Advanced Domain Decomposition Methods

Parallel Schwarz Preconditioning and an Introduction to FROSch

Alexander Heinlein[1]

DCSE Summerschool: Numerical Linear Algebra on High Performance Computers, TU Delft, June 5-9, 2023

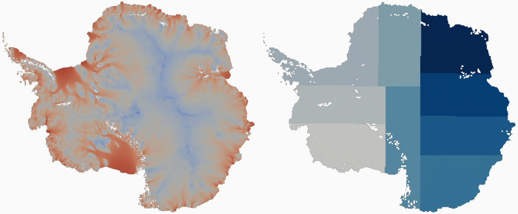
[1]TU Delft

Contents

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

Domain Decomposition Methods



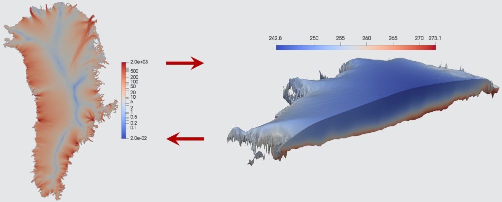
Graphics based on Heinlein, Perego, Rajamanickam (2022)

Idea

Decomposition of a large **global problem** into smaller **local problems**.

Parallel solvers

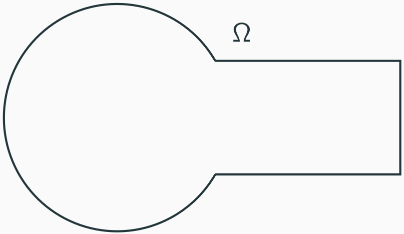
Coupled problems



velocity temperature

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**.



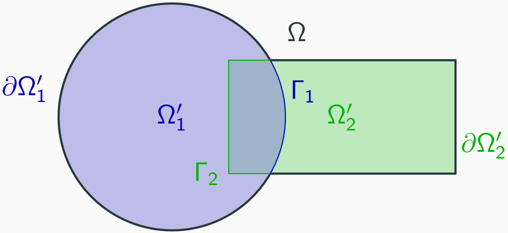
Classical "doorknob" geometry

Overlapping domain decomposition

We solve the Poisson equation

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

on the classical "doorknob" geometry.



The **alternating Schwarz iteration** corresponds to **solving alternatingly solving the local problems**

$$(D_1)\begin{cases} -\Delta u^{n+1/2} = f & \text{in } \Omega_1', \\ u^{n+1/2} = u^n & \text{auf } \Gamma_1 \\ u^{n+1/2} = u^n & \text{on } \Omega \setminus \overline{\Omega_1'} \end{cases}$$

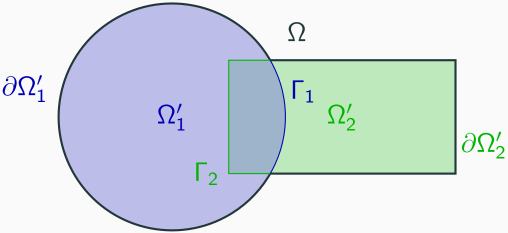
$$(D_2)\begin{cases} -u^{n+1''} = f & \text{in } \Omega_2', \\ u^{n+1} = u^{n+1/2} & \text{auf } \Gamma_2 \\ u^{n+1} = u^{n+1/2} & \text{on } \Omega \setminus \overline{\Omega_2'} \end{cases}$$

Overlapping domain decomposition

We solve the Poisson equation

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

on the classical "doorknob" geometry.



The **alternating Schwarz iteration** corresponds to **solving alternatingly solving the local problems**

$$(D_1)\begin{cases} -\Delta u^{n+1/2} = f & \text{in } \Omega_1', \\ u^{n+1/2} = u^n & \text{auf } \Gamma_1 \\ u^{n+1/2} = u^n & \text{on } \Omega \setminus \overline{\Omega_1'} \end{cases}$$

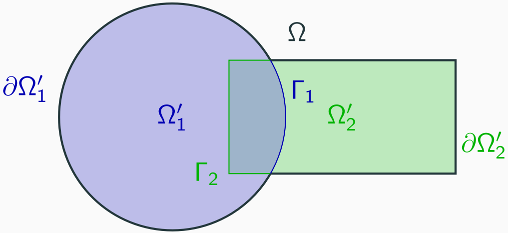
$$(D_2)\begin{cases} -u^{n+1''} = f & \text{in } \Omega_2', \\ u^{n+1} = u^{n+1/2} & \text{auf } \Gamma_2 \\ u^{n+1} = u^{n+1/2} & \text{on } \Omega \setminus \overline{\Omega_2'} \end{cases}$$

Overlapping domain decomposition

We solve the Poisson equation

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

on the classical "doorknob" geometry.



The **alternating Schwarz iteration** corresponds to **solving alternatingly solving the local problems**

$$(D_1)\begin{cases} -\Delta u^{n+1/2} = f & \text{in } \Omega'_1, \\ u^{n+1/2} = u^n & \text{auf } \Gamma_1 \\ u^{n+1/2} = u^n & \text{on } \Omega \setminus \overline{\Omega'_1} \end{cases}$$

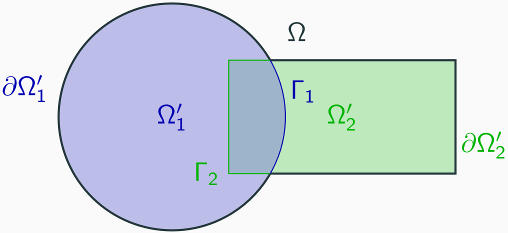
$$(D_2)\begin{cases} -u^{n+1''} = f & \text{in } \Omega'_2, \\ u^{n+1} = u^{n+1/2} & \text{auf } \Gamma_2 \\ u^{n+1} = u^{n+1/2} & \text{on } \Omega \setminus \overline{\Omega'_2} \end{cases}$$

Overlapping domain decomposition

We solve the Poisson equation

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

on the classical "doorknob" geometry.

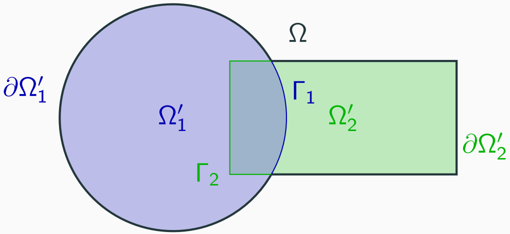


The **alternating Schwarz iteration** corresponds to **solving alternatingly solving the local problems**

$$
(D_1)\begin{cases}
-\Delta u^{n+1/2} = f & \text{in } \Omega_1',\\
u^{n+1/2} = u^n & \text{auf } \Gamma_1\\
u^{n+1/2} = u^n & \text{on } \Omega \setminus \overline{\Omega_1'}
\end{cases}
$$

$$
(D_2)\begin{cases}
-u^{n+1''} = f & \text{in } \Omega_2',\\
u^{n+1} = u^{n+1/2} & \text{auf } \Gamma_2\\
u^{n+1} = u^{n+1/2} & \text{on } \Omega \setminus \overline{\Omega_2'}
\end{cases}
$$

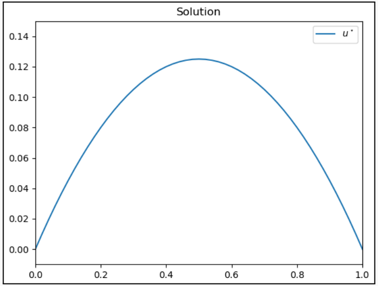
For the sake of simplicity, instead of the two-dimensional geometry,



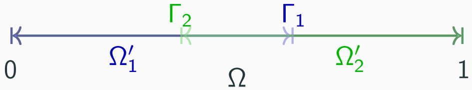
we consider the **one-dimensional Poisson equation**

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

Solution: $u(x) = -\frac{1}{2}x(x-1).$



Domain decomposition:



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**:

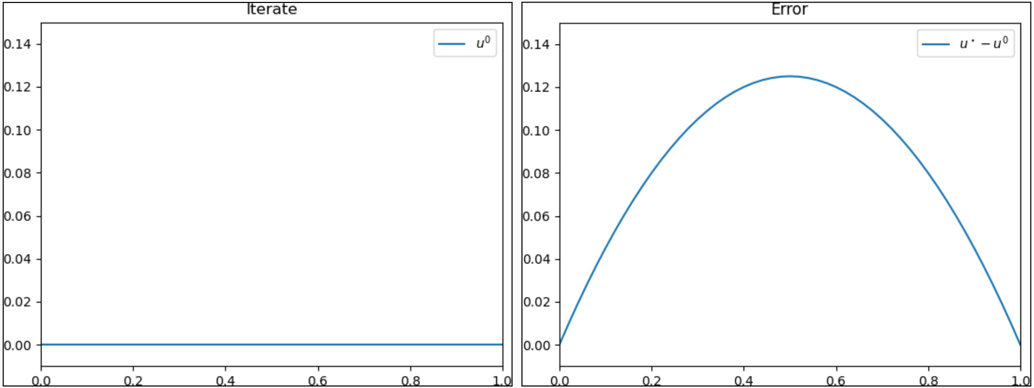


Figure 1: Iterate (left) and error (right) in iteration 0.

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**:

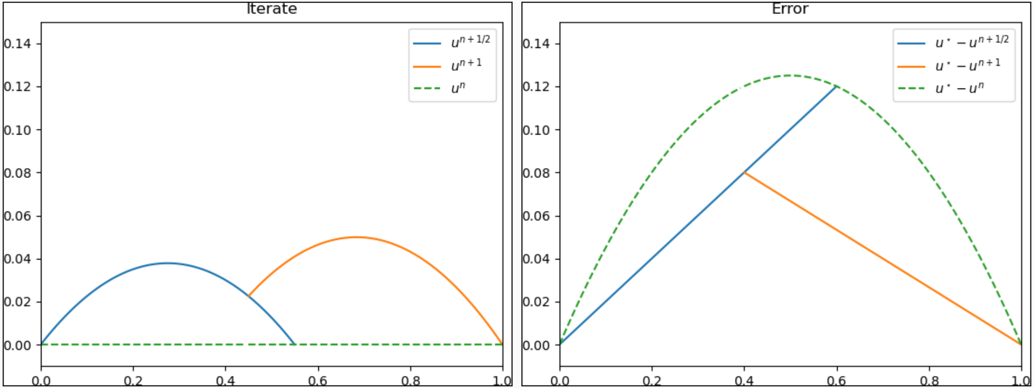


Figure 1: Iterate (left) and error (right) in iteration 1.

