in \mathcal{P}\ \forall i,j \in \mathcal{N}^k_\ell$$

that is, the $\mathbf{P}^k$ form a (cumulated) local representation of $\mathbf{P}$, we will call $\mathbf{P}$ hierarchic.
If $\mathbf{P}$ is hierarchic, then given a cumulated parallel vector $\mathbf{v} \cong (\mathbf{v}^i)_i$ on the coarse level, $\mathbf{P}\mathbf{v} \cong (\mathbf{P}^i\mathbf{v}^i)_{i\in\mathcal{P}}$ is a cumulated (fine) vector.
Conversely, given a distributed vector $\mathbf{v} \cong (\mathbf{v}^i)_i$ on the fine level, if $\mathbf{P}$ is hierarchic, $\mathbf{P}^T\mathbf{v} \cong (\mathbf{P}^{i,T}\mathbf{v}^i)_{i\in\mathcal{P}}$ is a distributed (coarse) vector.
The coarse matrix of $\mathbf{A}_C \cong (\mathbf{A}^k_C)_{k\in\mathcal{P}}$ defined by $\mathbf{A}_C = \mathbf{P}^T\mathbf{A}\mathbf{P}$ is represented by

$$\mathbf{A}_C \cong ((\mathbf{P}^k)^T\mathbf{A}^k\mathbf{P}^k)_{k\in\mathcal{P}}$$

Proof. We will show

$$\mathbf{P}\mathbf{v} \cong (\mathbf{P}\mathbf{v}^i)_i$$

that is

$$[\mathbf{P}\mathbf{v}]_{{}^f\mathbf{g}^k(i)} = \left[\mathbf{P}^k\mathbf{v}^k\right]_i \quad \forall i \in \mathcal{N}^k_\ell\ \forall k \in \mathcal{P}$$

For $i \in {}^c\mathcal{N}_\ell{}^k$ and $j \in {}^c\mathcal{N}$

$${}^f[{}^fg^k(i)]^v \lhd {}^c[j]^v \Rightarrow k \in {}^fI^{\mathcal{P}}_{{}^fg^k(i)} \subseteq {}^cI^{\mathcal{P}}_j \Rightarrow j \in {}^c\mathcal{N}^k_\ell$$

From this, and (8.15), it follows that we can replace the sum over ${}^c\mathcal{N}_\ell{}^k$ by the sum over ${}^c\mathcal{N}$ in $(*)$:

$$\left[\mathbf{P}^k\mathbf{v}^k\right]_i = \sum_{j\in{}^c\mathcal{N}_\ell{}^k} \mathbf{P}^k_{ij}\mathbf{v}^k_j = \sum_{j\in{}^c\mathcal{N}_\ell{}^k} \mathbf{P}_{{}^f g^k(i)\,{}^c g^k(j)}\mathbf{v}_{{}^c g^k(j)} =$$

$$\overset{(*)}{=} \sum_{j\in{}^c\mathcal{N}} \mathbf{P}_{{}^f g^k(i) j}\mathbf{v}_j = [\mathbf{P}\mathbf{v}]_{{}^f\mathbf{g}^k(i)}$$

The second claim, $\mathbf{P}^T\mathbf{v} \cong (\mathbf{P}^T\mathbf{v}^i)_i$, follows in the same way.
The last claim naturally follows from the first two: Given any cumulated vector $\mathbf{v} \cong (\mathbf{v}^k)_{k\in\mathcal{P}}$ on the coarse level, we have to show that $((\mathbf{P}^k)^T\mathbf{A}^k\mathbf{P}^k\mathbf{v}^k)_{k\in\mathcal{P}}$ is a distributed vector such that $\mathbf{A}_C\mathbf{v} \cong ((\mathbf{P}^k)^T\mathbf{A}^k\mathbf{P}^k\mathbf{v}^k)_{k\in\mathcal{P}}$. From the first two points we know that component-wise multiplication of $\mathbf{P}$ with a coarse cumulated vector yields a fine cumulated vector and $\mathcal{P}\mathbf{v} \cong (\mathbf{P}^k\mathbf{v}^k)_{k\in\mathcal{P}}$. Because $\mathbf{A}$ is a standard distributed parallel matrix, component-wise multiplication of $\mathbf{A}$ with a fine cumulated vector per definition yields a fine cumulated vector and $\mathbf{A}\mathbf{P}\mathbf{v} \cong (\mathbf{A}^k\mathbf{P}^k\mathbf{v}^k)_{k\in\mathcal{P}}$. Lastly, as we also know, component-wise multiplication of $\mathbf{P}^T$ with a fine distributed vector yields a coarse, distributed one and finally $\mathbf{P}^T\mathbf{A}\mathbf{P}\mathbf{v} \cong ((\mathbf{P}^k)^T\mathbf{A}^k\mathbf{P}^k\mathbf{v}^k)_{k\in\mathcal{P}}$. □

Note 8.28. Equation 8.15 alone guarantees the existence of local representations as defined in theorem 8.3, as if

$$\mathbf{P}_{ij} \neq 0 \Rightarrow {}^f[i]^v \lhd {}^c[j]^v \Rightarrow \exists k \in I_i^{\mathcal{P}} \cap I_j^{\mathcal{P}}$$

Which means that the value $\mathbf{P}_{ij}$ can be represented locally by proc k.

