PADOVA UNIVERSITY PRESS

Dolomites Research Notes on Approximation

Special issue of the "Seminari Padovani di Analisi Numerica 2018" (SPAN2018), Volume 11 · 2018 · Pages 11–22

Recent advancements in preconditioning techniques for large size linear systems suited for High Performance Computing

Andrea Franceschini[a] · Massimiliano Ferronato[a] · Carlo Janna[a] · Victor A. P. Magri[a]

Communicated by Ángeles Martínez

Abstract

The numerical simulations of real-world engineering problems create models with several millions or even billions of degrees of freedom. Most of these simulations are centered on the solution of systems of non-linear equations, that, once linearized, become a sequence of linear systems, whose solution is often the most time-demanding task. Thus, in order to increase the capability of modeling larger cases, it is of paramount importance to exploit the resources of High Performance Computing architectures. In this framework, the development of new algorithms to accelerate the solution of linear systems for many-core architectures is a really active research field. Our main focus is algebraic preconditioning and, among the various options, we elect to develop approximate inverses for symmetric and positive definite (SPD) linear systems [22], both as stand-alone preconditioner or smoother for AMG techniques. This choice is mainly supported by the almost perfect parallelism that intrinsically characterizes these algorithms. As basic kernel, the Factorized Sparse Approximate Inverse (FSAI) developed in its adaptive form by Janna and Ferronato [18] is selected. Recent developments are i) a robust multilevel approach for SPD problems based on FSAI preconditioning, which eliminates the chance of algorithmic breakdowns independently of the preconditioner sparsity [14] and ii) a novel AMG approach featuring the adaptive FSAI method as a flexible smoother as well as new approaches to adaptively compute the prolongation operator. In this latter work, a new technique to build the prolongation is also presented.

1 Introduction

The current engineering simulation models are quickly growing up to millions or even billions of unknowns, and the efficient solution to the related sparse linear systems of equations

$$\mathbf{A}x = b, \tag{1}$$

with $\mathbf{A} \in \mathbb{R}^{n\times n}$ a symmetric and positive definite (SPD) matrix, $b \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$, is often one of the most expensive tasks in several numerical applications. The linear system (1) can be solved by direct or iterative methods, but the last one is often the choice for large-scale engineering problems due to their lower complexity and higher scalability. It is known that an iterative method alone is not able to solve the linear system: a suitable preconditioner, imitating the action of $\mathbf{A}^{-1}$, is needed. Several techniques for building preconditioners are available in the literature with the distinct classes: purely algebraic or physics-based ones. In the first class, there are incomplete factorizations [30, 1], sparse approximate inverses [2, 17, 21], domain decomposition techniques [10, 23] and algebraic multigrid methods (AMG) [26, 35]. All these approaches build the preconditioners based on the matrix coefficients only.

Compared to incomplete factorizations, sparse approximate inverses are generally more robust and appropriate for parallel computational architectures. In particular, the adaptive pattern factorized sparse approximate inverse (aFSAI) [18, 21] is an algebraic preconditioner for SPD problems that proves effective in a wide range of applications. One of the most attractive features of aFSAI for modern computers is its intrinsic high degree of parallelism, indeed each row of the preconditioner can be fully formed independently of the others.

In this work, we review two ideas: i) the first one is based on the introduction of some sequentiality in the aFSAI computation, developing a multilevel preconditioner; ii) the latter one is a novel AMG method, based on the adaptivity concept and suited for high performance computing. The first idea is to use the information extracted from the earlier set-up stages for the remaining rows. This concept has been already introduced in the field of approximate inverses in [8, 29], and more recently in [27], where both a block tridiagonal and a domain decomposition approach have been used to improve the FSAI performance. Here, we develop a more general multilevel framework. The second approach is an AMG method: aSP-AMG [28], where the acronym aSP stands for *adaptive Smoothing and Prolongation*. In particular, the smoother is represented by the aFSAI and the prolongation operator construction is based on a least squares minimization variant, with a dynamic pattern selection scheme. Finally, following the adaptive and bootstrap AMG techniques [5, 6, 4], we assume no information about the near-null space of $\mathbf{A}$. Instead, we

[a]University of Padova

construct an approximation of this space by testing an initial set of candidates, both given as input (if available) or randomly generated, without any use of self-improvement ideas.

2 aFSAI preconditioner

The FSAI preconditioner $\mathbf{M}^{-1}$ of a symmetric positive definite matrix $\mathbf{A}$ is given by:

$$\mathbf{M}^{-1} = \mathbf{G}^T\mathbf{G} \approx \mathbf{A}^{-1}, \tag{2}$$

where the factor $\mathbf{G}$ is calculated by minimizing the Frobenius norm

$$||\mathbf{I} - \mathbf{GL}||_F \tag{3}$$

over the set of matrices with a given lower triangular nonzero pattern S. In Equation (3), the matrix $\mathbf{L}$ represents the exact lower Cholesky factor of $\mathbf{A}$, that is not required for the calculation of $\mathbf{G}$. Developing the minimization, i.e., deriving the equation with respect to the entries f_{ij}, we have:

$$[\mathbf{GA}]_{ij} = \begin{cases} 0, & \text{if } i \neq j \\ l_{ii}, & \text{if } i = j. \end{cases} \tag{4}$$

Introducing $\widetilde{\mathbf{G}} = \mathbf{D}^{-1}\mathbf{G}$, where $\mathbf{D} = [diag(\mathbf{G})]^{-1/2}$, it is possible to write:

$$\left[\widetilde{\mathbf{G}}\mathbf{A}\right]_{ij} = \delta_{ij}, \tag{5}$$

which is used to calculate the factor $\widetilde{\mathbf{G}}$ and finally the matrix $\mathbf{G}$.

The essential factor that determinates the performance of the FSAI preconditioner is the selection of the sparsity pattern. In the literature, a lot of strategies are available, e.g., using low powers of $\mathbf{A}$, but one of the most promising, based on the row-wise minimization of the Kaporin conditioning number, was developed by [18] (aFSAI). In this strategy, the construction of $\mathbf{G}$ is controlled by the following parameters:

- k_{max}: maximum number of steps for adding new entries.
- ρ_G: number of entries added to the sparsity pattern in each step.
- ϵ_G: stopping tolerance based on the relative reduction of the Kaporin number.

The main component of this algorithm is the gradient of the Kaporin number, computed row-wise, from which the indices of the ρ_G largest entries are added to the tentative pattern. For each row, a small dense linear system, obtained from gathering the components of the full matrix, is solved and the result is used to build the corresponding row of $\mathbf{G}$. This iterative process is repeated up to a k_{max} number of times or until the relative Kaporin number variation is below the input threshold ϵ_G. For more details about this algorithm, the reader is referred to the paper by [21].

3 Multilevel FSAI preconditioning

In this section, the multilevel preconditioner is presented with some considerations on its computation and eigenvalues. Moreover, we also show how to improve its quality with two different low-rank updates. This section is a review of the contents presented in [14].

