Exploiting symmetries for preconditioning Poisson's equation in CFD simulations

Àdel Alsalti-Baldellou[1,2] Carlo Janna[3,4] Xavier Álvarez-Farré[5]
F. Xavier Trias[1]

[1]Heat and Mass Transfer Technological Center, Polytechnic University of Catalonia

[2]Termo Fluids SL,
http://www.termofluids.com/

[3]Department of Civil, Environmental and Architectural Engineering, University of Padova

[4]M3E s.r.l.,
http://www.m3eweb.it/

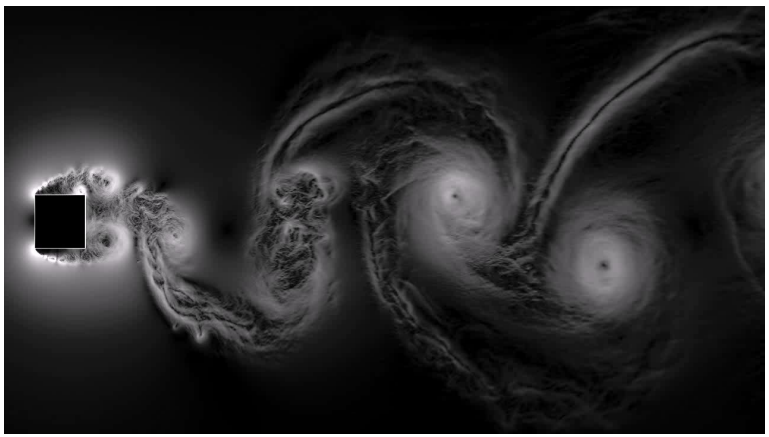
[5]High-Performance Computing Team, SURF

Platform for Advanced Scientific Computing Conference (PASC23)
June 26-28 2023 – Davos, Switzerland

Index

Context of the work

CFD applications – 1



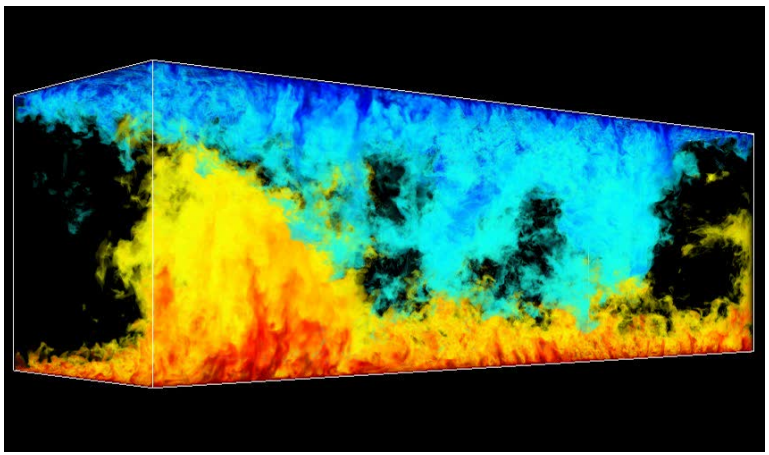


Figure: Simulation of flow around a square cylinder[1] and Rayleigh-Bénard convection[2].

[1] F.X. Trias et al. (2015). "Turbulent flow around a square cylinder at Reynolds number 22000: a DNS study" in *Computers and Fluids*.

[2] F. Dabbagh et al. (2017). "A priori study of subgrid-scale features in turbulent Rayleigh-Bénard convection" in *Physics of Fluids*.

CFD applications – 2

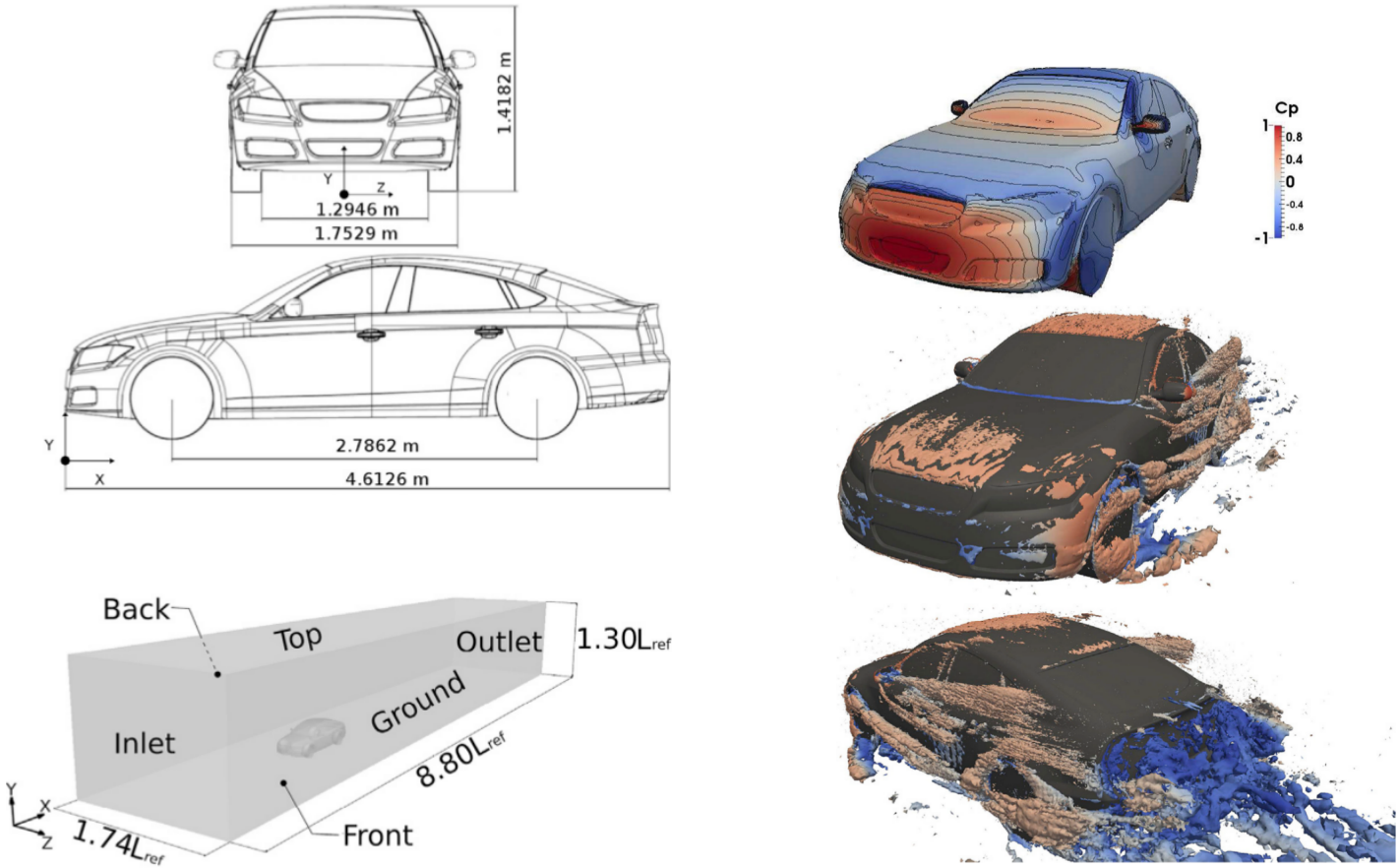


Figure: Simulation of turbulent flow over the DrivAer fastback vehicle model[3].

[3] D. E. Aljure et al. (2018). "Flow over a realistic car model: Wall modeled large eddy simulations assessment and unsteady effects" in *Journal of Wind Engineering and Industrial Aerodynamics*.

CFD applications – 3

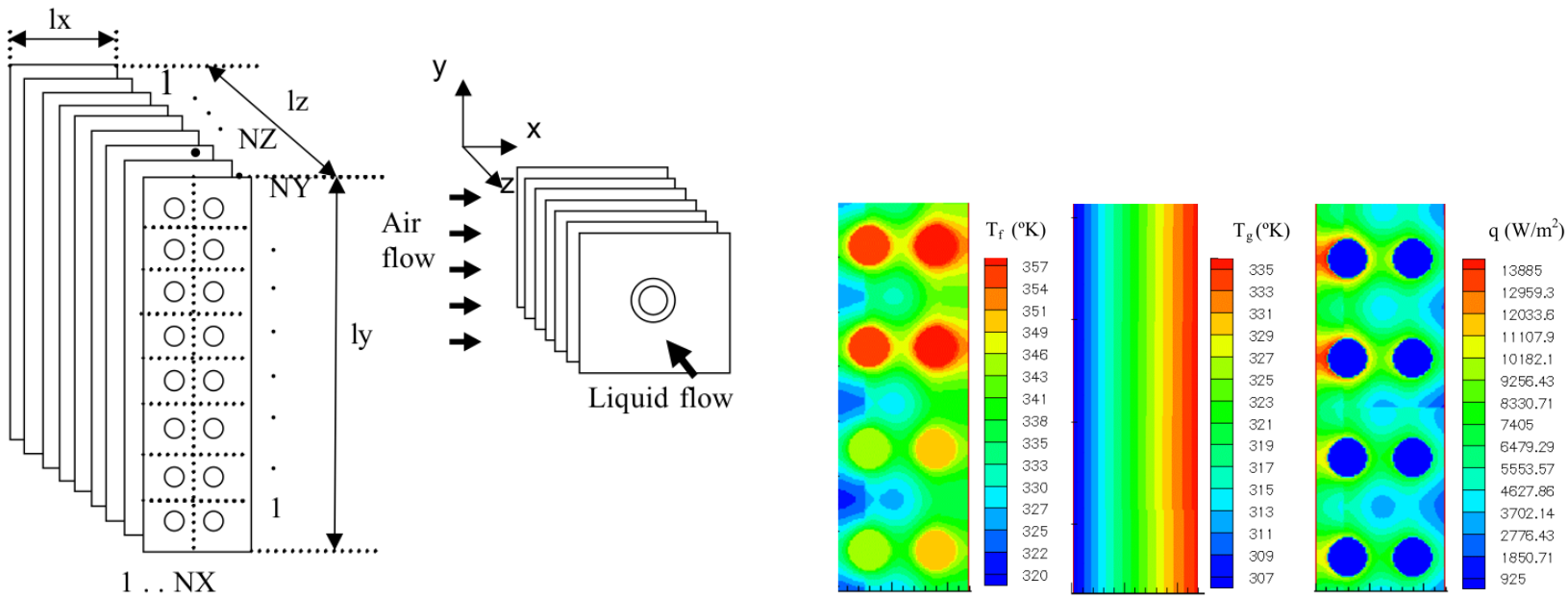


Figure: Simulation of brazed and expanded tube-fin heat exchangers[4].

[4] L. Paniagua et al. (2014). "Large Eddy Simulations (LES) on the Flow and Heat Transfer in a Wall-Bounded Pin Matrix" in *Numerical Heat Transfer, Part B: Fundamentals*.

