Overview Following SPAI Variants

1. Block SPAI
2. Iterative SPAI
3. Factorized SPAI (FSPAI)
4. Modified SPAI (MSPAI)

Further Variants of SPAI

1. Block SPAI

- Partition the matrix M in small blocks (e.g., 2×2 or 3×3) and apply the Frobenius norm minimization with blockwise pattern.
- Advantages:
  - Underlying block structure will also appear in the pattern of $A^{-1} \rightarrow$ improved pattern
  - Block operations are more efficient
  - Less least squares problems to solve

Further Variants of SPAI

1. Block SPAI

- Partition the matrix M in small blocks (e.g., 2×2 or 3×3) and apply the Frobenius norm minimization with blockwise pattern.
- Advantages:
  - Underlying block structure will also appear in the pattern of $A^{-1} \rightarrow$ improved pattern
  - Block operations are more efficient
  - Less least squares problems to solve

2. Iterative SPAI

- Start with pattern of $A \rightarrow M_1$
- Construct M_2 relative to new matrix AM_1
- Construct M_3 relative to new matrix AM_1M_2
- ...
- Advantage: cheaper (but inferior approximation)

3. Factorized SPAI, FSPAI

- Parallel Preconditioner for SPD matrices
- Approximate the inverse Cholesky factor of $A = L_A^T L_A$

$$A^{-1} = (L_A^T L_A)^{-1} = L_A^{-1} L_A^{-T} \approx L L^T$$

$$L_A^{-1} \approx L \qquad \rightarrow \qquad \min \| L_A L - I \|_F$$

- Normal equations give $A(\mathcal{J}_k, \mathcal{J}_k) L(\mathcal{J}_k) = \alpha e_k$

3. Factorized SPAI, FSPAI

- Parallel Preconditioner for SPD matrices
- Approximate the inverse Cholesky factor of $A = L_A^T L_A$

$$A^{-1} = (L_A^T L_A)^{-1} = L_A^{-1} L_A^{-T} \approx L L^T$$

$$L_A^{-1} \approx L \qquad \rightarrow \qquad \min \| L_A L - I \|_F$$

- Normal equations give $A(\mathcal{J}_k, \mathcal{J}_k) L(\mathcal{J}_k) = \alpha e_k$

4. Modified SPAI, MSPAI: Combining Targeting and Probing in classical SPAI formulation.

Targeting (for MSPAI)

$$\min_{M \in \mathcal{P}} \|AM - T\|_F$$

- Assume, T is a good sparse preconditioner for A. Improve T by computing M and solving the above Frobenius norm minimization.

Targeting (for MSPAI)

$$\min_{M \in \mathcal{P}} \|AM - T\|_F$$

- Assume, T is a good sparse preconditioner for A. Improve T by computing M and solving the above Frobenius norm minimization.
- Application: Assume that A is given by two parts, e.g. an advection part and a diffusion part.

  We can choose T as Laplacian relative to the diffusion part (easy to solve) and then we add M for improving T relative to the advection part.

Probing (for MSPAI)

- Find preconditioner of special form (tridiag, band) for preconditioning a matrix that is not given explicitly, but only by its action on certain (probing-)vectors, e.g.

$$e^T S = f^T.$$

  Example: S is Schur complement or general matrix.

- Choose, e.g. $e = (1, 1, \ldots, 1)^T$, $e = (1, -1, 1, -1, \ldots)^T$, $\ldots$

Probing (for MSPAI)

- Find preconditioner of special form (tridiag, band) for preconditioning a matrix that is not given explicitly, but only by its action on certain (probing-)vectors, e.g.

$$e^T S = f^T.$$

  Example: S is Schur complement or general matrix.

- Choose, e.g. $e = (1, 1, \ldots, 1)^T$, $e = (1, -1, 1, -1, \ldots)^T$, $\ldots$

- Disadvantage: Can use only very special pattern for M and special probing vectors e.
  Example: tridiagonal probing

4. Modified SPAI: SPAI with Targeting and Probing

Generalize Frobenius norm minimization to

$$\min_{M \in \mathcal{P}} \|CM - B\|_F = \min_{M \in \mathcal{P}} \left\| \begin{pmatrix} C_0 \\ \rho u^T \end{pmatrix} M - \begin{pmatrix} B_0 \\ \rho v^T \end{pmatrix} \right\|_F$$

For example: original SPAI extended by an additional norm minimization to deliver especially good results on vector e:

$$\min_{M \in \mathcal{P}} \|CM - B\|_F = \min_{M \in \mathcal{P}} \left\| \begin{pmatrix} A \\ \rho e^T A \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|_F$$

SPAI for Triangular Matrices

Solve $Lx = b$ with L triangular. Goal: Improve Jacobi iteration!

SPAI: $\min \|L\Lambda - I\|_F$ with pattern of Λ like L.

SPAI for Triangular Matrices

Solve $Lx = b$ with L triangular. Goal: Improve Jacobi iteration!

SPAI: $\min \|L\Lambda - I\|_F$ with pattern of Λ like L.

Modification: Sparse approximate block inverse.

Replace rectangular Least Squares problem in SPAI $L(I, J)\Lambda_j = e_j$ by square linear system $L(J, J)\Lambda_j = e_j$.

SPAI for Triangular Matrices

Solve $Lx = b$ with L triangular. Goal: Improve Jacobi iteration!

SPAI: $\min \|L\Lambda - I\|_F$ with pattern of Λ like L.

Modification: Sparse approximate block inverse.

Replace rectangular Least Squares problem in SPAI $L(I, J)\Lambda_j = e_j$ by square linear system $L(J, J)\Lambda_j = e_j$.

Stationary iterative method based on preconditioner Λ:

$$b = L\Lambda\Lambda^{-1}x = L\Lambda y$$

$$diag(L\Lambda)y^{k+1} = b - (L\Lambda)_0 y^k$$

Solution $x = \Lambda y$

Convergence again guaranteed, faster than Jacobi, fully parallel!

Hybrid SPAI - Gauss-Seidel

First compute SPAI-like preconditioner via $\min \|AM - I\|_F$.

Replace $b = Ax$ by $b = A(MM^{-1})x = (AM)\tilde{x} = \tilde{A}\tilde{x}$.

Apply Gauss-Seidel splitting on $\tilde{A} = D + L_0 + U_0 = L + U_0$.

Hybrid SPAI - Gauss-Seidel

First compute SPAI-like preconditioner via $\min \|AM - I\|_F$.

Replace $b = Ax$ by $b = A(MM^{-1})x = (AM)\tilde{x} = \tilde{A}\tilde{x}$.