3.1 Standard multilevel approach

The standard multilevel approach applied to the SPD matrix $\mathbf{A}$, with a total number n_l of levels, is based on a sequence of matrices $\mathbf{A}_l$, where each of them is the Schur complement at any levels $l \in [0, n_l - 1]$ of the previous level. Following for instance [19], we can partition A_l in four blocks and perform the factorization:

$$\mathbf{A}_l = \begin{bmatrix} \mathbf{K} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_K & \mathbf{0} \\ \mathbf{B}^T\mathbf{L}_K^{-T} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{L}_K^T & \mathbf{L}_K^{-1}\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{6}$$

where $\mathbf{L}_K$ is the exact Cholesky factorization of $\mathbf{K} \in \mathbb{R}^{n_1 \times n_1}$, and $\mathbf{S} = \mathbf{C} - \mathbf{B}^T\mathbf{K}^{-1}\mathbf{B}$, $\mathbf{S} \in \mathbb{R}^{n_2 \times n_2}$, is the Schur complement of $\mathbf{A}_l$ with respect to the partition (n_1, n_2), i.e., $\mathbf{A}_{l+1} \equiv \mathbf{S}$. As usual, being our goal to build a preconditioner, each block at the right-hand side of (6) is replaced with an approximation:

$$\widetilde{\mathbf{L}} \simeq \mathbf{L}_K, \qquad \widetilde{\mathbf{H}} \simeq \mathbf{L}_K^{-1}\mathbf{B}, \qquad \widetilde{\mathbf{S}} \simeq \mathbf{S} \tag{7}$$

From the recursive computation of (6), using the approximations (7), we have a general framework for a multilevel preconditioner $\mathbf{M}$ of $\mathbf{A}$, made by the list of factors $\mathbf{M}_l$, $l \in [0, n_l]$.

Due to the recursive nature of the $\mathbf{M}$ computation, we can restrict our analysis to the two-level case with no loss of generality. Hence, $\mathbf{A}_l$ in Equation (6) coincides with $\mathbf{A}$ and $n_1 + n_2 = n$. A widely used choice for the approximations in (7) is an incomplete factorization, i.e. setting $\widetilde{\mathbf{L}}$ equal to the Incomplete Cholesky (IC) factor of $\mathbf{K}$ as in [32, 24, 19, 7]. In the same framework, the use of aFSAI as the main kernel is straightforward. After computing $\mathbf{G}$ such that $\mathbf{G}^T\mathbf{G} \simeq \mathbf{K}$, the approximations in (7) become:

$$\widetilde{\mathbf{L}} \simeq \mathbf{G}^{-1}, \qquad \widetilde{\mathbf{H}} \simeq \mathbf{GB}, \qquad \widetilde{\mathbf{S}} \simeq \mathbf{C} - \widetilde{\mathbf{H}}^T\widetilde{\mathbf{H}} \tag{8}$$

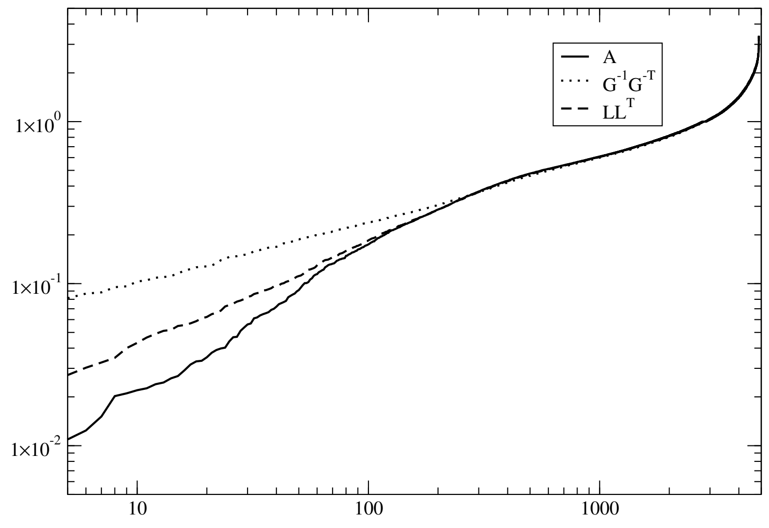

Figure 1: Comparison between the eigenspectra of $\mathbf{A}$, $\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^T$ and $(\mathbf{G}^T\mathbf{G})^{-1}$ for the `bcsstk16` matrix from the SuiteSparse Matrix Collection.

An interesting property related to the use of an aFSAI instead of an incomplete factorization is that no dropping is necessary to compute efficiently $\widetilde{\mathbf{H}}$ and $\widetilde{\mathbf{S}}$, because $\mathbf{G}$ is usually very sparse.

This straightforward implementation of the MF preconditioner is very prone to breakdowns. We observe that the Schur complement approximation $\widetilde{\mathbf{S}} = \mathbf{C} - \widetilde{\mathbf{H}}^T\widetilde{\mathbf{H}}$ is computed as the difference between two SPD matrices and can be indefinite. The reason for this indefiniteness resides in the poor approximation of the leftmost eigenvalues usually obtained by aFSAI. Figure 1 compares the eigenspectra of the `bcsstk16` matrix (structural problem, with 4,884 rows and 290,378 nonzero entries) from the SuiteSparse Matrix Collection [9] with its approximations by IC and aFSAI, $\widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^T$ and $(\mathbf{G}^T\mathbf{G})^{-1}$, respectively. $\widetilde{\mathbf{L}}$ and $\mathbf{G}$ are computed in such a way the number of nonzeros is approximately the same. From Figure 1, we have that both approximations are not able to capture the smallest eigenvalues of the matrix, though IC is better, as generally happens. The smallest eigenvalues of $\mathbf{K}$ are the largest of $\mathbf{K}^{-1}$ and control the most significant entries of $\mathbf{B}^T\mathbf{K}^{-1}\mathbf{B}$. Thus, $\widetilde{\mathbf{H}}^T\widetilde{\mathbf{H}}$ computed as in (8) often fails in approximating accurately the largest entries of $\mathbf{B}^T\mathbf{K}^{-1}\mathbf{B}$, leading to the indefiniteness.

Our aim is to describe a robust version of the MF preconditioner. This can be ensured by computing $\widetilde{\mathbf{S}}$ using aFSAI approximations only. To this aim, we can use the following result (Theorem 2.1 of [14]):

Theorem 3.1. *Let* $\mathbf{A} \in \mathbb{R}^{n\times n}$ *be an SPD* (2×2)*-block matrix:*

$$\mathbf{A} = \begin{bmatrix} \mathbf{K} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{bmatrix}, \tag{9}$$

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

$$\mathbf{V} = \begin{bmatrix} \mathbf{F}^T \\ \mathbf{I} \end{bmatrix}, \qquad \mathbf{D} = \begin{bmatrix} \mathbf{0} & \mathbf{Z} \end{bmatrix}, \tag{10}$$

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

For the proof, we refer the reader to the original work of [14]. The most important consequence of the previous theorem is that the Schur complement approximation $\widetilde{\mathbf{S}}$, obtained substituting $\mathbf{V}$ with an approximation $\widetilde{\mathbf{V}}$, is always SPD.