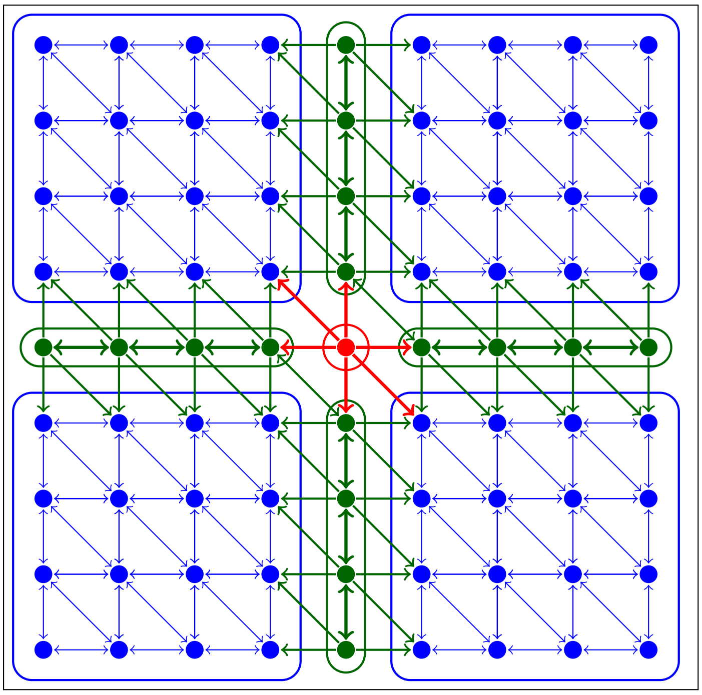


Figure 8.9: Under a hierarchic prolongation, information from the coarse level is transported “downwards” in the EQC-hierarchy. The graphic shows the maximal allowed matrix graph a matrix **B** (on the fine level), for the same case as figure 8.5, is allowed to have such that, given a piecewise prolongaion **P**, **BP** is hierarchic prolongation.

Hierarchic prolongations have two very nice features, firstly, the local coarse system matrix components are exactly the local coarse matrices of the local system matrix components and we can compute $\mathbf{A}_C^k = (\mathbf{P}^k)^T \mathbf{A}^k \mathbf{P}^k$ locally without required communication. Even better, we have shown that we can transport distributed vectors upwards across levels and cumulated vectors downwards by purely local matrix-vector multiplication without any needed communication. As we typically want to restrict (distributed) fine residuals to the coarse level and prolong (cumulated) coarse solutions to the fine level, this is a perfect fit.

We will now modify the smoothed prolongation $\mathbf{P}_s$ in such a way that it becomes a hierarchic prolongation. We will also set a limit for the nonzero entries per row of the new, hierarchic prolongation, usually by 3 or 4, by simply removing all but the strongest entries from $\hat{\mathbf{A}}$. In the resistor network this means that we only consider the connections with the highest conductivity when solving the local nodal problem to get the fine potential from the coarse one.

Definition 8.13: Hierarchic Smoothed Prolongation

Given the replacement matrix $\hat{\mathbf{A}}$, *an upper bound* M *for the nonzero entries in the prolongation matrix, a piecewise prolongation matrix* $\mathbf{P}$ *induced by a partition* $\mathcal{C}$ *and its induced fine-to-coarse DOF-map* $\mathbf{crs}$ *and for each DOF* $i \in \mathcal{N}$, *let the set of admissible DOFs be defined as* $\mathcal{A}_i \coloneqq \{j : j \notin D_{\mathcal{C}} \wedge \alpha_{ij} > \sigma \wedge {}^f[\mathbf{i}]^v \lhd {}^c[\mathbf{crs}(j)]^v\}$. *Next, let* $\widetilde{\mathcal{S}}_i$ *be the maximal subset of* $\mathcal{A}_i$ *consisting of the* k *dofs in* $\mathcal{A}_i$ *that have the highest weight such that* $|\mathbf{crs}(\widetilde{\mathcal{S}}_i)| \leq M$. *Define* $\mathcal{S}_i \coloneqq (\mathbf{crs}^{-1} \circ \mathbf{crs})(\widetilde{\mathcal{S}}_i)$, *that is the set of all dofs that are mapped to the same coarse DOF as some DOF in* $\widetilde{\mathcal{S}}_i$. *The sparsity- and hierarchy-filtered replacement matrix* $\hat{\mathbf{A}}^\circ \cong (\hat{\mathbf{A}}^{k,\circ})_{k\in\mathcal{P}}$ *is defined by*

$$\hat{\mathbf{A}}^{k,\circ}_{ij} = \begin{cases} \sum_{l\in((\mathbf{crs}\circ\mathbf{g}^k)^{-1}(\mathcal{S}_i))} \alpha_{il} & \textit{if } i = j \\ -\alpha_{ij} & \textit{if } (\mathbf{crs}\circ\mathbf{g}^k)(j)) \in \mathcal{S}_i \\ 0 & \textit{else} \end{cases}$$

Finally, the AscAMG hierarchic smoothed prolongation is defined as

$$\mathbf{P}_h \coloneqq (\mathbf{I} - (\mathbf{D}^\circ)^{-1}\mathbf{A}^\circ)\mathbf{P} \tag{8.16}$$

Note 8.29. By the last theorem and the definition of $\mathcal{A}_i$, $\mathbf{P}_h$ is a hierarchic prolongation that admits local representation.

Note 8.30. The matrix $\hat{\mathbf{A}}^{k,\circ}$ itself is not necessarily symmetric anymore. See also figure 8.9, where we have a directed graph, not an undirected one.

