Fast and Robust Overlapping Schwarz (FROSch) Domain Decomposition Preconditioners

Alexander Heinlein[1]

Seminar, University of Macau, Macao, August 7, 2024

[1] Delft University of Technology

Outline

1. One- and Two-Level Schwarz Preconditioners

2. The FROSch Package – Algebraic and Parallel Schwarz Preconditioners in Trilinos

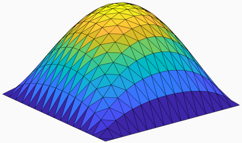
3. Coarse Spaces for Some Challenging Problems

4. Accelerating Time-to-Solution

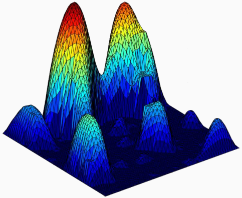
5. Learning Extension Operators Using Graph Neural Networks

One- and Two-Level Schwarz Preconditioners

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 $\frac{\max_x \alpha(x)}{\min_x \alpha(x)}$

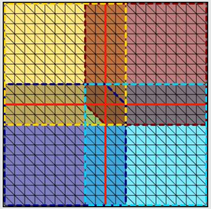
⇒ We introduce a preconditioner $\boldsymbol{M}^{-1} \approx \boldsymbol{A}^{-1}$ to improve the condition number:

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

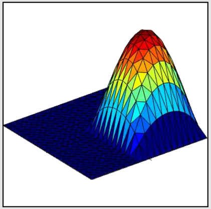
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



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

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

where R_i and $R_i^\top$ are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^\top$.

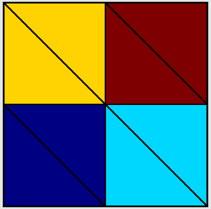
Condition number estimate:

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

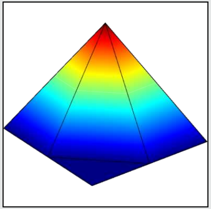
with subdomain size H and overlap width δ.

Lagrangian coarse space

Coarse triangulation



Coarse solution



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

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

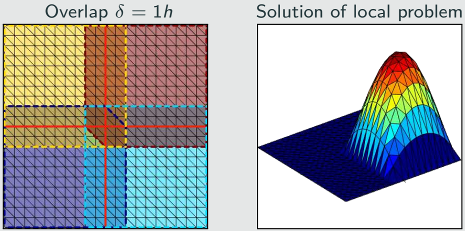
where Φ contains the coarse basis functions and $K_0 := \Phi^\top K \Phi$; cf., e.g., **Toselli, Widlund (2005)**. The construction of a Lagrangian coarse basis requires a coarse triangulation.

Condition number estimate:

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

Two-Level Schwarz Preconditioners

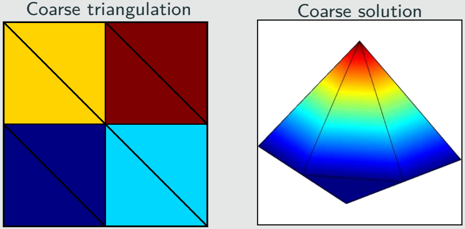
One-level Schwarz preconditioner



Overlap $\delta = 1h$

Solution of local problem

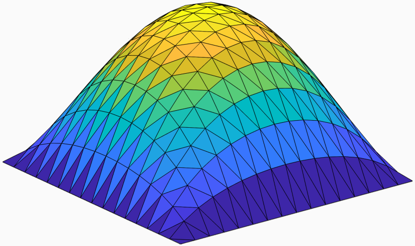
Lagrangian coarse space

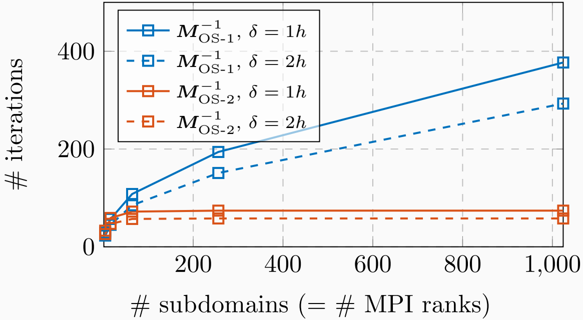


Coarse triangulation

Coarse solution

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




$M_{\text{OS-1}}^{-1}$, $\delta = 1h$
$M_{\text{OS-1}}^{-1}$, $\delta = 2h$
$M_{\text{OS-2}}^{-1}$, $\delta = 1h$
$M_{\text{OS-2}}^{-1}$, $\delta = 2h$
# iterations
0
200
400
200
400
600
800
1,000
# subdomains (= # MPI ranks)

The FROSch Package – Algebraic and Parallel Schwarz Preconditioners in Trilinos

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)
- Jascha Knepper (UCologne)
- Friederike Röver (TUBAF)
- Lea Saßmannshausen (UCologne)
- Alexander Heinlein (TU Delft)
- Axel Klawonn (UCologne)
- Siva Rajamanickam (SNL)
- Oliver Rheinbach (TUBAF)
- Ichitaro Yamazaki (SNL)

Partition of Unity

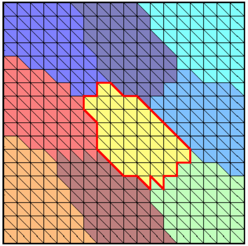
The **energy-minimizing extension** $v_i = H_{\partial\Omega_i \to \Omega_i}(v_{i,\partial\Omega_i})$ solves

$$\begin{aligned} -\Delta v_i &= 0 && \text{in } \Omega_i, \\ v_i &= v_{i,\partial\Omega_i} && \text{on } \partial\Omega_i. \end{aligned}$$

Hence, $v_i = E_{\partial\Omega_i \to \Omega_i}(\mathbb{1}_{\partial\Omega_i}) = \mathbb{1}$.

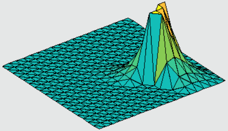
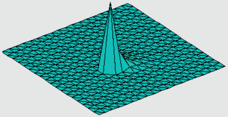
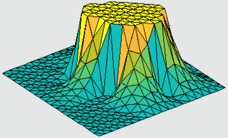
Due to **linearity of the extension operator**, we have

$$\sum_i \varphi_i = \mathbb{1}_{\partial\Omega_i} \Rightarrow \sum_i E_{\partial\Omega_i \to \Omega_i}(\varphi_i) = \mathbb{1}_{\Omega_i}$$



Null space property

Any extension-based coarse space built from a partition of unity on the domain decomposition interface satisfies the **null space property necessary for numerical scalability**:

$\sum_{\text{edges} \subset \partial\Omega_i}$  $+ \sum_{\text{vertices} \subset \partial\Omega_i}$  $=$ 

Algebraicity of the energy-minimizing extension

The computation of energy-minimizing extensions only requires $\boldsymbol{K}_{II}$ and $\boldsymbol{K}_{I\Gamma}$, **submatrices of the fully assembled matrix** $\boldsymbol{K}_i$.

$$\boldsymbol{v} = \begin{bmatrix} -\boldsymbol{K}_{II}^{-1}\boldsymbol{K}_{I\Gamma} \\ \boldsymbol{I}_{\Gamma} \end{bmatrix} \boldsymbol{v}_{\Gamma},$$

Algorithmic Framework for FROSch Preconditioners

Overlapping domain decomposition

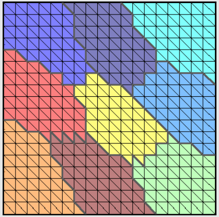
The **overlapping subdomains** are constructed by **recursively adding layers of elements** via the sparsity pattern of $\boldsymbol{K}$.

The corresponding matrices

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

can easily be extracted from $\boldsymbol{K}$.

Nonoverlapping DD



Algorithmic Framework for FROSch Preconditioners

Overlapping domain decomposition

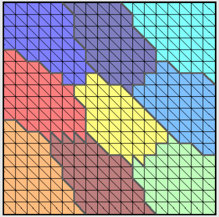
The **overlapping subdomains** are constructed by **recursively adding layers of elements** via the sparsity pattern of $\boldsymbol{K}$.

The corresponding matrices

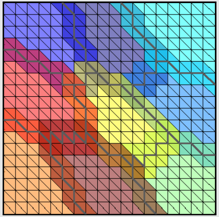
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

can easily be extracted from $\boldsymbol{K}$.

