Parallel Schwarz domain decomposition preconditioning and an introduction to FROSch

Alexander Heinlein

ECCOMAS Congress 2022, Oslo, Norway, June 5-9, 2022

TU Delft

- **Classical Schwarz algorithms**
- **Extending the ideas to linear preconditioning**
  Nonlinear Schwarz algorithms (as in David's talk) are inspired by these linear methods, however, nonlinear Schwarz methods will not be covered here
- **A parallel Schwarz domain decomposition solver package: `FROSch` (Fast and Robust Overlapping Schwarz)**
- **Exercises**
  You can finish those at home

Part I – Classical Schwarz domain decomposition methods

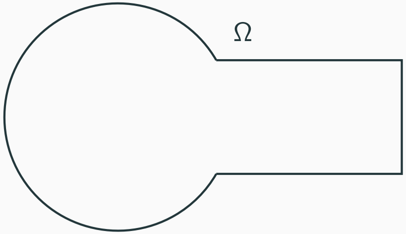
1 Literature on Domain Decomposition Methods

- Alfio Quarteroni and Alberto Valli
  Domain decomposition methods for partial differential equations
  Oxford University Press, 1999
- Barry Smith, Petter Bjorstad, and William Gropp
  Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations
  Cambridge University Press, 2004
- Andrea Toselli, and Olof Widlund
  Domain decomposition methods-algorithms and theory.
  Springer Science & Business Media, 2006
- Victorita Dolean, Pierre Jolivet, Frédéric Nataf
  An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation
  Society for Industrial and Applied Mathematics, 2016

2 The Alternating Schwarz Algorithm

Historical remarks: The **alternating Schwarz method** is the earliest **domain decomposition method (DDM)**, which has been invented by **H. A. Schwarz** and published in **1870**:

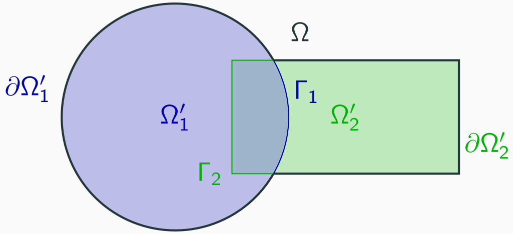
- Schwarz used the algorithm to establish the **existence of harmonic functions** with prescribed boundary values on **regions with nonsmooth boundaries**.
- The **regions are constructed recursively** by forming unions of pairs of regions **starting with “simple” regions** for which existence can be established by more elementary means.
- At the core of Schwarz’s work is a proof that **this iterative method converges in the maximum norm at a geometric rate**.



Classical “doorknob” geometry

We solve

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



We decompose Ω into two *overlapping subdomains* $\Omega_1', \Omega_2' \subset \Omega$ with

$$\begin{aligned} \overline{\Omega} &= \overline{\Omega_1' \cup \Omega_2'}, \\ \Gamma_1 &= \partial\Omega_1' \setminus \partial\Omega, \text{ and} \\ \Gamma_2 &:= \partial\Omega_2' \setminus \partial\Omega. \end{aligned}$$

The region $\Omega_1' \cap \Omega_2'$ is denoted as the **overlapping region** of the overlapping domain decomposition. This region is essential for the convergence of the following Schwarz algorithms.

The alternating Schwarz algorithm:

Given an initial guess u^0 which satisfies the boundary condition

$$u^0 = 0 \quad \text{on } \partial\Omega.$$

We perform the following **fixed point iteration**, solving **alternatingly two Dirichlet problems**:

$$(D_1) \begin{cases} -\Delta u^{n+1/2} & = & f & \text{in } \Omega_1', \\ u^{n+1/2} & = & u^n & \text{on } \partial\Omega_1' \\ u^{n+1/2} & = & u^n & \text{on } \Omega_2 := \Omega_2' \setminus \overline{\Omega_1'} \end{cases}$$

$$(D_2) \begin{cases} -\Delta u^{n+1} & = & f & \text{in } \Omega_2', \\ u^{n+1} & = & u^{n+1/2} & \text{on } \partial\Omega_2' \\ u^{n+1} & = & u^{n+1/2} & \text{on } \Omega_1 := \Omega_1' \setminus \overline{\Omega_2'} \end{cases}$$

We obtain continuous iterates which **satisfy the PDE within the overlapping subdomains** Ω_1' and Ω_2'.

Let us consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform an alternating Schwarz iteration:

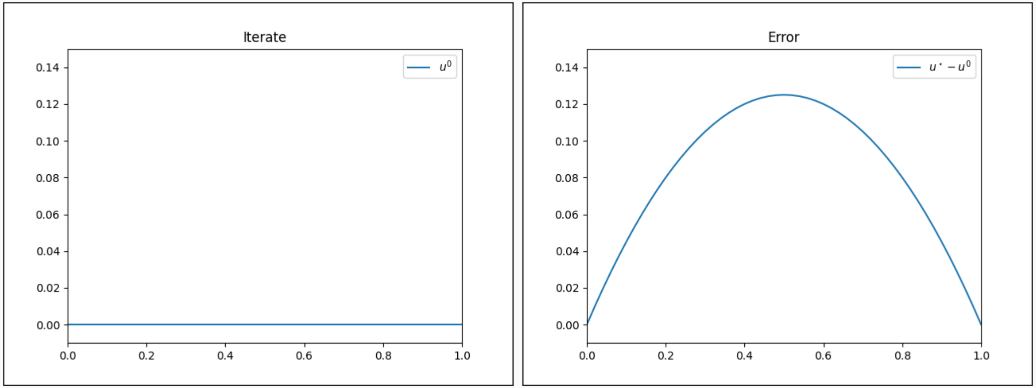


Figure 1: Iterate (left) and error (right) in iteration 0.

Let us consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform an alternating Schwarz iteration:

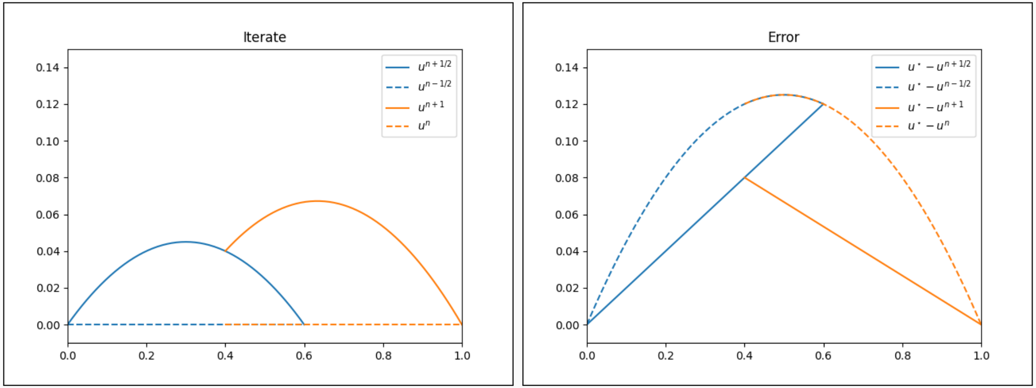


Figure 1: Iterate (left) and error (right) in iteration 1.

Let us consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform an alternating Schwarz iteration:

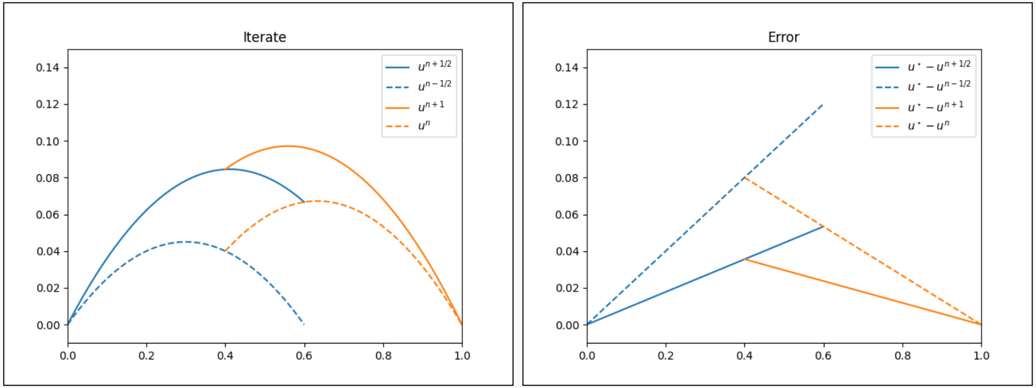


Figure 1: Iterate (left) and error (right) in iteration 2.

Let us consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform an alternating Schwarz iteration:

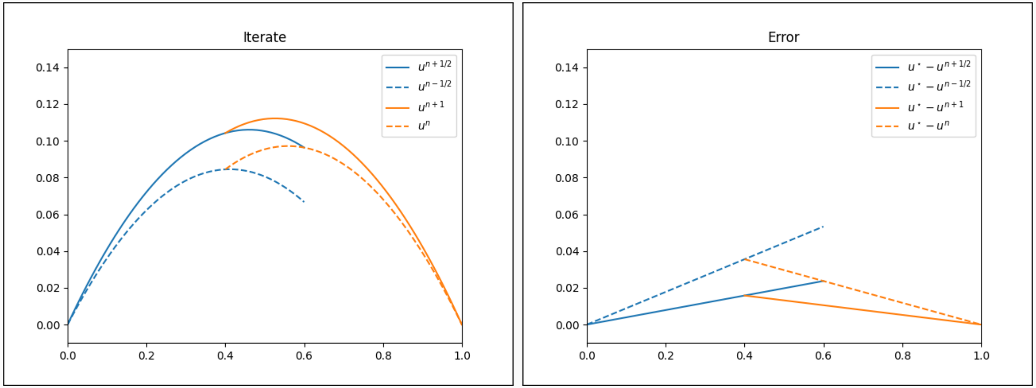


Figure 1: Iterate (left) and error (right) in iteration 3.

Let us consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform an alternating Schwarz iteration:

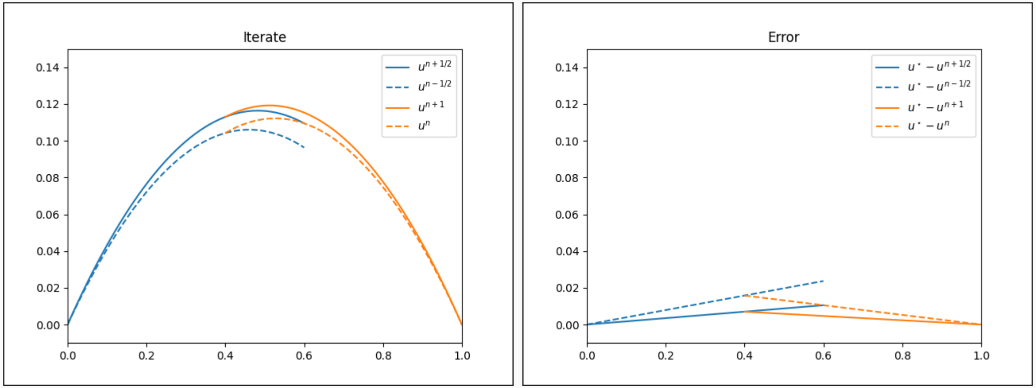
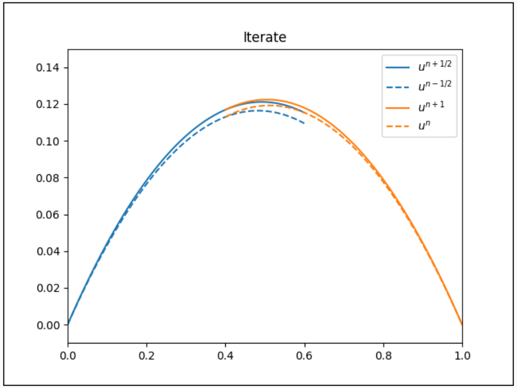


Figure 1: Iterate (left) and error (right) in iteration 4.

Let us consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform an alternating Schwarz iteration:



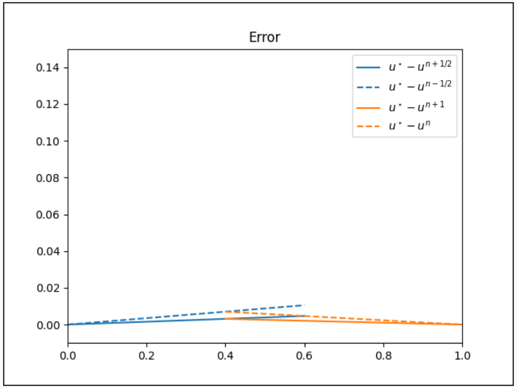


Figure 1: Iterate (left) and error (right) in iteration 5.