Apply Gauss-Seidel splitting on $\tilde{A} = D + L_0 + U_0 = L + U_0$.

In every iteration step we have to solve triangular system in L.

Apply sparse approximate block inverse preconditioner for L.

Hybrid SPAI - Gauss-Seidel

First compute SPAI-like preconditioner via $\min \|AM - I\|_F$.

Replace $b = Ax$ by $b = A(MM^{-1})x = (AM)\tilde{x} = \tilde{A}\tilde{x}$.

Apply Gauss-Seidel splitting on $\tilde{A} = D + L_0 + U_0 = L + U_0$.

In every iteration step we have to solve triangular system in L.

Apply sparse approximate block inverse preconditioner for L.

Everything fully parallel.

Improved convergence compared to plain SPAI or Gauss-Seidel

Outlook: Further Research

1. Apply MSPAI with the probing feature to multigrid or regularization problems (smoother, preconditioner).
   In connection with domain decomposition:
   - In the small domains use MSPAI as smoother
   - Use MSPAI as preconditioner for Schur complement

2. Find good block pattern
   - that reflects the physical connections
   - combines columns with very similar pattern to one LS problem

3. Factorized SPAI for indefinite matrices
   - Use a priori permutations (perfect matching)
   - and blocking

   to avoid breakdowns.

Available Code

- SPAI, MSPAI, FSPAI available in parallel version (MPI, C/C++)
  - SPAI (University of Basel):
    `http://www.computational.unibas.ch/software/spai/`
  - SPAI, MSPAI (TUM):
    `http://www5.in.tum.de/wiki/index.php/MSPAI`
  - FSPAI (TUM):
    `http://www5.in.tum.de/wiki/index.php/FSPAI`

    Publications on SPAI, MSPAI, FSPAI available on these websites!

Available Code

- SPAI, MSPAI, FSPAI available in parallel version (MPI, C/C++)
  - SPAI (University of Basel):
    `http://www.computational.unibas.ch/software/spai/`
  - SPAI, MSPAI (TUM):
    `http://www5.in.tum.de/wiki/index.php/MSPAI`
  - FSPAI (TUM):
    `http://www5.in.tum.de/wiki/index.php/FSPAI`

    Publications on SPAI, MSPAI, FSPAI available on these websites!
- Block SPAI also available for GPU and shared memory (OpenMP, C/C++).
- High priority: Parallel triangular solves.

Parallel Numerics, WT 2016/2017

6 Domain Decomposition

Contents

Concept of Domain Decomposition

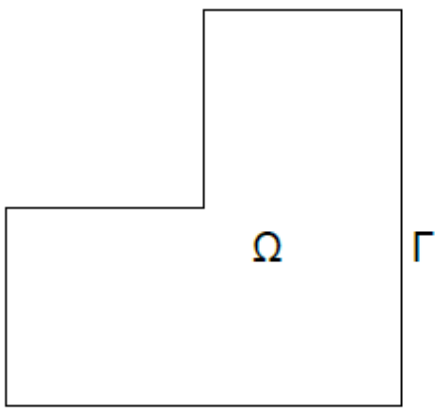
Consider elliptic PDE on region Ω with boundary Γ, e.g. with Dirichlet boundary conditions.

Example:

$$\Delta u = u_{xx} + u_{yy} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \in \Omega$$

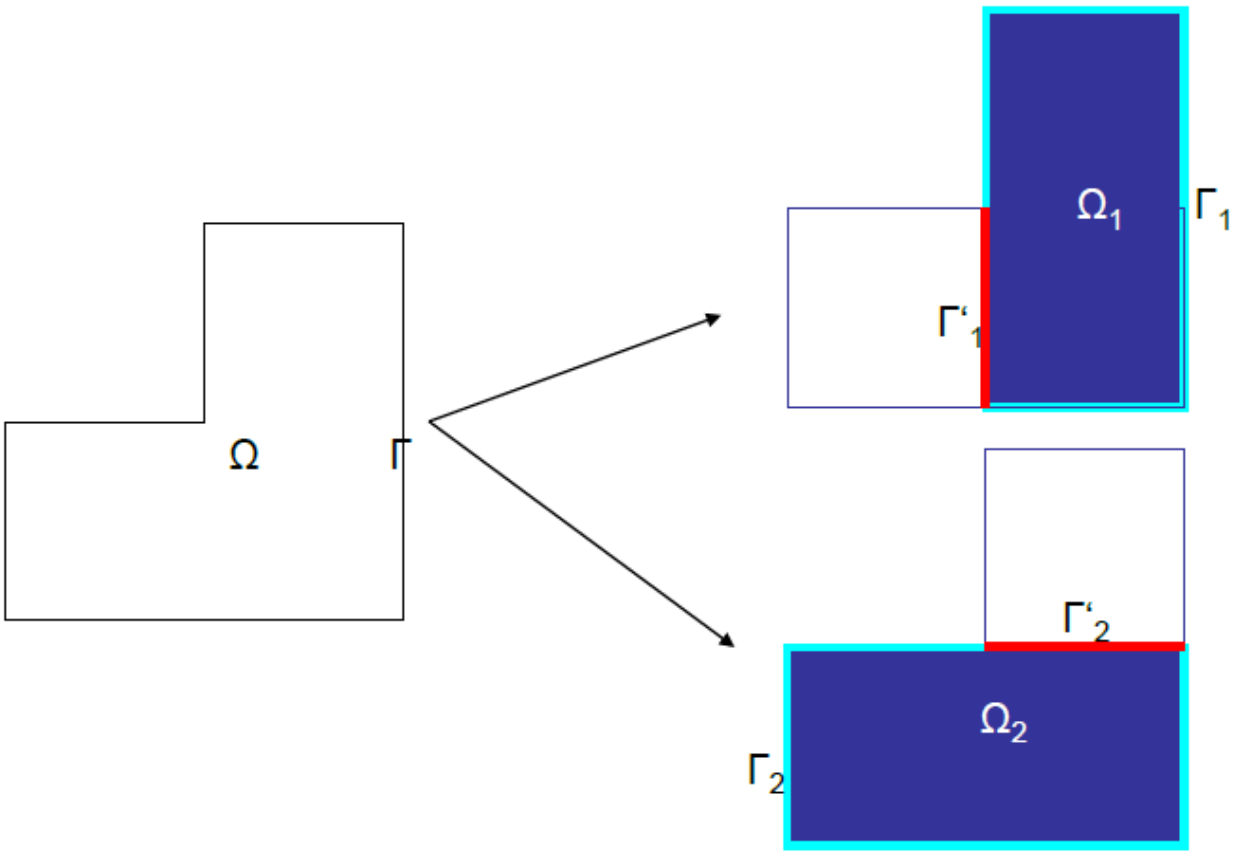
$$u(x, y)|_\Gamma = g(x, y) \in \Gamma$$

Parallel solution?