Nonoverlapping DD



Overlap $\delta = 1h$



Algorithmic Framework for FROSch Preconditioners

Overlapping domain decomposition

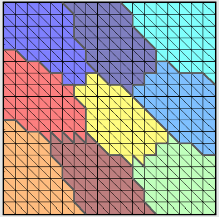
The **overlapping subdomains** are constructed by **recursively adding layers of elements** via the sparsity pattern of $\boldsymbol{K}$.

The corresponding matrices

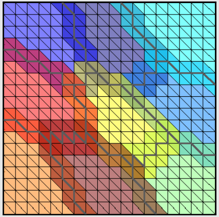
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

can easily be extracted from $\boldsymbol{K}$.

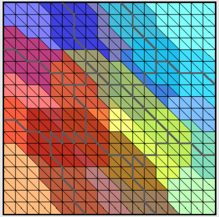
Nonoverlapping DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$



Algorithmic Framework for FROSch Preconditioners

Overlapping domain decomposition

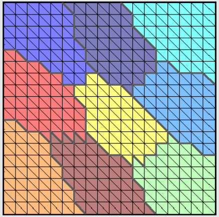
The **overlapping subdomains** are constructed by **recursively adding layers of elements** via the sparsity pattern of $\boldsymbol{K}$.

The corresponding matrices

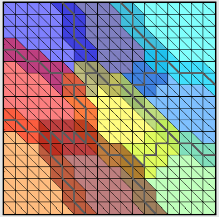
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

can easily be extracted from $\boldsymbol{K}$.

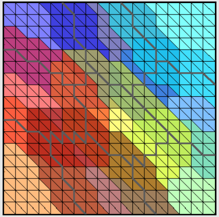
Nonoverlapping DD



Overlap $\delta = 1h$

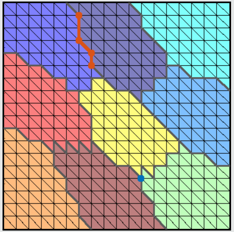


Overlap $\delta = 2h$



Coarse space

1. Interface components



Algorithmic Framework for FROSch Preconditioners

Overlapping domain decomposition

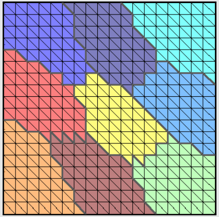
The **overlapping subdomains** are constructed by **recursively adding layers of elements** via the sparsity pattern of $\boldsymbol{K}$.

The corresponding matrices

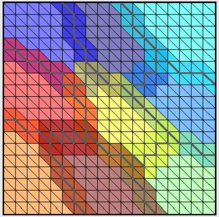
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

can easily be extracted from $\boldsymbol{K}$.

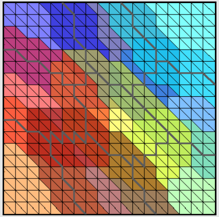
Nonoverlapping DD



Overlap $\delta = 1h$

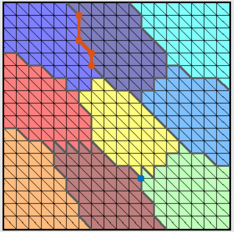


Overlap $\delta = 2h$

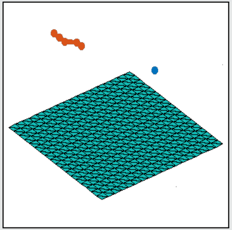


Coarse space

1. Interface components



2. Interface basis (partition of unity × null space)



For **scalar elliptic problems**, the **null space** consists only of **constant functions**.

Algorithmic Framework for FROSch Preconditioners

Overlapping domain decomposition

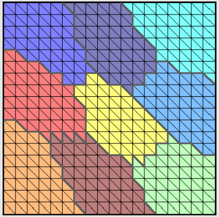
The **overlapping subdomains** are constructed by **recursively adding layers of elements** via the sparsity pattern of $\boldsymbol{K}$.

The corresponding matrices

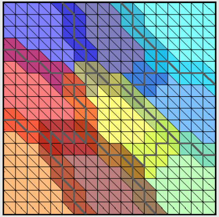
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

can easily be extracted from $\boldsymbol{K}$.

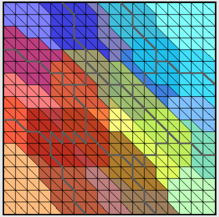
Nonoverlapping DD



Overlap $\delta = 1h$

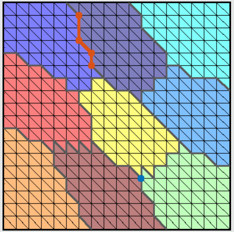


Overlap $\delta = 2h$

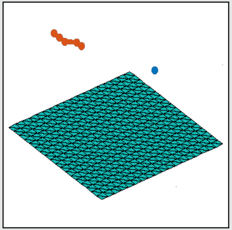


Coarse space

1. Interface components

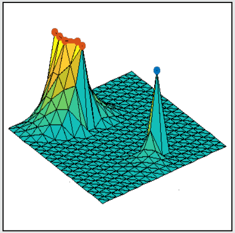


2. Interface basis (partition of unity × null space)



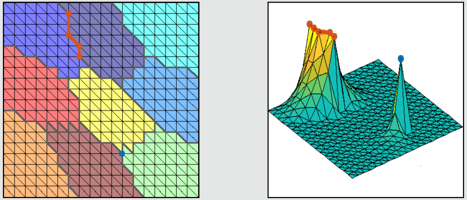
For **scalar elliptic problems**, the **null space** consists only of **constant functions**.

3. Extension



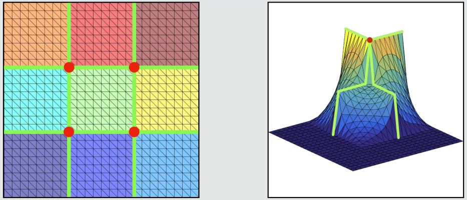
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)



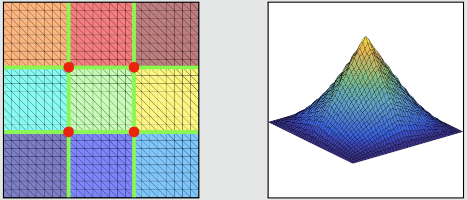
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)



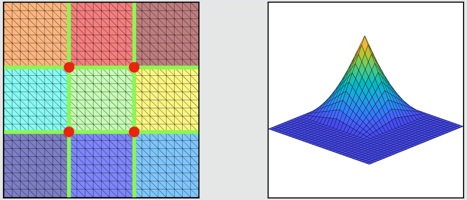
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

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

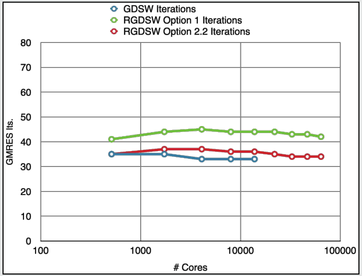


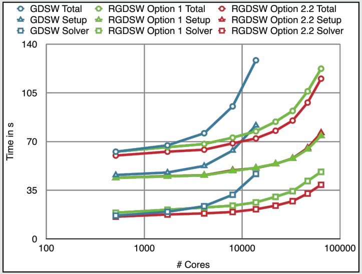
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

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

GDSW vs RGDSW (reduced dimension)

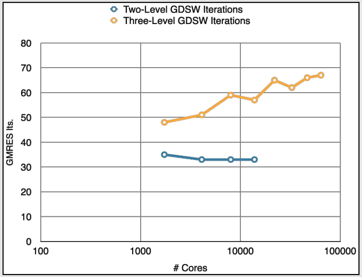
Heinlein, Klawonn, Rheinbach, Widlund (2019).

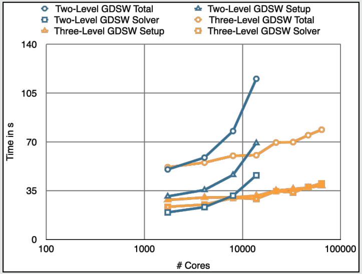




Two-level vs three-level GDSW

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



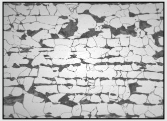


Coarse Spaces for Some Challenging Problems

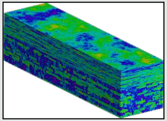
Spectral Extension-Based Coarse Spaces for Schwarz Preconditioners

Highly heterogeneous problems ...

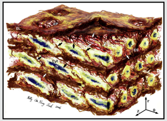
...appear in most areas of modern science and engineering:



Micro section of a dual-phase steel. Courtesy of **J. Schröder**.



Groundwater flow (SPE10); cf. **Christie and Blunt (2001)**.



Composition of arterial walls; taken from **O'Connell et al. (2008)**.

