RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

ONERA
THE FRENCH AEROSPACE LAB

Domain decomposition

Theory and implementation

Xavier JUVIGNY, AKOU, DAAA, ONERA
xavier.juvigny@onera.fr

Course Parallel Programming
- February 18th 2025 -

[1] ONERA, [2] DAAA

Table of contents

Overview

Main ideas

Principles

- Split a finite computing domain Ω (picture, spatial domain, ...) in some sub-domains Ω_i ;
- Apply a local method on each subdomains ;
- Apply if necessary a global method to find the global result on Ω ;
- The last two steps can be in a direct or iterative method.

Example : JPEG 2000 compression (Wikipedia)

- Color components transformation (RGB to YUV) ;
- Image is split into tiles (subdomains) ;
- Wavelet transformation of each tile ;
- Quantization of wavelet transformation of each tile.

Classification of domain decomposition methods

Types of domain decomposition methods

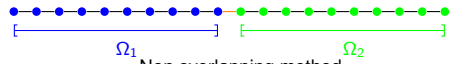
- **Overlapping domain decomposition methods** : The sub-domains share common cells (pixels, elements, ...) ;
- **Non Overlapping domain decomposition methods** : The sub-domains don't share common cells (pixels, elements, ...).

Non overlapping domain decomposition methods

Splitting Ω in n subdomains Ω_i, $i = 1, \cdots, n$ verifying

1. $\Omega_i \neq \emptyset, i = 1, \cdots, n$;
2. $\cup_{i=1}^{n} \overline{\Omega_i} = \overline{\Omega}$;
3. $\Omega_i \cap \Omega_j = \emptyset; i \neq j$;
4. $\overline{\Omega_i} \cap \overline{\Omega_j} \neq \emptyset; i \neq j$.

Overlapping method

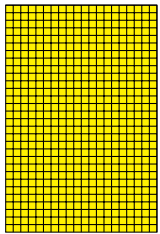


Non overlapping method

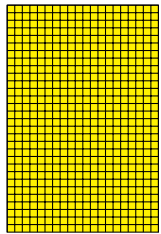
How to split a computing domain ?

Let Ω be a rectangular area in $\mathbb{R}^2$.
Some types of splitting :

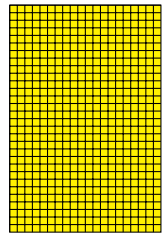
- One direction splitting : Easy to care, but not efficient for communications ;
- Alternate direction : Split in two, four or height subdomains the initial domain Ω and iterate the splitting for subdomains ; Harder to care for but more efficient for parallelization and communication ;
- Adaptive splitting : Anyway, for parallel efficiency, better to split in the direction where we have a minimal number of cells.



One direction splitting



Alternate direction splitting

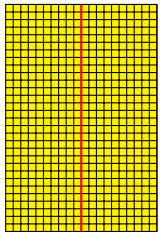


Adaptive direction splitting

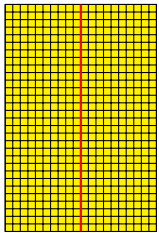
How to split a computing domain ?

Let Ω be a rectangular area in $\mathbb{R}^2$.
Some types of splitting :

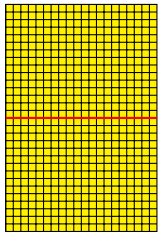
- One direction splitting : Easy to care, but not efficient for communications ;
- Alternate direction : Split in two, four or height subdomains the initial domain Ω and iterate the splitting for subdomains ; Harder to care for but more efficient for parallelization and communication ;
- Adaptive splitting : Anyway, for parallel efficiency, better to split in the direction where we have a minimal number of cells.



One direction splitting



Alternate direction splitting

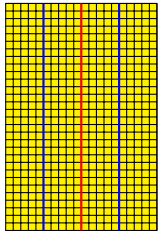


Adaptive direction splitting

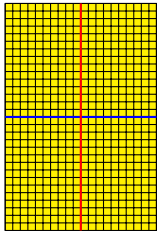
How to split a computing domain ?