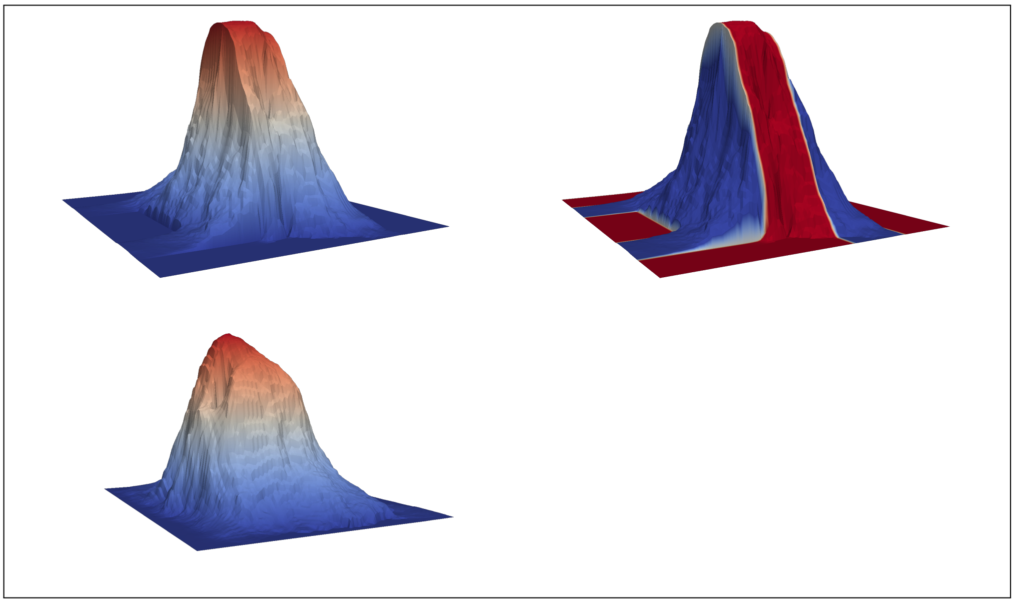


Figure 8.10: Basis functions of coarse level 12 resulting from hierarchic smoothed prolongation for the same problem as in figure 5.3. Nonzero entries per row of all prolongations bounded by 3. In the top two pictures, alpha jumps between 1 and 10^6 and in the bottom alpha is constant. In the top right picture, the color indices the value of α (red is higher).

Alternating Prolongations and Skipping Levels

The hierarchic smoothed prolongation on its own still underperforms. Especially the smoothing of the prolongation between levels 1 and 0 increases the operator complexity by a lot.

There is one, last, component missing that gives us another considerable boost in performance. What we informally call "alternating composite prolongation schemes" are just Multigrid V-cycles with two twists to them.

First of all, we do not use the same kind of prolongation on all levels. One level of piecewise prolongation might be followed by two levels of smoothed prolongation, follwed by another level of piecewise prolongation, etc. This is where the "alternating" comes from.

The "composite" comes from the fact that we do not necessarily need to smooth on each level. In most AMG solvers, the difference in NDOFs between to levels is much larger than a factor two, in fact, some of them start with an initial, massive coarsening step between levels 0 and 1, instantly going down in NDOF by a factor 10 or so.

We therefore choose not to include all levels in the V-cycle. If we decide to skip level k, we can just compose the prolongation that goes from level $k-1$ to k and the one going from $k-1$ to $k+1$ by multiplying them and directly build the coarse matrix on level $k+1$ from the one on level $k-1$.

A (PPH)-(NNB)-scheme, for example, would use (P)iecewise, (P)iecewise, (H)ierarchic-smoothed prolongations in this recurring order and always skip two levels and directly go to the third (the matrices are (N)ot built, (N)ot built and (B)uilt). In practice it has proven to be most efficient to arrange the scheme such that each level directly after an H-prolongation is built and those after P-prols are skipped.

Overall, the best choice seems to be a prefixed (PH)-(NB)-scheme. Prefixed here means that for the first couple coarse levels we do something special, usually involving skipping a few extra steps and then appending the (PH)-(NB)-scheme after that. We write such a scheme, for example as (PPHPPH)-(NNBNNB)//(PH)-(NB), which stands for two skipped P-prol levels, followed by a built H-level, another two skipped P-levels and a last built H-level and staring the (PH)-(NB)-scheme after that, on coarse level 7.

As we will se in chapter 9, after all of these optimizations, operator complexity is finally under control.

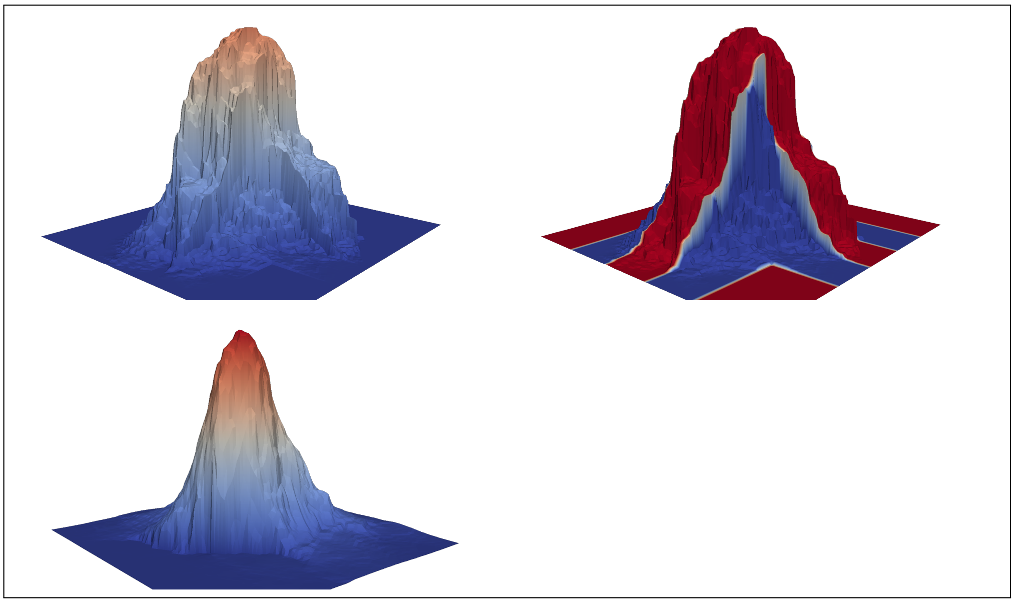


Figure 8.11: Showing the exact same situation as in figure 8.10, however using a (PH)-scheme.