Spectral coarse spaces

The coarse space is **enhanced** by eigenfunctions of **local edge and face eigenvalue problems** with eigenvalues below tolerances $tol_{\mathcal{E}}$ and $tol_{\mathcal{F}}$:

$$\kappa\left(M_{*}^{-1}K\right) \leq C\left(1+\frac{1}{tol_{\mathcal{E}}}+\frac{1}{tol_{\mathcal{F}}}+\frac{1}{tol_{\mathcal{E}}\cdot tol_{\mathcal{F}}}\right);$$

C does not depend on h, H, or the coefficients.
OS-ACMS & **adaptive GDSW (AGDSW)** (**Heinlein, Klawonn, Knepper, Rheinbach (2018, 2018, 2019)**).

Related works (non-exhaustive)

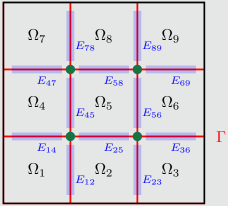
- **FETI & Neumann–Neumann: Bjørstad and Krzyzanowski (2002); Bjørstad, Koster, and Krzyzanowski (2001); Rixen and Spillane (2013); Spillane (2015, 2016)** ...
- **BDDC & FETI-DP: Mandel and Sousedík (2007); Sousedík (2010); Sístek, Mandel, and Sousedík (2012); Dohrmann and Pechstein (2013, 2016); Klawonn, Radtke, and Rheinbach (2014, 2015, 2016); Klawonn, Kühn, and Rheinbach (2015, 2016, 2017); Kim and Chung (2015); Kim, Chung, and Wang (2017); Beirão da Veiga et al. (2017); Calvo and Widlund (2016); Oh et al. (2017)** ...
- **Overlapping Schwarz: Galvis and Efendiev (2010, 2011); Nataf, Xiang, Dolean, and Spillane (2011); Spillane, Dolean, Hauret, Nataf, Pechstein, and Scheichl (2011); Gander, Loneland, and Rahman (preprint 2015); Eikeland, Marcinkowski, and Rahman (TR 2016); Marcinkowski and Rahman (2018)** ...
- **Spectral AMGe (ρAMGe)**: **Chartier, Falgout, Henson, Jones, Manteuffel, McCormick, Ruge, and Vassilevski (2003)** ...

Spectral Extension-Based Coarse Spaces for Schwarz Preconditioners

Local eigenvalue problems

Local generalized eigenvalue problems corresponding to the edges $\mathcal{E}$ and faces $\mathcal{F}$ of the domain decomposition:

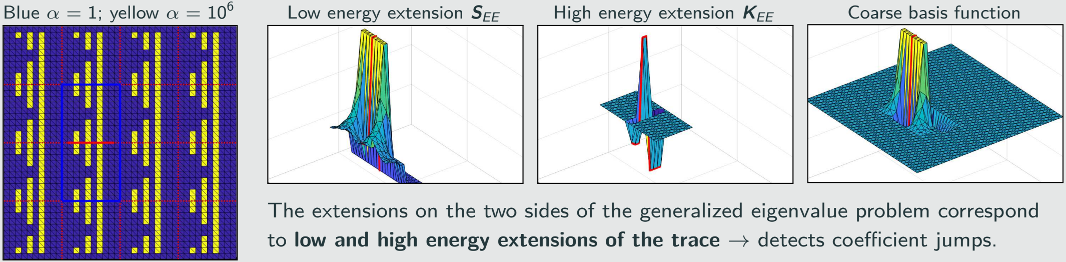
$$\forall E \in \mathcal{E}: \qquad \boldsymbol{S}_{EE}\tau_{*,E} = \lambda_{*,E}\boldsymbol{K}_{EE}\tau_{*,E}, \quad \forall \tau_{*,E} \in V_E,$$
$$\forall F \in \mathcal{F}: \qquad \boldsymbol{S}_{FF}\tau_{*,F} = \lambda_{*,F}\boldsymbol{K}_{FF}\tau_{*,F}, \quad \forall \tau_{*,F} \in V_F,$$



with **Schur complements** S_{EE}, S_{FF} with **Neumann boundary conditions** and **submatrices** K_{EE}, K_{FF} of K. We select eigenfunctions corresponding to **eigenvalues below tolerances** $tol_{\mathcal{E}}$ and $tol_{\mathcal{F}}$.

→ The corresponding coarse basis functions are **energy-minimizing extensions** into the interior of the subdomains.

Extensions in the generalized eigenvalue problem

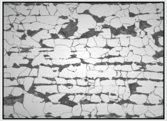


The extensions on the two sides of the generalized eigenvalue problem correspond to **low and high energy extensions of the trace** → detects coefficient jumps.

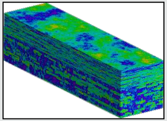
Spectral Extension-Based Coarse Spaces for Schwarz Preconditioners

Highly heterogeneous problems ...

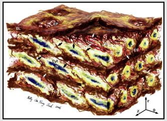
... appear in most areas of modern science and engineering:



Micro section of a dual-phase steel. Courtesy of **J. Schröder**.



Groundwater flow (SPE10); cf. **Christie and Blunt (2001)**.



Composition of arterial walls; taken from **O'Connell et al. (2008)**.

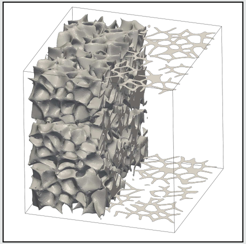
Spectral coarse spaces

The coarse space is **enhanced** by eigenfunctions of **local edge and face eigenvalue problems** with eigenvalues below tolerances $tol_{\mathcal{E}}$ and $tol_{\mathcal{F}}$:

$$\kappa\left(\boldsymbol{M}_{*}^{-1}\boldsymbol{K}\right) \leq C\left(1 + \frac{1}{tol_{\mathcal{E}}} + \frac{1}{tol_{\mathcal{F}}} + \frac{1}{tol_{\mathcal{E}} \cdot tol_{\mathcal{F}}}\right);$$

C does not depend on h, H, or the coefficients. **OS-ACMS** & **adaptive GDSW (AGDSW)** (**Heinlein, Klawonn, Knepper, Rheinbach (2018, 2018, 2019)**).

Foam coefficient function example



Solid phase: $\alpha = 10^6$; **transparent phase:** $\alpha = 1$; 100 subdomains

V_0	$tol_{\mathcal{E}}$	$tol_{\mathcal{F}}$	it.	κ	dim V_0	dim V_0/ dof
V_{GDSW}	—	—	**565**	$1.3 \cdot 10^6$	1 601	0.27 %
V_{AGDSW}	0.05	0.05	**60**	30.2	1 968	0.33 %
$V_{\text{OS-ACMS}}$	0.001	0.001	**57**	30.3	690	0.12 %

Cf. **Heinlein, Klawonn, Knepper, Rheinbach (2018, 2019)**.

Algebraic Spectral Extension-Based Coarse Spaces

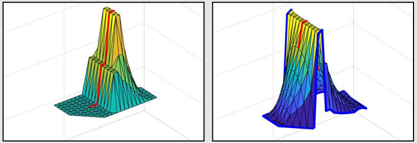
Two algebraic eigenvalue problems

Use the a-orthogonal decomposition

$$V_{\Omega_e} = V^0_{\Omega_e} \oplus \{E_{\partial\Omega_e \to \Omega_e}(v) : v \in V_{\partial\Omega_e}\}$$

to ***"split the AGDSW (Neumann) eigenvalue problem"*** into two:

- **Dirichlet eigenvalue problem** on $V^0_{\Omega_e}$
- **Transfer eigenvalue problem** on $V_{\Omega_e,\text{harm}}$; cf. **Smetana, Patera (2016)**



Condition number estimate

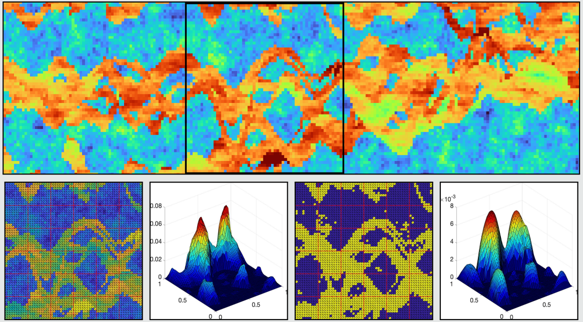
$$\kappa\left(M^{-1}_{\text{DIR\&TR}} K\right) \leq C \max\left\{{}^{1}/_{TOL_{\text{DIR}}}, {}^{TOL_{\text{TR}}}/_{\alpha_{\min}}\right\},$$

where C is independent of H, h, and the contrast of the coefficient function α.

