



Fast and Robust Overlapping Schwarz (FROSch) Preconditioners in Trilinos

New Developments and Applications

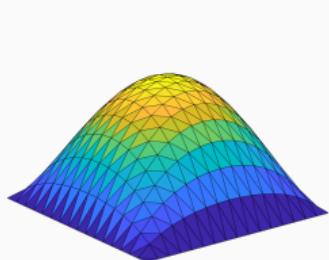
Alexander Heinlein¹

Trilinos User-Developer Group Meeting 2023 (Hybrid), CSRI, Albuquerque, USA, October 30 - November 2, 2023

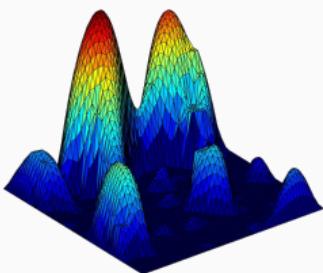
¹Delft University of Technology

Based on joint work with Oliver Rheinbach and Friederike Röver (Technische Universität Bergakademie Freiberg), Axel Klawonn and Lea Saßmannshausen (University of Cologne), and Sivasankaran Rajamanickam and Ichitaro Yamazaki (Sandia National Laboratories)

Solving A Model Problem



$$\alpha(x) = 1$$



$$\text{heterogeneous } \alpha(x)$$

Consider a **diffusion model problem**:

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

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

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

Direct solvers

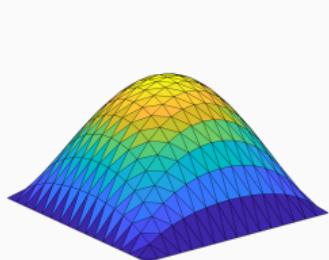
For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

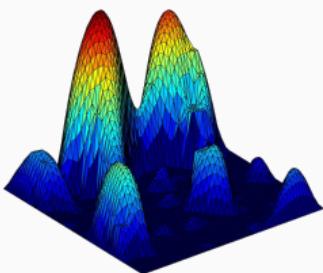
Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number $\kappa(\mathbf{A})$** . It deteriorates, e.g., for

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

Solving A Model Problem



$$\alpha(x) = 1$$



$$\text{heterogeneous } \alpha(x)$$

Consider a **diffusion model problem**:

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

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

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

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

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

Direct solvers

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

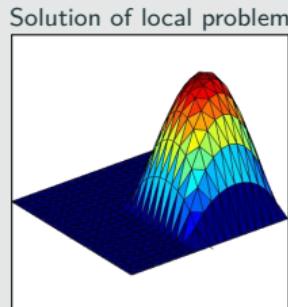
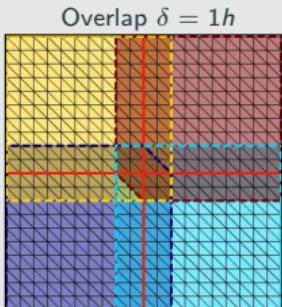
Iterative solvers

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

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

Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner



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

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

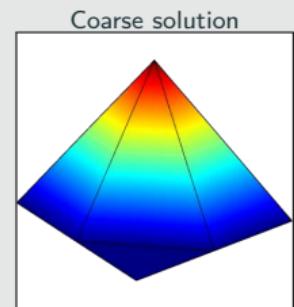
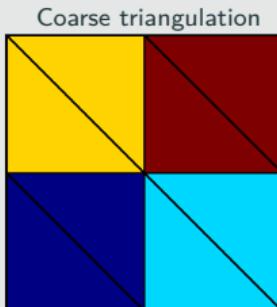
where R_i and R_i^T are restriction and prolongation operators corresponding to Ω'_i , and $K_i := R_i K R_i^T$.

Condition number estimate:

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

with subdomain size H and overlap width δ .

Lagrangian coarse space



The two-level overlapping Schwarz operator reads

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

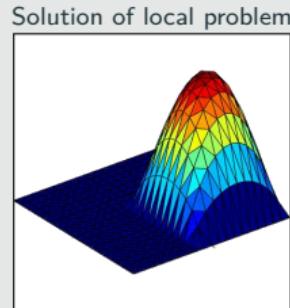
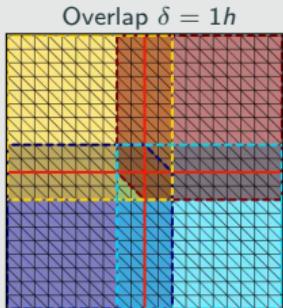
where Φ contains the coarse basis functions and $K_0 := \Phi^T K \Phi$; cf., e.g., [Toselli, Widlund \(2005\)](#).
The construction of a Lagrangian coarse basis requires a coarse triangulation.

Condition number estimate:

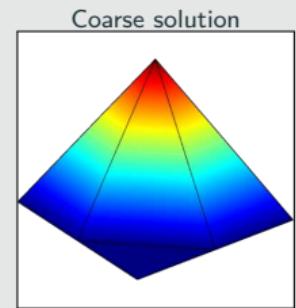
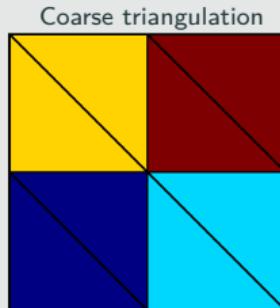
$$\kappa(M_{OS-2}^{-1} K) \leq C \left(1 + \frac{H}{\delta} \right)$$

Two-Level Schwarz Preconditioners

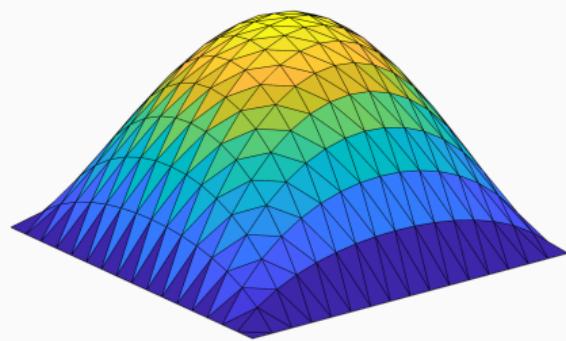
One-level Schwarz preconditioner



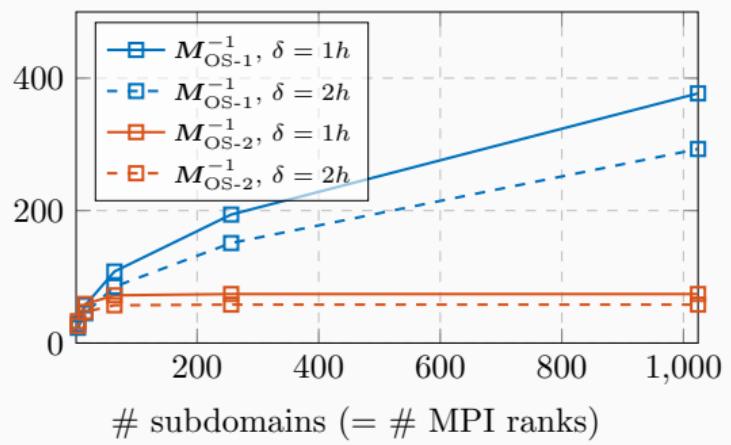
Lagrangian coarse space



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



iterations



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

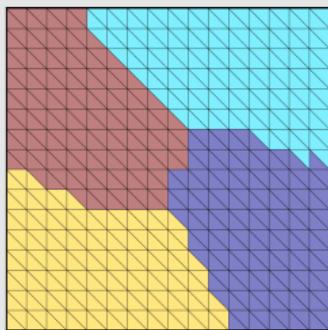
Team (active)

- | | |
|---------------------------------|------------------------------------|
| ▪ Alexander Heinlein (TU Delft) | ▪ Axel Klawonn (Uni Cologne) |
| ▪ Siva Rajamanickam (Sandia) | ▪ Oliver Rheinbach (TUBAF) |
| ▪ Friederike Röver (TUBAF) | ▪ Lea Saßmannshausen (Uni Cologne) |
| ▪ Ichitaro Yamazaki (Sandia) | |

Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

In FROSch, the overlapping subdomains $\Omega'_1, \dots, \Omega'_N$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K .