The alternating Schwarz algorithm is sequential because each local boundary value problem depends on the solution of the previous Dirichlet problem:

$$(D_1)\begin{cases} -\Delta u^{n+1/2} = f & \text{in } \Omega_1', \\ u^{n+1/2} = \mathbf{u^n} & \text{on } \partial\Omega_1' \\ u^{n+1/2} = \mathbf{u^n} & \text{on } \Omega_2 := \Omega_2' \setminus \overline{\Omega_1'} \end{cases}$$

$$(D_2)\begin{cases} -\Delta u^{n+1} = f & \text{in } \Omega_2, \\ u^{n+1} = \mathbf{u^{n+1/2}} & \text{on } \partial\Omega_2' \\ u^{n+1} = \mathbf{u^{n+1/2}} & \text{on } \Omega_1 := \Omega_1' \setminus \overline{\Omega_2'} \end{cases}$$

The alternating Schwarz algorithm is sequential because each local boundary value problem depends on the solution of the previous Dirichlet problem:

$$(D_1)\begin{cases} -\Delta u^{n+1/2} &= f & \text{in } \Omega_1', \\ u^{n+1/2} &= \mathbf{u^n} & \text{on } \partial\Omega_1' \\ u^{n+1/2} &= \mathbf{u^n} & \text{on } \Omega_2 := \Omega_2' \setminus \overline{\Omega_1'} \end{cases}$$

$$(D_2)\begin{cases} -\Delta u^{n+1} &= f & \text{in } \Omega_2, \\ u^{n+1} &= \mathbf{u^{n+1/2}} & \text{on } \partial\Omega_2' \\ u^{n+1} &= \mathbf{u^{n+1/2}} & \text{on } \Omega_1 := \Omega_1' \setminus \overline{\Omega_2'} \end{cases}$$

Idea: For all red terms, we **use the values from the previous iteration**. Then, the both Dirichlet problem **can be solved at the same time**.

3 The Parallel Schwarz Algorithm

The **parallel Schwarz algorithm** has been introduced by **Lions (1988)**. Therefore, given an initial guess $u^0 =: u_1^0 := u_2^0$ which satisfies the boundary condition $u = 0$ on $\partial\Omega$. Then, we perform the following fixed-point iteration, solving, again, two Dirichlet problems:

$$\begin{aligned}
(D_1) &\begin{cases} -\Delta u_1^{n+1} = f & \text{in } \Omega_1', \\ u_1^{n+1} = \mathbf{u_2^n} & \text{on } \partial\Omega_1' \end{cases} \\
(D_2) &\begin{cases} -\Delta u_2^{n+1} = f & \text{in } \Omega_2, \\ u_2^{n+1} = \mathbf{u_1^n} & \text{on } \partial\Omega_2' \end{cases}
\end{aligned} \tag{3.1}$$

Since u_1^n and u_2^n are both computed in the previous iteration, the problems can be solved independent of each other.

Note: The **cost for a single iteration is the same as in the alternating case**. However, in parallel computing, we could solve both problems at the same time.

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform a parallel Schwarz iteration:

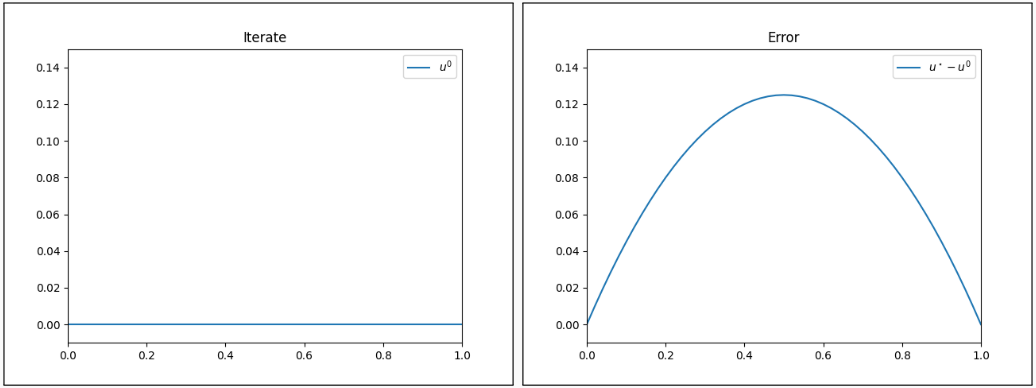


Figure 2: Iterate (left) and error (right) in iteration 0.

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform a parallel Schwarz iteration:

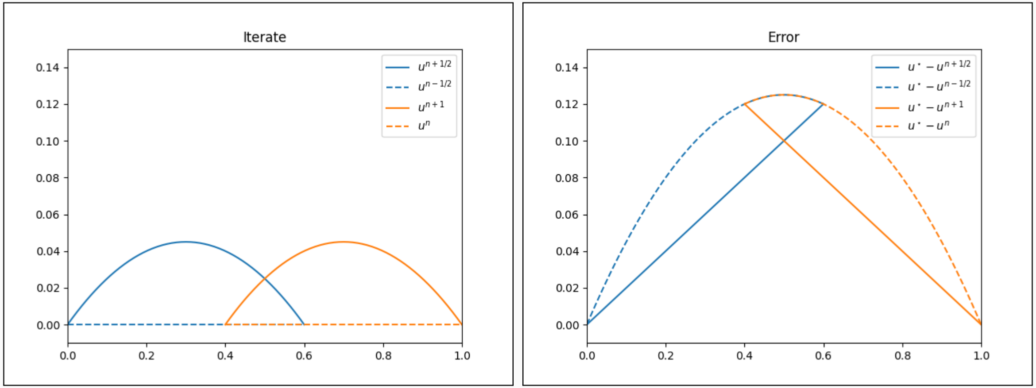


Figure 2: Iterate (left) and error (right) in iteration 1.

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform a parallel Schwarz iteration:

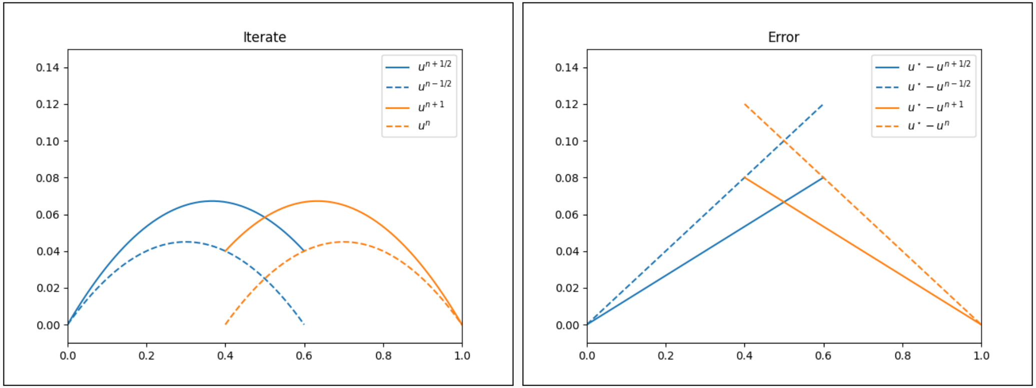


Figure 2: Iterate (left) and error (right) in iteration 2.

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform a parallel Schwarz iteration:

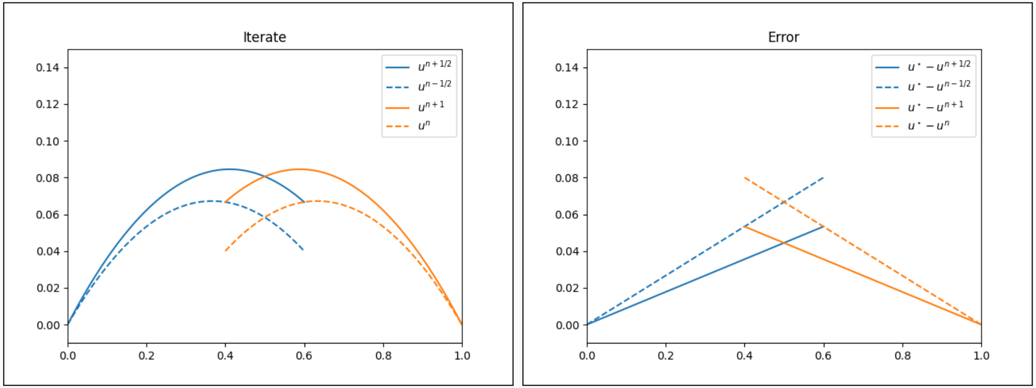


Figure 2: Iterate (left) and error (right) in iteration 3.

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform a parallel Schwarz iteration:

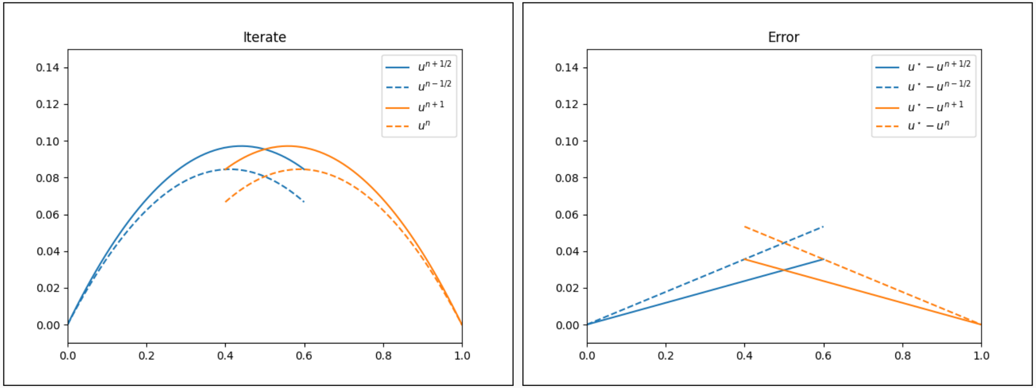


Figure 2: Iterate (left) and error (right) in iteration 4.

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

We perform a parallel Schwarz iteration:

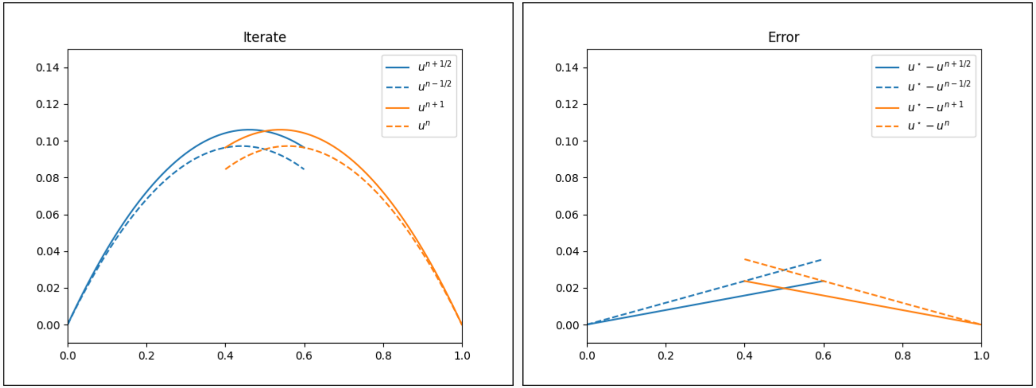
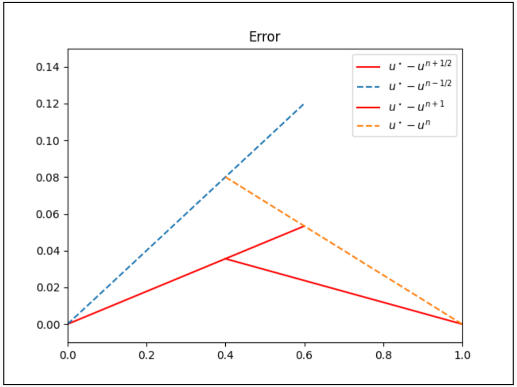


Figure 2: Iterate (left) and error (right) in iteration 5.

