TU Delft

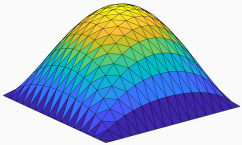
Fast and Robust Overlapping Schwarz (FROSch) Preconditioners in Trilinos

Alexander Heinlein

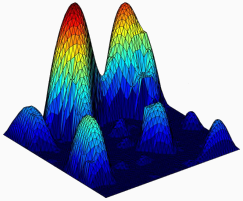
Seminar talk, Sandia National Laboratories, Albuquerque, U.S., March 19, 2024

Delft University of Technology

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

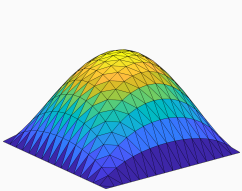
For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

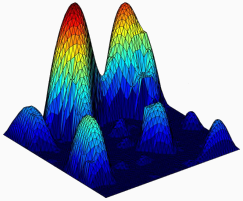
Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number** $\kappa(\boldsymbol{A})$. It deteriorates, e.g., for

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

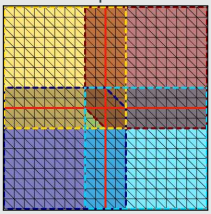
$\Rightarrow$ We introduce a preconditioner $\boldsymbol{M}^{-1} \approx \boldsymbol{A}^{-1}$ to improve the condition number:

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

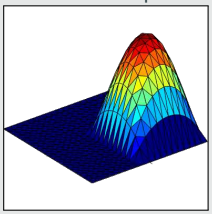
Two-Level Schwarz Preconditioners

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 $K_i := R_i K R_i^T$.

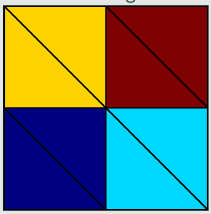
Condition number estimate:

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

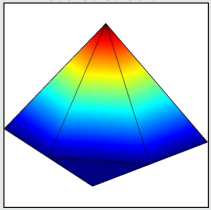
with subdomain size H and overlap width δ.

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.

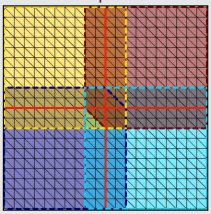
Condition number estimate:

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

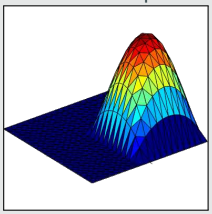
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$

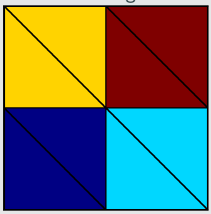


Solution of local problem

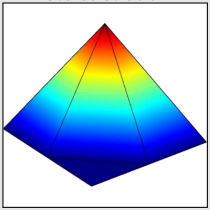


Lagrangian coarse space

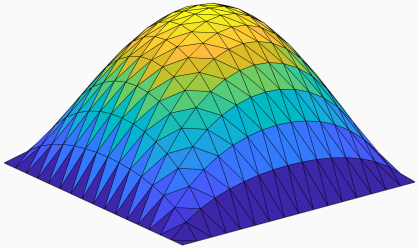
Coarse triangulation

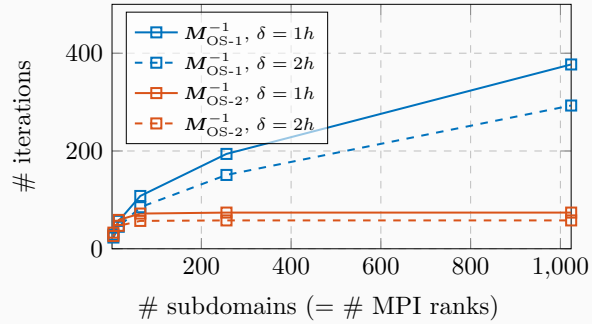


Coarse solution



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

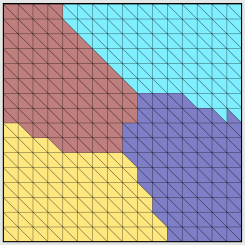
Team (active)

- Filipe Cumaru (TU Delft)
- Kyrill Ho (UCologne)
- 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)

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.

