Résolution de systèmes linéaires

Enseignement spécialisé "éléments finis" – S6133/5

Christophe Bovet (christophe.bovet@onera.fr)

Onera – The French Aerospace Lab F-92322 Chatillon, France, christophe.bovet@onera.fr

Paris, le 24 novembre 2016

Contexte

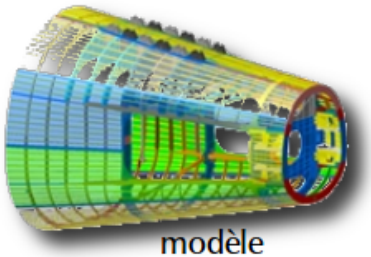
objet dans son environnement

modélisation mathématique MMC

$$\begin{cases} div\,\underline{\underline{\sigma}} = \underline{0} \\ \underline{\underline{\sigma}} = K\,\underline{\underline{\varepsilon}} \end{cases}$$

équations locales

formulation faible discrétisations temporelle et spatiale



modèle discrétisé

Problème linéaire ?

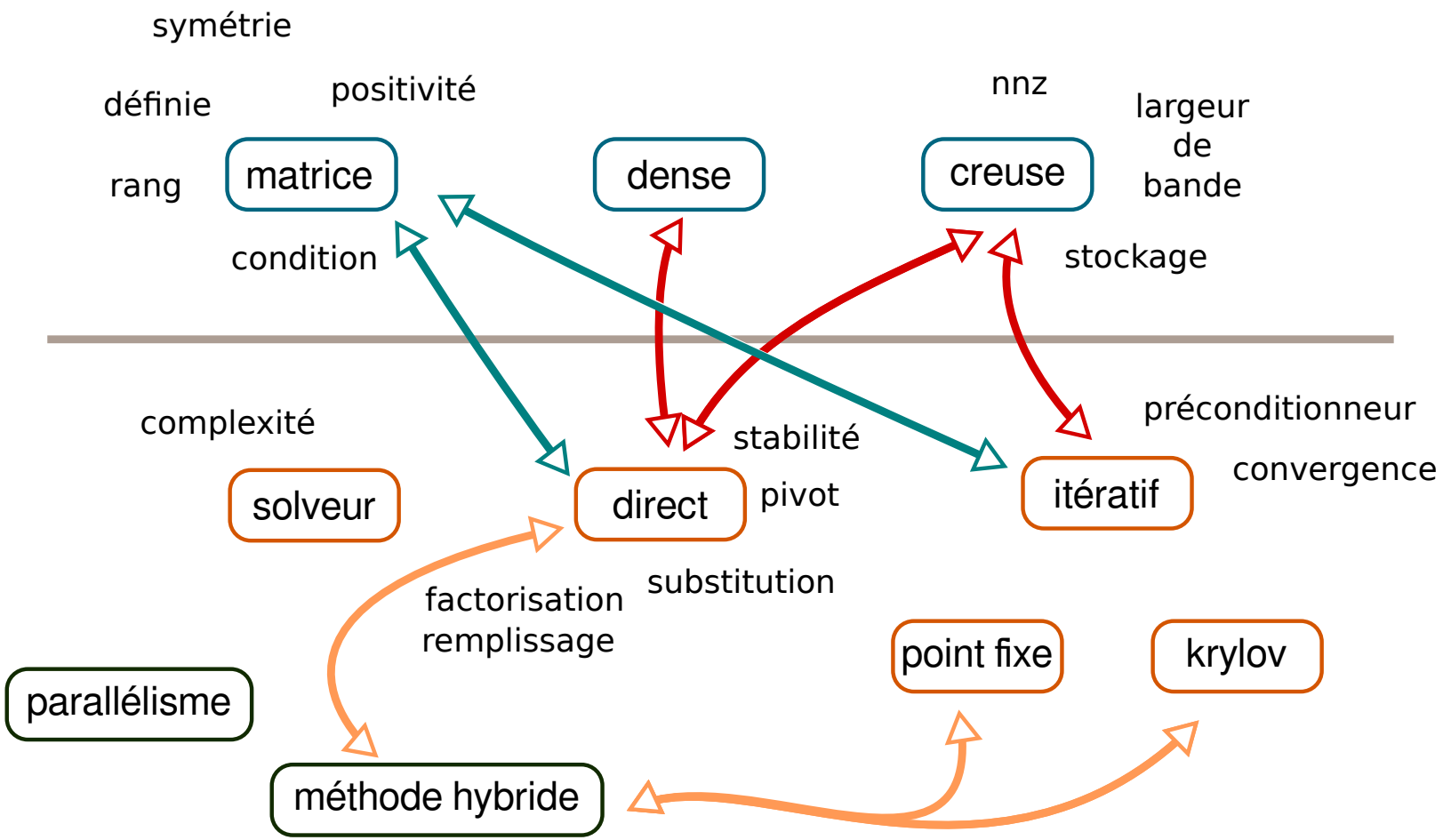
oui **non**

méthode de Newton-Raphson

à résoudre : $[A]\,[x] = [b]$

Comment résoudre efficacement un système linéaire de grande dimension (10^4 à 10^9) dans un contexte EF ?

Schéma synoptique



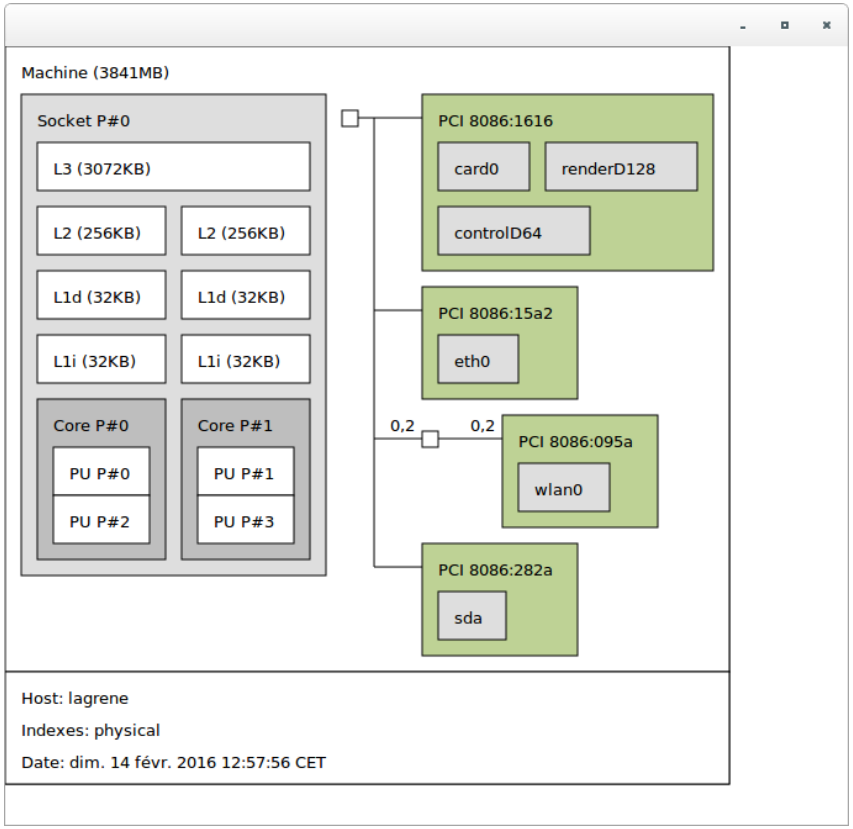
Objectifs et références

Objectifs

- Avoir un aperçu des méthodes de résolutions existantes
- Identifier les cas d'applications et spécificités des $\neq$ solutions
- Être sensibilisé aux problématiques du monde réel
  (complexité & besoin mémoire, stabilité & précision, parallélisme, ...)

Architecture d'un ordinateur

- **Unités de calculs** : coeur / processeur / noeud
- **Mémoire** : cache L1/L2/L3, RAM, disque dur
- **Communication** : bus / réseau (gigabit, infiniband)



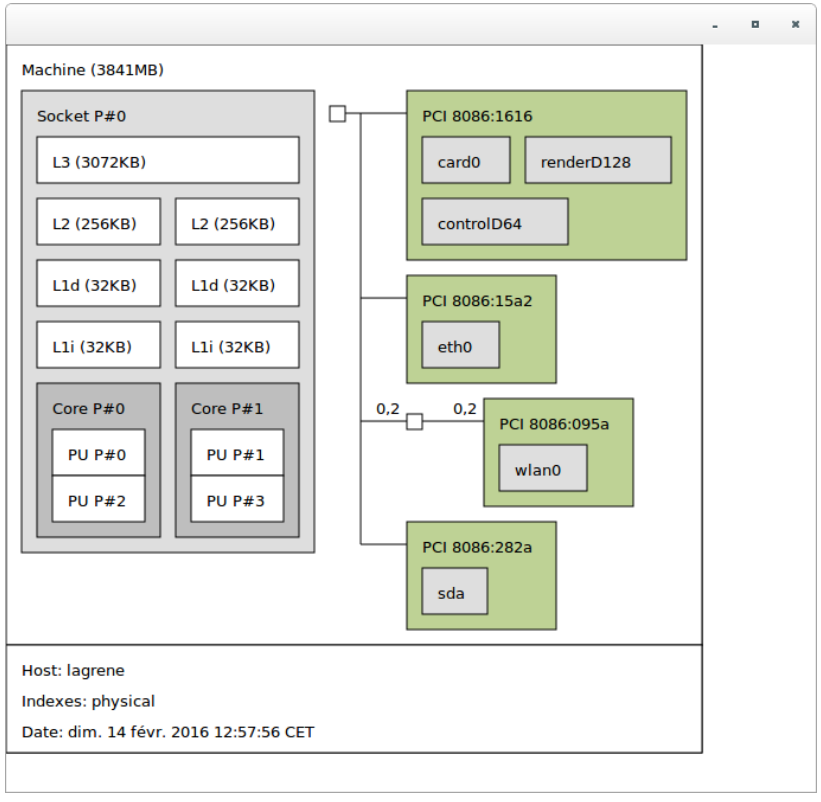
Architecture d'un ordinateur

Mémoire partagée (symmetric multiprocessing SMP)

