UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sede Amministrativa: Università degli Studi di Padova

Dipartimento di Ingegneria Civile, Edile e Ambientale (ICEA)

SCUOLA DI DOTTORATO DI RICERCA IN:
SCIENZE DELL'INGEGNERIA CIVILE ED AMBIENTALE
XXXI CICLO

Development of Scalable Linear Solvers for Engineering Applications

Tesi scritta con il contributo finanziario della Fondazione Cariparo.

Direttore della Scuola: Ch.mo Prof. Marco Marani

Relatore: Prof. Carlo Janna

Correlatore: Prof. Massimiliano Ferronato

Dottorando: Victor A. Paludetto Magri

UNIVERSITÀ DEGLI STUDI DI PADOVA

Abstract

Department of Civil, Architectural and Environmental Engineering

Doctor of Philosophy

Development of Scalable Linear Solvers for Engineering Applications

Victor A. Paludetto Magri

The numerical simulation of modern engineering problems can easily incorporate millions or even billions of unknowns. In several applications, particularly those with diffusive character, sparse linear systems with symmetric positive definite (SPD) matrices need to be solved, and multilevel methods represent common choices for the role of iterative solvers or preconditioners. The weak scalability showed by those techniques is one of the main reasons for their popularity, since it allows the solution of linear systems with growing size without requiring a substantial increase in the computational time and number of iterations. On the other hand, single-level preconditioners such as the adaptive Factorized Sparse Approximate Inverse (aFSAI) might be attractive for reaching strong scalability due to their simpler setup. In this thesis, we propose four multilevel preconditioners based on aFSAI targeting the efficient solution of ill-conditioned SPD systems through parallel computing. The first two novel methods, namely Block Tridiagonal FSAI (BTFSAI) and Domain Decomposition FSAI (DDFSAI), rely on graph reordering techniques and approximate block factorizations carried out by aFSAI. Then, we introduce an extension of the previous techniques called the Multilevel Factorization with Low-Rank corrections (MFLR) that ensures positive definiteness of the Schur complements as well as improves their approximation with the aid of tall-and-skinny correction matrices. Lastly, we present the adaptive Smoothing and Prolongation Algebraic MultiGrid (aSPAMG) preconditioner belonging to the adaptive AMG family that introduces the use of aFSAI as a flexible smoother; three strategies for

uncovering the near-null space of the system matrix and two new approaches to dynamically compute the prolongation operator. We assess the performance of the proposed preconditioners through the solution of a set of model problems along with real-world engineering test cases. Moreover, we perform comparisons to other approaches such as aFSAI, ILU (ILUPACK), and BoomerAMG (HYPRE), showing that our new methods prove comparable, if not superior, in many test cases.

UNIVERSITÀ DEGLI STUDI DI PADOVA

Sommario

Dipartimento di Ingegneria Civile, Edile e Ambientale

Dottorato di Ricerca

Development of Scalable Linear Solvers for Engineering Applications

Victor A. Paludetto Magri

Nell'ingegneria d'oggigiorno, le simulazioni numeriche possono facilmente arrivare alle dimensioni di milioni, se non miliardi, di incognite. Diverse applicazioni, specificatamente quelle a carattere diffusivo, richiedono la soluzione di sistemi lineari sparsi con matrici simmetriche definite positive (SPD). In questo contesto, i metodi multilivello rappresentano una scelta corrent, sia da usarsi per come solutori iterativi che come precondizionatori. La weak scalability mostrata da queste tecniche è una delle ragioni principali alla base della loro popolarità, dal momento che consente la soluzione di sistemi lineari di dimensioni crescenti senza un sostanziale aumento del tempo di calcolo e del numero di iterazioni. D'altro canto, i precondizionatori singolo livello, come l'adaptive Factorized Sparse Approximate Inverse (aFSAI), possono avere il vantaggio della strong scalability, principalmente grazie alla fase di setup più semplice. In questo lavoro di tesi, si propongono quattro precondizionatori multilivello, basati sulla tecnica aFSAI. L'obiettivo è la soluzione efficiente di problemi SPD mal condizionati, attraverso l'utilizzo di algoritmi adatti al calcolo parallelo. I primi due metodi, ovvero Block Tridiagonal FSAI (BTFSAI) e Domain Decomposition FSAI (DDFSAI), si basano su tecniche di riordinamento della matrice e su fattorizzazioni approssimate a blocchi, ottenute grazie alla aFSAI. In seguito, si presenta un'estensione delle tecniche precedenti, denominata Multilevel Factorization with Low-Rank corrections (MFLR), in grado di assicurare la definitezza (positiva) dei complementi di Schur e di migliorarne l'approssimazione grazie a delle matrici di correzione a rango basso. Infine, si introduce l'adaptive Smoothing and Prolongation Algebraic MultiGrid (aSPAMG), un

precondizionatore appartenente alla famiglia dei metodi multigrid adattivi, che si distingue per alcune caratteristiche, quali l'uso della aFSAI come smoother flessibile, tre strategie per approssimare il near-null space della matrice e due nuovi approcci per calcolare dinamicamente l'operatore d'interpolazione. Nel presente lavoro, si analizzano le caratteristiche dei precondizionatori proposti attraverso la soluzione sia di problemi artificiali che di matrici reali, ricavate da applicazioni ingegneristiche. Infine, si realizzano confronti con altri approcci molto noti, quali aFSAI, ILU (ILUPACK) e BoomerAMG (HYPRE), dimostrando che i metodi proposti hanno delle prestazioni comparabili, se non superiori, in diversi problemi.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Engineering problems are often solved through mathematical models that derive from the application of fundamental physical laws. Simple examples are the second Newton's equation of motion for computing the acceleration of a body or its statical equilibrium under the action of a set of mechanical forces; the Kirchhoff's circuit rules for computing currents and potentials values in closed electrical circuits or the thermodynamics' equations for calculating energy transfers and transformations across or inside systems. If written in their integral forms, these equations can frequently be solved analytically, i.e., after the application of a few arithmetic operations, an explicit formula for the unknown variable is obtained. However, when studying the infinitesimal distribution of these variables across a continuum media, these equations need to be restated in differential form according to the physical laws that originated them and their analytical solution is no longer straightforward to obtain. In fact, explicit solutions can be found only in artificial scenarios composed of regularly shaped domains and simple mathematical expressions representing the boundary conditions and materials properties.

When dealing with real-world applications, differential equations are solved only approximately through the use of numerical methods. These are techniques introduced in the last century that reformulate the mathematical problem valid for a continuous media into another one holding for a finite representation of the original domain associated with a limited number of degrees of freedom, a process commonly denoted as discretization. The first category of numerical methods to be proposed in the history are the finite difference methods (FDM) Strikwerda [2004]; LeVeque [2007], which

approximate derivatives at an arbitrary point by the truncation of the respective Taylor series according to the desired accuracy. After these, many other approaches were developed such as the finite element method (FEM) Zienkiewicz et al. [2013], which describes an approximate global solution as the superposition of local contributions at the element level weighted by a set of basis functions; the spectral element method (SEM) Gopalakrishnan et al. [2008], which extends FEM by using higher degree piecewise polynomials as basis functions and the finite volume method (FVM) LeVeque [2002], that guarantees local conservation of physical properties at control volumes.

One common aspect of all discretization strategies is that they require the solution of a sparse linear system of equations

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

where $A \in \mathcal{R}^{N \times N}$ is a sparse matrix; $\mathbf{b} \in \mathcal{R}^{N}$, the right hand side vector and $\mathbf{x} \in \mathcal{R}^{N}$, the approximate solution of the mathematical model being studied. For a given problem, linear systems with more unknowns and larger size normally improve the accuracy of the final solution. Also, the use of high order approximation schemes during the discretization normally leads to better solutions.

No matter the discretization method and order chose, the solution of sparse linear systems often represents the most significant portion of the computational time needed for numerical simulation. To certify this fact, we show in Figure 1.1, the wall-clock time associated with different phases of a numerical simulation where the mechanical deformation of a steel gear under the action of rotating forces in its teeth is calculated. We see that the linear system solution is the most expensive phase with more than 85% of the total computation time. Besides, this value can increase when refining the mesh or dealing with other mathematical models such as in the transient or non-linear cases where a sequence of linear systems needs to be solved to find the final solution of the model. Thus, it is of paramount importance the development of efficient algorithms for the solution of sparse linear systems.

Linear problems of the type (1.1) can be solved either by direct or iterative methods. Direct methods are robust approaches for finding an approximate solution $\mathbf{x}$ up to the

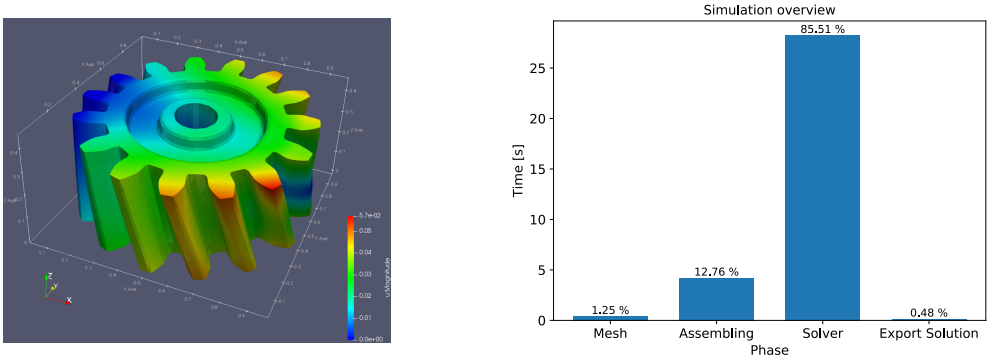

FIGURE 1.1: Overview of the numerical simulation costs.

machine precision. They consist of two major phases, namely setup, and solve. The first is usually the most expensive concerning computational time and is responsible for decomposing the matrix A as the product of two sparse triangular factors. The second one consists of solving the resulting triangular systems, something that is quickly done via backward and forward substitution. One of the main drawbacks of these methods is that their computational and memory complexities can grow up to $O(N^2)$ when solving matrices with a high number of nonzeros per row such as in the case of high order discretizations and three-dimensional problems. Also, their parallelization above the range of a thousand cores is not straightforward due to inherently sequential kernels needed in their construction such as the triangular solves.

Iterative methods build a sequence of vectors converging to the exact solution of (1.1) up to the machine precision. They generally have lower computational and memory complexities than direct methods, meaning that after a certain problem size $N^\star$, they become more advantageous than the former choice. See, e.g, Brussino and Sonnad [1989]; Fan et al. [1996]; Saad [2003]; Wang et al. [2007]; Fialko [2014]; Puzyrev et al. [2016]. Besides this, iterative methods primarily rely on first level linear algebra operations, i.e., those involving only vectors, and sparse linear algebra operations such as the sparse matrix/vector multiplication (SpMV), which are naturally parallel tasks allowing these methods to be reasonably scalable in parallel computational environments.

If used in their standalone form, iterative methods become inefficient since their convergence rate is unsatisfactory and, thus, a considerable computational effort may be required in their application phase. The most common remedy for this is preconditioning. These are linear operators whose action is sufficiently close but not equal to

A^{-1}. They can be divided into several categories, for instance, according to the specific type of problem that they are designed for and the family of algorithms they belong to. However, a common aspect of all these techniques is that they target the reduction of the conditioning number of the original linear system and, consequently, improves the convergence ratio of iterative methods.

1.1 Thesis objective

The general objective of this thesis is the development of preconditioners for solving (1.1) efficiently. We focus on the specific case of symmetric positive definite (SPD) matrices that are originated from the discretization of the Poisson and linear elasticity equations. Although, we note that the methods developed here are valid also for other applications leading to SPD matrices. Moreover, we aim at solving sufficiently large problem sizes starting range of $O(10^6)$ unknowns. We specifically follow the road of iterative methods and search for the construction of robust preconditioners relying on sparse approximate inverse techniques Kolotilina and Yeremin [1993]; Benzi and Tůma [1999]; Huckle [2003]; Janna et al. [2015b].

1.2 Engineering applications

The work developed in this thesis ultimately aims the efficient solution of general engineering problems modeled by elliptical partial differential equations which, once discretized through a numerical method, requires the solution of sparse linear systems formed by symmetric positive definite (SPD) matrices. Particular examples of engineering applications contemplated by the methods developed herein are:

1. the multiphase flow in porous media modeled by Darcy's law. This physical problem is solved, for instance, by oil reservoir simulators aiming to predict the cumulative fluids production, e.g., water and hydrocarbons, over a delimited time window Jenny et al. [2003]; Gerritsen and Durlofsky [2005]; Nordbotten and Celia [2011]; Horgue et al. [2015];

2. the study of geomechanical effects in subsurface rock formations modeled by the uncoupled poroelasticity equations. Deformation of these structures can occur due to anthropogenic actions towards hydrocarbons exploitation and underground gas storage. This phenomenon can furtherly lead to the subsidence/uplift of the ground surface or even promote induced seismicity. Thus, it is of great importance to forecast these events to prevent possible environmental impacts Kumpel [2002]; Teatini et al. [2005]; Zoback [2007]; Ferronato et al. [2010a];

3. the structural analysis of mechanical components modeled by the linear elasticity equations. This analysis is carried out, for example, during the design phase of new structures to compute their deformation under different loads and placing conditions with the ultimate aim to verify the structures' fitness for use prior than using real prototypes for physical tests Boresi et al. [2010]; Zienkiewicz et al. [2014]; Madenci and Guven [2015];

4. the creeping flow in open surfaces modeled by the Stokes' equations. This is the case of highly viscous fluids or flows with small velocities such that the Reynolds number is smaller than one Tuminaro et al. [2016]. Practical examples are the study of viscous polymers Vogel et al. [2014] and lava flow Dietterich et al. [2017];

5. the heat propagation into general domains due to diffusion modeled by the energy conservation equation. The solution of this mathematical model is the temperature distribution along the region being considered and, under certain conditions, can be applied to solid, liquid and gases Miah et al. [2018].

1.3 Contributions

In this work, we introduce four new algebraic preconditioners to be used in conjunction with the conjugate gradient method (CG) Hestenes and Stiefel [1952]. They share a common ingredient which is the use of the adaptive factorized sparse approximate inverse (aFSAI) Janna and Ferronato [2011] preconditioner at some stage of their construction. The block tridiagonal aFSAI (BTFSAI) is based on the LDU decomposition of

A viewed as a tridiagonal matrix and the further approximation of the Schur complement inverses utilizing the aFSAI algorithm. The domain decomposition aFSAI (DDFSAI) reorders the vertices of the adjacency graph of A via a multilevel K-way algorithm Karypis and Kumar [1998] that divides them into two categories, the method proceeds by computing the block factorization of the reordered matrix and uses aFSAI for approximating matrices inverses in an explicit form. A third multilevel preconditioner called multilevel FSAI with Low-Rank corrections (MFLR) improves the robustness of the former ones with the aid of low-rank corrections. Lastly, we propose the adaptive smoothing and prolongation algebraic multigrid preconditioner (aSP-AMG) which can be viewed as an enhanced black-box algebraic multigrid method since it does not assume any information about the constitution of the near-null space of A. Moreover, it employs aFSAI as the smoothing method rendering it especially attractive to massively parallel machines.

1.4 Outline

This thesis is organized as follows. In this chapter, we presented the problem being studied and its relevance for the solution of engineering problems modeled by partial differential equations. In chapter 2, we talk briefly about the most common numerical methods used in engineering and the main characteristics of the linear problems generated by them. Also, we make a literature overview of the most used direct and iterative methods, especially, employed for the solution of (1.1), pointing out the main pros and cons of each strategy. In chapter 3, we present two novel preconditioners based on aFSAI, namely BTFSAI and DDFSAI, that can be viewed as a multilevel extension of the former one. The robustness of BTFSAI is furtherly improved with low-rank corrections in chapter 4 giving rise to the MFLR preconditioner. In chapter 5, another class of multilevel preconditioners is exploited, namely the adaptive algebraic multigrid methods. We introduce the aSP-AMG preconditioner that employs a few ideas of sparse approximate inverse techniques in its construction and targets the solution of larger problem sizes than the previous preconditioners. The performance of each of those methods is assessed in their respective chapters through the solution of simple model problems

as well as real-world test cases. In chapter 6, we provide the main conclusions of this thesis and note a few closing remarks. Lastly, in appendix A, we provide details about the sparse linear matrices solved in this work that arise from real-world engineering applications.

Chapter 2

Background

In this chapter, we provide a closer look at the linear problem (1.1). Assuming that the mathematical model at hand is given by a PDE of elliptical type, we start by introducing the most fundamental numerical methods used in engineering that involves the solution of linear systems, paying special attention to algebraic characteristics of the generated matrices. Afterward, we present a succinct review of the most used direct and iterative methods for solving sparse linear systems and also we describe the most common preconditioners used for improving the convergence of the last methods.

2.1 Numerical methods

As already mentioned in the last chapter, numerical methods are essential for the solution of engineering problems modeled by PDEs. Several techniques have been developed in the previous century, and a common aspect of them is the generation of sparse linear systems. In this section, we recall the most used numerical methods in engineering and show essential algebraic features of the system matrices originated from the discretization process. For comparison purposes, we take as the standard model a homogeneous Poisson equation, which is typically used for modeling diffusion processes, considering a unitary square with zero Dirichlet boundary conditions everywhere and a unitary source term. This problem can be stated mathematically by:

$$\begin{aligned} -\Delta u &= 1, \quad \text{for } x \in \Omega = [0,1]^2; \\ u &= 0, \quad \text{on } \partial\Omega. \end{aligned} \tag{2.1}$$

All methods begin with the definition of a discrete representation of the domain being studied. Then, they proceed by approximating the PDE, in this case, Equation (2.1), into the new discretized domain. For a comprehensive review of numerical methods for engineering applications, we refer the reader to the works of Thomée [2001]; Larsson and Thomee [2003]; LeVeque [2007] and Grossmann et al. [2007].

2.1.1 Finite Differences Method (FDM)

In the finite differences method Strikwerda [2004]; LeVeque [2007], a mesh made of grid points is defined to represent the domain in which the PDE is valid. Then, continuous functions are approximated by their values at specific nodes of the mesh, while function derivatives are estimated by differences evaluated through Taylor's series, always using a limited set of grid points. Assuming that u is a smooth function of class $\mathcal{C}^\infty$, h is the mesh size and x_0 the x-coordinate of an arbitrary grid point, the Taylors' expansion of u around x_0 is:

$$u(x_0+h) = u(x_0) + \frac{u^{(1)}(x_0)}{1!}h + \frac{u^{(2)}(x_0)}{2!}h^2 + \cdots + \frac{u^{(n)}(x_0)}{n!}h^n + O(h^{n+1}), \tag{2.2}$$

with $O(h^{n+1})$, the order of the error involved with the approximation. After some algebraic manipulation, the first and second derivatives of u can be written as

$$u^{(1)}(x_0) \approx \frac{u(x_0+h) - u(x_0)}{h} + O(h^1); \tag{2.3}$$

$$u^{(2)}(x_0) \approx \frac{u(x_0+h) - 2u(x_0) + u(x_0-h)}{h} + O(h^2). \tag{2.4}$$

Besides, we note that higher order approximations could be obtained in case more grid points were used in these calculations. Choosing the expression (2.4) to estimate the second derivative of u, the Poisson Equation (2.1) can be discretized as

$$4u(x_0,y_0) - u(x_0+h,y_0) - u(x_0-h,y_0) - u(x_0,y_0+h) - u(x_0,y_0-h) = h^2, \tag{2.5}$$

which leads to a system matrix A having five nonzero entries per row at maximum, since boundary nodes have fewer connections than others. Also, the matrix A is SPD,

since

$$\mathbf{u}^T A\mathbf{u} = \sum_{i=2}^{N} (u_i - u_{i-1})^2 > 0, \tag{2.6}$$

with N the total number of unknowns and u_i the $i-th$ unknown. Lastly, it can be shown that A is irreducibly diagonally dominant Maliska. In figure 2.1 we show an example of discretized mesh for the problem (2.1) and the sparsity pattern of the matrix generated by FDM.

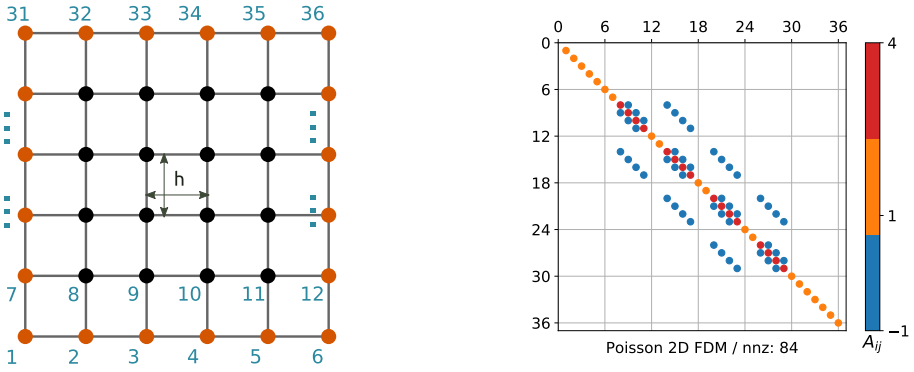

FIGURE 2.1: On the left, a Cartesian mesh with 6×6 grid points used for discretizing the Poisson equation via a finite difference method. The boundary points, depicted in orange, are associated with Dirichlet conditions. On the right, the sparsity pattern of the system matrix generated by FDM.

2.1.2 Finite Volumes Method (FVM)

In the classical finite volume method Patankar [1980]; LeVeque [2002], a mesh is composed of control volumes that do not intersect each other. Also, unknown variables attain a uniform value inside these entities; meanwhile, their derivatives at the volumes' interfaces are computed through finite difference formulas involving the unknown variables at neighboring control volumes. Then, discretization is performed by integrating the PDE over the control volumes through the divergence theorem, which ultimately produces a sparse linear system. This method is often used for the solution of PDEs deriving from conservation laws since numerical fluxes across the control volumes are guaranteed to be conserved locally, i.e., the partial differential equation integrated at each control volume is valid independently of the solution found for the

unknown variable. The classical FVM may use structured or unstructured grids, although the last ones are more common in another variant of FVM called the control volume finite element method variant (CVFEM) Baliga [1978]; Chen [2006].

We present a simplified recipe for solving the PDE (2.1) through the classical finite volume method. First, a mesh dividing the unit square into several partitions is created. A possible example is showed at the left portion of the figure 2.2 which depicts a cartesian 6×6 mesh composed by equally spaced control volumes of length $h \times h$. Then, the PDE can be integrated inside an arbitrary control volume as follows

$$\int_{y-\frac{h}{2}}^{y+\frac{h}{2}} \int_{x-\frac{h}{2}}^{x+\frac{h}{2}} \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) dx\, dy + \int_{y-\frac{h}{2}}^{y+\frac{h}{2}} \int_{x-\frac{h}{2}}^{x+\frac{h}{2}} \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}\right) dx\, dy = -\int_{y-\frac{h}{2}}^{y+\frac{h}{2}} \int_{x-\frac{h}{2}}^{x+\frac{h}{2}} 1\, dx\, dy, \tag{2.7}$$

where we omitted integration in the z-direction since we are dealing with a 2D geometry. Applying the divergence theorem to (2.7), we have

$$\int_{x-\frac{h}{2}}^{x+\frac{h}{2}} \left(\frac{\partial u}{\partial x}\right) dx + \int_{y-\frac{h}{2}}^{y+\frac{h}{2}} \left(\frac{\partial u}{\partial y}\right) dy = -h, \tag{2.8}$$

where volumetric integrals at the left-hand side of (2.7) are transformed into surface integrals giving the net flow across volumes' boundaries. Next, the first fundamental theorem of calculus gives

$$\left(\frac{\partial u}{\partial x}\right)\Bigg|^{x+\frac{h}{2}} - \left(\frac{\partial u}{\partial x}\right)\Bigg|^{x-\frac{h}{2}} + \left(\frac{\partial u}{\partial y}\right)\Bigg|^{y+\frac{h}{2}} - \left(\frac{\partial u}{\partial y}\right)\Bigg|^{y-\frac{h}{2}} = -h. \tag{2.9}$$

Lastly, the partial derivatives of (2.9) can be approximated with finite differences expressions such as (2.3) which leads to the following stencil equation for internal control volumes, i.e., with no contact with the domain boundaries:

$$\left(\frac{u_{(i+1,j)} - u_{(i,j)}}{h}\right) - \left(\frac{u_{(i,j)} - u_{(i-1,j)}}{h}\right) + \left(\frac{u_{(i,j+1)} - u_{(i,j)}}{h}\right) - \left(\frac{u_{(i,j)} - u_{(i,j-1)}}{h}\right) = -h, \tag{2.10}$$

with (i, j) denoting the index of the control volume centered at position $(x, y) = (i/5, j/5)$.

We note that the equation (2.10) can be rewritten as (2.6), thus we conclude that the discretization via FVM reduces to the finite difference case for internal control volumes. However, for boundary volumes, the equation (2.10) takes a different form. Considering the first control volume of the mesh we have

$$\left(\frac{u_{(i+1,j)} - u_{(i,j)}}{h}\right) - \left(\frac{u_{(i,j)} - u_w}{h/2}\right) + \left(\frac{u_{(i,j+1)} - u_{(i,j)}}{h}\right) - \left(\frac{u_{(i,j)} - u_s}{h/2}\right) = -h, \quad (2.11)$$

where u_s and u_w denote the Dirichlet conditions at the boundaries $x = 0$ and $y = 0$, respectively. Thus, the final expression for (2.11) can be written as

$$6u_{(i,j)} - u_{(i+1,j)} - u_{(i,j+1)} = h^2 + u_w + u_s, \quad (2.12)$$

differing from the way that FDM treats the boundary conditions. In the right portion of the figure (2.2) we show the sparsity pattern of the system matrix arising from FVM. In the same way as with FDM, the system matrix is symmetric positive definite and strictly diagonally dominant, it is also classified as an M-matrix.

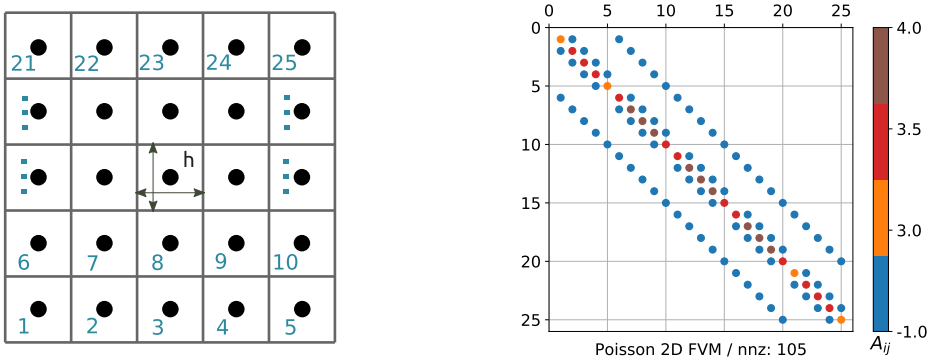

FIGURE 2.2: On the left, a Cartesian mesh made of 5×5 control volumes with uniform dimensions. On the right, the sparsity pattern of the system matrix generated by FVM.

2.1.3 Finite Element Method (FEM)

The Finite Element Method (FEM) was initially developed to solve problems of deformable continuous media and became famous in the solution of general engineering problems after the dissemination of the first edition of the book by Zienkiewicz et al. [2013]. A relevant characteristic of FEM is that it enables the use of complex meshes

such as unstructured grids, which are generally needed for representing realistic problems due to their complicated geometries. This method follows a similar approach to the FVM in the sense that it starts by defining a mesh as the union of elements with no intersection among themselves and proceeds by approximating the PDE at those entities. However, it differs in the way that this approximation is carried out. In the finite element methods, discrete functions such as the unknown u are generally variable inside an element. Their values at arbitrary points are interpolated from specific nodes by the use of basis functions, which also affects the integration of PDEs along elements. In the end, local matrices generated from the computation of integrals over the elements are summed up to assemble a global system matrix.

The finite element method can be used for solving the PDE problem (2.1) as follows. First, the continuous square domain is discretized into several elements of possibly arbitrary shapes. Here, we consider triangles, which are the most simple and commonly used elements, distributed in a structured grid configuration with elements size h equal to 0.2. We observe that unstructured grids with more general shapes could also be used with FEM, however, we prefer the structured choice to facilitate comparisons with the other discretization techniques. The obtained mesh is depicted in the left portion of the figure 2.3.

Before approximating the PDE via the finite element method, we need to define specific quantities. The first one is the test space $\mathcal{V}$ given by

$$\mathcal{V} = \left\{ v \in H^1(\Omega) \,\middle|\, v = 0 \text{ on } \partial\Omega \right\}, \tag{2.13}$$

where $H^1(\Omega)$ denotes the Sobolev space of infinitely differentiable functions compactly supported in Ω. Next, we define the trial space $\mathcal{U}$ as

$$\mathcal{U} = \left\{ v \in H^1(\Omega) \,\middle|\, v = u_0 \text{ on } \partial\Omega \right\}, \tag{2.14}$$

which contains the functions of $\mathcal{V}$ shifted by the boundary condition u_0. Lastly, we define a pair of discrete sets $\mathcal{V}_h \subset \mathcal{V}$ and $\mathcal{U}_h \subset \mathcal{U}$ which can be represented in terms of the basis functions $\phi(x)$ and $\psi(x)$, respectively. These sets are related to the mesh used for discretizing Ω, while the basis functions depend on the types of elements composing

the mesh. In the continuous Galerkin (classical) finite element method, the discrete spaces are represented by the same set of functions, thus $\psi(x) = \phi(x)$. Multiplying the PDE (2.1) by an arbitrary test function $v \in \mathcal{V}$ and integrating the result over the domain Ω, we find:

$$-\int_{\Omega} \nabla \cdot (\nabla u)\, v \; d\mathbf{x} = \int_{\Omega} 1\, v \; d\mathbf{x}. \tag{2.15}$$

Using the Green's formula, the left hand side of (2.15) can be rewritten as the composition of a volumetric and a surface integral leading to

$$\int_{\Omega} \nabla u \, \nabla v \; d\mathbf{x} - \int_{\partial\Omega} (\eta \cdot \nabla u)\; v \; d\mathbf{x} = \int_{\Omega} v \; d\mathbf{x}, \tag{2.16}$$

where η denotes a unitary vector normal to the boundary $\partial\Omega$. Recalling that $v = 0$ on $\partial\Omega$, the Equation (2.16) can be simplified to

$$\int_{\Omega} \nabla u \, \nabla v \; d\mathbf{x} = \int_{\Omega} v \; d\mathbf{x}, \tag{2.17}$$

which remains valid for the continuous domain Ω. However, this integral equation can be discretized over the sets U_h and V_h leading to

$$\int_{\Omega} \nabla u_h \, \nabla v_h \; d\mathbf{x} = \int_{\Omega} v_h \; d\mathbf{x}, \tag{2.18}$$

where v_h is an arbitrary function belonging to V_h and u_h is given by

$$u_h(x, y) = \sum_{k=1}^{N} \tilde{u}_k \; \phi_k(x, y), \tag{2.19}$$

with N, the dimension of the space U_k; $\tilde{u}_k$, the $k - th$ node unknown value belonging to the finite element and $\phi_k(x)$ its $k - th$ basis function evaluated at the coordinate x. This last quantity depends on the type of finite element chosen for creating the mesh.

In the case of linear triangular elements, the basis functions are given by

$$\phi_1(x,y) = \frac{1}{J}\left[(x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y\right]; \tag{2.20}$$

$$\phi_2(x,y) = \frac{1}{J}\left[(x_3y_1 - x_1y_3) + (y_3 - y_1)x + (x_1 - x_3)y\right]; \tag{2.21}$$

$$\phi_3(x,y) = \frac{1}{J}\left[(x_1y_2 - x_2y_1) + (y_1 - y_2)x + (x_2 - x_1)y\right]; \tag{2.22}$$

$$J = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2). \tag{2.23}$$

where (x_k, y_k) are the coordinates of the $k-th$ vertex of a triangle element. Substituting (2.19) at the Equation (2.18) and using the basis functions $\phi_k(x)$ to represent the discrete space V_h, we have:

$$\sum_{k=1}^{N} \tilde{u}_k \int_\Omega \nabla\phi_k \nabla\phi_i \, d\mathbf{x} = \int_\Omega \phi_i \, d\mathbf{x}, \quad \text{for } i = 1, 2, \cdots, N. \tag{2.24}$$

Thus, the approximate solution u_h can be found by solving the linear system $A\tilde{u} = b$ such that

$$A_{ik} = \int_\Omega \nabla\phi_k \nabla\phi_i \, d\mathbf{x}, \quad \text{and} \tag{2.25}$$

$$b_i = \int_\Omega \phi_i \, d\mathbf{x}. \tag{2.26}$$

We show the sparsity pattern of A at the right portion of Figure 2.3. In comparison to the previous discretization methods, it has a bigger stencil size which leads to more accurate solutions. Besides, the system matrix is still SPD and diagonally dominant; however, it contains off-diagonal coefficients that are positive, thus, it cannot be classified as an M-matrix.

Lastly, we provide a list of the most used open-source computational libraries that implement the necessary operations for running a finite element method.

- Deal.II - An open-source finite element library written in C++ Arndt et al. [2017].
- FEniCS - open-source computing platform for solving differential equations written mainly in C++ with interfaces to Python Alnæs et al. [2015].

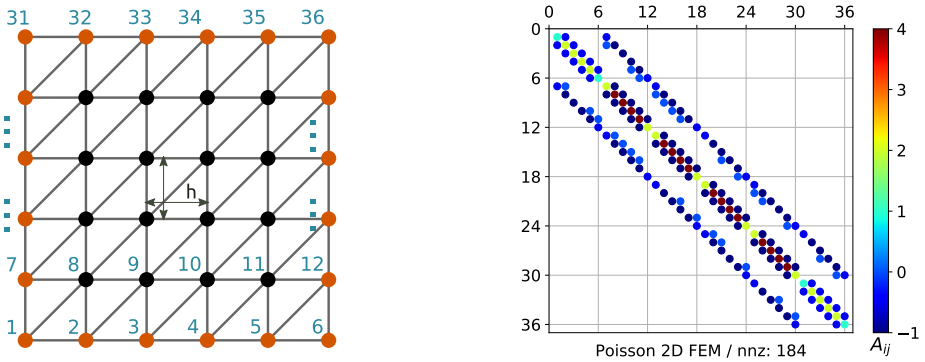

FIGURE 2.3: On the left, a mesh composed of linear triangle elements and used for discretizing the PDE problem (2.1) via the finite element method. On the right, the sparsity pattern of the system matrix generated by FEM.

- MFEM - A free, lighweight, scalable C++ library for finite element methods Kolev and Dobrev [2010].
- Libmesh - A framework for the numerical simulation of partial differential equations Kirk et al. [2006].
- GetFEM++ - An open-source finite element library Renard and Pommier [2002].
- FreeFem++ - A partial differential equation solver with its own language Hecht [2012].
- Feel++ - A scalable finite element embedded library written in C++ Prud'Homme et al. [2018].
- OOFEM - An objected oriented finite element solver written in C++ Patzák [2012].
- FEMPAR - An object-oriented parallel finite element framework Badia et al. [2018].

2.2 Sparse linear solvers

In the last section, we showed that the discretization of partial differential equations through numerical methods involve the solution of sparse linear systems. There are several techniques for solving these mathematical problems that can be appropriate or not depending on multiple factors such as the problem size, sparsity degree of the system matrix and inherent characteristics of the continuous problem originating the

linear system. In this section, we provide an overview of the most used approaches for solving sparse linear systems starting with direct methods and then moving to iterative methods.

2.2.1 Direct Methods

Direct methods are based on the decomposition of the system matrix as $A = LU$, where the right-hand side matrices are sparse and have a triangular sparsity pattern. These methods consist of four basic phases. Initially, a pre-processing step reorders the matrix A with the aim to minimize the fill-in of the triangular factors. Secondly, a symbolic factorization of the system matrix is carried out to compute the sparsity pattern of L and U. Then, a numerical factorization fills in the nonzero coefficients selected in the previous phase and finally, the linear system solution is computed through a pair a triangular solves involving L and U Duff et al. [2017].

Direct solvers are considered to be robust since a factorization can always be calculated, apart from cases where the matrix is numerically ill-conditioned. Also, due to their few configuration parameters and simplicity of use, they are frequently used as black-box solvers for industrial applications. However, these methods also have drawbacks which are mainly the high RAM consumption for computing and storing the triangular factors as well as a limited degree of parallelism for large-scale applications. Recent investigations by several authors Sourbier et al. [2009]; Gupta et al. [2009]; Rajamanickam et al. [2012]; Mary [2017] on the strong scalability of sparse direct solvers show that these methods typically have a good parallel efficiency up to the range of a hundred cores; however it starts to degrade significantly after that. A promising solution to improve the scalability of these solvers relies on the use of low-rank approximation techniques which decrease the space and time complexity of sparse direct solvers. These have been explored recently by the works Agullo et al. [2013]; Ghysels et al. [2016] and Rouet et al. [2016].

For an extended review of direct methods and their development over the last years, we refer the reader to the works Davis [2006]; Davis et al. [2016] and Duff et al. [2017]. Lastly, we present in Table 2.1 a list of commonly used open-source packages that provide sparse direct solver implementations.

TABLE 2.1: Open-source packages that implement sparse direct solvers.

Package	Parallelism	Matrices	Factorization	Reference
PARDISO	MPI/OpenMP	General CSR	Supernodal	Schenk et al. [2001]
HSL MA57	None	General CSR	Multifrontal	Duff [2004]
SuperLU	MPI/OpenMP	General CSR	Supernodal	Li [2005]
MUMPS	MPI/OpenMP	General CSR	Multifrontal	Amestoy et al. [2001]
STRUMPACK	MPI/OpenMP	General HSS	Multifrontal	Rouet et al. [2016]
CHOLMOD	None	SPD CSR	Supernodal	Chen et al. [2008]
UMFPACK	None	Unsym. CSR	Multifrontal	Davis [2004]
KLU	None	SPD CSR	Supernodal	Davis [2010]
PaStiX	MPI/OpenMP	General CSR	Supernodal	Hénon et al. [2002]

2.2.2 Iterative methods

Iterative methods compute a sequence of vectors starting from an initial guess x^0 that converges to the exact solution of (1.1) up to the machine precision. These methods share in common a few configuration parameters such as the maximum number of iterations and stopping tolerance based on the residual vector norm. On the other hand, they differ on how the sequence of vectors is built. They are often the only choice for nonlinear equations and particularly attractive for the solution of large-scale linear problems as far as the number of iterations remains small.

The simplest type of an iterative scheme is the Richardson method or fixed-point iteration which is given by

$$\mathbf{x}^k = \mathbf{x}^{k-1} + \omega M_1^{-1} \left(\mathbf{b} - A M_2^{-1} \mathbf{x}^{k-1} \right), \tag{2.27}$$

where the superscript k denotes the iteration count and ω a relaxation factor that controls the spectral radius of the iteration matrix. The operators M_1^{-1} and M_2^{-1} are denoted as left and right preconditioners, respectively, and will be explained detailed in the section 2.3.

A second class of iterative methods is given by the Krylov subspace based algorithms which builds a vector sequence of the type

$$\mathcal{K}_m = \left\{ \mathbf{r}_0 \, ; A\mathbf{r}_0 \, ; A^2\mathbf{r}_0 \, ; \cdots \, ; A^{m-1}\mathbf{r}_0 \right\}, \tag{2.28}$$

where $\mathbf{r}_0$ is the initial residual vector and m is the subspace dimension. Many algorithms can be defined to find an approximate solution of (1.1) through the use of the Krylov subspace (2.28). In the case of SPD system matrices, the most commonly used iterative method is the conjugate gradient Paige and Saunders [1975] which finds a solution minimizing the quantity

$$\underset{x \in \mathcal{K}_m + x_0}{\operatorname{argmin}} \; ||\mathbf{x} - \mathbf{x}^\star||_A \,, \tag{2.29}$$

where $\mathbf{x}^\star$ denotes the exact solution of Equation (1.1).

It is worth noting that the effectiveness of iterative methods is highly dependent on the preconditioner choice. In fact, advances on Krylov subspace methods have progressively faded over the last years while research on preconditioning techniques have gained increased interest Ferronato [2012].

2.2.3 Hybrid methods

Hybrid methods combine direct and iterative approaches to preserve the advantages and counterbalance drawbacks associated with both strategies. They commonly take a domain decomposition approach and solve the problem restricted to interior graphs with sparse direct solvers while the interface problem, which ties the interior problems altogether, is solved with a preconditioned iterative method applied to the accompanying Schur Complement. This way the amount of memory required for running direct solvers is decreased while the iterative counterpart couples the local solutions in a naturally parallel fashion.

The current approach is very suitable to parallel machines and somehow achieves better strong scalability than sparse direct methods on large clusters. Although overlapping techniques and multilevel techniques have been proposed, the property of weak scalability over the range of multiple millions of unknowns has not been shown yet. Lastly, this is an active field of research as described to the recent works Giraud et al. [2008]; Nakov [2015]; Agullo et al. [2016].

2.3 Preconditioning

Preconditioning regards techniques for altering the original linear system (1.1) to an equivalent one whose system matrix has a better conditioning number and, thus, provides a faster convergence when solved with iterative methods. The updated linear system can be written generically in the form

$$M_1^{-1} A M_2^{-1} \mathbf{y} = M_1 \mathbf{b} \tag{2.30}$$

$$\mathbf{x} = M_2^{-1} \mathbf{y}, \tag{2.31}$$

where M_1^{-1} and M_2^{-1} are denoted as the left and right preconditioners, respectively. We note that both operators are not necessarily different than the identity matrix I. In fact, most preconditioning methods choose $M_1^{-1} \neq I$ and $M_2^{-1} = I$. Furthermore, in the ideal case $M_1^{-1} = L^{-1}$ and $M_2^{-1} = U^{-1}$, which reduces actually to a direct method. Explicit representations of a preconditioner through a matrix are not always feasible. Instead, simply defining the sparse matrix/vector product $z = M^{-1} v$ is already enough for their application in the scope of iterative methods.

The central inquiry while developing preconditioners relies on how to choose operators M_1^{-1} and M_2^{-1} which effectively improve the convergence of iterative methods. Many researchers recognize that an optimal and general purpose approach applicable to a broad range of linear problems is unlikely to be found Benzi [2002]; Wathen [2015]. However, a list of necessary conditions that lead to efficient strategies exists and can be stated as follows:

1. the eigenspectrum relative to the preconditioned operator $M_1^{-1} A M_2^{-1}$ should be formed by one or more clusters of eigenvalues that are distant to the origin and sufficiently condensed or close to each other. In the context of iterative methods, the first criterion helps to reduce error components efficiently along eigenvectors associated with small eigenvalues, while the second criterion contributes to diminishing multiple error components in fewer iterations.

2. The construction time of a preconditioner should be the smallest possible for a given operator quality, which is measured by the convergence rate of the resultant

iterative method. Likewise, the application time of a preconditioner should be cost-effective.

3. The memory requirements for storing a preconditioner should be the lowest.

4. In case of large-scale problems, the construction and application phases of a preconditioner should also demonstrate the best parallel efficiency possible, which is predominantly influenced by the algorithm's degree of parallelism.

Preconditioners can be classified into two major classes: purely algebraic and problem-specific. In both cases, configuration parameters are used for controlling the quality and cost of a given strategy. However, they differ in the sense that methods belonging to the first category need no more information than the global system matrix A for defining the operators M_1^{-1} and M_2^{-1} while methods belonging to the second type use additional knowledge from the mathematical problem at hand such as the physics, geometry and discretization technique employed. Generally, the first ones are more robust however they often do not provide the optimal method for a given problem, a characteristic which is easier to hold in the second class of preconditioners. It is worth noting that, recently, a third category combining ideas of both algebraically and physically based approaches have been developed and denoted as "gray-box". For instance, particular physics-based pattern selection strategies can be used for algebraic sparse approximate inverse preconditioners as proposed by Carpentieri et al. [2005] and block variants of multilevel incomplete factorizations as in Janna et al. [2009b] and Carpentieri et al. [2014].

Below we give an overview of the most used algebraic approaches for building a preconditioner. For a comprehensive review of general preconditioning techniques, we refer the reader to the works Benzi [2002]; Ferronato [2012]; Wathen [2015]. Also, an extended review of multilevel preconditioners including incomplete factorizations algebraic multigrid and domain decomposition is given by Vassilevski [2008].

2.3.1 Regular splitting

The simplest preconditioners are based on the decomposition of the system matrix as

$$A = E + D + F, \tag{2.32}$$

where E, D and F are the lower triangular, diagonal and upper triangular part, respectively, of the matrix A. The damped (weighted) Jacobi preconditioner can be defined as

$$M_1^{-1} = \omega D^{-1}, \quad M_2^{-1} = I, \tag{2.33}$$

where ω is a relaxation factor rendering the preconditioner convergent if used with a Richardson iteration Richardson [1911]. Other common variants are the Successive Over Relaxation (SOR) preconditioner

$$M_1^{-1} = \omega \left(D + \omega E\right)^{-1}, \quad M_2^{-1} = I; \tag{2.34}$$

and its symmetric counterpart (SSOR)

$$\begin{aligned} M_1^{-1} &= \omega \left(2 - \omega\right) \left(D + \omega F\right)^{-1} D \left(D + \omega E\right)^{-1} \\ M_2^{-1} &= I. \end{aligned} \tag{2.35}$$

These preconditioners do not show calculation costs since they are readily available from the matrix A. If used solely, they are not effective strategies generally regarding convergence. However, when used as smoothers for multigrid methods they become very helpful Trottenberg et al. [2001].

