Efficient Schwarz Domain Decomposition Preconditioning Techniques on Current Hardware Using FROSch

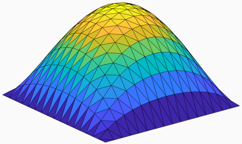
Alexander Heinlein[1] Sivasankaran Rajamanickam[2] Ichitaro Yamazaki[2]

SIAM Conference on Parallel Processing for Scientific Computing (PP24), Baltimore, Maryland, U.S., March 5 - 8, 2024

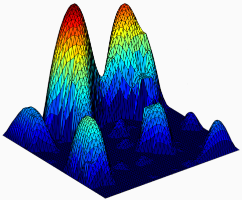
[1]Delft University of Technology

[2]Sandia National Laboratories

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

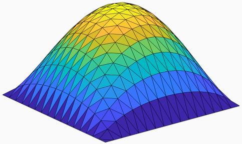
For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

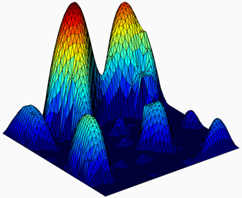
Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number** $\kappa(\boldsymbol{A})$. It deteriorates, e.g., for

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

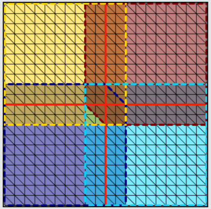
$\Rightarrow$ We introduce a preconditioner $\boldsymbol{M}^{-1} \approx \boldsymbol{A}^{-1}$ to improve the condition number:

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

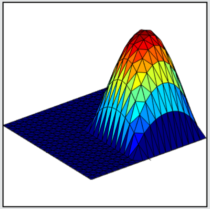
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



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

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

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

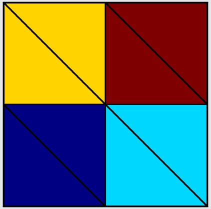
Condition number estimate:

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

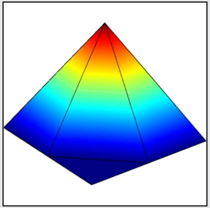
with subdomain size H and overlap width δ.

Lagrangian coarse space

Coarse triangulation



Coarse solution



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

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

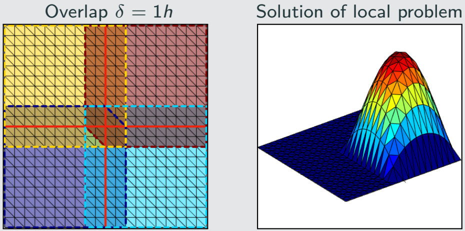
where Φ contains the coarse basis functions and $K_0 := \Phi^\top K \Phi$; cf., e.g., Toselli, Widlund (2005). The construction of a Lagrangian coarse basis requires a coarse triangulation.

Condition number estimate:

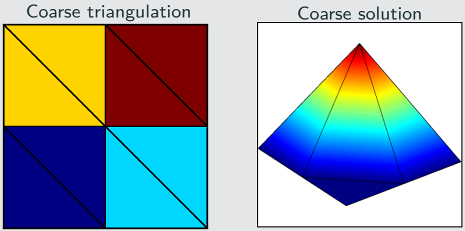
$$\kappa \left(M_{\text{OS-2}}^{-1} K \right) \leq C \left(1 + \frac{H}{\delta} \right)$$

Two-Level Schwarz Preconditioners

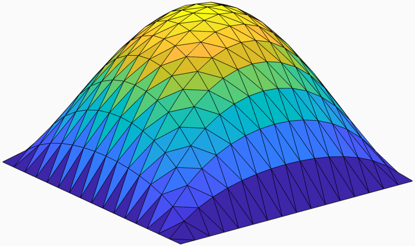
One-level Schwarz preconditioner

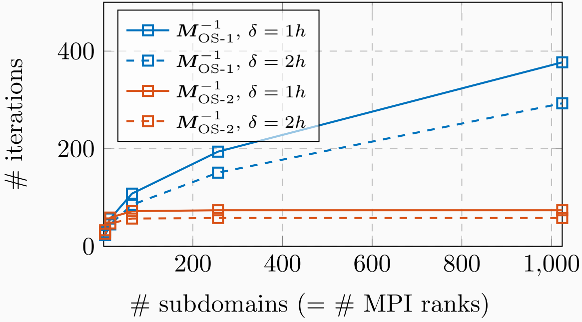


Lagrangian coarse space



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





FROSch (Fast and Robust Overlapping Schwarz) Framework in Trilinos

Software

- Object-oriented C++ domain decomposition solver framework with MPI-based distributed memory parallelization
- Part of TRILINOS with support for both parallel linear algebra packages EPETRA and TPETRA
- Node-level parallelization and performance portability on CPU and GPU architectures through KOKKOS and KOKKOSKERNELS
- Accessible through unified TRILINOS solver interface STRATIMIKOS