Let Ω be a rectangular area in $\mathbb{R}^2$.
Some types of splitting :

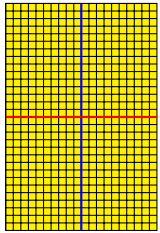
- One direction splitting : Easy to care, but not efficient for communications ;
- Alternate direction : Split in two, four or height subdomains the initial domain Ω and iterate the splitting for subdomains ; Harder to care for but more efficient for parallelization and communication ;
- Adaptive splitting : Anyway, for parallel efficiency, better to split in the direction where we have a minimal number of cells.



One direction splitting



Alternate direction splitting

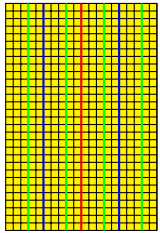


Adaptive direction splitting

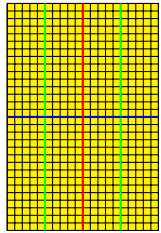
How to split a computing domain ?

Let Ω be a rectangular area in $\mathbb{R}^2$.
Some types of splitting :

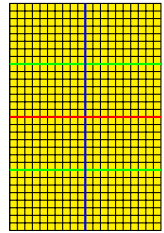
- One direction splitting : Easy to care, but not efficient for communications ;
- Alternate direction : Split in two, four or height subdomains the initial domain Ω and iterate the splitting for subdomains ; Harder to care for but more efficient for parallelization and communication ;
- Adaptive splitting : Anyway, for parallel efficiency, better to split in the direction where we have a minimal number of cells.



One direction splitting



Alternate direction splitting



Adaptive direction splitting

Domain decomposition for unstructured mesh

unstructured mesh

- Mesh where cells can't be located with (multi-)indices ;
- For example, a triangular mesh

How to split an unstructured mesh ?

- Minimize the size of the interfaces between sub-domains ;
- Optimal minimization is a NP-problem.
- Use some approximations or heuristics
- Some libraries exist for graph partitioning :
  - scotch (Inria) ;
  - metis (Karypis Lab) ;

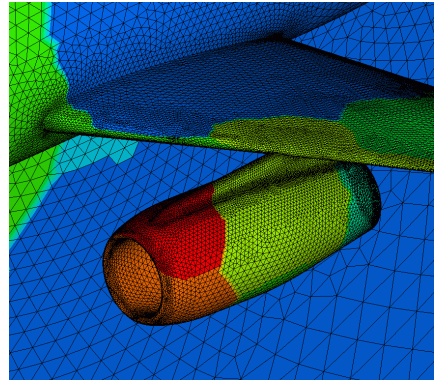


Figure – Fun 3D (NASA)

Overview

Baseline problem

Laplacian filter on a gray-scale image

- A pixel $p_{i,j}$ located by i and j indices on grid
- $v_{i,j}$ is the intensity of the pixel $p_{i,j}$;
- Discrete Laplacian scheme
  $u_{i,j} = v_{i-1,j} + v_{i+1,j} + v_{i,j-1} + v_{i,j+1} - 4v_{ij}$
- Consider pixel out of the image as black ($= 0$)

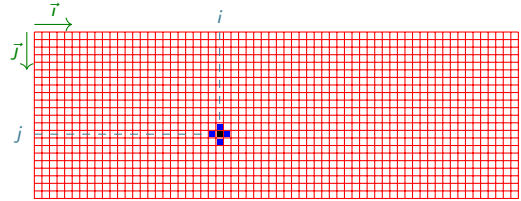


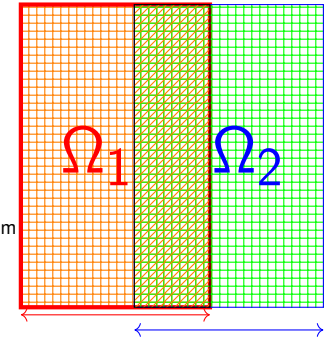
Figure – Before filter

Figure – After filter

Ghost cells

Parallelization of the laplacian filter

- Apply decomposition domain on image domain ;
- But need nearby pixels (right, left, above, bottom) ;
- Add a layer of image pixels in each direction
- These pixels are named *ghost cells* ;
- For sub-domain which contains boundary condition,
  fill ghost cells with adequate values (0 for laplacian filter).
- Application of laplacian filter is a nearly embarassingly parallel algorithm
- One must rebuild the complete filtered image after application
  of laplacian filter.