4 Comparison of the two Methods



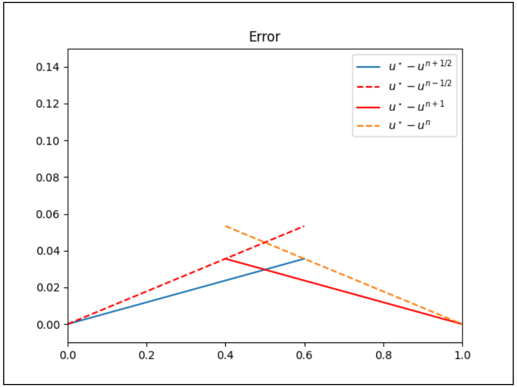


Figure 3: Error in **iterations 1 and 2 of the alternating Schwarz** iteration (left) and error in **iterations 3 and 4 of the parallel Schwarz iteration** (right).

We can see that the alternating Schwarz method convergence **twice as fast** as the parallel Schwarz method. However, the solutions on the two subdomains have to computed **sequentially**.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

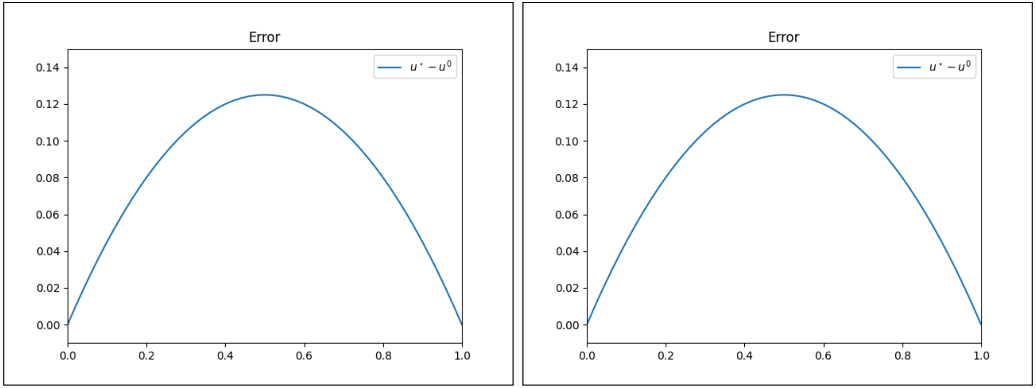
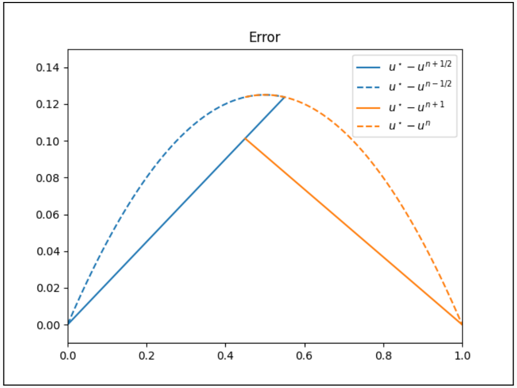


Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 0 of the alternating Schwarz iteration.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$



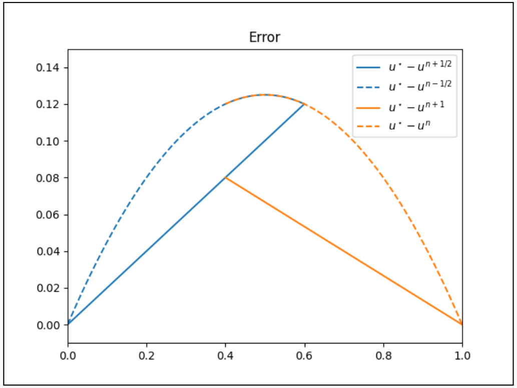


Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 1 of the alternating Schwarz iteration.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

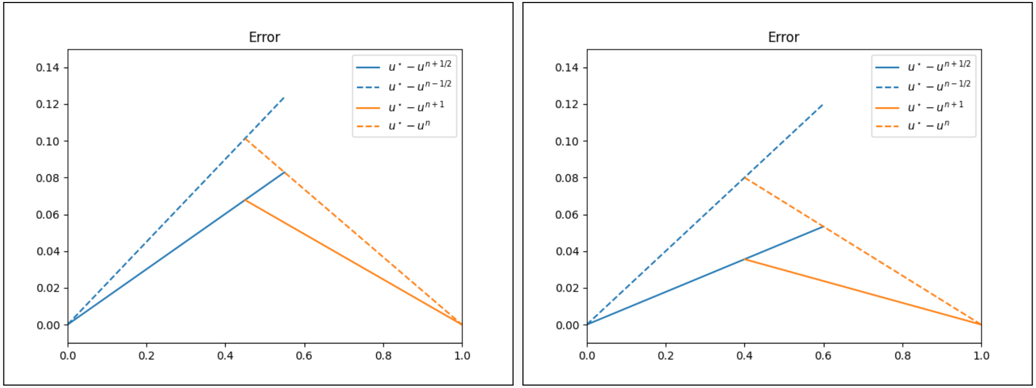
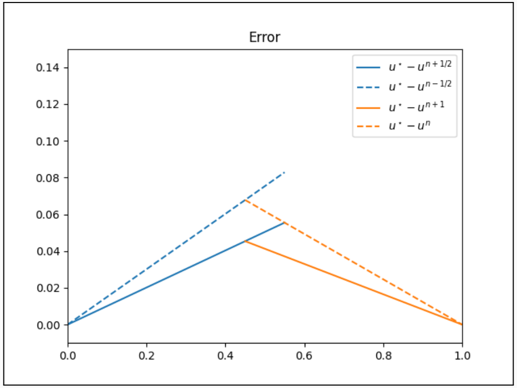


Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 2 of the alternating Schwarz iteration.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$



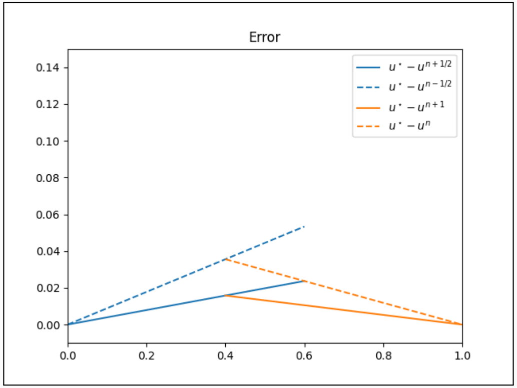
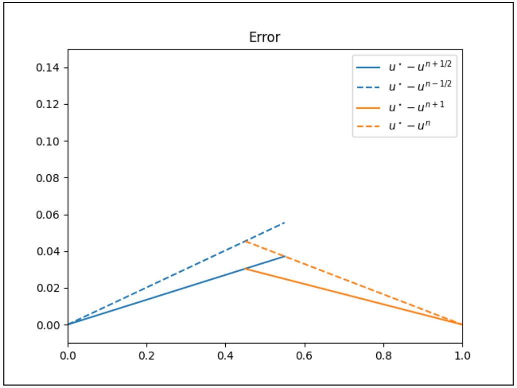


Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 3 of the alternating Schwarz iteration.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$



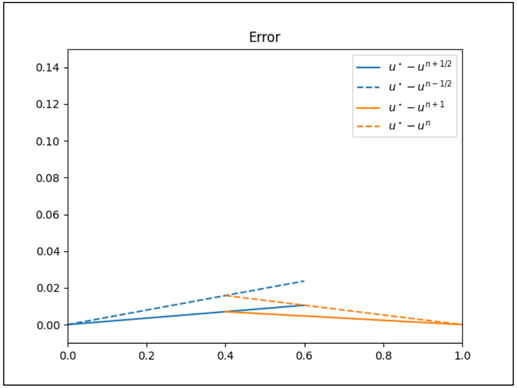


Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 4 of the alternating Schwarz iteration.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

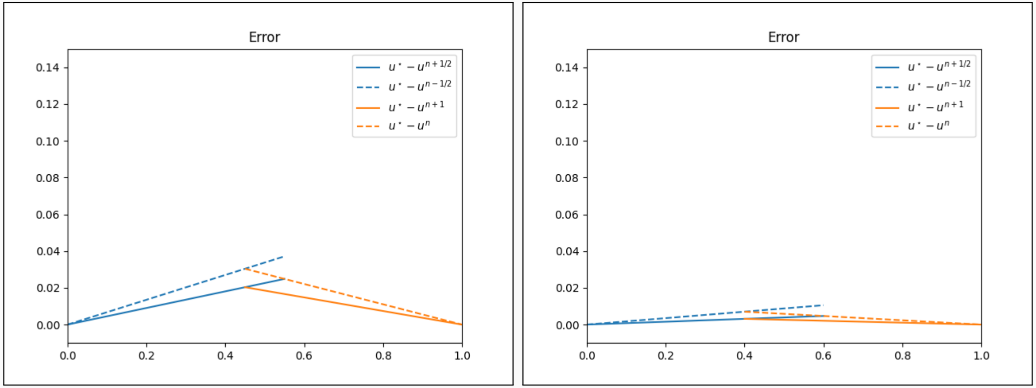


Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 5 of the alternating Schwarz iteration.

5 Effect of the Size of the Overlap

Let us again consider the simple boundary value problem: Find u such that

$$-u'' = 1, \text{ in } [0,1], \quad u(0) = u(1) = 0$$

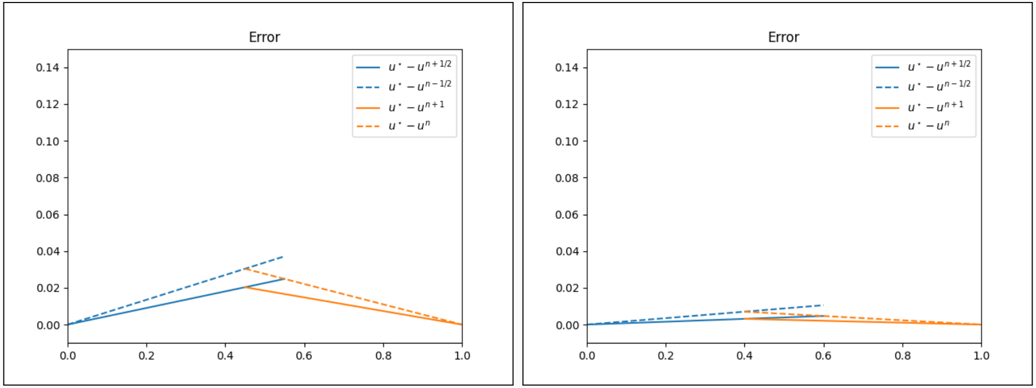


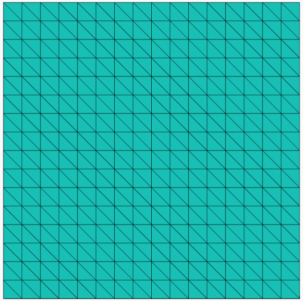
Figure 4: Error with an overlap of 0.05 (left) and an overlap of 0.1 (right) in iteration 5 of the alternating Schwarz iteration.

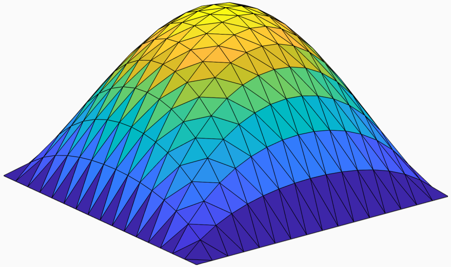
There are also nonoverlapping domain decomposition methods, based on **Dirichlet–Neumann**, **Neumann–Neumann** (BDD), and **Dirichlet–Dirichlet** (FETI) methods, but they will not be covered here.

Let us now apply this concept to construct **(parallel) domain decomposition preconditioners**.

Part II – Schwarz domain decomposition preconditioners

6 Model Problem



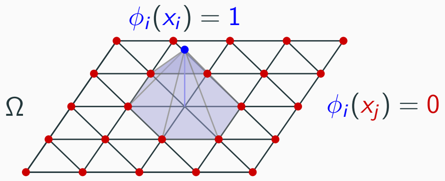


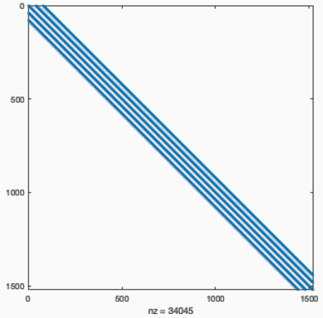
Let us consider the simple **diffusion model problem** ($\alpha(x) = 1$):

$$
\begin{aligned}
-\Delta u &= f \quad \text{in } \Omega = [0,1]^2, \\
u &= 0 \quad \text{on } \partial\Omega.
\end{aligned}
$$

Discretization using finite elements yields the linear equation system

$$Ku = f.$$





- Due to the local support of the finite element basis functions, the resulting system is **sparse**.
- However, due to the **superlinear complexity and memory cost**, the use of direct solvers becomes infeasible for fine meshes, that is, for the **resulting large sparse equation systems**.

