Scalable coarse spaces for monolithic Schwarz preconditioners

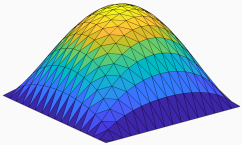
Alexander Heinlein[1]

28th International Conference on Domain Decomposition Methods (DD28), KAUST, Saudi Arabia, January 28 - February 1, 2024

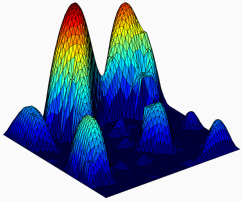
[1]Delft University of Technology

Based on joint work with Axel Klawonn and Lea Saßmannshausen (Universität zu Köln)

Solving A Model Problem



$\alpha(x) = 1$



heterogeneous $\alpha(x)$

Consider a **diffusion model problem**:

$$-\nabla \cdot (\alpha(x)\nabla u(x)) = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega.$$

Discretization using finite elements yields a **sparse** linear system of equations

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}.$$

Direct solvers

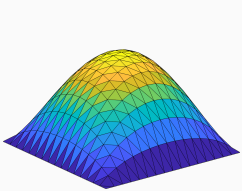
For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

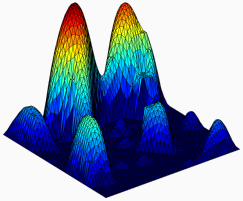
Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number** $\kappa(\boldsymbol{A})$. It deteriorates, e.g., for

- fine meshes, that is, small element sizes h
- large contrasts $\frac{\max_x \alpha(x)}{\min_x \alpha(x)}$

Solving A Model Problem



$\alpha(x) = 1$



heterogeneous $\alpha(x)$

Consider a **diffusion model problem**:

$$-\nabla \cdot (\alpha(x)\nabla u(x)) = f \quad \text{in } \Omega = [0,1]^2,$$
$$u = 0 \quad \text{on } \partial\Omega.$$

Discretization using finite elements yields a **sparse** linear system of equations

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}.$$

Direct solvers

For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number** $\kappa(\boldsymbol{A})$. It deteriorates, e.g., for

- fine meshes, that is, small element sizes h
- large contrasts $\frac{\max_x \alpha(x)}{\min_x \alpha(x)}$

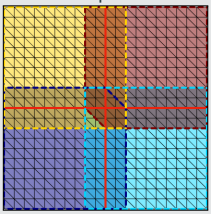
$\Rightarrow$ We introduce a preconditioner $\boldsymbol{M}^{-1} \approx \boldsymbol{A}^{-1}$ to improve the condition number:

$$\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f}$$

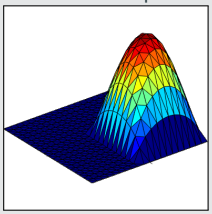
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



Based on an **overlapping domain decomposition**, we define a **one-level Schwarz operator**

$$M_{\text{OS-1}}^{-1}K = \sum_{i=1}^{N} R_i^T K_i^{-1} R_i K,$$

where R_i and R_i^T are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^T$.

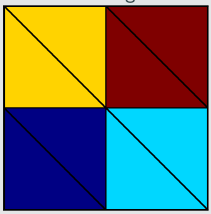
Condition number estimate:

$$\kappa\left(M_{\text{OS-1}}^{-1}K\right) \leq C\left(1 + \frac{1}{H\delta}\right)$$

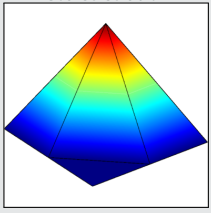
with subdomain size H and overlap width δ.

Lagrangian coarse space

Coarse triangulation



Coarse solution



The **two-level overlapping Schwarz operator** reads

$$M_{\text{OS-2}}^{-1}K = \underbrace{\Phi K_0^{-1}\Phi^T K}_{\text{coarse level – global}} + \underbrace{\sum_{i=1}^{N} R_i^T K_i^{-1} R_i K}_{\text{first level – local}},$$

where Φ contains the coarse basis functions and $K_0 := \Phi^T K \Phi$; cf., e.g., **Toselli, Widlund (2005)**. The construction of a Lagrangian coarse basis requires a coarse triangulation.

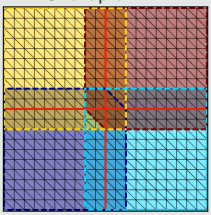
Condition number estimate:

$$\kappa\left(M_{\text{OS-2}}^{-1}K\right) \leq C\left(1 + \frac{H}{\delta}\right)$$

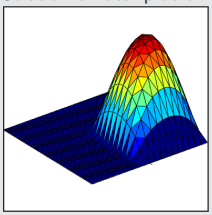
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$