Methodology

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

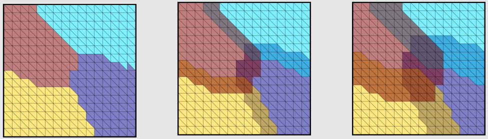
Team (active)

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

Algorithmic Framework for FROSch

First level – Overlapping DD

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



Nonoverl. DD | Overlap $\delta = 1h$ | Overlap $\delta = 2h$

First level – Computation K_i

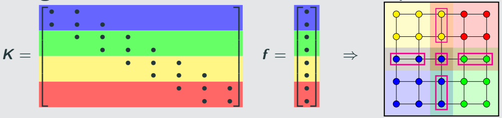
The overlapping matrices

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

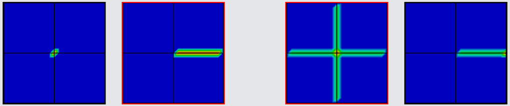
can easily be extracted from K since R_i is just a **global-to-local index mapping**.

Coarse level – Interface basis

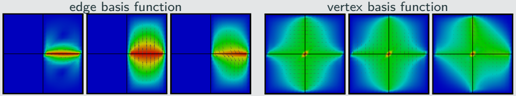
1. **Algebraic** identification of interface components:



2. Interface basis = partition of unity × null space



Coarse level – Extensions into interior



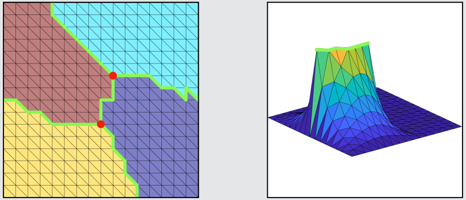
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

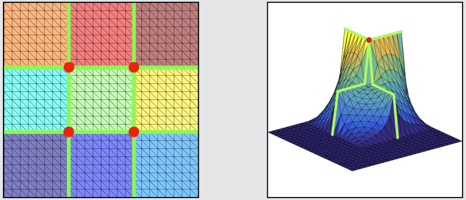
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)



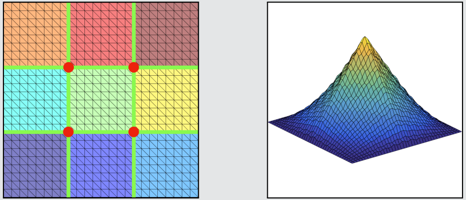
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)



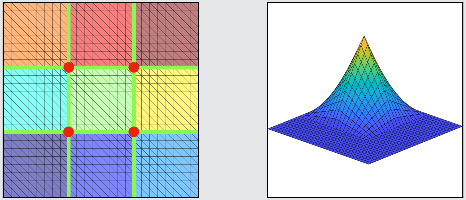
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

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



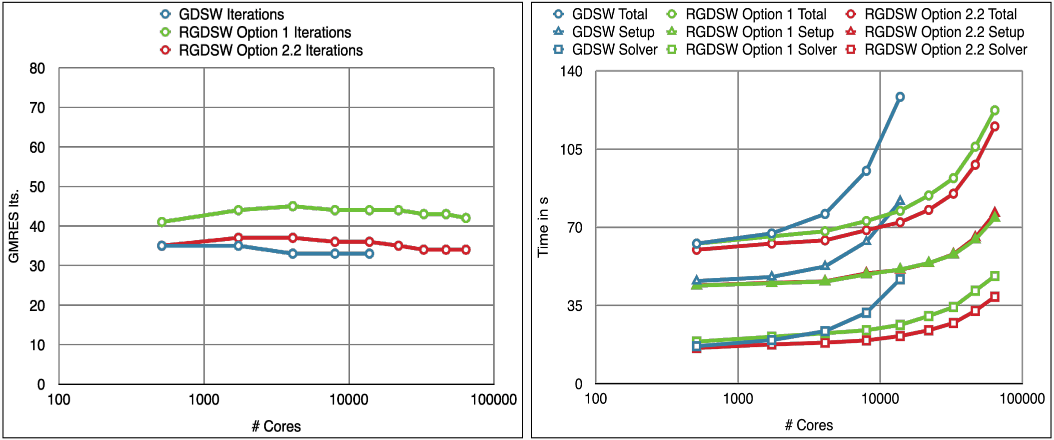
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

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.

