Exploiting Symmetries for Preconditioning Poisson's Equation in CFD Simulations

Àdel Alsalti-Baldellou
adel.alsalti@upc.edu
Universitat Politècnica de Catalunya – BarcelonaTech
Terrassa (Barcelona), Spain
Termo Fluids SL
Sabadell (Barcelona), Spain

Carlo Janna
carlo.janna@unipd.it
University of Padova
Padova, Italy
M3E S.r.l.
Padova, Italy

Xavier Álvarez-Farré
xavier.alvarezfarre@surf.nl
SURF
Amsterdam, Netherlands

F. Xavier Trias
francesc.xavier.trias@upc.edu
Universitat Politècnica de Catalunya – BarcelonaTech
Terrassa (Barcelona), Spain

ABSTRACT

Divergence constraints are present in the governing equations of many physical phenomena, and they usually lead to a Poisson equation whose solution is one of the most challenging parts of scientific simulation codes. Indeed, it is the main bottleneck of incompressible Computational Fluid Dynamics (CFD) simulations, and developing efficient and scalable Poisson solvers is a critical task. This work presents an enhanced variant of the Factored Sparse Approximate Inverse (FSAI) preconditioner. It arises from exploiting s spatial reflection symmetries, which are often present in academic and industrial configurations and allow transforming Poisson's equation into a set of 2^s fully-decoupled subsystems. Then, we introduce another level of approximation by taking advantage of the subsystems' close similarity and applying the same FSAI to all of them. This leads to substantial memory savings and notable increases in the arithmetic intensity resulting from employing the more compute-intensive sparse matrix-matrix product. Of course, recycling the same preconditioner on all the subsystems worsens its convergence. However, this effect was much smaller than expected and made us introduce relatively cheap but very effective low-rank corrections. A key feature of these corrections is that thanks to being applied to each subsystem independently, the more symmetries being exploited, the more effective they become, leading to up to 5.7x faster convergences than the standard FSAI. Numerical experiments on up to 1.07 billion grids confirm the quality of our low-rank corrected FSAI, which, despite being 2.6x lighter, outperforms the standard FSAI by a factor of up to 4.4x.

CCS CONCEPTS

• **Mathematics of computing → Solvers**; *Mathematical software performance*; • **Applied computing →** *Engineering*.

KEYWORDS

Poisson equation, FSAI, Low-Rank Corrections, Arithmetic Intensity, CFD

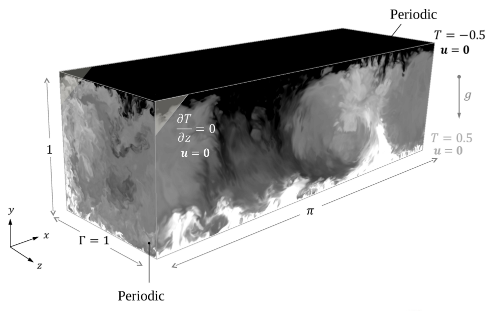
ACM Reference Format:
Àdel Alsalti-Baldellou, Carlo Janna, Xavier Álvarez-Farré, and F. Xavier Trias. 2023. Exploiting Symmetries for Preconditioning Poisson's Equation in CFD Simulations. In *Platform for Advanced Scientific Computing Conference (PASC '23), June 26–28, 2023, Davos, Switzerland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3592979.3593410

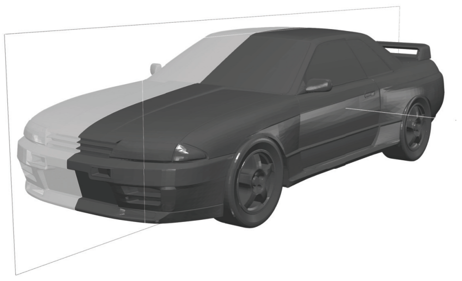
1 INTRODUCTION

Divergence constraints are ubiquitous in physical problems. Under certain assumptions, they follow basic conservation principles such as mass conservation, electrical charge conservation or the conservation of probability in quantum mechanics. Such a constraint leads to a Poisson equation for a sort of scalar potential. Hence, it is not surprising that Poisson's equation plays a fundamental role in many areas of science and engineering, such as computational fluid dynamics (CFD), linear elasticity, electrostatics and quantum mechanical continuum solvation models. Additionally, it is usually one of the most time-consuming and difficult to parallelise parts of scientific simulation codes.

In order to develop efficient and scalable solvers, it is necessary to identify the current computing devices' limitations and develop algorithms that overcome them. For instance, the low arithmetic intensity of most (sparse) linear algebra kernels motivated strategies like using mixed precision [4] or applying more compute-intensive algorithms [10]. Similarly, the large ratio of network to memory bandwidth led to implementations hiding or completely avoiding inter-node communications [16, 19, 21]. On its side, the limited available memory resulted in approaches like exploiting data sparsity [8, 3].

In this context, we focus on developing a parallel Poisson preconditioner flexible enough to run efficiently on different kinds of systems by mitigating several of the aforementioned limitations. The targeted applications are incompressible CFD simulations. In particular, direct numerical simulation (DNS) and large-eddy simulation (LES) of turbulent flows. The following four aspects, which are also relevant in the context of this paper, are commonly present in many DNS and LES applications:





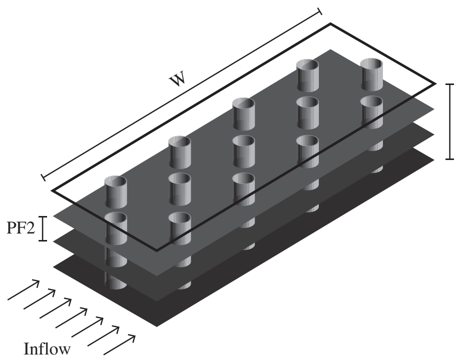


Figure 1: Incompressible flow simulations presenting up to three reflection symmetries. From top to bottom: Rayleigh-Bénard convection [5], car geometry and fin-and-tube heat exchanger.

- The same Poisson equation has to be solved repeatedly with different right-hand-side terms, *e.g.*, for DNS and LES problems the number of time-steps easily reach $O(10^6)$. Hence, a pre-processing stage with large computing demands can be accepted.
- The solution obtained in the previous time step(s) can be used as an initial guess for iterative solvers in order to accelerate the convergence.
- The boundary conditions for the Poisson equation are homogeneous Neumann regardless of the boundary conditions for the velocity and temperature fields.
- Domains with spatial reflection symmetries are often present in canonical flows but also in many industrial applications. Examples thereof are displayed in fig. 1.

