Fast and Robust Overlapping Schwarz (FROSch) Preconditioners in Trilinos

New Developments and Applications

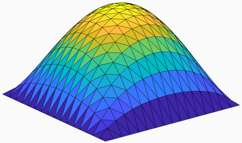
Alexander Heinlein[1]

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

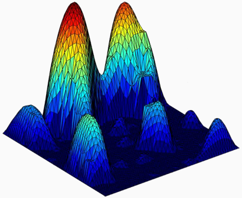
[1]Delft University of Technology

Based on joint work with Oliver Rheinbach and Friederike Röver (Technische Universität Bergakademie Freiberg), Axel Klawonn and Lea Saßmannshausen (Universität zu Köln), and Sivasankaran Rajamanickam and Ichitaro Yamazaki (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

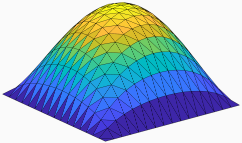
For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

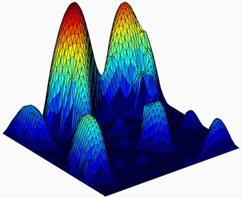
Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number** $\kappa(\boldsymbol{A})$. It deteriorates, e.g., for

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

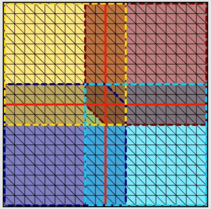
$\Rightarrow$ We introduce a preconditioner $\boldsymbol{M}^{-1} \approx \boldsymbol{A}^{-1}$ to improve the condition number:

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

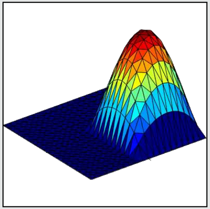
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



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

$$\boldsymbol{M}_{\text{OS-1}}^{-1}\boldsymbol{K} = \sum_{i=1}^{N} \boldsymbol{R}_i^T \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{K},$$

where $\boldsymbol{R}_i$ and $\boldsymbol{R}_i^T$ are restriction and prolongation operators corresponding to Ω_i', and $\boldsymbol{K}_i := \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$.

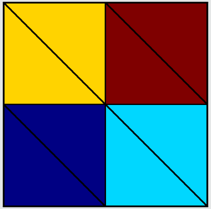
Condition number estimate:

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

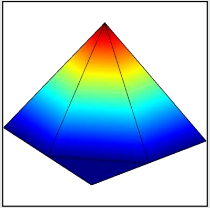
with subdomain size H and overlap width δ.

Lagrangian coarse space

Coarse triangulation



Coarse solution



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

$$\boldsymbol{M}_{\text{OS-2}}^{-1}\boldsymbol{K} = \underbrace{\Phi \boldsymbol{K}_0^{-1} \Phi^T \boldsymbol{K}}_{\text{coarse level – global}} + \underbrace{\sum_{i=1}^{N} \boldsymbol{R}_i^T \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{K}}_{\text{first level – local}},$$

where Φ contains the coarse basis functions and $\boldsymbol{K}_0 := \Phi^T \boldsymbol{K} \Phi$; cf., e.g., Toselli, Widlund (2005). The construction of a Lagrangian coarse basis requires a coarse triangulation.

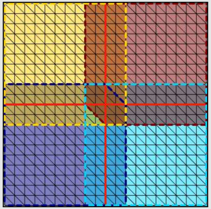
Condition number estimate:

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

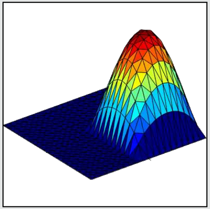
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$