CFD applications – 4

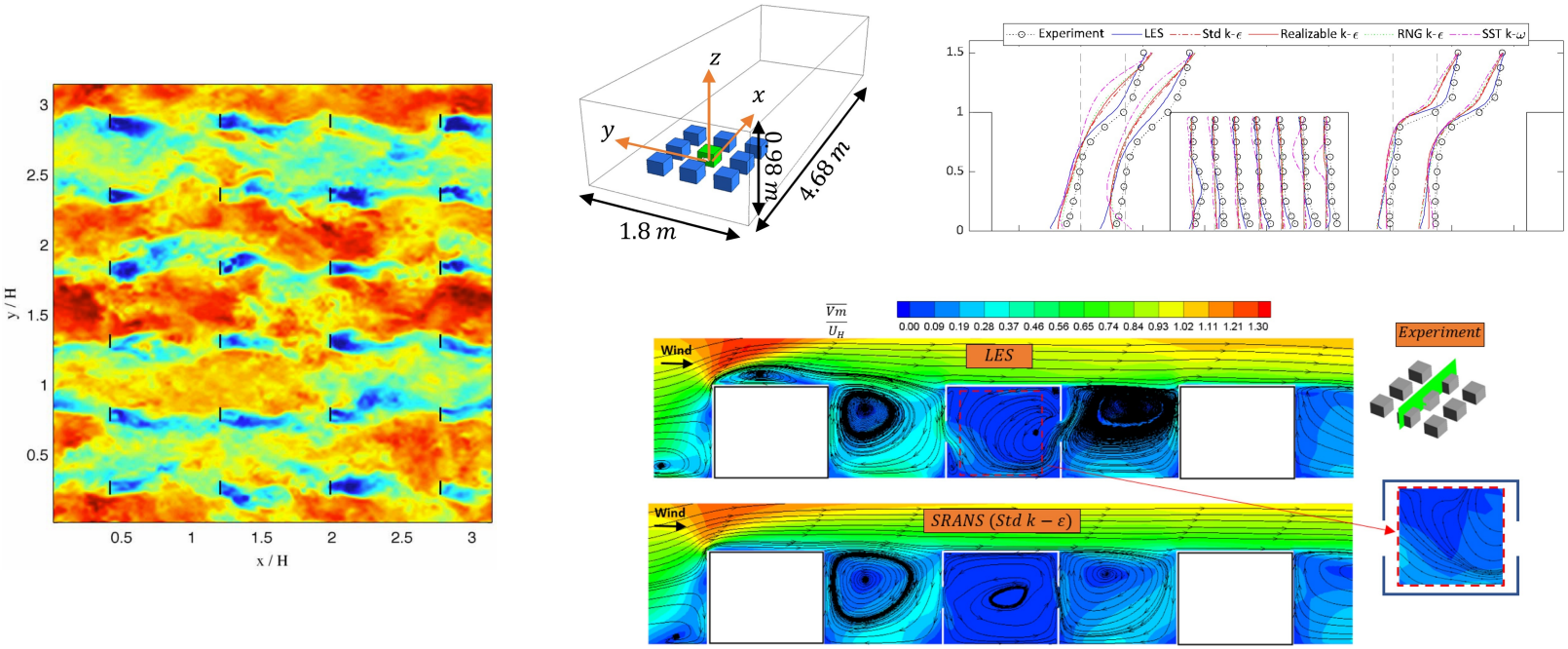


Figure: Simulation of wind plant[5] and array of "buildings"[6].

[5] M. Calaf et al. (2010). "Large eddy simulation study of fully developed wind-turbine array boundary layers" in *Physics of Fluids*.

[6] P. A. Mirzaei (2021). "CFD modeling of micro and urban climates: Problems to be solved in the new decade" in *Sustainable Cities and Society*.

Poisson's equation in incompressible CFD

Fractional Step Method (FSM)

1. Evaluate the auxiliar vector field $\mathbf{r}(\mathbf{v}^n) := -(\mathbf{v}\cdot\nabla)\mathbf{v} + \nu\Delta\mathbf{v}$
2. Evaluate the predictor velocity $\mathbf{v}^p := \mathbf{v}^n + \Delta t\left(\frac{3}{2}\mathbf{r}(\mathbf{v}^n) - \frac{1}{2}\mathbf{r}(\mathbf{v}^{n-1})\right)$
3. Obtain the pressure field by solving a **Poisson equation**:

$$\nabla\cdot\left(\frac{1}{\rho}\nabla p^{n+1}\right) = \frac{1}{\Delta t}\nabla\cdot\mathbf{v}^p$$

4. Obtain the new divergence-free velocity $\mathbf{v}^{n+1} = \mathbf{v}^p - \nabla p^{n+1}$

Poisson's equation in incompressible CFD

Fractional Step Method (FSM)

1. Evaluate the auxiliar vector field $\mathbf{r}(\mathbf{v}^n) := -(\mathbf{v}\cdot\nabla)\mathbf{v} + \nu\Delta\mathbf{v}$
2. Evaluate the predictor velocity $\mathbf{v}^p := \mathbf{v}^n + \Delta t\left(\frac{3}{2}\mathbf{r}(\mathbf{v}^n) - \frac{1}{2}\mathbf{r}(\mathbf{v}^{n-1})\right)$
3. Obtain the pressure field by solving a **Poisson equation**:

$$\nabla\cdot\left(\frac{1}{\rho}\nabla p^{n+1}\right) = \frac{1}{\Delta t}\nabla\cdot\mathbf{v}^p$$

4. Obtain the new divergence-free velocity $\mathbf{v}^{n+1} = \mathbf{v}^p - \nabla p^{n+1}$

Poisson's equation for incompressible single-phase flows

- Continuous:

$$\Delta p = \frac{\rho}{\Delta t}\nabla\cdot\mathbf{v}^p$$

Poisson's equation in incompressible CFD

Fractional Step Method (FSM)

1. Evaluate the auxiliar vector field $\mathbf{r}(\mathbf{v}^n) := -(\mathbf{v}\cdot\nabla)\mathbf{v} + \nu\Delta\mathbf{v}$
2. Evaluate the predictor velocity $\mathbf{v}^p := \mathbf{v}^n + \Delta t\left(\frac{3}{2}\mathbf{r}(\mathbf{v}^n) - \frac{1}{2}\mathbf{r}(\mathbf{v}^{n-1})\right)$
3. Obtain the pressure field by solving a **Poisson equation**:

$$\nabla\cdot\left(\frac{1}{\rho}\nabla p^{n+1}\right) = \frac{1}{\Delta t}\nabla\cdot\mathbf{v}^p$$

4. Obtain the new divergence-free velocity $\mathbf{v}^{n+1} = \mathbf{v}^p - \nabla p^{n+1}$

Poisson's equation for incompressible single-phase flows

- Continuous:

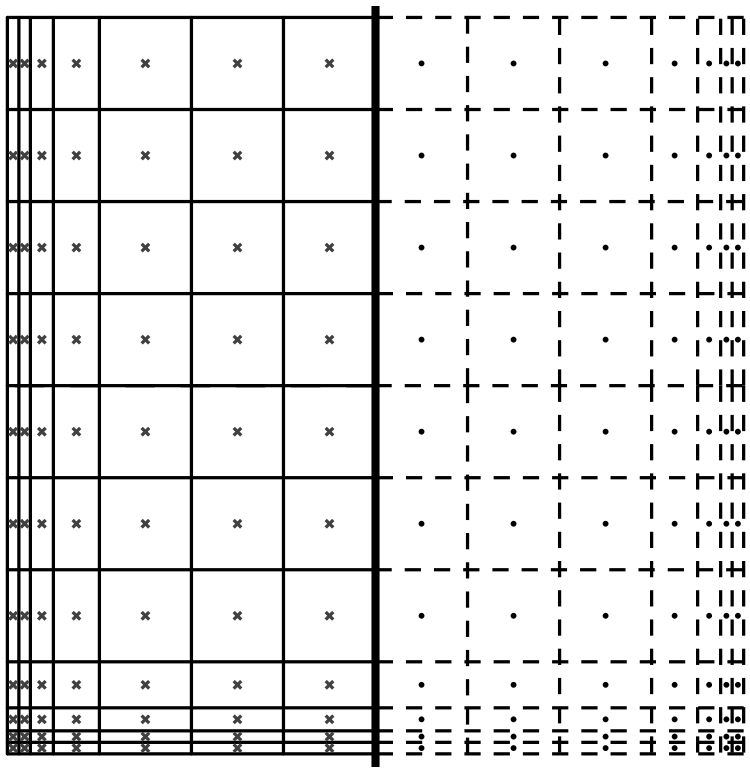
$$\Delta p = \frac{\rho}{\Delta t}\nabla\cdot\mathbf{v}^p$$

- Discrete:

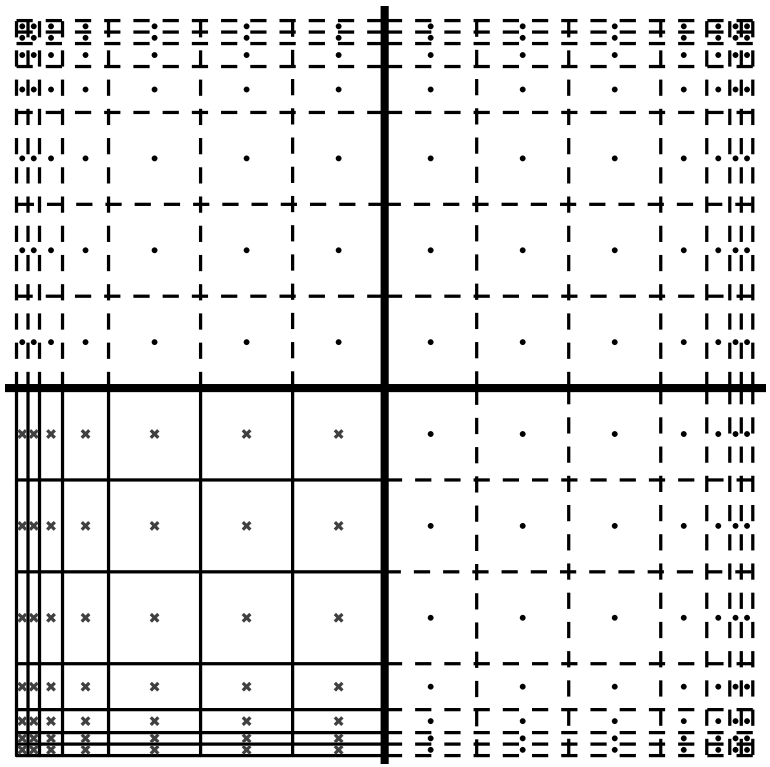
$$\mathsf{L}p_h = \frac{\rho}{\Delta t}\mathsf{M}v_h^p$$