Heinlein & Smetana (subm. 2023; preprint arXiv).

Numerical results – SPE10 benchmark

Layer 70 from model 2; cf. **Christie and Blunt (2001)**



V_0	TOL_{DIR}	TOL_{TR}	dim V_0	κ	its.
V_{GDSW}	-	-	85	$2.0\cdot10^5$	57
V_{AGDSW}	$1.0\cdot10^{-2}$		93	19.3	38
$V_{\text{DIR\&TR}-a}$	$1.0\cdot10^{-3}$	$1.0\cdot10^{5}$	90	19.4	39
$V_{\text{DIR\&TR}-l^2}$	$1.0\cdot10^{-3}$	$1.0\cdot10^{5}$	147	9.6	31
Original coefficient (without thresholding)					
V_{GDSW}	-	-	85	20.6	42

Linear & Nonlinear Preconditioning

Let us consider the nonlinear problem arising from the discretization of a partial differential equation

$$\boldsymbol{F}(\boldsymbol{u}) = 0.$$

We solve the problem using a **Newton-Krylov approach**, i.e., we solve a sequence of linearized problems using a Krylov subspace method:

$$D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right)\Delta\boldsymbol{u}^{(k+1)} = \boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right).$$

Linear preconditioning

In linear preconditioning, we **improve the convergence speed of the linear solver** by constructing a **linear operator** $\boldsymbol{M}^{-1}$ and solve linear systems

$$\boldsymbol{M}^{-1}D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right)\Delta\boldsymbol{u}^{(k+1)} = \boldsymbol{M}^{-1}\boldsymbol{F}(\boldsymbol{u}^{(k)}).$$

Goal:

- $\kappa\left(\boldsymbol{M}^{-1}D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right)\right) \approx 1.$

$\Rightarrow \boldsymbol{M}^{-1}D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right) \approx \boldsymbol{I}.$

Nonlinear preconditioning

In nonlinear preconditioning, we **improve the convergence speed of the nonlinear solver** by constructing a **nonlinear operator** G and solve the nonlinear system

$$(\boldsymbol{G}\circ\boldsymbol{F})(\boldsymbol{u}) = 0.$$

Goals:

- $\boldsymbol{G}\circ\boldsymbol{F}$ almost linear.
- Additionally: $\kappa\left(D(\boldsymbol{G}\circ\boldsymbol{F})(\boldsymbol{u})\right) \approx 1.$

Linear & Nonlinear Preconditioning

Let us consider the nonlinear problem arising from the discretization of a partial differential equation

$$\boldsymbol{F}(\boldsymbol{u}) = 0.$$

We solve the problem using a **Newton-Krylov approach**, i.e., we solve a sequence of linearized problems using a Krylov subspace method:

$$D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right)\Delta\boldsymbol{u}^{(k+1)} = \boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right).$$

Linear preconditioning

In linear preconditioning, we **improve the convergence speed of the linear solver** by constructing a **linear operator** $\boldsymbol{M}^{-1}$ and solve linear systems

$$\boldsymbol{M}^{-1}D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right)\Delta\boldsymbol{u}^{(k+1)} = \boldsymbol{M}^{-1}\boldsymbol{F}(\boldsymbol{u}^{(k)}).$$

Goal:

- $\kappa\left(\boldsymbol{M}^{-1}D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right)\right) \approx 1.$

$\Rightarrow \boldsymbol{M}^{-1}D\boldsymbol{F}\left(\boldsymbol{u}^{(k)}\right) \approx \boldsymbol{I}.$

Nonlinear preconditioning

In nonlinear preconditioning, we **improve the convergence speed of the nonlinear solver** by constructing a **nonlinear operator** G and solve the nonlinear system

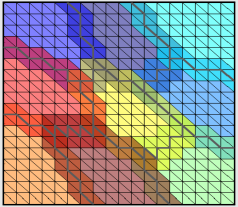
$$(\boldsymbol{G} \circ \boldsymbol{F})(\boldsymbol{u}) = 0.$$

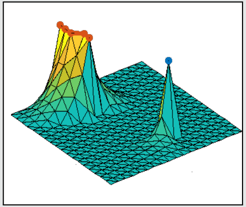
Goals:

- $\boldsymbol{G} \circ \boldsymbol{F}$ almost linear.
- Additionally: $\kappa\left(D(\boldsymbol{G} \circ \boldsymbol{F})(\boldsymbol{u})\right) \approx 1.$

Nonlinear Schwarz Methods

Two-level ASPEN & ASPIN methods





In **additive Schwarz preconditioned (in)exact Newton (ASPEN/ASPIN)** **(Cai and Keyes (2002))**, the nonlinear problem is modified

$$\boldsymbol{F}(\boldsymbol{u}) = 0 \quad \Leftrightarrow \quad \sum_{i=0}^{N} \boldsymbol{R}_i^T \boldsymbol{T}_i(\boldsymbol{u}) = 0$$

with corrections $\boldsymbol{T}_i(\boldsymbol{u})$ given by **nonlinear problems** on the **overlapping subdomains** / **coarse space**

$$\boldsymbol{R}_i \boldsymbol{F}(\boldsymbol{u} - \boldsymbol{R}_i^T \boldsymbol{T}_i(\boldsymbol{u})) = 0.$$

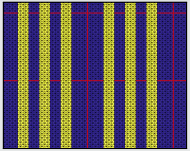
Coarse space via **Galerkin projection**: **Heinlein, Lanser (2020)**

Problem configuration

p-**Laplacian model problem** $(p = 4)$

$$\begin{aligned} -\alpha \Delta_p u &= 1 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

with $\alpha \Delta_p u := \operatorname{div}(\alpha |\nabla u|^{p-2} \nabla u)$ on a **domain decomposition into** 6×6 **subdomains** with $H/h = 32$ and overlap $1h$.



yellow: $\alpha = 10^3$
blue: $\alpha = 1$

size cp	method	coarse space	outer it.	local it. (avg.)	coarse it.	GMRES it. (sum)
no globalization						
145	ASPEN	AGDSW	5	27.0	35	77
25	ASPEN	MsFEM	>20	-	-	-
145	NK-AS	AGDSW	>20	-	-	-
inexact Newton backtracking (INB) **Eisenstat and Walker (1994)**						
145	ASPEN	AGDSW	5	24.8	21	77
25	ASPEN	MsFEM	18	83.9	75	852
145	NK-AS	AGDSW	13	-	-	207

Cf. **Heinlein, Klawonn, Lanser (2022)**

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

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

Monolithic GDSW preconditioner

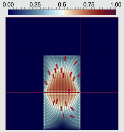
We construct a **monolithic GDSW preconditioner**

$$\mathcal{M}_{\text{GDSW}}^{-1} = \phi \mathcal{A}_0^{-1} \phi^\top + \sum\nolimits_{i=1}^{N} \mathcal{R}_i^\top \overline{\mathcal{P}}_i \mathcal{A}_i^{-1} \mathcal{R}_i,$$

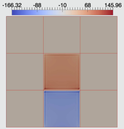
with **block matrices** $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$, **local pressure projections** $\overline{\mathcal{P}}_i$, and

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

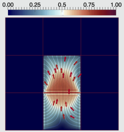
Using $\mathcal{A}$ to compute extensions: $\phi_I = -\mathcal{A}_{II}^{-1} \mathcal{A}_{I\Gamma} \phi_\Gamma$; cf. **Heinlein, Hochmuth, Klawonn (2019, 2020)**.



