Efficient Schwarz Preconditioning Techniques for Nonlinear Problems Using FROSch

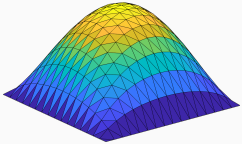
Alexander Heinlein[1]

10th International Congress on Industrial and Applied Mathematics (ICIAM 2023 Tokyo)
Waseda University, Tokyo, Japan, August 20-25, 2023

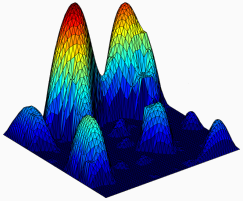
[1]Delft University of Technology

Based on joint work with Axel Klawonn and Lea Saßmannshausen (University of Cologne) and Mauro Perego, 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

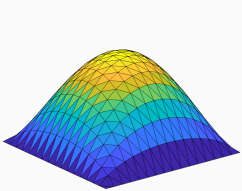
For fine meshes, solving the system using a direct solver is not feasible due to **superlinear complexity and memory cost**.

Iterative solvers

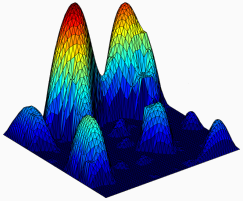
Iterative solvers are efficient for solving sparse linear systems of equations, however, the **convergence rate generally depends on the condition number** $\kappa(\boldsymbol{A})$. It deteriorates, e.g., for

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

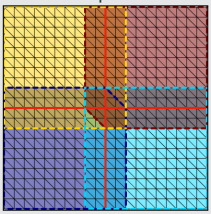
$\Rightarrow$ We introduce a preconditioner $\boldsymbol{M}^{-1} \approx \boldsymbol{A}^{-1}$ to improve the condition number:

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

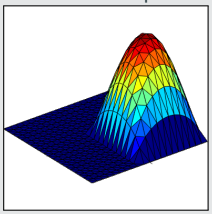
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$



Solution of local problem



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

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

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

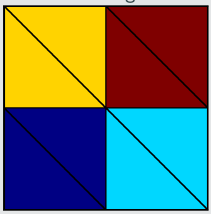
Condition number estimate:

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

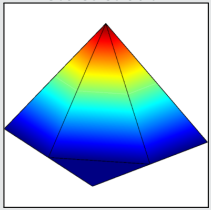
with subdomain size H and overlap width δ.

Lagrangian coarse space

Coarse triangulation



Coarse solution



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

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

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

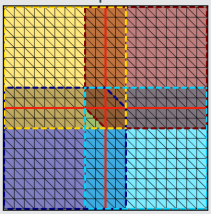
Condition number estimate:

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

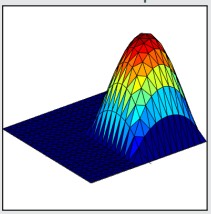
Two-Level Schwarz Preconditioners

One-level Schwarz preconditioner

Overlap $\delta = 1h$

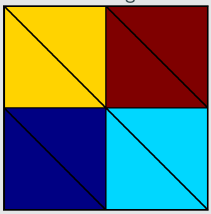


Solution of local problem

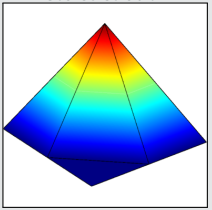


Lagrangian coarse space

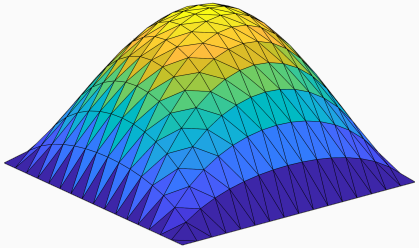
Coarse triangulation

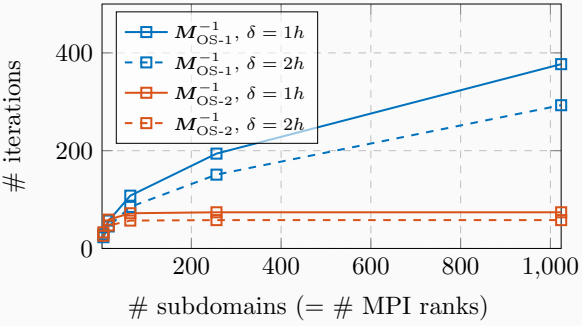


Coarse solution

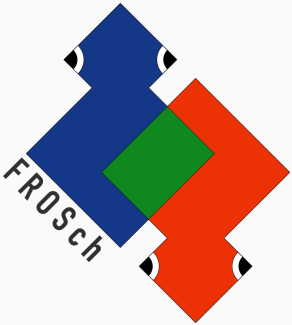


Diffusion model problem in two dimensions,
$H/h = 100$





FROSch (Fast and Robust Overlapping Schwarz) Framework in Trilinos



Software

- Object-oriented C++ domain decomposition solver framework with MPI-based distributed memory parallelization
- Part of Trilinos with support for both parallel linear algebra packages Epetra and Tpetra
- Node-level parallelization and performance portability on CPU and GPU architectures through Kokkos and KokkosKernels
- Accessible through unified Trilinos solver interface Stratimikos

Methodology

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

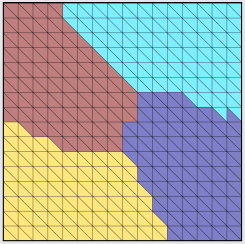
Team (active)

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

Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

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

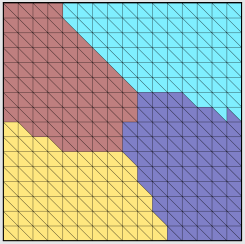


Nonoverlapping DD

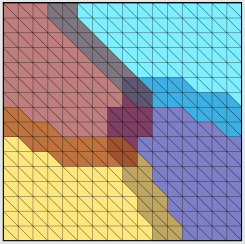
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

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



Nonoverlapping DD

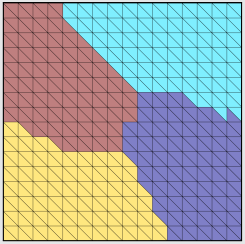


Overlap $\delta = 1h$

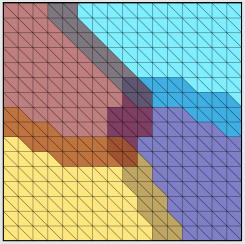
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

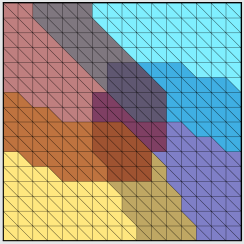
In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$

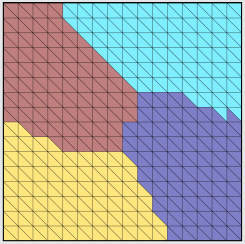


Overlap $\delta = 2h$

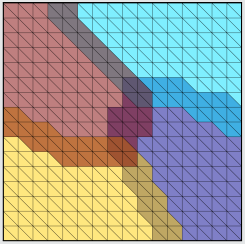
Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

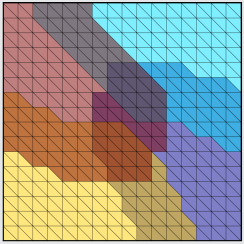
In FROSch, the overlapping subdomains $\Omega_1', ..., \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$

Computation of the overlapping matrices

The overlapping matrices

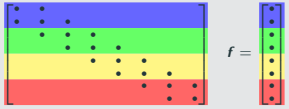
$$\boldsymbol{K}_i = \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$$

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

Algorithmic Framework for FROSch Coarse Spaces

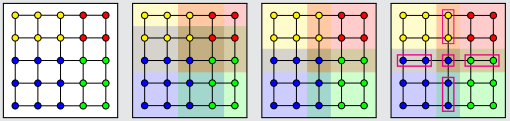
1. Identification interface components

$K =$ $f =$



Identification from **parallel distribution** of matrix:

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



2. Interface partition of unity (IPOU)

vertex & edge functions | vertex functions

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

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

3. Interface basis

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

×

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

4. Extension into the interior

edge basis function | vertex basis function

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

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

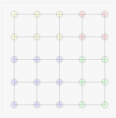
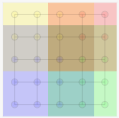
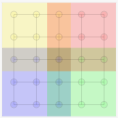
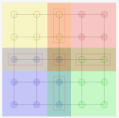
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

