An introduction to domain decomposition methods

Victorita Dolean in collaboration with: F. Nataf, P. Jolivet, P.-H. Tournier 2nd June 2021



# Outline

## Introduction

Connection with the Block-Jacobi algorithm Discrete setting Convergence analysis Algebraic Schwarz Methods

Schwarz methods using Freefem++

- Schwarz algorithms as solvers
- Schwarz algorithms as preconditioners
- Schwarz preconditioners using FreeFEM++

# Outline

# Introduction

Connection with the Block-Jacobi algorithm

Discrete setting

Convergence analysis

Algebraic Schwarz Methods

Schwarz methods using Freefem++

# Applications



Time-harmonic Maxwell's equations

 $-i\omega\varepsilon\mathbf{E} + \nabla\times\mathbf{H} - \sigma\mathbf{E} = \mathbf{J},$  $i\omega\mu\mathbf{H} + \nabla\times\mathbf{E} = \mathbf{0}.$ 



# Applications



Time-harmonic Maxwell's equations

$$-i\omega\varepsilon\mathbf{E} + \nabla\times\mathbf{H} - \sigma\mathbf{E} = \mathbf{J},$$
  
$$i\omega\mu\mathbf{H} + \nabla\times\mathbf{E} = \mathbf{0}.$$



After discretisation  $\Rightarrow$  large linear system  $A\mathbf{u} = \mathbf{b}$ . Matrix A inherits the properties of the underlying PDE (symmetric, positive definite, indefinite, etc...) A is in general sparse, large and ill conditioned.

# $A\mathbf{u} = \mathbf{b}$ ? Landscape of linear solvers

**Direct Solvers** 

MUMPS (J.Y. L'Excellent), SuperLU (Demmel, ...), PastiX, UMFPACK, PARDISO (O. Schenk):

#### **Iterative Methods**

• Fixed point iteration: Jacobi, Gauss-Seidel, SSOR

• Krylov type methods: Conjuguate Gradient (Stiefel-Hestenes), GMRES (Y. Saad), QMR (R. Freund), MinRes, BiCGSTAB (van der Vorst):



Gauss	$d = 1 \qquad d = 2$		d = 3	
dense matrix	$O(n^3)$	$\mathcal{O}(n^3)$	$O(n^3)$	
using band structure	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{7/3})$	
using sparsity	$\mathcal{O}(n)$	$\mathcal{O}(n^{3/2})$	$\mathcal{O}(n^2)$	

Complexity of t	the Gauss	factorisation	(sparse	linear	algebra
-----------------	-----------	---------------	---------	--------	---------

As of today:  $n = 10^6$  (2d) and  $n = 10^5$  (3d) is ok.

Different sparse direct solvers

- PARDISO (http://www.pardiso-project.org)
- SUPERLU (https://portal.nersc.gov/project/sparse/superlu/)
- MUMPS (http://graal.ens-lyon.fr/MUMPS/)
- UMFPACK, in SuiteSparse (https://people.engr.tamu.edu/davis/suitesparse.html)

# Need & Opportunities for massively parallel computing

Moore's law: the number of transistors doubles every two years



Dennard scaling: the power density stays constant



Parallel computers are more and more available to scientists and engineers

- Apple, Linux, Windows laptops, 4/8 cores
- Desktop Computers, 64/128 cores
- Laboratory cluster, 300 cores
- $\bullet$  University cluster,  $\sim$  10 000 cores
- Cloud computing on Data Mining machines
- Supercomputers via academic or commercial providers > 100k cores

All fields of computer science are impacted.

Parallel computers are more and more available to scientists and engineers

- Apple, Linux, Windows laptops, 4/8 cores
- Desktop Computers, 64/128 cores
- Laboratory cluster, 300 cores
- $\bullet$  University cluster,  $\sim$  10 000 cores
- Cloud computing on Data Mining machines

• Supercomputers via academic or commercial providers > 100k cores

All fields of computer science are impacted.

#### Hardware news

#### ARM (Advanced Risc Machines) and Soc

- A64FX (Fujitsu), M1 (Apple), Graviton (Amazon)
- Soc: System on a Chip, HMB: High Bandwidth Memory
- Fugaku HPC exclusively based on A64FX ranks on TOP500 1st for dense linear algebra, 1st for sparse linear algebra, 1st for AI and 1st on Graph500)
- A64FX  $\gg$  GPU even for AI and ML thanks to its Scalable Vector Extension SVE
- Programmable with standard Fortran/C/C++, OpenMP and MPI

# Need & Opportunities for massively parallel computing

Parallel computers are more and more available to scientists and engineers

- Apple, Linux, Windows laptops, 4/8 cores
- Desktop Computers, 64/128 cores
- Laboratory cluster, 300 cores
- $\bullet$  University cluster,  $\sim$  10 000 cores
- Cloud computing on Data Mining machines

• Supercomputers via academic or commercial providers > 100k cores

All fields of computer science are impacted.

## Hardware news

- ARM (Advanced Risc Machines) and Soc
  - A64FX (Fujitsu), M1 (Apple), Graviton (Amazon)
  - Soc: System on a Chip, HMB: High Bandwidth Memory
  - Fugaku HPC exclusively based on A64FX ranks on TOP500 1st for dense linear algebra, 1st for sparse linear algebra, 1st for AI and 1st on Graph500 )
  - A64FX  $\gg$  GPU even for AI and ML thanks to its Scalable Vector Extension SVE
  - Programmable with standard Fortran/C/C++, OpenMP and MPI

#### Software news

#### Softwa<u>re</u>

- Some New High level languages support natively some degree of parallelism: Julia, Rust and wrap MPI library with a simpler interface
- Parallel linear algebra package: PETSc (USA), Scalapack (USA), HPDDM (France)
- Free Finite element/volume packages allow for parallel computing: OpenFoam (UK), FreeFem (France), Firedrake (UK), Dune (Germany),

## The First Domain Decomposition Method

The original Schwarz Method (H.A. Schwarz, 1870)

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

Schwarz Method :  $(u_1^n,u_2^n) \rightarrow (u_1^{n+1},u_2^{n+1})$  with

$$\begin{split} &-\Delta(u_1^{n+1}) = f \quad \text{in } \Omega_1 & -\Delta(u_2^{n+1}) = f \quad \text{in } \Omega_2 \\ &u_1^{n+1} = 0 \text{ on } \partial\Omega_1 \cap \partial\Omega & u_2^{n+1} = 0 \text{ on } \partial\Omega_2 \cap \partial\Omega \\ &u_1^{n+1} = u_2^n \quad \text{on } \partial\Omega_1 \cap \overline{\Omega_2}. & u_2^{n+1} = u_1^{n+1} \quad \text{on } \partial\Omega_2 \cap \overline{\Omega_1} \end{split}$$

Parallel algorithm, converges but very slowly, overlapping subdomains only. The parallel version is called Jacobi Schwarz method (JSM).

The algorithm acts on the local functions  $(u_i)_{i=1,2}$ . To make things global, we need:

The algorithm acts on the local functions  $(u_i)_{i=1,2}$ . To make things global, we need: • extension operators,  $E_i$ , s.t. for a function  $w_i : \Omega_i \mapsto \mathbb{R}$ ,  $E_i(w_i) : \Omega \mapsto \mathbb{R}$  is the extension of  $w_i$  by zero outside  $\Omega_i$ . The algorithm acts on the local functions  $(u_i)_{i=1,2}$ . To make things global, we need:

• extension operators,  $E_i$ , s.t. for a function  $w_i : \Omega_i \mapsto \mathbb{R}$ ,  $E_i(w_i) : \Omega \mapsto \mathbb{R}$  is the extension of  $w_i$  by zero outside  $\Omega_i$ .

• partition of unity functions  $\chi_i : \Omega_i \mapsto \mathbb{R}$ ,  $\chi_i \ge 0$  and  $\chi_i(x) = 0$  for  $x \in \partial \Omega_i$  and s.t.

$$w = \sum_{i=1}^{2} E_i(\chi_i w_{|\Omega_i}).$$

The algorithm acts on the local functions  $(u_i)_{i=1,2}$ . To make things global, we need:

• extension operators,  $E_i$ , s.t. for a function  $w_i : \Omega_i \mapsto \mathbb{R}$ ,  $E_i(w_i) : \Omega \mapsto \mathbb{R}$  is the extension of  $w_i$  by zero outside  $\Omega_i$ .