→ We will employ iterative solvers:
For our elliptic model problem, the system matrix is symmetric positive definite, such that we can use the **preconditioned gradient descent (PCG) method**.

7 Preconditioned Conjugate Gradient (PCG) Method

Algorithm 1: Preconditioned conjugate gradient method

```
Result: Approximate solution of the linear equation system $Ax = b$
Given: Initial guess $x^{(0)} \in \mathbb{R}^n$ and tolerance $\varepsilon > 0$
$r^{(0)} := b - Ax^{(0)}$
$p^{(0)} := y^{(0)} := M^{-1} r^{(0)}$
while $\left\| r^{(k)} \right\| \geq \varepsilon \left\| r^{(0)} \right\|$ do
    $\alpha_k := \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(Ap^{(k)}, p^{(k)}\right)}$
    $x^{(k+1)} := x^{(k)} + \alpha_k y^{(k)}$
    $r^{(k+1)} := r^{(k)} - \alpha_k A p^{(k)}$
    $y^{(k+1)} := M^{-1} r^{(k+1)}$
    $\beta_k := \frac{\left(y^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$
    $p^{(k+1)} := r^{(k+1)} - \beta_k p^{(k)}$
end
```

Theorem 1

Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite. Then, for any $x^{(0)} \in \mathbb{R}^n$, the **(P)CG method** *converges to the solution x of the linear system $Ax = b$ in at most n steps.*

Theorem 1

Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Then, for any $x^{(0)} \in \mathbb{R}^n$, the **(P)CG method** *converges to the solution x of the linear system $Ax = b$ in at most n steps.*

The PCG methods solves the preconditioned system

$$M^{-1}Ax = M^{-1}b \qquad \overset{M^{-1}\,\text{inv.}}{\Leftrightarrow} \qquad Ax = b,$$

with the preconditioner M^{-1}. If $M^{-1} \approx A^{-1}$, this system is **easier to solve**.

Theorem 1

Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Then, for any $x^{(0)} \in \mathbb{R}^n$, the **(P)CG method** *converges to the solution x of the linear system $Ax = b$ in at most n steps.*

The PCG methods solves the preconditioned system

$$M^{-1}Ax = M^{-1}b \qquad \overset{M^{-1}\text{inv.}}{\Leftrightarrow} \qquad Ax = b,$$

with the preconditioner M^{-1}. If $M^{-1} \approx A^{-1}$, this system is **easier to solve**.

Theorem 2

Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite. Then the **PCG method** *converges and the following error estimate holds:*

$$\left\| e^{(k)} \right\|_A \leq 2 \left(\frac{\sqrt{\kappa\left(M^{-1}A\right)} - 1}{\sqrt{\kappa\left(M^{-1}A\right)} + 1} \right)^k \left\| e^{(0)} \right\|_A,$$

where $\kappa\left(M^{-1}A\right) = \frac{\lambda_{\max}\left(M^{-1/2}AM^{-1/2}\right)}{\lambda_{\min}\left(M^{-1/2}AM^{-1/2}\right)}$.

Do we need a preconditioner?

Do we need a preconditioner?

Theorem 3 (Condition number of the mass matrix)

There exists a constant $c > 0$, independent of h, such that

$$\kappa(M) \leq c \frac{h^d}{\left(\min_{T \in \tau_h} h_T\right)^d},$$

where $M = \left((\varphi_i, \varphi_j)_{L_2(\Omega)} \right)_{i,j}$ is the so-called mass matrix and $\kappa(M)$ the spectral condition number of M.

Note: *The mass matrix M is generally not related to the preconditioner M^{-1}.*

Do we need a preconditioner?

Theorem 3 (Condition number of the mass matrix)

There exists a constant $c > 0$, independent of h, such that

$$\kappa(M) \leq c \frac{h^d}{\left(\min_{T \in \tau_h} h_T\right)^d},$$

where $M = \left((\varphi_i, \varphi_j)_{L_2(\Omega)} \right)_{i,j}$ is the so-called mass matrix and $\kappa(M)$ the spectral condition number of M.

Note: *The mass matrix M is generally not related to the preconditioner M^{-1}.*

Theorem 4 (Condition number of the stiffness matrix)

Let K be the stiffness matrix and M be the mass matrix for the model problem. Then there exists a constant $c > 0$, independent of h, such that for the spectral condition number holds:

1. $\kappa\left(M^{-1}K\right) \leq c \left(\min_{T \in \tau_h} h_T\right)^{-2}$
2. $\kappa(K) \leq c \left(\min_{T \in \tau_h} h_T\right)^{-2} \kappa(M)$

Do we need a preconditioner?

Theorem 3 (Condition number of the mass matrix)

There exists a constant $c > 0$, independent of h, such that

$$\kappa\left(M\right) \leq c \frac{h^d}{\left(\min_{T \in \tau_h} h_T\right)^d},$$

where $M = \left((\varphi_i, \varphi_j)_{L_2(\Omega)} \right)_{i,j}$ is the so-called mass matrix and $\kappa\left(M\right)$ the spectral condition number of M.

Note: *The mass matrix M is generally not related to the preconditioner M^{-1}.*

Theorem 4 (Condition number of the stiffness matrix)

Let K be the stiffness matrix and M be the mass matrix for the model problem. Then there exists a constant $c > 0$, independent of h, such that for the spectral condition number holds:

1. $\kappa\left(M^{-1}K\right) \leq c \left(\min_{T \in \tau_h} h_T\right)^{-2}$
2. $\kappa\left(K\right) \leq c \left(\min_{T \in \tau_h} h_T\right)^{-2} \kappa\left(M\right)$

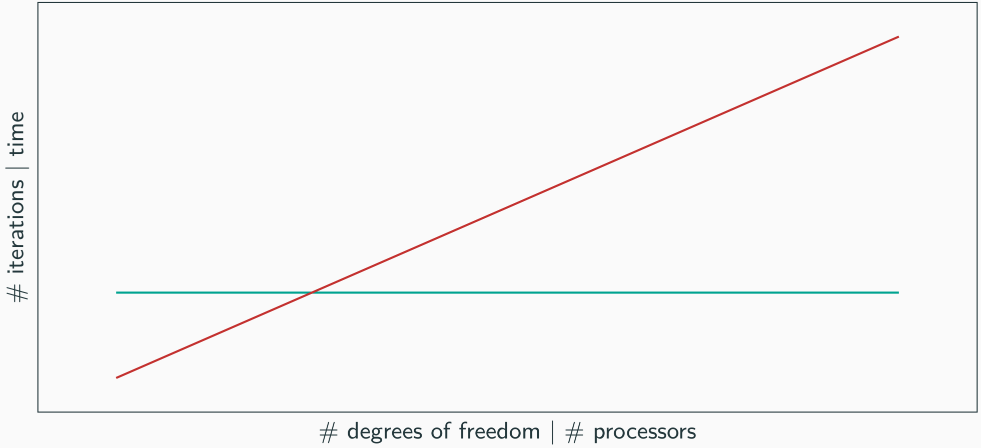
$\Rightarrow$ **Convergence of the PCG method will deteriorate** when refining the mesh.

Goal – Numerical & Parallel (Weak) Scalability

Increase the problem size while keeping

$$\frac{\#\text{ degrees of freedom}}{\#\text{ processors}}$$

fixed.

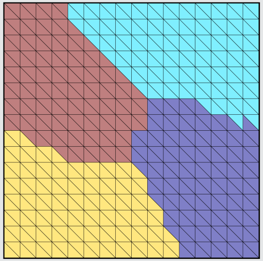


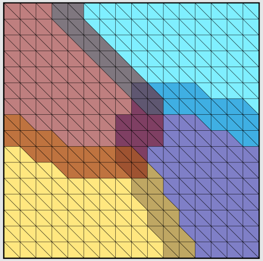
8 One-Level Overlapping Schwarz Preconditioners

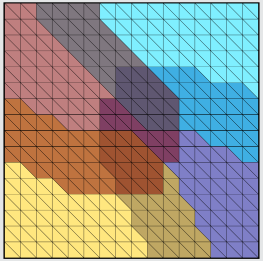
Overlapping domain decomposition

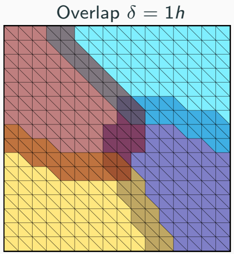
As the classical alternating and parallel Schwarz method **(overlapping) Schwarz preconditioners** are based on **overlapping decompositions** of the computational domain

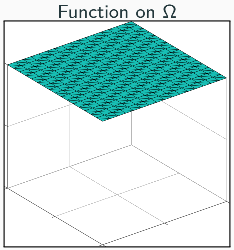
$$\Omega = \bigcup_{i=1}^{N} \Omega_i'.$$

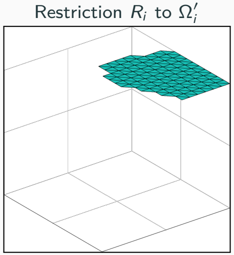

Nonoverlap. DD


Overlap $\delta = 1h$


Overlap $\delta = 2h$







Based on an **overlapping domain decomposition**, we define an additive **one-level Schwarz preconditioner**

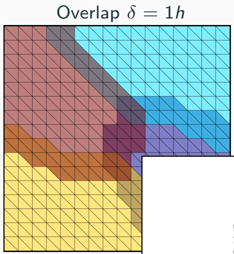
$$M_{\text{OS-1}}^{-1} = \sum\nolimits_{i=1}^{N} R_i^T K_i^{-1} R_i,$$

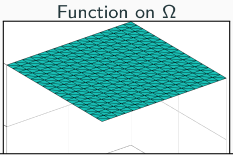
where R_i and R_i^T are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^T$. The K_i correspond to **local Dirichlet problems** on the overlapping subdomains.

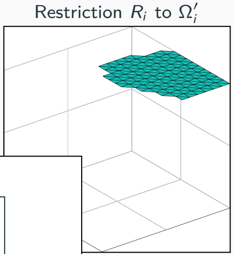
Condition number bound:

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

where the constant C is **independent of the subdomain size** H **and the width of the overlap** δ.

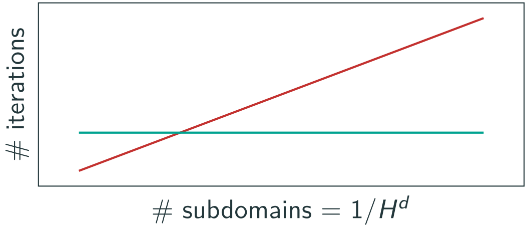






Based on an **ove** **e-level Schwarz preconditioner**

Numerical scalability



where R_i and R_i^T Ω_i', and $K_i := R_i K R_i^T$. The K_i correspond to **local Dirichlet problems** on the overlapping subdomains.

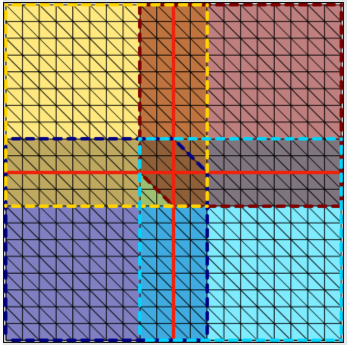
Condition number bound:

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

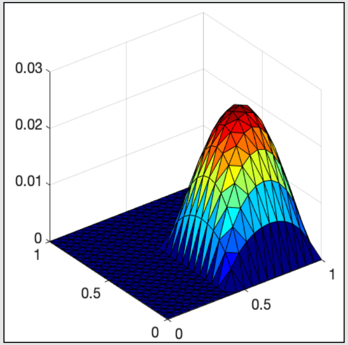
where the constant C is **independent of the subdomain size H and the width of the overlap δ**.