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**:

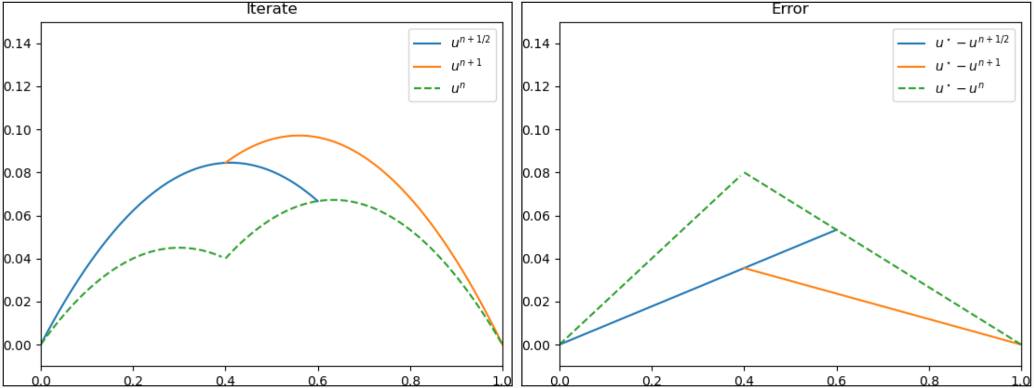


Figure 1: Iterate (left) and error (right) in iteration 2.

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**:

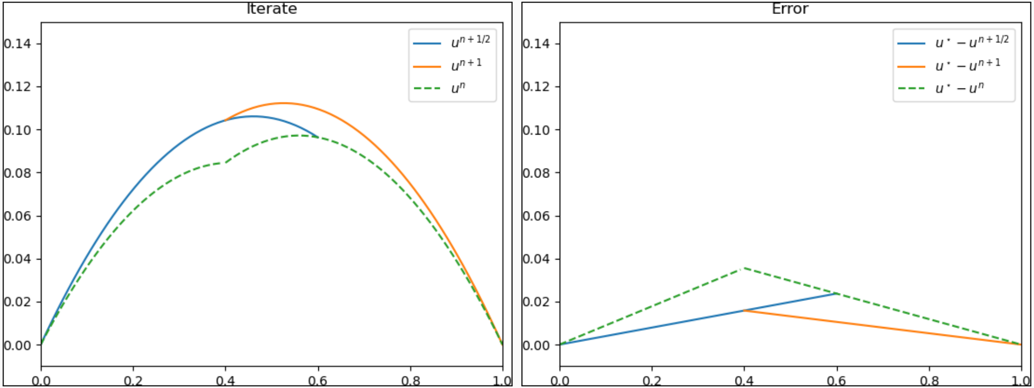


Figure 1: Iterate (left) and error (right) in iteration 3.

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**:

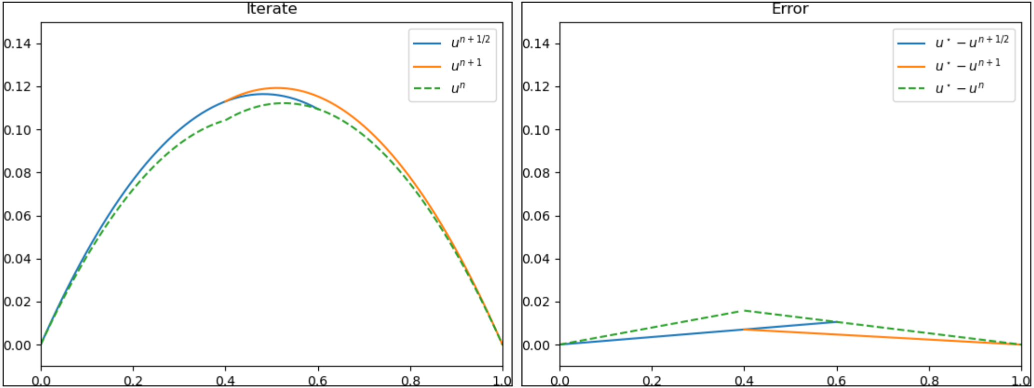


Figure 1: Iterate (left) and error (right) in iteration 4.

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**:

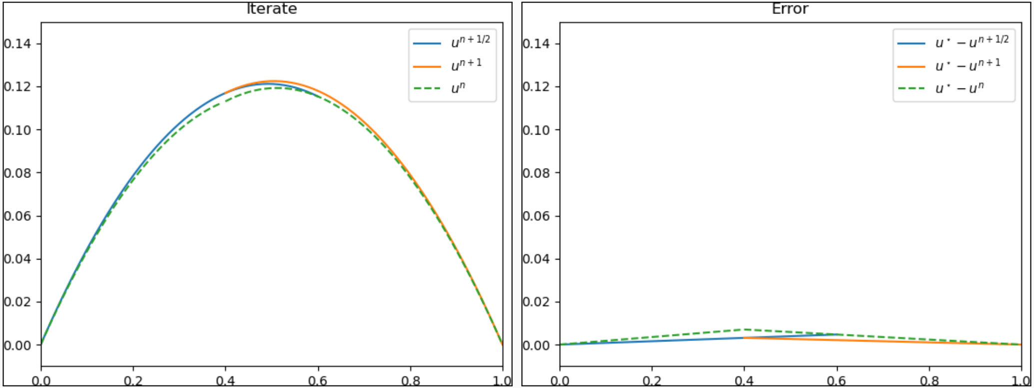


Figure 1: Iterate (left) and error (right) in iteration 5.

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

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

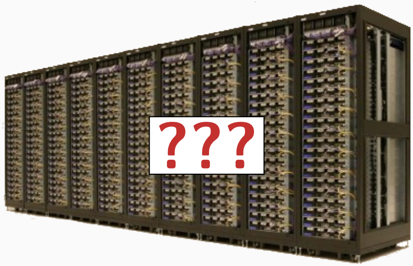
$$(D_2)\begin{cases} -\Delta u^{n+1} = f & \text{in } \Omega_2, \\ u^{n+1} = \mathbf{u^{n+1/2}} & \text{on } \partial\Omega_2' \\ u^{n+1} = \mathbf{u^{n+1/2}} & \text{on } \Omega \setminus \overline{\Omega_2'} \end{cases}$$

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**.

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

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

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



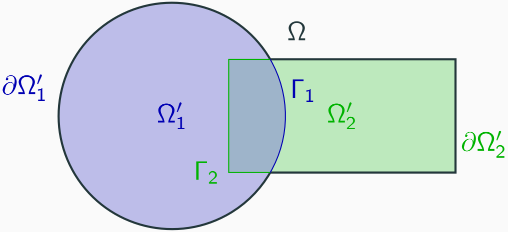
Idea: For all red terms, we **use the values from the previous iteration**. Then, the both Dirichlet problem **can be solved at the same time**.

3 The Parallel Schwarz Algorithm

The **parallel Schwarz algorithm** has been introduced by **Lions (1988)**. Here, we solve the local problems

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

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



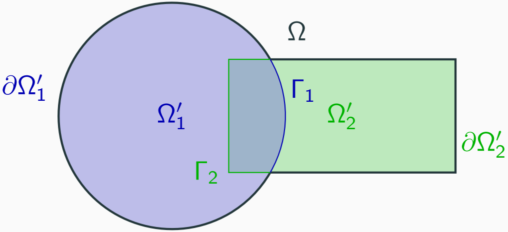
Since u_1^n and u_2^n are both computed in the previous iteration, the problems can be solved independent of each other.

3 The Parallel Schwarz Algorithm

The **parallel Schwarz algorithm** has been introduced by **Lions (1988)**. Here, we solve the local problems

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

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



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

This method is suitable for **parallel computing**!

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 the **parallel Schwarz iteration**:

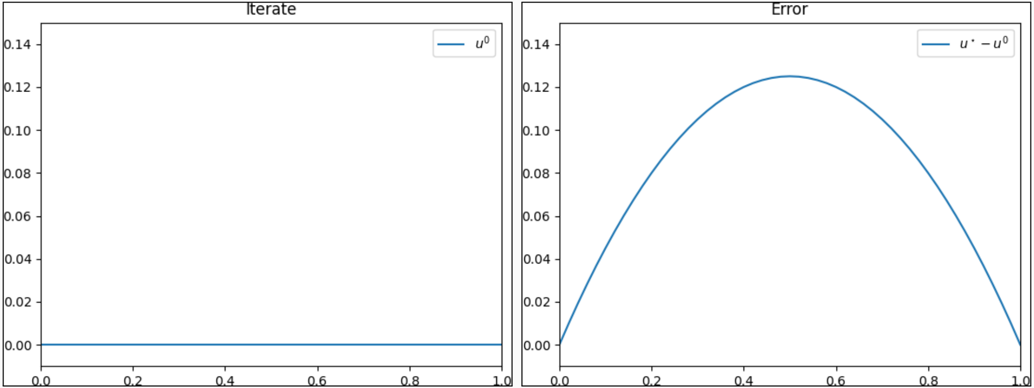


Figure 2: Iterate (left) and error (right) in iteration 0.

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 the **parallel Schwarz iteration**:

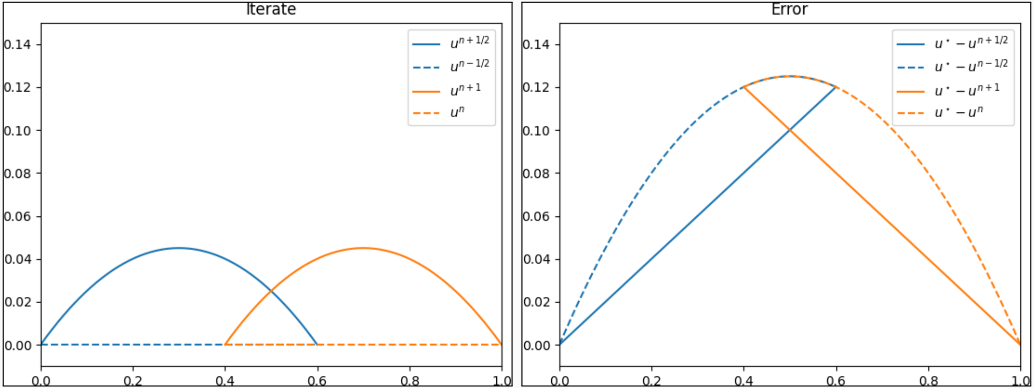


Figure 2: Iterate (left) and error (right) in iteration 1.

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 the **parallel Schwarz iteration**:

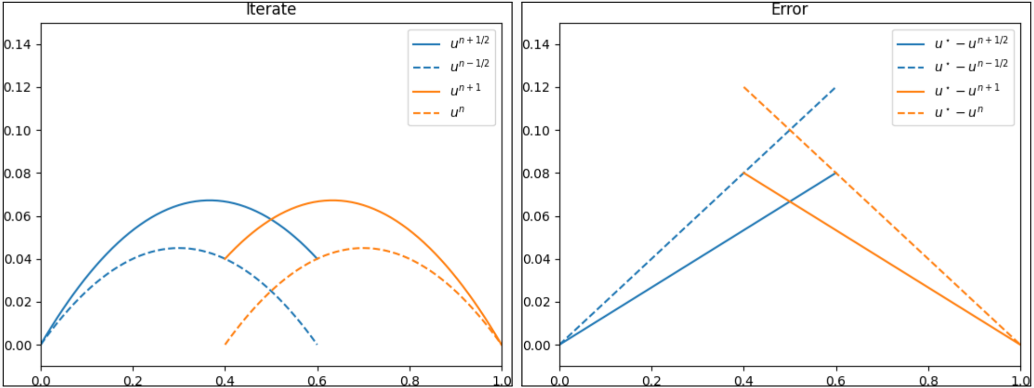


Figure 2: Iterate (left) and error (right) in iteration 2.