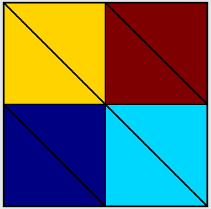


Solution of local problem

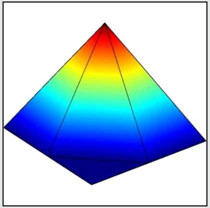


Lagrangian coarse space

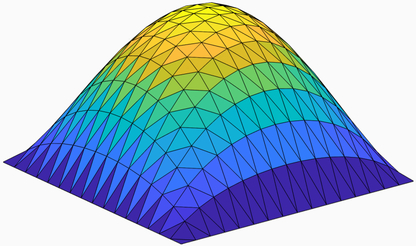
Coarse triangulation

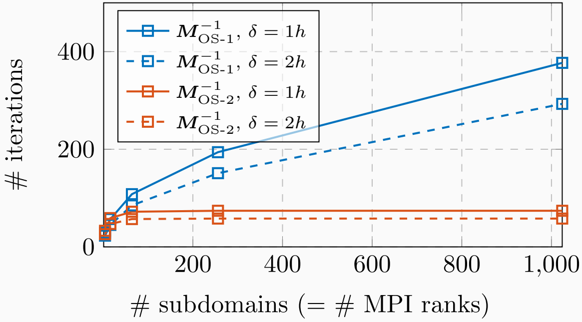


Coarse solution



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

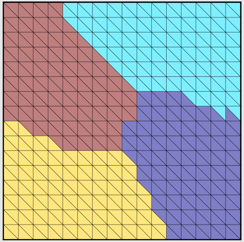
- Parallel scalable multi-level Schwarz domain decomposition preconditioners
- Algebraic construction based on the parallel distributed system matrix
- Extension-based coarse spaces

Team (active)

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

Overlapping domain decomposition

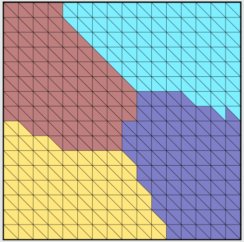
In FROSCH, the overlapping subdomains $\Omega_1', \ldots, \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



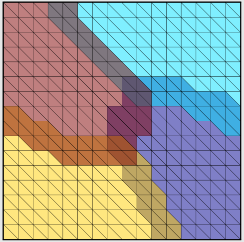
Nonoverlapping DD

Overlapping domain decomposition

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.



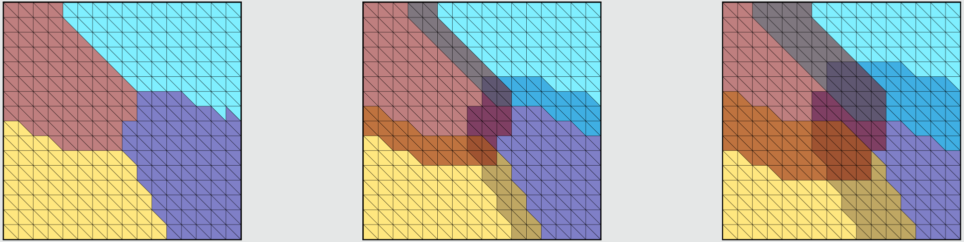
Nonoverlapping DD



Overlap $\delta = 1h$

Overlapping domain decomposition

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.

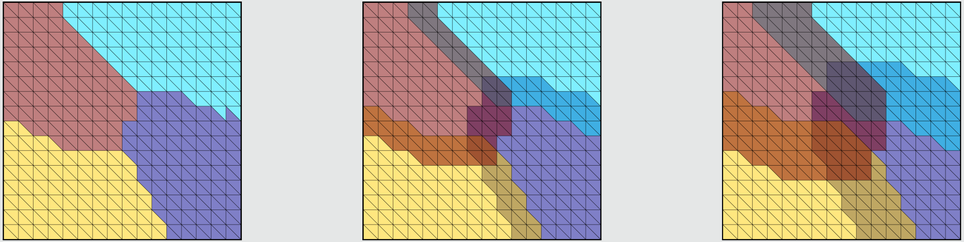


Nonoverlapping DD — Overlap $\delta = 1h$ — Overlap $\delta = 2h$

Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

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



Nonoverlapping DD — Overlap $\delta = 1h$ — Overlap $\delta = 2h$

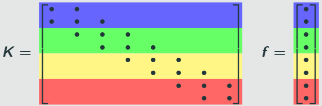
Computation of the overlapping matrices

The overlapping matrices

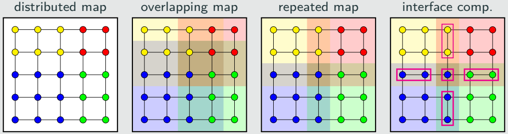
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

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

1. Identification interface components



Identification from **parallel distribution** of matrix:



2. Interface partition of unity (IPOU)

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

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

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**.

4. Extension into the interior

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

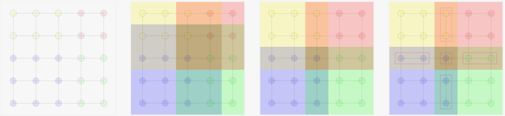
$$\Phi = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} -K_{II}^{-1} K_{\Gamma I}^T \Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix}.$$

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

Algorithmic Framework for FROSch Coarse Spaces

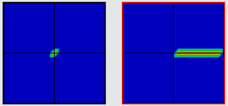
1. Identification interface components

Identification from **parallel distribution** of matrix:

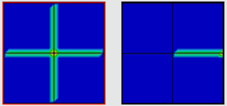


2. Interface partition of unity (IPOU)

vertex & edge functions

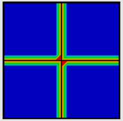


vertex functions



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

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



3. Interface basis

$\times$

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**.