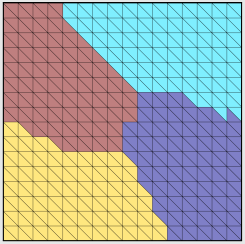


Nonoverlapping DD

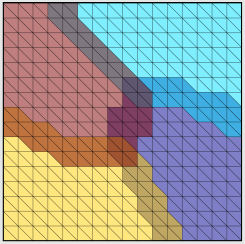
Algorithmic Framework for FROSch Overlapping Domain Decompositions

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

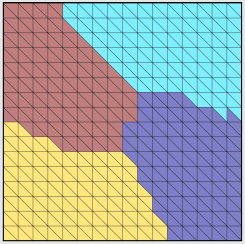


Overlap $\delta = 1h$

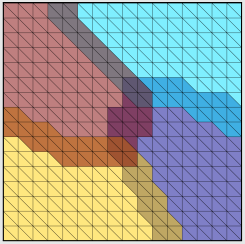
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

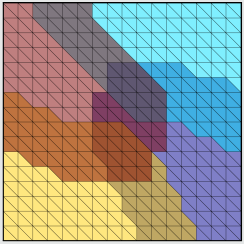
In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$

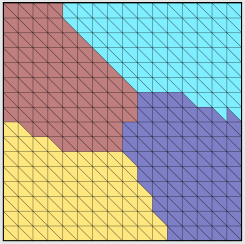


Overlap $\delta = 2h$

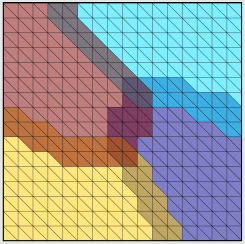
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

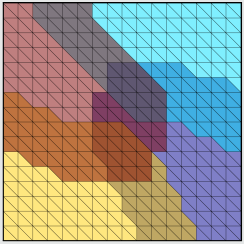
In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$

Computation of the overlapping matrices

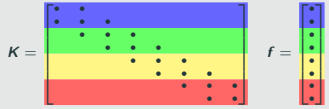
The overlapping matrices

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

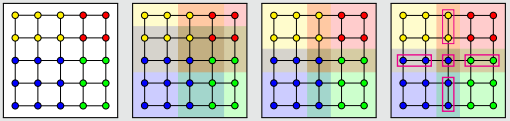
can easily be extracted from K since R_i is just a **global-to-local index mapping**.

Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components



Identification from **parallel distribution** of matrix:



2. Interface partition of unity (IPOU)

vertex & edge functions

vertex functions

Based on the interface components, construct an **interface partition of unity**:

$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

3. Interface basis

null space basis
(e.g., linear elasticity: **translations**, **linearized rotation(s)**)

×

The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

4. Extension into the interior

edge basis function

vertex basis function

The values in the interior of the subdomains are computed via the **extension operator**:

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

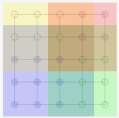
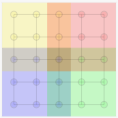
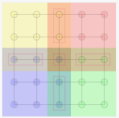
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

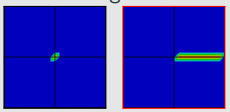
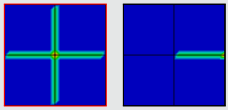
1. Identification interface components

$K =$ $f =$

Identification from **parallel distribution** of matrix:

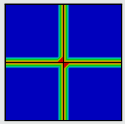
distributed map 
overlapping map 
repeated map 
interface comp. 

2. Interface partition of unity (IPOU)

vertex & edge functions 
vertex functions 

Based on the interface components, construct an **interface partition of unity**:

$$\sum_i \pi_i = 1 \text{ on } \Gamma$$



3. Interface basis

null space basis
(e.g., linear elasticity: **translations**, **linearized rotation(s)**)

$\times$

The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

4. Extension into the interior

edge basis function
vertex basis function

The values in the interior of the subdomains are computed via the **extension operator**:

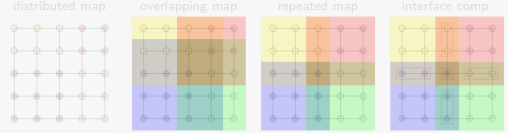
$$\Phi = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} -K_{II}^{-1} K_{\Gamma I}^T \Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix}.$$

(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

Identification from **parallel distribution** of matrix:



2. Interface partition of unity (IPOU)

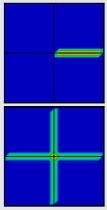
vertex & edge functions

vertex functions

Based on the interface components, construct an **interface partition of unity**:

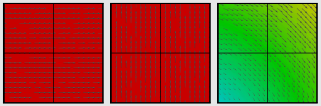
$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

3. Interface basis



null space basis
(e.g., linear elasticity: **translations**, **linearized rotation(s)**)

×



The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

4. Extension into the interior

edge basis function

vertex basis function

The values in the interior of the subdomains are computed via the **extension operator**:

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

(For elliptic problems: **energy-minimizing extension**)

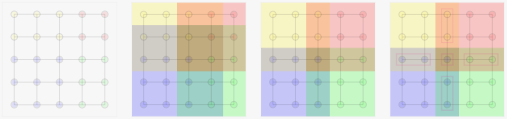
Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$K =$ $f =$

Identification from **parallel distribution** of matrix:

distributed map | overlapping map | repeated map | interface comp.