Remarks

- If possible, each process reads his part of the image ;
- Better to use communicator splitting and global communication than point to point communication when rebuilding complete image.

Inpainting algorithm

Definition

Inverse operation of laplacian filter. From filtered image, rebuild the original image.

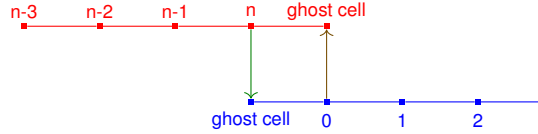
- It's an inverse laplacian problem ;
- This problem has a unique solution, thanks to the boundary (Dirichlet) condition (fixed value 0 out of the image) ;
- Inpainting is a research domain to achieve real-time decoding for 4K colour images of size 3840×2160 pixels for TV applications using GPGPU (Domain decomposition algorithms for real-time homogeneous diffusion inpainting in 4k, Niklas Kämper and Joachim Weickert, 14 February 2022).
- Symmetric definite positive linear system to solve ;
- The left hand side matrix is the laplacian filter !
- Use some iterative or direct solver (conjugate gradient or sparse LU algorithm for example).

Parallel conjugate gradient algorithm

Conjugate gradient algorithm

$p_0 = r_0 = b - A.x_0$;
$k = 0$;
while $\|r_k\| > \varepsilon$ **do**
  $\alpha_k = \frac{\|r_k\|^2}{p_k^T A p_k}$;
  $x_{k+1} = x_k + \alpha_k p_k$;
  $r_{k+1} = r_k - \alpha_k A p_k$;
  if $\|r_{k+1}\| < \varepsilon$ **then**
    exit loop ;
  end
  $\beta_k = \frac{\|r_{k+1}\|^2}{\|r_k\|^2}$;
  $p_{k+1} = r_{k+1} + \beta_k p_k$;
  $k \leftarrow k + 1$;
end
return x_{k+1} as the result ;

- Usually, if no smart x_0 can be chosen, one takes null vector as x_0 ;
- Very slow to converge $\Rightarrow$ use a local preconditioner ;
- Same splitting of the image for parallel implementation ;
- For parallel implementation, processes must exchanges values to update ghost cells value for each iteration of conjugate gradient ;



- Computing dot product or norm of vector needs global reduction.

Efficiency of parallel conjugate gradient algorithm

Efficiency

Given an image with resolution $W \times H$ pixels.
Let $N = W \times H$ be the number of pixels contained in the image. So, per iteration of conjugate gradient :

- About $5\frac{N}{nbp}$ additions and multiplications
- About $\frac{2}{\sqrt{\text{nbp}}}(W + H)$ data to send and receive when using alternating direction splitting, or $2\min(W, H)$ data to send and receive when using one direction splitting.
- For adaptative direction splitting, one has less than the best of formulae above.

Schwarz's Domain Decomposition Method

Presentation of the method

- Introduced by Schwarz (1869) to prove existence & uniqueness of Poisson problem, dividing domain in simpler geometry
- With the advent of digital computers, this method acquired a pratical interest as in iterative solver
- Parallel computers make it suited to this architecture.

The algorithm on two subdomains

Let Ω be a domain in $\mathbb{R}^n$ and Ω_1, Ω_2 two subdomains as $\Omega_i \subset \Omega$, $\Omega_1 \cup \Omega_2 = \Omega$ and $\Omega_1 \cap \Omega_2 \neq \emptyset$. The Schwarz method to solve the Poisson problem on Ω is an iterative method :

- Start with initial solutions $u_1^{(0)}, u_2^{(0)}$ on respectively Ω_1, Ω_2 ;
- To compute $u_1^{(n+1)}, u_2^{(n+1)}$ from $u_1^{(n)}, u_2^{(n)}$, solve :

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

Practical implementation

Tips

- Using ghostcells to update fixed values on $\partial\Omega_i \cap \Omega_{3-i}$, $i \in \{1, 2\}$
- Solve local values with direct or iterative method (GC by example)
- Direct solve is faster. Factorize local matrix before Schwarz iterative method ;
- Stop criterion : stop Schwarz iterations when $\max_{i=1,2} \left(\|u_i^{(n+1)} - u_i^{(n)}\| \right) < \varepsilon$
- Using more subdomains is straightforward !
- But when more subdomains are used, slower is the convergence.
- Some methods exist to speed-up the convergence of the method (coupling with multigrid method for example)