• partition of unity functions  $\chi_i : \Omega_i \mapsto \mathbb{R}$ ,  $\chi_i \ge 0$  and  $\chi_i(x) = 0$  for  $x \in \partial \Omega_i$  and s.t.

$$w = \sum_{i=1}^{2} E_i(\chi_i w_{|\Omega_i}).$$

Let  $u^n$  be an approximation to the solution to the global Poisson problem then  $u^{n+1}$  is computed by solving first local subproblems and then gluing them together.

Local problems to solve (i = 1, 2)

$$\begin{array}{ll} -\Delta(u_i^{n+1})=f & \text{ in } & \Omega_i \\ u_i^{n+1}=0 & \text{ on } & \partial\Omega_i\cap\partial\Omega \\ u_i^{n+1}=u^n & \text{ on } & \partial\Omega_i\cap\overline\Omega_{3-i}. \end{array}$$

Local problems to solve (i = 1, 2)

$$\begin{aligned} -\Delta(u_i^{n+1}) &= f & \text{in} \quad \Omega_i \\ u_i^{n+1} &= 0 & \text{on} \quad \partial \Omega_i \cap \partial \Omega \\ u_i^{n+1} &= u^n & \text{on} \quad \partial \Omega_i \cap \overline{\Omega}_{3-i} \end{aligned}$$

Two ways to "glue" solutions

• Using the partition of unity functions

Restricted Additive Schwarz (RAS) $u^{n+1} := \sum_{i=1}^2 E_i(\chi_i \, u_i^{n+1}) \, .$ 

Local problems to solve (i = 1, 2)

$$\begin{split} -\Delta(u_i^{n+1}) &= f & \text{ in } & \Omega_i \\ u_i^{n+1} &= 0 & \text{ on } & \partial\Omega_i \cap \partial\Omega \\ u_i^{n+1} &= u^n & \text{ on } & \partial\Omega_i \cap \overline{\Omega}_{3-i} \end{split}$$

Two ways to "glue" solutions

• Using the partition of unity functions

Restricted Additive Schwarz (RAS) $u^{n+1} := \sum_{i=1}^{2} E_i(\chi_i \, u_i^{n+1}) \, .$ 

• Not based on the partition of unity

Additive Schwarz (ASM)
$$u^{n+1} := \sum_{i=1}^2 E_i(u_i^{n+1}) \, .$$

Let us consider a linear system:

$$A\mathbf{U} = \mathbf{F}$$

with a matrix A of size  $m \times m$ , a right handside  $F \in \mathbb{R}^m$  and a solution  $\mathbf{U} \in \mathbb{R}^m$  where m is an integer. Let D be the diagonal of A:

 $D\mathbf{U}^{n+1} = D\mathbf{U}^n + (b - A\mathbf{U}^n),$ 

or equivalently,

 $\mathbf{U}^{n+1} = \mathbf{U}^n + D^{-1}(b - A\mathbf{U}^n) = \mathbf{U}^n + D^{-1}\mathbf{r}^n$ ,

where  $\mathbf{r}^n$  is the residual of the equation.

Let us consider a linear system:

 $A\mathbf{U} = \mathbf{F}$ 

with a matrix A of size  $m \times m$ , a right handside  $F \in \mathbb{R}^m$  and a solution  $\mathbf{U} \in \mathbb{R}^m$  where m is an integer. Let D be the diagonal of A:

 $D\mathbf{U}^{n+1} = D\mathbf{U}^n + (b - A\mathbf{U}^n),$ 

or equivalently,

 $\mathbf{U}^{n+1} = \mathbf{U}^n + D^{-1}(b - A\mathbf{U}^n) = \mathbf{U}^n + D^{-1}\mathbf{r}^n ,$ 

where  $\mathbf{r}^n$  is the residual of the equation.

# The block-Jacobi algorithm

The set of indices  $\{1,\ldots,m\}$  is partitioned into two sets

$$\mathcal{N}_1 := \{1, \dots, m_s\}$$
 and  $\mathcal{N}_2 := \{m_s + 1, \dots, m\}$ .

Let  $\mathbf{U}_1:=\mathbf{U}_{|\mathcal{N}_1}$ ,  $\mathbf{U}_2:=\mathbf{U}_{|\mathcal{N}_2}$  and similarly  $\mathbf{F}_1:=\mathbf{F}_{|\mathcal{N}_1},\,\mathbf{F}_2:=\mathbf{F}_{|\mathcal{N}_2}.$ 

Let us consider a linear system:

$$A\mathbf{U} = \mathbf{F}$$

with a matrix A of size  $m \times m$ , a right handside  $F \in \mathbb{R}^m$  and a solution  $\mathbf{U} \in \mathbb{R}^m$  where m is an integer. Let D be the diagonal of A:

 $D\mathbf{U}^{n+1} = D\mathbf{U}^n + (b - A\mathbf{U}^n),$ 

or equivalently,

$$\mathbf{U}^{n+1} = \mathbf{U}^n + D^{-1}(b - A\mathbf{U}^n) = \mathbf{U}^n + D^{-1}\mathbf{r}^n,$$

where  $\mathbf{r}^n$  is the residual of the equation.

### The block-Jacobi algorithm

The set of indices  $\{1,\ldots,m\}$  is partitioned into two sets

$$\mathcal{N}_1 := \{1, \dots, m_s\}$$
 and  $\mathcal{N}_2 := \{m_s + 1, \dots, m\}$ .

Let  $\mathbf{U}_1 := \mathbf{U}_{|\mathcal{N}_1}$ ,  $\mathbf{U}_2 := \mathbf{U}_{|\mathcal{N}_2}$  and similarly  $\mathbf{F}_1 := \mathbf{F}_{|\mathcal{N}_1}$ ,  $\mathbf{F}_2 := \mathbf{F}_{|\mathcal{N}_2}$ . The linear system has the following block form:

$$\left(\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right) \left(\begin{array}{c} \mathbf{U}_1 \\ \mathbf{U}_2 \end{array}\right) = \left(\begin{array}{c} \mathbf{F}_1 \\ \mathbf{F}_2 \end{array}\right)$$

where  $A_{ij} := A_{|\mathcal{N}_i \times \mathcal{N}_j}$ ,  $1 \le i, j \le 2$ .

Let us consider a linear system:

 $A\mathbf{U} = \mathbf{F}$ 

with a matrix A of size  $m \times m$ , a right handside  $F \in \mathbb{R}^m$  and a solution  $\mathbf{U} \in \mathbb{R}^m$  where m is an integer. Let D be the diagonal of A:

 $D\mathbf{U}^{n+1} = D\mathbf{U}^n + (b - A\mathbf{U}^n),$ 

or equivalently,

$$\mathbf{U}^{n+1} = \mathbf{U}^n + D^{-1}(b - A\mathbf{U}^n) = \mathbf{U}^n + D^{-1}\mathbf{r}^n,$$

where  $\mathbf{r}^n$  is the residual of the equation.

## The block-Jacobi algorithm

The set of indices  $\{1,\ldots,m\}$  is partitioned into two sets

$$\mathcal{N}_1 := \{1, \dots, m_s\}$$
 and  $\mathcal{N}_2 := \{m_s + 1, \dots, m\}$ .

Let  $\mathbf{U}_1 := \mathbf{U}_{|\mathcal{N}_1}$ ,  $\mathbf{U}_2 := \mathbf{U}_{|\mathcal{N}_2}$  and similarly  $\mathbf{F}_1 := \mathbf{F}_{|\mathcal{N}_1}$ ,  $\mathbf{F}_2 := \mathbf{F}_{|\mathcal{N}_2}$ . The block-Jacobi algorithm reads:

$$\begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix} \begin{pmatrix} \mathbf{U}_1^{n+1} \\ \mathbf{U}_2^{n+1} \end{pmatrix} = \begin{pmatrix} F_1 - A_{12} \mathbf{U}_2^n \\ F_2 - A_{21} \mathbf{U}_1^n \end{pmatrix}.$$

Let us consider a linear system:

 $A\mathbf{U} = \mathbf{F}$ 

with a matrix A of size  $m \times m$ , a right handside  $F \in \mathbb{R}^m$  and a solution  $\mathbf{U} \in \mathbb{R}^m$  where m is an integer. Let D be the diagonal of A:

 $D\mathbf{U}^{n+1} = D\mathbf{U}^n + (b - A\mathbf{U}^n),$ 

or equivalently,

$$\mathbf{U}^{n+1} = \mathbf{U}^n + D^{-1}(b - A\mathbf{U}^n) = \mathbf{U}^n + D^{-1}\mathbf{r}^n,$$

where  $\mathbf{r}^n$  is the residual of the equation.