2. Interface partition of unity (IPOU)

vertex & edge functions | vertex functions

Based on the interface components, construct an **interface partition of unity**:

$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

3. Interface basis

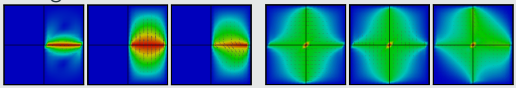
null space basis
(e.g., linear elasticity: **translations**, **linearized rotation(s)**)

$\times$

The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

4. Extension into the interior

edge basis function | vertex basis function



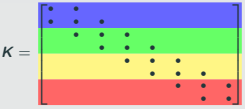
The values in the interior of the subdomains are computed via the **extension operator**:

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

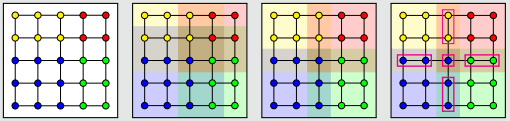
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

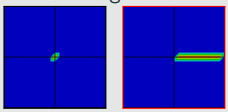


Identification from **parallel distribution** of matrix:

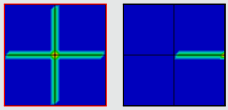


2. Interface partition of unity (IPOU)

vertex & edge functions

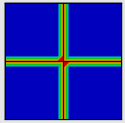


vertex functions

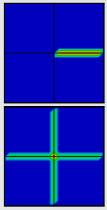


Based on the interface components, construct an **interface partition of unity**:

$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

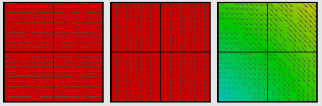


3. Interface basis



null space basis
(e.g., linear elasticity: **translations**, **linearized rotation(s)**)

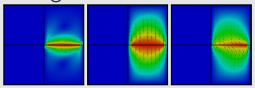
×



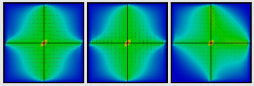
The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

4. Extension into the interior

edge basis function



vertex basis function



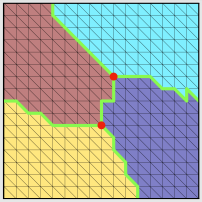
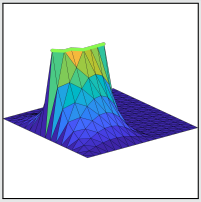
The values in the interior of the subdomains are computed via the **extension operator**:

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

(For elliptic problems: **energy-minimizing extension**)

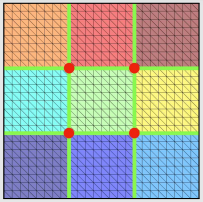
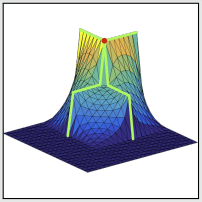
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

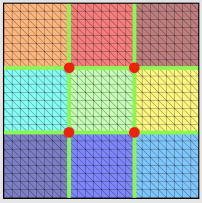
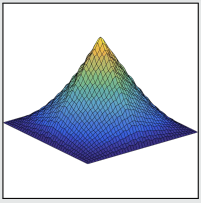
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

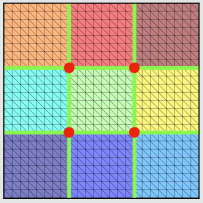
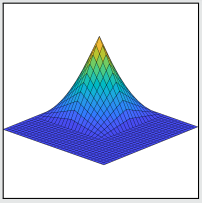
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)

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

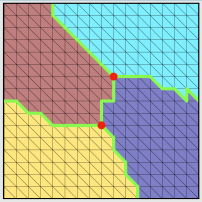
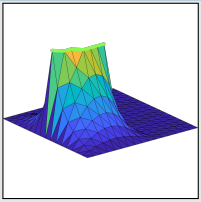
Q1 Lagrangian / piecewise bilinear

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

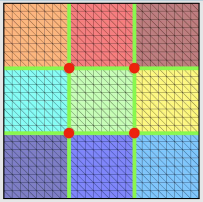
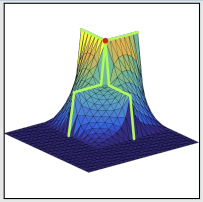
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

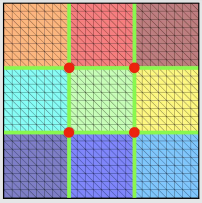
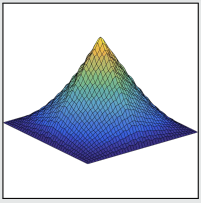
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

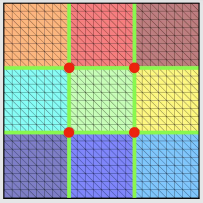
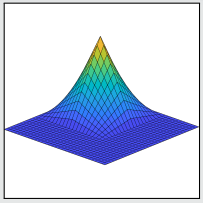
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