Solving a local subdomain problem

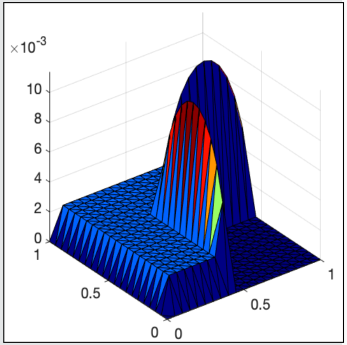
Overlap $\delta = 2h$



Solution on Ω_2



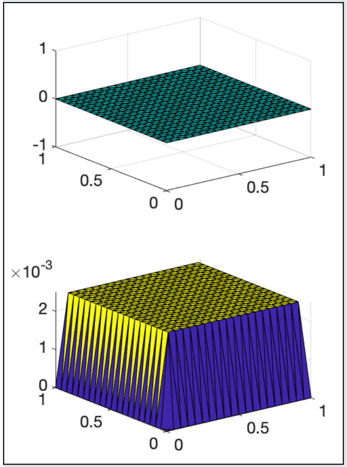
Corresponding residual



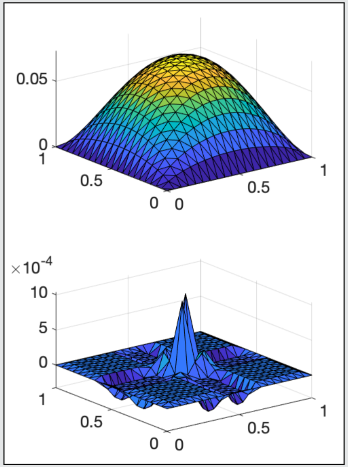
$\rightarrow$ **Zero residual only inside this subdomain** but **particularly large residual inside the overlap**.

Convergence of the PCG method with a one-level Schwarz preconditioner

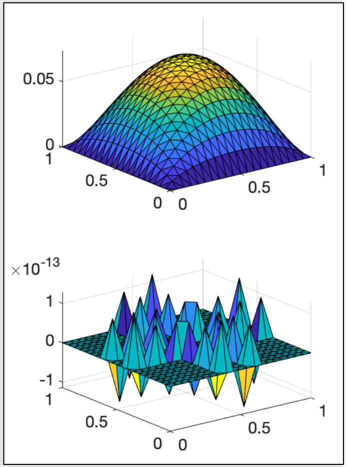
Initial guess



5 PCG iterations



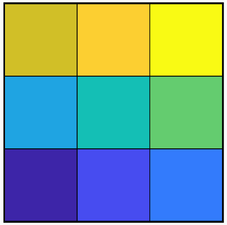
Converged (13 its)



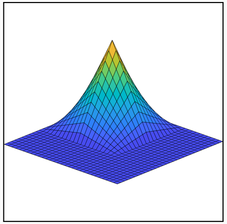
→ **Fast convergence** of the preconditioned gradient decent (PCG) method **(low number of subdomains)**.

9 Two-Level Overlapping Schwarz Preconditioners

Coarse triangulation



Nodal bilinear basis function



The additive **two-level Schwarz preconditioner** reads

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

where Φ contains the coarse basis functions and $K_0 := \Phi^T K \Phi$.

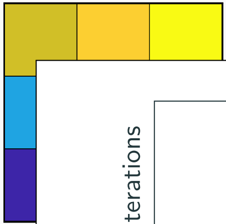
Condition number bound:

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

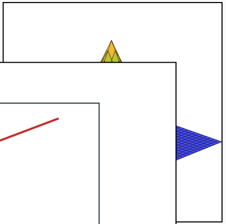
where the constant C is **independent of** h**,** δ**, and** H; cf., e.g., **Toselli, Widlund (2005)**.

9 Two-Level Overlapping Schwarz Preconditioners

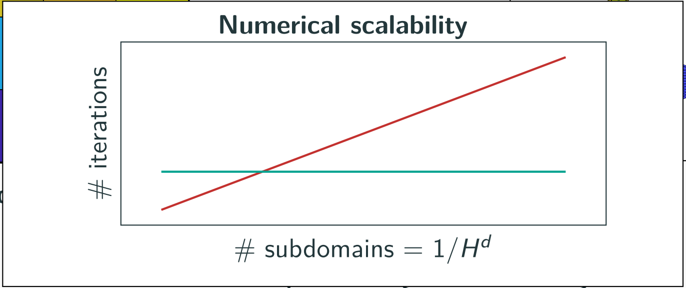
Coarse triangulation



Nodal bilinear basis function



The additive **tw**



coarse level – global

first level – local

where Φ contains the coarse basis functions and $K_0 := \Phi^T K \Phi$.

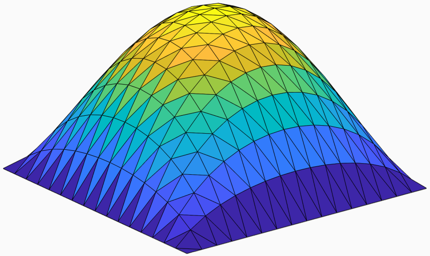
Condition number bound:

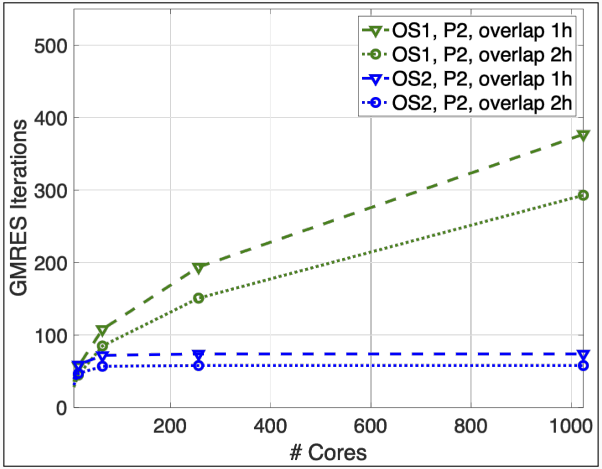
$$\kappa\left(M_{\text{OS-2}}^{-1} K\right) \leq C \left(1 + \frac{H}{\delta}\right)$$

where the constant C is **independent of** h**,** δ**, and** H; cf., e.g., **Toselli, Widlund (2005)**.

One- Vs Two-Level Schwarz Preconditioners

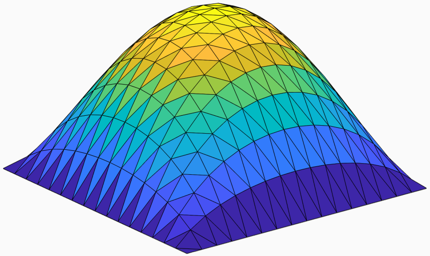
Diffusion model problem in two dimensions, # subdomains = # cores, $H/h = 100$

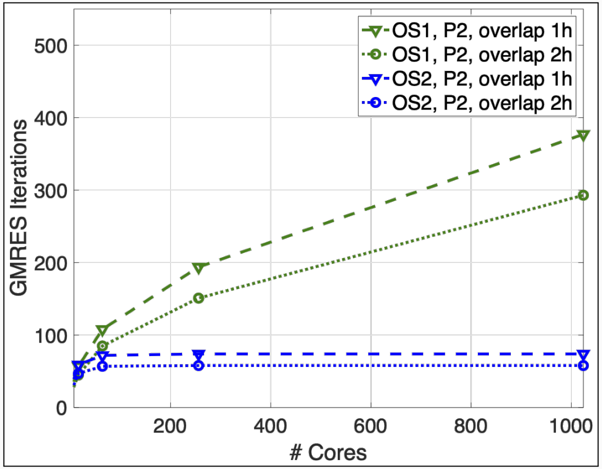




One- Vs Two-Level Schwarz Preconditioners

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

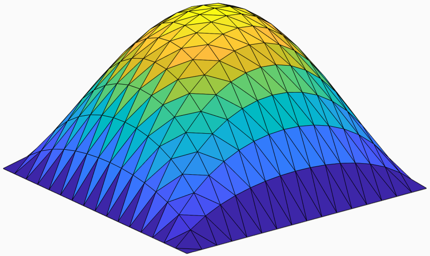


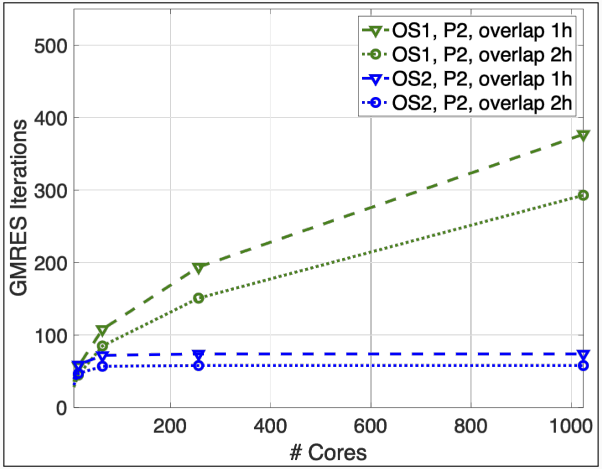


→ We only obtain **numerical scalability** if a **coarse level** is used.

One- Vs Two-Level Schwarz Preconditioners

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





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

→ Convergence is **faster** for **larger overlaps**.

10 A Brief Overview Over the Theoretical Framework

In order to establish a condition number bound for $\kappa\left(M_{\mathrm{ad}}^{-1}K\right)$ based on the **abstract Schwarz framework**, we have to verify the following **three assumptions**:

Assumption 1: Stable Decomposition

There exists a constant C_0 such that, for every $u \in V$, there exists a decomposition $u = \sum_{i=0}^{N} R_i^T u_i$, $u_i \in V_i$, with

$$\sum_{i=0}^{N} a_i(u_i, u_i) \leq C_0^2 a(u, u).$$

Assumption 2: Strengthened Cauchy-Schwarz Inequality

There exist constants $0 \leq \epsilon_{ij} \leq 1$, $1 \leq i, j \leq N$, such that

$$\left|a(R_i^T u_i, R_j^T u_j)\right| \leq \epsilon_{ij} \left(a(R_i^T u_i, R_i^T u_i)\right)^{1/2} \left(a(R_j^T u_j, R_j^T u_j)\right)^{1/2}$$

for $u_i \in V_i$ and $u_j \in V_j$. (Consider $\mathcal{E} = (\varepsilon_{ij})$ and $\rho\left(\mathcal{E}\right)$ its spectral radius)

Assumption 3: Local Stability

There exists $\omega < 0$, such that

$$a(R_i^T u_i, R_i^T u_i) \leq \omega a_i(u_i, u_i), \quad u_i \in \operatorname{range}\left(\tilde{P}_i\right), \quad 0 \leq i \leq N.$$

General Condition Number Bound

With Assumption 1–3, we have

$$\kappa\left(M_{\mathrm{ad}}^{-1} K\right) \leq C_0^2 \omega \left(\rho\left(\mathcal{E}\right) + 1\right)$$

for

$$M_{\mathrm{ad}}^{-1} = \sum_{i=0/1}^{N} R_i^T K_i^{-1} R_i;$$

see, e.g., Toselli, Wildund (2005).

To obtain a condition number bound for a specific additive Schwarz preconditioner, we have to bound ω, $\rho\left(\mathcal{E}\right)$, and C_0^2.

The constants ω and $\rho\left(\mathcal{E}\right)$ can often easily be bounded.

Exact Solvers

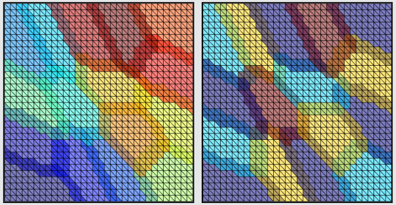
If we choose the local bilinear forms as

$$a_i(u_i, u_i) := a(R_i^T u_i, R_i^T u_i),$$

we obtain $K_i = R_i K R_i^T$ and $\omega = 1$.

$\rightarrow$ For exact **exact local and coarse solvers**, ω does not depend on the coefficient.

Coloring Constant



The spectral radius $\rho\left(\mathcal{E}\right)$ is bounded by the number of colors N^c of the domain decomposition.

$\rightarrow$ N^c depends only on the **domain decomposition** but not on the coefficient function.

Assumption 3 is typically proved by constructing functions $u_i \in V_i$, $i = 0, \dots, N$, such that

$$u = \sum_{i=0}^{N} R_i^T u_i \text{ and } \sum_{i=0}^{N} a_i(u_i, u_i) \leq C_0^2 a(u, u)$$

for any given function $u \in V$. Let us sketch the **difference between the one- and two-level preconditioners**.