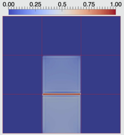
Φ_{u,u_0}



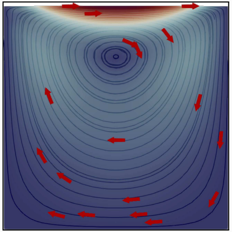
Φ_{p,u_0}



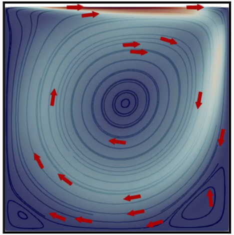
Φ_{u,p_0}



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

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

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

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

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

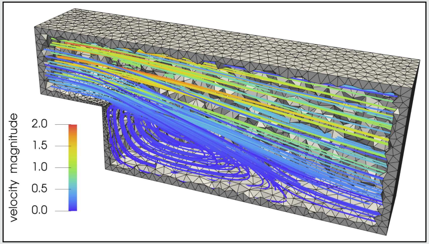
$$\mathcal{M}_{\text{SIMPLE}}^{-1} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}^{-1}\boldsymbol{B} \\ 0 & \alpha \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\hat{\boldsymbol{S}}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \hat{\boldsymbol{S}}^{-1} \end{bmatrix};$$

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

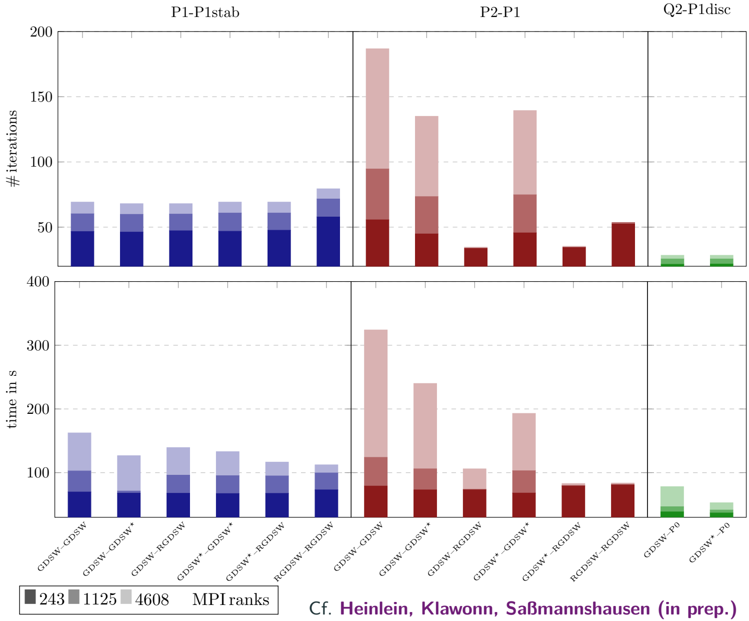


Steady-state Navier–Stokes equations

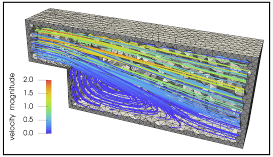
prec.	# MPI ranks	243	1 125	15 562
Monolithic RGDSW (FROSch)	setup	39.6 s	57.9 s	95.5 s
	solve	57.6 s	69.2 s	74.9 s
	total	**97.2 s**	**127.7 s**	**170.4 s**
SIMPLE RGDSW (Teko & FROSch)	setup	39.2 s	38.2 s	68.6 s
	solve	86.2 s	106.6 s	127.4 s
	total	125.4 s	144.8 s	196.0 s

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

Balancing the Velocity and Pressure Coarse Spaces

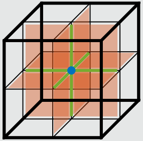


Cf. **Heinlein, Klawonn, Saßmannshausen (in prep.)**

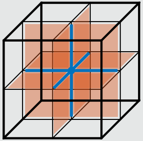


Varying the POU

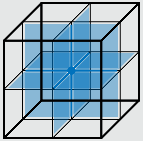
GDSW:



GDSW*:

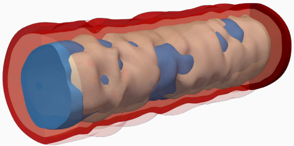


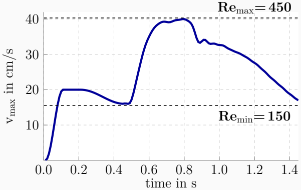
RGDSW:

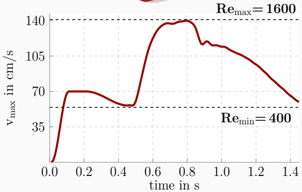


Results for Blood Flow Simulations

- **3D unsteady flow simulation** within the **geometry of a realistic artery** (from **Balzani et al. (2012)**) and kinematic viscosity $\nu = 0.03\,\mathrm{cm}^2/\mathrm{s}$
- **Parabolic inflow profile** is prescribed at inlet of geometry
- **Time discretization**: BDF-2; **space discretization**: P2-P1 elements







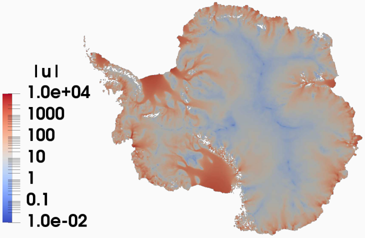
prec.	# MPI ranks	16	64	256
Monolithic RGDSW (FROSch)	avg. #its.	33	31	30
	setup	4 825 s	1 422 s	701 s
	solve	3 198 s	1 004 s	463 s
	total	8 023 s	2 426 s	**1 164 s**
SIMPLE RGDSW (Teko & FROSch)	avg. #its.	82	82	87
	setup	3 046 s	824 s	428 s
	solve	4 679 s	1 533 s	801 s
	total	**7 725 s**	**2 357 s**	1 229 s

prec.	# MPI ranks	16	64	256
Monolithic RGDSW (FROSch)	avg. #its.	36	36	36
	setup	4 808 s	1 448 s	688 s
	solve	3 490 s	1 186 s	538 s
	total	**8 298 s**	**2 634 s**	**1 226 s**
SIMPLE RGDSW (Teko & FROSch)	avg. #its.	157	164	169
	setup	3 071 s	842 s	432 s
	solve	9 541 s	3 210 s	1 585 s
	total	12 612 s	4 052 s	2 017 s

FROSch Preconditioners for Land Ice Simulations

Albany

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**)			Greenland (**multiphysics vel. & temperature**)		
	4 km resolution, 20 layers, 35 m dofs			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)

Accelerating Time-to-Solution

Inexact Subdomain Solvers in FROSch

$$M_{\text{OS-2}}^{-1} K = \Phi K_0^{-1} \Phi^T K + \sum_{i=1}^{N} R_i^T \mathrm{K}_i^{-1} R_i K$$

3D Laplacian; 512 MPI ranks = 512 (= 8 × 8 × 8) subdomains; $H/\delta = 10$; RGDSW coarse space.

		subdomain solver						
		direct	ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
		solver	k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, ≈ 14 k dofs per rank	iter	**26**	33	30	31	28	34	31
	setup time	1.89 s	0.97 s	1.01 s	0.89 s	0.91 s	**0.73 s**	**0.71 s**
	apply time	0.39 s	**0.27 s**	0.31 s	0.31 s	0.35 s	0.30 s	0.30 s
	prec. time	2.28 s	1.24 s	1.32 s	1.20 s	1.26 s	1.03 s	**1.01 s**
$H/h = 40$, ≈ 105 k dofs per rank	iter	**30**	55	46	52	41	59	51
	setup time	12.09 s	6.14 s	6.26 s	5.74 s	5.89 s	**5.55 s**	5.64 s
	apply time	4.21 s	**1.84 s**	1.96 s	2.66 s	3.28 s	2.52 s	2.47 s
	prec. time	16.30 s	**7.98 s**	8.22 s	8.40 s	9.18 s	8.16 s	8.11 s
$H/h = 60$, ≈ 350 k dofs per rank	iter	OOM	81	64	76	**56**	88	74
	setup time		47.29 s	47.87 s	45.14 s	**45.08 s**	45.44 s	45.49 s
	apply time		10.79 s	**9.98 s**	13.00 s	16.16 s	11.95 s	12.09 s
	prec. time		58.08 s	57.85 s	58.15 s	61.25 s	**57.39 s**	57.59 s

INTEL MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from IFPACK2.

Parallel computations on dual-socket Intel Xeon Platinum machine at Sandia National Laboratories (Blake).

Inexact Subdomain Solvers in FROSch

$$\boldsymbol{M}_{\text{OS-2}}^{-1}\boldsymbol{K} = \Phi \boldsymbol{K}_0^{-1}\Phi^T \boldsymbol{K} + \sum\nolimits_{i=1}^{N} \boldsymbol{R}_i^T \mathbf{K}_\mathbf{i}^{-1} \boldsymbol{R}_i \boldsymbol{K}$$

3D Laplacian; 512 MPI ranks = 512 ($= 8 \times 8 \times 8$) subdomains; $H/\delta = 10$; RGDSW coarse space.

		subdomain solver						
		direct	ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
		solver	k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**26**	33	30	31	28	34	31
	setup time	1.89 s	0.97 s	1.01 s	0.89 s	0.91 s	**0.73 s**	**0.71 s**
	apply time	0.39 s	**0.27 s**	0.31 s	0.31 s	0.35 s	0.30 s	0.30 s
	prec. time	2.28 s	1.24 s	1.32 s	1.20 s	1.26 s	1.03 s	**1.01 s**
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**30**	55	46	52	41	59	51
	setup time	12.09 s	6.14 s	6.26 s	5.74 s	5.89 s	**5.55 s**	5.64 s
	apply time	4.21 s	**1.84 s**	1.96 s	2.66 s	3.28 s	2.52 s	2.47 s
	prec. time	16.30 s	**7.98 s**	8.22 s	8.40 s	9.18 s	8.16 s	8.11 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	OOM	81	64	76	**56**	88	74
	setup time		47.29 s	47.87 s	45.14 s	**45.08 s**	45.44 s	45.49 s
	apply time		10.79 s	**9.98 s**	13.00 s	16.16 s	11.95 s	12.09 s
	prec. time		58.08 s	57.85 s	58.15 s	61.25 s	**57.39 s**	57.59 s

Intel MKL Pardiso; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from Ifpack2.