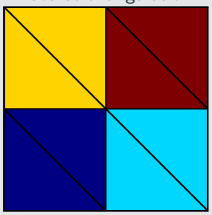


Solution of local problem

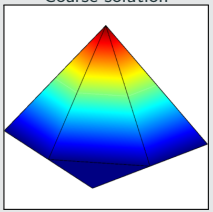


Lagrangian coarse space

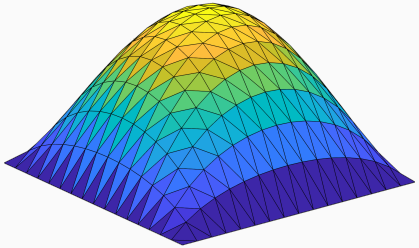
Coarse triangulation

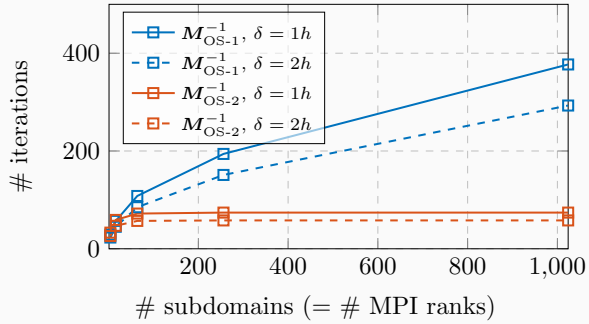


Coarse solution

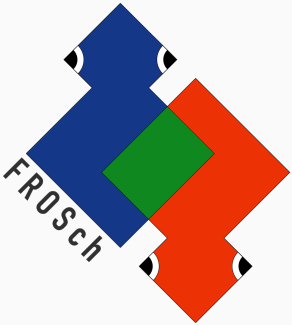


Diffusion model problem in two dimensions,
$H/h = 100$





FROSch (Fast and Robust Overlapping Schwarz) Framework in Trilinos



Software

- Object-oriented C++ domain decomposition solver framework with MPI-based distributed memory parallelization
- Part of Trilinos with support for both parallel linear algebra packages Epetra and Tpetra
- Node-level parallelization and performance portability on CPU and GPU architectures through Kokkos and KokkosKernels
- Accessible through unified Trilinos solver interface Stratimikos

Methodology

- Parallel scalable multi-level Schwarz domain decomposition preconditioners
- Algebraic construction based on the parallel distributed system matrix
- Extension-based coarse spaces

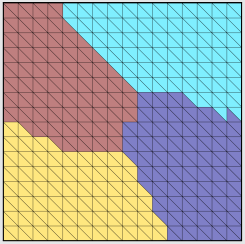
Team (active)

- Filipe Cumaru (TU Delft)
- Kyrill Ho (UCologne)
- Siva Rajamanickam (SNL)
- Oliver Rheinbach (TUBAF)
- Ichitaro Yamazaki (SNL)
- Alexander Heinlein (TU Delft)
- Axel Klawonn (UCologne)
- Friederike Röver (TUBAF)
- Lea Saßmannshausen (UCologne)

Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD

Computation of the overlapping matrices

The overlapping matrices

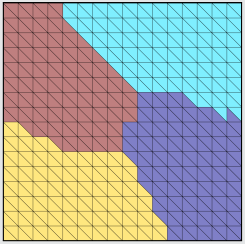
$$K_i = R_i K R_i^T$$

can easily be extracted from K since R_i is just a **global-to-local index mapping**.

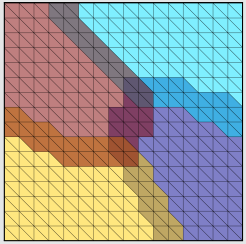
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$

Computation of the overlapping matrices

The overlapping matrices

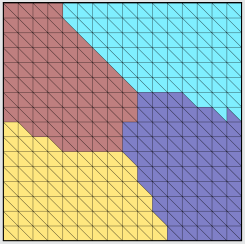
$$K_i = R_i K R_i^T$$

can easily be extracted from K since R_i is just a **global-to-local index mapping**.

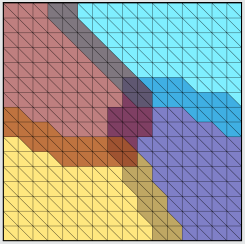
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

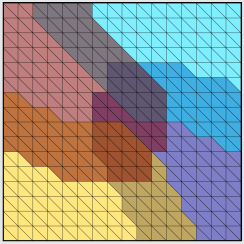
In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$

Computation of the overlapping matrices

The overlapping matrices

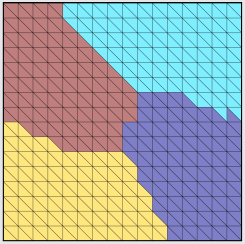
$$K_i = R_i K R_i^T$$

can easily be extracted from K since R_i is just a **global-to-local index mapping**.

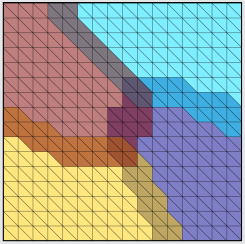
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

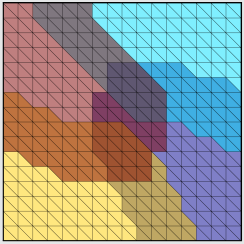
In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$

Computation of the overlapping matrices

The overlapping matrices

$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

can easily be extracted from K since R_i is just a **global-to-local index mapping**.

Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Algorithmic Framework for FROSch Coarse Spaces

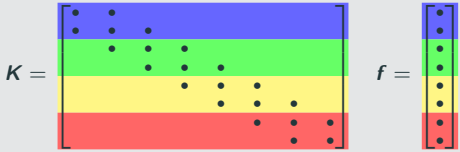
FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

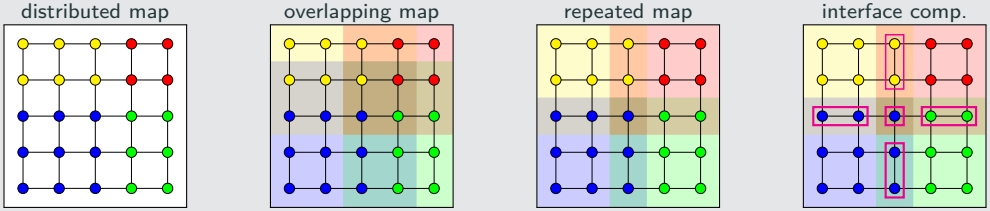
1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Identification of the domain decomposition interface

If not provided by the user, FROSch will construct a **repeated map** where the interface (Γ) nodes are shared between processes from the parallel distribution of the matrix rows (**distributed map**).