Pratical algorithm

Choose u_i ;
$u_i^{(old)} \leftarrow 0$;
while *reduce*$(\|u_i - u_i^{(old)}\|, max) < \varepsilon$ **do**
    $u_i^{(old)} \leftarrow u_i$;
    update ghost cells for u_i ;
    Solve(u_i, f) using ghostcells as boundary conditions ;
end

Properties of Schwarz's algorithm

Properties

- Some mathematical theorems show than the algorithm converges
- Greater is the overlapping between domains, faster is the convergence of the algorithm
- But solving local problems is slower
- And more memory is consumed

Complexity analysis

- For each iteration, computing complexity is the complexity to solve the linear system issued from Laplacian operator ;
- Beware, Gradient Conjugate algorithm will be purely local if used ;
- For communication, for each iteration of Schwarz's algorithm, updating ghostcells needs communication as : $\frac{2}{\sqrt{\text{nbp}}}(W + H)$ data to send and receive when using alternating direction splitting, or $2\min(W, H)$ data to send and receive when using one direction splitting.
- For adaptative direction splitting, one has less than the best of formulae above.

Overview

Non overlapping methods

Non overlapping domain decomposition

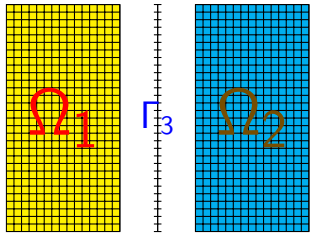
- $\{\Omega_i\}_{i=1,\cdots,n}$ is a non overlapping decomposition method of Ω if :
  - $\Omega_i \bigcap \Omega_j = \emptyset$ where $i \neq j$
  - $\bigcup_{i=1,\cdots,n} \overline{\Omega_i} = \overline{\Omega}$
  - $\overline{\Omega_i} \bigcap \overline{\Omega_j} \neq \emptyset$
- Example : $\Omega =]0;1[\Rightarrow \Omega_1 =]0;\frac{1}{2}[$ and $\Omega_2 =]\frac{1}{2};1[$

Non overlapping methods for image

- In fact, to split an image in two sub-images, we must share a common interface, i.e a simple row (or column) of pixels
- This common interface can be a third sub-images or the boundary condition of both sub-images.

Primal schur factorization

Principle with two sub-domains and an interface



- This subdivision of the domain is a bissection ;
- Care the interface as a third sub-domain ;
- Thanks to the finite difference laplacian scheme, the interaction between Ω_1 and Ω_2 is null ;
- Numbering Ω_1 nodes at first, then nodes contained in Ω_2 and finally the nodes in Γ_3 gives the following matrix :

$$A = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

- This type of matrix is called an *arrowhead (block) matrix*

Resolution of an arrowhead block matrix

Block factorization

if A_{11} and A_{22} are invertible, then :

$$A = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & 0 \\ A_{31} & A_{32} & S_{33} \end{pmatrix} \begin{pmatrix} I & 0 & A_{11}^{-1}A_{13} \\ 0 & I & A_{22}^{-1}A_{23} \\ 0 & 0 & I \end{pmatrix}$$

where $S_{33} = A_{33} - A_{31}A_{11}^{-1}A_{13} - A_{32}A_{22}^{-1}A_{23}$

Solving linear system $A.x = b$ from block factorization

- Solve $A_{11}.y_1 = b_1$ and $A_{22}.y_2 = b_2$
- Solve $S_{33}.x_3 = b_3 - A_{31}.y_1 - A_{32}.y_2$
- Compute $x_1 = y_1 - A_{11}^{-1}.A_{13}.x_3$ and $x_2 = y_2 - A_{22}^{-1}.A_{23}.x_3$.