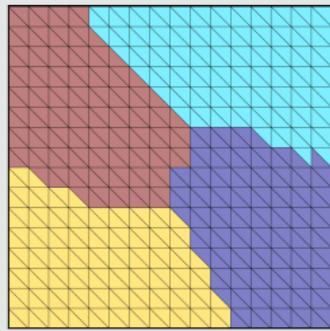


Nonoverlapping DD

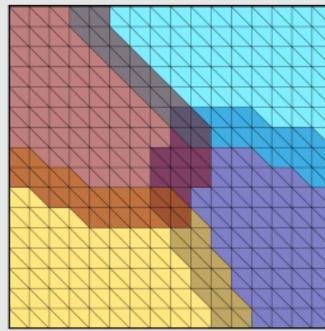
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

In FROSch, the overlapping subdomains $\Omega'_1, \dots, \Omega'_N$ 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

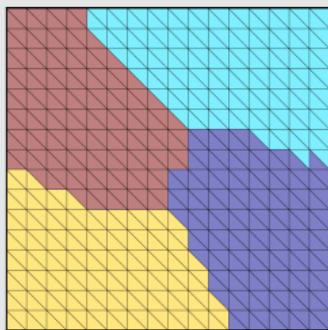


Overlap $\delta = 1h$

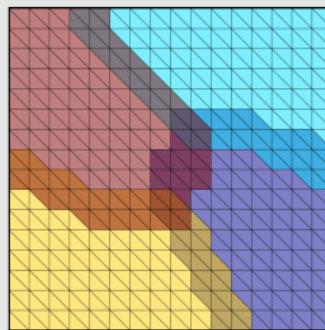
Algorithmic Framework for FROSCh Overlapping Domain Decompositions

Overlapping domain decomposition

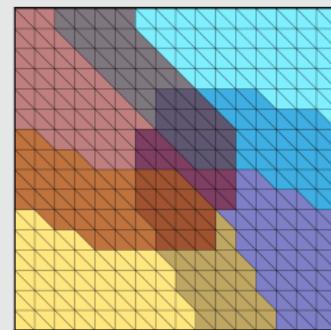
In FROSCh, the overlapping subdomains $\Omega'_1, \dots, \Omega'_N$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K .



Nonoverlapping DD



Overlap $\delta = 1h$

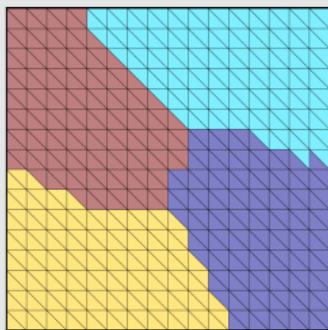


Overlap $\delta = 2h$

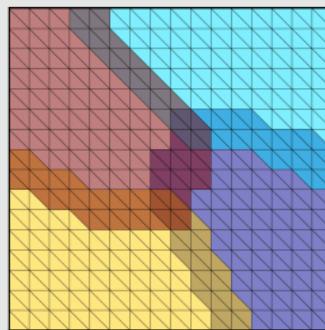
Algorithmic Framework for FROSCh Overlapping Domain Decompositions

Overlapping domain decomposition

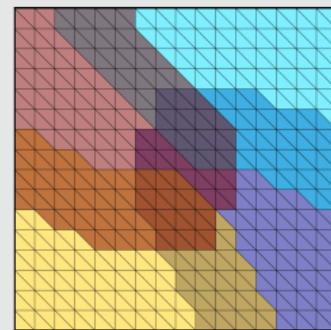
In FROSCh, the overlapping subdomains $\Omega'_1, \dots, \Omega'_N$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K .



Nonoverlapping DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$

Computation of the overlapping matrices

The overlapping matrices

$$K_i = R_i K R_i^T$$

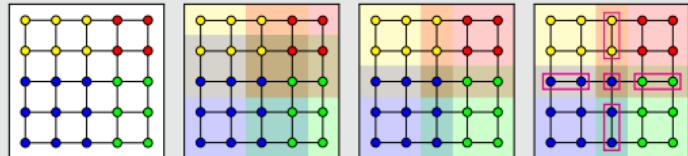
can easily be extracted from K since R_i is just a **global-to-local index mapping**.

Algorithmic Framework for FROSch Coarse Spaces

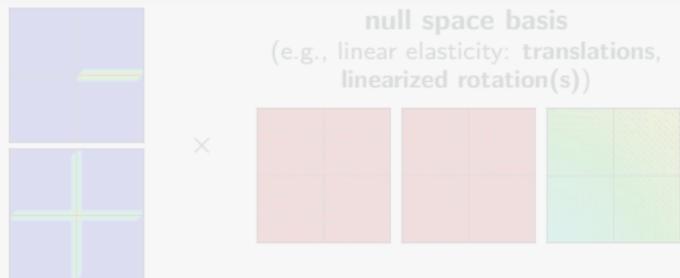
1. Identification interface components

$$K = \begin{bmatrix} & & \\ & & \\ & & \\ & & \\ & & \end{bmatrix} \quad f = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

Identification from parallel distribution of matrix:
distributed map overlapping map repeated map interface comp.



3. Interface basis



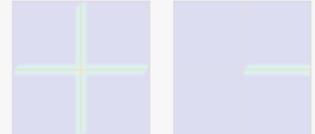
The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

2. Interface partition of unity (IPOU)

vertex & edge functions

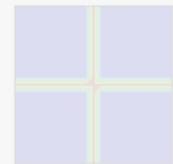


vertex functions



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

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



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:

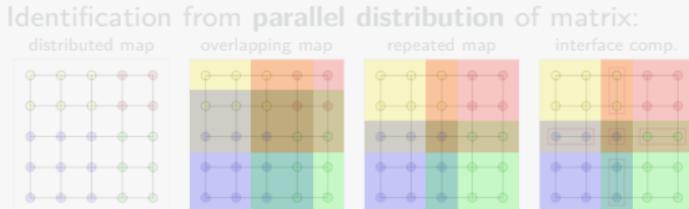
$$\Phi = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} -K_{II}^{-1} K_{\Gamma I}^T \Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix}.$$

(For elliptic problems: energy-minimizing extension)

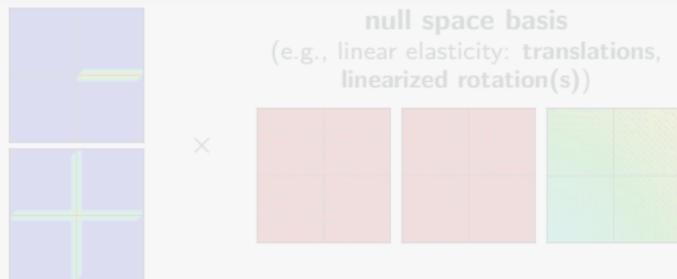
Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$$K = \begin{bmatrix} & & \\ & \cdot & \end{bmatrix} \quad f = \begin{bmatrix} & & \\ & \cdot & \end{bmatrix}$$