A few works exploiting symmetries for solving Poisson's equation already exist [2, 1, 9, 22]. In the present paper, we extend them to propose an improved variant of the Factored Sparse Approximate Inverse (FSAI) preconditioner [15]. Namely, given an arbitrarily complex geometry presenting s reflection symmetries, we have started by transforming the resulting Laplace operator into a set of 2^s decoupled subsystems. Then, given their close similarity, we have added an extra level of approximation by applying the same FSAI to all the subsystems. Surprisingly enough, this did not result in significantly slower convergences. Nevertheless, we introduced relatively cheap but very effective low-rank corrections [17, 18]. A key feature of these corrections is that, the more symmetries being exploited, the more effective they are, resulting in considerably faster convergences. On the other hand, exploiting symmetries and recycling the same FSAI on all the subsystems allowed increasing considerably the arithmetic intensity of our preconditioner. This was done by applying it through the more compute-intensive sparse matrix-matrix product (`SpMM`) instead of using the standard sparse matrix-vector product (`SpMV`). Remarkably enough, the strategy presented in this work is naturally extensible to virtually any preconditioner explicitly factorisable such as those based on incomplete factorisations.

The remaining sections of this work are organised as follows. Section 2 derives the proposed low-rank corrected FSAI, and details its construction and application. Section 3 discusses its practical implementation, which has been made publicly available. Section 4 analyses the benefits of our proposal by comparing it with a standard FSAI in meaningful numerical experiments. Finally, section 5 gives some concluding remarks.

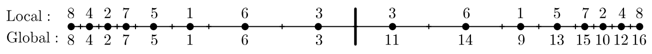

Local : 8 4 2 7 5 1 6 3 3 6 1 5 7 2 4 8
Global : 8 4 2 7 5 1 6 3 11 14 9 13 15 10 12 16

Figure 2: Single-symmetry 1D mesh with mirrored ordering.

2 INCREASED ARITHMETIC INTENSITY FSAI

The aim of this section is to show how to exploit spatial reflection symmetries for preconditioning Poisson's equation. First, we will show how to transform the original Laplace operator into a set of decoupled subsystems. Then, we will show how to take advantage of their regular structure to improve the FSAI preconditioner.

2.1 Exploiting symmetries

Given an arbitrary mesh presenting a single reflection symmetry, *e.g.*, such as fig. 2, let us order its grid points by first indexing the ones lying on one half and then those in the other. If we impose to the resulting two subdomains the same local ordering (mirrored by the symmetry's hyperplane), we ensure that all the scalar fields satisfy:

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^n, \tag{1}$$

where n stands for the mesh size and $x_1, x_2 \in \mathbb{R}^{n/2}$ for x's restriction to each of the subdomains. Mirrored grid points are in the same position within the subvectors, and the discrete Laplace operator reads:

$$A = \begin{pmatrix} A_{\text{inn}} & A_{\text{out}} \\ A_{\text{out}} & A_{\text{inn}} \end{pmatrix} \in \mathbb{R}^{n \times n}, \tag{2}$$

where $A_{\text{inn}}, A_{\text{out}} \in \mathbb{R}^{n/2 \times n/2}$ correspond to the inner- and outer-subdomain couplings, respectively.

In such a context, by denoting with $\mathbb{I}_k$ the identity matrix of order k, we can define the following change-of-basis matrix [9]:

$$P := \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbb{I}_{n/2} & \mathbb{I}_{n/2} \\ \mathbb{I}_{n/2} & -\mathbb{I}_{n/2} \end{pmatrix} \in \mathbb{R}^{n \times n}, \quad (3)$$

which satisfies $P^{-1} = P$. Then, applying this change of basis to A leads to:

$$PAP^{-1} = \begin{pmatrix} A_{\text{inn}} + A_{\text{out}} & 0 \\ 0 & A_{\text{inn}} - A_{\text{out}} \end{pmatrix}, \quad (4)$$

transforming eq. (2) into two fully-decoupled and half-sized subsystems, $\hat{A}_1 := A_{\text{inn}} + A_{\text{out}}$ and $\hat{A}_2 := A_{\text{inn}} - A_{\text{out}}$.

This strategy to exploit symmetries can be applied recursively by defining the following change of basis [2]:

$$P_s := \prod_{i=1}^{s} \left(\mathbb{I}_{2^{i-1}} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \mathbb{I}_{n/2^i} \right) \in \mathbb{R}^{n \times n}, \quad (5)$$

which satisfies $P_s^{-1} = P_s$ and allows transforming the original Laplace operator, A, into 2^s decoupled subsystems:

$$\hat{A} := P_s A P_s^{-1} = \begin{pmatrix} \hat{A}_1 & & 0 \\ & \ddots & \\ 0 & & \hat{A}_{2^s} \end{pmatrix}. \quad (6)$$

This decomposition of Poisson's equation is very beneficial. Leaving the computational advantages for section 3, let us note that reducing the size of the linear systems makes the time complexity of the solvers considerably lower. For instance, solving separately the 2^s decoupled subsystems $\hat{A}_1, \ldots, \hat{A}_{2^s}$ instead of the entire A leads to faster convergence. Algorithm 1 summarises the resulting algorithm.

Algorithm 1 Poisson solver exploiting s reflection symmetries

```
Require: $\hat{A}_1, \ldots, \hat{A}_{2^s}$, $P_s$ and $b \in \text{range}(A) \subseteq \mathbb{R}^n$
1: procedure SOLVE($b$)
2:     Transform forward: $\hat{b} = P_s b$
3:     Decoupled solution of $\hat{A}_i \hat{x}_i = \hat{b}_i \;\; \forall i \in \{1, \ldots, 2^s\}$
4:     Transform backward: $x = P_s \hat{x}$
5:     return $x$
6: end procedure
```

2.2 Low-Rank Corrections for FSAI

Our aim now is to show how to exploit symmetries for enhancing the FSAI preconditioner by means of low-rank corrections.