4. Extension into the interior

edge basis function

vertex basis function

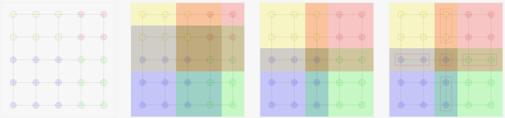
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

1. Identification interface components

Identification from **parallel distribution** of matrix:

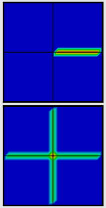


2. Interface partition of unity (IPOU)

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

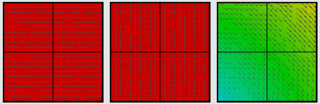
$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

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

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

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

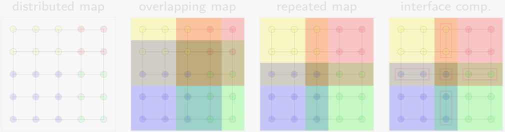
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$K =$ $f =$

Identification from **parallel distribution** of matrix:



2. Interface partition of unity (IPOU)

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

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

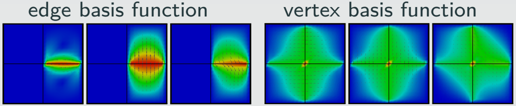
3. Interface basis

$\times$

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**.

4. Extension into the interior



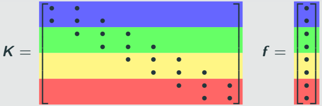
The values in the interior of the subdomains are computed via the **extension operator**:

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

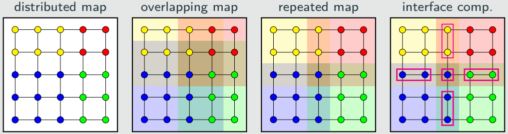
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for `FROSch` Coarse Spaces

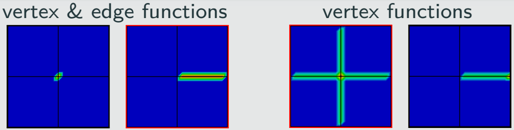
1. Identification interface components



Identification from **parallel distribution** of matrix:

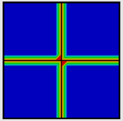


2. Interface partition of unity (IPOU)

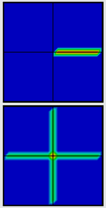


Based on the interface components, construct an **interface partition of unity**:

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

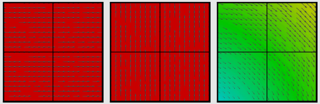


3. Interface basis



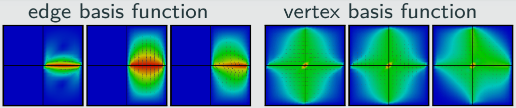
$\times$

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**.

4. Extension into the interior



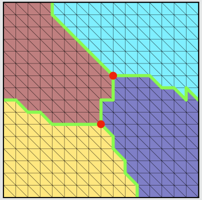
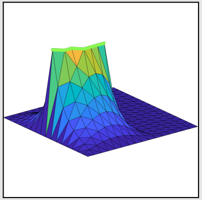
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

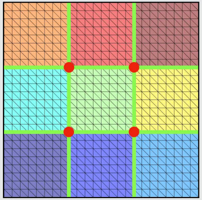
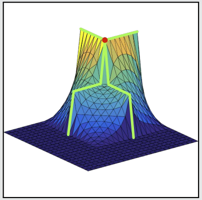
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

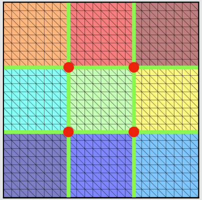
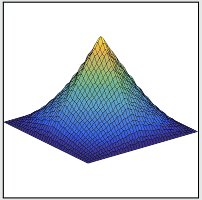
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

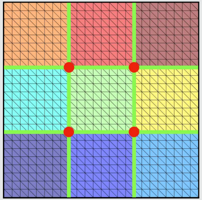
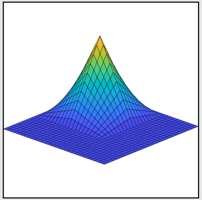
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)