2.3.2 Incomplete LU factorization

Incomplete LU Factorization (ILU), as the name clearly suggests, is based on the approximate decomposition of the system matrix such that

$$A \approx \tilde{L}\tilde{U}, \tag{2.36}$$

where $\tilde{L}$ and $\tilde{U}$ are the lower and upper triangular approximated factors. This leads to the following preconditioner

$$M^{-1} = \tilde{U}^{-1}\tilde{L}^{-1}. \tag{2.37}$$

There are several techniques for building an ILU preconditioner. The simplest one is to enforce a sparsity pattern that is selected a priori for the triangular factors. This strategy gives rise to the ILU(k) preconditioner where k denotes the fill-in factor, i.e., the ratio between the number of nonzeros of the triangular factors and A. Although very effective for M-matrices, the ILU(k) algorithm could fail for general matrices. This motivated the development of a more robust approach by Saad [1994], namely the incomplete LU preconditioner based on threshold (ILUT). In this strategy, the sparsity pattern of the triangular factors is built dynamically in accordance to a maximum number of nonzero coefficients per row as well as a control on their sizes through the elimination of sufficiently small entries with respect to a threshold. Subsequently, many other ILU variants were designed mainly to overcome particular issues involving the previous ones. We limit our presentation by simply citing a few of the most important variants such as the Crout Incomplete LU (ILUC) targetting better computational efficiency Li et al. [2003]; the Block Incomplete LU (BILU) for treating problems deriving from systems of PDEs Chow and Saad [1997a] and the Modified Incomplete threshold LU (MILUT) which produces approximate triangular factors such that $\tilde{L}\,\tilde{U}\,\mathbf{1} = A\,\mathbf{1}$, a very desired property for the solution of Poisson problems MacLachlan et al. [2012]. For comparisons between different ILU preconditioners we refer the reader to the works Chow and Saad [1997b]; Lin and Moré [1999]; Benzi and Tůma [2003]; Gupta [2017]

In general, incomplete factorization techniques lead to iterative methods with satisfactory convergence rates; however, their applicability in parallel machines is somehow limited since their algorithms normally carry a degree of sequentiality difficulting parallel implementations as well as because that they show superlinear complexities, which does not lead to weakly scalable solutions. Multilevel incomplete factorization techniques were developed to cope with these deficiencies. Their central idea is to use domain decomposition concepts for splitting recursively the adjacency graph of A into several levels. Efforts towards this direction were proposed by the works Li and Saad

[2013a]; Xi et al. [2016]; Li et al. [2016]; Carpentieri et al. [2016]; Bu et al. [2016].

Recently, many authors have worked on the extension of incomplete factorization techniques to massively parallel computing architectures. Examples can be found in the works by Li and Saad [2013b]; Chow and Patel [2015]; Chow et al. [2018] and Anzt et al. [2018a]. Besides, the development of specialized algorithms for sparse matrices with low-rank structures has been explored in the articles by Xia [2013a,b]; Ghysels et al. [2016]; Rouet et al. [2016]. Most of these scientific publications developed also open-source numerical packages and the table 2.2 lists a few of the pertinent ones.

TABLE 2.2: Open-source packages that implement incomplete factorization preconditioners.

Package	**Parallelism**	**Matrices**	**Method**	**Reference**
BPKIT	None	General CSR	Block	Chow and Heroux [1998]
ILUPACK	OpenMP	General CSR	Multilevel	Bollhöfer et al. [2011]
HSL_MI28	None	SPD CSR	Single-level	Scott and Tůma [2014]
HSL_MI30	None	Saddle-Point CSR	Single-level	Scott and Tůma [2014a]
ARMS	None	General CSR	Multilevel	Saad and Suchomel [2002]
pARMS	MPI	General CSR	Multilevel	Saad and Sosonkina [2002]
VBARMS	None	General CSR	Blk. Multilevel	Carpentieri et al. [2014]
pVBARMS	MPI	General CSR	Blk. Multilevel	Carpentieri et al. [2016]
STRUMPACK	MPI/OpenMP	General HSS	Multifrontal	Rouet et al. [2016]

2.3.3 Sparse Approximate Inverses (SAI)

Sparse approximate inverse preconditioners propose an explicit approximation for A^{-1} in the format of a sparse matrix. These are very efficient preconditioners for the case that the entry values of A^{-1} demonstrate a fast decay in magnitude from the diagonal Benzi and Tůma [1999]. Sparse matrices arising from the discretization of elliptic equations frequently fall into this category.

These techniques can be divided into two major classes: unfactored and factored. Methods belonging to the first category define the operators $M_2^{-1} = I$ and M_1^{-1} as the solution of the following constrained minimization problem:

$$\min_{M \in \mathcal{S}} \left|\left|I - M_1^{-1}A\right|\right|_F, \tag{2.38}$$

where $\mathcal{S}$ denotes a set of sparse matrices sharing the same sparsity pattern. This approach produces a left inverse only. A right inverse preconditioner could be obtained by solving the equivalent problem

$$\min_{M \in \mathcal{S}} \left|\left| I - A^T M_2^{-T} \right|\right|_F, \tag{2.39}$$

which implies $M_1^{-1} = I$. This distinction can be important for nonsymmetric matrices since there are cases where solving Equation (2.38) is easier than (2.39) or vice-versa Benzi and Tůma [1999]. Perhaps the most well-known method of this type is the SPAI preconditioner proposed by Grote and Huckle [1997]; Huckle and Sedlacek [2011] which assumes a candidate sparsity pattern informed by the user and fills it according to an iterative algorithm controlled by dropping tolerances; the number of entries to be added and maximum number per row of M^{-1}. Recently, this preconditioner has been improved for the solution of low-frequency error components by the incorporation of probing Huckle et al. [2010].

Factored techniques are based on incomplete inverse factorizations of the system matrix i.e., they search for operators such that $M_1^{-1} \approx L^{-1}$ and $M_2^{-1} \approx U^{-1}$. This strategy is specially attractive for SPD system matrices since the property $M_1^{-1} = M_2^{-T}$ is guaranteed to hold, which may not be true in the other class of sparse approximate inverses. This feature helps to reduce the preconditioner setup cost and enables its use with symmetric iterative solvers such as the conjugate gradient method. The first strategy of this type was proposed by Kolotilina and Yeremin [1993] with a naturally parallel and robust algorithm since it could not break down. Next, Benzi et al. [1996] proposed the AINV preconditioner constructed by a biconjungation algorithm that computes a triangular factorization of A^{-1} without its explicit knowledge. Depending on the problem, this strategy could lead to break down and this flaw was remediated with the stabilized AINV by Benzi et al. [2000]. Other techniques have been proposed such as the FSPAI by Huckle [2003]; the adaptive factorized sparse approximate inverse (aFSAI) by Janna and Ferronato [2011] and the incomplete sparse approximate inverse (ISAI) by Anzt et al. [2018b]. We refer the reader to the work by Benzi and Tůma [1999] for a comparison of sparse approximate inverses techniques.

In general, sparse approximate inverses do not show high dependence on the matrix ordering. One of the significant challenges in computing these preconditioners consists of defining the sparsity pattern of the matrices belonging to the set $\mathcal{S}$ Benzi et al. [2001]. Ideally, a full pattern would lead to the calculation of the exact inverse of A. For obvious reasons, this choice is not feasible and selecting a sufficiently sparse pattern leading to an efficient preconditioner can be complicated. Dynamic pattern selections tend to be more robust than static, i.e., a priori, choices; however, they are more costly. Recently, novel algorithms lying into the first category have been proposed by Janna and Ferronato [2011]; Janna et al. [2015b] and Janna et al. [2015a]. We give special attention to the adaptive Factorized Sparse Approximate Inverse method (aFSAI) in the section 3.2.

Lastly, we show in Table 2.3 a list of open-source numerical packages that implement sparse approximate inverses preconditioners.

TABLE 2.3: Open-source packages that implement sparse approximate inverse preconditioners.

Package	Parallelism	Matrices	Pattern	Reference
Parasails	MPI	General CSR	Static	Chow [2001]
Sparslab	None	General CSR	Dynamic	Benzi et al. [2001]
HSL_MI12	None	Unsymmetric CSR	Dynamic	Gould and Scott [1998]
SPAI	MPI	General CSR	Dynamic	Huckle and Sedlacek [2011]
MSPAI	MPI	General CSR	Static/Dynamic	Huckle et al. [2010]
FSPAI	MPI	SPD CSR	Static/Dynamic	Huckle [2003]
FSAIPACK	OpenMP	SPD CSR	Static/Dynamic	Janna et al. [2015b]

2.3.4 Domain Decomposition (DD)

Domain decomposition preconditioners are mainly motivated by the solution of large-scale linear systems through parallel computing. Roughly speaking, their basic idea is to split the problem's domain into several parts and distribute them among processors on a parallel machine. A global solution is then formed by gathering local solutions computed in each subdomain. This strategy is often referred to as divide-and-conquer. The first solution methods of this class date several years before the advent of parallel computing and even computers themselves. Schwarz methods, as proposed in 1870 by Schwarz [1870], aimed to find the approximate solution of elliptical PDEs in domains

formed by the superposition of regularly shaped geometric forms through restricting the original problem to each of the subdomains.

There are two main families of domain decomposition algorithms: overlapping Schwarz and iterative substructuring. The central idea of both methods is to obtain convergence rates that are independent of the number of subdomains. In general, an additional component to provide global transfer of numerical information across all subproblems is needed for achieving weak scalability. They differ on how the subdomain problems are defined and coupled to yield the global solution.

In the case of overlapping Schwarz, subdomains may share common regions with their neighbors. These methods are generally based on solving Dirichlet problems and the overlap plays a significant role in rendering them robust regarding convergence. The simplest examples are the Additive Schwarz (AS) and Multiplicative Schwarz (MS) methods as presented by Chan and Mathew [1994].

Iterative substructuring or non-overlapping methods are based on the implicit elimination of degrees of freedom associated with the interior of each subdomain. The resulting Schur complement system is then solved iteratively. In the balancing Neumann-Neumann (BNN) method from Goldfeld et al. [2003], Neumann problems are solved at the subdomain boundaries, which requires additional information arising from the unassembled or partially assembled matrices. A more recent approach namely the Balancing Domain Decomposition by Constraints (BDDC) enforces continuity between subdomains at every iteration Zampini and Keyes [2016]; Zampini [2016]; Badia et al. [2016] while the FETI-DP aims the same objective through the use of Lagrange multipliers Farhat et al. [2001]; Klawonn et al. [2002].

For a comprehensive review of domain decomposition methods, we refer the reader to Chan and Mathew [1994]; Smith et al. [1996]; Toselli and Widlund [2005]; Dolean et al. [2015]. Lastly, we show in Table 2.3 a list of open-source numerical packages that implement domain decomposition preconditioners.

2.3.5 Algebraic Multigrid (AMG)

Algebraic MultiGrid (AMG) is a family of iterative methods, also used as preconditioners, built upon a hierarchy of levels associated with linear problems of decreasing size.

TABLE 2.4: Open-source packages that implement domain decomposition preconditioners.

Package	Parallelism	Matrices	Methods	Reference
AztecOO	MPI	General CSR	AS	Heroux [2004]
IFPACK	MPI	General CSR	AS	Prokopenko et al. [2016]
PETSc	MPI	General CSR	AS/BNN/BDDC	Balay et al. [2018]
HPDDM	MPI	General CSR	AS/FETI/BDD	Jolivet et al. [2013]
BDDCML	MPI	General CSR	BDDC	Sousedík et al. [2013]

Four essential components define a method belonging to this class: the restriction and prolongation (interpolation) operators, coarsening process, smoothing and application strategies. Varying these elements gives rise to different AMG methods; however, sharing a common characteristic which is indeed the central idea of algebraic multigrid: they all work by reducing the error between the iterative and exact solution of the linear system (1.1) while targeting different error components across their level hierarchy. The high-frequency part of the error is reduced in the fine level system while the coarse levels handle the low-frequency counterpart. In this way, optimality and efficiency are achieved by combining the two complementary processes: relaxation and coarse-grid correction.

Multigrid and multilevel domain decomposition methods share so many features that one method can usually be written as a special case of the other. In multigrid, moderate coarsening ratios are generally used and inexact solves or smoothers are applied in each level through the use of simple preconditioners that reduces error components locally while domain decomposition methods use very aggressive coarsening and strong smoothers. Ultimately, both preconditioning methods aim to reduce the dependency of the conditioning number on the mesh size. We reserve a more in-depth overview of these methods in the chapter 5. For a general review of multigrid techniques, we refer the reader to the works by Stüben [2001]; Falgout [2006] and Xu and Zikatanov [2017]. Lastly, we show in Table 2.5 a list of open-source numerical packages that implement geometric and algebraic multigrid preconditioners.

TABLE 2.5: Open-source packages implementing multigrid preconditioners.

Package	**Implementation**	**Matrices**	**Methods**	**Reference**
HYPRE	C/MPI	General CSR	GMG/AMG	Henson and Yang [2002]
HSL_MI20	Fortran	SPD CSR	classical AMG	Boyle et al. [2010]
PETSc	C/MPI	General CSR	agg. AMG	Balay et al. [2018]
ML	C++/MPI	General CSR	agg. AMG	Gee et al. [2006]
MueLu	C++/MPI+OpenMP	General CSR	agg. AMG	Prokopenko et al. [2014]
AGMG	Fortran/MPI	General CSR	agg. AMG	Notay [2010]
AmgX	C++/Cuda/MPI	General CSR	GMG/AMG	Naumov et al. [2015]
PyAMG	C/Python	General CSR	GMG/AMG	Bell et al. [2015]

Chapter 3

Multilevel aFSAI Preconditionining

This chapter presents two multilevel preconditioners based on a sparse approximate inverse technique, namely the block tridiagonal factorized sparse approximate inverse (BTFSAI) and the domain decomposition factorized sparse approximate inverse (DDFSAI). It extends the theory and results published in the work by Paludetto Magri et al. [2018].

3.1 Introduction

As presented in chapter 2, preconditioning is the principal ingredient for building robust and efficient iterative solvers. Given the increasing number of unknowns associated with realistic engineering models and the fact that parallel computing recently became the common tool for solving large computational simulations, it is natural to invest on preconditioning methods with good parallel efficiency. In this thesis, we concentrate on factorized sparse approximate inverses mainly because they offer embarrassingly parallel algorithms for both setup and application phases. However, regarding convergence, these preconditioners are not weakly scalable, i.e., the number of iterations required for solving problems with decreasing mesh size may grow substantially. To illustrate this behavior, we show in Figure 3.1 the number of iterations required for solving a 2D homogeneous Poisson model with increasing number of degree of freedoms when using the adaptive factorized sparse approximate inverse (aFSAI) preconditioner along with the conjugate gradient method.

This theoretical deficiency of aFSAI makes the amount of computational work associated with its application phase to grow superlinearly and thus limits its applicability

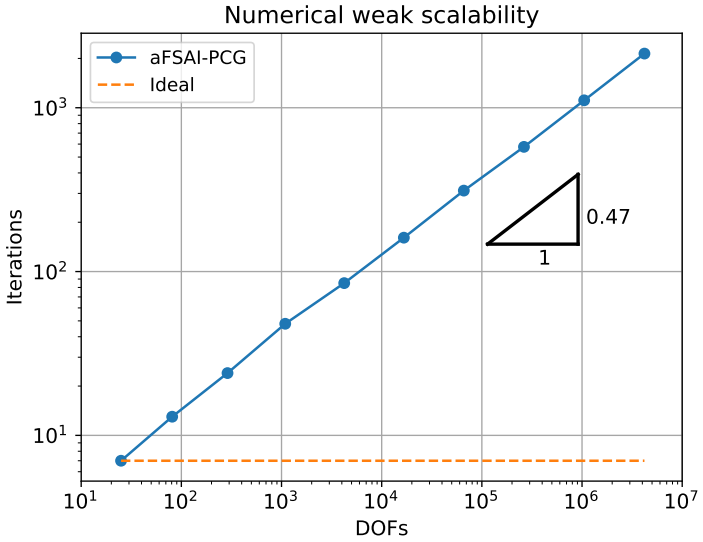

FIGURE 3.1: Numerical weak scalability of the aFSAI-PCG solver for the solution of a 2D homogeneous Poisson problem.

in parallel computing architectures for the solution of big[1] sparse linear systems. The preconditioners developed in this work aim to improve this flaw while maintaining the inherent parallelism showed by aFSAI to an acceptable degree.

In this chapter, we introduce two novel preconditioners based on the adaptive factorized sparse approximate inverse, namely the Block Tridiagonal FSAI (BTFSAI) and the Domain Decomposition FSAI (DDFSAI). The basic concept developed with these techniques is borrowed from incomplete factorizations, where the computation of a row takes advantage of the information gained from the previous ones. We note that this idea was already exploited in the context of sparse approximate inverses Benzi et al. [1996]; Chow and Saad [1998]; Raghavan and Teranishi [2010], however in this work we take a different approach with respect to the previous works. In the BTFSAI preconditioner, the system matrix is reordered with a band minimizing method and viewed as a block tridiagonal matrix; then, the aFSAI preconditioner is used to compute Schur complements and approximate its inverses. The DDFSAI preconditioner is based on the combination of a two-level domain decomposition approach and the FSAI preconditioner, which is used for computing the approximate inverse of the inner submatrices and the Schur complement.

[1] Number of degrees of freedom above 10^6.

This chapter is organized as follows: in section 3.2, we review the aFSAI preconditioner and present its algorithm. In sections 3.3 and 3.4, we introduce the BTFSAI and DDFSAI preconditioners, respectively. Next, in section 3.5, we evaluate their performance considering system matrices deriving from simple model problems as well as real engineering applications. Lastly, in section 3.6, we draw conclusions on the approaches presented in this chapter.

3.2 aFSAI Preconditioning

The Factorized Sparse Approximate Inverse (FSAI) Kolotilina and Yeremin [1993, 1995]; Kolotilina et al. [1999]; Yeremin et al. [2000] preconditioner M_{FSAI}^{-1} of a symmetric positive definite matrix A is given by

$$M_{FSAI}^{-1} = G^T G \approx A^{-1}, \tag{3.1}$$

where the factor G is calculated by minimizing the Frobenius norm

$$||I - GL||_F \tag{3.2}$$

over the set $\mathcal{W}_S$ of matrices sharing the same lower triangular nonzero pattern S. In the same equation, L is the exact lower Cholesky factor of A. Note that, although this matrix is used in Equation (3.2), it is not required for the calculation of G. Indeed, setting to zero the derivative of (3.2) with respect to the entries $[G]_{ij}$ yields

$$[GA]_{ij} = \begin{cases} 0, & \text{if } i \neq j, \quad (i,j) \in S \\ [L]_{ii}, & \text{if } i = j. \end{cases} \tag{3.3}$$

Since $[L]_{ii}$ is unknown, we can instead compute a matrix $\hat{G}$ such that:

$$\left[\hat{G}A\right]_{ij} = \delta_{ij}, \qquad (i,j) \in S; \tag{3.4}$$

where δ_{ij} is the Kronecker delta. Finally, the matrix G is obtained by scaling $\hat{G}$ with the square root of its diagonal entries, i.e., $G = D^{-1}\hat{G}$, with $D^{-1} = [\text{diag}(\hat{G})]^{1/2}$. This way,

the diagonal entries of the preconditioned matrix GAG^T are equal to one, which is a desirable property since $LAL^T = I$, with I the identity matrix. Once G has been calculated, the density μ_{FSAI} of the factorized sparse approximate inverse preconditioner is defined as

$$\mu_{\text{FSAI}} = \frac{nnz\,(G)}{nnz\,(A)}, \tag{3.5}$$

with $nnz\,(M)$ the function returning the number of nonzeros stored for the matrix M.

A key factor affecting the performance of FSAI is the selection of its sparsity pattern. In fact, there is a trade-off between the preconditioner quality in terms of convergence and its computational efficiency since denser sparsity patterns naturally lead to better convergence, however, with higher setup costs. The most straightforward choice of sparsity pattern is given by (small) powers of the system matrix, which is employed by the static FSAI algorithm. Although cheap and straightforward to calculate, this strategy lacks robustness regarding preconditioning quality. A more robust approach referenced as the adaptive Factorized Sparse Approximate Inverse (aFSAI) was proposed by Janna and Ferronato [2011] where the sparsity pattern of G is built dynamically such that an upper bound of the Kaporin number of the preconditioned matrix is minimized. Later, several approaches including the last two were compared by Janna et al. [2015b] and the authors concluded that aFSAI was the most robust one. For this reason, we focus on aFSAI in the present work and describe its algorithm next.

Suppose that an initial guess G_0 has already been computed statically by solving Equation (3.4) for every pair (i, j) lying in the initial pattern S_0. Then, scale G_0 with its diagonal entries, i.e.,

$$G_0 = \widetilde{D}_0 \widetilde{G}_0, \tag{3.6}$$

where $\widetilde{D}_0 = \text{diag}(G_0)$. The target of the dynamic pattern generation is to enlarge S_0 in order to reduce as much as possible a measure of the conditioning of the preconditioned matrix. To this aim, let us consider the Kaporin conditioning number of $G_0 A G_0^T$ Kaporin [1990, 1994]:

$$\kappa = \frac{\text{tr}(G_0 A G_0^T)}{n\,\det(G_0 A G_0^T)^{1/n}}. \tag{3.7}$$

Recalling that $G_0 A G_0^T$ and $\widetilde{G}_0$ have unitary diagonal entries, it follows that:

$$\kappa = \frac{1}{\det(A)^{1/n}} \frac{1}{\det(\widetilde{D}_0)^{2/n}}. \tag{3.8}$$

Since:

$$\operatorname{diag}\left(G_0 A G_0^T\right) = I, \tag{3.9}$$

by computing the determinant of both sides of (3.9) and using Equation (3.6), we have:

$$\det(\widetilde{D}_0)^2 \det[\operatorname{diag}(\widetilde{G}_0 A \widetilde{G}_0^T)] = 1, \tag{3.10}$$

and introducing Equation (3.10) into (3.8) finally yields:

$$\kappa = \det(A)^{-1/n} \det[\operatorname{diag}(\widetilde{G}_0 A \widetilde{G}_0^T)]. \tag{3.11}$$

Denoting by $\widetilde{G}_0[i,:]$ the i-th row of $\widetilde{G}_0$, Equation (3.11) reads:

$$\kappa = \det(A)^{-\frac{1}{n}} \prod_{i=1}^{n} \psi_{0,i}, \tag{3.12}$$

where $\psi_{0,i} = \widetilde{G}_0[i,:] A \widetilde{G}_0[i,:]^T$. The objective is to find an augmented pattern S_1 allowing for a reduction of κ in Equation (3.12) by minimizing each factor $\psi_{0,i}$. Moreover, since each of these factors depends only on the i-th row of $\widetilde{G}_0$, it follows that the dynamic pattern generation can be carried out in parallel on a row-wise basis. Defining $\mathcal{P}_i = \{j \mid (i,j) \in S_1,\ j < i\}$, the idea is to compute the gradient of κ with respect to the entries of $\widetilde{G}_0$ and add its largest components to the index set $\mathcal{P}_i$. After that, the coefficients $G_1[i, \mathcal{P}_i]$, i.e., those relative to the off-diagonal part of the i-th row of G, are computed by the following set of equations

$$\begin{cases} A[\mathcal{P}_i, \mathcal{P}_i]\ \widetilde{G_1^T}[i, \mathcal{P}_i] = -A[\mathcal{P}_i, i], & \text{(3.13a)} \\ G_1[i,i] = \dfrac{1}{\sqrt{A[i,i] + A[\mathcal{P}_i, i] \widetilde{G}_1[i, \mathcal{P}_i]}}, & \text{(3.13b)} \\ G_1[i, \mathcal{P}_i] = \dfrac{\widetilde{G}_1[i, \mathcal{P}_i]}{G_1[i,i]}; & \text{(3.13c)} \end{cases}$$

Algorithm 1 Adaptive FSAI setup.

procedure aFSAI_SetUp(k_g,ρ_g,ϵ_g,A)
  $\widetilde{G} \leftarrow I$
  for $i = 1, n$ **do**
    $\psi_{0,i} \leftarrow [A]_{ii}$
    for $k = 1, k_g$ **do**
      Compute $\nabla\psi_{k,i}$
      Add to $\mathcal{P}_i$ the ρ_g indices of the largest components of $\nabla\psi_{k,i}$
      Gather $A[\mathcal{P}_i, \mathcal{P}_i]$ and $A[\mathcal{P}_i, i]$ from A
      Solve $A[\mathcal{P}_i, \mathcal{P}_i]\widetilde{\mathbf{g}}_i = -A[\mathcal{P}_i, i]$
      if $\psi_{k,i} \leq \epsilon_g \cdot \psi_{k-1,i}$ **then**
        Exit the loop over k
      end if
    end for
    $\widetilde{d}_{ii} \leftarrow (-\widetilde{\mathbf{g}}_i^T A[\mathcal{P}_i, i])^{-\frac{1}{2}}$
    $\mathbf{g}_i \leftarrow \widetilde{d}_{ii}\widetilde{\mathbf{g}}_i$
    Scatter the components of $\mathbf{g}_i$ into $G[i, \mathcal{P}_i]$
  end for
  return G
end procedure

where the matrix $A[\mathcal{P}_i, \mathcal{P}_i]$ is small, dense, SPD and built up by gathering the coefficients of A referring to the positions $(i, j) \in S_1 \times S_1$. Analogously, the vector $A[\mathcal{P}_i, i]$ is composed by gathering the coefficients A_{ji} such that $j \in \mathcal{P}_1$. The Equation (3.13a) calculates the scaled coefficients of the i-th row of G while the Equation (3.13c) scales these values with the aim to ensure that

$$G_k[i,:]AG_k[i,:]^T = 1, \tag{3.14}$$

i.e., the preconditioned matrix has a unitary diagonal.

The procedure just described can be naturally iterated for a user-specified number of steps by adding an arbitrary quantity of new nonzero entries at each iteration, and can be stopped when the current value of $\psi_{k,i}$, being k the step counter, is reduced below some tolerance with respect to the initial value $\psi_{0,i}$. When either $\psi_{k,i}$ meets the stopping criterion, or a maximum number of steps are performed, the dynamic procedure ends. With these considerations in mind, the Algorithm 1 outlines a complete method for constructing the aFSAI.

At last, we note that the following user-specified parameters control the preconditioner density:

1. k_{max}, the maximum number of steps for the dynamic procedure;

2. ρ_F, the number of entries added per row at each step;

3. ϵ_F, the exit tolerance based on the relative reduction of $\psi_{k,i}$.

Regarding the computational cost of aFSAI, it is possible to show that it is proportional to the fourth power of μ_{aFSAI}. A more detailed description of the setup algorithm of FSAI and its complexity is provided in the work Janna et al. [2015b].

3.3 Block Tridiagonal FSAI (BTFSAI)

In the Block Tridiagonal aFSAI (BTFSAI), the basic idea is to identify a block tridiagonal structure in A and perform a block LDU decomposition. The inverses of the diagonal blocks are approximated explicitly by the aFSAI algorithm introduced in section 3.2. With this approach, the global matrix is subdivided into small blocks, that should be approximated with less effort, and the information of the first rows is used to compute the last rows of the preconditioner. This feature is counterbalanced by the introduction of an intrinsic sequentiality whose importance grows as the number of blocks increases. However, parallelism can still be exploited for each level of the decomposition.

The matrix A is first reordered by the reverse Cuthill-McKee algorithm Cuthill and McKee [1969], with the aim of reducing its bandwidth, then it is partitioned into a block tridiagonal structure for a given number of blocks. Let n be the number of partitions, A_i the SPD diagonal submatrix and B_i the upper diagonal submatrix of the $i-th$ partition. Thus, the matrix A can be written as:

$$\hat{A} = P^T A P = \begin{bmatrix} A_1 & B_1 & & & \\ B_1^T & A_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & B_{n-2}^T & A_{n-1} & B_{n-1} \\ & & & B_{n-1}^T & A_n \end{bmatrix}. \tag{3.15}$$

For example, considering a 3D Poisson problem discretized in a 10x10x10 cube, we can compare the original sparsity pattern of the system matrix to the reordered pattern used by the BTFSAI preconditioner configured with four partitions in Figure 3.2.

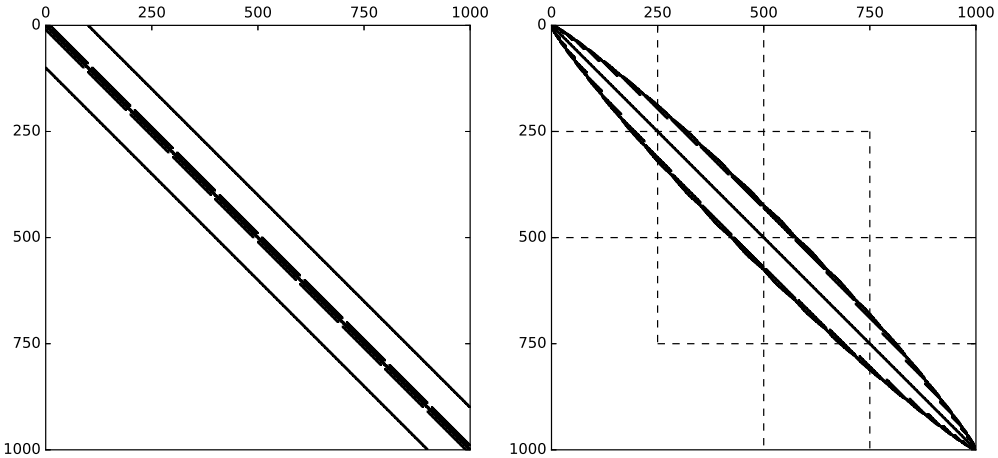

FIGURE 3.2: Left, sparsity pattern of the $10 \times 10 \times 10$ Poisson problem with native ordering. Right, reordered sparsity pattern employes by the BTFSAI preconditioner.

Computing the block LDU decomposition of the reordered matrix, we have:

$$\hat{A} = \begin{bmatrix} I_1 & & & \\ B_1^T S_1^{-1} & I_2 & & \\ & \ddots & \ddots & \\ & & B_{n-1}^T S_{n-1}^{-1} & I_n \end{bmatrix} \begin{bmatrix} S_1 & & & \\ & S_2 & & \\ & & \ddots & \\ & & & S_n \end{bmatrix} \begin{bmatrix} I_1 & S_1^{-1} B_1 & & \\ & I_2 & S_2^{-1} B_2 & \\ & & \ddots & \ddots \\ & & & I_n \end{bmatrix}, \quad (3.16)$$

where

$$S_i = A_i - B_{i-1}^T S_{i-1}^{-1} B_{i-1}, \qquad i = 2 \ldots n, \quad (3.17)$$

is the $i-th$ Schur complement and $S_1 = A_1$, by definition.

Let $S_i^{-1} \approx G_i^T G_i$ be an approximate inverse of the Schur complement computed by aFSAI and the matrix H_i be equal to the product $G_i B_i$. It follows that the Schur complement defined by Equation (3.17) can be calculated recursively as $S_i = A_i - H_{i-1}^T H_{i-1}$. If we substitute these relationships into Equation (3.16), the matrix $\hat{A}$ can be

approximated as

$$\begin{bmatrix} I_1 & & & \\ H_1^T G_1 & I_2 & & \\ & \ddots & \ddots & \\ & & H_{n-1}^T G_{n-1} & I_n \end{bmatrix} \begin{bmatrix} G_1^T G_1 & & & \\ & G_2^T G_2 & & \\ & & \ddots & \\ & & & G_n^T G_n \end{bmatrix}^{-1} \begin{bmatrix} I_1 & G_1^T H_1 & & \\ & I_2 & G_2^T H_2 & \\ & & \ddots & \ddots \\ & & & I_n \end{bmatrix}. \quad (3.18)$$

Computing the inverse of (3.18), we have

$$\begin{bmatrix} I_1 & -G_1^T H_1 & \dots & \dots \\ & I_2 & -G_2^T H_2 & \vdots \\ & & \ddots & \vdots \\ & & & I_n^T \end{bmatrix} \begin{bmatrix} G_1^T G_1 & & & \\ & G_2^T G_2 & & \\ & & \ddots & \\ & & & G_n^T G_n \end{bmatrix} \begin{bmatrix} I_1 & & & \\ -H_1^T G_1 & I_2 & & \\ \vdots & \ddots & \ddots & \\ \dots & \dots & -H_{n-1}^T G_{n-1} & I_n \end{bmatrix}, \quad (3.19)$$

which finally leads to the BTFSAI preconditioner:

$$\hat{A}^{-1} \approx \begin{bmatrix} G_1^T & -G_1^T H_1 G_2^T & \dots & \dots \\ & G_2^T & \ddots & \vdots \\ & & \ddots & -G_{n-1}^T H_{n-1} G_n^T \\ & & & G_n^T \end{bmatrix} \begin{bmatrix} G_1 & & & \\ -G_2 H_1^T G_1 & G_2 & & \\ \vdots & \ddots & \ddots & \\ \dots & \dots & -G_n H_{n-1}^T G_{n-1} & G_n \end{bmatrix}. \quad (3.20)$$

The steps for building the BTFSAI preconditioner are summarized in the Algorithm 2. As input parameters we have:

Algorithm 2 BTFSAI setup.

1: **procedure** BTFSAI_SETUP(n, k_{max}, ρ_F, ϵ_F, A)
2: Compute $A_1^{-1} \approx G_1^T G_1$, the approximate inverse of A_1
3: **for** $i = 2, \dots, n$ **do**
4: Compute $H_{i-1} = G_{i-1} B_{i-1}$
5: Compute $S_i = A_i - H_{i-1}^T H_{i-1}$
6: Compute $S_i^{-1} \approx G_i^T G_i$, the approximate inverse of the Schur complement
7: **end for**
8: **return** $G_{1 \cdots n}$; $H_{1 \cdots (n-1)}$
9: **end procedure**

1. A, the linear system matrix itself;
2. n, the number of desired partitions;
3. k_{max}, ρ_F and ϵ_F, the aFSAI configuration parameters.

Calling $bw(A)$ the bandwidth of A, i.e., the largest difference between the column indices of an arbitrary nonzero term of A to its diagonal, we note that the maximum number of partitions allowed for covering the entire matrix in a uniform block tridiagonal fashion is equal to $n_{max} = n(A)/bw(A)$. A larger number of levels could be chosen if the matrices B_i were not square; however, this would not necessarily lead to a more efficient preconditioner since the quantity of Schur-complements being approximated would also be higher. From our experience, a good estimate for the optimal n is given by $n_{max}/2$. Better performance can probably be found in the interval $n_{max}/2 < n < n_{max}$; however, it is likely that a marginal decrease in the total solution time will be verified in this case.

Regarding the setup algorithm, the outer loop on the i index is sequential and requires a thread synchronization before starting with a new level. The operations at each level, however, are almost perfectly parallel. Note that the matrices H_i are calculated explicitly, however the BTFSAI application does not require the off-diagonal terms $G_{i+1}H_i^T G_i$ and $G_i^T H_i G_{i+1}^T$ given in Equation (3.20) to be stored. Hence, only G_i, $i = 1, ..., n$, and H_i, $i = 1, ..., (n-1)$, have to be stored and the density μ_{BTFSAI} of the resulting preconditioner can be computed as

$$\mu_{\text{BTFSAI}} = \left(\sum_{i=0}^{n} nnz\left(G_i\right) + \sum_{i=0}^{n-1} nnz\left(H_i\right) \right) \Big/ nnz\left(A\right) \tag{3.21}$$

The procedure for applying the BTFSAI preconditioner to an arbitrary vector $\mathbf{v}$ is presented in Algorithm 3.

Algorithm 3 BTFSAI application.

procedure BTFSAI_APPLY(n,G_i,H_i,$\mathbf{v}$)
  $\mathbf{w}_1 = G_1\mathbf{v}_1$
  for $i = 2, \ldots, n$ **do**
    $\mathbf{w}_i = G_i\left(\mathbf{v}_i - H_{i-1}^T\mathbf{w}_{i-1}\right)$
  end for
  $\mathbf{u}_n = G_n^T\mathbf{w}_n$
  for $i = n-1, \ldots, 1$ **do**
    $\mathbf{u}_i = G_i^T\left(\mathbf{w}_i - H_i\mathbf{u}_{i+1}\right)$
  end for
  return u
end procedure

A weak point for this approach may arise in the computation of the $n-1$ Schur complements along with their aFSAI approximations. While theoretically the exact S_i should be SPD for every i, since it is computed numerically as the difference between two SPD matrices, A_i and $H_{i-1}^T H_{i-1}$, it could be indefinite depending on the approximations introduced in H_{i-1}. This difficulty tends to vanish when the quality of the approximate inverses increases because the current Schur complement becomes closer to the exact one. In fact, when a breakdown occurs, the aFSAI relative to the previous Schur complement is recomputed by using new k_{max} and ρ_F parameters such that μ_{aFSAI} is increased by a user-specified factor with a default value of 1.15. From our experience, a few restart steps can restore the positive definiteness of the Schur complement and permit the setup algorithm to evolve. However, the preconditioner effectiveness with respect to the reduction of PCG iterations can deteriorate. Another solution to this problem consists in rewriting the Schur complement S_i as the summation of SPD matrices instead of subtraction, as presented in the next chapter. This idea, which will be described in the next chapter, ensures greater robustness at the cost of a higher computational setup effort.

3.4 Domain Decomposition FSAI (DDFSAI)

An alternative way to partition A into many blocks relies on using a Domain Decomposition approach. Roughly speaking, domain decomposition refers to a family of techniques which solve a problem defined on a large domain Ω through the solution of smaller problems defined over subdomains $\Omega_k, k = 1, \ldots, n$. Each subproblem can be solved independently of the others for the internal variables, thus exploiting the intrinsic parallelism of any algorithm, and the solutions are eventually gathered to build the outcome of the full problem.

In the Domain Decomposition aFSAI (DDFSAI), the graph of the original matrix is adequately reordered, as proposed by Benzi et al. [1999] in another sparse approximate inverse context, according to the K-way partitioning algorithm implemented in the METIS software library Karypis and Kumar [1998]. With this method, the number of edges connecting a subdomain to the others is minimized, thus reducing the pieces

of information that need to be communicated among different partitions. At the same time, the subdomain sizes are kept approximately at the same order to ensure a satisfactory load balance among distinct threads. After this, each independent block of the matrix is reordered according to the reverse Cuthill-McKee algorithm for reducing its bandwidth.

Let Q be the permutation matrix employed by the DDFSAI preconditioner. The reordered system matrix can be written as:

$$\tilde{A} = Q^T A Q = \begin{bmatrix} C & B \\ B^T & D \end{bmatrix} = \begin{bmatrix} C_1 & & & & B_1 \\ & C_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & C_n & B_n \\ B_1^T & B_2^T & \dots & B_n^T & D \end{bmatrix}, \tag{3.22}$$

where the submatrices C_i represent the connections between the internal nodes of the $i-th$ subdomain; the matrix D, those between interface nodes, and B_i, the coupling between internal and interface nodes belonging to the $i-th$ subdomain. For instance, for the 3D Poisson problem used in Section 3.3, the difference between the original sparsity pattern of the system matrix and the one employed by the DDFSAI preconditioner when configured with four partitions is shown in the figure 3.3.

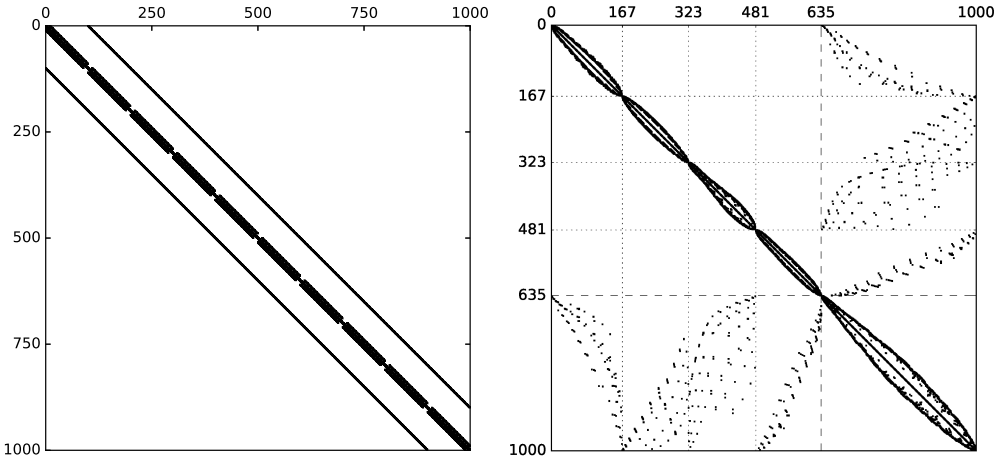

FIGURE 3.3: Sparsity pattern of the original test case on the left and the reordered pattern employed by the DDFSAI preconditioner on the right.

We can proceed as done formally in Section 3.3 for the BTFSAI preconditioner, just considering a 2-level partition. Writing the LDU decomposition of $\tilde{A}$ we have:

$$\tilde{A} = \begin{bmatrix} I_1 & \\ B^T C^{-1} & I_2 \end{bmatrix} \begin{bmatrix} C & \\ & S \end{bmatrix} \begin{bmatrix} I_1 & C^{-1}B \\ & I_2 \end{bmatrix}, \tag{3.23}$$

with $S = D - B^T C^{-1} B$ the Schur complement and I_i, $i = 1, 2$, the identity matrices with size equal to the number of internal and interface nodes, respectively. Note that the exact computation of S can be very expensive because of the high number of fill-in and linear solves required by the triple sparse matrix product $B^T C^{-1} B$. It is therefore desirable to approximate this calculation in a cheaper way. Indeed, following the same approach as in the BTFSAI preconditioner, we approximate S by $\tilde{S} = D - H^T H$ with $H = G_1 B$, where G_1 is the adaptive FSAI factor of C. This gives a sparse approximate inverse of the matrix $\tilde{A}$:

$$\tilde{A}^{-1} = \begin{bmatrix} I_1 & -C^{-1}B \\ & I_2 \end{bmatrix} \begin{bmatrix} C^{-1} & \\ & S^{-1} \end{bmatrix} \begin{bmatrix} I_1 & \\ -B^T C^{-1} & I_2 \end{bmatrix}. \tag{3.24}$$

and the inverse S^{-1} is approximated in the same way as C^{-1}, i.e., by the aFSAI algorithm. Finally, the DDFSAI preconditioner can be written as

$$M^{-1}_{\text{DDFSAI}} = \begin{bmatrix} I_1 & -G_1^T H \\ & I_2 \end{bmatrix} \begin{bmatrix} G_1^T G_1 & \\ & G_2^T G_2 \end{bmatrix} \begin{bmatrix} I_1 & \\ -H^T G_1 & I_2 \end{bmatrix}. \tag{3.25}$$

or even more concisely as:

$$M^{-1}_{\text{DDFSAI}} = G^T G = \begin{bmatrix} G_1^T & -G_1^T H G_2^T \\ & G_2^T \end{bmatrix} \begin{bmatrix} G_1 & \\ -G_2 H^T G_1 & G_2 \end{bmatrix}. \tag{3.26}$$

As already observed in the last section, the triple products $G_1^T H G_2^T$ and $G_2 H^T G_1$ are not assembled explicitly, but their action on a vector is calculated by three SpMV operations. The expression for computing the density of the DDFSAI preconditioner

turns out to be a special case of Equation (3.21) where only two levels are used:

$$\mu_{\text{DDFSAI}} = \left[nnz\left(H\right) + nnz\left(\sum_{i=1}^{2} G_i\right)\right] \Big/ nnz\left(A\right). \tag{3.27}$$

The procedure for building the DDFSAI preconditioner is almost the same as Algorithm 2 restricted to two levels, with the only difference consisting in the initial matrix reordering and also in the possibility to use different input parameters for building the aFSAI G_1 and G_2 factors. The same argument also applies to the preconditioner application. Notice here that the sequentiality introduced by the level partitioning is much less significant than the one characterizing the BTFSAI variant, hence the degree of parallelism ensured by DDFSAI is expected to be almost optimally preserved.

3.5 Numerical Results

In this section, we present a two-stage analysis of the BTFSAI and DDFSAI preconditioners. First, the impact of the DDFSAI and BTFSAI configuration parameters on their performance is studied considering the solution of two reference problems arising from the discretization of the Poisson and elasticity equations. In the second stage, both computational performance and parallel efficiency of an OpenMP implementation are evaluated for a set of real-world engineering problems described in the appendix A. The performance of the aFSAI algorithm as implemented in the FSAIPACK package Janna et al. [2015b] is also shown for comparison purposes.

We calculate the right-hand side vector associated with a unitary solution. Moreover, the preconditioned conjugate gradient method (PCG) is used as the iterative solver and convergence is reached when the relative Euclidean norm of the residual with respect to the right-hand side vector is smaller than 10^{-10}.