Then, FROSch automatically identifies vertices, edges, and (in 3D) faces, by the multiplicities of the nodes.





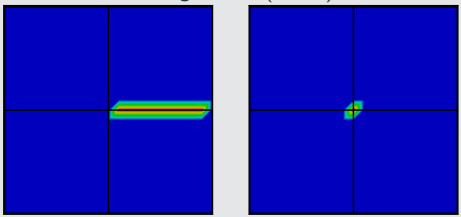
Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

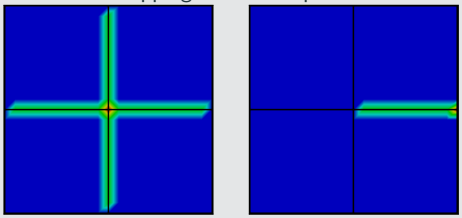
1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Construction of a partition of unity on the interface

vertices, edges, and (in 3D) faces



overlapping vertex components

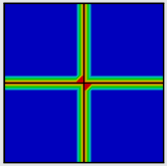


We construct a **partition of unity (POU)** $\{\pi_i\}_i$ with

$$\sum_i \pi_i = 1$$

on the interface Γ.

$\Rightarrow$



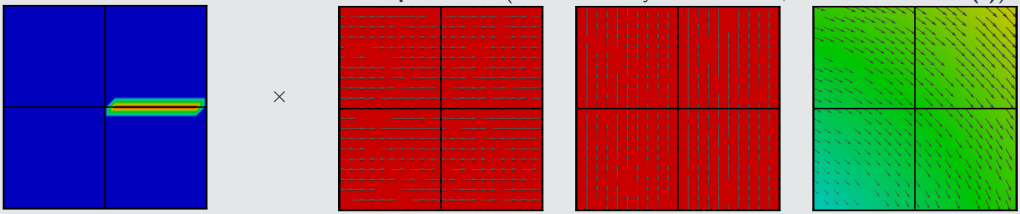
Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Computation of a coarse basis on the interface

interface POU function $\times$ **null space basis** (linear elasticity: **translations**, **linearized rotation(s)**)



For each partition of unity function π_i, we compute a basis for the space

$$\text{span}\left(\{\pi_i \times z_j\}_j\right),$$

where $\{z_j\}_j$ is a null space basis. In case of **linear dependencies**, we perform a **local QR factorization** to construct a basis.

This yields an **interface coarse basis** Φ_Γ.

The linearized rotation

$$\begin{bmatrix} y \\ -x \end{bmatrix}$$

depends on coordinates (geometric information).

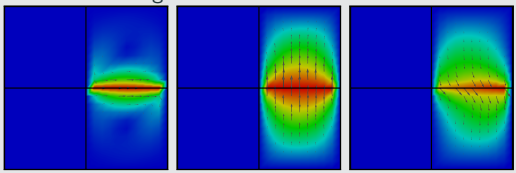
Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

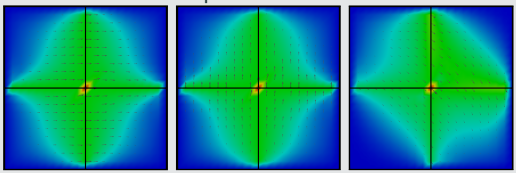
1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Harmonic extensions into the interior

edge coarse basis functions



vertex component basis functions



For each **interface coarse basis function**, we compute the interior values Φ_I by computing **harmonic / energy-minimizing extensions**:

$$\Phi = \begin{bmatrix} -\boldsymbol{K}_{II}^{-1}\boldsymbol{K}_{I\Gamma}^{T}\Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix}.$$

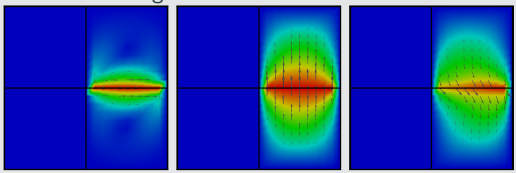
Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

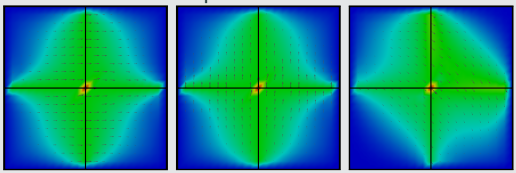
1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Harmonic extensions into the interior

edge coarse basis functions



vertex component basis functions



For each **interface coarse basis function**, we compute the interior values Φ_I by computing **harmonic / energy-minimizing extensions**:

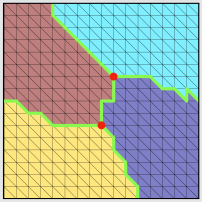
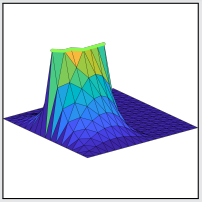
$$\Phi = \begin{bmatrix} -\boldsymbol{K}_{II}^{-1}\boldsymbol{K}_{I\Gamma}^{T}\Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix}.$$

Resulting condition number bound:

$$\kappa\left(\boldsymbol{M}_{\text{OS-2}}^{-1}\boldsymbol{K}\right) \leq C\left(1+\frac{H}{\delta}\right)\left(1+\log\left(\frac{H}{h}\right)\right)^2$$