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

$$\mathbf{M}_0^{-1} = \mathbf{P}_b\mathbf{P}_a. \tag{11}$$

An explicit approximation of $\mathbf{K}^{-1}$ is computed as $\mathbf{G}^T\mathbf{G}$ using the aFSAI procedure [21] and introduced in $\mathbf{P}_a$:

$$\mathbf{P}_a = \begin{bmatrix} \mathbf{G} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{12}$$

Then, the first stage preconditioned matrix $\mathbf{P}_a\mathbf{A}\mathbf{P}_a^T$ is computed:

$$\mathbf{P}_a\mathbf{A}\mathbf{P}_a^T = \begin{bmatrix} \mathbf{G}\mathbf{K}\mathbf{G}^T & \mathbf{G}\mathbf{B} \\ \mathbf{B}^T\mathbf{G}^T & \mathbf{C} \end{bmatrix} \tag{13}$$

and the adaptive Block FSAI [18] of $\mathbf{P}_a\mathbf{A}\mathbf{P}_a^T$ is computed for the second factor:

$$\mathbf{P}_b = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{F} & \mathbf{I} \end{bmatrix}. \tag{14}$$

The zero-level preconditioned matrix $\mathbf{M}_0^{-1}\mathbf{A}\mathbf{M}_0^{-T}$ reads:

$$\mathbf{M}_0^{-1}\mathbf{A}\mathbf{M}_0^{-T} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{F} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{G}\mathbf{K}\mathbf{G}^T & \mathbf{G}\mathbf{B} \\ \mathbf{B}^T\mathbf{G}^T & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{F}^{\mathrm{T}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{G}\mathbf{K}\mathbf{G}^T & -\mathbf{R}_F^T \\ -\mathbf{R}_F & \widetilde{\mathbf{S}} \end{bmatrix} \tag{15}$$

where $\mathbf{R}_F = -\mathbf{F}(\mathbf{G}\mathbf{K}\mathbf{G}^T) - \mathbf{B}^T\mathbf{G}^T$ is the residual on $\mathbf{F}$, i.e., $\mathbf{R}_F$ approaches the null matrix as the accuracy in the computation on $\mathbf{F}$ increases. Finally, the (2,2) block of $\mathbf{M}_0^{-1}\mathbf{A}\mathbf{M}_0^{-T}$ is the approximation of the first-level Schur complement:

$$\widetilde{\mathbf{S}} = \mathbf{C} + \mathbf{F}\mathbf{G}\mathbf{B} + \mathbf{B}^T\mathbf{G}^T\mathbf{F}^T + \mathbf{F}\mathbf{G}\mathbf{K}\mathbf{G}^T\mathbf{F}^T \tag{16}$$

that becomes the new matrix for the next level. As $\widetilde{\mathbf{S}}$ in (16) is SPD for any $\mathbf{F}$ and $\mathbf{G}$, no breakdown is possible.

The MF preconditioner construction is controlled by three user-specified parameters:

- n_l: number of levels. Therefore, $(n_l - 1)$ is the number of computed Schur complements;
- ϵ_G: tolerance for the adaptive computation of $\mathbf{G}$, see [21];
- ϵ_F: tolerance for the adaptive computation of $\mathbf{F}$, see [18];

3.2 Improving the MF performance with Low-Rank corrections

In [34] is introduced the idea of using low-rank corrections in a multilevel framework, giving rise to the robust Multilevel Schur complement-based Low-Rank (MSLR) preconditioner. The basic concept is as follows. Define the matrix:

$$\mathbf{Y} = \mathbf{L}_C^{-1}\mathbf{B}^T\mathbf{K}^{-1}\mathbf{B}\mathbf{L}_C^{-T} = \mathbf{L}_C^{-1}(\mathbf{C}-\mathbf{S})\mathbf{L}_C^{-T} \tag{17}$$

where $\mathbf{L}_C$ is the exact lower factor of $\mathbf{C}$. The eigenvalues σ_i of $\mathbf{Y}$ are bounded:

$$0 \le \sigma_{n_2} \le \dots \le \sigma_1 < 1 \tag{18}$$

The separation of the eigenvalues θ_i of $\mathbf{X} = \mathbf{L}_C^T(\mathbf{S}^{-1} - \mathbf{C}^{-1})\mathbf{L}_C$ is larger than that of $\mathbf{Y}$, because:

$$\begin{gathered} \theta_i = \frac{\sigma_i}{1-\sigma_i} \qquad i = 1, \dots, n_2 \\ \theta_i - \theta_{i+1} = \frac{\sigma_i - \sigma_{i+1}}{(1-\sigma_i)(1-\sigma_{i+1})} \qquad i = 1, \dots, n_2 - 1 \end{gathered} \tag{19}$$

As shown in [34], the separation of the eigenvalues of $\mathbf{L}_C^{-T}\mathbf{X}\mathbf{L}_C^{-1} = \mathbf{S}^{-1} - \mathbf{C}^{-1}$ has a stronger impact on the performance of the MSLR preconditioner, however studying $\mathbf{X}$ is easier. Equation (19) suggests that approximating with a low-rank matrix $(\mathbf{S}^{-1} - \mathbf{C}^{-1})$ is easier than $(\mathbf{S} - \mathbf{C})$ because of the faster eigenvalue decay. An improved version of $\mathbf{S}^{-1}$ can be computed as:

$$\mathbf{S}^{-1} \simeq \mathbf{C}^{-1} + \mathbf{W}_k\Theta_k\mathbf{W}_k^T \tag{20}$$

with $\mathbf{W}_k\Theta_k\mathbf{W}_k^T$ a rank-k approximation of $\mathbf{L}_C^{-T}\mathbf{X}\mathbf{L}_C^{-1}$, obtained from the eigendecomposition of $\mathbf{Y}$. Retaining the k largest eigenvalues and corresponding eigenvectors of $\mathbf{Y}$ we can write:

$$\mathbf{Y} \simeq \mathbf{U}_k\Sigma_k\mathbf{U}_k^T \tag{21}$$

Noting that:

$$\mathbf{S}^{-1} - \mathbf{C}^{-1} = \mathbf{L}_C^{-T}[(\mathbf{I}-\mathbf{Y})^{-1} - \mathbf{I}]\mathbf{L}_C^{-1} = \mathbf{L}_C^{-T}[\mathbf{Y}(\mathbf{I}-\mathbf{Y})^{-1}]\mathbf{L}_C^{-1} \tag{22}$$

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

$$\Theta_k = \Sigma_k(\mathbf{I} - \Sigma_k)^{-1} \tag{23}$$

and $\mathbf{W}_k = \mathbf{L}_C^{-T}\mathbf{U}_k$.

In our framework, we have an already computed Schur complement and we want to improve the action of $\widetilde{\mathbf{S}}^{-1}$. Due to the positive definiteness of $\widetilde{\mathbf{S}}^{-1}$, the eigenvalues σ_i of the matrix:

$$\overline{\mathbf{Y}} = \mathbf{L}_{\widetilde{S}}^{-1}(\widetilde{\mathbf{S}} - \mathbf{S})\mathbf{L}_{\widetilde{S}}^{-T} \tag{24}$$

satisfy the condition (18). Thus, with the same procedure outlined above, we can compute the k largest eigenpairs of $\overline{\mathbf{Y}}$:

$$\overline{\mathbf{Y}} \simeq \overline{\mathbf{U}}_k\overline{\Sigma}_k\overline{\mathbf{U}}_k^T \tag{25}$$

and get the expression of the corrected Schur complement inverse:

$$\mathbf{S}^{-1} \simeq \widetilde{\mathbf{S}}^{-1} + \overline{\mathbf{W}}_k\overline{\Theta}_k\overline{\mathbf{W}}_k^T \tag{26}$$

where $\overline{\Theta}_k = \overline{\Sigma}_k(\mathbf{I} - \overline{\Sigma}_k)^{-1}$ and $\overline{\mathbf{W}} = \mathbf{L}_{\widetilde{S}}^{-T}\overline{\mathbf{U}}$.