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

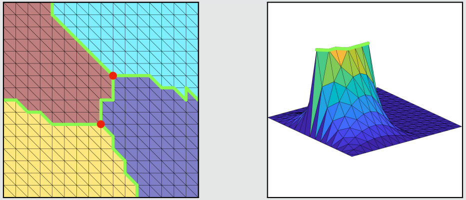
Q1 Lagrangian / piecewise bilinear

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

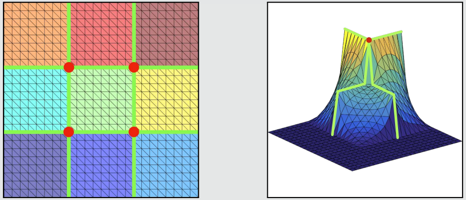
Examples of `FROSch` Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)



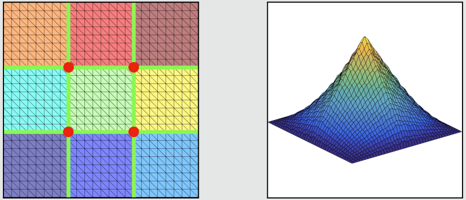
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)



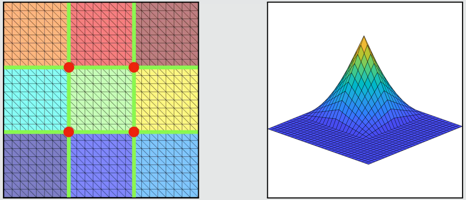
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

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



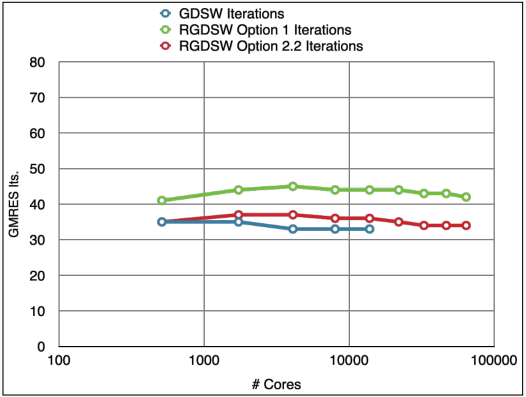
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

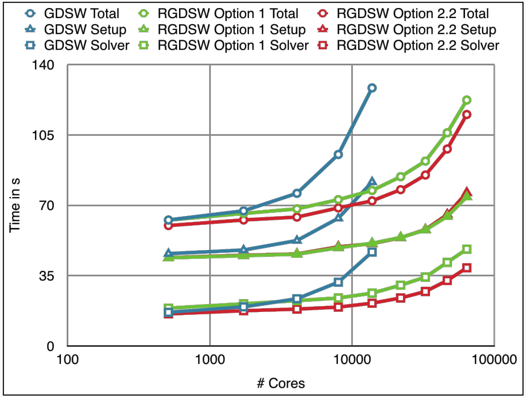
Weak Scalability up to $64\,\mathrm{k}$ MPI ranks / $1.7\,\mathrm{b}$ Unknowns (3D Poisson; Juqueen)

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)

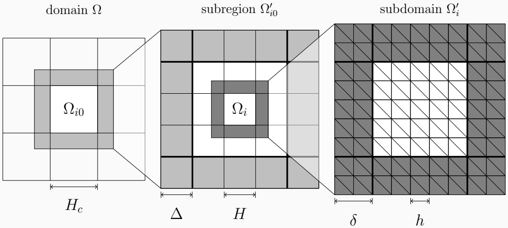
2 Monolithic Schwarz Preconditioners in FROSch

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

3 FROSch Preconditioners on GPUs

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

Multilevel Schwarz Preconditioners in FROSch



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**

$\rightarrow$ **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

---

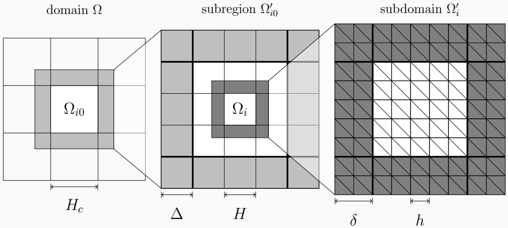
Function FROSch($\boldsymbol{K}$,$\boldsymbol{x}$,l):

    $\boldsymbol{x} = \Phi^\top \boldsymbol{x}$; /* 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: $\boldsymbol{M}_{\text{FROSch}}^{-1} = \Phi \boldsymbol{K}_0^{-1} \Phi^T \boldsymbol{K} + \sum_{i=1}^{N} \boldsymbol{R}_i^T \boldsymbol{K}_i^{-1} \boldsymbol{R}_i \boldsymbol{K}$