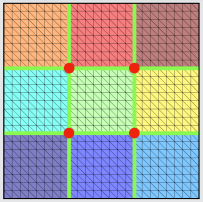
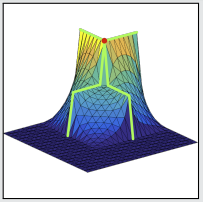
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

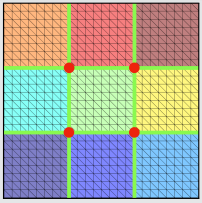
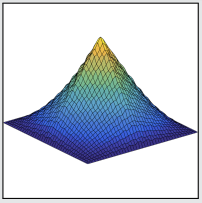
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

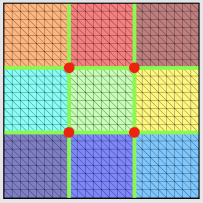
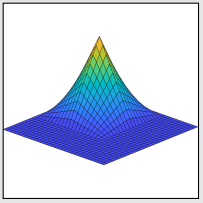
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)

- Hou (1997), Efendiev and Hou (2009)
- Buck, Iliev, and Andrä (2013)
- H., Klawonn, Knepper, Rheinbach (2018)

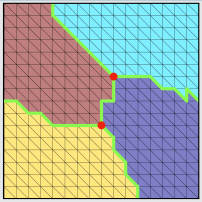
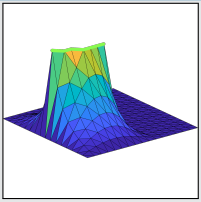
Q1 Lagrangian / piecewise bilinear

Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

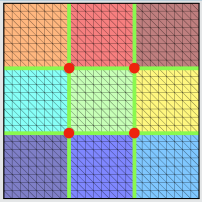
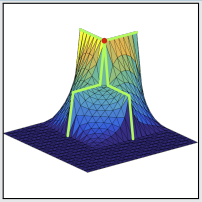
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

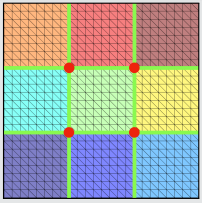
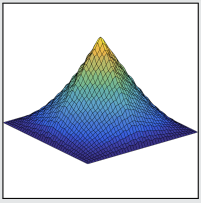
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

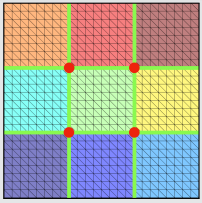
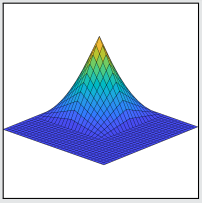
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)

- Hou (1997), Efendiev and Hou (2009)
- Buck, Iliev, and Andrä (2013)
- H., Klawonn, Knepper, Rheinbach (2018)

Q1 Lagrangian / piecewise bilinear

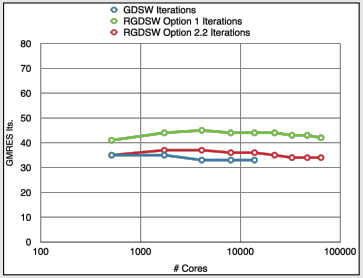



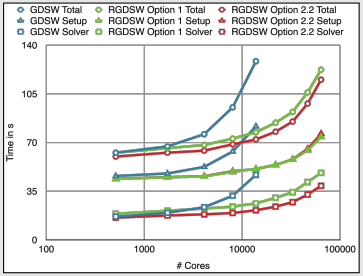
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

Weak Scalability up to $64\,\mathrm{k}$ MPI Ranks / $1.7\,\mathrm{b}$ Unknowns (3D Poisson; Juqueen)

GDSW vs RGDSW (reduced dimension)

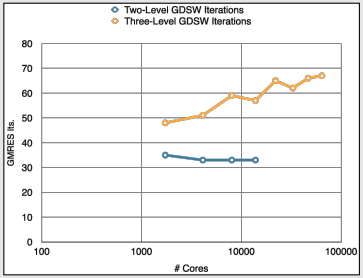
Heinlein, Klawonn, Rheinbach, Widlund (2019).

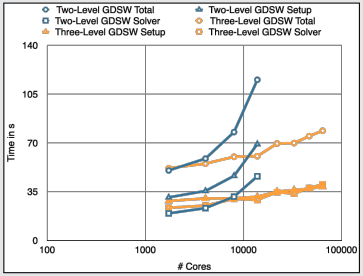




Two-level vs three-level GDSW

Heinlein, Klawonn, Rheinbach, Röver (2019, 2020).





Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

$$\mathcal{A}x = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} = b.$$

Monolithic GDSW preconditioner

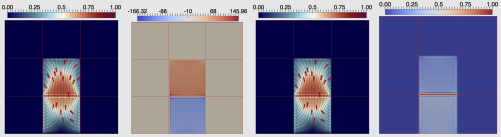
We construct a **monolithic GDSW preconditioner**

$$M_{\text{GDSW}}^{-1} = \phi \mathcal{A}_0^{-1} \phi^\top + \sum_{i=1}^{N} \mathcal{R}_i^\top \mathcal{A}_i^{-1} \mathcal{R}_i,$$

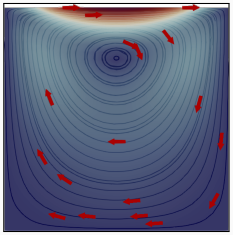
with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$, and

$$\mathcal{R}_i = \begin{bmatrix} \mathcal{R}_{u,i} & \boldsymbol{0} \\ \boldsymbol{0} & \mathcal{R}_{p,i} \end{bmatrix} \quad \text{and} \quad \phi = \begin{bmatrix} \Phi_{u,u_0} & \Phi_{u,p_0} \\ \Phi_{p,u_0} & \Phi_{p,p_0} \end{bmatrix}.$$

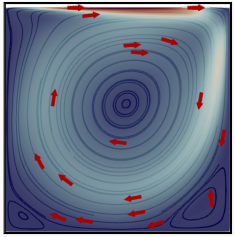
Using $\mathcal{A}$ to compute extensions: $\phi_I = -\mathcal{A}_{II}^{-1} \mathcal{A}_{I\Gamma} \phi_\Gamma$; cf. Heinlein, Hochmuth, Klawonn (2019, 2020).