Since we are working in a multilevel framework, $\widetilde{\mathbf{S}}^{-1}$ will not be used exactly. Instead a new approximation $\widehat{\mathbf{S}}^{-1} \simeq \widetilde{\mathbf{S}}^{-1}$ will be computed. Thus, $\widehat{\mathbf{S}}$ will be the new target of the low-rank correction. Moreover, the same idea can be used to improve the approximation $\mathbf{G}$ used in the first stage, i.e. $\mathbf{P}_a$. Therefore, we use two low-rank correction techniques for the preconditioner set-up:

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

3.2.1 Descending Low-Rank corrections

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

$$\overline{\mathbf{Y}} = \mathbf{G}[(\mathbf{G}^T\mathbf{G})^{-1} - \mathbf{K}]\mathbf{G}^T = \mathbf{I} - \mathbf{G}\mathbf{K}\mathbf{G}^T \tag{27}$$

and compute its rank-k approximation:

$$\overline{\mathbf{Y}} \simeq \overline{\mathbf{U}}_k \overline{\Sigma}_k \overline{\mathbf{U}}_k^T \tag{28}$$

Now the computation of $\overline{\mathbf{U}}_k$ and $\overline{\Sigma}_k$ is less expensive than in (25), because all components are explicitly known. The enhanced preconditioner for **K** reads:

$$\mathbf{K}^{-1} \simeq \mathbf{G}^T\mathbf{G} + \overline{\mathbf{W}}_k \overline{\Theta}_k \overline{\mathbf{W}}_k^T \tag{29}$$

Due to the positive definiteness of $\mathbf{G}\mathbf{K}\mathbf{G}^T$, the eigenvalues of $\overline{\mathbf{Y}}$ are bounded from above by 1, but a lower bound is missing. This is not a problem, as our approach is based on the computation of the eigenvalues $\overline{\sigma}_i$ of $\overline{\mathbf{Y}}$ closest to 1. To obtain a split preconditioner, a symmetrically split operator is needed. Further details are provided in [14].

3.2.2 Ascending Low-Rank corrections

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

$$\widehat{\mathbf{Y}} = \mathbf{I} - \widehat{\mathbf{G}}\widetilde{\mathbf{S}}\widehat{\mathbf{G}}^T \simeq \widehat{\mathbf{U}}_k \widehat{\Sigma}_k \widehat{\mathbf{U}}_k^T \tag{30}$$

where $\widehat{\mathbf{G}}$ is the lower inverse factor of $\widehat{\mathbf{S}}$, i.e. $(\widehat{\mathbf{G}}^T\widehat{\mathbf{G}})^{-1} = \widehat{\mathbf{S}}$, which is explicitly available by the approximation of lower levels. Because the explicit expression of every matrix is known, the computation of (30) is quite cheap. The new approximation to $\widetilde{\mathbf{S}}$ is given by:

$$\widetilde{\mathbf{S}}^{-1} \simeq \widehat{\mathbf{G}}^T\widehat{\mathbf{G}} + \widehat{\mathbf{W}}_k \widehat{\Theta}_k \widehat{\mathbf{W}}_k^T \tag{31}$$

where $\widehat{\mathbf{W}}_k = \widehat{\mathbf{G}}^T\widehat{\mathbf{U}}_k$ and $\widehat{\Theta}_k = \widehat{\Sigma}_k(\mathbf{I} - \widehat{\Sigma}_k)^{-1}$. Notice that, during the set-up, we use a split update because it is operatively necessary to compute $\widehat{\mathbf{G}}\widetilde{\mathbf{S}}\widehat{\mathbf{G}}^T$, however, during the preconditioner application, the use of Equation (31) is more efficient as it only requires one update.

The low-rank correction in the MFLR preconditioner depends on two parameters:

- dlr: Descending Low-Rank correction size, i.e., number of eigenpairs used to enrich **G**. This is a local improvement;
- alr: Ascending Low-Rank correction size, i.e., number of eigenpairs used to enrich $\widehat{\mathbf{S}}^{-1}$. This is a global improvement, being the Schur complement size increasing up to the size of the zero-level partition.

4 aSP-AMG: adaptive Smoothing and Prolongation MultiGrid

Standard components of all AMG methods are a multilevel hierarchy, interpolation operators, the use of smoothers and a complementary between coarse-grid correction and relaxation. In aSP-AMG, to define coarse nodes, we adopt the classical AMG approach, i.e., coarse variables are a subset of the fine level ones.

One of the main components for an AMG method is the smoother, which usually is a stationary iterative method whose task is to eliminate the error components associated with larger eigenvalues of **A**, i.e., high-frequency errors. In many AMG methods, the smoother is a pointwise relaxation method such as (block) Jacobi or Gauss-Seidel. The second one is often preferred, because of its effectiveness, but its parallel implementation is not straightforward. In aSP-AMG, we use the aFSAI [21] as a smoother.

To build an effective AMG, the coarse grid must be able to correct errors that the smoother is not able to deal with. Defining Ω as the index set $1, 2, ..., n$, with n the size of **A**, we need to find two disjoint sets $\mathcal{F}$ and $\mathcal{C}$ representing the fine and coarse nodes, respectively, naming with $n_f = |\mathcal{F}|$ and $n_c = |\mathcal{C}|$ their size.

Numerous of the developed coarsening algorithms [36] are based on the classical concept of strength of connection (SoC). This measure of the influences exerted between two nodes plays a fundamental role in the coarsening phase. However, it relies on the assumption that **A** is an M-matrix, which is far to be true for general discretizations. We employ another definition of SoC based on the concept of *affinity* recently introduced by [25]. The affinity-based SoC is more general but requires the availability of a suitable test space representing the algebraically smooth vectors. In some problems, the test space is known a priori, e.g., the rigid body modes for the linear elasticity. Our current implementation can take advantage of an already existing kernel or estimating it through a Lanczos process. Let us call **X** the $n \times n_t$ matrix whose columns form a basis of the test space Φ and let us denote as x_i^T the i-th row of X. Using the same notation of [35], the connection strength between two nodes i and j is:

$$s_c(i,j) = \frac{(x_i^T x_j)^2}{(x_i^T x_i)(x_j^T x_j)}. \tag{32}$$

The SoC matrix $\mathbf{S_c}$ is built on the same pattern of **A**, with entries defined by (32), then it is filtered by dropping weak connections. Coarse nodes are finally chosen by finding a maximum independent set (MIS) of nodes on the filtered adjacency graph. The sparsity of the SoC matrix is controlled specifying the integer parameter:

- θ: the average number of connections per node.

Once the nodes belonging to the fine level have been subdivided into the two sets $\mathcal{F}$ and $\mathcal{C}$, it is possible to set-up the prolongation operator who is responsible for transferring information from the coarse to the fine space. Using the conventional F/C ordering (i.e., first fine nodes, then coarse nodes), the prolongation operator P will be written as:

$$\mathbf{P} = \begin{pmatrix} \mathbf{W} \\ \mathbf{I} \end{pmatrix}, \tag{33}$$

where $\mathbf{W}$ is a $n_f \times n_c$ matrix containing the weights for coarse-to-fine variable interpolation. Finally, as we are dealing with an SPD matrix, a Galerkin approach in defining the restriction operator is used, so $\mathbf{R}$ is the transpose of $\mathbf{P}$, with the coarse level matrix $\mathbf{A_c}$ given merely by the triple matrix product:

$$\mathbf{A_c} = \mathbf{P}^T\mathbf{AP}. \tag{34}$$

It is straightforward to extend this two-level approach to a more efficient multilevel version by recursion.

4.1 Test space generation

As long as we base our multigrid method on the affinity measure (32), a key ingredient is an accurate representation of the smooth vectors, i.e., a suitable test space Φ. As almost all single level preconditioners, aFSAI can accurately represent the higher part of the spectrum of an SPD matrix while the approximation of lower eigenpairs is quite rough.

