Exercise Session 2

Parallel Preconditioning With FROSch

Alexander Heinlein

November 25, 2021

TU Delft

Schwarz domain decomposition preconditioners in `FROSch`

Domain Decomposition Methods in Trilinos

By Sandia National Laboratories

- Teko: **Block preconditioners** for multi-physics problems
- Ifpack/Ifpack2: **One-level overlapping Schwarz preconditioners**
  → **Algebraic** but **not scalable**
- ShyLU/BDDC: **BDDC** (Balancing Domain Decomposition by Constraints) **preconditioner**
  → **Scalable** but **less algebraic**

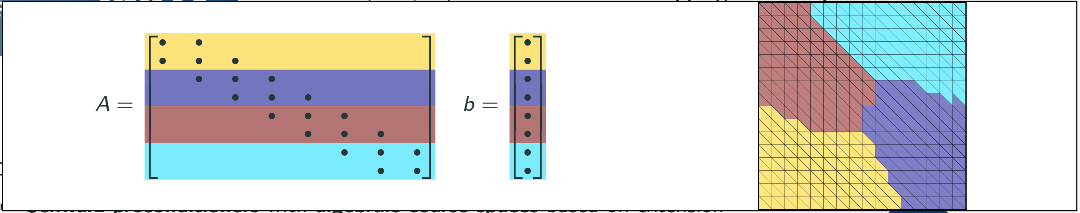
FROSch (**F**ast and **R**obust **O**verlapping **Sch**warz)

- **Schwarz preconditioners** with **algebraic coarse spaces** based on extension operators, e.g., **GDSW** (Generalized–Dryja–Smith–Widlund) coarse spaces
  → **Algebraic** and **scalable**
- Part of the package ShyLU:
  (Joint work with the Scalable Algorithms group of the **Sandia National Laboratories (SNL)**, Albuquerque, USA)
- Implementation based on Xpetra
  → Can be used with Epetra and Tpetra (linear algebra packages)
  Extension to current architectures, e.g., **GPUs**, *using the* Kokkos *programming model*

Easy access to FROSch through unified Trilinos solver interface Thyra.

Domain Decomposition Methods in Trilinos

- Teko: **Block preconditioners** for multi-physics problems
- Ifpack/Ifpack2: **One-level overlapping Schwarz preconditioners**

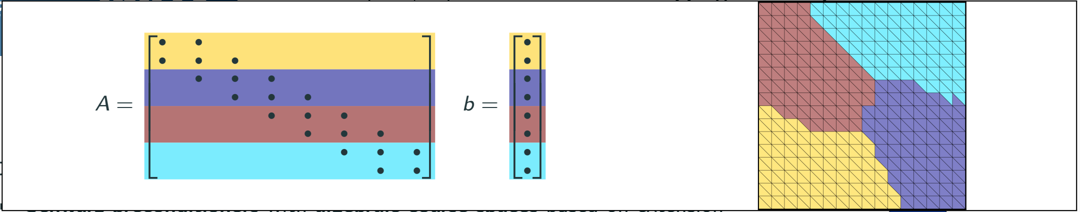


FR

operators, e.g., **GDSW** (Generalized–Dryja–Smith–Widlund) coarse spaces
  → **Algebraic** and **scalable**
- Part of the package ShyLU:
  (Joint work with the Scalable Algorithms group of the **Sandia National Laboratories (SNL)**, Albuquerque, USA)
- Implementation based on Xpetra
  → Can be used with Epetra and Tpetra (linear algebra packages)
  Extension to current architectures, e.g., **GPUs**, *using the* Kokkos *programming model*

Easy access to FROSch through unified Trilinos solver interface Thyra.

- Teko: **Block preconditioners** for multi-physics problems
- Ifpack/Ifpack2: **One-level overlapping Schwarz preconditioners**

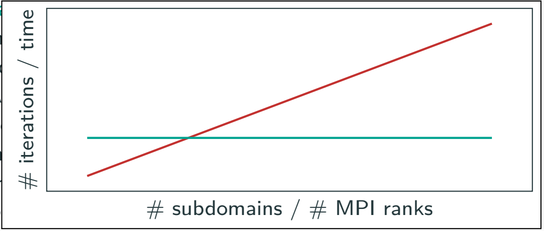


FRO

operators, e.g., **GDSW** (Generalized–Dryja–Smith–Widlund) coarse spaces

→ Algebraic and sca

- Part of the package Sh
(Joint work with the S
Laboratories (SNL), A
- Implementation based
→ Can be used with
Extension to curr
programming mo



Easy access to FROSch through unified Trilinos solver interface Thyra.