8.2.4 smoothers

In this last section of the chapter, we will discuss the (small) range of smoothers that come with AscAMG . By small, we really mean only two, and one of them, the distributed Gauss Seidel smoother, turned to be extremely, extremely difficult to implement efficiently, in fact we would not recommend using it for problems running on more than about 500 procs. Up to that point, however, it perforems pretty well. Besides, it is an interesting piece of software that will also give us another opportunity to use the DOF-eqc terminology developed in section 8.1 so we will describe it briefly despite its' shortcomings. In [2], multiple viable, scalable options for multigrid smoothers are presented and analyzed, including the ℓ^1-hybrid Gauss Seidel smoother, which is the other smoother available in AscAMG .

Hybrid Gauss Seidel

Definition 8.14: ℓ^1-Hybrid Gauss Seidel Smoother

For $k \in \mathcal{P}$, let J^k be its set of master DOFs,

$$J^k := \{j \in \mathcal{N}^k : k = \min(I_j^{\mathcal{P}})\}$$

Let $\mathbf{A}_{J^k} \in \mathbb{R}^{|J^k| \times |J^k|}$ be the submatrix of $\mathbf{A}$ for indices in J^k. For each $k \in \mathcal{P}$, let the diagonal matrix $\widetilde{\mathbf{D}}^k = diag(d_1, d_2, \ldots, d_{|J^k|)}) \in \mathbb{R}^{|J^k| \times |J^k|}$ be defined by its's entries

$$d_i = \sum_{j \in \mathcal{N}^k \setminus J^k} |\mathbf{A}_{ij}|$$

The ℓ^1-Hybrid Gauss Seidel Smoother is the SLIM defined by the block-diagonal $\mathbf{W} \cong (\mathbf{W}^k)_{k \in \mathcal{P}}$, where the $\mathbf{W}^k$ are the purely matrices

$$\mathbf{W}^k := \mathbf{L}_k + \mathbf{D}_k + \frac{1}{2}\widetilde{\mathbf{D}}_k$$

Note 8.31. As can be seen easily, per construction of $\widetilde{\mathbf{D}}$, we have $\mathbf{W}_H + \mathbf{W}_H^T - \mathbf{A} > 0$

Note 8.32. Although it is a bit of a misnomer, this smoother will simply be referred to as HGSS.

This is just the application of local Gauss-Seidel on every proc, except that we have a modified diagonal and that we are only doing this on the submatrix corresponding to the master-DOFs. This is easily implemented, and communication-wise the only thing we need to do is to cumulate the solution vector after the local Gauss-Seidel sweeps have finished. For more details on hybrid smoothers and see [2].

Distributed Gauss Seidel

What is internally reffered to as DGSS is a distributed parallel implementation of standard Gauss Seidel. It is based on a particular, semi-manual coloring of the DOFs that allows it to hide much of the communication overhead incurred due to the serial nature of Gauss Seidel behind local computations.

Definition 8.15

We will call $\mathcal{I} \coloneqq \{i \in \mathcal{N} : |[i]^v| = 1\}$ the set of all vol-DOFs and define the sets of all face-DOFs $\mathcal{F}$ and all wire-DOFs $\mathcal{W}$ accordingly as $\mathcal{F} \coloneqq \{i \in \mathcal{N} : |[i]^v| = 2\}$ and $\mathcal{W} \coloneqq \{i \in \mathcal{N} : |[i]^v| \geq 3\}$.
The set $\mathring{\mathcal{F}}$ of all face-interior DOFs will be

$$\mathring{\mathcal{F}} \coloneqq \{k \in \mathcal{F} : \mathbf{A}_{kj} = 0 \quad \forall j \in \mathcal{F} \setminus [k]^v\}$$

The matrix $\mathbf{A}$ restricted to the indices in $\mathring{\mathcal{F}}$ is block-diagonal, each diagonal block corresponds to the "interior" of one vol-eqc. Similarly, the set $\mathring{\mathcal{W}}$ is defined by

$$\mathring{\mathcal{W}} \coloneqq \{k \in \mathcal{W} : \mathbf{A}_{kj} = 0 \quad \forall j \in \mathcal{W} \setminus [k]^v\}$$

Doing the same for $\mathcal{I}$ would be pointless, no vol-DOFs can be connected with a vol-DOF of another proc in the matrix graph of $\mathbf{A}$. Instead, we define the set of "absolutely local" DOFs as

$$\mathring{\mathcal{I}} \coloneqq \{k \in \mathcal{I} : \mathbf{A}_{kl} = 0 \quad \forall l \in \mathcal{N} \ |[l]^c| > 0\}$$

This is the set of all interior DOFs that are at least one "DOF-layer" removed from any interface. The set $\mathcal{M} \coloneqq \mathcal{I} \setminus \mathring{\mathcal{I}}$, which consists exactly of this connecting "DOF-layer", will be called the mortar-DOF set.
All of the remaining DOFs will be called "type-C" dofs and the set of all type-C DOFs is

$$\mathcal{C} \coloneqq (\mathcal{F} \setminus \mathring{\mathcal{F}}) \cup (\mathcal{W} \setminus \mathring{\mathcal{W}})$$

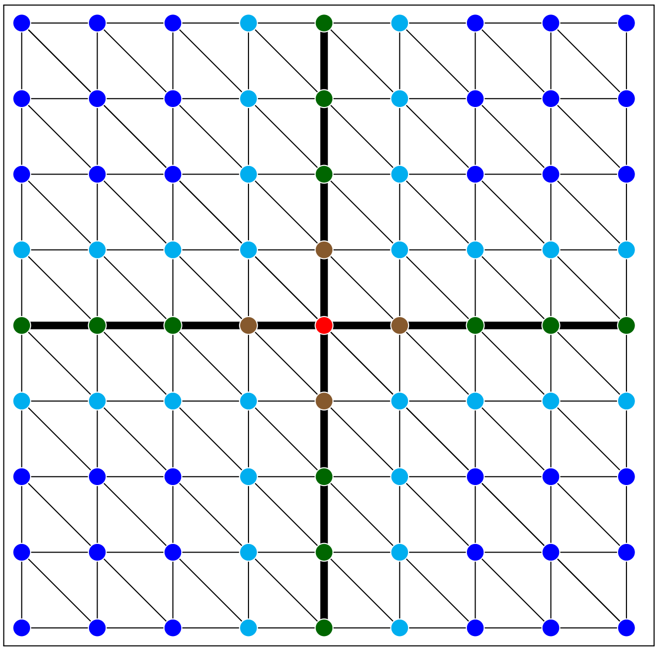


Figure 8.12: Classification of DOFs for the same case as in figure 8.5. dark blue:$\mathring{\mathcal{I}}$, light blue:$\mathcal{M}$, green:$\mathring{\mathcal{F}}$, red:$\mathring{\mathcal{W}}$, brown:$\mathcal{C}$

