Conception d'un solveur linéaire creux parallèle hybride direct-itératif

Soutenance de thèse

Jérémie GAIDAMOUR

LaBRI et INRIA Bordeaux - Sud-Ouest (équipe Bacchus)

8 décembre 2009

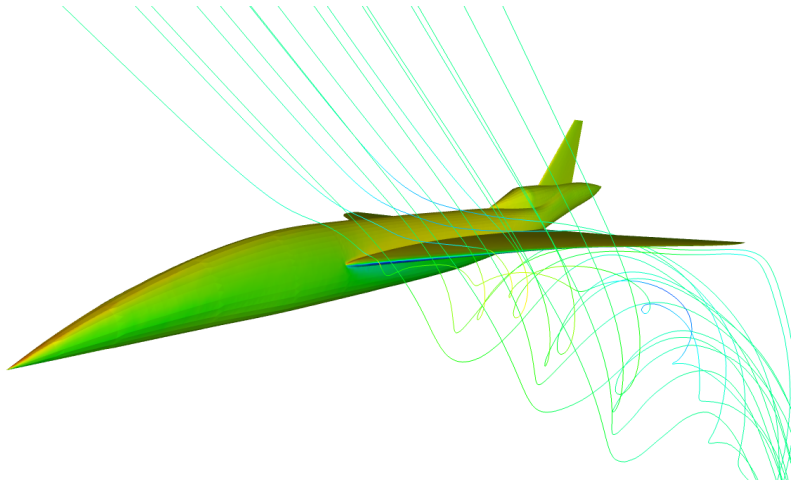
Plan

Contexte

On s'intéresse à la résolution de systèmes linéaires creux :

$$A.x = b$$

C'est une brique fondamentale de nombreuses simulations numériques (ex : résolution d'équations aux dérivées partielles par la méthode des éléments finis).



Définition

On dit qu'une matrice est creuse lorsqu'elle comporte une forte proportion de coefficients nuls.

⇒ On utilise des algorithmes et des structures de données qui prennent en compte la structure creuse des matrices pour réduire les coûts en mémoire et en calcul.

Objectif

Objectif :

Concevoir un solveur linéaire creux parallèle capable de résoudre des systèmes difficiles et/ou avec une grande précision à moindre coût mémoire.

On souhaite se positionner entre 2 grandes familles de solveurs :

- Les méthodes directes (factorisation exacte).
- Les méthodes itératives (méthode de Krylov préconditionnée, décomposition de domaine, multigrille algébrique, ...).

Factorisation

On s'intéresse aux différentes méthodes de résolution utilisant des factorisations :

A une permutation près, une matrice A régulière peut être décomposée sous la forme $A = L.U$ où :

- L est une matrice triangulaire inférieure à diagonale unité,
- U est une matrice triangulaire supérieure.

La résolution du système $A.x = b$ se ramène alors à des résolutions de systèmes triangulaires :

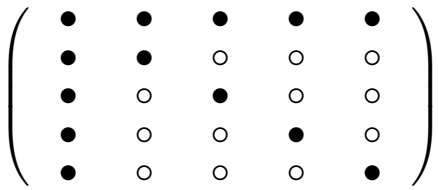
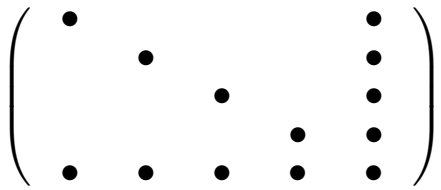
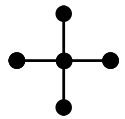
$$\begin{cases} L.y &= b \\ U.x &= y \end{cases}$$

Remplissage

Phénomène de remplissage des matrices creuses :

Au cours de la factorisation, de nouveaux termes non nuls apparaissent dans la matrice.

Le remplissage dépend de l'ordre d'élimination des inconnues :

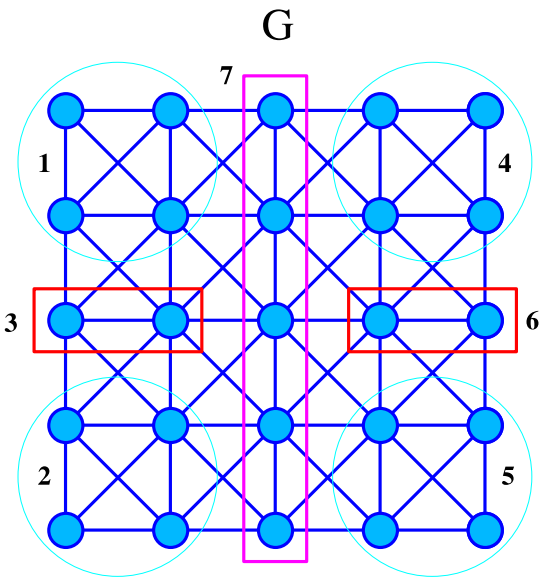
$A =$  $PAP^T =$  $G =$ 

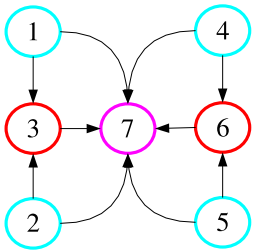
Outils d'analyse : Graphe G associé à A : arête $(i,j) \Leftrightarrow a_{ij} \neq 0$

Théorème de remplissage :

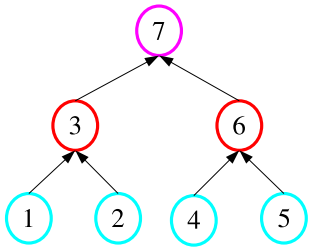
Tout terme $a_{ij} = 0$ de A devient non nul dans L (ou U) ssi il existe un chemin dans G du sommet i au sommet j ne passant que par des sommets de numéro inférieur à i et j.

Méthode directe (1/3)

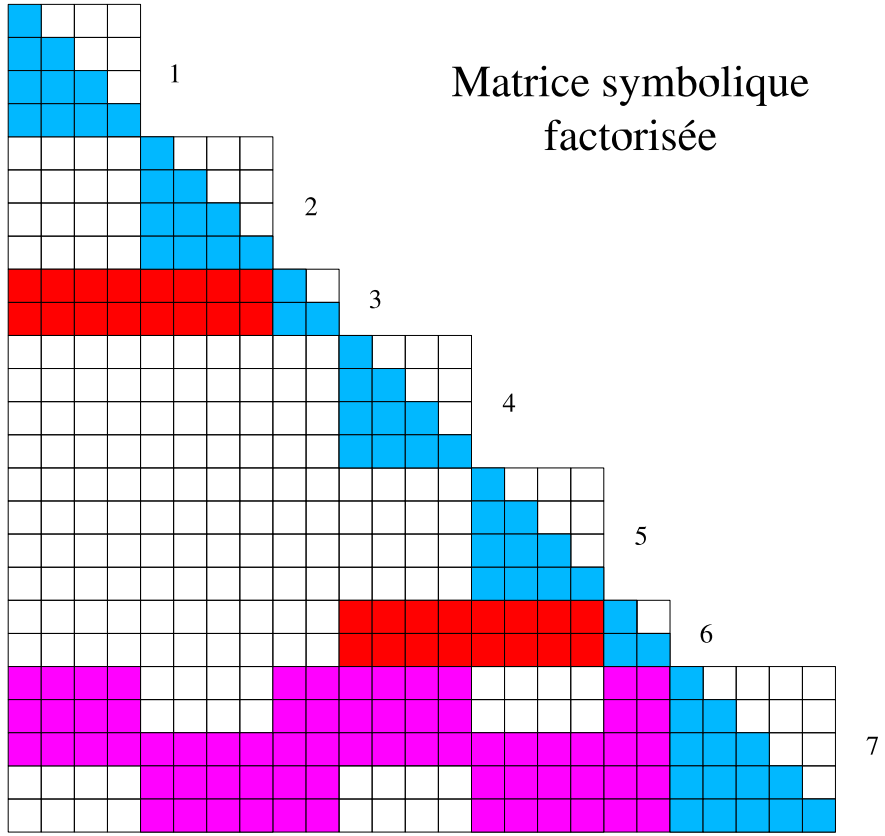




G*/ P



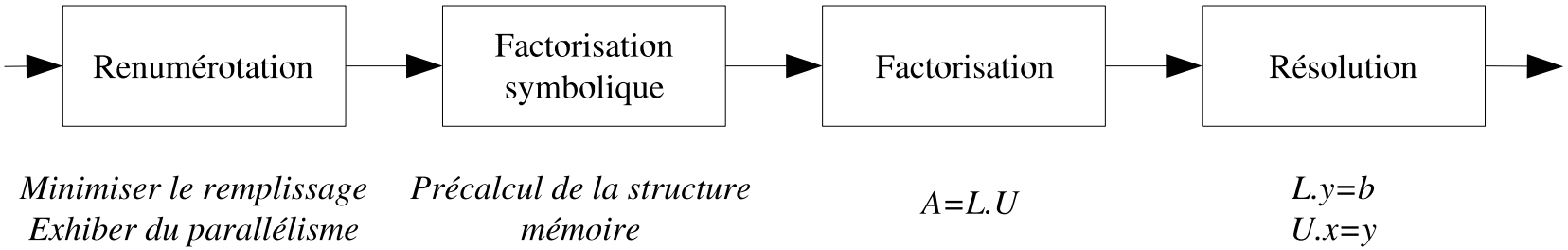
Arbre d'élimination



Matrice symbolique factorisée

Méthode directe (2/3)

Chaîne de résolution d'une méthode directe :



Renumérotation :

- Numérotation de type dissection emboîtée.

Factorisation symbolique par blocs :

- Partition P des inconnues.
- Graphe quotient $G^*/P \quad \Rightarrow (G/P)^* = G^*/P$.

Méthode directe (3/3)

Avantages :

- Utilisable comme boîte noire (approche générique).
- Très performante : algorithmiques par blocs denses (BLAS 3).
- Solution très précise.
- Robuste numériquement.
- Ressources nécessaires prédictibles (équilibrage de charge).

Inconvénient :

- Mémoire et nombre d'opérations très importants (3D).

Méthode itérative préconditionnée (1/2)

Méthode de Krylov :

Calculer successivement des approximations de la solution jusqu'à une précision voulue (ex : Krylov).

Préconditionnement :

$$M^{-1}.A.x = M^{-1}.b$$

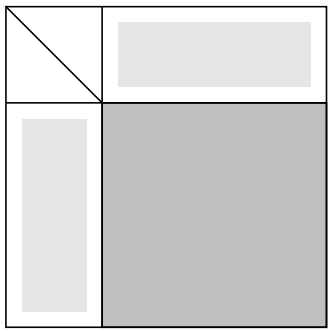
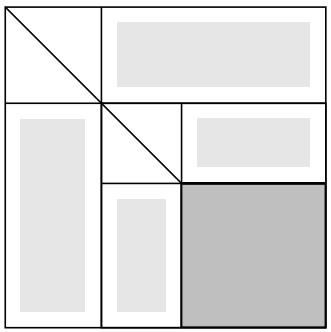
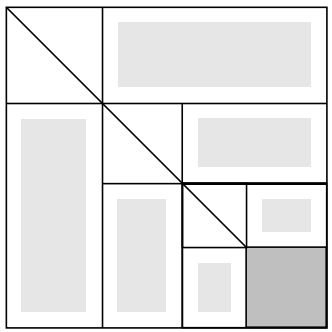
Choix possible pour M^{-1} : $\simeq A^{-1} = U^{-1}.L^{-1}$

Le principe d'une factorisation incomplète (ILU) est de limiter en cours de factorisation le remplissage des facteurs en ignorant une partie des coefficients des facteurs.

Méthode itérative préconditionnée (2/2)

Exemples :

- les factorisations ILU(k) utilisent un critère combinatoire pour limiter le remplissage.
- les factorisations ILUT (pour « Incomplete LU Threshold ») contrôlent le remplissage en fonction des valeurs numériques des coefficients.
- les factorisations ILUM utilisent une approche récursive (ex : ARMS).

Préconditionnement par factorisation incomplète

Avantages :

- Coût théorique faible.
- Faible consommation mémoire.

Inconvénients :

- Sensible à la difficulté du problème.
- Algorithmes généralement scalaires.
- Factorisation symbolique impossible en ILUT, de complexité du même ordre que la factorisation numérique pour ILU(k).
- Parallélisation souvent difficile.

Bilan

Ce qu'il faut pour être performant :

- Maîtriser le remplissage.
- Fournir des algorithmes par blocs qui permettent de mieux exploiter les calculateurs hautes performances.
- Exhiber du parallélisme.

On a besoin :

- d'une numérotation adaptée.
- d'un partitionnement ($\Rightarrow$bloc dense = BLAS).
- d'une phase d'analyse qui permet de bien gérer le parallélisme.

Plan

Introduction

But : construire un solveur itératif **robuste**, générique et parallèle

- Construire un préconditionneur algébrique pour une méthode de Krylov.
- Chercher un compromis en fonction de la difficulté numérique.