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**

$\rightarrow$ **Hierarchy of domain decompositions**

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

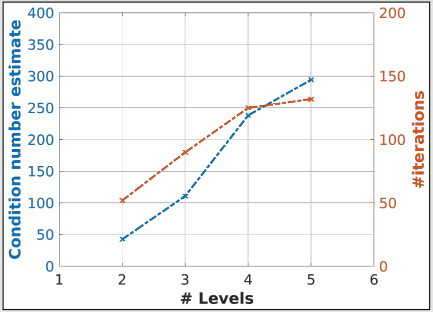
Influence of the inexact coarse solver

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**

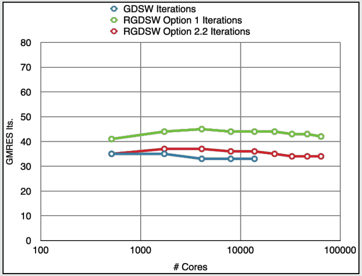
Let us discuss the **effect on the computing times next**.

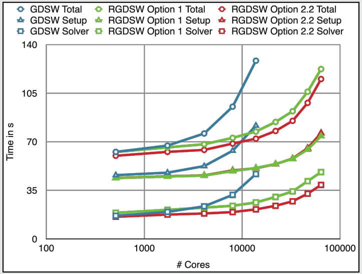


Weak Scalability up to $64\,\mathrm{k}$ MPI ranks / $1.7\,\mathrm{b}$ Unknowns (3D Poisson; Juqueen)

GDSW vs RGDSW (reduced dimension)

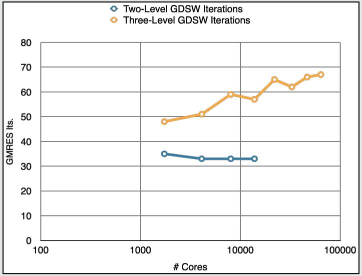
Heinlein, Klawonn, Rheinbach, Widlund (2019).

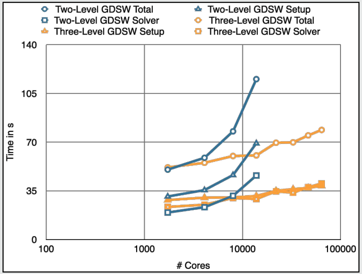




Two-level vs three-level GDSW

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

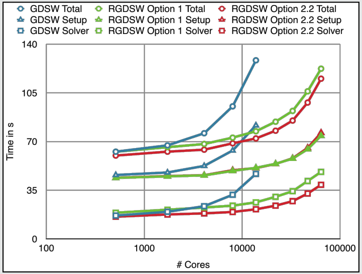




Weak Scalability up to $64\,\mathrm{k}$ MPI ranks / $1.7\,\mathrm{b}$ Unknowns (3D Poisson; Juqueen)

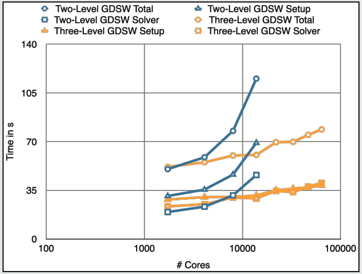
GDSW vs RGDSW (reduced dimension)

Heinlein, Klawonn, Rheinbach, Widlund (2019).



Two-level vs three-level GDSW

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



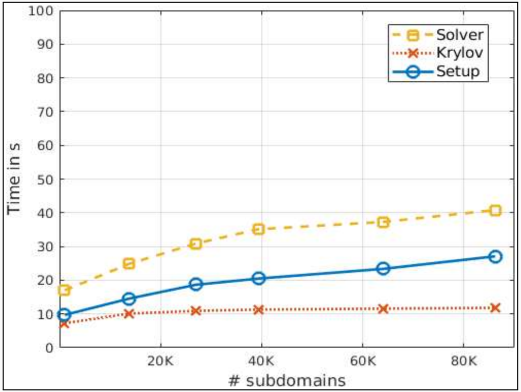
# subdomains (=#cores)		1 728	4 096	8 000	13 824	21 952	32 768	46 656	64 000
GDSW	Size of K_0	10 439	25 695	51 319	89 999	-	-	-	-
	Size of K_{00}	98	279	604	1 115	1 854	2 863	4 184	5 589
RGDSW	Size of K_0	1 331	3 375	6 859	12 167	19 683	29 791	42 875	59 319
	Size of K_{00}	8	27	64	125	216	343	512	729