One-level Schwarz preconditioner

During the proof of the condition number, we have to use an L^2-norm using Friedrich's inequality globally on Ω:

$$\sum\nolimits_{i=1}^{N} \|u\|_{L_2(\Omega_i)}^2 = \|u\|_{L_2(\Omega)}^2 \leq C \, |u|_{H^1(\Omega)}^2 ,$$

This results in

$$\sum_{i=1}^{N} a_i \left(u_i, u_i\right) \leq C \left(1 + \frac{H}{\delta}\right) a\left(u, u\right) + C \frac{1}{H\delta} a\left(u, u\right)$$

Since $\frac{H}{\delta} \leq \frac{1}{H\delta}$, we obtain

$$\sum\nolimits_{i=1}^{N} a_i \left(u_i, u_i\right) \leq C \left(1 + \frac{\mathbf{1}}{\mathbf{H}\delta}\right) a\left(u, u\right) .$$

Two-level Schwarz preconditioner

In contrast to the one-level method, we can estimate the L^2-norm locally since we instead have the term $u - u_0$

$$\sum\nolimits_{i=1}^{N} \|u - u_0\|_{L_2(\Omega_i')}^2 \leq \sum_{i=1}^{N} CH^2 \, |u|_{H^1\left(\omega_{\Omega_i}\right)}^2 .$$

Different from the one-level preconditioner, we obtain an H^2 term in the final estimate:

$$\sum_{i=1}^{N} a_i \left(u_i, u_i\right) \leq C \left(1 + \frac{H}{\delta}\right) a\left(u, u\right) + C \frac{1}{H\delta} \mathbf{H^2} a\left(u, u\right)$$
$$\leq C \left(1 + \frac{\mathbf{H}}{\delta}\right) a\left(u, u\right)$$

11 Some Comments on Constructing Schwarz Preconditioners

Combining Schwarz operators

Given Schwarz operators $P_0, \ldots, P_N$ (e.g, for one-level or two-level Schwarz preconditioners),

$$P_i = R_i^T K_i^{-1} R_i K,$$

can be combined in several ways, e.g.:

Additive (parallel):

$$P_{\mathrm{ad}} = \sum_{i=0}^{N} P_i = \sum_{i=0}^{N} R_i^T K_i^{-1} R_i K$$

Multiplicative (sequential):

$$P_{\mathrm{mu}} = I - (I - P_N)(I - P_{N-1}) \cdots (I - P_0)$$
$$P_{\mathrm{mu-sym}} = I - (I - P_0) \cdots (I - P_{N-1})(I - P_N)(I - P_{N-1}) \cdots (I - P_0)$$

Hybrid (parallel & sequential):

$$P_{\mathrm{hy-1}} = I - (I - P_0)\left(I - \sum_{i=0}^{N} P_i\right)(I - P_0)$$
$$P_{\mathrm{hy-2}} = \alpha P_0 + I - (I - P_N) \cdots (I - P_1);$$

cf. Toselli and Widlund (2005).

Restricted Schwarz Preconditioner (Cai and Sarkis (1999))

Replace the prolongation R_i^T by $\widetilde{R}_i^T$,

$$M_{\text{OS-1}}^{-1} = \sum\nolimits_{i=1}^{N} \widetilde{R}_i^T K_i^{-1} R_i,$$

where

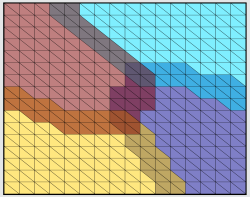
$$\sum\nolimits_{i=1}^{N} \widetilde{R}_i^T = I.$$

Therefore, we can just introduce a diagonal scaling matrix D, such that

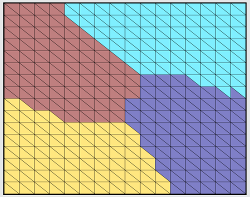
$$\widetilde{R}_i^T = D R_i^T,$$

for example based on a nonoverlapping domain decomposition or an inverse multiplicity scaling.

R_i



$\widetilde{R}_i^T$



This often **improves the convergence**, however, the preconditioner becomes **unsymmetric**.

Changing the local and coarse solvers

For solving

$$K_i^{-1}, \quad i = 0, \ldots, N,$$

we can employ **inexact solvers** instead of direct solvers, such as

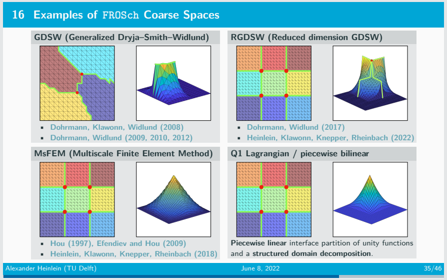
- iterative solvers
- preconditioners

to **speedup the computing times**. Of course, **convergence might slow down** a bit a the same time.

Choose another coarse basis

As it turns out, the choice of a **suitable coarse basis** is one of the more important ingredients for a **scalable and robust domain decomposition solver**.

We will discuss this again in a few slides.



16 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)
- Heinlein, Klawonn, Knepper, Rheinbach (2022)

MsFEM (Multiscale Finite Element Method)

- Hou (1997), Efendiev and Hou (2009)
- Heinlein, Klawonn, Knepper, Rheinbach (2018)

Q1 Lagrangian / piecewise bilinear

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

Alexander Heinlein (TU Delft) June 8, 2022 35/46

Part III – Schwarz domain decomposition preconditioners in FROSch

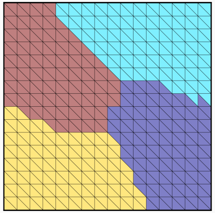
12 Wishlist for a Parallel Schwarz Preconditioning Package

Parallel distributed system

$$Ax = b$$

with

$$A = \begin{bmatrix} \bullet & \bullet & & & & & & \\ \bullet & \bullet & \bullet & & & & & \\ & \bullet & \bullet & \bullet & & & & \\ & & \bullet & \bullet & \bullet & & & \\ & & & \bullet & \bullet & \bullet & & \\ & & & & \bullet & \bullet & \bullet & \\ & & & & & \bullet & \bullet & \bullet \\ & & & & & & \bullet & \bullet \end{bmatrix} \qquad b = \begin{bmatrix} \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \\ \bullet \end{bmatrix}$$



Wishlist:

- **Parallel scalability** (includes numerical scalability)
- Usability $\rightarrow$ **algebraicity**
- **Generality**
- **Robustness**

13 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
- Accessible through unified Trilinos solver interface Stratimikos

Methoddology

- 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)
- Axel Klawonn (Uni Cologne)
- Oliver Rheinbach (TUBAF)
- Ichitaro Yamazaki (Sandia)

Trilinos Overview

From the report

M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, and A. G. Salinger
An overview of the Trilinos project.
ACM Transactions on Mathematical Software (TOMS) 31.3 (2005): 397-423.

"The Trilinos Project is an effort to facilitate the design, development, integration, and ongoing support of mathematical software libraries within an object-oriented framework for the solution of large-scale, complex multiphysics engineering and scientific problems."

Trilinos is a collection of more than 50 software packages:

- Each Trilinos package is a *self-contained, independent piece of software* with its *own set of requirements, its own development team*[1] *and group of users*.
- However, there are often certain *dependencies between different Trilinos packages*. Some Trilinos packages also *depend on third party libraries*.
- Generally, a *certain degree of interoperability* of the different Trilinos packages is provided.

Why using `Trilinos`?

Wide range of functionality

Data services	Vectors, matrices, graphs and similar data containers, and related operations
Linear and eigen-problem solvers	For large, distributed systems of equations
Nonlinear solvers and analysis tools	Includes basic nonlinear approaches, continuation methods and similar
Discretizations	Tools for the discretization of integral and differential equations
Framework	Tools for building, testing, and integrating Trilinos capabilities

Portable parallelism

`Trilinos` is targeted for all major parallel architectures, including

- distributed-memory using the Message Passing Interface (MPI),
- multicore using a variety of common approaches,
- accelerators using common and emerging approaches, and
- vectorization.

"*. . . as long as a given algorithm and problem size contain enough latent parallelism, **the same Trilinos source code** can be compiled and execution on **any reasonable combination of distributed, multicore, accelerator and vectorizing computing devices**.*" — Trilinos Website

Trilinos Packages

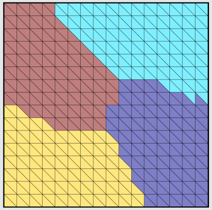
	MPI (Epetra-based)	MPI+X (Tpetra-based)
Linear algebra	Epetra & EpetraExt	Tpetra
Direct sparse solvers	Amesos	Amesos2
Iterative solvers	AztecOO	Belos
Preconditioners:		
• One-level (incomplete) factorization	IFPACK	Ifpack2
• Multigrid	ML	MueLu
• Domain decomposition		ShyLU
Eigenproblem solvers		Anasazi
Nonlinear solvers	NOX & LOCA	
Partitioning	Isorropia & Zoltan	Zoltan2
Example problems	Galeri	
Performance portability		Kokkos & KokkosKernels
Interoperability	Stratimikos & Thyra	
Tools	Teuchos	
⋮	⋮	⋮

More details on https://trilinos.github.io.

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.

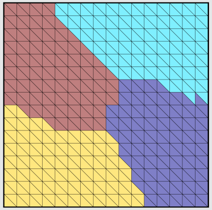


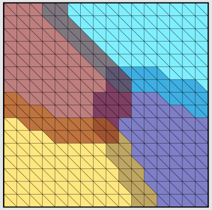
Nonoverlapping DD

Algorithmic Framework for FROSch Overlapping Domain Decompositions

Overlapping domain decomposition

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

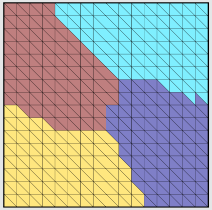

Nonoverlapping DD

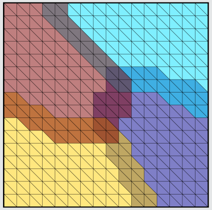

Overlap $\delta = 1h$

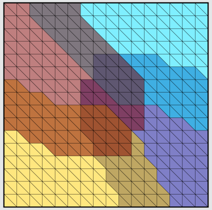
Algorithmic Framework for FROSch Overlapping Domain Decompositions

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

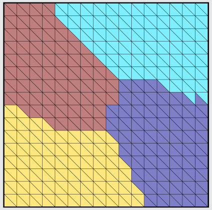

Overlap $\delta = 1h$

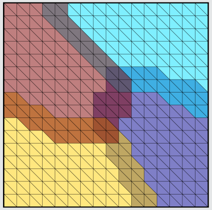

Overlap $\delta = 2h$

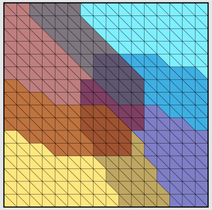
Algorithmic Framework for `FROSch` Overlapping Domain Decompositions

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

$$K_i = R_i K R_i^T$$

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

14 Algorithmic Framework for `FROSch` Coarse Spaces

`FROSch` preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

14 Algorithmic Framework for FROSch Coarse Spaces

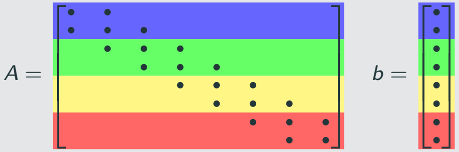
FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

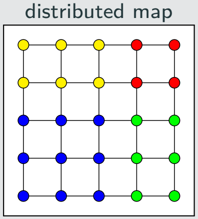
1. Identification of the **domain decomposition interface**

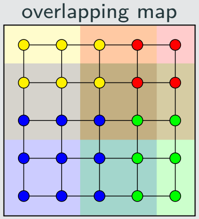
Identification of the domain decomposition interface

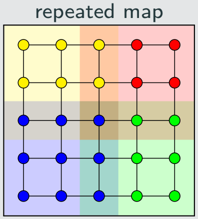
If not provided by the user, FROSch will construct a **repeated map** where the interface (Γ) nodes are shared between processes from the parallel distribution of the matrix rows (**distributed map**).

Then, FROSch automatically identifies vertices, edges, and (in 3D) faces, by the multiplicities of the nodes.