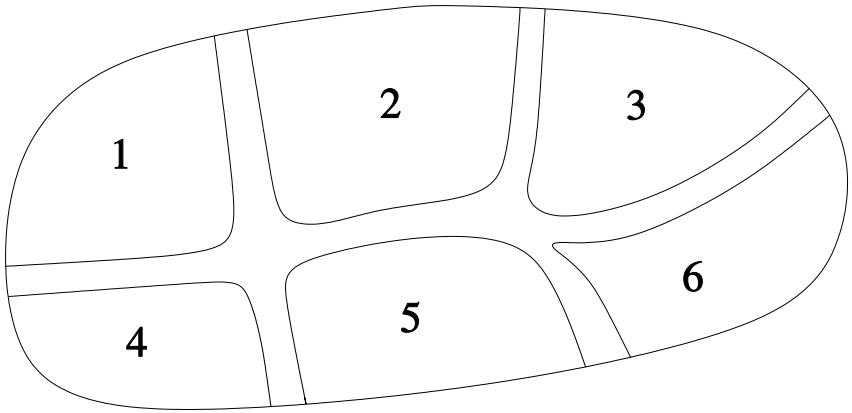
$\Rightarrow$ Coupler méthodes directes et itératives pour construire un solveur boîte grise.

Stratégie :

- Réutiliser des technologies du direct (BLAS, arbre d'élimination, factorisation symbolique).
- Tirer partie du parallélisme *naturel* des méthodes de type décomposition de domaine.

Méthode de décomposition de domaine

Ensemble de techniques s'appuyant sur le principe *diviser pour régner* :



Exemple : Méthode de Schwarz additive : $M = \sum_{i \in \{1,\ldots,6\}} A_i^{-1}$

Complément de Schur (1/2)

Comment coupler méthodes directes et itératives ?

Le système linéaire $A.x = b$ peut s'écrire sous la forme :

$$\begin{pmatrix} A_B & F \\ E & A_C \end{pmatrix} \cdot \begin{pmatrix} x_B \\ x_C \end{pmatrix} = \begin{pmatrix} y_B \\ y_C \end{pmatrix} \quad (1)$$

La système $A.x = B$ peut être résolu en trois étapes successives :

$$\begin{cases} A_B.z_B = y_B \\ S.x_C = y_C - E.z_B \\ A_B.x_B = y_B - F.x_C \end{cases} \quad (2)$$

où $S = A_C - E.\mathbf{A_B^{-1}}.F = A_C - E.\mathbf{U_B^{-1}}.\mathbf{L_B^{-1}}.F$

Complément de Schur (2/2)

Utilisation du complement de Schur pour coupler une méthode directe et une méthode itérative :

$$\begin{cases} A_B.z_B = r_B & (1) \\ S.x_C = r_C - E.z_B & (2) \\ A_B.x_B = r_B - F.x_C & (3) \end{cases}$$

$A_B = L_B.U_B$: (1) et (3) résolus par factorisation directe,

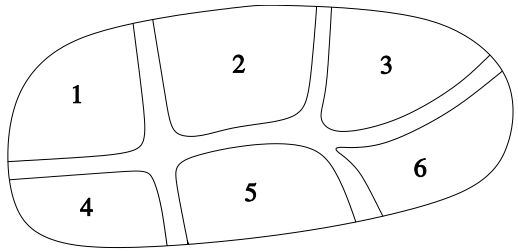
$S \simeq \tilde{L_S}.\tilde{U_S}$: (2) résolu par une méthode itérative.

Le processus itératif n'est nécessaire que sur le système réduit associé à S.

Utilisation d'une décomposition de domaine (1/2)

Décomposition de domaine :

$$A = \begin{pmatrix} A_B & F \\ E & A_C \end{pmatrix}$$



B : Nœuds intérieurs aux sous-domaines (factorisation exacte).

C : Nœuds de l'interface.

⇒ *On se concentre sur la résolution du système du complément de Schur par une méthode itérative.*

Ex : A. Haidar, L. Giraud, 2008

Utilisation d'une décomposition de domaine (2/2)

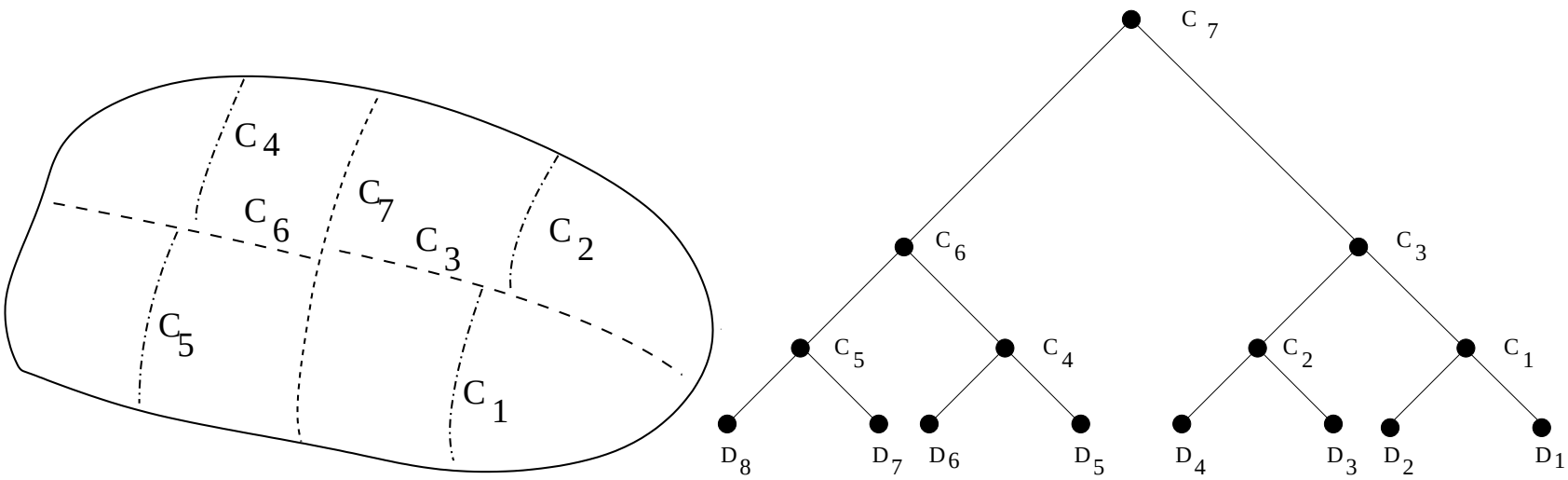
Etapes du solveur :

- **Construction d'une décomposition du graphe.**

  Décomposition du graphe d'adjacence de la matrice en sous-domaines (recouvrement de 1 noeuds).
- Méthode directe à l'intérieur des sous-domaines (factorisation LU).
- **Construction d'un préconditionneur pour le système réduit aux noeuds de l'interface**.
- Résolution itérative sur S (ex : GMRES).

Décomposition de domaine algébrique (1/3)

La décomposition du graphe est construite à partir d'une renumérotation de type dissection emboîtée (ex : SCOTCH, METIS).

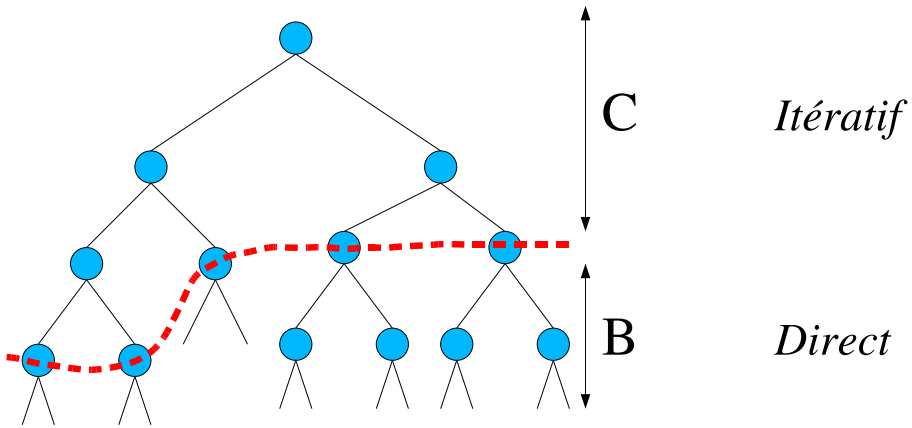


$\Rightarrow$ Minimise le recouvrement entre les domaines, qualité de l'interface.

Décomposition de domaine algébrique (2/3)

On choisit un *niveau* dans l'arbre d'élimination d'une méthode directe :

- Les sous-arbres enracinés à ce niveau constituent l'intérieur des sous-domaines.
- La partie haute de l'arbre correspond à l'interface.



Il est possible de choisir le ratio direct/iteratif en fonction de la difficulté du problème et de la précision voulue.

Décomposition de domaine algébrique (3/3)

Compromis entre le nombre de sous-domaines et la taille de l'interface ?

- Dépendant du problème.
- Paramètre d'ajustement du solveur, pour contrôler la mémoire et la convergence.

On souhaite utiliser de petits sous-domaines ($\simeq$ 1000 nœuds) :

- Nécessite moins de mémoire (moins de direct).
- Parallélisme induit très important.

Factorisation ILU de l'interface

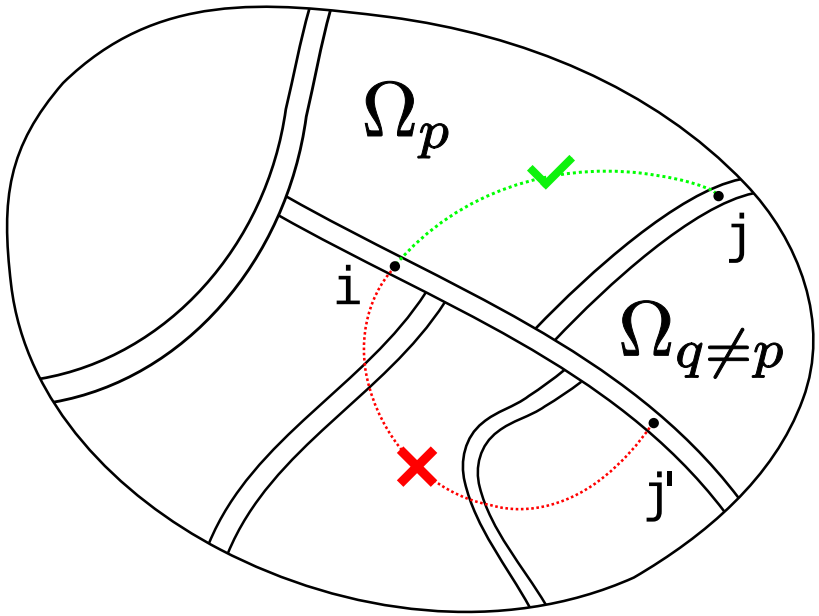
Pourquoi construire un préconditionneur **robuste** sur l'interface :

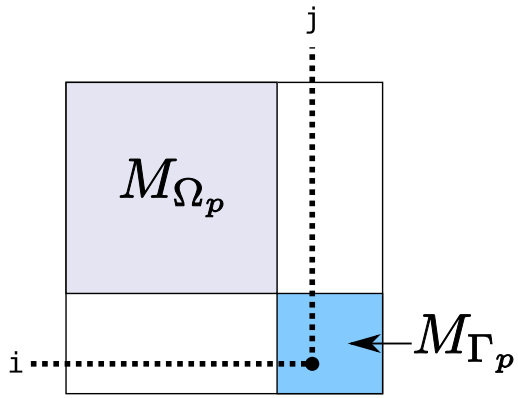
- Pour résoudre des problèmes difficiles.
- Pour utiliser de petits sous-domaines.

Le parallélisme des méthodes de décomposition de domaine est séduisant mais la convergence peut décroître rapidement avec le nombre de sous-domaines.

$\Rightarrow$ Construire un préconditionneur global du complément de Schur (ILU) en autorisant du remplissage que dans les matrices locales aux sous-domaines.

Notion de remplissage compatible





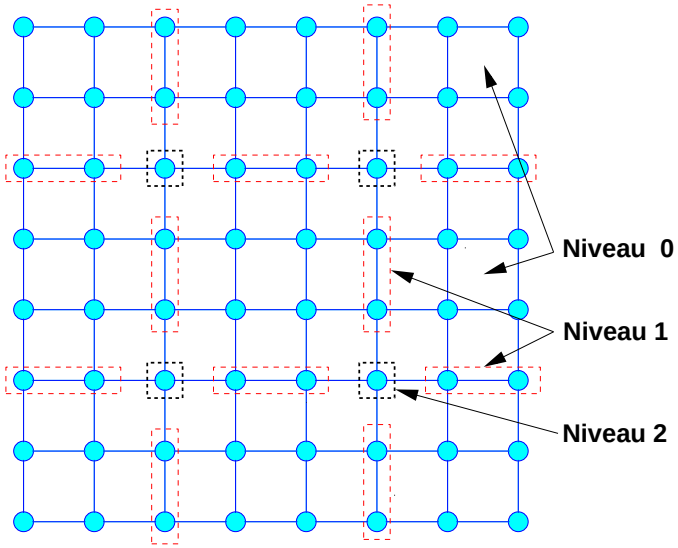
Partition des nœuds de l'interface en fonction des domaines auquels ils appartiennent.