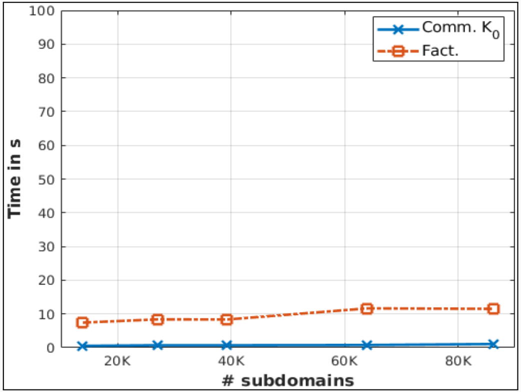
Weak Scalability of the Three-Level RGDSW Preconditioner – SuperMUC-NG

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)



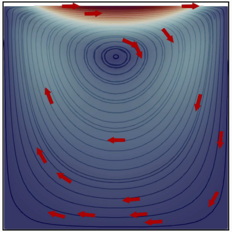


Cf. Heinlein, Rheinbach, Röver (2022); computations performed on SuperMUC-NG, LRZ, Germany.

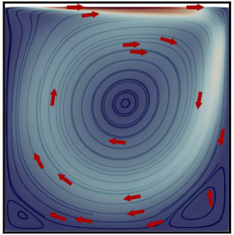
Monolithic Schwarz Preconditioners in FROSch

Consider the discrete saddle point problem

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



Stokes flow



Navier–Stokes flow

Monolithic GDSW preconditioner

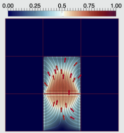
We construct a **monolithic GDSW preconditioner**

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

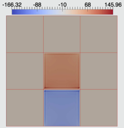
with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$, and

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

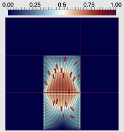
Using $\mathcal{A}$ to compute extensions: $\phi_I = -\mathcal{A}_{II}^{-1} \mathcal{A}_{I\Gamma} \phi_\Gamma$; cf. **Heinlein, Hochmuth, Klawonn (2019, 2020)**.



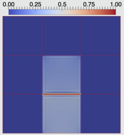
Φ_{u,u_0}



Φ_{p,u_0}



Φ_{u,p_0}



Φ_{p,p_0}

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

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

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

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

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:

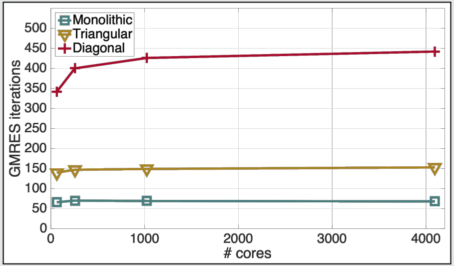
$$\mathcal{M}_{\text{D}}^{-1} = \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ 0 & \boldsymbol{S}^{-1} \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{M}_{\text{GDSW}}^{-1}(\boldsymbol{K}) & 0 \\ 0 & \boldsymbol{M}_{\text{OS-1}}^{-1}(\boldsymbol{M_p}) \end{bmatrix}$$

Block-triangular preconditioner:

$$\mathcal{M}_{\text{T}}^{-1} = \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\boldsymbol{S}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \boldsymbol{S}^{-1} \end{bmatrix}$$

$$\approx \begin{bmatrix} \boldsymbol{M}_{\text{GDSW}}^{-1}(\boldsymbol{K}) & 0 \\ -\boldsymbol{M}_{\text{OS-1}}^{-1}(\boldsymbol{M_p})\boldsymbol{B}\boldsymbol{M}_{\text{GDSW}}^{-1}(\boldsymbol{K}) & \boldsymbol{M}_{\text{OS-1}}^{-1}(\boldsymbol{M_p}) \end{bmatrix}$$

Monolithic vs. block prec. (Stokes)