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

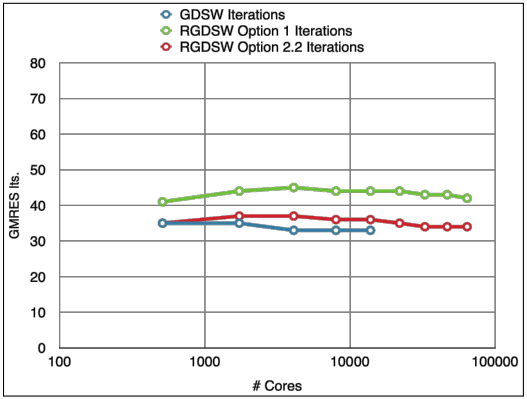



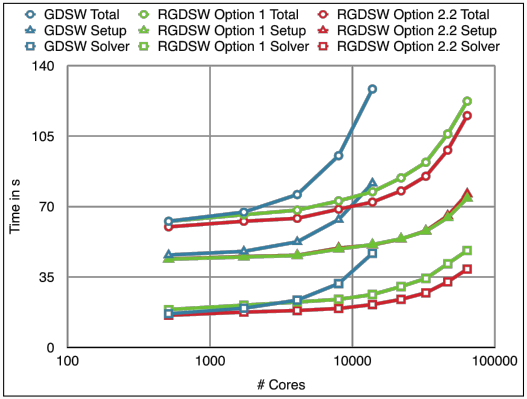
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

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

Model problem: **Poisson equation in 3D** **Coarse solver: MUMPS (direct)**

Largest problem: **374 805 361 / 1 732 323 601 unknowns**





Cf. Heinlein, Klawonn, Rheinbach, Widlund (2017); computations performed on Juqueen, JSC, Germany.

1 Multilevel Schwarz Preconditioners in FROSch

Based on joint work with **Oliver Rheinbach** and **Friederike Röver** (Technische Universität Bergakademie Freiberg)

2 Monolithic Schwarz Preconditioners in FROSch

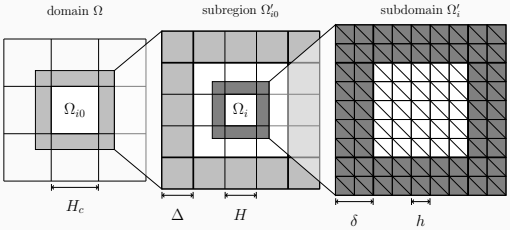
Based on joint work with **Christian Hochmuth**, **Axel Klawonn**, and **Lea Saßmannshausen** (Universität zu Köln) and **Mauro Perego** and **Sivasankaran Rajamanickam** (Sandia National Laboratories)

3 FROSch Preconditioners With Inexact Solvers

Based on joint work with **Sivasankaran Rajamanickam** and **Ichitaro Yamazaki** (Sandia National Laboratories)

Multilevel Schwarz Preconditioners in FROSch

Multi-Level GDSW Preconditioner



Heinlein, Klawonn, Rheinbach, Röver (2019, 2020),
Heinlein, Rheinbach, Röver (2022, 2023)

Recursive implementation

- Instead of solving the coarse problem exactly, we construct and apply a **FROSch preconditioner** as an **inexact coarse solver**

→ **Hierarchy of domain decompositions**

- **Interpolation of the null space** to coarse spaces

Algorithm 1: Application of the lth level of an L level FROSCH preconditioner

Function FROSch($\boldsymbol{K}$,$\boldsymbol{x}$,l):

$\boldsymbol{x} = \Phi^\top \boldsymbol{x}$; /* coarse interpolation */

if $l < L$ **then** $\boldsymbol{x}$ = FROSch($\boldsymbol{K}_0$,$\boldsymbol{x}$,$l+1$); /* exact coarse solver */

else $\boldsymbol{x} = \boldsymbol{K}_0^{-1}\boldsymbol{x}$; /* inexact coarse solver */

$\boldsymbol{x} = \Phi \boldsymbol{x}$; /* fine interpolation */

for $i := 1$ **to** $N^{(l)}$ **do** $\boldsymbol{x} = \boldsymbol{x} + \boldsymbol{R}_i^\top \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{x}$; /* fine level updates */

return $\boldsymbol{x}$;