3. Interface basis



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

2. Interface partition of unity (IPOU)

vertex & edge functions

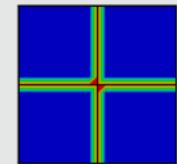


vertex functions



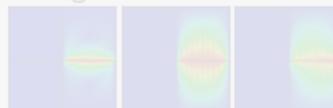
Based on the interface components, construct an **interface partition of unity**:

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



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:

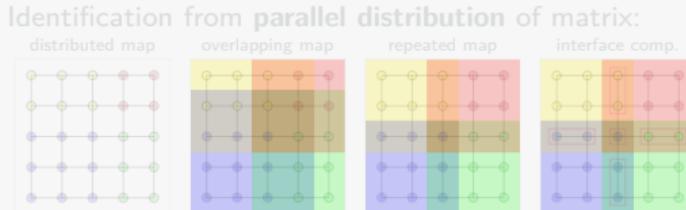
$$\Phi = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} -K_{II}^{-1} K_{\Gamma I}^T \Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix}.$$

(For elliptic problems: energy-minimizing extension)

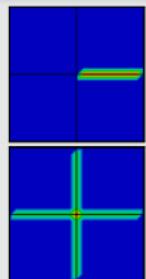
Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$$K = \begin{bmatrix} & & \\ & \cdot & \\ & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot & \cdot & \cdot \\ & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & & & & & & \end{bmatrix} \quad f = \begin{bmatrix} & \\ & \cdot \end{bmatrix}$$



3. Interface basis



\times



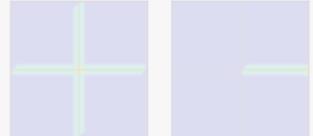
The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

2. Interface partition of unity (IPOU)

vertex & edge functions

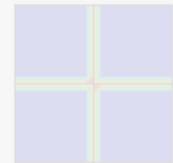


vertex functions



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

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



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:

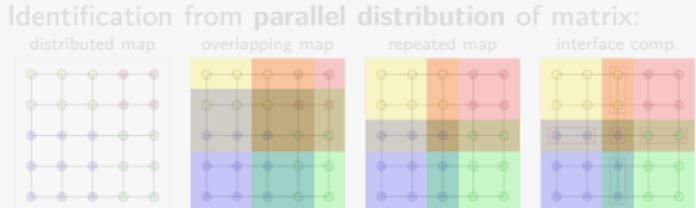
$$\Phi = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} -K_{II}^{-1} K_{\Gamma I}^T \Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix}.$$

(For elliptic problems: energy-minimizing extension)

Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$$K = \begin{bmatrix} & & \\ & \ddots & \\ & & \end{bmatrix} \quad f = \begin{bmatrix} \\ \\ \vdots \\ \end{bmatrix}$$



3. Interface basis



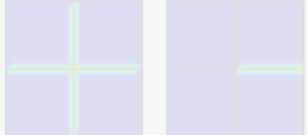
The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

2. Interface partition of unity (IPOU)

vertex & edge functions



vertex functions



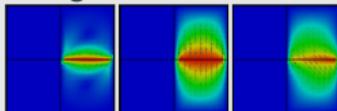
Based on the interface components, construct an **interface partition of unity**:

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

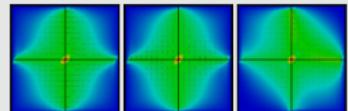


4. Extension into the interior

edge basis function



vertex basis function



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

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

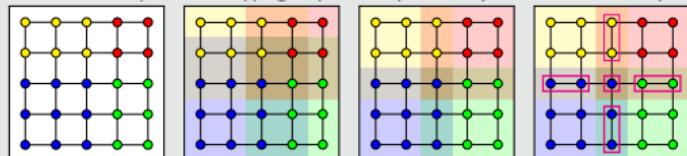
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

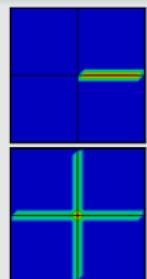
1. Identification interface components

$$K = \begin{bmatrix} & & \\ & \vdots & \\ & & \end{bmatrix} \quad f = \begin{bmatrix} & \\ & \vdots \\ & \end{bmatrix}$$

Identification from parallel distribution of matrix:
distributed map overlapping map repeated map interface comp.



3. Interface basis



\times



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**.

2. Interface partition of unity (IPOU)

vertex & edge functions

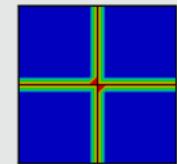


vertex functions



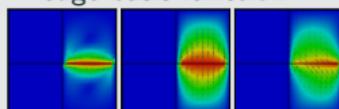
Based on the interface components,
construct an **interface partition of unity**:

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

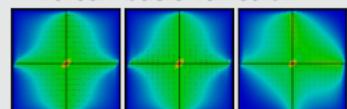


4. Extension into the interior

edge basis function



vertex basis function



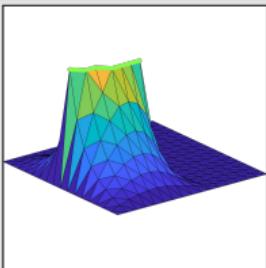
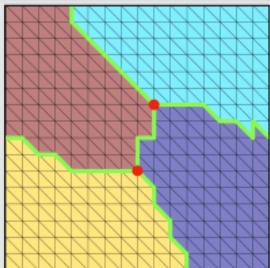
The values in the interior of the subdomains are computed via the **extension operator**:

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

(For elliptic problems: energy-minimizing extension)

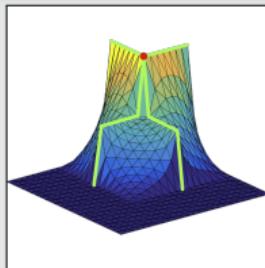
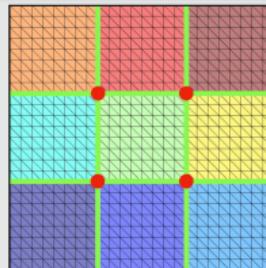
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)



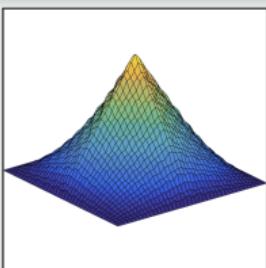
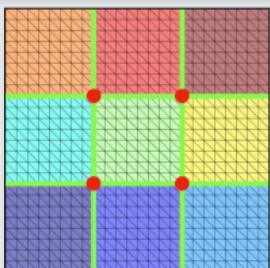
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)



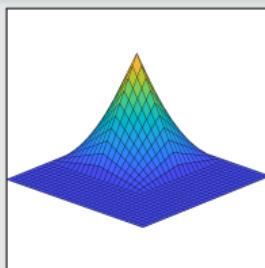
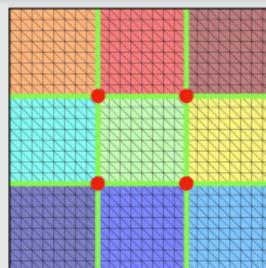
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)