Domain Decomposition Methods in Trilinos

By Sandia National Laboratories

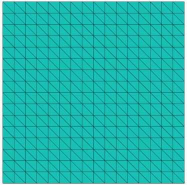
- Teko: **Block preconditioners** for multi-physics problems
- Ifpack/Ifpack2: **One-level overlapping Schwarz preconditioners**
  → **Algebraic** but **not scalable**
- ShyLU/BDDC: **BDDC** (Balancing Domain Decomposition by Constraints) **preconditioner**
  → **Scalable** but **less algebraic**

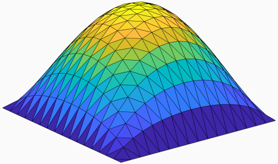
FROSch (**F**ast and **R**obust **O**verlapping **Sch**warz)

- **Schwarz preconditioners** with **algebraic coarse spaces** based on extension operators, e.g., **GDSW** (Generalized–Dryja–Smith–Widlund) coarse spaces
  → **Algebraic** and **scalable**
- Part of the package ShyLU:
  (Joint work with the Scalable Algorithms group of the **Sandia National Laboratories (SNL)**, Albuquerque, USA)
- Implementation based on Xpetra
  → Can be used with Epetra and Tpetra (linear algebra packages)
  Extension to current architectures, e.g., **GPUs**, *using the* Kokkos *programming model*

Easy access to FROSch through unified Trilinos solver interface Thyra.

Simple Model Problem & Overlapping Domain Decomposition





Consider a **Poisson model problem** on $[0,1]^2$:

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

Discretize (e.g., using finite elements)

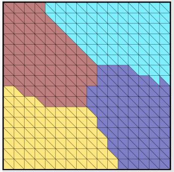
$$Kx = b.$$

$\Rightarrow$ Construct a **parallel scalable preconditioner** M^{-1} using **overlapping Schwarz domain decomposition methods**.

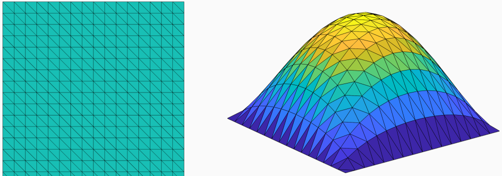
Overlapping domain decomposition

Overlapping Schwarz methods are based on **overlapping decompositions** of the computational domain Ω.

Overlapping subdomains $\Omega_1', \dots, \Omega_N'$ can be constructed by **recursively adding layers of elements** to nonoverlapping subdomains $\Omega_1, \dots, \Omega_N$.



Nonoverlap. DD



Consider a **Poisson model problem** on $[0,1]^2$:

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

Discretize (e.g., using finite elements)

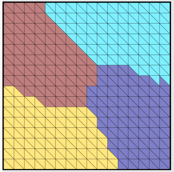
$$Kx = b.$$

$\Rightarrow$ Construct a **parallel scalable preconditioner** M^{-1} using **overlapping Schwarz domain decomposition methods**.

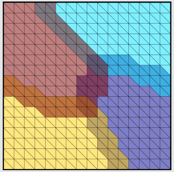
Overlapping domain decomposition

Overlapping Schwarz methods are based on **overlapping decompositions** of the computational domain Ω.

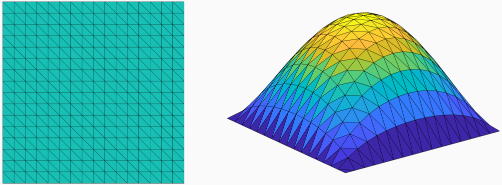
Overlapping subdomains $\Omega_1', ..., \Omega_N'$ can be constructed by **recursively adding layers of elements** to nonoverlapping subdomains $\Omega_1, ..., \Omega_N$.



Nonoverlap. DD



Overlap $\delta = 1h$



Consider a **Poisson model problem** on $[0,1]^2$:

$$
\begin{aligned}
-\Delta u &= f \quad \text{in } \Omega, \\
u &= 0 \quad \text{on } \partial\Omega.
\end{aligned}
$$

Discretize (e.g., using finite elements)

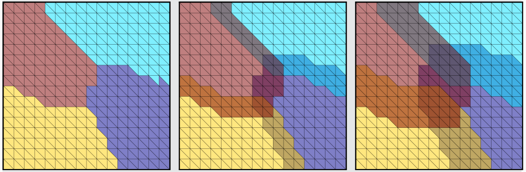
$$Kx = b.$$

$\Rightarrow$ Construct a **parallel scalable preconditioner** M^{-1} using **overlapping Schwarz domain decomposition methods**.

