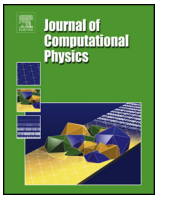
Journal of Computational Physics 486 (2023) 112133

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Exploiting spatial symmetries for solving Poisson's equation

Àdel Alsalti-Baldellou [a,b], Xavier Álvarez-Farré [a], F. Xavier Trias [a,*], Assensi Oliva [a]

[a] Heat and Mass Transfer Technological Center, Universitat Politècnica de Catalunya - BarcelonaTech, Carrer de Colom 11, 08222, Terrassa (Barcelona), Spain
[b] Termo Fluids SL, Carrer de Magí Colet 8, 08204, Sabadell (Barcelona), Spain

ARTICLE INFO

Article history:
Received 17 November 2022
Received in revised form 2 April 2023
Accepted 5 April 2023
Available online 11 April 2023

Keywords:
Poisson equation
Spatial symmetries
SpMM
Arithmetic intensity
Memory footprint
CFD

ABSTRACT

This paper presents a strategy to accelerate virtually any Poisson solver by taking advantage of s spatial reflection symmetries. More precisely, we have proved the existence of an inexpensive block diagonalisation that transforms the original Poisson equation into a set of 2^s fully decoupled subsystems then solved concurrently. This block diagonalisation is identical regardless of the mesh connectivity (structured or unstructured) and the geometric complexity of the problem, therefore applying to a wide range of academic and industrial configurations. In fact, it simplifies the task of discretising complex geometries since it only requires meshing a portion of the domain that is then mirrored implicitly by the symmetries' hyperplanes. Thus, the resulting meshes naturally inherit the exploited symmetries, and their memory footprint becomes 2^s times smaller. Thanks to the subsystems' better spectral properties, iterative solvers converge significantly faster. Additionally, imposing an adequate grid points' ordering allows reducing the operators' footprint and replacing the standard sparse matrix-vector products with the sparse matrix-matrix product, a higher arithmetic intensity kernel. As a result, matrix multiplications are accelerated, and massive simulations become more affordable. Finally, we include numerical experiments based on a turbulent flow simulation and making state-of-the-art solvers exploit a varying number of symmetries. On the one hand, algebraic multigrid and preconditioned Krylov subspace methods require up to 23% and 72% fewer iterations, resulting in up to 1.7x and 5.6x overall speedups, respectively. On the other, sparse direct solvers' memory footprint, setup and solution costs are reduced by up to 48%, 58% and 46%, respectively.

© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Poisson equation is present in many physical problems, usually arising from basic conservation principles. Hence, it is not surprising that Poisson solvers play a fundamental role in numerous areas of science and engineering, such as computational fluid dynamics (CFD), linear elasticity and magnetohydrodynamics. It is usually one of the most time-consuming and difficult to parallelise parts of scientific simulation codes. Consequently, the development of efficient and scalable Poisson solvers is of great interest. The reader is referred to [25,48,9] for a comprehensive review of most standard algorithms.

* Corresponding author.
E-mail addresses: adel.alsalti@upc.edu (A. Alsalti-Baldellou), xavier.alvarez.farre@upc.edu (X. Álvarez-Farré), francesc.xavier.trias@upc.edu (F.X. Trias), asensio.oliva@upc.edu (A. Oliva).

https://doi.org/10.1016/j.jcp.2023.112133
0021-9991/© 2023 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Basic theoretical properties such as the operation count or the convergence rate are not enough to evaluate the adequacy of a Poisson solver for a particular application and computing system. Indeed, computational aspects such as the memory footprint, the arithmetic intensity or the required communications can critically affect its feasibility and overall performance. For instance, despite their robustness, direct solvers have limited applicability due to their excessive memory requirements and are usually relegated to relatively small ill-conditioned subsystems [12,49,24].

In the last decades, the advent of highly scalable iterative solvers and preconditioners was motivated by the sustained growth of massively parallel high-performance computing (HPC) systems. This trend in HPC has also motivated communication avoiding and overlapping strategies for both iterative solvers [16] and preconditioners [26] to minimise the parallelisation overheads. Likewise, their sequential nature made traditional incomplete factorisation preconditioners [43,31] lose ground against other more recent alternatives, such as those based on sparse approximate inverses [33,10,22]. One of the major challenges the scientific computing community faces is that most of the algorithms have a rather low arithmetic intensity, *i.e.*, the ratio of flop to memory traffic in bytes. This, combined with the fact that the memory bandwidth tends to grow much slower than the peak performance, leads to strongly memory-bound codes. Approaches trying to remedy this are generally based on reducing memory traffic [1], solving multiple right-hand sides (RHSs) [35], using mixed-precision algorithms [8] or adapting more compute-intensive methods [7,30,17].

Regarding the applications and without loss of generality, our research work is targeted at transient CFD simulations of incompressible flows. In particular, direct numerical simulation (DNS) and large-eddy simulation (LES) of turbulent flows. The following aspects, which are relevant for this paper, are usually present in many other applications:

- The Poisson equation has to be solved repeatedly. Indeed, in DNS or LES the number of time-steps can easily reach $\mathcal{O}(10^6)$. While the RHSs vary, the coefficient matrix remains constant and a preprocessing stage with great computing demands may be acceptable.
- The solution obtained in the previous time-step(s) can be used as an initial guess to accelerate the iterative solvers' convergence.
- Wall-bounded flows and flows around internal obstacles are common in most applications. Therefore, arbitrary unstructured meshes may be required to solve all relevant turbulent scales near the walls. Hence, no *a priori* assumptions on the meshes connectivity are made.
- The boundary conditions for the Poisson equation are homogeneous Neumann regardless of the boundary conditions for the velocity and temperature fields [32].
- Spatial reflection symmetries are present in many industrial and scientific applications in at least one direction.

Regarding the last property, reflection symmetries are indeed present in many configurations ranging from simplified academic cases to more complex industrial devices, such as the car and fin-and-tube heat exchanger of Fig. 1. For instance, most vehicles generally present one symmetry [2,13]. Cases with two symmetries are also quite common, including jets [19], building [45] and urban simulations [27]. Finally, three symmetries can still be found on many industrial devices such as heat exchangers [42,46] and nuclear reactors [21,44].

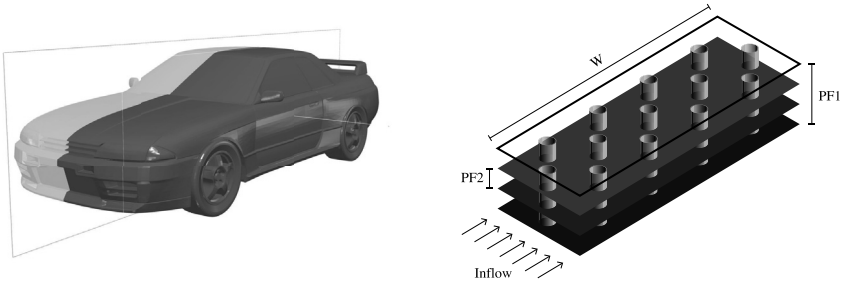


Fig. 1. Configurations presenting one (left) and three (right) reflection symmetries.

A few works taking advantage of a single reflection symmetry already exist [50,24]. They focus on canonical academic cases of high interest in which memory requirements often are the limiting factor. The present work generalises them, providing a strategy to accelerate virtually any Poisson solver by exploiting an arbitrary number of reflection symmetries. Such a strategy is identically applicable regardless of the mesh connectivity (structured or unstructured) and the geometric complexity of the problem. In fact, it simplifies the task of discretising complex geometries since it only requires to mesh a portion of the domain that is then mirrored implicitly by the symmetries' hyperplanes. Thus, the resulting meshes naturally inherit the exploited symmetries, and their footprint is reduced considerably. Finally, we present new theoretical results that allow mitigating several of the aforementioned computing challenges. In particular, we prove the existence and several relevant properties of an inexpensive change of basis that block diagonalises the discrete Laplace operator. By doing so, the original Poisson equation is decomposed into a set of fully decoupled subsystems then solved concurrently. On the one hand, their smaller size makes methods with excessive memory or computational requirements applicable

to larger cases. On the other, their better spectral properties make iterative solvers like Algebraic Multigrid (AMG) and Krylov subspace methods converge significantly faster. Proving that the block diagonalisation preserves matrix symmetry and negative-semidefiniteness allows employing specific solvers and preconditioners like Preconditioned Conjugate Gradient (PCG) and incomplete Cholesky factorisations. Additionally, proving that all the subsystems exhibit a regular structure, we can split them into a common part plus another subsystem-dependent whose size is, in practice, negligible. Such a splitting allows replacing the standard sparse matrix-vector product (`SpMV`) with the sparse matrix-matrix product (`SpMM`) [6,34,40], a kernel with significantly higher arithmetic intensity and, therefore, performance. We also show how to extend its use beyond the Laplace operator by imposing an adequate grid points' ordering. In this manner, the substantial accelerations and memory footprint reductions granted by `SpMM` can be exploited throughout the solver and simulation. Recalling that `SpMV` is the most computationally expensive operation on which iterative solvers generally rely, `SpMM`'s use accelerates their (fewer) iterations substantially, particularly benefiting those methods involving many matrix multiplications, such as AMG. Finally, imposing an adequate domain partitioning we avoid additional communications within the changes of basis, which become a negligible overhead. Regarding platform portability, it is naturally guaranteed since the whole algorithm relies on a small set of basic linear algebra operations included in standard math libraries.

The remainder of the paper is organised as follows. Section 2 proves the existence and several properties of the block diagonalisation, and gives the resulting algorithm. Section 3 presents the main computational improvements allowed by exploiting reflection symmetries. Finally, section 4 presents the results obtained in the numerical experiments, and section 5 gives the concluding remarks.

2. Mathematical framework

For the sake of clarity, section 2.1 presents Laplace operator's block diagonalisation in its simplest form, namely, applied to a single-symmetry 1D case, and section 2.2 extends it to its general form. Finally, section 2.3 presents the resulting algorithm.

2.1. Starting point: the single-symmetry 1D case

2.1.1. Spatial discretisation

Given a 1D domain with a single reflection symmetry, let us consider a spatial discretisation preserving it. Moreover, we will define its grid points ordering by first indexing the unknowns of one half and then those of the other. Finally, let us impose the same local ordering to both halves, *i.e.*, let us set any order to one of them and mirror it to the other by the symmetry's hyperplane (adding the corresponding offset).

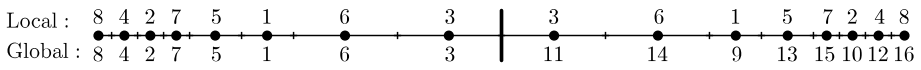


Fig. 2. Single-symmetry 1D mesh whose (random) local ordering is mirrored by the symmetry's hyperplane.

Such a discretisation leads to a mesh analogous to Fig. 2, and makes scalar fields and discrete operators exhibit a very favourable block structure. Namely, given a scalar field $\mathbf{b}$, we have that:

$$\mathbf{b} = \begin{pmatrix} \mathbf{b_1} \\ \mathbf{b_2} \end{pmatrix} \in \mathbb{R}^n, \tag{1}$$

being $n = 16$ the number of grid points and corresponding $\mathbf{b_1}, \mathbf{b_2} \in \mathbb{R}^{n/2}$ to the scalar field at each of the two subdomains. Moreover, the discrete Laplace operator reads:

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

where L is generally symmetric negative-semidefinite, and $\mathsf{L}_{\text{inn}}, \mathsf{L}_{\text{out}} \in \mathbb{R}^{n/2 \times n/2}$ account for the inner- and outer-subdomain couplings, respectively. This is true for virtually any existing discretisation method, such as finite volume (FVM), element and difference methods. For instance, a standard second-order FVM discretisation on the mesh of Fig. 2 leads to the following discrete Laplace operator in the form of eq. (2):

$$\mathsf{L}_{\text{inn}} = \begin{pmatrix} -H_1^{-1} & 0 & 0 & 0 & h_{1,5}^{-1} & h_{1,6}^{-1} & 0 & 0 \\ 0 & -H_2^{-1} & 0 & h_{2,4}^{-1} & 0 & 0 & h_{2,7}^{-1} & 0 \\ 0 & 0 & -H_3^{-1} & 0 & 0 & h_{3,6}^{-1} & 0 & 0 \\ 0 & h_{2,4}^{-1} & 0 & -H_4^{-1} & 0 & 0 & 0 & h_{4,8}^{-1} \\ h_{1,5}^{-1} & 0 & 0 & 0 & -H_5^{-1} & 0 & h_{5,7}^{-1} & 0 \\ h_{1,6}^{-1} & 0 & h_{3,6}^{-1} & 0 & 0 & -H_6^{-1} & 0 & 0 \\ 0 & h_{2,7}^{-1} & 0 & 0 & h_{5,7}^{-1} & 0 & -H_7^{-1} & 0 \\ 0 & 0 & 0 & h_{4,8}^{-1} & 0 & 0 & 0 & -H_8^{-1} \end{pmatrix} \tag{3}$$

and

$$(\mathsf{L}_{\text{out}})_{i,j} = \begin{cases} h_{3,11}^{-1}, & \text{if } i = j = 3 \\ 0, & \text{elsewhere} \end{cases}, \tag{4}$$

where $h_{i,j}$ stands for the distance between the ith and jth grid points, and H_i^{-1} for L's ith row off-diagonal sum, *e.g.*, $H_3^{-1} = h_{3,6}^{-1} + h_{3,11}^{-1}$.