Parallel algorithm

For two processes :

Inplace parallel factorization

For the processus with rank=i :

- Factorize $A_{ii} \leftarrow L_{ii}.U_{ii}$
- Compute $A_{i3} \leftarrow A_{ii}^{-1} A_{i3}$
- Compute $A_{33}^{(i)} \leftarrow \frac{1}{2} A_{33} - A_{3i} A_{i3}$
- **Sum reduction** $A_{33} \leftarrow A_{33}^{(1)} + A_{33}^{(2)}$
- Factorize A_{33}

Parallel inversion of the linear system

For the process with rank=i :

- Solve $y_i \leftarrow A_{ii}^{-1}.b_i$
- Compute $b_3^{(i)} \leftarrow \frac{1}{2} b_3 - A_{3i}.y_i$
- **Sum reduction** $b_3 \leftarrow b_3^{(1)} + b_3^{(2)}$
- Solve $x_3 \leftarrow A_{33}^{-1}.b_3$
- Compute $x_i \leftarrow y_i - A_{i3}.x_3$

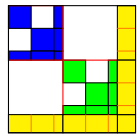
Generalization for more processes

Nested bissection

- The main idea : apply recursively the bissection on subdomains
- The factorization can be computed recursively
- The solving part is computed recursively.

Properties

- For square or rectangular domains (image for example), ideal algorithm to sparsify the factorization ;
- Drawback : A can be invertible but a matrix A_{ii} could be not invertible



Principe of FETI method on two subdomains

Algebraic approach

- Same domain decomposition as Primal Schur decomposition domain :
$$\begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$
- Decompose $A_{33} = A_{33}^{(1)} + A_{33}^{(2)}$ and $b_3 = b_3^{(1)} + b_3^{(2)}$
- Equivalent linear system :
$$\begin{cases} \begin{pmatrix} A_{11} & A_{13} \\ A_{31} & A_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_3^{(1)} + \left(b_3^{(2)} - A_{32}x_2 - A_{33}^{(2)}x_3^{(2)} \right) \end{pmatrix} \\ \begin{pmatrix} A_{22} & A_{23} \\ A_{32} & A_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_2 \\ b_3^{(2)} + \left(b_3^{(1)} - A_{31}x_1 - A_{33}^{(1)}x_3^{(1)} \right) \end{pmatrix} \end{cases}$$
- $\lambda = b_3^{(2)} - A_{32}x_2 - A_{33}^{(2)}x_3^{(2)} = -\left(b_3^{(1)} - A_{31}x_1 - A_{33}^{(1)}x_3^{(1)} \right)$
- Equivalent to find $(x_1, x_2, x_3^{(1)}, x_3^{(2)}, \lambda)$ verifying :
$$\begin{cases} \mathcal{A}_i.u_i = \begin{pmatrix} A_{ii} & A_{i3} \\ A_{3i} & A_{33}^{(i)} \end{pmatrix} \begin{pmatrix} x_i \\ x_3^{(i)} \end{pmatrix} = \begin{pmatrix} b_i \\ b_3^{(i)} \pm \lambda \end{pmatrix} = v_i \pm P_i^{\ddagger}.\lambda, \; i = 1, 2; \; P_i^{\ddagger} \text{ prolongement de } \Gamma_3 \to \overline{\Omega_i} \\ x_3^{(1)} = x_3^{(2)} \Leftrightarrow P_1.v_1 = P_2.v_2; \; P_i \text{ restriction de } \overline{\Omega_i} \to \Gamma_3 \end{cases}$$

Algebraic approach of the FETI method

FETI Method

- Must solve $\begin{pmatrix} \mathcal{A}_1 & 0 & P_1^t \\ 0 & \mathcal{A}_2 & -P_2^t \\ P_1 & -P_2 & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \lambda \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ 0 \end{pmatrix}$
- u_1 and u_2 could be considered as linear operator with variable λ : $u_1(\lambda) = \mathcal{A}_1^{-1}\left(v_1 - P_1^t\lambda\right)$ and $u_2(\lambda) = \mathcal{A}_2^{-1}\left(v_2 + P_2^t\lambda\right)$
- Idea of FETI Method is to solve linear system $P_1.u_1(\lambda) - P_2.u_2(\lambda) = 0$ with an iterative method (Conjugate gradient, GMRES, and so.)