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

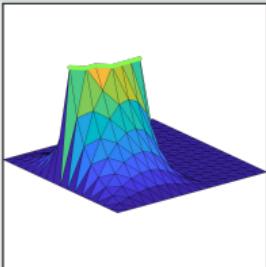
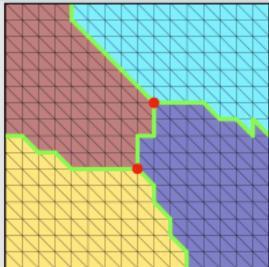
Q1 Lagrangian / piecewise bilinear



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

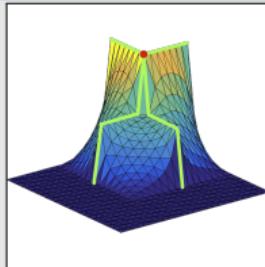
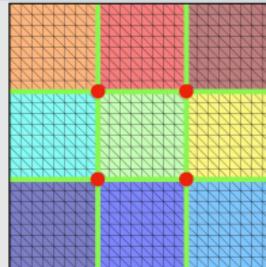
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)



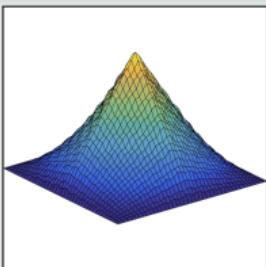
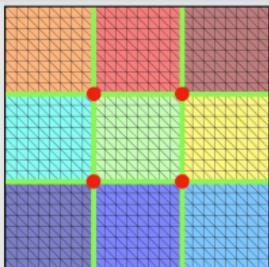
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)



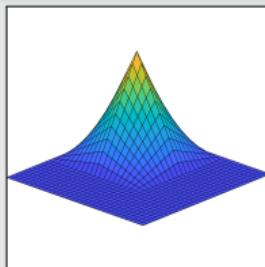
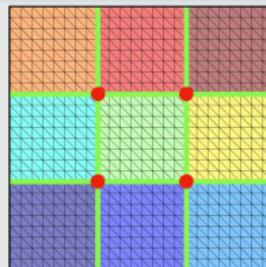
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

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



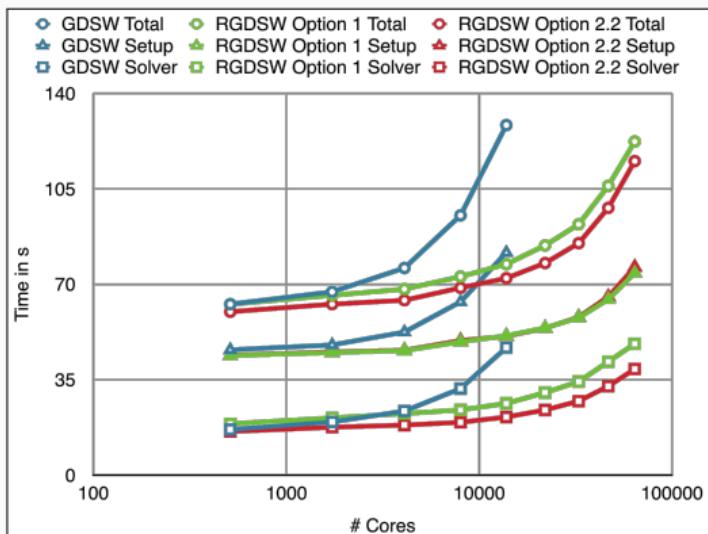
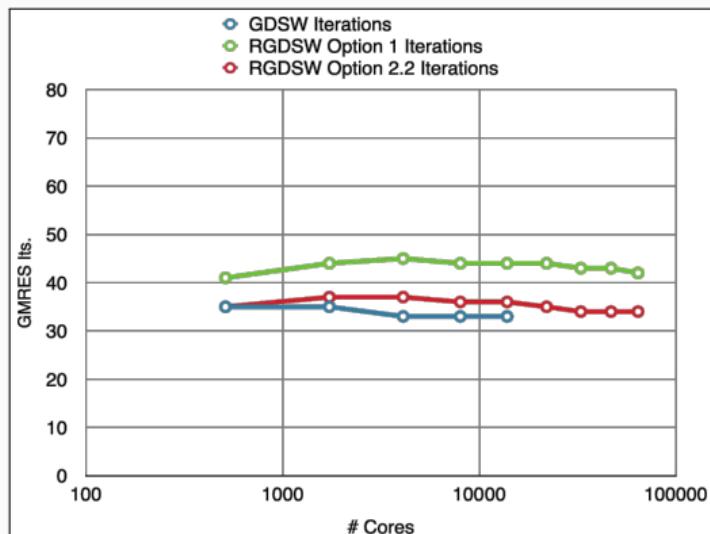
Piecewise linear interface partition of unity functions and a structured domain decomposition.

Weak Scalability up to 64 k MPI ranks / 1.7 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.

⇒ Using the **reduced dimension coarse space**, we can **improve parallel scalability**.

To extend the scalability even further, we consider **multi-level Schwarz preconditioners**.

1 Multilevel Schwarz Preconditioners in FROSCH

Based on joint work with **Oliver Rheinbach** and **Friederike Röver** (University of Cologne)

2 Monolithic Schwarz Preconditioners in FROSCH

Based on joint work with **Axel Klawonn** and **Lea Saßmannshausen** (University of Cologne)

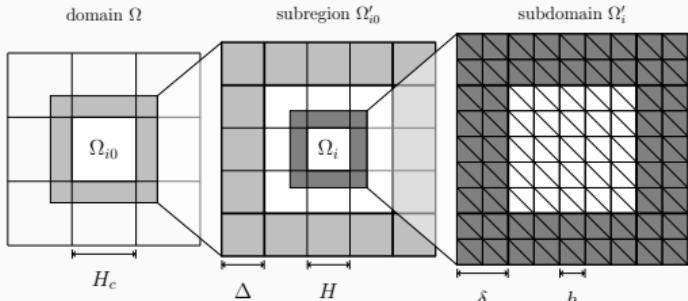
3 FROSCH Preconditioners on GPUs

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

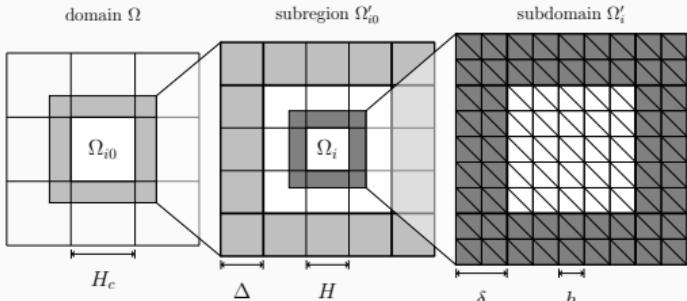
Algorithm 1: Application of the l th level of an L level FROSCH preconditioner