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 solver	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	**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) k = 2	ILU(k) k = 3	symm. Gauß–Seidel 5 sweeps	symm. Gauß–Seidel 10 sweeps	Chebyshev polyn. p = 6	Chebyshev polyn. 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

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

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
$H/h = 20$, ≈ 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
$H/h = 40$, ≈ 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
$H/h = 60$, ≈ 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

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

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.

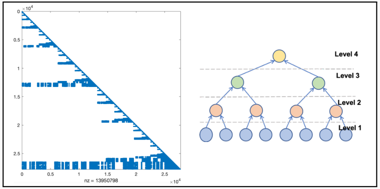
Parallel computations on dual-socket Intel Xeon Platinum machine at Sandia National Laboratories (Blake).

Performing the Subdomain Solves on GPUs

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)**

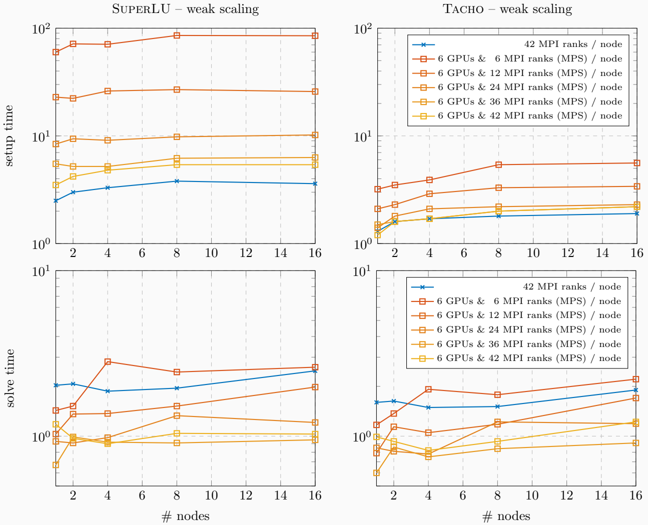


Tacho

- **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**
CPU	ND	1.6	2.6	4.4	7.4
GPU	KK(No)	1.4	1.5	1.8	2.4
GPU	KK(ND)	1.7	2.0	2.9	5.2
GPU	Fast(No)	**1.5**	**1.6**	**2.1**	**3.2**
GPU	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)**
CPU	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)
GPU	KK(ND)	2.89 (227)	4.27 (134)	5.57 (105)	6.36 (88)
GPU	Fast(No)	**1.14 (173)**	**1.11 (141)**	**1.26 (134)**	**1.43 (126)**
GPU	Fast(ND)	1.49 (227)	1.15 (137)	1.10 (109)	1.22 (100)
speedup		**2.2**×	**3.2**×	**4.3**×	**4.8**×

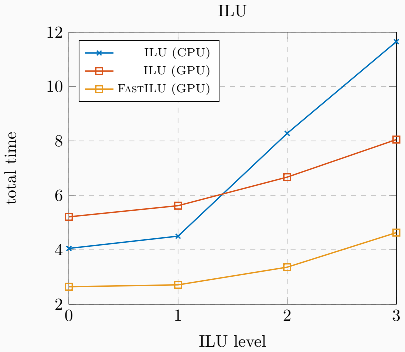
Computations on Summit (OLCF): 42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

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)

Learning Extension Operators Using Graph Neural Networks

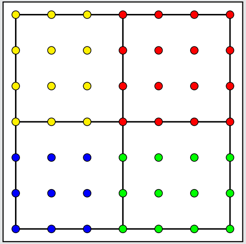
Why Learning Extension Operators

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:

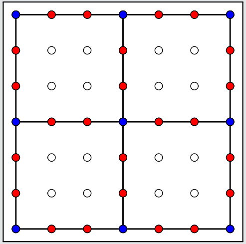
Subdomain-based



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

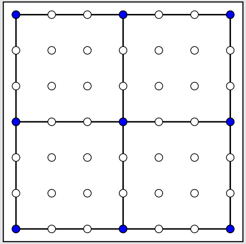
Cf. Nicolaides (1987)

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

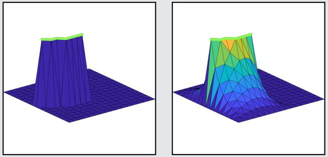
Energy-minimizing extensions

- are algebraic:

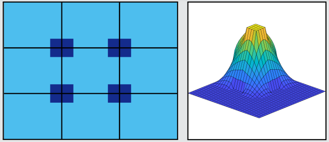
$$\boldsymbol{v}_I = -\boldsymbol{K}_{II}^{-1}\boldsymbol{K}_{I\Gamma}\boldsymbol{v}_\Gamma$$

(with Dirichlet b. c.)