Échange rapide (accès direct)
Besoin de protéger les données (mutex/semaphore)
⇒ multithreading, protocole openmp

Mémoire séparée

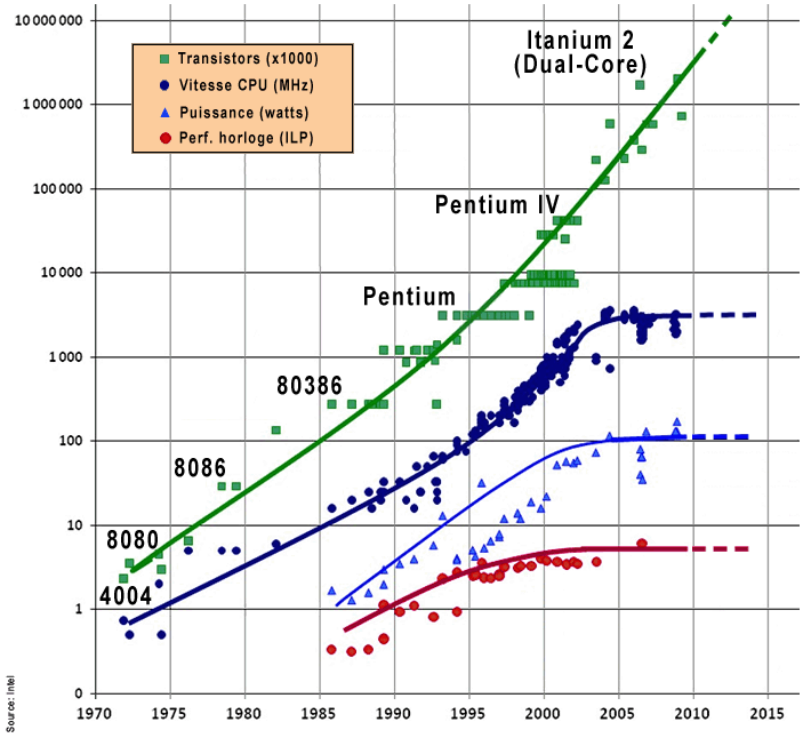
Données protégées
Il faut s'occuper d'échanger les données.
⇒ Protocole MPI
(message passing interface)



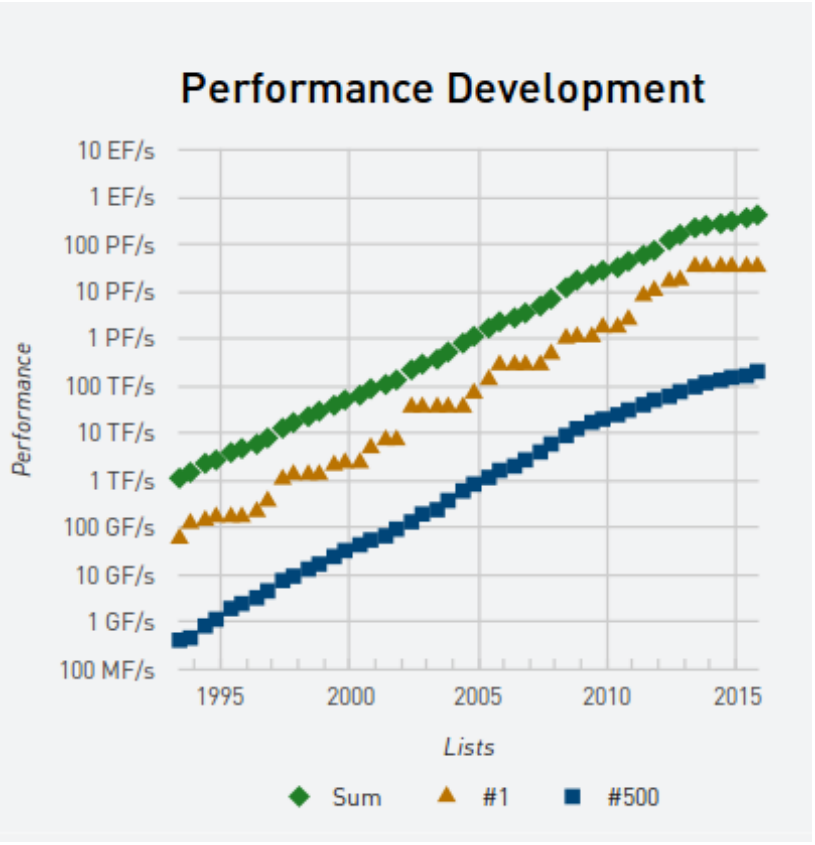
Architectures actuelles

Architectures mixtes plusieurs noeuds en réseau avec plusieurs coeurs à mémoire partagée.