end

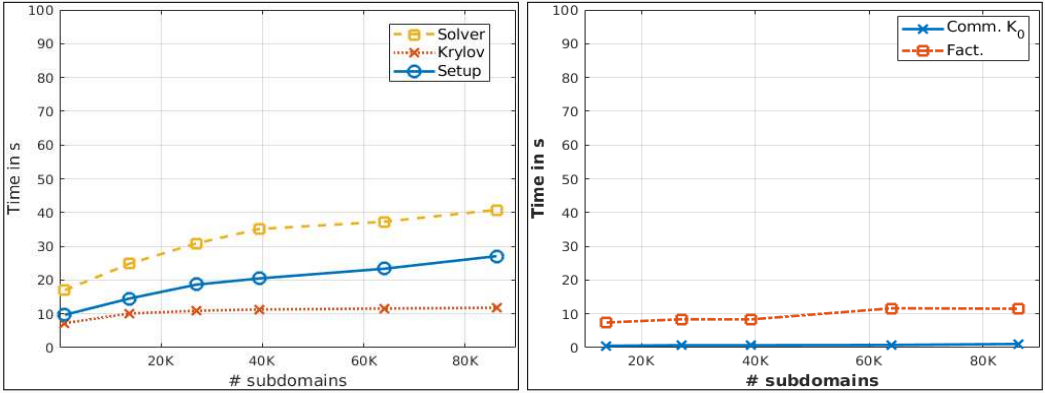
Compare a two-level FROSCH preconditioner: $\boldsymbol{M}^{-1}_{\text{FROSCH}} = \Phi \boldsymbol{K}_0^{-1} \Phi^T \boldsymbol{K} + \sum_{i=1}^{N} \boldsymbol{R}_i^T \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{K}$

Weak Scalability of the Three-Level RGDSW Preconditioner – SuperMUC-NG

In Heinlein, Rheinbach, Röver (2022), it has been shown that the **null space can be transferred algebraically to higher levels**.

Model problem: **Linear elasticity in 3D** **Coarse solver level 3: Intel MKL Pardiso (direct)**

Largest problem: **2 044 416 000 unknowns**



Cf. Heinlein, Rheinbach, Röver (2022); computations performed on SuperMUC-NG, LRZ, Germany.

Monolithic Schwarz Preconditioners in FROSch

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

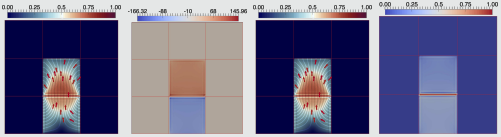
We construct a **monolithic GDSW preconditioner**

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

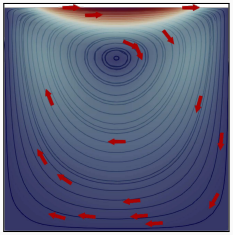
with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$, and

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

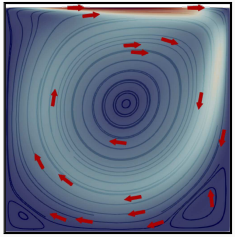
Using $\mathcal{A}$ to compute extensions: $\phi_I = -\mathcal{A}_{II}^{-1} \mathcal{A}_{I\Gamma} \phi_\Gamma$; cf. Heinlein, Hochmuth, Klawonn (2019, 2020).



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

SIMPLE block preconditioner

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

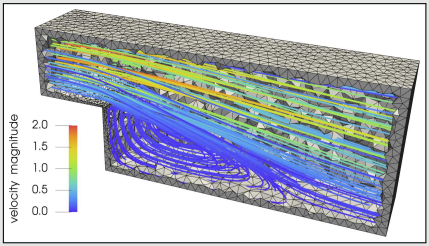
$$M_{\text{SIMPLE}}^{-1} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}^{-1}\boldsymbol{B} \\ 0 & \alpha \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\hat{\boldsymbol{S}}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \hat{\boldsymbol{S}}^{-1} \end{bmatrix};$$

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

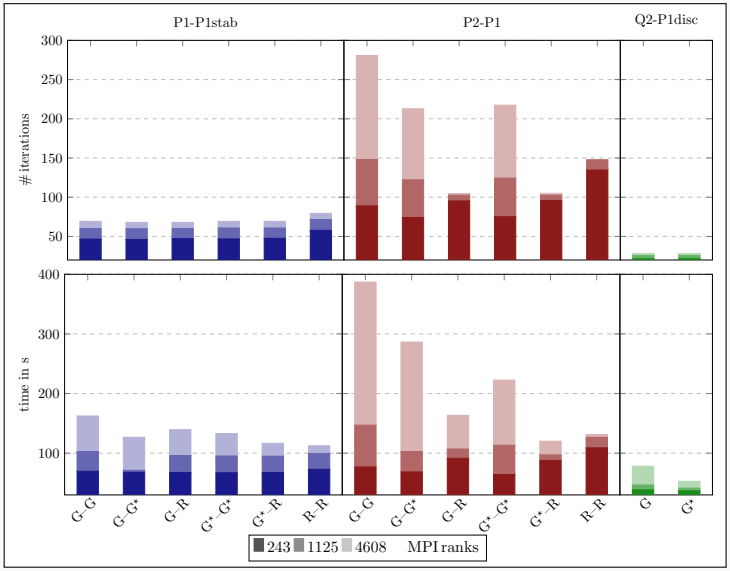


Steady-state Navier–Stokes equations

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.