Overlapping domain decomposition

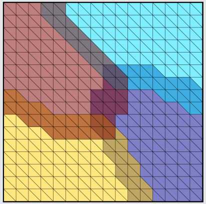
Overlapping Schwarz methods are based on **overlapping decompositions** of the computational domain Ω.

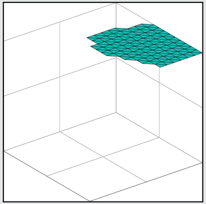
Overlapping subdomains $\Omega_1', \ldots, \Omega_N'$ can be constructed by **recursively adding layers of elements** to nonoverlapping subdomains $\Omega_1, \ldots, \Omega_N$.



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

One-Level Schwarz preconditioner





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

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

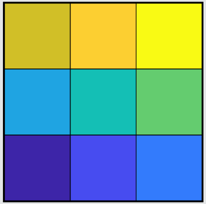
where R_i and R_i^T are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^T$.
$\rightarrow$ algebraic

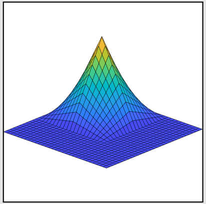
Condition number estimate:

$$\kappa \left(P_{\mathrm{OS-1}} \right) \leq C \left(1 + \frac{1}{H\delta} \right)$$

with subdomain size H and the width of the overlap δ.

Adding a Lagrangian coarse space





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

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

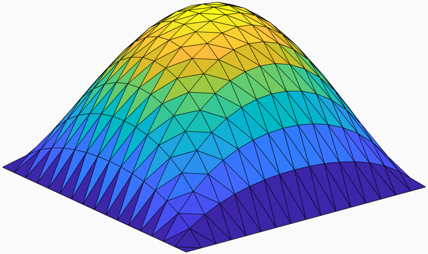
where Φ contains the coarse basis functions and $K_0 := \Phi^T K \Phi$; cf., e.g., **Toselli, Widlund (2005)**.

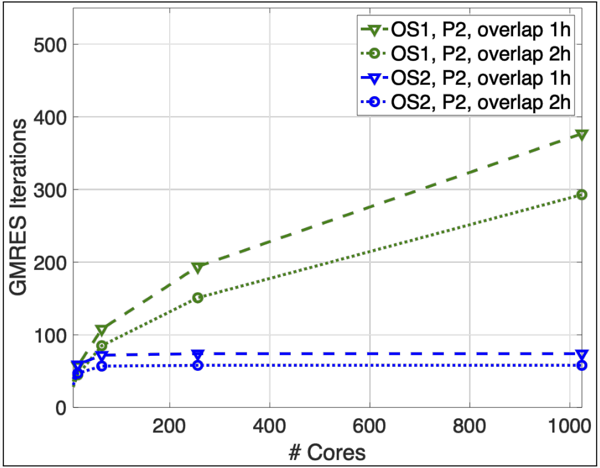
A Lagrangian coarse basis requires a coarse triangulation (geometric information) $\rightarrow$ not algebraic

$$\Rightarrow \kappa \left(P_{\mathrm{OS-2}} \right) \leq C \left(1 + \frac{H}{\delta} \right)$$

One- Vs Two-Level Schwarz Preconditioners

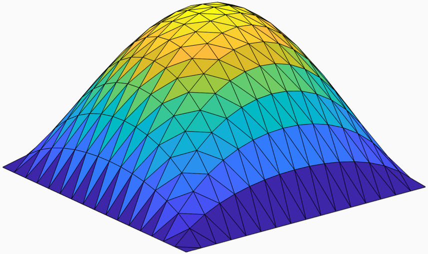
Laplace model problem in two dimensions, # subdomains = # cores, $H/h = 100$