Constat sur les architectures

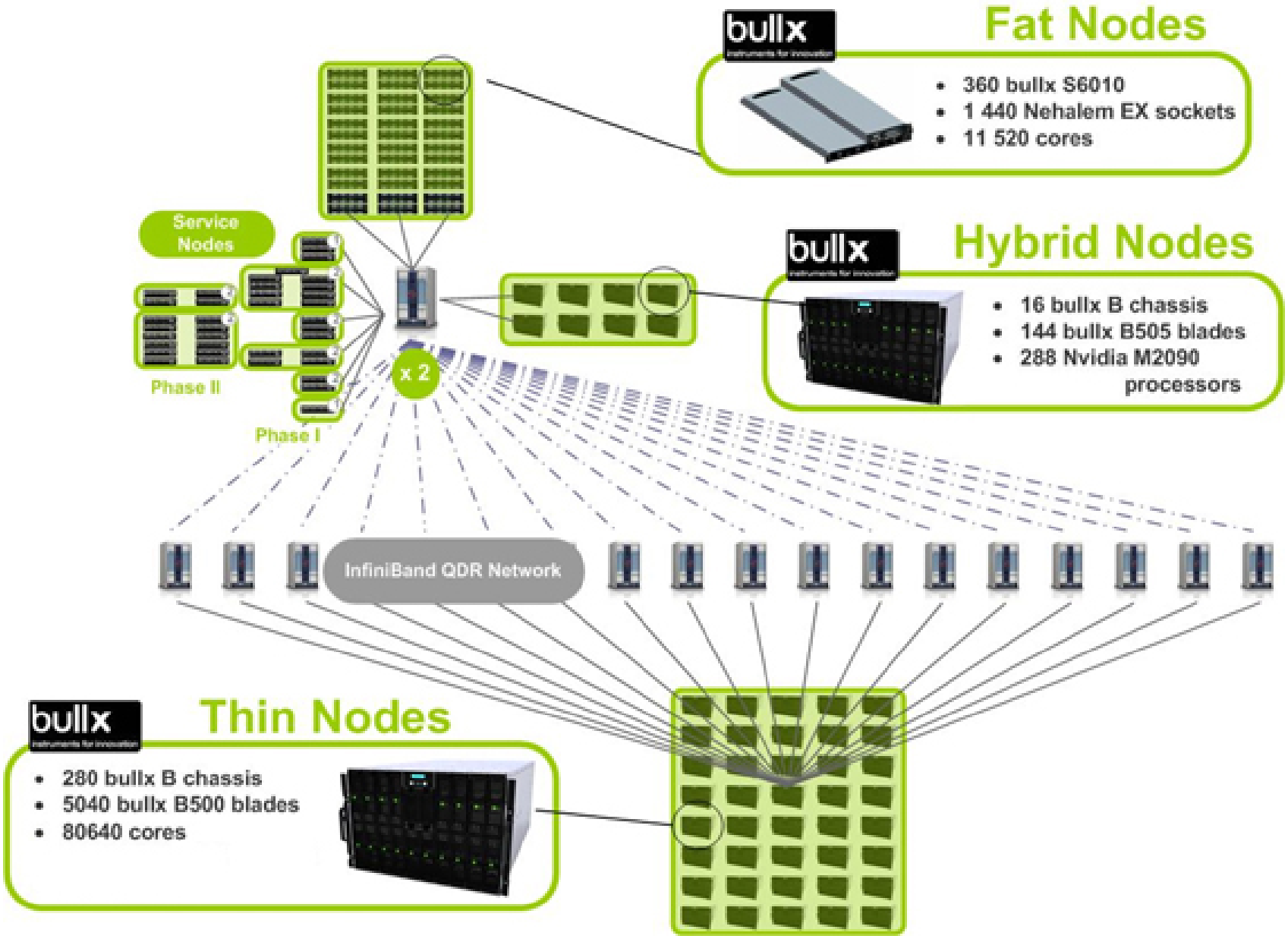


Source : www.astrosurf.com/luxorion/loi-moore.html



Source : www.top500.org

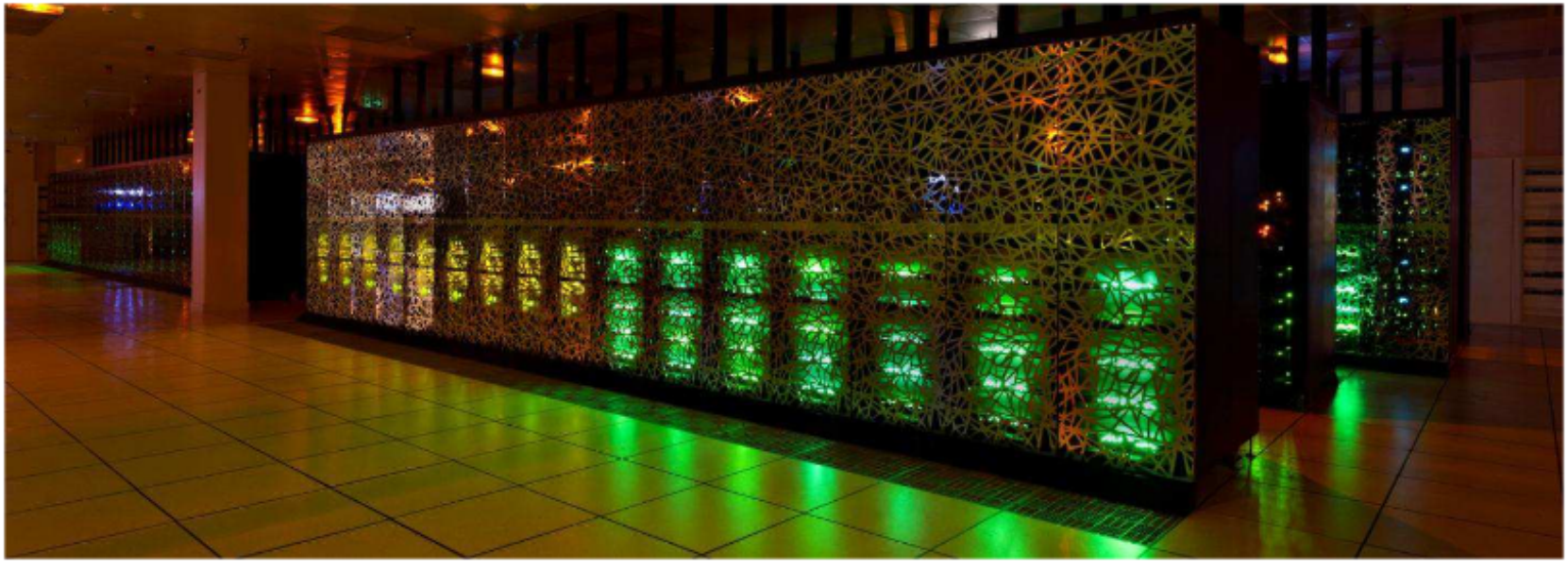
Présentation d'un Cluster
Le calculateur Curie du TGCC



Présentation d'un Cluster

Le calculateur Curie du TGCC

Fat nodes	Thin nodes
360 BullX-S6010	5040 BullX B500
Intel NH EX 2,26 Ghz	Intel Sandy Bridge 2,7Ghz
11 520 coeurs	80 640 coeurs
128 coeurs/noeud	16 coeurs/noeuds
512 Go/noeud	4 Go/coeurs
105 TFlops	1 740 TFlops



Plan

Plan

Vocabulaire et notations

Cadre d'étude

- Soit le système linéaire suivant :

$$\begin{array}{cccc} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 & & & \\ \vdots \quad\quad \vdots & & \vdots \quad\quad \vdots & \Leftrightarrow \quad \boldsymbol{Ax} = \boldsymbol{b} \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n & & & \end{array}$$

- $\boldsymbol{A}$ (taille $n \times n$) et $\boldsymbol{b}$ (second membre, *right hand side)* sont donnés
- On suppose $\boldsymbol{A}$ réelle et inversible ($\det(\boldsymbol{A}) \neq 0$)

Vocabulaire

- $\boldsymbol{A}$ est dite **dense** s'il y a peu de a_{ij} nuls $\Rightarrow$ stockage de toutes les valeurs (alignées en mémoire)
- Sinon $\boldsymbol{A}$ est dite **creuse**, seules les *nnz* valeurs non nulles sont stockées (CSR, COO, ...)

Vocabulaire et notations

Rappels

- $\boldsymbol{A}$ est symétrique si $\boldsymbol{A}^\top = \boldsymbol{A}$
- $\boldsymbol{A}$ est positive si $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0 \quad \forall \boldsymbol{x} \in \mathbb{R}^n$
- $\boldsymbol{A}$ est définie si pour $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = 0 \Rightarrow \boldsymbol{x} = \boldsymbol{0}$
- Norme matricielle :

$$\|\boldsymbol{A}\| = \max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\| = \max_{\|\boldsymbol{x}\|\neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|}{\|\boldsymbol{x}\|}$$

- Condition de $\boldsymbol{A}$ (inversible) (*condition number*) :

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\| \left\|\boldsymbol{A}^{-1}\right\| = \left(\max_{\|\boldsymbol{x}\|=1} \|\boldsymbol{A}\boldsymbol{x}\|\right) / \left(\min_{\|\boldsymbol{y}\|=1} \|\boldsymbol{A}\boldsymbol{y}\|\right)$$

On a toujours $\kappa(\boldsymbol{A}) \geq 1$

Élimination de Gauss

- Retour sur le système initial, on suppose $a_{11} \neq 0$.
- On pose $\ell_{i1} = a_{i1}/a_{11}$, on réalise ligne $i - \ell_{i1}$ ligne 1

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \quad\vdots \qquad\quad \vdots \qquad\qquad \vdots \qquad \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{array} \Rightarrow \begin{array}{r} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ a_{22}^{(1)}x_2 + \cdots + a_{2n}^{(1)}x_n = b_2^{(1)} \\ \vdots \qquad\qquad \vdots \qquad \vdots \\ a_{n2}^{(1)}x_2 + \cdots + a_{nn}^{(1)}x_n = b_n^{(1)} \end{array}$$

- On a $a_{ij}^{(2)} = a_{ij}^{(1)} - \dfrac{a_{i1}^{(1)} a_{1j}^{(1)}}{a_{11}^{(1)}}$ $\quad a_{11}^{(1)}$ est appelé pivot
- Permutation de ligne si $a_{22}^{(2)} = 0$
- Succession de système lin. $(\boldsymbol{A}^{(k)}, \boldsymbol{b}^{(k)})$ jusqu'à obtenir un syst. tri. sup.

$$\boldsymbol{A}^{(n-1)}\boldsymbol{x} = \boldsymbol{U}\boldsymbol{x} = \boldsymbol{b}^{(n-1)} \tag{1}$$

Élimination de Gauss

Theorem

Soit $\boldsymbol{A}$ *inversible, l'élimination de Gauss donne*

$$\boldsymbol{PA} = \boldsymbol{LU}$$

où $\boldsymbol{P}$ *est une matrice de permutation et*

$$\boldsymbol{L} = \begin{pmatrix} 1 & & \\ \ell_{21} & 1 & \\ \vdots & \ddots & \ddots \\ \ell_{n1} & \dots & \ell_{nn-1} \; 1 \end{pmatrix} \quad et \quad \boldsymbol{U} = \begin{pmatrix} u_{11} & \dots & \dots & u_{1n} \\ & u_{22} & \dots & u_{2n} \\ & & \ddots & \vdots \\ & & & u_{nn} \end{pmatrix}$$

- Matrice de permutation

$$\boldsymbol{P} = \begin{pmatrix} 0\,1\,0 \\ 1\,0\,0 \\ 0\,0\,1 \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{P} \begin{pmatrix} ligne1 \\ ligne2 \\ ligne3 \end{pmatrix} = \begin{pmatrix} ligne2 \\ ligne1 \\ ligne3 \end{pmatrix}$$

Plan

Méthodes directes

Élimination de Gauss & factorisation $\boldsymbol{LU}$

En pratique, on veut souvent résoudre plusieurs second membres. La substitution à la volée de $\boldsymbol{b}$ n'est pas souhaitée

1. Calcul de la factorisation $\boldsymbol{LU}$

$$\boldsymbol{LU} = \boldsymbol{PA}$$

Alors

$$\boldsymbol{LUx} = \boldsymbol{PAx} = \boldsymbol{Pb}$$

2. Résolution d'un système tri. inf. à diagonale unitaire

$$\boldsymbol{Ly} = \boldsymbol{Pb} \qquad \text{(descente, } \textit{forward substitution}\text{)}$$

3. Résolution d'un système tri. supérieur

$$\boldsymbol{Ux} = \boldsymbol{y} \qquad \text{(remontée, } \textit{backward substitution}\text{)}$$

Élimination de Gauss & factorisation $\boldsymbol{LU}$

Remarques

- Suivant les propriétés de $\boldsymbol{A}$, on sait d'avance si aucune permutation n'est nécessaire
  - Matrices à diagonale dominante
  - Matrices SPD

Theorem (Existence et unicité)

Soit la matrice $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. La factorisation $\boldsymbol{LU}$ existe et est unique ssi toutes les sous-matrices $\boldsymbol{A}_i$ d'ordre $i = 1, \ldots, n-1$ sont inversibles.

Remarques

- Si $\boldsymbol{A}$ est inversible, on peut toujours se ramener au cas ci-dessus grâce à des permutations.

Élimination de Gauss & factorisation $\boldsymbol{LU}$

Complexité

- Passage de $\boldsymbol{A}$ à $\boldsymbol{A}^{(1)}$: $(n-1)$ divisions + $(n-1)^2$ additions et multiplications.
- Coût du calcul de $\boldsymbol{LU} \simeq 2/3n^3 + 1/2n^2 + o\left(n^2\right)$
- Coût de résolution d'un système triangulaire $\simeq n^2$
- Coût de stockage, en dense il suffit d'un vecteur supplémentaire pour $\boldsymbol{P}$.