## The block-Jacobi algorithm

The set of indices  $\{1,\ldots,m\}$  is partitioned into two sets

$$\mathcal{N}_1 := \{1, \dots, m_s\}$$
 and  $\mathcal{N}_2 := \{m_s + 1, \dots, m\}$ .

Let  $\mathbf{U}_1 := \mathbf{U}_{|\mathcal{N}_1}$ ,  $\mathbf{U}_2 := \mathbf{U}_{|\mathcal{N}_2}$  and similarly  $\mathbf{F}_1 := \mathbf{F}_{|\mathcal{N}_1}$ ,  $\mathbf{F}_2 := \mathbf{F}_{|\mathcal{N}_2}$ . Let  $\mathbf{U}^n = (\mathbf{U}_1^n, \mathbf{U}_2^n)^T$ , algorithm becomes

$$\begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix} \mathbf{U}^{n+1} = F - \begin{pmatrix} 0 & A_{12} \\ A_{21} & 0 \end{pmatrix} \mathbf{U}^n$$

Let us consider a linear system:

 $A\mathbf{U} = \mathbf{F}$ 

with a matrix A of size  $m \times m$ , a right handside  $F \in \mathbb{R}^m$  and a solution  $\mathbf{U} \in \mathbb{R}^m$  where m is an integer. Let D be the diagonal of A:

 $D\mathbf{U}^{n+1} = D\mathbf{U}^n + (b - A\mathbf{U}^n),$ 

or equivalently,

$$\mathbf{U}^{n+1} = \mathbf{U}^n + D^{-1}(b - A\mathbf{U}^n) = \mathbf{U}^n + D^{-1}\mathbf{r}^n,$$

where  $\mathbf{r}^n$  is the residual of the equation.

# The block-Jacobi algorithm

The set of indices  $\{1,\ldots,m\}$  is partitioned into two sets

$$\mathcal{N}_1 := \{1, \dots, m_s\}$$
 and  $\mathcal{N}_2 := \{m_s + 1, \dots, m\}$ .

Let  $\mathbf{U}_1 := \mathbf{U}_{|\mathcal{N}_1}$ ,  $\mathbf{U}_2 := \mathbf{U}_{|\mathcal{N}_2}$  and similarly  $\mathbf{F}_1 := \mathbf{F}_{|\mathcal{N}_1}$ ,  $\mathbf{F}_2 := \mathbf{F}_{|\mathcal{N}_2}$ . Or equivalently

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \begin{pmatrix} A_{11} & 0\\ 0 & A_{22} \end{pmatrix}^{-1} \mathbf{r}^n$$

where

$$\mathbf{r}^n := F - A \mathbf{U}^n, \mathbf{r}^n_i := \mathbf{r}^n_{|\mathcal{N}_i}, \ i = 1, 2.$$

#### Notations for a compact form

- $R_1$  the restriction operator from  $\mathcal{N}$  into  $\mathcal{N}_1$
- $R_2$  the restriction operator from  $\mathcal{N}$  into  $\mathcal{N}_2$ . The transpose operator  $R_1^T$  is an extension operator from  $\mathcal{N}_1$  into  $\mathcal{N}$  and the same holds for  $R_2^T$ .

Block-Jacobi in compact form  $\mathbf{U}^{n+1} = \mathbf{U}^n + (R_1^T A_1^{-1} R_1 + R_2^T A_2^{-1} R_2) \mathbf{r}^n .$ where  $\mathbf{r}^n := \mathbf{F} - A \mathbf{U}^n , \mathbf{r}_i^n := \mathbf{r}_{|\mathcal{N}_i}^n, \ i = 1, 2 .$   $A_i = R_i A R_i^T, \ i = 1, 2 .$ 

# Schwarz algorithms as block Jacobi methods

Let  $\Omega:=(0,1)$  and consider the following  $\mathsf{BVP}$ 

$$\begin{split} -\Delta u &= f \text{ in } \Omega \\ u(0) &= u(1) = 0 \,. \end{split}$$

discretized by a three point FD on the grid  $x_j := j h, 1 \le j \le m$  where h := 1/(m + 1). Let  $u_j \simeq u(x_j), f_j := f(x_j), 1 \le j \le m$  and the discrete problem

$$A\mathbf{U} = F, \, \mathbf{U} = (u_j)_{1 \le j \le m}, \, F = (f_j)_{1 \le j \le m}$$

where  $A_{jj} := 2/h^2$  and  $A_{jj+1} = A_{j+1j} := -1/h^2$ . Let domains  $\Omega_1 := (0, (m_s + 1) h)$  and  $\Omega_2 := (m_s h, 1)$  define an overlapping decomposition with a minimal overlap of width h. The discretization of the Jacobi-Schwarz for domain  $\Omega_1$  reads

$$\begin{pmatrix}
-\frac{u_{1,j-1}^{n+1} - 2u_{1,j}^{n+1} + u_{1,j+1}^{n+1}}{h^2} = f_j, \ 1 \le j \le m_s \\
u_{1,0}^{n+1} = 0 \\
u_{1,m_s+1}^{n+1} = u_{2,m_s+1}^n
\end{cases}$$

Solving for  $\mathbf{U}_1^{n+1} = (u_{1,j}^{n+1})_{1 \leq j \leq m_s}$  corresponds to solving a Dirichlet BVP in subdomain  $\Omega_1$  with Dirichlet data taken from the other subdomain at the previous step.

 $A_{11}\mathbf{U}_1^{n+1} + A_{12}\mathbf{U}_2^n = F_1,$  $A_{22}\mathbf{U}_2^{n+1} + A_{21}\mathbf{U}_1^n = F_2.$ 

# Schwarz as block Jacobi methods - III

The discrete counterpart of the extension operator  $E_1$  (resp.  $E_2$ ) is defined by  $E_1(U_1) = (U_1, 0)^T$  (resp.  $E_2(U_2) = (0, U_2)^T$ ).



then 
$$E_1(U_1) + E_2(U_2) = E_1(\chi_1 U_1) + E_2(\chi_2 U_2) = \begin{pmatrix} U_1 \\ U_2 \end{pmatrix}$$
.

When the overlap is minimal, the discrete counterparts of the three Schwarz methods (AS, RAS, JS) are equivalent to the same block Jacobi algorithm.

#### **Continuous level**

 $\bullet$  Domain  $\Omega$  and an overlapping decomposition

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

- A function  $u: \Omega \to \mathbb{R}$ .
- Restriction of  $u: \Omega \to \mathbb{R}$  to  $\Omega_i$ ,  $1 \le i \le N$ .
- The extension operator  $E_i$  of a function

$$u_i:\Omega_i\to\mathbb{R}$$

to a function  $\Omega \to \mathbb{R}$ .

• Partition of unity functions  $\chi_i$ ,  $1 \le i \le N$ .

#### Continuous level

#### Discrete level

• Domain  $\Omega$  and an overlapping decomposition

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

- A function  $u: \Omega \to \mathbb{R}$ .
- Restriction of  $u: \Omega \to \mathbb{R}$  to  $\Omega_i, 1 \leq i \leq N$ .
- The extension operator  $E_i$  of a function

$$u_i:\Omega_i\to\mathbb{R}$$

to a function  $\Omega \to \mathbb{R}$ .

• Partition of unity functions 
$$\chi_i$$
,  $1 \le i \le N$ .

• A set of d.o.f. 
$${\mathcal N}$$
 and a decomposition

$$\mathcal{N} = \cup_{i=1}^{N} \mathcal{N}_i.$$

- A vector  $U \in \mathbb{R}^{\#\mathcal{N}}$ .
- Restriction  $R_i$   $\#\mathcal{N}_i \times \#\mathcal{N}$  boolean matrix • The extension matrix  $R_i^T$ .
- Partition of unity diagonal matrices with positive entries, of size  $\#\mathcal{N}_i \times \#\mathcal{N}_i$  s. t.

$$Id = \sum_{i=1}^{N} R_i^T D_i R_i.$$

Let  $(\mathcal{T}_{h,i})_{1 \leq i \leq N}$  define an overlapping mesh decomposition of  $\mathcal{T}_h$  and  $\Omega_i := \bigcup_{K \in \mathcal{T}_{h,i}} K$ . Define  $V_{h,i}$  the finite element space on  $\mathcal{T}_{h,i}$ .

Let  $\mathcal{T}_h$  be a mesh of a domain  $\Omega$  and  $u_h$  some discretization of a function u which is the solution of an elliptic Dirichlet BVP.