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 the **parallel Schwarz iteration**:

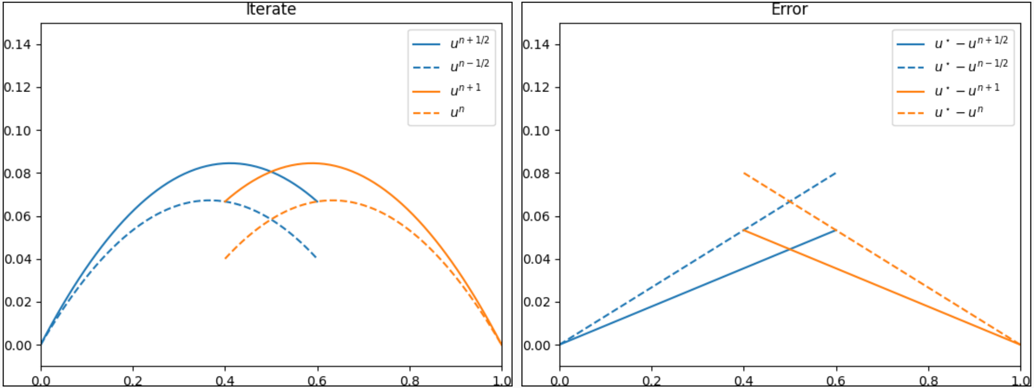


Figure 2: Iterate (left) and error (right) in iteration 3.

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 the **parallel Schwarz iteration**:

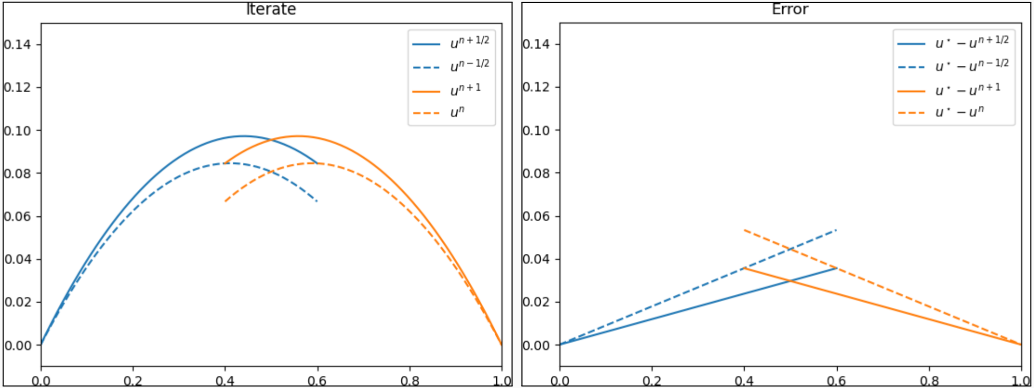


Figure 2: Iterate (left) and error (right) in iteration 4.

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 the **parallel Schwarz iteration**:

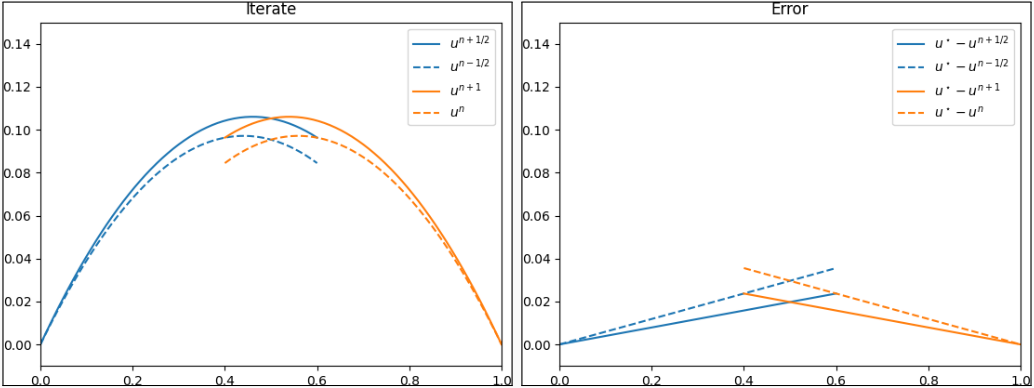


Figure 2: Iterate (left) and error (right) in iteration 5.

4 Comparison of the two Methods

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

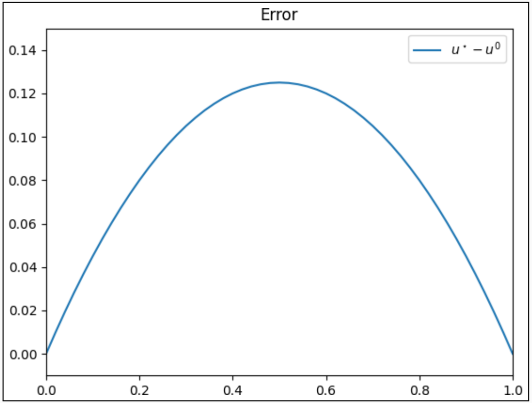


Figure 3: Error in iteration 0.

Parallel Schwarz iteration

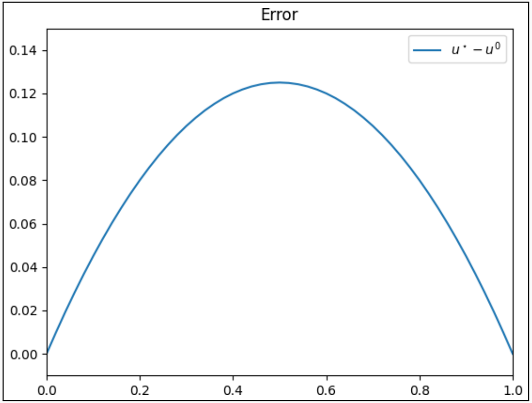


Figure 4: Error in iteration 0.

4 Comparison of the two Methods

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

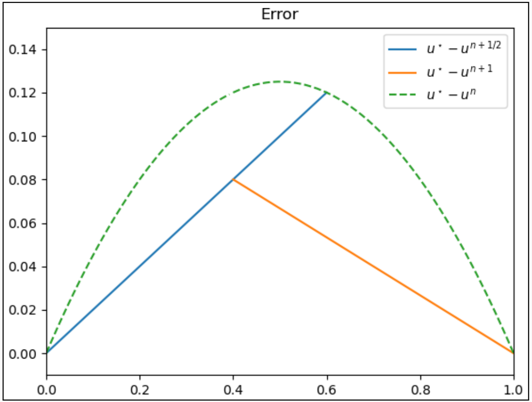


Figure 3: Error in iteration 1.

Parallel Schwarz iteration

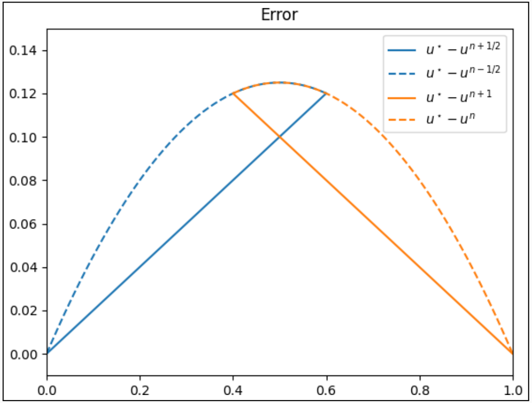


Figure 4: Error in iteration 1.

4 Comparison of the two Methods

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

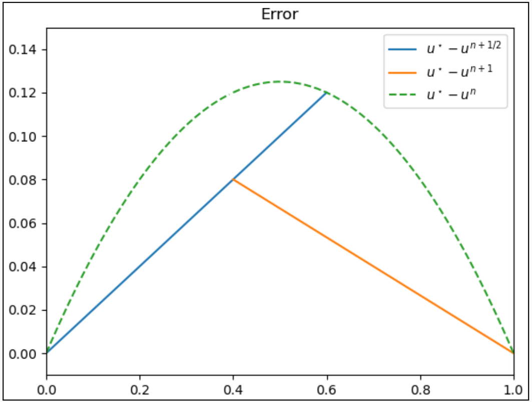


Figure 3: Error in iteration 1.

Parallel Schwarz iteration

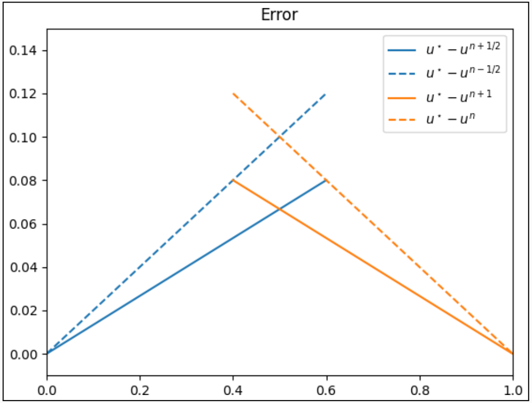


Figure 4: Error in iteration 2.

4 Comparison of the two Methods

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

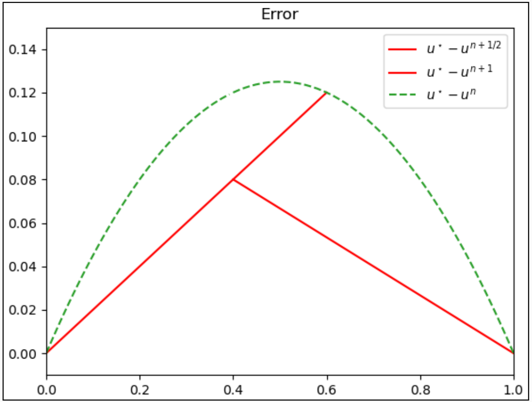


Figure 3: Error in **iteration 1**.

Parallel Schwarz iteration

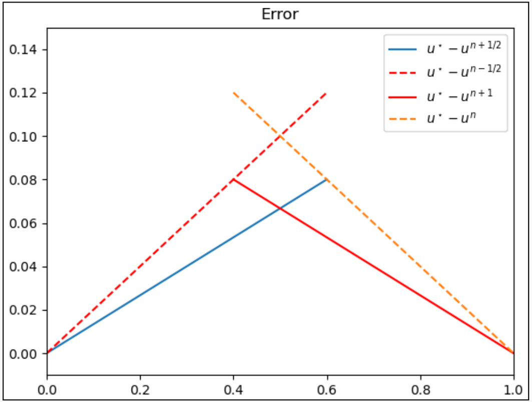


Figure 4: Error in **iteration 2**.