2.1.2. Block diagonalisation

Let us define the following matrix:

$$\mathsf{P} := \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbb{I}_{n/2} & \mathbb{I}_{n/2} \\ \mathbb{I}_{n/2} & -\mathbb{I}_{n/2} \end{pmatrix} \in \mathbb{R}^{n \times n}. \tag{5}$$

Then, according to eq. (2), changing the basis of L by means of P results into:

$$\mathsf{P}\mathsf{L}\mathsf{P}^{-1} = \begin{pmatrix} \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}} & 0 \\ 0 & \mathsf{L}_{\text{inn}} - \mathsf{L}_{\text{out}} \end{pmatrix}, \tag{6}$$

decomposing the original discrete Laplace operator into two decoupled and half-sized subsystems.

2.2. Extension to the general case

2.2.1. Spatial discretisation

Given an N-dimensional domain with s reflection symmetries, let us consider a spatial discretisation analogous to Fig. 2. That is, a discretisation preserving all s symmetries and with a grid points' ordering that subsequently indexes each of the 2^s subdomains, imposing to each of them the same local ordering mirrored by the s symmetries' hyperplanes. Hereafter, we will name such discretisations *symmetry-aware*.

Regardless of being structured or unstructured, a *symmetry-aware* discretisation leads to a mesh consisting of 2^s subdomains. Analogously to eq. (2), the resulting discrete Laplace operator has the following form:

$$\mathsf{L} = \begin{pmatrix} \mathsf{L}_{1,1} & \mathsf{L}_{1,2} & \dots & \mathsf{L}_{1,2^s} \\ \mathsf{L}_{2,1} & \mathsf{L}_{2,2} & \dots & \mathsf{L}_{2,2^s} \\ \vdots & \vdots & \ddots & \vdots \\ \mathsf{L}_{2^s,1} & \mathsf{L}_{2^s,2} & \dots & \mathsf{L}_{2^s,2^s} \end{pmatrix} \in \mathbb{R}^{n \times n}, \tag{7}$$

where the diagonal blocks account for the inner-subdomain, and the off-diagonal blocks for the outer-subdomain couplings, belonging all to $\mathbb{R}^{n/2^s \times n/2^s}$. Clearly, every row consists of the same 2^s sub-blocks differently ordered according to the position of its corresponding subdomain within the full mesh. Such sub-blocks should be regarded as the *base mesh*'s couplings with itself and its $2^s - 1$ mirrored counterparts.

We aim to give an accurate description of L's structure to later propose a change of basis that vanishes all its outer couplings, making it block diagonal. To do so, given $k \in \{1, \dots, s\}$, let us define the auxiliary k-dimensional vector:

$$\vec{\pi}_k = (\pi_1, \dots, \pi_k) \in \{0, 1\}^k,$$

which will be used to identify the disjoint union of 2^{s-k} subdomains, *i.e.*, of 2^{s-k} different mirrorings of the *base mesh*. Each of its coordinates, π_i, equals either 0 or 1 depending on whether $\vec{\pi}_k$ lies on one half (tagged with 0) or the other (tagged with 1) of the full mesh with respect to the ith symmetry. Fig. 3 illustrates all the possible $\vec{\pi}_1$ and $\vec{\pi}_2$ when exploiting $s = 2$ symmetries on a 2D unstructured grid. Each of the mesh halves is identified either with the one-dimensional vector (0) or (1) and consists of two *base meshes* (*i.e.*, of two mesh quarters). Then, an extra component is added to identify the mesh quarters arising from (0): (0, 0), (0, 1), and (1): (1, 0), (1, 1).

For clarity, let us also consider a 3D mesh with $s = 3$ symmetries. Then, while $\vec{\pi}_1 \in \{(0), (1)\}$ corresponds to each mesh half arising from the first symmetry and consisting of four *base meshes* (*i.e.*, of four mesh eighths), $\vec{\pi}_2 \in$

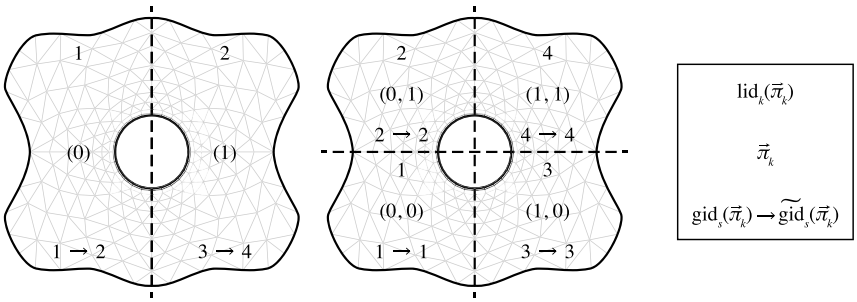


Fig. 3. Application of $\vec{\pi}_k$, $\mathrm{lid_k}\,(\vec{\pi}_k)$, $\mathrm{gid_s}\,(\vec{\pi}_k)$ and $\widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}_k)$ to a 2D mesh with $s = 2$ symmetries.

$\{(0,0),(0,1),(1,0),(1,1)\}$ and $\vec{\pi}_3 \in \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0),\ (1,0,1),(1,1,0),(1,1,1)\}$ correspond to each of the mesh quarters and eighths arising from the first two and three symmetries and consisting of two and one *base meshes*, respectively.

Besides that, given $k \in \{1,\dots,s\}$ and a subdomain $\vec{\pi}_k$, let us define the following row-major indexing functions:

$$\mathrm{lid_k}\,(\vec{\pi}_k) \coloneqq 1 + \sum_{i=1}^{k} \pi_i 2^{k-i} \in \left\{1,\dots,2^k\right\}, \tag{8}$$

$$\mathrm{gid_s}\,(\vec{\pi}_k) \coloneqq 1 + \sum_{i=1}^{k} \pi_i 2^{s-i} \in \left\{1,\dots,2^s\right\}, \tag{9}$$

$$\widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}_k) \coloneqq \mathrm{gid_s}\,(\vec{\pi}_k) + 2^{s-k} - 1 \in \left\{1,\dots,2^s\right\}. \tag{10}$$

While the first two correspond to a local and a global indexing of the different (in position and size) subdomains identified by $\vec{\pi}_k$, the third one satisfies that, given a larger $\vec{\pi}_{k_1}$, and a smaller $\vec{\pi}_{k_2}$ (thus $k_1 < k_2$), then $\vec{\pi}_{k_2}$ is contained within $\vec{\pi}_{k_1}$ if, and only if:

$$\mathrm{gid_s}\,(\vec{\pi}_{k_1}) \le \mathrm{gid_s}\,(\vec{\pi}_{k_2}) \le \widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}_{k_1}).$$

Fig. 3 illustrates the above definitions and properties by applying them to the 2D mesh with $s = 2$ symmetries.

Finally, given $k \le s$ and the subdomains $\vec{\pi}_k$ and $\vec{\pi}'_k$, let us define the following submatrix of L:

$$\mathsf{L}_{\vec{\pi}_k,\vec{\pi}'_k} \coloneqq \begin{pmatrix} \mathsf{L}_{\mathrm{gid_s}\,(\vec{\pi}_k),\mathrm{gid_s}\,(\vec{\pi}'_k)} & \cdots & \mathsf{L}_{\mathrm{gid_s}\,(\vec{\pi}_k),\widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}'_k)} \\ \vdots & \ddots & \vdots \\ \mathsf{L}_{\widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}_k),\mathrm{gid_s}\,(\vec{\pi}'_k)} & \cdots & \mathsf{L}_{\widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}_k),\widetilde{\mathrm{gid}}_\mathrm{s}\,(\vec{\pi}'_k)} \end{pmatrix} \in \mathbb{R}^{n/2^k \times n/2^k}, \tag{11}$$

which accounts for the couplings between the mesh's 2^kth fractions $\vec{\pi}_k$ and $\vec{\pi}'_k$. Despite being its general form somewhat cryptic, the definition of $\mathsf{L}_{\vec{\pi}_k,\vec{\pi}'_k}$ is very convenient. For instance, it allows us to denote the coupling between the two mesh halves, $\mathsf{L}_{(0),(1)}$, identically regardless of the number of symmetries. Indeed, for $s = 1$ we have that $\mathsf{L}_{(0),(1)} = \mathsf{L}_{1,2} \in \mathbb{R}^{n/2\times n/2}$, whereas for $s = 2$:

$$\mathsf{L}_{(0),(1)} = \begin{pmatrix} \mathsf{L}_{1,3} & \mathsf{L}_{1,4} \\ \mathsf{L}_{2,3} & \mathsf{L}_{2,4} \end{pmatrix} \in \mathbb{R}^{n/2\times n/2},$$

and for $s = 3$:

$$\mathsf{L}_{(0),(1)} = \begin{pmatrix} \mathsf{L}_{1,5} & \mathsf{L}_{1,6} & \mathsf{L}_{1,7} & \mathsf{L}_{1,8} \\ \mathsf{L}_{2,5} & \mathsf{L}_{2,6} & \mathsf{L}_{2,7} & \mathsf{L}_{2,8} \\ \mathsf{L}_{3,5} & \mathsf{L}_{3,6} & \mathsf{L}_{3,7} & \mathsf{L}_{3,8} \\ \mathsf{L}_{4,5} & \mathsf{L}_{4,6} & \mathsf{L}_{4,7} & \mathsf{L}_{4,8} \end{pmatrix} \in \mathbb{R}^{n/2\times n/2}.$$

Hence, combining eqs. (7) and (11), we can express L as:

$$\mathsf{L} = \begin{pmatrix} \mathsf{L}_{(0),(0)} & \mathsf{L}_{(0),(1)} \\ \mathsf{L}_{(1),(0)} & \mathsf{L}_{(1),(1)} \end{pmatrix}. \tag{12}$$

At this point, we are ready to describe L's structure accurately and generalise the strategy of section 2.1. Certainly, the use of a *symmetry-aware* discretisation ensures the following recursive block structure of L. For all $k \in \{1,\dots,s-1\}$ and subdomains $\vec{\pi}_k$ and $\vec{\pi}'_k$:

$$\mathsf{L}_{\vec{\pi}_k,\vec{\pi}'_k} = \begin{pmatrix} \mathsf{L}_{(\vec{\pi}_k,0),(\vec{\pi}'_k,0)} & \mathsf{L}_{(\vec{\pi}_k,0),(\vec{\pi}'_k,1)} \\ \mathsf{L}_{(\vec{\pi}_k,1),(\vec{\pi}'_k,0)} & \mathsf{L}_{(\vec{\pi}_k,1),(\vec{\pi}'_k,1)} \end{pmatrix}, \tag{13}$$

$$\mathsf{L}_{(\vec{\pi}_k,0),(\vec{\pi}'_k,0)} = \mathsf{L}_{(\vec{\pi}_k,1),(\vec{\pi}'_k,1)} \text{ and } \mathsf{L}_{(\vec{\pi}_k,0),(\vec{\pi}'_k,1)} = \mathsf{L}_{(\vec{\pi}_k,1),(\vec{\pi}'_k,0)}. \tag{14}$$

As hinted earlier, if we let the *base mesh* be $\vec{\pi}_s = \vec{0}$, it follows that L is only based on its 2^s couplings, $\mathsf{L}_{1,1},\dots,\mathsf{L}_{1,2^s}$, and the remaining $\mathsf{L}_{2,1},\dots,\mathsf{L}_{2,2^s},\dots,\mathsf{L}_{2^s,2^s}$ equal them according to eqs. (12) to (14). Hence, from now on and for the sake of simplicity, given $k \in \{1,\dots,s\}$ and $\vec{0},\vec{\pi}_k \in \{0,1\}^k$, we will denote:

$$\mathsf{L}_k \coloneqq \mathsf{L}_{1,k} \text{ and } \mathsf{L}_{\vec{\pi}_k} \coloneqq \mathsf{L}_{\vec{0},\vec{\pi}_k}.$$