When aFSAI is used, the smoother operator takes the following form:

$$\mathbf{S} = \mathbf{I} - \mathbf{G}^T\mathbf{GA}. \tag{35}$$

Since the spectral radius of $\mathbf{G}^T\mathbf{GA}$ may be bigger than one, a relaxation factor ω is introduced to render the smoother convergent:

$$\mathbf{S}_\omega = \mathbf{I} - \omega\mathbf{G}^T\mathbf{GA}, \tag{36}$$

where:

$$\omega < \frac{2}{\lambda_{\max}(\mathbf{G}^T\mathbf{GA})}. \tag{37}$$

This factor does not affect the lowest frequencies and the correction of the error components along the smooth vectors must be committed to the coarse grid operator. We aim to find a good approximation of the eigenvectors of $\mathbf{S}$ paired to the eigenvalues closer to 1, i.e., a consistent estimate of the near-null space of $\mathbf{A}$. A simple plain Lanczos algorithm [31] is used to extract these eigenpairs.

Being the Lanczos iterative method designed for symmetric matrices, we need to recast our problem as follows:

$$\overline{\mathbf{S}} = \mathbf{I} - \mathbf{GAG}^T, \tag{38}$$

where the matrix $\overline{\mathbf{S}}$ is similar to $\mathbf{S}$.

It is well known that the Lanczos algorithm converges rapidly to the extreme eigenvalues, thus, in this case, these eigenvalues are close to 1 on the rightmost part of the spectrum and to $1-\lambda_{\max}(\mathbf{GAG}^T)$ on the other side. Both these extremes are necessary, as the former one represents the smooth vector space Φ and the latter is used to compute ω, according to (37), to transform the given smoother in a converging one, i.e. $\mathbf{S}_\omega$.

The following user-specified parameters control the test space generation:

- n_t: the dimension of the test space;
- ϵ_t: the relative accuracy in estimating eigenpairs, that is (λ, v) is added to the test space if it satisfies $\|\overline{\mathbf{S}}v - \lambda v\| \le \epsilon_t \|v\|$.

The steps to build the test space are collected in Algorithm 1.

From a practical viewpoint, aSP-AMG needs an approximation of the test space. Lanczos algorithm does not provide the whole extreme parts of the spectrum. We use anyway the n_t rightmost converged eigenvectors for the test space and the leftmost converged eigenvalue to compute ω. The number of Lanczos iterations is limited between 5 and 6 times the desired test space dimension n_t, with the accuracy ϵ_t usually ranging between 10^{-3} and 10^{-2}.

4.2 Prolongation operators

Once the coarse nodes are selected, we can move to the computation of the prolongation operator $\mathbf{P}$. Recalling that the complementarity between smoothing and coarse-grid correction is essential for achieving an efficient multigrid method, it is crucial to calculate a prolongation operator that represents accurately slow converging modes of the error. Assuming a fine-coarse (F/C) ordering of nodes, we can define the prolongation operator by specifying the matrix $\mathbf{W}$ in (33).

For a given F/C partition of the system variables, the ideal interpolation, which minimizes a weak approximation property [11], is defined as:

$$\mathbf{W}_{\text{ideal}} = -\mathbf{A}_{ff}^{-1}\mathbf{A}_{fc}. \tag{39}$$

Although the ideal interpolation may give rise to the best convergence rate, its use is impractical because it is costly to compute and to apply as $\mathbf{W}_{\text{ideal}}$ as, in general, it is a dense matrix.

The prolongation $\mathbf{P}$ in several AMG methods is computed trying to approximate $\mathbf{W}_{\text{ideal}}$, for instance starting by a given pattern of $\mathbf{W}$ and running a few Jacobi or Gauss-Seidel sweeps for the approximate solution to (39). Very often, to limit the number of nonzeros in $\mathbf{W}$, only strong connections are considered in the path. However, $\mathbf{S_c}$ connectivity may vary significantly, especially in tough problems, and an *a priori* selected nonzero patterns may give rise to poor prolongation operator. The use of

Algorithm 1 **Test space generation by Lanczos algorithm**

```
procedure CPTTESTSPACE($N$,$n_t$,$\epsilon_t$,$\mathbf{A}$,$\mathbf{G}$,$\mathbf{V}$, $\omega$)
    Set $z_1$, random vector with $\|z_1\|_2 = 1$, $\beta_1 = 0$, $z_0 = 0$;
    for $i = 1, N$ do
        $w_j = (\mathbf{I} - \mathbf{G}\mathbf{A}\mathbf{G}^T)z_j - \beta_j z_{j-1}$;
        $\alpha_j = w_j^T z_j$;
        $w_j = w_j - \alpha_j z_j$;
        $\beta_{j+1} = \|w_j\|_2$;
        $z_{j+1} = w_j / \beta_{j+1}$;
    end for
    Find eigenpairs of the matrix $\mathbf{H}$, with $h_{ii} = \alpha_i$ and $h_{i-1,i} = h_{i,i+1} = \beta_i$: $\mathbf{H} = \mathbf{U}\Lambda\mathbf{U}^T$;
    Compute the approximate eigenvectors $\mathbf{Z} \leftarrow \mathbf{Z}\mathbf{U}$;
    Compute residuals for all eigenvectors $z_i$: $r_i = \|(\mathbf{I} - \mathbf{G}\mathbf{A}\mathbf{G}^T)z_i - \Lambda_i z_i\| / \|\Lambda_i z_i\|$;
    Form the test space matrix $\mathbf{V}$ containing the first $n_t$ vectors with $\Lambda_i$ closest to 1 satisfying $r_i < \epsilon_t$;
    Using the smallest (negative) converged $\Lambda_i$, compute $\omega = 2/(1 - \Lambda_i)$;
end procedure
```

Algorithm 2 **Dynamic Pattern Least Squares Prolongation**

```
procedure CPTPROLONGATION($d_p$,$\epsilon_p$,$S_c$,$\mathbf{V}_f$,$\mathbf{V}_c$, $\mathbf{W}$)
    for all nodes $i \in \mathcal{F}$ do
        $k = 0$; $\mathcal{C}_i = \emptyset$; $r = \overline{v}_i$;
        Form $\mathcal{J}_i = \{j \in \mathcal{C} \mid$ there is a path from $i$ to $j$ shorter than $d_p\}$;
        while $\|r\| \geq \epsilon_p \|v_i\|$ do
            $k = k + 1$;
            Select $\bar{j} \in \mathcal{J}_i \setminus \mathcal{C}_i$ for which $\overline{v}_j$ has maximal affinity with $r$;
            Update $\mathcal{C}_i = \mathcal{C}_i \cup \{\bar{j}\}$;
            Compute the Householder's rotation able to zero $\overline{v}_{\bar{j}}$ from component $k + 1$;
            Apply the Householder's rotation to $r$;
            for all $j \in \mathcal{J}_i \setminus \mathcal{C}_i$ do
                Apply the Householder's rotation to $\overline{v}_j$;
            end for
        end while
        Form the $n_t \times k$ upper triangular matrix $R$ collecting $\overline{v}_j$, for all $j \in \mathcal{C}_i$;
        Compute $w_i \leftarrow R^{-1} r$;
    end for
end procedure
```

long-distance neighbors can alleviate these problems, however with the expense of producing denser coarser levels that might require sparsification to reduce communication issues when running the algorithm in parallel [3].