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

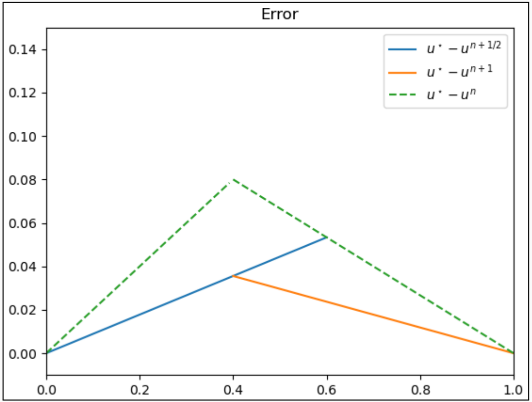


Figure 3: Error in iteration 2.

Parallel Schwarz iteration

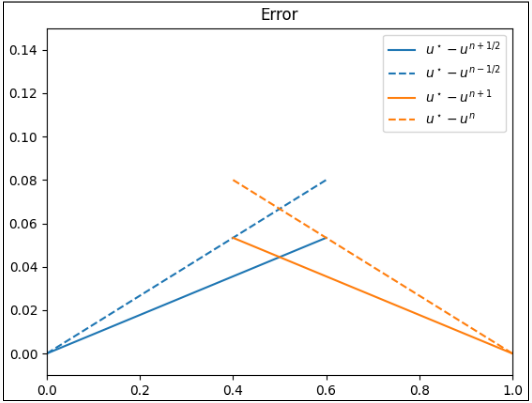


Figure 4: Error in iteration 3.

4 Comparison of the two Methods

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

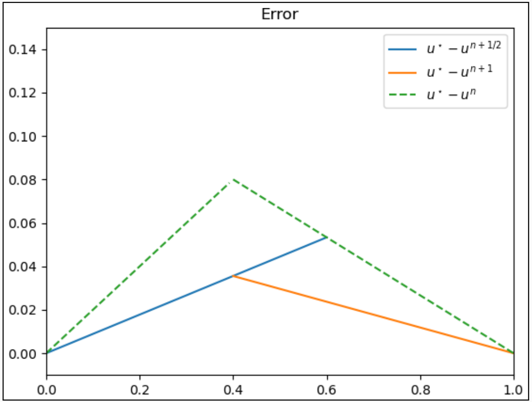


Figure 3: Error in iteration 2.

Parallel Schwarz iteration

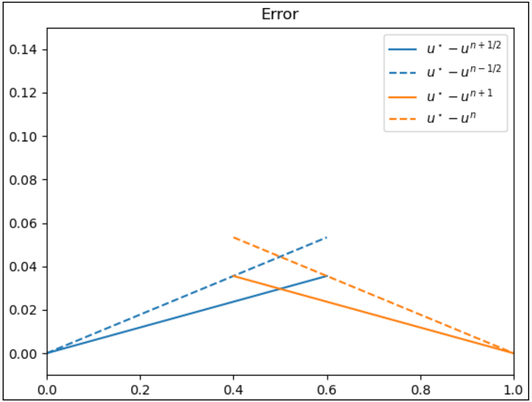


Figure 4: Error in iteration 4.

4 Comparison of the two Methods

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

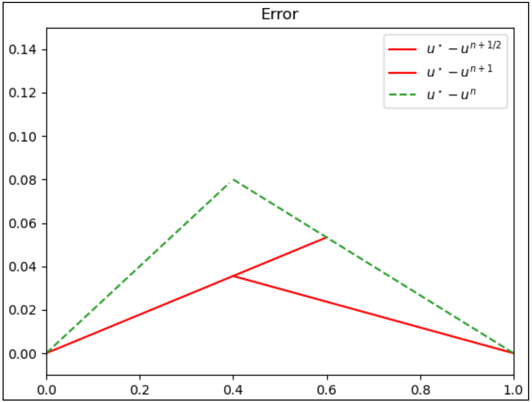


Figure 3: Error in **iteration 2**.

Parallel Schwarz iteration

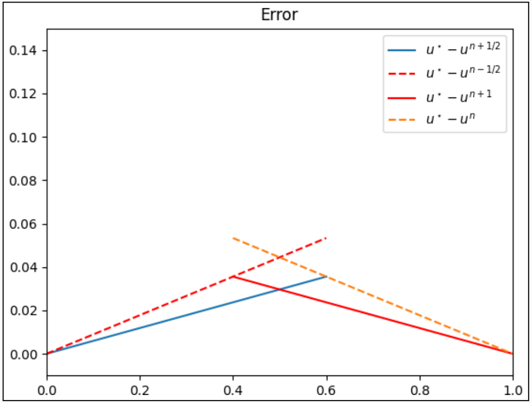


Figure 4: Error in **iteration 4**.

Next, we compare the convergence of the two methods using the error plots:

Alternating Schwarz iteration

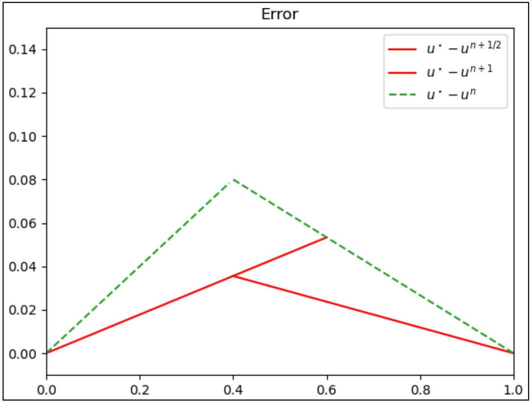


Figure 3: Error in **iteration 2**.

Parallel Schwarz iteration

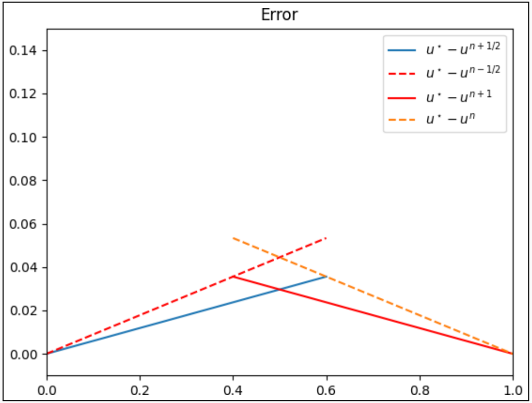
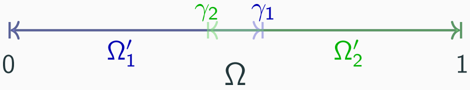


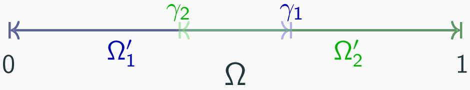
Figure 4: Error in **iteration 4**.

The alternating Schwarz method **converges twice as fast** as the parallel Schwarz method.
However, the **local solutions** have to be computed **sequentially**.

We investigate the convergence of the methods (using the alternating method as an example) depending on the **size of the overlap**:



Overlap 0.05



Overlap 0.1

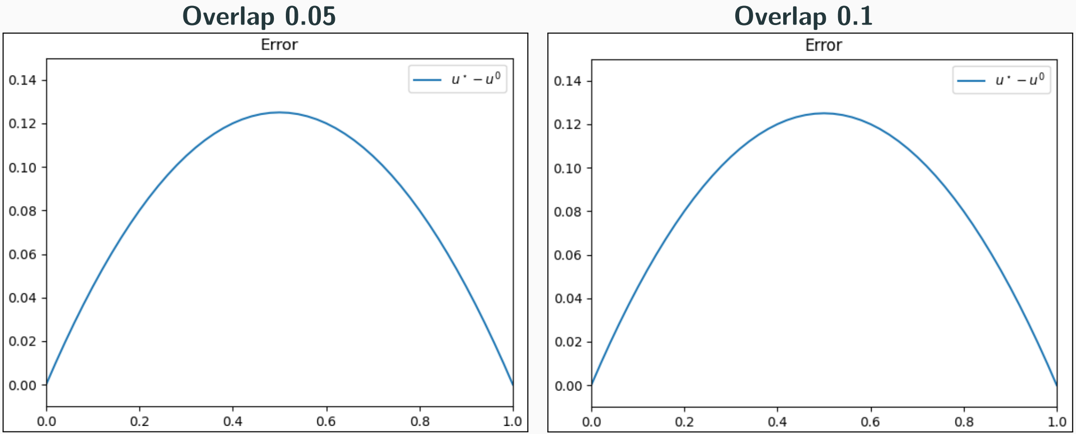


Figure 5: Error in iteration 0.

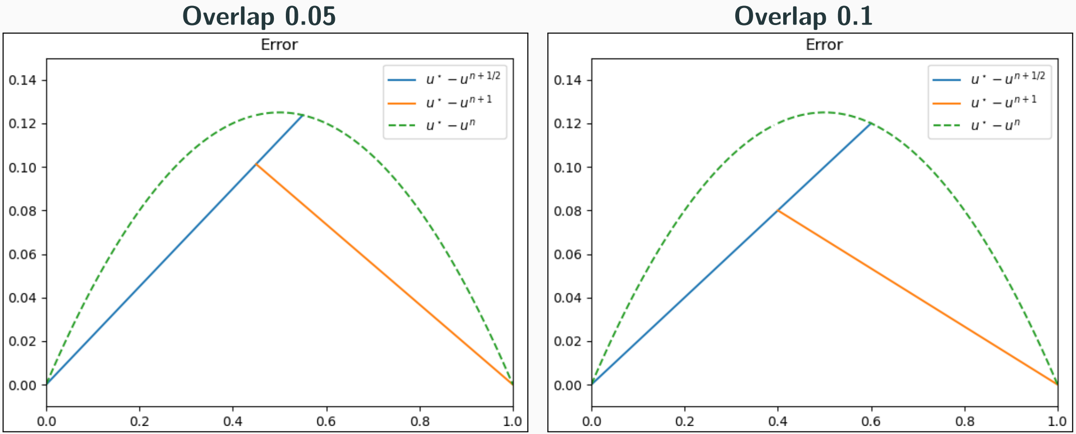
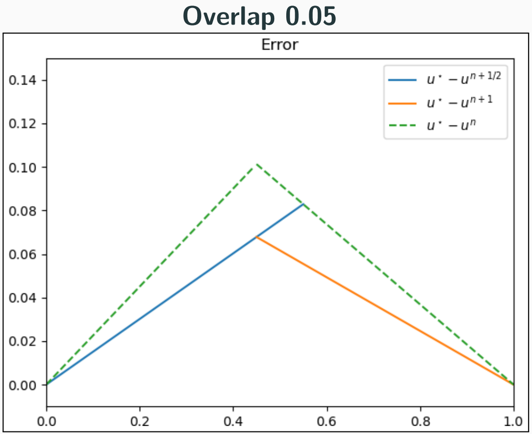


Figure 5: Error in iteration 1.