Let us start noting that, similarly to eq. (4), the matrix $\hat{A}$ defined in eq. (6) can be split as follows:

$$\hat{A} = \mathbb{I}_{2^s} \otimes A_{\text{inn}} + \begin{pmatrix} A_{\text{out},1} & & 0 \\ & \ddots & \\ 0 & & A_{\text{out},2^s} \end{pmatrix}, \quad (7)$$

where all the outer-couplings, $A_{\text{out},i} \in \mathbb{R}^{n/2^s \times n/2^s}$, are substantially sparser than A_{inn}, and generally present the same sparsity pattern, approximately having as many non-zeros as grid points adjacent to any of the symmetries' hyperplanes. Therefore, $\text{rank}(A_{\text{out},i})$ is of the order of $O((n/2^s)^{2/3})$ and:

$$\text{rank}\left(A_{\text{out},i}\right) \ll \text{rank}\left(A_{\text{inn}}\right) \quad \forall i \in \{1, \ldots, 2^s\}, \quad (8)$$

where $\text{rank}(A_{\text{inn}}) \simeq n/2^s$.

At this point, let us consider the FSAI of A_{inn}. It provides an approximation to the inverse of A_{inn}'s lower Cholesky factor, $G_{\text{inn}} \simeq L_{\text{inn}}^{-1}$, which ensures that:

$$G_{\text{inn}}^T G_{\text{inn}} \simeq A_{\text{inn}}^{-1}. \quad (9)$$

Then, we can define the following auxiliary matrix for each subsystem $\hat{A}_i$:

$$Y := \mathbb{I}_{n/2^s} - G_{\text{inn}} \hat{A}_i G_{\text{inn}}^T \in \mathbb{R}^{n/2^s \times n/2^s}, \quad (10)$$

whose definition yields $Y(\mathbb{I}_{n/2^s} - Y)^{-1} = (G_{\text{inn}} \hat{A}_i G_{\text{inn}}^T)^{-1} - \mathbb{I}_{n/2^s}$, finally leading to:

$$\hat{A}_i^{-1} = G_{\text{inn}}^T G_{\text{inn}} + G_{\text{inn}}^T Y (\mathbb{I}_{n/2^s} - Y)^{-1} G_{\text{inn}}. \quad (11)$$

Although being very useful, eq. (11) cannot be applied given the unaffordable costs of the "full-rank" correction it provides. However, by virtue of eq. (8), we can expect such a correction to have a very high data sparsity, *i.e.*, its action can be well represented by a low-rank approximation [17]. With this assumption, let us truncate Y's eigendecomposition to account for its k most *relevant* eigenpairs:

$$Y \simeq V_k \Sigma_k V_k^T \quad (12)$$

such that $V_k \in \mathbb{R}^{n/2^s \times k}$ and $\Sigma_k \in \mathbb{R}^{k \times k}$. Then, eqs. (11) and (12) can be combined to give the following low-rank correction:

$$\hat{A}_i^{-1} \simeq G_{\text{inn}}^T G_{\text{inn}} + W_k \Theta_k W_k^T, \quad (13)$$

where $W_k := G_{\text{inn}}^T V_k \in \mathbb{R}^{n/2^s \times k}$ and $\Theta_k := \Sigma_k (\mathbb{I}_k - \Sigma_k)^{-1} \in \mathbb{R}^{k \times k}$.

When it comes to selecting the most relevant eigenpairs, it is worth remarking the numerical meaning of Y. Namely, it gives a measure of how far it is each preconditioned subsystem, $G_{\text{inn}} \hat{A}_i G_{\text{inn}}^T$, from the identity matrix. Recalling the harmful effect of the small eigenvalues in the Preconditioned Conjugate Gradient (PCG) convergence [23], it becomes clear that the k most effective eigenpairs for the low-rank correction are those associated with the smallest eigenvalues of $X := G_{\text{inn}} \hat{A}_i G_{\text{inn}}^T$. Then, we can compute such a low-rank representation of X:

$$X \simeq U_k \Lambda_k U_k^T, \quad (14)$$

which conveniently leads to:

$$Y \simeq U_k (\mathbb{I}_k - \Lambda_k) U_k^T. \quad (15)$$

Note that computing the smallest eigenpairs of a matrix is generally a complex problem, often even more challenging than solving a linear system. However, only rough and cost-effective approximations of U_k and Λ_k are needed for preconditioning.

Hence, applying the above procedure to each of the 2^s subsystems results in the following approximation of $\hat{A}^{-1}$:

$$\mathbb{I}_{2^s} \otimes G_{\text{inn}}^T G_{\text{inn}} + \begin{pmatrix} W_{k,1} \Theta_{k,1} W_{k,1}^T & & 0 \\ & \ddots & \\ 0 & & W_{k,2^s} \Theta_{k,2^s} W_{k,2^s}^T \end{pmatrix}, \quad (16)$$

which corresponds to a preconditioner henceforth named low-rank corrected FSAI, and denoted by LRCFSAI(k). Algorithm 2 summarises its setup and application.

Algorithm 2 Low-rank corrected FSAI exploiting s symmetries

```
Require: $A_{\text{inn}}, A_{\text{out},1}, \dots, A_{\text{out},2^s}$ and $k \geq 0$
 1: procedure SETUP($k, \hat{A}_1, \dots, \hat{A}_{2^s}$)
 2:     Compute FSAI factors of $A_{\text{inn}}$: $G_{\text{inn}}$ and $G_{\text{inn}}^T$
 3:     for $i = 1, \dots, 2^s$ do
 4:         Compute rank-$k$ decomposition of $X \simeq U_k \Lambda_k U_k^T$
 5:         Compute $\Sigma_k = \mathbb{I}_k - \Lambda_k$
 6:         Compute rank-$k$ correction:
                $\Theta_{k,i} = \Sigma_k (\mathbb{I}_k - \Sigma_k)^{-1}$ and $W_{k,i} = U_k$
 7:     end for
 8: end procedure
 9: procedure APPLY($\mathbf{r}$)
10:     Apply FSAI: $p = (\mathbb{I}_{2^s} \otimes G_{\text{inn}}^T G_{\text{inn}}) r$
11:     for $i = 1, \dots, 2^s$ do
12:         Extract $r_i$ from $r = (r_1^T, \dots, r_{2^s}^T)^T$
13:         Apply $i$th rank-$k$ correction: $q_i = W_{k,i} \Theta_{k,i} W_{k,i}^T r_i$
14:     end for
15:     Reconstruct $q = (q_1^T, \dots, q_{2^s}^T)^T$
16:     return $z = p + q$
17: end procedure
```