Function FROSCH(K, x, l):

```
for  $i := 1$  to  $N^{(l)}$  do construct  $R_i$  and compute  $K_i = R_i K R_i^\top$ ;           /* fine level */  
construct  $\Phi$  and compute  $K_0 = \Phi^\top K \Phi$ ;  
 $x = \Phi^\top x$ ;  
if  $l < L$  then  $x = K_0^{-1} x$ ;  
else  $x = \text{FROSCH}(K_0, x, l + 1)$ ;  
 $x = \Phi x$ ;  
for  $i := 1$  to  $N^{(l)}$  do  $x = x + R_i^\top K_i^{-1} R_i x$ ;  
return  $x$ ;  
/* coarse level */  
/* coarse interpolation */  
/* exact coarse solver */  
/* inexact coarse solver */  
/* fine interpolation */  
/* fine level updates */
```

end

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**

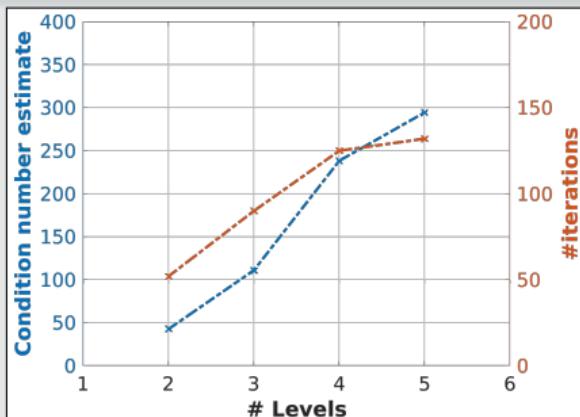
Influence of the inexact coarse solver

Two-dimensional Laplacian model problem with

- fixed global problem size: $\approx 530 \text{ m}$
- fixed number of subdomains on the first level: 16 384

Increasing the number of levels results in a **slight increase in the condition number and iteration count**

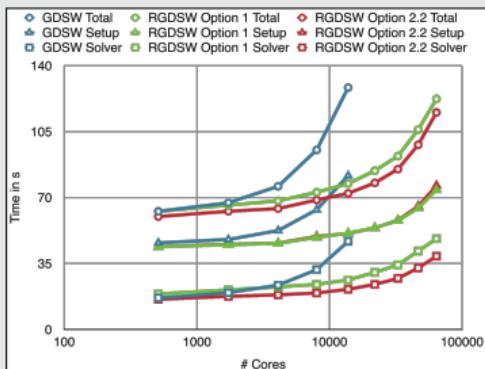
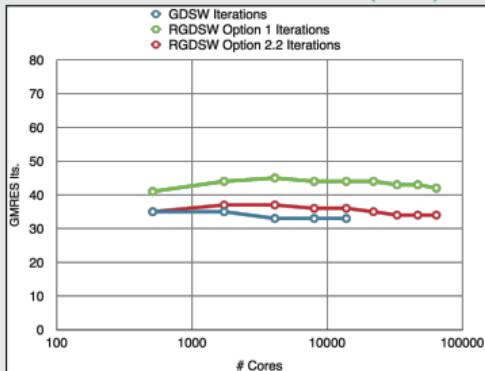
Let us discuss the effect on the computing times next.



Weak Scalability up to 64 k MPI ranks / 1.7 b Unknowns (3D Poisson; Juqueen)

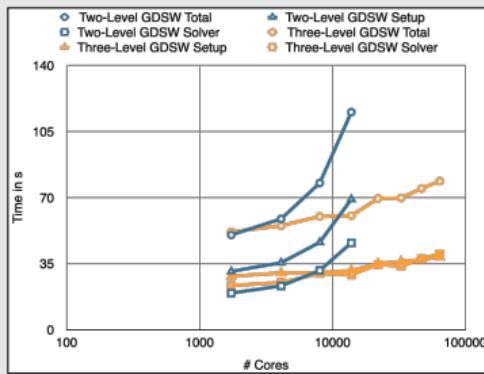
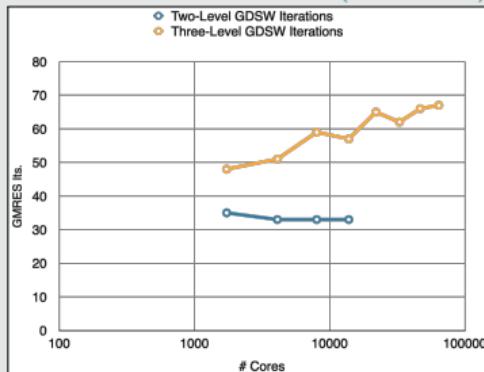
GDSW vs RGDSW (reduced dimension)

Heinlein, Klawonn, Rheinbach, Widlund (2019).



Two-level vs three-level GDSW

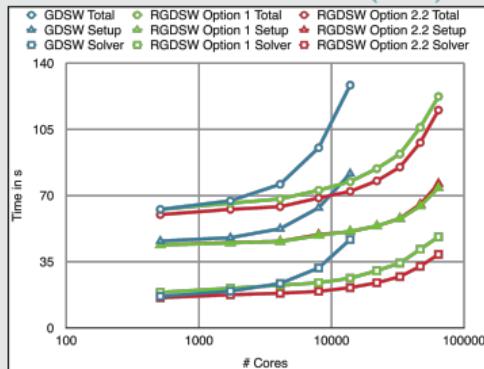
Heinlein, Klawonn, Rheinbach, Röver (2019, 2020).



Weak Scalability up to 64 k MPI ranks / 1.7 b Unknowns (3D Poisson; Juqueen)

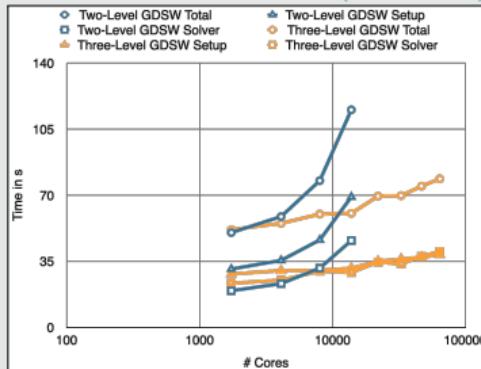
GDSW vs RGDSW (reduced dimension)

Heinlein, Klawonn, Rheinbach, Widlund (2019).



Two-level vs three-level GDSW

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



# subdomains (=cores)		1 728	4 096	8 000	13 824	21 952	32 768	46 656	64 000
GDSW	Size of K_0	10 439	25 695	51 319	89 999	-	-	-	-
	Size of K_{00}	98	279	604	1 115	1 854	2 863	4 184	5 589
RGDSW	Size of K_0	1 331	3 375	6 859	12 167	19 683	29 791	42 875	59 319
	Size of K_{00}	8	27	64	125	216	343	512	729