For the above reasons, a novel iterative procedure called Dynamic Pattern Least Squares (DPLS) that shows a few analogies with approximate inverses [15, 18] is established to construct the prolongation pattern dynamically during set-up. First of all, the test space components v_i are copied in $\overline{v}_i$. 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 the matrix row w_i^T. 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 $\overline{v}_j$ giving the best possible approximation of $\overline{v}_i$ 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 $\overline{v}_i$ and $\overline{v}_{\bar{j}}$ is minimal. To choose the second most promising entry, it is necessary to update the angle estimate by computing the residuals, i.e. removing the components of both $\overline{v}_i$ and all the $\overline{v}_j$ along $\overline{v}_{\bar{j}}$. The selection of new entries stops when the following criterium is met:

$$\|\overline{v}_i - \sum_{j \in \overline{\mathcal{J}}_i} \beta_j \overline{v}_j\| \leq \epsilon_p \|v_i\|, \tag{40}$$

where $\overline{\mathcal{J}}_i \subseteq \mathcal{J}_i$ contains only the selected entries. Two parameters control the construction of the DPLS prolongation, they are:

- d_p: the maximum path length between fine nodes and interpolating coarse nodes;
- ϵ_p: the relative exit tolerance in the iterative procedure.

Algorithm 2 summarizes the procedure to compute the DPLS prolongation.

5 Numerical results

In this section, we evaluate the performance of the two proposed preconditioners for the solution of a set of real-world SPD problems from the SuiteSparse Matrix Collection [9]. Precisely, we show the computational times for building the preconditioner and solving the linear system; the number of iterations needed and density measures indicating RAM consumption. Moreover, both methods are compared to the native aFSAI algorithm as implemented in the FSAIPACK package [21] and, for the aSP-AMG, also to the BoomerAMG solver as provided by the Hypre package, version 2.13 [12].

The right hand side vector used in the test cases discussed here is 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 $10^{-10} \cdot \|b\|_2$.

To evaluate the memory occupation and the cost of applying MFLR, we define the following preconditioner density ρ_l:

$$\rho_l = \frac{1}{\text{nnz}(\mathbf{A})} \sum_{i=0}^{n_l-1} \left(\text{nnz}(\mathbf{Q}^i_{alr}) + \text{nnz}(\mathbf{Q}^i_M) \right) \tag{41}$$

where $\mathbf{Q}_{alr}$ is a queue collecting all data structures to apply the Ascending Low-Rank corrections and $\mathbf{Q}_M$ is another queue, with all matrices, such as $\mathbf{G}$, $\mathbf{F}$ and the Descending Low-Rank corrections, to build $\mathbf{P}_a$ and $\mathbf{P}_b$.

With the same aim, for aSP-AMG in a single V(1,1)-cycle, we use the classical definition of operator and grid complexities, shown in Equation (42), respectively:

$$C_{op} = \sum_{l=0}^{nl-1} \text{nnz}\left(\mathbf{A}_l\right) \Big/ \text{nnz}\left(\mathbf{A}_0\right), \quad C_{gd} = \sum_{l=0}^{nl-1} \text{nrows}\left(\mathbf{A}_l\right) \Big/ \text{nrows}\left(\mathbf{A}_0\right). \tag{42}$$

The first accounts for the space needed to store the sparse systems belonging to the multigrid hierarchy while the second measure is related to the size of auxiliary vectors employed in the preconditioner application.

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

$$\mu_G = \frac{\text{nnz}\left(\mathbf{G}\right)}{\text{nnz}\left(\mathbf{A}\right)}. \tag{43}$$

In the aSP-AMG, when aFSAI is considered as a smoother, its density is simply given by an average through levels:

$$\mu_G = \frac{\sum_{l=0}^{n_l-1} \text{nnz}\left(\mathbf{G}_l\right)}{\text{nnz}\left(\mathbf{A}_0\right)}. \tag{44}$$

We present a shared memory implementation developed by using OpenMP directives for both algorithms and run all the tests on a workstation equipped with two Intel Xeon E5-2643 processors at 3.30GHz with four physical cores each and 256 GB of RAM memory.

5.1 MFLR performance

Since the main goal of this work is to prove the robustness and effectiveness of a multilevel framework in explicit preconditioning and, in this stage, we are not looking for the optimal implementation, all the CPU times presented in this section are obtained using 1 thread on the described computer.

The dimension, number of nonzeros, average number of nonzeros per row together with a brief description of the chosen matrices are listed in Table 1.

Table 1: Test matrices for MFLR representing the real-world engineering problems.

Matrix name	nrows	nnz	avg. nnzr	Short description
`af_shell3`	504,855	17,588,875	35	3D elasticity
`af_shell8`	504,855	17,579,155	35	3D elasticity
`Emilia_923`	923,136	40,373,538	44	3D elasticity
`StocF_1465`	1,465,137	21,005,389	14	3D fluid flow

In Table 2 we reported densities, number of iterations and times to build the preconditioner and solve the linear system using the presented multilevel approach. We note that these results were obtained with the best possible configuration for each of the preconditioners tested regarding solution time.

For results in Table 2, the MFLR preconditioner set-up 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 significant. For instance, in the `af_shell3` and `af_shell8` 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 set-up time can be amortized along several linear solves. On the other hand, if there is an increase in the number of iterations, such as in the `StocF_1465` test case, the aFSAI proves more efficient than the MFLR preconditioner.

Table 2: Performance comparison between aFSAI and MFLR preconditioners in the solution of the real-world engineering problems.

Matrix name	method	ρ_l	μ_G	n_{it}	$T_p[s]$	$T_s[s]$	$T_t[s]$
`af_shell3`	aFSAI	—	0.89	1059	75.3	59.6	135.0
	MFLR	4.50	—	139	201.7	29.4	231.2
`af_shell8`	aFSAI	—	0.86	1136	68.9	63.5	132.4
	MFLR	3.22	—	144	148.3	26.7	175.0
`Emilia_923`	aFSAI	—	0.11	1664	4.2	119.9	124.2
	MFLR	0.25	—	633	70.3	75.0	145.3
`StocF_1465`	aFSAI	—	1.45	598	106.0	86.8	192.8
	MFLR	1.66	—	1028	220.8	188.3	409.1

Table 3: Test matrices for aSP-AMG representing the real-world engineering problems.

Matrix name	**nrows**	**nnz**	**avg. nnzr**	**Short description**
`pflow742`	742,793	37,138,461	49	3D fluid flow
`spe10`	1,122,005	7,780,175	6	3D fluid flow
`StocF_1465`	1,465,137	21,005,389	14	3D fluid flow
`flan1565`	1,564,794	114,165,372	72	3D elasticity
`abq_powtrain`	1,609,950	68,660,476	42	3D elasticity

Table 4: Parameters used to obtain the best performance of aSP-AMG preconditioners in the solution of the real-world engineering problems.