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



Observation 2

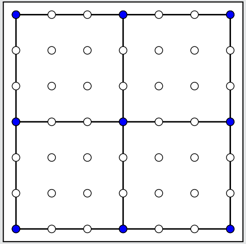


Heterogeneous: $\alpha_{\text{light}} = 1$; $\alpha_{\text{dark}} = 10^8$

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.

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:

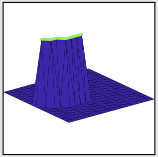
Observation 1

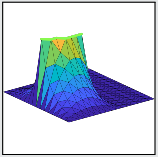
Energy-minimizing extensions

- are **algebraic**:

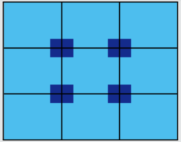
$$\boldsymbol{v}_I = -\boldsymbol{K}_{II}^{-1}\boldsymbol{K}_{I\Gamma}\boldsymbol{v}_\Gamma$$

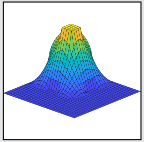
  (with Dirichlet b. c.)
- can be **costly**: solving a **problem in the interior**





Observation 2



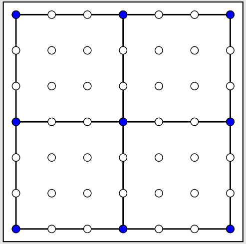


Heterogeneous: $\alpha_{\text{light}} = 1$; $\alpha_{\text{dark}} = 10^8$

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:** 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:

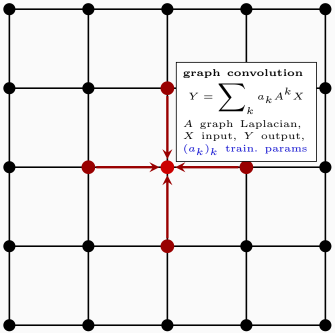
A. Heinlein, A. Klawonn, M. Lanser, J. Weber
Combining machine learning and domain decomposition methods for the solution of partial differential equations — A review
GAMM-Mitteilungen. 2021.

A. Klawonn, M. Lanser, J. Weber
Machine learning and domain decomposition methods – a survey
arXiv:2312.14050. 2023

Prediction via Graph Convolutional Networks

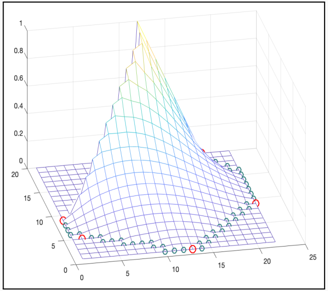
Graph convolutional networks (GCNs) introduced in **Kipf and Welling (2017)** are an example of **graph neural networks (GNNs)** and are well-suited for learning operations on simulation meshes:

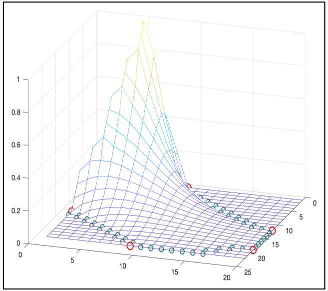
- **Generalization** of classical convolutional neural networks (CNNs) **LeCun (1998)** to **graph-based data sets**.
- Consist of **message passing layers**, which perform a **graph convolution operation on each node of the graph**.
- Graph convolutions are **invariant to position** and **permutation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

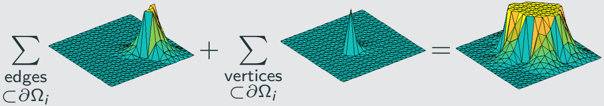




Theory-Inspired Design of the GNN-Based Coarse Space

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

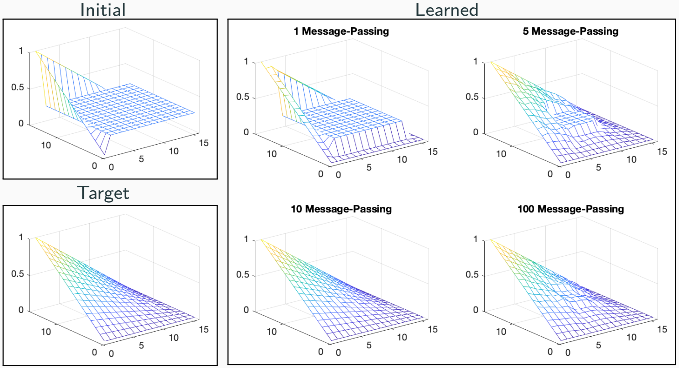
To **explicitly enforce** that the basis functions $(\varphi_j)_j$ form **a partition of unity**