6.1. Overlapping Domain Decomposition

Partition region Ω in two regions Ω_1 and Ω_2, with new boundaries Γ_1 and Γ_2 which are partially given by old boundary Γ and some unknown parts Γ_1' and Γ_2' :



Overlapping Domain Decomposition (cont.)

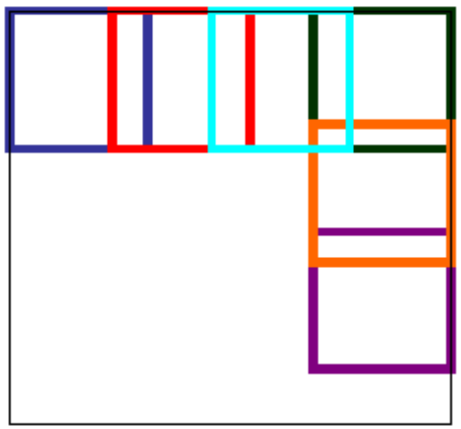
- We discretize and solve given PDE on Ω_1 with boundary Γ_1, but we need the values of $u(x, y)$ on new artificial boundary Γ_1'.
- First, we assume any values for Γ_1', e.g. $u(x, y) = 0$.
- Then we solve the linear system relative to region Ω_1.
- The same can be done in parallel for Ω_2.

Overlapping Domain Decomposition (cont.)

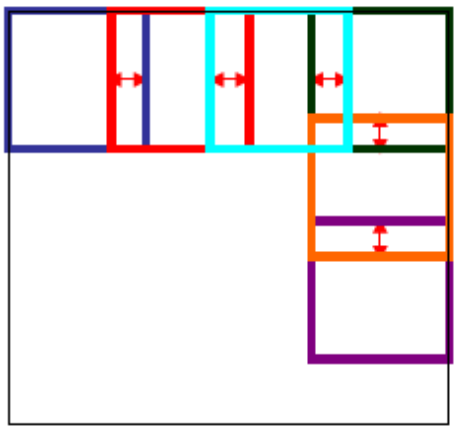
- We discretize and solve given PDE on Ω_1 with boundary Γ_1, but we need the values of $u(x, y)$ on new artificial boundary Γ_1'.
- First, we assume any values for Γ_1', e.g. $u(x, y) = 0$.
- Then we solve the linear system relative to region Ω_1.
- The same can be done in parallel for Ω_2.
- Values of resulting solution of 1) on Γ_2' can be used to compute solution in the next step for region Ω_2 with boundary Γ_2.
- In same way we use the resulting solution of 2) on Γ_1', to compute the solution for region Ω_1 with boundary Γ_1.
- So we generate solutions on partial regions which provide us with approximate values for unknown boundary values of the other partial solution.
- The sequence of solutions converges in each region against solution on Ω.

Overlapping Domain Decompos. (cont. 2)

1. Solve in parallel the PDE on all small subdomains with certain boundary cond.
2. Exchange boundary values



1st step

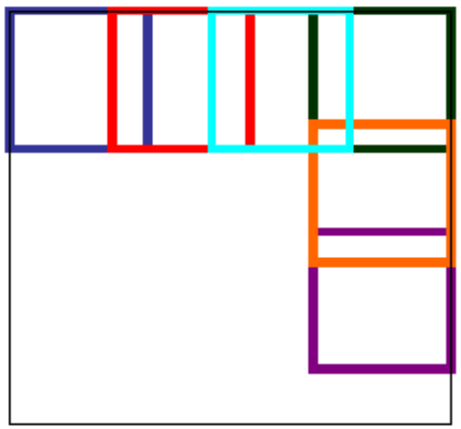


2nd step

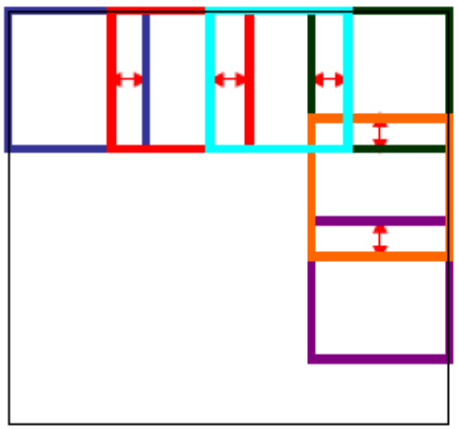
3. Repeat until convergence.

Overlapping Domain Decompos. (cont. 2)