Raisonnement simplifié

- On n'utilise pas les spécificités de $\boldsymbol{A}$
- Algèbre linéaire dense
- Arithmétique exacte

Élimination de Gauss et calculs à virgule flottante

Pivot partiel $\boldsymbol{PA} = \boldsymbol{LU}$

- Exemple ($k \ll 1$)

$$\boldsymbol{A} = \begin{bmatrix} k & 1 \\ 1 & 1 \end{bmatrix} \quad \boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ (1/k) & 1 \end{bmatrix} \quad \boldsymbol{U} = \begin{bmatrix} k & 1 \\ 0 & (1 - 1 \cdot (1/k)) \end{bmatrix}$$

- Au lieu de se satisfaire de $a_{kk}^{(k)} \neq 0$, on permute la ligne i qui maximise $|a_{ik}^{(k)}|, i \geq k$

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & x & x & x \\ & + & + & + \\ & \star & \star & \star \end{pmatrix} \Rightarrow \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & \star & \star & \star \\ & + & + & + \\ & x & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ & \star & \star & \star \\ & & \times & \times \\ & & \times & \times \end{pmatrix}$$

- N'affecte pas les lignes de $\boldsymbol{U}$ déjà factorisées
- Si ce n'est pas suffisant, il existe le pivotage complet $\Rightarrow \boldsymbol{PAQ}^\top = \boldsymbol{LU}$

Autres factorisations classiques

- Variante de la factorisation $\boldsymbol{LU}$:

$$\boldsymbol{LDU}^\star = \boldsymbol{PA}$$

  Avec $\boldsymbol{D}$ diagonale et $\boldsymbol{U}^\star$ tri. sup. à diagonale unitaire.
- Ces factorisations ne profite pas des propriétés de $\boldsymbol{A}$.
- Si $\boldsymbol{A}$ est symétrique $\Rightarrow$ factorisation de Crout (pivotage symétrique)

$$\boldsymbol{LDL}^\top = \boldsymbol{PAP}^\top$$

  Coût du calcul $\simeq 1/3n^3$
- Si $\boldsymbol{A}$ est SDP, tous les termes $\boldsymbol{D}_{ii} > 0 \Rightarrow$ factorisation de Cholesky.

$$\boldsymbol{L}_c\boldsymbol{L}_c^\top = \boldsymbol{A}$$

  Coût du calcul $\simeq 1/3n^3$, pas besoin de pivoter *a priori*.

Quelques remarques

Remarque

Les méthodes `solve` utilisent une ***LU***

Quelques références

- Langages interprétés :
  - Python Scipy : `scipy.linalg`
  - Octave / Matlab : `lu`
- Librairies :

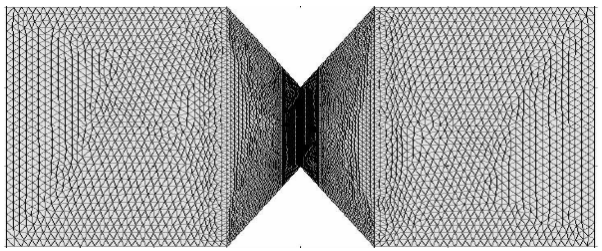
  `http://en.wikipedia.org/wiki/Comparison_of_linear_algebra_libraries`
  - Eigen
  - LAPACK
  - GNU Scientific library

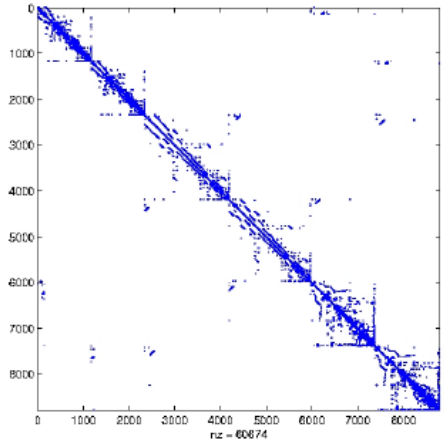
Systèmes creux

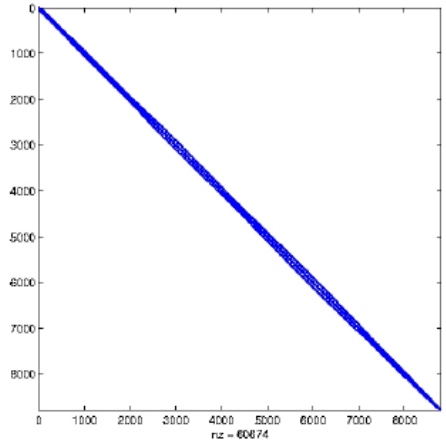
- En **dense**, coût d'une facto ***LU*** et temps de calcul

	Flops de l'ordinateur		
n	10^9	10^{12}	10^{15}
10^4	10 min	1 sec	1 μs
10^6	20 ans	7 mois	10 min
10^8	...	...	20 ans

- Heureusement les systèmes sont souvent **creux**







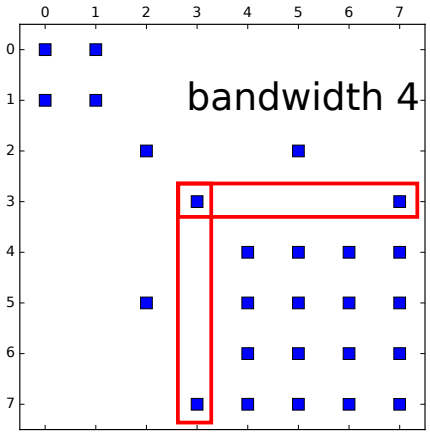
Source : Analyse des solides déformables par la méthode des éléments finis, Bonnet, Frangi

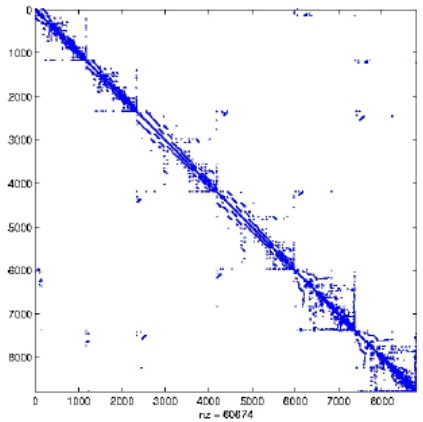
Stockage creux

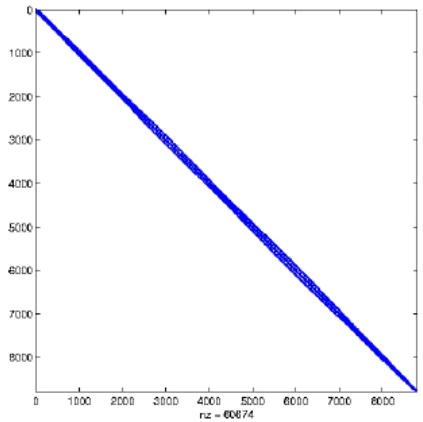
Largeur de bande

La matrice $\boldsymbol{A}$ a une largeur de bande b si

$$\begin{cases} \forall i, a_{ij} = 0 & \text{si} \quad i > j + b \\ \forall j, a_{ij} = 0 & \text{si} \quad j > i + b \end{cases}$$







Stockage creux

- Plusieurs stockages possibles (COO, CSR, Skyline, ...)
- Exemple du stockage COO `(i,j,aij)`

```
i = [ 1 1 2 3 4]
j = [ 1 2 2 3 2]
aij = [ 11 6 45 22 3]
n = 4
m = 4
nnz = 5
```

$$\boldsymbol{A} = \begin{bmatrix} 11 & 45 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 22 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}$$

- `nnz` nombre de valeurs non nulles

Factorisation et stockage creux

- On considère la factorisation LU de la matrice $\boldsymbol{A}$ creuse
- Les matrices $\boldsymbol{L}$ et $\boldsymbol{U}$ sont en générale creuses également
- Elles sont par contre (beaucoup) plus remplies

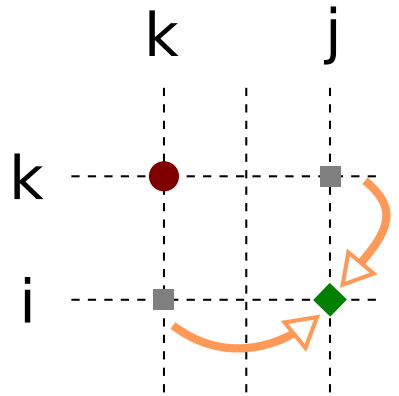
$$nnz(\boldsymbol{A}) \ll nnz(\boldsymbol{L}) \quad \text{et} \quad nnz(\boldsymbol{A}) \ll nnz(\boldsymbol{U})$$

- Le remplissage ou *fill-in* est la différence de *nnz* entre $\boldsymbol{A}$ et $\boldsymbol{L}$ ou $\boldsymbol{U}$
- Le *fill-in* augmente la mémoire nécessaire et le coût de la factorisation
- Comment minimiser ce remplissage ?