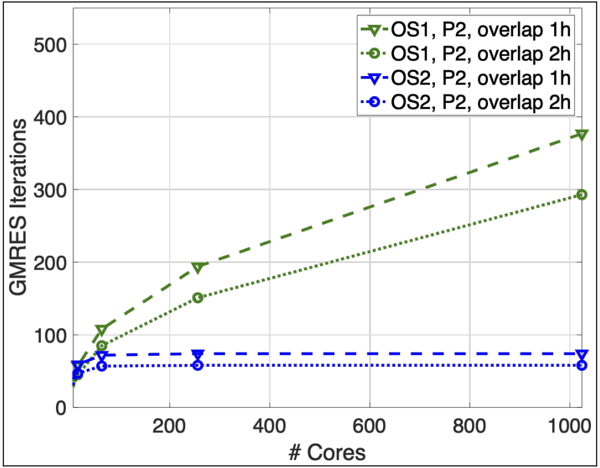




One- Vs Two-Level Schwarz Preconditioners

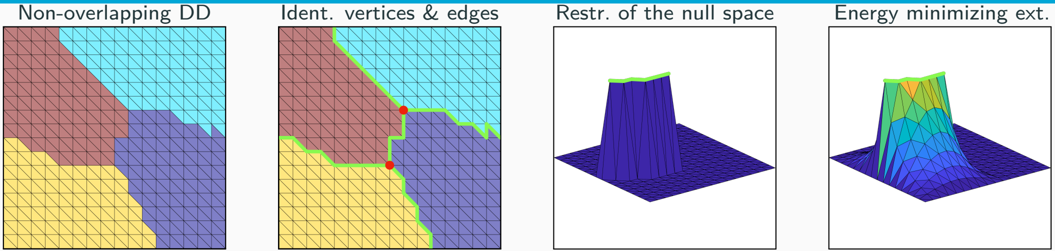
Laplace model problem in two dimensions, # subdomains = # cores, $H/h = 100$





→ We only obtain **scalability** if a **coarse level** is used.

Extension-Based GDSW Coarse Spaces



In **GDSW (Generalized–Dryja–Smith–Widlund) coarse spaces**, the coarse basis functions are chosen as **energy minimizing extensions** of functions Φ_Γ that are defined on the interface Γ:

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

The functions Φ_Γ are **restrictions of the null space of global Neumann matrix** to the **edges, vertices, and, in 3D, faces (partition of unity)** of the non-overlapping decomposition.

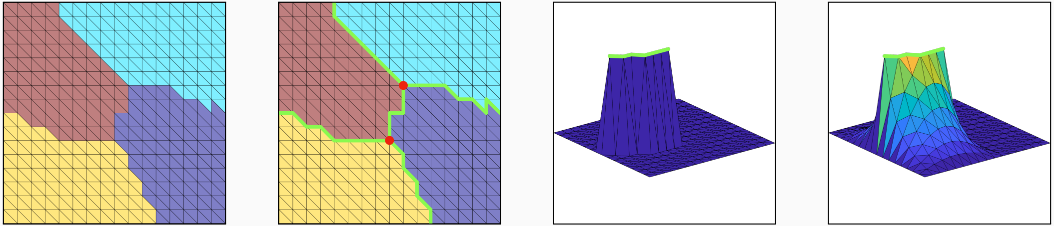
The **condition number of the GDSW operator** is bounded by

$$\kappa\left(M_{\mathrm{GDSW}}^{-1} K\right) \leq C \left(1 + \frac{H}{\delta}\right) \left(1 + \log\left(\frac{H}{h}\right)\right)^2;$$

cf. **Dohrmann, Klawonn, Widlund (2008)**, **Dohrmann, Widlund (2009, 2010, 2012)**.

→ We only obtain the exponent 2 for very irregular subdomains.

→ **Scalable and algebraic!**



In **GDSW (Generalized–Dryja–Smith–Widlund) coarse spaces**, the coarse basis functions are chosen as **energy minimizing extensions** of functions Φ_Γ that are defined on the interface Γ:

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

The functions Φ_Γ are **restrictions of the null space of global Neumann matrix** to the **edges, vertices, and, in 3D, faces (partition of unity)** of the non-overlapping decomposition.

The **condition number of the GDSW operator** is bounded by

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

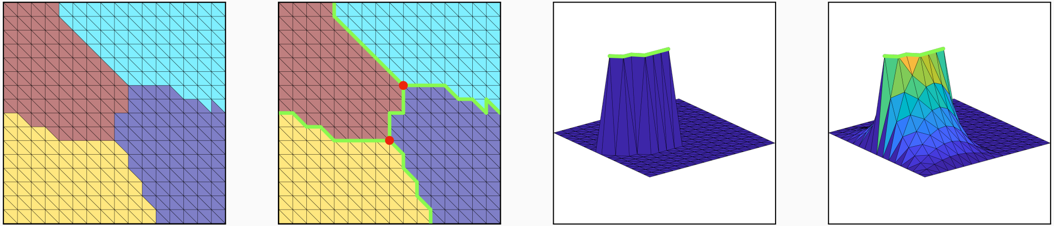
cf. **Dohrmann, Klawonn, Widlund (2008)**, **Doh** ... **012)**.