2.2.2. Block diagonalisation

For all $k \in \{1,\dots,s\}$, let us define the following collection of change of basis matrices:

$$\mathsf{P}_k \coloneqq \prod_{i=1}^{k} \mathsf{P}^{(i)} \in \mathbb{R}^{n\times n}, \tag{15}$$

where:

$$\mathsf{P}^{(i)} \coloneqq \mathbb{I}_{2^{i-1}} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \mathbb{I}_{n/2^i} \in \mathbb{R}^{n\times n}. \tag{16}$$

Several immediate properties follow:

Proposition 1. *For all $k \geq 1$:*

$$\mathsf{P}_k = \frac{1}{\sqrt{2^k}} \left[\bigotimes_{i=1}^{k} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^k}.$$

Proof. Given the definition of $\mathsf{P}^{(1)}$, it holds for P_1. By induction on k, assuming it is satisfied by P_k and using the mixed-product property leads to:

$$\mathsf{P}_{k+1} = \mathsf{P}^{(k+1)}\mathsf{P}_k = \frac{1}{\sqrt{2^k}} \left[\bigotimes_{i=1}^{k} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \otimes \mathbb{I}_{n/2^{k+1}} =$$
$$= \frac{1}{\sqrt{2^{k+1}}} \left[\bigotimes_{i=1}^{k+1} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^{k+1}}. \quad \square$$

Proposition 2. *For all $k \geq 1$, P_k is symmetric orthogonal.*

Proof. Given $k \geq 1$, P_k is symmetric by construction. Additionally:

$$\mathsf{P}_k^{-1} = \left[\bigotimes_{i=1}^{k} \left(\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right)^{-1} \right] \otimes \mathbb{I}_{n/2^k} = \left[\bigotimes_{i=1}^{k} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^k} = \mathsf{P}_k. \quad \square$$

Proposition 3. *For all $k \geq 1$, the 2-norm is invariant under P_k (and P_k^{-1}) transformations, i.e.,*

$$\|\mathsf{P}_k\mathbf{x}\|_2 = \|\mathbf{x}\|_2 \ \ \forall \mathbf{x} \in \mathbb{R}^n.$$

Proof. As an immediate consequence of Proposition 2, given $k \geq 1$ and $\mathbf{x} \in \mathbb{R}^n$:

$$\|\mathsf{P}_k\mathbf{x}\|_2^2 = (\mathsf{P}_k\mathbf{x})^t\,\mathsf{P}_k\mathbf{x} = \mathbf{x}^t\mathbf{x} = \|\mathbf{x}\|_2^2. \quad \square$$

When it comes to the effect of changing the basis of L by P_k, the following theorem can be established:

Theorem 1. *Let L be a Laplace operator with s symmetries and discretised in a symmetry-aware manner, i.e., satisfying eqs. (12) to (14). Then, given $k \in \{1,\dots,s\}$:*

$$\mathsf{P}_k\mathsf{L}\mathsf{P}_k^{-1} = \sum_{\vec{\pi}_k\in\{0,1\}^k} \mathsf{D}_{\vec{\pi}_k}\otimes\mathsf{L}_{\vec{\pi}_k}, \tag{17}$$

with $\mathsf{D}_{\vec{\pi}_k} = \mathrm{diag}\left(\mathbf{d}_{lid_k(\vec{\pi}_k)}\right)$ *and* $\mathbf{d}_{lid_k(\vec{\pi}_k)}\in\{-1,1\}^{2^k}$ *equal to the* $lid_k(\vec{\pi}_k)$*-th row of* $\bigotimes_{i=1}^{k}\left(\begin{smallmatrix}1&1\\1&-1\end{smallmatrix}\right)$.

Proof. Let us prove it by induction on k. For $k=1$, it follows from eq. (12):

$$\mathsf{P}_1\mathsf{L}\mathsf{P}_1^{-1} = \mathbb{I}_2\otimes\mathsf{L}_{(0)} + \begin{pmatrix}1&0\\0&-1\end{pmatrix}\otimes\mathsf{L}_{(1)}.$$

Then, assuming:

$$\mathsf{P}_k\mathsf{L}\mathsf{P}_k^{-1} = \sum_{\vec{\pi}_k\in\{0,1\}^k} \mathsf{D}_{\vec{\pi}_k}\otimes\mathsf{L}_{\vec{\pi}_k},$$

and recalling Proposition 2 and eqs. (13) and (14) leads to:

$$\begin{aligned}\mathsf{P}_{k+1}\mathsf{L}\mathsf{P}_{k+1}^{-1} &= \mathsf{P}^{(k+1)}\left(\sum_{\vec{\pi}_k\in\{0,1\}^k}\mathsf{D}_{\vec{\pi}_k}\otimes\left(\mathbb{I}_2\otimes\mathsf{L}_{(\vec{\pi}_k,0)} + \begin{pmatrix}0&1\\1&0\end{pmatrix}\otimes\mathsf{L}_{(\vec{\pi}_k,1)}\right)\right)\mathsf{P}^{(k+1)} = \\ &= \sum_{\vec{\pi}_k\in\{0,1\}^k}\left(\mathsf{D}_{\vec{\pi}_k}\otimes\mathbb{I}_2\otimes\mathsf{L}_{(\vec{\pi}_k,0)} + \mathsf{D}_{\vec{\pi}_k}\otimes\begin{pmatrix}1&0\\0&-1\end{pmatrix}\otimes\mathsf{L}_{(\vec{\pi}_k,1)}\right).\end{aligned}$$

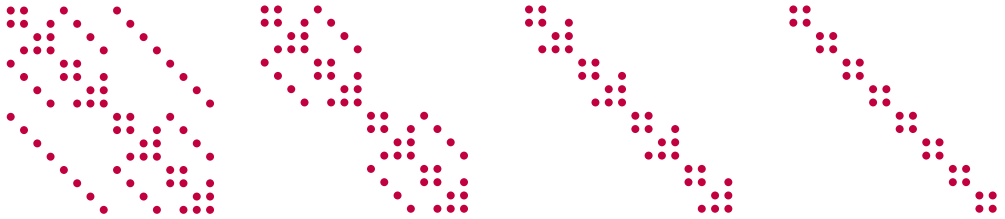
By the induction hypothesis, $\mathsf{D}_{\vec{\pi}_k}$ is a diagonal matrix whose diagonal equals the $\mathrm{lid}_k(\vec{\pi}_k)$-th row of $\bigotimes_{i=1}^{k}\left(\begin{smallmatrix}1&1\\1&-1\end{smallmatrix}\right)$. Given that the diagonals of $\mathbb{I}_2$ and $\left(\begin{smallmatrix}1&0\\0&-1\end{smallmatrix}\right)$ equal the first and second rows of $\left(\begin{smallmatrix}1&1\\1&-1\end{smallmatrix}\right)$, respectively, it follows that $\mathsf{D}_{\vec{\pi}_k}\otimes\mathbb{I}_2$ and $\mathsf{D}_{\vec{\pi}_k}\otimes\left(\begin{smallmatrix}1&0\\0&-1\end{smallmatrix}\right)$ correspond to the rows $2\,\mathrm{lid}_k(\vec{\pi}_k)-1$ and $2\,\mathrm{lid}_k(\vec{\pi}_k)$ of $\bigotimes_{i=1}^{k+1}\left(\begin{smallmatrix}1&1\\1&-1\end{smallmatrix}\right)$. Additionally:

$$\mathrm{lid}_{k+1}(\vec{\pi}_k,0) = 2\,\mathrm{lid}_k(\vec{\pi}_k)-1 \text{ and } \mathrm{lid}_{k+1}(\vec{\pi}_k,1) = 2\,\mathrm{lid}_k(\vec{\pi}_k)$$

and, therefore:

$$\mathsf{P}_{k+1}\mathsf{L}\mathsf{P}_{k+1}^{-1} = \sum_{\vec{\pi}_{k+1}\in\{0,1\}^{k+1}}\mathsf{D}_{\vec{\pi}_{k+1}}\otimes\mathsf{L}_{\vec{\pi}_{k+1}}. \quad \square$$

By virtue of Theorem 1, P_k removes $\sum_{i=1}^{k}2^{s-i}$ (out of 2^s-1) outer-subdomain couplings. Hence, the effect of P_s on L is to vanish all its off-diagonal blocks $\mathsf{L}_2,\ldots,\mathsf{L}_{2^s}$, making $\mathsf{P}_s\mathsf{L}\mathsf{P}_s^{-1}$ block diagonal and transforming the original system of size n into 2^s decoupled subsystems of size $n/2^s$. Fig. 4 illustrates this on a 3D discrete Laplace operator with three reflection symmetries. As predicted by Theorem 1, P_1, P_2 and P_3 transform L into 2, 4 and 8 subsystems of size $n/2$, $n/4$ and $n/8$, respectively.



(a) L (b) $\mathsf{P}_1\mathsf{L}\mathsf{P}_1^{-1}$ (c) $\mathsf{P}_2\mathsf{L}\mathsf{P}_2^{-1}$ (d) $\mathsf{P}_3\mathsf{L}\mathsf{P}_3^{-1}$

Fig. 4. Sparsity patterns of L, $\mathsf{P}_1\mathsf{L}\mathsf{P}_1^{-1}$, $\mathsf{P}_2\mathsf{L}\mathsf{P}_2^{-1}$ and $\mathsf{P}_3\mathsf{L}\mathsf{P}_3^{-1}$ based on a 3D structured mesh of size $2\times2\times4$ with $s=3$ symmetries.

Finally, several properties of the resulting subsystems can be established:

Corollary 1. *Let* L *be a Laplace operator with s symmetries, discretised in a symmetry-aware manner, i.e., satisfying eqs. (12) to (14), and such that* $\ker(\mathsf{L}) = \langle\mathbf{1}\rangle\subset\mathbb{R}^n$*. Then, for all* $k\in\{1,\ldots,s\}$*, the singularity of* L *is only inherited by the first (out of* 2^k*) decoupled subsystem of* $\mathsf{P}_k\mathsf{L}\mathsf{P}_k^{-1}$*:*

$$\hat{\mathsf{L}}_1 = \sum_{\vec{\pi}_k\in\{0,1\}^k}\mathsf{L}_{\vec{\pi}_k}\in\mathbb{R}^{n/2^k\times n/2^k},$$

whose kernel is also $\langle \mathbf{1} \rangle \subset \mathbb{R}^{n/2^k}$.

Proof. By Theorem 1, $\hat{\mathsf{L}} = \mathsf{P}_k \mathsf{L} \mathsf{P}_k^{-1}$ is block diagonal and $\hat{\mathsf{L}}_1 = \sum_{\vec{\pi}_k \in \{0,1\}^k} \mathsf{L}_{\vec{\pi}_k}$. Moreover, recalling that $\ker(\mathsf{L}) = \langle \mathbf{1} \rangle \subset \mathbb{R}^n$, it follows that $\mathbf{1} \in \ker(\hat{\mathsf{L}}_1) \subset \mathbb{R}^{n/2^k}$ and $\dim(\ker(\hat{\mathsf{L}}_1)) \geq 1$. Finally, as L and $\hat{\mathsf{L}}$ are similar, then $\mathrm{rank}(\mathsf{L}) = \mathrm{rank}(\hat{\mathsf{L}})$ and $\hat{\mathsf{L}}$'s subsystems, $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^k}$, satisfy:

$$\dim\left(\ker\left(\hat{\mathsf{L}}_i\right)\right) = \begin{cases} 1, & \text{if } i = 1 \\ 0, & \text{elsewhere.} \end{cases} \quad \square$$

Proposition 4. *For all* $k \geq 1$, *if* L *is symmetric negative-(semi)definite, then so is* $\mathsf{P}_k \mathsf{L} \mathsf{P}_k^{-1}$.