Factorisation ILU de l'interface

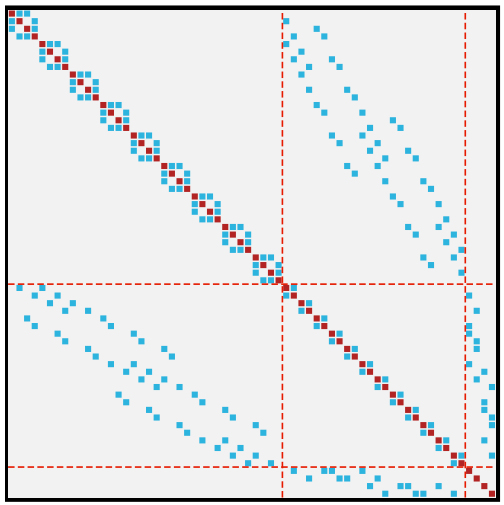
ILU global de S avec le motif de remplissage des matrices de Schur locales :

1. Pas de création d'arête de remplissage à l'extérieur des matrices locales aux domaines (conserver le parallélisme / un stockage local).
2. La factorisation globale nécessite une **renumérotation globale** des nœuds de l'interface (renumérotation compatible entre les domaines voisins).

- Décomposition hiérarchique de l'interface en connecteurs :



Grille 8×8



Matrice renumérotée

On utilise le graphe d'adjacence induit par cette partition pour définir une factorisation incomplète **par blocs**.

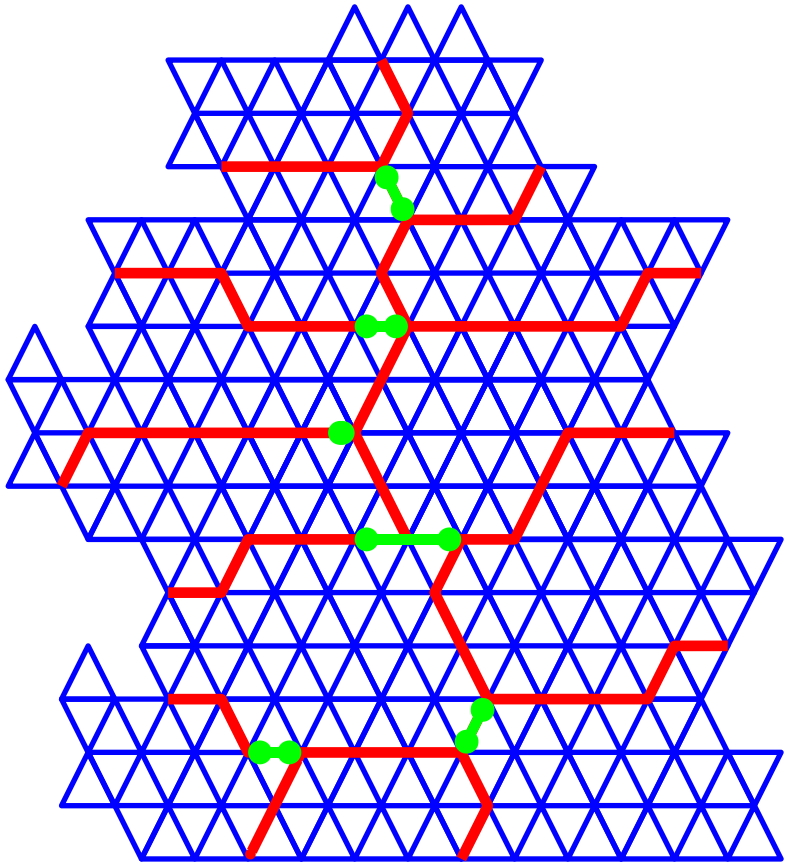
Remplissage compatible : on n'autorise que des arêtes entre les connecteurs adjacents au même sous-domaine.

Partition de l'interface pour des graphes irréguliers

Deux règles à respecter :

1. Les connecteurs d'un même niveau ne doivent pas être connectés
2. Un connecteur du niveau k est un séparateur pour au moins deux connecteurs du niveau $k-1$.

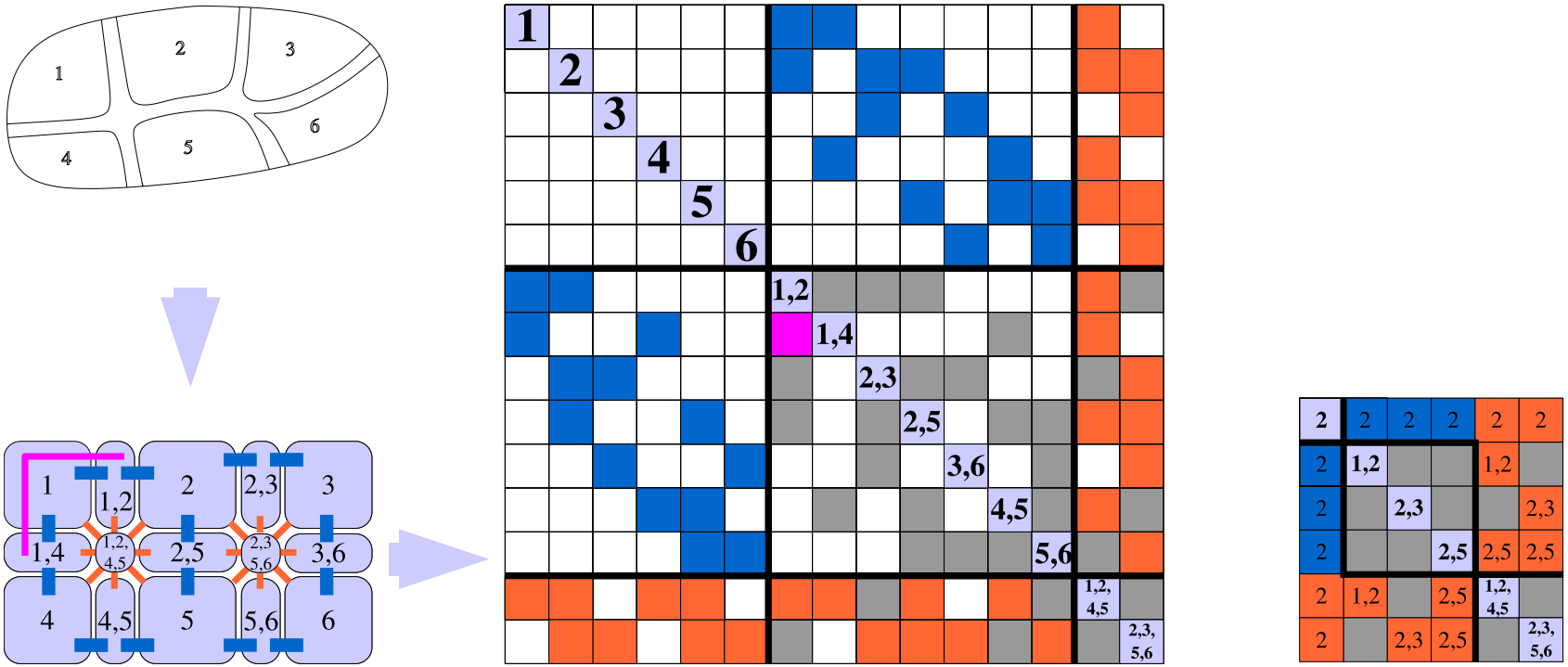
Exemple :



P. Hénon, Y. Saad, SIAM, 2006

Matrices (1/2)

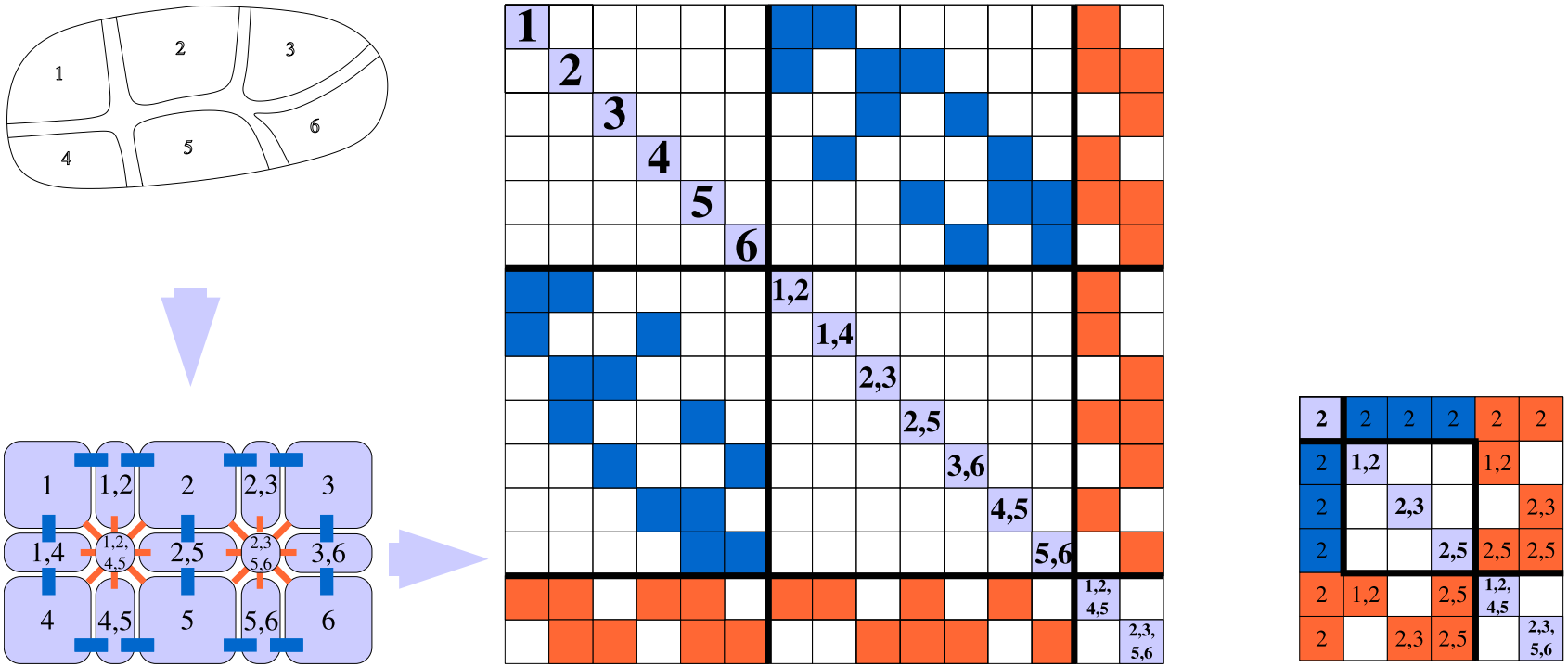
Factorisation ILU par blocs :



$\mathscr{R}_S$: Remplissage autorisé partout dans les matrices locales aux sous-domaines.

Matrices (2/2)

Factorisation ILU par blocs :

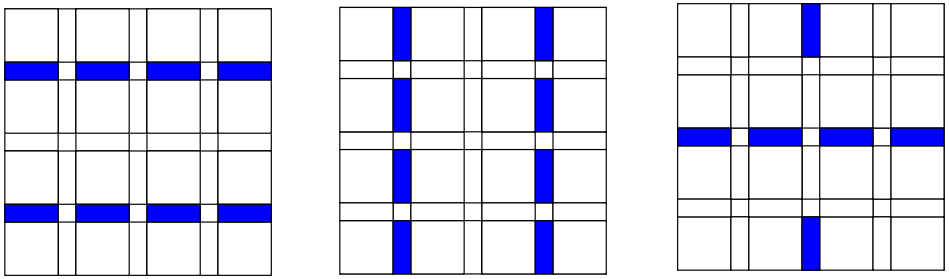


$\mathscr{R}_C$: Remplissage de la matrice initiale C (pas de remplissage entre connecteurs d'un même niveau).