Φ_{u,u_0} Φ_{p,u_0} Φ_{u,p_0} Φ_{p,p_0}



Stokes flow



Navier–Stokes flow

Related work:

- Original work on monolithic Schwarz preconditioners: Klawonn and Pavarino (1998, 2000)
- Other publications on monolithic Schwarz preconditioners: e.g., Hwang and Cai (2006), Barker and Cai (2010), Wu and Cai (2014), and the presentation Dohrmann (2010) at the *Workshop on Adaptive Finite Elements and Domain Decomposition Methods* in Milan.

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

$$\mathcal{A}x = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} = b.$$

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

$$M_{\text{GDSW}}^{-1} = \phi \mathcal{A}_0^{-1} \phi^\top + \sum_{i=1}^{N} \mathcal{R}_i^\top \mathcal{A}_i^{-1} \mathcal{R}_i,$$

with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$.

Block diagonal & triangular preconditioners

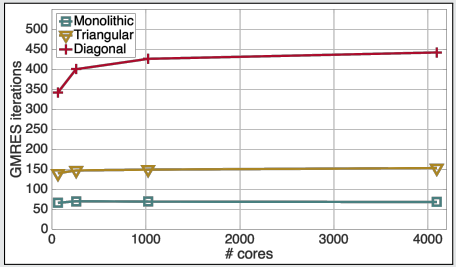
Block-diagonal preconditioner:

$$M_{\text{D}}^{-1} = \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ 0 & \boldsymbol{S}^{-1} \end{bmatrix} \approx \begin{bmatrix} M_{\text{GDSW}}^{-1}(\boldsymbol{K}) & 0 \\ 0 & M_{\text{OS-1}}^{-1}(\boldsymbol{M_p}) \end{bmatrix}$$

Block-triangular preconditioner:

$$M_{\text{T}}^{-1} = \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\boldsymbol{S}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \boldsymbol{S}^{-1} \end{bmatrix} \approx \begin{bmatrix} M_{\text{GDSW}}^{-1}(\boldsymbol{K}) & 0 \\ -M_{\text{OS-1}}^{-1}(\boldsymbol{M_p})\boldsymbol{B}M_{\text{GDSW}}^{-1}(\boldsymbol{K}) & M_{\text{OS-1}}^{-1}(\boldsymbol{M_p}) \end{bmatrix}$$

Monolithic vs. block prec. (Stokes)



prec.	# MPI ranks	64	256	1 024	4 096
mono.	time	154.7 s	170.0 s	175.8 s	188.7 s
	effic.	**100 %**	91 %	88 %	82 %
triang.	time	309.4 s	329.1 s	359.8 s	396.7 s
	effic.	50 %	47 %	43 %	39 %
diag.	time	736.7 s	859.4 s	966.9 s	1 105.0 s
	effic.	21 %	18 %	16 %	14 %

Computations performed on **magnitUDE (University Duisburg-Essen)**.

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

$$\mathcal{A}x = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} = b.$$

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

$$M_{\text{GDSW}}^{-1} = \phi \mathcal{A}_0^{-1} \phi^\top + \sum_{i=1}^{N} \mathcal{R}_i^\top \mathcal{A}_i^{-1} \mathcal{R}_i,$$

with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$.

SIMPLE block preconditioner

We employ the **SIMPLE (Semi-Implicit Method for Pressure Linked Equations)** block preconditioner

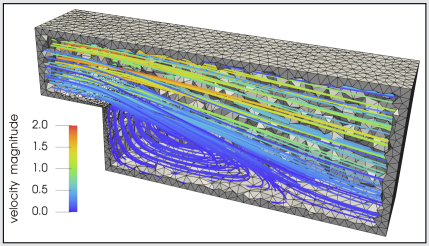
$$M_{\text{SIMPLE}}^{-1} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}^{-1}\boldsymbol{B} \\ 0 & \alpha \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\hat{\boldsymbol{S}}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \hat{\boldsymbol{S}}^{-1} \end{bmatrix};$$

see Patankar and Spalding (1972). Here,

- $\hat{\boldsymbol{S}} = -\boldsymbol{B}\boldsymbol{D}^{-1}\boldsymbol{B}^\top$, with $\boldsymbol{D} = \operatorname{diag} \boldsymbol{K}$
- α is an under-relaxation parameter

We **approximate the inverses** using (R)GDSW preconditioners.

Monolithic vs. SIMPLE preconditioner



Steady-state Navier–Stokes equations

prec.	# MPI ranks	243	1 125	15 562
Monolithic	setup	39.6 s	57.9 s	95.5 s
RGDSW	solve	57.6 s	69.2 s	74.9 s
(FROSch)	total	**97.2 s**	**127.7 s**	**170.4 s**
SIMPLE	setup	39.2 s	38.2 s	68.6 s
RGDSW (Teko	solve	86.2 s	106.6 s	127.4 s
& FROSch)	total	125.4 s	144.8 s	196.0 s

Computations on Piz Daint (CSCS). Implementation in the finite element software FEDDLib.