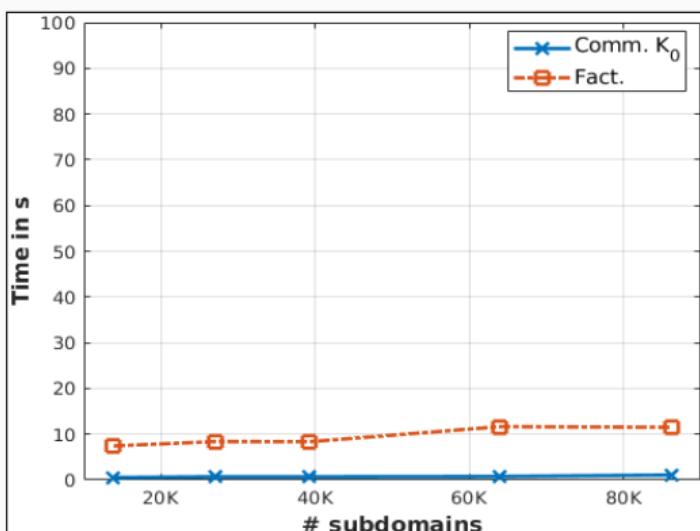
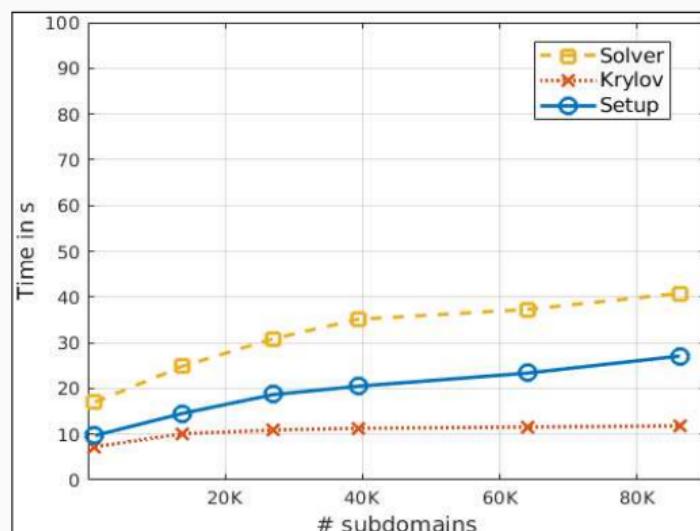
Weak Scalability of the Three-Level RGDSW Preconditioner – SuperMUC-NG

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

Largest problem: 2 044 416 000 unknowns

Coarse solver level 3: Intel MKL Pardiso (direct)



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

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

Monolithic GDSW preconditioner

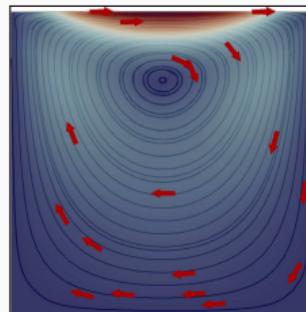
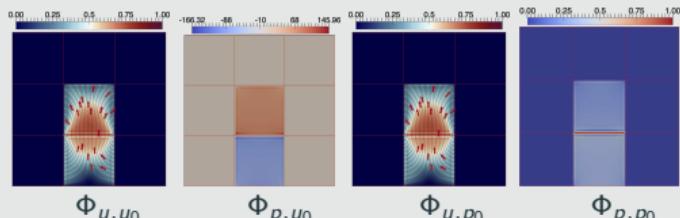
We construct a **monolithic GDSW preconditioner**

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

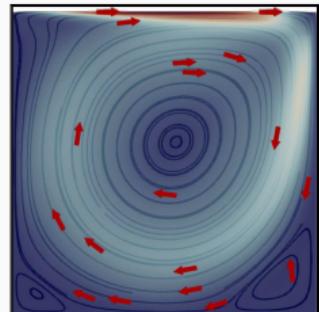
with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$, and

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

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



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

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

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

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

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

Block diagonal & triangular preconditioners

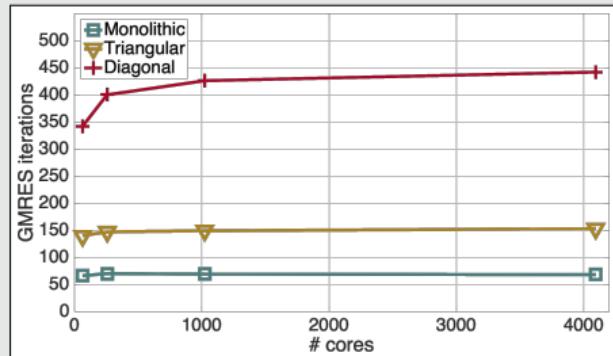
Block-diagonal preconditioner:

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

Block-triangular preconditioner:

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

Monolithic vs. block prec. (Stokes)



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

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

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

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

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

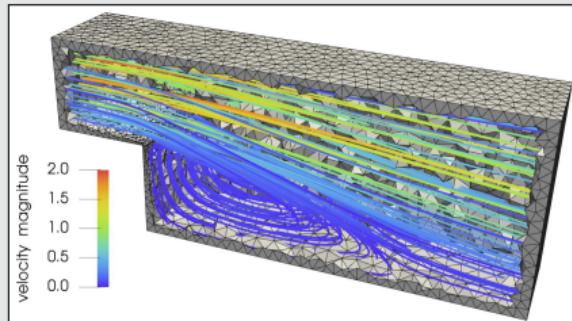
$$m_{\text{SIMPLE}}^{-1} = \begin{bmatrix} I & -D^{-1}B \\ 0 & \alpha I \end{bmatrix} \begin{bmatrix} K^{-1} & 0 \\ -\hat{S}^{-1}BK^{-1} & \hat{S}^{-1} \end{bmatrix};$$

see [Patankar and Spalding \(1972\)](#). Here,

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

We approximate the inverses using (R)GDSW preconditioners.