Find  $\mathbf{U} \in \mathbb{R}^{\#\mathcal{N}}$ s.t.  $A\mathbf{U} = F$ .

Define the restriction operator  $r_i = E_i^T$ :

 $r_i: u_h \mapsto u_{h|\Omega_i}$ 

 $R_i$  boolean matrix corresponding to  $r_i$ :

$$R_i := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots \end{bmatrix}$$
$$R_i : \mathbb{R}^{\#\mathcal{N}} \longmapsto \mathbb{R}^{\#\mathcal{N}_i}$$

$$R_i: \mathbb{R}^{\#\mathcal{N}} \longmapsto \mathbb{R}^{\#\mathcal{N}_i}$$

 $R_i: \mathbb{R}^{\#\mathcal{N}} \longmapsto \mathbb{R}^{\#\mathcal{N}_i}$ 

## Prolongation operator

 $R_i^T : \mathbb{R}^{\#\mathcal{N}_i} \longmapsto \mathbb{R}^{\#\mathcal{N}}.$ 

 $R_i: \mathbb{R}^{\#\mathcal{N}} \longmapsto \mathbb{R}^{\#\mathcal{N}_i}$ 

## Prolongation operator

 $R_i^T : \mathbb{R}^{\#\mathcal{N}_i} \longmapsto \mathbb{R}^{\#\mathcal{N}}.$ 

Local Dirichlet matrices

 $A_i := R_i A R_i^T \,.$ 

$$R_i: \mathbb{R}^{\#\mathcal{N}} \longmapsto \mathbb{R}^{\#\mathcal{N}_i}$$

Prolongation operator

 $R_i^T : \mathbb{R}^{\#\mathcal{N}_i} \longmapsto \mathbb{R}^{\#\mathcal{N}}.$ 

Local Dirichlet matrices

 $A_i := R_i A R_i^T \,.$ 

Partition of unity defined by matrices  $D_i$ 

$$D_i : \mathbb{R}^{\#\mathcal{N}_i} \longmapsto \mathbb{R}^{\#\mathcal{N}_i}$$
$$\sum_{i=1}^N R_i^T D_i R_i = Id$$

Let  $\mathcal{N}:=\{1,\ldots,5\}$  be partitioned into  $\mathcal{N}_1:=\{1,2,3\} \text{ and } \mathcal{N}_2:=\{4,5\}\,.$ 


## Examples. Two subdomain case: 1d algebraic setting

Let  $\mathcal{N} := \{1, \dots, 5\}$  be partitioned into  $\mathcal{N}_1 := \{1, 2, 3\}$  and  $\mathcal{N}_2 := \{4, 5\}$ .



Restriction/partition of unity matrices  $R_1$ ,  $R_2$ ,  $D_1$  and  $D_2$ :

$$R_{1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \text{ and } R_{2} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
$$D_{1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } D_{2} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$



#### Two subdomain case: 1d algebraic setting II



Restriction/partition of unity matrices  $R_1$ ,  $R_2$ ,  $D_1$  and  $D_2$ :

$$R_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \text{ and } R_2 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$
$$D_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 \end{pmatrix} \text{ and } D_2 = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

1/2

0 0 0

Intro

## Partition of the 1D mesh corresponds to an ovr. decomp. of $\mathcal{N}$ : $\mathcal{N}_1 := \{1, 2, 3\}$ and $\mathcal{N}_2 := \{3, 4, 5\}$ .



### Two subdomain case: 1d finite element decomposition

Partition of the 1D mesh corresponds to an ovr. decomp. of  $\mathcal{N}$ :

$$\mathcal{N}_1 := \{1, 2, 3\}$$
 and  $\mathcal{N}_2 := \{3, 4, 5\}$ 



Restriction/partition of unity matrices  $R_1$ ,  $R_2$ ,  $D_1$  and  $D_2$ :

$$R_{1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \text{ and } R_{2} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$
$$D_{1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/2 \end{pmatrix} \text{ and } D_{2} = \begin{pmatrix} 1/2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

An overlapping partition.

$$\mathcal{N}_1^{\delta=1} := \{1, 2, 3, 4\} \text{ and } \mathcal{N}_2^{\delta=1} := \{2, 3, 4, 5\}.$$



#### Two subdomain case: 1d finite element decomposition - II

0

0 0 An overlapping partition.

$$\mathcal{N}_1^{\delta=1}:=\{1,2,3,4\} \text{ and } \mathcal{N}_2^{\delta=1}:=\{2,3,4,5\}$$
 .



Restriction/partition of unity matrices  $R_1$ ,  $R_2$ ,  $D_1$  and  $D_2$ :

0

0

0

0

0

1/2

Intro

The set of indices  ${\cal N}$  can be partitioned by an automatic graph partitioner such as METIS or SCOTCH.

The set of indices N can be partitioned by an automatic graph partitioner such as **METIS** or **SCOTCH**. • From the input matrix A, a connectivity graph is created. The set of indices N can be partitioned by an automatic graph partitioner such as **METIS** or **SCOTCH**. • From the input matrix A, a connectivity graph is created.

• Two indices  $i, j \in \mathcal{N}$  are connected if the matrix coefficient  $A_{ij} \neq 0$ .

The set of indices  ${\cal N}$  can be partitioned by an automatic graph partitioner such as METIS or SCOTCH.

- From the input matrix A, a connectivity graph is created.
- Two indices  $i, j \in \mathcal{N}$  are connected if the matrix coefficient  $A_{ij} \neq 0$ .
- Even if matrix A is not symmetric, the connectivity graph is symmetrized.

The set of indices  ${\cal N}$  can be partitioned by an automatic graph partitioner such as METIS or SCOTCH.

- From the input matrix A, a connectivity graph is created.
- Two indices  $i, j \in \mathcal{N}$  are connected if the matrix coefficient  $A_{ij} \neq 0$ .
- Even if matrix A is not symmetric, the connectivity graph is symmetrized.
- Algorithms that find a good partitioning of highly unstructured graphs are used.

The set of indices  $\mathcal{N}$  can be partitioned by an automatic graph partitioner such as **METIS** or **SCOTCH**.

- From the input matrix A, a connectivity graph is created.
- Two indices  $i, j \in \mathcal{N}$  are connected if the matrix coefficient  $A_{ij} \neq 0$ .
- Even if matrix A is not symmetric, the connectivity graph is symmetrized.
- Algorithms that find a good partitioning of highly unstructured graphs are used.
- This distribution must be done so that the number of elements assigned to each processor is roughly the same (balance the computations among the processors).

The set of indices  $\mathcal N$  can be partitioned by an automatic graph partitioner such as METIS or SCOTCH.

- From the input matrix A, a connectivity graph is created.
- Two indices  $i, j \in \mathcal{N}$  are connected if the matrix coefficient  $A_{ij} \neq 0$ .
- Even if matrix A is not symmetric, the connectivity graph is symmetrized.
- Algorithms that find a good partitioning of highly unstructured graphs are used.
- This distribution must be done so that the number of elements assigned to each processor is roughly the same (balance the computations among the processors).
- The number of adjacent elements assigned to different processors is minimized (minimize the communication between different processors).

## Multi-D algebraic setting

Extend each disjoint subset  $\mathcal{N}_i$  with its direct neighbors to form  $\mathcal{N}_i^{\delta=1}$ .

$$\mathcal{N} := \bigcup_{i=1}^N \mathcal{N}_i, \quad \mathcal{N}_i \cap \mathcal{N}_j = \emptyset \text{ for } i \neq j.$$



 $R_i$  be the restriction matrix from set  $\mathcal{N}$  to the subset  $\mathcal{N}_i^{\delta=1}$ . Partition of unity:  $D_i$  a diagonal matrix of size  $\#\mathcal{N}_i^{\delta=1}\times\#\mathcal{N}_i^{\delta=1}$ ,  $1\leq i\leq N$  such that

$$(D_i)_{jj} := 1/\#\mathcal{M}_j, \ \mathcal{M}_j := \{1 \le i \le N | \ j \in \mathcal{N}_i^{\delta=1}\}.$$



#### Computational domain



Create overlapping subdomains resolved by the finite element meshes:

$$\Omega_i = \bigcup_{\tau \in \mathcal{T}_{i,h}} \tau \quad \text{for } 1 \le i \le N \,. \tag{1}$$



Let  $\{\phi_k\}_{k\in\mathcal{N}}$  be a basis of the finite element space.