Solving Poisson's equation

Meshes with symmetries



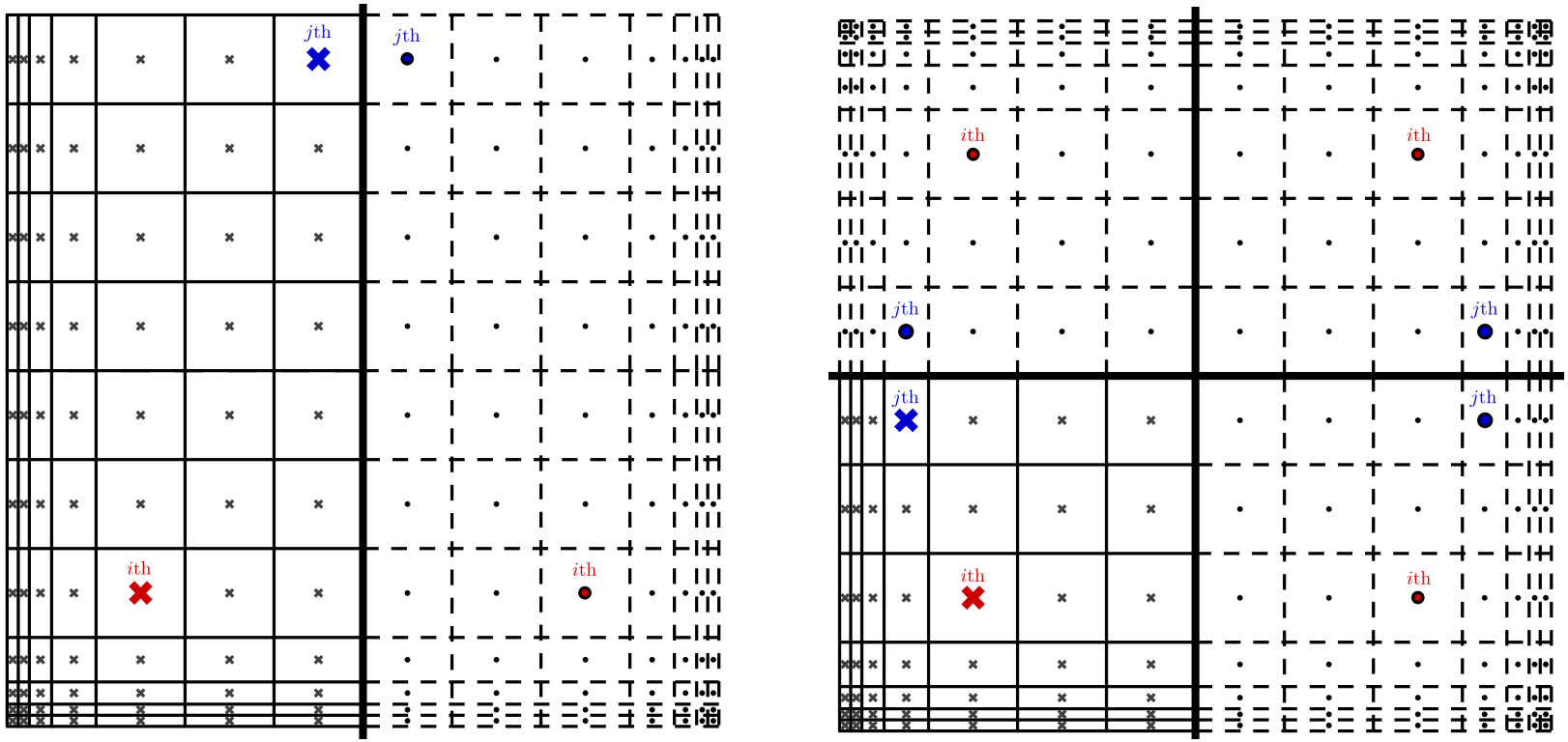
(a) 1 symmetry



(b) 2 symmetries

Figure: 2D meshes with varying number of symmetries.

"Mirrored" unknowns' ordering



(a) 1 symmetry

(b) 2 symmetries

Figure: "Mirrored" ordering on 2D meshes with a varying no. of symmetries.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

Then, thanks to the "mirrored" ordering, P transforms L:

$$\mathsf{L} = \begin{pmatrix} \mathsf{L}_{1-1} & \cdots & \mathsf{L}_{1-2^s} \\ \vdots & \ddots & \vdots \\ \mathsf{L}_{2^s-1} & \cdots & \mathsf{L}_{2^s-2^s} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

into 2^s decoupled subsystems:

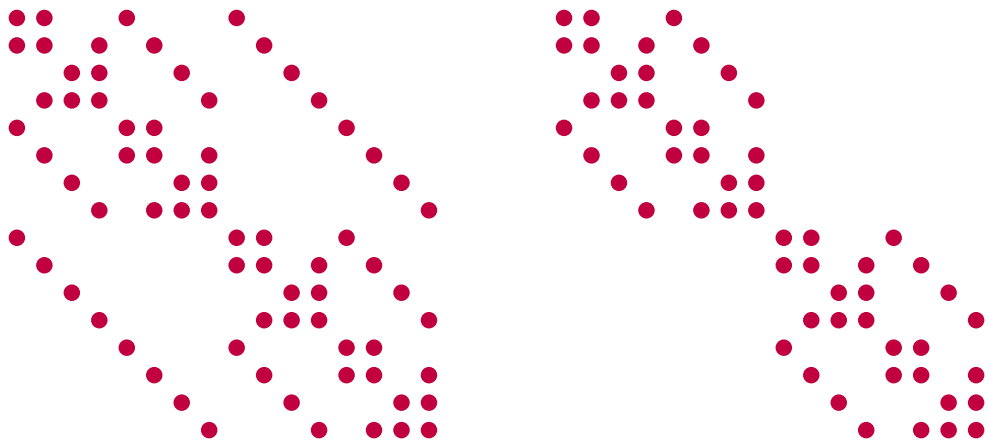
$$\hat{\mathsf{L}} = \begin{pmatrix} \hat{\mathsf{L}}_1 & & \\ & \ddots & \\ & & \hat{\mathsf{L}}_{2^s} \end{pmatrix} \in \mathbb{R}^{n \times n}$$

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$



(a) L (b) $\hat{\mathsf{L}} = \mathsf{PLP}^{-1}$

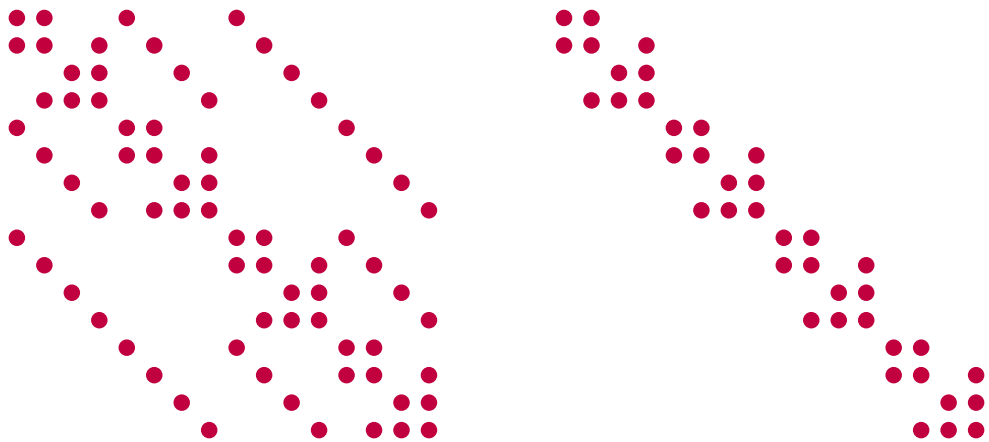
Figure: 3D structured mesh exploiting $s = 1$ symmetries.

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$



(a) L (b) $\hat{\mathsf{L}} = \mathsf{PLP}^{-1}$

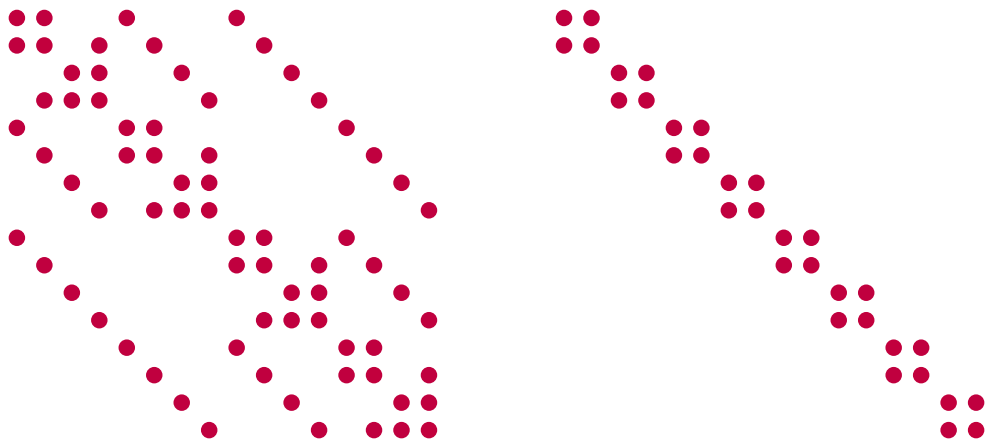
Figure: 3D structured mesh exploiting $s = 2$ symmetries.

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$



(a) L (b) $\hat{\mathsf{L}} = \mathsf{PLP}^{-1}$

Figure: 3D structured mesh exploiting $s = 3$ symmetries.