Flexible Construction of Monolithic Extension-Based Coarse Spaces in FROSch

FROSch allows for the **flexible construction of extension-based coarse spaces**. For monolithic preconditioners, the interface basis for the different blocks in the system

$$\mathcal{A}x = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{B}^\top \\ \boldsymbol{B} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{p} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} = b.$$

can be **defined independently**

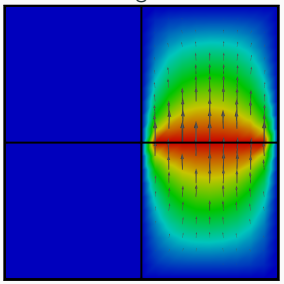
$$\phi_\Gamma = \begin{bmatrix} \phi_{\Gamma,u} & \boldsymbol{0} \\ \boldsymbol{0} & \phi_{\Gamma,p} \end{bmatrix}$$

and then extended in a monolithic way

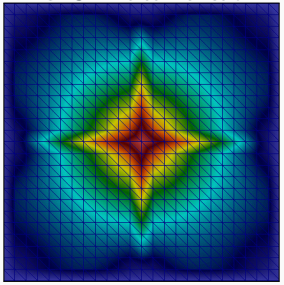
$$\phi = \begin{bmatrix} \phi_I \\ \phi_\Gamma \end{bmatrix} = \begin{bmatrix} -\mathcal{A}_{II}^{-1}\mathcal{A}_{I\Gamma}\phi_\Gamma \\ \phi_\Gamma \end{bmatrix}.$$

As a result, we can have use a different coarse space one each of the blocks.

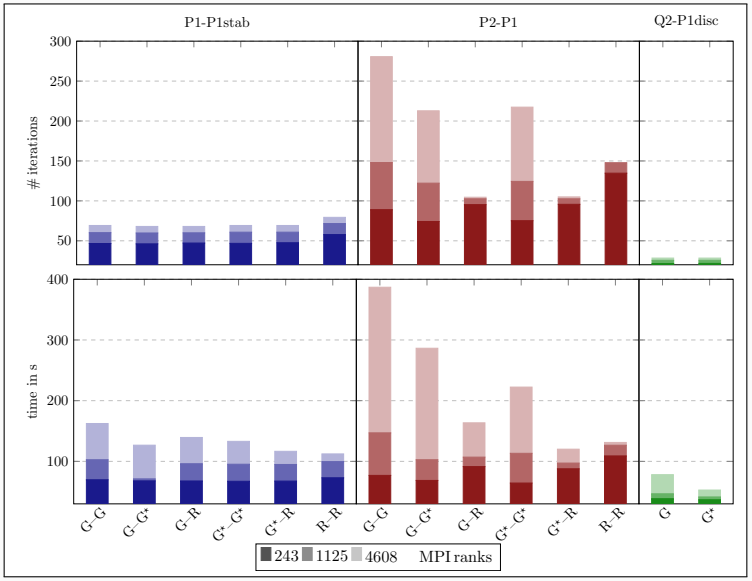
GDSW edge function



RGDSW vertex function



Coarse Spaces for Monolithic FROSch Preconditioners for CFD Simulations

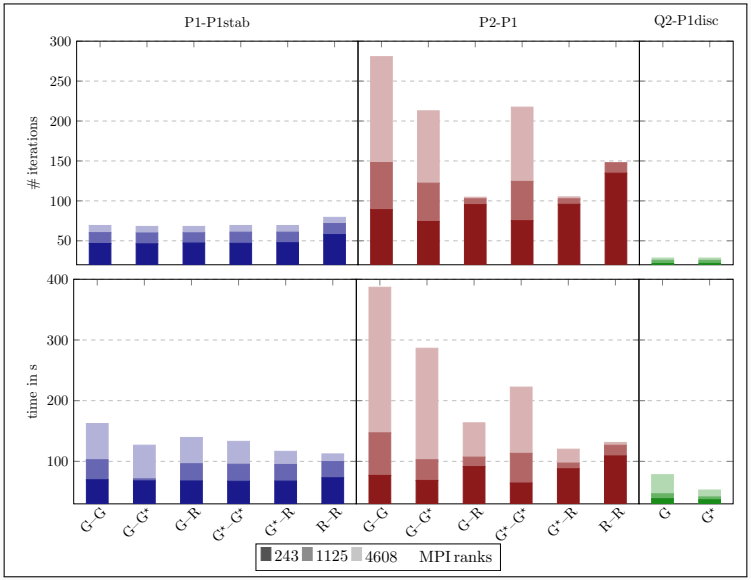


Comparison of coarse spaces

Coarse spaces based on different interface partitions of unity (IPOU):

- **G** (GDSW):
  IPOU: faces, edges, vertices
- **G*** (GDSW*):
  IPOU: faces, vertex-based
- **R** (RGDSW):
  IPOU: vertex-based

Coarse Spaces for Monolithic FROSch Preconditioners for CFD Simulations



Comparison of coarse spaces

Coarse spaces based on different interface partitions of unity (IPOU):

- **G** (GDSW):
  IPOU: faces, edges, vertices
- **G*** (GDSW*):
  IPOU: faces, vertex-based
- **R** (RGDSW):
  IPOU: vertex-based

$\Rightarrow$ Generally **good performance** for **stabilized or discontinuous pressure discretizations**. Otherwise, performance depends on the **combination of velocity and pressure coarse spaces**.