Règles de remplissage (1/2)

$\mathscr{R}_S$:

- Stratégie la plus robuste.
- Dépendances dans l'élimination entre connecteurs voisins :



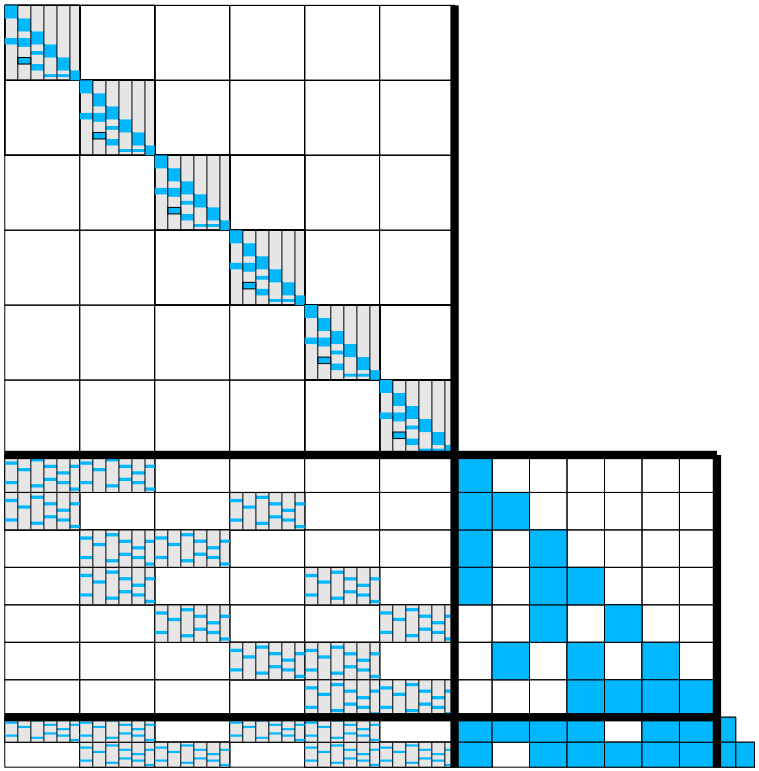
$\mathscr{R}_C$:

- Faible consommation mémoire et moins de calculs.
- Les connecteurs d'un même niveau peuvent être éliminés en parallèle.

$\mathscr{R}_S$+ ILUT :

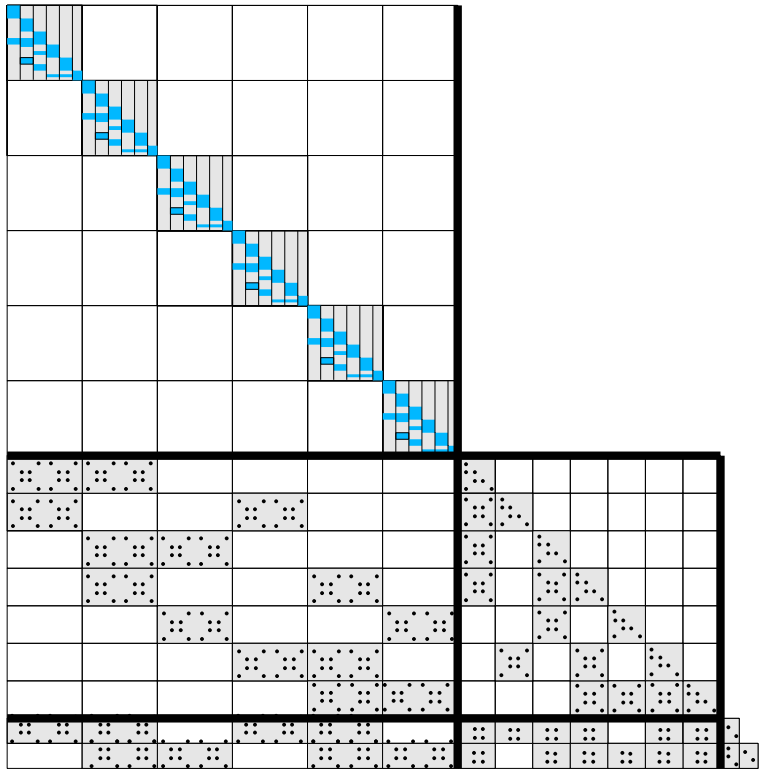
- Réduit la mémoire de la stratégie $\mathscr{R}_S$.
- Capture des termes importants en dehors du motif de $\mathscr{R}_C$.

Règles de remplissage (2/2)



Factorisation directe | ILU par blocs

R_S



Factorisation directe | ILUT scalaire

$R_S + \tau$

Expérimentations

AUDI : maillage 3D, mécanique des structures (PARASOL)

Matrice	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L,U)	OPC
AUDI ($\mathbb{R}$ sym.)	943 695	39 297 771	$41,2$	$5,4\,.\,10^{12}$

Conditions expérimentales :

- GMRES (pas de paramètre de "restart").
- Partitionneur : Scotch.
- $||b - A.x||/||b|| < 10^{-7}$.
- 2.6 Ghz quadri dual-core Opteron, 32 Go de mémoire.

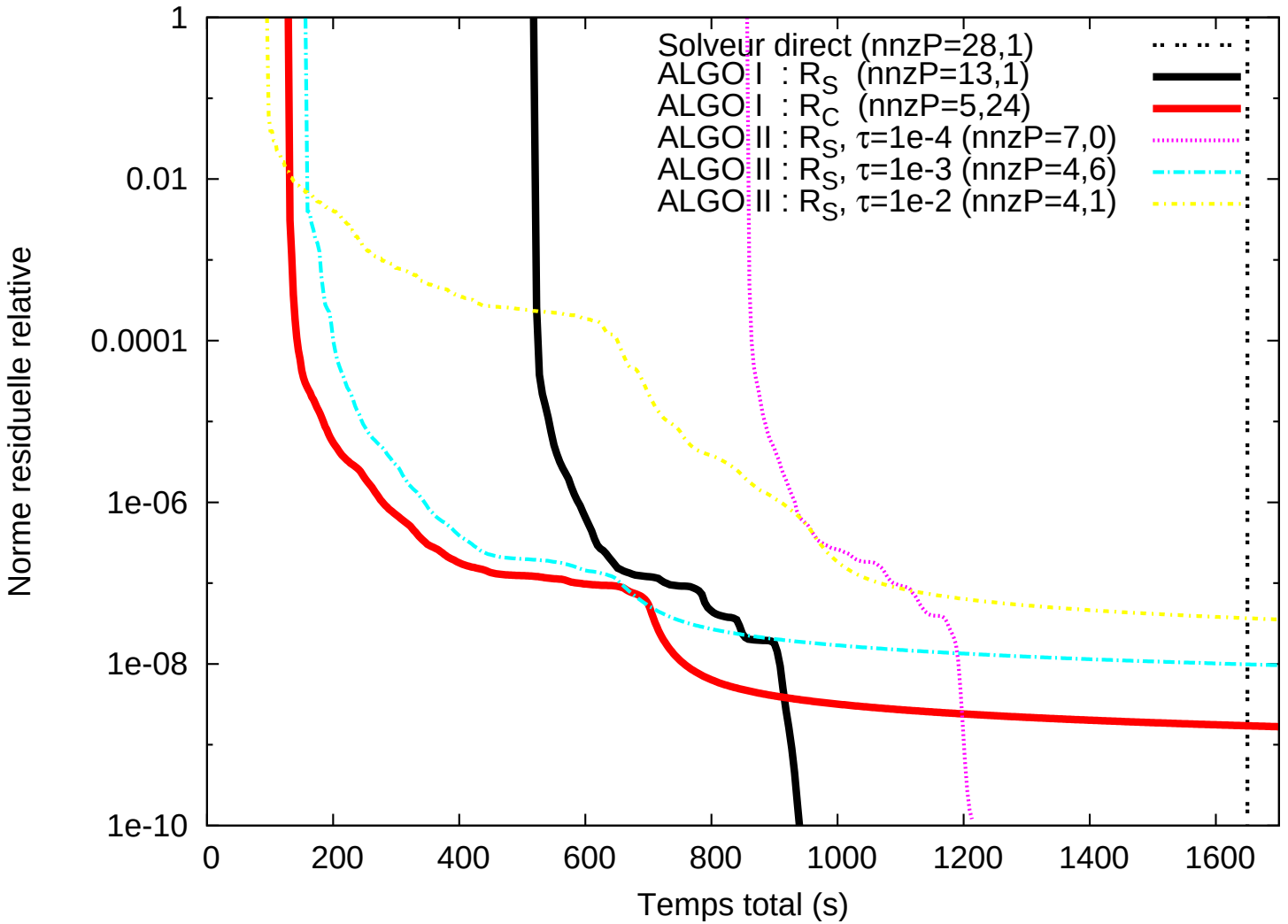
Expérimentations

Comparaison avec PETSc :

- GMRES préconditionné par méthode de Schwarz additive.
- Solveur local : solveur direct (MUMPS).
- Décomposition de domaine de HIPS (recouvrement=1) et avec un recouvrement augmenté (=5).

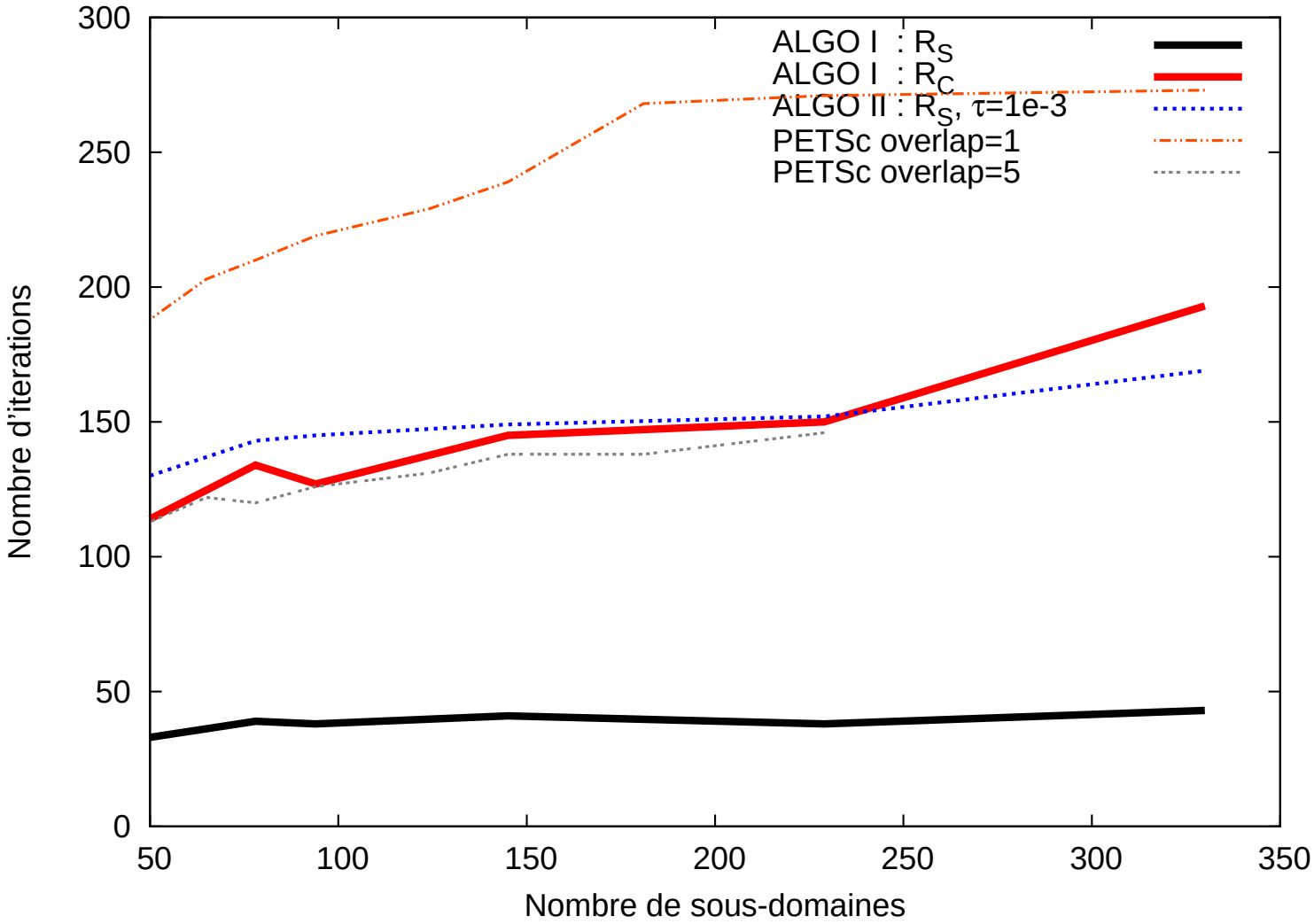
Historique de convergence (en temps)

AUDI (943 695) : 180 domaines (taille des domaines : 2000 noeuds)