Matrix name	Smoother			Coarsening	Test space		Prolongation	
	k_{max}	ρ_G	ϵ_G	θ	n_t	ϵ_t	d_p	ϵ_p
`pflow742`	5	3	10^{-3}	5	10	10^{-2}	2	10^{-2}
`spe10`	10	2	10^{-3}	2	8	10^{-2}	1	10^{-2}
`StocF_1465`	10	1	10^{-3}	2	10	10^{-2}	2	10^{-2}
`flan1565`	15	1	10^{-8}	7	15	10^{-2}	2	10^{-2}
`abq_powtrain`	10	2	10^{-8}	5	15	10^{-2}	2	10^{-3}

Table 5: 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
`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	1.10	64	5.3	3.5	8.8
`StocF_1465`	aFSAI	—	—	1.45	597	12.9	21.4	34.3
	BoomerAMG	1.59	1.62	—	32	1.3	4.0	5.3
	aSP-AMG	2.11	2.27	1.09	51	14.7	6.4	21.1
`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

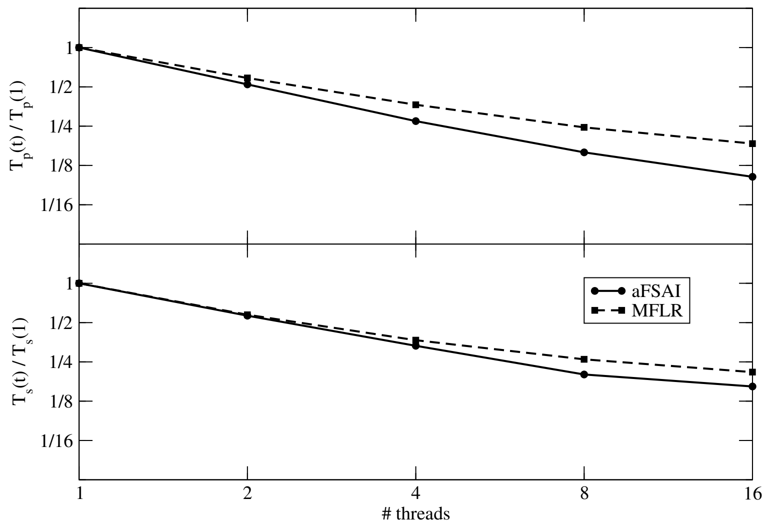

Figure 2: MFLR: strong scalability test for a regular 150^3 Laplacian.

5.2 aSP-AMG performance

Unlike the other cases, here tests are performed with 16 threads, i.e. all the available cores in a shared-memory environment. The description of the matrices used to evaluate aSP-AMG performance is given in Table 3. Table 4 shows the combination of parameters used to obtain the best total solution time for the preconditioner considered here. Finally, Table 5 collects grid and operator complexities, aFSAI density, number of iterations and computational times for all matrices.

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. Another notable 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. Thus, the aFSAI smoother yields a faster strategy, showing a higher degree of parallelism.

Comparing the results obtained with BoomerAMG and aSP-AMG, we note that the first one provides a faster set-up, which can be explained by the construction of the smoother and the test space, phases owned by aSP-AMG only. However, aSP-AMG is still competitive against BoomerAMG because the high set-up time is compensated by the faster convergence of the aSP-AMG 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 suggests that this method proves to be more efficient when employed in the solution of matrices with larger near-null space dimension.

To see how the presented approaches mutually compare, for the case `StocF_1465` we run the MFLR preconditioner with 16 threads, obtaining a preconditioning time of $T_p = 87.0s$ and a solution time of $T_s = 82.9s$, for a total time equal to $T_t = 169.9s$. From this result, we can state that the aSP-AMG reveals to be superior to the MFLR approach.

6 Strong scalability

In this section the potential parallelism of the described algorithms is presented. The scalability test has been carried out on a discrete Laplacian computed over a regular 150 × 150 × 150 grid. The numerical experiment is performed on a local cluster with two Intel Xeon E5-2680 v2 processors at 2.80GHz with ten physical cores each, and 256 GB of RAM memory. As measure of the speed-up we use the relative preconditioning and solution times with respect to times obtained using 1 thread:

$$S_p = \frac{T_p(t)}{T_p(1)}, \quad S_s = \frac{T_s(t)}{T_s(1)} \tag{45}$$

where $T_p(t)$ and $T_s(t)$ are the preconditioner set-up and solution phase wall-clock times measured with t threads, respectively. Figures 2 and 3 show the speed-ups S_p and S_s compared to those of the aFSAI algorithm, which shows an almost ideal speed-up [20]. The performance on the specific computational architecture used for the numerical experiments depends on the hardware properties, such as speed of memory access.

It is well-known that bandwidth limited algorithms such as iterative solvers, characterized by a low bit per flop ratio, cannot obtain ideal speed-ups in the case of the used multi-core processors. Nevertheless, both algorithms have a good degree of parallelism, comparable with that of aFSAI.

7 Conclusions

In this work we presented two preconditioning approaches for SPD matrices. The first one is a robust multilevel framework, which uses the FSAI preconditioner as basic kernel. We overcome the difficulties arising in a standard multilevel approach with an alternative way of computing the Schur complement. In this way, the positive definiteness of all Schur complements is

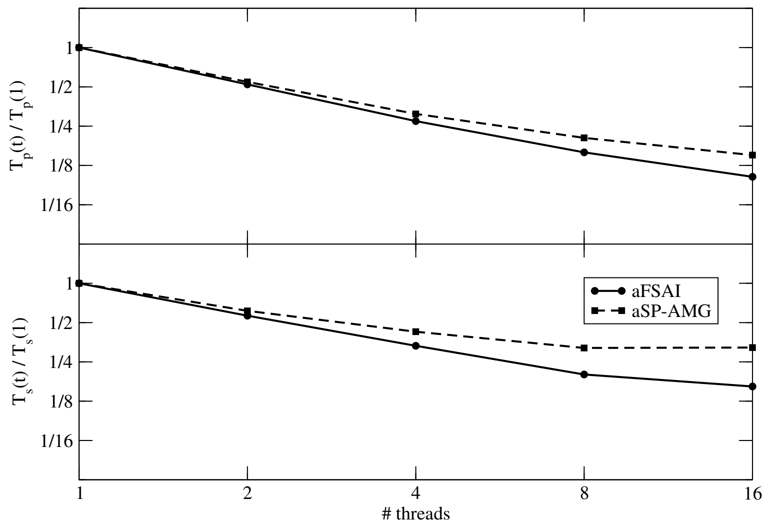

Figure 3: aSP-AMG: strong scalability test for a regular 150^3 Laplacian.

ensured independently of the preconditioner sparsity. 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 the computational performance. The numerical results show that the proposed approach is generally able to significantly accelerate the solver convergence rate. Mainly due to the computation of the eigenpairs needed by the Low-Rank corrections, we have large set-up costs, which make this method attractive especially for those applications where the preconditioner can be effectively recycled along a number of linear solves. Further investigations will be devoted in the development of a faster set-up stage, by using for instance randomized approaches while computing low rank corrections [16].

The second idea presented in this work is a novel AMG package based on an approximate inverse as smoother and a new adaptive strategy for computing the prolongation operator. Moreover, this method, named aSP-AMG, belongs to the bootstrap AMG family, which may not assume any information about the near-null space of the original matrix.

We compared the performance of aSP-AMG to the aFSAI and BoomerAMG preconditioners in the solution of real-world problems and proved that our method leads to the faster solution in most of the cases regarding both iteration time as well as total execution time. The next steps of the present research will concern the development of new techniques for predicting the smooth vector space aiming to reduce the set-up time. Lastly, a leading goal will be the efficient implementation of this package on modern massively parallel computers for the solution of large-scale problems.

A topic of increasing interest is the development of similar preconditioners for non-symmetric linear systems. Currently, we are working to extent the aSP-AMG to general system, using ideas such as those presented in [13, 33].

References

[1] M. Benzi, *Preconditioning Techniques for Large Linear Systems: A Survey*, Journal of Computational Physics, 182 (2002), pp. 418–477, http://www.sciencedirect.com/science/article/pii/S0021999102971767.