Parallel computations on dual-socket Intel Xeon Platinum machine at Sandia National Laboratories (Blake).

Inexact Extension Solvers in FROSch

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

3D Laplacian; 512 MPI ranks = 512 (= 8 × 8 × 8) subdomains; $H/\delta = 10$; RGDSW coarse space.

extension solver (10 Gauss–Seidel sweeps for the subdomain solver)		direct solver	**preconditioned GMRES** (rel. tol. $= 10^{-4}$)					
			ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**28**	**28**	**28**	**28**	**28**	**28**	**28**
	setup time	0.89 s	0.93 s	0.89 s	**0.78 s**	0.83 s	0.79 s	0.84 s
	apply time	0.35 s	0.35 s	**0.34 s**	0.36 s	**0.34 s**	0.35 s	**0.34 s**
	prec. time	1.23 s	1.28 s	1.23 s	**1.14 s**	1.17 s	**1.14 s**	1.18 s
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**41**	**41**	**41**	**41**	**41**	**41**	**41**
	setup time	5.72 s	**4.16 s**	4.61 s	4.26 s	4.64 s	4.27 s	4.33 s
	apply time	3.33 s	3.33 s	3.30 s	3.33 s	3.30 s	**3.28 s**	3.29 s
	prec. time	9.04 s	**7.49 s**	7.92 s	7.59 s	7.95 s	7.55 s	7.62 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	**56**	**56**	**56**	**56**	**56**	**56**	**56**
	setup time	45.16 s	**17.75 s**	18.16 s	17.98 s	19.34 s	17.93 s	18.04 s
	apply time	**15.83 s**	18.04 s	17.08 s	16.26 s	15.81 s	16.19 s	16.44 s
	prec. time	60.99 s	35.79 s	35.25 s	34.24 s	35.15 s	**34.12 s**	34.49 s

INTEL MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from IFPACK2.

Parallel computations on dual-socket Intel Xeon Platinum machine at Sandia National Laboratories (Blake).

Inexact Extension Solvers in FROSch

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

3D Laplacian; 512 MPI ranks = 512 (= 8 × 8 × 8) subdomains; $H/\delta = 10$; RGDSW coarse space.

extension solver (10 Gauss–Seidel sweeps for the subdomain solver)		direct solver	**preconditioned GMRES** (rel. tol. $= 10^{-4}$)					
			ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**28**	**28**	**28**	**28**	**28**	**28**	**28**
	setup time	0.89 s	0.93 s	0.89 s	**0.78 s**	0.83 s	0.79 s	0.84 s
	apply time	0.35 s	0.35 s	**0.34 s**	0.36 s	**0.34 s**	0.35 s	**0.34 s**
	prec. time	1.23 s	1.28 s	1.23 s	**1.14 s**	1.17 s	**1.14 s**	1.18 s
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**41**	**41**	**41**	**41**	**41**	**41**	**41**
	setup time	5.72 s	**4.16 s**	4.61 s	4.26 s	4.64 s	4.27 s	4.33 s
	apply time	3.33 s	3.33 s	3.30 s	3.33 s	3.30 s	**3.28 s**	3.29 s
	prec. time	9.04 s	**7.49 s**	7.92 s	7.59 s	7.95 s	7.55 s	7.62 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	**56**	**56**	**56**	**56**	**56**	**56**	**56**
	setup time	45.16 s	**17.75 s**	18.16 s	17.98 s	19.34 s	17.93 s	18.04 s
	apply time	**15.83 s**	18.04 s	17.08 s	16.26 s	15.81 s	16.19 s	16.44 s
	prec. time	60.99 s	35.79 s	35.25 s	34.24 s	35.15 s	**34.12 s**	34.49 s

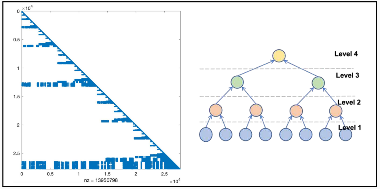
Intel MKL Pardiso; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from Ifpack2.

Parallel computations on dual-socket Intel Xeon Platinum machine at Sandia National Laboratories (Blake).

Sparse Triangular Solver in KokkosKernels (Amesos2 – SuperLU/Tacho)

SuperLU & SpTRSV

- **Supernodal LU factorization** with partial pivoting
- **Triangular solver** with **level-set scheduling** (KokkosKernels); cf. **Yamazaki, Rajamanickam, Ellingwood (2020)**.

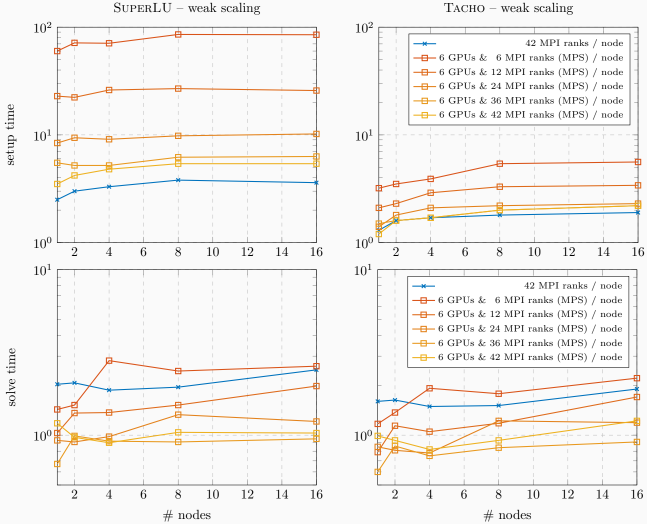


Tacho

- **Multifrontal factorization** with pivoting inside frontal matrices
- Implementation using Kokkos using **level-set scheduling**

Cf. **Kim, Edwards, Rajamanickam (2018)**.

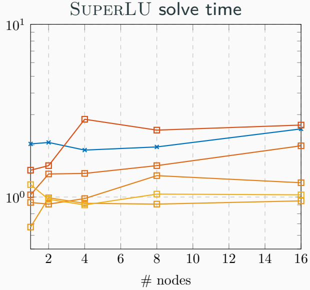
Three-Dimensional Linear Elasticity – Weak Scalability of FROSch

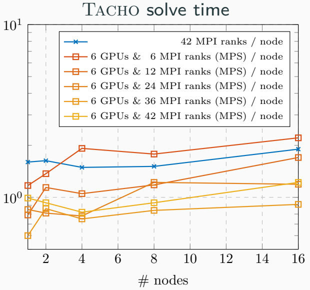


Computations on Summit (OLCF): 42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

Yamazaki, Heinlein, Rajamanickam (2023)

Three-Dimensional Linear Elasticity – Weak Scalability


SuperLU solve time


Tacho solve time

# nodes	1	2	4	8	16
# dofs	375 K	750 K	1.5 M	3 M	6 M

SuperLU solve					
CPUs	**2.03 (75)**	**2.07 (69)**	**1.87 (61)**	**1.95 (58)**	**2.48 (69)**
$n_p/\text{GPU} = 1$	1.43 (47)	1.52 (53)	2.82 (77)	2.44 (68)	2.61 (75)
2	1.03 (46)	1.36 (65)	1.37 (60)	1.52 (65)	1.98 (86)
4	0.93 (59)	0.91 (53)	0.98 (59)	1.33 (77)	1.21 (66)
6	0.67 (46)	0.99 (65)	0.92 (57)	0.91 (57)	0.95 (57)
7	**1.03 (75)**	**1.04 (69)**	**0.90 (61)**	**0.97 (58)**	**1.18 (69)**
speedup	**2.0**×	**2.0**×	**2.1**×	**2.0**×	**2.1**×