 $\mathcal{N}_i := \{k \in \mathcal{N} : \mathsf{supp}(\phi_k) \cap \Omega_i \neq \emptyset\}.$ 

For all degree of freedom  $k \in \mathcal{N}$ , define

 $\mu_k := \# \left\{ j: \, 1 \leq j \leq N \text{ and } \operatorname{supp}(\phi_k) \cap \Omega_j \neq \emptyset \right\}.$ 



Let  $\{\phi_k\}_{k\in\mathcal{N}}$  be a basis of the finite element space.

$$\mathcal{N}_i := \{k \in \mathcal{N} : \mathsf{supp}(\phi_k) \cap \Omega_i \neq \emptyset\}.$$

For all degree of freedom  $k \in \mathcal{N}$ , define

 $\mu_k := \# \left\{ j: \ 1 \le j \le N \text{ and } \operatorname{supp}(\phi_k) \cap \Omega_j \neq \emptyset \right\}.$ 

 $R_i$  be the restriction matrix from  $\mathcal{N}$  to  $\mathcal{N}_i$ .

 $\begin{array}{l} \mbox{Partition of unity } D_i: \mbox{ diagonal matrix of size} \\ \#\mathcal{N}_i \times \#\mathcal{N}_i, \ 1 \leq i \leq N \ {\rm s.t.} \end{array}$ 

 $(D_i)_{kk} := 1/\mu_k, \ k \in \mathcal{N}_i.$ 

Let the discretised Poisson problem:  $A\mathbf{U} = \mathbf{F} \in \mathbb{R}^n$ .



Let the discretised Poisson problem:  $A\mathbf{U} = \mathbf{F} \in \mathbb{R}^n$ . Given a decomposition of  $[\![1;n]\!]$ ,  $(\mathcal{N}_1, \mathcal{N}_2)$ , define: the restriction operator  $R_i$  from  $\mathbb{R}^{[\![1;n]\!]}$  into  $\mathbb{R}^{\mathcal{N}_i}$ ,  $R_i^T$  as the extension by 0 from  $\mathbb{R}^{\mathcal{N}_i}$  into  $\mathbb{R}^{[\![1;n]\!]}$ .



Let the discretised Poisson problem:  $A\mathbf{U} = \mathbf{F} \in \mathbb{R}^n$ . Given a decomposition of  $[\![1;n]\!]$ ,  $(\mathcal{N}_1, \mathcal{N}_2)$ , define: the restriction operator  $R_i$  from  $\mathbb{R}^{[\![1;n]\!]}$  into  $\mathbb{R}^{\mathcal{N}_i}$ ,  $R_i^T$  as the extension by 0 from  $\mathbb{R}^{\mathcal{N}_i}$  into  $\mathbb{R}^{[\![1;n]\!]}$ . Find  $\mathbf{U}^m \longrightarrow \mathbf{U}^{m+1}$  by solving concurrently:

$$\mathbf{U}_{j}^{m+1} = \mathbf{U}_{j}^{m} + A_{j}^{-1} R_{j} (\mathbf{F} - A \mathbf{U}^{m}), \ j = 1, 2$$

where  $\mathbf{U}_i^m = R_i \mathbf{U}^m$  and  $A_i := R_i A R_i^T$ .



## An introduction to Additive Schwarz II

We have effectively divided, but we have yet to conquer.

Duplicated unknowns coupled via a partition of unity:



## An introduction to Additive Schwarz II

We have effectively divided, but we have yet to conquer.

Duplicated unknowns coupled via a partition of unity:



Then, 
$$\mathbf{U}^{m+1} = \sum_{i=1}^{N} R_i^T D_i \mathbf{U}_i^{m+1}.$$

#### An introduction to Additive Schwarz II

We have effectively divided, but we have yet to conquer.

Duplicated unknowns coupled via a partition of unity:



Schwarz algorithm iterates on a pair of local functions  $(u_m^1, u_m^2)$ RAS algorithm iterates on the global function  $u^m$  Schwarz algorithm iterates on a pair of local functions  $(u_m^1, u_m^2)$ RAS algorithm iterates on the global function  $u^m$ 

Schwarz and RAS (Efstathiou and Gander, 2002)  
Discretization of the classical Schwarz algorithm and the iterative RAS algorithm:  
$$\mathbf{U}^{n+1} = \mathbf{U}^n + M_{RAS}^{-1}\mathbf{r}^n, \mathbf{r}^n := \mathbf{F} - A \mathbf{U}^n.$$
are equivalent
$$\mathbf{U}^n = R_1^T D_1 \mathbf{U}_1^n + R_2^T D_2 \mathbf{U}_2^n.$$

Operator  $M_{RAS}^{-1}$  is used as a preconditioner in Krylov methods for non symmetric problems.

$$M_{RAS}^{-1} := \sum_{i=1}^{N} R_i^T D_i A_i^{-1} R_i.$$

A symmetrized version: Additive Schwarz Method (ASM),

$$M_{ASM}^{-1} := \sum_{i=1}^{N} R_i^T A_i^{-1} R_i$$
(2)

are used as a preconditioner for the conjugate gradient (CG) method. Later on, we introduce

$$M_{SORAS}^{-1} := \sum_{i=1}^{N} R_i^T D_i B_i^{-1} D_i R_i$$
(3)

where  $(B_i)_{1 \le i \le N}$  are some local invertible matrices. Although RAS is more efficient, ASM is amenable to theory. (1)

Let L > 0,  $\Omega = (0, L)$  be decomposed into two subodmains  $\Omega_1 := (0, L_1)$  and  $\Omega_2 := (l_2, L)$  with  $l_2 \le L_1$ . The error  $e_i^n := u_i^n - u_{|\Omega_i}$ , i = 1, 2 satisfies

$$-\frac{d^2 e_1^{n+1}}{dx_1^2} = 0, x \in (0, L_1)$$
  
$$e_1^{n+1}(0) = 0$$
  
$$e_1^{n+1}(L_1) = e_2^n(L_1)$$

Let L > 0,  $\Omega = (0, L)$  be decomposed into two subodmains  $\Omega_1 := (0, L_1)$  and  $\Omega_2 := (l_2, L)$  with  $l_2 \le L_1$ . The error  $e_i^n := u_i^n - u_{|\Omega_i|}$ , i = 1, 2 satisfies

$$-\frac{d^2 e_2^{n+1}}{dx^2} = 0, x \in (l_2, L)$$
  
$$e_2^{n+1}(l_2) = e_1^{n+1}(l_2)$$
  
$$e_2^{n+1}(L) = 0.$$

Errors are affine functions in each subdomain:

$$e_1^{n+1}(x) = e_2^n(L_1) \frac{x}{L_1} \\ e_2^{n+1}(x) = e_1^{n+1}(l_2) \frac{L-x}{L-l_2}$$

Thus, we have

$$e_2^{n+1}(L_1) = e_1^{n+1}(l_2) \frac{L-L_1}{L-l_2} = e_2^n(L_1) \frac{l_2}{L_1} \frac{L-L_1}{L-l_2}.$$

Errors are affine functions in each subdomain:

$$\begin{array}{rcl} e_1^{n+1}(x) & = & e_2^n(L_1) \, \frac{x}{L_1} \\ e_2^{n+1}(x) & = & e_1^{n+1}(l_2) \, \frac{L-x}{L-l_2} \end{array}$$

Thus, we have

$$e_2^{n+1}(L_1) = e_1^{n+1}(l_2) \frac{L-L_1}{L-l_2} = e_2^n(L_1) \frac{l_2}{L_1} \frac{L-L_1}{L-l_2}.$$

Let  $\delta := L_1 - l_2$  (overlap). Interface iteration

$$e_2^{n+1}(L_1) = \frac{l_2}{l_2+\delta} \frac{L-l_2-\delta}{L-l_2} e_2^n(L_1) \\ = \frac{1-\delta/(L-l_2)}{1+\delta/l_2} e_2^n(L_1).$$

It is clear that  $\delta>0$  is sufficient and necessary to have convergence.

Errors are affine functions in each subdomain:

$$\begin{array}{rcl} e_1^{n+1}(x) & = & e_2^n(L_1) \, \frac{x}{L_1} \\ e_2^{n+1}(x) & = & e_1^{n+1}(l_2) \, \frac{L-x}{L-l_2} \end{array}$$

Thus, we have

$$e_2^{n+1}(L_1) = e_1^{n+1}(l_2) \frac{L-L_1}{L-l_2} = e_2^n(L_1) \frac{l_2}{L_1} \frac{L-L_1}{L-l_2}.$$

Let  $\delta := L_1 - l_2$  (overlap). Interface iteration

$$e_2^{n+1}(L_1) = \frac{l_2}{l_2+\delta} \frac{L-l_2-\delta}{L-l_2} e_2^n(L_1) = \frac{1-\delta/(L-l_2)}{1+\delta/l_2} e_2^n(L_1) .$$

It is clear that  $\delta>0$  is sufficient and necessary to have convergence.