1. Solve in parallel the PDE on all small subdomains with certain boundary cond.
2. Exchange boundary values



1st step



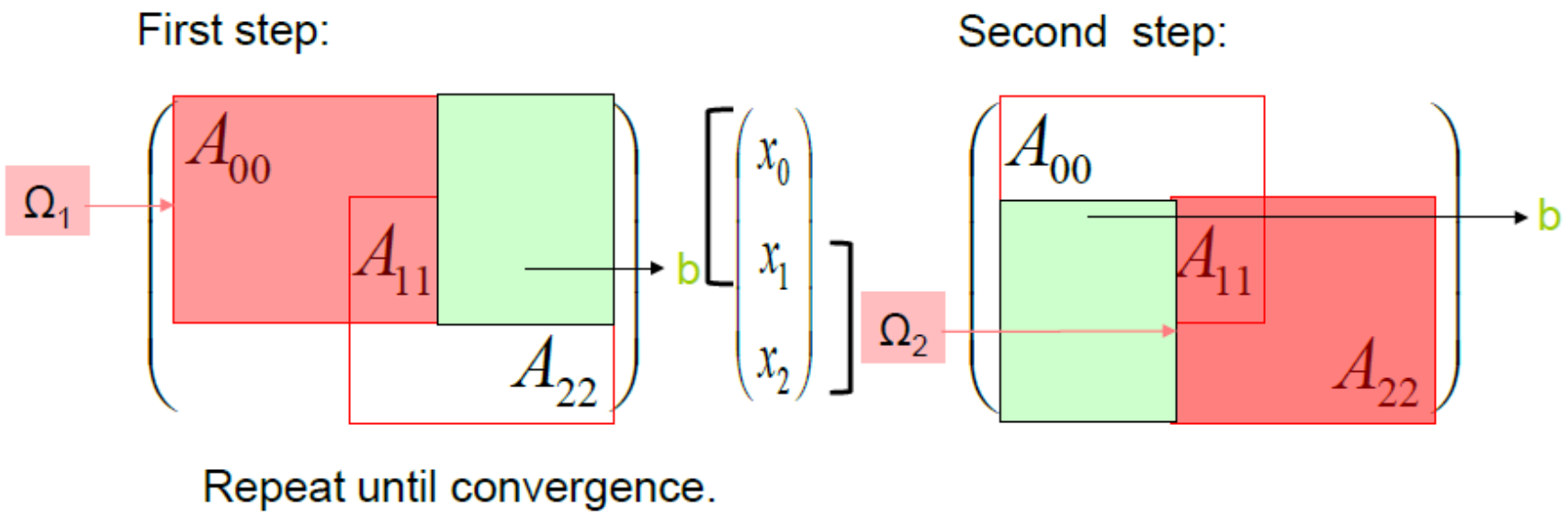
2nd step

3. Repeat until convergence.

For getting better initial values for interior boundary nodes: solve the whole problem on a coarse mesh and interpolate.

Overlapping Domain Decompos. (cont. 3)

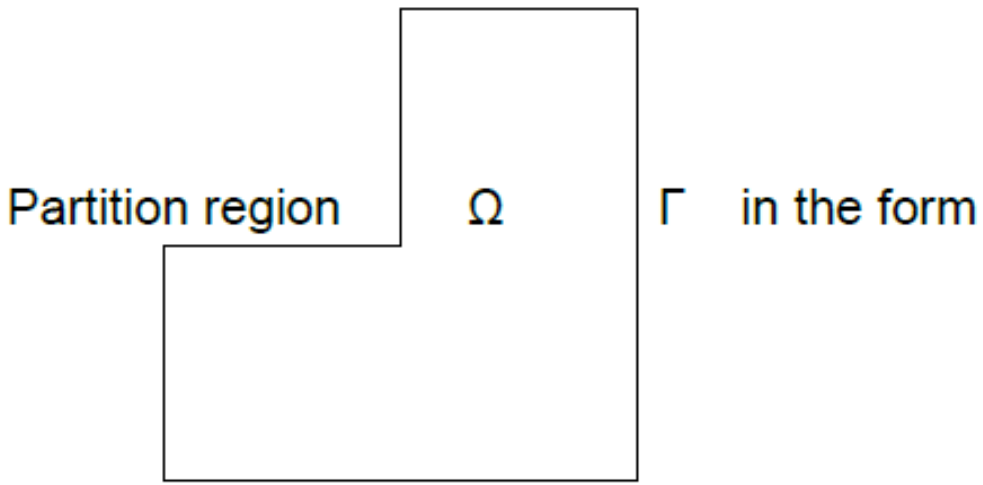
Matrix representation of overlapping domain decomposition:

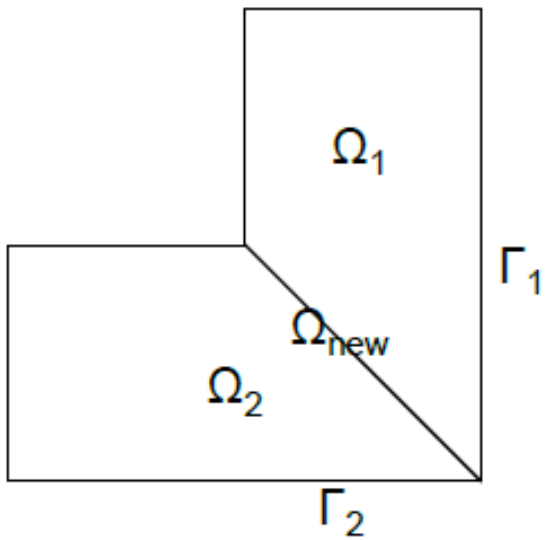


Repeat until convergence.

Green parts are related to the other domain and we asssume to know the related components in the vector *x* of unknowns. They are moved to the right-hand side *b*.

6.2. Non-overlapping Domain Decomposition





Discretization of the original problem with numbering of the unknowns relative to the partitioning given by Ω_1 and Ω_2 leads to a linear system with a matrix in dissection form.

Non-overlapping DD (cont. 2)

Poisson equation on domain Ω

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

is equivalent to

$$\begin{aligned} -\Delta u_1 &= f && \text{in} \quad \Omega_1, \\ u_1 &= 0 && \text{on} \quad \partial\Omega_1 \setminus \Gamma, \end{aligned}$$

$$\begin{aligned} u_1 &= u_2 && \text{on} \quad \Gamma, \\ \frac{\partial u_1}{\partial n_1} &= -\frac{\partial u_2}{\partial n_2} && \text{on} \quad \Gamma, \end{aligned}$$

$$\begin{aligned} -\Delta u_2 &= f && \text{in} \quad \Omega_2, \\ u_2 &= 0 && \text{on} \quad \partial\Omega_2 \setminus \Gamma. \end{aligned}$$

Non-overlapping DD (cont. 3)

In matrix-vector notation $Au = f$ can be written as

$$A = \begin{pmatrix} A_{I,I}^{(1)} & 0 & A_{I,\Gamma}^{(1)} \\ 0 & A_{I,I}^{(2)} & A_{I,\Gamma}^{(2)} \\ A_{\Gamma,I}^{(1)} & A_{\Gamma,I}^{(2)} & A_{\Gamma,\Gamma} \end{pmatrix}, u = \begin{pmatrix} u_I^{(1)} \\ u_I^{(2)} \\ u_\Gamma \end{pmatrix}, f = \begin{pmatrix} f_I^{(1)} \\ f_I^{(2)} \\ f_\Gamma \end{pmatrix}, \quad (1)$$

where the degrees of freedom are partitioned into those internal to Ω_1, and to Ω_2, and those of the interior of Γ.

On next slide we formulate problem (1) in a more general notation.

Non-overlapping DD (cont. 4)

- A_3 is the so called interface matrix:

$$f = Au = \begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

Better reduce the original problem to two partial subproblems and one interface Schur complement system.

Non-overlapping DD (cont. 4)

- A_3 is the so called interface matrix:

$$f = Au = \begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

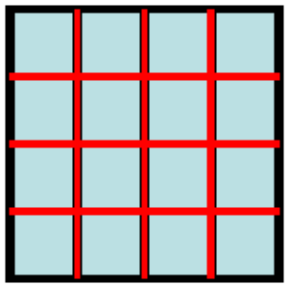
  Better reduce the original problem to two partial subproblems and one interface Schur complement system.