Monolithic vs. SIMPLE preconditioner

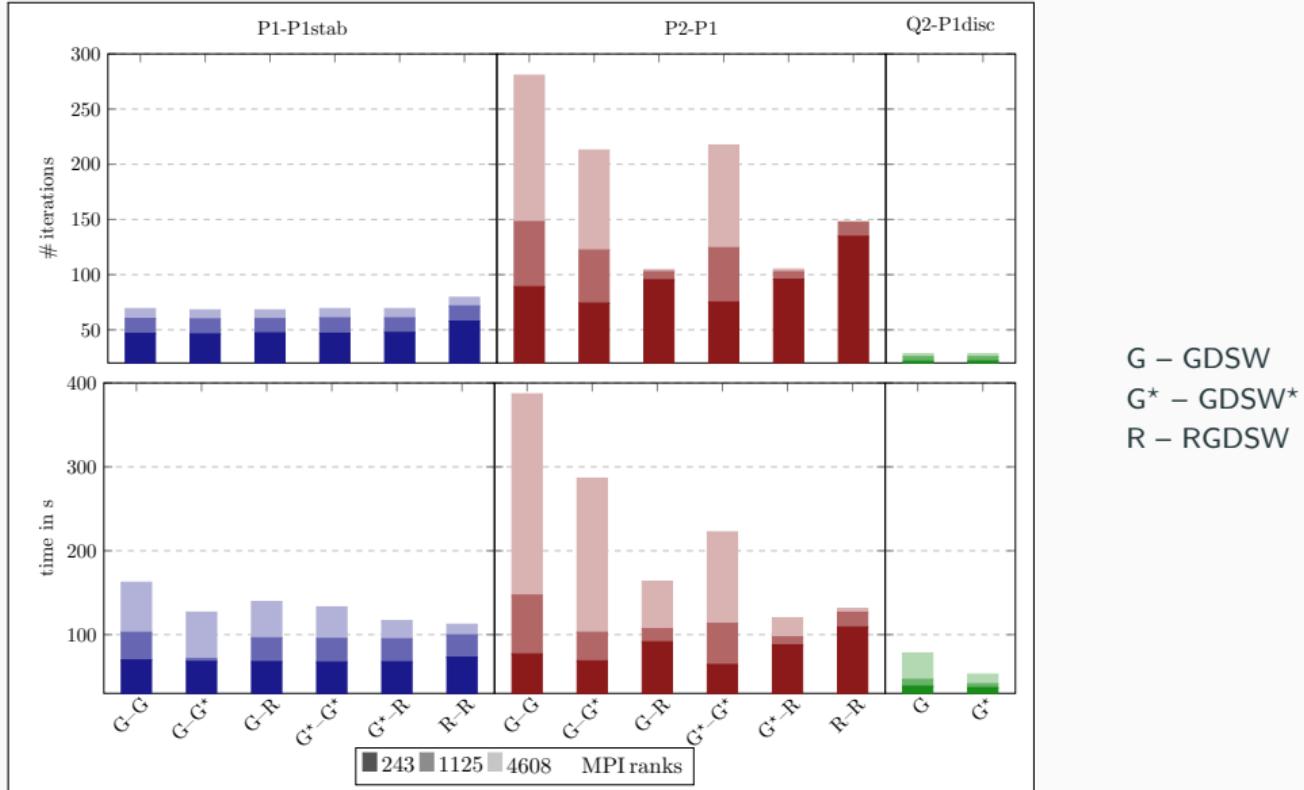


Steady-state Navier–Stokes equations

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

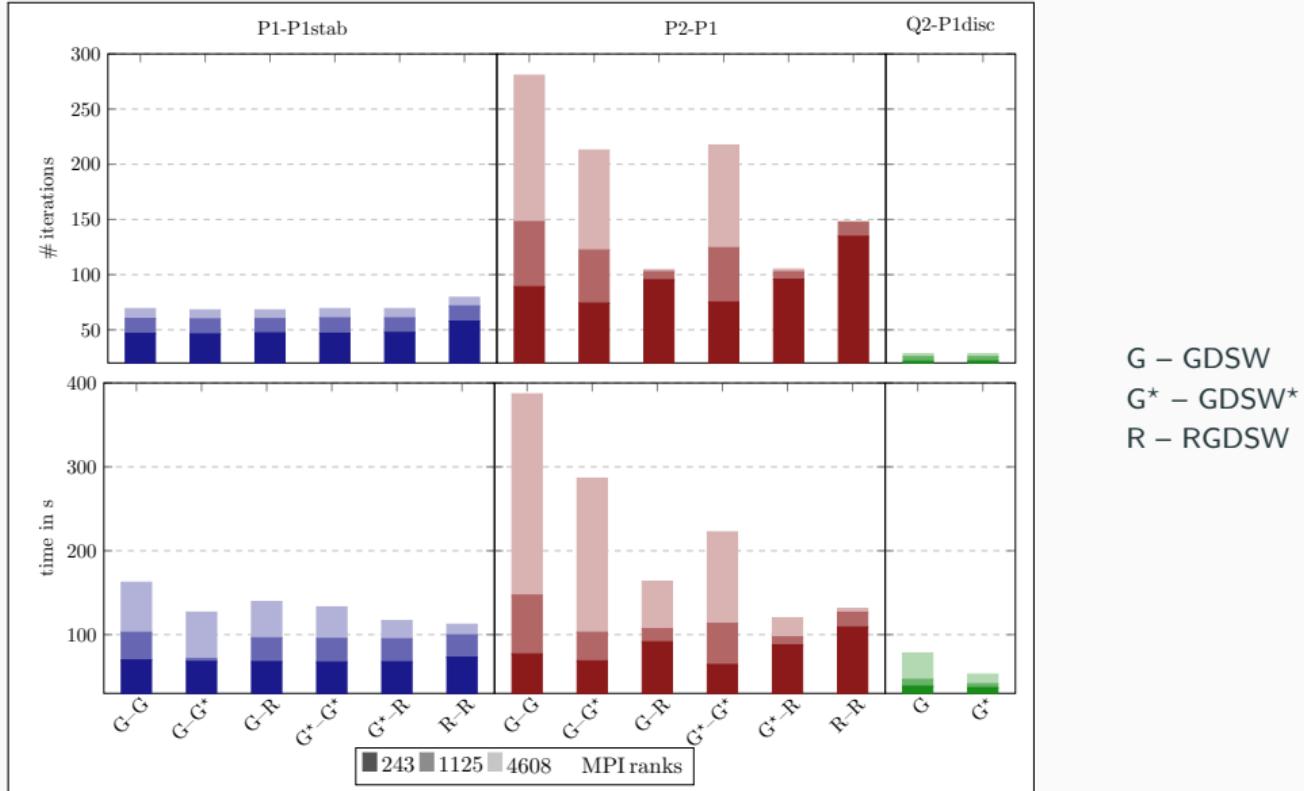
Computations on Piz Daint (CSCS). Implementation in the finite element software FEDDLib.

Coarse Spaces for Monolithic FROSch Preconditioners for CFD Simulations



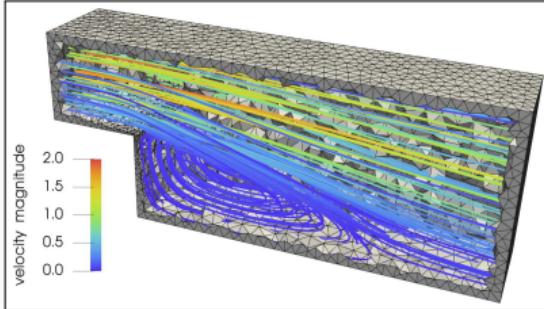
G – GDSW
G* – GDSW*
R – RGDSW

Coarse Spaces for Monolithic FROSch Preconditioners for CFD Simulations



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

Monolithic Vs. Block (R)GDSW Preconditioners for CFD Simulations



Problem:
Steady-state
Navier–Stokes
equations

Computations on
Fritz (FAU).
Implementation in
the finite element
software FEDDLIB.

Monolithic GDSW preconditioner

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

SIMPLE block preconditioner

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

P1–P1 stabilized, $H/h = 20$

prec.	# MPI ranks	243	1 125	4 608
Monolithic RGDSW (FROSCH)	# its.	57.8 (5)	71.6 (5)	79.4 (5)
	setup	33.6 s	39.6 s	40.3 s
	solve	39.4 s	60.0 s	72.3 s
	total	73.0 s	99.6 s	112.3 s
SIMPLE RGDSW (TEKO & FROSCH)	# its.	168.4 (5)	196.8 (5)	200.0 (5)
	setup	20.7 s	32.2 s	26.9 s
	solve	108.2 s	156.0 s	175.0 s
	total	128.9 s	189.4 s	209.6 s

P2–P1, $H/h = 9$

prec.	# MPI ranks	243	1 125	4 608
Monolithic GDSW*– RGDSW (FROSCH)	# its.	95.7 (6)	102.6 (5)	105.0 (5)
	setup	31.1 s	33.8 s	42.2 s
	solve	56.7 s	63.4 s	77.9 s
	total	87.8 s	97.2 s	120.1 s
SIMPLE RGDSW (TEKO & FROSCH)	# its.	157.5 (6)	161.8 (5)	169.8 (5)
	setup	22.0 s	25.3 s	26.1 s
	solve	86.8 s	92.2 s	112.4 s
	total	108.8 s	117.5 s	138.5 s