Let  $\mathbb{R}^2$  decomposed into two half-planes  $\Omega_1 = (-\infty, \delta) \times \mathbb{R}$  and  $\Omega_2 = (0, \infty) \times \mathbb{R}$  with an overlap of size  $\delta > 0$  and the problem

 $\begin{array}{rl} (\eta-\Delta)(u) &=f \quad \mbox{in} \quad \mathbb{R}^2, \\ u & \mbox{is bounded at infinity}\,, \end{array}$ 

Let  $\mathbb{R}^2$  decomposed into two half-planes  $\Omega_1=(-\infty,\delta)\times\mathbb{R}$  and  $\Omega_2=(0,\infty)\times\mathbb{R}$  with an overlap of size  $\delta>0$  and the problem

 $\begin{array}{rl} (\eta-\Delta)(u) &=f \quad \mbox{in} \quad \mathbb{R}^2, \\ u & \mbox{is bounded at infinity}\,, \end{array}$ 

By linearity, the errors  $e_i^n := u_i^n - u|_{\Omega_i}$  satisfy

 $\begin{array}{ll} (\eta-\Delta)(e_1^{n+1}) &= 0 \quad \text{in} \quad \Omega_1 \\ e_1^{n+1} & \text{is bounded at infinity} \\ e_1^{n+1}(\delta,y) &= e_2^n(\delta,y), \end{array}$
$\begin{array}{rl} (\eta-\Delta)(u) &=f \quad \mbox{in} \quad \mathbb{R}^2, \\ u & \mbox{is bounded at infinity}\,, \end{array}$ 

By linearity, the errors  $e_i^n := u_i^n - u|_{\Omega_i}$  satisfy

$$\begin{array}{rll} (\eta-\Delta)(e_2^{n+1}) &= 0 \quad \text{in} \quad \Omega_2 \\ & e_2^{n+1} & \text{is bounded at infinity} \\ & e_2^{n+1}(0,y) &= e_1^n(0,y). \end{array}$$

 $\begin{array}{rl} (\eta-\Delta)(u) &=f \quad \mbox{in} \quad \mathbb{R}^2,\\ u & \mbox{is bounded at infinity}\,, \end{array}$ 

Partial Fourier transform of the equation in the  $\boldsymbol{y}$  direction:

$$\left(\eta - \frac{\partial^2}{\partial x^2} + k^2\right) (\hat{e}_j^{n+1}(x,k)) = 0 \quad \text{in} \quad \Omega_j.$$

 $\begin{array}{rl} (\eta-\Delta)(u) &=f \quad \mbox{in} \quad \mathbb{R}^2,\\ u & \mbox{is bounded at infinity}\,, \end{array}$ 

Partial Fourier transform of the equation in the  $\boldsymbol{y}$  direction:

$$\left(\eta - \frac{\partial^2}{\partial x^2} + k^2\right) (\hat{e}_j^{n+1}(x,k)) = 0 \quad \text{in} \quad \Omega_j.$$

$$(\eta - \Delta)(u) = f$$
 in  $\mathbb{R}^2$ ,  
 $u$  is bounded at infinity.

Partial Fourier transform of the equation in the  $\boldsymbol{y}$  direction:

$$\left(\eta - \frac{\partial^2}{\partial x^2} + k^2\right) (\hat{e}_j^{n+1}(x,k)) = 0 \quad \text{in} \quad \Omega_j.$$

For a given k, solutions

$$\hat{e}_{j}^{n+1}(x,k) = \gamma_{+}^{n+1}(k)^{\lambda^{+}(k)x} + \gamma_{-}^{n+1}(k)e^{\lambda^{-}(k)x},$$

must be bounded at  $x = \mp \infty$ .

$$\begin{array}{rl} (\eta-\Delta)(u) &=f \quad \mbox{in} \quad \mathbb{R}^2, \\ u & \mbox{is bounded at infinity} \end{array}$$

Partial Fourier transform of the equation in the  $\boldsymbol{y}$  direction:

$$\left(\eta - \frac{\partial^2}{\partial x^2} + k^2\right) (\hat{e}_j^{n+1}(x,k)) = 0 \quad \text{in} \quad \Omega_j.$$

For a given k, solutions

 $\hat{e}_{j}^{n+1}(x,k) = \gamma_{+}^{n+1}(k)^{\lambda^{+}(k)x} + \gamma_{-}^{n+1}(k)e^{\lambda^{-}(k)x},$ 

must be bounded at  $x = \mp \infty$ .

This implies

$$\hat{e}_1^{n+1}(x,k) = \gamma_+^{n+1}(k)e^{\lambda^+(k)x}$$
$$\hat{e}_2^{n+1}(x,k) = \gamma_-^{n+1}(k)e^{\lambda^-(k)x}$$

$$egin{array}{rcl} (\eta-\Delta)(u)&=f \quad {
m in}\quad \mathbb{R}^2,\ u& {
m is bounded at infinity}\,, \end{array}$$

Partial Fourier transform of the equation in the  $\boldsymbol{y}$  direction:

$$\left(\eta - \frac{\partial^2}{\partial x^2} + k^2\right) (\hat{e}_j^{n+1}(x,k)) = 0 \quad \text{in} \quad \Omega_j.$$

For a given k, solutions

$$\hat{e}_j^{n+1}(x,k) = \gamma_+^{n+1}(k)^{\lambda^+(k)x} + \gamma_-^{n+1}(k)e^{\lambda^-(k)x)},$$
 must be bounded at  $x = \pm \infty.$ 

From the interface conditions we get

$$\gamma_{+}^{n+1}(k) = \gamma_{-}^{n}(k) e^{\lambda^{-}(k)x}, \ \gamma_{-}^{n+1}(k) = \gamma_{+}^{n}(k) e^{-\lambda^{+}(k)x}.$$

Combining these two and denoting  $\lambda(k) = \lambda^+(k) = -\lambda^-(k)$ , we get for i = 1, 2,

$$\gamma_{\pm}^{n+1}(k) = \rho(k;\eta,\delta)^2 \, \gamma_{\pm}^{n-1}(k)$$

The convergence factor  $\rho$  is given by:  $\rho(k;\eta,\delta)=e^{-\lambda(k)\delta},\,\lambda(k)=\sqrt{\eta+k^2}$ 

The convergence factor  $\rho$  is given by:

$$\rho(k;\eta,\delta) = e^{-\lambda(k)\delta}, \, \lambda(k) = \sqrt{\eta + k^2}$$



#### Properties

- $\bullet$  For all  $k\,\in\,\mathbb{R},\;\rho(k)\,<\,e^{-\sqrt{\eta}\,\delta}\,<\,1$  so that
- $\gamma_i^n(k) \to 0$  uniformly as n goes to infinity.
- $\rho \to 0$  as k tends to infinity, high frequency modes of the error converge very fast.
- When there is no overlap ( $\delta = 0$ ),  $\rho = 1$  and there is stagnation of the method.

Introduction

Schwarz methods using Freefem++

Schwarz algorithms as solvers

Schwarz algorithms as preconditioners

Schwarz preconditioners using FreeFEM++

FreeFem++ allows a very simple and natural way to solve a great variety of variational problems.

It is possible to have access to the underlying linear algebra such as the stiffness or mass matrices.

A very detailed documentation of FreeFem++ is available on the official website http://www.freefem.org/
Codes and lecture notes for the domain decomposition part available here:
http://www.victoritadolean.com/p/book.html

Let a homogeneous Dirichlet boundary value problem for a Laplacian defined on a unit square  $\Omega=]0,1[^2:$ 

$$\begin{aligned} -\Delta u &= f & \text{dans } \Omega \\ u &= 0 & \text{sur } \partial \Omega \end{aligned} \tag{4}$$

The variational formulation of the problem

Find 
$$u \in H_0^1(\Omega) := \{ w \in H^1(\Omega) : w = 0, \text{ on } \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4 \}$$

such that

$$\int_{\Omega} \nabla u \cdot \nabla v dx - \int_{\Omega} f v dx = 0, \forall v \in H_0^1(\Omega) \,.$$

Feature of Freefem++: penalization of Dirichlet BC. Let TGV (*Très Grande Valeur* in French) be a very large value, the above variational formulation is approximated by Find  $u \in H^1(\Omega)$  such that

$$\int_{\Omega} \nabla u \cdot \nabla v \, dx + T G V \int_{\bigcup_{i=1,\dots,4} \Gamma_i} u \, v - \int_{\Omega} f v \, dx = 0, \forall v \in H^1(\Omega)$$

The following FreeFem++ script is solving this problem

```
// Number of mesh points in x and y directions
int Nbnoeuds=10;
```

The text after // symbols are comments ignored by the FreeFem++ language.