Proof. Given $k \geq 1$, recalling $\mathsf{P}_k^2 = \mathbb{I}_n$, we have that $\det\left(\mathsf{P}_k^2\right) = 1$. Thus, $\det(\mathsf{P}_k) \neq 0$ and $\ker(\mathsf{P}_k) = \langle \mathbf{0} \rangle$. Consequently, for all $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, $\exists \mathbf{y} = \mathsf{P}_k^{-1}\mathbf{x} = \mathsf{P}_k \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ and, if L is negative-(semi)definite, then so is $\hat{\mathsf{L}}$, as $x^t \hat{\mathsf{L}} x = y^t \mathsf{L} y$. $\square$

Proposition 5. *For all* $k \geq 1$, *let the eigenvectors and eigenvalues of* L *and* $\mathsf{P}_k \mathsf{L} \mathsf{P}_k^{-1}$ *be* $\{\mathbf{w}_i\}_{1 \leq i \leq n}, \{\hat{\mathbf{w}}_i\}_{1 \leq i \leq n} \subset \mathbb{R}^n$ *and* $\{\lambda_i\}_{1 \leq i \leq n}$, $\{\hat{\lambda}_i\}_{1 \leq i \leq n} \subset \mathbb{R}$, *respectively. Then,* $\hat{\lambda_i} = \lambda_i$ *and* $\hat{\mathbf{w}}_i = \mathsf{P}_k \mathbf{w}_i$ *for all* $i \in \{1, \ldots, n\}$.

Proof. It follows from the fact that $\left(\mathsf{P}_k \mathsf{L} \mathsf{P}_k^{-1}\right) \mathsf{P}_k \mathbf{w}_i = \mathsf{P}_k \mathsf{L} \mathbf{w}_i = \lambda_i \mathsf{P}_k \mathbf{w}_i$. $\square$

2.3. Resulting algorithm

The results derived so far lead to a very effective strategy to accelerate virtually any Poisson solver by exploiting spatial reflection symmetries. Roughly, given a discrete Poisson equation:

$$\mathsf{L}\mathbf{x} = \mathbf{b}, \tag{18}$$

it consists of changing the basis of L and solving the equivalent system:

$$\hat{\mathsf{L}}\hat{\mathbf{x}} = \hat{\mathbf{b}}, \tag{19}$$

where $\hat{\mathsf{L}} = \mathsf{P}_s \mathsf{L} \mathsf{P}_s^{-1}$, $\hat{\mathbf{x}} = \mathsf{P}_s \mathbf{x}$ and $\hat{\mathbf{b}} = \mathsf{P}_s \mathbf{b}$. By Theorem 1, eq. (19) is block diagonal and can be written as:

$$\begin{pmatrix} \hat{\mathsf{L}}_1 & & 0 \\ & \ddots & \\ 0 & & \hat{\mathsf{L}}_{2^s} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_{2^s} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{2^s} \end{pmatrix}, \tag{20}$$

allowing to solve concurrently the 2^s subsystems instead of the entire L.

Such an approach has multiple benefits, both from theoretical and computational points of view. In the first place, their smaller size translates into reducing the solvers' time complexity. While this is clear for direct methods, whose operation count is proportional to the size of the matrices inverted, iterative solvers' convergence rates highly depend on their spectral properties [48]. Remarkably enough, Proposition 5 ensures that the spectrums of L and $\hat{\mathsf{L}}$ are identical and, therefore, both lead to very similar convergence rates. Regarding $\hat{\mathsf{L}}$'s subsystems, let the eigenvectors and eigenvalues of $\hat{\mathsf{L}}$ and $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$ be $\{\hat{\mathbf{w}}_i\}_{1 \leq i \leq n} \subset \mathbb{R}^n$, $\{\hat{\mathbf{w}}_i^1\}_{1 \leq i \leq n/2^s}, \ldots, \{\hat{\mathbf{w}}_i^{2^s}\}_{1 \leq i \leq n/2^s} \subset \mathbb{R}^{n/2^s}$ and $\{\hat{\lambda}_i\}_{1 \leq i \leq n}, \{\hat{\lambda}_i^1\}_{1 \leq i \leq n/2^s}, \ldots, \{\hat{\lambda}_i^{2^s}\}_{1 \leq i \leq n/2^s} \subset \mathbb{R}$, respectively. Then, letting $\mathbf{e}_j \in \mathbb{R}^{2^s}$ denote the jth canonical basis vector, the following holds for all $i \in \{1, \ldots, n/2^s\}$ and $j \in \left\{1, \ldots, 2^s\right\}$:

$$\hat{\mathsf{L}}_j \hat{\mathbf{w}}_i^j = \hat{\lambda}_i^j \hat{\mathbf{w}}_i^j \implies \hat{\mathsf{L}}\left(\mathbf{e}_j \otimes \hat{\mathbf{w}}_i^j\right) = \hat{\lambda}_i^j \left(\mathbf{e}_j \otimes \hat{\mathbf{w}}_i^j\right)$$

and, therefore:

$$\{\hat{\mathbf{w}}_i\}_{1 \leq i \leq n} = \left\{\mathbf{e}_1 \otimes \hat{\mathbf{w}}_1^1, \ldots, \mathbf{e}_1 \otimes \hat{\mathbf{w}}_{n/2^s}^1, \ldots, \mathbf{e}_{2^s} \otimes \hat{\mathbf{w}}_{n/2^s}^{2^s}, \right\}, \tag{21}$$

$$\{\hat{\lambda}_i\}_{1 \leq i \leq n} = \{\hat{\lambda}_i^1\}_{1 \leq i \leq n/2^s} \cup \cdots \cup \{\hat{\lambda}_i^{2^s}\}_{1 \leq i \leq n/2^s}. \tag{22}$$

Equations (21) and (22) generally apply to block diagonal matrices and raise the question of how are L's eigenvalues distributed among $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$. A direct application of Gershgorin's circle theorem leads to poor bounds on their better-conditioning. Alternatively, by Theorem 1, $\hat{\mathsf{L}}_1 = \sum_{\vec{\pi}_k \in \{0,1\}^k} \mathsf{L}_{\vec{\pi}_k}$ corresponds to a Laplace operator built upon the *base mesh* identically to L. We could expect it to be significantly better-conditioned, as it results from shortening the largest length scale (the domain's size) while keeping constant the smallest one (the grid's size) [47]. However, numerical experiments

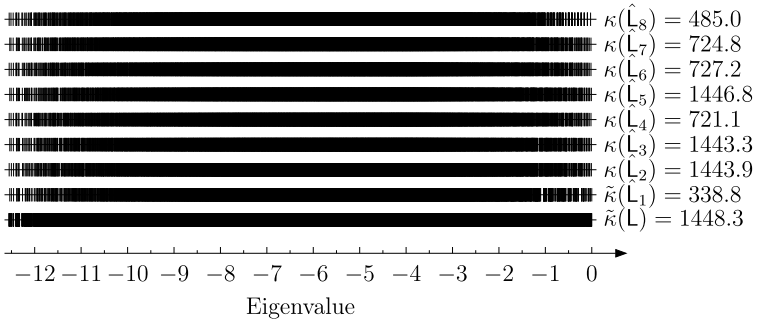


Fig. 5. Spectrums of L and $\hat{\mathsf{L}}_1, \dots, \hat{\mathsf{L}}_{2^s}$ on a $32 \times 32 \times 32$ structured mesh with $s = 3$ symmetries. κ and $\tilde{\kappa}$ denote the condition number and the ratio between the largest and the smallest (in magnitude) non-zero eigenvalues, respectively.

have showed that the spectrums of $\hat{\mathsf{L}}$ and $\hat{\mathsf{L}}_1, \dots, \hat{\mathsf{L}}_{2^s}$ are actually quite similar. Fig. 5 illustrates this on a 3D discrete Laplace operator exploiting three symmetries.

Despite not really improving the condition number, exploiting symmetries accelerates iterative solvers' convergence considerably. Namely, most iterative methods can be regarded as projection techniques that rely on selecting an approximate solution from a certain search space (*e.g.*, a Krylov subspace [54]) by making its related residual orthogonal to some other vector subspace [48]. Therefore, the higher their dimension is, the more accurate the approximate solution becomes. In this sense, given that a local iteration on $\hat{\mathsf{L}}_1, \dots, \hat{\mathsf{L}}_{2^s}$ produces a global search space of dimension 2^s times higher than that of a global iteration on L, solving the 2^s subsystems separately results into considerably better approximations. For instance, each PCG iteration minimises the A-norm of the error over a certain Krylov subspace [48]. Then, replacing a single global iteration with 2^s local iterations improves considerably such a minimisation. Similarly, applying a single deflation or augmentation [41,38] vector locally, *i.e.*, onto $\hat{\mathsf{L}}_1, \dots, \hat{\mathsf{L}}_{2^s}$, corresponds to applying 2^s vectors globally, *i.e.*, onto L.

On top of that, the superlinear convergence of certain well-known iterative methods agrees with a modest convergence of Ritz values towards the extreme eigenvalues of the coefficient matrix [52,55]. In addition, the more isolated these eigenvalues are, the faster the Ritz values converge [54,51]. Thence, solving $\hat{\mathsf{L}}_1, \dots, \hat{\mathsf{L}}_{2^s}$ separately ensures a two-fold improvement of the superlinear convergence. On the one hand, the rather homogeneous distribution of $\hat{\mathsf{L}}$'s eigenvalues among its subsystems makes the extreme ones more isolated, therefore accelerating Ritz values convergence. On the other, the fact that each subsystem presents fewer extreme eigenvalues makes superlinear convergence occur earlier and more intensely. The qualitative nature of the above arguments is due to the high dependence of the convergence rates on the iterative method employed. Additionally, the actual reductions in the iterations' count are determined by unrelated factors like the discretisation itself or the initial residual considered, which may be orthogonal to the eigenvectors associated with the most harmful eigenvalues.

Apart from accelerating convergence, exploiting symmetries allows for substantial computational improvements. On the one hand, section 3.1 will show how to accelerate the matrix multiplications by replacing the standard `SpMV` with the more compute-intensive `SpMM`. On the other, apart from the immediate reductions in the memory footprint derived from $\hat{\mathsf{L}}$'s block diagonal structure, section 3.2 will present a strategy to further lighten the solvers' memory requirements. Besides, the fact that $\hat{\mathsf{L}}_1, \dots, \hat{\mathsf{L}}_{2^s}$ are fully decoupled leaves a place for higher parallelism, for instance, in the construction and application of complex preconditioners. Indeed, $\hat{\mathsf{L}}$'s regular block structure may be used to apply low-rank approximations as in [36].

Algorithm 1 summarises the resulting strategy for making solvers exploit reflection symmetries. Regarding the procedure *setup* in line 1, the construction of P_s represents a relatively simple task through Proposition 1. On the other hand, to build $\hat{\mathsf{L}}$, we no longer need to discretise the entire domain but a 2^s times smaller fraction, the *base mesh*, that each of the symmetries' hyperplanes will implicitly mirror by applying Theorem 1 and eqs. (12) to (14) to its couplings, $\mathsf{L}_1, \dots, \mathsf{L}_{2^s}$.

Algorithm 1 Poisson solver exploiting s reflection symmetries.

Require: $s \geq 1$, $\mathsf{L}_{1,1}, \dots, \mathsf{L}_{1,2^s} \in \mathbb{R}^{n/2^s \times n/2^s}$ and $\mathbf{b} \in \mathrm{im}(\mathsf{L}) \subseteq \mathbb{R}^n$
1: **procedure** SETUP($s, \mathsf{L}_1, \dots, \mathsf{L}_{2^s}$)
2: $\quad \mathsf{P}_s = (1/\sqrt{2^s}) \left[\bigotimes_{i=1}^{s} \left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix} \right) \right] \otimes \mathbb{I}_{n/2^s}$ and $\hat{\mathsf{L}} = \mathsf{P}_s \mathsf{L} \mathsf{P}_s$
3: **end procedure**
4: **procedure** SOLVE($\mathbf{b}, \hat{\mathsf{L}}, \mathsf{P}_s$)
5: $\quad$ Transform $\hat{\mathbf{b}} = \mathsf{P}_s \mathbf{b}$
6: $\quad$ Decoupled solution of $\hat{\mathsf{L}}_i \hat{\mathbf{x}}_i = \hat{\mathbf{b}}_i$ for all $i \in \{1, \dots, 2^s\}$
7: $\quad$ Inverse transform $\mathbf{x} = \mathsf{P}_s \hat{\mathbf{x}}$
8: **end procedure**

When it comes to the procedure *solve* in line 4, it requires a forward and an inverse transforms of the RHS and solution vectors, respectively. Recalling that Proposition 2 allows computing them as a matrix multiplication by P_s, section 3.3 will present a domain partitioning making such products communication-free.