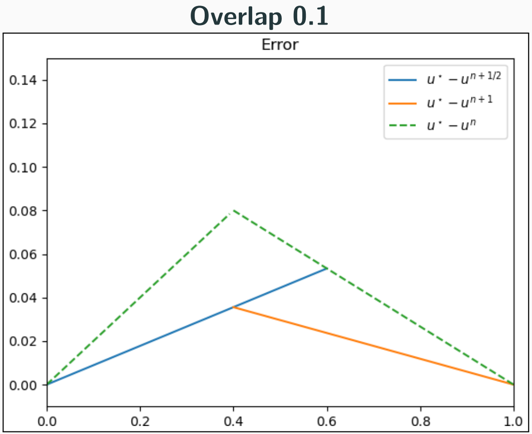


Figure 5: Error in iteration 2.

5 Effect of the Size of the Overlap

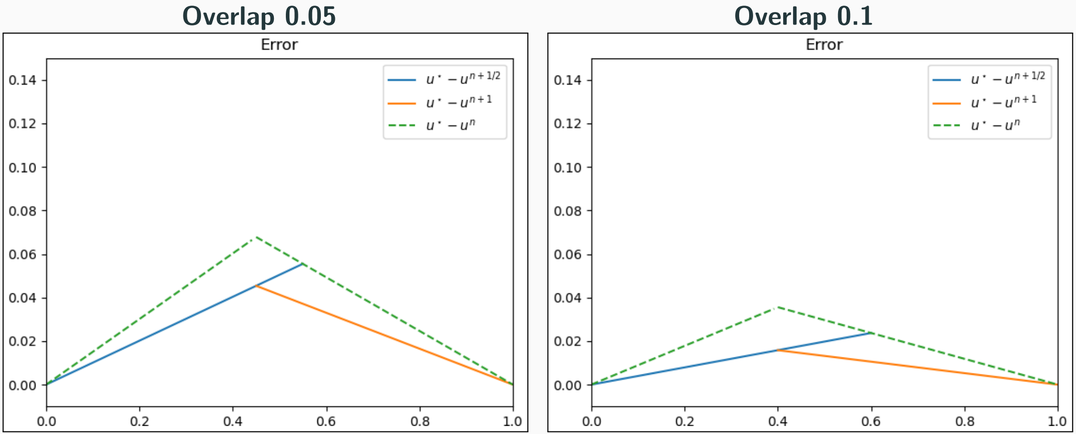
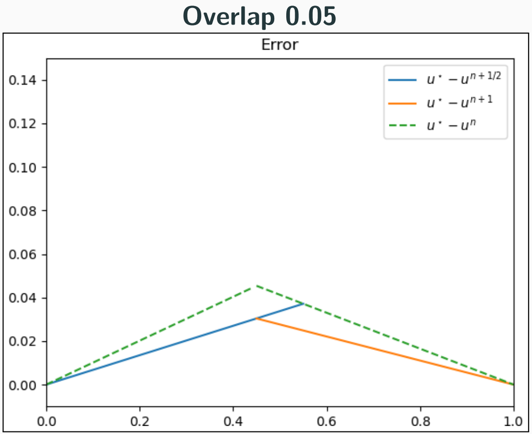


Figure 5: Error in iteration 3.



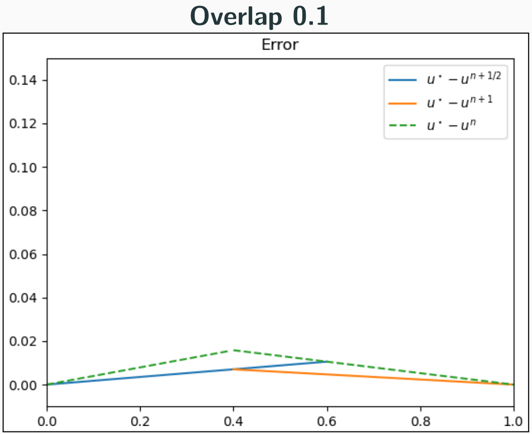


Figure 5: Error in iteration 4.

5 Effect of the Size of the Overlap

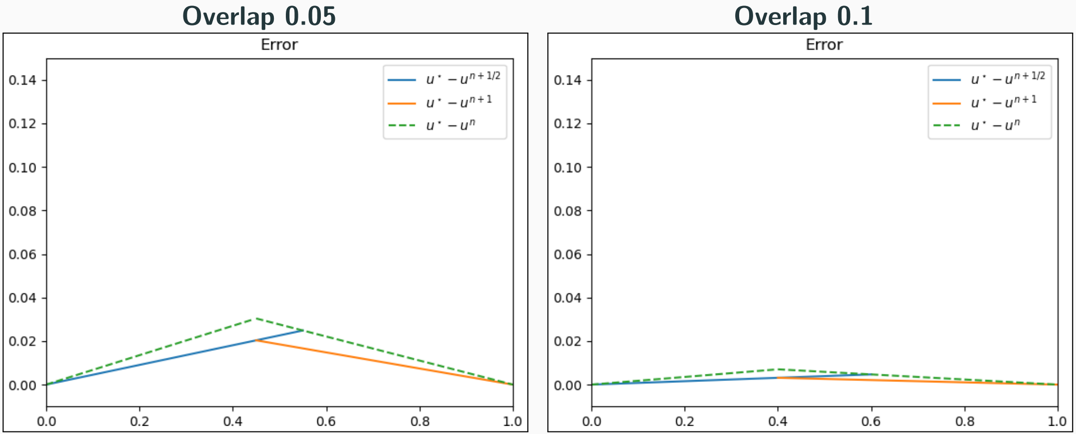


Figure 5: Error in iteration 5.

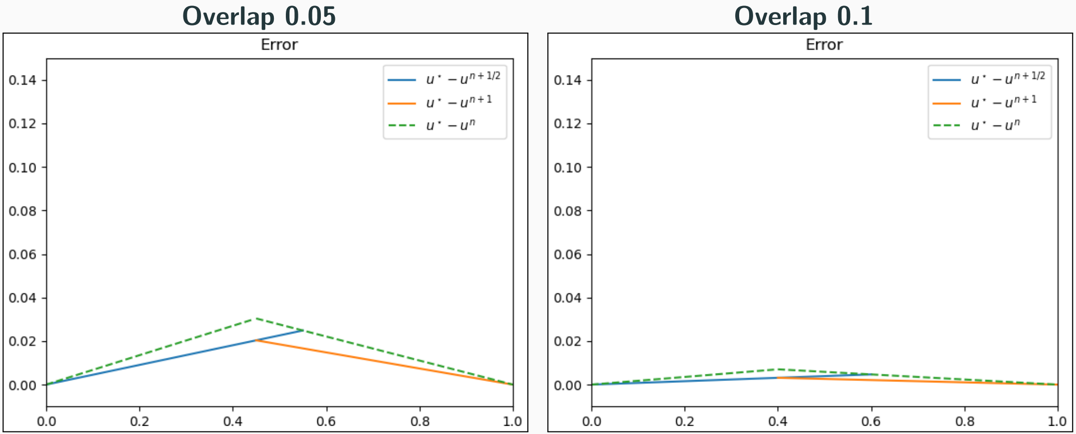
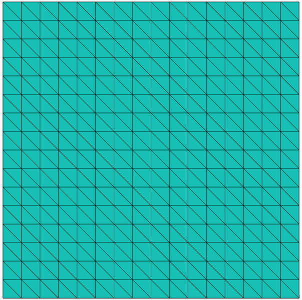


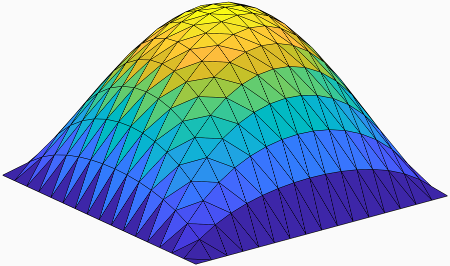
Figure 5: Error in iteration 5.

$\Rightarrow$ A **larger overlap** leads to **faster convergence**.

Part II — Schwarz Domain Decomposition Preconditioners

6 Model Problem



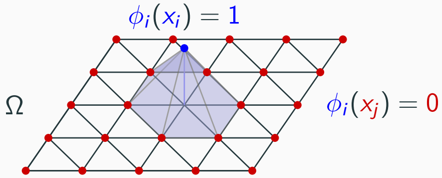


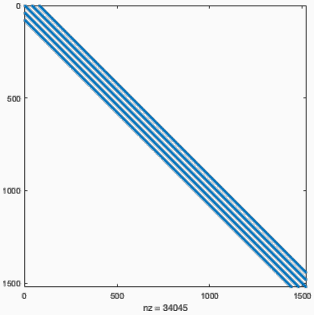
Let us consider the simple **diffusion model problem**:

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

Discretization using finite elements yields the linear equation system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}.$$





- 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**.

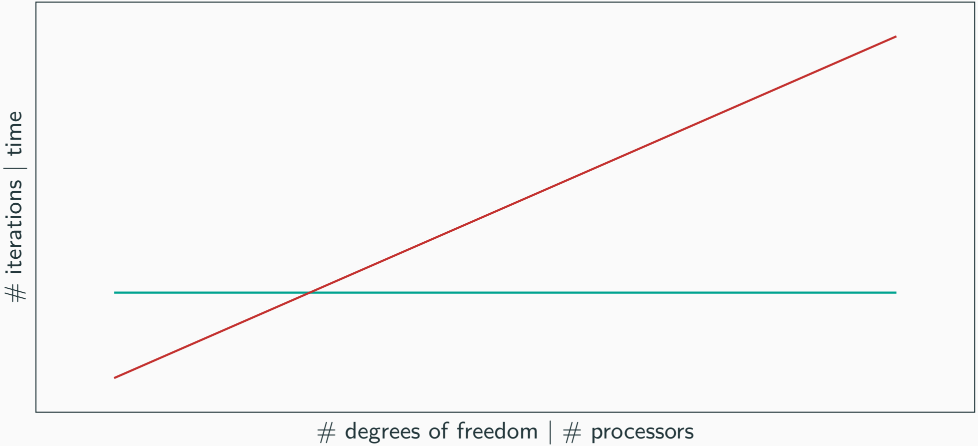
→ We will employ iterative solvers:
For our elliptic model problem, the system matrix is symmetric positive definite, such that we can use the **preconditioner gradient descent (PCG) method**.

Goal – Numerical & Parallel (Weak) Scalability

Increase the problem size while keeping

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

fixed.