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

In general, $\hat{\mathsf{L}}$ can be split as:

$$\hat{\mathsf{L}} = \cdots = \begin{pmatrix} \mathsf{L}_{\text{inn}} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{inn}} \end{pmatrix} + \begin{pmatrix} \mathsf{L}_{\text{out}}^{(1)} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{out}}^{(2^s)} \end{pmatrix}$$

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

In general, $\hat{\mathsf{L}}$ can be split as:

$$\hat{\mathsf{L}} = \cdots = \begin{pmatrix} \mathsf{L}_{\text{inn}} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{inn}} \end{pmatrix} + \begin{pmatrix} \mathsf{L}_{\text{out}}^{(1)} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{out}}^{(2^s)} \end{pmatrix}$$

In particular, compact stencils only coupling adjacent nodes result in:

$$\hat{\mathsf{L}}\mathbf{v} = \underbrace{\left(\mathbb{I}_{2^s} \otimes \mathsf{L}_{\text{inn}}\right)\mathbf{v}}_{\substack{\text{Sparse matrix-matrix} \\ \text{product (SpMM)}}} + \underbrace{\operatorname{diag}\left(\mathbf{l}_{\text{out}}\right)\mathbf{v}}_{\substack{\text{Element-wise product} \\ \text{of vectors (axty)}}}$$

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

Given $\mathbf{v} \in \mathbb{R}^n$, the products by $\hat{\mathsf{L}}$ can be accelerated by replacing:

$$\texttt{SpMV:} \quad \begin{pmatrix} \mathsf{L}_{\text{inn}} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{inn}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{2^s} \end{pmatrix}$$

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

Given $\mathbf{v} \in \mathbb{R}^n$, the products by $\hat{\mathsf{L}}$ can be accelerated by replacing:

$$\texttt{SpMV:} \begin{pmatrix} \mathsf{L}_{\text{inn}} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{inn}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{2^s} \end{pmatrix} \text{ with } \texttt{SpMM:} \ \mathsf{L}_{\text{inn}} \begin{pmatrix} \mathbf{v}_1 \dots \mathbf{v}_{2^s} \end{pmatrix}$$

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Discrete Laplace operator and mesh symmetries

Let L be the discrete Laplace operator arising from a mesh with s symmetries, and let us define the following change of basis[7]:

$$\mathsf{P} = \frac{1}{\sqrt{2^s}} \left[\bigotimes_{i=1}^{p} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \right] \otimes \mathbb{I}_{n/2^s} \in \mathbb{R}^{n \times n}$$

Given $\mathbf{v} \in \mathbb{R}^n$, the products by $\hat{\mathsf{L}}$ can be accelerated by replacing:

$$\texttt{SpMV}: \begin{pmatrix} \mathsf{L}_{\text{inn}} & & \\ & \ddots & \\ & & \mathsf{L}_{\text{inn}} \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_{2^s} \end{pmatrix} \text{ with } \texttt{SpMM}: \mathsf{L}_{\text{inn}} \begin{pmatrix} \mathbf{v}_1 \dots \mathbf{v}_{2^s} \end{pmatrix}$$

Hence:

- Since SpMV and SpMM are memory-bound kernels, SpMM's acceleration equals $\mathsf{I}_{\texttt{SpMM}}/\mathsf{I}_{\texttt{SpMV}}$
- SpMM reads L_{inn} once, whereas SpMV reads L_{inn} 2^s times.

[7] A. Alsalti-Baldellou et al. (2023). "Exploiting spatial symmetries for solving Poisson's equation" in *Journal of Computational Physics*.

Resulting algorithm

Algorithm Poisson solver exploiting s mesh symmetries

1. Transform forward the RHS: $\hat{b} = \mathsf{P}b$
2. Decoupled solution of the 2^s subsystems: $\hat{\mathsf{L}}\hat{x} = \hat{b}$
3. Transform backward the solution: $x = \mathsf{P}^{-1}\hat{x}$

where:

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

and Step 2 corresponds to inverting:

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

Summary

Summary:

- The overhead of the two (communication-free) transforms is negligible.

Summary

Summary:

- The overhead of the two (communication-free) transforms is negligible.
- Exploiting symmetries **reduces the setup costs** of the matrices.
- Exploiting symmetries **reduces the memory footprint** of the matrices.
- Exploiting symmetries **reduces the time complexity** of the solvers.

Summary

Summary:

- The overhead of the two (communication-free) transforms is negligible.
- Exploiting symmetries **reduces the setup costs** of the matrices.
- Exploiting symmetries **reduces the memory footprint** of the matrices.
- Exploiting symmetries **reduces the time complexity** of the solvers.
- `SpMM` naturally applies to all operators of the form $\hat{\mathrm{A}} = \mathbb{I}_{2^s} \otimes \mathrm{A}$.
- `SpMM` **increases considerably the I** of *all* the matrix multiplications.

Summary

Summary:

- The overhead of the two (communication-free) transforms is negligible.
- Exploiting symmetries **reduces the setup costs** of the matrices.
- Exploiting symmetries **reduces the memory footprint** of the matrices.
- Exploiting symmetries **reduces the time complexity** of the solvers.
- `SpMM` naturally applies to all operators of the form $\hat{\mathrm{A}} = \mathbb{I}_{2^s} \otimes \mathrm{A}$.
- `SpMM` **increases considerably the I** of *all* the matrix multiplications.

Still missing...

Since all the subsystems are (slightly) different, so are their preconditioners, and `SpMM` cannot be applied with them!

Preconditioning Poisson's equation

Right, left and split preconditioning

Let $A \in \mathbb{R}^n$ and $x, b \in \mathbb{R}^n$. Then, given the linear system $Ax = b$, we can consider the following preconditioning techniques:

Left preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the left-preconditioned system is:

$$M^{-1}Ax = M^{-1}b$$

Right, left and split preconditioning

Let $A \in \mathbb{R}^n$ and $x, b \in \mathbb{R}^n$. Then, given the linear system $Ax = b$, we can consider the following preconditioning techniques:

Left preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the left-preconditioned system is:

$$M^{-1}Ax = M^{-1}b$$

Right preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the right-preconditioned system is:

$$AM^{-1}y = b, \text{ where } Mx = y$$

Right, left and split preconditioning

Let $A \in \mathbb{R}^n$ and $x, b \in \mathbb{R}^n$. Then, given the linear system $Ax = b$, we can consider the following preconditioning techniques:

Left preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the left-preconditioned system is:

$$M^{-1}Ax = M^{-1}b$$

Right preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the right-preconditioned system is:

$$AM^{-1}y = b, \text{ where } Mx = y$$

Split preconditioning

Given the preconditioner $M^{-1} = M_1^{-1}M_2^{-1} \simeq A^{-1}$, the split-preconditioned system is:

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b, \text{ where } M_2x = y$$

Right, left and split preconditioning

Let $A \in \mathbb{R}^n$ and $x, b \in \mathbb{R}^n$. Then, given the linear system $Ax = b$, we can consider the following preconditioning techniques:

Left preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the left-preconditioned system is:

$$M^{-1}Ax = M^{-1}b$$

Right preconditioning

Given the preconditioner $M^{-1} \simeq A^{-1}$, the right-preconditioned system is:

$$AM^{-1}y = b, \text{ where } Mx = y$$

Split preconditioning

Given the preconditioner $M^{-1} = M_1^{-1}M_2^{-1} \simeq A^{-1}$, the split-preconditioned system is:

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b, \text{ where } M_2x = y$$

Thus, preconditioning reduces to operations of the type: $y = M^{-1}x$

"Full-rank" corrections – 1

"Full-rank" corrections[8]

Given two SPD matrices A and B, let the Cholesky decomposition of B be $B = LL^t$. Then, given $Y := (\mathbb{I} - L^{-1}AL^{-t})$, the following holds:

$$A^{-1} = L^{-t}L^{-1} + W\Theta W^t,$$

where $Y = U\Sigma U^t$ is the eigendecomposition of Y, and:

$$\Theta := \Sigma\left(\mathbb{I} - \Sigma\right)^{-1} \text{ and } W := L^{-t}U.$$

[8] R. Li and Y. Saad (2013). "Divide and conquer low-rank preconditioners for symmetric matrices" in *SIAM Journal on Scientific Computing*.

"Full-rank" corrections – 1

"Full-rank" corrections[8]

Given two SPD matrices A and B, let the Cholesky decomposition of B be $B = LL^t$. Then, given $Y := (\mathbb{I} - L^{-1}AL^{-t})$, the following holds:

$$A^{-1} = L^{-t}L^{-1} + W\Theta W^t,$$

where $Y = U\Sigma U^t$ is the eigendecomposition of Y, and:

$$\Theta := \Sigma\left(\mathbb{I} - \Sigma\right)^{-1} \text{ and } W := L^{-t}U.$$

Taking a closer look to the spectral decomposition of Y, we have:

$$Y = U\Sigma U^t, \text{ where } \Sigma = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix}.$$

[8] R. Li and Y. Saad (2013). "Divide and conquer low-rank preconditioners for symmetric matrices" in *SIAM Journal on Scientific Computing*.

"Full-rank" corrections – 2

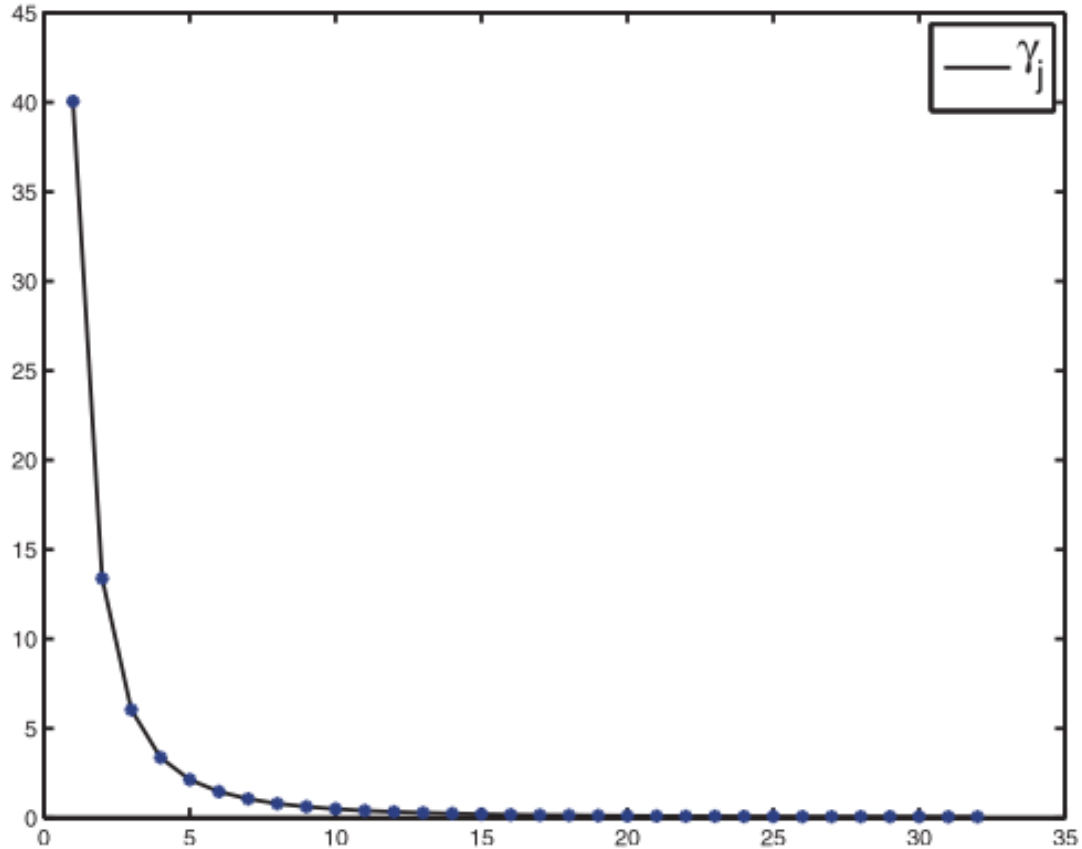


Figure: Example of rapidly decaying eigenvalues[8].

[8] R. Li and Y. Saad (2013). "Divide and conquer low-rank preconditioners for symmetric matrices" in *SIAM Journal on Scientific Computing*.