Taking a look at line 10 in algorithm 2, the main motivation for developing LRCFSAI(k) is to allow replacing the SpMV with the SpMM, a considerably more compute-intensive kernel (see section 3). Additionally, by reusing the same FSAI on all the subsystems, we reduce the approximate inverses' memory footprint and setup costs by a factor of 2^s. Of course, this comes at the price of using lower quality approximations since G_{inn} does not account for their outer-subdomain couplings, $A_{\text{out},i}$. However, the rapidly decaying spectrums of $\hat{A}_i^{-1} - A_{\text{inn}}^{-1}$ make it possible to overcome this drawback by introducing low-rank corrections, representing a relatively small overhead. Of course, a balance must be sought in terms of improving convergence by applying higher-rank corrections and bearing their increased costs. Finally, let us remark that the approach presented in section 2.1 to exploit symmetries is particularly well-suited for low-rank corrections (or for similar techniques like deflation or augmentation [6]). Indeed, applying a rank-k correction to each of the 2^s subsystems separately, corresponds to applying a rank-($2^s k$) correction on the global system, $\hat{A}$.

3 PRACTICAL IMPLEMENTATION

The aim of this section is to address the most important aspects regarding the distributed memory implementation of the LRCFSAI(k)[1] preconditioner, which has been made publicly available on GitHub[1]. Firstly, we will detail how to discretise domains with symmetries in order to effectively save memory, reduce communications, and increase the arithmetic intensity of the matrix multiplications. Secondly, we will present the FSAI preconditioner and Lanczos eigensolver on top of which LRCFSAI(k) has been built.

[1]Source code publicly available at https://github.com/adalbal/LRCFSAI.

3.1 Adequate domain partitioning

Discretising complex geometries becomes considerably simpler and more affordable when exploiting reflection symmetries. Indeed, the current approach only requires meshing a $1/2^s$ fraction of the entire domain, henceforth named *base mesh*. Then, the algorithm will mirror it (implicitly) by each of the symmetries' hyperplanes, thus not having to worry about building exactly symmetric meshes and saving a considerable amount of memory and computational effort.

Figure 3 illustrates the above procedure on an arbitrary unstructured grid. It also shows an adequate domain partitioning allowing for several computational advantages. Roughly, the user is free to distribute the base mesh among the available computing resources, but then such a distribution is mirrored to the rest of subdomains.

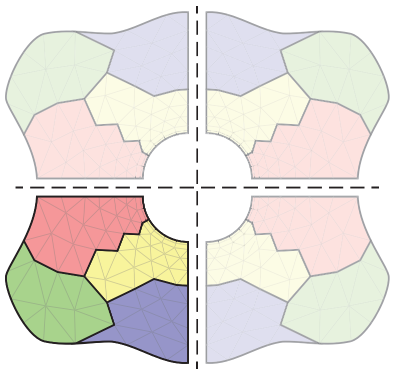


Figure 3: Adequate partitioning of a mesh with 2 symmetries.

3.1.1 *Communication-free transforms.*

The first advantage granted by such a domain partitioning is to make communication-free the two transforms of lines 2 and 4 in algorithm 1. Indeed, P_s only couples spatially symmetric nodes that, thanks to the partitioning, will always belong to the same memory space. Recalling that forward and backward transforms are computed as matrix multiplications by P_s, it becomes clear that they do not entail any communication and, therefore, represent a negligible overhead.

3.1.2 *Sparse matrix-matrix multiplication.*

When exploiting symmetries, practically all the operators involved in the simulations exhibit the following (or a very similar and compatible) block structure:

$$\hat{H} = \mathbb{I}_{2^s} \otimes H, \tag{17}$$

where $\hat{H} \in \mathbb{R}^{n \times n}$ stands for the operator itself and $H \in \mathbb{R}^{n/2^s \times n/2^s}$ for its restriction to the base mesh.

The standard approach for applying $\hat{H}$ to a vector is through an SpMV call, which performs the following operation:

$$y = \begin{pmatrix} H & & \\ & \ddots & \\ & & H \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_{2^s} \end{pmatrix} \in \mathbb{R}^n, \tag{18}$$

However, it is possible to take advantage of eq. (17) to replace SpMV with SpMM, algebraically corresponding to:

$$(y_1 \dots y_{2^s}) = H(x_1 \dots x_{2^s}) \in \mathbb{R}^{n/2^s \times 2^s}. \tag{19}$$

Although both kernels are memory-bound, SpMV needs to read H 2^s times, whereas SpMM only once. As a result, SpMM's arithmetic intensity is considerably higher and, therefore, so is its performance. Remarkably enough, eq. (17) also leads to substantial reductions in the memory footprint of the matrices, since only their restrictions to the base mesh are required. Examples of matrices satisfying eq. (17) include, among many others, $\mathbb{I}_{2^s} \otimes A_{\text{inn}}$ in eq. (7) and $\mathbb{I}_{2^s} \otimes G_{\text{inn}}^T G_{\text{inn}}$ in eq. (16).

3.2 Distributed memory implementation of aFSAI and Lanczos

Besides the PCG, the two main ingredients required for the proposed approach are an FSAI implementation able to get G_{inn} from A_{inn} and an effective eigensolver to approximate the smallest eigenpairs Λ_k and U_k. To this aim, we use the adaptive FSAI (aFSAI) preconditioner proposed in [13] and the classical Lanczos iterative method [20]. Although both algorithms are almost perfectly parallel, their implementation in a distributed memory system is not trivial as it involves a large volume of communications. Hence, special care in its design is required to hide communication latency by superposing computations. In this work, we rely on Chronos [11, 7], a linear algebra library for sparse matrix operations on high-performance computers. Chronos implements CPU-only and GPU-accelerated iterative methods and preconditioners for the solution of large-size systems and eigenproblems. All the classical iterative methods, such as PCG, GMRES, BiCGstab, Lanczos or Rayleigh quotient minimisation, are provided. On the other hand, for scalability reasons, preconditioning is mainly based on approximate inverses and algebraic multigrid.