Coarse Spaces for Monolithic FROSch Preconditioners for CFD Simulations



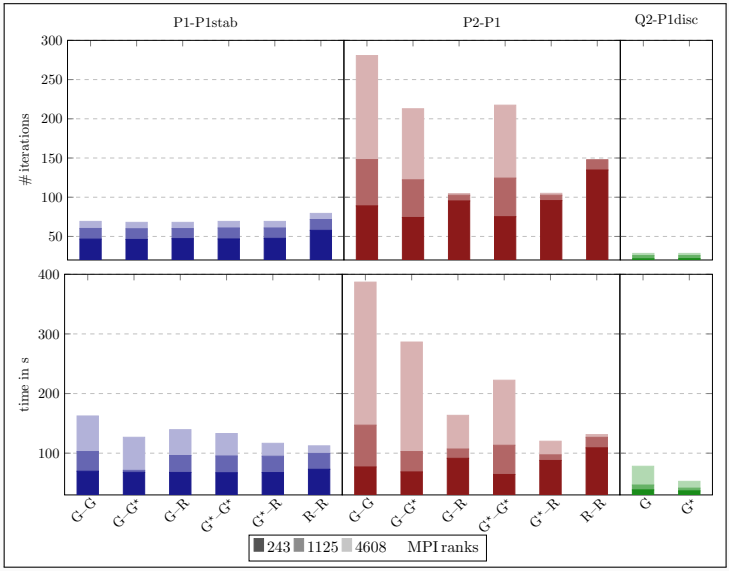
FROSch allows for the **flexible construction of extension-based coarse spaces** based on **various choices for the interface partition of unity (IPOU)**: IPOUHarmonicCoarseOperator

Comparison of coarse spaces

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

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

Coarse Spaces for Monolithic FROSch Preconditioners for CFD Simulations



FROSch allows for the **flexible construction of extension-based coarse spaces** based on **various choices for the interface partition of unity (IPOU)**: IPOUHarmonicCoarseOperator

Comparison of coarse spaces

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

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

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

FROSch Preconditioners With Inexact Solvers

Inexact Subdomain Solvers in FROSch

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

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 Subdomain Solvers in FROSch

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

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

		subdomain solver						
		direct solver	ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			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 Extension Solvers in FROSch

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

3D Laplacian; 512 MPI ranks $= 512\ (= 8 \times 8 \times 8)$ 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

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

3D Laplacian; 512 MPI ranks $= 512\ (= 8 \times 8 \times 8)$ 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

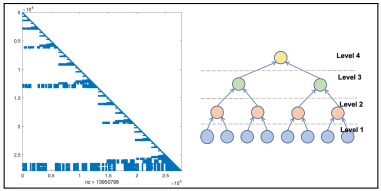
Intel MKL Pardiso; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from Ifpack2.

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)

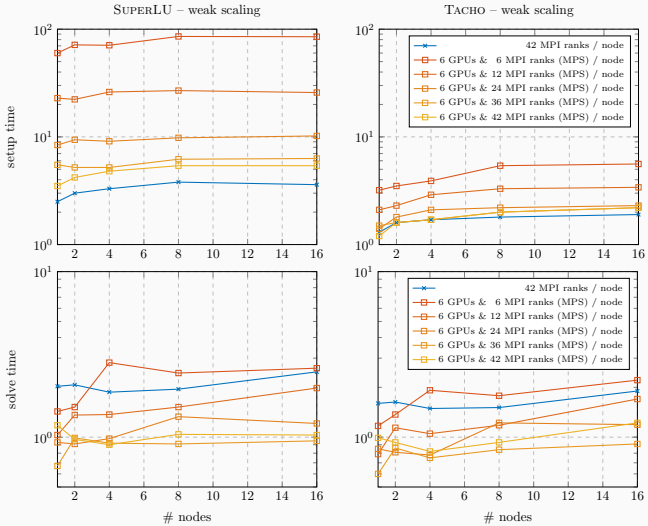


Tacho

- **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 – 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**×

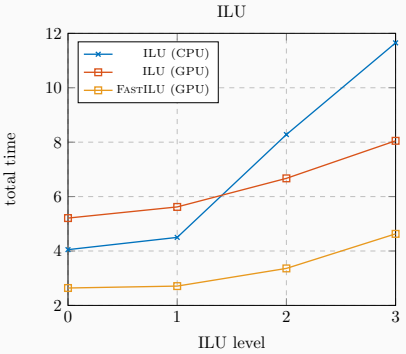
Computations on Summit (OLCF):
42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

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
GPU	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)
GPU	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 GPU:** Luo, Yang, Zhao, Cai (2011)
- **Solves of dense local Schur complement matrices in BDDC on GPUs:** Šístek & Oberhuber (2022)