Low-rank corrections

Hence, whenever Y's spectrum decays rapidly we can consider truncated eigendecompositions accounting for its k most relevant eigenpairs:

$$Y = U\Sigma U^t \simeq U_k \Sigma_k U_k^t \in \mathbb{R}^{n\times n}$$

where $\Sigma_k \in \mathbb{R}^{k\times k}$ and $U_k \in \mathbb{R}^{n\times k}$.

Low-rank corrections

Hence, whenever Y's spectrum decays rapidly we can consider truncated eigendecompositions accounting for its k most relevant eigenpairs:

$$Y = U\Sigma U^t \simeq U_k \Sigma_k U_k^t \in \mathbb{R}^{n\times n}$$

where $\Sigma_k \in \mathbb{R}^{k\times k}$ and $U_k \in \mathbb{R}^{n\times k}$.

Low-rank correction[9]

Given two SPD matrices A and B, let the Cholesky decomposition of B be $B = LL^t$. Then, given $Y \coloneqq (\mathbb{I} - L^{-1}AL^{-t})$, the following holds:

$$A^{-1} \simeq L^{-t}L^{-1} + \mathbf{W_k \Theta_k W_k^t},$$

where $\mathbf{Y \simeq U_k \Sigma_k U_k^t}$ and:

$$\mathbf{\Theta_k \coloneqq \Sigma_k \left(\mathbb{I} - \Sigma_k\right)^{-1}} \text{ and } \mathbf{W_k \coloneqq L^{-t} U_k}.$$

[9] A. Franceschini et al. (2018). "A robust multilevel approximate inverse preconditioner for symmetric positive definite matrices" in *SIAM Journal on Matrix Analysis and Applications*.

Low-rank corrected FSAI

As we saw, s symmetric directions allow decomposing Poisson's equation, $\mathsf{L}x = b$, into 2^s decoupled subsystems with the following structure:

$$\begin{pmatrix} \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(1)} & & 0 \\ & \ddots & \\ 0 & & \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(2^s)} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_{2^s} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{2^s} \end{pmatrix},$$

and such that:

$$\boxed{\operatorname{rank}(\mathsf{L}_{\text{out}}^{(i)}) = n_{\text{ifc}} \ll \operatorname{rank}(\mathsf{L}_{\text{inn}}) = n}$$

Low-rank corrected FSAI

As we saw, s symmetric directions allow decomposing Poisson's equation, $\mathsf{L}x = b$, into 2^s decoupled subsystems with the following structure:

$$\begin{pmatrix} \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(1)} & & 0 \\ & \ddots & \\ 0 & & \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(2^s)} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_{2^s} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{2^s} \end{pmatrix},$$

and such that:

$$\boxed{\operatorname{rank}(\mathsf{L}_{\text{out}}^{(i)}) = n_{\text{ifc}} \ll \operatorname{rank}(\mathsf{L}_{\text{inn}}) = n}$$

Eureka!

Let M_{inn} be a preconditioner for L_{inn}, *i.e.*, $M_{\text{inn}}^{-1} \simeq \mathsf{L}_{\text{inn}}^{-1}$. Then, we can seek low-rank corrections for M_{inn} such that:

$$\hat{\mathsf{L}}^{-1} \simeq \mathbb{I}_{2^s} \otimes M_{\text{inn}} + \begin{pmatrix} W_k^{(1)} \Theta_k^{(1)} W_k^{(1)^t} & & 0 \\ & \ddots & \\ 0 & & W_k^{(2^s)} \Theta_k^{(2^s)} W_k^{(2^s)^t} \end{pmatrix},$$

Low-rank corrected FSAI

As we saw, s symmetric directions allow decomposing Poisson's equation, $\mathsf{L}x = b$, into 2^s decoupled subsystems with the following structure:

$$\begin{pmatrix} \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(1)} & & 0 \\ & \ddots & \\ 0 & & \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(2^s)} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_{2^s} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{2^s} \end{pmatrix},$$

and such that:

$$\boxed{\operatorname{rank}(\mathsf{L}_{\text{out}}^{(i)}) = n_{\text{ifc}} \ll \operatorname{rank}(\mathsf{L}_{\text{inn}}) = n}$$

Eureka!

Let M_{inn} be a preconditioner for L_{inn}, *i.e.*, $M_{\text{inn}}^{-1} \simeq \mathsf{L}_{\text{inn}}^{-1}$. Then, we can seek low-rank corrections for M_{inn} such that:

$$\hat{\mathsf{L}}^{-1} \simeq \mathbb{I}_{2^s} \otimes M_{\text{inn}} + \begin{pmatrix} W_k^{(1)} \Theta_k^{(1)} W_k^{(1)^t} & & 0 \\ & \ddots & \\ 0 & & W_k^{(2^s)} \Theta_k^{(2^s)} W_k^{(2^s)^t} \end{pmatrix},$$

As a result: **lower setup costs**, **decoupled corrections** and `SpMM`!

Low-rank corrected FSAI

As we saw, s symmetric directions allow decomposing Poisson's equation, $\mathsf{L}x = b$, into 2^s decoupled subsystems with the following structure:

$$\begin{pmatrix} \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(1)} & & 0 \\ & \ddots & \\ 0 & & \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(2^s)} \end{pmatrix} \begin{pmatrix} \hat{\mathbf{x}}_1 \\ \vdots \\ \hat{\mathbf{x}}_{2^s} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{b}}_1 \\ \vdots \\ \hat{\mathbf{b}}_{2^s} \end{pmatrix},$$

and such that:

$$\boxed{\operatorname{rank}(\mathsf{L}_{\text{out}}^{(i)}) = n_{\text{ifc}} \ll \operatorname{rank}(\mathsf{L}_{\text{inn}}) = n}$$

LRCFSAI(k): Low-rank corrected FSAI

Let the aFSAI[10] of L_{inn} be $G_{\text{inn}}^t G_{\text{inn}} \simeq \mathsf{L}_{\text{inn}}^{-1}$.
For each subsystem $\hat{\mathsf{L}}_i = \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}^{(i)}$, let $Y := (\mathbb{I} - G_{\text{inn}} \hat{\mathsf{L}}_i G_{\text{inn}}^t)$.
Then:

$$\hat{\mathsf{L}}_i^{-1} \simeq G_{\text{inn}}^t G_{\text{inn}} + W_k \Theta_k W_k^t,$$

where $Y \simeq U_k \Sigma_k U_k^t$ and $\Theta_k := \Sigma_k \left(\mathbb{I} - \Sigma_k\right)^{-1}$ and $W_k := L^{-t} U_k$.

[10] C. Janna and M. Ferronato (2011). "Adaptive pattern research for block FSAI preconditioning" in *SIAM Journal on Scientific Computing*.

Low-rank corrected FSAI: residual convergence

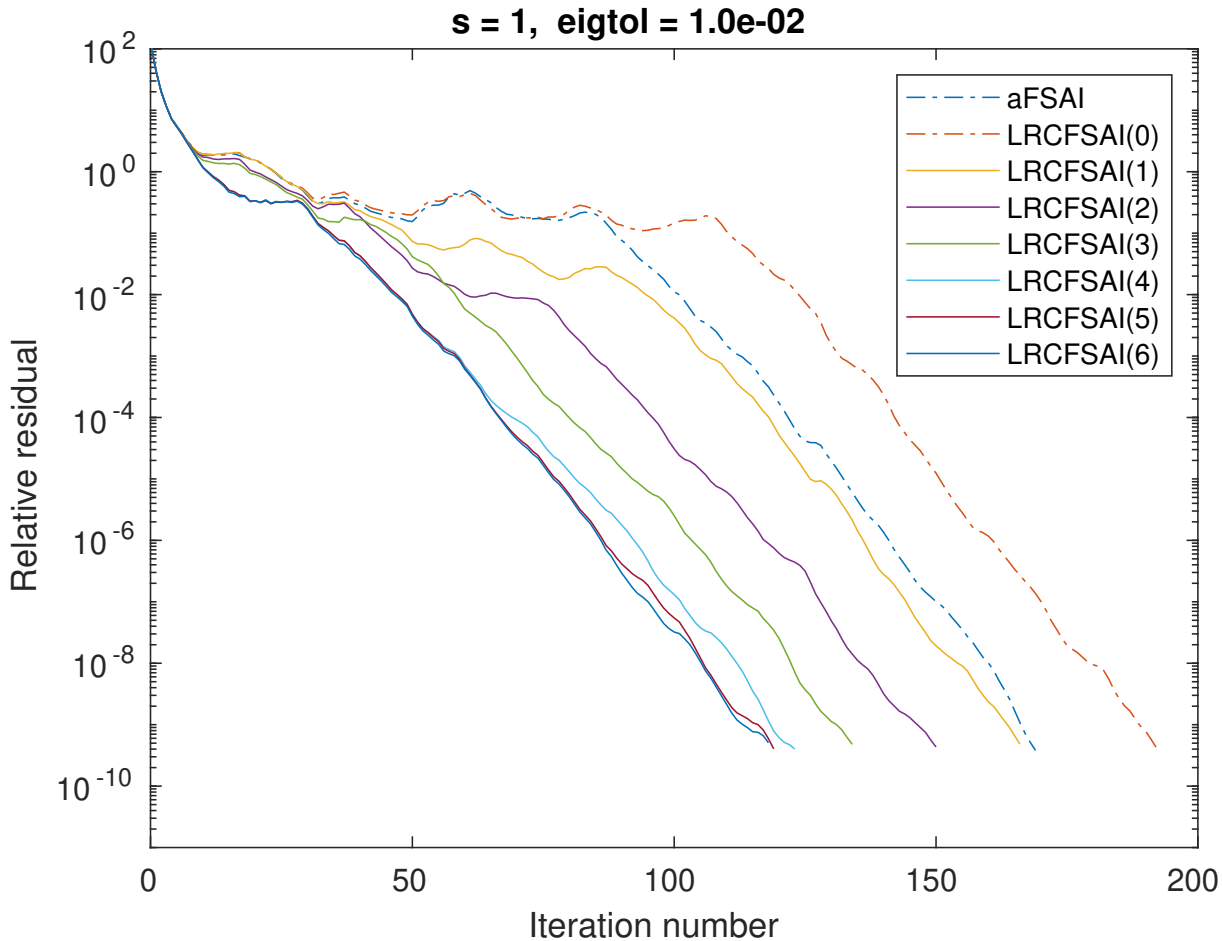


Figure: Convergence of PCG+LRCFSAI(k) on a 32^3 mesh with $s = 1$ symmetries.