The approximate inverse we choose for this work is a factored one based on the work [15], which will be denoted as aFSAI for simplicity. Given a matrix A (in this section, we omit the subindex "inn" for simplicity), aFSAI aims to compute an approximation of A^{-1} in a factored form, that is:

$$G^T G \simeq A^{-1}, \tag{20}$$

where G is a lower triangular sparse matrix. The factor G is computed by minimising the Frobenius norm of:

$$||I - GL|| \rightarrow \min \tag{21}$$

over all the matrices G having a prescribed non-zero pattern $\mathcal{S}$ and L is the exact Cholesky factor of A, *i.e.*, $LL^T \equiv A$. Since the parallel computation of L is not trivial at all and may be quite expensive, a key feature of FSAI is that L is not explicitly needed. In fact, after some algebra it can be shown that L disappears from the equations, and the explicit computation of the G entries can be performed by solving the following entrywise equations:

$$[GA]_{ij} = \delta_{ij} \qquad \forall (i,j) \in \mathcal{S} \tag{22}$$

where $[\cdot]_{ij}$ denote the ij component of a matrix and δ is the Kronecker delta. From a practical viewpoint, the condition imposed by eq. (22) is satisfied by solving a long sequence of small and dense linear systems. More precisely, we need to solve one dense system for the computation of every row of G, hence the procedure exhibits a very high degree of parallelism. The main drawback of the original FSAI set-up was the choice of the non-zero pattern $\mathcal{S}$ which is at the same time crucial and difficult. In fact, using a too sparse $\mathcal{S}$ results in an inaccurate preconditioner while choosing a too dense one may become exceedingly expensive. An effective remedy to this has been proposed in [13], where a dynamic procedure is implemented to choose the non-zero pattern dynamically during preconditioner set-up. The position of most promising non-zeroes is determined by estimating the variation of the Kaporin number of the preconditioned matrix, which is an alternative measure to estimate PCG convergence speed [14]. In [12], it is shown that an efficient implementation of aFSAI on distributed memory systems is possible, and its performance with up to 4096 CPU-cores and 512 GPUs has been proven in real industrial applications.

The other component needed for our approach is the estimation of the smallest eigenpairs of $G_{\text{inn}} \hat{A}_i G_{\text{inn}}^T$. To this aim, we rely on a classical Lanczos implementation without restarting. The choice of avoiding restarting strategies in Lanczos was motivated by the relatively low accuracy needed in the eigenpairs estimate. As the numerical experiments will show, requiring a high level of accuracy does not contribute to accelerating PCG convergence, thus resulting in a waste of resources. However, the numerical experiments will also show that, when dealing with large problems, the excessive memory demands of the non-restarted Lanczos may require switching to a restarted implementation. Once the user defines the rank of the correction, the dimension of the Lanczos space is set accordingly, and a basis of the subspace is constructed through the Gram-Schmidt procedure with the only simple care of re-orthogonalising the base vectors whenever linear independence is lost. The Lanczos eigensolver is a high-level procedure in Chronos since, like most iterative solution algorithms, it only requires basic linear algebra operations, such as matrix by vector products, scalar products and vector updates, which are already available in Chronos as low-level kernels.

4 NUMERICAL EXPERIMENTS

The aim of this section is to show the benefits of the LRCFSAI(k) developed in the preceding sections by comparing it with a standard application of the aFSAI in meaningful numerical experiments. All the executions rely on combined MPI and multithreaded parallelism, and have been conducted on the MARCONI100 supercomputer at the Italian Center for High Performance Computing (CINECA). Its non-uniform memory access (NUMA) nodes are equipped with two IBM POWER9 AC922 (16 cores, 2.6 GHz, 27.5 MB and 120 GB/s memory bandwidth), linked to 256 GB of RAM, and interconnected through 12.5GB/s Mellanox IB EDR DragonFly++.

The domain considered for the experiments is analogous to that of fig. 1. As discussed in section 3.1, we did only have to discretise a fraction of it in accordance with the number of symmetries being exploited. Namely, a half, a quarter and an eighth of the domain when exploiting $s = 1, 2, 3$ symmetries, respectively. This was done using a standard 7-point stencil and a refinement at the walls defined by the following hyperbolic tangent function:

$$x_i = \frac{1}{2}\left(1 + \frac{\tanh\left(\gamma_x \left(2\frac{(i-1)}{n_x} - 1\right)\right)}{\tanh\left(\gamma_x\right)}\right) \quad \forall i \in \{1, \ldots, n_x + 1\}, \tag{23}$$

which was applied analogously in the y- and z-directions. Then, the three test meshes considered were equally stretched according to: $\gamma = (1.35, 1.2, 1.45)$, and of increasing size: $n = 256^3, 512^3, 1024^3$. The right-hand side (RHS) vectors were randomly generated, and the initial guesses assumed null. Regarding the convergence criteria, we set a relative tolerance equal to 10^{-8} without any limitation in the iterations' count.

It is worth remarking that, when exploiting symmetries, compact stencils only coupling adjacent nodes give rise to a Laplace operator with diagonal outer-couplings:

$$\begin{pmatrix} A_{\text{out},1} & & 0 \\ & \ddots & \\ 0 & & A_{\text{out},2^s} \end{pmatrix} = \operatorname{diag}(a_{\text{out}}) \in \mathbb{R}^{n \times n}, \tag{24}$$

which can be naturally stored like the rest of vectors:

$$a_{\text{out}} = (a_{\text{out},1} \dots a_{\text{out},2^s}) \in \mathbb{R}^{n/2^s \times 2^s}. \tag{25}$$

Then, by virtue of eq. (7), all the coefficient matrix multiplications required by PCG can be computed as a combination of an SpMM and an element-wise product of vectors (axty):

$$\begin{aligned} \hat{A}(x_1 \dots x_{2^s}) = {} & A_{\text{inn}}(x_1 \dots x_{2^s}) + \\ & + (a_{\text{out},1} \dots a_{\text{out},2^s}) \odot (x_1 \dots x_{2^s}), \end{aligned} \tag{26}$$

therefore replacing the less compute-intensive SpMV.