The two key observations behind any attempt to parallelize Gauss-Seidel are that firstly, the exact order in which we update the residuals and solution vectors is not actually important, as long as we do it one DOF at a time. This means that we can permute $\mathbf{A}$ and apply Gauss-Seidel to the permuted Matrix. Let us take permutation such that first take all DOFs in $\mathring{\mathcal{I}}$, then all in $\mathcal{I}$, ctc. The order of sets will be $\mathring{\mathcal{I}}, \mathcal{M}, \mathcal{F}, \mathcal{W}, \mathcal{C}$. The permuted matrix now looks like this

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{\mathring{\mathcal{I}}} & * & 0 & 0 & 0 \\ * & \mathbf{A}_{\mathcal{M}} & * & * & * \\ 0 & * & \mathbf{A}_{\mathcal{F}} & * & * \\ 0 & * & * & \mathbf{A}_{\mathcal{W}} & * \\ 0 & * & * & * & \mathbf{A}_{\mathcal{C}} \end{pmatrix}$$

The second observation is that we can update DOFs i and j at the same time if $A_{ij} = 0$. On first glance, permuting $\mathbf{A}$ has not really put us a large step forward. Splitting of the mortar-DOFs from the absolutely interior DOFs allows us to update $\mathring{\mathcal{I}}$ at the same time as $\mathring{\mathcal{F}}, \mathring{\mathcal{W}}, \mathcal{C}$, but $\mathring{\mathcal{F}}, \mathring{\mathcal{W}}$ and $\mathcal{C}$ are connected among each other.

The submatrices $\mathbf{A}_{\mathring{\mathcal{I}}}$, $\mathbf{A}_{\mathcal{M}}$, $\mathbf{A}_{\mathring{\mathcal{F}}}$ and $\mathbf{A}_{\mathring{\mathcal{W}}}$ are all block-diagonal.

This is clear for $\mathbf{A}_{\mathring{\mathcal{I}}}$ and $\mathbf{A}_{\mathcal{M}}$ as these are vol-EQC DOFs that can never connect to vol-EQC DOFs on other procs, so the subblocks for each domain are not connected to each other. In a way tere are type A-DOFs, which were always “easy” to parallelize to begin with.

$\mathbf{A}_{\mathring{\mathcal{F}}}$ and $\mathbf{A}_{\mathring{\mathcal{W}}}$ are block diagonal by design, in the definition of $\mathring{\mathcal{F}}$ and $\mathring{\mathcal{W}}$ we threw out all DOFs that connect face-eqcs to face-eqcs or wire-eqcs to wire-eqcs. These are, in a way, type B-DOFs, they were not parallelizable to begin with, but with a bit of work, and by removing all troublesome DOFs, we are still able to find a way.

This brings us to the very troublesome type C-DOFs, which are, unfortunately, basically unparallelizable. They do, however, at least on the finer levels, make up the vast minority of all DOFs.

Let us first formulate the distributed Gauss Seidel algorithm for the rest of the DOFs and discuss what to do about the C-DOFs afterwards.

In figure 8.13 we can see the the basic flow chart of the DGSS algorithm. First, we are smoothing the $\mathcal{M}$-DOFs, these are $\mathcal{I}$-DOFs, so we can compute the residuals and update the solution vectors locally. After that, the $\mathring{\mathcal{I}}$ dofs can be updated at the same time as the $\mathring{\mathcal{F}}$-, $\mathring{\mathcal{W}}$- and $\mathcal{C}$-DOFs.

Updating the solution on the $\mathring{\mathcal{F}}$ and $\mathring{\mathcal{W}}$-dofs requires first local computation of partial residuals (which can be thought of as a partial distributed parallel vector), then a communication step to cumulate these values and finally, once the full residual values are present, a local update to the solution vector.

Updating the $\mathring{\mathcal{I}}$-DOFs can again be done completely locally.We try to hide as much of the time that is spent on waiting for messages on the $\mathring{\mathcal{W}}$, $\mathring{\mathcal{F}}$, and $\mathcal{C}$-parts with local computation as possible. Overlapping communication and computation manually can be tricky. One has to very carefully split up the local work into smaller chunks and whenever one is waiting for a message work off a couple of them. We use more of a brute force approach.

On construction of the first DistributedGaussSeidelSmoother-object, a thread is created that immediately goes to sleep. Wenever we do a Gauss-Seidel sweep, this thread is woken up and told to to the $\mathring{\mathcal{I}}$-portion of the work. The main proc itself meanwhile keeps on working on the $\mathring{\mathcal{F}}$, $\mathring{\mathcal{W}}$, and $\mathcal{C}$-portions. Each proc only creates one thread for all DGSS-objects (in particular, only one for all levels!). Depending on the MPI-libeary in use, the way it was compiled and the machine we are running on, this sometimes works well and sometimes not so much.