The performance of the resulting preconditioned iterative solvers is evaluated through the total number of iterations needed for convergence n_{it}, the wall clock time needed for building the preconditioner T_p and the time for solving the linear system T_s. The machine used for running the sequential test cases was a local cluster equipped with two Intel Xeon E5645 processors at 2.4GHz and 256 GB of RAM memory. On the other hand, the scalability tests were performed in a single node of the Marconi A1 cluster at

the CINECA - Center for High Performance Computing - which is equipped with 1512 nodes containing each two 18-core Intel(R) Xeon(R) E5-2697 v4 at 2.30GHz and 128 GB RAM.

3.5.1 Preconditioner analysis

A comprehensive analysis of the multilevel preconditioners proposed in this work is first carried out by solving two reference test cases arising from the discretization of elliptic partial differential equations, which are known to produce SPD linear systems of the type (1.1). The test cases are summarized below:

- `ps3d160`: discretization of the Poisson equation in a unitary cube by the standard 7-point finite difference method. Setting 160 nodes in each direction, the discretization gives rise to a matrix with 4,096,000 rows and 28,518,400 nonzeros entries;

- `el3d50`: discretization of the linear elasticity problem in a unitary cube by the finite element method. By employing 50 hexahedral elements in each direction, the discretization gives rise to a matrix with 397,953 rows and 30,986,559 nonzeros entries.

Initially, we find the optimal parameter configuration for the adaptive FSAI preconditioner, providing the lowest total wall-clock time, and set this configuration as the reference one. The search for the best setup parameters of the preconditioner is accomplished by using the guidelines suggested in Janna and Ferronato [2011] and Janna et al. [2015b]. Specifically, we varied the number of steps and the dropping tolerance used in the calculation of the aFSAI factor and later the step size while keeping the preconditioner density below 4.0, thus searching for a balance between preconditioner cost and quality. Then, the same input parameters are used for building the BTFSAI and DDFSAI preconditioners, with the only additional information being the number of blocks or partitions in both cases. The aFSAI parameters relative to the interface nodes for DDFSAI are also set independently.

In order to evaluate how the number of selected partitions affects the solution process in the `ps3d160` test case, we vary this quantity over a specific range and show in

the figure 3.4 the resulting number of iterations needed for convergence, the times for building these preconditioners and the times for solving the linear systems.

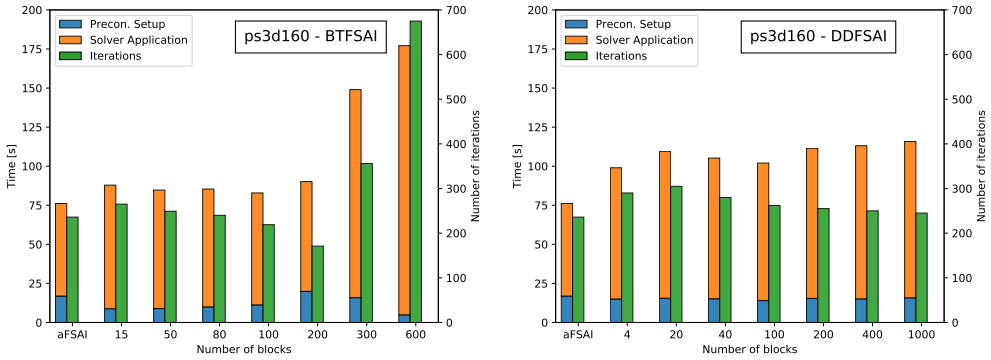

FIGURE 3.4: Matrix `ps3d160` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

Considering first the BTFSAI preconditioner, we note that the number of iterations needed for convergence decreases with respect to the reference case up to 200 partitions, then it starts to increase again. This behavior is caused by the quality of the approximated Schur complements, which degrades as the number of blocks increases, and by the fact that with a high number of levels the nonzero entries located farther from the main diagonal are neglected in the block tridiagonal factorization process. On the other hand, we see that both the number of iterations and computational times for the DDFSAI grows moderately with respect to the aFSAI and BTFSAI preconditioners while remaining quite stable for any number of partitions.

The same analysis is carried out for the `el3d50` test case and showed in the figure 3.5. As before, the number of iterations for convergence given by the BTFSAI preconditioner decreases up to a certain number of blocks and then starts to increase again. However the optimal number of blocks, in this case, is ten times smaller than before showing that this is a very problem-dependent parameter. Generally speaking, the larger is the number of nonzero off-diagonal terms, the smaller is the value for the optimal number of levels. Considering the DDFSAI preconditioner, we notice that the number of iterations can be reduced with respect to aFSAI when using 100 partitions and the total time becomes closer to the reference even when using more than 400 partitions.

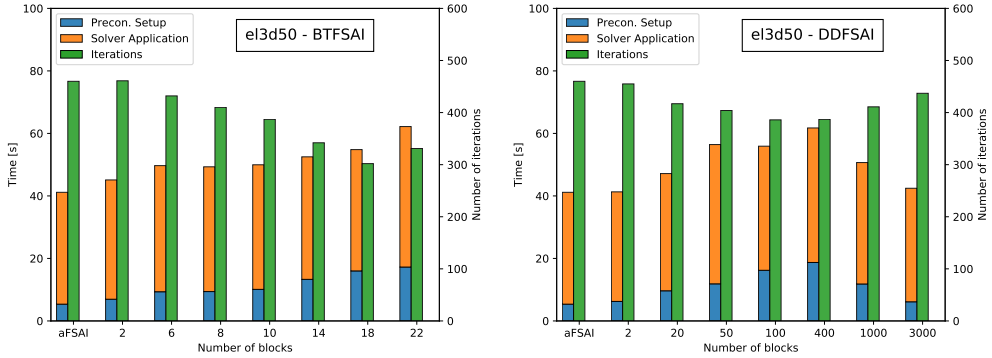

FIGURE 3.5: Matrix `el3d160` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

Next, fixing the optimal number of partitions for each multilevel preconditioner, we vary the input parameters which are required for the construction of the aFSAI factors. This analysis aims to evaluate whether and how the optimal configuration parameters for the BTFSAI and DDFSAI preconditioners vary with respect to the native FSAI algorithm applied to the whole matrix. The preconditioner density μ; number of iterations for convergence n_{it}; preconditioner setup time T_p; solution time T_s and total time T_t obtained while using the BTFSAI and DDFSAI preconditioners are reported in Tables 3.1 and 3.2, respectively. From these tables, we see that it is possible to improve the quality of the multilevel preconditioners by adjusting the aFSAI factor computations. However, such parameters are generally quite close to the ones used for the optimal aFSAI. This evidence is quite useful from a practical point of view because it simplifies the parameter setting for the multilevel FSAI variants.

Finally, we report in Table 3.3 the optimal configuration of each preconditioner regarding total solution time for both test cases.

3.5.2 Real-world engineering applications

The performance and scalability of BTFSAI and DDFSAI are analyzed considering a set of five matrices arising from different engineering applications, such as structural mechanics, geomechanics, electromagnetism and multiphase flow. A detailed description of each test case is presented in Appendix A. As in the previous subsection, we

TABLE 3.1: Tentative optimization of the BTFSAI preconditioner for the test matrices `ps3d160` and `el3d50`. The best configuration regarding T_t for each preconditioner is highlighted.

Case	Prec.	Input parameters				Output parameters				
		k_{max}	ρ_F	ϵ_F	n	μ	n_{it}	T_p [s]	T_s [s]	T_t [s]
ps3d160	aFSAI	2	3	0.0	–	1.00	236	16.9	59.2	76.1
	BTFSAI	**2**	**3**	**0.0**	**180**	**1.91**	**177**	**18.1**	**69.3**	**87.4**
		2	1	0.0	180	0.88	295	8.0	90.0	98.0
		5	1	0.0	180	1.61	197	28.0	71.8	99.9
		5	3	0.0	180	3.78	129	111.0	72.3	183.3
		5	3	1e-3	180	3.57	130	100.0	70.8	170.7
		10	1	0.0	180	2.74	152	120.7	72.0	192.8
		10	1	1e-3	180	2.70	152	116.8	71.4	188.2
		15	1	1e-3	180	2.97	128	177.6	63.2	241.0
el3d50	aFSAI	3	3	0	–	0.13	460	5.4	36.2	41.5
	BTFSAI	**3**	**3**	**0.0**	**18**	**0.60**	**302**	**16.0**	**38.8**	**54.1**
		3	1	0.0	18	0.42	467	7.2	49.8	57.0
		5	1	0.0	18	0.47	423	12.7	48.8	61.6
		5	3	0.0	18	0.78	247	31.1	36.6	67.3
		10	1	0.0	18	0.62	289	28.7	34.2	63.0
		10	3	0.0	18	1.18	195	142.3	35.7	178.1
		10	3	1e-3	18	1.16	195	135.8	35.4	171.2
		15	1	0.0	18	0.78	248	62.6	33.0	95.6
		15	3	1e-3	18	1.37	174	255.3	33.5	290.3

TABLE 3.2: Tentative optimization of the DDFSAI preconditioner for the test matrices `ps3d160` and `el3d50`. The best configuration is highlighted and the aFSAI input parameters for building the F_1 and F_2 are differentiated by the subscripts 1 and 2.

Case	**Prec.**	**Input parameters**							**Output parameters**				
		k_{max1}	ρ_{F1}	ϵ_{F1}	k_{max2}	ρ_{F2}	ϵ_{F2}	n	μ	n_{it}	T_p [s]	T_s [s]	T_t [s]
`ps3d160`	aFSAI	2	3	0.0	–	–	–	–	1.00	236	16.9	59.2	76.1
	DDFSAI	**2**	**3**	**0.0**	**2**	**3**	**0.0**	**2**	**1.01**	**285**	**17.5**	**79.2**	**96.7**
		3	1	0.0	3	3	0.0	2	0.59	355	22.6	84.4	107.0
		3	3	0.0	3	3	0.0	2	1.45	277	29.8	80.3	110.1
		5	1	0.0	5	1	0.0	2	0.87	306	37.8	80.3	118.1
		5	3	0.0	5	3	0.0	2	0.59	306	72.6	87.2	159.8
`el3d50`	aFSAI	3	3	0.0	–	–	–	–	0.13	460	5.4	36.2	41.5
	DDFSAI	**3**	**3**	**0.0**	**3**	**3**	**0.0**	**2**	**0.17**	**455**	**6.3**	**35.0**	**41.3**
		5	3	0.0	5	1	0.0	2	0.25	400	14.0	36.8	50.8
		5	3	0.0	5	3	0.0	2	0.27	385	14.8	32.8	47.6
		5	1	0.0	5	1	0.0	2	0.12	610	7.4	43.6	51.0
		5	1	0.0	5	3	0.0	2	0.13	608	7.9	47.8	55.7
		10	1	0.0	10	1	0.0	2	0.19	434	20.8	37.5	58.3

TABLE 3.3: Optimal configurations of the aFSAI, BTFSAI and DDFSAI preconditioners for the test matrices `ps3d160` and `el3d50`.

Case	**Prec.**	**Input parameters**							**Output parameters**				
		k_{max1}	ρ_{F1}	ϵ_{F1}	k_{max2}	ρ_{F2}	ϵ_{F2}	n	μ	n_{it}	T_p [s]	T_s [s]	T_t [s]
`ps3d160`	aFSAI	2	3	0.0	–	–	–	–	1.00	236	16.9	59.2	76.1
	BTFSAI	2	3	0.0	–	–	–	180	1.91	177	18.1	69.3	87.4
	DDFSAI	2	3	0.0	2	3	0.0	2	1.01	285	17.5	79.2	96.7
`el3d50`	aFSAI	3	3	0.0	–	–	–	–	0.13	460	5.4	36.2	41.5
	BTFSAI	3	3	0.0	–	–	–	18	0.60	302	16.0	38.2	54.8
	DDFSAI	3	3	0.0	3	3	–	2	0.17	455	6.2	35.0	41.2

show in Figure 3.6 through 3.10 how the performance of both preconditioners vary according to the number of partitions in each test case. Again, we use the optimal aFSAI input parameters for building the multilevel preconditioners, and then we evaluate their performance for a different number of partitions.

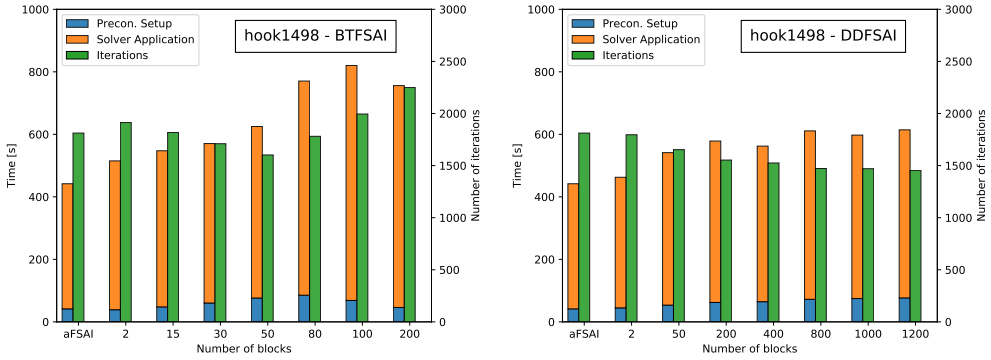

FIGURE 3.6: Matrix `hook1498` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

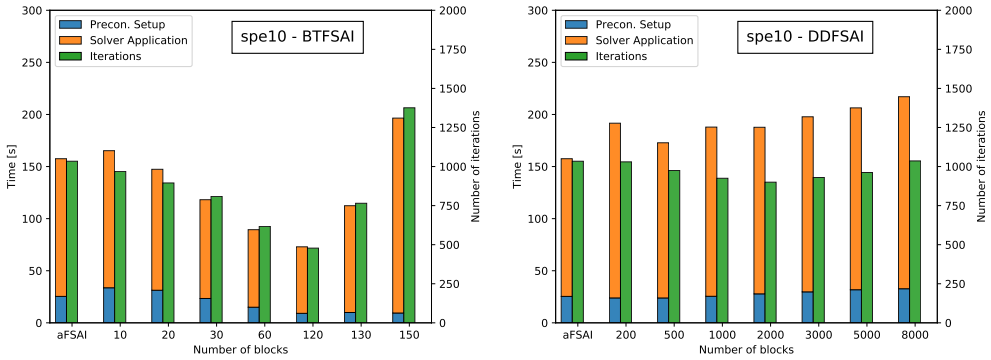

FIGURE 3.7: Matrix `spe10` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

In general, when compared to aFSAI, the BTFSAI method is able to reduce the number of iterations in most cases, showing up to a three-fold reduction, as in the `cylinder` test case. Consequently, the total execution time of linear solve is also reduced with a significant wall-clock time reduction. As already detected before, there is a problem-dependent optimal number of blocks for building this preconditioner. After this optimal number, we observe a sudden performance degradation, while too few blocks do not allow for thoroughly exploiting the block tridiagonal structure of the preconditioner. However, BTFSAI still can perform better than the pure FSAI.

The DDFSAI preconditioner can also decrease the solution time and the total number of iterations, although this effect is not as pronounced as in the BTFSAI case. Its behavior, however, is always very stable with the number of levels. When the time needed for building a good preconditioner is relatively large, like in the `cylinder` case, the use of DDFSAI becomes more advantageous since the inverse of the Schur complement can be approximated with a low accuracy showing no significant loss of effectiveness.

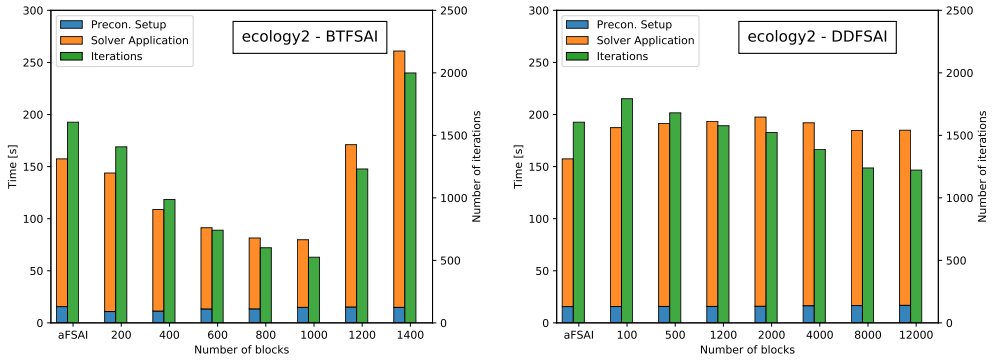

FIGURE 3.8: Matrix `ecology2` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

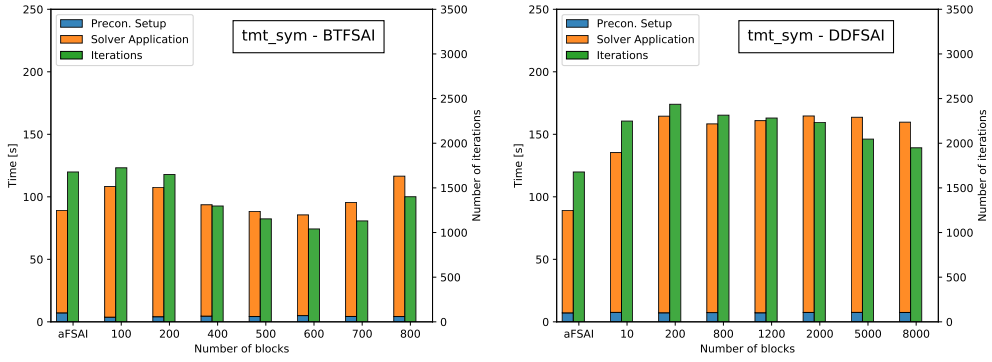

FIGURE 3.9: Matrix `tmt-sym` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

Next, given the optimal number of blocks for each test case and multilevel preconditioner, we vary the aFSAI configuration parameters looking for the optimal setting. The best results for each test case are reported in Table 3.4. We can conclude again that once the optimal aFSAI parameters are known, they can be used for the construction

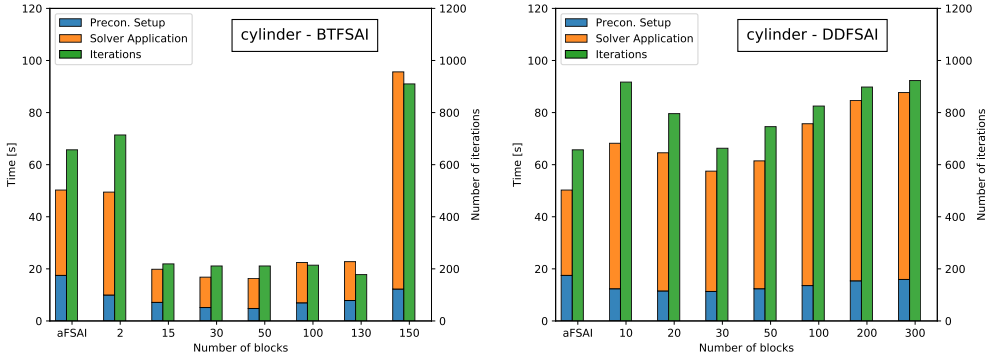

FIGURE 3.10: Matrix `cylinder` - solution times and number of iterations for the BTFSAI preconditioner on the left and for the DDFSAI preconditioner on the right.

of efficient multilevel preconditioners and the only parameter which still needs to be selected is the number of partitions.

TABLE 3.4: Tentative optimization of the multilevel preconditioners relative to the test matrices.

Case	Prec.	Input parameters							Output parameters				
		k_{max1}	ρ_{F1}	ϵ_{F1}	k_{max2}	ρ_{F2}	ϵ_{F2}	n	μ	n_{it}	T_p [s]	T_s [s]	T_t [s]
`hook1498`	aFSAI	5	3	0.0	–	–	–	–	0.40	1809	44.0	397.5	441.5
	BTFSAI	5	3	0.0	–	–	–	2	0.41	1904	48.5	474.1	522.6
	DDFSAI	5	3	0.0	5	3	0.0	2	0.41	1796	44.7	417.8	462.5
`spe10`	aFSAI	10	3	1e-2	–	–	–	–	2.51	1050	23.9	134.1	158.0
	BTFSAI	5	3	1e-3	–	–	–	110	2.41	392	14.7	57.0	71.7
	DDFSAI	10	3	1e-2	10	3	1e-2	500	2.58	975	23.8	148.7	172.5
`ecology2`	aFSAI	5	3	0.0	–	–	–	–	3.00	1605	15.5	141.9	157.4
	BTFSAI	10	3	1e-3	–	–	–	1000	3.72	387	17.1	48.7	65.8
	DDFSAI	5	3	0.0	5	3	0.0	100	1.82	1239	16.6	168.1	184.7
`tmt-sym`	aFSAI	5	1	0.0	–	–	–	–	0.86	1678	7.2	82.0	89.2
	BTFSAI	10	3	1e-3	–	–	–	600	3.21	313	21.1	36.0	57.1
	DDFSAI	5	1	0.0	5	1	0.0	10	0.87	1874	7.6	127.9	136.7
`cylinder`	aFSAI	10	1	1e-2	–	–	–	–	0.22	743	13.7	37.8	51.5
	BTFSAI	10	1	1e-3	–	–	–	30	0.18	238	3.6	12.5	16.1
	DDFSAI	10	1	1e-2	10	1	1e-2	30	0.42	663	11.3	46.2	57.5

3.5.3 Strong scalability

Lastly, we analyze the strong scalability of the multilevel preconditioners using up to 32 threads of a single node of the Marconi A1 cluster. We evaluate three different metrics

for measuring the degree of parallelism tackling different aspects. The first one is the inverse speedup which is given by

$$\frac{1}{s\left(p\right)} = \frac{T_t\left(p\right)}{T_t\left(1\right)}, \tag{3.28}$$

where p is the number of threads used and $T_t\left(p\right)$ is the total solution time relative to an arbitrary preconditioner. For a given solution strategy, this quantity shows how much the total solution time decreases with respect to the sequential case. On the other hand, the parallel efficiency, given by

$$e\left(p\right) = 100\left(\frac{s\left(p\right)}{p}\right), \tag{3.29}$$

uses the concept of speed-up to measure in percentage how close to perfectly parallel a solution strategy is. Finally, we consider the relative parallel efficiency as

$$re\left(p\right) = 100\left(\frac{T_t\left(1\right)}{T_{\text{FSAI}}\left(1\right)}\right)\left(\frac{T_{\text{FSAI}}\left(p\right)}{T_t\left(p\right)}\right) \tag{3.30}$$

which measures the relationship between the parallelism showed by the multilevel preconditioners and the native aFSAI. Here, a value greater than 100% means that the multilevel preconditioner scales better than aFSAI.

The metrics presented in Equations (3.28) to (3.30) are computed for aFSAI, BTFSAI and DDFSAI for all real-world test cases. The results are shown in Figure 3.11 through 3.15. Considering first the DDFSAI preconditioner, we note from the relative parallel efficiency graphs that its scalability is very close to aFSAI. Moreover, according to the inverse speedup graphs, it is close to the ideal profile up to 16 threads. If 32 threads are used for solving the linear systems, all scalability metrics suffer from the cluster's limited bandwidth and this is the reason why the performance of all preconditioners decays.

On the other hand, taking into consideration the BTFSAI preconditioner, we note that its scalability is worse than the one of aFSAI and DDFSAI. This is expected from the premises of the algorithm and is caused by the number of synchronization barriers

needed by BTFSAI, which increases with the number of partitions. These are responsible for increasing the load imbalance between threads as well as their overhead time, thus leading to a reduced parallel efficiency. We note, however, that when the number of partitions is sufficiently small, like for the `hook1498` test case, the BTFSAI preconditioner can lead to a scalability which is very similar to aFSAI.

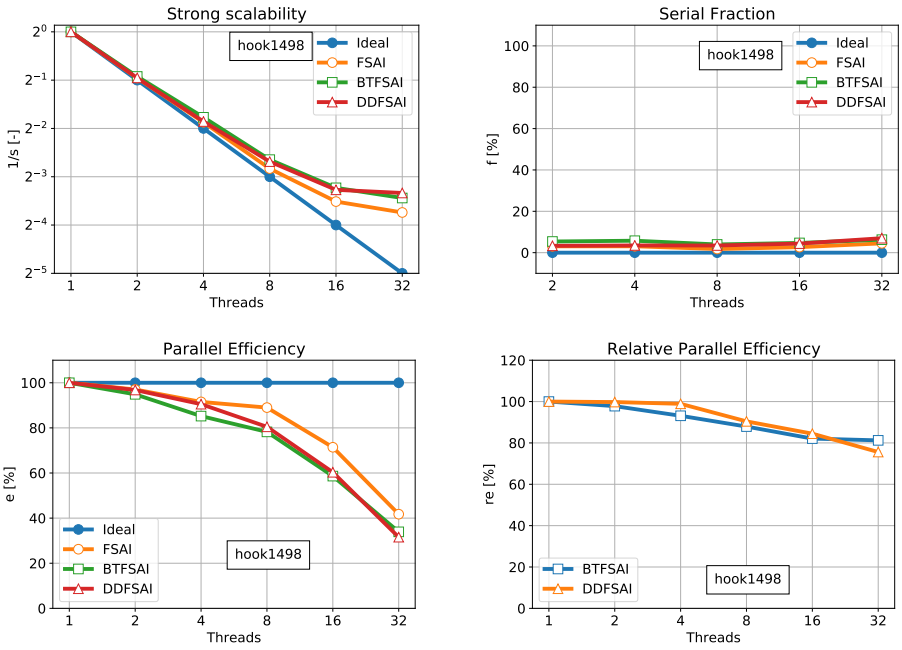

FIGURE 3.11: Scalability profiles for the matrix `hook1498`.

3.6 Conclusions

In this chapter, two new multilevel variants of the aFSAI preconditioner were discussed, namely the DDFSAI and BTFSAI algorithms. Their main idea is to improve the preconditioning quality given by aFSAI for a given memory footprint at the cost of reducing its degree of parallelism to a reasonable value. The multilevel variants of aFSAI were applied in the serial and parallel solution of SPD linear problems arising from model problems as well as distinct engineering applications. The results show that the BTFSAI technique is very promising in terms of decreasing the total number of iterations and time for achieving the solution in comparison to the original aFSAI preconditioner. On the other hand, the computational gain obtained with the DDFSAI technique is smaller than the one observed with BTFSAI; however, its stability and

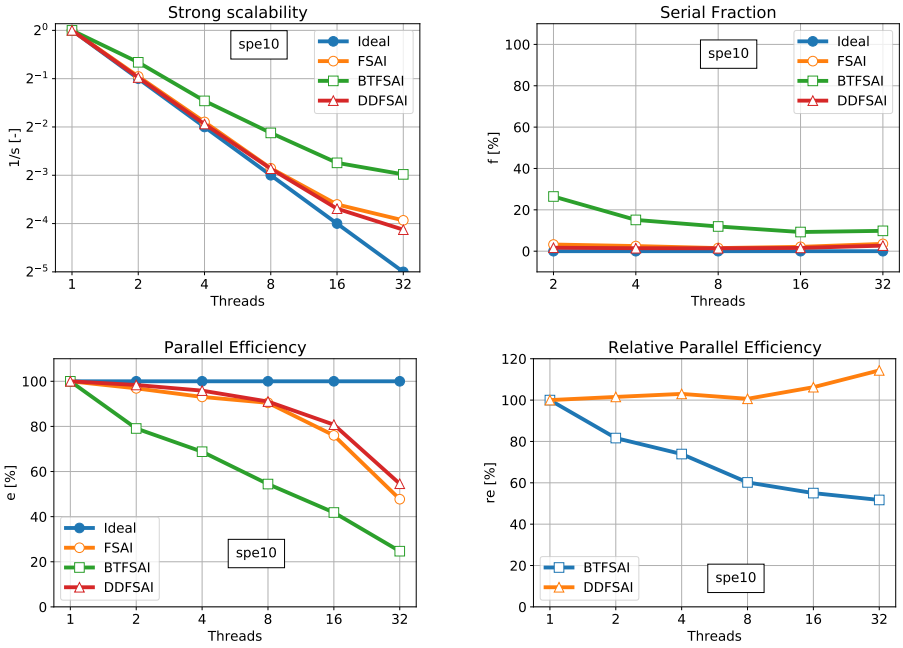

FIGURE 3.12: Scalability profiles for the matrix `spe10`.

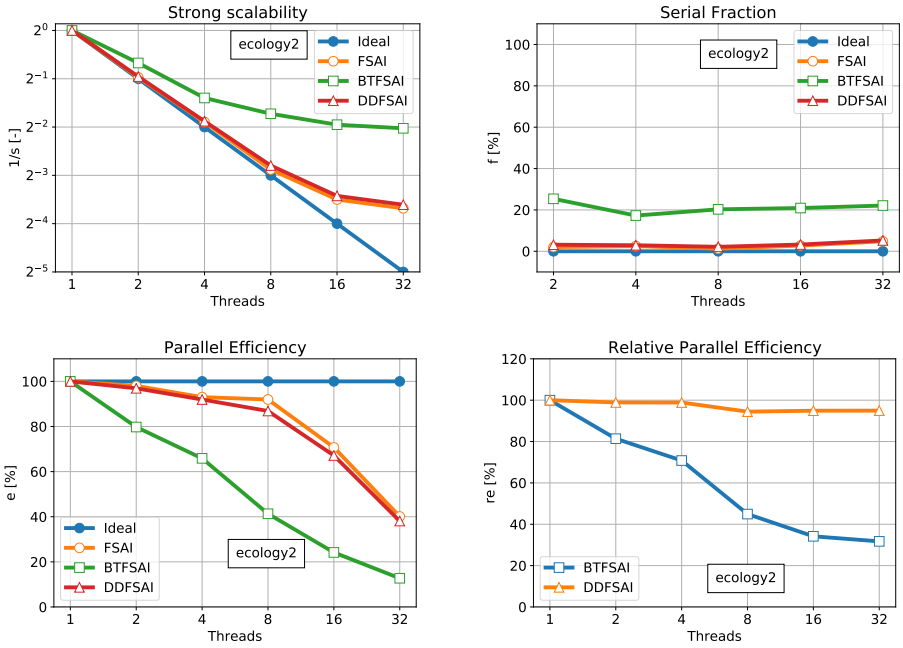

FIGURE 3.13: Scalability profiles for the matrix `ecology2`.

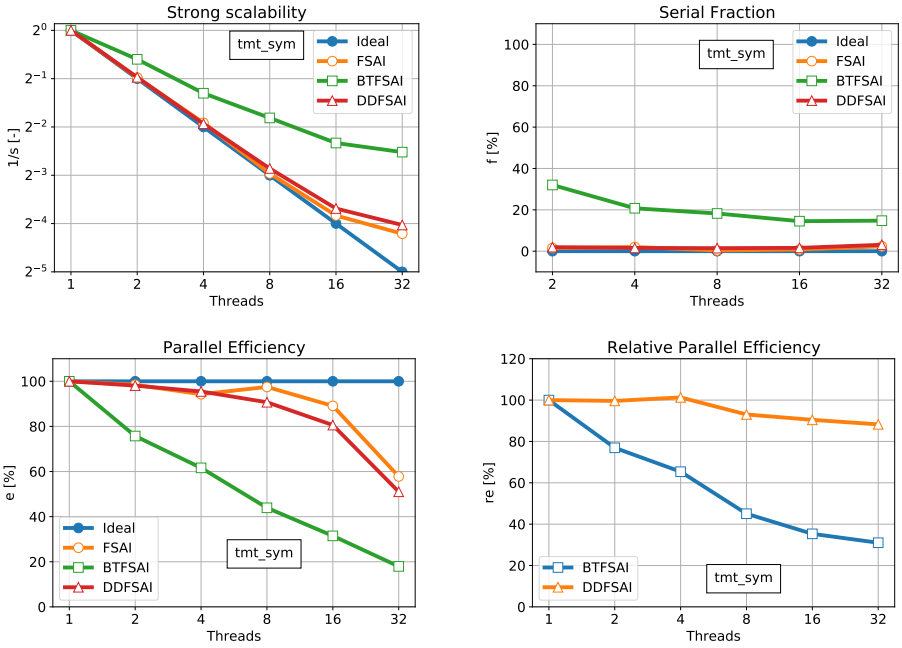

FIGURE 3.14: Scalability profiles for the matrix `tmt-sym`.

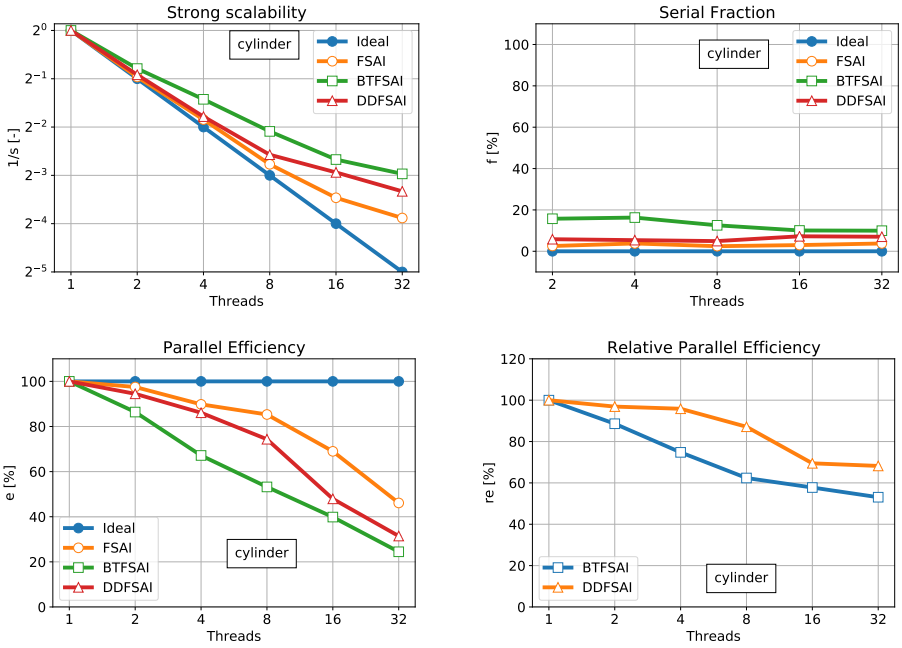

FIGURE 3.15: Scalability profiles for the matrix `cylinder`.

scalability are better in general. Finally, we note that all of the performance metrics of the DDFSAI preconditioner are very similar to the ones shown by the aFSAI preconditioner indicating that it can easily substitute the reference preconditioner.

Chapter 4

Multilevel aFSAI with Low-Rank Corrections

This chapter presents a multilevel preconditioner based on BTFSAI that enhances its robustness with the aid of low-rank corrections. An extension of the theory and results exposed in the present chapter can be found in the work by Franceschini et al. [2018].

4.1 Introduction

As discussed in the last chapter, one of the major difficulties that arise in the multilevel generalization of aFSAI is related to the accuracy in computing the Schur complements and their inverses at each level. If the earlier levels are not well approximated, the resulting Schur complement can be indefinite since it is computed as the difference between two SPD matrices and aFSAI cannot be employed for approximating the Schur complement inverses.

The mathematical reason behind this flaw relies on the poor approximation of the leftmost eigenvalues usually obtained by aFSAI. For illustrating this, we consider a system matrix A deriving from a two-dimensional Poisson problem over a 64×64 structured grid and discretized with linear triangular finite elements. We show in Figure 4.1 the exact eigenspectra of A as well as LL^T and $(G^TG)^{-1}$, i.e., the approximation of A through an Incomplete Cholesky with 10^{-2} threshold dropping, ICT(10^{-2}), and through the aFSAI preconditioner with 5 nonzeros per row, aFSAI(5), respectively. We see that both ICT and aFSAI preconditioners are not able to capture the set of smallest eigenvalues of A. Recalling the terminology of the last chapter, if we divide A

in two arbitrary partitions, the smallest eigenvalues of A_1 will be the largest ones of A_1^{-1}, and therefore controlling the most significant entries of the Schur complement $S = C - B^T K^{-1} B$. We note that this behavior is reproduced also when Incomplete Factorizations are used as the central kernel in a multilevel preconditioning framework.

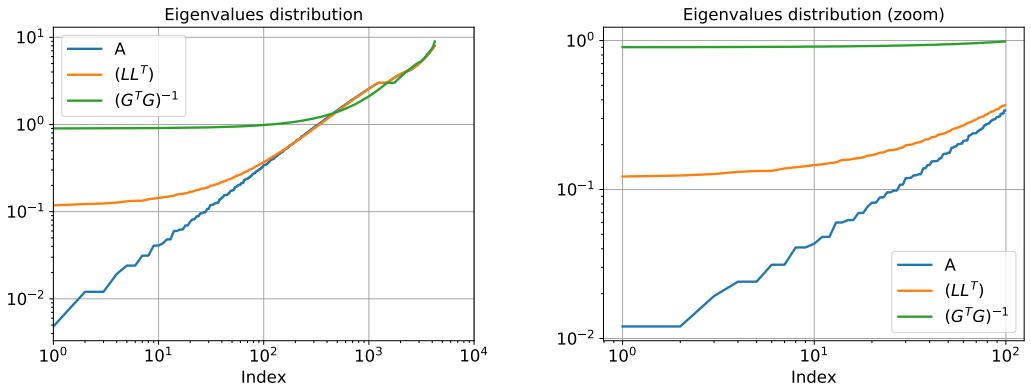

FIGURE 4.1: Comparison between the eigenspectra of A, LL^T and $(G^TG)^{-1}$ for a homogeneous Poisson test case discretized with P1 FEM in a 64×64 mesh.

To address this issue, some stabilization techniques, e.g., based on diagonal shifts Janna et al. [2009b]; Scott and Tůma [2014b], have been successfully introduced. Unfortunately, these strategies do not provide satisfactory results when used for the computation of $\widetilde{S}$ with the aid of FSAI as a kernel. A more prominent solution has been recently investigated in the context of multilevel incomplete factorizations with the assistance of low-rank corrections Xi et al. [2016]; however, no contributions were made yet to the field of sparse approximate inverses.

Low-rank compression algorithms are gaining an increasing attention especially in direct linear solution methods, e.g., Amestoy et al. [2015]; Wang et al. [2013]; Xia [2012, 2013a,b]; Xia et al. [2010], with the basic idea of taking advantage from data sparsity instead of structural sparsity. In fact, during the factorization process of a system matrix arising from a PDE problem, the off-diagonal submatrices, which are characterized by low-rank properties, are decomposed by the singular value decomposition (SVD) and compressed by neglecting those components corresponding to the smallest singular values. There are several schemes to carry out the compression on the entire matrix, ranging from hierarchical matrices ($\mathcal{H}$-matrices) Hackbusch [1999]; Börm et al. [2003]; Hackbusch [2015], HSS methods Sheng et al. [2007]; Ghysels et al. [2016]; Rouet et al.

[2016] to BLR tree structured compression Amestoy et al. [2015]; Aminfar et al. [2016]; Amestoy et al. [2017]. Typically a drop tolerance is set below which the singular values are neglected. In the case of sufficiently large drop tolerances, this process gives rise to a preconditioner, as investigated initially by Li and Saad [2013a] in the context of divide and conquer methods.

Within the multilevel aFSAI framework presented in the last chapter, low-rank corrections are introduced for both enhancing the preconditioner quality at the earlier levels (Descending Low-Rank, DLR) and improving the accuracy in the Schur complement computation (Ascending Low-Rank, ALR). This chapter is organized as follows. We start by presenting the Multilevel Factorization (MF) preconditioner featuring positive definite Schur complements by construction. Then, we demonstrate how low-rank corrections can improve the quality of the Schur matrices. The effectiveness of the approaches introduced here is evaluated through the solution of real-world test cases. Lastly, we conclude of the developments proposed in the present chapter.

4.2 Multilevel Factorization with Block FSAI

A generic remedy for solving the problem of the indefinite Schur complement relies on rewriting it as the summation, instead of subtraction of SPD matrices. In this section, we propose the MF preconditioner, a modification to the multilevel aFSAI preconditioners introduced in the last chapter with the aid of the adaptive Block FSAI algorithm Janna et al. [2010]. We start by proving a few theoretical results in order to build the modified version of multilevel aFSAI.

Theorem 4.2.1 *Let $A \in \mathbb{R}^{n\times n}$ be an SPD (2×2)-block matrix:*

$$A = \begin{bmatrix} K & B \\ B^T & C \end{bmatrix}, \tag{4.1}$$

with $K \in \mathbb{R}^{n_1 \times n_1}$, $B \in \mathbb{R}^{n_1 \times n_2}$ and $C \in \mathbb{R}^{n_2 \times n_2}$, and $V \in \mathbb{R}^{n \times n_2}$ and $D \in \mathbb{R}^{n_2 \times n}$ the 2-block rectangular matrices:

$$V = \begin{bmatrix} F^T \\ I \end{bmatrix}, \qquad D = \begin{bmatrix} 0 & Z \end{bmatrix}, \tag{4.2}$$

with $F \in \mathbb{R}^{n_2 \times n_1}$ and $Z \in \mathbb{R}^{n_2 \times n_2}$, such that the Frobenius norm $\|D - V^T L\|_F$ is minimum for any Z, L being the lower Cholesky factor of A. Then, $S = V^T AV$ is the Schur complement of A with respect to the partition (n_1, n_2).

Proof. The lower Cholesky factor of A can be written as:

$$L = \begin{bmatrix} L_K & 0 \\ B^T L_K^{-T} & L_S \end{bmatrix}, \tag{4.3}$$

where L_S is the lower Cholesky factor of the Schur complement of A, i.e., $L_S L_S^T = C - B^T K^{-1} B$. The matrix $(D - V^T L)$ is therefore:

$$D - V^T L = \begin{bmatrix} FL_K + B^T L_K^{-T}, & Z - L_S \end{bmatrix}, \tag{4.4}$$

whose Frobenius norm is minimum for any Z if $FL_K + B^T L_K^{-T} = 0$, i.e.:

$$F = -B^T K^{-1}. \tag{4.5}$$

The matrix $S = V^T AV$ reads:

$$S = FKF^T + B^T F^T + FB + C. \tag{4.6}$$

Introducing equation (4.5) into (4.6) provides $V^T AV = C - B^T K^{-1} B$.

Remark. The matrix F of Theorem 4.2.1 is generally dense. If Equation (4.5) is enforced only for the entries located in a prescribed set of positions $\widetilde{\mathcal{S}} \subset \mathcal{S} = \{(i,j) : 1 \leq i \leq n_1, 1 \leq j \leq n_2\}$, the sparsity of F can be retained at a workable level. This definition for F coincides with the Block FSAI preconditioner introduced in Janna et al. [2010] and Janna and Ferronato [2011], where the nonzero pattern $\widetilde{\mathcal{S}}$ is defined either statically

or dynamically during the computation of F. Using a sparse F into Equation (4.6) produces an approximation $\widetilde{S}$ of the exact Schur complement S of A.

Corollary 4.2.2 *The Schur complement approximation $\widetilde{S}$ computed with Equation (4.6) and a sparse Block FSAI F is SPD.*

Proof. The expression of $\widetilde{S}$ can be easily rearranged by adding and subtracting $B^T K^{-1} B$:

$$\widetilde{S} = C - B^T K^{-1} B + \left(F + B^T K^{-1}\right) K \left(K^{-T} B + F^T\right) = S + W^T K W. \tag{4.7}$$

The result immediately follows by noting that $W^T K W$ is SPD.

Based on these results, the MF preconditioner is built as follows. The zero-level preconditioner M_0^{-1} is made by two factors:

$$M_0^{-1} = P_b P_a. \tag{4.8}$$

An explicit approximation of K^{-1} is computed as $G^T G$ using an adaptive FSAI procedure Janna et al. [2015b] and introduced in P_a:

$$P_a = \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix}. \tag{4.9}$$

Then, the preconditioned matrix $P_a A P_a^T$ is computed:

$$P_a A P_a^T = \begin{bmatrix} GKG^T & GB \\ B^T G^T & C \end{bmatrix} \tag{4.10}$$

and the adaptive Block FSAI Janna and Ferronato [2011] of $P_a A P_a^T$ is computed for the second factor:

$$P_b = \begin{bmatrix} I & 0 \\ F & I \end{bmatrix}. \tag{4.11}$$

Algorithm 4 Multilevel block aFSAI setup.

procedure MF_SETUP(k_g,ρ_g,ϵ_g,n_l,A,M^{-1})
  Set $A_1 = A$
  for all $l = 1, \ldots, n_l - 1$ **do**
    Partition A_l as $\begin{bmatrix} K & B \\ B^T & C \end{bmatrix}$
    Compute the aFSAI factor G of K through the Algorithm 1
    Set $P_a = \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix}$
    Compute $P_b = \begin{bmatrix} I & 0 \\ F & I \end{bmatrix}$ the adaptive Block FSAI approximation of $P_a A_l P_a^T$
    Compute $\widetilde{S} = C + FGB + B^T G^T F^T + FGKG^T F^T$
    Set $M_l^{-1} = P_b P_a$
    Set $A_{l+1} = \widetilde{S}$
  end for
  Compute the aFSAI factor $M_{n_l}^{-1}$ of A_{n_l} through the Algorithm 1
  Set $M^{-1} = \{M_1^{-1}, M_2^{-1}, \ldots, M_{n_l}^{-1}\}$
end procedure

The first-level preconditioned matrix $M_1^{-1} A M_1^{-T}$ reads:

$$M_1^{-1} A M_1^{-T} = \begin{bmatrix} I & 0 \\ F & I \end{bmatrix} \begin{bmatrix} GKG^T & GB \\ B^T G^T & C \end{bmatrix} \begin{bmatrix} I & F^T \\ 0 & I \end{bmatrix} = \begin{bmatrix} GKG^T & -R_F^T \\ -R_F & \widetilde{S} \end{bmatrix} \tag{4.12}$$

where $R_F = -F(GKG^T) - B^T G^T$ is the residual on F, i.e., R_F approaches the null matrix as the accuracy in the computation on F increases. Finally, the (2,2) block of $M_1^{-1} A M_1^{-T}$ is the approximation of the first-level Schur complement:

$$\widetilde{S} = C + FGB + B^T G^T F^T + FGKG^T F^T \tag{4.13}$$

that becomes the new matrix for the next level. As $\widetilde{S}$ in (4.13) is SPD for any F and G (see Corollary 4.2.2), no breakdown is possible. The operations required for building the robust MF preconditioner are provided in Algorithm 4.