→ V ... very irreg ...

The coarse basis functions were **inspired by FETI-DP and BDDC methods**
⇒ We get the same log term.

→ **Scalable and algebraic!**

Extension-Based GDSW Coarse Spaces



In **GDSW (Generalized–Dryja–Smith–Widlund) coarse spaces**, the coarse basis functions are chosen as **energy minimizing extensions** of functions Φ_Γ that are defined on the interface Γ:

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

The functions Φ_Γ are **restrictions of the null space of global Neumann matrix** to the **edges, vertices, and, in 3D, faces (partition of unity)** of the non-overlapping decomposition.

The **condition number of the GDSW operator** is bounded by

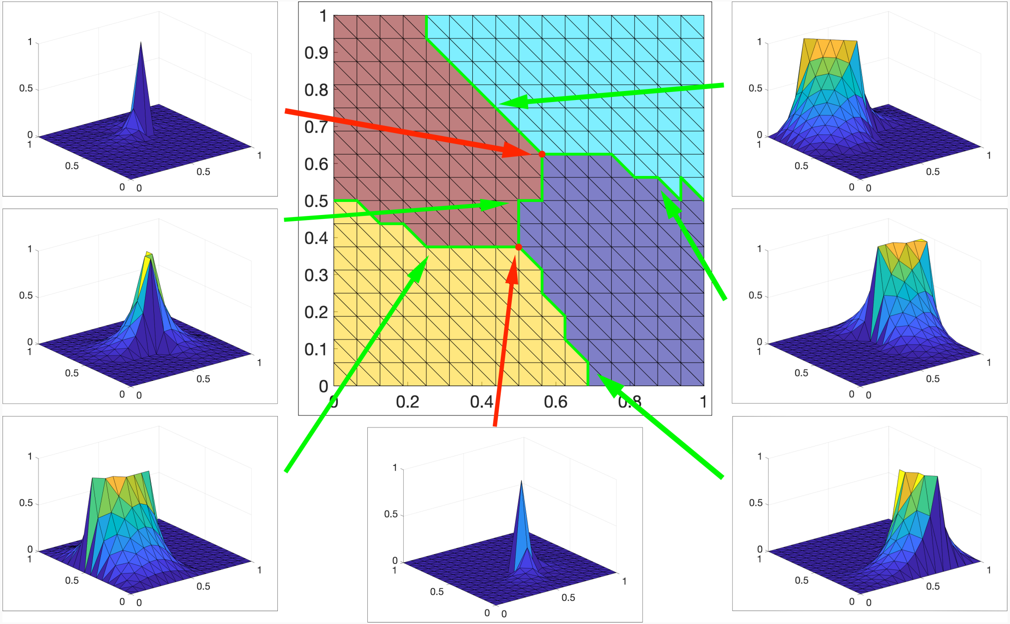
$$\kappa \left(M_{\mathrm{GDSW}}^{-1} K \right) \leq C \left(1 + \frac{H}{\delta} \right) \left(1 + \log \left(\frac{H}{h} \right) \right)^2;$$

cf. **Dohrmann, Klawonn, Widlund (2008)**, **Dohrmann, Widlund (2009, 2010, 2012)**.

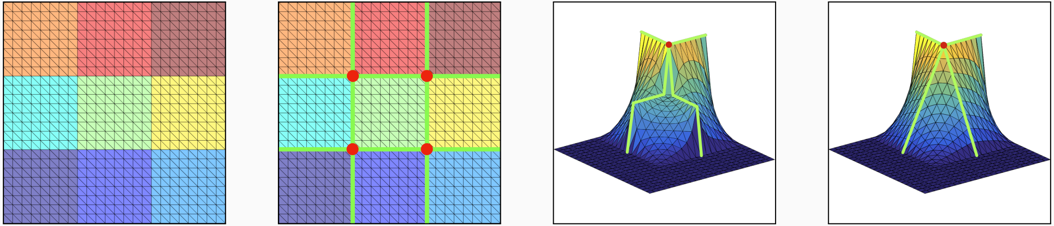
→ We only obtain the exponent 2 for very irregular subdomains.