Finally, two practical observations about the concurrent solution of $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$ in line 6 can be made. Firstly, if an iterative method is employed, Proposition 3 provides a natural way to keep track of the original system's convergence in terms of the 2^s subsystems' local convergences. Indeed, letting $\mathbf{r} = \mathbf{b} - \mathsf{L}\mathbf{x}$ and $\hat{\mathbf{r}} = \hat{\mathbf{b}} - \hat{\mathsf{L}}\hat{\mathbf{x}}$, $\hat{\mathbf{r}} = \mathsf{P}_s\mathbf{r}$ holds, and by Proposition 3:

$$\|\mathbf{r}\|_2 = \|\hat{\mathbf{r}}\|_2 = \sqrt{\sum_{i=1}^{2^s} \|\hat{\mathbf{r}}_i\|_2^2}, \tag{23}$$

corresponding $\hat{\mathbf{r}}_i = \hat{\mathbf{b}}_i - \hat{\mathsf{L}}_i\hat{\mathbf{x}}_i$ to the ith subsystem's local residual. On the other hand, it is worth recalling that the Laplace operator's properties are usually preserved at discrete level, *i.e.*, L is symmetric negative-semidefinite and $\ker(\mathsf{L}) = \langle \mathbf{1} \rangle$ (see, for instance, eqs. (3) and (4)). Then, a common approach for specific solvers and preconditioners only applicable to symmetric positive-definite (SPD) matrices is to remove L's singularity and change the sign of the resulting system of equations. In this regard, Proposition 4 ensures that the block diagonalisation preserves matrix symmetry and definiteness, and Corollary 1 clarifies how to tackle $\hat{\mathsf{L}}$'s singularity.

3. Computational framework

3.1. Increasing the arithmetic intensity

Iterative methods generally rely on linear algebra operations, being (sparse) matrix-vector products the most computationally expensive. Algorithm 2 presents a pseudocode summarising the implementation of `SpMV`, the kernel typically used to compute them. Despite existing many works on developing architecture-specific sparse matrix storage formats and implementations [56,23], it is a memory-bound kernel with irregular memory accesses and unavoidable cache misses. As a result, its theoretically achievable performance falls to a small fraction of the computing systems' peak performances.

When exploiting symmetries, it is possible to replace `SpMV` with `SpMM`, a higher performance kernel also known as sparse matrix-dense matrix product or sparse matrix-multivector product [6]. It is summarised in Algorithm 3 and represents the multiplication of a sparse matrix by a set of dense vectors, which results in significantly greater data reuse. Indeed, in line 5 every vector reuses the coefficients of the sparse matrix.

`SpMM` naturally applies to matrices satisfying the following block structure:

$$\hat{\mathsf{A}} = \mathbb{I}_{2^s} \otimes \mathsf{A}, \tag{24}$$

where $\hat{\mathsf{A}} \in \mathbb{R}^{n \times n}$ and $\mathsf{A} \in \mathbb{R}^{n/2^s \times n/2^s}$. Then, thanks to the *symmetry-aware* discretisation, `SpMM` can be used on the products by virtually all the simulations' matrices with 2^s (sub)vectors, one per subdomain. In this sense, while operators like the discrete divergence satisfy eq. (24) exactly, others like the transformed Laplacian require a specialised implementation.

Algorithm 2 `SpMV` implementation using the standard CSR matrix format. Algebraically, given $\hat{\mathsf{A}} \in \mathbb{R}^{n \times m}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^m$ and $c \in \mathbb{R}$, it corresponds to $\mathbf{y} = c\hat{\mathsf{A}}\mathbf{x}$.

Require: Â, x, c
Ensure: y
1: **for** $i \leftarrow 1$ to n **do**
2: sum ← 0.0
3: **for** $j \leftarrow$ Â.rptr[i] to Â.rptr[$i+1$] **do**
4: sum ← sum + Â.coef[j] · x[Â.cidx[j]]
5: **end for**
6: y[i] ← c · sum
7: **end for**

Algorithm 3 `SpMM` implementation using the standard CSR matrix format. Algebraically, given $\mathsf{A} \in \mathbb{R}^{n/d \times m/d}$, $\mathbf{y} \in \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^d$, it corresponds to $\mathbf{y} = (\mathrm{diag}(\mathbf{c}) \otimes \mathsf{A})\mathbf{x}$.

Require: A, x, c
Ensure: y
1: **for** $i \leftarrow 1$ to n/d **do**
2: sum ← *zeros*(d)
3: **for** $j \leftarrow$ A.rptr[i] to A.rptr[$i+1$] **do**
4: **for** $k \leftarrow 1$ to d **do**
5: sum[k] ← sum[k] + A.coef[j] · x[A.cidx[j]][k]
6: **end for**
7: **end for**
8: **for** $k \leftarrow 1$ to d **do**
9: y[i][k] ← c[k] · sum[k]
10: **end for**
11: **end for**

Indeed, when it comes to $\hat{\mathsf{L}}$'s multiplications, a direct application of Theorem 1 allows using SpMM on each of the 2^s summands of eq. (17). Nevertheless, while the inner-subdomain couplings, L_1, resemble an $(n/2^s)$-sized Laplace operator, the outer ones, $\mathsf{L}_2, \dots, \mathsf{L}_{2^s}$, have very few non-zero entries, if any. Then, it makes sense to define the following splitting of $\hat{\mathsf{L}}$:

$$\hat{\mathsf{L}} = \mathbb{I}_{2^s} \otimes \hat{\mathsf{L}}_{\text{inn}} + \hat{\mathsf{L}}_{\text{out}}, \tag{25}$$

where, according to Theorem 1:

$$\hat{\mathsf{L}}_{\text{inn}} \coloneqq \mathsf{L}_{\vec{0}} \in \mathbb{R}^{n/2^s \times n/2^s} \text{ and } \hat{\mathsf{L}}_{\text{out}} \coloneqq \sum_{\vec{\pi}_s \neq \vec{0}} \mathsf{D}_{\vec{\pi}_s} \otimes \mathsf{L}_{\vec{\pi}_s} \in \mathbb{R}^{n \times n}. \tag{26}$$

Hence, by accumulating $\hat{\mathsf{L}}$'s outer-subdomain couplings into $\hat{\mathsf{L}}_{\text{out}}$, we can compute $\hat{\mathsf{L}}$'s multiplications as a combination of a single SpMM, SpMV and linear combination of vectors (axpy), reducing considerably the number of SpMM calls at a very low cost given the high sparsity of $\hat{\mathsf{L}}_{\text{out}}$. For instance, when dealing with compact stencils only coupling adjacent nodes, then at most s out of $2^s - 1$ outer-subdomain couplings are not null. In such cases, $\hat{\mathsf{L}}_{\text{out}}$ is diagonal (as in Fig. 4) and can be treated as a (sparse) vector, allowing to further accelerate $\hat{\mathsf{L}}$'s multiplications by replacing $\hat{\mathsf{L}}_{\text{out}}$'s SpMV with an element-wise product of vectors (axty).

Recalling that SpMV and SpMM are both memory-bound, the maximum speedup achievable by replacing an SpMV with an SpMM on matrices satisfying eq. (24) equals their arithmetic intensity ratio, $I_{\texttt{SpMM}}/I_{\texttt{SpMV}}$. According to Algorithm 2, considering double-precision flop and the standard CSR matrix format, the arithmetic intensity of SpMV reads:

$$I_{\texttt{SpMV}} = \frac{2\text{nnz}(\hat{\mathsf{A}}) + 1}{8\text{nnz}(\hat{\mathsf{A}}) + 4\text{nnz}(\hat{\mathsf{A}}) + 4(n+1) + 8n + 8m + 8}, \tag{27}$$

where $\text{nnz}(\hat{\mathsf{A}})$, n and m are the number of non-zeros, rows and columns in the matrix, respectively. Similarly, the arithmetic intensity of SpMM reads:

$$I_{\texttt{SpMM}} = \frac{(2\text{nnz}(\mathsf{A}) + 1) \cdot d}{8\text{nnz}(\mathsf{A}) + 4\text{nnz}(\mathsf{A}) + 4(n/d + 1) + (8n/d + 8m/d + 8) \cdot d}, \tag{28}$$

where $d = 2^s$ stands for the number of subvectors. It is clear, then, that the larger d is, the higher the speedup becomes and, on its limit for square matrices ($n = m$):

$$\lim_{d \to \infty} \frac{I_{\texttt{SpMM}}}{I_{\texttt{SpMV}}} \underset{n \gg 1}{\simeq} \frac{12\text{nnz}(\hat{\mathsf{A}})/n + 20}{16}.$$

Remarkably enough, being the speedup proportional to the average number of non-zeros per row, $\text{nnz}(\hat{\mathsf{A}})/n$, high-order schemes strengthen the benefits of SpMM. Additionally, the more matrix multiplications the solvers involve, the greater the accelerations provided by SpMM will be, particularly benefiting methods like AMG.

It should be noted that our approach to exploiting symmetries for accelerating matrix multiplications increases the communication to computation ratio. Indeed, while the computational work, in flops, and the memory traffic, in bytes, are typically functions of the mesh size, the communication traffic depends on the mesh boundaries resulting from the workload distribution. Because of this, the communication to computation ratio highly depends on the mesh size per process, and distributing a smaller *base mesh* increases the communication overheads, reducing the room for communication-hiding strategies. Although this had no impact on the studies conducted for this work, there is a limit where parallel scalability is no longer possible. However, exploring this limit is out of the scope of this paper.

3.2. Reducing the memory footprint

In certain cases, especially when using GPUs, the memory requirements become the limiting factor in the problems' size. In this regard, using a *symmetry-aware* discretisation effectively reduces the footprint of virtually all the matrices required by the simulations.

Regarding those operators satisfying eq. (24), SpMM no longer requires their entire matrices but their restriction to the *base mesh*, whose size is 2^s times smaller. When it comes to $\hat{\mathsf{L}}$, if we recall eq. (25) to replace SpMV with a combination of SpMM, SpMV and axpy, $\hat{\mathsf{L}}$'s memory footprint reduces to that of $\hat{\mathsf{L}}_{\text{inn}}$ and $\hat{\mathsf{L}}_{\text{out}}$. Considering $\hat{\mathsf{L}}_{\text{out}}$ negligible and since $\hat{\mathsf{L}}_{\text{inn}}$ resembles an $(n/2^s)$-sized Laplace operator, we can expect reductions by a factor of around 2^s in $\hat{\mathsf{L}}$'s footprint. For instance, a standard 7-point stencil produces a diagonal $\hat{\mathsf{L}}_{\text{out}}$ and $\hat{\mathsf{L}}$'s resulting size is $\mathcal{O}(n + 7n/2^s)$.

Table 1 summarises the reductions in the memory requirements of several standard solvers after exploiting a varying number of symmetries. The block diagonal structure of $\hat{\mathsf{L}}$ makes dense direct solvers 2^s times lighter, and reduces the footprint of sparse solvers and preconditioners, such as approximate inverses or incomplete factorisations, up to the same factor (in accordance with their sparsity patterns).

Table 1
Relative memory requirements of several standard solvers after exploiting s reflection symmetries. Results obtained considering that, apart from the coefficient matrix, CG and GMRES(4) require 3 and 10 vectors, respectively. $\hat{\mathsf{L}}$ is assumed to arise from a 7-point stencil, $\mathcal{O}(n+7n/2^s)$, AMG's memory footprint is simplified to that of its matrices, $\mathcal{O}(n/2^s)$, and direct solvers are assumed dense, $\mathcal{O}(2^s(n/2^s)^2)$.

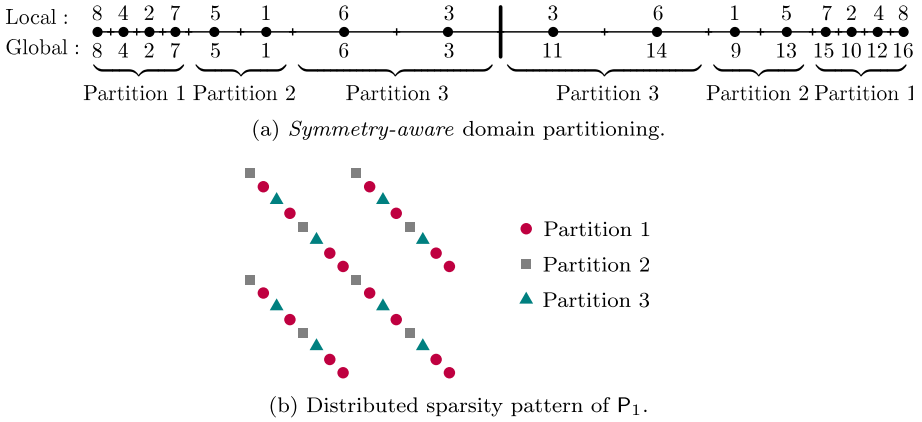
s	CG	GMRES(4)	AMG	Dense direct solvers
0	1.00	1.00	1.00	1.00
1	0.68	0.85	0.50	0.50
2	0.52	0.75	0.25	0.25
3	0.44	0.70	0.13	0.13

As a result, exploiting symmetries reduces the simulation's setup costs and memory requirements along with the sizes of their matrices, making massive simulations considerably more affordable, and methods with excessive memory or computational requirements applicable to larger cases.