4.2.1 Theoretical properties

In this section, we will obtain some theoretical bounds on the eigenspectrum of the preconditioned matrix according to the different approximations introduced in the MF computation. At every level of the MF preconditioner setup, the partial factorization of

the approximated Schur complement from Equation (4.13) is computed. This operation introduces additional approximations level after level, potentially yielding to a Schur complement quite different from the exact one. Consider the first-level preconditioner M_1^{-1} of Equation (4.8):

$$M_1^{-1} = \begin{bmatrix} G & 0 \\ FG & I \end{bmatrix}. \tag{4.14}$$

The natural choice for the next level preconditioner is:

$$\widetilde{M}_2^{-1} = \begin{bmatrix} I & 0 \\ 0 & L_{\widetilde{S}}^{-1} \end{bmatrix} \tag{4.15}$$

where $L_{\widetilde{S}} L_{\widetilde{S}}^T = \widetilde{S}$. However, $\widetilde{S}$ is an approximated Schur complement. If available S, one could use

$$M_2^{-1} = \begin{bmatrix} I & 0 \\ 0 & L_S^{-1} \end{bmatrix} \tag{4.16}$$

with $L_S L_S^T = S$. Using either $\widetilde{M}_2^{-1}$ or M_2^{-1} as the next level preconditioner of A leads to a different performance. The two propositions that follow provide a theoretical upper bound for the eigenvalues of the preconditioned matrices $\widetilde{M}^{-1} A \widetilde{M}^{-T}$ and $M^{-1} A M^{-T}$, in order to allow for an a priori assessment of the preconditioner quality.

Proposition 4.2.3 *The eigenvalues λ of the preconditioned matrix $\widetilde{M}^{-1} A \widetilde{M}^{-T}$, with $\widetilde{M}^{-1} = \widetilde{M}_2^{-1} M_1^{-1}$ as defined in equations (4.14) and (4.15), satisfy:*

$$|\lambda - 1| \leq \frac{\|E_K\| + \sqrt{\|E_K\|^2 + 4\|\widetilde{Q}^T\|\|\widetilde{Q}\|}}{2} \tag{4.17}$$

where $\widetilde{Q} = G(KG^T F^T + B) L_{\widetilde{S}}^{-T} = -R_F^T L_{\widetilde{S}}^{-T}$ and $E_K = GKG^T - I$, for any consistent matrix norm.

Proof. The preconditioned matrix $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ reads:

$$\begin{aligned}\widetilde{M}^{-1}A\widetilde{M}^{-T} &= \begin{bmatrix} GKG^T & G\left(KG^TF^T+B\right)L_{\widetilde{S}}^{-T} \\ L_{\widetilde{S}}^{-1}\left(FGK+B^T\right)G^T & I \end{bmatrix} \\ &= \begin{bmatrix} I+E_K & \widetilde{Q} \\ \widetilde{Q}^T & I \end{bmatrix}\end{aligned} \tag{4.18}$$

Its eigenpairs $(\lambda, \mathbf{w})$, $\mathbf{w} = [\mathbf{u}, \mathbf{v}]^T$, satisfy by definition the relationship:

$$\begin{bmatrix} I+E_K & \widetilde{Q} \\ \widetilde{Q}^T & I \end{bmatrix}\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \lambda\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \tag{4.19}$$

which is equivalent to:

$$\begin{bmatrix} E_K & \widetilde{Q} \\ \widetilde{Q}^T & 0 \end{bmatrix}\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = (\lambda-1)\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \tag{4.20}$$

Taking consistent norms at both sides of the first and second set of equations we have:

$$\begin{cases} \|E_K\|\|\mathbf{u}\| + \|\widetilde{Q}\|\|\mathbf{v}\| \geq |\lambda-1|\|\mathbf{u}\| \\ \|\widetilde{Q}^T\|\|\mathbf{u}\| \geq |\lambda-1|\|\mathbf{v}\| \end{cases} \tag{4.21}$$

which, by setting $t = \|\mathbf{v}\|/\|\mathbf{u}\|$, can be rearranged as:

$$\begin{cases} |\lambda-1| \leq \|E_K\| + \|\widetilde{Q}\|t \\ |\lambda-1| \leq \|\widetilde{Q}^T\|/t \end{cases} \tag{4.22}$$

If $\|\mathbf{u}\| = 0$, then trivially $\mathbf{v} \in \text{Ker}(\widetilde{Q})$ and $\lambda = 1$, thus satisfying the inequality (4.17). The right-hand side of the first and second inequality in (4.22) increases and decreases monotonically with t, respectively (Figure 4.2). The intersection point is:

$$\bar{t} = \frac{-\|E_K\| + \sqrt{\|E_K\|^2 + 4\|\widetilde{Q}^T\|\|\widetilde{Q}\|}}{2\|\widetilde{Q}\|} \tag{4.23}$$

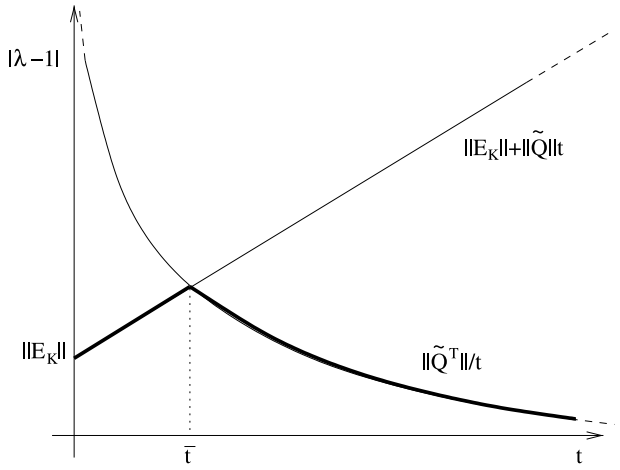

FIGURE 4.2: Schematic representation of the system of inequalities (4.22).

so for any t we have:

$$|\lambda - 1| \leq \frac{\|E_K\| + \sqrt{\|E_K\|^2 + 4\|\widetilde{Q}^T\|\|\widetilde{Q}\|}}{2} \tag{4.24}$$

Proposition 4.2.4 *The eigenvalues λ of the preconditioned matrix $M^{-1}AM^{-T}$, with $M^{-1} = M_1^{-1}M_0^{-1}$ as defined in equations (4.8) and (4.16), satisfy:*

$$|\lambda - 1| \leq \frac{\|E_K\|+\|Q^T\|\|(I+E_K)^{-1}\|\|Q\|+\sqrt{(\|E_K\|-\|Q^T\|\|(I+E_K)^{-1}\|\|Q\|)^2+4\|Q^T\|\|Q\|}}{2} \tag{4.25}$$

where $Q = G(KG^TF^T + B)L_S^{-T} = -R_F^TL_S^{-T}$ and $E_K = GKG^T - I$, for any consistent matrix norm.

Proof. Operating as in the proof of Proposition 4.2.3, it can be shown that the eigenpairs of $M^{-1}AM^{-T}$ satisfy:

$$\begin{bmatrix} I + E_K & Q \\ Q^T & I + Q^T\left(I + E_K\right)^{-1}Q \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \lambda \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \tag{4.26}$$

which is equivalent to:

$$\begin{bmatrix} E_K & Q \\ Q^T & Q^T\left(I + E_K\right)^{-1}Q \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = (\lambda - 1) \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} \tag{4.27}$$

Again, taking consistent norms at both sides of the first and second set of equations, and setting $t = \|\mathbf{v}\|/\|\mathbf{u}\|$, we get:

$$\begin{cases} |\lambda - 1| \leq \|E_K\| + \|Q\| t \\ |\lambda - 1| \leq \|Q^T\|/t + \|Q^T\| \| \left(I + E_K\right)^{-1} \| \|Q\| \end{cases} \tag{4.28}$$

If $\|\mathbf{u}\| = 0$, then trivially $\mathbf{v} \in \text{Ker}(Q)$ and $\lambda = 1$, thus satisfying the inequality (4.25). Otherwise, denoting by $\bar{t}$ the intersection point between the right-hand sides of (4.28):

$$\bar{t} = \frac{-\|E_K\| + \|Q^T\| \| \left(I + E_K\right)^{-1} \| \|Q\| + \sqrt{(\|E_K\| - \|Q^T\| \| \left(I + E_K\right)^{-1} \| \|Q\|)^2 + 4\|Q^T\|\|Q\|}}{2\|Q\|} \tag{4.29}$$

for any t we have:

$$|\lambda - 1| \leq \frac{\|E_K\| + \|Q^T\| \| (I+E_K)^{-1} \| \|Q\| + \sqrt{(\|E_K\| - \|Q^T\| \| (I+E_K)^{-1} \| \|Q\|)^2 + 4\|Q^T\| \|Q\|}}{2} \tag{4.30}$$

Now, to understand which preconditioner, either $\widetilde{M}^{-1}$ or M^{-1}, is expected to ensure a faster convergence, we compare the upper bounds provided in Propositions 4.2.3 and 4.2.4. These bounds depend on the norms of E_K and Q or $\widetilde{Q}$, which on their turn are controlled by the accuracy in the computation of G and F, respectively. As $G^T G$ approaches K^{-1}, $\|E_K\| \to 0$, and in the limit the bounds (4.17) and (4.25) read:

$$|\lambda - 1| \leq \sqrt{\|\widetilde{Q}\| \|\widetilde{Q}^T\|} \tag{4.31}$$

and

$$|\lambda - 1| \leq \frac{1}{2} \left[\|Q\| \|Q^T\| + \sqrt{\|Q\| \|Q^T\| \left(4 + \|Q\| \|Q^T\|\right)} \right] \tag{4.32}$$

respectively. Similarly, as F approaches $-B^T K^{-1} G^{-1}$, $\|Q\|, \|\widetilde{Q}\| \to 0$ and the bounds (4.17) and (4.25) trivially provide:

$$|\lambda - 1| \leq \|E_K\| \tag{4.33}$$

The previous relationships hold true for any consistent matrix norm. However, to compare the bounds in an easier way, we restrict our attention to the matrix norm induced

by the 2-norm of vectors. In this case we have:

$$\|E_K\|_2 = \lambda_1(E_K) = \epsilon_1, \quad \|Q\|_2 = \sqrt{\lambda_1(Q^TQ)} = \eta_1, \quad \|\widetilde{Q}\|_2 = \sqrt{\lambda_1(\widetilde{Q}^T\widetilde{Q})} = \tilde{\eta}_1,$$

$$\|(I+E_K)^{-1}\|_2 = \lambda_n^{-1}(GKG^T) = \kappa_n^{-1} \tag{4.34}$$

and the bounds (4.17) and (4.25) respectively become:

$$|\lambda - 1| \leq \frac{\epsilon_1 + \sqrt{\epsilon_1^2 + 4\tilde{\eta}_1^2}}{2} \tag{4.35}$$

$$|\lambda - 1| \leq \frac{\epsilon_1 + \eta_1^2\kappa_n^{-1} + \sqrt{\left(\epsilon_1 - \eta_1^2\kappa_n^{-1}\right)^2 + 4\eta_1^2}}{2} \tag{4.36}$$

Remark. When $\epsilon_1 = 0$, it is easy to prove that the maximum and minimum eigenvalues of $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ are $1 + \tilde{\eta}_1$ and $1 - \tilde{\eta}_1$, respectively, and the maximum and minimum eigenvalues of $M^{-1}AM^{-T}$ are $1 + (\eta_1^2 + \sqrt{\eta_1^4 + 4\eta_1^2})/2$ and $1 + (\eta_1^2 - \sqrt{\eta_1^4 + 4\eta_1^2})/2$, respectively.

The following results suggest the use of $\widetilde{M}^{-1}$ instead of M^{-1} as the MF preconditioner of A.

Theorem 4.2.5 *For any choice of G and F in (4.14), the bound (4.35) is narrower than or equal to the bound (4.36).*

Proof. Using the arguments of Corollary 4.2.2, it follows that $\widetilde{S} = S + \widetilde{H}$, with $\widetilde{H}$ a symmetric positive semidefinite matrix, hence $\|S\widetilde{S}^{-1}\|_2 \leq 1$. In particular:

$$\begin{aligned}
\widetilde{S} &= C + FGKG^TF^T + FGB + B^TG^TF^T = \\
&= C - B^TK^{-1}B + (F + B^TK^{-1}G^{-1})(GKG^T)(F + B^TK^{-1}G^{-1})^T = \\
&= S + R_F(GKG^T)^{-1}R_F^T
\end{aligned} \tag{4.37}$$

Moreover, the matrix $\widetilde{Q}^T\widetilde{Q}$ is similar to $L_S^T\widetilde{S}^{-1}L_SQ^TQ$. In fact, recalling that $Q = -R_F^TL_S^{-T}$ and $\widetilde{Q} = -R_F^TL_{\widetilde{S}}^{-T}$, we obtain:

$$R_F^TR_F = L_SQ^TQL_S^T = L_{\widetilde{S}}\widetilde{Q}^T\widetilde{Q}L_{\widetilde{S}}^T \tag{4.38}$$

from which the similarity follows. Hence:

$$\|\widetilde{Q}^T\widetilde{Q}\|_2 = \|L_S^T\widetilde{S}^{-1}L_SQ^TQ\|_2 \leq \|L_S^T\widetilde{S}^{-1}L_S\|_2\|Q^TQ\|_2 \leq \|Q^TQ\|_2 \tag{4.39}$$

because $L_S^T\widetilde{S}^{-1}L_S$ is similar to $S\widetilde{S}^{-1}$. As a consequence, $\tilde{\eta}_1 = \alpha\eta_1$ for some $\alpha \leq 1$. The thesis of the theorem reads:

$$\frac{\epsilon_1 + \sqrt{\epsilon_1^2 + 4\tilde{\eta}_1^2}}{2} \leq \frac{\epsilon_1 + \eta_1^2\kappa_n^{-1} + \sqrt{\left(\epsilon_1 - \eta_1^2\kappa_n^{-1}\right)^2 + 4\eta_1^2}}{2} \tag{4.40}$$

Introducing $\tilde{\eta}_1 = \alpha\eta_1$ in (4.40), after some algebra we obtain:

$$\alpha^2 \leq 1 + \frac{\sqrt{\left(\epsilon_1 - \eta_1^2\kappa_n^{-1}\right)^2 + 4\eta_1^2} - \left(\epsilon_1 - \eta_1^2\kappa_n^{-1}\right)}{2} \tag{4.41}$$

which holds true for any F and G.

Theorem 4.2.5 suggests that the use of $\widetilde{S}$ in the MF preconditioner is likely to be more appropriate than the exact Schur complement S. In particular, in the theoretical case of $G^TG = K^{-1}$ it is possible to compute explicitly the ratio between the conditioning numbers of the preconditioned matrices $M^{-1}AM^{-T}$ and $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ as follows.

Theorem 4.2.6 *If $E_K = 0$, the ratio between the conditioning number of the preconditioned matrices (4.26) and (4.18) is:*

$$r(\eta_1) = \frac{\left(1 + \dfrac{\eta_1^2 + \sqrt{\eta_1^4 + 4\eta_1^2}}{2}\right)^2}{1 + 2\eta_1^2 + 2\sqrt{\eta_1^4 + \eta_1^2}} \tag{4.42}$$

Proof. If $E_K = 0$, it can be easily verified from Equation (4.37) that $\widetilde{S} = S + R_F R_F^T$ and:

$$L_S^{-1}\widetilde{S}L_S^{-T} = I + L_S^{-1}R_F R_F^T L_S^{-T} = I + Q^T Q \tag{4.43}$$

Recalling from the proof of Theorem 4.2.5 that $\widetilde{Q}^T\widetilde{Q}$ is similar to $L_S^T\widetilde{S}^{-1}L_S Q^T Q$ we have:

$$\|\widetilde{Q}^T\widetilde{Q}\|_2 = \|(I + Q^T Q)^{-1}Q^T Q\|_2 \tag{4.44}$$

Trivially, $(I+Q^TQ)^{-1}Q^TQ$ is symmetric positive definite and has the same eigenvectors as Q^TQ. Denoting by λ an eigenvalue of Q^TQ, the norm $\|(I + Q^TQ)^{-1}Q^TQ\|_2$ is the maximum of the function:

$$f(\lambda) = \frac{\lambda}{1+\lambda} \tag{4.45}$$

As $f(\lambda)$ monotonically increases with λ, its maximum value is attained for the largest eigenvalue of Q^TQ, i.e., $\|Q^TQ\|_2 = \eta_1^2$. Hence:

$$\|\widetilde{Q}^T\widetilde{Q}\|_2 = \frac{\|Q^TQ\|_2}{1+\|Q^TQ\|_2} \Rightarrow \tilde{\eta}_1 = \sqrt{\frac{\eta_1^2}{1+\eta_1^2}} \tag{4.46}$$

The proof is completed by introducing Equation (4.46) in the results of Remark 4.2.1.

Remark. The ratio $r(\eta_1)$ monotonically increases with η_1 and takes value 1 for $\eta_1 = 0$, i.e., $F = -B^TK^{-1}G^{-1}$ and $\widetilde{S} = S$. For $\eta_1 \to \infty$, r monotonically diverges to infinity as the second power of η_1. Hence, the sparser or more inaccurate F, the more important is using $\widetilde{S}$ instead of S in the MF preconditioner.

4.3 MF with Low-Rank corrections

The framework developed in the last section provides the formulation for a robust multilevel preconditioner that can be computed in a stable way for any configuration of the aFSAI factors. Using low fill-in degrees obviously keeps the computational cost of the preconditioner setup and application under control at the cost of a poor convergence. The quality of the MF preconditioner can be improved by using low-rank corrections giving rise to the MFLR scheme. The basic concept of combining low-rank corrections

to multilevel preconditioners is given as follows. Define the matrices:

$$Y = L_C^{-1} B^T K^{-1} B L_C^{-T} = L_C^{-1} (C - S) L_C^{-T} \tag{4.47}$$

and

$$X = L_C^T \left(S^{-1} - C^{-1} \right) L_C \tag{4.48}$$

where L_C is the exact lower factor of C, i.e., $C = L_C L_C^T$. It is straighforward to see that the eigenvalues σ_i of Y follow the inequality:

$$0 < \sigma_{n_2} \leq \cdots \leq \sigma_1 < 1. \tag{4.49}$$

Theorem 4.3.1 *The eigenvalues σ_i of matrix Y and θ_i of matrix X are related through the formula:*

$$\theta_i = \frac{\sigma_i}{1 - \sigma_i} \qquad i = 1, \ldots, n_2 \tag{4.50}$$

Proof. Note that $(I - Y) = L_C^{-1} S L_C^{-T}$ and $(I + X) = L_C^T S^{-1} L_C$, thus:

$$(I - Y)^{-1} = I + X. \tag{4.51}$$

From the Sherman-Morrison-Woodbury (SMW) formula, the matrix inverse on the left-hand side of Equation (4.51) can be written as

$$(I - Y)^{-1} = I + \left(Y^{-1} - I \right)^{-1}. \tag{4.52}$$

Substituting this equivalence in Equation (4.51), we have:

$$X = \left(Y^{-1} - I \right)^{-1}. \tag{4.53}$$

Finally, if $\mathbf{p}_i$ is an eigenvector of Y associated with the eigenvalue σ_i:

$$\begin{aligned} & Y\mathbf{p}_i = \sigma_i \mathbf{p}_i \\ \rightarrow \quad & Y^{-1}\mathbf{p}_i = (1/\sigma_i)\,\mathbf{p}_i \\ \rightarrow \quad & \left(Y^{-1} - I\right)\mathbf{p}_i = (1/\sigma_i - 1)\,\mathbf{p}_i \\ \rightarrow \quad & \left(Y^{-1} - I\right)^{-1}\mathbf{p}_i = (1/\sigma_i - 1)^{-1}\,\mathbf{p}_i \\ \rightarrow \quad & X\mathbf{p}_i = (1/\sigma_i - 1)^{-1}\,\mathbf{p}_i \\ \therefore \quad & \theta_i = (1/\sigma_i - 1)^{-1} = \frac{\sigma_i}{1-\sigma_i}. \end{aligned} \tag{4.54}$$

From Theorem 4.3.1, the distance between consecutive eigenvalues of X is larger than those of Y. Moreover, their difference can be written as

$$\theta_i - \theta_{i+1} = \frac{\sigma_i - \sigma_{i+1}}{(1-\sigma_i)\,(1-\sigma_{i+1})} \qquad i = 1, \ldots, (n_2 - 1). \tag{4.55}$$

The separation of the eigenvalues of $L_C^{-T} X L_C^{-1} = S^{-1} - C^{-1}$ has a stronger impact on the performance of the MSLR preconditioner Xi et al. [2016], however studying X is easier and the main results for X are at some extent still valid for $S^{-1} - C^{-1}$. Equation (4.55) suggests that approximating with a low-rank matrix $(S^{-1} - C^{-1})$ is easier than $(S - C)$ because of the faster eigenvalue decay. A better approximation of S^{-1} can be computed as:

$$S^{-1} \simeq C^{-1} + W_k \Theta_k W_k^T \tag{4.56}$$

with $W_k \Theta_k W_k^T$ a rank-k approximation of $L_C^{-T} X L_C^{-1}$ which can be obtained from the eigendecomposition of Y. In fact, by retaining the k largest eigenvalues and corresponding eigenvectors of Y we can write:

$$Y \simeq U_k \Sigma_k U_k^T \tag{4.57}$$

Noting that:

$$S^{-1} - C^{-1} = L_C^{-T}[(I-Y)^{-1} - I]L_C^{-1} = L_C^{-T}[Y(I-Y)^{-1}]L_C^{-1} \tag{4.58}$$

the rank-k correction to $S^{-1} - C^{-1}$ is found by setting:

$$\Theta_k = \Sigma_k (I - \Sigma_k)^{-1} \tag{4.59}$$

and $W_k = L_C^{-T} U_k$.

In the original formulation outlined above, the low rank corrections are used to make the action of C^{-1} closer to that of S^{-1}. By distinction, we use low-rank corrections to improve the action of $\widetilde{S}^{-1}$. A consequence of Corollary 4.2.2 is that the eigenvalues σ_i of the matrix:

$$\overline{Y} = L_{\widetilde{S}}^{-1} (\widetilde{S} - S) L_{\widetilde{S}}^{-T} \tag{4.60}$$

satisfy the condition (4.49). Thus, following the procedure outlined above for C, we can compute the k largest eigenpairs of $\overline{Y}$:

$$\overline{Y} \simeq \overline{U}_k \overline{\Sigma}_k \overline{U}_k^T \tag{4.61}$$

and get the expression of the corrected Schur complement inverse:

$$S^{-1} \simeq \widetilde{S}^{-1} + \overline{W}_k \overline{\Theta}_k \overline{W}_k^T \tag{4.62}$$

where $\overline{\Theta}_k = \overline{\Sigma}_k (I - \overline{\Sigma}_k)^{-1}$ and $\overline{W} = L_{\widetilde{S}}^{-T} \overline{U}$. However, the idea of correcting the application of $\widetilde{S}^{-1}$ so as to better resemble the one of S^{-1} is not that good. First, the computation of (4.61) may be quite expensive, since every multiplication by S requires a solution of a linear system with K. Second, according to Theorems 4.2.5 and 4.2.6, the use of S^{-1} in our multilevel framework is not optimal.

The low-rank corrections can be more effectively implemented in this other way. Since we are working in a multilevel framework, $\widetilde{S}^{-1}$ will not be used exactly. Rather, a new approximation, say $\widehat{S}^{-1} \simeq \widetilde{S}^{-1}$, will be computed. Thus, $\widehat{S}$ will be the new target of the low-rank correction. Moreover, as shown by equation (4.17), reducing $\|E_K\|$ is also useful for improving the convergence. Also this task can be performed by a low-rank correction. Therefore, we use two low-rank correction techniques for the preconditioner setup:

- *Descending low-rank corrections*: computed at each level, from the first to the last, to reduce $\|E_K\|$;
- *Ascending low-rank corrections*: computed at each level, from the last to the first, to reduce the gap between $\widehat{S}$ and $\widetilde{S}$.

4.3.1 Descending Low-Rank corrections

The aim of this correction is to enhance the approximation of the inverse of K. We can define the matrix:

$$\overline{Y} = G\left[\left(G^TG\right)^{-1} - K\right]G^T = I - GKG^T \tag{4.63}$$

obtained from Equation (4.60) where $\left(G^TG\right)^{-1}$ and K replace $\widetilde{S}$ and S, respectively, and compute its rank-k approximation:

$$\overline{Y} \simeq \overline{U}_k\overline{\Sigma}_k\overline{U}_k^T \tag{4.64}$$

Note that the computation of $\overline{U}_k$ and $\overline{\Sigma}_k$ is less expensive than in (4.61), because both G and K are explicitly known. The enhanced preconditioner for K reads:

$$K^T \simeq \left(G^TG\right) + \overline{W}_k\overline{\Theta}_k\overline{W}_k^T \tag{4.65}$$

The eigenvalues of $\overline{Y}$ are bounded from above by one as GKG^T is positive definite, but there is not a lower bound in this case. Actually this is not a problem, as we are mainly interested in the computation of the eigenvalue $\overline{\sigma}_i$ of $\overline{Y}$ closest to one. From the implementation point of view, it is better to dispose of a symmetrically splitted operator. Hence, we define:

$$\widetilde{G} = (I + \overline{U}_k\overline{\Psi}_k\overline{U}_k^T)G \tag{4.66}$$

in order to have:

$$\widetilde{G}^T\widetilde{G} = G^T(I + \overline{U}_k\overline{\Psi}_k\overline{U}_k^T)(I + \overline{U}_k\overline{\Psi}_k\overline{U}_k^T)G = G^TG + \overline{W}_k\overline{\Theta}_k\overline{W}_k^T \tag{4.67}$$

Recalling that $\overline{W}_k = G^T\overline{U}_k$, the diagonal $k \times k$ matrix $\overline{\Psi}_k$ is simply found by solving:

$$I + 2\overline{U}_k\overline{\Psi}_k\overline{U}_k^T + \overline{U}_k\overline{\Psi}_k^2\overline{U}_k^T = I + \overline{U}_k\overline{\Theta}_k\overline{U}_k^T \tag{4.68}$$

Using Equation (4.55), the entries of $\overline{\Psi}_k$ read:

$$\overline{\psi}_i = -1 + \sqrt{\frac{1}{1-\overline{\sigma}_i}}, \qquad i = 1, \dots, k \tag{4.69}$$

Since GKG^T is positive definite, $\overline{\sigma}_i < 1$ and $\overline{\psi}_i$ is real for any i.

This correction can significantly reduce $\|E_K\|$. However, the update in $\widetilde{G}$ propagates in the other blocks of the preconditioned matrix potentially shattering the overall procedure efficiency:

$$\begin{aligned} P_a A P_a^T = \begin{bmatrix} \widetilde{G}K\widetilde{G}^T & \widetilde{G}B \\ B^T\widetilde{G}^T & C \end{bmatrix} = \begin{bmatrix} GKG^T & GB \\ B^TG^T & C \end{bmatrix} + \\ + \begin{bmatrix} \overline{U}_k\overline{\Psi}_k\overline{U}_k^TGKG^T + GKG^T\overline{U}_k\overline{\Psi}_k\overline{U}_k^T + \overline{U}_k\overline{\Psi}_k\overline{U}_k^TGKG^T\overline{U}_k\overline{\Psi}_k\overline{U}_k^T & \overline{U}_k\overline{\Psi}_k\overline{U}_k^TGB \\ B^TG^T\overline{U}_k\overline{\Psi}_k\overline{U}_k^T & C \end{bmatrix} \end{aligned} \tag{4.70}$$

Actually, this is not the case. In fact, the Block FSAI $\widetilde{F}$ computed as the (2,1) block of P_b when using $\widetilde{G}$ is the approximate solution of the multiple right-hand-side system:

$$\widetilde{F}^T \simeq -(\widetilde{G}K\widetilde{G}^T)^{-1}\widetilde{G}B = -\widetilde{G}^{-T}K^{-1}B \tag{4.71}$$

By expanding Equation (4.71) with $\widetilde{G}$ definition, we note that $\widetilde{F}$ can be easily found as:

$$\widetilde{F} = F(I + \overline{U}_k\overline{\Psi}_k\overline{U}_k^T)^{-1} \tag{4.72}$$

where F is the standard Block FSAI computed using G. Moreover, from (4.72) and (4.66) we notice that $FG = \widetilde{F}\widetilde{G}$. This implies that the approximate Schur complement (4.13) is not affected by the use of $\widetilde{F}$ and $\widetilde{G}$:

$$\widetilde{S} = C + \widetilde{F}\widetilde{G}B + B^T\widetilde{G}^T\widetilde{F}^T + \widetilde{F}\widetilde{G}K\widetilde{G}^T\widetilde{F}^T = C + FGB + B^TG^TF^T + FGKG^TF^T \tag{4.73}$$

As a consequence, descending low-rank corrections on G have only a local impact with no changes for the following levels.

4.3.2 Ascending Low-Rank corrections

We define the matrix $\widehat{Y}$ and compute its rank-k approximation:

$$\widehat{Y} = I - \widehat{G}\widetilde{S}\widehat{G}^T \simeq \widehat{U}_k\widehat{\Sigma}_k\widehat{U}_k^T \tag{4.74}$$

where $\widehat{G}$ is the lower inverse factor of $\widehat{S}$, i.e. $(\widehat{G}^T\widehat{G})^{-1} = \widehat{S}$, which is explicitly available by the approximation of lower levels. The computation of (4.74) is relatively cheap, because the explicit expression of every matrix is known. The new approximation to $\widetilde{S}$ is given by:

$$\widetilde{S}^{-1} \simeq \widehat{G}^T\widehat{G} + \widehat{W}_k\widehat{\Theta}_k\widehat{W}_k^T \tag{4.75}$$

where $\widehat{W}_k = \widehat{G}^T\widehat{U}_k$ and $\widehat{\Theta}_k = \widehat{\Sigma}_k(I - \widehat{\Sigma}_k)^{-1}$. Notice that, during the setup, we use a splitted update, as done for the descending low-rank corrections, because it is operatively necessary to compute $\widehat{G}\widetilde{S}\widehat{G}^T$. However, during the preconditioner application, the use of Equation (4.75) is more efficient as it only requires one update.

4.3.3 MFLR algorithm

The multilevel aFSAI preconditioner with Low-Rank corrections (MFLR) can be constructed through the recursive Algorithm 5. The input parameters for its configuration are listed as follows:

1. n_l: number of levels composing the preconditioner hierarchy.
2. ϵ_g: drop tolerance for computing the aFSAI factor G, see Janna et al. [2015b];
3. ϵ_f: drop tolerance for computing the block aFSAI factor F, see Janna and Ferronato [2011];
4. n_{dlr}: size of the descending low-rank correction, i.e., number of eigenpairs used to enrich G;

Algorithm 5 MFLR setup.

procedure MFLR_SETUP(l,n_l,n_{alr},n_{dlr},ϵ_g,ϵ_f,A)
  if $l < (n_l - 1)$ **then**
    Partition A_l as $\begin{bmatrix} K & B \\ B^T & C \end{bmatrix}$
    Compute the aFSAI factor G of K through the Algorithm 1
    Set $P_a = \begin{bmatrix} G & 0 \\ 0 & I \end{bmatrix}$
    Compute $P_b = \begin{bmatrix} I & 0 \\ F & I \end{bmatrix}$, the block aFSAI approximation of $P_a A_l P_a^T$
    Compute $\widetilde{S} = C + FGB + B^T G^T F^T + FGKG^T F^T$
    Compute a rank k approximation $\overline{U}_k \overline{\Sigma}_k \overline{U}_k^T$ of $\overline{Y} = I - GKG^T$
    Set $\widetilde{G} = (I + \overline{U}_k \overline{\Psi}_k \overline{U}_k^T)G$ with $\overline{\Psi}_k = (I - \overline{\Sigma}_k)^{-1/2} - I$
    Set $P = \begin{bmatrix} \widetilde{G} & 0 \\ FG & I \end{bmatrix}$
    $[Q_{alr},\ Q_M]$ = MFLR_SETUP($l+1$,n_l,n_{alr},n_{dlr},ϵ_g,ϵ_f,$\widetilde{S}$)
    Use Q_{alr} and Q_M to compute the rank k correction $\widehat{W}_k$ and $\widehat{\Theta}_k$ as in (4.75)
    Set $\widetilde{P} = \begin{bmatrix} I & 0 \\ 0 & I + \widehat{W}_k \widehat{\Theta}_k \widehat{W}_k^T \end{bmatrix}$
    Push $\widetilde{P}$ in the head of Q_{alr}
    Push P in the head of Q_M
  else if $l = n_L$ **then**
    Compute the aFSAI factor G of A through the Algorithm 1
    Set $Q_{alr} = \varnothing$
    Set $Q_M = \{G\}$
  end if
  return $Q_{alr},\ Q_M$
end procedure

5. n_{alr}: size of the ascending low-rank correction, i.e., number of eigenpairs used to enrich $\widehat{S}^{-1}$.

For simplicity, we set the maximum number of nonzeros per row and the number of coeficients added to the aFSAI factors at each inner cycle of the Algorithm 1 as unlimited and one, respectively.

The application of MFLR to a random vector is performed recursively for each level of the preconditioner hierarchy as given by the Algorithm 6.

We note that, in this work, the Algorithms 5 and 6 are implemented computationally by using the Fortran 90 language and OpenMP directives for their parallelization in shared memory architectures.

Algorithm 6 MFLR application.

procedure MFLR_APPLY(l,n_l,Q_{alr},Q_M,$\mathbf{x}$)
  if $l < (n_l - 1)$ **then**
    Pop P from the head of Q_M
    Pop $\widetilde{P}$ from the head of Q_{alr}
    Compute $\mathbf{y} = P\mathbf{x}$
    Partition $\mathbf{y}$ into $\mathbf{y}_1 = \mathbf{y}(1 : n_1)$ and $\mathbf{y}_2 = \mathbf{y}(n_1 + 1 : n_1 + n_2)$
    Compute $\mathbf{z}_2$ = MFLR_APPLY($l + 1$,n_l,Q_{alr},Q_M,$\mathbf{y}_2$)
    Form $\mathbf{z}$ with $\mathbf{y}_1$ and $\mathbf{z}_2$
    Compute $\mathbf{y} = \widetilde{P}\mathbf{z}$
    Update $\mathbf{y} \leftarrow P^T\mathbf{y}$
    Push P in the head of Q_M
    Push $\widetilde{P}$ in the head of Q_{alr}
  else if $l = n_L$ **then**
    Pop G from the head of Q_M
    Compute $\mathbf{y} = G^T G\mathbf{x}$
    Push G in the head of Q_M
  end if
  return $\mathbf{y}$
end procedure

4.4 Numerical results

In this section, we investigate the behavior of MFLR for the solution of linear systems along with the preconditioned conjugate gradient method. First, we verify experimentally the theoretical properties MFLR presented in section 4.3. Next, we perform a sensitivity analysis on the input parameters for building MFLR in order to better understand how they control its quality and computational efficiency. Lastly, we employ this preconditioner for the solution of real-world test problems with large size.

4.4.1 Theoretical properties

As our model problem, we choose the `bcsstk38` SPD matrix Davis and Hu [2011] with 8,032 rows and 355,460 nonzeroes in its scaled form so as to have unitary diagonal. We refer this matrix as A. Assuming a uniform partioning into two levels ($n_1 = n_2 = 4,016$), we calculate the exact Schur complement S and the approximate Schur complement $\widetilde{S}$ through Equation (4.6) where F is the adaptive block FSAI of A Janna and Ferronato [2011] built with a variable number $k_{F,max}$ of entries retained per

row. Figure 4.3 shows the complete eigspectrum of A in the left and the nonzero eigenspectrum of $(\widetilde{S} - S)$. As expected by Corollary 4.2.2, the last quantity is strictly positive for any choice of F.

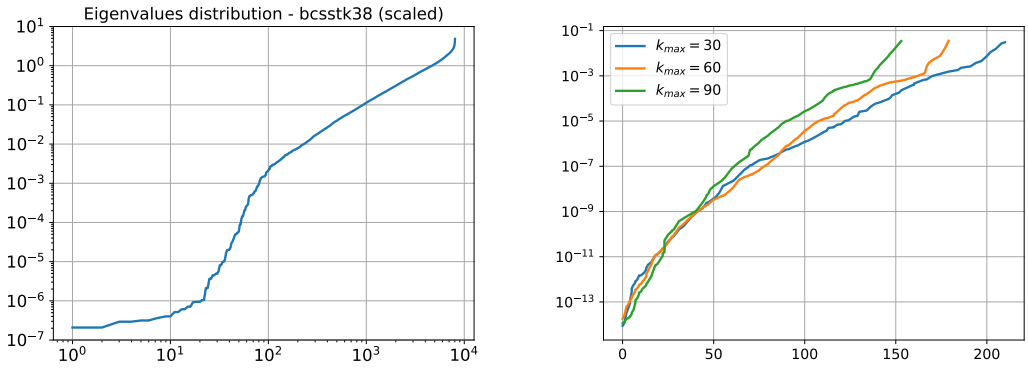

FIGURE 4.3: `bcsstk38` test case: eigenspectrum of A (left) and $\left(\widetilde{S} - S\right)$ (right) for different configurations of the adaptive block FSAI factor F.

The analysis is performed by introducing new ingredients to the MFLR preconditioner step-by-step. In particular:

1. the exact inverse of K is used to evaluate the effect of using either $\widetilde{S}$ or S;
2. the approximation $G^T G \simeq K^{-1}$ is introduced;
3. low-rank corrections are added to improve the preconditioner

First, we want to verify that the main claim of Section 4.2, i.e., the use of $\widetilde{S}$ is more appropriate than that of S in the presented multilevel framework. With this aim, we report in Table 4.1 the ratio between the conditioning numbers of $M^{-1}AM^{-T}$ and $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ for different configurations of the block FSAI factor F. Should F be computed exactly as a full matrix, there would be no difference between $\widetilde{M}^{-1}$ and M^{-1}. In constrast, decreasing the quality of F means increasing $\eta_1 = \|Q\|_2$ and $\tilde{\eta}_1 = \|\widetilde{Q}\|_2$. As a consequence, the effectiveness of $\widetilde{M}^{-1}$ improves with respect to M^{-1}, i.e., the sparser F, the more important is to use $\widetilde{S}$ instead of S. For this test problem, the ratio $r(\eta_1)$ between the conditioning numbers of the matrix $M^{-1}AM^{-T}$ from Equation (4.26) and $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ from Equation (4.18) increases up to 5.709.

Theorem 4.2.6 no longer holds if we introduce the approximation $G^T G$ for K^{-1}. However, it is still true that the theoretical eigenvalue bounds for $\widetilde{M}^{-1}$ are tighter than

TABLE 4.1: `bcsstk38` test case: comparison between the conditioning numbers of (4.26) and (4.18) by varying F with $\|E_K\| = 0$. λ and $\tilde{\lambda}$ denote the eigenvalues of $M^{-1}AM^{-T}$ and $\widetilde{M}^{-1}A\widetilde{M}^{-T}$, respectively.

$k_{F,\max}$	η_1	$\tilde{\eta}_1$	$[\lambda_n, \lambda_1]$	$[\tilde{\lambda}_n, \tilde{\lambda}_1]$	$r(\eta_1)$
30	4.395	0.975	[0.0470, 21.273]	[0.0249, 1.975]	5.709
60	2.867	0.944	[0.0988, 10.121]	[0.0558, 1.944]	2.939
90	2.248	0.914	[0.1448, 6.9069]	[0.0863, 1.914]	2.153

those for M^{-1}. The matrix G is computed as the aFSAI factor of K, thus, its quality is controlled mainly by the maximum number of entries $k_{G,max}$ retained per row. First of all, notice that $\widetilde{S}$ computed as in equation (3.17) might be indefinite if G is not accurate enough. For instance, even with $k_{G,max} = 250$ the approximate Schur complement $\left(C - B^T G^T G B\right)$ still has one negative eigenvalue.

Changing the fill-in degree of G modifies the values obtained from the bounds (4.35) and (4.36), which become tighter as $k_{G,max}$ increases. Such bounds along with the actual eigenvalue intervals are reported in Table 4.2 for different choices of G and F. It can be observed that the role played by ϵ_1 and $\tilde{\eta}_1$, i.e., a measure of the quality of G and F, respectively, has a similar impact on the actual eigenvalue distribution of $\widetilde{M}^{-1}A\widetilde{M}^{-T}$, as expected from the right-hand side of inequality (4.35). Hence, one may argue that improving both G and F is essential for a better MF performance. On the other hand, the bound (4.36) is less significant because of the presence of κ_n^{-1}, which can be quite large. Nonetheless, the actual eigenvalue distribution of $M^{-1}AM^{-T}$ is in any case worse than that of $\widetilde{M}^{-1}A\widetilde{M}^{-T}$.

Finally, we analyze the Descending and Ascending Low-Rank corrections. We fix $k_{G,max} = 30$ and $k_{F,max} = 60$, and change the number of eigenpairs computed to correct either G (dlr) or $\widehat{S}^{-1}$ (alr). Their effect on the variation of the conditioning number of $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ is shown in Table 4.3. Using both low-rank strategies greatly helps reduce the conditioning number. For example, in this case, retaining 20 eigenpairs for both corrections yields a reduction of the conditioning number of about two orders of magnitude, i.e., from about 24,000 to 500. By distinction, notice that correcting $\widetilde{S}^{-1}$ towards S^{-1} is not as effective, as expected from the MFLR theoretical properties. It is

TABLE 4.2: `bcsstk38` test case: eigenvalue distribution of $M^{-1}AM^{-T}$ and $\widetilde{M}^{-1}A\widetilde{M}^{-T}$ while varying F and G. The same notation of Table 4.1 is used.

$k_{G,\max}$	$k_{F,\max}$	ϵ_1	η_1	$\tilde{\eta}_1$	$[\lambda_n, \lambda_1]$	$[\tilde{\lambda}_n, \tilde{\lambda}_1]$	bound (4.36)	bound (4.35)
30	30	1.08	4.50	0.81	[1.8e-4, 47.3]	[1.7e-4, 2.1]	19473.3	2.51
30	60	1.08	3.18	0.79	[1.8e-4, 33.2]	[1.8e-4, 2.1]	9707.6	2.49
30	90	1.08	2.62	0.65	[1.8e-4, 29.1]	[1.8e-4, 2.1]	6570.8	2.39
60	30	1.24	4.39	0.72	[3.9e-4, 40.0]	[3.9e-4, 2.2]	8291.3	2.57
60	60	1.24	3.04	0.64	[4.0e-4, 29.5]	[4.0e-4, 2.2]	3963.8	2.51
60	90	1.24	2.47	0.59	[4.0e-4, 25.8]	[4.0e-4, 2.2]	2613.6	2.47
90	30	1.09	4.30	0.74	[6.6e-4, 36.0]	[6.5e-4, 2.1]	5353.9	2.46
90	60	1.09	2.94	0.65	[6.8e-4, 26.8]	[6.7e-4, 2.1]	2495.6	2.40
90	90	1.09	2.43	0.62	[6.9e-4, 23.3]	[6.8e-4, 2.1]	1716.8	2.37

also interesting to observe that the largest eigenvalue is the most sensitive to the selection of the target matrix for the Ascending Low-Rank corrections, while it is insensitive to the Descending Low-Rank corrections, which mainly affect the smallest eigenvalue.

4.4.2 Sensitivity analysis

To test the influence of the user-specified parameters controlling the behavior of MFLR, we employ a medium-sized test case referred as `cube190k` arising from discretizing the linear elasticity model in a unitary cube with homogeneous material properties by P1 finite elements. The resulting system matrix has 190,581 unknowns and 7,531,389 nonzero entries. We use a PCG iterative solver preconditioned with MFLR with a convergence tolerance based on the relative residual norm equal to 10^{-8}. The right-hand side vector $\mathbf{b}$ is such that the components of the solution $\mathbf{x}$ are $x_j = j + 1$, $j = 1, \ldots, n$. All tests reported here are obtained using a machine equipped with Intel(R) Xeon(R) E5-2680 v2 processors at 2.80 GHz and 256 Gbyte of RAM. Each CPU has ten cores. For these preliminary tests, just one thread is used. For the computation of the eigenpairs needed by the Low-Rank corrections, we used the Laneig software, that is part of the Filtlan package Fang and Saad [2012].