Each new variable must be declared with its type (here int designs integers).

```
//Mesh definition
mesh Th=square(Nbnoeuds,Nbnoeuds,[x,y]);
```

The function square returns a structured mesh of the square, the sides of the square are labelled from 1 to 4 in trigonometrical sense.

Define the function representing the right hand side

```
// Function of x and y
func f=x*y;
```

and the  $P_1$  finite element space Vh over the mesh Th.

```
// Finite element space on the mesh Th
fespace Vh(Th,P1);
//uh and vh are of type Vh
Vh uh,vh;
```

The functions  $u_h$  and  $v_h$  belong to the  $P_1$  finite element space  $V_h$  which is an approximation to  $H^1(\Omega)$ .

```
// variational problem definition
problem heat(uh,vh,solver=LU)=
    int2d(Th)(dx(uh)*dx(vh)+dy(uh)*dy(vh))
        -int2d(Th)(f*vh)
        +on(1,2,3,4,uh=0);
```

The keyword problem allows the definition of a variational problem (without solving it)

$$\int_{\Omega} \nabla u_h \cdot \nabla v_h dx + TGV \int_{\bigcup_{i=1,\dots,4} \Gamma_i} u_h v_h - \int_{\Omega} f v_h dx = 0, \forall v_h \in V_h.$$

where TGV is equal to  $10^{30}$ .

The parameter solver sets the method that will be used to solve the resulting linear system. To solve the problem we need

```
//Solving the problem
heat;
// Plotting the result
plot(uh,wait=1);
```

The Freefem++ script can be saved with your favourite text editor (e.g. under the name heat.edp). In order to execute the script write the shell command

```
FreeFem++ heat.edp
```

The result will be displayed in a graphic window.

Solve Neumann or Fourier boundary conditions such as

$$\begin{cases} -\Delta u + u = f & \text{dans } \Omega\\ \frac{\partial u}{\partial n} = 0 & \text{sur } \Gamma_1\\ u = 0 & \text{sur } \Gamma_2\\ \frac{\partial u}{\partial n} + \alpha u = g & \text{sur } \Gamma_3 \cup \Gamma_4 \end{cases}$$

The new variational formulation consists in determining  $u_h \in V_h$  such that

$$\int_{\Omega} \nabla u_h \cdot \nabla v_h dx + \int_{\Gamma_3 \cup \Gamma_4} \alpha u_h v_h + TGV \int_{\Gamma_2} u_h \cdot v_h \\ - \int_{\Gamma_3 \cup \Gamma_4} gv_h - \int_{\Omega} fv_h dx = 0, \forall v_h \in V_h.$$

The Freefem++ definition of the problem

(5)

In order to use some **linear algebra** package, we need the matrices. The keyword varf allows the definition of a variational formulation

Here rhsglobal is a FE function and the associated vector of d.o.f. is rhsglobal[]. The linear system is solved by using UMFPACK

```
// Solving the problem by a sparse LU sover
uh[] = Aglobal^-1*rhsglobal[];
```

To build the overlapping decomposition and the associated algebraic call the routine SubdomainsPartitionUnity.

Output:

- overlapping meshes aTh[i]
- the restriction/interpolation operators Rih[i] from the local finite element space Vh[i] to the global one Vh
- the diagonal local matrices Dih[i] from the partition of unity.

```
include "./data.edp"
include "./decomp.idp"
include "./createPartition.idp"
SubdomainsPartitionUnity(Th,part[],sizeovr,aTh,Rih,Dih,Ndeg,AreaThi);
```

We first need to define the global data.

```
Aglobal = vaglobal(Vh,Vh,solver = UMFPACK); // global matrix
rhsglobal[] = vaglobal(0,Vh); // global rhs
uglob[] = Aglobal^-1*rhsglobal[];
plot(uglob,value=1,fill=1,wait=1,cmm="Solution by a direct method",dim=3);
```

And then the local problems

```
for (int i = 0; i < npart; ++ i)
{
    cout << " Domain :" << i << "/" << npart << endl;
    matrix aT = Aglobal*Rih[i]';
    aA[i] = Rih[i]*aT;
    set(aA[i], solver = UMFPACK); // direct solvers
}</pre>
```

```
ofstream filei("Conv.m"):
Vh un = 0:
                                       // initial guess
Vh rn = rhsglobal;
for(int iter = 0:iter<maxit:++iter)</pre>
 { real err = 0, res;
   Vh er = 0:
    for (int i = 0; i < npart; ++i)
      { real[int] bi = Rih[i]*rn[]; // restriction to the local domain
        real[int] ui = aA[i]^{-1} * bi; // local solve
        bi = Dih[i]*ui:
        // bi = ui; // uncomment this line to test the ASM method as a solver
        er[] += Rih[i]'*bi;}
    un[] += er[]: // build new iterate
    rn[] = Aglobal*un[]; // computes global residual
    rn[] = rn[] - rhsglobal[]:
    rn[] *= -1;
    err = sqrt(er[]'*er[]);
    res = sart(rn[]'*rn[]):
    cout << "Iteration: " << iter << " Correction = " << err << " Residual = " << res << endl:
    plot (un, wait=1, value=1, fill=1, dim=3.cmm="Approximate solution at step " + iter):
    int i = iter + 1:
    // Store the error and the residual in Matlab/Scilab/Octave form
    filei << "Convhist("+i+".:)=[" << err << " " << res <<"]:" << endl:
    if (err < tol) break: \}
plot(un, wait=1, value=1, fill=1, dim=3.cmm="Final solution");
```

## Convergence

Convergence history of the RAS solver for different values of the overlapping parameter.



Note that this convergence, not very fast even in a simple configuration of 4 subdomains. The iterative version of ASM does not converge. For this reason, the ASM method is always used a preconditioner for a Krylov method such as CG, GMRES or BiCGSTAB.

#### Fixed point method

Consider the linear system

A x = b

A possible iterative method is a fixed point algorithm

 $x^{n+1} = x^n + B^{-1}(b - Ax^n)$ 

and x is a fixed point of the operator:

 $x \longmapsto x + B^{-1}(b - Ax).$ 

Let  $r_0 := b - Ax^0$  and  $C := B^{-1}A$ , then

$$x^n = \sum_{i=0}^n (I_d - C)^i B^{-1} r_0 + x^0.$$

We have convergence iff the spectral radius of the matrix  $I_d - C$  is smaller than one.

### Krylov method

Consider a preconditioned linear system:

 $B^{-1}Ax = B^{-1}b$ 

Let  $x^0$  an initial guess and  $r^0:=B^{-1}\,b-C\,x^0$  the initial residual. Then  $y:=x-x^0$  solves

 $C y = r^0$ .

The basis for Krylov methods is the following

 $\label{eq:lemma} \begin{array}{l} \mbox{Lemma} \\ \mbox{Let } C \mbox{ be an invertible matrix of size } N \times N. \\ \mbox{Then, there exists a polynomial } \mathcal{P} \mbox{ of degree} \\ p < N \mbox{ such that} \end{array}$ 

 $C^{-1} = \mathcal{P}(C) \,.$ 

By a constructive proof

$$x = x^{0} + \sum_{i=1}^{d} \left(-\frac{a_{i}}{a_{0}}\right) C^{i-1} r^{0}$$

Thus, it makes sense to introduce Krylov spaces,  $\mathcal{K}^n(C,r^0)$ 

 $\mathcal{K}^{n}(C, r^{0}) := Span\{r^{0}, Cr^{0}, \dots, C^{n-1}r^{0}\}, n \ge 1.$ 

to seek  $y^n$  an approximation to y.

By a constructive proof

$$x = x^{0} + \sum_{i=1}^{d} \left( -\frac{a_{i}}{a_{0}} \right) C^{i-1} r^{0}.$$

Thus, it makes sense to introduce Krylov spaces,  $\mathcal{K}^n(C,r^0)$ 

$$\mathcal{K}^{n}(C, r^{0}) := Span\{r^{0}, Cr^{0}, \dots, C^{n-1}r^{0}\}, n \ge 1.$$

to seek  $y^n$  an approximation to y.