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

FROSch Preconditioners on GPUs

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

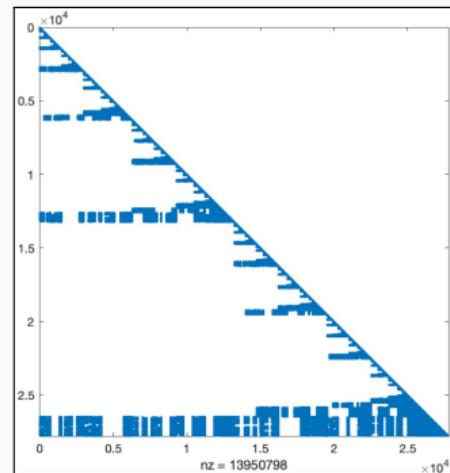
The sparse triangular solver is an **important kernel** in many codes (including FROSch) but is **challenging to parallelize**

- Factorization using a **sparse direct solver** typically leads to triangular matrices with **dense blocks** called **supernodes**
- In **supernodal triangular solvers**, rows/columns with a similar sparsity pattern are merged into a supernodal block, and the **solve is then performed block-wise**
- The **parallelization potential** for the triangular solver is **determined by the sparsity pattern**

Parallel supernode-based triangular solver:

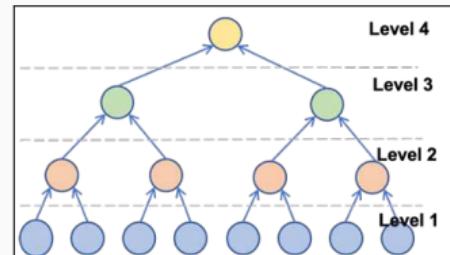
1. **Supernode-based level-set scheduling**, where **all leaf-supernodes within one level are solved in parallel**
(batched kernels for hierarchical parallelism)
2. **Partitioned inverse** of the submatrix associated with each level:
SpTRSV is transformed into a sequence of SpMVs

See [Yamazaki, Rajamanickam, and Ellingwood \(2020\)](#) for more details.

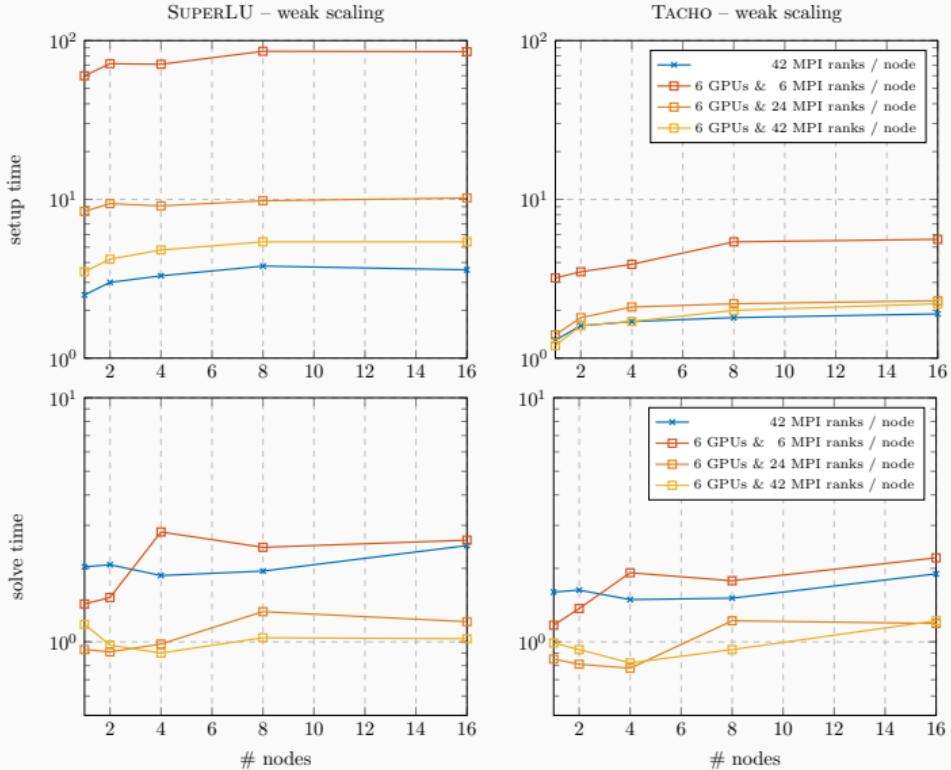


Lower-triangular matrix – SuperLU

with METIS nested dissection ordering



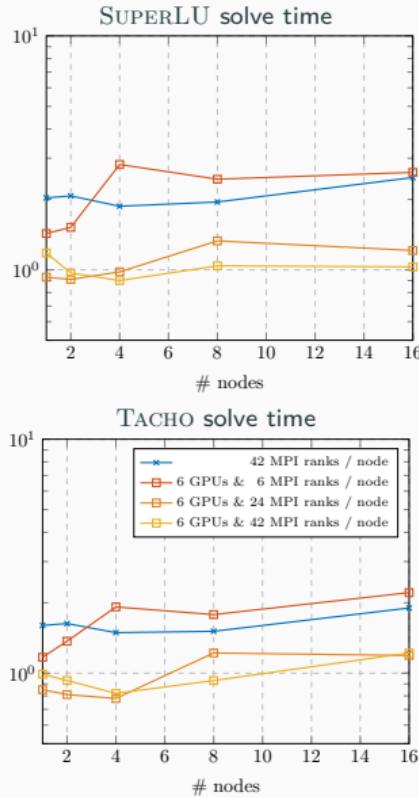
Three-Dimensional Linear Elasticity – Weak Scalability



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



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

# nodes	1	2	4	8	16
# dofs	375K	750K	1.5M	3M	6M
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)
4	0.93 (59)	0.91 (53)	0.98 (59)	1.33 (77)	1.21 (66)
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)
4	0.85 (59)	0.81 (53)	0.78 (59)	1.22 (77)	1.19 (66)
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×

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×

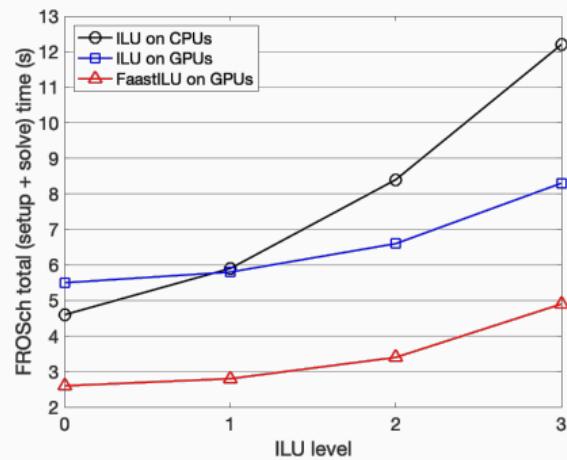
Computations on Summit (OLCF):
42 IBM Power9 CPU cores and 6 NVIDIA
V100 GPUs per node.

Yamazaki, Heinlein,
Rajamanickam (2023)

ILU variants

- KOKKOSKERNELS ILU (KK)
- 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

# 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
	Fast	1.5	2.2	2.3	2.5
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)
	Fast	1.2 (154)	1.0 (133)	1.1 (130)	1.3 (117)
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)

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,
 - multi-level preconditioners,
 - monolithic coarse spaces,
 - and GPU capabilitieshave been developed further.

Outlook

- Nonlinear preconditioners
- Robust coarse spaces for heterogeneous problems

Acknowledgements

- **Financial support:** DFG (KL2094/3-1, RH122/4-1), DFG SPP 2311 project number 465228106
- **Computing resources:** Summit (OLCF), Cori (NERSC), magnitUDE (UDE), Piz Daint (CSCS), Fritz (FAU)

Thank you for your attention!