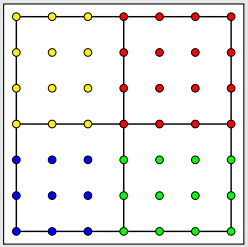
Why Learning Extension Operators

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:

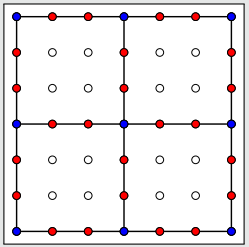
Subdomain-based



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

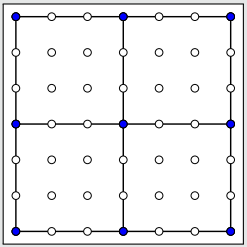
Cf. Nicolaides (1987)

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

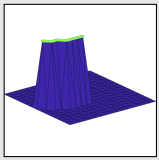
Energy-minimizing extensions

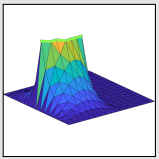
- are **algebraic**:

$$v_I = -K_{II}^{-1} K_{I\Gamma} v_\Gamma$$

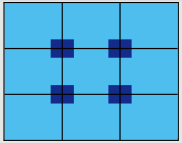
(with Dirichlet b. c.)

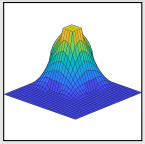
- can be **costly**: solving a **problem in the interior**





Observation 2



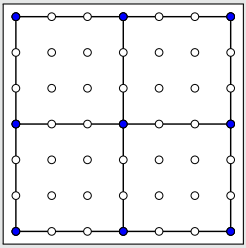


Heterogeneous: $\alpha_{\text{light}} = 1$; $\alpha_{\text{dark}} = 10^8$

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.

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

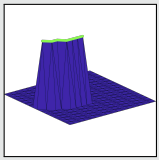
Energy-minimizing extensions

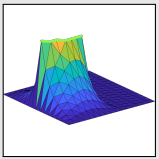
- are **algebraic**:

$$v_I = -K_{II}^{-1} K_{I\Gamma} v_\Gamma$$

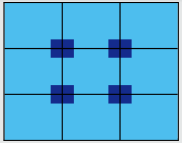
(with Dirichlet b. c.)

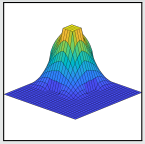
- can be **costly**: solving a **problem in the interior**





Observation 2



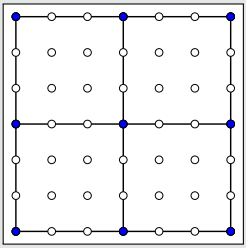


Heterogeneous: $\alpha_{\text{light}} = 1$; $\alpha_{\text{dark}} = 10^8$

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:** 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

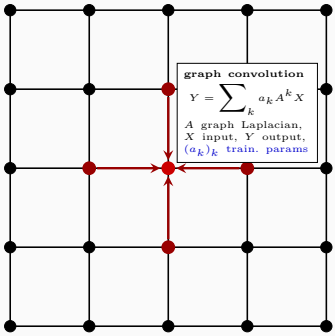
Machine learning and domain decomposition methods – a survey

arXiv:2312.14050. 2023

Prediction via Graph Convolutional Networks

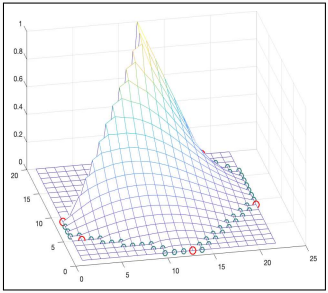
Graph convolutional networks (GCNs) introduced in **Kipf and Welling (2017)** are an example of **graph neural networks (GNNs)** and are well-suited for learning operations on simulation meshes:

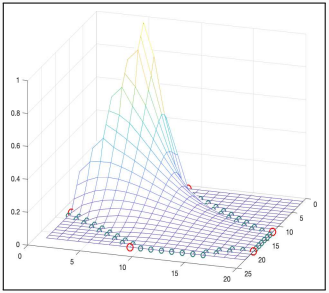
- **Generalization** of classical convolutional neural networks (CNNs) **LeCun (1998)** to **graph-based data sets**.
- Consist of **message passing layers**, which perform a **graph convolution operation on each node of the graph**.
- Graph convolutions are **invariant to position** and **permutation of the input vector**.