The `cube190k` test matrix is reordered with the Reverse Cuthill-McKee algorithm and uniformly partitioned into equal-size levels. The results are evaluated in terms of

TABLE 4.3: `bcsstk38` test case: effect of Descending and Ascending Low-Rank corrections, dlr and alr, respectively, with $k_{G,max} = 30$ and $k_{F,max} = 60$. The target of Ascending Low-Rank correction is $\widetilde{S}^{-1}$ and S^{-1} in the upper and lower table, respectively.

alr target	alr	dlr	$[\lambda_n, \lambda_1]$	λ_1/λ_n
$\widetilde{S}^{-1}$	0	0	[1.682e-04, 2.337e+00]	2.389e+04
	0	10	[4.391e-04, 2.337e+00]	5.321e+03
	0	20	[4.438e-04, 2.337e+00]	5.265e+03
	10	0	[1.682e-04, 2.076e+00]	1.234e+04
	10	10	[4.391e-04, 2.077e+00]	4.731e+03
	10	20	[4.438e-04, 2.137e+00]	4.816e+03
	20	0	[1.744e-04, 2.076e+00]	1.190e+04
	20	10	[3.380e-03, 2.077e+00]	6.145e+02
	20	20	[4.413e-03, 2.137e+00]	4.843e+02
S^{-1}	10	0	[1.753e-04, 1.467e+01]	8.371e+04
	10	10	[4.342e-03, 1.482e+01]	3.414e+03
	10	20	[6.308e-03, 1.493e+01]	2.367e+03
	20	0	[1.772e-04, 3.220e+01]	1.817e+05
	20	10	[5.475e-03, 3.270e+01]	5.973e+03
	20	20	[9.065e-03, 3.290e+01]	3.630e+03

number of iterations, n_{it}, time needed to compute the preconditioner, T_p, the time spent in the PCG iterations, T_s, and the preconditioner density, ρ, defined as:

$$\rho = \frac{1}{\text{nnz}(A)} \sum_{i=1}^{n_l} \left(\text{nnz}(Q_{alr}^i) + \text{nnz}(Q_M^i)\right) \tag{4.76}$$

where n_l is the number of levels and the matrices Q_{alr}^i and Q_M^i are defined in Algorithm 5. For these preliminary tests, the sparse matrix/matrix products are fully computed, i.e., any use of thresholds and/or levels of fill is avoided.

Similarly to the previous section, the analysis is carried out step-by-step. First, the performance of the robust MF preconditioner with no low-rank corrections (Algorithm 4) is investigated by changing the fill-in degree of G and F. Then, low-rank corrections are added to clarify their effect. It is worth noting that only the robust MF preconditioner is addressed here because the standard BTFSAI preconditioner (Algorithm 2) breaks down due to the computation of indefinite Schur complements.

Table 4.4 shows the results obtained by varying the number of levels n_l and the

TABLE 4.4: `cube190k` test case: performance of MFLR without low-rank corrections as a function of n_l and ϵ_G ($\epsilon_F = 10^{-2}$, $dlr = alr = 0$).

ϵ_G	n_l	n_{it}	ρ	T_p	T_s	ϵ_G	n_l	n_{it}	ρ	T_p	T_s
10^{-1}	10	930	0.43	14.5	26.8	10^{-2}	10	640	0.77	34.5	21.9
	20	843	0.73	132.0	34.3		20	591	1.17	265.9	29.3
	50	664	1.47	349.9	49.1		50	503	2.04	441.7	41.6
	100	612	2.78	515.1	79.2		100	464	3.67	612.9	67.0
10^{-3}	10	446	1.97	268.4	24.0	10^{-4}	10	362	3.83	2232.9	37.3
	20	416	2.53	881.2	29.5		20	338	4.67	3741.6	35.7
	50	386	3.45	829.9	41.8		50	329	5.74	2404.2	47.4
	100	367	5.71	956.9	66.3		100	318	8.88	2053.8	73.1

tolerance ϵ_G for building the adaptive block FSAI factor. The other tolerance ϵ_F is set to 10^{-2} and no low-rank corrections are applied, i.e., $dlr = alr = 0$. As expected, the iteration count decreases progressively as ϵ_G decreases, i.e., G is more accurate, and n_l grows. The preconditioner density also increases, so that the setup time becomes very high and the cost per iteration grows. Although the setup cost could be reduced by introducing thresholds and level-of-fills in the sparse matrix-matrix computations with no substantial loss in the PCG acceleration, it appears that the proposed multilevel approach is of interest whenever the preconditioner can be reused multiple times, e.g., in eigensolvers or in some transient simulations, so that its setup cost can be properly amortized and set apart in the present analysis. Hence, we focus on the solution time T_s only.

Considering the best value for ϵ_G given by the table 4.4, i.e., 10^{-2}, we vary again the number of levels n_l but now also the drop tolerance for computing the F factor, i.e, ϵ_F. The results, provided in Table 4.5, show that the MFLR preconditioner appears to be less sensitive to the quality of F than of G. In fact, the iteration count is more stable than in Table 4.4.

Now, we test the effects of Descending and Ascending Low-Rank corrections by considering $\epsilon_G = 10^{-2}$, $\epsilon_F = 10^{-2}$ and $n_l = 10$. The results for different low-rank corrections are displayed in Table 4.6. Increasing the rank size, the number of iterations decreases for both approaches, but Ascending Low-Rank corrections have a stronger

TABLE 4.5: `cube190k` test case: MFLR performance varying n_l and ϵ_F ($\epsilon_G = 10^{-2}$, $dlr = alr = 0$).

ϵ_G	n_l	n_{it}	ρ	T_p	T_s	ϵ_G	n_l	n_{it}	ρ	T_p	T_s
10^{-1}	10	660	0.65	29.1	22.8	10^{-2}	10	640	0.77	34.5	21
	20	632	0.94	216.8	29.5		20	591	1.17	265.9	29.3
	50	601	1.61	369.8	47.8		50	503	2.04	441.7	41.6
	100	579	2.81	498.0	78.7		100	464	3.67	612.9	67.0
10^{-3}	10	621	1.13	85.7	27.0	10^{-4}	10	585	2.304	1163.3	35.1
	20	544	1.79	639.2	32.9		20	460	3.445	4513.5	39.3
	50	403	3.21	1129.2	43.0		50	309	5.550	4020.9	43.6
	100	348	6.98	2278.7	71.2		100	255	11.192	6029.8	68.6

impact on the solution time T_s. This is somewhat expected because Ascending Low-Rank corrections have a global effect on the overall preconditioner, while Descending Low-Rank corrections improve locally the current level approximation of K^{-1}.

TABLE 4.6: `cube190k` test case: MFLR performance varying either dlr or alr ($\epsilon_G = \epsilon_F = 10^{-2}$, $n_l = 10$).

dlr	n_{it}	ρ	T_p	T_s
5	534	0.90	71.3	29.3
10	485	1.01	66.1	29.4
20	450	1.24	65.9	25.5
50	405	1.92	75.0	30.1

alr	n_{it}	ρ	T_p	T_s
5	289	1.35	95.1	17.0
10	213	1.92	78.0	15.1
20	196	3.06	88.1	14.1
50	186	6.48	214.5	36.7

Finally, the combined effect of both corrections is investigated in table 4.7. This allows for obtaining the best result in terms of both iteration count and solution time T_s decrease. In particular, the latter is more than halved with respect to the case with no corrections with at least $dlr = alr = 10$. The solver acceleration is paid with a bigger setup cost, that makes the MFLR approach interesting when the preconditioner can be recycled.

Although the present analysis cannot be thoroughly exhaustive, it is possible to observe that dlr and alr appear to be the most sensitive user-specified parameters. The number of levels n_l strongly depends on the size of the matrix and should be selected such that each level is not too small. In the `cube190k` test case it can be seen that with more than 20 levels, i.e., less than 10,000 unknowns per level, the preconditioner

TABLE 4.7: `cube190k` test case: MFLR performance varying both dlr and alr ($\epsilon_G = \epsilon_F = 10^{-2}$,$n_l = 10$).

dlr	alr	n_{it}	ρ	T_p	T_s
1	1	525	0.92	76.3	34.4
1	10	204	1.95	87.7	14.8
1	20	188	3.09	112.5	18.5
10	1	381	1.13	71.7	25.0
10	10	148	2.15	83.6	10.9
10	20	134	3.29	100.0	11.3
20	1	341	1.35	86.6	23.9
20	10	132	2.38	100.5	10.4
20	20	123	3.52	100.1	10.8

application cost grows quickly and is no longer compensated by the iteration count reduction. By distinction, the selection of the tolerances ϵ_G and ϵ_F does not appear to be overly difficult. In fact, the MFLR performance does not change much with respect to a variation of these parameters in the interval $[10^{-3}, 10^{-1}]$.

4.4.3 Preconditioner performance

In this section, we evaluate the use of MFLR as a preconditioner to PCG in the solution of real-world problems arising from structural and fluid flow models and described with details in the appendix A. In Table 4.8, we show the best results obtained by the MFLR preconditioner in terms of the solution time T_s as well as the results for the aFSAI, ILUPACK[1] Bollhöfer et al. [2011] and Trilinos_ML Gee et al. [2006] preconditioners. It is worth mentioning that the native adaptive FSAI is used as a benchmark because the BTFSAI and DDFSAI preconditioners generally give rise to indefinite Schur complements in these test cases. Hence, only the robust MF algorithm presented here can be effectively used, with or without low-rank corrections. The latter may improve the convergence rate, but have no effect on the preconditioner robustness.

As already observed, the MFLR preconditioner setup can be quite expensive, especially because of the computation of the eigenpairs needed by the low-rank correction procedures. However, its effectiveness in the iteration count and CPU time can be quite

[1]Since ILUPACK gives a non-symmetric preconditioner, we use GMRes Saad and Schultz [1986] as the iterative solver.

TABLE 4.8: Performance comparison between the aFSAI, ILUPACK, Trilinos_ML and MFLR preconditioners in the solution of real-world engineering problems.

Matrix name	**Method**	n_{it}	ρ	$T_p[s]$	$T_s[s]$	$T_t[s]$
`afshell3`	aFSAI	963	0.89	88.8	63.9	152.7
	ILUPACK	79	4.12	64.0	24.6	88.6
	Trilinos_ML	356	1.01	1.2	80.1	81.3
	MFLR	126	4.50	237.8	31.5	269.3
`afshell8`	aFSAI	1033	0.86	81.2	68.0	149.2
	ILUPACK	78	4.12	62.2	24.4	86.4
	Trilinos_ML	309	1.02	1.2	57.4	58.6
	MFLR	131	3.22	174.8	28.6	203.4
`emilia923k`	aFSAI	1513	0.11	5.0	128.5	133.5
	ILUPACK	—	—	—	—	—
	Trilinos_ML	586	1.15	6.5	367.0	373.5
	MFLR	575	0.25	82.8	80.4	163.2
`geo1438k`	aFSAI	491	0.31	59.4	89.6	149.0
	ILUPACK	878	0.50	57.3	385.8	443.1
	Trilinos_ML	118	1.15	9.6	93.7	103.3
	MFLR	456	0.49	159.8	110.4	270.2
`stocF1465k`	aFSAI	937	0.93	51.3	133.4	184.7
	ILUPACK	108	1.50	49.2	33.3	82.5
	Trilinos_ML	759	1.16	3.8	278.0	281.8
	MFLR	936	1.66	260.4	222.2	482.6

significant. For instance, in the `afshell3` and `afshell8` test cases, n_{it} is approximately reduced by a factor 10 and T_s is more than halved. Hence, the use of the MFLR preconditioner can be of great interest whenever the setup time can be amortized along several linear solves. On the other hand, if the reduction of the number of iterations is marginal, such as in the `geo1438k` and `stocF1465k` test cases, the Adaptive FSAI proves more efficient than the MFLR preconditioner.

The ILUPACK parameters are set in such a way to obtain approximately the same memory footprint as MFLR. By distinction, the default parameters for configuring Trilinos_ML are kept. The results show that the ILUPACK performance in terms of solution time T_s is close to MFLR for `afshell3` and `afshell8`, much better for `stocF1465k` and much lower for `emilia923k` and `geo1438k`. In particular, the ILU density for the `emilia923k` test case must be increased up to 3.8 to attain convergence in just 62 iterations, otherwise the convergence is not reached after 3,000 iterations. Recalling that ILUPACK cannot thoroughly exploit a growing parallelism of the computational architecture, the comparison with MFLR appears to be quite satisfactory. As far as the results with Trilinos_ML are concerned, it can be observed that with the default parameters the MFLR preconditioner is always superior, with the exception of the `geo1438k` test case where the two performances are roughly equivalent.

Finally, to show the potential parallelism of the MFLR preconditioner, a scalability test has been carried out on a discrete Laplacian computed over a regular $300\times300\times300$ grid. The numerical experiment is performed on the A1 partition of the Marconi cluster at CINECA (Center of High Performance Computing), Bologna, Italy. It consists of 1,512 nodes with 128 Gbyte of RAM each. Every node is equipped with 2 Intel Xeon E5-2697v4 Broadwell processors at 2.3 GHz with 36 cores. This preliminary implementation of MFLR makes use of shared-memory parallelism through the OpenMP directives, hence in the strong scalability test provided in Figure 4.4, a single node only is used with up to 32 threads.

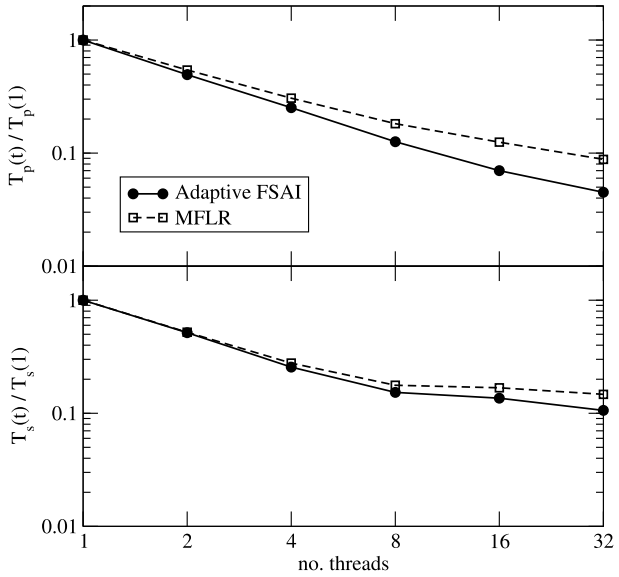

FIGURE 4.4: Strong scalability test for a regular 300^3 Laplacian.

The strong scalability of the MFLR preconditioner is compared to that of the Adaptive FSAI algorithm as far as both the setup and the iteration time, T_p and T_s, is concerned. In particular, Figure 4.4 provides the speedup:

$$S_p = \frac{T_p(t)}{T_p(1)}; \qquad S_s = \frac{T_s(t)}{T_s(1)}, \tag{4.77}$$

with $T_p(t)$ and $T_s(t)$ the setup and iteration wall-clock times measured with t threads. It has been already verified that aFSAI practically has an ideal speedup Janna et al. [2015b], according to the hardware properties of the specific computational architecture used for the numerical experiments. In the case of the multi-core processors of the Marconi cluster, it is well-known that it is virtually impossible to obtain ideal speedups with iterative solvers as these algorithms are bandwidth limited being characterized by a low bit per flop ratio. The MFLR preconditioner is theoretically less parallel than FSAI, because of the intrinsic sequentiality introduced by the multilevel framework. Nevertheless, Figure 4.4 shows that the strong scalability is only marginally affected by such a sequentiality and the MFLR preconditioner still preserves a good degree of parallelism.

4.5 Conclusions

The development of a multilevel framework is often useful in several applications. However, the use of the FSAI preconditioner as basic kernel in a standard multilevel approach may arise some difficulties related with the approximations introduced in the computation of the Schur complement at each level. With SPD problems, such a Schur complement might be indefinite, thus causing a break down of the multilevel algorithm.

We proposed a robust multilevel framework for SPD matrices based on the use of aFSAI as the main kernel. An alternative way of computing the Schur complement is introduced so as to guarantee its positive definiteness independently of the preconditioner sparsity. A theoretical analysis is formulated with the aim of providing appropriate bounds for the eigenspectrum of the preconditioned matrix. The multilevel FSAI preconditioner is further enhanced by introducing low-rank corrections at both a local and a global level, namely Descending and Ascending low-rank corrections, respectively, thus producing the MFLR preconditioning framework.

The MFLR preconditioner has been investigated in a set of test problems to analyze: (i) the relative influence of the user-specified parameters controlling the algorithm setup, and (ii) the computational performance and potential scalability in a parallel environment. The numerical results show that the proposed approach is generally able to significantly accelerate the solver convergence rate still preserving a good degree of parallelism. At the present time, the solver acceleration is paid off by a large setup cost, which is mainly due to the computation of the eigenpairs needed by the low-rank corrections. The increase of the cost for building the preconditioner with respect to aFSAI makes this approach attractive especially for those applications where the preconditioner can be effectively recycled along a number of linear solves.

Chapter 5

Adaptive Smoothing and Prolongation based AMG

This chapter presents an algebraic multigrid preconditioner called adaptive smoothing and prolongation based AMG (aSP-AMG). This presentation extends the one given by the journal article Paludetto Magri et al. [2019] which, by the time of writing this document, was under peer review.

5.1 Introduction

Although the MFLR preconditioner, with a novel way of approximating Schur complements and enhancing the application phase via low-rank corrections, represented an improvement in terms of robustness with respect to the other multilevel FSAI preconditioners such as BTFSAI and DDFSAI, its applicability remained limited by the setup costs. Due to this reason, we decided to explore another algebraic preconditioning field which has a lot in common with multilevel factorization techniques: the algebraic multigrid methods (AMG).

AMG was firstly introduced in the early 1980s in the form of classical AMG (C-AMG), whose main feature is that coarse level nodes are obtained as a subset of the fine level nodes and the interpolation operator is built upon the assumption that the unitary vector is the dominant component in the near-null space of A McCormick [1982]; Stüben [1983]; Brandt et al. [1984]; Brandt [1986]. Since then, researches have been conducted in the direction of improving the components of classical AMG as well as creating new AMG techniques with the final aim of improving AMG efficiency and

expanding its applicability to challenging problems characterized by complex geometries, distorted grids, strong discontinuities in the physical properties and anisotropy.

The most important AMG variants comprise unsmoothed and smoothed aggregation multigrid whose coarsening is based on agglomeration of nodes and the prolongation operator is built in a column-wise fashion in order to interpolate a set of vectors in the near-null space usually provided as an input Vaněk [1992]; Vaněk et al. [1996]; Notay [2012]. The element-based AMG family composed by the energy-minimization AMGe Brezina et al. [2001], element-free AMGe Henson and Vassilevski [2001] and spectral AMGe Chartier et al. [2003], where the coarse spaces are constructed via an energy minimization process, were proposed to improve the robustness of these methods by alleviating the heuristics based on M-matrices properties implemented in classical AMG. More recently the adaptive and Bootstrap AMG (αAMG and BAMG, respectively) were designed for the solution of harder problems where the classical and smoothed AMG may fail or show poor convergence Brezina et al. [2005, 2006]; Brandt and Ron [2003]; Brandt et al. [2011]; Brezina et al. [2012]; Brandt et al. [2015]; D'Ambra et al. [2018]. The most important feature of the above methods is that no preliminary assumption is made about the near-null space of A but it is approximated adaptively during the AMG hierarchy construction by starting from one or multiple candidate vectors. Moreover, the computed multigrid hierarchy can be used itself as a smoother to better expose smooth vectors in a self-improvement fashion. On the one hand, this attribute renders those methods more general and capable of achieving better convergence rates, but on the other, it increases the setup time substantially. A detailed review of the most recent and effective AMG variants developed so far can be found in Xu and Zikatanov [2017].

In this work, we propose a novel AMG package that we name aSP-AMG where the acronym aSP stands for *adaptive Smoothing and Prolongation*. The reason for the choice of the word adaptive is manifold. The first motivation is that we follow the perspective of adaptive and bootstrap AMG that is to assume no information about the near-null space of A, but we construct the space of smooth vectors, denoted as test space in the remainder of the paper, by testing an initial set of candidates, however without making use of the self-improvement concept in the current implementation.

Another novel contribution provided by aSP-AMG is the introduction of the adaptive pattern factorized sparse approximate inverse (aFSAI) as a smoother. This improves the smoothing capabilities of the resulting method as aFSAI is more effective than Jacobi and usually much sparser than Gauss-Seidel for the same accuracy. Moreover, aFSAI has been shown to be, both theoretically and experimentally, strongly scalable Janna et al. [2015b]; Bernaschi et al. [2016], and this fact fosters the implementation of the package in massively parallel computers, even if it is not our main focus at the moment. Coarsening is carried out as in classical AMG by dividing variables into fine and coarse, but the strength of connection is computed by means of the affinity between components of the test space, which we believe is a concept that better adapts to general problems. Finally, the main contribution that we want to stress consists of the presentation of three new techniques for building the prolongation operator which are based on different minimization processes. The rationale of the first two techniques derives from the adaptive block FSAI preconditioning Janna and Ferronato [2011] while the third one, similarly to Brandt et al. [2011], is based on least squares, however with a dynamic pattern selection scheme enhancing its quality especially in real-world problems.

This chapter is organized as follows. In section 5.2, the fundamentals of the classical AMG preconditioner are reviewed and the setup and application algorithms of aSP-AMG are presented. Section 5.3 introduces the aFSAI preconditioner as a smoother, pointing out its advantages over other choices for relaxation and providing details of the setup algorithm. The following section 5.4 exposes how the near-null space of A is approximated via the calculation of a subspace of $\mathbb{R}^{n \times n_t}$ with $n_t << n$. In section 5.6, we introduce the three new aforementioned techniques for building prolongation. In section 5.7, the impact of each configuration parameter on the aSP-AMG performance is analyzed in Poisson and linear elasticity model problems and an extensive set of matrices arising from real-world applications is used to compare aSP-AMG to the well-established and popular BoomerAMG preconditioner from the Hypre library Falgout and Yang [2002] and the one-level aFSAI.

5.2 The Classical Algebraic Multigrid (C-AMG) approach

As mentioned before, there are several families of AMG methods, each of them sharing common components such as a multilevel hierarchy built upon interpolation operators, the use of smoothers and the quest for a suitable interplay between coarse-grid correction and relaxation. In the present work, we adopt the classical AMG approach in which coarse variables are chosen as a subset of the fine level ones and the interpolation operators are usually built in a row-wise fashion.

We start by recalling the basic ideas behind classical AMG methods. For a more detailed and rigorous description, we refer the reader to the works by works Stüben [2001]; Trottenberg et al. [2001]; Xu and Zikatanov [2017]. For the sake of clearness, we restrict our explanation here and in the remainder of the paper to a two-level method, and note that the multilevel extension can be easily obtained by applying the two-level scheme recursively for the coarse levels solution. Recalling the linear system given by (1.1), to simplify notation, we reorder the system matrix A according to the Fine/Coarse (F/C) partitioning of the unknowns:

$$A = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{fc}^T & A_{cc} \end{bmatrix} \tag{5.1}$$

even though this ordering is never really needed in the solver implementation.

The first component that needs to be defined in C-AMG is the smoother, which is a stationary iterative method responsible for eliminating the error components associated with large eigenvalues of A, usually referring to the high frequency error components. Generally, it is given by a simple pointwise relaxation method such as (block) Jacobi or Gauss-Seidel, with the second one often preferred even though its efficient parallel implementation is not straightforward. In aSP-AMG, we introduce the usage of aFSAI as a smoother. Regardless of the choice, the smoother operator S can be represented by

$$S = I - \omega M^{-1} A, \tag{5.2}$$

where I is the identity matrix, ω is a relaxation factor and M, the preconditioning operator.

The following step for building a multigrid hierarchy is called coarsening. Defining Ω as the index set $1, 2, ..., n$, with n the size of A, we want to find two disjoint sets $\mathcal{F}$ and $\mathcal{C}$ representing the fine and coarse nodes, respectively, such that $\Omega = \mathcal{C} \cup \mathcal{F}$ with $n_f = |\mathcal{F}|$ and $n_c = |\mathcal{C}|$. Strategies for computing the F/C splitting normally rely on a classical strength of connection concept which is taylored for M-matrices and shows practical limitations when solving more general matrices. Aiming a more robust approach, in aSP-AMG we employ an affinity-based strength of connection introduced by Livne and Brandt [2012] and explained detailed in section 5.5.

Next, it is possible to build the prolongation operator, which is responsible for transferring information from the coarse to the fine space. Using the conventional F/C ordering defined above (5.1), the prolongation operator P will be written as:

$$P = \begin{bmatrix} W \\ I \end{bmatrix}, \tag{5.3}$$

where W is a $n_f \times n_c$ matrix containing the weights for coarse-to-fine variable interpolation. Finally, as the system matrix is SPD, we assume a Galerkin approach in defining the restriction operator R as P^T, with the coarse level matrix A_c simply given by the triple matrix product:

$$A_c = P^T A P. \tag{5.4}$$

In practice, we want fast convergence with a rapid coarsening, i.e. high F/C ratios, and possibly small sets of interpolatory variables, also referred as the prolongation *c*aliber, to allow for small and relatively sparse coarse grid operators. With aSP-AMG, we propose three algorithms for constructing effective prolongation operators that conciliate these conflicting requirements.

Once all the aforementioned components are well defined, the setup phase of the two-level multigrid method can be completed and its iteration matrix is given by:

$$(S)^{\nu_2} \left(I - P {A_c}^{-1} P^T A\right) (S)^{\nu_1} \tag{5.5}$$

with ν_1 and ν_2 representing the number of pre and post smoothing steps, respectively.

Algorithm 7 AMG setup.

procedure AMG_SETUP(A_k)
  Define Ω_k as the set of the n_k vertices of the adjacency graph of A_k
  if n_k is small enough to allow for a direct factorization **then**
    Compute $A_k = L_k L_k^T$
  else
    Compute M_k such that $M_k^{-1} \simeq A_k^{-1}$
    Define the smoother as $S_k = \left(I_k - \omega_k M_k^{-1} A_k\right)$
    Compute the $n_k \times n_t$ test space matrix X_k
    Partition Ω_k into the disjoint sets $\mathcal{C}_k$ and $\mathcal{F}_k$ via coarsening
    Compute the prolongation matrix P_k from $\mathcal{C}_k$ to Ω_k
    Compute the new coarse level matrix $A_{k+1} = P_k^T A_k P_k$
    Call AMG_SetUp(A_{k+1})
  end if
end procedure

As anticipated, extending this two-level approach to a more efficient multilevel version is straigthforward by using recursivity. Algorithms 7 and 8 briefly report the general AMG setup phase and application in a V-cycle, respectively, where it is conventionally assumed that $A_0 = A$, $\mathbf{y}_0 = \mathbf{y}$ and $\mathbf{z}_0 = \mathbf{z}$. The details regarding all the computational building blocks used in our implementation of Algorithm 7 will be given in the following sections.

Algorithm 8 Application of the AMG V-cycle preconditioner.

procedure AMGV_APPLY(A_k, $\mathbf{y}_k$, $\mathbf{z}_k$)
  if k is the last level **then**
    Solve $L_k L_k^T \mathbf{z}_k = \mathbf{y}_k$
  else
    $\mathbf{s}_k \leftarrow$ apply ν_1 smoothing steps to $A_k \mathbf{s}_k = \mathbf{y}_k$ with $\mathbf{s}_0 = \mathbf{0}$
    $\mathbf{r}_k \leftarrow \mathbf{y}_k - A_k \mathbf{s}_k$
    $\mathbf{r}_{k+1} \leftarrow P_k^T \mathbf{r}_k$
    Call AMGv_Apply($A_{k+1}, \mathbf{r}_{k+1}, \mathbf{d}_{k+1}$)
    $\mathbf{d}_k \leftarrow P_k \mathbf{d}_{k+1}$
    $\mathbf{s}_k \leftarrow \mathbf{s}_k + \mathbf{d}_k$
    $\mathbf{z}_k \leftarrow$ apply ν_2 smoothing steps to $A_k \mathbf{z}_k = \mathbf{y}_k$ with $\mathbf{z}_0 = \mathbf{s}_k$
  end if
end procedure

5.3 aFSAI smoothing

The development of robust and efficient multigrid smoothers for parallel architectures is an active field of research. On the one hand, some of the best methods used in

sequential multigrid implementations such as Forward/Backward Gauss-Seidel, Successive Over Relaxation and ILU are hard to parallelize. On the other hand, highly parallel approaches such as hybrid Gauss-Seidel, (damped) l1-Jacobi/Gauss-Seidel and Chebyshev polynomials Adams et al. [2003]; Baker et al. [2011]; Ghysels et al. [2012] might lead to poor convergence especially when dealing with anisotropic and discontinuous coefficients problems as well as system of PDEs. Thus, the problem of finding strategies showing a good compromise between parallelism and effectiveness is yet to be solved.

Given the good qualities of sparse approximate inverses mentioned in section 2.3.3, we exploit the idea of using aFSAI as the smoother method for aSP-AMG. In this case, the operator (5.2) is written as

$$S = I - \omega G^T G A \tag{5.6}$$

with the highest frequencies of $G^T GA$ compromising the smoothing property of aFSAI while the lowest counterpart being responsible to the slow-to-converge modes. The first issue can be handled by setting the relaxation factor as

$$\omega = \frac{2}{\lambda_{max}\left(G^T GA\right)}, \tag{5.7}$$

thus ensuring the overall smoother convergence Briggs et al. [2000], i.e., $||S|| \leq 1$.

Despite the high number of publications dealing with sparse approximate inverses, very few analyzed their use as smoothing techniques, and none of them have studied aFSAI specifically. The very first research was introduced by Tang and Wan [2000], who analyzed the smoothing property of a modified least squares SAI method and compared it to other techniques such as Gauss-Seidel for the solution of anisotropic problems showing that they are very effective techniques. Later, Bröker et al. [2001] and Bröker and Grote [2002] analyzed more robust SAI smoothers based on SPAI variants that are built with the same sparsity pattern of A as well as with more general patterns that are obtained dynamically during the smoother setup. Moreover, they introduce theoretical results proving the smoothing property of SPAI. However, in general, the

SPAI smoother is not symmetric which thus limits the applicability of AMG as a preconditioner to PCG. Also, for a given factor density, the convergence given by SPAI is inferior to aFSAI when used as single-level preconditioners.

The accuracy and effectiveness of the aFSAI smoother can be easily controlled by allowing for larger densities of G, which is not possible with Jacobi or Gauss-Seidel relaxations. Depending on the type of matrix being solved, aFSAI can even yield sparser smoothers ,when looking at the the application phase, than Gauss-Seidel while maintaining its smoothing property, i.e., $||S|| \leq 1$. An example proving this phenomenon is showed in Figure 5.1 which presents the sparsity pattern and the number of nonzeros coefficients of the linear system operators A_k and aFSAI factors G_k formed by aSP-AMG with $k = \{1, 2, 3\}$ when considering a 3D linear elasticity problem discretized with P1 FEM. By distinction to ILU preconditioners, the setup of aFSAI is not affected by reordering of unknowns either by the occurrence of zero or negative pivots. Lastly, they form SPD smoothers naturally, and their setup and application phases follow highly parallel operations, thus bringing together the two ingredients necessary for achieving smoothers well-suited for parallel architectures.

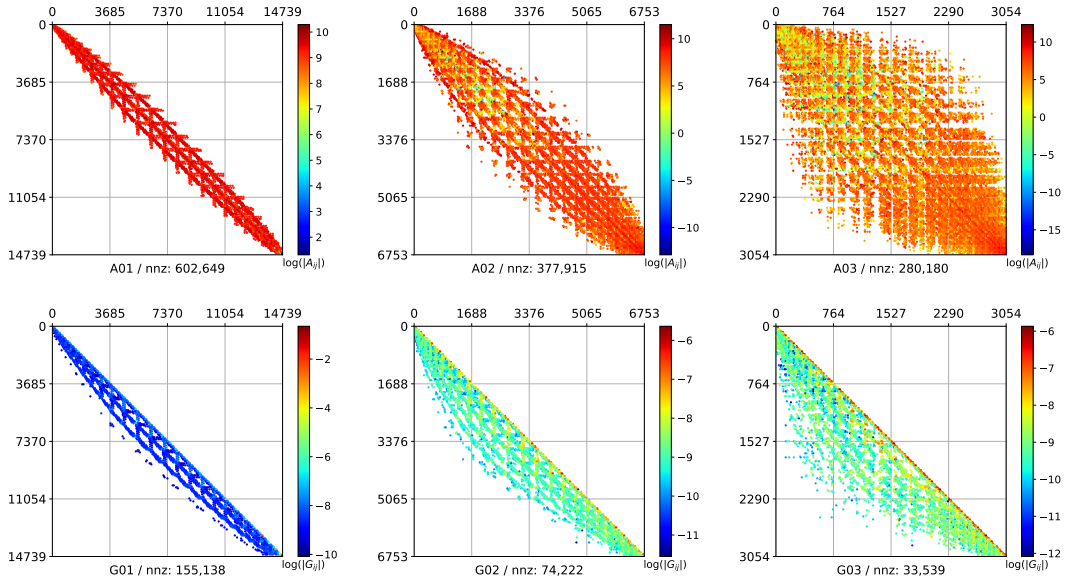

FIGURE 5.1: Comparison between the sparsity patterns of the system matrices A_k (upper frame) and aFSAI factors G_k (lower frame) belonging to the finest (left), second (center) and coarsest (right) levels of aSP-AMG.

5.4 Test space generation

A crucial step in the setup of aSP-AMG or any other adaptive AMG method is the identification of smooth modes, i.e., components of the error vector that are not eliminated sufficiently during smoothing. The reason for this importance is that the smooth modes need to be well represented by the range of the prolongation operator in order to be eliminated by coarse grid correction and consequently result in an efficient solver. Like most of the other single level preconditioners, aFSAI is able to represent accurately only the upper part of the spectrum of an SPD matrix while it fails in approximating lower counterpart. In Figure 5.2 (left frame), we show a typical example of such occurrence by comparing the spectra of A and different configurations of $(G^TG)^{-1}$ for the `bcsstk38` sparse matrix from the SuiteSparse collection. No matter the accuracy used for computing the aFSAI factor G, the lowest end of the eigenspectrum of $(G^TG)^{-1}$ (right frame) does not appropriately represent the one of A.

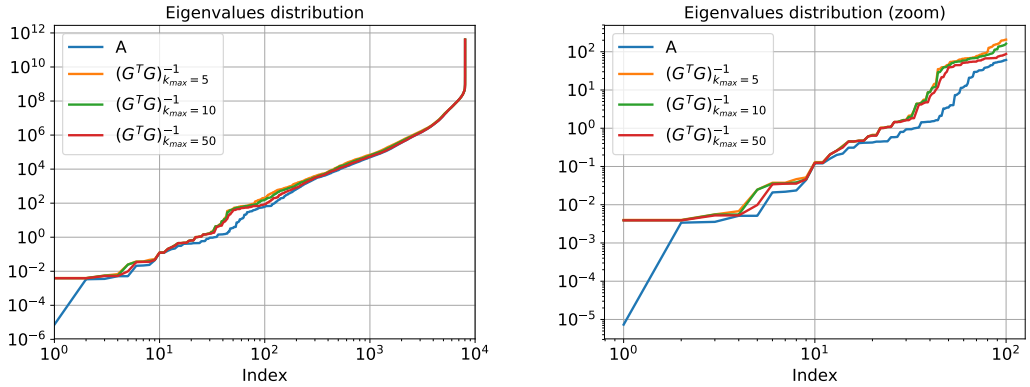

FIGURE 5.2: In the left, LogLog plot of the eigenspectrum of A and different configurations of $(G^TG)^{-1}$ for the `bcsstk38` test case. In the right, a zoom into the lowest end of the eigenspectra.

In Bootstrap AMG, Brandt et al. [2011], smooth modes are interpreted as the near-null space of A which is approximated by a small subspace of test vectors, called test space, after the application of a multilevel eigensolver. This idea was proven to be robust and to originate an AMG method with fast convergence for anisotropic Poisson problems. However, it involves reconstruction of the multigrid hierarchy and would be inefficient if used within the aSP-AMG framework given the aFSAI setup cost. Moreover, as shown recently by Brannick et al. [2018], the near-null space of A is not the

optimal representation of smooth modes, instead they are given by the eigenmodes corresponding to the smallest eigenvalues of

$$A\mathbf{x} = \lambda \tilde{M}\mathbf{x}, \tag{5.8}$$

with $\tilde{M}$ the symmetrized preconditioner matrix. In aSP-AMG, we follow the last approach and search for an approximation of the eigenvectors related to the n_t largest eigenvalues of

$$\left(I - \omega G A G^T\right)\mathbf{y} = \bar{\lambda}\mathbf{y} \tag{5.9}$$

$$\rightarrow \bar{S}\mathbf{y} = \bar{\lambda}\mathbf{y}, \tag{5.10}$$

with $\bar{S}$ the symmetrized smoother operator. Note that the Equations (5.8) and (5.10) are related via $\tilde{M} = G$, $\mathbf{x} = G^T\mathbf{y}$ and $\bar{\lambda} = (1 - \lambda)$. The n_t solutions $\mathbf{x}$ form the column vectors of a test space matrix X. In aSP-AMG, we have three strategies for computing the test space matrix as detailed below.

5.4.1 Power method

The simplest approach for solving Equation (5.10) for its largest eigenvalues in magnitude is a variant of the power method, presented by the Algorithm 9. This technique, which belongs to the class of single-vector iteration methods, is based on the fact that a vector which is multiplied repeatedly by a matrix tends to align in the direction of the eigenvector associated to its dominant eigenvalue. In aSP-AMG, we assume that this component gives the principal direction on what the smoother reduces the error least efficiently and, thus, which should be well characterized by the formation of robust prolongation operators. Lastly, we note that the only input parameter needed for configuring the power method is the number of iterations n_{it}.

5.4.2 Plain Lanczos

The Lanczos method was originally proposed by Lanczos [1952] and was later enhanced with an explicit orthogonalization step for a better accuracy in finite precision

Algorithm 9 Test Space generation by Power method.

procedure CPT_POWER(n_t, n_{it}, A, ω, G, G^T)
  for $k = 1, n_t$ **do**
    Initialize $\mathbf{x}_k$ to a random vector
    for $i = 1, n_{it}$ **do**
      $\mathbf{x}_k \leftarrow \left(I - \omega G A G^T\right) \mathbf{x}_k$
      $\mathbf{x}_k \leftarrow \mathbf{x}_k / \left|\left|\mathbf{x}_k\right|\right|_2$
    end for
    $\mathbf{x}_k \leftarrow G^T \mathbf{x}_k$
  end for
  return X
end procedure

computations by Paige [1972] and Paige [1980]. In the context of aSP-AMG, it is used to compute the upper part the eigenspectrum of $\tilde{S}$. The method starts with the observation that $\tilde{S}$ can be reduced it to a tridiagonal form T with size $n_s \times n_s, (n_s << n)$ by means of the three-term recurrence formula:

$$\beta_k \mathbf{v}_k = \tilde{S}\mathbf{v}_{k-1} - \alpha_{k-1}\mathbf{v}_{k-1} - \beta_k \mathbf{v}_{k-2}, \tag{5.11}$$

with $\mathbf{v}_k$ an arbitrary vector of unitary norm and the coefficients

$$\begin{aligned} \alpha_k &= \mathbf{v}_k^T \left(\tilde{S}\mathbf{v}_k - \beta_k \mathbf{v}_{k-1}\right), \text{ and,} \\ \beta_k &= \left|\left|\tilde{S}\mathbf{v}_{k-1} - \alpha_{k-1}\mathbf{v}_{k-1} - \beta_{k-1}\mathbf{v}_{k-2}\right|\right|_2, \end{aligned} \tag{5.12}$$

such that the tridiagonal matrix T is written as

$$T = \begin{bmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \beta_3 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{n_s} \\ & & & \beta_{n_s} & \alpha_{n_s} \end{bmatrix}. \tag{5.13}$$

Defining $V = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_{n_s}]$, the following relation holds

$$\tilde{S}V - VT = \beta_{n_s+1}\mathbf{v}_{n_s+1}\mathbf{e}_m^T. \tag{5.14}$$

Multiplying the last equation by V^T, we obtain:

$$V^T A V = T, \tag{5.15}$$

since the property $V^T \mathbf{v}_{n_s+1} = \mathbf{0}$ holds by construction of the method. From Equation (5.15), we can conclude that T represents the orthogonal projection of $\tilde{S}$ onto the Krylov subspace which allows for the computation of the Rayleigh-Ritz approximations of its eigenpairs. Let $(\lambda_k, \mathbf{e}_k)$ be an eigenpair of T belonging to the upper part of its eigenspectrum, then the Ritz value λ_k and the Ritz vector $\mathbf{V}\mathbf{e}_k$ can be taken as approximations to the eigenpair of $\tilde{S}$.

The process described above is given by the Algorithm 10, which depends on the following user-specified parameters:

1. n_t, the dimension of the test space;
2. n_s, the dimension of the trial space;
3. ϵ_t, the relative accuracy in estimating eigenpairs, i.e., $(\lambda, \mathbf{v})$ is added to the test space if it satisfies $\left|\left|\tilde{S}\mathbf{v} - \lambda\mathbf{v}\right|\right|_2 \leq \epsilon_t \left|\left|\mathbf{v}\right|\right|_2$.

5.4.3 Simultaneous Rayleigh Quotient Minimization (SRQCG)

If an approximation X_0 for the (partial) near-null space of A is available beforehand, it is natural to use this information as an initial guess for building a possibly larger test space X. Knowing the physical problem that originates A usually can give a clue on the matrix X_0. For instance, when solving linear elasticity problems via a finite element discretization, it is known that the rigid body modes constitute the kernel of the differential operator prior to application of the boundary conditions, thus, they can be used to define X_0. As a matter of fact, such information is essential to the effectiveness of other methods like the aggregation-based AMG Adams [2002] and the deflation-based preconditioning Frank and Vuik [2001]; Jonsthovel et al. [2013]; Baggio et al. [2017].

Given the limitation of the plain Lanczos and power methods on employing more than a single initial vector for computing X, we propose the use of a Rayleigh quotient minimization algorithm for this specific situation. The method is called Simultaneous

Algorithm 10 Test Space generation by Lanczos.

procedure CPT_LANCZOS(n_t, n_s, ϵ_t, A, ω, G, G^T)
  Initialize $\mathbf{x}_1$ to a random vector
  $\mathbf{v}_1 \leftarrow \left(I - \omega GAG^T\right)\mathbf{x}_1$
  $\mathbf{v}_1 \leftarrow \mathbf{v}_1 / ||\mathbf{v}_1||_2$
  for $i = 1, n_s$ **do**
    $\mathbf{v}_{i+1} \leftarrow \left(I - \omega GAG^T\right)\mathbf{v}_i$
    if $(i > 1)$ **then** $\mathbf{v}_{i+1} \leftarrow \mathbf{v}_{i+1} - \beta_{i-1}\mathbf{v}_{i-1}$ **end if**
    $\alpha_i \leftarrow \mathbf{v}_{i+1}^T \mathbf{v}_i$
    $\mathbf{v}_{i+1} \leftarrow \mathbf{v}_{i+1} - \alpha_i \mathbf{v}_i$
    $\beta_i \leftarrow ||\mathbf{v}_{i+1}||_2$
  end for
  Define T via Equation (5.13)
  Solve the eigenvalue problem $TE = E\Lambda$
  Set Ω as the indices associated with the n_t largest eigenvalues from Λ
  Set $\tilde{V}$ as $[\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_{n_s}]$ restricted to Ω
  Set $\tilde{E}$ as E restricted to Ω
  for $i = 1, n_t$ **do**
    if $\left|\left|\tilde{S}\tilde{E}_i - \lambda\tilde{E}_i\right|\right|_2 \leq \epsilon_t \left|\left|\tilde{E}_i\right|\right|_2$ **then**
      $X_i \leftarrow G^T\tilde{V}\tilde{E}_i$
    end if
  end for
  return X
end procedure

Rayleigh Quotient Minimization by Conjugate Gradients (SRQCG) and was introduced by Longsine and McCormick [1980]. As the name clearly suggests, it is based in the minimization of Rayleigh quotients over several independent vectors simultaneously. A conjugate gradient technique is then used to form each new vector through the linear combination of current iterates and correction vectors. Moreover, as in other subspace-based methods, SRQCG uses Ritz projections to accelerate convergence.

The SRQCG method is described by the Algorithm (11). For a practical investigation of SQRCG through numerical experiments based on real-world applications, we refer the reader to the works by Bergamaschi et al. [2006, 2012] and Ferronato et al. [2012b]

5.5 Affinity-based coarsening

Numerous coarsening algorithms have been developed over the years such as classical Ruge-Stüeben (RS), Cleary-Luby-Jones-Plasman (CLJP), Parallel Maximal Independent Set (PMIS) and Hybrid MIS (HMIS) Yang [2006]. All of them rely on the concept of strength of connection (SoC), which measures the influences exerted between two neighboring nodes. The commonly used definition of strength of connection, however, is based on the assumption that A is an M-matrix or it is applied to the M-matrix relative of A which jeopardizes its applicability to more general discretizations. In aSP-AMG, we employ another definition of SoC based on the concept of *affinity* recently introduced by Livne and Brandt [2012], that we believe more flexible and with a wider range of applicability.