→ **Scalable and algebraic!**

GDSW Coarse Basis Functions



Non-overlapping DD | Ident. vertices & edges | RGDSW option 1 | RGDSW option 2.2



Reduced dimension GDSW coarse spaces are constructed from **nodal interface functions** (different partition of unity compared to GDSW); as in classical GDSW coarse spaces, **energy minimizing extensions** define the values in the interior degrees of freedom; cf. **Dohrmann, Widlund (2017)**.

Option 1

We define the interface values based on the **number of adjacent vertices**

$$\Phi_i(n) = \begin{cases} \frac{1}{|\mathcal{C}_n|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

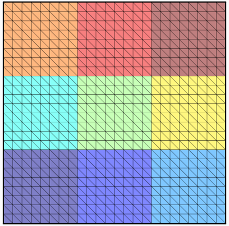
Option 2.2

We define the interface values based on the **distance to adjacent vertices**

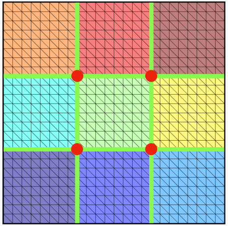
$$\Phi_i(n) = \begin{cases} \frac{1/\|\mathcal{V}_i - n\|}{\sum_{\mathcal{V}_j \in \mathcal{C}_n} 1/\|\mathcal{V}_j - n\|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

Reducing the Dimension – RGDSW Coarse Spaces

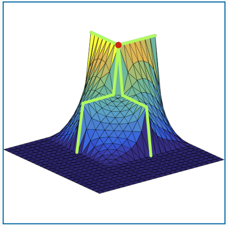
Non-overlapping DD



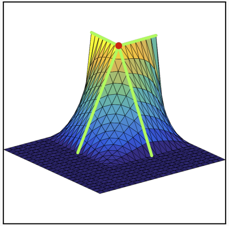
Ident. vertices & edges



RGDSW option 1



RGDSW option 2.2



Reduced dimension GDSW coarse spaces are constructed from **nodal interface functions** (different partition of unity compared to GDSW); as in classical GDSW coarse spaces, **energy minimizing extensions** define the values in the interior degrees of freedom; cf. **Dohrmann, Widlund (2017)**.

Option 1

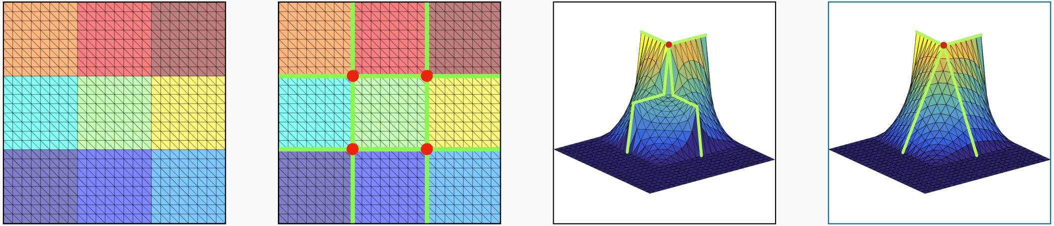
We define the interface values based on the **number of adjacent vertices**

$$\Phi_i(n) = \begin{cases} \frac{1}{|\mathcal{C}_n|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

Option 2.2

We define the interface values based on the **distance to adjacent vertices**

$$\Phi_i(n) = \begin{cases} \frac{1/\|\mathcal{V}_i - n\|}{\sum_{\mathcal{V}_j \in \mathcal{C}_n} 1/\|\mathcal{V}_j - n\|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$



Reduced dimension GDSW coarse spaces are constructed from **nodal interface functions** (different partition of unity compared to GDSW); as in classical GDSW coarse spaces, **energy minimizing extensions** define the values in the interior degrees of freedom; cf. **Dohrmann, Widlund (2017)**.