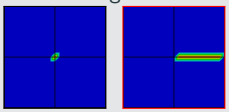
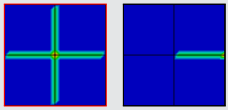
1. Identification interface components

$K =$ $f =$

Identification from **parallel distribution** of matrix:

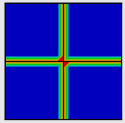
distributed map 
overlapping map 
repeated map 
interface comp. 

2. Interface partition of unity (IPOU)

vertex & edge functions 
vertex functions 

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

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



3. Interface basis

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

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

4. Extension into the interior

edge basis function
vertex basis function

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

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

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

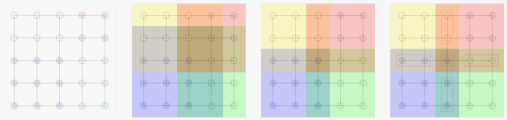
Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$K =$ $f =$

Identification from **parallel distribution** of matrix:

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



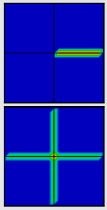
2. Interface partition of unity (IPOU)

vertex & edge functions | vertex functions

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

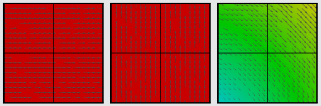
$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

3. Interface basis



$\times$

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**)

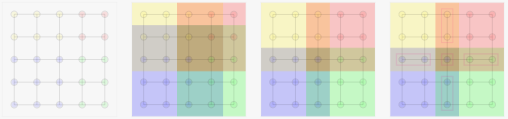
Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

$K =$ $f =$

Identification from **parallel distribution** of matrix:

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



2. Interface partition of unity (IPOU)

vertex & edge functions | vertex functions

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

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

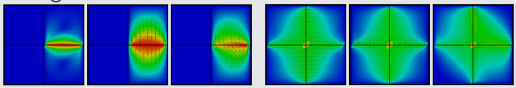
3. Interface basis

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

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

4. Extension into the interior

edge basis function | vertex basis function



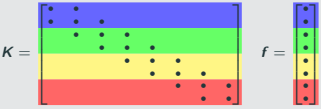
The values in the interior of the subdomains are computed via the **extension operator**:

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

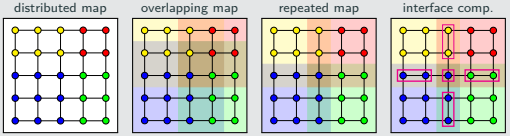
(For elliptic problems: **energy-minimizing extension**)

Algorithmic Framework for FROSch Coarse Spaces

1. Identification interface components

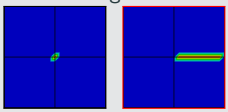


Identification from **parallel distribution** of matrix:

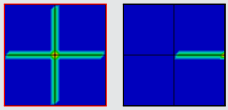


2. Interface partition of unity (IPOU)

vertex & edge functions

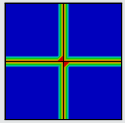


vertex functions

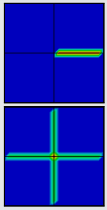


Based on the interface components, construct an **interface partition of unity**:

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

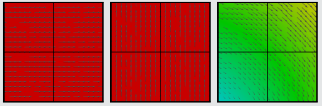


3. Interface basis



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

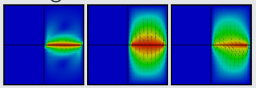
×



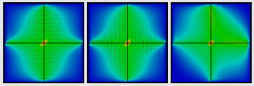
The interface values of the basis of the coarse space is obtained by **multiplication with the null space**.