- We can solve $Au = f$ iteratively with PCG and preconditioner:

$$\begin{pmatrix} A_1^{-1} & 0 & 0 \\ 0 & A_2^{-1} & 0 \\ 0 & 0 & M \end{pmatrix}$$

  Here, for M we can use the identity or an approximate inverse for the Schur complement.

Non-overlapping DD (cont. 5)

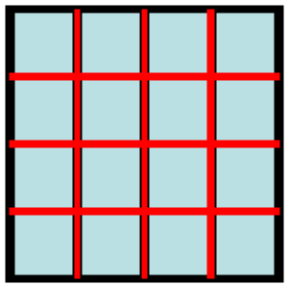


Leads to 16 block matrices on the diagonal $A_1, \ldots, A_{16}$ and Schur complement S.

$$A = \begin{pmatrix} A_1 & & & F_1 \\ & \ddots & & \vdots \\ & & A_{16} & F_{16} \\ G_1 & \cdots & G_{16} & A_{17} \end{pmatrix}$$

$$\left(S = A_{17} - G_1 A_1^{-1} F_1 - \ldots - G_{16} A_{16}^{-1} F_{16} \right)$$

Non-overlapping DD (cont. 5)



Leads to 16 block matrices on the diagonal $A_1, \ldots, A_{16}$ and Schur complement S.

$$A = \begin{pmatrix} A_1 & & & F_1 \\ & \ddots & & \vdots \\ & & A_{16} & F_{16} \\ G_1 & \cdots & G_{16} & A_{17} \end{pmatrix}$$

$$\left(S = A_{17} - G_1 A_1^{-1} F_1 - \ldots - G_{16} A_{16}^{-1} F_{16} \right)$$

General comments on DD:

- Solve small problems e.g. with multigrid in parallel.
- Overlapping is easy to parallelize, but slow convergence
- Non-overlapping is harder to parallelize, more influence on convergence in S.

6.3. Schur Complements

Write matrix A from (1) in block factorized form $A = LR$

$$L = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A^{(1)}_{\Gamma,I} A^{(1)^{-1}}_{I,I} & A^{(2)}_{\Gamma,I} A^{(2)^{-1}}_{I,I} & I \end{pmatrix}, \qquad R = \begin{pmatrix} A^{(1)}_{I,I} & 0 & A^{(1)}_{I,\Gamma} \\ 0 & A^{(2)}_{I,I} & A^{(2)}_{I,\Gamma} \\ 0 & 0 & S \end{pmatrix}$$

leading to the resulting linear system

$$\begin{pmatrix} A^{(1)}_{I,I} & 0 & A^{(1)}_{I,\Gamma} \\ 0 & A^{(2)}_{I,I} & A^{(2)}_{I,\Gamma} \\ 0 & 0 & S \end{pmatrix} u = \begin{pmatrix} f^{(1)}_I \\ f^{(2)}_I \\ b_\Gamma \end{pmatrix},$$

where

$$S = A_{\Gamma,\Gamma} - A^{(1)}_{\Gamma,I} A^{(1)^{-1}}_{I,I} A^{(1)}_{I,\Gamma} - A^{(2)}_{\Gamma,I} A^{(2)^{-1}}_{I,I} A^{(2)}_{I,\Gamma}$$

being the Schur complement relative to the unknowns Γ.

Schur Complements (cont.)

We can transform $Au = f$ into $LRu = f$, resp. $Ru = L^{-1}f = b$ with

$$L^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A^{(1)}_{\Gamma,I}A^{(1)}_{I,I}{}^{-1} & A^{(2)}_{\Gamma,I}A^{(2)}_{I,I}{}^{-1} & I \end{pmatrix}^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ -A^{(1)}_{\Gamma,I}A^{(1)}_{I,I}{}^{-1} & -A^{(2)}_{\Gamma,I}A^{(2)}_{I,I}{}^{-1} & I \end{pmatrix}$$

$$\begin{pmatrix} A^{(1)}_{I,I} & 0 & A^{(1)}_{I,\Gamma} \\ 0 & A^{(2)}_{I,I} & A^{(2)}_{I,\Gamma} \\ 0 & 0 & S \end{pmatrix} u = \begin{pmatrix} f^{(1)}_I \\ f^{(2)}_I \\ b_\Gamma \end{pmatrix}$$

$$b_\Gamma = f_\Gamma - A^{(1)}_{\Gamma,I}A^{(1)}_{I,I}{}^{-1} f^{(1)}_I - A^{(2)}_{\Gamma,I}A^{(2)}_{I,I}{}^{-1} f^{(2)}_I$$

Once u_Γ is found, the internal components can be found by

$$u^{(i)}_I = A^{(i)}_{I,I}{}^{-1}(f^{(i)}_I - A^{(i)}_{I,\Gamma}u_\Gamma).$$

Direct Derivation of the Schur Complement

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$\begin{aligned} \Rightarrow \quad & A_1 x_1 + F_1 x_3 = b_1 \\ & A_2 x_2 + F_2 x_3 = b_2 \\ & G_1 x_1 + G_2 x_2 + A_3 x_3 = b_3 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & x_1 = A_1^{-1} b_1 - A_1^{-1} F_1 x_3 \quad \text{and} \\ & x_2 = A_2^{-1} b_2 - A_2^{-1} F_2 x_3 \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & (G_1 A_1^{-1} b_1 - G_1 A_1^{-1} F_1 x_3) + (G_2 A_2^{-1} b_2 - G_2 A_2^{-1} F_2 x_3) + A_3 x_3 = b_3 \\ \Rightarrow \quad & (A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2) x_3 = b_3 - G_1 A_1^{-1} b_1 - G_2 A_2^{-1} b_2 \\ \Rightarrow \quad & S x_3 = \widetilde{b}_3 \end{aligned}$$