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

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

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

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

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

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

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

SIMPLE block preconditioner

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

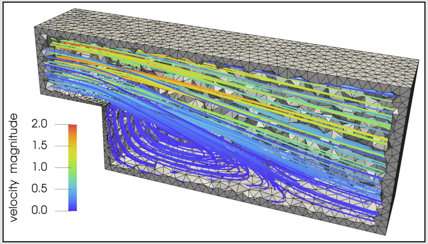
$$\mathcal{M}_{\text{SIMPLE}}^{-1} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}^{-1}\boldsymbol{B} \\ 0 & \alpha \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\hat{\boldsymbol{S}}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \hat{\boldsymbol{S}}^{-1} \end{bmatrix};$$

see Patankar and Spalding (1972). Here,

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

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

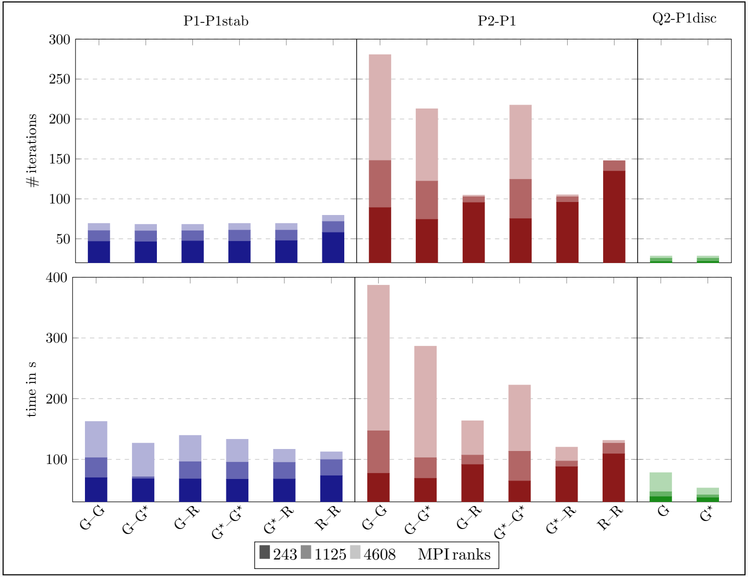
Monolithic vs. SIMPLE preconditioner



Steady-state Navier–Stokes equations

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

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

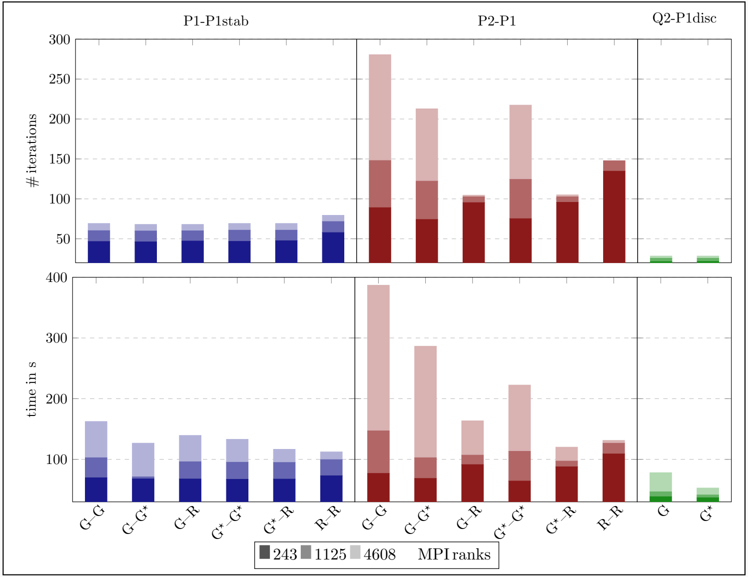


FROSch allows for the **flexible construction of extension-based coarse spaces** based on **various choices for the interface partition of unity (IPOU)**:
IPOUHarmonicCoarseOperator

Comparison of coarse spaces

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

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

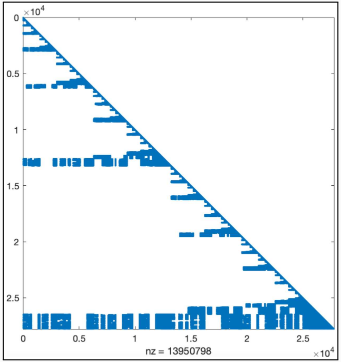
⇒ Generally **good performance** for **stabilized or discontinuous pressure discretizations**. Otherwise, performance depends on the **combination of velocity and pressure coarse spaces**.