Low-rank corrected FSAI: residual convergence

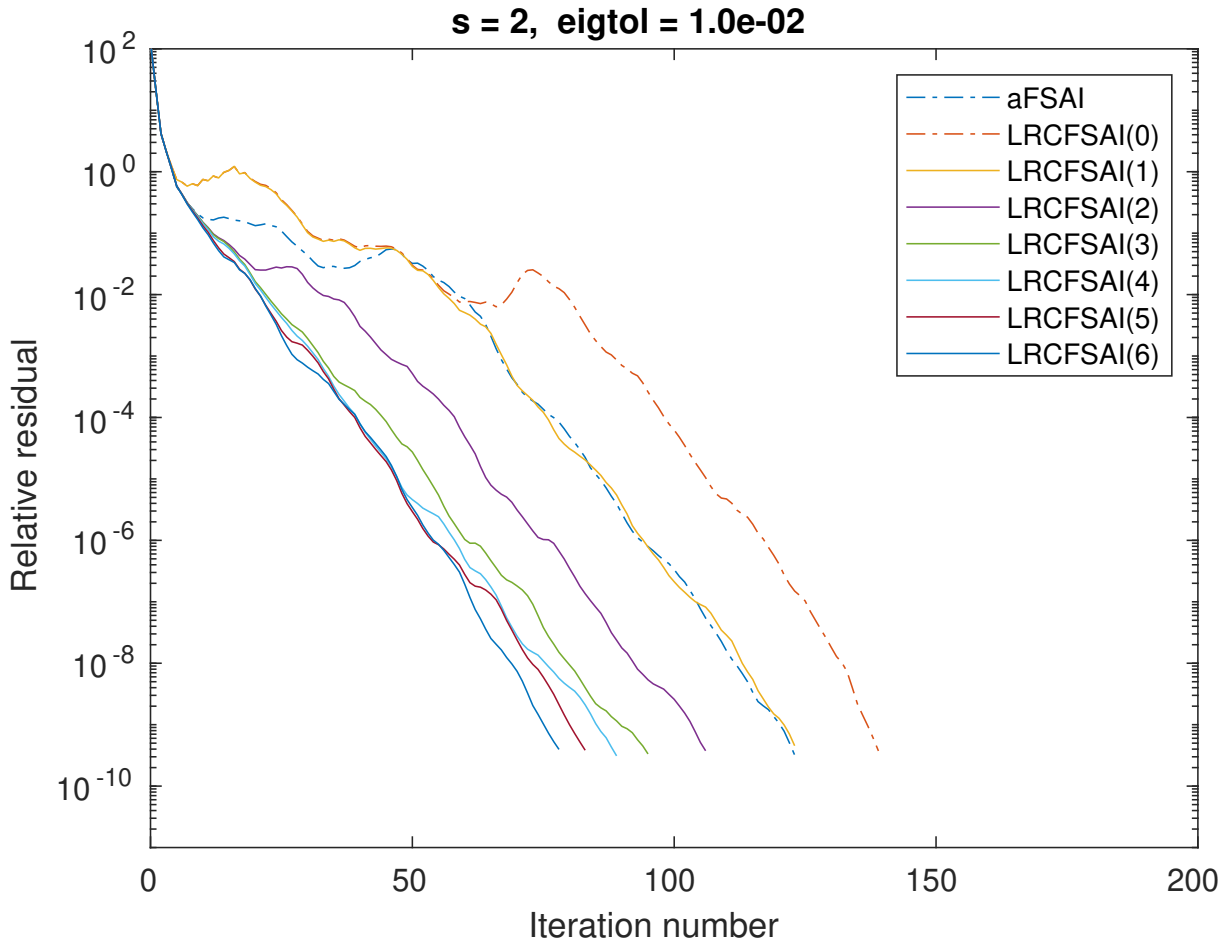


Figure: Convergence of PCG+LRCFSAI(k) on a 32^3 mesh with $s = 2$ symmetries.

Low-rank corrected FSAI: residual convergence

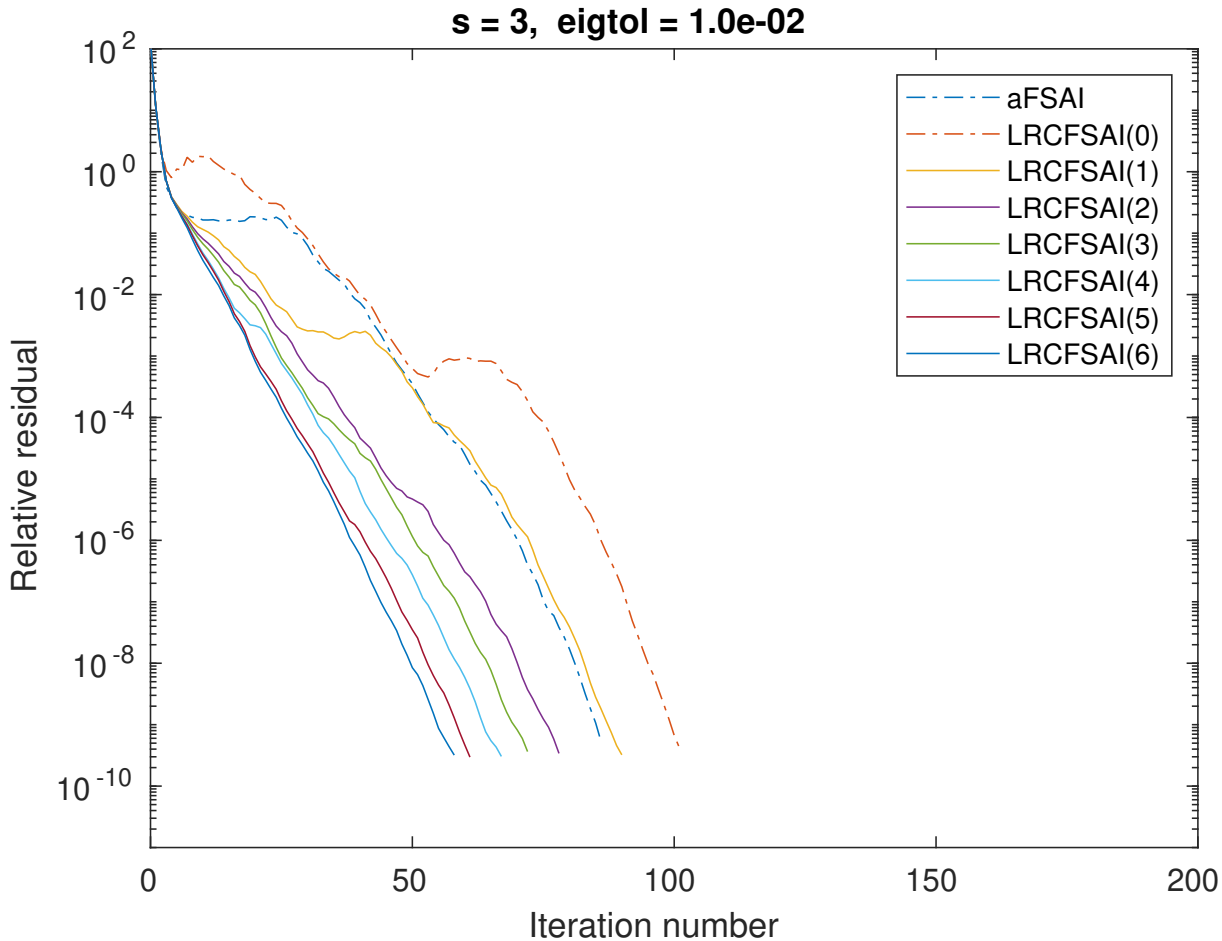


Figure: Convergence of PCG+LRCFSAI(k) on a 32^3 mesh with $s = 3$ symmetries.

Low-rank corrected FSAI: eigensolvers' accuracy

DOF	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

Table: Influence of Lanczos tolerance on the PCG iterations.