We are left with discussing what can be done about the $\mathcal{C}$-dofs. So far, there were two differnt attempts to master this problem. One is to just redistribute the entire matrix block $\mathbf{A}_{\mathcal{C}}$ to the master proc P^0. It has been mentioned when talking about parallel meshes in NGSolve that the master proc does not posess a subdomain, which means that most of the time, it has nothing to do. From that point of view, and if also considering that usually $\mathcal{C}$ should only contain very few DOFs, this appears to be a good idea. This is however not always the case on the coarse levels, as the coarsening algorithm tends to make DOFs more and more global as the number of levels increases. Nonetheless, this works pretty well up to around 500 cores or so, provided $\mathcal{C}$ stays small. After that, problems start to arise even if $\mathcal{C}$ is small. Every time we have to smooth the $\mathcal{C}$-DOFs, we have to send the residual data for all $\mathcal{C}$-DOFs, which come from all over the computational domain, to the master proc. This is basically a global Gather-operatioon, which, as it has to be done for every Gauss-Seidel sweep, becomes prohobitively expensive when many procs are involved.

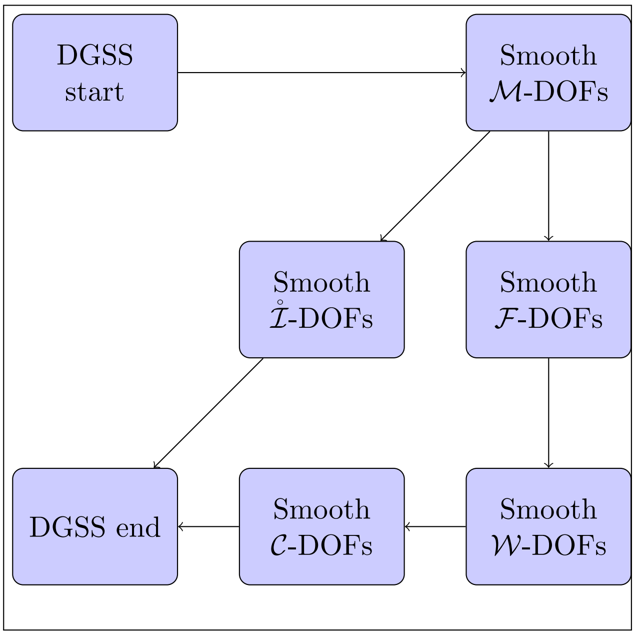

DGSS start
Smooth $\mathcal{M}$-DOFs
Smooth $\mathring{\mathcal{I}}$-DOFs
Smooth $\mathcal{F}$-DOFs
DGSS end
Smooth $\mathcal{C}$-DOFs
Smooth $\mathcal{W}$-DOFs

Figure 8.13: Simple FlowChart for the DGSS algorithm.

The other approad was to divide the $\mathcal{C}$ dofs back into their EQC-blocks and then work through one block after another. For that, a global $\mathcal{C}$-EQC-block-graph, consisting of a node for each block, and an edge e_{ij} whenever a DOF in block i is connected to a DOF in block j in the matrix graph of $\mathbf{A}$ is cunstructed. Then, a coloring for this graph is found. All blocks of the same color can be updated at the same time, after which residuals have to be updated and cumulated before one can proceed with the next color. This corresponds to construcing a directed acyclic graph on the $\mathcal{C}$-EQC-block-graph.

Unfortunately, every color in the graph "costs" one round of communication, and as the $\mathcal{C}$-EQC-block-graph becomes denser and denser on coarse levels, this is just too costly.

Currently a mixture of the two is employed. We take all $\mathcal{C}$-dofs in the largest M colors and go through them block-wise and gather the rest on the master-proc. The parameter M can me modified.

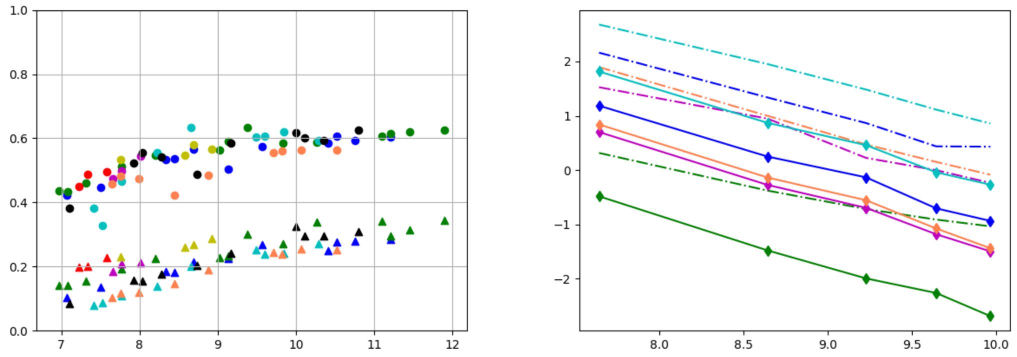
Ultimately, the distributed Gauss Seidel algorith performs acceptably well when the number of cores does not get too large and can perform very good in circumstances where a lot of local computation has to be done which hides some of the latencies. In terms of scalability however, it does not even come close to the ℓ_1 hybrid Gauss Seidel smoother.

Note 8.33. The created thread is a C++-11 thread that is notified via a condition_variable when it has work to do.

9 Performance and Benchmarks

All results in this section have been obtained on the COEUS clutster at Portland State University, thanks to the support by Professor Jay Gopalakrishnan and the Portland Institute for Computational Science and its resources acquired using NSF Grant DMS 1624776.

Smoother Performance



a: x-axis: $\ln(\frac{n}{n_p})$. y-axis: $\frac{t_{\text{mult}}}{t_{\text{smooth}}}$. triangles are DGSS. Colors are the number of procs: red:20, yellow:40, pirple:100, green:200, blue:400, black:600, orange:800 and turquoise:1000.