Regarding the preconditioners, we compared the standard aFSAI:

$$\hat{G}^T \hat{G} := \begin{pmatrix} G_1 G_1^T & & 0 \\ & \ddots & \\ 0 & & G_{2^s} G_{2^s}^T \end{pmatrix} \simeq \hat{A}^{-1}, \tag{27}$$

satisfying $G_i G_i^T \simeq \hat{A}_i^{-1}$ for all $i \in \{1, \dots, 2^s\}$, with the LRCFSAI(k) of eq. (16) resulting from $k \in \{0, 1, 2, 8, 16\}$.

As explained in section 3.2, for the estimation of the smallest eigenpairs of $G_{\text{inn}} \hat{A}_i G_{\text{inn}}^T$, we relied on a classical non-restarted Lanczos implementation. Remarkably enough, the similarity of $\hat{A}_1, \dots, \hat{A}_{2^s}$ allows improving the eigensolvers convergence by recycling previously calculated eigenpairs as initial guesses for subsequent subsystems. This resulted very beneficial given the large memory demands of performing too many non-restarted Lanczos iterations. In fact, in the largest cases we were unable to meet the desired accuracy in the computation of LRCFSAI(k), making it clear the need for a restarted implementation. Table 1 illustrates the impact of the eigenpairs' accuracy on the quality of LRCFSAI(k). According to it, a tolerance in between 10^{-3} and 10^{-4} is enough to get the most out of the rank-k corrections. Therefore, in our experiments we imposed to Lanczos a tolerance equal to 10^{-3}, and allowed as many iterations as permitted by the nodes capacity.

Regarding the memory footprint of the solver, *i.e.*, the footprint of both the coefficient matrix and the preconditioner, fig. 4 summarises the very important reductions granted by exploiting symmetries. In this sense, only having to store A_{inn}, a_{out}, G_{inn} and G_{inn}^T made the extra space required by the low-rank corrections, $W_{k,1}, \Theta_{k,1}, W_{k,1}^T, \dots, W_{k,1}, \Theta_{k,2^s}, W_{k,2^s}^T$, more than acceptable. Indeed, the heaviest preconditioner considered, LRCFSAI(16), resulted in 2.6x memory savings with respect to aFSAI.

When it comes to the convergence of LRCFSAI(k), the higher the rank of the correction, the fewer the iterations required by PCG. Additionally, thanks to being applied independently, a rank-k correction on each subsystem corresponds to a correction on the entire system of 2^s times higher rank. Therefore, the more symmetries being exploited, the more effective the low-rank corrections are, as can be confirmed from fig. 4. These trends are clearly confirmed in fig. 4 by the 16.8M grid. The contradictory behaviour of LRCFSAI(k) on the 134M and 1.07B grids is solely due to the aforementioned low accuracy in the calculation of the smallest eigenpairs, which leads to ineffective corrections. For clarity, the points corresponding to cases in which non-restarted Lanczos was unable to meet the desired tolerance are left empty. The reason why such cases are limited to higher-rank corrections on larger grids and exploiting fewer (if any) symmetries is simple. The larger the subsystems are, the more iterations non-restarted Lanczos requires and, furthermore, the larger each vector of the Krylov subspace basis is. As a result, the limited available memory results in poor corrections. Remarkably enough, this can be easily addressed by switching to a restarted Lanczos implementation, which is a subject of ongoing work.

Table 1: Lanczos tolerance influence on the PCG iterations

n	k	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
16.8M	1	406	413	399	385	372	359
	2	454	431	369	320	327	327
	8	388	228	205	206	205	205
	16	357	184	169	161	160	160
134M	1	960	853	838	803	770	744
	2	927	881	842	663	665	664
	8	921	497	425	426	425	425
	16	898	405	348	333	328	327

Some comments were already made about the existing trade-off between the faster convergences granted by higher-rank corrections and the extra overhead their application introduces. Figure 5 illustrates this by decomposing the time spent per PCG iteration in multiplications by the coefficient matrix, aFSAI and low-rank corrections. The advantages of replacing SpMM with SpMV are clear. Indeed, applying the standard aFSAI of eq. (27), which is not compatible with SpMM, results almost twice as expensive as ignoring the Laplacian's outer-couplings and applying the same aFSAI, $G_{\text{inn}}^T G_{\text{inn}}$, to all the subsystems. On top of that, fig. 4 confirms that introducing this extra level of approximation does not harm convergence, generally requiring LRCFSAI(0) almost the same iterations as the standard aFSAI. It is worth noting that the benefits of SpMM are not so apparent for the coefficient matrix multiplications due to its high sparsity, which makes them a very fast operation. Finally, it is clear from fig. 5 the considerable overhead derived from the low-rank corrections, especially for larger values of k. LRCFSAI(k) will only be advantageous as long as the added cost of its rank-k corrections is well hidden behind the fewer iterations and faster multiplications. In this sense, the actual solution times presented in fig. 4 confirm the benefits of LRCFSAI(k). Indeed, while rank-1 and rank-2 corrections are not so effective compared to LRCFSAI(0), the considerably faster convergences attained by LRCFSAI(16) make it up to 2.5x faster than the standard application of aFSAI exploiting $s = 3$ symmetries, and up to 4.4x faster than aFSAI without exploiting symmetries.