Source du *fill-in*

- Retour sur l'élimination de Gauss

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}}$$



- À l'étape $k+1$, $a_{ij}^{(k+1)} \neq 0$ si $\begin{cases} a_{ij}^{(k)} \neq 0 \\ a_{ik}^{(k)} \neq 0 \text{ et } a_{kj}^{(k)} \neq 0 \text{ (structural fill-in)} \end{cases}$
- Exemple

$$\begin{pmatrix} x & x & x & x \\ x & x & & \\ x & & x & \\ x & & & x \end{pmatrix} \Rightarrow \begin{pmatrix} \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \end{pmatrix} \quad \text{mais} \quad \begin{pmatrix} x & & & x \\ & x & & x \\ & & x & x \\ x & x & x & x \end{pmatrix} \Rightarrow \begin{pmatrix} \star & & & \star \\ & \star & & \star \\ & & \star & \star \\ \star & \star & \star & \star \end{pmatrix}$$

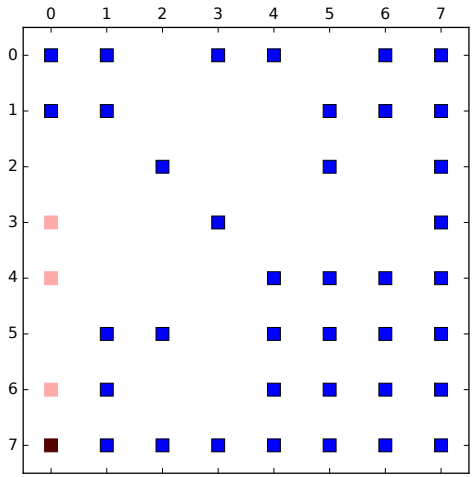
Factorisation et stockage creux

- Les permutations de lignes et colonnes de $\boldsymbol{A}$ permettent de réduire significativement le remplissage.
- Cela fait appel à de notions complexes de théorie des graphes.
- Les permutations ont donc un rôle double
  - Garantir la stabilité et limiter la propagation des erreurs d'arrondis
  - Limiter le remplissage
- Pivots candidats *(threshold pivot)* : au lieu de choisir

$$p = \underset{i \geq k}{\operatorname{argmax}}(|a_{ik}^{(k)}|)$$

on choisit le pivot qui minimise le *fill-in* parmi les pivots candidats vérifiant :

$$|a_{ik}^{(k)}| \geq \tau |\max a_{pk}^{(k)}|$$



Synthèse

Les solveurs directs sont

- des variantes de l'élimination de Gauss
- ils fonctionnent en 3 phases

Fact. symbolique $\Rightarrow$ Fact. numérique $\Rightarrow$ descente-remontée

- robustes : ils fournissent la solution exacte (arithmétique exacte) en un nombre fini d'opération (qui peut être grand)
- coûteux en mémoire (les ressources croissent fortement avec la taille du problème)
- le conditionnement de $\boldsymbol{A}$ influe uniquement sur la qualité de la solution (remède : pivot, scaling)

Plan

Pourquoi utiliser des méthodes itératives ?

- Les solveurs directs sont robustes mais très gourmand en mémoire
- La parallélisation est possible mais pas évidente (échange de complément de Schur)
- Pour les très gros problèmes il faut penser aux méthodes itératives ou hybride
- **En dense**, un produit MV en $O(n^2) \Rightarrow$ uniquement utile pour le creux
- Solveurs itératifs presque *embarrassingly parallel*

$$\boldsymbol{A}\boldsymbol{x} \quad \Rightarrow \quad \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix}$$

Méthodes itératives stationnaires

Principe

- On souhaite résoudre $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ avec $\boldsymbol{A}$ très grande
- Construire une suite de vecteurs tels que $\lim_{k \to +\infty} \boldsymbol{x}_k = \boldsymbol{A}^{-1}\boldsymbol{b} = \boldsymbol{x}^\star$
- Soit $\boldsymbol{x}_k$ l'approximation au pas k

$$\boldsymbol{x}^\star = \boldsymbol{x}_k + \boldsymbol{e}_k \quad \text{(erreur)}$$

$$\boldsymbol{A}\boldsymbol{e}_k = \boldsymbol{A}\boldsymbol{x}^\star - \boldsymbol{A}\boldsymbol{x}_k := \boldsymbol{r}_k \quad \text{(résidu)}$$

On a donc

$$\boldsymbol{x}^\star = \boldsymbol{x}_k + \boldsymbol{A}^{-1}\boldsymbol{r}_k$$

- L'idée est de remplacer $\boldsymbol{A}$ par une matrice proche mais facilement inversible

Méthodes itératives stationnaires

Principe

- Soit $\boldsymbol{M}$ une matrice inversible qui
  - est une bonne approximation de $\boldsymbol{A}$
  - soit facile à calculer
  - permette de résoudre facilement le système $\boldsymbol{Mz} = \boldsymbol{r}$
- $\boldsymbol{M}$ est appelé **préconditionneur**
- Au lieu de résoudre $\boldsymbol{x}^\star = \boldsymbol{x}_k + \boldsymbol{A}^{-1}\boldsymbol{r}_k$, on itère :

$$\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{M}^{-1}\boldsymbol{r}_k$$

- Trois étapes :
  calcul de résidu $\Rightarrow$ résolution du pb préconditionné $\Rightarrow$ maj solution

Préconditionnement

Comment construire un préconditionneur

- Décomposer la matrice $\boldsymbol{A} = \boldsymbol{M} - \boldsymbol{N}$ où $\boldsymbol{M}$ est (facilement) inversible
- On a l'équivalence

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{M}^{-1}\boldsymbol{r}_k \Leftrightarrow \boldsymbol{M}\boldsymbol{x}_{k+1} = \boldsymbol{N}\boldsymbol{x}_k + \boldsymbol{b}$$

- Quelques manipulations algébriques mènent à la formulation sous forme de point fixe

$$\boldsymbol{x}_{k+1} = \boldsymbol{B}\boldsymbol{x}_k + \boldsymbol{c} \quad \text{avec } \boldsymbol{B} = \boldsymbol{M}^{-1}\boldsymbol{N} \text{ et } \boldsymbol{c} = \boldsymbol{M}^{-1}\boldsymbol{b}$$

- $\boldsymbol{B}$ est la matrice d'itération $\boldsymbol{e}_{k+1} = \boldsymbol{B}\boldsymbol{e}_k$

Méthodes itératives stationnaires

Définition

- Une méthode itérative est convergente ssi la suite $(\boldsymbol{x}_n)_n \to \boldsymbol{A}^{-1}\boldsymbol{f}$ $\forall$ init. $\boldsymbol{x}_0$
- Dans notre cas, la méthode converge uniquement si le rayon spectral de la matrice d'itération

$$\rho(\boldsymbol{B}) = \rho(\boldsymbol{M}^{-1}\boldsymbol{N}) < 1$$

- Suivant le choix de $\boldsymbol{M}$ on obtient les méthodes Jacobi, Gauss-Seidel, de relaxation ...
- ... et des propriétés de convergence différentes.

Critère d'arrêt et exemple

Critères d'arrêt

- Norme du résidu $\|\boldsymbol{r}_k\| \leq \epsilon \|\boldsymbol{r}_0\|$
- Stagnation de la solution $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| \leq \epsilon \|\boldsymbol{x}_k\|$

Un petit comparatif

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ 1 & 300 \end{pmatrix} \quad \boldsymbol{b} = (1, 0) \quad \boldsymbol{x}_0 = (1., 1.5)$$

Méthodes itératives de type Krylov

- Avec les méthodes précédentes

$$\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{M}^{-1}\boldsymbol{r}_{k-1} = \boldsymbol{x}_{k-1} + \boldsymbol{z}_{k-1}$$

- On construit successivement $\boldsymbol{z}_k = Q_k(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{z}_0$ où Q_k est un polynôme de degré k.
- Déf : Espace de Krylov :

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0) = \text{Vect}(\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{r}_0)$$

- Les méthodes de projection ajoutent une contrainte sur $\boldsymbol{r}_m$ tel que

$$\begin{cases} \boldsymbol{x}_m \in \ \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \\ \boldsymbol{r}_m \perp_? \ \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \end{cases}$$

- Dans le cas précédent $\boldsymbol{K} \leftarrow \boldsymbol{M}^{-1}\boldsymbol{A}$ et $\boldsymbol{r}_0 \leftarrow \boldsymbol{z}_0$

Méthodes itératives de type Krylov

- Les méthodes de projection ajoutent une contrainte sur $\boldsymbol{r}_m$ tel que

$$\begin{cases} \boldsymbol{x}_m \in \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \\ \boldsymbol{r}_m \perp_? \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \end{cases}$$

- Dans le cas précédent $\boldsymbol{K} \leftarrow \boldsymbol{M}^{-1}\boldsymbol{A}$ et $\boldsymbol{r}_0 \leftarrow \boldsymbol{z}_0$
- Suivant les propriétés de $\boldsymbol{K}$, le choix du type l'orthogonalité permet de définir plusieurs approches (CG, GMRes, OrthoDir, etc.)
- Construction d'une base de l'espace de Krylov (procédé d'Arnoldi)
- Si $\boldsymbol{A}$ est symétrique définie positive le gradient conjugué est la meilleure solution

$$\begin{cases} \boldsymbol{x}_m \in \boldsymbol{x}_0 + \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \\ \boldsymbol{r}_m \perp \mathcal{K}_m(\boldsymbol{K}, \boldsymbol{r}_0) \end{cases}$$

Méthode du gradient conjugué

Construction d'une base de $\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}_0)$.