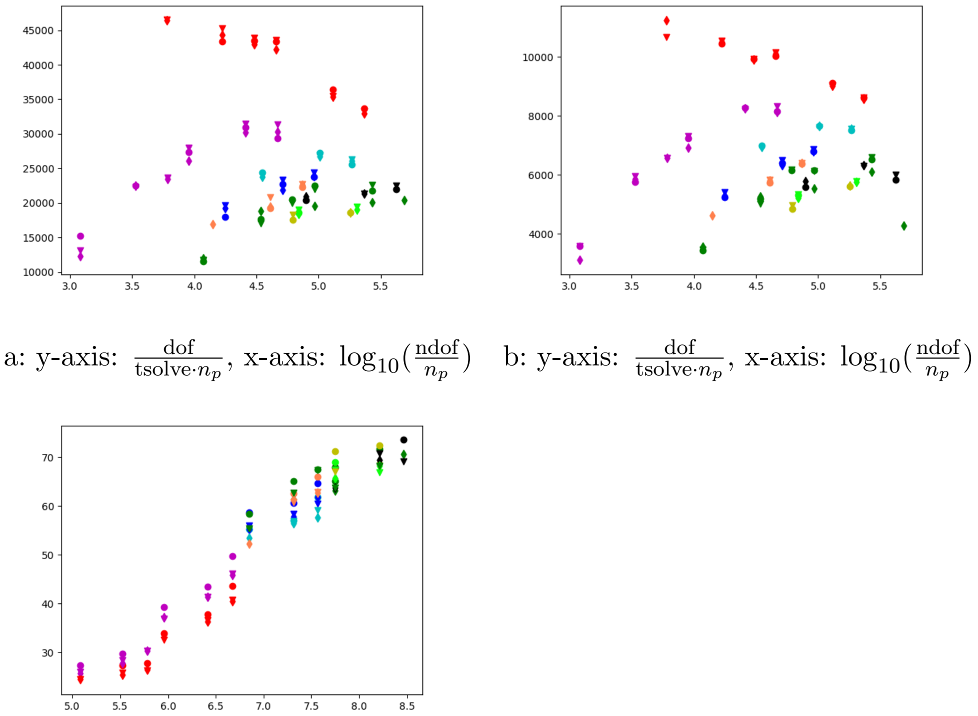
b: x-axis: $\log_2(n_p)$. y-axis: $\log_2(\text{time})$ for 25 iterations. DGSS is dashed. Colors are problem sizes: green: $5.7 \cdot 10^6$, purple: $14.2 \cdot 10^6$, orange: $14.8 \cdot 10^6$, blue: $18.8 \cdot 10^6$ and turquoise: $29.5 \cdot 10^6$

Figure 9.1: Comparison of DGSS and HGSS. On the left: Efficiency of HGSS(circles) and DGSS(triangles). The Y-axis shows time for a matrix-vector multiplication divided by time for one smoothing step. On te right: n_p versus wall time for 25 smoothing interations

Overall, DGSS performs acceptably where there is sufficient local work to hide communication but is still outperformed by HGSS across the board. It has to be said, however, that these computations were done on the finest levels. On coarser levels, the comparison is even less favorable towards DGSS.

AscAMG Performance

Figure 9.2 shows results for $\beta = 0$ and homogeneous dirichlet boundary conditions on $\Omega = (-1,1)^3$. α is constant on $\Omega_1 = (-0.3, 0.3)^3$ and either 10, 10^3 or 10^6. On $\Omega \setminus \Omega_1$, $\alpha = 1$. The mesh size and number of procs varies. Obtained by AscAMG -preconditioned PCG. The empleysd smoother is HGSS. Figure 9.2a does not take the time for assembling and AscAMG setup into account, 9.2b includes it. We prescribed that AscAMG uses the scheme (PPHPPH)-(NNBNNB)//(PH)-(NB) and left it to its' own devices otherwise. The cutoff-points for contraction and number of levels before the left over coarse system is solved directly have been left up to AscAMG to decide.

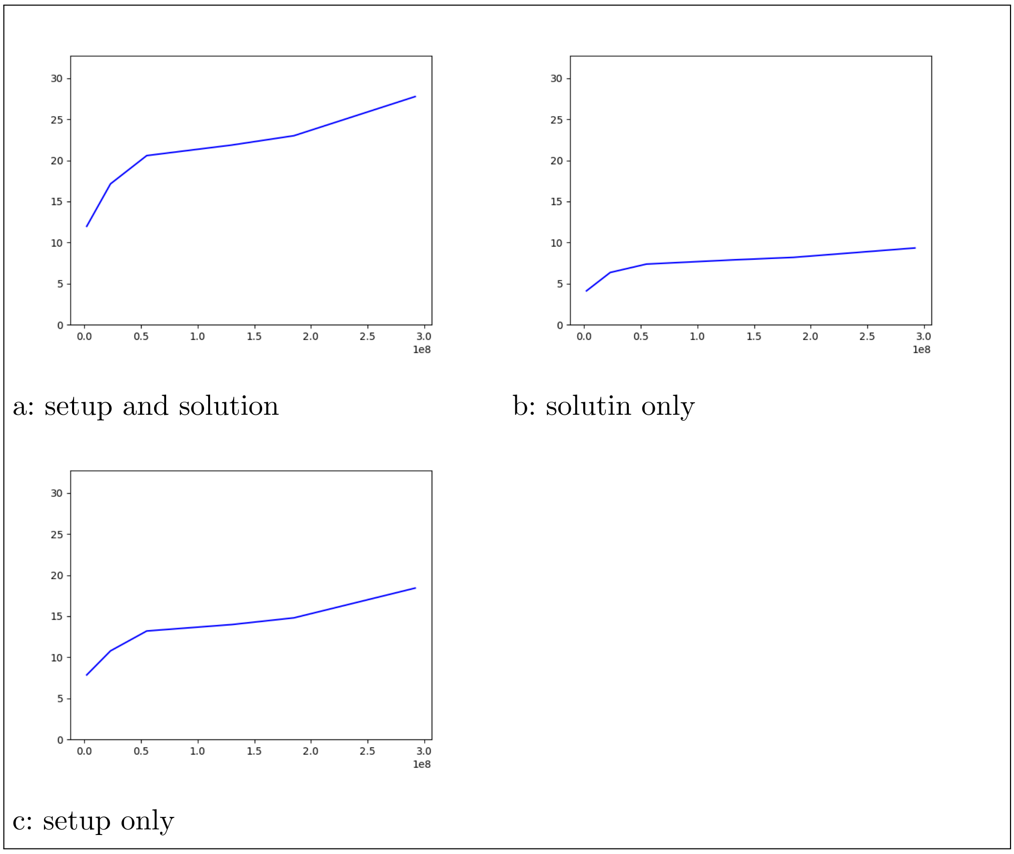


a: y-axis: $\frac{\text{dof}}{\text{tsolve}\cdot n_p}$, x-axis: $\log_{10}(\frac{\text{ndof}}{n_p})$ b: y-axis: $\frac{\text{dof}}{\text{tsolve}\cdot n_p}$, x-axis: $\log_{10}(\frac{\text{ndof}}{n_p})$