Low-rank corrected FSAI: time per iteration

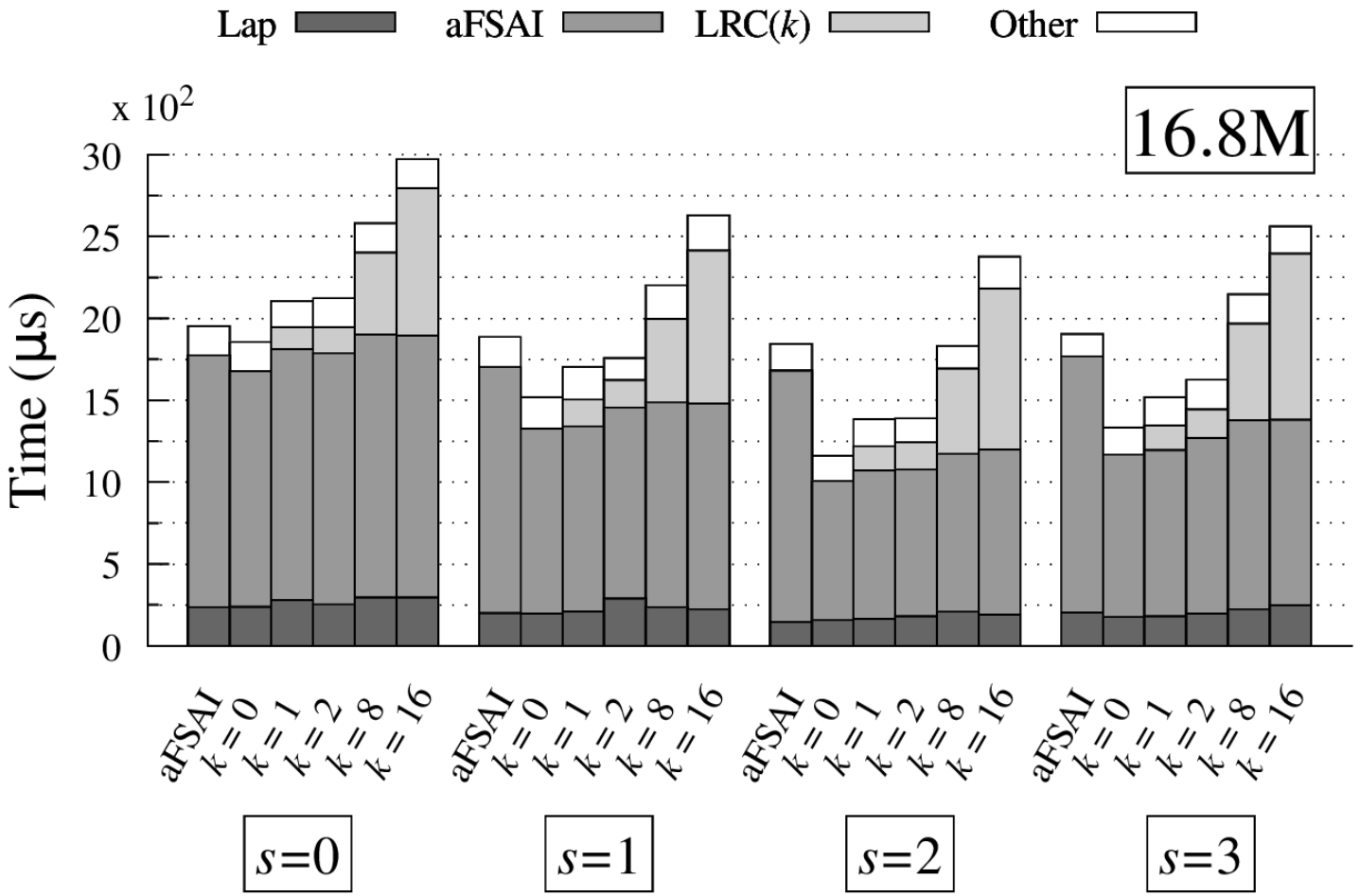


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

Low-rank corrected FSAI: time per iteration

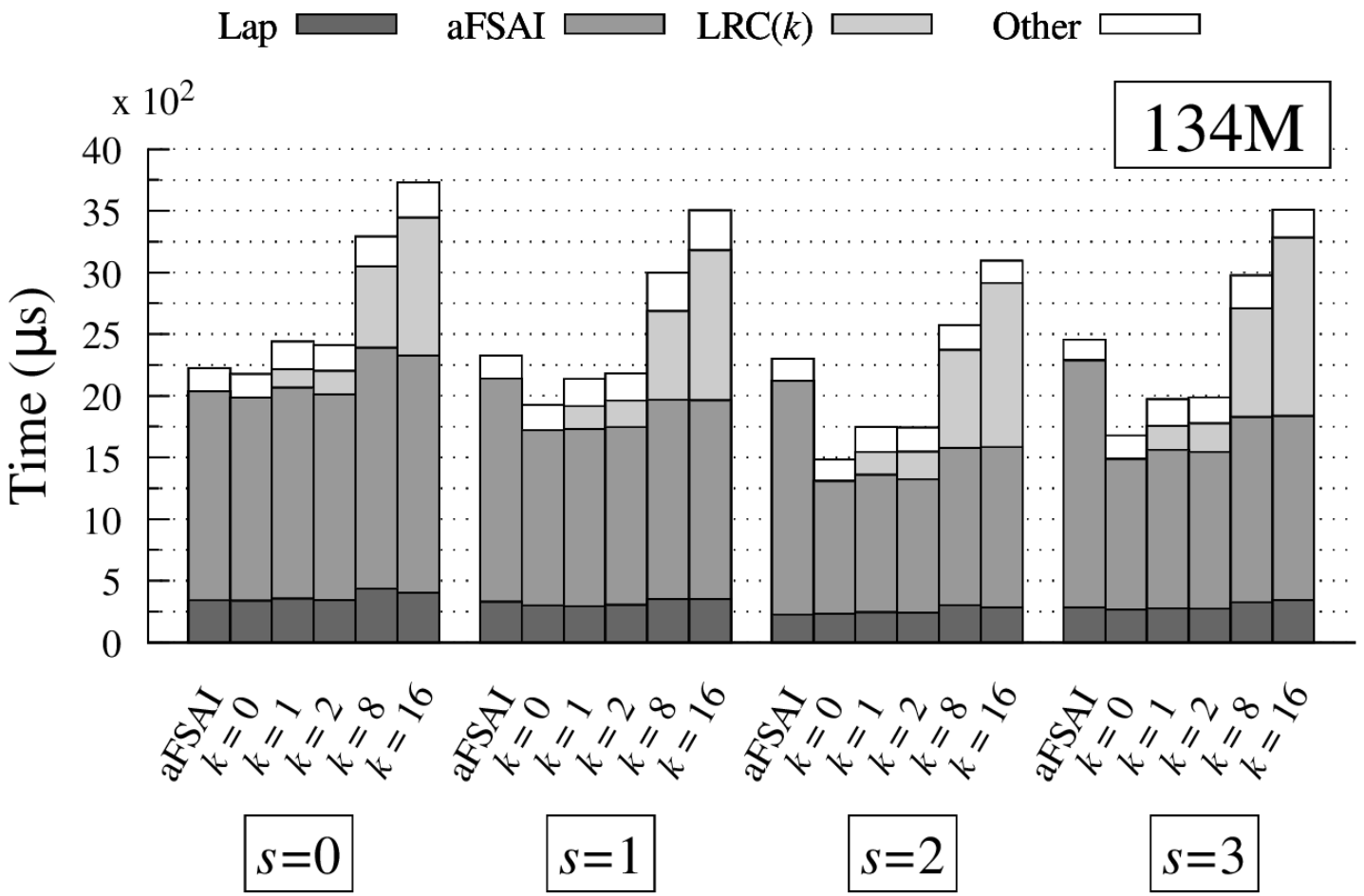


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

Low-rank corrected FSAI: time per iteration

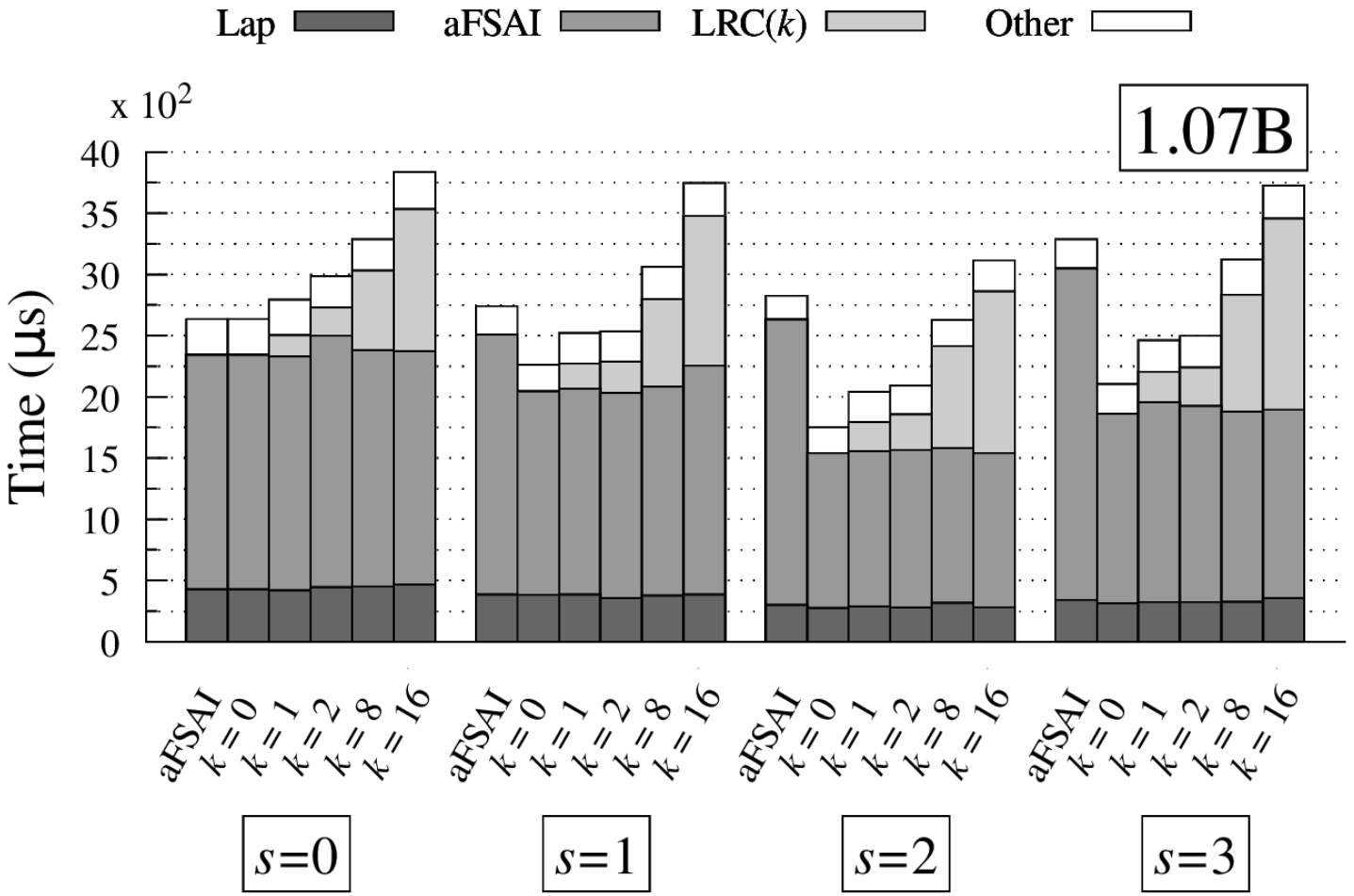


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

Low-rank corrected FSAI: overview

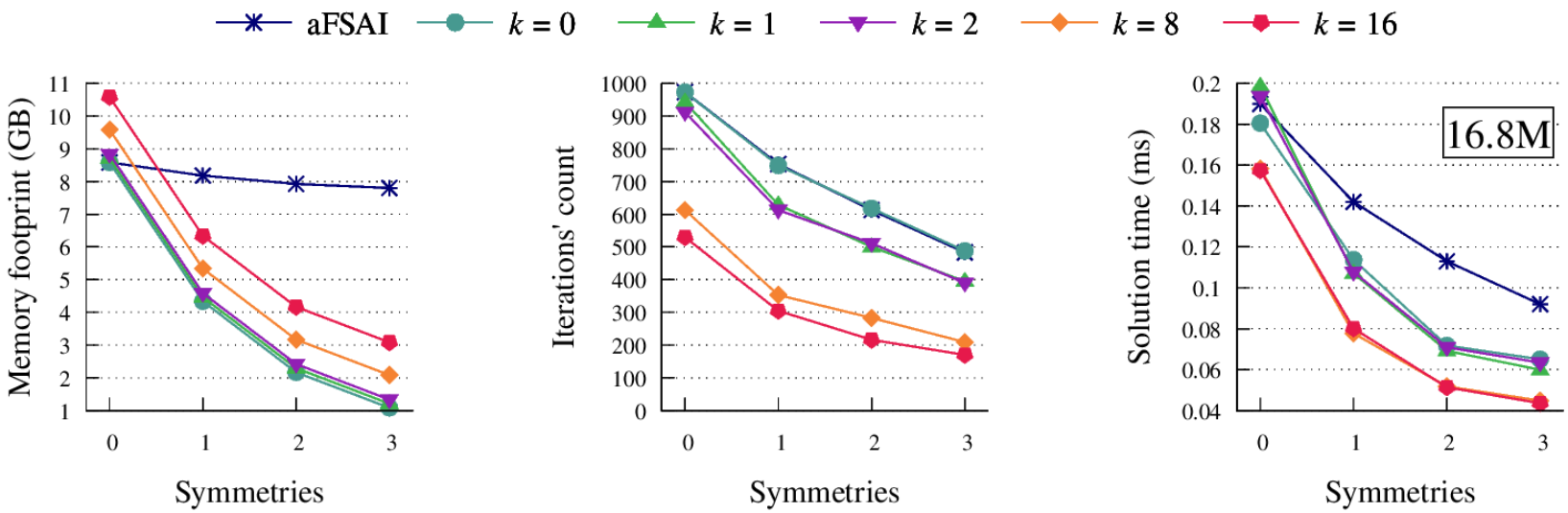


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

Low-rank corrected FSAI: overview

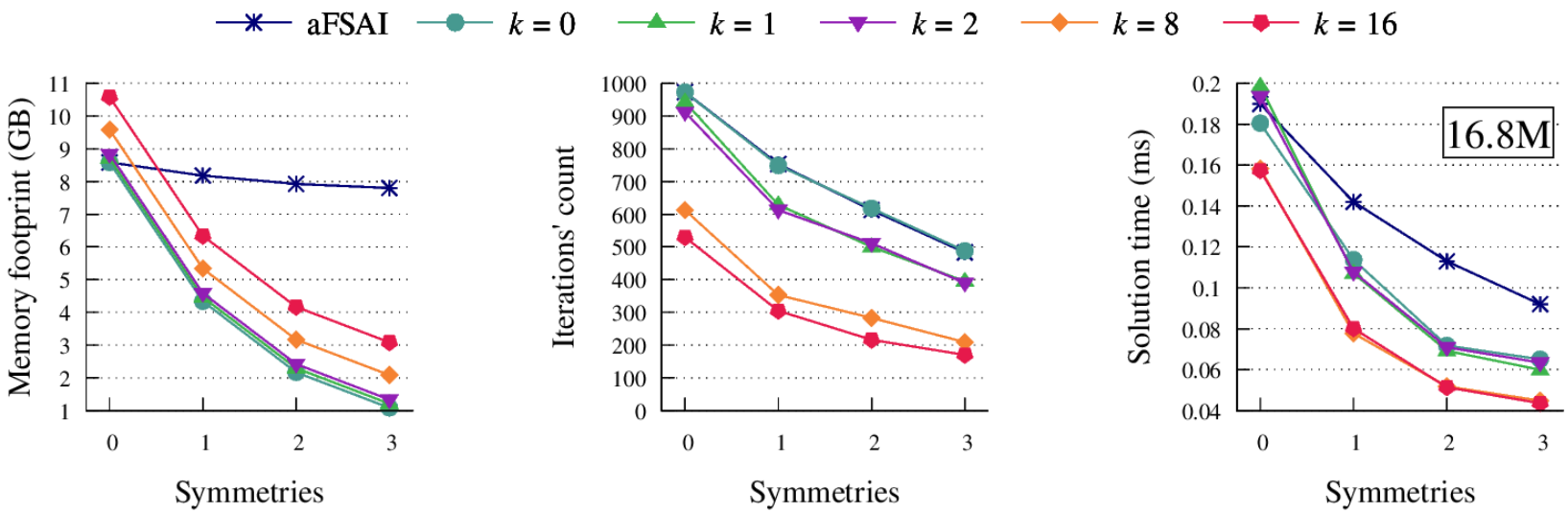


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

In summary, LRCFSAI(k) enhances the standard FSAI in many aspects:

- LRCFSAI(k) has up to **2.6x lower memory requirements**.

Low-rank corrected FSAI: overview

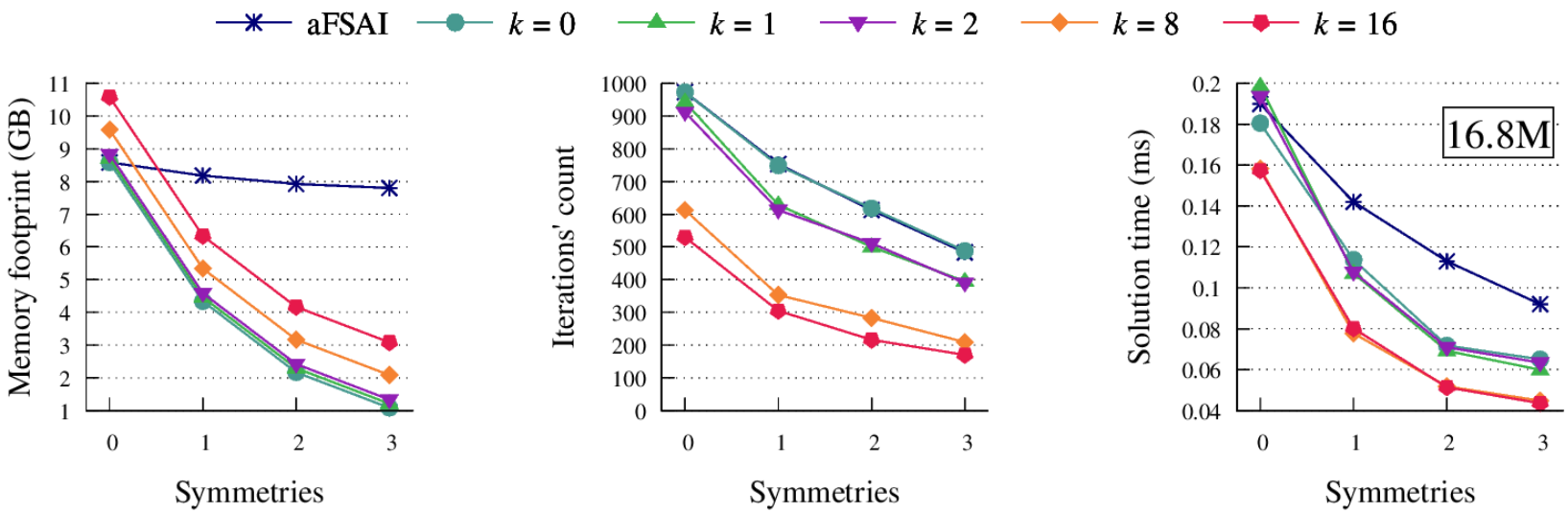


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

In summary, LRCFSAI(k) enhances the standard FSAI in many aspects:

- LRCFSAI(k) has up to **2.6x lower memory requirements**.
- LRCFSAI(k) requires up to **5.7x fewer iterations**

Low-rank corrected FSAI: overview

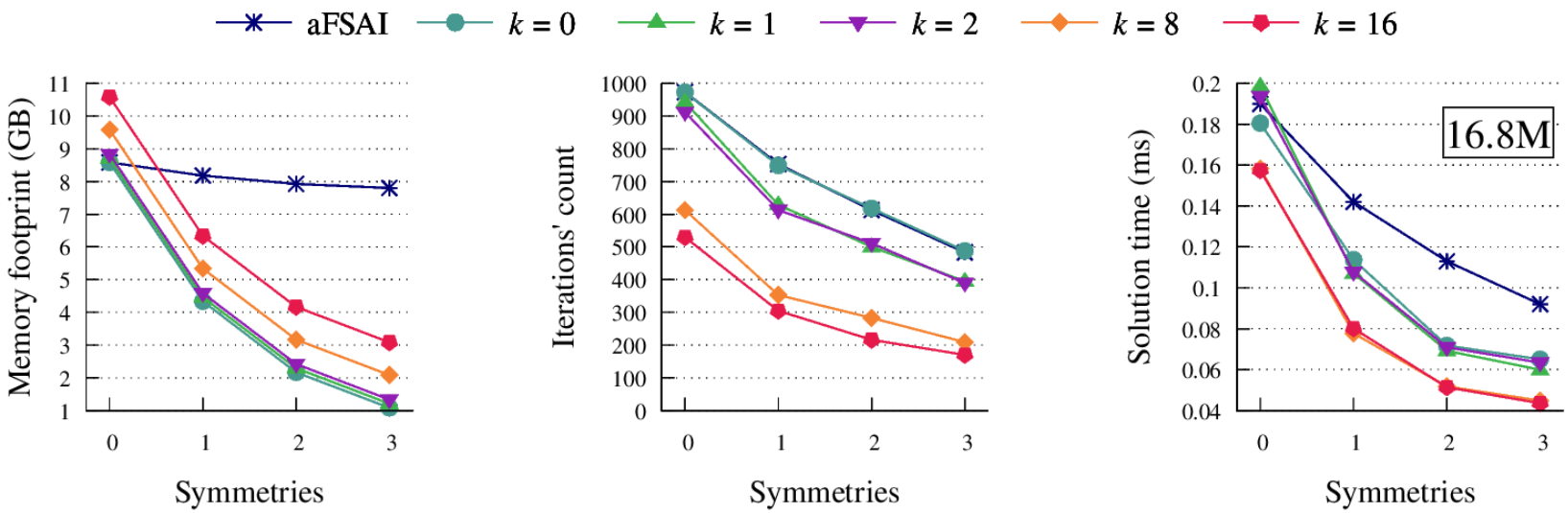


Figure: Normalised time per PCG+LRCFSAI(k) iteration on MARCONI.

In summary, LRCFSAI(k) enhances the standard FSAI in many aspects:

- LRCFSAI(k) has up to **2.6x lower memory requirements**.
- LRCFSAI(k) requires up to **5.7x fewer iterations**
- LRCFSAI(k) results in up to **4.4x overall speed-ups**.

Concluding remarks

Conclusions

Summary:

- Low-rank corrections make FSAI compatible with `SpMM`.

Conclusions

Summary:

- Low-rank corrections make FSAI compatible with `SpMM`.