- La base naturelle $(\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{m-1}\boldsymbol{r}_0)$ est peu adaptée
- On suppose $\boldsymbol{A}$ SPD, soit $(\lambda_p, \boldsymbol{v}_p)$ les éléments propres de $\boldsymbol{A}$

$$\text{Si } \boldsymbol{r}_0 = \sum_p \alpha_p \boldsymbol{v}_p \quad \Rightarrow \quad \boldsymbol{A}\boldsymbol{r}_0 = \sum_p (\alpha_p \cdot \lambda_p)\boldsymbol{v}_p$$

Algorithme 1 : Procédé d'Arnoldi, création d'une base $\boldsymbol{A}$-orthogonale

Initialisation $\boldsymbol{v}_1 / \|\boldsymbol{v}_1\|_2 = 1$
for $j = 1, \ldots m$ **do**
  Calcul de $h_{ij} = \boldsymbol{v}_i^\top \boldsymbol{A}\boldsymbol{v}_j$ for $i = 1, \ldots j$
  Calcul de $\boldsymbol{w}_j = \boldsymbol{A}\boldsymbol{v}_j - \sum_{i=1}^{j} h_{ij}\boldsymbol{v}_i$
  $h_{j+1j} = \|\boldsymbol{w}_j\|_2$
  if $h_{j+1j} = 0$ **then** stop
  else $\boldsymbol{v}_{j+1} h_{j+1j}$

Méthode du gradient conjugué

- Si $\boldsymbol{A}$ est symétrique la matrice d'Hessemberg (h_{ij}) est tridiagonale $\Rightarrow$récurrence courte

Algorithme 2 : Gradient conjugué avec récurrence courte

Initialisation $\boldsymbol{r}_0 = \boldsymbol{A}\boldsymbol{x}_0 - \boldsymbol{b}, \boldsymbol{w}_0 = \boldsymbol{r}_0, j = 0$

while $\sqrt{\boldsymbol{r}_j^T \boldsymbol{r}_j}$ **do**

$\boldsymbol{q}_j = \boldsymbol{A}\boldsymbol{w}_j$

$\alpha_j = (\boldsymbol{r}_j^T \boldsymbol{r}_j)/(\boldsymbol{q}_j^T \boldsymbol{w}_j)$

$\boldsymbol{x}_{j+1} = \boldsymbol{x}_j + \alpha_j \boldsymbol{w}_j$

$\boldsymbol{r}_{j+1} = \boldsymbol{r}_j - \alpha_j \boldsymbol{q}_j$

$\beta_j = (\boldsymbol{r}_{j+1}^T \boldsymbol{r}_{j+1})/(\boldsymbol{r}_j^T \boldsymbol{r}_j)$

$\boldsymbol{w}_{j+1} = \boldsymbol{r}_{j+1} + \beta_j \boldsymbol{w}_j$

$j \leftarrow j + 1$

Remarques

Propriétés

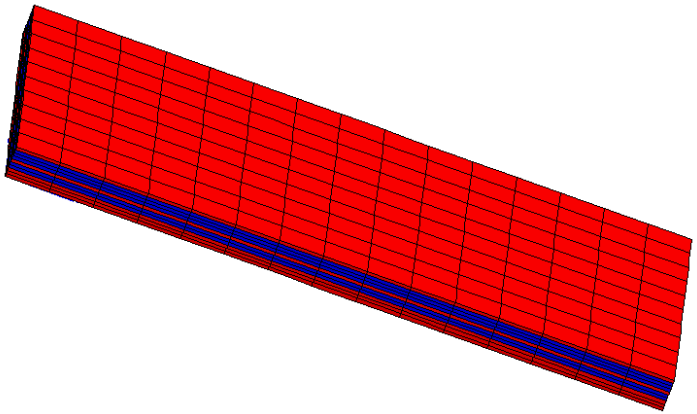
- Le gradient conjugué converge en au plus n itérations.
- À chaque itération, $\boldsymbol{x}_k$ minimise la $\boldsymbol{A}$-norme de l'erreur sur l'espace de Krylov.

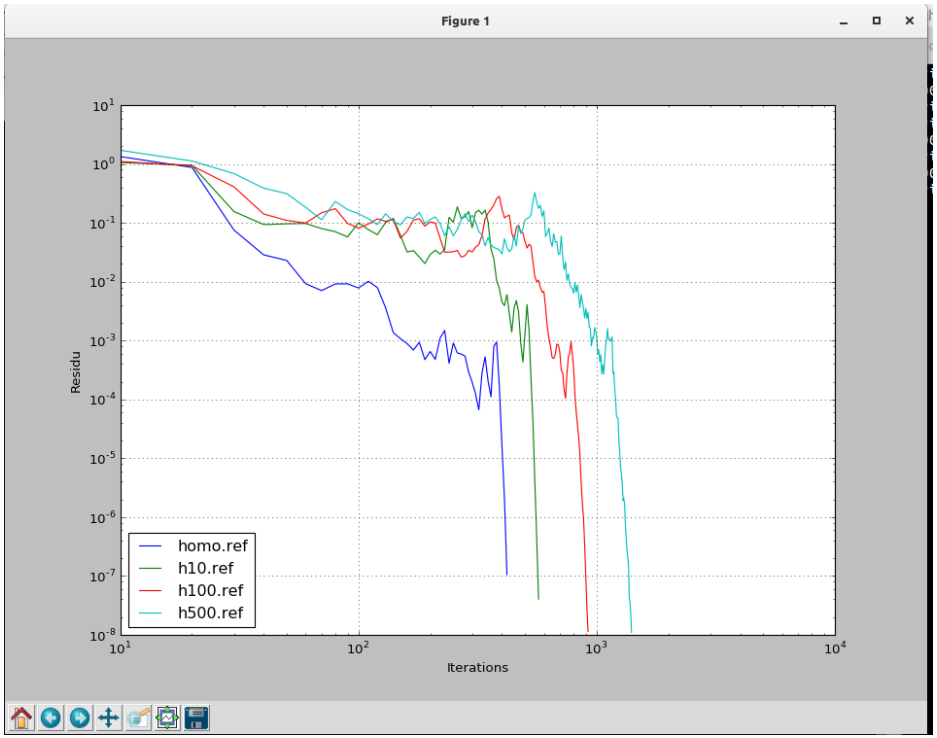
Un petit comparatif

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ 1 & 300 \end{pmatrix} \quad \boldsymbol{b} = (1, 0) \quad \boldsymbol{x}_0 = (1., 1.5)$$

Solveurs itératifs sensibles au conditionnement

- Plaque lamifiée élastique linéaire $n \simeq 21\ 000$, CG & Chol.





Synthèse

- Les méthodes itératives nécessitent très peu de mémoire
- Elles peuvent donc résoudre de très gros problèmes
- Choisir un bon préconditionneur n'est pas toujours évident
- La vitesse de convergence dépend du conditionnement de l'opérateur à résoudre
- Les méthodes de Krylov convergent généralement plus rapidement
- Elles sont cependant moins *embarrassingly parallel*

Plan

Méthodes hybrides

Idée de base : combiner les avantages des méthodes directes et itératives

- Méthodes itératives par blocs (solveurs directes sur les sous-blocs)
- Méthodes multi-grilles
- Méthode de type décomposition de domaine, solveurs directs pour les problèmes locaux et solveurs de Krylov pour équilibrer l'interface

Méthodes hybrides

Idée de base : combiner les avantages des méthodes directes et itératives

- Méthodes itératives par blocs (solveurs directes sur les sous-blocs)
- Méthodes multi-grilles
- Méthode de type décomposition de domaine, solveurs directs pour les problèmes locaux et solveurs de Krylov pour équilibrer l'interface

Origines des méthodes de décomposition de domaine

Méthodes de Schur

Système global

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f} \quad \text{avec} \quad \boldsymbol{K} \text{ SDP}$$

Décomposition sans recouvrement

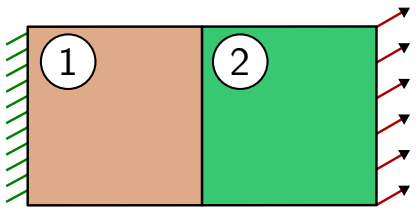
$$\boldsymbol{K}^{(s)}\boldsymbol{u}^{(s)} = \boldsymbol{f}^{(s)} + \boldsymbol{t}^{(s)\,T}\boldsymbol{\lambda}_b^{(s)}$$

$$\sum_s \boldsymbol{A}^{(s)}\boldsymbol{t}^{(s)}\boldsymbol{\lambda}^{(s)} = \sum_s \boldsymbol{A}^{(s)}\boldsymbol{\lambda}_b^{(s)} = \boldsymbol{0}$$

$$\sum_s \boldsymbol{B}^{(s)}\boldsymbol{t}^{(s)}\boldsymbol{u}^{(s)} = \sum_s \boldsymbol{B}^{(s)}\boldsymbol{u}_b^{(s)} = \boldsymbol{0}$$

Topologie

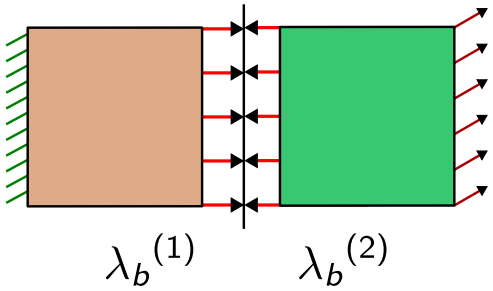
$\boldsymbol{t}^{(s)}$ op. de trace $(\boldsymbol{u}_b^{(s)} = \boldsymbol{t}^{(s)}\boldsymbol{u}^{(s)})$
$\boldsymbol{B}^{(s)}$ op. d'assemblage dual
$\boldsymbol{A}^{(s)}$ op. d'assemblage primal



Problème init.