4. Extension into the interior

edge basis function



vertex basis function



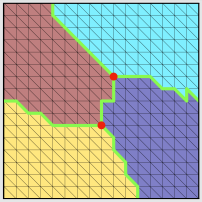
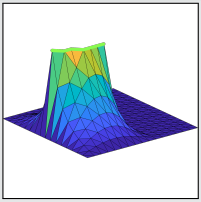
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

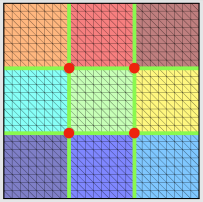
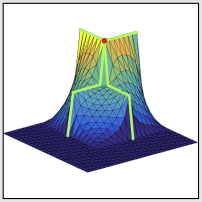
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

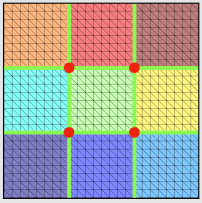
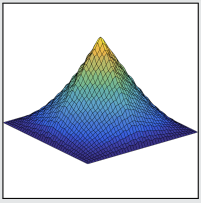
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

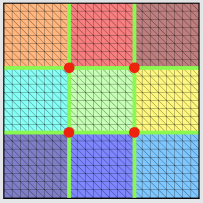
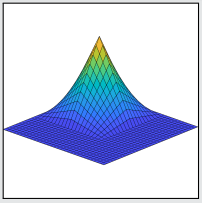
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)

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

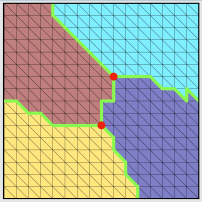
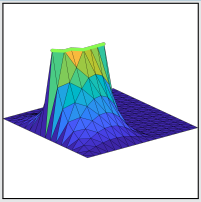
Q1 Lagrangian / piecewise bilinear

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

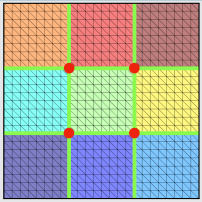
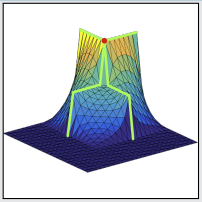
Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)

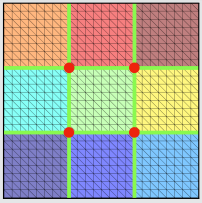
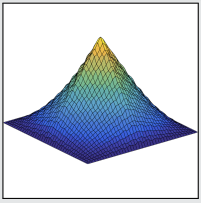
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)

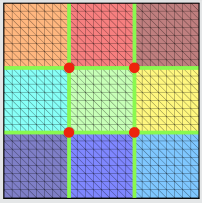
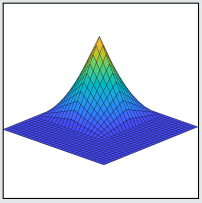
- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

MsFEM (Multiscale Finite Element Method)

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

Q1 Lagrangian / piecewise bilinear

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

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

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

Monolithic GDSW preconditioner

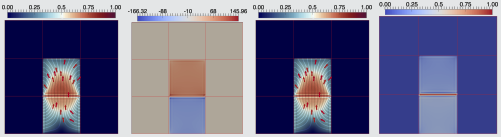
We construct a **monolithic GDSW preconditioner**

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

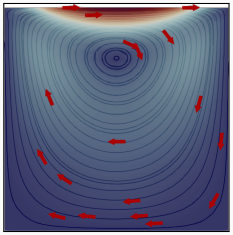
with block matrices $\mathcal{A}_0 = \phi^\top \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^\top$, and

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

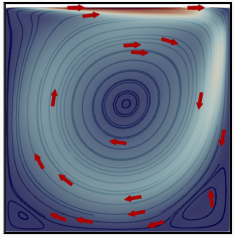
Using $\mathcal{A}$ to compute extensions: $\phi_I = -\mathcal{A}_{II}^{-1} \mathcal{A}_{I\Gamma} \phi_\Gamma$; cf. Heinlein, Hochmuth, Klawonn (2019, 2020).



Φ_{u,u_0} Φ_{p,u_0} Φ_{u,p_0} Φ_{p,p_0}



Stokes flow



Navier–Stokes flow

Related work:

- Original work on monolithic Schwarz preconditioners: Klawonn and Pavarino (1998, 2000)
- Other publications on monolithic Schwarz preconditioners: e.g., Hwang and Cai (2006), Barker and Cai (2010), Wu and Cai (2014), and the presentation Dohrmann (2010) at the *Workshop on Adaptive Finite Elements and Domain Decomposition Methods* in Milan.