Option 1

We define the interface values based on the **number of adjacent vertices**

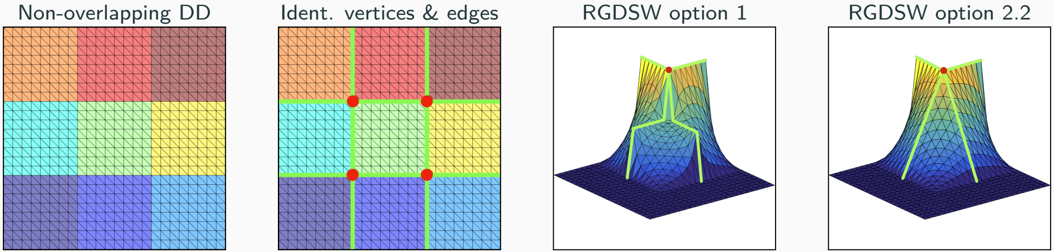
$$\Phi_i(n) = \begin{cases} \frac{1}{|\mathcal{C}_n|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

Option 2.2

We define the interface values based on the **distance to adjacent vertices**

$$\Phi_i(n) = \begin{cases} \frac{1/\|\mathcal{V}_i - n\|}{\sum_{\mathcal{V}_j \in \mathcal{C}_n} 1/\|\mathcal{V}_j - n\|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

Reducing the Dimension – RGDSW Coarse Spaces



Reduced dimension GDSW coarse spaces are constructed from **nodal interface functions** (different partition of unity compared to GDSW); as in classical GDSW coarse spaces, **energy minimizing extensions** define the values in the interior degrees of freedom; cf. **Dohrmann, Widlund (2017)**.

Option 1

We define the interface values based on the **number of adjacent vertices**

$$\Phi_i(n) = \begin{cases} \frac{1}{|\mathcal{C}_n|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

Option 2.2

We define the interface values based on the **distance to adjacent vertices**

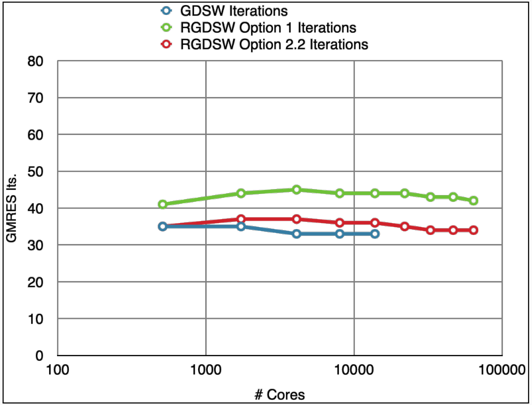
$$\Phi_i(n) = \begin{cases} \frac{1/\|\mathcal{V}_i - n\|}{\sum_{\mathcal{V}_j \in \mathcal{C}_n} 1/\|\mathcal{V}_j - n\|} & \text{if } \mathcal{V}_i \in \mathcal{C}_n, \\ 0 & \text{otherwise.} \end{cases}$$

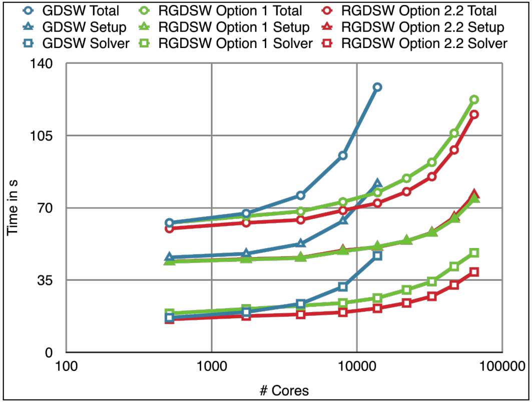
$\rightarrow$ **Less communication and global work**.

Weak Scalability of FROSch Preconditioners

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

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

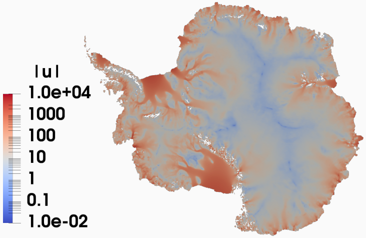




Cf. Heinlein, Klawonn, Rheinbach, Widlund (2017); computations performed on Juqueen, JSC, Jülich, Germany.

Albany

https://github.com/SNLComputation/Albany



The velocity of the ice sheet in Antarctica and Greenland is modeled by a **first-order-accurate Stokes approximation model**,

$$-\nabla \cdot (2\mu \dot{\epsilon}_1) + \rho g \frac{\partial s}{\partial x} = 0, \quad -\nabla \cdot (2\mu \dot{\epsilon}_2) + \rho g \frac{\partial s}{\partial y} = 0,$$

with a **nonlinear viscosity model** (Glen's law); cf., e.g., **Blatter (1995)** and **Pattyn (2003)**.