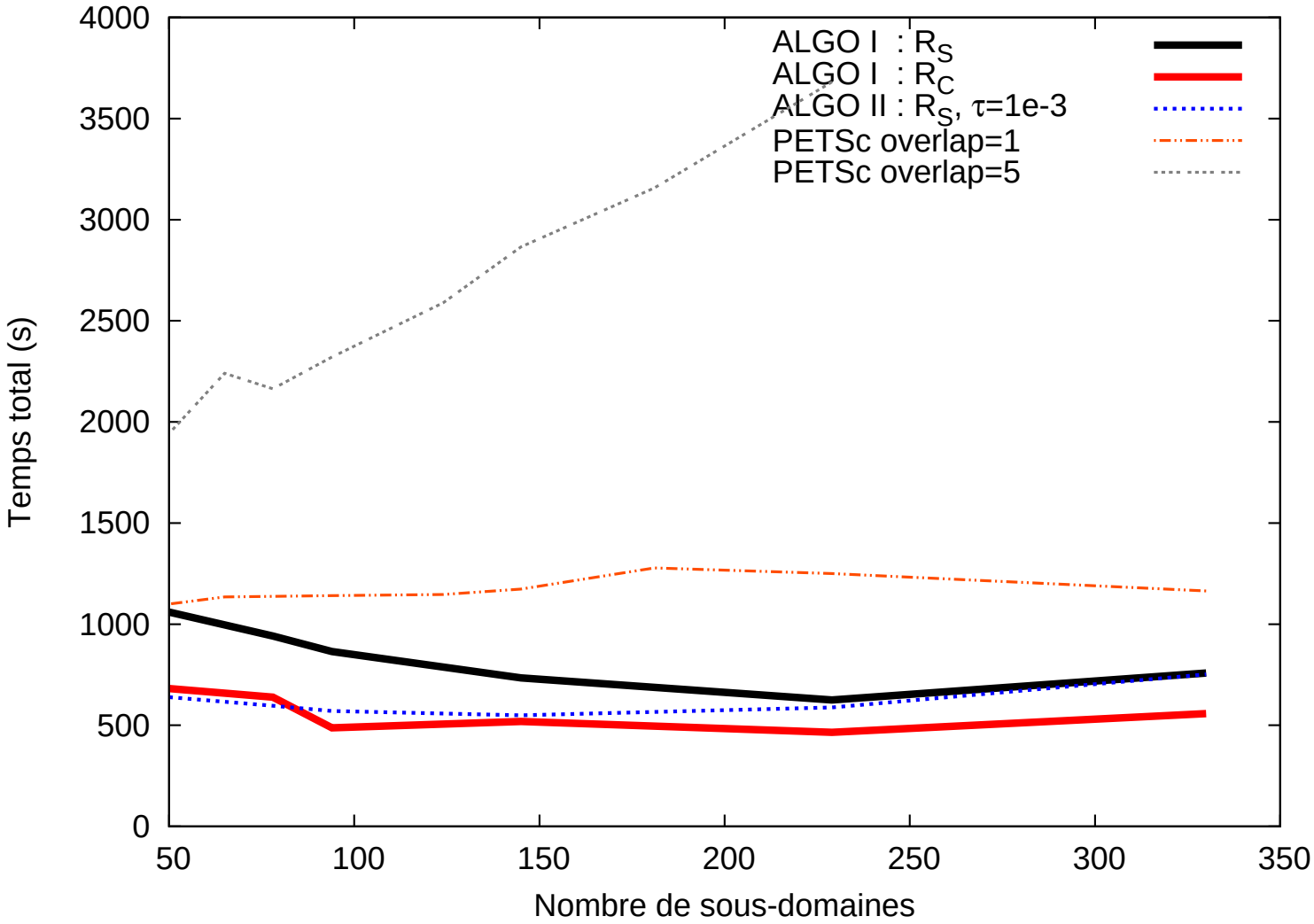
Itérations/nb. de domaines (norme rel. du résidu $= 10^{-7}$)

AUDI (943 695) :



Temps/nb. de domaines (norme rel. du résidu $= 10^{-7}$)

AUDI (943 695) :



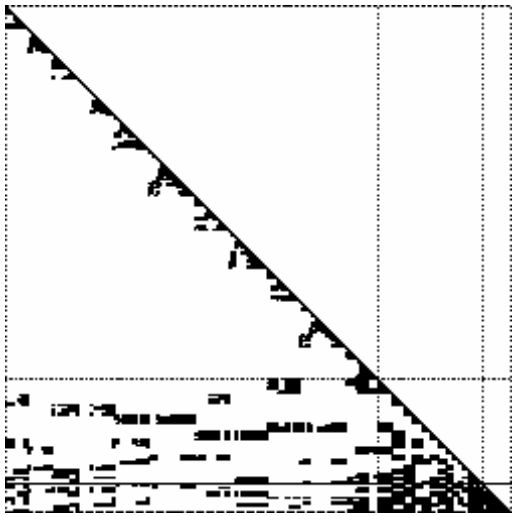
Plan

Problématique (1/2)

Question : Comment construire le préconditionneur du complément de Schur en limitant la consommation mémoire ?

$$\begin{pmatrix} L_B, U_B & L_B^{-1}F \\ EU_B^{-1} & S \end{pmatrix}$$

Factorisation symbolique de la matrice BCSSTK14 :



$\Rightarrow$ réduire le coût de EU_B^{-1}, $L_B^{-1}F$ et S.

Problématique (2/2)

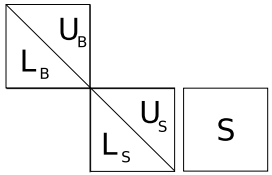
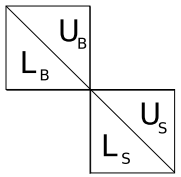
Réduire le coût de EU_B^{-1}, $L_B^{-1}F$ et S.

2 remarques importantes :

- EU_B^{-1} et $L_B^{-1}F$ sont seulement des matrices temporaires,
- La résolution itérative a uniquement besoin du calcul du produit $S.x$ (produit de Schur).

Produit de Schur (1/2)

Calcul du produit $S.x$:

Produit explicite	$S.x$	
Produit implicite	$(C - E.U^{-1}.L^{-1}.F).x$	

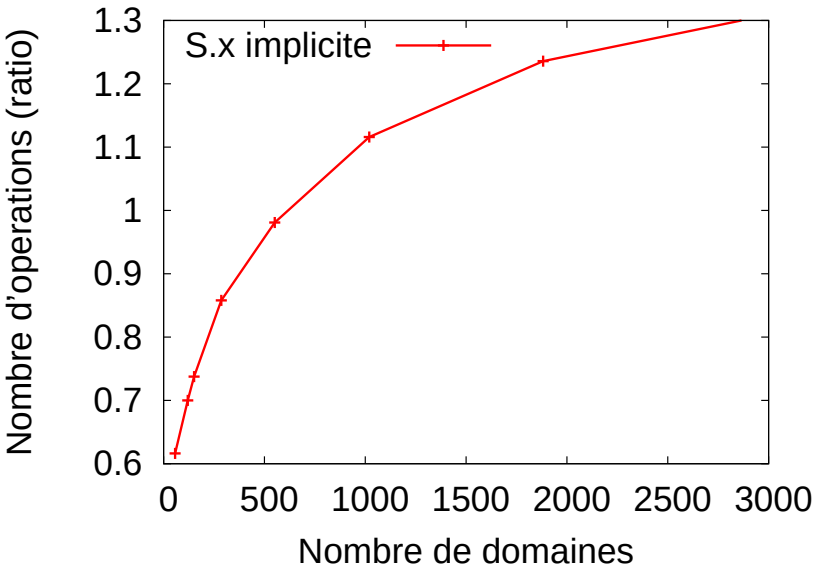
$$\begin{pmatrix} B & F \\ E & C \end{pmatrix}, \quad B = L_B.U_B$$

Produit de Schur (2/2)

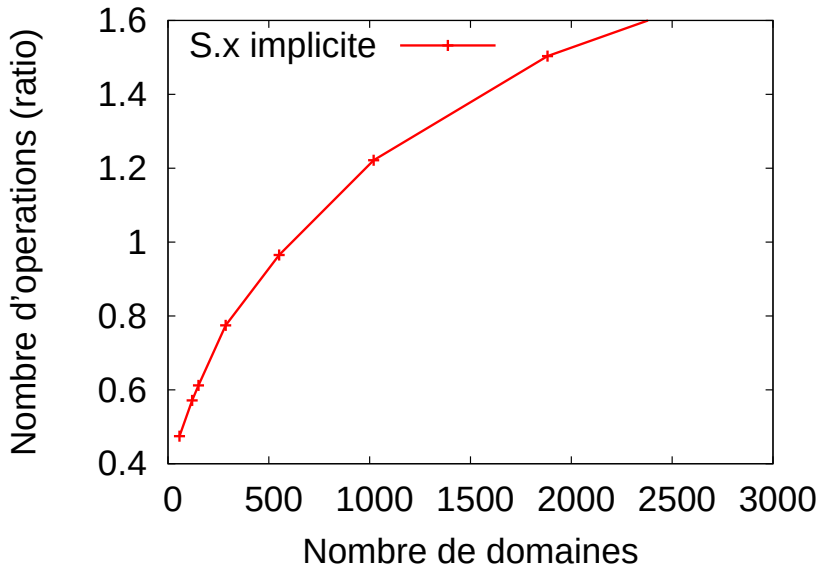
HALTERE : maillage 3D, problème d'électromagnétisme

Matrice	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L,U)	OPC
HALTERE ($\mathbb{C}$ sym.)	1 288 825	10 476 775	38,7	$7,5 \cdot 10^{12}$

Remplissage $\mathscr{R}_S$

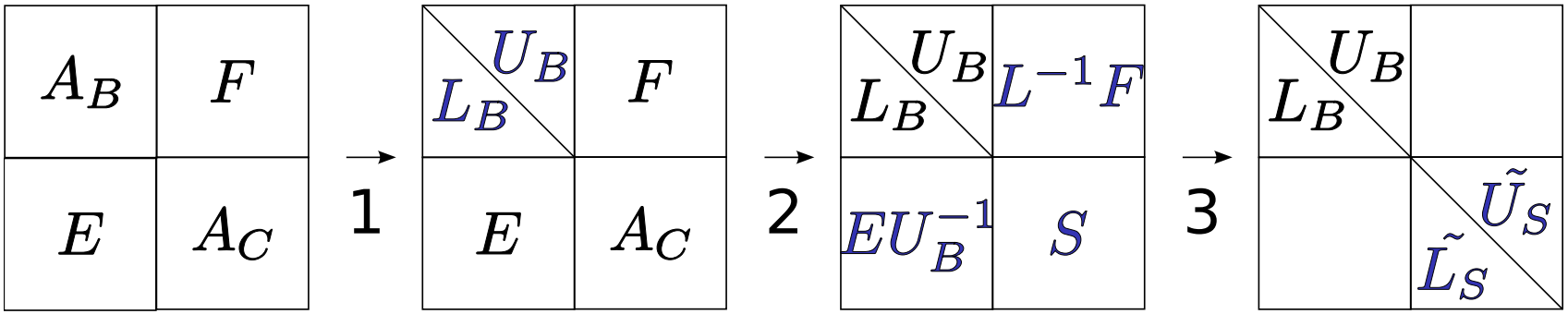


Remplissage $\mathscr{R}_C$



Algorithme I (1/2)

Factorisation de L_S, U_S à partir du calcul exacte de S : $\tau = 0$



Etapes de l'algorithme supernodal :

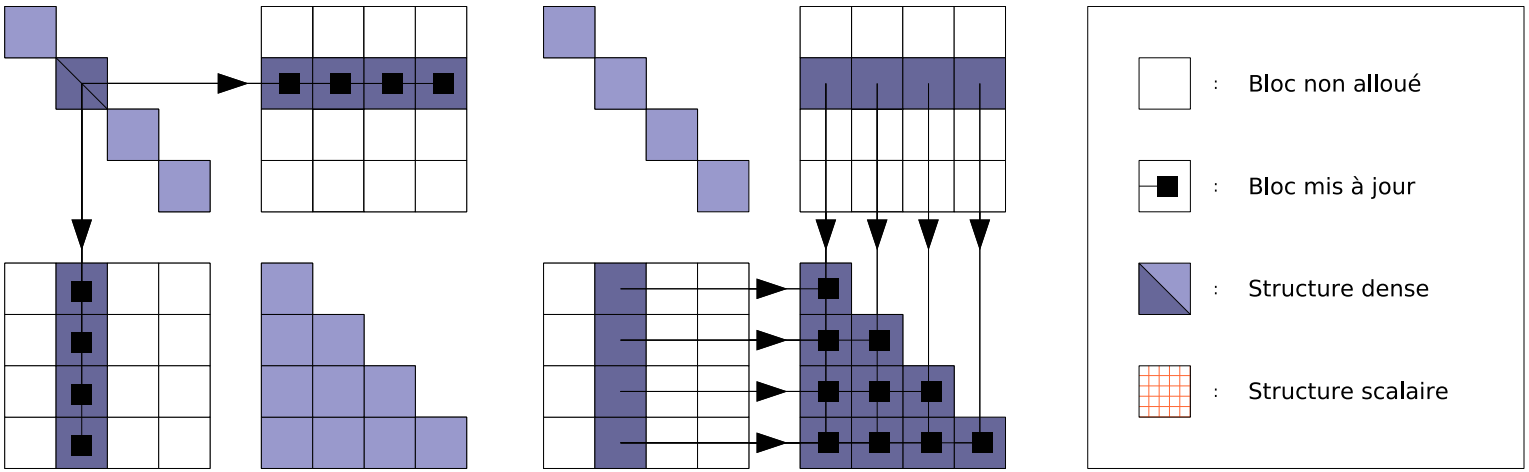
1. Factorisation exacte $A_B = L_B.U_B$.
2. Calcul de $W = EU_B^{-1}$, $G = L_B^{-1}F$ et du complément de Schur exact S selon $\mathscr{R}_C$ ou $\mathscr{R}_S$.
3. Factorisation incomplète de S selon $\mathscr{R}_C$ ou $\mathscr{R}_S$.