Monolithic (R)GDSW Preconditioners for CFD Simulations

Consider the discrete saddle point problem

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

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

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

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

Block diagonal & triangular preconditioners

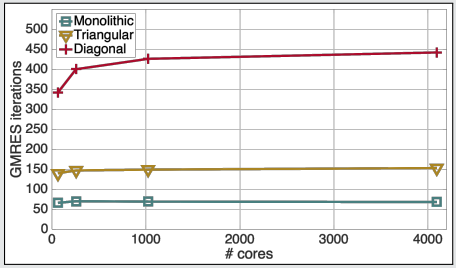
Block-diagonal preconditioner:

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

Block-triangular preconditioner:

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

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

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

Monolithic GDSW preconditioner

We construct a **monolithic GDSW preconditioner**

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

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

SIMPLE block preconditioner

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

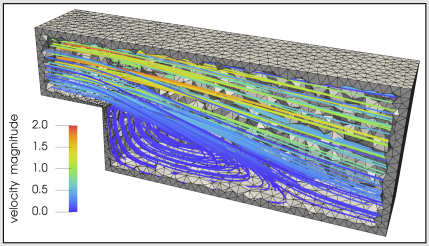
$$M_{\text{SIMPLE}}^{-1} = \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{D}^{-1}\boldsymbol{B} \\ 0 & \alpha \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{K}^{-1} & 0 \\ -\hat{\boldsymbol{S}}^{-1}\boldsymbol{B}\boldsymbol{K}^{-1} & \hat{\boldsymbol{S}}^{-1} \end{bmatrix};$$

see Patankar and Spalding (1972). Here,

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

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

Monolithic vs. SIMPLE preconditioner

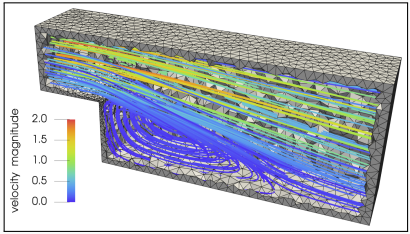


Steady-state Navier–Stokes equations

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

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

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

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

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	39.6 s	50.9 s	49.8 s
	solve	**38.2 s**	**58.7 s**	**71.7 s**
	total	**77.8 s**	**109.8 s**	**121.5 s**
SIMPLE RGDSW (Teko & FROSch)	# its.	168.4(5)	196.8(5)	200.0(5)
	setup	**21.2 s**	**32.2 s**	**26.9 s**
	solve	106.2 s	156.0 s	175.0 s
	total	127.4 s	188.2 s	201.9 s

P2–P1, $H/h = 9$

prec.	# MPI ranks	243	1 125	4 608
Monolithic RGDSW (FROSch)	# its.	**84.2(6)**	**100.4(5)**	**108.6(5)**
	setup	44.2 s	48.5 s	49.7 s
	solve	**50.0 s**	**63.9 s**	**88.0 s**
	total	**94.2 s**	**112.4 s**	**137.7 s**
SIMPLE RGDSW (Teko & FROSch)	# its.	157.5(6)	161.8(5)	169.8(5)
	setup	**26.8 s**	**31.7 s**	**28.5 s**
	solve	84.8 s	90.4 s	111.5 s
	total	111.6 s	122.1 s	140.0 s

Heinlein, Klawonn, and Saßmannshausen (in preparation)

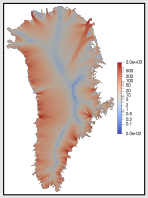
FROSch Preconditioners for Land Ice Simulations

Stationary velocity problem

We use a **first-order (or Blatter-Pattyn) approximation of the Stokes equations**

$$\begin{cases} -\nabla\cdot(2\mu\,\dot\epsilon_1) &= -\rho_i\,|\boldsymbol{g}|\,\partial_x s, \\ -\nabla\cdot(2\mu\,\dot\epsilon_2) &= -\rho_i\,|\boldsymbol{g}|\,\partial_y s, \end{cases}$$

with ice density ρ_i, ice surface elevation s, gravity acceleration $\boldsymbol{g}$, and strain rates $\dot\epsilon_1$ and $\dot\epsilon_2$; cf. Blatter (1995) and Pattyn (2003).