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**

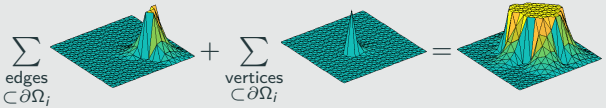




Theory-Inspired Design of the GNN-Based Coarse Space

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

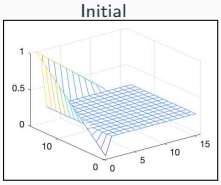
To **explicitly enforce** that the basis functions $\left(\varphi_j\right)_j$ form **a partition of unity**

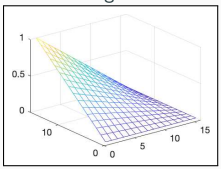
$$\varphi_j = \frac{\hat{\varphi}_j}{\sum_k \hat{\varphi}_k},$$

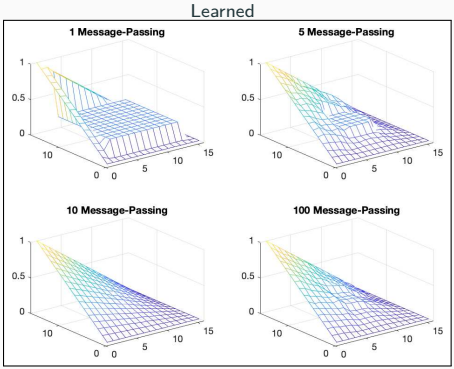
where the $\hat{\varphi}_k$ are the outputs of the GNN.

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



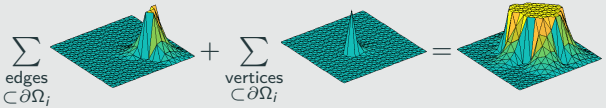




Theory-Inspired Design of the GNN-Based Coarse Space

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 $\left(\varphi_j\right)_j$ form **a partition of unity**

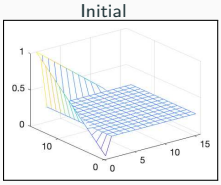
$$\varphi_j = \frac{\hat{\varphi}_j}{\sum_k \hat{\varphi}_k},$$

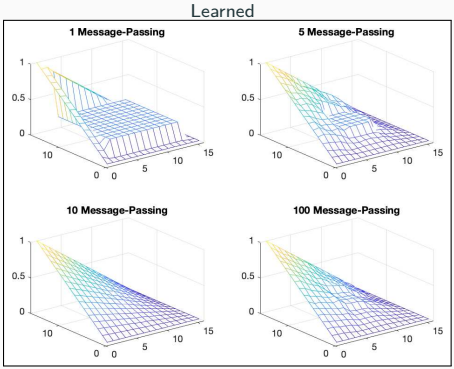
where the $\hat{\varphi}_k$ are the outputs of the GNN.

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**



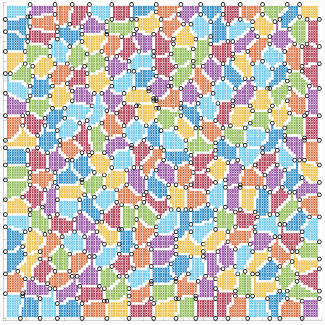


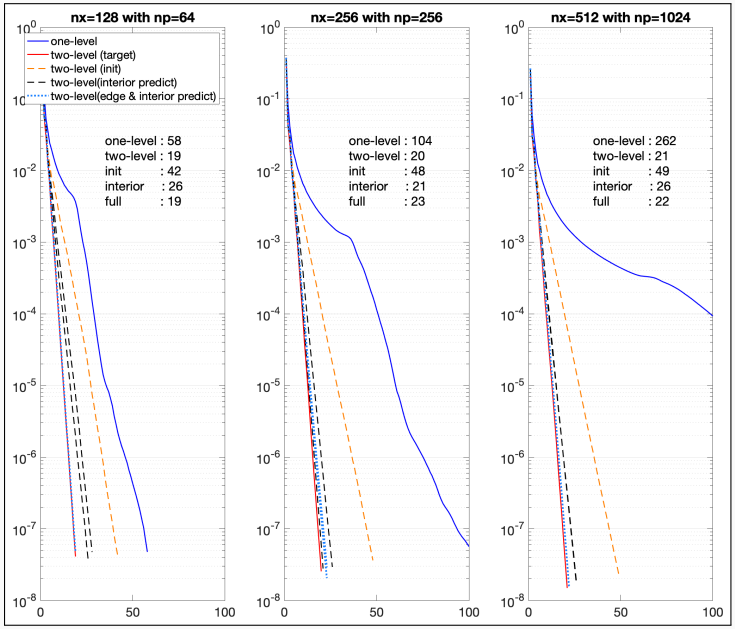
Numerical Results – Weak Scaling Study

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

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

decomposed using METIS:





Summary

- 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**.
- Recently, the following directions have been further developed:
  - **Multi-level preconditioners**
  - **Monolithic coarse spaces**
  - **GPU capabilities**
  - **Integration of machine learning techniques**

Outlook

- **Nonlinear Schwarz preconditioners**
- **Robust coarse spaces for heterogeneous problems** → Next talk by **Jascha Knepper**

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!