Comment éviter le stockage simultané de (W, G) et S ?

Algorithme I (2/2)

Détails de l'étape 2 : Calcul de $W = EU_B^{-1}$, $G = L_B^{-1}F$ et S :

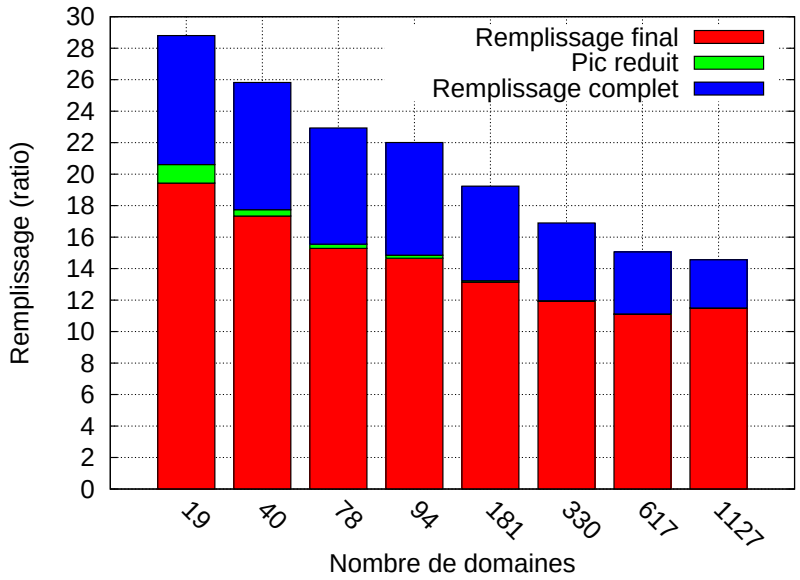
1. Préallocation de S (factorisation symbolique).
2. Pour chaque sous-domaine, faire :
   - Calcul supernodal du bloc-colonne de W (bloc-ligne de G) correspondant au sous-domaine.
   - Calcul des contributions de ce bloc-colonne/bloc-ligne à S.
   - Désallocation du bloc-colonne de W (bloc-ligne de G).



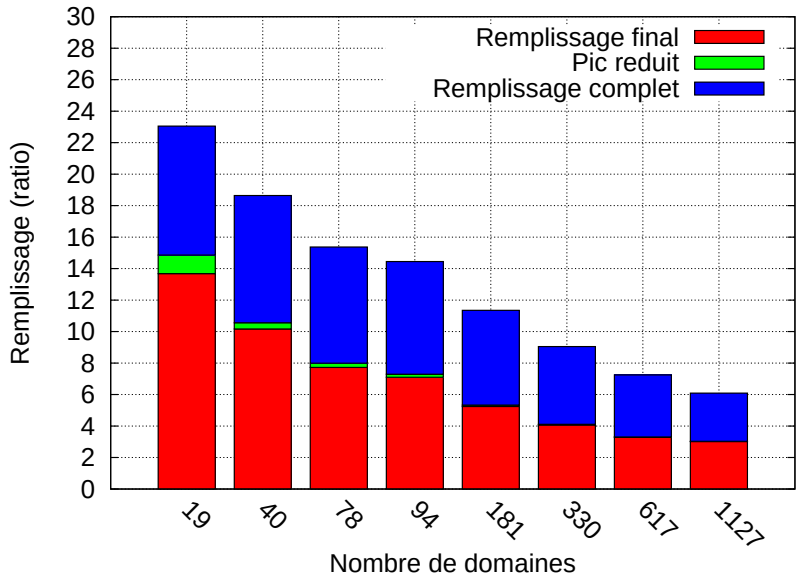
Calcul de S par un algorithme supernodal « Right-looking »

Algorithme I : Résultats

AUDI : Remplissage $\mathscr{R}_S$



AUDI : Remplissage $\mathscr{R}_C$



Algorithme II (1/3)

Factorisation L_S, U_S à partir d'une approximation de S : $\tau \neq 0$

$$S \simeq \tilde{S} \simeq \tilde{L_S}.\tilde{U_S}.$$

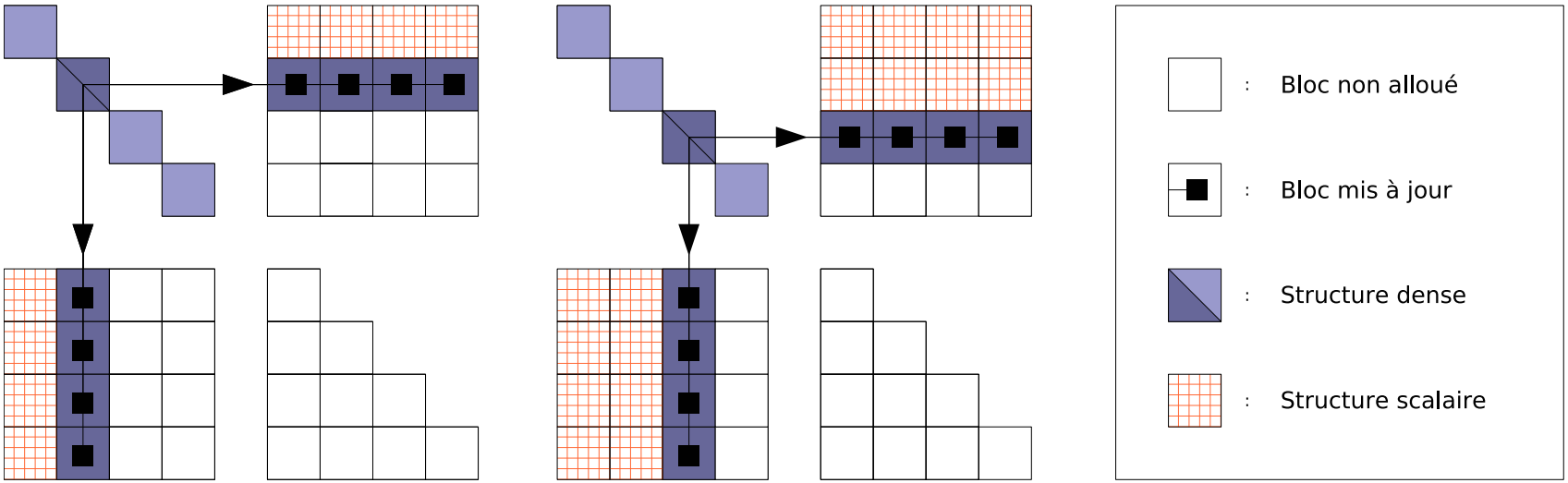
$\Rightarrow$ On accepte de réduire la qualité du préconditionneur pour consommer moins de mémoire.

Etapes :

1. Factorisation exacte de $A_B = L_B.U_B$ (algorithme supernodal).
2. Calcul d'une approximation de $W = EU_B^{-1}$, $G = L_B^{-1}F$.
3. Factorisation incomplète ILU(τ_S) de $\tilde{S}$.

Algorithme II (2/3)

2. Calcul d'une approximation de $W = EU_B^{-1}$, $G = L_B^{-1}F$:
   - Les matrices W et G sont calculées de manière supernodale.
   - Elles sont creusées en cours de calcul (seuillage numérique τ).

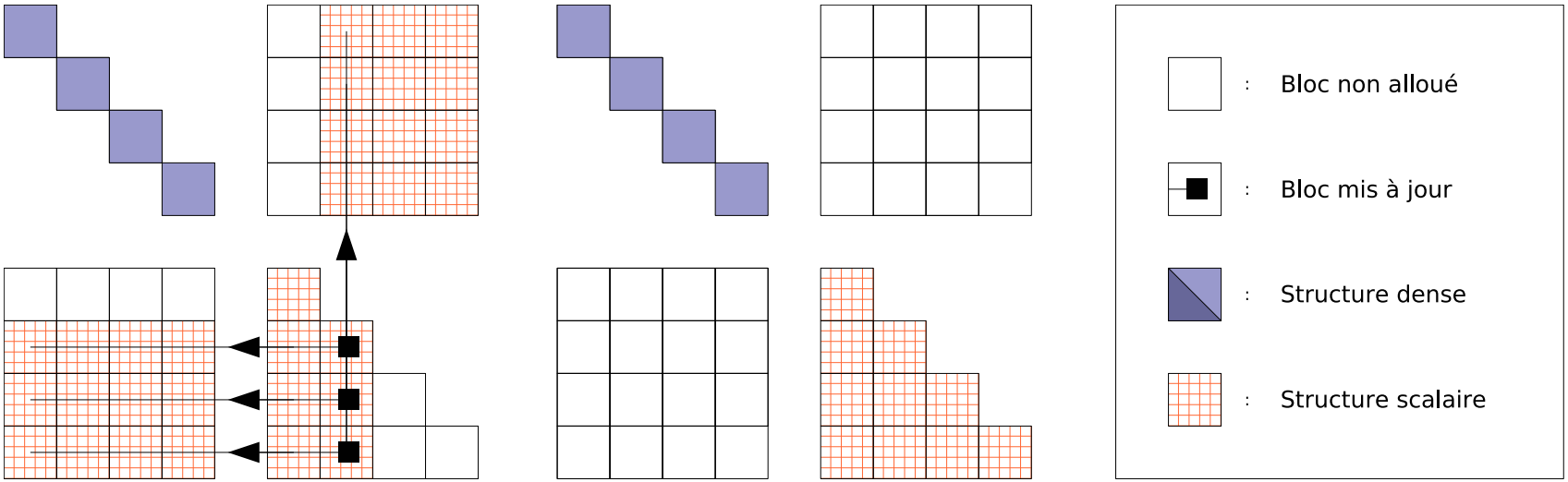


Calcul de $\tilde{S}$ par un algorithme supernodal « Left-Looking »

Algorithme II (3/3)

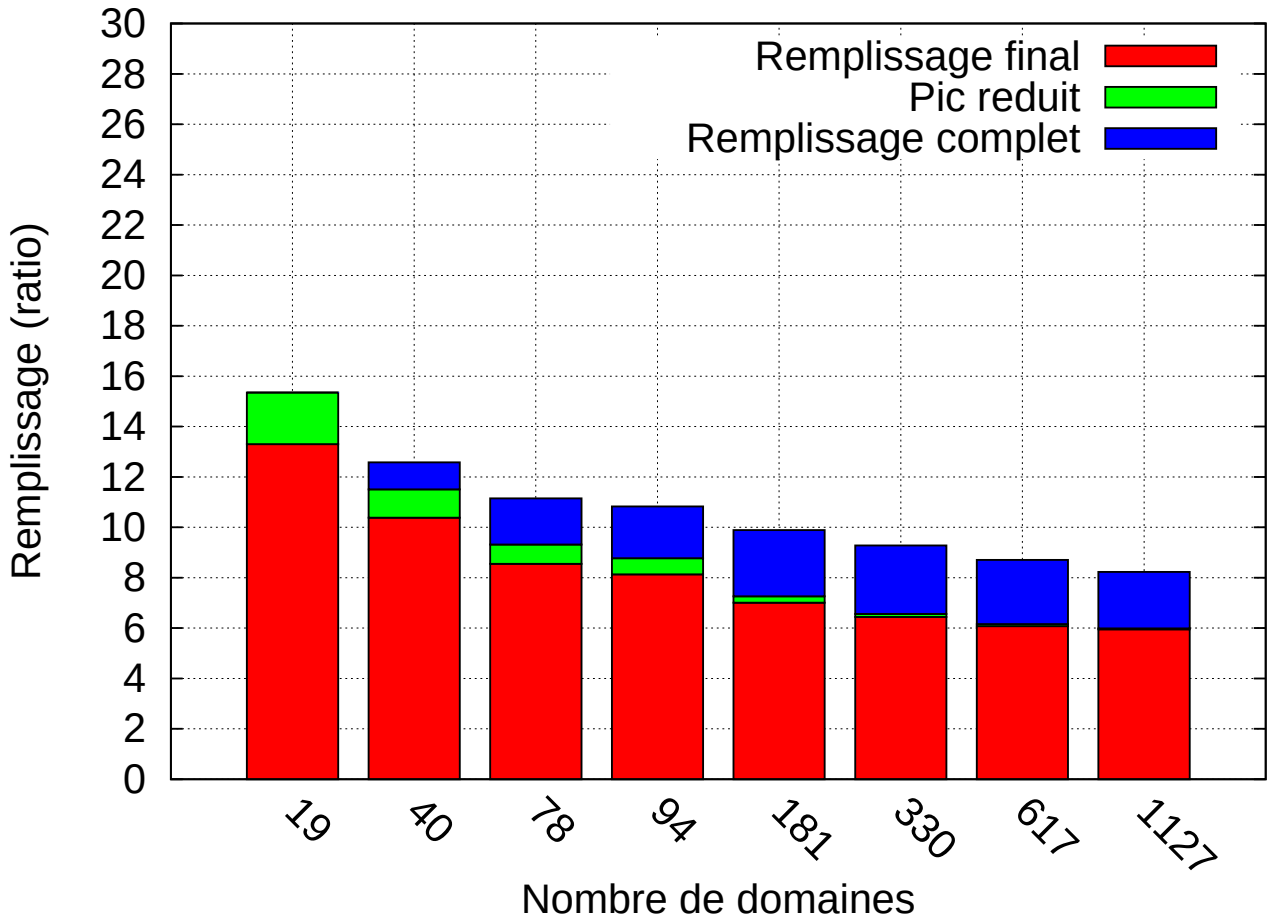
3. Factorisation incomplète « Left-Looking » de $\tilde{S}$:
   Algorithme colonne selon la partition supernodale de A_B. Le calcul de $\tilde{S}$ est entrelacé à la factorisation ILU(τ_S).