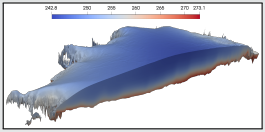
Ice viscosity modeled by **Glen's law**: $\mu = \frac{1}{2}A(T)^{-\frac{1}{n}}\,\dot\epsilon_e^{\frac{1-n}{n}}$.

Stationary temperature problem

The **enthalpy equation** reads

$$\nabla\cdot\boldsymbol{q}(h) + \boldsymbol{u}\cdot\nabla h = 4\mu\,\epsilon_e^2$$

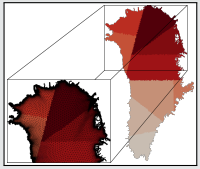
with the **enthalpy flux**



$$\boldsymbol{q}(h) = \begin{cases} \frac{k}{\rho_i c_i}\nabla h, & \text{for cold ice } (h \le h_m), \\ \frac{k}{\rho_i c_i}\nabla h_m + \rho_w L\boldsymbol{j}(h), & \text{for temperate ice.} \end{cases}$$

Set of complex boundary conditions: Dirichlet, Neumann, Robin, and Stefan boundary and coupling conditions.

FROSch preconditioners



We compute the **nonoverlapping domain decomposition** based on the **surface mesh**.

For the coupled problem, we construct a **monolithic two-level (R)GDSW preconditioner** (Heinlein, Hochmuth, Klawonn (2019, 2020))

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

where the linearized system is of the form

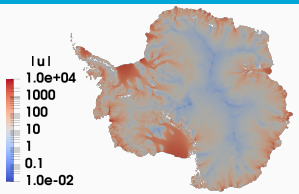
$$\mathcal{A}x := \begin{bmatrix} A_u & C_{uT} \\ C_{Tu} & A_T \end{bmatrix}\begin{bmatrix} x_u \\ x_T \end{bmatrix} = \begin{bmatrix} \tilde r_u \\ \tilde r_T \end{bmatrix} =: r.$$

For single-physics problems, we employ a **standard (R)GDSW preconditioner**.

Antarctica Velocity Problem – Reuse Strategies (Strong Scaling)

We employ different **reuse strategies** to **reduce the setup costs** of the two-level preconditioner

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



reuse	restriction operators + symbolic fact. (1st level)			+ coarse basis + symbolic fact. (2nd level)			+ coarse matrix		
MPI ranks	avg. its (nl its)	avg. setup	avg. solve	avg. its (nl its)	avg. setup	avg. solve	avg. its (nl its)	avg. setup	avg. solve
512	**41.9** (11)	25.10 s	**12.29 s**	42.6 (11)	14.99 s	12.50 s	46.7 (11)	**14.94 s**	13.81 s
1 024	**43.3** (11)	9.18 s	**5.85 s**	44.5 (11)	**5.65 s**	6.08 s	49.2 (11)	5.75 s	6.78 s
2 048	**41.4** (11)	4.15 s	**2.63 s**	42.7 (11)	3.11 s	2.79 s	47.7 (11)	**2.92 s**	3.10 s
4 096	**41.2** (11)	1.66 s	**1.49 s**	42.5 (11)	1.07 s	1.54 s	48.9 (11)	**0.95 s**	1.75 s
8 192	**40.2** (11)	1.26 s	**1.06 s**	42.0 (11)	1.20 s	1.16 s	50.1 (11)	**0.63 s**	1.35 s

Problem: Velocity **Mesh:** Antarctica, 4 km hor. resolution, 20 vert. layers **Size:** 35.3 m degrees of freedom (P1 FE) **Coarse space:** RGDSW

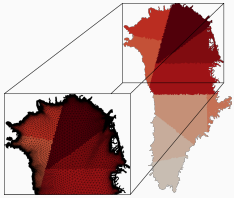
Cf. **Heinlein, Perego, Rajamanickam (2022)**