3.3. Communication-free transforms

In distributed parallel processing, kernels involving matrix multiplications usually require communications. Roughly, a non-zero a_{ij} represents a coupling between the ith and jth elements of the output and input vectors, and requires a data exchange whenever they are located in different memory spaces.

By the definition of P_k given in eqs. (15) and (16), its ith row's off-diagonal non-zeros are located at the columns corresponding to ith grid point's mirrored counterparts (by each of the first k symmetries). For instance, recalling the single-symmetry 1D mesh of section 2.1 and according to Fig. 6, the 2nd row of P_1 is coupled with its 10th column, whose local index is also 2.



(a) *Symmetry-aware* domain partitioning.

(b) Distributed sparsity pattern of P_1.

Fig. 6. *Symmetry-aware* domain partitioning for communication-free transforms.

Then, since both transforms in lines 5 and 7 of Algorithm 1 are computed through `SpMV` calls by P_s, their data exchanges can be entirely avoided using a *symmetry-aware* domain partitioning. That is, a domain partitioning in which all elements with the same local index belong to the same partition. Fig. 6 illustrates this by defining a random *symmetry-aware* partitioning and giving the resulting rows distribution of P_1.

4. Numerical experiments

This section evaluates the benefits of exploiting spatial reflection symmetries in a realistic application. To do so, we have extracted a discrete Laplace operator, RHS and initial guess from a DNS of a fully-developed turbulent flow. Then, we have solved the resulting Poisson equation making several state-of-the-art solvers exploit a varying number of symmetries. All the numerical experiments rely on combined MPI and multithreaded parallelism, and have been conducted on the JFF cluster at the Heat and Mass Transfer Technological Center. Its non-uniform memory access (NUMA) nodes are equipped with two Intel Xeon 6230 CPUs (20 cores, 2.1 GHz, 27.5 MB L3 cache and 140 GB/s memory bandwidth) linked to 288 GB of RAM and interconnected through 7 GB/s FDR Infiniband.

The case considered corresponds to a buoyancy-driven flow within a cubic cavity heated from below. Fig. 7 illustrates the case and details its boundary conditions. The governing equations of such a configuration, generally known as Rayleigh-Bénard convection, are the Navier-Stokes and the temperature transport equations. Considering an incompressible Newtonian fluid and using the Boussinesq approximation to model the buoyancy effects, they read:

$$\partial_t \mathbf{v} + (\mathbf{v}\cdot\nabla)\,\mathbf{v} = \sqrt{\frac{\mathrm{Pr}}{\mathrm{Ra}}}\Delta\mathbf{v} - \nabla p - \theta\mathbf{z} \quad \text{and} \quad \nabla\cdot\mathbf{v} = 0, \tag{29}$$

$$\partial_t \theta + (\mathbf{v}\cdot\nabla)\,\theta = \frac{1}{\sqrt{\mathrm{RaPr}}}\Delta\theta; \tag{30}$$

being $\mathbf{v}$, θ and p the non-dimensional velocity, temperature and pressure fields, and corresponding Ra and Pr to the Rayleigh and Prandtl numbers, respectively.

The pressure-velocity coupling of eqs. (29) and (30) has been solved using a classical fractional step projection method [14]. Roughly, at each time iteration a predictor velocity is computed to, later, project it onto a divergence-free space by means of the pressure gradient, which is obtained by solving the following discrete Poisson equation:

$$\mathsf{L}\tilde{\mathbf{p}}^{i+1} = \mathsf{M}\mathbf{v}^*, \tag{31}$$

being $\tilde{\mathbf{p}}^{i+1} = \mathbf{p}^{i+1}\Delta t$ the discrete pseudo-pressure, and M and $\mathbf{v}^*$ the discrete divergence and predictor velocity, respectively (see [53] for further details about their discretisation). The time integration scheme is a second-order Adams-Bashforth and, to ensure its stability, a classical CFL condition has been imposed to the time-step, Δt. Further details about the algorithm and its implementation can be found in [3]. Henceforth, the initial guess and RHS vectors used in the numerical experiments are extracted once the flow is fully-developed, and equal to:

$$\mathbf{x}_0 = \tilde{\mathbf{p}}^i \text{ and } \mathbf{b} = \mathsf{M}\mathbf{v}^*,$$

being $\tilde{\mathbf{p}}^i$ previous time iteration's pseudo-pressure. The dimensionless parameters used are $\mathrm{Ra} = 10^9$ and $\mathrm{Pr} = 0.71$, which correspond to a turbulent flow within an air-filled cavity. Fig. 7 displays the temperature field of such a fully-developed flow.

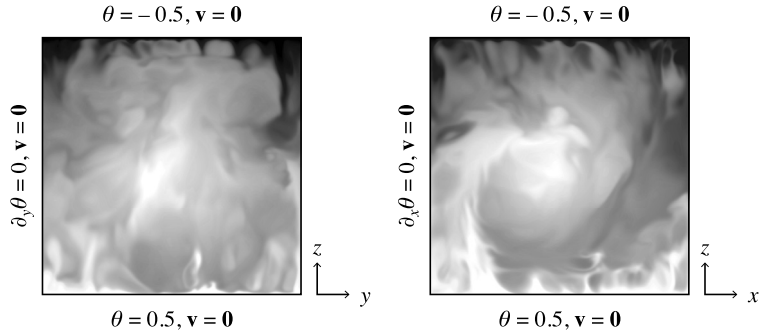


Fig. 7. Rayleigh-Bénard configuration displayed together with a fully-developed temperature field obtained on a 512^3 mesh ($\mathrm{Ra} = 10^9$, $\mathrm{Pr} = 0.71$).

Without loss of generality, the three test meshes considered in our experiments are structured and refined at the walls according to the following hyperbolic tangent function (analogously applied in the y- and z-directions):

$$x_i = \frac{1}{2}\left(1 + \frac{\tanh\left(\gamma_x\left(2\frac{(i-1)}{n_x} - 1\right)\right)}{\tanh\left(\gamma_x\right)}\right) \quad \forall i \in \{1, \ldots, n_x + 1\}, \tag{32}$$

which ensures having three symmetries. For the sake of simplicity, they are equally defined along each direction and correspond to $\gamma_x = \gamma_y = \gamma_z = 1.35$ and $n_x = n_y = n_z \in \{128, 256, 512\}$, thus being their sizes $n \in \{128^3, 256^3, 512^3\}$. This grid point's distribution follows the recommendation criteria proposed in [18] for DNS of Rayleigh-Bénard convection. In this regard, the estimated number of grid points to fully resolve all the flow scales at $\mathrm{Ra} = 10^9$ falls between our two finest meshes. Therefore, our experiments range from a relatively coarse-mesh simulation ($n = 128^3$) to a well-resolved DNS ($n = 512^3$).

As mentioned earlier, when exploiting s symmetries, we only need to discretise a $1/2^s$ fraction of the domain regardless of its geometric complexity and without having to worry about symmetrically dividing it. Fig. 8 summarises the domain portions meshed for $s \in \{0, 1, 2, 3\}$. Then, we can assemble virtually all the simulations' matrices from the *base mesh*'s couplings with itself and with its $2^s - 1$ mirrored counterparts, whose geometric information is inferred from the *base mesh*. For instance, we have used Theorem 1 and the classical second-order 7-point stencil to build $\hat{\mathsf{L}}$ from $\mathsf{L}_1, \ldots, \mathsf{L}_{2^s}$. Additionally, we have made $\hat{\mathsf{L}}$ SPD by recalling Proposition 4 and Corollary 1, making eq. (31) compatible with PCG and more robust since it is full-rank.

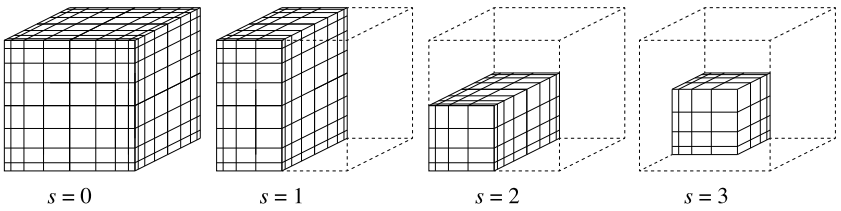


Fig. 8. Schema of the domain's portions meshed when exploiting s symmetries.

4.1. Naïve implementation

In order to evaluate the benefits of exploiting symmetries on state-of-the-art solvers, we have firstly tested a naïve implementation of Algorithm 1. More concretely, we have built $\hat{\mathsf{L}}$'s subsystems explicitly, and inverted them through sequential calls to external libraries. Despite not taking advantage of SpMM and coming with the overhead of requiring extra function calls, this approach has allowed testing our strategy on a wide variety of methods. Additionally, it has confirmed that even such a naïve implementation results into substantial gains.

As it is well known, the excessive memory requirements and setup costs of sparse direct solvers like those included in MUMPS [4,5], PARDISO [11] and SuperLU [37] make them impractical for large 3D cases. In this context, we have tested the impact of exploiting symmetries on solving $\hat{\mathsf{L}}$'s subsystems, $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$, through MUMPS' sparse direct solver, which relies on a block low-rank multifrontal factorisation.

Before analysing the results obtained, let us consider a general *symmetry-aware* (structured or unstructured) mesh with n grid points. Then, assuming a compact-stencil only coupling adjacent nodes, s symmetries would give rise to $\mathcal{O}(2sn^{2/3})$ outer-subdomain couplings later vanished by L's block diagonalisation and, consequently, $\mathrm{nnz}(\hat{\mathsf{L}}) \simeq \mathrm{nnz}(\mathsf{L}) - 2sn^{2/3}$.

Hence, the advantages of making sparse direct solvers exploit symmetries are two-fold. On the one hand, passing the decoupled subsystems instead of the entire L to MUMPS simplifies its reorderings and prevents it from adding extra fill-in outside $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$. On the other hand, $\hat{\mathsf{L}}$'s fewer non-zeros further alleviate the factorisation phase and, thanks to both combined factors, MUMPS' memory footprint, setup and solution costs are reduced by up to 48%, 58% and 46%, respectively. Figs. 9 and 10 present such reductions and illustrate how, thanks to exploiting symmetries, MUMPS became applicable to the 256^3 case. Remarkably enough, the lower performance obtained combining MPI and OpenMP parallelism made us switch to MPI-only executions. As a result, we faced substantial memory overheads that, despite being negligible for the two finer meshes, made MUMPS' memory footprint rather constant for the 128^3 test mesh.

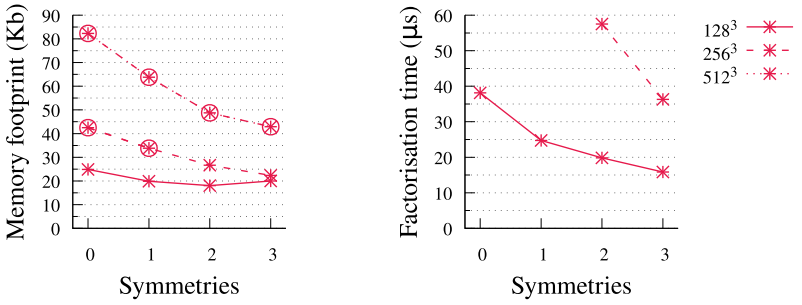


Fig. 9. MUMPS' factorisation time and memory footprint normalised by mesh size. Results obtained on the three test meshes. Circled values were predicted by MUMPS and not computed due to their excessive memory requirements.

Regarding the iterative solution of $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$, our goal now is to illustrate the faster convergences in which their better spectral properties result. To do so, we have solved $\hat{\mathsf{L}}_1, \ldots, \hat{\mathsf{L}}_{2^s}$ through many of the iterative solvers and preconditioners included in *hypre* [20]. They have been selected with the aim of providing a good overview of the impact of exploiting symmetries on a wide variety of methods, somehow compensating the lack of a general result proving their much faster convergences despite their slightly better conditioning. Therefore, our experiments include an algebraic multigrid (BoomerAMG [28]) used both as a solver and as a preconditioner for PCG and BICGSTAB, which have also been preconditioned with a sparse approximate inverse (ParaSails [15]) and a Parallel ILU (Euclid [29]).