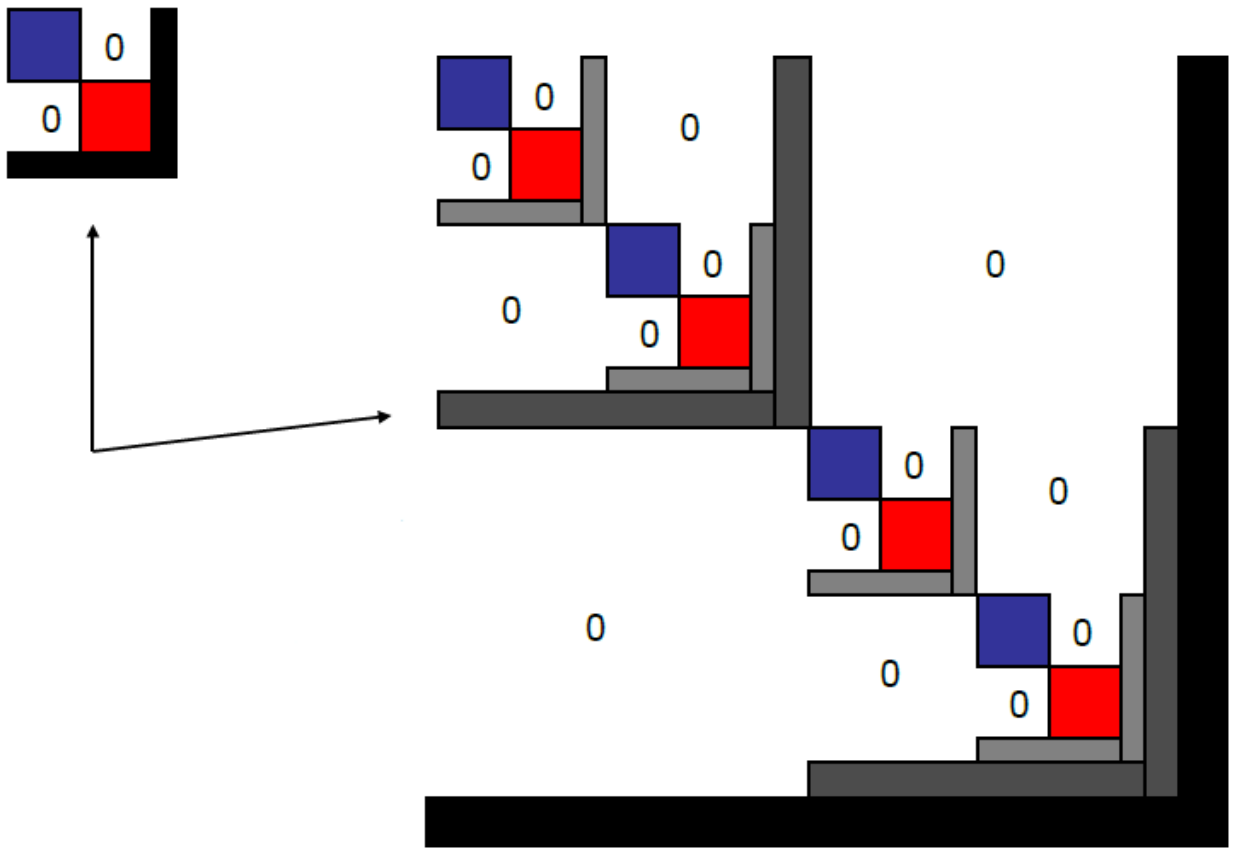
(Parallel) Algorithm to solve $Ax = b$ based on Schur Complement

1. Compute S by using inv(A_1) and inv(A_2)
2. Solve $Sx_3 = \widetilde{b}_3$
3. Compute x_1 and x_2 by using inv(A_1) and inv(A_2)

- The explicit computation of S can be avoided by solving the linear system in S iteratively, e.g. Jacobi, PCG, ...
- Then we need only a part of S and in every iteration step we have to compute $S *$ intermediate vector.
- To achieve fast convergence, a preconditioner (approximation) for S has to be used!
- Precondition Schur complement e.g. with MSPAI

Recursive Form of Non-overlapping DD

$\leftrightarrow$ Nested (recursive) dissection:



Domain Decomposition: Outlook

- Approach can be generalized to
  - non-conforming discretizations (mortar methods/Lagrange multipliers/FETI)
  - time-dependent systems
  - ...
- Literature:
  - A. Toselli, O. Widlund: *Domain Decomposition Methods—Algorithms and Theory*, Springer, 2004
  - A. Quarteroni, A. Valli: *Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, 1999

Solving the Schur Complement

$$S = A_{k+1} - G_1 A_1^{-1} F_1 - \cdots - G_k A_k^{-1} F_k \Rightarrow Sx = b$$

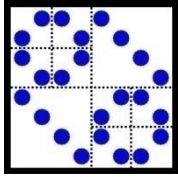
S dense and large. So one should avoid computing S explicitly!

For iterative solver you have only to compute Sx in every step – good.

But also a preconditioner is necessary - bad.

MSPAI for S:
Use SPAI for sparse approximation of A_k^{-1} and S by S‘.
Choose probing vectors e.

Solve: $\min \left\| \begin{pmatrix} S' \\ \rho e^T S \end{pmatrix} M - \begin{pmatrix} I \\ \rho e^T \end{pmatrix} \right\|$ for preconditioner M



Multigrid

Starting point: Solve Partial Differential Equation, e.g.

$$-u_{xx} - u_{yy} = f(x, y)$$

with boundary conditions, e.g. Dirichlet BC.

Discretization leads to system of linear equations, resp. matrix A.

A is sparse, (structured,) and ill-conditioned.
Looking for O(n) solver.

Direct solver $O(n^2)$ or $O(n \log(n))$
PCG also > O(n) because matrix is ill-conditioned

Idea

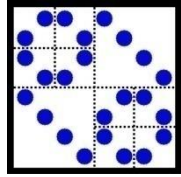
Vector of unknowns:

(1) Project the fine discretization on a coarser (smaller) problem

(2) Solve the coarse matrix (smaller matrix)

(3) Project the coarse solution back on fine grid

Problems: - only possible for smooth vector
without high oscillatory components
- backprojection introduces (high-oscillatory)
errors that have to be removed, too.



Observation:

For typical PDE matrices high-oscillatory vectors are related to the subspace to large eigenvalues and are removed e.g. by the stationary Gauss-Seidel iteration:

$\| I - M^{-1}A \|$ small for eigenvectors to large eigenvalues!