**Example**: The CG methods applies to symmetric positive definite (SPD) matrices and minimizes the  $A^{-1}$ -norm of the residual when solving Ax = b:

$$\mathsf{C}G \; \left\{ \begin{array}{l} \mathsf{Find} \; y^n \in \mathcal{K}^n(A,r^0) \; \mathsf{such \; that} \\ \|A \, y^n - r^0\|_{A^{-1}} = \min_{w \in \mathcal{K}^n(A,r^0)} \|A \, w - r^0\|_{A^{-1}} \, . \end{array} \right.$$

By a constructive proof

$$x = x^{0} + \sum_{i=1}^{d} \left( -\frac{a_{i}}{a_{0}} \right) C^{i-1} r^{0}$$

Thus, it makes sense to introduce Krylov spaces,  $\mathcal{K}^n(C,r^0)$ 

$$\mathcal{K}^{n}(C, r^{0}) := Span\{r^{0}, Cr^{0}, \dots, C^{n-1}r^{0}\}, n \ge 1.$$

to seek  $y^n$  an approximation to y.

A detailed analysis reveals that  $x^n=y^n+x_0$  can be obtained by the quite cheap recursion formula:

for 
$$i = 1, 2, ...$$
 do  
 $\rho_{i-1} = (r_{i-1}, r_{i-1})_2$   
if  $i = 1$  then  
 $p_1 = r_0$   
else  
 $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$   
 $p_i = r_{i-1} + \beta_{i-1}p_{i-1}$   
end if  
 $q_i = Ap_{i-1}$   
 $\alpha_i = \frac{\rho_{i-1}}{(p_i, q_i)_2}$   
 $x_i = x_{i-1} + \alpha_i p_i$   
 $r_i = r_{i-1} - \alpha_i q_i$   
check convergence; continue if necessary  
end for

By solving an optimization problem:

$$\mathsf{G}MRES \; \left\{ \begin{array}{l} \mathsf{Find} \; y^n \in \mathcal{K}^n(C,r^0) \; \mathsf{such that} \\ \|C\,y^n - r^0\|_2 = \min_{w \in \mathcal{K}^n(C,r^0)} \|C\,w - r^0\|_2 \end{array} \right.$$

By solving an optimization problem:

$$\mathsf{G}MRES \ \left\{ \begin{array}{l} \mathsf{Find} \ y^n \in \mathcal{K}^n(C,r^0) \text{ such that} \\ \|C \ y^n - r^0\|_2 = \min_{w \in \mathcal{K}^n(C,r^0)} \|C \ w - r^0\|_2 \end{array} \right.$$

a preconditioned Krylov solve will generate an optimal  $\boldsymbol{x}_{\boldsymbol{K}}^n$  in

$$\mathcal{K}^{n}(C, B^{-1}r_{0}) := x_{0} + Span\{B^{-1}r_{0}, CB^{-1}r_{0}, \dots, C^{n-1}B^{-1}r_{0}\}.$$

By solving an optimization problem:

$$\mathsf{G}MRES \; \left\{ \begin{array}{l} \mathsf{Find} \; y^n \in \mathcal{K}^n(C,r^0) \; \mathsf{such that} \\ \|C\,y^n - r^0\|_2 = \min_{w \in \mathcal{K}^n(C,r^0)} \|C\,w - r^0\|_2 \end{array} \right.$$

a preconditioned Krylov solve will generate an optimal  $\boldsymbol{x}_{K}^{n}$  in

$$\mathcal{K}^{n}(C, B^{-1}r_{0}) := x_{0} + Span\{B^{-1}r_{0}, CB^{-1}r_{0}, \dots, C^{n-1}B^{-1}r_{0}\}.$$

**Remark**. This minimization problem is of size n. When n is small w.r.t. N, its solving has a marginal cost. Thus,  $x_K^n$  has a computing cost similar to that of  $x^n$ . But, since  $x^n \in \mathcal{K}^n(B^{-1}A, B^{-1}r_0)$  as well but with "frozen" coefficients, we have that  $x_n$  is less optimal (actually much much less) than  $x_K^n$ .

In the previous Krylov methods we can use as preconditioner • RAS (in conjunction with BiCGStab or GMRES)

$$B^{-1} := M_{RAS}^{-1} = \sum_{i=1}^{N} R_i^T D_i (R_i A R_i^T)^{-1} R_i$$

• ASM (in a CG methods)

$$B^{-1} := M_{ASM}^{-1} = \sum_{i=1}^{N} R_i^T (R_i A R_i^T)^{-1} R_i$$

•  $M_{ASM}^{-1}$  as a preconditioner • a Krylov method: conjugate gradient since  $M_{ASM}^{-1}$ and A are symmetric.

At iteration  $\boldsymbol{m}$  the error for the PCG method is bounded by:

$$\begin{aligned} ||\bar{x} - x_m||_{\substack{M_{ASM}^{-\frac{1}{2}} AM_{ASM}^{-\frac{1}{2}} \\ 2\left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right]^{m}} ||\bar{x} - x_0||_{\substack{M_{ASM}^{-\frac{1}{2}} AM_{ASM}^{-\frac{1}{2}} \\ M_{ASM}^{-\frac{1}{2}} AM_{ASM}^{-\frac{1}{2}} }. \end{aligned}$$

where  $\kappa$  is the condition number of  $M_{ASM}^{-1}A$  and  $\bar{x}$  is the exact solution.

The CG with the ASM preconditioner becomes:

for i = 1.9 do

At iteration  $\boldsymbol{m}$  the error for the PCG method is bounded by:

$$\begin{aligned} ||\bar{x} - x_m|| & \frac{1}{M_{ASM}^{-\frac{1}{2}} AM_{ASM}^{-\frac{1}{2}}}{2\left[\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right]^m} ||\bar{x} - x_0|| & \frac{1}{M_{ASM}^{-\frac{1}{2}} AM_{ASM}^{-\frac{1}{2}}}. \end{aligned}$$

where  $\kappa$  is the condition number of  $M_{ASM}^{-1}A$  and  $\bar{x}$  is the exact solution.

$$\begin{array}{l} \rho_{i-1} = (r_{i-1}, M_{ASM}^{-1} r_{i-1})_2 \\ \text{if } i = 1 \text{ then} \\ p_1 = M_{ASM}^{-1} r_0 \\ \text{else} \\ \beta_{i-1} = \rho_{i-1} / \rho_{i-2} \\ p_i = M_{ASM}^{-1} r_{i-1} + \beta_{i-1} p_{i-1} \\ \text{end if} \\ q_i = A p_{i-1} \\ \alpha_i = \frac{\rho_{i-1}}{(p_i, q_i)_2} \\ x_i = x_{i-1} + \alpha_i p_i \\ r_i = r_{i-1} - \alpha_i q_i \\ \text{check convergence; continue if necessary} \\ \text{end for} \end{array}$$

# Action of the preconditioner

The action of the global matrix and preconditioner

```
func real[int] A(real[int] &x)
       // Matrix vector product with the global matrix
       Vh Ax:
       Ax[] = Aglobal * x;
       return Ax[];
// and the application of the preconditioner
func real[int] AS(real[int] &|)
  // Application of the ASM preconditioner
  // M^{-1}*v = sum Ri^{T}*Ai^{-1}*Ri*v
  // Ri restriction operators, Ai =Ri*A*Ri^T local matrices
  Vh s = 0:
  for(int i=0:i<npart:++i)</pre>
      real[int] bi = Rih[i]*l; // restricts rhs
      real[int] ui = aA[i] ^-1 * bi; // local solves
      s[] += Rih[i]'*ui;
                         // prolongation
     ι
  return s[];
```

The Krylov method applied in this case is the CG. The performance is now less sensitive to the overlap size.





We can also use RAS as a preconditioner, by taking into account the partition of unity

```
func real[int] RAS(real[int] &l)
{
    // Application of the RAS preconditioner
    // M^{-1}*y = \sum Ri^T*Di*Ai^{-1}*Ri*y
    // Ri restriction operators, Ai =Ri*A*Ri^T local matrices
    Vh s = 0;
    for(int i=0;i<npat;++i) {
        real[int] bi = Rih[i]*l; // restricts rhs
        real[int] ui = aA[i] ^-1 * bi; // local solves
        bi = Dih[i]*ui; // partition of unity
        s[] += Rih[i]'*bi; // prolongation
    }
    return s[];
}</pre>
```

this time in conjuction with BiCGStab since we deal with non-symmetric problems.