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**

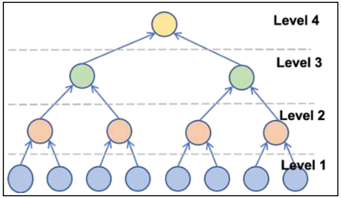
- Factorization using a **sparse direct solver** typically leads to triangular matrices with **dense blocks** called **supernodes**
- In **supernodal triangular solvers**, rows/columns with a similar sparsity pattern are merged into a supernodal block, and the **solve is then performed block-wise**
-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



Lower-triangular matrix – SuperLU with METIS nested dissection ordering

Parallel supernode-based triangular solver:

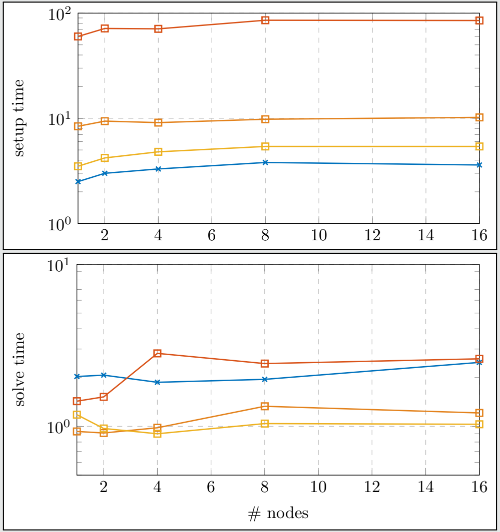
1. **Supernode-based level-set scheduling**, where **all leaf-supernodes within one level are solved in parallel** (batched kernels for hierarchical parallelism)
2. **Partitioned inverse** of the submatrix associated with each level: **SpTRSV is transformed into a sequence of SpMVs**



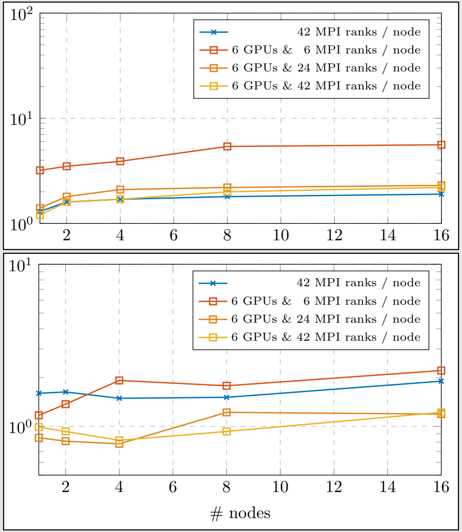
See **Yamazaki, Rajamanickam, and Ellingwood (2020)** for more details.

Three-Dimensional Linear Elasticity – Weak Scalability

SuperLU – weak scaling

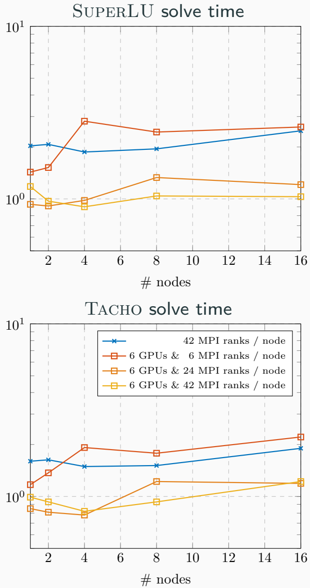


Tacho – weak scaling



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

Yamazaki, Heinlein, Rajamanickam (2023)



# 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/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/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**×

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**×

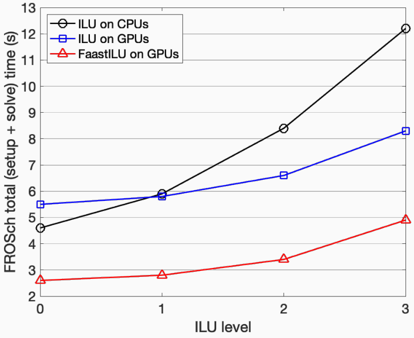
Computations on Summit (OLCF): 42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

Yamazaki, Heinlein, Rajamanickam (2023)

ILU variants

- KokkosKernels ILU (KK)
- 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	2.8
	Fast	**1.5**	**2.2**	**2.3**	**2.5**	**2.8**
speedup		**1.3**×	**1.0**×	**1.0**×	**1.0**×	**0.9**×
solve						
CPU		**3.60 (112)**	**7.26 (84)**	**6.93 (78)**	**6.41 (75)**	**4.1 (109)**
GPU	KK	4.3 (119)	3.9 (110)	4.8 (105)	4.3 (97)	4.9 (109)
	Fast	**1.2 (154)**	**1.0 (133)**	**1.1 (130)**	**1.3 (117)**	**1.6 (131)**
speedup		**3.3**×	**3.8**×	**3.4**×	**2.5**×	**2.6**×

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

Yamazaki, Heinlein, Rajamanickam (2023)

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 capabilities

  have 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, 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!