Monolithic and SIMPLE Preconditioners for FSI

Monolithic fluid–structure interaction (FSI)

We consider the monolithic FSI problem

$$\left[\begin{array}{cc|c|c} \boldsymbol{F}(\boldsymbol{u}_f^{n+1}) & \boldsymbol{0} & \boldsymbol{C}_1^\top \boldsymbol{\lambda}^{n+1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S}(\mathbf{d}_s^{n+1}) & \boldsymbol{C}_2^\top \boldsymbol{\lambda}^{n+1} & \boldsymbol{0} \\ \hline \boldsymbol{C}_1 \mathbf{u}_f^{n+1} & \boldsymbol{C}_2 \mathbf{d}_s^{n+1}/\Delta t & \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{C}_3 \mathbf{d}_s^{n+1} & \boldsymbol{0} & \boldsymbol{G}\mathbf{d}_f^{n+1} \end{array}\right] = \left[\begin{array}{c} \mathbf{b}_f \\ \mathbf{b}_s \\ \hline \boldsymbol{0} \\ \hline \boldsymbol{0} \end{array}\right]$$

- $\boldsymbol{\lambda}$: Lagrange multipliers enforcing stress balance.
- $\boldsymbol{F}$, $\boldsymbol{S}$, and $\boldsymbol{G}$: fluid, structure, and geometry operators.
- $\boldsymbol{C}_1$, $\boldsymbol{C}_2$, and $\boldsymbol{C}_3$: transfer operators.

Linearization

In each time step, we solve the FSI system using **Newton's method**

$$\mathcal{J}_{\text{FSI}}(\mathbf{x}_k^{n+1})\delta_{k+1} = -\mathbf{r}(\mathbf{x}_k^{n+1}),$$

with the Jacobian:

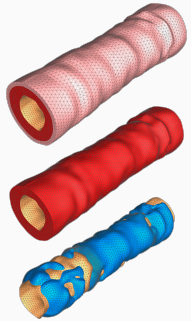
$$\mathcal{J}_{\text{FSI}} = \left[\begin{array}{cc|c|c} \boldsymbol{D}_{\boldsymbol{u}}\boldsymbol{F} & \boldsymbol{0} & \boldsymbol{C}_1^\top & \boldsymbol{D}_{\boldsymbol{d}_f}\boldsymbol{F} \\ \boldsymbol{0} & \boldsymbol{D}_{\boldsymbol{d}_s}\boldsymbol{S} & \boldsymbol{C}_2^\top & \boldsymbol{0} \\ \hline \boldsymbol{C}_1 & \boldsymbol{C}_2/\Delta t & \boldsymbol{0} & \boldsymbol{0} \\ \hline \boldsymbol{0} & \boldsymbol{C}_3 & \boldsymbol{0} & \boldsymbol{G} \end{array}\right]$$

We solve the linearized systems using GMRES using the FaCSI (**Fa**ctorization – static **C**ondensation – **SI**MPLE) preconditioner introduced in **(Deparis et al., 2015)**:

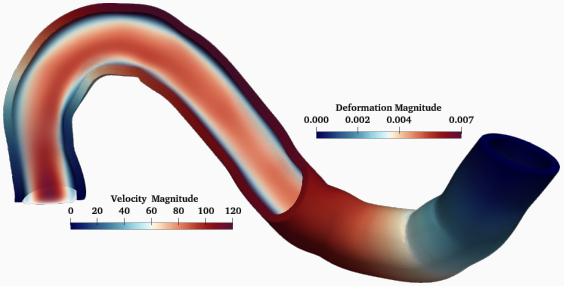
$$\mathcal{J}_{\text{FSI}} \approx \underbrace{\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}}_{=\mathcal{P}_{\boldsymbol{S}}} \underbrace{\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}_3 & \boldsymbol{0} & \boldsymbol{G} \end{bmatrix}}_{\mathcal{P}_{\boldsymbol{G}}} \underbrace{\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{D} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}_2 & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}}_{\mathcal{P}_{\boldsymbol{F}}^{(1)}} \underbrace{\begin{bmatrix} \boldsymbol{F} & \boldsymbol{0} & \boldsymbol{C}_1^\top & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C}_1 & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}}_{\mathcal{P}_{\boldsymbol{F}}^{(2)}}$$

Here, in $\mathcal{P}_{\boldsymbol{F}}^{(2)}$,

- we formally **eliminate the interface degrees of freedom via static condensation** and
- apply approximate the inverse via a **monolithic or SIMPLE preconditioner**.

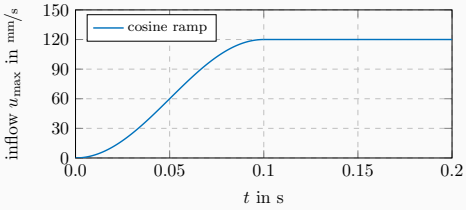


Results: Block and Monolithic Preconditioner in Fluid Component in FSI



fluid	Navier–Stokes	P2–P1
solid	St. Venant Kirchhoff	P2
geometry	Laplace	P2

≈ 1.0 M degrees of freedom



Strong scaling: comparison of **preconditioners for the fluid block F**:

Monolithic RGDSW preconditioner

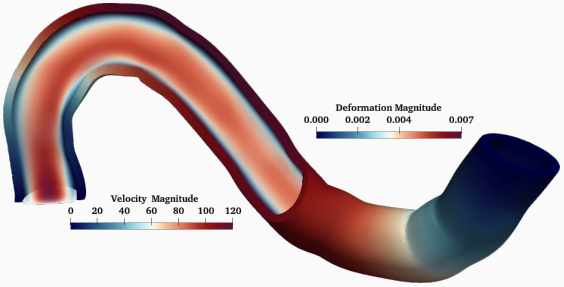
# MPI ranks	$\overline{\#its}$	setup	solve	total
60	**48.3**	318.4 s	529.1 s	**847.5** s
120	**60.4**	204.7 s	411.4 s	**616.1** s
240	**64.8**	128.8 s	279.2 s	**408.0** s
360	**64.7**	100.4 s	228.7 s	**329.1** s
480	**64.8**	90.9 s	228.2 s	**319.2** s
600	**63.9**	85.2 s	227.4 s	**312.6** s