   Pour chaque sous-domaine, faire :
   - Calcul d'un bloc-colonne de $\tilde{S}$ à partir de $\tilde{W}$ et $\tilde{G}$.
   - Factorisation ILU(τ) immédiate de cette colonne pour limiter le remplissage.



Algorithme II : Résultats

AUDI : Remplissage $\mathscr{R}_S+ \tau = 10^{-4}$



Plan

Parallélisation

Opérations locales aux sous-domaines :

- Factorisation exacte de $A_B = L_B.U_B$.
- Calcul de W, G (ou $\tilde{W}$, $\tilde{G}$).
- Calcul de S (ou $\tilde{S}$).

  $\longrightarrow$ Pas de communication.

Opérations globales :

- Factorisation ILU par blocs ou ILUT du complément de Schur ($L_S.U_S$) :

  $\longrightarrow$ Communications uniquement entre sous-domaines voisins.

Interprétation des résultats séquentiels

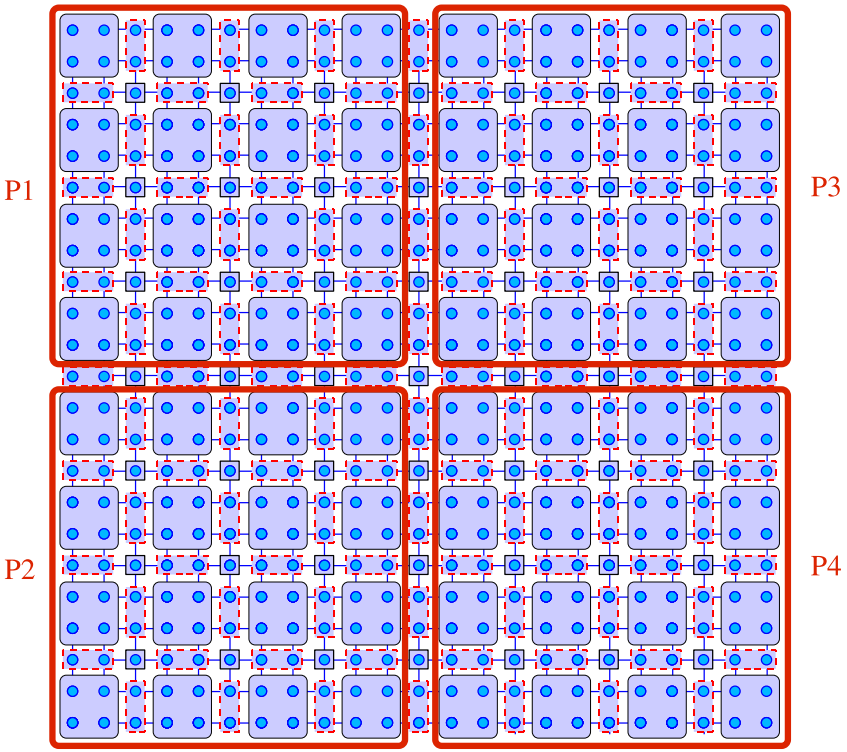
- Préconditionneur **robuste** du complément de Schur : le nombre d'itérations dépend peu du nombre de sous-domaines.
- Compromis *consommation mémoire/performance* controlé par la taille des sous-domaines :

  $\Rightarrow$ **Petits sous-domaines** ($\simeq$ 1000 nœuds).

  $\Rightarrow$ **Plusieurs sous-domaines par processeur.**

Elimination parallèle des inconnues

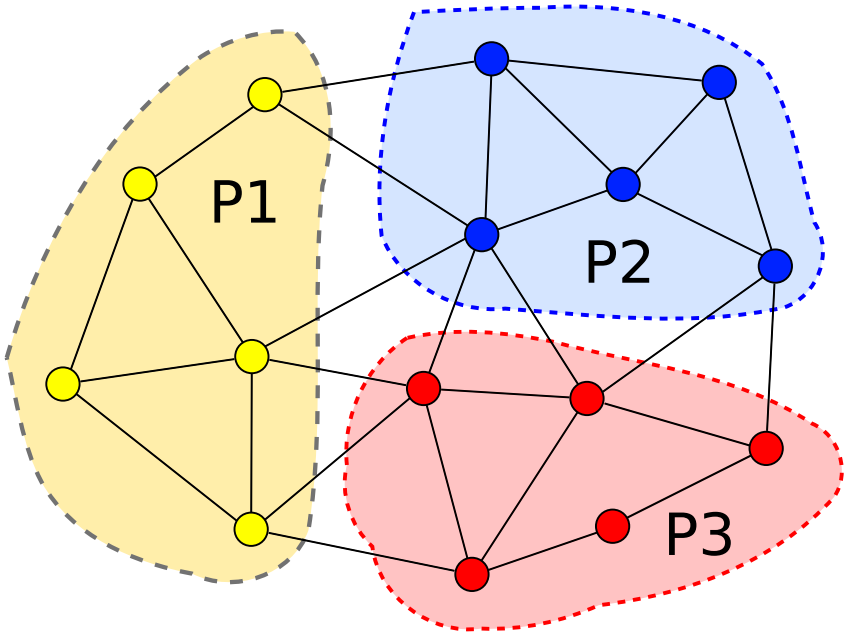
Le schéma de parallélisation est directement induit par la décomposition en petit sous-domaines :



Parallélisation (équilibrage de charge)

Distribution des sous-domaines aux processeurs :

- Obtenu par partitionnement du graphe quotient de la décomposition de domaine.
- Equilibrage du calcul du produit $S.x$ (étape de résolution) en calculant le nombre de non-zéros de l'intérieur des sous-domaines (pondération du graphe utilisant une factorisation symbolique).



Expérimentations

Cas tests :

- AUDI : mécanique des structures (3D, PARASOL).
- MHD1 : magnéto-hydrodynamique (3D, Univ. du Québec).
- HALTERE et AMANDE : électromagnétisme (3D, CEA).

Matrices		Nombre d'inconnues	Nombre de termes	Factorisation directe	
				nnz(L,U)	OPC
AUDI	($\mathbb{R}$ sym.)	943 695	39 297 771	$41,2$	$5,4\,.\,10^{12}$
MHD1	($\mathbb{R}$ non sym.)	485 597	24 233 141	$52,4$	$9,0\,.\,10^{12}$
HALTERE	($\mathbb{C}$ sym.)	1 288 825	10 476 775	$38,7$	$7,5\,.\,10^{12}$
AMANDE	($\mathbb{C}$ sym.)	6 994 683	58 477 383	$53,9$	$1,5\,.\,10^{13}$

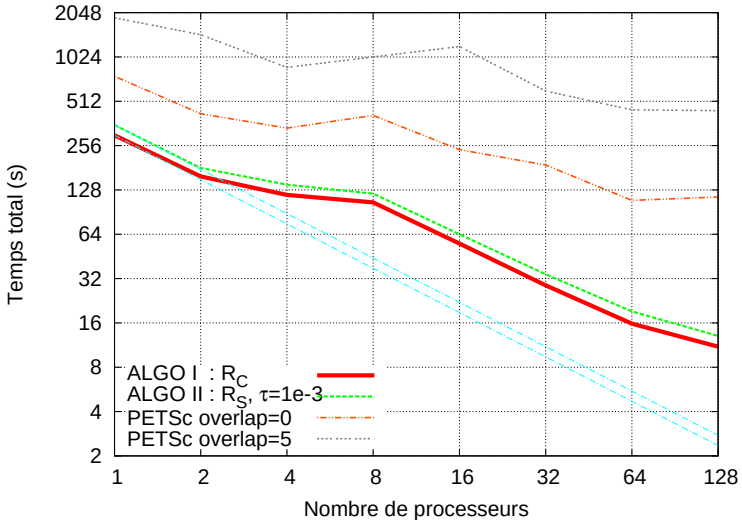
Conditions expérimentales :

Supercalculateur Jade du GENCI-CINES :

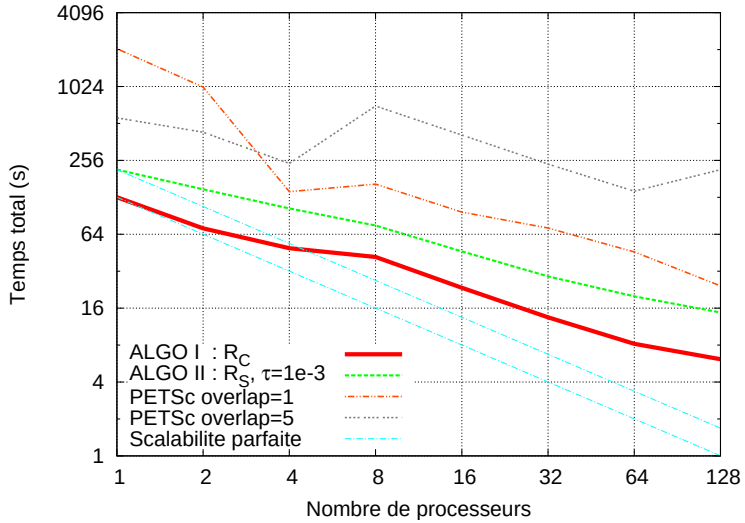
- Noeuds : 2 Intel Xeon Quad-Core 3,0 GHz.
- 32 Go de mémoire par nœud.
- Réseau Infiniband.

Scalabilité en temps (norme rel. du résidu $= 10^{-7}$)

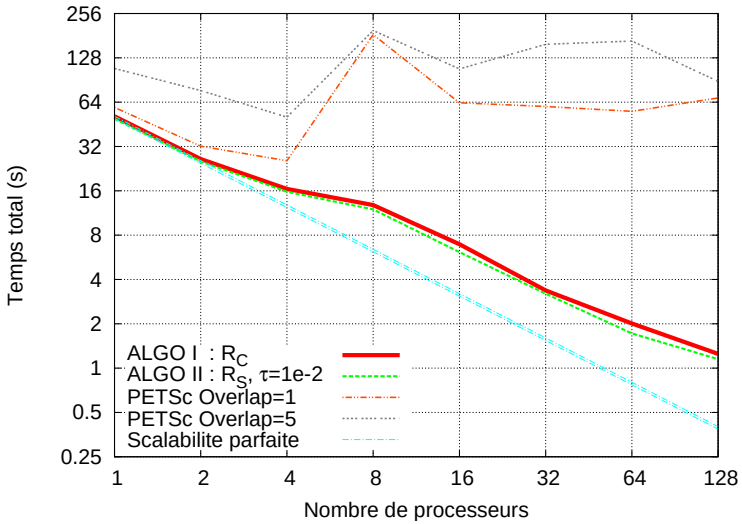
AUDI (181 domaines)



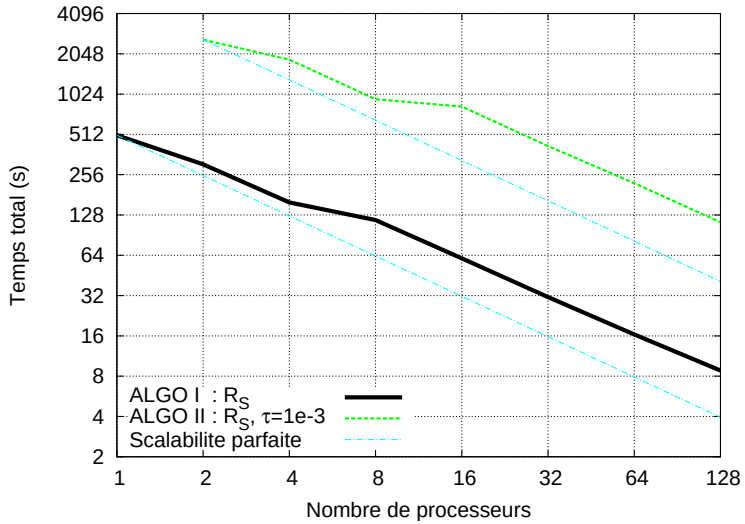
MHD1 (133 domaines)