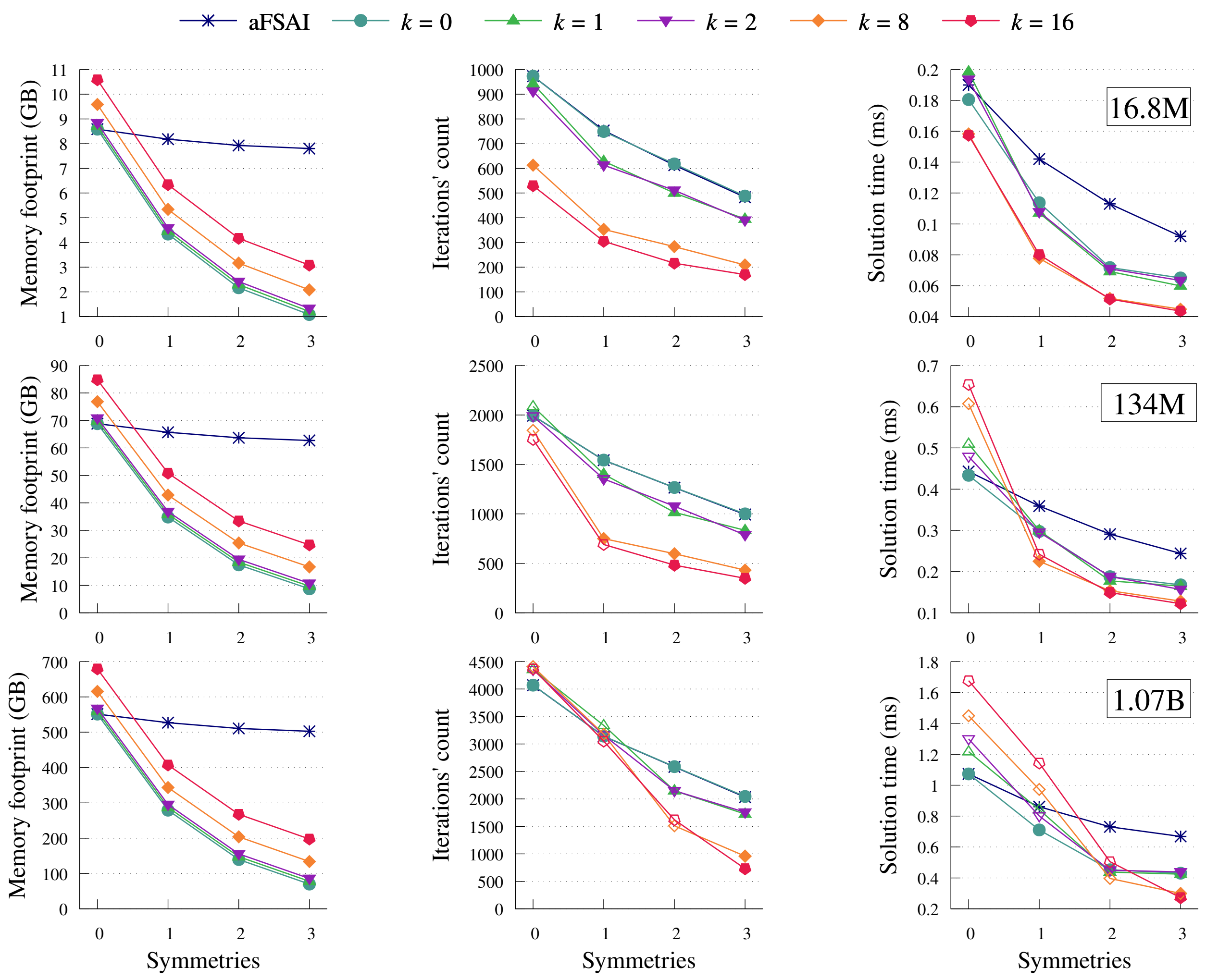


Figure 4: Memory footprint, iterations' count, and solution time normalised by the mesh size and CPU cores' number.

5 CONCLUSIONS

The fact that hardware's memory bandwidth tends to grow much slower than its peak performance has led to strongly memory-bound codes. Hence, increasing the arithmetic intensity of the Poisson solvers, which are the main bottleneck of incompressible CFD (and many other) simulations, is critical to develop efficient HPC codes. It is in this context that we have developed LRCFSAI(k), a more compute-intensive variant of the FSAI preconditioner.

In particular, we have recalled a strategy to exploit s spatial reflection symmetries for transforming the original Poisson equation into a set of 2^s decoupled subsystems. Given their close similarity, it has been possible to introduce an extra level of approximation and apply the same preconditioner to all the subsystems. Surprisingly enough, this did not harm convergence considerably. In fact, it set the ground for applying relatively cheap but very effective low-rank corrections. A key feature of these corrections is that, thanks to being applied to each subsystem independently, they correspond to a correction on the entire system of 2^s times higher rank. Therefore, the more symmetries being exploited, the more effective the low-rank corrections are. Indeed, LRCFSAI(16) reaches up to 2.9x and 5.7x faster convergences compared to the standard aFSAI exploiting $s = 3$ and no symmetries, respectively.

Apart from allowing very effective low-rank corrections, exploiting symmetries and recycling the same aFSAI on all the subsystems allowed making its application considerably more compute-intensive by replacing the standard SpMV with SpMM. On the one hand, this made the FSAI's application up to 1.7x faster. On the other, it reduced considerably the memory footprint of LRCFSAI(k). For instance, its heaviest version considered, LRCFSAI(16), was 2.6x lighter than aFSAI.

The combination of all the aforementioned factors made our LRCFSAI(k) outperform the standard aFSAI by a factor of up to 4.4x. Remarkably enough, the strategy presented in this work is naturally extensible to virtually any preconditioner explicitly factorisable, *e.g.*, those based on incomplete factorisations. Then, the usage of

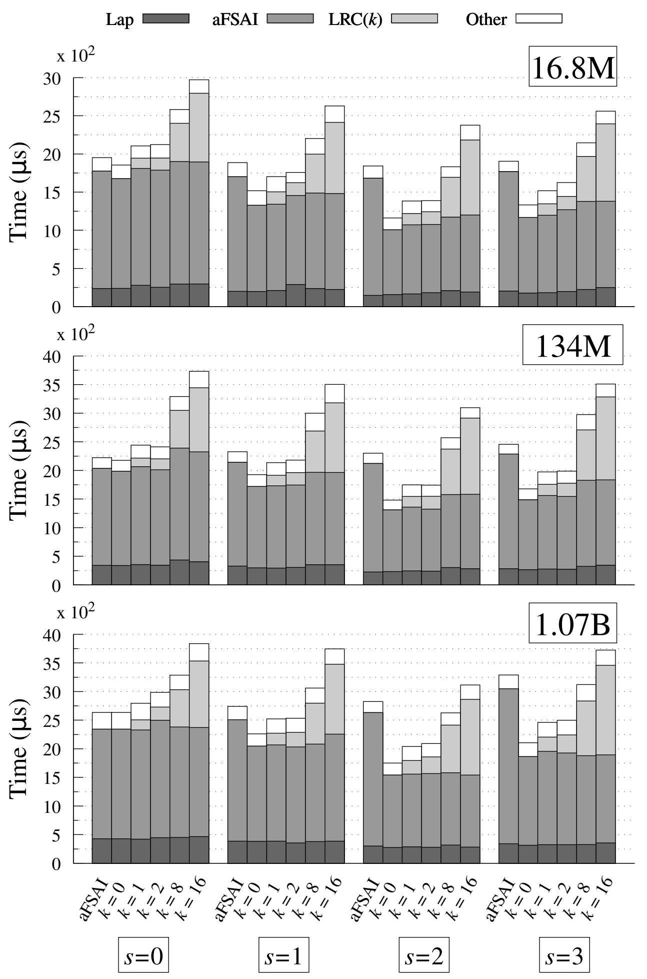


Figure 5: Decomposition of PCG iteration normalised by mesh size and CPU cores' number.