Fig. 10 presents the resulting times and iterations' count required to solve eq. (31). Again, the overall accelerations are the result of multiple factors. Firstly, the subsystems' better spectral properties reduced substantially the iterations' count of all the methods. Regarding SAI- and ILU-preconditioned PCG and BICGSTAB, exploiting one, two and three symmetries lead to around 30%, 50% and 70% fewer iterations, respectively. Conversely, the very few V-cycles required by AMG to converge

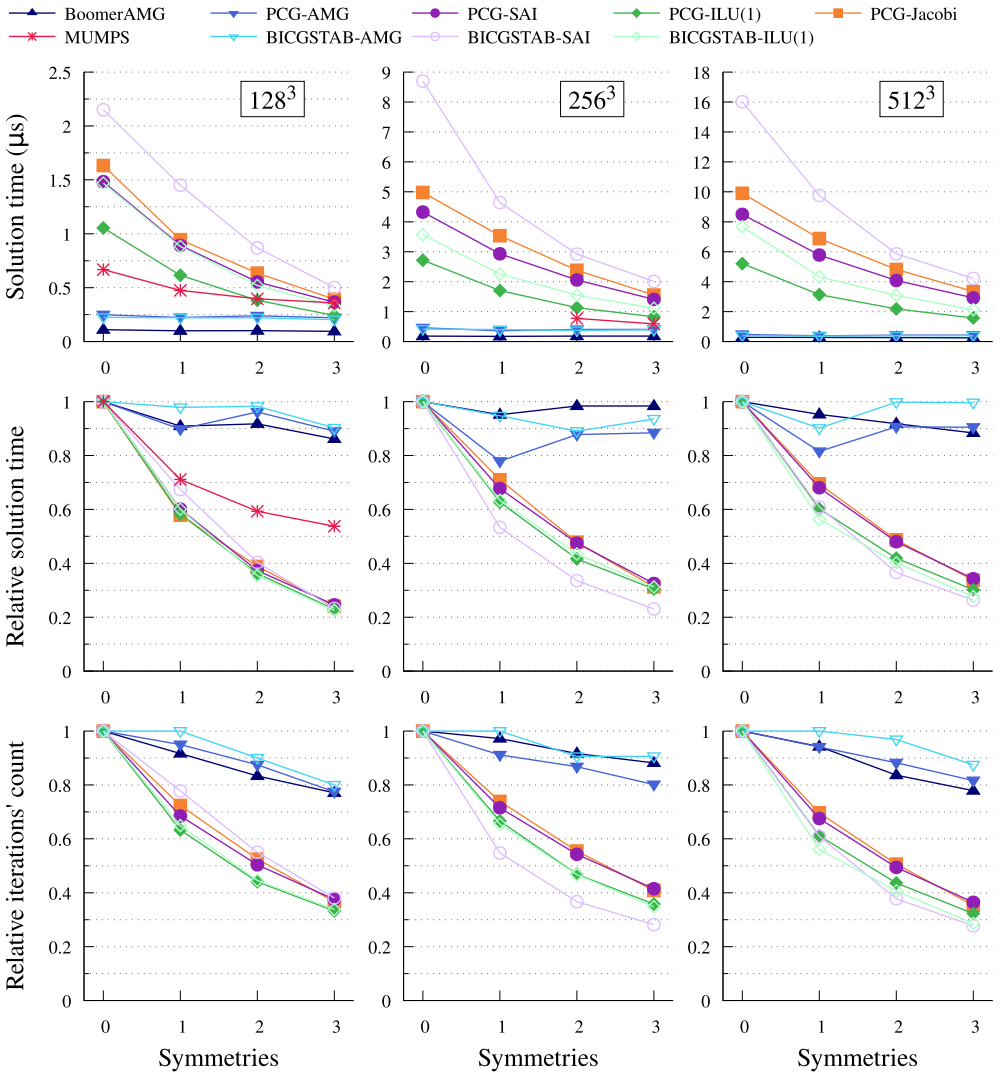


Fig. 10. Solution time normalised by mesh size (top) and L's solution time (middle). Bottom: iterations' relative reduction. First, second and third columns correspond to the 128^3, 256^3 and 512^3 test meshes, respectively. Convergence criterion: $\sqrt{2^s}\|\mathbf{r}_i\|_2/\|\mathbf{b}\|_2 \leq 10^{-9}$ for all $i \in \{1, \ldots, 2^s\}$.

made it reach, at most, around 23% iterations' reductions. Besides that, the higher sparsity of $\hat{\mathrm{L}}$ and the greater cache reuse derived from the subsystems' smaller size made BoomerAMG and AMG-preconditioned PCG and BICGSTAB up to 1.3x faster. For their part, SAI- and ILU-preconditioned PCG and BICGSTAB were up to 4.3x faster. Finally, it is worth remarking that, thanks to the *symmetry-aware* domain partitioning, the forward and inverse transforms' overhead was indeed negligible and represented at most 0.2% of the overall solution time.

4.2. SpMM-based implementation

The naïve implementation of virtually all iterative methods can be improved by replacing standard SpMV calls with the SpMM, which has a higher arithmetic intensity (AI). As was already discussed, thanks to the *symmetry-aware* discretisation, operators applying identically on the different subdomains satisfy eq. (24) and allow a direct application of SpMM. Additionally, Theorem 1 allows computing $\hat{\mathrm{L}}$'s multiplications as a combination of SpMM, axty and axpy. Fig. 11 presents the speedups resulting from replacing SpMV with our implementation of the SpMM on both kinds of products. Apart from reaching up to 2.5× speedups, SpMM also reduces the memory footprint of the matrices (and their setup costs) by a factor of 2^s.

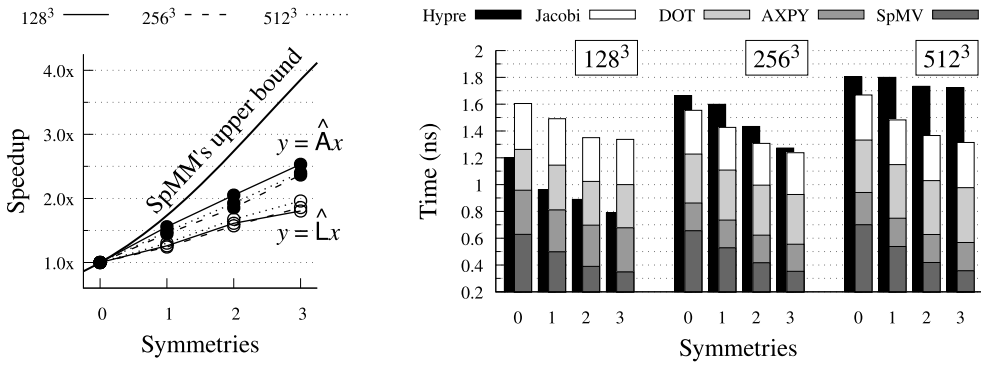


Fig. 11. Left: speedups in matrix products by $\hat{\mathsf{L}} = \mathbb{I}_{2^s} \otimes \hat{\mathsf{L}}_{\text{inn}} + \hat{\mathsf{L}}_{\text{out}}$ and $\hat{\mathsf{A}} = \mathbb{I}_{2^s} \otimes \mathsf{A}$ topped by SpMM's maximum speedup, $I_{\text{SpMM}}/I_{\text{SpMV}}$. Right: Comparison of time per *hypre* versus SpMM-based Jacobi-PCG iteration. Results normalised by the mesh sizes.

In order to illustrate the benefits of an eventual SpMM-based implementation, we have studied the relative weight within *hypre*'s solvers of those matrix multiplications compatible with SpMM. Then, by extrapolating the results of Fig. 11, we have calculated the overall speedups expected from their SpMM-based implementations. Fig. 12 summarises such speedups and confirms the potential of making methods relying on matrix multiplications exploit SpMM. For instance, AMG and Krylov subspace methods would reach up to 1.7x and 5.6x overall speedups, respectively. Remarkably enough, Fig. 11 shows how exploiting symmetries have also accelerated *hypre*'s Jacobi-PCG iterations (on the 128^3 and 256^3 meshes) despite not utilising SpMM. This is due to the much greater cache reuse derived from solving the smaller subsystems separately. However, given the current HPC trend of relying on massively parallel devices not counting with cache memory, we do not consider such accelerations an actual benefit of exploiting symmetries and take the 512^3 results as the standard.

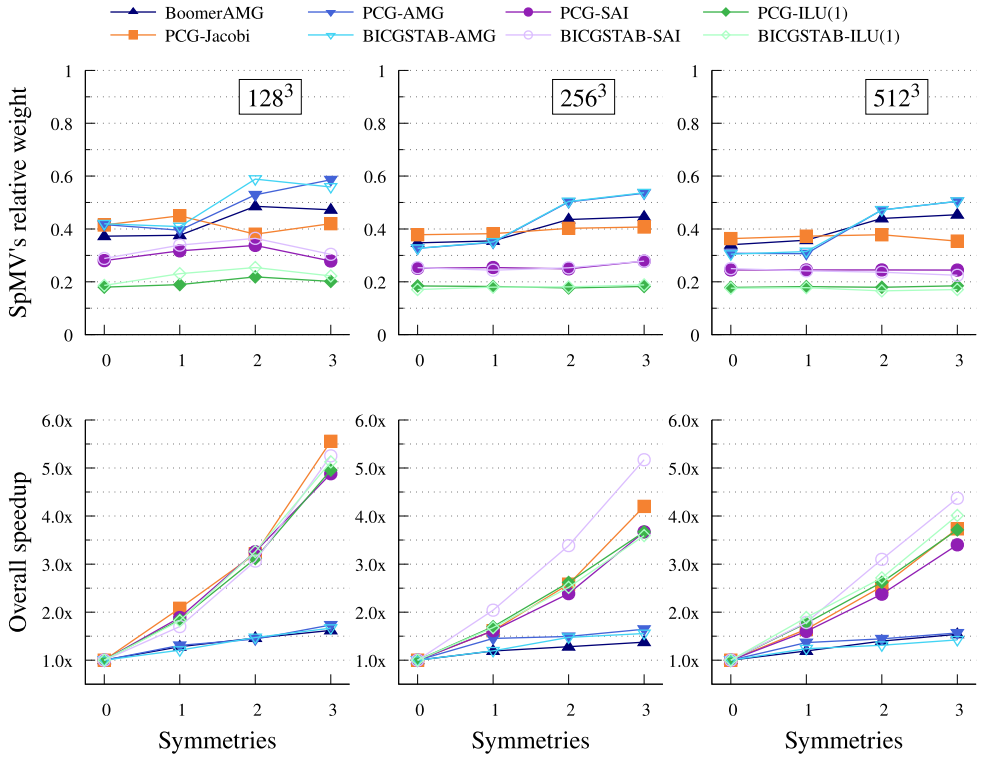


Fig. 12. Top: relative SpMV's weight per iteration. Bottom: Projected overall speedups resulting from the actual solution time (Fig. 10), SpMM's speedup (Fig. 11) and SpMV's relative weight (Top).

5. Conclusions

We have presented a strategy to accelerate virtually any Poisson solver by taking advantage of spatial reflection symmetries. More precisely, we have proved the existence and several relevant properties of an inexpensive change of basis used to block diagonalise the discrete Laplace operator. In this manner, we can replace the original Poisson equation with a set of fully decoupled subsystems then solved concurrently. This change of basis is identical regardless of the mesh connectivity (structured or unstructured) and the geometric complexity of the problem, therefore applying to a wide range of academic and industrial configurations.

This strategy has several advantages. Namely, the subsystems' better spectral properties make iterative solvers, such as Krylov subspace methods and AMG, converge significantly faster. For instance, numerical experiments based on a real turbulence flow simulation showed up to 3.6x faster convergences with ILU- and SAI-preconditioned PCG and BICGSTAB. Conversely, exploiting symmetries with *hypre*'s BoomerAMG and AMG-preconditioned PCG and BICGSTAB lead to 1.3x faster convergences. Furthermore, imposing an adequate ordering of the unknowns allowed replacing `SpMV` with `SpMM`, a more compute-intensive kernel whose performance is up to 2.5x faster. Since iterative solvers generally rely on basic linear algebra operations, among which matrix multiplications are the most computationally expensive, the use of `SpMM` would accelerate their (fewer) iterations considerably, making AMG and Krylov subspace methods attain up to 1.7x and 5.6x overall speedups, respectively.