Preconditioned Conjugate Gradient (PCG) Method

Algorithm 1: Preconditioned conjugate gradient (PCG) method

Result: Approximate solution of the linear equation system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$

Given: Initial guess $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$ and tolerance $\varepsilon > 0$

$\boldsymbol{r}^{(0)} := \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}$

$\boldsymbol{p}^{(0)} := \boldsymbol{y}^{(0)} := \boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}$

while $\left\|\boldsymbol{r}^{(k)}\right\| \geq \varepsilon \left\|\boldsymbol{r}^{(0)}\right\|$ **do**

$\quad \alpha_k := \frac{\left(\boldsymbol{p}^{(k)}, \boldsymbol{r}^{(k)}\right)}{\left(\boldsymbol{A}\boldsymbol{p}^{(k)}, \boldsymbol{p}^{(k)}\right)}$

$\quad \boldsymbol{x}^{(k+1)} := \boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{y}^{(k)}$

$\quad \boldsymbol{r}^{(k+1)} := \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{A}\boldsymbol{p}^{(k)}$

$\quad \boldsymbol{y}^{(k+1)} := \boldsymbol{M}^{-1}\boldsymbol{r}^{(k+1)}$

$\quad \beta_k := \frac{\left(\boldsymbol{y}^{(k+1)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}{\left(\boldsymbol{p}^{(k)}, \boldsymbol{A}\boldsymbol{p}^{(k)}\right)}$

$\quad \boldsymbol{p}^{(k+1)} := \boldsymbol{r}^{(k+1)} - \beta_k \boldsymbol{p}^{(k)}$

end

Theorem 6.1

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

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

where $\kappa\left(\boldsymbol{M}^{-1}\boldsymbol{A}\right) = \frac{\lambda_{\max}(\boldsymbol{M}^{-1}\boldsymbol{A})}{\lambda_{\min}(\boldsymbol{M}^{-1}\boldsymbol{A})}$ *is condition number of the preconditioned matrix* $\boldsymbol{M}^{-1}\boldsymbol{A}$.

Do we need a preconditioner?

The condition number of the stiffness matrix $\boldsymbol{K}$ for the diffusion problem behaves as follows:

$$\kappa\left(\boldsymbol{K}\right) \leq C \frac{\left(\max_{T \in \tau_h} h_T\right)^d}{\left(\min_{T \in \tau_h} h_T\right)^{d+2}} \overset{\text{quasi uniform}}{\equiv} C \frac{1}{h^2},$$

where τ_h is the triangulation and d is the problem dimension (for instance, $d = 2, 3$).

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

Theorem 6.1

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

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

where $\kappa\left(\boldsymbol{M}^{-1}\boldsymbol{A}\right) = \frac{\lambda_{\max}(\boldsymbol{M}^{-1}\boldsymbol{A})}{\lambda_{\min}(\boldsymbol{M}^{-1}\boldsymbol{A})}$ is condition number of the preconditioned matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$.

Do we need a preconditioner?

The condition number of the stiffness matrix $\boldsymbol{K}$ for the diffusion problem behaves as follows:

$$\kappa\left(\boldsymbol{K}\right) \leq C \frac{\left(\max_{T \in \tau_h} h_T\right)^d}{\left(\min_{T \in \tau_h} h_T\right)^{d+2}} \overset{\text{quasi uniform}}{\equiv} C \frac{1}{h^2},$$

where τ_h is the triangulation and d is the problem dimension (for instance, $d = 2, 3$).

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

Theorem 6.1

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

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

where $\kappa\left(\boldsymbol{M}^{-1}\boldsymbol{A}\right) = \frac{\lambda_{\max}\left(\boldsymbol{M}^{-1}\boldsymbol{A}\right)}{\lambda_{\min}\left(\boldsymbol{M}^{-1}\boldsymbol{A}\right)}$ *is condition number of the preconditioned matrix* $\boldsymbol{M}^{-1}\boldsymbol{A}$.

Do we need a preconditioner?

The condition number of the stiffness matrix $\boldsymbol{K}$ for the diffusion problem behaves as follows:

$$\kappa\left(\boldsymbol{K}\right) \leq C \frac{\left(\max_{T \in \tau_h} h_T\right)^d}{\left(\min_{T \in \tau_h} h_T\right)^{d+2}} \overset{\text{quasi uniform}}{\equiv} C \frac{1}{h^2},$$

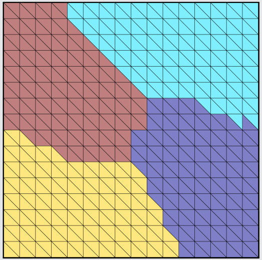
where τ_h is the triangulation and d is the problem dimension (for instance, $d = 2, 3$).

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

Overlapping domain decomposition

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

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

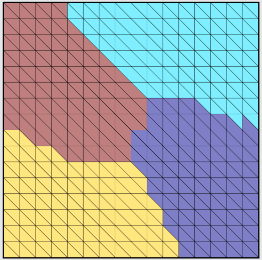


Nonoverlap. DD

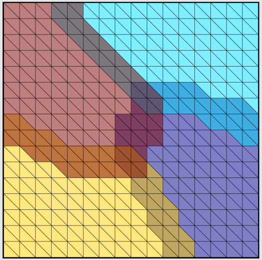
Overlapping domain decomposition

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

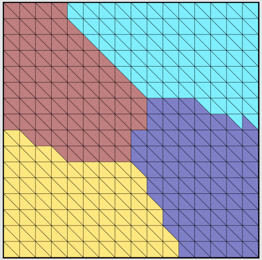


Overlap $\delta = 1h$

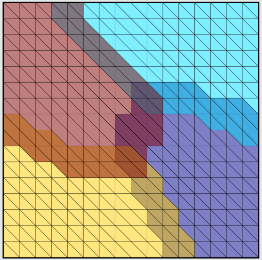
Overlapping domain decomposition

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

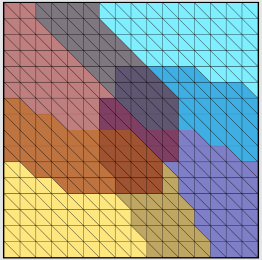
$$\Omega = \bigcup_{i=1}^{N} \Omega_i'.$$



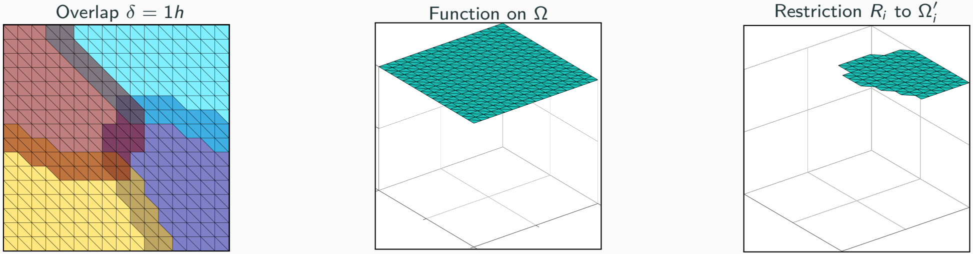
Nonoverlap. DD



Overlap $\delta = 1h$



Overlap $\delta = 2h$



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

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

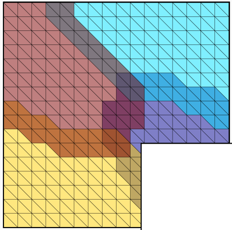
where R_i and R_i^T are restriction and prolongation operators corresponding to Ω_i', and $K_i := R_i K R_i^T$. The K_i correspond to **local Dirichlet problems** on the overlapping subdomains.

Condition number bound:

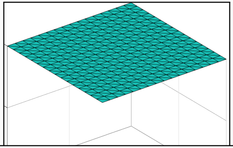
$$\kappa\left(M_{\text{OS-1}}^{-1} K\right) \leq C \left(1 + \frac{1}{H\delta}\right)$$

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

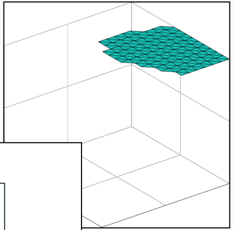
Overlap $\delta = 1h$

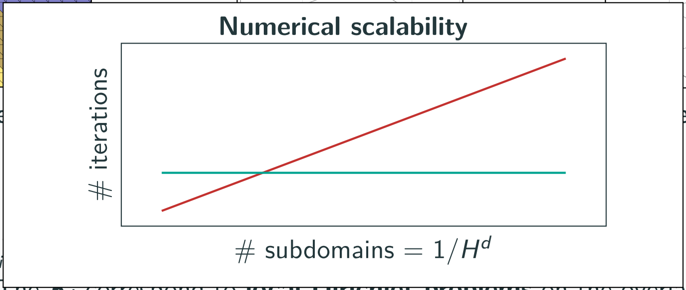


Function on Ω



Restriction R_i to Ω_i'





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

where $\boldsymbol{R}_i$ and $\boldsymbol{R}_i$ … Ω_i', and $\boldsymbol{K}_i := \boldsymbol{R}_i \boldsymbol{K} \boldsymbol{R}_i^T$. … on the overlapping subdomains.

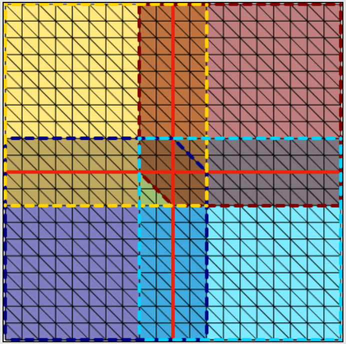
Condition number bound:

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

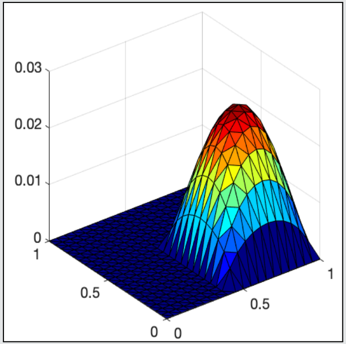
where the constant C is **independent of the subdomain size** H **and the width of the overlap** δ.