Affinity-based SoC requires the availability of a suitable test space X, i.e, with its column vectors representing the smooth modes. Our current implementation is able to both take advantage of an already existing test space, if available, or estimating it through the algorithms discussed in section 5.4. In any case, let us denote as $\mathbf{x}_i^T$ the i-th row vector of X. The connection strength between two adjacent degrees of freedom i and j is given by

$$SoC[i,j] = \frac{\left(\mathbf{x}_i^T \mathbf{x}_j\right)^2}{\left(\mathbf{x}_i^T \mathbf{x}_i\right)\left(\mathbf{x}_j^T \mathbf{x}_j\right)}. \tag{5.16}$$

Algorithm 11 Test Space generation by SRQCG.

procedure CPT_SRQCG(n_t, n_{it}, A, G, G^T, X_0)
  $Z_0 \leftarrow \left(I - GAG^T\right) X_0$
  Orthogonalize the column vectors of Z_0
  $SZ_0 \leftarrow GAG^T Z_0$
  $\beta_0 = 0$; $P_0 = 0$
  for $k = 1, n_{it}$ **do**
    for $i = 1, n_t$ **do**
      $\mathbf{q}_{k-1,i} \leftarrow \left(SZ_{k-1,i}\right)^T Z_{k-1,i}$
      $R_{k-1,i} \leftarrow SZ_{k-1,i} - \mathbf{q}_{k-1,i} Z_{k-1,i}$
      if $k > 1$ **then**
        $\beta_{k-1} \leftarrow 2\left(SP_{k-1,i}\right)^T \left(R_{k-1,i}\right) \Big/ \mathbf{n}_{k-1,i}$
      end if
      $P_{k,i} \leftarrow 2R_{k-1,i} - \beta_{k-1} P_{k-1,i}$
      $SP_{k,i} \leftarrow GAG^T P_{k-1,i}$
      $\mathbf{m}_{k,i} \leftarrow \left(SZ_{k,i}\right)^T \left(P_{k,i}\right)$
      $\mathbf{n}_{k,i} \leftarrow \left(SP_{k,i}\right)^T \left(P_{k,i}\right)$
      $\mathbf{p}_{k,i} \leftarrow \left(Z_{k,i}\right)^T \left(P_{k,i}\right)$
      $\mathbf{q}_{k,i} \leftarrow \left(P_{k,i}\right)^T \left(P_{k,i}\right)$
      $\mathbf{r}_{k,i} \leftarrow \left(SZ_{k,i}\right)^T \left(Z_{k,i}\right)$
      $a_0 \leftarrow \mathbf{m}_{k,i} - \mathbf{p}_{k,i}\mathbf{r}_{k,i}$
      $a_1 \leftarrow \mathbf{q}_{k,i}\mathbf{r}_{k,i} - \mathbf{n}_{k,i}$
      $a_2 \leftarrow \mathbf{n}_{k,i}\mathbf{p}_{k,i} - \mathbf{m}_{k,i}\mathbf{q}_{k,i}$
      $\alpha_{k,i} \leftarrow \left(a_1 + \sqrt{a_1^2 - 4a_2a_0}\right) \Big/ 2a_2$
      $Z_{k,i} \leftarrow Z_{k,i} + \alpha_{k,i}\left(P_{k,i}\right)$
      $SZ_{k,i} \leftarrow SZ_{k,i} + \alpha_{k,i}\left(SP_{k,i}\right)$
    end for
    $D_k \leftarrow Z_k^T \left(SZ_k\right)$
    $E_k \leftarrow Z_k^T Z_k$
    Solve $D_k U_k = E_k U_k \Lambda_k$
    $Z_k \leftarrow Z_k U_k$
    $SZ_k \leftarrow \left(SZ_k\right) U_k$
  end for
  $X \leftarrow G^T Z_{nit}$
  Orthogonalize the column vectors of X
  return X
end procedure

Algorithm 12 F/C splitting based on MIS

```
1: procedure CPT_FCSPLITTING(n, X, SoC, C, F)
2:     for i = 1, n do
3:         nrmx_i ← ||X_{i,:}||_2
4:     end for
5:     Put nrmx in descending order stored as idx;
6:     Set C = ∅ and F = ∅;
7:     for k = 1, n do
8:         i = idx_k
9:         if i is a Free node then
10:            C ← C ∪ i
11:            for all strong neighbors j of i do
12:                if i is a Free node then F ← F ∪ j; end if
13:            end for
14:        end if
15:    end for
16: end procedure
```

With this definition, the strength of connection matrix SoC is formed initially on the same pattern of A and then filtered by dropping weak connections. Coarse nodes are finally chosen by finding a maximum independent set (MIS) of nodes on the filtered adjacency graph. Unfortunately, affinity-based SoC gives usually rise to connections whose numerical values are distributed in a narrow interval, so it is tricky to define an absolute threshold for dropping. For this reason, we prefer to control the sparsity of the SoC matrix, and thus the rapidity of coarsening, by specifying the integer parameter:

- θ, the average number of connections per node,

that is we prescribe the maximum number of entries in SoC.

5.6 Prolongation operators

Once the coarsening stage is concluded and the coarse unknowns are selected, we can define the prolongation operator P. To achieve an effective two-grid method, it is crucial to compute a prolongation whose action is complementary to the fine level smoother, that is, P should be able to accurately represent slow converging modes of the error. In this section, we present three new techniques for building P where the first two rely on approximating the ideal prolongation operator Falgout and Vassilevski [2004], while the last one approximates the optimal prolongation Brannick et al. [2018].

5.6.1 Adaptive Block FSAI (ABF)

For a given F/C splitting of variables, the ideal prolongation is the absolute minimizer of a weak approximation property Falgout and Vassilevski [2004]; Vassilevski [2008]. Recalling the form (5.3) of the prolongation, it can be completely defined by

$$W_{\text{ideal}} = -A_{ff}^{-1} A_{fc}. \tag{5.17}$$

Although being attractive from a convergence viewpoint, the direct use of such operator is impractical. The simple reason is that computing A_{ff}^{-1} explicitly or its action to an arbitrary vector is expensive and, in most practical cases, leads to dense W_{ideal} operators. In many AMG methods, prolongation operators are designed to target the ideal form W_{ideal} while maintaining sparsity and cheap setup. For instance, the standard interpolation employs a prescribed pattern to W and a few Jacobi or Gauss-Seidel to improve its quality Trottenberg et al. [2001]; also, in Olson et al. [2011], diverse energy-minimization techniques are proposed to approximate W_{ideal} through a root-node multigrid type Manteuffel et al. [2017]. It can be shown, see e.g., Theorem 12.2 Xu and Zikatanov [2017], that computing the ideal interpolation (5.17) is equivalent to solving the trace minimization problem:

$$P^* = \underset{P \in \mathcal{P}}{\operatorname{argmin}} \operatorname{tr}(P^T A P) \tag{5.18}$$

where $\mathcal{P}$ represent the set of $n \times n_c$ matrices of the form specified in (5.3). In the Adaptive Block Factorization (ABF) prolongation, we propose an approximation of W_{ideal} based on the work by in Janna and Ferronato [2011] where an adaptive procedure is developed for minimizing each entry $[FAF^T]_{ii}$ for any $i \in \{1, \ldots, n\}$ with F a block lower triangular matrix (see Theorem 2.4, Janna and Ferronato [2011]). The analogy with ideal prolongation is readily found by considering a matrix F partioned in two blocks with dimensions n_f and n_c, respectively, such that:

$$F = \begin{bmatrix} I_{n_f} & 0 \\ \widetilde{F} & I_{n_c} \end{bmatrix} \tag{5.19}$$

Due to the format of F, the following identity holds:

$$\mathrm{tr}(FAF^T) = \mathrm{tr}(A_{ff}) + \sum_{i \in \mathcal{C}} [FAF^T]_{ii}, \tag{5.20}$$

which shows that reaching the minimum of each $[FAF^T]_{ii}$ is equivalent to choosing $\widetilde{F} = W_{\text{ideal}}$. Using an approach similar to the one presented in Franceschini et al. [2018], we approximate the ideal prolongator by running a few iterations of the Algorithm 1 with configuration parameters k_p, ρ_p and ϵ_p, where in place of $\mathbf{g}_i$ we compute $\mathbf{w}_i^T$, the i-th row of W.

5.6.2 Dynamic Pattern Least Squares (DPLS)

The latter method usually provides a fast AMG cycle in terms of convergence; however, at the cost of a dense sparsity pattern for W. For this reason, we developed another strategy for computing the prolongation operator where the nonzero pattern of W is selected through a least squares minimization procedure inspired by the work Brandt et al. [2011].

A common way to select the nonzero pattern of W is to choose for each row vector $\mathbf{w}_i^T$ of W nonzero coefficients in correspondence to coarse nodes with a sufficiently small connecting path to i. Usually, a length equal to two is already enough for building good operators when using moderate coarsening ratios, while distance three might be used in case of aggressive coarsening. Normally, to limit the number of nonzeroes in W, only strong connections are considered. However, the connectivity of the strength of connection matrix may vary significantly, especially in difficult problems, with the consequence that an *a priori* selected nonzero patterns may give rise to both overdetermined and underdetermined row systems. The first occurrence causes unnecessary large operator complexity, while the latter prevents the construction of an effective prolongation as the target range cannot be represented locally. Very often, to avoid the second occurrence, it is necessary to include long-distance neighbors, thus producing dense prolongation that needs to be post-filtered in order to limit setup and application costs.

With the Dynamic Pattern Least Squares (DPLS), we propose an iterative procedure, showing some analogies with approximate inverses Grote and Huckle [1997]; Janna and Ferronato [2011], that constructs the prolongation pattern dynamically during setup. Its first stage is very similar to the static pattern selection of FSAI. For any fine node i, we choose a set of coarse nodes that can be reached from i with a path of strong connections shorter than a given distance d_p, and form the set $\mathcal{J}_i$ of potential column indices to be considered in row $\mathbf{w}_i^T$. The set $\mathcal{J}_i$ is intentionally larger than what is really needed and the selection procedure can be efficiently implemented through a level set traversal starting from i. Once $\mathcal{J}_i$ is formed, the problem is to choose a fixed number k of entries $j \in \mathcal{J}_i$ such that there exists a linear combination of $\overline{\mathbf{v}}_j$ giving the best possible approximation of $\overline{\mathbf{v}}_i$ in the Euclidean norm sense. When $k = 1$, the optimal solution is easily found by selecting the index $\bar{j}$ for which the angle comprised between $\overline{\mathbf{v}}_i$ and $\overline{\mathbf{v}}_{\bar{j}}$ is minimal or, in other words, the affinity

$$\alpha_{i\bar{j}} = \frac{|\overline{\mathbf{v}}_i^T \overline{\mathbf{v}}_{\bar{j}}|}{||\overline{\mathbf{v}}_i||_2 \left|\left|\overline{\mathbf{v}}_{\bar{j}}\right|\right|_2} \tag{5.21}$$

is maximal. To choose the second most promising entry, it is necessary to update the affinity estimate by removing the components of both $\overline{\mathbf{v}}_i$ and all the $\overline{\mathbf{v}}_j$ along $\overline{\mathbf{v}}_{\bar{j}}$. This operation is crucial, since, if not present, another connection carrying similar information of node $\bar{j}$ could be added to the pattern of P. Orthogonalization can be conveniently carried out by applying Householder reflections to $\overline{\mathbf{v}}_i$ and all the remaining $\overline{\mathbf{v}}_j$. Given a vector $\mathbf{u} \in \mathbb{R}^n$, the rotation matrix Q nullifying its components from $k+1$ to n is given by

$$Q = I - 2\mathbf{h}\mathbf{h}^T, \tag{5.22}$$

where $\mathbf{h} = \mathbf{z} \,/\, ||\mathbf{z}||_2$ is a unit vector obtained from $\mathbf{z}$ whose components statisfy:

$$z_i = \begin{cases} 0, & \text{if} \quad i < k; \\ u_i + \text{sign}(u_i) \times \sqrt{\sum_{j=k}^{n} u_j^2}, & \text{if} \quad i = k; \\ u_i, & \text{if} \quad i > k. \end{cases} \tag{5.23}$$

The selection of interpolating coarse nodes from $\mathcal{J}_i$ continues until the desired accuracy

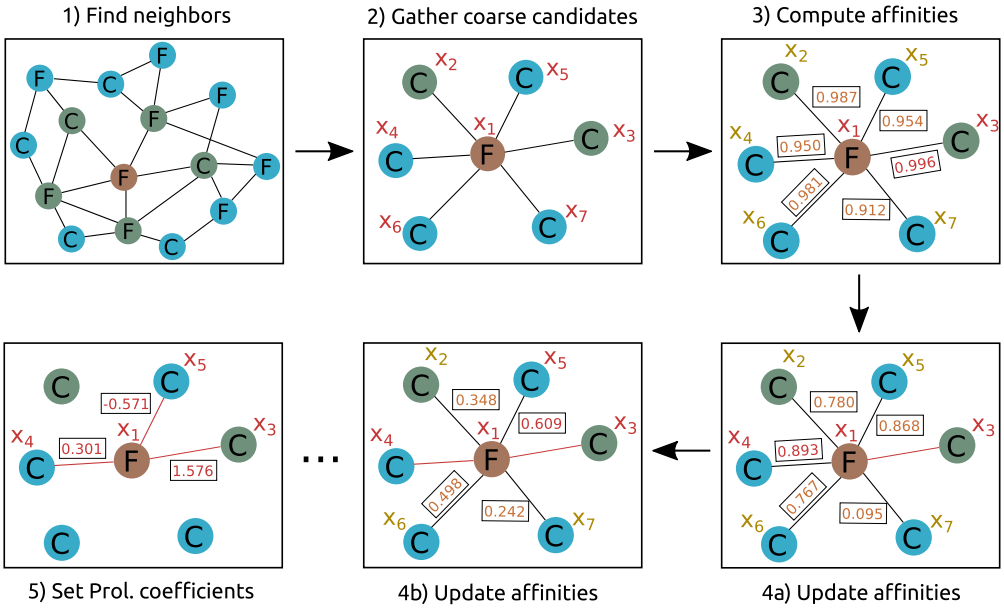

FIGURE 5.3: Practical demonstration of Algorithm 13 with numerical values taken from a linear elasticity test case. We want to find the interpolation weights for the central fine node F, depicted in brown. In this example, we set $d_p = 2$ and $\epsilon_p = 10^{-2}$. Initially, i.e., upper right frame, we find the set of nodes that are strongly connected to F with algebraic length up to d_p. Note that the first level neighbors are depicted in green while the second level neighbors, blue. Then, in frame (2), the set of interpolation candidates $\mathcal{J}_i$ and the row vectors $\mathbf{x}_i$ are gathered from the test space matrix. In (3), we compute the affinities of $\mathbf{x}_i$ for $i = \{2, 3, \cdots, 7\}$ with respect to $\mathbf{x}_1$ and select the coarse node with the biggest one as the first interpolation node. In frame (4a), the affinities are updated according to the Householder reflection defined by Equation (5.22) and, again, the coarse candidate with the biggest value is included in the set $\overline{\mathcal{J}}_i$. This procedure is repeated in frame (4b), where convergence according to Equation (5.24) is finally met. Lastly, in frame (5), the interpolation weights for the selected coarse nodes are computed.

is reached, that is the relative difference between $\overline{\mathbf{v}}_i$ and the optimal linear combination of selected $\overline{\mathbf{v}}_j$ falls below a prescribed tolerance:

$$\left\| \overline{\mathbf{v}}_i - \sum_{j \in \overline{\mathcal{J}}_i} \beta_j \overline{\mathbf{v}}_j \right\|_2 \leq \epsilon_p \, ||\mathbf{v}_i||_2 \tag{5.24}$$

where $\overline{\mathcal{J}}_i \subseteq \mathcal{J}_i$ contains only the selected entries. Obviously, if k reaches n_t, the vector $\overline{\mathbf{v}}_i$ is perfectly matched and the procedure stops anyway. For a better understanding, we provide in Figure 5.3 a practical example of the procedure just described.

In summary, the DPLS prolongation is controlled by the user-defined parameters:

1. d_p, the maximum path length for interpolation among fine and coarse nodes;

Algorithm 13 Dynamic Pattern Least Squares Prolongation

procedure DPLS_SETUP(d_p,ϵ_p,S_c,V_f,V_c,W)
  for all nodes $i \in \mathcal{F}$ **do**
    $k \leftarrow 0$; $\mathcal{C}_i \leftarrow \varnothing$; $\mathbf{r} \leftarrow \mathbf{v}_i$; $\overline{\mathbf{v}}_i \leftarrow \mathbf{v}_i$
    Form $\mathcal{J}_i \leftarrow \{j \in \mathcal{C} \mid$ there is a path from i to j shorter than $d_p\}$
    while $\|\mathbf{r}\| \geq \epsilon_p \|\mathbf{v}_i\|$ **do**
      $k \leftarrow k + 1$
      Select $\bar{j} \in \mathcal{J}_i \setminus \mathcal{C}_i$ for which $\overline{\mathbf{v}}_j$ has maximal affinity with $\mathbf{r}$
      Update $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \{\bar{j}\}$
      Compute the Householder reflection operator Q by Equation 5.22
      Compute $\mathbf{r} \leftarrow Q\,\mathbf{r}$
      for all $j \in \mathcal{J}_i \setminus \mathcal{C}_i$ **do**
        Compute $\overline{\mathbf{v}}_j \leftarrow Q\,\overline{\mathbf{v}}_j$
      end for
    end while
    Form the $n_t \times k$ upper triangular matrix R collecting $\overline{\mathbf{v}}_j$, for all $j \in \mathcal{C}_i$
    Compute $\mathbf{w}_i \leftarrow R^{-1}\mathbf{r}$
  end for
end procedure

2. ϵ_p, the relative exit tolerance in the iterative procedure.

Note that, with no danger of confusion, we use the same symbol ϵ_p for both ABF and DPLS. Lastly, we present in Algorithm 13 the complete procedure for computing such prolongation operator.

The ability of DPLS prolongation to adapt to the problem at hand is shown in Figure 5.4 where an anisotropic diffusion problem is considered. The Poisson equation is solved on a two-dimensional square domain characterized by different diffusion tensors in the four regions NE, NW, SE and SW. SW is isotropic, while in the other regions a ratio of 100 between diffusivities in orthogonal directions and a different slope are used. Figure 5.4 (right) shows how aSP-AMG follows the material properties.

5.7 Numerical results

In this section, we present an analysis of the aSP-AMG preconditioner in two stages. First, the impact of the configuration parameters in its performance is studied considering the solution of three model problems. Next, the computational performance of aSP-AMG in terms of CPU time and memory occupation is evaluated considering a set of SPD sparse matrices arising from real-world engineering applications. Moreover, we

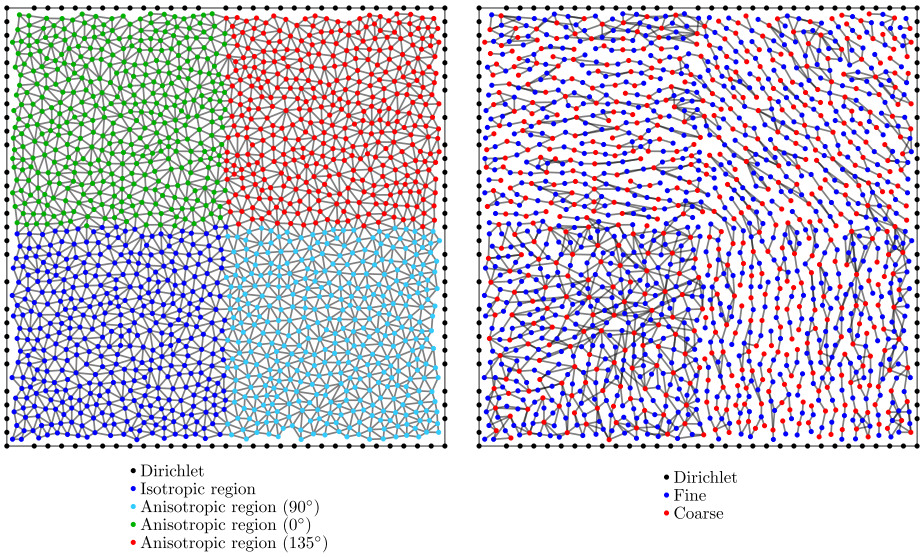

FIGURE 5.4: Anisotropic diffusion problem. Computational grid (left), adjacency graph of the DPLS prolongation (right).

compare our new method with three other preconditioners, namely the native aFSAI algorithm, as implemented in the FSAIPACK package Janna et al. [2015b]; BoomerAMG, as provided by the Hypre package, version 2.13 Falgout and Yang [2002] and GAMG, a smoothed aggregation algebraic multigrid implementation present from the PETSc package, version 3.9.0 Balay et al. [2018].

The right hand side vector used in the test cases discussed here are unitary. The linear systems are solved by the preconditioned conjugate gradient method (PCG) with initial solution equal to the null vector. Lastly, convergence is considered achieved when the PCG iterative residual becomes smaller than $10^{-10} \cdot ||\mathbf{b}||_2$.

To evaluate the memory occupation and the cost for applying aSP-AMG in a single V-cycle, we use the classical definition of operator and grid complexities, shown in equations (5.25) and (5.26), respectively:

$$C_{op} = \sum_{l=1}^{nl} \text{nnz}\,(A_l) \Big/ \text{nnz}\,(A_1), \tag{5.25}$$

$$C_{gd} = \sum_{l=1}^{nl} \text{nrows}\,(A_l) \Big/ \text{nrows}\,(A_1). \tag{5.26}$$

The first accounts for the space needed to store the sparse systems belonging to the multigrid hiearchy while the second measure is related to the size of auxiliary vectors employed in the preconditioner application. Moreover, the grid complexity also gives

an idea of how fast unknowns are coarsened in the multilevel hierarchy and can be easily related to the average grid contraction factor r_{avg} by the following expression:

$$C_{gd} = \sum_{l=1}^{nl} r_{\text{avg}}^{l} = \frac{1 - r_{\text{avg}}^{nl}}{1 - r_{\text{avg}}}. \tag{5.27}$$

The third measure is the cycle complexity which is borrowed from the work Manteuffel et al. [2017]. This gives an estimate of the application cost of one multigrid cycle in comparison to the cost of one SpMV operation involving the original matrix A. This quantity accounts for the cost of application of the smoother and interpolation operators as well as the residual computations present in the algorithm 8. It is also dependent on the type of multigrid cycle used and for the V-cycle it is given by

$$C_{cv} = \left[\sum_{l=1}^{nl} (1 + \nu_T) \text{nnz}\,(A_l) + 2 \sum_{l=1}^{nl-1} (\nu_T) \text{nnz}\,(F_l) + \text{nnz}\,(P_l) \right] \Big/ \text{nnz}\,(A_1), \tag{5.28}$$

in which the variable ν_T is equal to $\nu_1 + \nu_2$, i.e., the total number of pre and post-smoothing steps per level.

Lastly, we consider the aFSAI density μ_G, i.e. the ratio between the nonzeroes in the G factor and the nonzeroes in A, which gives an idea of the cost for storing as well as for applying of aFSAI preconditioner:

$$\mu_G = \frac{\text{nnz}\,(G)}{\text{nnz}\,(A)}. \tag{5.29}$$

When aFSAI is considered as a smoother, its density is simply given by an average through levels:

$$\mu_G = \frac{\sum_{l=0}^{n_l-1} \text{nnz}\,(G_l)}{\sum_{l=0}^{n_l-1} \text{nnz}\,(A_l)}. \tag{5.30}$$

We stress once more that our focus is not on the parallelism of the proposed approach but on its algorithmic definition. Nevertheless, to show that there are no obstacles to its parallelization, we present a shared memory implementation developed by

using OpenMP directives and run all the tests on a workstation equipped with two Intel Xeon E5-2643 processors at 3.30GHz with 4 cores each and 256 GB of RAM memory.

5.7.1 Sensitivity analysis

The proposed solver depends upon a considerable number of user-defined parameters that varies according to the strategies selected for computing each of its building blocks. Then, it is desirable to understand which of these parameters play a significant role in the construction of an efficient preconditioner as well as their range of suitable values. For this reason, we run a sensitivity analysis on aSP-AMG by testing several setup configurations and applying the resulting preconditioner for the solution of three model problems that exhibit laborious characteristics from the numerical viewpoint. To better understand the relative impact of each parameter, we vary only one or few of them at a time, leaving the other unchanged to the "default" values listed in Table 5.1.

TABLE 5.1: Default parameters for the sensitivity analysis.

Phase	**Method**	**Parameter**	**Value**
Smoother	aFSAI	k_g	5
		ρ_g	3
		ϵ_g	10^{-2}
Test Space	Lanczos	n_t	10
		n_s	100
		ϵ_t	10^{-1}
Coarsening	Affinity-based	θ	5
Prolongation	ABF	k_p	20
		ρ_p	1
		ϵ_p	10^{-2}
	DPLS	d_p	2
		ϵ_p	10^{-2}

A detailed description of the model problems is given below

P1: a three-dimensional Poisson simulation ($\nabla \cdot (K \nabla u) = 0$) on a unitary cube composed by two anisotropic materials, namely soft and hard, that are disposed in a

$16 \times 16 \times 16$ checkboard pattern with conductivity matrices given by

$$K_{soft} = \begin{bmatrix} 10.0 & 5.0 & 2.0 \\ 5.0 & 8.0 & 2.0 \\ 2.0 & 2.0 & 2.0 \end{bmatrix}, \text{ and } K_{hard} = \begin{bmatrix} 1.0 & 0.8 & 0.2 \\ 0.8 & 0.5 & 0.4 \\ 0.2 & 0.4 & 4.0 \end{bmatrix}. \tag{5.31}$$

The boundary faces located on the XY, and XZ planes are impermeable ($\nabla u \cdot \mathbf{n} = 0$) while the remaining ones on plane YZ show dirichlet boundary conditions equal to 0 and 1 for $x = 0$ and $x = 1$, respectively. The domain is discretized with $262,144$ ($64 \times 64 \times 64$) linear hexahedral elements leading to a sparse matrix with $274,625$ rows and $7,189,057$ nonzero terms. The Figure 5.5 ilustrates the problem geometry and mesh on the left and a plot of the resultant sparse matrix on the right.

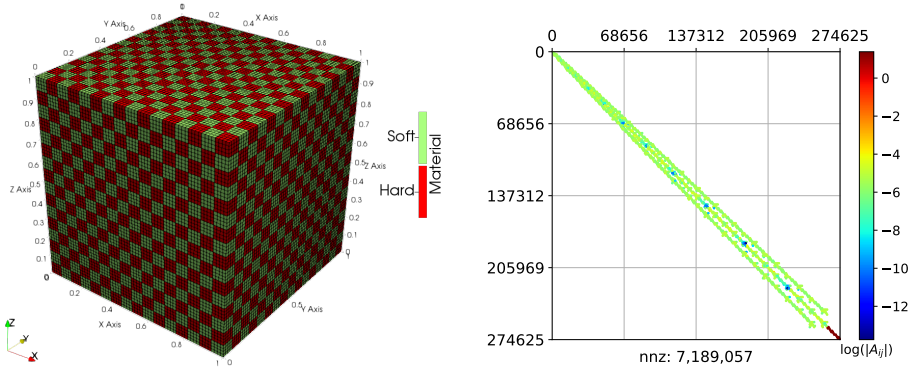

FIGURE 5.5: Left: the discretized domain of problem **P1** with colors representing different materials. Right: sparsity pattern of the pertinent discretization matrix colored according to the order of magnitude of the nonzero coefficients. The last matrix coefficients depicted in dark red represent the boundary conditions.

P2: a three-dimensional linear elasticity problem where the Cauchy indefinite equilibrium equations are solved on a cube with unitary edges. The domain is firstly discretized with 1,193 linear tetrahedral finite elements and the bottom face is completely fixed as boundary condition. A constant Poisson ratio $\nu = 0.3$ is assumed while the Young modulus E varies in the range $[10^{-\alpha}, 10^{\alpha}]$, with $\alpha = 3$. For each element, E_i is computed drawing σ_i from a uniform distribution and computing $E_i = 10^{\sigma_i}$. Once the material properties are defined in the whole

domain, the mesh is refined thus generating 610,816 tetrahedra with every new finite element inheriting the material properties of the original tetrahedron in which it is located. In such a way, we have homogeneous aggregates of finite elements with jumps in the material properties among aggregates. The final number of unknowns and nonzero entries for this matrix are 393,102 and 15,767,442, respectively. Figure 5.6 provides the FE mesh with different colors used to distinguish materials.

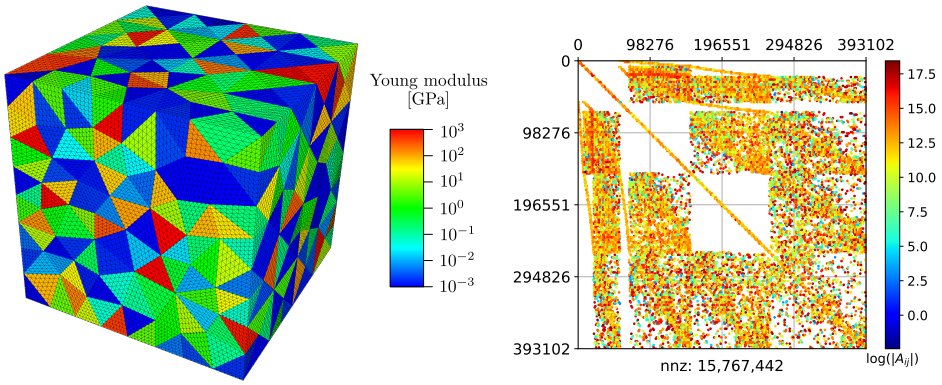

FIGURE 5.6: Distribution of the Young modulus across the domain of the problem **P2**.

P3: mechanical equilibrium of a heterogeneous cylinder composed by four coaxial layers of different materials. The cylinder has unitary diameter and height and it is under compression in the top face by a uniformly distributed pressure of $10MPa$ and clamped in the bottom face. Linear elasticity equations are used to model the equilibrium of forces on this structure. Mechanical properties referrent to the layers as well as their limits are provided in Table 5.2 The mesh is built

TABLE 5.2: Materials properties and layers limits for problem **P3**. r_i and r_o denotes the inner and outer radius of each layer, respectively; E and ν are the Young modulus and Poisson's coefficient of the material.

Layer	$\mathbf{r_i}[m]$	$\mathbf{r_o}[m]$	$\mathbf{E}[GPa]$	$\boldsymbol{\nu}$
1	0.00	0.10	1.0	0.35
2	0.10	0.25	10.0	0.45
3	0.25	0.40	0.1	0.25
4	0.40	0.50	1.0	0.20

with Gmsh Geuzaine and Remacle [2009] by using the default program options

and a maximum element size of 0.1 which gives 590,266 tetrahedron. Discretization of the mathematical model is done by first order continuous Galerkin FEM which generated a sparse matrix with $316,623$ rows and $13,851,585$ nonzero coefficients. In Figure 5.7 we show the deformed configuration of the body as well as a sparsity pattern plot of the matrix.

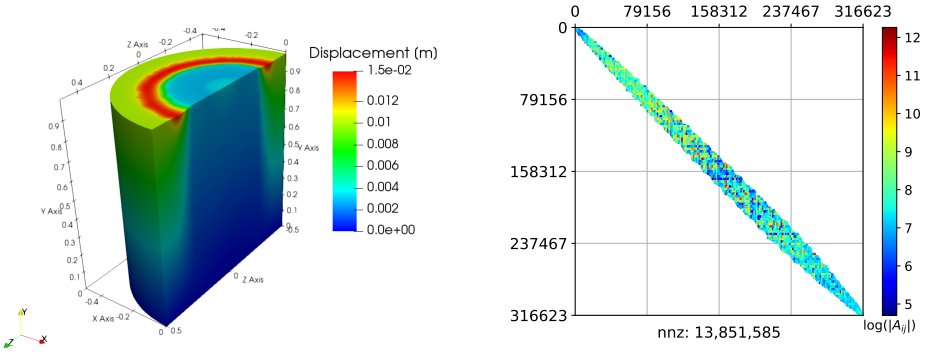

FIGURE 5.7: Left: a cross-section of the deformed mesh colored with the magnitude of the displacement vector after the solution of **P3**. Right: sparsity pattern of the matrix relative to **P3** where the colors indicate the values of the nonzero coefficients.

In particular, we provide the following information in each analysis:

- n_l: number of levels;
- C_{gd}: grid complexity;
- C_{op}: operator complexity;
- C_{fs}: aFSAI complexity;
- n_{it}: number of iterations for convergence;
- T_{ts}: time spent in the test space calculation phase;
- T_{cs}: time spent in the coarsening phase;
- T_{sm}: time spent in the aFSAI smoother calculation phase;
- T_{pl}: time spent in the prolongation computation phase;
- T_p: time spent in the setup of aSP-AMG;

- T_s: time spent in the solver application;
- T_t: total solution time spent until convergence.

We start the sensitivity analysis by varying the input parameters for building the smoother. Six configurations are used and the results obtained are shown in Table 5.3. The first and simplest configuration gives rise to a weighted Jacobi relaxation, meanwhile the following ones calculate aFSAI factors G with increasing density. From Table 5.3, we see that the iteration counts for all test problems reduce when C_{fs} increases, i.e., the situation where more accurate aFSAI factors are computed. We note that such effect is basically driven by increasing k_g and ρ_g or decreasing ϵ_g. We see also that the smoother has little impact on the multigrid hierarchy since the grid and operator complexities remain practically constant for each test problem. As shown by Janna et al. [2015b], the setup cost of G is proportional to the fourth power of its average number of nonzeros per row, thus, the smoother setup cost can be high even for small values of k_g. In fact, the smoother calculation time T_{sm} increases with C_{fs} up to the point of representing more than 90% of the total solution time T_t as for the last configuration. This leads to the conclusion that an efficient aFSAI smoother must be associated with a sufficiently sparse factor G with cheap computation cost while maintaining a good smoothing property. In particular, we note that the second configuration provides the most efficient option one for all problems, except for **P1** since, due to its simplicity, weighted Jacobi has proven to be the most efficient relaxation option. Such configuration is in close accordance with the one considered in Table 5.1.

For the test space computation, we divide the sensitivity analysis in three steps by considering first the Lanczos method given by Algorithm 10 and later, the SRQCG method given by Algorithm 11 with different initial guesses for the test space X_0. For the Lanczos method, the input parameters being varied are the number of test vectors n_t, the dimension of the Lanczos space n_v, and the convergence tolerance ϵ_s. Seven configurations are tested and the results obtained are showed in Table 5.4. Although the grid and operator complexities are more affected by the test space selection than by the smoother, such effect is still small. The same observation can be verified for C_{fs}. In

TABLE 5.3: aSP-AMG sensitivity to the smoother selection.

Test case	Parameters			Results										
	k_g	ρ_g	ϵ_g	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P1	0	1	10^0	1.97	2.54	0.30	38	1.2	0.8	0.0	0.1	2.2	0.8	3.0
	5	1	10^0	2.00	2.53	1.77	26	1.9	1.0	1.5	0.2	4.7	0.9	5.6
	5	3	10^0	2.01	2.53	4.74	20	1.9	0.7	2.2	0.1	5.1	0.7	5.8
	10	3	10^{-3}	2.02	2.51	9.05	20	2.9	0.8	10.3	0.2	14.2	0.9	15.1
	30	1	10^{-3}	2.02	2.54	7.84	19	2.5	0.7	19.4	0.1	22.8	0.8	23.6
	30	3	10^{-3}	2.04	2.49	17.00	16	3.9	0.7	46.8	0.1	51.6	1.1	52.6
P2	0	1	10^0	1.81	2.66	0.18	687	2.4	2.0	0.0	0.4	5.0	34.5	39.5
	5	1	10^0	1.81	2.93	1.01	127	3.3	2.4	2.4	0.5	8.5	7.8	16.3
	5	3	10^0	1.81	2.97	2.62	87	4.9	3.4	6.1	0.7	15.1	7.0	22.1
	10	3	10^{-3}	1.81	3.00	4.12	65	5.3	2.6	15.4	0.5	23.8	5.6	29.4
	30	1	10^{-3}	1.81	3.02	2.86	74	4.4	2.5	26.2	0.5	33.6	5.9	39.5
	30	3	10^{-3}	1.81	3.02	6.74	52	6.5	2.8	91.2	0.5	101.1	5.6	106.7
P3	0	1	10^0	1.68	3.02	0.15	184	2.7	2.7	0.0	0.6	6.1	8.1	14.2
	5	1	10^0	1.70	3.02	0.92	90	3.1	2.7	2.2	0.6	8.6	4.7	13.3
	5	3	10^0	1.70	2.99	2.45	76	3.6	2.6	4.5	0.6	11.4	5.2	16.6
	10	3	10^{-3}	1.71	2.95	4.70	61	3.5	1.8	12.5	0.4	18.2	4.0	22.2
	30	1	10^{-3}	1.71	2.96	4.13	60	3.5	1.8	30.4	0.4	36.1	3.7	39.9
	30	3	10^{-3}	1.72	2.90	10.39	41	5.3	1.7	83.0	0.4	90.5	3.8	94.3

terms of iteration count, the results for **P1** are very similar, however, we see that the remaining test cases require more test vectors for a better convergence as long as smaller accuracy in their calculation is allowed, i.e., small values of ϵ_s. The cost of computing the test space, measured by T_{ts}, increases fast with the Lanczos dimension and the default configuration value for n_v gives the smallest computation times. Overall, the test space computation cost can represent up to 80% of the total solution time, while the optimal test space configuration represents nearly $25 - 40\%$ of T_t.

One of the main drawbacks of the Lanczos method is the fact that it does not allow the use of a priori knowledge about the near-null space components of A. With the SRQCG method, presented in section 5.4.3, an initial guess X_0 can be used for computing the test space matrix. Now, we evaluate such strategy in two different scenarios: first, a random initial guess is considered, while, in the second, the rigid body modes, which are the nullspace components of the linear elasticity operator prior application of the boundary conditions, are used. Table 5.5 gives the results for the first case. In terms of iteration count, we note that the SQRCG gives slightly better results than Lanczos, also, the total solution times of the fastest runs are pretty similar to the fastest runs

TABLE 5.4: aSP-AMG sensitivity to the test space selection - Algorithm 10.

Test case	Parameters			Results										
	n_t	n_v	ϵ_s	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P1	10	100	10^{-1}	2.11	2.17	4.97	21	2.0	0.6	2.1	0.1	4.9	0.7	5.6
	20	100	10^{-1}	2.07	2.31	4.88	20	2.6	0.7	2.1	0.1	5.5	0.7	6.2
	30	100	10^{-1}	2.02	2.41	4.78	19	3.6	1.0	2.7	0.2	7.7	0.8	8.5
	10	150	10^{-1}	2.12	2.14	5.00	20	3.3	0.8	2.8	0.2	7.0	0.9	7.9
	30	200	10^{-1}	2.08	2.29	4.92	19	6.3	0.9	2.8	0.2	10.3	0.8	11.1
	30	200	10^{-2}	2.06	2.35	4.86	18	15.6	1.0	2.7	0.2	19.6	0.8	20.4
	30	200	10^{-3}	2.10	2.24	4.95	20	18.9	0.9	2.8	0.2	22.9	0.9	23.8
P2	10	100	10^{-1}	1.92	2.23	2.61	121	4.6	1.8	4.0	0.3	10.8	10.9	21.7
	20	100	10^{-1}	1.91	2.26	2.59	118	6.2	2.1	4.5	0.5	13.3	7.8	21.1
	30	100	10^{-1}	1.89	2.31	2.57	119	9.0	2.1	4.4	0.5	16.1	8.0	24.0
	10	150	10^{-1}	1.91	2.27	2.60	109	6.6	2.0	4.6	0.4	13.6	9.9	23.5
	30	200	10^{-1}	1.86	2.54	2.54	86	11.8	2.7	4.6	0.6	19.8	6.4	26.2
	30	200	10^{-2}	1.89	2.44	2.57	146	25.8	2.0	3.8	0.4	32.1	10.0	42.1
	30	200	10^{-3}	1.94	2.23	2.63	314	27.5	1.7	4.0	0.3	33.7	20.7	54.4
P3	10	100	10^{-1}	1.79	2.33	2.58	114	3.4	1.7	3.6	0.4	9.2	7.0	16.2
	20	100	10^{-1}	1.69	2.93	2.44	64	4.3	2.7	3.8	0.7	11.5	4.4	16.0
	30	100	10^{-1}	1.66	3.08	2.39	62	4.3	2.1	3.0	0.5	10.0	3.4	13.4
	10	150	10^{-1}	1.83	2.20	2.63	105	4.4	1.6	3.5	0.3	9.9	6.4	16.3
	30	200	10^{-1}	1.69	2.91	2.43	59	9.0	2.6	4.4	0.6	16.7	3.4	20.1
	30	200	10^{-2}	1.78	2.64	2.56	213	20.1	1.8	3.4	0.5	25.8	10.7	36.5
	30	200	10^{-3}	1.89	2.32	2.73	220	17.9	1.3	3.0	0.3	22.6	10.8	33.4

given by the Lanczos method. From these results we see that even if a innacurate approximation of the near-null space components are used, the SRQCG method can still find a sufficiently good test space matrix.

TABLE 5.5: aSP-AMG sensitivity to the test space selection - Algorithm 11 with random initial guess X_0.

Test case	Parameters			Results										
	n_t	n_{it}	ϵ	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P1	6	5	10^{-1}	2.06	2.25	1.22	17	1.0	0.8	2.7	0.1	4.8	0.8	5.6
	6	10	10^{-1}	2.08	2.20	1.23	18	1.0	0.7	2.5	0.1	4.6	0.8	5.3
	6	30	10^{-1}	2.08	2.20	1.23	18	0.9	0.6	2.1	0.1	3.9	0.6	4.5
	10	10	10^{-1}	2.08	2.24	1.23	16	1.5	0.6	2.2	0.1	4.6	0.6	5.2
	20	10	10^{-1}	2.07	2.29	1.22	14	3.9	0.9	2.8	0.2	8.0	0.6	8.6
	30	10	10^{-2}	2.07	2.31	1.22	14	7.4	0.9	2.6	0.2	11.3	0.5	11.8
	30	10	10^{-3}	2.07	2.31	1.22	14	6.8	0.8	2.1	0.1	10.1	0.5	10.7
P2	6	5	10^{-1}	1.76	2.89	0.61	184	2.0	3.5	4.8	0.7	11.7	17.9	29.6
	6	10	10^{-1}	1.78	2.85	0.61	163	1.5	2.3	3.9	0.5	8.8	11.6	20.4
	6	30	10^{-1}	1.79	2.76	0.62	142	2.7	2.4	4.0	0.5	10.4	10.2	20.5
	10	10	10^{-1}	1.77	2.90	0.61	115	3.9	3.4	4.7	0.7	13.6	11.3	24.8
	20	10	10^{-1}	1.76	2.97	0.61	97	7.3	3.5	4.5	0.8	17.0	7.2	24.2
	30	10	10^{-2}	1.76	2.94	0.61	89	14.3	3.6	4.7	0.8	24.2	8.7	33.0
	30	10	10^{-3}	1.76	2.94	0.61	89	12.6	2.8	4.2	0.7	21.0	6.4	27.5
P3	6	5	10^{-1}	1.68	2.89	0.60	126	1.4	2.6	3.8	0.6	8.8	8.6	17.4
	6	10	10^{-1}	1.70	2.84	0.61	90	1.6	2.5	3.8	0.5	8.8	4.7	13.5
	6	30	10^{-1}	1.74	2.65	0.63	74	2.0	1.5	2.9	0.3	7.2	3.8	11.0
	10	10	10^{-1}	1.68	2.91	0.61	66	2.4	2.2	3.5	0.6	9.0	3.5	12.6
	20	10	10^{-1}	1.68	3.01	0.60	58	5.1	2.2	3.5	0.5	11.7	3.1	14.8
	30	10	10^{-2}	1.68	2.99	0.60	54	10.0	2.8	3.8	0.7	17.8	3.7	21.5
	30	10	10^{-3}	1.68	2.99	0.60	54	8.8	2.2	3.5	0.5	15.4	2.9	18.4

In Table 5.6, we show results for the case of using the rigid body modes as an initial guess for Algorithm 11. Since this makes sense only for linear elasticity problems, results for the test case **P2** are ommitted. Initially, we note that the grid, operator and aFSAI complexities follow the same trend as in the last scenario. If only 6 components are used, which is the number of rigid body modes for the three-dimensional test cases, the initial guess gives a sufficiently good approximation to X lowering the computation time of the test space T_{ts} and, consequently, decreasing the total run time T_t. In case more test vectors than the ones provided by X_0 are used, the performance of SRQCG is practically the same wheter using an initial guess X_0 or not.

In Table 5.7, we show the sensitivity analysis of aSP-AMG with respect to its coarsening calculation phase. We want to test the performance of the method when the

TABLE 5.6: aSP-AMG sensitivity to the test space selection - Algorithm 11 with X_0 given by the rigid body modes.