An iteration of iterative solver with FETI Method with two processors

For a processor, the matrix-vector product in FETI method is :

- Compute $f_i = v_i \mp P_i^t\lambda$
- Solve $\mathcal{A}_i.u_i = f$
- Exchange $Pi.u_i = u_i|_{\Gamma_3}$ with other processor
- Compute $P_1.u_1 - P_2.u_2$

FETI-2LM (complement)

FETI-2LM

A more robust variant of FETI method :

- We take two lagragian variables instead of one lagragian variable
  $$\left\{ \begin{array}{l} \mathcal{A}_i.u_i = \begin{pmatrix} A_{ii} & A_{i3} \\ A_{3i} & A_{33}^{(i)} + R_{33}^{(i)} \end{pmatrix} \begin{pmatrix} x_i \\ x_3^{(i)} \end{pmatrix} = \begin{pmatrix} b_i \\ b_3^{(i)} + \lambda^{(i)} \end{pmatrix} = v_i + P_i^t.\lambda^{(i)}, i = 1, 2;\ P_i^t \text{ prolongement de } \Gamma_3 \rightarrow \overline{\Omega_i} \\ x_3^{(1)} = x_3^{(2)} \Leftrightarrow P_1.v_1 = P_2.v_2; P_i \text{ restriction de } \overline{\Omega_i} \rightarrow \Gamma_3 \end{array} \right.$$
  with $R_{33}^{(i)}$ to define.
- In global discrete system : $A_{31}.x_1 + A_{32}.x_2 + A_{33}.x_3 = b_3$, so $A_{31}.x_1 + A_{32}.x_2 + A_{33}^{(1)}.x_3^{(1)} + A_{33}^{(2)}.x_3^{(2)} = b_3^{(1)} + b_3^{(2)}$
- So using the last equation of local discrete problem : $A_{3i}.x_i + A_{33}^{(i)}.x_3^{(i)} + R_{33}^{(i)}.x_3^{(i)} = b_3^{(i)} + \lambda^{(i)}$, we have $R_{33}^{(1)}x_3^{(1)} + R_{33}^{(2)}.x_3^{(2)} = \lambda^{(1)} + \lambda^{(2)}$
- Using $x_3^{(1)} = x_3^{(2)}$, we obtain : $\left\{ \begin{array}{lcl} R_{33}^{(1)}.x_3^{(1)} + R_{33}^{(2)}.x_3^{(1)} & = & \lambda^{(1)} + \lambda^{(2)} \\ R_{33}^{(1)}.x_3^{(2)} + R_{33}^{(2)}.x_3^{(2)} & = & \lambda^{(1)} + \lambda^{(2)} \end{array} \right.$
- Relation between λ_i and $x_3^{(i)}$ can be explicitly computed : $\left(A_{33}^{(i)} - A_{3i}A_{ii}^{-1}A_{i3} + R_{33}^{(i)}\right) x_3^{(i)} = \lambda^{(i)} + b_3^{(i)} - A_{3i}A_{ii}^{-1}b_i$

FETI-2LM...

FETI-2LM (continued)

- Denote the schur complement $S^{(i)} = A_{33}^{(i)} - A_{3i}.A_{ii}^{-1}.A_{i3}$, and the condensed right-hand side $c_3^{(i)} = b_3^{(i)} - A_{3i}.A_{ii}^{-1}.b_i$.
- Replace $x_3^{(1)}$ and $x_3^{(2)}$ by their values as function of $\lambda^{(1)}$ and $\lambda^{(2)}$:

$$\begin{pmatrix} I & I - (R_{33}^{(1)} + R_{33}^{(2)})(S^{(2)} + R_{33}^{(2)})^{-1} \\ I - (R_{33}^{(1)} + R_{33}^{(2)})(S^{(1)} + R_{33}^{(1)})^{-1} & I \end{pmatrix} \begin{pmatrix} \lambda^{(1)} \\ \lambda^{(2)} \end{pmatrix} = \begin{pmatrix} (S^{(2)} + R_{33}^{(2)})^{-1} c_3^{(2)} \\ (S^{(1)} + R_{33}^{(1)})^{-1} c_3^{(1)} \end{pmatrix}$$

- "Best choice" for $R_{33}^{(1)}$ and $R_{33}^{(2)}$: $R_{33}^{(1)} = S^{(2)}$ and $R_{33}^{(2)} = S^{(1)}$ but too expensive to compute
- Try to build a sparse approximation of $S^{(i)}$ for $R_{33}^{(i)}$.