Solving a local subdomain problem

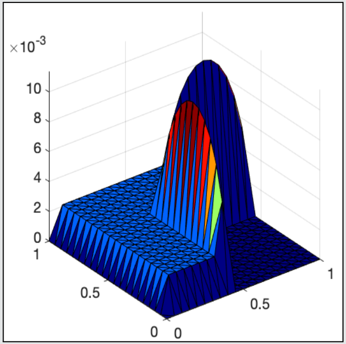
Overlap $\delta = 2h$



Solution on Ω_2

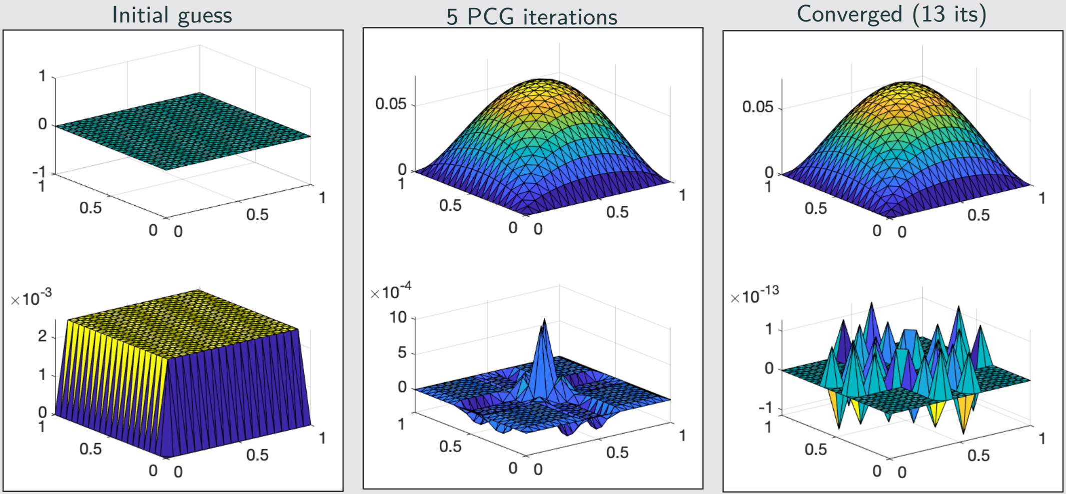


Corresponding residual

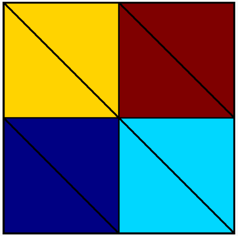


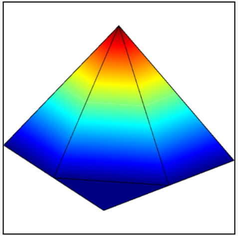
→ **Zero residual only inside this subdomain** but **particularly large residual inside the overlap**.

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



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





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

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

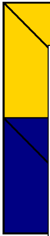
where Φ contains the coarse basis functions and $K_0 := \Phi^T K \Phi$.

Condition number bound:

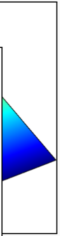
$$\kappa\left(M_{\text{OS-2}}^{-1} K\right) \leq C \left(1 + \frac{H}{\delta}\right)$$

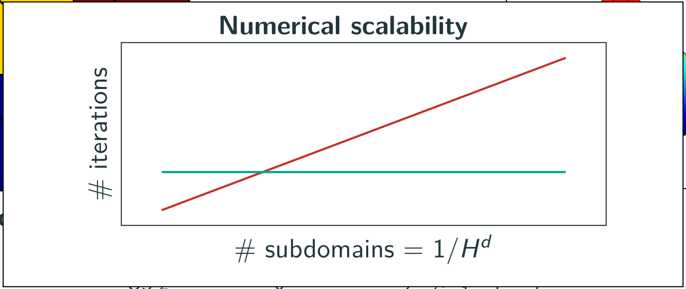
where the constant C is **independent of** h**,** δ**, and** H; cf., e.g., Toselli, Widlund (2005).

Coarse triangulation



Nodal bilinear basis function





The additive **two**

coarse level – global

first level – local

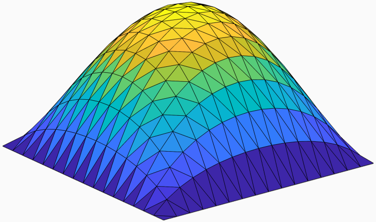
where Φ contains the coarse basis functions and $\boldsymbol{K}_0 := \Phi^T \boldsymbol{K} \Phi$.

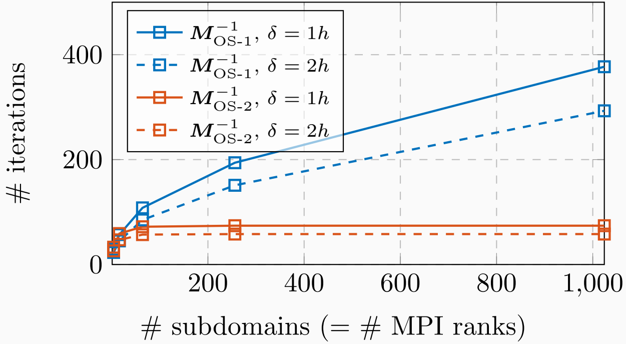
Condition number bound:

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

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

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



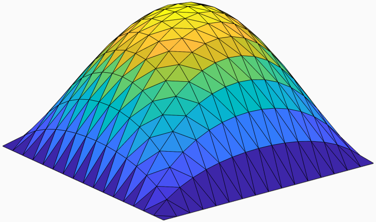


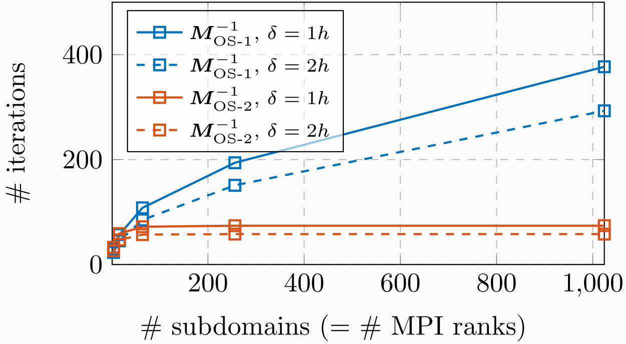
→ We only obtain **numerical scalability** if a **coarse level** is used.

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

One- Vs Two-Level Schwarz Preconditioners

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



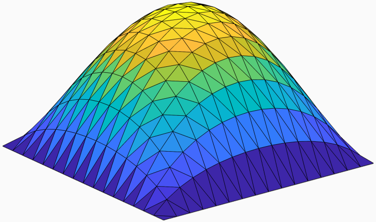


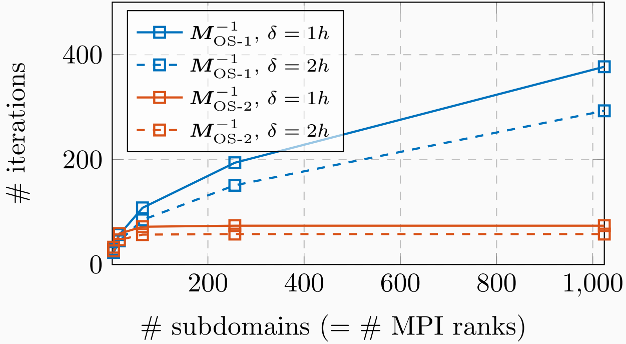
→ We only obtain **numerical scalability** if a **coarse level** is used.

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

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**.

9 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 $\rho(\mathcal{E})$ its spectral radius)

Assumption 3: Local Stability

There exists $\omega < 0$, such that

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

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 be handled easily.

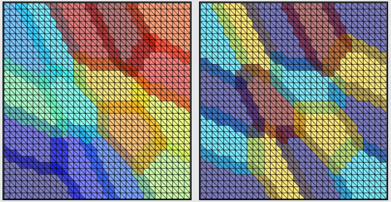
Exact Solvers

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 $u_i \in V_i$, $i = 0, \ldots, N$, 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:

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

This results in

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

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

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

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 $u - u_0$

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

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

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

Restricted Schwarz Preconditioner (Cai and Sarkis (1999))

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

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

where

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

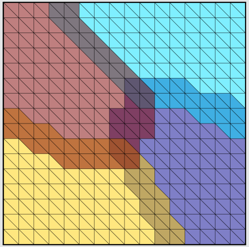
Therefore, we can just introduce a diagonal scaling matrix $\boldsymbol{D}$, such that

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

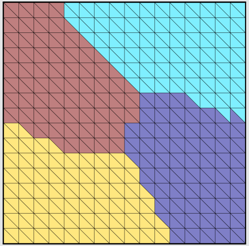
for example based on a nonoverlapping domain decomposition or an inverse multiplicity scaling.

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

R_i



$\widetilde{R}_i^T$



Changing the local and coarse solvers

For solving

$$\boldsymbol{K}_i^{-1}, \quad i = 0, \ldots, N,$$

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

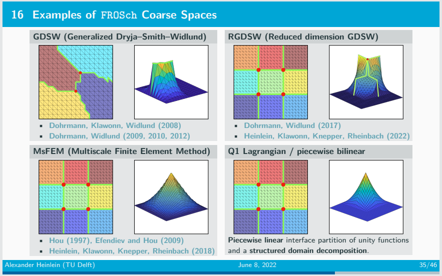
- iterative solvers
- preconditioners

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.

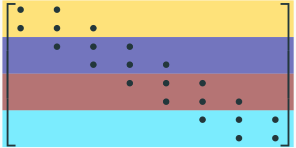
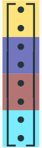


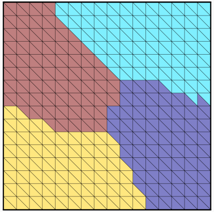
Part III — Schwarz Domain Decomposition Preconditioners in FROSch

Parallel distributed system

$$Ax = b$$

with

$A =$  $b =$ 



Wishlist:

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

12 FROSch (Fast and Robust Overlapping Schwarz) Framework in Trilinos

Software

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

Methodology

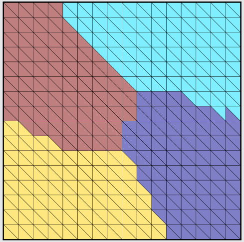
- Parallel scalable multi-level Schwarz domain decomposition preconditioners
- Algebraic construction based on the parallel distributed system matrix
- Extension-based coarse spaces

Team (active)

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

Overlapping domain decomposition

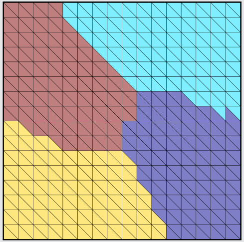
In FROSch, the overlapping subdomains $\Omega_1', \ldots, \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



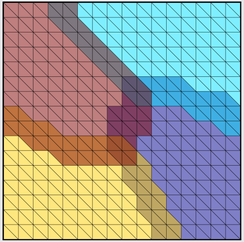
Nonoverlapping DD

Overlapping domain decomposition

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



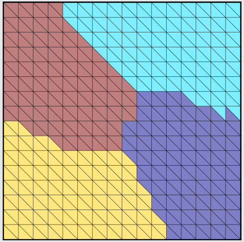
Nonoverlapping DD



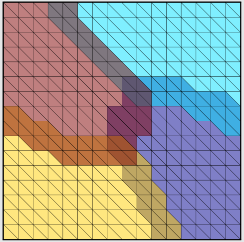
Overlap $\delta = 1h$

Overlapping domain decomposition

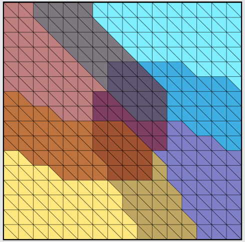
In FROSch, the overlapping subdomains $\Omega_1', \ldots, \Omega_N'$ are constructed by **recursively adding layers of elements** to the nonoverlapping subdomains; this can be performed based on the sparsity pattern of K.