Test case	Parameters			Results										
	n_t	n_{it}	ϵ	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P2	6	5	10^{-1}	1.93	2.15	0.66	151	1.7	1.9	4.5	0.4	9.2	13.6	22.8
	6	10	10^{-1}	1.94	2.16	0.66	153	2.3	1.9	4.6	0.4	9.7	13.9	23.6
	6	30	10^{-1}	1.93	2.16	0.66	153	1.5	1.3	3.7	0.2	7.4	10.2	17.6
	10	10	10^{-1}	1.80	2.67	0.62	101	2.9	2.8	4.6	0.6	11.6	7.7	19.3
	20	10	10^{-1}	1.76	2.93	0.61	84	7.2	3.4	5.7	0.8	17.9	6.1	24.0
	30	10	10^{-2}	1.77	2.93	0.61	80	14.4	3.6	4.8	0.8	24.4	7.8	32.2
	30	10	10^{-3}	1.77	2.93	0.61	81	11.1	2.5	4.0	0.5	18.9	5.9	24.8
P3	6	5	10^{-1}	1.80	2.15	0.65	90	0.8	1.1	2.9	0.2	5.3	4.2	9.5
	6	10	10^{-1}	1.81	2.14	0.65	88	0.9	1.1	2.9	0.2	5.4	4.1	9.5
	6	30	10^{-1}	1.81	2.14	0.65	88	1.4	1.5	3.5	0.3	7.2	5.0	12.2
	10	10	10^{-1}	1.74	2.65	0.62	66	1.6	1.6	3.0	0.3	6.9	3.4	10.3
	20	10	10^{-1}	1.69	2.95	0.61	56	4.1	1.9	3.1	0.5	9.9	3.0	12.9
	30	10	10^{-2}	1.68	2.97	0.60	53	7.8	1.9	3.0	0.5	13.7	2.8	16.5
	30	10	10^{-3}	1.68	2.97	0.60	53	10.0	2.8	3.8	0.7	17.8	3.7	21.4

average number of strong connections per node θ from a lower bound equal to 2 up to an upper bound equal to 8. A common behavior to all numerical runs is the decrease in the grid and operator complexities when increasing the value of θ, followed by a slightly increase in the total solution time. Moreover, looking at the iteration counts we conclude that the preconditioning operator quality deteriorates when increasing θ towards 8. As expected, the parameter θ affects mostly the coarsening time. However, it is worth noting that the test space and smoother computation times (T_{ts} and T_{sm}) are influenced indirectly by the way that the multigrid hierarchy is built, and consequently, by the value of θ. In summary, we see that such parameter has a significant effect in all setup phases aSP-AMG and therefore is an important parameter in our method. Moreover, we note that the default choice of θ lead to a satisfactory performance in all test cases.

In table 5.8, the DPLS prolongation parameters are varied. It is seen that both d_p and ϵ_p are important. A too large value of d_p, e.g. $d_p = 3$, causes a serious increase of the operator complexity, with a larger preconditioning time, which is not counterbalanced by a consequent reduction in the number of iterations. If ϵ_p is too high, e.g. $\epsilon_p = 10^{-1}$, the prolongation is inaccurate and the AMG-cycle becomes ineffective.

In table 5.9, the ABF prolongation is tested. Despite the fact that both the number of

TABLE 5.7: aSP-AMG sensitivity to the coarsening variable.

Test case	Param.	Results										
	θ	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P1	2	1.75	1.86	1.02	24	1.8	0.5	2.2	0.2	4.8	0.7	5.4
	5	1.46	1.62	0.85	28	1.6	0.5	1.7	0.4	4.2	0.8	5.0
	8	1.33	1.51	0.77	31	1.4	0.5	1.5	0.5	3.9	0.9	4.8
P2	2	1.61	1.96	0.55	145	4.2	1.8	3.7	0.7	10.5	9.8	20.3
	5	1.33	1.87	0.45	233	2.8	1.4	2.8	1.2	8.2	12.0	20.2
	8	1.22	1.73	0.41	346	3.3	2.0	2.8	2.3	10.4	17.0	27.4
P3	2	1.48	2.23	0.53	120	2.4	1.3	2.5	0.6	6.8	5.2	12.1
	5	1.27	1.95	0.46	163	2.3	1.5	2.4	1.4	7.7	6.5	14.2
	8	1.19	1.73	0.43	304	2.1	1.4	2.1	1.9	7.6	11.0	18.6

TABLE 5.8: aSP-AMG sensitivity to the prolongation selection - DPLS.

Test case	Parameters			Results									
	d_p	n_{max}	ϵ_p	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P1	1	10	10^{-3}	2.03	2.45	1.20	20	2.0	0.7	2.2	0.1	5.0	0.7	5.7
	2	10	10^{-3}	1.99	2.62	1.17	20	2.4	1.0	2.5	0.2	6.1	0.9	7.0
	3	10	10^{-3}	1.91	3.31	1.13	20	2.6	1.4	2.6	0.7	7.3	1.0	8.3
	2	5	10^{-3}	1.99	2.62	1.17	20	2.2	0.8	2.6	0.2	5.7	0.7	6.5
	2	5	10^{-2}	2.01	2.53	1.19	20	2.3	0.9	2.5	0.2	6.0	0.9	6.9
	2	5	10^{-1}	2.11	2.09	1.24	21	2.3	0.7	2.5	0.1	5.6	0.9	6.5
P2	1	10	10^{-3}	1.85	2.57	0.63	96	3.9	1.7	3.9	0.2	9.7	6.7	16.3
	2	10	10^{-3}	1.79	3.41	0.62	98	5.9	5.2	5.2	1.1	17.4	10.6	28.0
	3	10	10^{-3}	1.72	5.14	0.60	149	6.5	12.2	6.1	17.9	42.7	20.2	62.9
	2	5	10^{-3}	1.79	3.27	0.62	76	4.4	2.9	4.3	0.6	12.3	5.9	18.2
	2	5	10^{-2}	1.81	3.04	0.62	81	4.7	3.1	4.6	0.6	13.1	6.1	19.1
	2	5	10^{-1}	1.96	2.08	0.66	120	4.5	1.8	4.3	0.3	11.1	10.6	21.7
P3	1	10	10^{-3}	1.76	2.51	0.63	93	3.5	1.9	3.6	0.2	9.3	5.9	15.3
	2	10	10^{-3}	1.68	3.38	0.60	87	3.9	3.5	4.1	1.0	12.5	6.7	19.2
	3	10	10^{-3}	1.61	5.31	0.58	115	4.7	6.3	4.7	10.5	26.3	8.9	35.2
	2	5	10^{-3}	1.68	3.31	0.61	88	3.1	2.2	3.2	0.7	9.3	5.0	14.3
	2	5	10^{-2}	1.70	3.12	0.61	92	3.7	2.9	4.0	0.7	11.3	6.6	17.9
	2	5	10^{-1}	1.86	2.04	0.67	112	3.0	1.4	3.5	0.3	8.2	6.6	14.8

iterations and the solution time are significantly higher than the previous tests where DPLS is used as default, it is seen that the only parameter really affecting this approach is the choice of the least squares correction. When the pattern of a poor ABF (e.g. $k_p = 20$ and $\epsilon_p = 10^{-2}$) is used, updating the prolongation in a least square sense reduces both the number of iterations and the solution time. By contrast, in case of a very accuarate ABF (e.g. $k_p = 40$ and $\epsilon_p = 10^{-4}$), the least square correction increases both the number of iterations and the solution time.

TABLE 5.9: aSP-AMG sensitivity to the prolongation selection - ABF.

Test case	Parameters			Results									
	κ_p	ρ_p	ϵ_p	C_{gd}	C_{op}	C_{fs}	n_{it}	$T_{ts}[s]$	$T_{cs}[s]$	$T_{sm}[s]$	$T_{pl}[s]$	$T_p[s]$	$T_s[s]$	$T_t[s]$
P1	5	1	10^{-3}	1.86	5.09	4.36	49	3.1	0.9	2.9	0.7	8.8	2.4	11.2
	5	3	10^{-3}	1.86	7.84	3.94	38	3.3	1.1	3.3	1.6	12.1	2.4	14.5
	15	1	10^{-3}	1.69	7.54	3.94	38	3.4	1.3	3.5	4.2	14.7	2.3	17.0
	15	3	10^{-3}	1.63	11.13	3.81	32	4.2	1.6	3.8	8.7	23.9	2.6	26.5
	30	1	10^{-3}	1.66	8.47	3.88	35	4.0	1.5	3.9	10.2	23.9	2.3	26.2
P2	5	1	10^{-3}	1.81	3.90	2.48	727	4.5	1.0	4.4	0.9	13.5	63.2	76.8
	5	3	10^{-3}	1.72	6.51	2.37	529	7.6	2.7	6.8	2.9	30.8	68.3	99.2
	15	1	10^{-3}	1.73	5.89	2.38	571	6.4	2.0	5.3	5.5	25.5	64.0	89.5
	15	3	10^{-3}	1.60	8.66	2.20	412	7.9	2.7	6.5	11.3	41.4	61.1	102.5
	30	1	10^{-3}	1.67	6.36	2.30	479	8.2	2.7	7.4	16.4	45.9	63.9	109.9
P3	5	1	10^{-3}	1.74	3.87	2.48	285	3.3	1.1	3.9	0.9	10.2	18.5	28.7
	5	3	10^{-3}	1.63	5.71	2.34	230	4.3	1.5	3.9	1.9	14.9	19.4	34.3
	15	1	10^{-3}	1.65	5.94	2.37	227	4.6	1.6	4.1	5.7	19.4	19.3	38.7
	15	3	10^{-3}	1.55	9.63	2.22	215	7.6	2.6	6.9	13.7	41.8	26.6	68.4
	30	1	10^{-3}	1.59	7.45	2.28	203	5.6	2.2	4.7	16.5	34.2	20.6	54.8

This sensitivity analysis shows that the DPLS prolongation is by far the most effective prolongation strategy, and the only parameters that really impact the perfomance of aSP-AMG are those controlling DPLS. As to the other parameters, especially those controlling the smoother, it is better to avoid configurations leading to too high setup costs. According to this general guideline, different choices in the user-defined parameters lead to almost the same performance, which simplifies the tuning of aSP-AMG towards its optimal configuration parameters.

5.7.2 Weak scalability

In this section we evaluate the weak scalability of aSP-AMG, that is how the iteration count needed for convergence changes by varying the size of a problem arising from

the discretization of PDEs. Due to its major effectiveness, only DPLS prolongation is considered in this analysis that comprises two test problems, a diffusion and a linear elasticity test case. The software Gmsh Geuzaine and Remacle [2009] is used to build the geometry and the meshes considered herein while the numerical discretization relies on the MFEM package Kolev and Dobrev [2010]. For comparison purposes, we provide also the results obtained with the BoomerAMG preconditioner configured with its default parameters as available in the current package Henson and Yang [2002], version 2.11.2, which is one of the most commonly used classical AMG implementations available in the literature.

Starting with the diffusion test case, we consider an isotropic Poisson model with unitary diffusivity coefficients and homogenous boundary conditions, as described by the PDE problem:

$$\begin{aligned} \Delta u &= 1, \quad \text{for } \Pi \in \mathbb{R}^3 \mid \Pi = [0,1]^3 \\ u &= 0 \quad \text{on } \partial\Pi. \end{aligned} \tag{5.32}$$

The system is discretized by linear hexahedral finite elements with size h. In table 5.10, we report the main characteristics of the sparse matrices generated from the FEM discretization of (5.32) as well as the performance of the aSP-AMG and BoomerAMG preconditioners in terms of grid and operator complexities as well as number of iterations to converge according to the mesh refinement.

TABLE 5.10: Weak scalability of the Poisson problem.

Matrix Info			**aSP-AMG(2)**			**aSP-AMG(N)**			**BoomerAMG**		
h^{-1}	nrows	nnz	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}
8	729	15,623	1.07	1.07	4	1.07	1.07	4	1.32	1.51	6
16	4,913	117,645	1.33	1.35	5	1.51	1.55	5	1.60	1.72	6
32	35,937	912,669	1.40	1.47	6	1.71	2.03	7	1.79	1.98	6
64	274,625	7,189,053	1.50	1.53	7	1.99	2.29	10	1.92	2.21	7
128	2,146,689	57,066,621	1.52	1.57	8	2.00	2.43	12	1.98	2.38	10

As expected, we note only a slight increase in the number of iterations for both aSP-AMG(2) and aSP-AMG(N) preconditioners when refining the mesh. We highlight the fact that aSP-AMG presents the same order of magnitude for n_{it} when going from

the two-level version to the multilevel one. This is a relevant observation showing that the inexact smoothing steps executed across the multigrid hierarchy degrades only marginally its convergence performance. Lastly, we remark that aSP-AMG(N) presents a very similar convergence behavior as BoomerAMG without having to raise substantially the operator complexity.

The second problem concerns an unitary cube formed by an homogeneous material under the hypothesis of small deformation. A linear elastic material property is assumed and the resulting PDE for calculating the deformation of this structure reads:

$$\begin{aligned} \nabla \cdot \left[\lambda \, tr\left(\frac{\left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right)}{2} \right) I + \mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^T\right) \right] = \mathbf{0} \quad & \text{for } \Pi = [0,1]^3, \\ \mathbf{u} = \mathbf{0} \quad & \text{on } \partial\Pi_D = [0,1] \times [0,1] \times \{0\}, \\ \boldsymbol{\sigma} \cdot \mathbf{n} = 10 \quad & \text{on } \partial\Pi_N = [0,1] \times [0,1] \times \{1\}, \\ \boldsymbol{\sigma} \cdot \mathbf{n} = 0 \quad & \text{on } \partial\Pi \setminus \{\partial\Pi_D \cup \partial\Pi_N\} \end{aligned} \tag{5.33}$$

with λ and μ the Lamè constants corresponding to Young modulus and Poisson ratio equal to 10^3 GPa and 0.3, respectively.

Again, the problem is discretized by using linear hexahedral finite elements and the results are reported in table 5.11. First of all, we observe that aSP-AMG shows a slightly smaller increase of n_{it} with mesh in comparison to the diffusion model problem suggesting that this preconditioner is even more suitable for solving elasticity problems. Finally, we mention that aSP-AMG(N) performance is very close to aSP-AMG(2) and both lead to better convergence behaviors than BoomerAMG while still having similar grid and operator complexities.

5.7.3 Real-world engineering problems

In this section we evaluate how the aSP-AMG preconditioner behaves in the solution of sparse linear systems arising from the application of different discretization techniques to real-world engineering problems such as the the fluid flow in petroleum reservoirs, pressure-temperature field evolution in porous media, geomechanical simulations and mechanical equilibrium of linear elastic materials. The matrices used for these tests

TABLE 5.11: Weak scalability on the linear elasticity problem.

Matrix Info			**aSP-AMG(2)**			**aSP-AMG(N)**			**BoomerAMG**		
h^{-1}	nrows	nnz	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}	C_{gd}	C_{op}	n_{it}
8	2,187	140,625	1.36	1.48	10	1.51	1.73	12	1.67	1.87	14
16	14,739	1,058,841	1.38	1.59	11	1.54	2.00	13	1.65	1.97	17
32	107,811	8,214,057	1.41	1.68	11	1.56	2.26	13	1.68	2.15	19
64	823,875	64,701,513	1.41	1.73	12	1.56	2.40	14	1.69	2.28	24
128	6,440,067	513,599,625	1.40	1.76	14	1.54	2.42	21	1.70	2.35	31

have been chosen to be representative of ill-conditioned problems characterized by complex geometries and strong heterogeneities in both material properties and element size.

The dimension, number of nonzeroes, average number of nonzeroes per row together with a brief description of the above matrices are listed in the table A.1. For the sake of completeness, we also solve this set of problems with aFSAI, in order to understand how the multilevel hierarchy of aSP-AMG is able to complement its single level smoother and, lastly, with the BoomerAMG preconditioner.

Table 5.12 shows grid and operator complexities, aFSAI density, number of iterations and computational times for the preconditioners considered here. We note that these results were obtained with the best possible configuration for each of the preconditioners tested in terms of total solution time. In particular, for BoomerAMG, a vast package collecting several options for any AMG components, we carried out an optimization process involving all the available prolongation and coarsening algorithms over an extensive range of values for their input parameters. We stress that, for elasticity problems, the parameter bs defining the block size used for interpolation turned out to be of paramount importance to achieve fast convergence. Detailed information about the input parameters used in the setup of the preconditioners is provided in the appendix 5.8.

The main aspect to be noted here is that aSP-AMG leads to an efficient solution method in all test cases. More precisely, comparing the aSP-AMG to aFSAI, we see that the first one shows a reduction up to ten times in the number of iterations n_{it}. Also, the total computational time T_t is smaller in all experiments except for `bump2911` which

TABLE 5.12: Performance comparison between aFSAI, BoomerAMG and aSP-AMG preconditioners in the solution of the real-world engineering problems.

Matrix name	method	C_{gd}	C_{op}	μ_G	n_{it}	$T_p[s]$	$T_s[s]$	$T_t[s]$
pflow742	aFSAI	—	—	0.28	1465	1.4	24.9	26.3
	BoomerAMG	1.31	1.14	—	577	0.3	38.4	38.7
	aSP-AMG	1.54	1.87	0.58	85	9.5	5.7	15.2
finger	aFSAI	—	—	2.20	3711	5.2	199.0	204.2
	BoomerAMG	1.67	2.21	—	10	1.0	2.2	3.2
	aSP-AMG	1.71	2.22	2.05	43	18.2	8.3	26.5
spe10	aFSAI	—	—	1.55	1767	1.0	22.7	23.7
	BoomerAMG	1.55	2.23	—	36	0.4	2.2	2.6
	aSP-AMG	1.90	2.68	1.1	64	5.3	3.5	8.8
bump2911	aFSAI	—	—	0.46	507	13.6	45.3	58.9
	BoomerAMG	1.58	1.62	—	250	4.5	89.3	93.8
	aSP-AMG	2.17	2.94	0.53	42	59.9	23.8	83.7
flan1565	aFSAI	—	—	0.22	4230	9.9	179.2	189.1
	BoomerAMG	1.27	1.38	—	244	2.0	73.1	75.1
	aSP-AMG	1.38	1.74	0.21	136	31.9	30.6	62.5
abq_powtrain	aFSAI	—	—	0.72	2792	11.6	123.1	134.6
	BoomerAMG	1.47	1.45	—	526	3.7	241.4	245.1
	aSP-AMG	1.49	2.06	0.56	123	27.7	24.7	52.4
hook1498	aFSAI	—	—	0.40	2010	3.0	64.5	67.4
	BoomerAMG	1.60	3.16	—	72	5.8	35.6	41.4
	aSP-AMG	1.54	2.30	0.42	135	19.4	21.0	40.4

already present a fairly good convergence behavior when solved with aFSAI. However, even in these cases, we observe that the solution time T_s for aSP-AMG is smaller up to three times and if this preconditioner could be recycled such as in some transient simulation, its setup time would be amortized after two or three linear solves only.

Another interesting fact observed for our AMG is that its aFSAI density μ_G is always smaller than the operator complexity C_{op}, which in turn gives the cost for Gauss-Seidel smoothing. It follows that the aFSAI smoother yields a faster strategy, besides showing a higher degree of parallelism and thus be amenable to the implementation in massively parallel computers.

Comparing the results obtained with BoomerAMG and aSP-AMG, we note that the first one provides a faster method in terms of setup for all test cases, which can be explained by the fact that in aSP-AMG we have two additional setup phases contributing to T_p, i.e., the construction of the smoother and the test space. However, aSP-AMG is still competitive against BoomerAMG as we can see in the `pflow742`, `bump2911`, `flan1565` and `abq_powtrain` test cases (see Figure 5.8 for a comprehensive comparison). Indeed, the high setup time of aSP-AMG is compensated by the faster convergence of the method providing a smaller total computation time T_t. Ultimately, we observe that aSP-AMG tends to behave better in elasticity problems in comparison to the diffusion problems. This fact was already observed in the last section and suggests that this method proves to be more efficient when employed in the solution of matrices with larger near-null space dimension.

5.8 Input parameters for configuring the preconditioners

In Table 5.13 we show the input parameters used for configuring the BoomerAMG preconditioner applied in the solution of the real-world problems. We kept fixed most of the configurable parameters for this preconditioner and varied only the ones that could impact most the performance of the preconditioner, i.e., the interpolation and coarsening methods; the block size of the matrix bs; the number of levels with aggressive coarsening n_{agg}; the strong threshold θ and the maximum number of nonzeros per row of the prolongation operator P_{max}. Note that b_s is the only input parameter

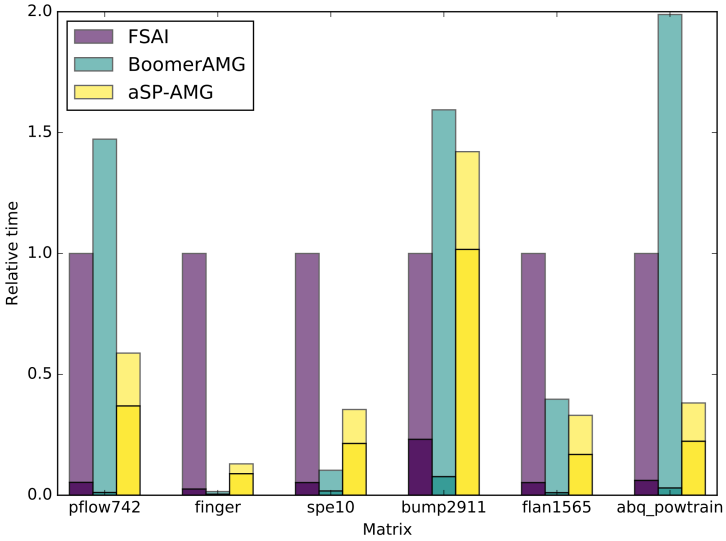

FIGURE 5.8: Total time comparison relative to the aFSAI preconditioner. The lower segment of each column represents the relative setup time while the upper segment denotes the solution counterpart.

which depends on the problem being handled. The best configuration was selected as the one which produced the fastest preconditioner in terms of total computational time (T_t). For completness, a list of other input parameters which were kept fixed are given below:

- relaxation method: symmetric-SOR/Jacobi with C/F ordering and $\omega = 1.0$;
- coarse system solver: Gaussian-elimination;
- measure type: local;
- interpolation truncation factor: 0.0;
- interpolation maximum row sum: 0.9;
- cycle type: V(1,1).

The input parameters for the aFSAI and aSP-AMG preconditioners are given in Table 5.14. Lastly, after an extensive optimization process for aSP-AMG, we remark that the best parameters offer a very similar performance as given by the default setup configuration listed in Table 5.1.

TABLE 5.13: Input parameters for configuring the BoomerAMG preconditioner.

Matrix name	**Interpolation**	**Coarsening**	b_s	n_{agg}	θ	P_{max}
`pflow742`	classical	CLJP	1	1	0.99	—
`finger`	classical	Falgout	1	0	0.25	—
`spe10`	classical	HMIS	1	1	0.70	10
`bump2911`	ext+i	Falgout	3	1	0.99	5
`flan1565`	ext+i	HMIS	3	1	0.70	5
`abq_powtrain`	classical	Falgout	3	1	0.99	—

TABLE 5.14: Input parameters for aFSAI and aSP-AMG, respectively.

Matrix name	**aFSAI**			**aSP-AMG**						
	k_g	ρ_g	ϵ_g	k_g	ρ_g	ϵ_g	n_t	θ	d_p	ϵ_p
`pflow742`	5	3	10^{-3}	5	4	10^{-3}	10	6	2	10^{-2}
`finger`	5	2	0	5	1	10^{-3}	5	5	1	10^{-3}
`spe10`	10	2	10^{-3}	3	1	0	8	4	1	10^{-2}
`bump2911`	10	2	10^{-3}	10	1	10^{-3}	10	2	2	10^{-3}
`flan1565`	15	1	0	5	2	0	15	8	2	10^{-2}
`abq_powtrain`	10	2	0	5	3	10^{-3}	15	6	2	10^{-3}

5.9 Conclusions

In this chapter, we propose a novel AMG package featuring approximate inverse smoothing and three new strategies for computing the prolongation operator. This method, entitled as aSP-AMG, belongs to the recent adaptive and bootstrap AMG family, which may not assume any information about the constitution of the near null space of A, rendering it more general and robust than the classical and aggregation based AMG methods.

A set of artificial and real-world problems are solved in order to assess the performance of aSP-AMG. Initially, a sensitivity analysis is carried out to uncover the most important configuration parameters for aSP-AMG as well as their suitable range of usability. From this analysis, we show that the DPLS prolongation technique is more efficient than ABF. After this, it is verified that the aSP-AMG preconditioner configured with the DPLS prolongation is weakly scalable in the solution of two model problems discretized with linear finite elements and having up to six millions of unknowns. Lastly, the performance of the aSP-AMG is compared to the aFSAI and BoomerAMG preconditioners in the solution of real-world problems showing that aSP-AMG leads

to the faster solution method in most of the cases in terms of both iteration time as well as total execution time.

The next steps of the present research will concern the implementation of other application techniques such as the powerful F-cycle Trottenberg et al. [2001] and K-cycle Notay [2010]. Further, we want to develop new techniques for predicting the smooth vector space aiming to reduce the setup time as well as improving the quality of the test vectors, which play a fundamental role in defining coarse nodes and setup interpolation. Another enhancement could be the introduction of adaptivity also in the coarsening process. Lastly, a major goal will be the efficient implementation of this package on modern massively parallel computers for the solution of very large size problems.

Chapter 6

Conclusions

In this thesis, we presented four SPD preconditioners targeted to the solution of large size linear problems obtained from engineering applications with diffusive character. All of these techniques are based on aFSAI with the main idea of enhancing its weak scalability through multilevel strategies. The two multilevel approaches presented in Chapter 3, i.e., BTFSAI and DDFSAI, showed improvements in terms of iteration count and solution time when compared to aFSAI; however, their applicability was limited to the number of partitions allowed for computing SPD Schur complements. This constraint was eliminated by the MFLR preconditioner presented in Chapter 4 which also introduced low-rank corrections that reduced iteration counts, however at the cost of longer setup time. Lastly, in Chapter 5, we explored a different multilevel framework by proposing an adaptive AMG method that employs aFSAI as a flexible smoother and strategies for dynamically generating the prolongation pattern among other innovative features. We run several sensitivity analysis with challenging test cases to understand the importance of the input configuration parameters of the preconditioners and, consequently, we were able to identify suitable ranges of these parameters giving rise to efficient methods. Also, we solved real-world linear problems arising from the numerical discretization of diverse engineering applications and proved that the preconditioners proposed in this thesis had a better performance, in general, when compared to aFSAI, ILU and other AMG implementations. Special attention is given to the aSP-AMG method which proved to be the most robust and efficient strategy over the other methods proposed herein. Future steps of the present work should focus on such method and aim to improve particularly its C/F splitting selection. Moreover, further enhancements on the computational implementation of aSP-AMG could reduce its setup time

and the development of a distributed parallel code would make possible the solution of problems in order of billions of unknowns.

Appendices

Appendix A

Sparse matrices arising from real-world applications

Here we talk bout the Sparse Matrices arising from Real-World applications.

1. `afshell3`: simulation of sheet metal forming Davis and Hu [2011];
2. `afshell8`: simulation of sheet metal forming Davis and Hu [2011];
3. `emilia923k`: mechanical equilibrium of a 3D gas reservoir with tetrahedral Finite Elements. An irregular unstructured grid was used due to the complexity of the geological formation Ferronato et al. [2010b];
4. `geo1438k`: a 3D geomechanical problem discretizing a region of the earth crust subject to underground deformation. The computational domain is a box with an areal extent of 50 x 50 km and 10 km deep and the disretization is done via regularly shaped tetrahedral Finite Elements Janna et al. [2010];
5. `stocF1465k`: pressure distribution in an three dimensional underground aquifer with stochastic permeabilty field. The computational grid consists of tetrahedral Finite Elements Ferronato et al. [2012a];
6. `pflow742`, arising from a 3D simulation of the pressure-temperature field in a multilayered porous media discretized by Q2 hexahedral FE Janna et al. [2015a];
7. `finger`, a 2D discretization of multiphase flow simulating viscous fingering in porous media Janna et al. [2015b];

8. `spe10`, a finite volume discretization of the single-phase flow simulation in a 3D heterogeneous reservoir with properties defined by the SPE10 dataset Christie et al. [2001];

9. `bump2911`, a 3D geomechanical model of a gas-reservoir discretized by linear tetrahedral FE with mechanical properties of the medium varying in depth Janna et al. [2013];

10. `flan1565`, a 3D mechanical equilibrium problem of a steel flange discetized with linear hexahedral finite elements. The computational grid is a structured mesh with regularly shaped elements Janna et al. [2009a];

11. `abq_powtrain`, a 3D powertrain model, provided by © Dassault Systèmes Simulia Corp., mainly discretized by Q1 hexahedral elements, with non-linear gaskets, penalty frictional contact and pre-tensions;

12. `ecology2`: Landscape ecology problem, using electrical network theory to model animal movement an gene flow. The computational domain is two-dimensional; mesh is formed by 1000x1000 points and the discretization is done via the standard finite difference method (5-points stencil) McRae et al. [2008];

13. `tmt_sym`: a symmetric electromagnetics problem Davis and Hu [2011];

14. `cylinder`: mechanical equilibrium of a cylinder containing a central hole. The medium is loosely clamped. Discretization is performed via linear tetrahedral finite elements by using an in-house numerical code;

15. `fault639`: a 3D faulted gas reservoir discretized with tetrahedral Finite Elements while the faults are discretized with triangular Interface Elements and modeled through a penalty formulation Massimiliano et al. [2007];

16. `hook1498`: a 3D mechanical equilibrium problem of a steel hook discetized with linear tetrahedral finite elements Baggio et al. [2017];

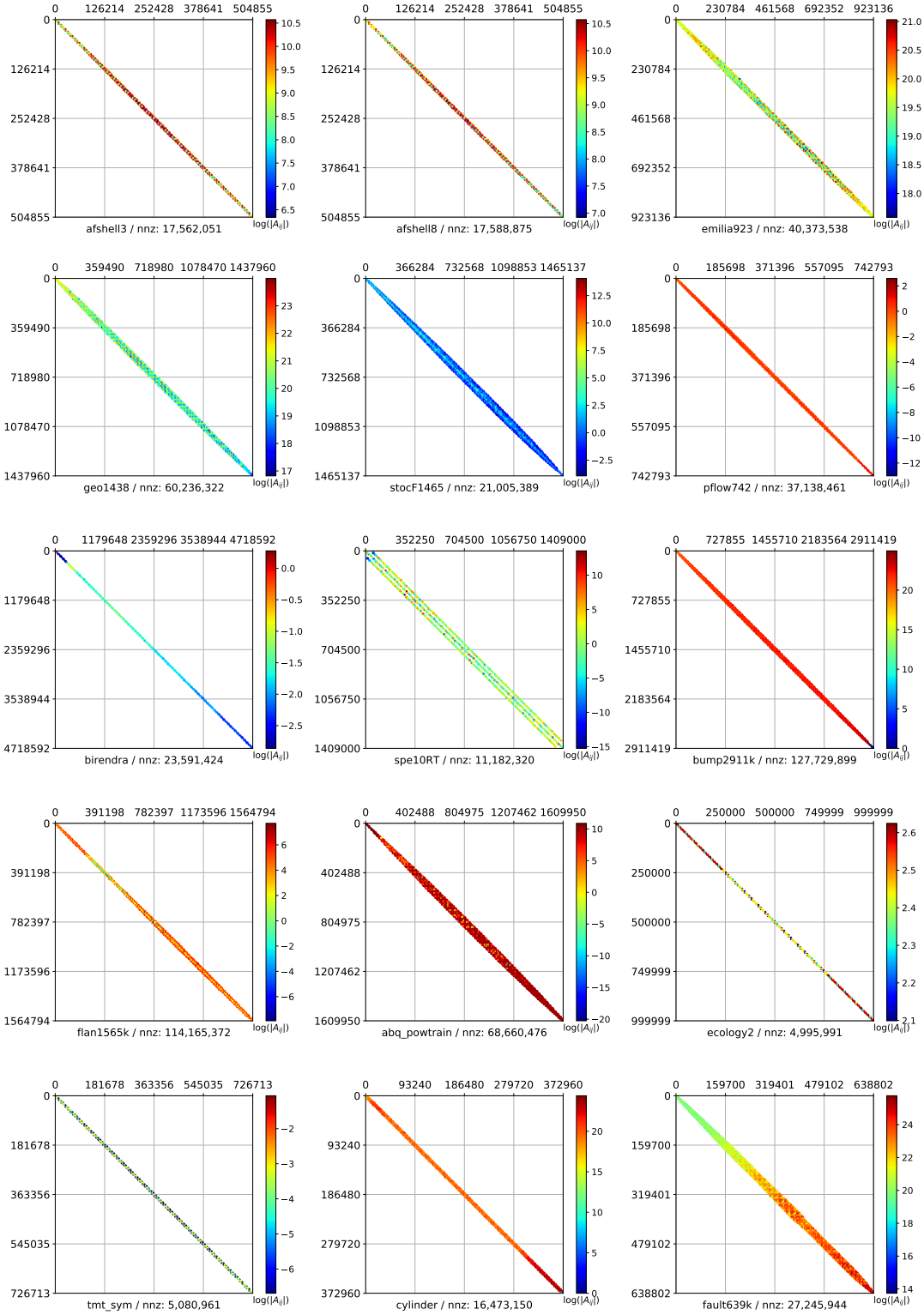

FIGURE A.1: Sparsity pattern of the real-world test matrices from Table A.1. The colors represent the order of magnitude of the nonzero coefficients.

TABLE A.1: Test matrices deriving from real-world problems.

ID	Matrix name	n(A)	nnz(A)	avg. nnzr.	Short Description
1	afshell3	504,855	17,588,875	35	2D/3D structural problem
2	afshell8	504,855	17,579,155	35	2D/3D structural problem
3	emilia923k	923,136	40,373,538	44	3D elasticity - P1 FEM
4	geo1438k	1,437,960	60,236,322	42	3D elasticity - P1 FEM
5	stocF1465k	1,465,137	21,005,389	14	3D poisson - P1 FEM
6	pflow742k	742,793	37,138,461	49	3D poisson - Q2 FEM
7	finger	4,718,592	23,591,424	5	2D poisson - FDM
8	spe10	1,122,005	7,780,175	6	3D poisson - FVM
9	bump2911k	2,911,419	127,729,899	43	3D elasticity - P1 FEM
10	flan1565k	1,564,794	114,165,372	72	3D elasticity - Q1 FEM
11	abq_powtrain	1,609,950	68,660,476	42	3D elasticity - P1 FEM
12	ecology2	999,999	4,995,991	5	2D poisson - FDM
13	tmt-sym	726,713	5,080,961	7	Electromagnetic problem
14	cylinder	372,960	16,473,150	44	3D elasticity - P1 FEM
15	fault639k	638,802	28,614,564	42	3D elasticity - P1/IE FEM
16	hook1498	1,498,023	59,374,451	40	3D elasticity - P1 FEM

Bibliography

Adams, M. Evaluation of three unstructured multigrid methods on 3D finite element problems in solid mechanics. *International Journal for Numerical Methods in Engineering*, 55(5):519–534, 2002. ISSN 1097-0207. URL http://dx.doi.org/10.1002/nme.506.

Adams, M., Brezina, M., Hu, J., and Tuminaro, R. Parallel multigrid smoothing: polynomial versus Gauss–Seidel. *Journal of Computational Physics*, 188(2):593–610, 2003. ISSN 0021-9991. URL http://dx.doi.org/10.1016/S0021-9991(03)00194-3.

Agullo, E., Amestoy, P. R., Buttari, A., Guermouche, A., Joslin, G., L'Excellent, J.-Y., Li, X. S., Napov, A., Rouet, F.-H., Sid-Lakhdar, W. M., Wang, S., Weisbecker, C., and Yamazaki, I. Recent advances in sparse direct solvers. In *SMiRT 2013 - 22nd International Conference on Structural Mechanics in Reactor Technology*, Nuclear Engineering and Design, pages 1–10, Aug 2013. URL https://hal.inria.fr/hal-01060301.

Agullo, E., Giraud, L., Nakov, S., and Roman, J. Hierarchical hybrid sparse linear solver for multicore platforms. Research Report RR-8960, INRIA Bordeaux, Oct 2016. URL https://hal.inria.fr/hal-01379227.

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M., and Wells, G. The FEniCS project version 1.5. *Archive of Numerical Software*, 3(100), 2015. ISSN 2197-8263. doi: 10.11588/ans.2015.100.20553. URL http://journals.ub.uni-heidelberg.de/index.php/ans/article/view/20553.

Amestoy, P., Buttari, A., L'Excellent, J., and Mary, T. On the complexity of the block low-rank multifrontal factorization. *SIAM Journal on Scientific Computing*, 39(4):

A1710–A1740, 2017. doi: 10.1137/16M1077192. URL https://doi.org/10.1137/16M1077192.

Amestoy, P., Ashcraft, C., Boiteau, O., Buttari, A., L'Excellent, J.-Y., and Weisbecker, C. Improving multifrontal methods by means of block low-rank representations. *SIAM Journal on Scientific Computing*, 37(3):A1451–A1474, 2015. URL https://doi.org/10.1137/120903476.

Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Koster, J. MUMPS: A general purpose distributed memory sparse solver. In Sørevik, T., Manne, F., Gebremedhin, A. H., and Moe, R., editors, *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, pages 121–130, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-70734-9. doi: 10.1007/3-540-70734-4_16. URL https://doi.org/10.1007/3-540-70734-4_16.

Aminfar, A., Ambikasaran, S., and Darve, E. A fast block low-rank dense solver with applications to finite-element matrices. *Journal of Computational Physics*, 304: 170 – 188, 2016. ISSN 0021-9991. doi: 10.1016/j.jcp.2015.10.012. URL http://www.sciencedirect.com/science/article/pii/S0021999115006750.

Anzt, H., Chow, E., and Dongarra, J. ParILUT—A New Parallel Threshold ILU Factorization. *SIAM Journal on Scientific Computing*, 40(4):C503–C519, 2018a. doi: 10.1137/16M1079506. URL https://doi.org/10.1137/16M1079506.

Anzt, H., Huckle, T. K., Bräckle, J., and Dongarra, J. Incomplete sparse approximate inverses for parallel preconditioning. *Parallel Computing*, 71(Supplement C):1–22, 2018b. ISSN 0167-8191. doi: 10.1016/j.parco.2017.10.003. URL http://dx.doi.org/10.1016/j.parco.2017.10.003.

Arndt, D., Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Pelteret, J.-P., Turcksin, B., and Wells, D. The `deal.II` library, version 8.5. *Journal of Numerical Mathematics*, 25(3):137–146, 2017. doi: 10.1515/jnma-2016-1045. URL https://doi.org/10.1515/jnma-2016-1045.

Badia, S., Martín, A. F., and Principe, J. Multilevel balancing domain decomposition at extreme scales. *SIAM Journal on Scientific Computing*, 38(1):C22–C52, 2016. URL https://doi.org/10.1137/15M1013511.

Badia, S., Martín, A. F., and Principe, J. FEMPAR: an object-oriented parallel finite element framework. *Archives of Computational Methods in Engineering*, 25(2):195–271, Apr 2018. ISSN 1886-1784. doi: 10.1007/s11831-017-9244-1. URL https://doi.org/10.1007/s11831-017-9244-1.

Baggio, R., Franceschini, A., Spiezia, N., and Janna, C. Rigid body modes deflation of the preconditioned conjugate gradient in the solution of discretized structural problems. *Computers & Structures*, 185:15–26, 2017. URL https://doi.org/10.1016/j.compstruc.2017.03.003.

Baker, A. H., Falgout, R. D., Kolev, T. V., and Yang, U. M. Multigrid smoothers for ultraparallel computing. *SIAM Journal on Scientific Computing*, 33(5):2864–2887, 2011. doi: 10.1137/100798806. URL https://doi.org/10.1137/100798806.

Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. PETSc users manual. Technical Report ANL-95/11 - Revision 3.9, Argonne National Laboratory, 2018. URL http://www.mcs.anl.gov/petsc.

Baliga, B. R. *A control-volume based finite element method for convective heat and mass transfer*. PhD thesis, Minnesota Univ., Minneapolis, December 1978.

Bell, W. N., Olson, L. N., and Schroder, J. B. PyAMG: Algebraic multigrid solvers in Python v3.0, 2015. URL https://github.com/pyamg/pyamg. Release 3.2.

Benzi, M., Cullum, J., and Tuma, M. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 22(4):1318–1332, 2000. doi: 10.1137/S1064827599356900. URL https://doi.org/10.1137/S1064827599356900.

Benzi, M. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418–477, 2002. ISSN 0021-9991. URL http://www.sciencedirect.com/science/article/pii/S0021999102971767.

Benzi, M. and Tůma, M. A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics*, 30(2):305 – 340, 1999. ISSN 0168-9274. doi: 10.1016/S0168-9274(98)00118-4. URL http://www.sciencedirect.com/science/article/pii/S0168927498001184.

Benzi, M. and Tůma, M. A robust incomplete factorization preconditioner for positive definite matrices. *Numerical Linear Algebra with Applications*, 10(5-6):385–400, 2003. ISSN 1099-1506. doi: 10.1002/nla.320. URL http://dx.doi.org/10.1002/nla.320.

Benzi, M., Meyer, C. D., and Tůma, M. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM Journal on Scientific Computing*, 17(5):1135–1149, 1996. URL https://doi.org/10.1137/S1064827594271421.

Benzi, M., Marín, J., and Tůma, M. A two-level parallel preconditioner based on sparse approximate inverses, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.177.

Benzi, M., Kouhia, R., and Tůma, M. Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics. *Computer Methods in Applied Mechanics and Engineering*, 190(49):6533 – 6554, 2001. ISSN 0045-7825. doi: 10.1016/S0045-7825(01)00235-3. URL http://www.sciencedirect.com/science/article/pii/S0045782501002353.

Bergamaschi, L., Ángeles Martínez, and Pini, G. Parallel preconditioned conjugate gradient optimization of the rayleigh quotient for the solution of sparse eigenproblems. *Applied Mathematics and Computation*, 175(2):1694 – 1715, 2006. ISSN 0096-3003. doi: 10.1016/j.amc.2005.09.015. URL http://www.sciencedirect.com/science/article/pii/S0096300305007320.

Bergamaschi, L., Martínez, Á., and Pini, G. Parallel rayleigh quotient optimization with FSAI-Based preconditioning. *Journal of Applied Mathematics*, 2012, Feb 2012. doi: 10.1155/2012/872901. URL http://dx.doi.org/10.1155/2012/872901.

Bernaschi, M., Bisson, M., Fantozzi, C., and Janna, C. A factored sparse approximate inverse preconditioned conjugate gradient solver on graphics processing units. *SIAM Journal on Scientific Computing*, 38(1):C53–C72, jan 2016. URL https://doi.org/10.1137/15M1027826.

Bollhöfer, M., Aliaga, J. I., Martín, A. F., and Quintana Ortí, E. S. *ILUPACK*, pages 917–926. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_513. URL https://doi.org/10.1007/978-0-387-09766-4_513.

Boresi, A. P., Chong, K., and Lee, J. D. *Elasticity in Engineering Mechanics*. Wiley Publishing, 3rd edition, 2010. ISBN 9780470402559, 9780470950005. doi: 10.1002/9780470950005. URL https://doi.org/10.1002/9780470950005.

Boyle, J., Mihajlović, M., and Scott, J. HSL_MI20: An efficient amg preconditioner for finite element problems in 3D. *International Journal for Numerical Methods in Engineering*, 82(1):64–98, 4 2010. ISSN 1097-0207. doi: 10.1002/nme.2758. URL http:https://doi.org/10.1002/nme.2758.

Brandt, A., McCormick, S., and Ruge, J. Algebraic multigrid AMG for sparse matrix equations. *Sparsity and its Applications*, pages 257–284, 1984. URL http://amath.colorado.edu/pub/multigrid/amg1.pdf.

Brandt, A., Brannick, J., Kahl, K., and Livshits, I. Bootstrap AMG. *SIAM Journal on Scientific Computing*, 33(2):612–632, 2011. URL https://doi.org/10.1137/090752973.

Brandt, A. Algebraic multigrid theory: The symmetric case. *Appl. Math. Comput.*, 19(1-4):23–56, jul 1986. ISSN 0096-3003. URL http://dx.doi.org/10.1016/0096-3003(86)90095-0.

Brandt, A. and Ron, D. *Multigrid Solvers and Multilevel Optimization Strategies*, pages 1–69. Springer US, Boston, MA, 2003. ISBN 978-1-4757-3748-6. URL https://doi.org/10.1007/978-1-4757-3748-6_1.

Brandt, A., Brannick, J., Kahl, K., and Livshits, I. Bootstrap algebraic multigrid: Status report, open problems, and outlook. *Numerical Mathematics: Theory, Methods and Applications*, 8(1):112–135, 2015. URL https://doi.org/10.4208/nmtma.2015.w06si.

Brannick, J., Cao, F., Kahl, K., Falgout, R., and Hu, X. Optimal interpolation and compatible relaxation in classical algebraic multigrid. *SIAM Journal on Scientific Computing*, 40(3):A1473–A1493, 2018. doi: 10.1137/17M1123456. URL https://doi.org/10.1137/17M1123456.

Brezina, M., Cleary, A. J., Falgout, R. D., Henson, V. E., Jones, J. E., Manteuffel, T. A., McCormick, S. F., and Ruge, J. W. Algebraic multigrid based on element interpolation (AMGe). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, 2001. URL https://doi.org/10.1137/S1064827598344303.

Brezina, M., Falgout, R., MacLachlan, S., Manteuffel, T., McCormick, S., and Ruge, J. Adaptive smoothed aggregation (αSA) multigrid. *SIAM Review*, 47(2):317–346, 2005. URL https://doi.org/10.1137/050626272.

Brezina, M., Falgout, R., MacLachlan, S., Manteuffel, T., McCormick, S., and Ruge, J. Adaptive algebraic multigrid. *SIAM Journal on Scientific Computing*, 27(4):1261–1286, 2006. URL https://doi.org/10.1137/040614402.