The use of `SpMM` and the smaller subsystems' size allowed for considerable reductions in the solvers' footprint and setup costs, making methods with excessive memory or computational requirements applicable to larger cases. For instance, exploiting symmetries made MUMPS' sparse direct solver applicable to a 16.8M grid points mesh, being its memory footprint, setup and solution costs reduced by up to 55%, 85% and 60%, respectively.

Remarkably enough, exploiting symmetries has also simplified the discretisation of complex geometries, as it only requires meshing a portion of the physical domain. Although it was out of the scope of this work, we plan to exploit more than 3 symmetries on domains like extruded regular polygons [44,39] or arrays of repeated geometries [27]. The *symmetry-aware* discretisation allows extending `SpMM`'s use beyond the solvers, and we also want to exploit its higher performance and reduced memory footprint on most of the simulations' matrices, making massive problems considerably more affordable.

Finally, we would like to enhance our current implementation of the `SpMM` to approach its maximum achievable performance (see Fig. 11). Other future lines of work include seeking strategies to build complex preconditioners for a single subsystem (for instance, the worst-conditioned) and later extend it to the others through cheaper matrix manipulations. In this line, we are already working on an `SpMM`-based AMG.

CRediT authorship contribution statement

Àdel Alsalti-Baldellou: Conceptualization, Methodology, Software. **Xavier Álvarez-Farré:** Methodology, Software. **F. Xavier Trias:** Conceptualization, Methodology, Supervision. **Assensi Oliva:** Funding acquisition, Resources.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work has been financially supported by two competitive R+D projects: RETOtwin (PDC2021-120970-I00), given by MCIN/AEI/10.13039/501100011033 and European Union Next GenerationEU/PRTR, 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] M.F. Adams, J. Brown, M. Knepley, R. Samtaney, Segmental refinement: a multigrid technique for data locality, SIAM J. Sci. Comput. 38 (2016) C426–C440.

[2] D. Aljure, J. Calafell, A. Baez, A. Oliva, Flow over a realistic car model: wall modeled large eddy simulations assessment and unsteady effects, J. Wind Eng. Ind. Aerodyn. 174 (2018) 225–240.

[3] X. Álvarez-Farré, A. Gorobets, F.X. Trias, A hierarchical parallel implementation for heterogeneous computing. Application to algebra-based CFD simulations on hybrid supercomputers, Comput. Fluids 214 (2021) 104768.

[4] P.R. Amestoy, A. Buttari, J.Y. L'Excellent, T. Mary, Performance and scalability of the block low-rank multifrontal factorization on multicore architectures, ACM Trans. Math. Softw. 45 (2019) 1–26.
[5] P.R. Amestoy, I.S. Duff, J.Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM J. Matrix Anal. Appl. 23 (2001) 15–41.
[6] H. Anzt, S. Tomov, J. Dongarra, On the performance and energy efficiency of sparse linear algebra on GPUs, Int. J. High Perform. Comput. Appl. 31 (2017) 375–390.
[7] T. Askham, A. Cerfon, An adaptive fast multipole accelerated Poisson solver for complex geometries, J. Comput. Phys. 344 (2017) 1–22.
[8] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, S. Tomov, Accelerating scientific computations with mixed precision algorithms, Comput. Phys. Commun. 180 (2009) 2526–2533.
[9] M. Benzi, Preconditioning techniques for large linear systems: a survey, J. Comput. Phys. 182 (2002) 418–477.
[10] D. Bertaccini, S. Filippone, Sparse approximate inverse preconditioners on high performance GPU platforms, Comput. Math. Appl. 71 (2016) 693–711.
[11] M. Bollhöfer, O. Schenk, R. Janalik, S. Hamm, K. Gullapalli, State-of-the-art sparse direct solvers, in: Model. Simul. Sci. Eng. Technol., 2020, pp. 3–33.
[12] A. Buttari, M. Huber, P. Leleux, T. Mary, U. Rüde, B. Wohlmuth, Block low-rank single precision coarse grid solvers for extreme scale multigrid methods, Numer. Linear Algebra Appl. (2021) 1–15.
[13] H. Choi, J. Lee, H. Park, Aerodynamics of heavy vehicles, Annu. Rev. Fluid Mech. 46 (2014) 441–468.
[14] A.J. Chorin, Numerical solution of the Navier-Stokes equations, Math. Comput. 22 (1968) 745.
[15] E. Chow, A priori sparsity patterns for parallel sparse approximate inverse preconditioners, SIAM J. Sci. Comput. 21 (2000) 1804–1822.
[16] S. Cools, W. Vanroose, The communication-hiding pipelined BiCGstab method for the parallel solution of large unsymmetric linear systems, Parallel Comput. 65 (2017) 1–20.
[17] P. Costa, A FFT-accelerated multi-block finite-difference solver for massively parallel simulations of incompressible flows, Comput. Phys. Commun. 271 (2022) 108194.
[18] F. Dabbagh, F.X. Trias, A. Gorobets, A. Oliva, On the evolution of flow topology in turbulent Rayleigh-Bénard convection, Phys. Fluids 28 (2016) 115105.
[19] M. Duponcheel, Y. Bartosiewicz, Direct numerical simulation of turbulent heat transfer at low Prandtl numbers in planar impinging jets, Int. J. Heat Mass Transf. 173 (2021) 121179.
[20] R.D. Falgout, U.M. Yang, hypre: a library of high performance preconditioners, in: Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 2331, 2002, pp. 632–641.
[21] J. Fang, D.R. Shaver, A. Tomboulides, M. Min, P. Fischer, Y.H. Lan, R. Rahaman, P. Romano, S. Benhamadouche, Y.A. Hassan, A. Kraus, E. Merzari, Feasibility of full-core pin resolved CFD simulations of small modular reactor with momentum sources, Nucl. Eng. Des. 378 (2021) 111143.
[22] M. Ferronato, A. Franceschini, C. Janna, N. Castelletto, H.A. Tchelepi, A general preconditioning framework for coupled multiphysics problems with application to contact- and poro-mechanics, J. Comput. Phys. 398 (2019) 108887.
[23] S. Filippone, V. Cardellini, D. Barbieri, A. Fanfarillo, Sparse matrix-vector multiplication on GPGPUs, ACM Trans. Math. Softw. 43 (2017) 1–49.
[24] A. Gorobets, F.X. Trias, R. Borrell, O. Lehmkuhl, A. Oliva, Hybrid MPI+OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction, Comput. Fluids 49 (2011) 101–109.
[25] A. Greenbaum, Iterative Methods for Solving Linear Systems, Society for Industrial and Applied Mathematics, 1997.
[26] L. Grigori, S. Moufawad, Communication avoiding ILU0 preconditioner, SIAM J. Sci. Comput. 37 (2015) C217–C246.
[27] J. Hang, Y. Li, M. Sandberg, R. Buccolieri, S. Di Sabatino, The influence of building height variability on pollutant dispersion and pedestrian ventilation in idealized high-rise urban areas, Build. Environ. 56 (2012) 346–360.
[28] V.E. Henson, U.M. Yang, BoomerAMG: a parallel algebraic multigrid solver and preconditioner, Appl. Numer. Math. 41 (2002) 155–177.
[29] D. Hysom, A. Pothen, A scalable parallel algorithm for incomplete factor preconditioning, SIAM J. Sci. Comput. 22 (2001) 2194–2215.
[30] H. Ibeid, R. Yokota, J. Pestana, D. Keyes, Fast multipole preconditioners for sparse matrices arising from elliptic equations, Comput. Vis. Sci. 18 (2018) 213–229.
[31] D.S. Kershaw, The incomplete Cholesky—conjugate gradient method for the iterative solution of systems of linear equations, J. Comput. Phys. 26 (1978) 43–65.
[32] J. Kim, P. Moin, Application of a fractional-step method to incompressible Navier-Stokes equations, J. Comput. Phys. 59 (1985) 308–323.
[33] L.Y. Kolotilina, A.A. Nikishin, A.Y. Yeremin, Factorized sparse approximate inverse preconditionings. IV: Simple approaches to rising efficiency, Numer. Linear Algebra Appl. 6 (1999) 515–531.
[34] B.I. Krasnopolsky, An approach for accelerating incompressible turbulent flow simulations based on simultaneous modelling of multiple ensembles, Comput. Phys. Commun. 229 (2018) 8–19.
[35] B.I. Krasnopolsky, Revisiting performance of BiCGStab methods for solving systems with multiple right-hand sides, Comput. Math. Appl. 79 (2020) 2574–2597.
[36] R. Li, Y. Saad, low-rank correction methods for algebraic domain decomposition preconditioners, SIAM J. Matrix Anal. Appl. 38 (2017) 807–828.
[37] X.S. Li, An overview of SuperLU: algorithms, implementation, and user interface, ACM Trans. Math. Softw. 31 (2005) 302–325.
[38] J. van der Linden, T. Jönsthövel, A. Lukyanov, C. Vuik, The parallel subdomain-levelset deflation method in reservoir simulation, J. Comput. Phys. 304 (2016) 340–358.
[39] H.R. Liu, C.S. Ng, K.L. Chong, D. Lohse, R. Verzicco, An efficient phase-field method for turbulent multiphase flows, J. Comput. Phys. 446 (2021) 110659.
[40] X. Liu, E. Chow, K. Vaidyanathan, M. Smelyanskiy, Improving the performance of dynamical simulations via multiple right-hand sides, in: 2012 IEEE 26th Int. Parallel Distrib. Process. Symp., IEEE, 2012, pp. 36–47.
[41] R. Löhner, F. Mut, J.R. Cebral, R. Aubry, G. Houzeaux, Deflated preconditioned conjugate gradient solvers for the pressure-Poisson equation: extensions and improvements, Int. J. Numer. Methods Eng. 87 (2011) 2–14.
[42] B. Lotfi, M. Zeng, B. Sundén, Q. Wang, 3D numerical investigation of flow and heat transfer characteristics in smooth wavy fin-and-elliptical tube heat exchangers using new type vortex generators, Energy 73 (2014) 233–257.
[43] J.A. Meijerink, H.A. van der Vorst, An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix, Math. Comput. 31 (1977) 148–162.
[44] E. Merzari, A. Obabko, P. Fischer, M. Aufiero, Wall resolved large eddy simulation of reactor core flows with the spectral element method, Nucl. Eng. Des. 364 (2020) 110657.
[45] N. Morozova, F.X. Trias, R. Capdevila, C.D. Pérez-Segarra, A. Oliva, On the feasibility of affordable high-fidelity CFD simulations for indoor environment design and control, Build. Environ. 184 (2020) 107144.
[46] L. Paniagua, O. Lehmkuhl, C. Oliet, C.D. Pérez-Segarra, Large Eddy Simulations (LES) on the flow and heat transfer in a wall-bounded pin matrix, Numer. Heat Transf., Part B, Fundam. 65 (2014) 103–128.
[47] J. Ruano, A. Baez Vidal, J. Rigola, F.X. Trias, A new general method to compute dispersion errors on Cartesian stretched meshes for both linear and non-linear operators, Comput. Phys. Commun. 271 (2022) 108192.
[48] Y. Saad, Iterative Methods for Sparse Linear Systems, Society for Industrial and Applied Mathematics, 2003.
[49] P. Sanan, D.A. May, M. Bollhöfer, O. Schenk, Pragmatic solvers for 3D Stokes and elasticity problems with heterogeneous coefficients: evaluating modern incomplete LDL^t; preconditioners, Solid Earth 11 (2020) 2031–2045.

[50] O. Shishkina, A. Shishkin, C. Wagner, Simulation of turbulent thermal convection in complicated domains, J. Comput. Appl. Math. 226 (2009) 336–344.
[51] V. Simoncini, D.B. Szyld, On the occurrence of superlinear convergence of exact and inexact Krylov subspace methods, SIAM Rev. 47 (2005) 247–272.
[52] A. van der Sluis, H.A. van der Vorst, The rate of convergence of conjugate gradients, Numer. Math. 48 (1986) 543–560.
[53] F.X. Trias, O. Lehmkuhl, A. Oliva, C.D. Pérez-Segarra, R.W.C.P. Verstappen, Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids, J. Comput. Phys. 258 (2014) 246–267.
[54] H.A. van der Vorst, Iterative Krylov Methods for Large Linear Systems, vol. 56, Cambridge University Press, 2003.
[55] H.A. van der Vorst, C. Vuik, The superlinear convergence behaviour of GMRES, J. Comput. Appl. Math. 48 (1993) 327–341.
[56] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, J. Demmel, Optimization of sparse matrix-vector multiplication on emerging multicore platforms, in: Proc. 2007 ACM/IEEE Conf. Supercomput. - SC '07, ACM Press, New York, USA, 2007, p. 1.