Nonoverlapping DD



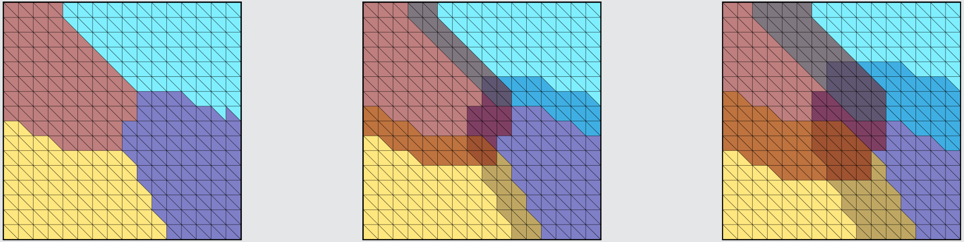
Overlap $\delta = 1h$



Overlap $\delta = 2h$

Overlapping domain decomposition

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



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

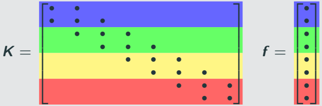
Computation of the overlapping matrices

The overlapping matrices

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

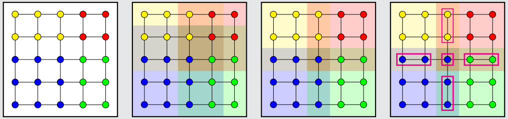
can easily be extracted from K since R_i is just a **global-to-local index mapping**.

1. Identification interface components



Identification from **parallel distribution** of matrix:

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



2. Interface partition of unity (IPOU)

vertex & edge functions | vertex functions

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

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

3. Interface basis

×

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

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

4. Extension into the interior

edge basis function | vertex basis function

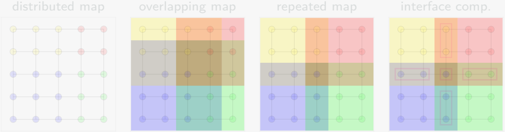
The values in the interior of the subdomains are computed via the **extension operator**:

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

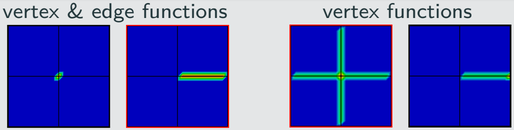
(For elliptic problems: **energy-minimizing extension**)

1. Identification interface components

Identification from **parallel distribution** of matrix:

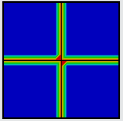


2. Interface partition of unity (IPOU)



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

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



3. Interface basis

×

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

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

4. Extension into the interior

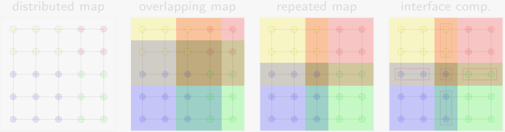
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

1. Identification interface components

Identification from **parallel distribution** of matrix:

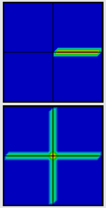


2. Interface partition of unity (IPOU)

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

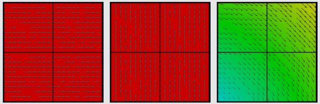
$$\sum_i \pi_i = 1 \text{ on } \Gamma$$

3. Interface basis



×

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



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

4. Extension into the interior

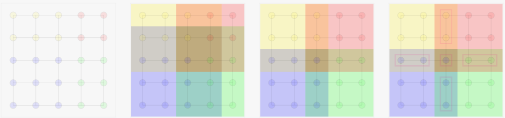
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

1. Identification interface components

Identification from **parallel distribution** of matrix:



2. Interface partition of unity (IPOU)

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

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

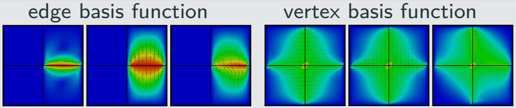
3. Interface basis

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

×

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

4. Extension into the interior

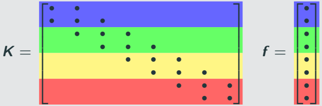


The values in the interior of the subdomains are computed via the **extension operator**:

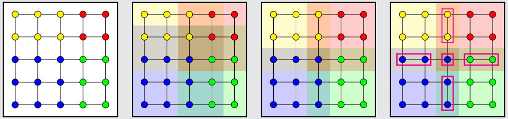
$$\Phi = \begin{bmatrix} \Phi_I \\ \Phi_\Gamma \end{bmatrix} = \begin{bmatrix} -K_{II}^{-1} K_{\Gamma I}^T \Phi_\Gamma \\ \Phi_\Gamma \end{bmatrix}.$$

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

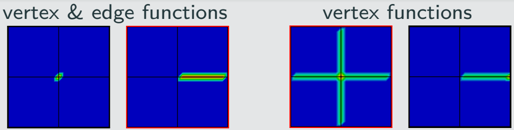
1. Identification interface components



Identification from **parallel distribution** of matrix:

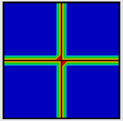


2. Interface partition of unity (IPOU)

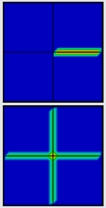


Based on the interface components, construct an **interface partition of unity**:

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

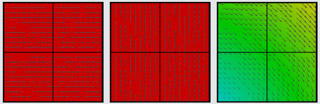


3. Interface basis



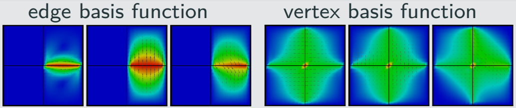
$\times$

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



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

4. Extension into the interior

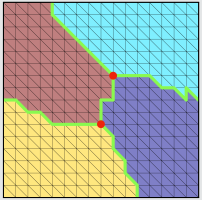


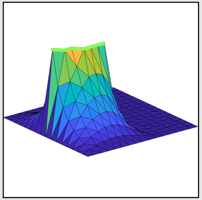
The values in the interior of the subdomains are computed via the **extension operator**:

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

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

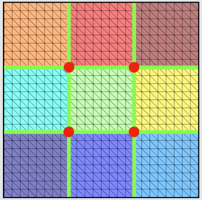
GDSW (Generalized Dryja–Smith–Widlund)

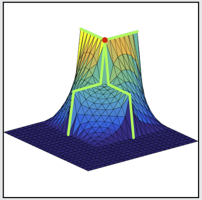




- Dohrmann, Klawonn, Widlund (2008)
- Dohrmann, Widlund (2009, 2010, 2012)

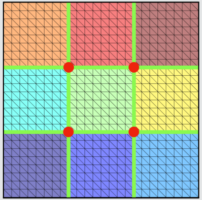
RGDSW (Reduced dimension GDSW)

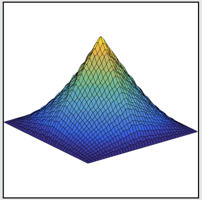




- Dohrmann, Widlund (2017)
- H., Klawonn, Knepper, Rheinbach, Widlund (2022)

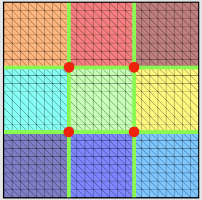
MsFEM (Multiscale Finite Element Method)

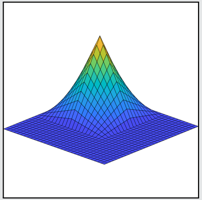




- Hou (1997), Efendiev and Hou (2009)
- Buck, Iliev, and Andrä (2013)
- H., Klawonn, Knepper, Rheinbach (2018)

Q1 Lagrangian / piecewise bilinear



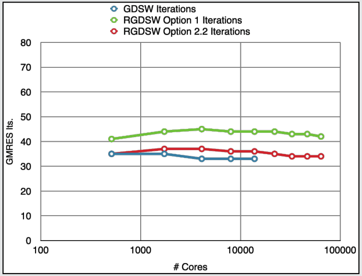


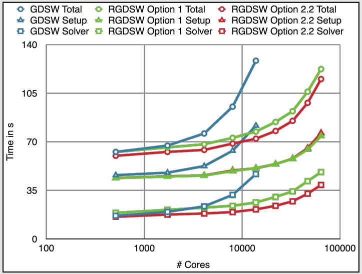
Piecewise linear interface partition of unity functions and a **structured domain decomposition**.

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

GDSW vs RGDSW (reduced dimension)

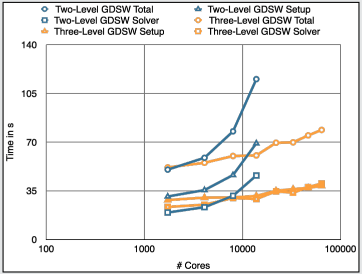
Heinlein, Klawonn, Rheinbach, Widlund (2019).





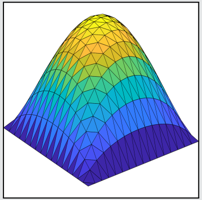
Two-level vs three-level GDSW

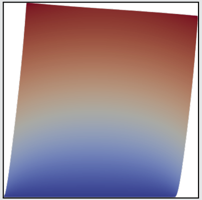
Heinlein, Klawonn, Rheinbach, Röver (2019, 2020).



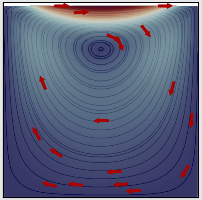
Exemplary Applications

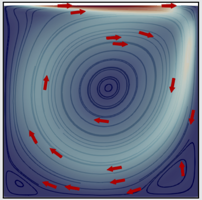
Elliptic problems


Diffusion

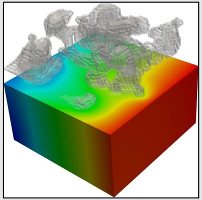

Elasticity

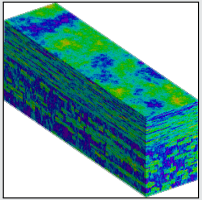
Fluid flow problems


Stokes flow

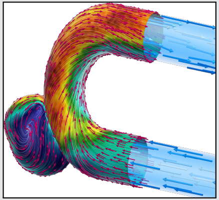

Navier–Stokes flow

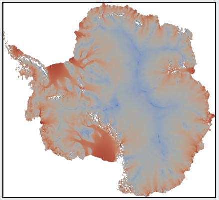
Heterogeneous problems


Dual-phase steel


Groundwater flow

Multi-physics problems


Fluid–structure interaction


Land ice simulations

16 Exercises

All the material for the exercises of this session can be found in the folder `lab2` of the **GitHub repository** of the summer school: https://github.com/jthies/dcse-summerschool

Important are the last two **exercises**

- exercise 3 – Implementing a One-Level Schwarz Preconditioner Using FROSch
- exercise 4 – Implementing a GDSW Preconditioner Using FROSch

and the **corresponding solutions** (in the subfolder `solution`),

Each exercise has **two parts**:

1. **Implement the missing code**; step-by-step explanations 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 > 1 subdomains, you have to increase the number of MPI ranks. For instance, for 4 MPI ranks / subdomains: `mpirun —n 4 ./main.x`

Thank you for your attention!

Questions?

Want to **try out FROSch at home**?

$\rightarrow$ https://github.com/searhein/frosch-demo for a **demo** with **simple installation via Docker**.