SpMM would be replaced by any other computational enhancement derived from having 2^s right-hand sides.

Immediate lines of work include switching from a non-restarted to a restarted Lanczos implementation, as well as improving our current low-rank corrections to make their application computationally cheaper, and attain even higher speed-ups. Additionally, we plan to port our current implementation of the LRCFSAI(k) to GPUs, and to evaluate its performance in large-scale CFD simulations of complex geometries. Finally, we want to study ways to apply similar strategies to preconditioners that cannot be explicitly factored, such as Algebraic Multigrid, which would be particularly benefited from using SpMM.

ACKNOWLEDGMENTS

We would like to thankfully acknowledge the team of M3E S.r.l., and most notably PhD. Matteo Frigo, for their support with Chronos throughout the implementation of LRCFSAI(k).

This work was financially supported by two competitive R+D projects: RETOtwin (PDC2021-120970-I00), given by MCIN/AEI/10.13039/501100011033 and European Union Next GenerationEU, and FusionCAT (001-P-001722), given by *Generalitat de Catalunya* RIS3CAT-FEDER. Àdel Alsalti-Baldellou has also been supported by the predoctoral grants DIN2018-010061 and 2019-DI-90, given by MCIN/AEI/10.13039/501100011033 and the Catalan Agency for Management of University and Research Grants (AGAUR), respectively.

REFERENCES

[1] A. Alsalti-Baldellou, X. Álvarez-Farré, A. Gorobets, and F.X. Trias. 2022. Efficient strategies for solving the variable Poisson equation with large contrasts in the coefficients. In vol. 273. CIMNE, 416–434.
[2] A. Alsalti-Baldellou, X. Álvarez-Farré, F.X. Trias, and A. Oliva. 2023. Exploiting spatial symmetries for solving Poisson's equation. *J. Comput. Phys.*, 486, 112133.
[3] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. 2015. Improving multifrontal methods by means of block low-rank representations. *SIAM Journal on Scientific Computing*, 37, A1451–A1474, 3.
[4] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, and S. Tomov. 2009. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180, 2526–2533, 12.
[5] F. Dabbagh, F.X. Trias, A. Gorobets, and A. Oliva. 2020. Flow topology dynamics in a three-dimensional phase space for turbulent Rayleigh-Bénard convection. *Physical Review Fluids*, 5, 024603.
[6] J. Frank and C. Vuik. 2001. On the construction of deflation-based preconditioners. *SIAM Journal on Scientific Computing*, 23, 442–462, 2.
[7] M. Frigo, G. Isotton, and C. Janna. 2021. Chronos web page. https://www.m3eweb.it/chronos. (2021).
[8] P. Ghysels, X. S. Li, F. H. Rouet, S. Williams, and A. Napov. 2016. An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM Journal on Scientific Computing*, 38, S358–S384, 5.
[9] A. Gorobets, F.X. Trias, R. Borrell, O. Lehmkuhl, and A. Oliva. 2011. Hybrid MPI+OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction. *Comput. Fluids*, 49, 1, 101–109.
[10] H. Ibeid, L. Olson, and W. D. Gropp. 2020. FFT, FMM, and multigrid on the road to exascale: performance challenges and opportunities. *Journal of Parallel and Distributed Computing*, 136, 63–74.
[11] G. Isotton, M. Frigo, N. Spiezia, and C. Janna. 2021. Chronos: a general purpose classical AMG solver for high performance computing. *SIAM J. Sci. Comput.*, 43, 5, C335–C357.
[12] G. Isotton, C. Janna, and M. Bernaschi. 2022. A GPU-accelerated adaptive FSAI preconditioner for massively parallel simulations. *The International Journal of High Performance Computing Applications*, 36, 2, 153–166.
[13] C. Janna and M. Ferronato. 2011. Adaptive pattern research for block FSAI preconditioning. *SIAM J. Sci. Comput.*, 33, 6, 3357–3380.
[14] I. E. Kaporin. 1994. New convergence results and preconditioning strategies for the conjugate gradient method. *Numer. Linear Algebra Appl.*, 1, 2, 179–210.
[15] L. Yu. Kolotilina and A. Yu. Yeremin. 1993. Factorized sparse approximate inverse preconditionings. I. Theory. *SIAM J. Matrix Anal. Appl.*, 14, 1, 45–58.
[16] S. Laut, M. Casas, and R. Borrell. 2022. Communication-aware sparse patterns for the factorized approximate inverse preconditioner. In ACM, 148–158. ISBN: 9781450391993.
[17] R. Li and Y. Saad. 2013. Divide and conquer low-rank preconditioners for symmetric matrices. *SIAM Journal on Scientific Computing*, 35, A2069–A2095, 4.
[18] R. Li and Y. Saad. 2017. Low-rank correction methods for algebraic domain decomposition preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 38, 807–828, 3.
[19] P. Nayak, T. Cojean, and H. Anzt. 2021. Evaluating asynchronous Schwarz solvers on GPUs. *The International Journal of High Performance Computing Applications*, 35, 226–236, 3.
[20] Y. Saad. 2011. *Numerical methods for large eigenvalue problems*. Society for Industry and Applied Mathematics.
[21] P. Sanan, S. M. Schnepp, and D. A. May. 2016. Pipelined, flexible Krylov subspace methods. *SIAM Journal on Scientific Computing*, 38, C441–C470, 5.
[22] O. Shishkina, A. Shishkin, and C. Wagner. 2009. Simulation of turbulent thermal convection in complicated domains. *Journal of Computational and Applied Mathematics*, 226, 336–344, 2.
[23] A. van der Sluis and H. A. van der Vorst. 1986. The rate of convergence of Conjugate Gradients. *Numerische Mathematik*, 48, 543–560, 5.