To solve Ax=f:

(1) Apply a few steps Gauss-Seidel smoothing steps → x_a
Residual equation $A(x_a+x)=f$ → $Ax = f - Ax_a = r$

(2) Project (restrict) r, resp. A on coarse grid by mean value
Solve $A_c x_c = r_c$

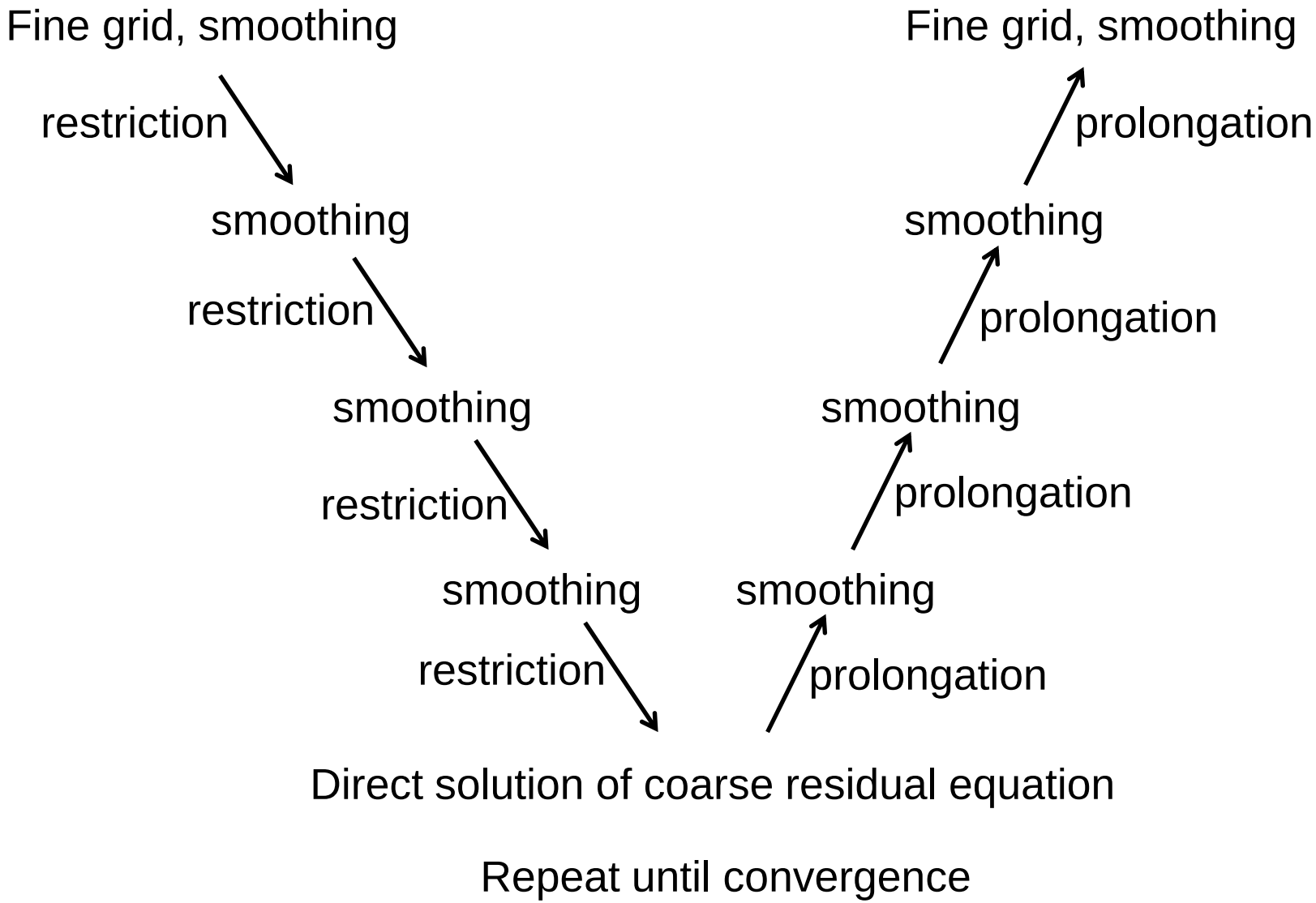
(3) Project (prolongate) x_c back on fine grid x by interpolation
x_c → x → new approximate solution $x_a + x$