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

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

Initial and target

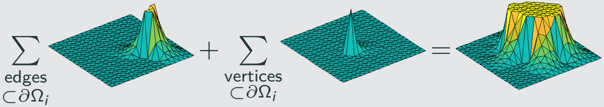
- **Initial function:** partition of unity that is constant in the interior
- **Target function:**
  - linear on the edges
  - energy-minimizing in the interior



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**

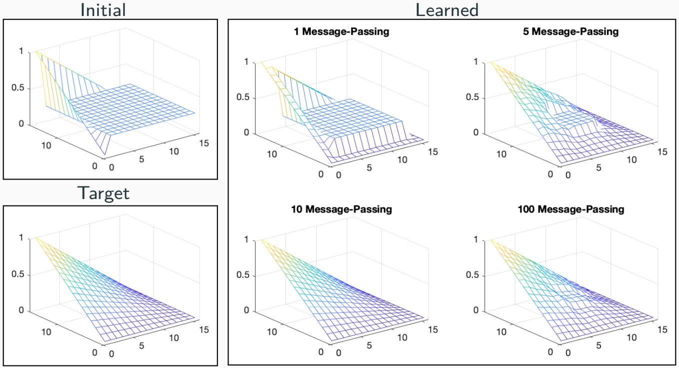
$$\varphi_j = \frac{\hat{\varphi}_j}{\sum_k \hat{\varphi}_k},$$

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**

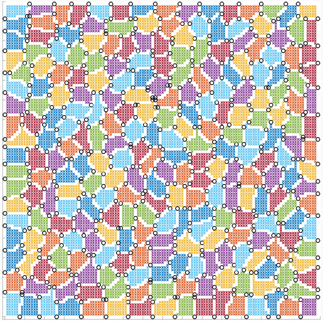


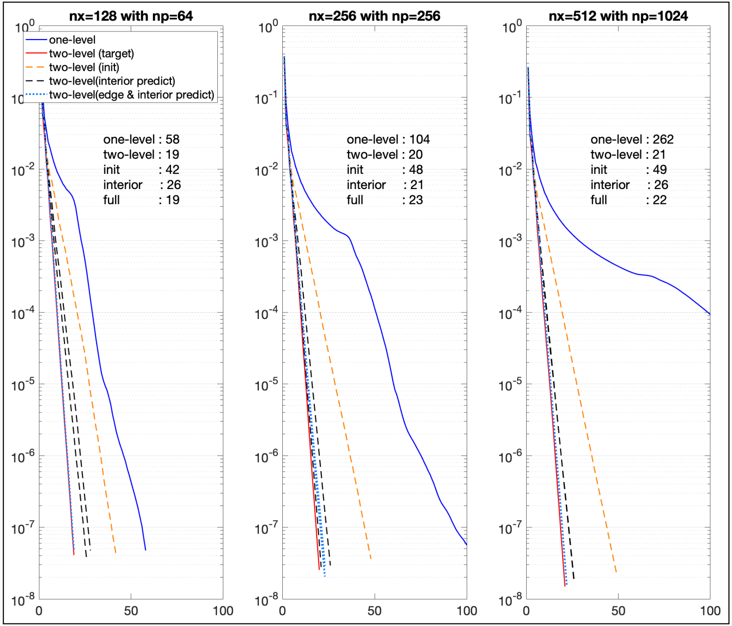
Numerical Results – Weak Scaling Study

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:





FROSch

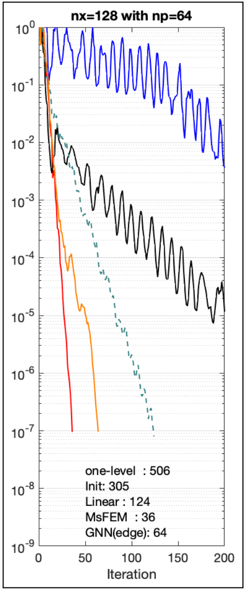
- FROSch is based on the **Schwarz framework** and **energy-minimizing coarse spaces**, which provide **numerical scalability** using **only algebraic information** for a **variety of applications**

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!