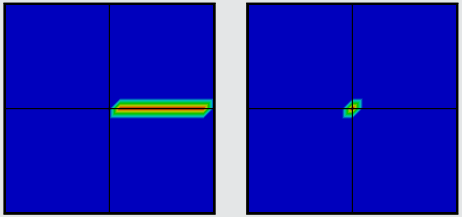
14 Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use algebraic coarse spaces that are constructed in **four algorithmic steps**:

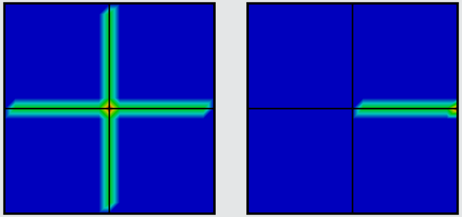
1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface

Construction of a partition of unity on the interface

vertices, edges, and (in 3D) faces



overlapping vertex components

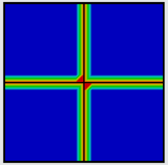


We construct a **partition of unity (POU)** $\{\pi_i\}_i$ with

$$\sum_i \pi_i = 1$$

on the interface Γ.

$\Rightarrow$



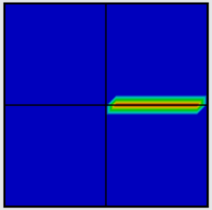
Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use algebraic coarse spaces that are constructed in **four algorithmic steps**:

1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**

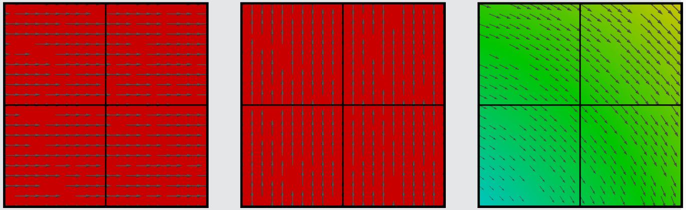
Computation of a coarse basis on the interface

interface POU function



$\times$

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



For each partition of unity function π_i, we compute a basis for the space

$$\operatorname{span}\{\pi_i \times z_j\}_j,$$

where $\{z_j\}_j$ is a null space basis. In case of **linear dependencies**, we perform a **local QR factorization** to construct a basis.

This yields an **interface coarse basis** Φ_Γ.

The linearized rotation

$$\begin{bmatrix} y \\ -x \end{bmatrix}$$

depends on coordinates (geometric information).

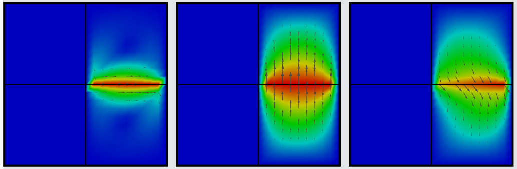
Algorithmic Framework for FROSch Coarse Spaces

FROSch preconditioners use **algebraic coarse spaces** that are constructed in **four algorithmic steps**:

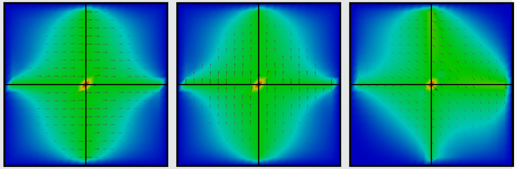
1. Identification of the **domain decomposition interface**
2. Construction of a **partition of unity (POU)** on the interface
3. Computation of a **coarse basis on the interface**
4. Harmonic extensions into the interior to obtain a **coarse basis** on the whole domain

Harmonic extensions into the interior

edge coarse basis functions



vertex component basis functions

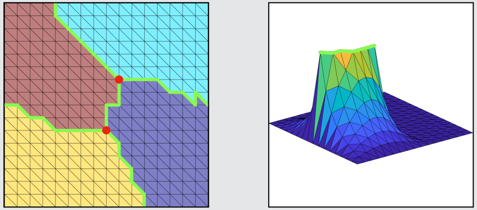


For each **interface coarse basis function**, we compute the interior values Φ_I by computing **harmonic / energy-minimizing extensions**:

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

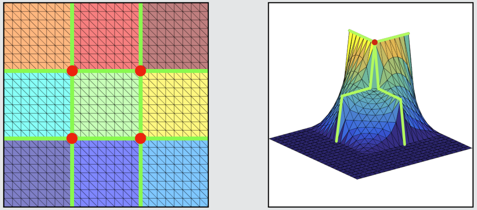
15 Examples of FROSch Coarse Spaces

GDSW (Generalized Dryja–Smith–Widlund)



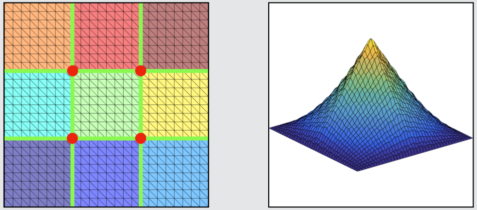
- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

RGDSW (Reduced dimension GDSW)



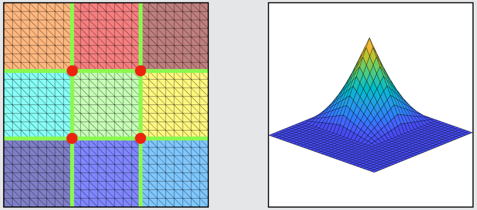
- Dohrmann, Widlund (2017)
- Heinlein, Klawonn, Knepper, Rheinbach (2022)

MsFEM (Multiscale Finite Element Method)



- Hou (1997), Efendiev and Hou (2009)
- Heinlein, Klawonn, Knepper, Rheinbach (2018)

Q1 Lagrangian / piecewise bilinear

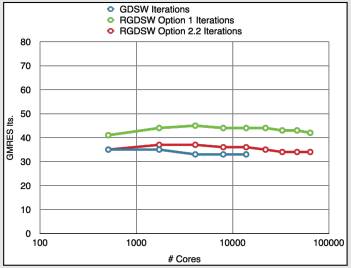


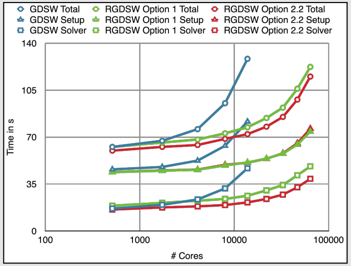
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

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

GDSW vs RGDSW (reduced dimension)

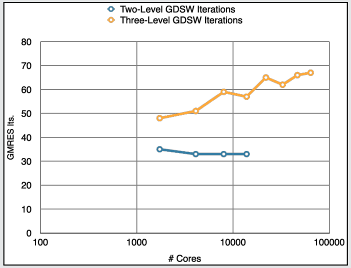
Heinlein, Klawonn, Rheinbach, Widlund (2019).

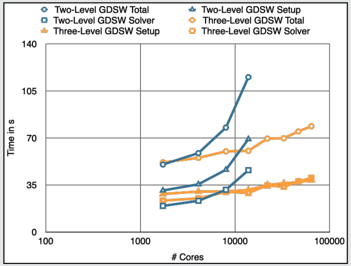




Two-level vs three-level GDSW

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

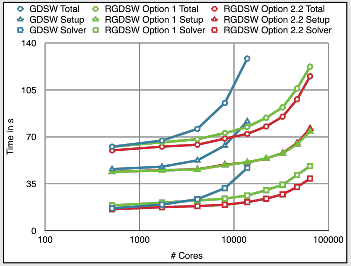




Weak Scalability up to $64\,\mathrm{k}$ MPI Ranks / $1.7\,\mathrm{b}$ Unknowns (3D Poisson; Juqueen)

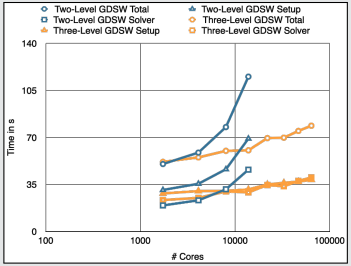
GDSW vs RGDSW (reduced dimension)

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

Algebraic FROSch Preconditioners for Elasticity

$$\begin{aligned} \operatorname{div} \boldsymbol{\sigma} &= (0,-100,0)^T && \text{in } \Omega := [0,1]^3, \\ \boldsymbol{u} &= 0 && \text{on } \partial\Omega_D := \{0\} \times [0,1]^2, \\ \boldsymbol{\sigma} \cdot \boldsymbol{n} &= 0 && \text{on } \partial\Omega_N := \partial\Omega \setminus \partial\Omega_D \end{aligned}$$

St. Venant Kirchhoff material, P2 finite elements, $H/h = 9$; implementation in FEDDLib. (timings: setup + solve = **total**)

prec.	type	#cores	64	512	4 096
GDSW	rotations	#its.	16.3	17.3	19.3
		time	40.1 + 5.9 = **46.0**	55.0 + 8.5 = **63.5**	223.3 + 24.4 = **247.7**
	no rotations	#its.	24.5	29.3	32.3
		time	32.5 + 8.4 = **40.9**	38.4 + 11.8 = **46.7**	102.2 + 20.0 = **122.2**
	fully algebraic	#its.	57.5	74.8	78.0
		time	42.0 + 20.5 = **62.5**	46.0 + 29.9 = **75.9**	124.8 + 50.5 = **175.3**
RGDSW	rotations	#its.	18.8	21.3	19.8
		time	27.8 + 6.4 = **34.2**	31.1 + 8.0 = **39.1**	41.3 + 8.9 = **50.2**
	no rotations	#its.	29.0	32.8	35.5
		time	26.2 + 9.4 = **35.6**	27.3 + 11.8 = **39.1**	31.1 + 14.3 = **45.4**
	fully algebraic	#its.	60.7	78.5	83.0
		time	27.9 + 19.9 = **47.8**	28.7 + 27.9 = **56.6**	34.1 + 33.1 = **67.2**

4 Newton iterations (with backtracking) were necessary for convergence (relative residual reduction of 10^{-8}) for all configurations.

Computations on magnitUDE (University Duisburg-Essen). **Heinlein, Hochmuth, and Klawonn (2021)**

Monolithic (R)GDSW Preconditioners for CFD Simulations

Monolithic GDSW preconditioner

Consider the discrete saddle point problem

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

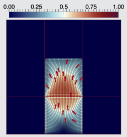
We construct a **monolithic GDSW preconditioner**

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

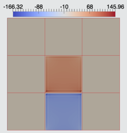
with block matrices $\mathcal{A}_0 = \phi^T \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^T$, and

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

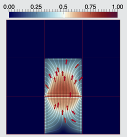
Using $\mathcal{A}$ to compute extensions: $\phi_I = -\mathcal{A}_{II}^{-1} \mathcal{A}_{I\Gamma} \phi_\Gamma$; cf. Heinlein, Hochmuth, Klawonn (2019, 2020).



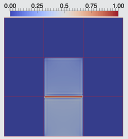
Φ_{u,u_0}



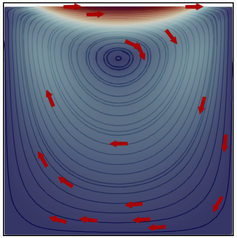
Φ_{p,u_0}

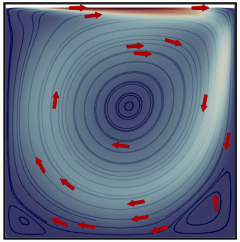


Φ_{u,p_0}



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

Monolithic GDSW preconditioner

Consider the discrete saddle point problem

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

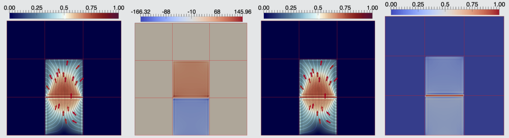
We construct a **monolithic GDSW preconditioner**

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

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

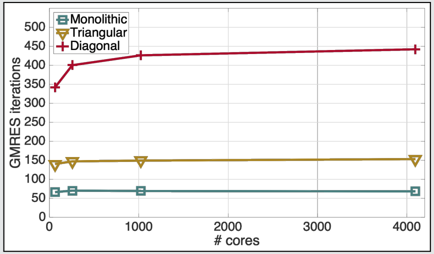
$$\mathcal{R}_i = \begin{bmatrix} \mathcal{R}_{u,i} & 0 \\ 0 & \mathcal{R}_{p,i} \end{bmatrix} \quad \text{and} \quad \phi = \begin{bmatrix} \Phi_{u,u_0} & \Phi_{u,p_0} \\ \Phi_{p,u_0} & \Phi_{p,p_0} \end{bmatrix}.$$

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}

Monolithic vs Block Preconditioners



prec.	MPI ranks	64	256	1 024	4 096
monolithic	time	154.7s	170.0s	175.8s	188.7s
	effic.	**100%**	91%	88%	82%
triangular	time	309.4s	329.1s	359.8s	396.7s
	effic.	50%	47%	43%	39%
diagonal	time	736.7s	859.4s	966.9s	1105.0s
	effic.	21%	18%	16%	14%

Computations performed on magnitUDE, University Duisburg-Essen.