MPI ranks	Antarctica (**velocity**) 4 km resolution, 20 layers, 35 m dofs			Greenland (**multiphysics vel. & temperature**) 1-10 km resolution, 20 layers, 69 m dofs		
	avg. its	avg. setup	avg. solve	avg. its	avg. setup	avg. solve
512	41.9 (11)	25.10 s	12.29 s	41.3 (36)	18.78 s	4.99 s
1 024	43.3 (11)	9.18 s	5.85 s	53.0 (29)	8.68 s	4.22 s
2 048	41.4 (11)	4.15 s	2.63 s	62.2 (86)	4.47 s	4.23 s
4 096	41.2 (11)	1.66 s	1.49 s	68.9 (40)	2.52 s	2.86 s
8 192	40.2 (11)	1.26 s	1.06 s	-	-	-

Computations on Cori (NERSC).

Heinlein, Perego, Rajamanickam (submitted 2021)

There are several **extensions of the classical GDSW coarse space**, e.g.,

- **Monolithic GDSW coarse spaces for block systems:** Heinlein, Hochmuth, Klawonn (SISC 2019, IJNME 2020), Heinlein, Perego, Rajamanickam (submitted 2021)
- **Adaptive GDSW coarse spaces:** Heinlein, Klawonn, Knepper, Rheinbach (Springer LNCSE 2019,SISC 2019), Heinlein, Klawonn, Knepper, Rheinbach, Widlund (SISC 2021)
- **Multilevel GDSW preconditioners:** Heinlein, Klawonn, Rheinbach, Röver (Springer LNCSE 2019, Springer LNCSE 2020, accepted 2021), Heinlein, Rheinbach, Röver (submitted 2021)
- **Nonlinear Schwarz method with GDSW type coarse spaces:** Heinlein, Lanser (SISC 2021), Heinlein, Klawonn, Lanser (submitted 2021)

Already implemented in FROSch

Announcement: Trilinos User-Developer Group Meeting 2021

Dates

November 30th: Keynote, 20th Anniversary Celebration and Product Areas Presentations

December 1st: Applications Session

December 2nd: Developers Session

Location

Virtual Meeting

Registration

There is no registration fee for attendance; however, registration is required for our reporting purposes (see attached). **Registration may be submitted through November 30, 2021**.

More details

https://trilinos.github.io/trilinos_user-developer_group_meeting_2021.html

Exercises – Parallel Preconditioning with FROSch

FROSch

Software Environment

All the material for the exercises can be found in the **GitHub repository**

https://github.com/searhein/frosch-demo

It contains:

- A **dockerfile for automatically installing the software environment**
- **Three exercises**:
  - Exercise 1 – Implementing a Krylov Solver Using `Belos`
  - Exercise 2 – Implementing a One-Level Schwarz Preconditioner Using `FROSch`
  - Exercise 3 – Implementing a GDSW Preconditioner Using `FROSch`
- A code that includes the **solution for all three exercises**.

The GitHub repository also contains detailed **step-by-step instructions** for installing the software environment, compiling the exercises, and testing the software.

You should have received the link to the GitHub repository on Monday and **installed the software by now**. Otherwise, there will **not enough time to set up the software now and still work on the exercises**.

Working on the Exercises

Each exercise has **two parts**:

1. **Implement the missing code**; step-by-step explanations can be found in the `README.md` files.
2. **Perform numerical experiments** to investigate the behavior of the methods.

Parallelization

The code assumes a **one-to-one correspondence of MPI ranks and subdomains**. In order to run with larger numbers of subdomains, you have to increase the number of MPI ranks. For instance, for 4 MPI ranks / subdomains: `mpirun —n 4 ./EXECUTABLE`
Depending on your hardware (and the number of available processors), you can also study **computing times of the computations**.

The **solution code**

- can serve as a **reference for solving the implementation part** of the exercises.
- can be used to **directly work on the numerical experiments and skip the implementation part**.

Remainder of the Session

First, I will

- walk you through the **basic structure of the code**,
- show you how to **run the code**, and
- show you how to **visualize the solution** using Paraview.

Then, you can

- **start working on the exercises** as described in the `README.md` files and
- **ask questions** about the code and the exercises.

Please **take your time** to **look into the code** and **run numerical experiments**. I do **not expect you to finish the exercises** within the one hour. However, the `README.md` files should provide enough information to **continue working on the exercises after the session**.