Tacho solve					
CPUs	**1.60 (75)**	**1.63 (69)**	**1.49 (61)**	**1.51 (58)**	**1.90 (69)**
$n_p/\text{GPU} = 1$	1.17 (47)	1.37 (53)	1.92 (77)	1.78 (68)	2.21 (75)
2	0.79 (46)	1.14 (65)	1.05 (60)	1.18 (65)	1.70 (86)
4	0.85 (59)	0.81 (53)	0.78 (59)	1.22 (77)	1.19 (66)
6	0.60 (46)	0.86 (65)	0.75 (57)	0.84 (57)	0.91 (57)
7	**0.99 (75)**	**0.93 (69)**	**0.82 (61)**	**0.93 (58)**	**1.22 (69)**
speedup	**1.6**×	**1.8**×	**1.8**×	**1.6**×	**1.6**×

Computations on Summit (OLCF): 42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

Yamazaki, Heinlein, Rajamanickam (2023)

Three-Dimensional Linear Elasticity – ILU Subdomain Solver

ILU level		0	1	2	3
setup					
CPU	No	**1.5**	**1.9**	**3.0**	**4.8**
	ND	1.6	2.6	4.4	7.4
GPU	KK(No)	1.4	1.5	1.8	2.4
	KK(ND)	1.7	2.0	2.9	5.2
	Fast(No)	**1.5**	**1.6**	**2.1**	**3.2**
	Fast(ND)	1.5	1.7	2.5	4.5
speedup		**1.0**×	**1.2**×	**1.4**×	**1.5**×
solve					
CPU	No	**2.55 (158)**	**3.60 (112)**	**5.28 (99)**	**6.85 (88)**
	ND	4.17 (227)	5.36 (134)	6.61 (105)	7.68 (88)
GPU	KK(No)	3.81 (158)	4.12 (112)	4.77 (99)	5.65 (88)
	KK(ND)	2.89 (227)	4.27 (134)	5.57 (105)	6.36 (88)
	Fast(No)	**1.14 (173)**	**1.11 (141)**	**1.26 (134)**	**1.43 (126)**
	Fast(ND)	1.49 (227)	1.15 (137)	1.10 (109)	1.22 (100)
speedup		**2.2**×	**3.2**×	**4.3**×	**4.8**×

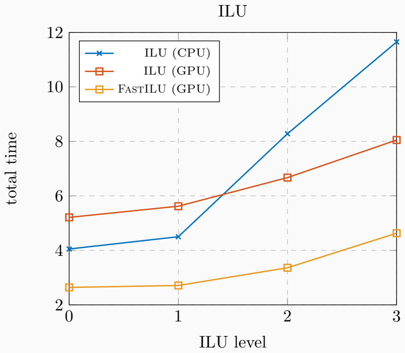
Computations on Summit (OLCF):
42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

Yamazaki, Heinlein, Rajamanickam (2023)

ILU variants

- KokkosKernels ILU (KK)
- Iterative FastILU (Fast); cf. **Chow, Patel (2015)** and **Boman, Patel, Chow, Rajamanickam (2016)**

No reordering (**No**) and nested dissection (**ND**)



Three-Dimensional Linear Elasticity – Weak Scalability Using ILU(1)

# nodes		1	2	4	8	16
# dofs		648 K	1.2 M	2.6 M	5.2 M	10.3 M
setup						
CPU		**1.9**	**2.2**	**2.4**	**2.4**	**2.6**
GPU	KK	1.4	2.0	2.2	2.4	2.8
	Fast	**1.5**	**2.2**	**2.3**	**2.5**	**2.8**
speedup		**1.3**×	**1.0**×	**1.0**×	**1.0**×	**0.9**×
solve						
CPU		**3.60 (112)**	**7.26 (84)**	**6.93 (78)**	**6.41 (75)**	**4.1 (109)**
GPU	KK	4.3 (119)	3.9 (110)	4.8 (105)	4.3 (97)	4.9 (109)
	Fast	**1.2 (154)**	**1.0 (133)**	**1.1 (130)**	**1.3 (117)**	**1.6 (131)**
speedup		**3.3**×	**3.8**×	**3.4**×	**2.5**×	**2.6**×

Computations on Summit (OLCF): 42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

Yamazaki, Heinlein, Rajamanickam (2023)

Related works

- **One-level Schwarz with local solves on GPUs: Luo, Yang, Zhao, Cai (2011)**
- **Solves of dense local Schur complement matrices in the balancing domain decomposition by constraints (BDDC) method on GPUs: Šístek & Oberhuber (2022)**

Learning Extension Operators Using Graph Neural Networks

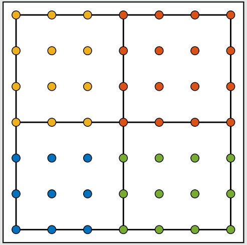
Why Learning Extension Operators

Most coarse spaces for Schwarz preconditioners are constructed based on a **characteristic functions**

$$\varphi_i(\omega_j) = \delta_{ij},$$

on specifically chosen sets of nodes $\{\omega_j\}_j$. The **values in the remaining nodes** are then obtained by **extending the values into the adjacent subdomains**. Examples:

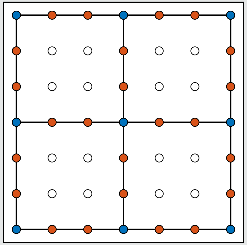
Subdomain-based



- The ω_j are based on nonoverl. subdomains Ω_j
- **No extensions** needed

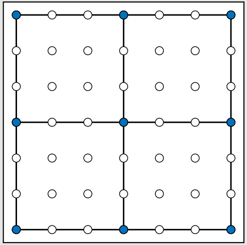
Cf. **Nicolaides (1987)**.

GDSW



- The ω_j are based on **partition of the interface**
- **Energy-minimizing** exts.

Vertex-based



- **Lagrangian**: **geometric** ext.
- **MsFEM**: **geometric** and **energy-minimizing** exts.
- **RGDSW**: **algebraic** and **energy-minimizing** exts.

Why Learning Extension Operators

Most coarse spaces for Schwarz preconditioners are constructed based on a **characteristic functions**

$$\varphi_i(\omega_j) = \delta_{ij},$$

on specifically chosen sets of nodes $\{\omega_j\}_j$. The **values in the remaining nodes** are then obtained by **extending the values into the adjacent subdomains**. Examples:

Observation 1

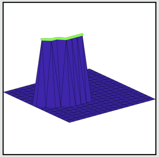
Energy-minimizing extensions

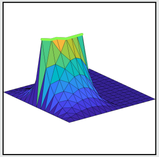
- are algebraic:

$$\boldsymbol{v}_I = -\boldsymbol{K}_{II}^{-1}\boldsymbol{K}_{I\Gamma}\boldsymbol{v}_\Gamma$$

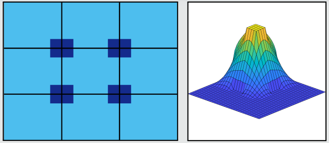
(with Dirichlet b. c.)

- can be costly: solving a **problem in the interior**





Observation 2

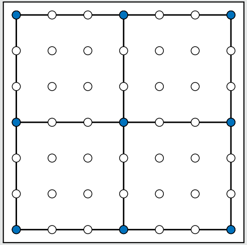


Heterogeneous: $\alpha_{\text{light}} = 1$; $\alpha_{\text{dark}} = 10^8$

The performance may **strongly depend on extension operator**:

coarse space	its.	κ
—	**163**	$4.06 \cdot 10^7$
Q1	**138**	$1.07 \cdot 10^6$
MsFEM	**24**	8.05

Vertex-based



- Lagrangian: **geometric** ext.
- MsFEM: **geometric** and **energy-minimizing** exts.
- RGDSW: **algebraic** and **energy-minimizing** exts.

→ **Improving efficiency & robustness** via machine learning.

Related Works

This overview is **not exhaustive**:

Coarse spaces for domain decomposition methods

- **Prediction of the geometric location of adaptive constraints (adaptive BDDC & FETI–DP as well as AGDSW): Heinlein, Klawonn, Lanser, Weber (2019, 2020, 2021, 2021, 2021, 2022)**
- **Prediction of the adaptive constraints: Klawonn, Lanser, Weber (preprint 2023, 2024)**
- **Prediction of spectral coarse spaces for BDDC for stochastic heterogeneities: Chung, Kim, Lam, Zhao (2021)**
- **Learning interface conditions and coarse interpolation operators: Taghibakhshi et al. (2022, 2023)**

Algebraic multigrid (AMG)

- **Prediction of coarse grid operators: Luz et al. (2020), Tomasi, Krause (2023)**
- **Coarsening: Taghibakhshi, MacLachlan, Olson, West (2021); Antonietti, Caldana, Dede (2023)**

An overviews of the **state-of-the-art on domain decomposition and machine learning** in early 2021 and 2023:

📕 A. Heinlein, A. Klawonn, M. Lanser, J. Weber

Combining machine learning and domain decomposition methods for the solution of partial differential equations — A review

GAMM-Mitteilungen. 2021.