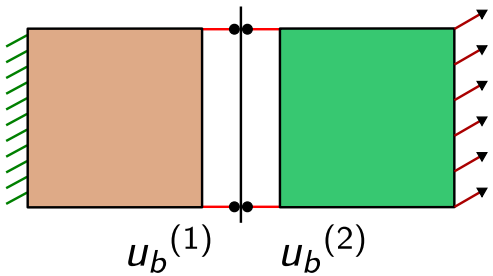
⇩

Décomposition



Équilibre

$\lambda_b{}^{(1)} = -\lambda_b{}^{(2)}$

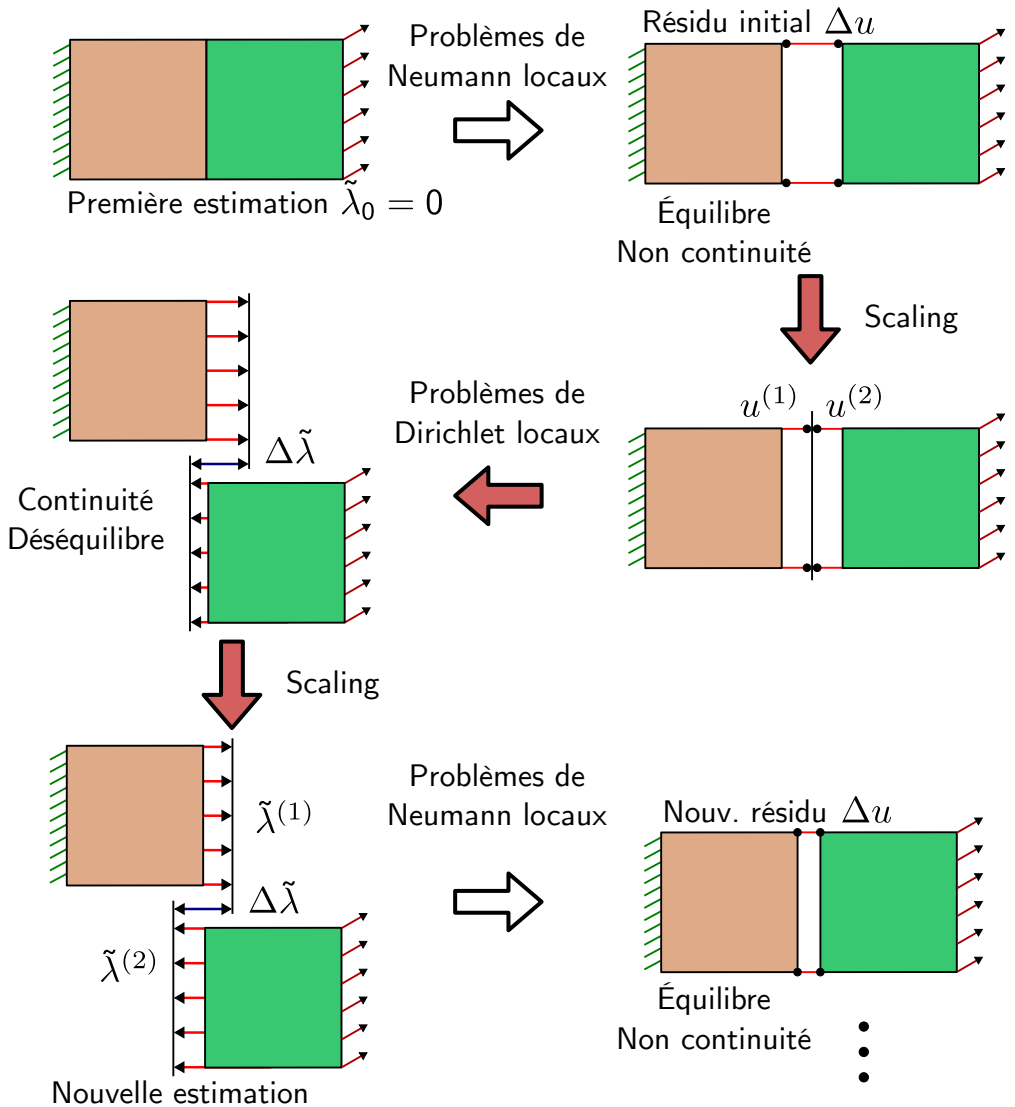


Continuité

$u_b{}^{(1)} = u_b{}^{(2)}$

Algorithme 3 : FETI-I

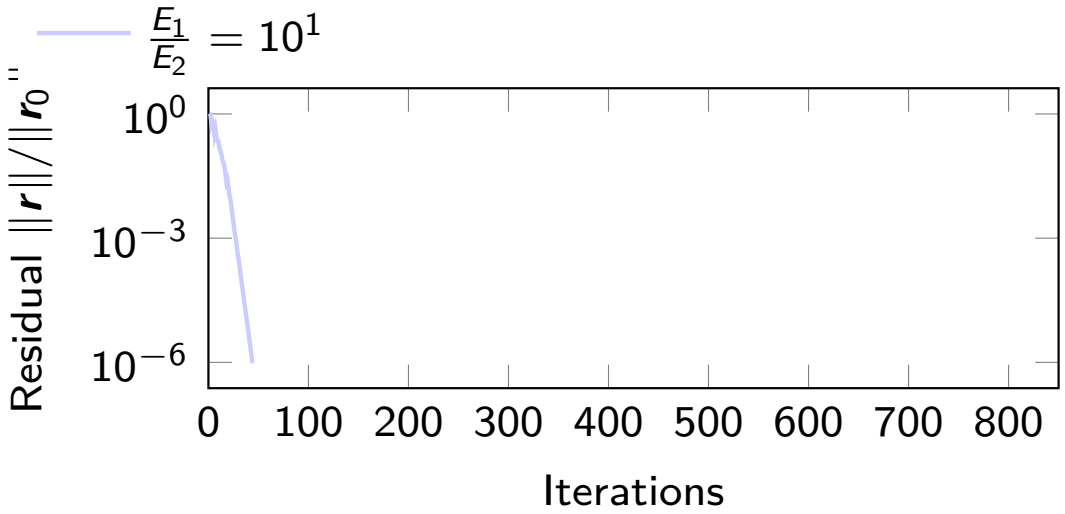
$r_0 = P^T(d - F\lambda_0)$; $z_0 = \tilde{S} r_0$
$w_0 = P z_0$; $\tilde{\lambda}_0 = 0$; $i = 0$
while $\sqrt{r_i^T z_i}$ **do**
  $q_i = F w_i$
  $\delta_i = q_i^T w_i$
  $\gamma_i = z_i^T r_i$
  $\alpha_i = \gamma_i / \delta_i$
  $\tilde{\lambda}_{i+1} = \tilde{\lambda}_i + w_i \alpha_i$
  $r_{i+1} = r_i - P^T q_i \alpha_i$
  $z_{i+1} = \tilde{S} r_{i+1}$
  $w_{i+1} = P z_{i+1}$
  for $0 \leq j \leq i$ **do**
    $\Phi_{i,j} = q_j^T w_{i+1}$
    $w_{i+1} \leftarrow w_{i+1} - (\Phi_{i,j}/\delta_j) w_j$
  $i \leftarrow i + 1$
$\lambda = \lambda_0 + \tilde{\lambda}_i$

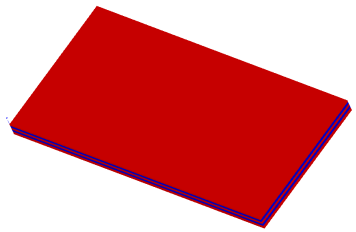


Méthode FETI-I

Exemple de résultats

- Les méthodes DD classiques restent sensibles au conditionnement du problème
- Cas pathologiques (hétérogénéité, quasi incompressibilité, ...)
- Des solutions existent !



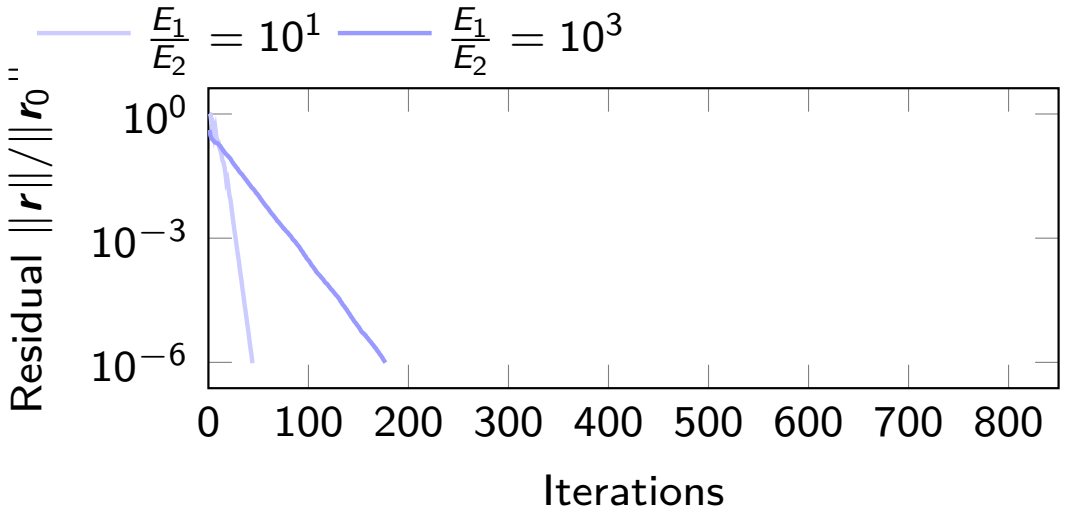


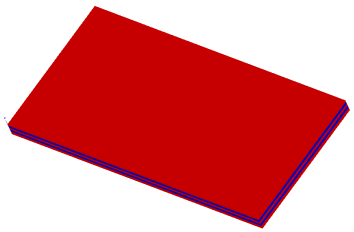
Heterogeneous plate

Méthode FETI-I

Exemple de résultats

- Les méthodes DD classiques restent sensibles au conditionnement du problème
- Cas pathologiques (hétérogénéité, quasi incompressibilité, ...)
- Des solutions existent !