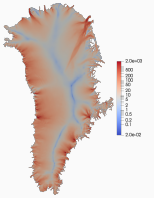
Greenland Coupled Problem – Coarse Spaces

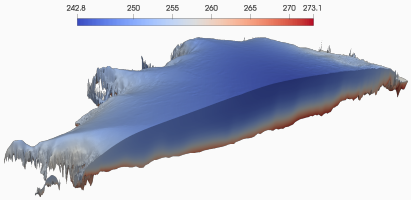
fully coupled extensions							
		no reuse			reuse **coarse basis**		
MPI ranks	dim V_0	avg. its (nl its)	avg. setup	avg. solve	avg. its (nl its)	avg. setup	avg. solve
256	1 400	100.1 (27)	4.10 s	6.40 s	**18.5** (70)	**2.28 s**	**1.07 s**
512	2 852	129.1 (28)	1.88 s	4.20 s	**24.6** (38)	**1.04 s**	**0.70 s**
1 024	6 036	191.2 (65)	1.21 s	4.76 s	**34.2** (32)	**0.66 s**	**0.70 s**
2 048	12 368	237.4 (30)	0.96 s	4.06 s	**37.3** (30)	**0.60 s**	**0.58 s**
decoupled extensions							
		no reuse			reuse **coarse basis**		
MPI ranks	dim V_0	avg. its (nl its)	avg. setup	avg. solve	avg. its (nl its)	avg. setup	avg. solve
256	1 400	23.6 (29)	3.90 s	1.32 s	**21.5** (34)	**2.23 s**	**1.18 s**
512	2 852	27.5 (30)	1.83 s	0.78 s	**26.4** (33)	**1.13 s**	**0.78 s**
1 024	6 036	30.1 (29)	1.19 s	**0.60 s**	**28.6** (43)	**0.66 s**	0.61 s
2 048	12 368	36.4 (30)	0.69 s	0.56 s	**31.2** (50)	**0.57 s**	**0.55 s**

Problem: Coupled **Mesh:** Greenland 3-30 km hor. resolution 20 vert. layers **Size:** 7.5 m degrees of freedom (P1 FE) **Coarse space:** RGDSW

Greenland Coupled Problem – Large Problem







	decoupled (no reuse)			**fully coupled** (reuse coarse basis)			**decoupled** (reuse 1st level symb. fact. + coarse basis)		
MPI ranks	avg. its. (nl its)	avg. setup	avg. solve	avg. its (nl its)	avg. setup	avg. solve	avg. its (nl its)	avg. setup	avg. solve
512	**41.3** (36)	18.78 s	**4.99** s	45.3 (32)	11.84 s	5.35 s	45.0 (35)	**10.53** s	5.36 s
1 024	53.0 (29)	8.68 s	4.22 s	**47.8** (37)	5.36 s	**3.82 s**	54.3 (32)	**4.59 s**	4.31 s
2 048	62.2 (86)	4.47 s	4.23 s	66.7 (38)	2.81 s	4.53 s	**59.1** (38)	**2.32 s**	**3.99 s**
4 096	**68.9** (40)	2.52 s	**2.86 s**	79.1 (36)	1.61 s	3.30 s	78.7 (38)	**1.37 s**	3.30 s

Problem: Coupled **Mesh:** Greenland, 1-10 km hor. resolution, 20 vert. layers **Size:** 68.6 m degrees of freedom (P1 FE) **Coarse space:** RGDSW

Cf. **Heinlein, Perego, Rajamanickam (2022)**

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

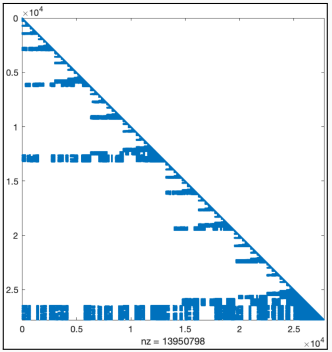
The sparse triangular solver is an **important kernel** in many codes (including FROSch) but is **challenging to parallelize**

- 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**

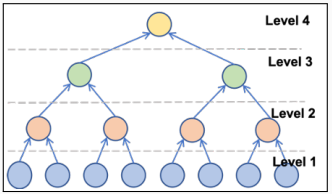
Parallel supernode-based triangular solver:

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**

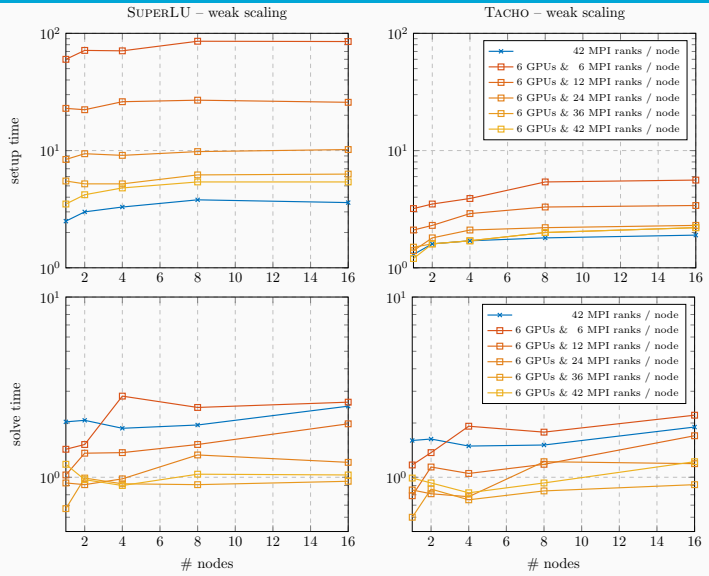
See **Yamazaki, Rajamanickam, and Ellingwood (2020)** for more details.



Lower-triangular matrix – SuperLU with METIS nested dissection ordering



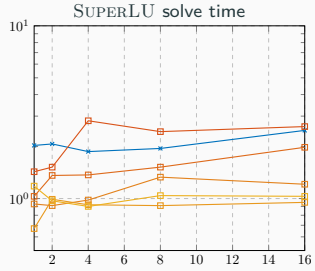
Three-Dimensional Linear Elasticity – Weak Scalability



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



# 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)
2	1.03 (46)	1.36 (65)	1.37 (60)	1.52 (65)	1.98 (86)
4	0.93 (59)	0.91 (53)	0.98 (59)	1.33 (77)	1.21 (66)
6	0.67 (46)	0.99 (65)	0.92 (57)	0.91 (57)	0.95 (57)
7	**1.03 (75)**	**1.04 (69)**	**0.90 (61)**	**0.97 (58)**	**1.18 (69)**
speedup	**2.0**×	**2.0**×	**2.1**×	**2.0**×	**2.1**×

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

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

Yamazaki, Heinlein, Rajamanickam (2023)

Three-Dimensional Linear Elasticity – ILU Subdomain Solver

ILU level		0	1	2	3
setup					
CPU	No	**1.5**	**1.9**	**3.0**	**4.8**
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**×

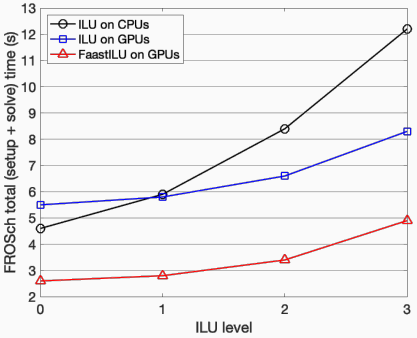
Computations on Summit (OLCF): 42 IBM Power9 CPU cores and 6 NVIDIA V100 GPUs per node.

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)

Thank you for your attention!

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** including **nonlinear multi-physics problems**
- For nonlinear problems,
  - **the reuse of components of the preconditioner** and
  - **the speedup of the solver phase (e.g., using GPUs)**

  can significantly help to improve the solver performance.

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!

→ Talk by **Ichitaro Yamazaki** on Tuesday (00911 (2/2)): **Related nonlinear Schwarz methods**
→ Talk by **Martin Lanser** on Thursday (01054 (2/3)): **FROSch on GPUs**
→ Talk by **Friederike Röver** on Friday (01054 (3/3)): **FROSch for chemo-mechanics problems**