📕 A. Klawonn, M. Lanser, J. Weber

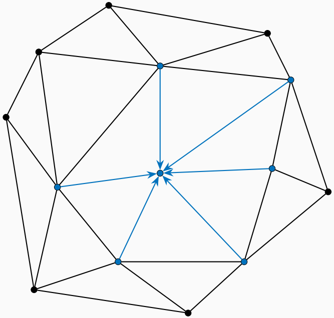
Machine learning and domain decomposition methods – a survey

arXiv:2312.14050. 2023

Prediction via Graph Convolutional Networks

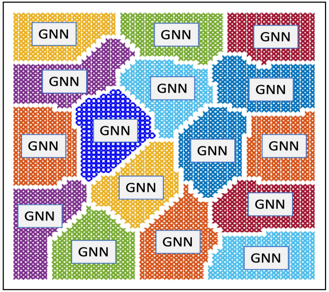
Graph neural networks (GNNs) introduced in **Gori, Monfardini, and Scarselli (2005)** are well-suited for learning on data based on simulation meshes:

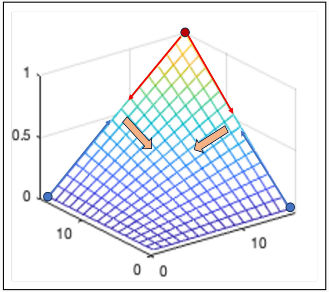
- **Generalization** of classical **convolutional neural networks (CNNs) LeCun (1998)** to **graph-based data sets**.
- **Aggregation and transmission of features** of **neighboring nodes in the graph** via **message passing layers**.
- **Invariance and equivariance** with respect to **position and permutation** of the nodes of the graph.



Local approach

- **Input:** subdomain matrix $\boldsymbol{K}_i$
- **Output:** basis functions $\{\varphi_j^{\Omega_i}\}_j$ on the same subdomain
- Training on **subdomains with varying geometry**
- Inference on **unseen subdomains**

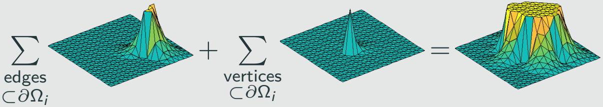




Theory-Inspired Design of the GNN-Based Coarse Space

Null space property

Any extension-based coarse space built from a partition of unity on the domain decomposition interface satisfies the **null space property necessary for numerical scalability**:



Explicit partition of unity

To **explicitly enforce** that the basis functions $(\varphi_j)_j$ form **a partition of unity**

$$\varphi_j = \frac{\hat{\varphi}_j}{\sum_k \hat{\varphi}_k},$$

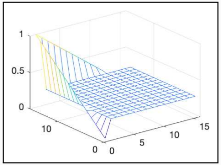
where the $\hat{\varphi}_k$ are the outputs of the GNN.

Initial and target

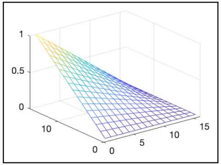
- **Initial function:** partition of unity that is constant in the interior
- **Target function:**
  - linear on the edges
  - energy-minimizing in the interior

→ **Information transport** via **message passing**

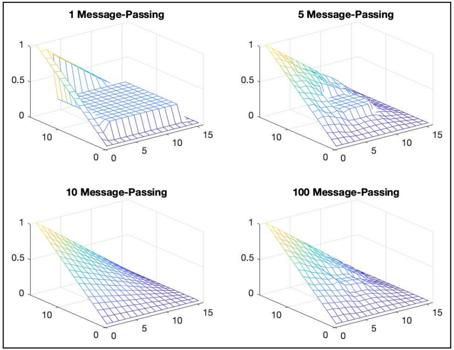
Initial



Target



Learned

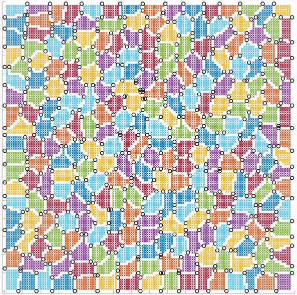


Numerical Results for Homogeneous Laplacian – Weak Scaling Study

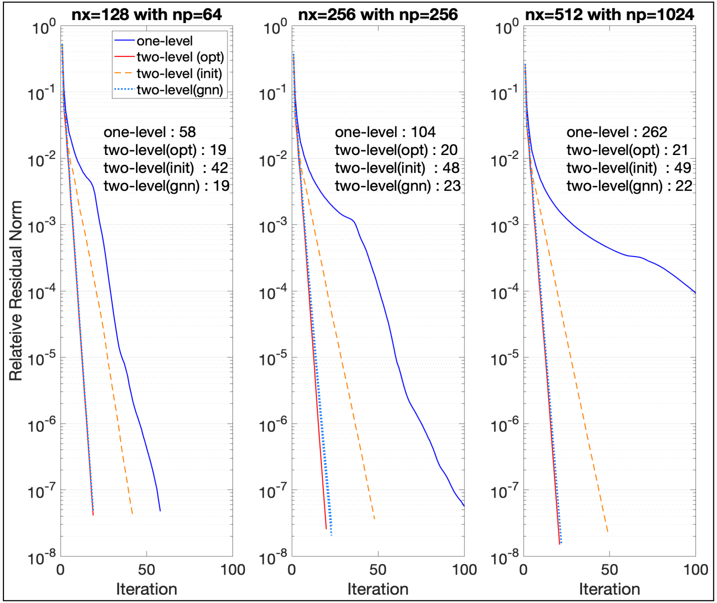
Model problem: 2D Laplacian model problem discretized using finite differences on a structured grid

$$
\begin{aligned}
-\Delta u &= 1 \quad \text{in } \Omega, \\
u &= 0 \quad \text{on } \partial\Omega,
\end{aligned}
$$

decomposed using METIS:



- The GNN has been **trained on 64 subdomains**.

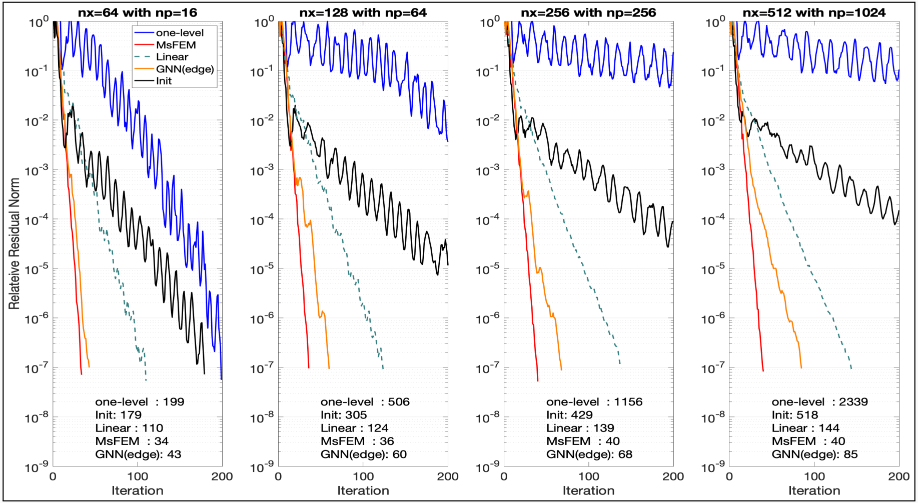


Yamazaki, Heinlein, Rajamanickam (subm. 2024)

Numerical Results for Heterogeneous Laplacian – Weak Scaling Study

Heterogeneous Laplacian with $\alpha_{\max}/\alpha_{\min} = 10^3$:

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



Yamazaki, Heinlein, Rajamanickam (subm. 2024)

FROSch

- FROSch is based on the **Schwarz framework** and **energy-minimizing coarse spaces**, which provide **numerical scalability** using **only algebraic information** for a **variety of applications**

Subdomain solves on GPUs

- Subdomain solves make up a **major part of the total solver time**.
- Using the **GPU triangular solve** from KokkosKernels, we can **speed up** the **solve phase** of FROSch. It can be **further improved** using **ILU**.

Learning extension operators

- **Extensions** are a major component in the **construction of coarse spaces** for domain decomposition methods.
- Using **GNNs** and **known properties from the theory**, we can **learn extension operators** that lead to a **scalable coarse spaces**.

Thank you for your attention!