- Need for a balance with higher-rank corrections: fewer vs slower iterations.

Conclusions

Summary:

- Low-rank corrections make FSAI compatible with `SpMM`.
- Need for a balance with higher-rank corrections: fewer vs slower iterations.
- LRCFSAI(k) reduces the **setup costs** of FSAI.
- LRCFSAI(k) reduces the **memory footprint** of FSAI.
- LRCFSAI(k) reduces the **time-to-solution** of FSAI.

Conclusions

Summary:

- Low-rank corrections make FSAI compatible with `SpMM`.
- Need for a balance with higher-rank corrections: fewer vs slower iterations.
- LRCFSAI(k) reduces the **setup costs** of FSAI.
- LRCFSAI(k) reduces the **memory footprint** of FSAI.
- LRCFSAI(k) reduces the **time-to-solution** of FSAI.

Ongoing work:

- Extend `SpMM`'s use to AMG.

Conclusions

Summary:

- Low-rank corrections make FSAI compatible with `SpMM`.
- Need for a balance with higher-rank corrections: fewer vs slower iterations.
- LRCFSAI(k) reduces the **setup costs** of FSAI.
- LRCFSAI(k) reduces the **memory footprint** of FSAI.
- LRCFSAI(k) reduces the **time-to-solution** of FSAI.

Ongoing work:

- Extend `SpMM`'s use to AMG.
- Extend `SpMM`'s use to repeated geometries.

Conclusions

Summary:

- Low-rank corrections make FSAI compatible with `SpMM`.
- Need for a balance with higher-rank corrections: fewer vs slower iterations.
- LRCFSAI(k) reduces the **setup costs** of FSAI.
- LRCFSAI(k) reduces the **memory footprint** of FSAI.
- LRCFSAI(k) reduces the **time-to-solution** of FSAI.

Ongoing work:

- Extend `SpMM`'s use to AMG.
- Extend `SpMM`'s use to repeated geometries.
- Extend `SpMM`'s use to structural mechanics.

Thanks for your attention!