SIMPLE RGDSW preconditioner

# MPI ranks	$\overline{\#its}$	setup	solve	total
60	54.8	306.4 s	685.3 s	991.7 s
120	68.3	183.1 s	484.7 s	667,8 s
240	79.6	114.9 s	349.1 s	464.0 s
360	83.7	91.8 s	290.6 s	382.4 s
480	92.3	82.0 s	302.2 s	384.2 s
600	97.3	76.6 s	295.2 s	371.8 s

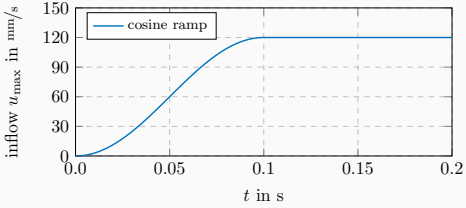
Computations on Fritz (FAU). Implementation in the finite element software FEDDLib. Overlap $\delta/h = 1$

Results: Block and Monolithic Preconditioner in Fluid Component in FSI



fluid	Navier–Stokes	P2–P1
solid	St. Venant Kirchhoff	P2
geometry	Laplace	P2

≈ 1.0 M degrees of freedom



Strong scaling: comparison of **preconditioners for the fluid block F**:

Monolithic RGDSW preconditioner

# MPI ranks	$\overline{\#its}$	setup	solve	total
60	**48.3**	318.4 s	529.1 s	**847.5** s
120	**60.4**	204.7 s	411.4 s	**616.1** s
240	**64.8**	128.8 s	279.2 s	**408.0** s
360	**64.7**	100.4 s	228.7 s	**329.1** s
480	**64.8**	90.9 s	228.2 s	**319.2** s
600	**63.9**	85.2 s	227.4 s	**312.6** s

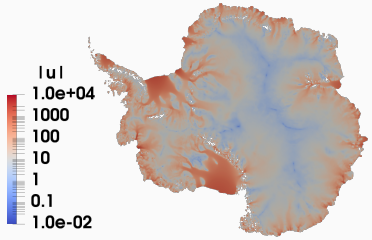
SIMPLE RGDSW preconditioner

# MPI ranks	$\overline{\#its}$	setup	solve	total
60	54.8	306.4 s	685.3 s	991.7 s
120	68.3	183.1 s	484.7 s	667,8 s
240	79.6	114.9 s	349.1 s	464.0 s
360	83.7	91.8 s	290.6 s	382.4 s
480	92.3	82.0 s	302.2 s	384.2 s
600	97.3	76.6 s	295.2 s	371.8 s

Computations on Fritz (FAU). Implementation in the finite element software FEDDLib. Overlap $\delta/h = 1$

FROSch Preconditioners for Land Ice Simulations

https://github.com/SNLComputation/Albany



The velocity of the ice sheet in Antarctica and Greenland is modeled by a **first-order-accurate Stokes approximation model**,

$$-\nabla\cdot(2\mu\dot{\epsilon}_1)+\rho g\frac{\partial s}{\partial x}=0,\quad -\nabla\cdot(2\mu\dot{\epsilon}_2)+\rho g\frac{\partial s}{\partial y}=0,$$

with a **nonlinear viscosity model** (Glen's law); cf., e.g., **Blatter (1995)** and **Pattyn (2003)**.

	Antarctica (**velocity**) 4 km resolution, 20 layers, 35 m dofs			Greenland (**multiphysics vel. & temperature**) 1-10 km resolution, 20 layers, 69 m dofs		
MPI ranks	avg. its	avg. setup	avg. solve	avg. its	avg. setup	avg. solve
512	41.9 (11)	25.10 s	12.29 s	41.3 (36)	18.78 s	4.99 s
1 024	43.3 (11)	9.18 s	5.85 s	53.0 (29)	8.68 s	4.22 s
2 048	41.4 (11)	4.15 s	2.63 s	62.2 (86)	4.47 s	4.23 s
4 096	41.2 (11)	1.66 s	1.49 s	68.9 (40)	2.52 s	2.86 s
8 192	40.2 (11)	1.26 s	1.06 s	-	-	-

Computations performed on Cori (NERSC).

Heinlein, Perego, Rajamanickam (2022)

Summary

- FROSch is based on the **Schwarz framework** and **extension-based coarse spaces**, which provide **numerical scalability** using **only algebraic information** for a **variety of problems**, including **fluid**, **solid**, and **multiphysics** problems.
- We presented a detailed study
  - **comparing monolithic and block preconditioners** and
  - **parallel scalability results** for up to more than **15 k MPI ranks (= cores)**

  for **fluid and fluid–structure interaction problems**.

Outlook

- **Theoretical investigations of monolithic extension-based coarse spaces**

Acknowledgements

- **Financial support:** DFG (KL2094/3-1, RH122/4-1), DFG SPP 2311 project number 465228106, DOE SciDAC-5 FASTMath Institute (Contract no. DE-AC02-05CH11231)
- **Computing resources:** Summit (OLCF), Cori (NERSC), magnitUDE (UDE), Piz Daint (CSCS), Fritz (FAU)

Thank you for your attention!