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

[3] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel communication in algebraic multigrid through sparsification*, SIAM Journal on Scientific Computing, 38 (2016), pp. S332–S357, https://doi.org/10.1137/15M1026341.

[4] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap Algebraic Multigrid: Status Report, Open Problems, and Outlook*, Numerical Mathematics: Theory, Methods and Applications, 8 (2015), pp. 112–135, https://doi.org/10.4208/nmtma.2015.w06si.

[5] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, *Adaptive Algebraic Multigrid*, SIAM Journal on Scientific Computing, 27 (2006), pp. 1261–1286, https://doi.org/10.1137/040614402.

[6] M. Brezina, C. Ketelsen, T. Manteuffel, S. McCormick, M. Park, and J. Ruge, *Relaxation-corrected Bootstrap Algebraic Multigrid (rBAMG)*, Numerical Linear Algebra with Applications, 19 (2012), pp. 178–193, https://doi.org/10.1002/nla.1821.

[7] Y. Bu, B. Carpentieri, Z. Shen, and T. Huang, *A hybrid recursive multilevel incomplete factorization preconditioner for solving general linear systems*, Applied Numerical Mathematics, 104 (2016), pp. 141–157, http://www.sciencedirect.com/science/article/pii/S0168927416000271.

[8] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023, https://doi.org/10.1137/S1064827594270415.

[9] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Softw., 38 (2011), pp. 1:1–1:25, http://doi.acm.org/10.1145/2049662.2049663.

[10] V. Dolean, P. Jolivet, and F. Nataf, *An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation*, SIAM, 2015, https://hal.inria.fr/cel-01100932v3/document.

[11] R. D. Falgout and P. S. Vassilevski, *On Generalizing the Algebraic Multigrid Framework*, SIAM Journal on Numerical Analysis, 42 (2004), pp. 1669–1693, https://doi.org/10.1137/S0036142903429742.

[12] R. D. Falgout and U. M. Yang, *hypre: a Library of High Performance Preconditioners*, in Preconditioners, Lecture Notes in Computer Science, 2002, pp. 632–641, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.1202.

[13] M. Ferronato, C. Janna, and G. Pini, *A generalized Block FSAI preconditioner for nonsymmetric linear systems*, Journal of Computational and Applied Mathematics, 256 (2014), pp. 230–241.

[14] A. Franceschini, V. A. Paludetto Magri, M. Ferronato, and C. Janna, *A robust multilevel approximate inverse preconditioner for symmetric positive definite matrices*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 123–147, https://doi.org/10.1137/16M1109503.

[15] M. J. Grote and T. Huckle, *Parallel Preconditioning with Sparse Approximate Inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853, https://doi.org/10.1137/S1064827594276552.

[16] N. Halko, P.-G. Martinsson, and J. A. Tropp, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM review, 53 (2011), pp. 217–288, https://doi.org/10.1137/090771806.

[17] T. Huckle, *Factorized Sparse Approximate Inverses for Preconditioning*, The Journal of Supercomputing, 25 (2003), pp. 109–117, https://doi.org/10.1023/A:1023988426844.

[18] C. Janna and M. Ferronato, *Adaptive Pattern Research for Block FSAI Preconditioning*, SIAM Journal on Scientific Computing, 33 (2011), pp. 3357–3380, https://doi.org/10.1137/100810368.

[19] C. Janna, M. Ferronato, and G. Gambolati, *Multi-level incomplete factorizations for the iterative solution of non-linear finite element problems*, International journal for numerical methods in engineering, 80 (2009), pp. 651–670, https://doi.org/10.1002/nme.2664.

[20] C. Janna, M. Ferronato, and G. Gambolati, *The use of supernodes in factored sparse approximate inverse preconditioning*, SIAM Journal on Scientific Computing, 37 (2015), pp. C72–C94, https://doi.org/10.1137/140956026.

[21] C. Janna, M. Ferronato, F. Sartoretto, and G. Gambolati, *FSAIPACK: A Software Package for High-Performance Factored Sparse Approximate Inverse Preconditioning*, ACM Trans. Math. Softw., 41 (2015), pp. 10:1–10:26, http://doi.acm.org/10.1145/2629475.

[22] L. Y. Kolotilina and A. Y. Yeremin, *Factorized sparse approximate inverse preconditionings I. Theory*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 45–58, https://doi.org/10.1137/0614004.

[23] R. Li and Y. Saad, *Low-Rank Correction Methods for Algebraic Domain Decomposition Preconditioners*, SIAM Journal on Matrix Analysis and Applications, 38 (2017), pp. 807–828, https://doi.org/10.1137/16M110486X.

[24] Z. Li, Y. Saad, and M. Sosonkina, *pARMS: a parallel version of the algebraic recursive multilevel solver*, Numerical linear algebra with applications, 10 (2003), pp. 485–509, https://doi.org/10.1002/nla.325.

[25] O. E. Livne and A. Brandt, *Lean Algebraic Multigrid (LAMG): Fast Graph Laplacian Linear Solver*, SIAM Journal on Scientific Computing, 34 (2012), pp. B499–B522, https://doi.org/10.1137/110843563.

[26] J. W. McCormick, S. F.; Ruge, *Multigrid Methods for Variational Problems*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 924–929, https://doi.org/10.1137/0719067.

[27] V. A. Paludetto Magri, A. Franceschini, M. Ferronato, and C. Janna, *Multilevel approaches for FSAI preconditioning*, Numerical Linear Algebra with Applications, (2018), pp. 1–18, https://doi.org/10.1002/nla.2183.

[28] V. A. Paludetto Magri, A. Franceschini, and C. Janna, *A novel AMG approach based on adaptive smoothing and prolongation for ill-conditioned systems*, SIAM Journal on Scientific Computing, (2018). Manuscript submitted for publication.

[29] P. Raghavan and K. Teranishi, *Parallel hybrid preconditioning: Incomplete factorization with selective sparse approximate inversion*, SIAM Journal on Scientific Computing, 32 (2010), pp. 1323–1345, https://doi.org/10.1137/080739987.

[30] Y. Saad, *ILUT: a Dual Threshold Incomplete LU Factorization*, Numerical Linear Algebra with Applications, 1 (1994), pp. 387–402, http://dx.doi.org/10.1002/nla.1680010405.

[31] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, SIAM, 2011, http://epubs.siam.org/doi/abs/10.1137/1.9781611970739.

[32] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical linear algebra with applications, 9 (2002), pp. 359–378, https://doi.org/10.1002/nla.279.

[33] B. Southworth, T. A. Manteuffel, and J. Ruge, *Nonsymmetric Algebraic Multigrid Based on Local Approximate Ideal Restriction (LAIR)*, arXiv preprint arXiv:1708.06065, (2017).

[34] Y. Xi, R. Li, and S. Y., *An Algebraic Multilevel Preconditioner with Low-Rank Corrections for Sparse Symmetric Matrices*, SIAM Journal on Matrix Analysis and Applications, 37 (2016), pp. 235–259, http://dx.doi.org/10.1137/15M1021830.

[35] J. Xu and L. Zikatanov, *Algebraic multigrid methods*, Acta Numerica, 26 (2017), pp. 591–721, https://doi.org/10.1017/S0962492917000083.

[36] U. M. Yang, *Parallel Algebraic Multigrid Methods — High Performance Preconditioners*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 209–236, https://doi.org/10.1007/3-540-31619-1_6.