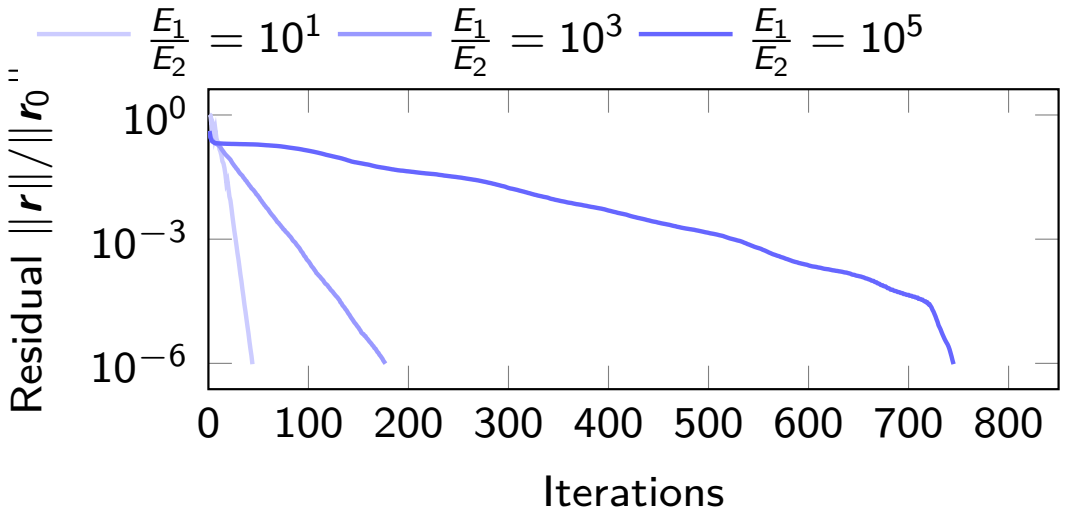


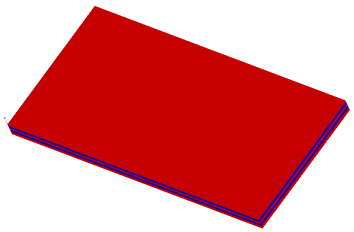
Heterogeneous plate

Méthode FETI-I

Exemple de résultats

- Les méthodes DD classiques restent sensibles au conditionnement du problème
- Cas pathologiques (hétérogénéité, quasi incompressibilité, ...)
- Des solutions existent !



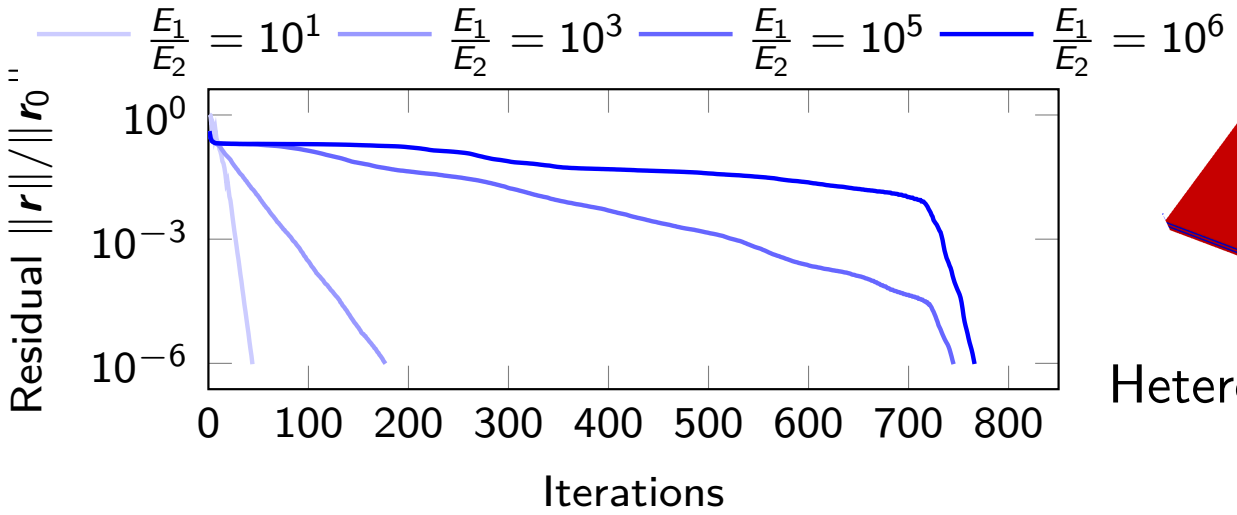


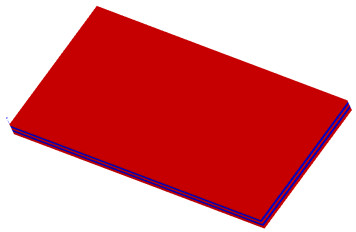
Heterogeneous plate

Méthode FETI-I

Exemple de résultats

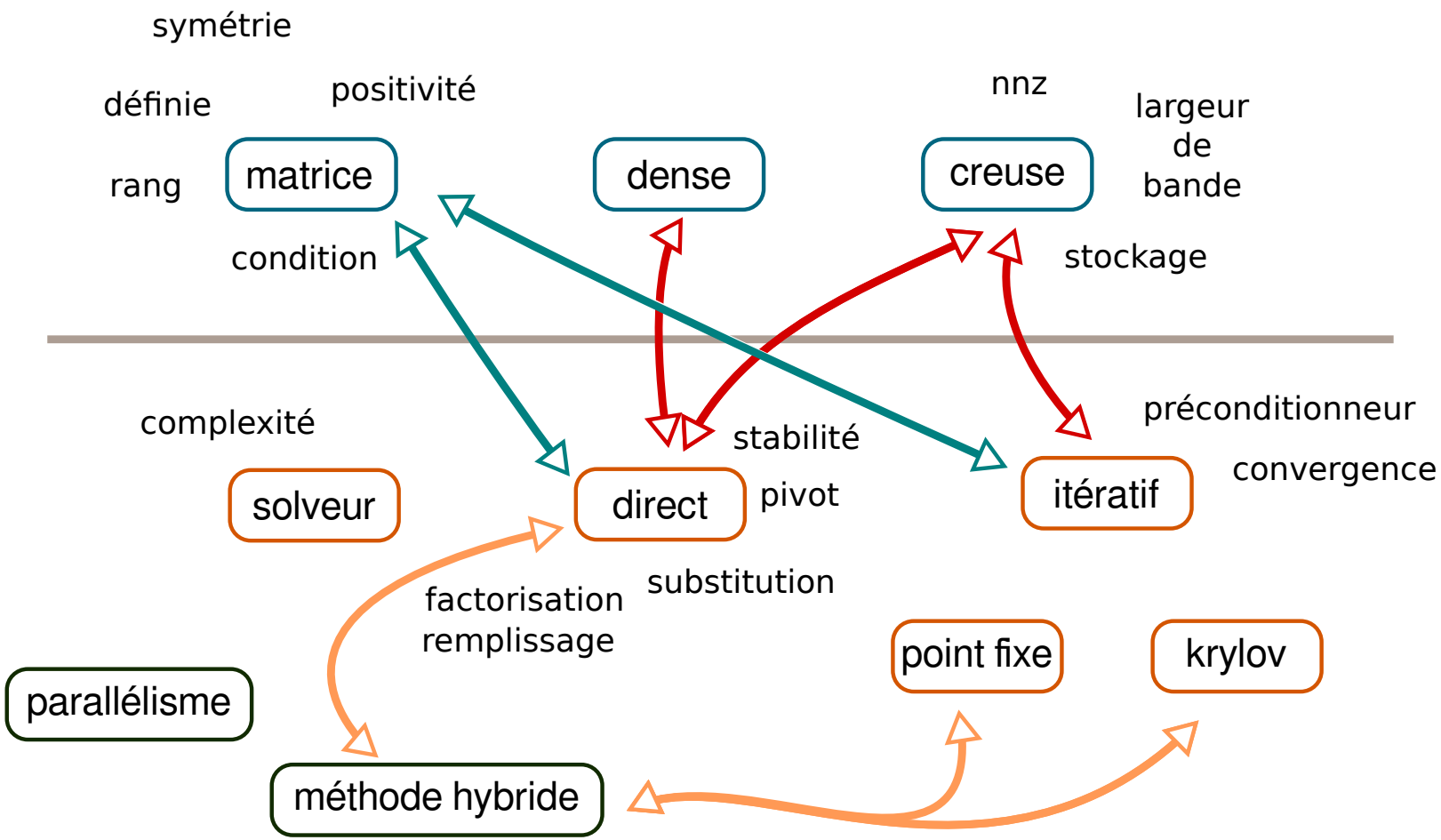
- Les méthodes DD classiques restent sensibles au conditionnement du problème
- Cas pathologiques (hétérogénéité, quasi incompressibilité, ...)
- Des solutions existent !





Heterogeneous plate

Schéma synoptique



Références

- Méthodes numériques en général
  - Quarteroni, A. M., Sacco, R., & Saleri, F. (2008). Méthodes numériques : algorithmes, analyse et applications. Springer Science & Business Media.
- Algèbre linéaire dense
  - Golub & van Loan : Matrix Computations, 3rd ed., Johns Hopkins, 1996.
- Solveurs directs creux
  - I. Duff, A. Erisman, J. Reid : Direct Methods for Sparse Matrices, Oxford University Press, 1986.
  - T. Davis : Direct Methods for Sparse Linear Systems, SIAM,2006.
- Solveurs de Krylov
  - Y. Saad : Iterative Methods for Sparse Linear Systems, 2nd ed., pp. 103–128, SIAM, 2003.
  - Van der Vorst, H. A. (2003). Iterative Krylov methods for large linear systems (Vol. 13). Cambridge University Press.