(4) Improve approximate solution by Gauss-Seidel steps

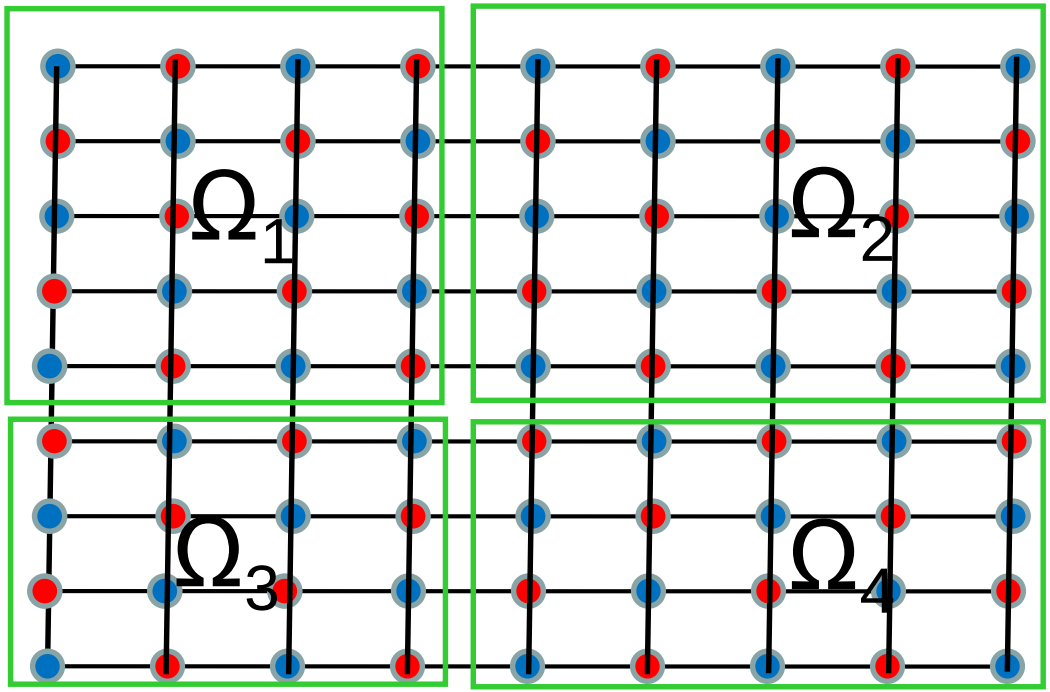
Repeat until convergence

Apply also recursively for solving coarse equations

V-Cycle

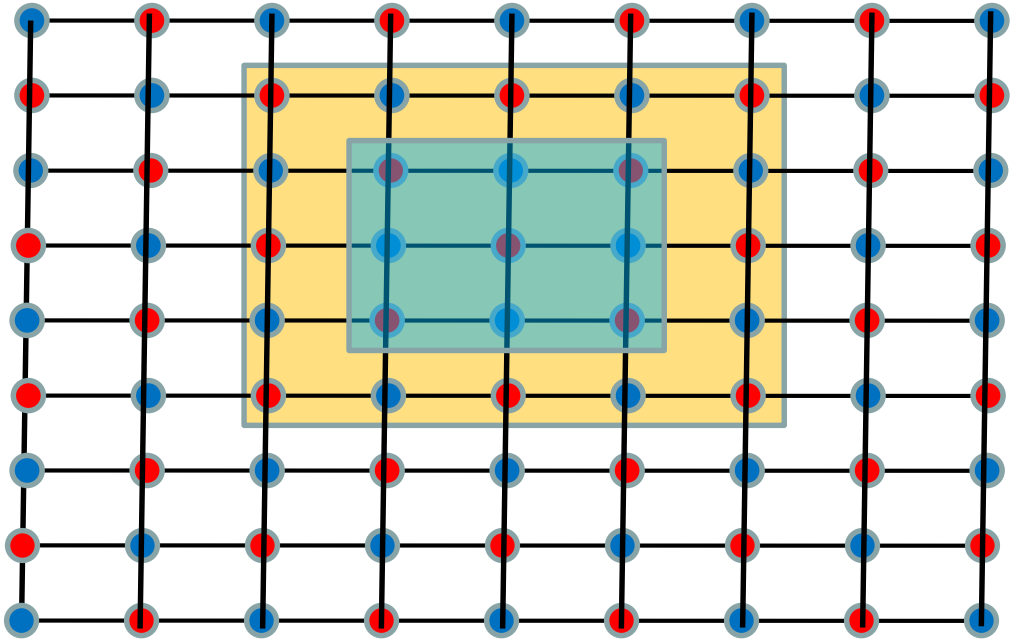

Fine grid, smoothing
restriction
smoothing
restriction
smoothing
restriction
smoothing
restriction
Direct solution of coarse residual equation
prolongation
smoothing
prolongation
smoothing
prolongation
smoothing
prolongation
Fine grid, smoothing
Repeat until convergence

Parallel Aspects



Each processor p_r has data for domain Ω_r and does projections and smoothing for its components.
Needs data from neighboring processor for projection and GS.
Load balancing!

Ghost layers



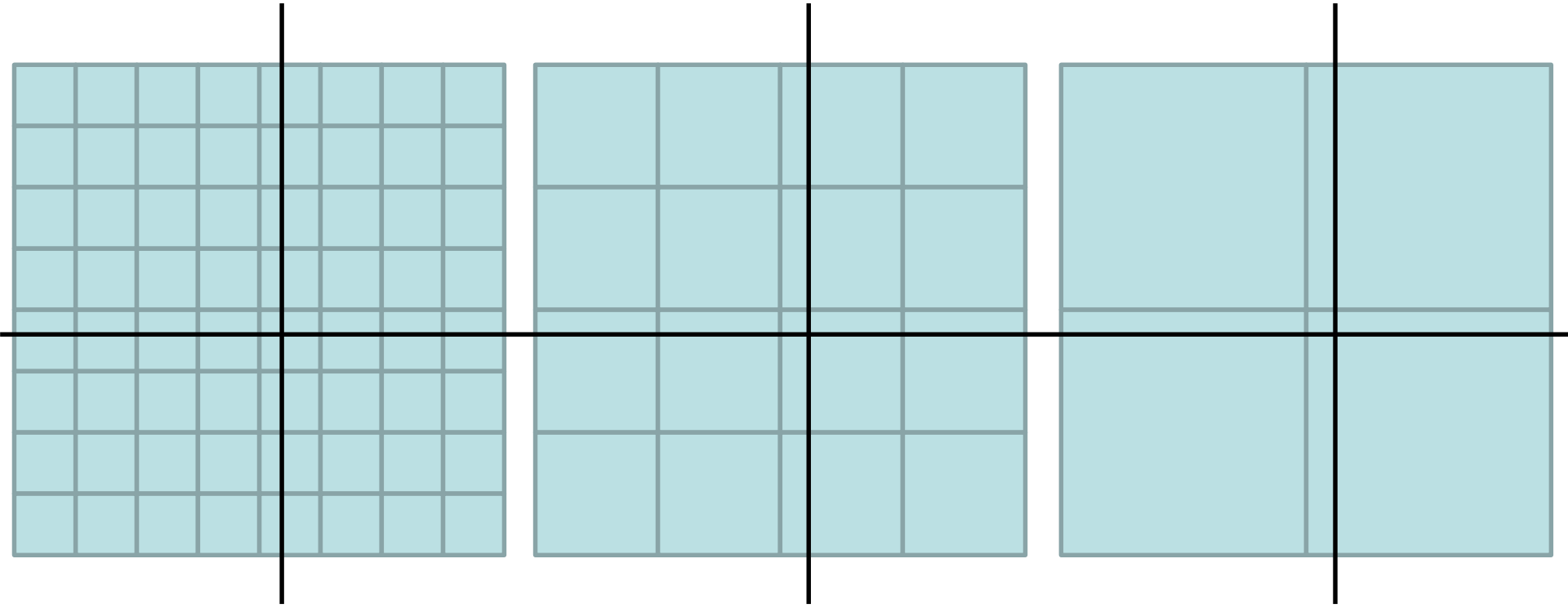
Jacobi smoothing:

(1) Each process performs Jacobi iteration (independently)

(2) Send messages to update ghost layers

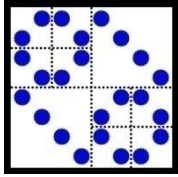
(3) Use communication/computation overlap for interior computations and exchange of boundary data

Restriction/Prolongation



Problems with work load: On coarse grids less computations, more communication!

Efficiency goes like 1/log(p) with the number of processors.



Modifications

Agglomeration: Coarse grid partitions are no longer aligned with the finer grid partitions (in order to avoid inefficiency on coarsest grids)

→ More communication in grid transfer, less in coarse grid solve.

Larger ghost layers can reduce the communication

Reduce number of V-cycle steps by using more powerful projections and smoothers or aggressive coarsening → less data transfer

Additive Multigrid

Consider the different levels at the same time,
Compute the related corrections in parallel and sum up the corrections.

$$x := x + \alpha \sum_{l=0}^{L} A_l^{-1} P_l (f - Ax)$$

Collect all matrices from different levels in one big linear system and apply smoothing on each level in parallel.

A_0
P_1, A_1
P_2, A_2
P_3, A_3
.
.
P_L, A_L

$$x = x_0 + x_1 + \ldots + x_L$$

Hybrid conjugate gradient iteration with MG as preconditioner
→ Less V-cycles