HALTERE (551 domaines)

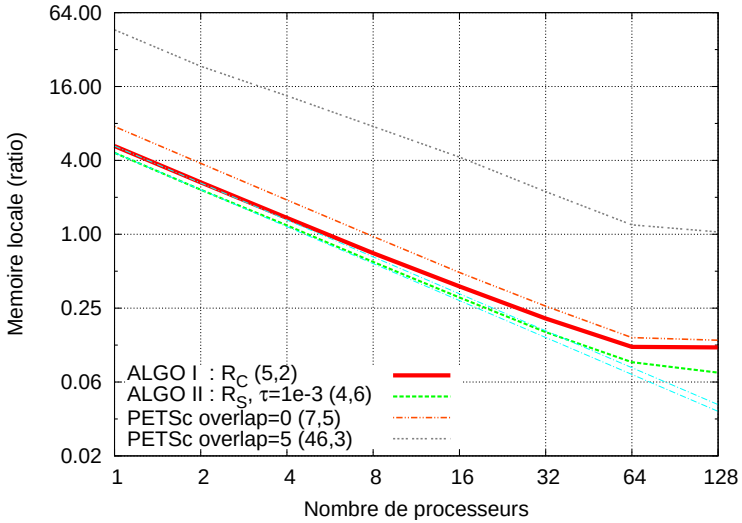


AMANDE (3034 domaines)

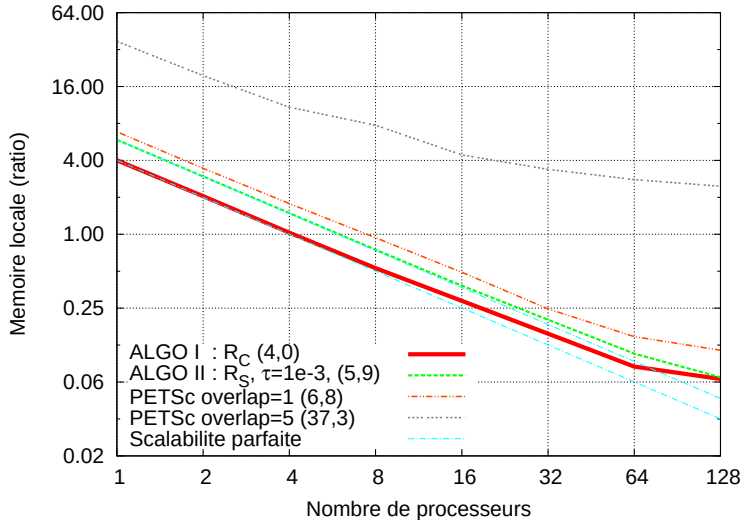


Remplissage local

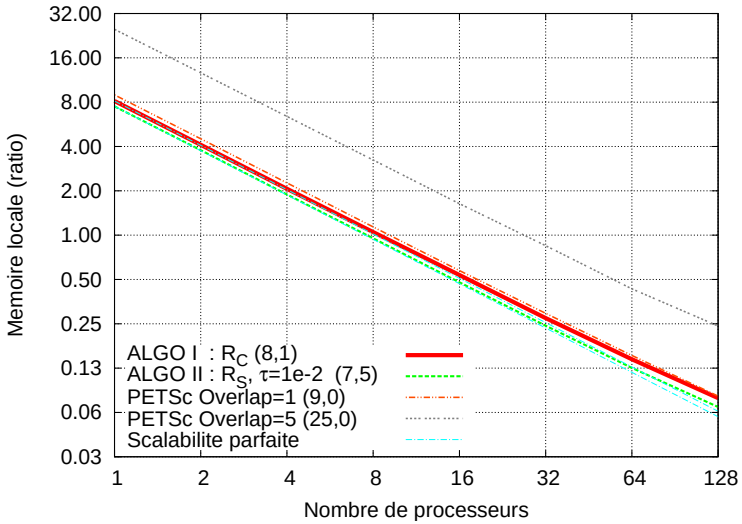
AUDI (181 domaines)



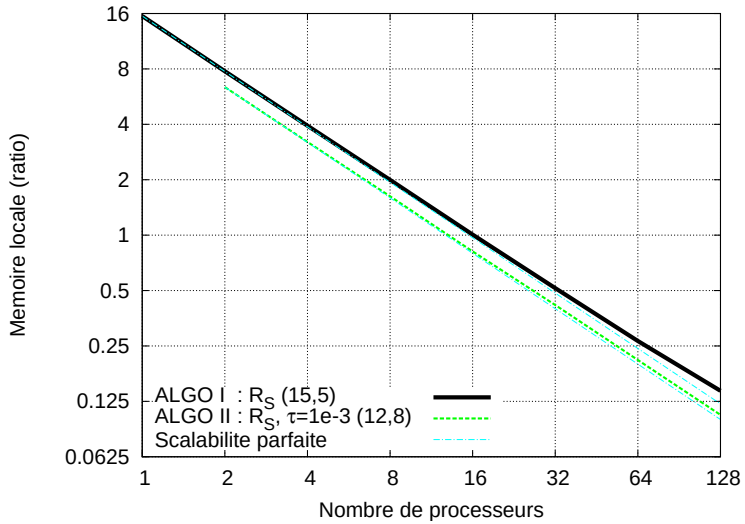
MHD1 (133 domaines)



HALTERE (551 domaines)

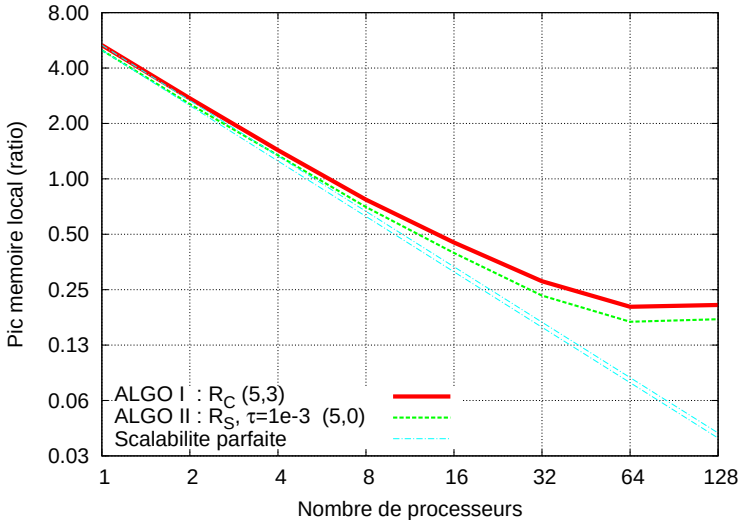


AMANDE (3034 domaines)

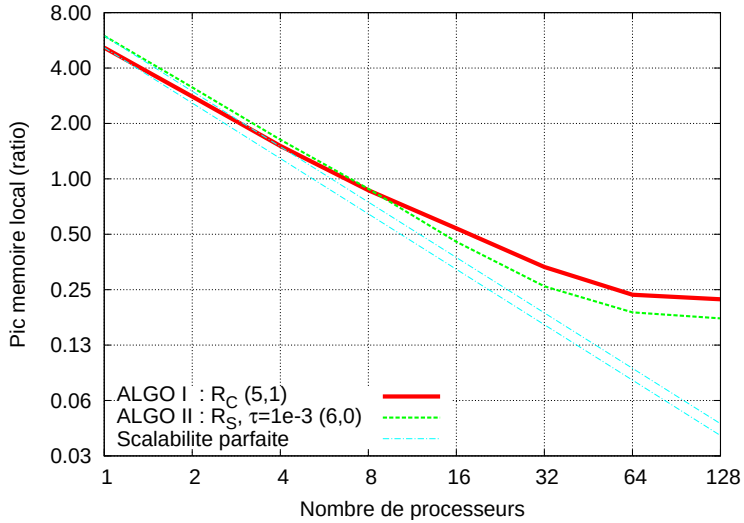


Pic mémoire local

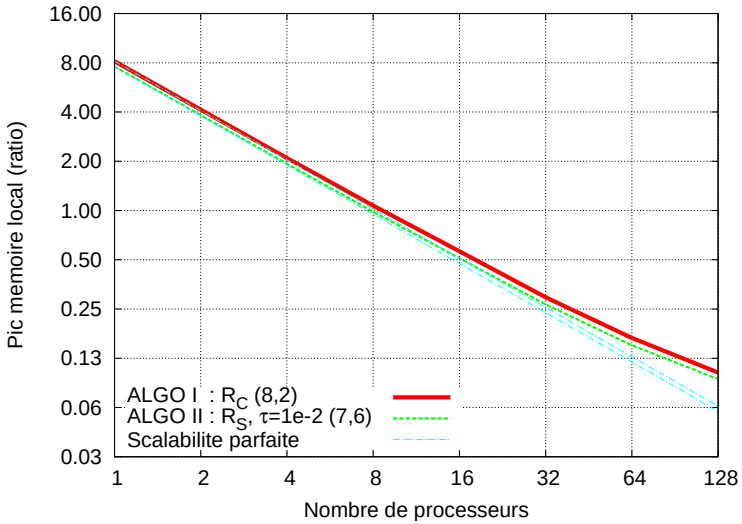
AUDI (181 domaines)



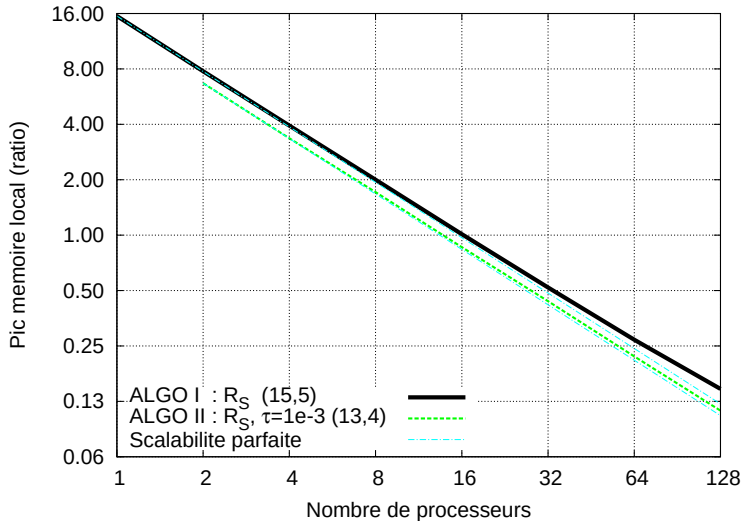
MHD1 (133 domaines)



HALTERE (551 domaines)



AMANDE (3034 domaines)



10Millions : maillage 3D, problème d'électromagnétisme (CEA)

Matrice	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L,U)	OPC
10Millions	10 423 737	89 072 871	$75,49$	$4,4\,.\,10^{13}$

Solveur direct : 277 secondes sur 512 processeurs.

2193 domaines - $\mathscr{R}_S$ - 10^{-7} - nnz(P) $= 23,07$

# proc	Précond. (s)	Résol. (s)	Total (s)	Mémoire	
				Préc.	Résol.
16	125,01	359,91	484,92	24,12	23,94
32	65,06	181,25	246,31	24,75	24,4
64	35,42	94,57	129,99	26,27	25,61
128	19,34	48,43	67,77	27,75	26,76
256	10,2	31,78	41,97	31,13	29,22
512	6,33	17,38	23,71	38,77	35,66
1024	7,08	10,52	17,6	48,08	41,86

Plan

Conclusion (1/3)

Conclusion :

- Approche algébrique générique hybride couplant direct et itératif.
- Préconditionneur flexible, nombreuses stratégies différentes implémentées.
- Couplage fin entre algorithmes supernodaux et algorithmes scalaires.
- Schéma de parallélisation : plusieurs sous-domaines par processeur.
- Etude expérimentale sur de grands cas tests irréguliers.

Conclusion (2/3)

Perspectives :

- Critère automatique pour choisir la taille des sous-domaines.
- Parallélisation de la phase de prétraitement.
- Etude comparative à d'autres méthodes de résolution hybride.

Conclusion (3/3)

http://hips.gforge.inria.fr

- Langage de programmation : C / MPI.
- Fonctionnalités : symétrique, non-symétrique (Cholesky/LU, ICC(t)/ILU(t)), systèmes réels ou complexes.
- Méthodes : Hybride, ICC(t) et ILU(t) multi-niveaux.
- Compatible avec les partitionneurs SCOTCH et METIS.
- Licence Cecill-C (licence de type LGPL).