Brezina, M., Ketelsen, C., Manteuffel, T., McCormick, S., Park, M., and Ruge, J. Relaxation-corrected bootstrap algebraic multigrid (rBAMG). *Numerical Linear Algebra with Applications*, 19(2):178–193, 2012. ISSN 1099-1506. URL https://doi.org/10.1002/nla.1821.

Briggs, W., Henson, V., and McCormick, S. *A Multigrid Tutorial, Second Edition*. Society for Industrial and Applied Mathematics, second edition, 2000. URL http://epubs.siam.org/doi/abs/10.1137/1.9780898719505.

Brussino, G. and Sonnad, V. A comparison of direct and preconditioned iterative techniques for sparse, unsymmetric systems of linear equations. *International Journal for Numerical Methods in Engineering*, 28(4):801–815, 1989. doi: 10.1002/nme.1620280406. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620280406.

Bröker, O. and Grote, M. J. Sparse approximate inverse smoothers for geometric and algebraic multigrid. *Applied Numerical Mathematics*, 41(1):61–80, 2002. ISSN 0168-9274. doi: 10.1016/S0168-9274(01)00110-6. URL http://www.sciencedirect.com/science/article/pii/S0168927401001106. Developments and Trends in Iterative Methods for Large Systems of Equations - in memorium Rudiger Weiss.

Bröker, O., Grote, M. J., Mayer, C., and Reusken, A. Robust parallel smoothing for multigrid via sparse approximate inverses. *SIAM Journal on Scientific Computing*, 23 (4):1396–1417, 2001. doi: 10.1137/S1064827500380623. URL https://doi.org/10.1137/S1064827500380623.

Bu, Y., Carpentieri, B., Shen, Z., and Huang, T.-Z. A hybrid recursive multilevel incomplete factorization preconditioner for solving general linear systems. *Appl. Numer. Math.*, 104(C):141–157, jun 2016. ISSN 0168-9274. doi: 10.1016/j.apnum.2015.12.007. URL https://doi.org/10.1016/j.apnum.2015.12.007.

Börm, S., Grasedyck, L., and Hackbusch, W. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements*, 27(5):405 – 422, 2003. ISSN 0955-7997. doi: 10.1016/S0955-7997(02)00152-2. URL http://www.sciencedirect.com/science/article/pii/S0955799702001522. Large scale problems using BEM.

Carpentieri, B., Duff, I. S., Giraud, L., and Sylvand, G. Combining fast multipole techniques and an approximate inverse preconditioner for large electromagnetism calculations. *SIAM Journal on Scientific Computing*, 27(3):774–792, 2005. doi: 10.1137/040603917. URL https://doi.org/10.1137/040603917.

Carpentieri, B., Liao, J., and Sosonkina, M. VBARMS: A variable block algebraic recursive multilevel solver for sparse linear systems. *Journal of Computational*

and Applied Mathematics, 259:164 – 173, 2014. ISSN 0377-0427. doi: 10.1016/j.cam.2013.04.036. URL http://www.sciencedirect.com/science/article/pii/S0377042713002227. Proceedings of the Sixteenth International Congress on Computational and Applied Mathematics (ICCAM-2012), Ghent, Belgium, 9-13 July, 2012.

Carpentieri, B., Liao, J., Sosonkina, M., Bonfiglioli, A., and Baars, S. Using the VBARMS method in parallel computing. *Parallel Computing*, 57:197 – 211, 2016. ISSN 0167-8191. doi: 10.1016/j.parco.2016.01.005. URL http://www.sciencedirect.com/science/article/pii/S0167819116000351.

Chan, T. F. and Mathew, T. P. Domain decomposition algorithms. *Acta Numerica*, 3:61–143, 1 1994. ISSN 1474-0508. URL http://journals.cambridge.org/article_S0962492900002427.

Chartier, T., Falgout, R. D., Henson, V. E., Jones, J., Manteuffel, T., McCormick, S., Ruge, J., and Vassilevski, P. S. Spectral AMGe (ρAMGe). *SIAM Journal on Scientific Computing*, 25(1):1–26, 2003. URL https://doi.org/10.1137/S106482750139892X.

Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14, oct 2008. ISSN 0098-3500. doi: 10.1145/1391989.1391995. URL http://doi.acm.org/10.1145/1391989.1391995.

Chen, Z. On the control volume finite element methods and their applications to multiphase flow. *Networks & Heterogeneous Media*, 1:689, 2006. ISSN 1556-1801. doi: 10.3934/nhm.2006.1.689. URL http://aimsciences.org//article/id/db725959-34e3-4c9c-953a-96b361964eb3.

Chow, E. and Patel, A. Fine-grained parallel incomplete lu factorization. *SIAM Journal on Scientific Computing*, 37(2):C169–C193, 2015. doi: 10.1137/140968896. URL https://doi.org/10.1137/140968896.

Chow, E. and Saad, Y. Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing*, 18(6):1657–1675, 1997a. doi: 10.1137/S1064827595281575. URL https://doi.org/10.1137/S1064827595281575.

Chow, E. Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns. *The International Journal of High Performance Computing Applications*, 15(1):56–74, 2001. doi: 10.1177/109434200101500106. URL https://doi.org/10.1177/109434200101500106.

Chow, E. and Heroux, M. A. An object-oriented framework for block preconditioning. *ACM Trans. Math. Softw.*, 24(2):159–183, jun 1998. ISSN 0098-3500. doi: 10.1145/290200.287639. URL http://doi.acm.org/10.1145/290200.287639.

Chow, E. and Saad, Y. Experimental study of ilu preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86(2):387–414, dec 1997b. ISSN 0377-0427. doi: 10.1016/S0377-0427(97)00171-4. URL http://dx.doi.org/10.1016/S0377-0427(97)00171-4.

Chow, E. and Saad, Y. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19(3):995–1023, 1998. doi: 10.1137/S1064827594270415. URL https://doi.org/10.1137/S1064827594270415.

Chow, E., Anzt, H., Scott, J., and Dongarra, J. Using jacobi iterations and blocking for solving sparse triangular systems in incomplete factorization preconditioning. *Journal of Parallel and Distributed Computing*, 119:219 – 230, 2018. ISSN 0743-7315. doi: 10.1016/j.jpdc.2018.04.017. URL http://www.sciencedirect.com/science/article/pii/S0743731518303034.

Christie, M., Blunt, M., et al. Tenth SPE comparative solution project: A comparison of upscaling techniques. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2001. URL https://doi.org/10.2118/72469-PA.

Cuthill, E. and McKee, J. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM. URL http://doi.acm.org/10.1145/800195.805928.

D'Ambra, P., Filippone, S., and Vassilevski, P. S. BootCMatch: A software package for bootstrap AMG based on graph weighted matching. *ACM Trans. Math. Softw.*, 44(4):39:1–39:25, jun 2018. ISSN 0098-3500. doi: 10.1145/3190647. URL http://doi.acm.org/10.1145/3190647.

Davis, T. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006. doi: 10.1137/1.9780898718881. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718881.

Davis, T. A. Algorithm 832: UMFPACK v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, jun 2004. ISSN 0098-3500. doi: 10.1145/992200.992206. URL http://doi.acm.org/10.1145/992200.992206.

Davis, T. A. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37(3):36:1–36:17, sep 2010. ISSN 0098-3500. doi: 10.1145/1824801.1824814. URL http://doi.acm.org/10.1145/1824801.1824814.

Davis, T. A. and Hu, Y. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, dec 2011. ISSN 0098-3500. URL http://doi.acm.org/10.1145/2049662.2049663.

Davis, T. A., Rajamanickam, S., and Sid-Lakhdar, W. M. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016. doi: 10.1017/S0962492916000076. URL https://doi.org/10.1017/S0962492916000076.

Dietterich, H. R., Lev, E., Chen, J., Richardson, J. A., and Cashman, K. V. Benchmarking computational fluid dynamics models of lava flow simulation for hazard assessment, forecasting, and risk management. *Journal of Applied Volcanology*, 6(1):9, May 2017. ISSN 2191-5040. doi: 10.1186/s13617-017-0061-x. URL https://doi.org/10.1186/s13617-017-0061-x.

Dolean, V., Jolivet, P., and Nataf, F. *An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation*. SIAM, 2015. URL https://hal.inria.fr/cel-01100932v3/document.

Duff, I. S. MA57 — a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw.*, 30(2):118–144, jun 2004. ISSN 0098-3500. doi: 10.1145/992200.992202. URL http://doi.acm.org/10.1145/992200.992202.

Duff, I., Erisman, A., and Reid, J. *Direct Methods for Sparse Matrices*. Numerical Mathematics and Scientific Computation. OUP Oxford, 2017. ISBN

9780192507501. URL https://global.oup.com/academic/product/direct-methods-for-sparse-matrices-9780198508380.

Falgout, R. D. An introduction to algebraic multigrid. *Computing in Science and Engg.*, 8 (6):24–33, November 2006. ISSN 1521-9615. URL http://dx.doi.org/10.1109/MCSE.2006.105.

Falgout, R. D. and Vassilevski, P. S. On generalizing the algebraic multigrid framework. *SIAM Journal on Numerical Analysis*, 42(4):1669–1693, 2004. URL https://doi.org/10.1137/S0036142903429742.

Falgout, R. D. and Yang, U. M. hypre: a library of high performance preconditioners. In *Preconditioners, Lecture Notes in Computer Science*, pages 632–641, 2002. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.1202.

Fan, Q., Forsyth, P., McMacken, J., and Tang, W. Performance issues for iterative solvers in device simulation. *SIAM Journal on Scientific Computing*, 17(1):100–117, 1996. doi: 10.1137/0917009. URL https://doi.org/10.1137/0917009.

Fang, H.-r. and Saad, Y. A filtered lanczos procedure for extreme and interior eigenvalue problems. *SIAM Journal on Scientific Computing*, 34(4):A2220–A2246, 2012. doi: 10.1137/110836535. URL https://doi.org/10.1137/110836535.

Farhat, C., Lesoinne, M., LeTallec, P., Pierson, K., and Rixen, D. FETI-DP: a dual–primal unified FETI method—part i: A faster alternative to the two-level FETI method. *International Journal for Numerical Methods in Engineering*, 50(7):1523–1544, 2001. doi: 10.1002/nme.76. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.76.

Ferronato, M. Preconditioning for sparse linear systems at the dawn of the 21st century: History, current developments, and future perspectives. *ISRN Applied Mathematics*, 2012. URL http://www.hindawi.com/journals/isrn.applied.mathematics/2012/127647/.

Ferronato, M., Castelletto, N., and Gambolati, G. A fully coupled 3-D mixed finite element model of biot consolidation. *Journal of Computational Physics*, 229(12):4813 –

4830, 2010a. ISSN 0021-9991. doi: 10.1016/j.jcp.2010.03.018. URL https://doi.org/10.1016/j.jcp.2010.03.018.

Ferronato, M., Gambolati, G., Janna, C., and Teatini, P. Geomechanical issues of anthropogenic CO2 sequestration in exploited gas fields. *Energy Conversion and Management*, 51(10):1918 – 1928, 2010b. ISSN 0196-8904. doi: 10.1016/j.enconman.2010.02.024. URL http://www.sciencedirect.com/science/article/pii/S0196890410000907.

Ferronato, M., Janna, C., and Pini, G. Shifted FSAI preconditioners for the efficient parallel solution of non-linear groundwater flow models. *International Journal for Numerical Methods in Engineering*, 89(13):1707–1719, 2012a. doi: 10.1002/nme.3309. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.3309.

Ferronato, M., Janna, C., and Pini, G. Efficient parallel solution to large-size sparse eigenproblems with block fsai preconditioning. *Numerical Linear Algebra with Applications*, 19(5):797–815, 2012b. doi: 10.1002/nla.813. URL https://doi.org/10.1002/nla.813.

Fialko, S. Iterative methods for solving large-scale problems of structural mechanics using multi-core computers. *Archives of Civil and Mechanical Engineering*, 14(1):190 – 203, 2014. ISSN 1644-9665. doi: 10.1016/j.acme.2013.05.009. URL http://www.sciencedirect.com/science/article/pii/S1644966513000666.

Franceschini, A., Paludetto Magri, V. A., Ferronato, M., and Janna, C. A robust multilevel approximate inverse preconditioner for symmetric positive definite matrices. *SIAM Journal on Matrix Analysis and Applications*, 39:123–147, jan 2018. doi: 10.1137/16M1109503. URL https://doi.org/10.1137/16M1109503.

Frank, J. and Vuik, C. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23(2):442–462, 2001. doi: 10.1137/S1064827500373231. URL https://doi.org/10.1137/S1064827500373231.

Gee, M., Siefert, C., Hu, J., Tuminaro, R., and Sala, M. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006. URL https://trilinos.org/packages/ml/.

Gerritsen, M. G. and Durlofsky, L. J. Modeling fluid flow in oil reservoirs. *Annual Review of Fluid Mechanics*, 37(1):211–238, 2005. doi: 10.1146/annurev.fluid.37.061903.175748. URL https://doi.org/10.1146/annurev.fluid.37.061903.175748.

Geuzaine, C. and Remacle, J.-F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009. ISSN 1097-0207. URL http://dx.doi.org/10.1002/nme.2579.

Ghysels, P., Kłosiewicz, P., and Vanroose, W. Improving the arithmetic intensity of multigrid with the help of polynomial smoothers. *Numerical Linear Algebra with Applications*, 19(2):253–267, 2012. doi: 10.1002/nla.1808. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.1808.

Ghysels, P., Li, X. S., Rouet, F.-H., Williams, S., and Napov, A. An efficient multicore implementation of a novel hss-structured multifrontal solver using randomized sampling. *SIAM Journal on Scientific Computing*, 38, 01 2016. doi: 10.1137/15M1010117. URL https://dx.doi.org/10.1137/15M1010117.

Giraud, L., Haidar, A., and Watson, L. T. Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Comput.*, 34(6-8):363–379, jul 2008. ISSN 0167-8191. doi: 10.1016/j.parco.2008.01.006. URL http://dx.doi.org/10.1016/j.parco.2008.01.006.

Goldfeld, P., Pavarino, L. F., and Widlund, O. B. Balancing neumann-neumann preconditioners for mixed approximations of heterogeneous problems in linear elasticity. *Numerische Mathematik*, 95(2):283–324, Aug 2003. ISSN 0945-3245. doi: 10.1007/s00211-002-0450-9. URL https://doi.org/10.1007/s00211-002-0450-9.

Gopalakrishnan, S., Chakraborty, A., and Mahapatra, D. R. *Spectral Finite Element Method*. Springer-Verlag Berlin Heidelberg, 1 edition, 2008. doi: 10.1007/978-1-84628-356-7. URL https://www.springer.com/la/book/9781846283550.

Gould, N. and Scott, J. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM Journal on Scientific Computing*, 19(2):605–625, 1998. doi: 10.1137/S1064827595288425. URL https://doi.org/10.1137/S1064827595288425.

Grossmann, C., Roos, H.-G., and Stynes, M. *Numerical Treatment of Partial Differential Equations*. Springer-Verlag Berlin Heidelberg, 1st edition, 2007. ISBN 978-3-540-71584-9. doi: 10.1007/978-3-540-71584-9. URL https://www.springer.com/book/9783540715825.

Grote, M. J. and Huckle, T. Parallel preconditioning with sparse approximate inverses. *SIAM Journal on Scientific Computing*, 18(3):838–853, 1997. URL https://doi.org/10.1137/S1064827594276552.

Gupta, A. Enhancing performance and robustness of ILU preconditioners by blocking and selective transposition. *SIAM Journal on Scientific Computing*, 39(1):A303–A332, 2017. doi: 10.1137/15M1053256. URL https://doi.org/10.1137/15M1053256.

Gupta, A., Koric, S., and George, T. Sparse matrix factorization on massively parallel computers. Technical Report RC24809, IBM Research Division, 2009. URL http://domino.research.ibm.com/library/cyberdig.nsf/papers/794A7318E4757DD6852575E1005A2C13.

Hackbusch, W. A sparse matrix arithmetic based on H-matrices. Part I: Introduction to H-matrices. *Computing*, 62(2):89–108, Apr 1999. ISSN 1436-5057. doi: 10.1007/s006070050015. URL https://doi.org/10.1007/s006070050015.

Hackbusch, W. *Hierarchical Matrices: Algorithms and Analysis*. Springer-Verlag Berlin Heidelberg, 1 edition, 2015. ISBN 978-3-662-47323-8. doi: 10.1007/978-3-662-47324-5. URL https://link.springer.com/book/10.1007%2F978-3-662-47324-5.

Hecht, F. New development in FreeFem++. *Journal of Numerical Mathematics*, 20(3-4):251–265, 2012. ISSN 1570-2820. URL http://hal.upmc.fr/hal-01476313/document.

Henson, V. E. and Yang, U. M. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155 – 177, 2002. ISSN 0168-9274. URL http://www.sciencedirect.com/science/article/pii/S0168927401001155. Developments and Trends in Iterative Methods for Large Systems of Equations - in memorium Rudiger Weiss.

Henson, V. E. and Vassilevski, P. S. Element-free AMGe: General algorithms for computing interpolation weights in AMG. *SIAM Journal on Scientific Computing*, 23(2): 629–650, 2001. URL https://doi.org/10.1137/S1064827500372997.

Heroux, M. A. AztecOO User's Guide. Technical Report SAND2004-3796, Sandia National Labs, 2004.

Hestenes, M. R. and Stiefel, E. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952. doi: 10.6028/jres.049.044. URL http://dx.doi.org/10.6028/jres.049.044.

Horgue, P., Soulaine, C., Franc, J., Guibert, R., and Debenest, G. An open-source toolbox for multiphase flow in porous media. *Computer Physics Communications*, 187:217 – 226, 2015. ISSN 0010-4655. doi: 10.1016/j.cpc.2014.10.005. URL https://doi.org/10.1016/j.cpc.2014.10.005.

Huckle, T. Factorized sparse approximate inverses for preconditioning. *The Journal of Supercomputing*, 25:109–117, may 2003. URL https://doi.org/10.1023/A:1023988426844.

Huckle, T. and Sedlacek, M. *SPAI (SParse Approximate Inverse)*, pages 1867–1870. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_144. URL https://doi.org/10.1007/978-0-387-09766-4_144.

Huckle, T., Kallischko, A., Roy, A., Sedlacek, M., and Weinzierl, T. An efficient parallel implementation of the mspai preconditioner. *Parallel Computing*, 36(5-6):273–284, jun 2010. doi: 10.1016/j.parco.2009.12.007. URL https://doi.org/10.1016/j.parco.2009.12.007.

Hénon, P., Ramet, P., and Roman, J. PaStiX: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Computing*, 28(2): 301 – 321, 2002. ISSN 0167-8191. doi: https://doi.org/10.1016/S0167-8191(01)00141-7. URL http://www.sciencedirect.com/science/article/pii/S0167819101001417.

Janna, C. and Ferronato, M. Adaptive pattern research for block FSAI preconditioning. *SIAM Journal on Scientific Computing*, 33(6):3357–3380, 2011. URL https://doi.org/10.1137/100810368.

Janna, C., Comerlati, A., and Gambolati, G. A comparison of projective and direct solvers for finite elements in elastostatics. *Advances in Engineering Software*, 40(8):675 – 685, 2009a. ISSN 0965-9978. doi: 10.1016/j.advengsoft.2008.11.010. URL http://www.sciencedirect.com/science/article/pii/S0965997808001968.

Janna, C., Ferronato, M., and Gambolati, G. Multi-level incomplete factorizations for the iterative solution of non-linear finite element problems. *International Journal for Numerical Methods in Engineering*, 80(5):651–670, 2009b. doi: 10.1002/nme.2664. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2664.

Janna, C., Ferronato, M., and Gambolati, G. A block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems. *SIAM Journal on Scientific Computing*, 32(5):2468–2484, 2010. doi: 10.1137/090779760. URL https://doi.org/10.1137/090779760.

Janna, C., Ferronato, M., and Gambolati, G. Enhanced block FSAI preconditioning using domain decomposition techniques. *SIAM Journal on Scientific Computing*, 35 (5):S229–S249, 2013. doi: 10.1137/120880860. URL https://doi.org/10.1137/120880860.

Janna, C., Ferronato, M., and Gambolati, G. The use of supernodes in factored sparse approximate inverse preconditioning. *SIAM Journal on Scientific Computing*, 37:C72–C94, 2015a. URL https://doi.org/10.1137/140956026.

Janna, C., Ferronato, M., Sartoretto, F., and Gambolati, G. FSAIPACK: A software package for high-performance factored sparse approximate inverse preconditioning.

ACM Trans. Math. Softw., 41(2):10:1–10:26, February 2015b. ISSN 0098-3500. URL http://doi.acm.org/10.1145/2629475.

Jenny, P., Lee, S. H., and Tchelepi, H. A. Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. *J. Comput. Phys.*, 187(1):47–67, may 2003. ISSN 0021-9991. doi: 10.1016/S0021-9991(03)00075-5. URL http://dx.doi.org/10.1016/S0021-9991(03)00075-5.

Jolivet, P., Hecht, F., Nataf, F., and Prud'homme, C. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 80:1–80:11, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2378-9. doi: 10.1145/2503210.2503212. URL http://doi.acm.org/10.1145/2503210.2503212.

Jonsthovel, T., van Gijzen, M. B., Vuik, C., and Scarpas, A. On the use of rigid body modes in the deflated preconditioned conjugate gradient method. *SIAM Journal on Scientific Computing*, 35(1):B207–B225, 2013. URL https://doi.org/10.1137/100803651.

Kaporin, I. E. A preconditioned conjugate-gradient method for solving discrete analogs of differential problems. *Differential Equations*, 26(7):897–906, 1990.

Kaporin, I. E. New convergence results and preconditioning strategies for the conjugate gradient method. *Numerical Linear Algebra with Applications*, 1(2):179–210, 1994. ISSN 1099-1506. URL http://dx.doi.org/10.1002/nla.1680010208.

Karypis, G. and Kumar, V. Multilevel K-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1):96–129, jan 1998. ISSN 0743-7315. URL http://dx.doi.org/10.1006/jpdc.1997.1404.

Kirk, B. S., Peterson, J. W., Stogner, R. H., and Carey, G. F. `libMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations. *Engineering with Computers*, 22(3–4):237–254, 2006. doi: 10.1007/s00366-006-0049-3. URL https://doi.org/10.1007/s00366-006-0049-3.

Klawonn, A., Widlund, O., and Dryja, M. Dual-primal feti methods for three-dimensional elliptic problems with heterogeneous coefficients. *SIAM Journal on Numerical Analysis*, 40(1):159–179, 2002. doi: 10.1137/S0036142901388081. URL https://doi.org/10.1137/S0036142901388081.

Kolev, T. and Dobrev, V. MFEM: Modular Finite Element Methods library, 6 2010. URL https://github.com/mfem/mfem.

Kolotilina, L. Y. and Yeremin, A. Y. Factorized sparse approximate inverse preconditionings I. theory. *SIAM Journal on Matrix Analysis and Applications*, 14(1):45–58, 1993. doi: 10.1137/0614004. URL https://doi.org/10.1137/0614004.

Kolotilina, L. Y. and Yeremin, A. Y. Factorized sparse approximate inverse ppreconditioning II: Solution of 3D FE systems on massively parallel computers. *International Journal of High Speed Computing*, 07(02):191–215, 1995. doi: 10.1142/S0129053395000117. URL https://doi.org/10.1142/S0129053395000117.

Kolotilina, L. Y., Nikishin, A. A., and Yeremin, A. Y. Factorized sparse approximate inverse preconditionings. iv: Simple approaches to rising efficiency. *Numerical Linear Algebra with Applications*, 6(7):515–531, 1999. doi: 10.1002/(SICI)1099-1506(199910/11)6:7<515::AID-NLA176>3.0.CO;2-0. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291099-1506%28199910/11%296%3A7%3C515%3A%3AAID-NLA176%3E3.0.CO%3B2-0.

Kumpel, H.-J. Theory of linear poroelasticity - with applications to geomechanics and hydrogeology. *Geophysical Journal International*, 150(3):828–829, 2002. doi: 10.1046/j.1365-246X.2002.01757.x. URL https://doi.org/10.1046/j.1365-246X.2002.01757.x.

Lanczos, C. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards*, 49(1):33–53, 1952. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.130.3118.

Larsson, S. and Thomee, V. *Partial Differential Equations with Numerical Methods*. Springer-Verlag Berlin Heidelberg, 1 edition, 2003. doi: 10.1007/978-3-540-88706-5. URL https://www.springer.com/book/9783540017721.

LeVeque, R. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics Classics in Applied Mathemathics)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007. ISBN 0898716292, 9780898716290. URL https://dl.acm.org/citation.cfm?id=1355322.

LeVeque, R. J. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. doi: 10.1017/CBO9780511791253. URL https://doi.org/10.1017/CBO9780511791253.

Li, N., Saad, Y., and Chow, E. Crout versions of ilu for general sparse matrices. *SIAM Journal on Scientific Computing*, 25(2):716–728, 2003. doi: 10.1137/S1064827502405094. URL https://doi.org/10.1137/S1064827502405094.

Li, R. and Saad, Y. Divide and conquer low-rank preconditioners for symmetric matrices. *SIAM Journal on Scientific Computing*, 35(4):A2069–A2095, 2013a. doi: 10.1137/120872735. URL https://doi.org/10.1137/120872735.

Li, R. and Saad, Y. Gpu-accelerated preconditioned iterative linear solvers. *J. Supercomput.*, 63(2):443–466, feb 2013b. ISSN 0920-8542. doi: 10.1007/s11227-012-0825-3. URL http://dx.doi.org/10.1007/s11227-012-0825-3.

Li, R., Xi, Y., and Saad, Y. Schur complement-based domain decomposition preconditioners with low-rank corrections. *Numerical Lin. Alg. with Applic.*, 23(4):706–729, 2016. doi: 10.1002/nla.2051. URL https://doi.org/10.1002/nla.2051.

Li, X. S. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw.*, 31(3):302–325, sep 2005. ISSN 0098-3500. doi: 10.1145/1089014.1089017. URL http://doi.acm.org/10.1145/1089014.1089017.

Lin, C.-J. and Moré, J. J. Incomplete cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, 21(1):24–45, 1999. URL https://doi.org/10.1137/S1064827597327334.

Livne, O. E. and Brandt, A. Lean algebraic multigrid (LAMG): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, 34(4):B499–B522, 2012. URL https://doi.org/10.1137/110843563.

Longsine, D. and McCormick, S. Simultaneous rayleigh-quotient minimization methods. *Linear Algebra and its Applications*, 34:195 – 234, 1980. ISSN 0024-3795. doi: 10.1016/0024-3795(80)90166-4. URL http://www.sciencedirect.com/science/article/pii/0024379580901664.

MacLachlan, S., Osei-Kuffuor, D., and Saad, Y. Modification and compensation strategies for threshold-based incomplete factorizations. *SIAM J. Sci. Comput.*, 34(1):48–75, jan 2012. ISSN 1064-8275. doi: 10.1137/110834986. URL http://dx.doi.org/10.1137/110834986.

Madenci, E. and Guven, I. *The Finite Element Method and Applications in Engineering Using ANSYS ®*. Springer International Publishing, second edition, 2015. ISBN 978-1-4899-7549-2. doi: 10.1007/978-1-4899-7550-8. URL https://www.springer.com/book/9781489975492.

Maliska, C. *Transferencia de calor e mecanica dos fluidos computacional*. LTC, 2 edition. ISBN 9788521610465. URL https://books.google.it/books?id=osq-nAEACAAJ.

Manteuffel, T. A., Olson, L. N., Schroder, J. B., and Southworth, B. S. A root-node based algebraic multigrid method. *SIAM Journal on Scientific Computing*, 39(5):S723–S756, 2017. URL https://doi.org/10.1137/16M1082706.

Mary, T. *Block low-rank multifrontal solvers: complexity, performance and scalability*. PhD thesis, Université de Toulouse 3 Paul Sabatier, 2017. URL http://mumps.enseeiht.fr/doc/Thesis_TheoMary.pdf.

Massimiliano, F., Giuseppe, G., Carlo, J., and Pietro, T. Numerical modelling of regional faults in land subsidence prediction above gas/oil reservoirs. *International Journal for Numerical and Analytical Methods in Geomechanics*, 32(6):633–657, 2007. doi: 10.1002/nag.640. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nag.640.

McCormick, J. W., S. F.; Ruge. Multigrid methods for variational problems. *SIAM Journal on Numerical Analysis*, 19:924–929, 10 1982. URL https://doi.org/10.1137/0719067.

McRae, B. H., Dickson, B. G., Keitt, T. H., and Shah, V. B. Using circuit theory to model connectivity in ecology, evolution, and conservation. *Ecology*, 89(10):2712–2724, 2008. doi: 10.1890/07-1861.1. URL https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1890/07-1861.1.

Miah, M. I., Elhaj, M. A., Ahmed, S., and Hossain, M. E. Modeling of temperature distribution and oil displacement during thermal recovery in porous media: A critical review. *Fuel*, 226:423 – 440, 2018. ISSN 0016-2361. doi: 10.1016/j.fuel.2018.04.018. URL http://www.sciencedirect.com/science/article/pii/S0016236118306410.

Nakov, S. *On the Design of Sparse Hybrid Linear Solvers for Modern Parallel Architectures*. PhD thesis, Université de Bourdaix, 2015. URL https://tel.archives-ouvertes.fr/tel-01304315/file/NAKOV_STOJCE_2015.pdf.

Naumov, M., Arsaev, M., Castonguay, P., Cohen, J., Demouth, J., Eaton, J., Layton, S., Markovskiy, N., Reguly, I., Sakharnykh, N., Sellappan, V., and Strzodka, R. AmgX: A library for GPU accelerated algebraic multigrid and preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 37(5):S602–S626, 2015. doi: 10.1137/140980260. URL https://doi.org/10.1137/140980260.

Nordbotten, J. M. and Celia, M. A. *Geological Storage of CO2: Modeling Approaches for Large-Scale Simulation*. Wiley Publishing, 1st edition, 2011. ISBN 9780470889466, 9781118137086. doi: 10.1002/9781118137086. URL https://doi.org/10.1002/9781118137086.

Notay, Y. An aggregation-based algebraic multigrid method. *Electronic transactions on numerical analysis*, 37(6):123–146, 2010. URL http://etna.mcs.kent.edu/volumes/2001-2010/vol37/abstract.php?vol=37&pages=123-146.

Notay, Y. Aggregation-based algebraic multigrid for convection-diffusion equations. *SIAM Journal on Scientific Computing*, 34(4):A2288–A2316, 2012. URL https://doi.org/10.1137/110835347.

Olson, L. N., Schroder, J. B., and Tuminaro, R. S. A general interpolation strategy for algebraic multigrid using energy minimization. *SIAM Journal on Scientific Computing*, 33(2):966–991, 2011. URL https://doi.org/10.1137/100803031.

Paige, C. Computational variants of the lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972. doi: 10.1093/imamat/10.3.373. URL http://dx.doi.org/10.1093/imamat/10.3.373.

Paige, C. and Saunders, M. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975. doi: 10.1137/0712047. URL https://doi.org/10.1137/0712047.

Paige, C. Accuracy and effectiveness of the lanczos algorithm for the symmetric eigenproblem. *Linear Algebra and its Applications*, 34:235 – 258, 1980. ISSN 0024-3795. doi: 10.1016/0024-3795(80)90167-6. URL http://www.sciencedirect.com/science/article/pii/0024379580901676.

Paludetto Magri, V. A., Franceschini, A., Ferronato, M., and Janna, C. Multilevel approaches for fsai preconditioning. *Numerical Linear Algebra with Applications*, 0(0): e2183, 2018. doi: 10.1002/nla.2183. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.2183. e2183 nla.2183.

Paludetto Magri, V. A., Franceschini, A., , and Janna, C. A novel AMG approach based on adaptive smoothing and prolongation for ill-conditioned systems. *SIAM Journal on Scientific Computing*, 2019. submitted.

Patankar, S. V. *Numerical heat transfer and fluid flow*. Series on Computational Methods in Mechanics and Thermal Science. Hemisphere Publishing Corporation (CRC

Press, Taylor & Francis Group), 1980. ISBN 978-0891165224. URL http://www.crcpress.com/product/isbn/9780891165224.

Patzák, B. OOFEM - an object-oriented simulation tool for advanced modeling of materials and structures. *Acta Polytechnica*, 52(6):59–66, 2012. URL https://ojs.cvut.cz/ojs/index.php/ap/article/view/1678.

Prokopenko, A., Hu, J. J., Wiesner, T. A., Siefert, C. M., and Tuminaro, R. S. MueLu user's guide 1.0. Technical Report SAND2014-18874, Sandia National Labs, 2014. URL http://trilinos.org/packages/muelu.

Prokopenko, A., Siefert, C. M., Hu, J. J., Hoemmen, M., and Klinvex, A. Ifpack2 User's Guide 1.0. Technical Report SAND2016-5338, Sandia National Labs, 2016.

Prud'Homme, C., Metivet, T., Chabannes, V., and Dollé, G. Feel++: Finite Element Embedded Language and Library in C++, 2018. URL https://github.com/feelpp/feelpp.

Puzyrev, V., Koric, S., and Wilkin, S. Evaluation of parallel direct sparse linear solvers in electromagnetic geophysical problems. *Computers & Geosciences*, 89:79 – 87, 2016. ISSN 0098-3004. doi: 10.1016/j.cageo.2016.01.009. URL http://www.sciencedirect.com/science/article/pii/S0098300416300164.

Raghavan, P. and Teranishi, K. Parallel hybrid preconditioning: Incomplete factorization with selective sparse approximate inversion. *SIAM Journal on Scientific Computing*, 32(3):1323–1345, 2010. doi: 10.1137/080739987. URL https://doi.org/10.1137/080739987.

Rajamanickam, S., Boman, E. G., and Heroux, M. A. ShyLU: A hybrid-hybrid solver for multicore platforms. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium*, pages 631–643, May 2012. doi: 10.1109/IPDPS.2012.64. URL https://ieeexplore.ieee.org/document/6267865/.

Renard, Y. and Pommier, J. Getfem++. an open source generic c++ library for finite element methods, 2002. URL http://home.gna.org/getfem.

Richardson, L. F. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 210:307–357, 1911. ISSN 02643952. URL http://www.jstor.org/stable/90994.

Rouet, F.-H., Li, X. S., Ghysels, P., and Napov, A. A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. *ACM Trans. Math. Softw.*, 42(4):27:1–27:35, jun 2016. ISSN 0098-3500. doi: 10.1145/2930660. URL http://doi.acm.org/10.1145/2930660.

Saad, Y. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, second edition, 2003. doi: 10.1137/1.9780898718003. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

Saad, Y. and Schultz, M. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986. doi: 10.1137/0907058. URL https://doi.org/10.1137/0907058.

Saad, Y. and Sosonkina, M. pARMS: A package for solving general sparse linear systems on parallel computers. In *Parallel Processing and Applied Mathematics*, pages 446–457, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-48086-0. doi: 10.1007/3-540-48086-2_49. URL https://doi.org/10.1007/3-540-48086-2_49.

Saad, Y. and Suchomel, B. ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9(5):359–378, 2002. doi: 10.1002/nla.279. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.279.

Saad, Y. ILUT: a Dual Threshold Incomplete LU Factorization. *Numerical Linear Algebra with Applications*, 1(4):387–402, 1994. ISSN 1099-1506. URL http://dx.doi.org/10.1002/nla.1680010405.

Schenk, O., Gärtner, K., Fichtner, W., and Stricker, A. PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation. *Future Gener. Comput. Syst.*, 18(1):69–78, sep 2001. ISSN 0167-739X. doi: 10.1016/S0167-739X(00)00076-5. URL http://dx.doi.org/10.1016/S0167-739X(00)00076-5.

Schwarz, H. A. II. Ueber einen Grenzübergang durch alternirendes Verfahren. Wolf J. XV. 272-286. 1870 (1870)., 1870. URL https://zbmath.org/?format=complete&q=an:02.0214.02.

Scott, J. and Tůma, M. On signed incomplete cholesky factorization preconditioners for saddle-point systems. *SIAM Journal on Scientific Computing*, 36(6):A2984–A3010, 2014a. doi: 10.1137/140956671. URL https://doi.org/10.1137/140956671.

Scott, J. and Tůma, M. HSL_MI28: An efficient and robust limited-memory incomplete cholesky factorization code. *ACM Trans. Math. Softw.*, 40(4):24:1–24:19, jul 2014. ISSN 0098-3500. doi: 10.1145/2617555. URL http://doi.acm.org/10.1145/2617555.

Scott, J. and Tůma, M. On positive semidefinite modification schemes for incomplete cholesky factorization. *SIAM Journal on Scientific Computing*, 36(2):A609–A633, 2014b. doi: 10.1137/130917582. URL https://doi.org/10.1137/130917582.

Sheng, Z., Dewilde, P., and Chandrasekaran, S. *Algorithms to Solve Hierarchically Semi-separable Systems*, pages 255–294. Birkhäuser Basel, Basel, 2007. ISBN 978-3-7643-8137-0. doi: 10.1007/978-3-7643-8137-0_5. URL https://doi.org/10.1007/978-3-7643-8137-0_5".

Smith, B. F., Bjørstad, P. E., and Gropp, W. D. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, NY, USA, 1996. ISBN 0-521-49589-X.

Sourbier, F., Operto, S., Virieux, J., Amestoy, P., and L'Excellent, J.-Y. FWT2D: A massively parallel program for frequency-domain full-waveform tomography of wide-aperture seismic data—part 2: Numerical examples and scalability analysis. *Computers & Geosciences*, 35(3):496 – 514, 2009. ISSN 0098-3004. doi: 10.

1016/j.cageo.2008.04.012. URL http://www.sciencedirect.com/science/article/pii/S0098300408002677.

Sousedík, B., Sístek, J., and Mandel, J. Adaptive-multilevel bddc and its parallel implementation. *Computing*, 95(12):1087–1119, Dec 2013. ISSN 1436-5057. doi: 10.1007/s00607-013-0293-5. URL https://doi.org/10.1007/s00607-013-0293-5.

Strikwerda, J. *Finite Difference Schemes and Partial Differential Equations, Second Edition*. Society for Industrial and Applied Mathematics, 2nd edition, 2004. doi: 10.1137/1.9780898717938. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898717938.

Stüben, K. Algebraic multigrid (AMG): Experiences and comparisons. *Appl. Math. Comput.*, 13(3-4):419–451, jan 1983. ISSN 0096-3003. URL http://dx.doi.org/10.1016/0096-3003(83)90023-1.

Stüben, K. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics*, 128(1):281 – 309, 2001. ISSN 0377-0427. URL http://www.sciencedirect.com/science/article/pii/S0377042700005161. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

Tang, W.-P. and Wan, W. L. Sparse approximate inverse smoother for multigrid. *SIAM Journal on Matrix Analysis and Applications*, 21(4):1236–1252, 2000. doi: 10.1137/S0895479899339342. URL https://doi.org/10.1137/S0895479899339342.

Teatini, P., Ferronato, M., Gambolati, G., Bertoni, W., and Gonella, M. A century of land subsidence in ravenna, italy. *Environmental Geology*, 47(6):831–846, Apr 2005. ISSN 1432-0495. doi: 10.1007/s00254-004-1215-9. URL https://doi.org/10.1007/s00254-004-1215-9.

Thomée, V. From finite differences to finite elements: A short history of numerical analysis of partial differential equations. *Journal of Computational and Applied Mathematics*, 128(1):1 – 54, 2001. ISSN 0377-0427. doi: 10.1016/S0377-0427(00)00507-0. URL http://www.sciencedirect.com/science/article/pii/S0377042700005070. Numerical Analysis 2000. Vol. VII: Partial Differential Equations.

Toselli, A. and Widlund, O. B. *Domain Decomposition Methods — Algorithms and Theory*. Elsevier, 1 edition, 2005. doi: 10.1007/b137868. URL https://doi.org/10.1007/b137868.

Trottenberg, U., Oosterlee, C., and Schüller, A. *Multigrid*. Academic Press, 2001. ISBN 9780127010700. URL https://www.elsevier.com/books/multigrid/trottenberg/978-0-08-047956-9.

Tuminaro, R., Perego, M., Tezaur, I., Salinger, A., and Price, S. A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling. *SIAM Journal on Scientific Computing*, 38(5):C504–C532, 2016. doi: 10.1137/15M1040839. URL https://doi.org/10.1137/15M1040839.

Vaněk, P. Acceleration of convergence of a two-level algorithm by smoothing transfer operator. *Applications of Mathematics*, 37:265–274, 1992. URL https://eudml.org/doc/15715.

Vaněk, P., Mandel, J., and Brezina, M. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, Sep 1996. ISSN 1436-5057. URL https://doi.org/10.1007/BF02238511.

Vassilevski, P. S. *Multilevel Block Factorization Preconditioners*. Springer International Publishing, 1st edition, 2008. ISBN 978-0-387-71563-6. doi: 10.1007/978-0-387-71564-3. URL https://doi.org/10.1007/978-0-387-71564-3.

Vogel, F., Göktepe, S., Steinmann, P., and Kuhl, E. Modeling and simulation of viscous electro-active polymers. *European Journal of Mechanics - A/Solids*, 48:112 – 128, 2014. ISSN 0997-7538. doi: 10.1016/j.euromechsol.2014.02.001. URL http://www.sciencedirect.com/science/article/pii/S099775381400014X. Frontiers in Finite-Deformation Electromechanics.

Wang, S., Li, X. S., Xia, J., Situ, Y., and de Hoop, M. V. Efficient scalable algorithms for solving dense linear systems with hierarchically semiseparable structures. *SIAM Journal on Scientific Computing*, 35(6):C519–C544, 2013. doi: 10.1137/110848062. URL https://doi.org/10.1137/110848062.

Wang, S., Sturler, E. d., and Paulino, G. H. Large-scale topology optimization using preconditioned krylov subspace methods with recycling. *International Journal for Numerical Methods in Engineering*, 69(12):2441–2468, 2007. doi: 10.1002/nme.1798. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1798.

Wathen, A. J. Preconditioning. *Acta Numerica*, 24:329–376, 2015. doi: 10.1017/S0962492915000021. URL https://doi.org/10.1017/S0962492915000021.

Xi, Y., Li, R., and Saad, Y. An algebraic multilevel preconditioner with low-rank corrections for sparse symmetric matrices. *SIAM Journal on Matrix Analysis and Applications*, 37(1):235–259, 2016. URL http://dx.doi.org/10.1137/15M1021830.

Xia, J. On the complexity of some hierarchical structured matrix algorithms. *SIAM Journal on Matrix Analysis and Applications*, 33(2):388–410, 2012. doi: 10.1137/110827788. URL https://doi.org/10.1137/110827788.

Xia, J. Randomized sparse direct solvers. *SIAM Journal on Matrix Analysis and Applications*, 34(1):197–227, 2013a. doi: 10.1137/12087116X. URL https://doi.org/10.1137/12087116X.

Xia, J. Efficient structured multifrontal factorization for general large sparse matrices. *SIAM Journal on Scientific Computing*, 35(2):A832–A860, 2013b. doi: 10.1137/120867032. URL https://doi.org/10.1137/120867032.

Xia, J., Chandrasekaran, S., Gu, M., and Li, X. S. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications*, 17(6):953–976, 2010. doi: 10.1002/nla.691. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.691.

Xu, J. and Zikatanov, L. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017. URL http://dx.doi.org/10.1017/S0962492917000083.

Yang, U. M. *Parallel Algebraic Multigrid Methods — High Performance Preconditioners*, pages 209–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-31619-0. URL https://doi.org/10.1007/3-540-31619-1_6.

Yeremin, A. Y., Kolotilina, L. Y., and Nikishin, A. A. Factorized sparse approximate inverse preconditionings. III. iterative construction of preconditioners. *Journal of Mathematical Sciences*, 101(4):3237–3254, Sep 2000. ISSN 1573-8795. doi: 10.1007/BF02672769. URL https://doi.org/10.1007/BF02672769.

Zampini, S. PCBDDC: A class of robust dual-primal methods in PETSc. *SIAM Journal on Scientific Computing*, 38(5):S282–S306, 2016. URL https://doi.org/10.1137/15M1025785.

Zampini, S. and Keyes, D. E. On the robustness and prospects of adaptive bddc methods for finite element discretizations of elliptic pdes with high-contrast coefficients. In *Proceedings of the Platform for Advanced Scientific Computing Conference*, PASC '16, pages 6:1–6:13, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4126-4. doi: 10.1145/2929908.2929919. URL http://doi.acm.org/10.1145/2929908.2929919.

Zienkiewicz, O., Taylor, R., and Zhu, J. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, 7 edition, 2013. doi: 10.1016/C2009-0-24909-9. URL https://doi.org/10.1016/C2009-0-24909-9.

Zienkiewicz, O., Taylor, R., and Fox, D. Chapter 1 - general problems in solid mechanics and nonlinearity. In Zienkiewicz, O., Taylor, R., and Fox, D., editors, *The Finite Element Method for Solid and Structural Mechanics (Seventh Edition)*, pages 1 – 20. Butterworth-Heinemann, Oxford, seventh edition edition, 2014. ISBN 978-1-85617-634-7. doi: 10.1016/B978-1-85617-634-7.00001-6. URL https://doi.org/10.1016/B978-1-85617-634-7.00001-6.

Zoback, M. D. *Reservoir Geomechanics*. Cambridge University Press, 2007. doi: 10.1017/CBO9780511586477. URL https://doi.org/10.1017/CBO9780511586477.