c: y-axis: condition number, x-axis: $\log_{10}(\text{ndof})$

Figure 9.2: Circles: $\alpha_1 = 10$. Rectangles: $\alpha_1 = 10^3$. Triangles: $\alpha_1 = 10^6$. Colors indicate number of procs; red:20, violet:100, turquoise:200, dark blue:400, orange:500, green:600, black:700, light green:800, ochre:900

We show also results of a test run where we solved the same equation, with $\alpha = 1$ and kept $\frac{\text{ndof}}{\text{nprocs}} \approx 2 \cdot 10^5$ constant while increasing both problem size and the number of procs evenly. This goes up to 1460 procs. In this case we did do some handtuning, by adding additional $\mathcal{P}$-prefix-stages to the scheme as the problem became bigger.



a: setup and solution

b: solutin only

c: setup only

Figure 9.3: y-axis:time, x-axis:ndof. All the while $\frac{\text{ndof}}{\text{proc}}$ is kept constant.

Concluding this chapter, let us take one computation and look a little closer at a lingle, ver large computation. The problem is again the same as above, with $\alpha = 1$, $\beta = 0$. The number of DOFs is about $3 \cdot 10^8$ and we are using 1800 cores, which represents 90% of COEUS's 100 standard compute nodes with 20 cores each. We use a total of 21 coarse levels, the scheme was the rather monstrous looking (PPHPPHPHPHPHPHPHPH)-(BNNBNNBNBNBNBNBNBNBNB)//(H)-(B). Again, we use HGSS. We solve the equation in 63 iterations and 6.9 seconds, the time needed for assembling the bilinearform and setting up the *AscAMG* preconditioner was 25 seconds. This gives us a very respectable performance of solving 29×10^3 unknowns per core and second. Counting the setup time, we get 5×10^3. The condition number of the preconditioned system is 107.161. A total of 6 contraction steps have been used, in total reducing 1800 procs on the finest level to only 43 active procs in the coarsest. The operator complexity of the V-cycle is 1.65. The maximal local operator complexity is 1.77, which tells us that the contraction happened at good breakpoints and spaced out enough that operator complexity was not overly impacted. The degrees of freedom where reduced from about $3 \cdot 10^8$ on the finest level to 2497 on the coarsest one.

10 Conclusion and Outlook

Conclusion

In this thesis we have introduced AscAMG . We have shown a new way to define strong connections via the replacement matrix that has resulted in a new variation of smoothed prolongation. Particular emphasis has been placed on all aspects of the the parallelization of AscAMG , for which a formalism has been introduced that allows us to write down parallel algorithms with great ease. Scalable Smoothers have been discussed and an, as far as I know, original approach to Distributed Gauss-Seidel has been presented. AscAMG has been shown to perform well and scale to at least 1800 cores, although without manual tuning it lacks consistency.

Future Work

While AscAMG has been shown to be able to perform well if tuned manually, we have not yet managed to find a good algorithm that can let AscAMG tune itself. This currently limits the consistency of the method and means that its usefulness for the average user who does not know all the peculiarities of the method is limited.

I have also personally not given up hope on making the distributed Gauss Seidel work respectably. While it is probably impossible to outperform the ℓ^1-Hybrid Gauss Seidel smoother, it remains nonetheless a very intriguing problem. One interesting approach would be to combine DGSS with a hybridization only for the $\mathcal{C}$-DOFs.

The next truly major step would be to hybridize AscAMG . NGSolve already provides very efficient parallelization either by MPI or by shared memory parallelization via C++-11-threads. One could build on these established foundations and combine the two approaches, into the effort of which AscAMG could be integrated.

Bibliography

[1] Mpi: A message-passing interface standard version 3.1. `http://mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`. Accessed: 2017-12-11.

[2] Allison H. Baker, Robert D. Falgout, Tzanio V. Kolev, and Ulrike Meier Yang. Multigrid smoothers for ultraparallel computing. *SIAM J. Sci. Comput.*, 33(5):2864–2887, October 2011.

[3] Dietrich Braess. *Finite Elemente.* Springer Berlin Heidelberg, Berlin, Heidelberg, vierte, berarbeitete und erweiterte auflage. edition, 2010.

[4] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[5] Joachim Schöberl. Netgen an advancing front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1(1):41–52, Jul 1997.

[6] Joachim Schöberl. C++11 implementation of finite elements in ngsolve. *Institute for Analysis and Scientific Comuting, Vienna University of Technology*, 2014.

[7] K. Stüben. *Algebraic Multigrid (AMG): An Introduction with Applications ; Updated Version of GMD Report No 53, March 1999.* GMD-Report. GMD-Forschungszentrum Informationstechnik, 1999.

[8] Petr Vaněk. Fast multigrid solver. *Appl. Math.*, 40(1):1–20, 1995.

[9] Petr Vaněk, Marian Brezina, and Jan Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numer. Math.*, 88(3):559–579, 2001.

[10] Panayot S. Vassilevski. Lecture notes on multigrid methods, 2010.

[11] Jinchao Xu. Iterative methods by space decomposition and subspace correction. *SIAM Rev.*, 34(4):581–613, 1992.