Monolithic (R)GDSW Preconditioners for CFD Simulations

Monolithic GDSW preconditioner

Consider the discrete saddle point problem

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

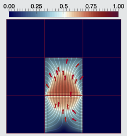
We construct a **monolithic GDSW preconditioner**

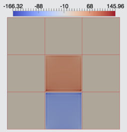
$$\mathcal{M}_{\mathrm{GDSW}}^{-1} = \phi \mathcal{A}_0^{-1} \phi^T + \sum\nolimits_{i=1}^{N} \mathcal{R}_i^T \mathcal{A}_i^{-1} \mathcal{R}_i,$$

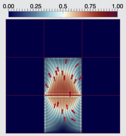
with block matrices $\mathcal{A}_0 = \phi^T \mathcal{A} \phi$, $\mathcal{A}_i = \mathcal{R}_i \mathcal{A} \mathcal{R}_i^T$, and

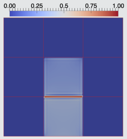
$$\mathcal{R}_i = \begin{bmatrix} \mathcal{R}_{u,i} & 0 \\ 0 & \mathcal{R}_{p,i} \end{bmatrix} \quad \text{and} \quad \phi = \begin{bmatrix} \Phi_{u,u_0} & \Phi_{u,p_0} \\ \Phi_{p,u_0} & \Phi_{p,p_0} \end{bmatrix}.$$

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

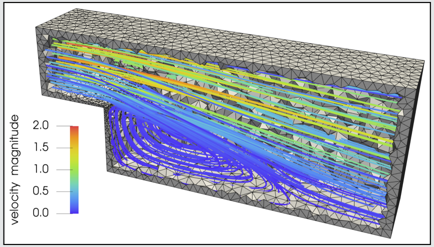

Φ_{u,u_0}


Φ_{p,u_0}


Φ_{u,p_0}


Φ_{p,p_0}

Monolithic vs SIMPLE



Steady-state Navier-Stokes equations

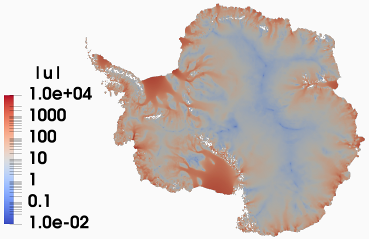
prec.	MPI ranks	243	1 125	15 562
Monolithic RGDSW (`FROSch`)	setup	39.6 s	57.9 s	95.5 s
	solve	57.6 s	69.2 s	74.9 s
	total	**97.2 s**	**127.7 s**	**170.4 s**
SIMPLE RGDSW (`Teko` & `FROSch`)	setup	39.2 s	38.2 s	68.6 s
	solve	86.2 s	106.6 s	127.4 s
	total	125.4 s	144.8 s	196.0 s

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

FROSch Preconditioners for Land Ice Simulations

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)**.

	Antarctica (**velocity**) 4 km resolution, 20 layers, 35 m dofs			Greenland (**multiphysics vel. & temperature**) 1-10 km resolution, 20 layers, 69 m dofs		
MPI ranks	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 (2022)**

Inexact Subdomain Solvers in FROSch

3D Laplacian; 512 MPI ranks $= 512$ $(= 8 \times 8 \times 8)$ subdomains; $H/\delta = 10$; RGDSW coarse space.

		subdomain solver						
		direct solver	ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**26**	33	30	31	28	34	31
	setup time	1.89 s	0.97 s	1.01 s	0.89 s	0.91 s	**0.73 s**	**0.71 s**
	apply time	0.39 s	**0.27 s**	0.31 s	0.31 s	0.35 s	0.30 s	0.30 s
	prec. time	2.28 s	1.24 s	1.32 s	1.20 s	1.26 s	1.03 s	**1.01 s**
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**30**	55	46	52	41	59	51
	setup time	12.09 s	6.14 s	6.26 s	5.74 s	5.89 s	**5.55 s**	5.64 s
	apply time	4.21 s	**1.84 s**	1.96 s	2.66 s	3.28 s	2.52 s	2.47 s
	prec. time	16.30 s	**7.98 s**	8.22 s	8.40 s	9.18 s	8.16 s	8.11 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	OOM	81	64	76	**56**	88	74
	setup time	-	47.29 s	47.87 s	45.14 s	**45.08 s**	45.44 s	45.49 s
	apply time	-	10.79 s	**9.98 s**	13.00 s	16.16 s	11.95 s	12.09 s
	prec. time	-	58.08 s	57.85 s	58.15 s	61.25 s	**57.39 s**	57.59 s

Intel MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from Ifpack2.

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

Inexact Subdomain Solvers in FROSch

3D Laplacian; 512 MPI ranks $= 512$ ($= 8 \times 8 \times 8$) subdomains; $H/\delta = 10$; RGDSW coarse space.

		subdomain solver						
		direct solver	ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**26**	33	30	31	28	34	31
	setup time	1.89 s	0.97 s	1.01 s	0.89 s	0.91 s	**0.73 s**	**0.71 s**
	apply time	0.39 s	**0.27 s**	0.31 s	0.31 s	0.35 s	0.30 s	0.30 s
	prec. time	2.28 s	1.24 s	1.32 s	1.20 s	1.26 s	1.03 s	**1.01 s**
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**30**	55	46	52	41	59	51
	setup time	12.09 s	6.14 s	6.26 s	5.74 s	5.89 s	**5.55 s**	5.64 s
	apply time	4.21 s	**1.84 s**	1.96 s	2.66 s	3.28 s	2.52 s	2.47 s
	prec. time	16.30 s	**7.98 s**	8.22 s	8.40 s	9.18 s	8.16 s	8.11 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	OOM	81	64	76	**56**	88	74
	setup time	-	47.29 s	47.87 s	45.14 s	**45.08 s**	45.44 s	45.49 s
	apply time	-	10.79 s	**9.98 s**	13.00 s	16.16 s	11.95 s	12.09 s
	prec. time	-	58.08 s	57.85 s	58.15 s	61.25 s	**57.39 s**	57.59 s

Intel MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from `Ifpack2`.

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

Inexact Extension Solvers in FROSch

3D Laplacian; 512 MPI ranks $= 512$ $(= 8 \times 8 \times 8)$ subdomains; $H/\delta = 10$; RGDSW coarse space.

extension solver (10 Gauss–Seidel sweeps for the subdomain solver)		direct solver	**preconditioned GMRES** (rel. tol. $= 10^{-4}$)					
			ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**28**	**28**	**28**	**28**	**28**	**28**	**28**
	setup time	0.89 s	0.93 s	0.89 s	**0.78 s**	0.83 s	0.79 s	0.84 s
	apply time	0.35 s	0.35 s	**0.34 s**	0.36 s	**0.34 s**	0.35 s	**0.34 s**
	prec. time	1.23 s	1.28 s	1.23 s	**1.14 s**	1.17 s	**1.14 s**	1.18 s
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**41**	**41**	**41**	**41**	**41**	**41**	**41**
	setup time	5.72 s	**4.16 s**	4.61 s	4.26 s	4.64 s	4.27 s	4.33 s
	apply time	3.33 s	3.33 s	3.30 s	3.33 s	3.30 s	**3.28 s**	3.29 s
	prec. time	9.04 s	**7.49 s**	7.92 s	7.59 s	7.95 s	7.55 s	7.62 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	**56**	**56**	**56**	**56**	**56**	**56**	**56**
	setup time	45.16 s	**17.75 s**	18.16 s	17.98 s	19.34 s	17.93 s	18.04 s
	apply time	**15.83 s**	18.04 s	17.08 s	16.26 s	15.81 s	16.19 s	16.44 s
	prec. time	60.99 s	35.79 s	35.25 s	34.24 s	35.15 s	**34.12 s**	34.49 s

Intel MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from `Ifpack2`.

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

Inexact Extension Solvers in FROSch

3D Laplacian; 512 MPI ranks = 512 (= $8 \times 8 \times 8$) subdomains; $H/\delta = 10$; RGDSW coarse space.

extension solver (10 Gauss–Seidel sweeps for the subdomain solver)		direct solver	**preconditioned GMRES** (rel. tol. = 10^{-4})					
			ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\ k$ dofs per rank	iter	**28**	**28**	**28**	**28**	**28**	**28**	**28**
	setup time	0.89 s	0.93 s	0.89 s	**0.78 s**	0.83 s	0.79 s	0.84 s
	apply time	0.35 s	0.35 s	**0.34 s**	0.36 s	**0.34 s**	0.35 s	**0.34 s**
	prec. time	1.23 s	1.28 s	1.23 s	**1.14 s**	1.17 s	**1.14 s**	1.18 s
$H/h = 40$, $\approx 105\ k$ dofs per rank	iter	**41**	**41**	**41**	**41**	**41**	**41**	**41**
	setup time	5.72 s	**4.16 s**	4.61 s	4.26 s	4.64 s	4.27 s	4.33 s
	apply time	3.33 s	3.33 s	3.30 s	3.33 s	3.30 s	**3.28 s**	3.29 s
	prec. time	9.04 s	**7.49 s**	7.92 s	7.59 s	7.95 s	7.55 s	7.62 s
$H/h = 60$, $\approx 350\ k$ dofs per rank	iter	**56**	**56**	**56**	**56**	**56**	**56**	**56**
	setup time	45.16 s	**17.75 s**	18.16 s	17.98 s	19.34 s	17.93 s	18.04 s
	apply time	**15.83 s**	18.04 s	17.08 s	16.26 s	15.81 s	16.19 s	16.44 s
	prec. time	60.99 s	35.79 s	35.25 s	34.24 s	35.15 s	**34.12 s**	34.49 s

Intel MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from `Ifpack2`.

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

Inexact Extension Solvers in FROSch

3D Laplacian; 512 MPI ranks $= 512$ $(= 8 \times 8 \times 8)$ subdomains; $H/\delta = 10$; RGDSW coarse space.

extension solver (10 Gauss–Seidel sweeps for the subdomain solver)		direct solver	**preconditioned GMRES** (rel. tol. $= 10^{-4}$)					
			ILU(k)		symm. Gauß–Seidel		Chebyshev polyn.	
			k = 2	k = 3	5 sweeps	10 sweeps	p = 6	p = 8
$H/h = 20$, $\approx 14\,k$ dofs per rank	iter	**28**	**28**	**28**	**28**	**28**	**28**	**28**
	setup time	0.89 s	0.93 s	0.89 s	**0.78 s**	0.83 s	0.79 s	0.84 s
	apply time	0.35 s	0.35 s	**0.34 s**	0.36 s	**0.34 s**	0.35 s	**0.34 s**
	prec. time	1.23 s	1.28 s	1.23 s	**1.14 s**	1.17 s	**1.14 s**	1.18 s
$H/h = 40$, $\approx 105\,k$ dofs per rank	iter	**41**	**41**	**41**	**41**	**41**	**41**	**41**
	setup time	5.72 s	**4.16 s**	4.61 s	4.26 s	4.64 s	4.27 s	4.33 s
	apply time	3.33 s	3.33 s	3.30 s	3.33 s	3.30 s	**3.28 s**	3.29 s
	prec. time	9.04 s	**7.49 s**	7.92 s	7.59 s	7.95 s	7.55 s	7.62 s
$H/h = 60$, $\approx 350\,k$ dofs per rank	iter	**56**	**56**	**56**	**56**	**56**	**56**	**56**
	setup time	45.16 s	**17.75 s**	18.16 s	17.98 s	19.34 s	17.93 s	18.04 s
	apply time	**15.83 s**	18.04 s	17.08 s	16.26 s	15.81 s	16.19 s	16.44 s
	prec. time	60.99 s	35.79 s	35.25 s	34.24 s	35.15 s	**34.12 s**	34.49 s

Intel MKL PARDISO; ILU / symmetric Gauß–Seidel / Chebyshev polynomials from Ifpack2.

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

$\Rightarrow$ The use of inexact subdomain solvers may **significantly improve the time to solution**, in particular, for large subdomain problems.

Part IV – Exercises – Parallel Preconditioning with FROSch

17 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 be enough time to set up the software now and still work on the exercises**.

18 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**.

19 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**.

Thank you for your attention!

Questions?