Preconditioning the Schur complement in a hybrid direct/iterative solver

SparseDays 2010

Jérémie GAIDAMOUR Pascal HÉNON

LaBRI and INRIA Bordeaux - Sud-Ouest, France
This work was supported by the ANR project "Solstice"

June 16th 2010

Outlines

Motivations for hybrid solvers

Goal solve $A.x = b$, A sparse matrix , $n = \dim(A)$

Popular methods :

- Multilevel methods : $O(n)$ operations, $O(n)$ storage (do not depend on h) but not very flexible.
- Preconditioned Krylov methods : number of operations depends on cond(A), storage $O(n)$ (depends on preconditioner) : quite robust and flexible
- Direct methods ($A = L.U$) :
  A from a 2D mesh : $O(n^{\frac{3}{2}})$ operations, storage $O(n.\log(n))$
  A from a 3D mesh : $O(n^2)$ operations, storage $O(n^{\frac{4}{3}})$
  Very robust and flexible (black box)

Motivations for hybrid solvers

Goal solve $A.x = b$, A sparse matrix , $n = \dim(A)$

- Improvement of one of these methods is not easy !
- Mixing two of these methods allows to find a trade-off between robustness/flexibility and memory/CPU-time.

Hybrid Direct-iterative solver

We want to build an hybrid direct-iterative method by using the Schur complement in a domain decomposition framework

Schur complement approach (1/2)

Reminder of the Schur complement approach :

The linear system $A.x = b$ can be written as :

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

The system $A.x = B$ can be solved in three steps :

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

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

Schur complement approach (2/2)

Using the Schur complement approach to mix direct / iterative methods :

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

$A_B = L_B.U_B$: (1) and (3) solved by a direct resolution.

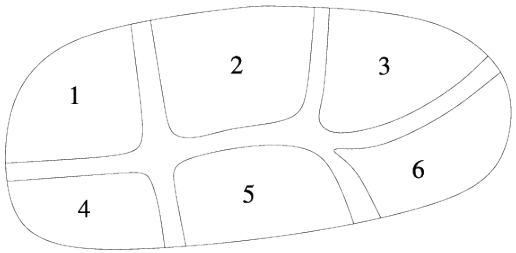
$S \simeq \tilde{L_S}.\tilde{U_S}$: (2) solved by a preconditioned Krylov subspace method.

We iterate only on the Schur complement : we need $S.x$ and $L_S.U_S \approx S$.

Domain decomposition framework

Based on a domain decomposition : interface one node-wide (no overlap in DD lingo)

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



B : Interior nodes of subdomains (direct factorization).

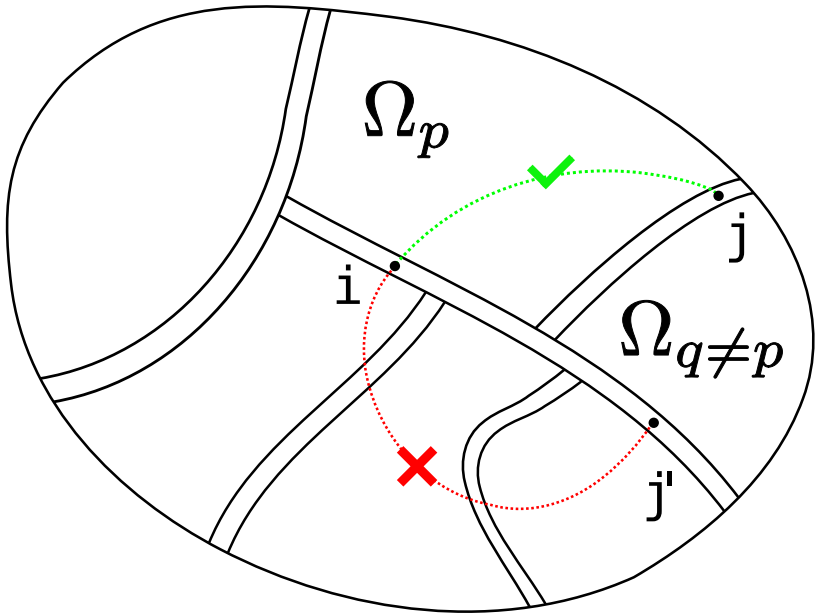
C : Interface nodes.

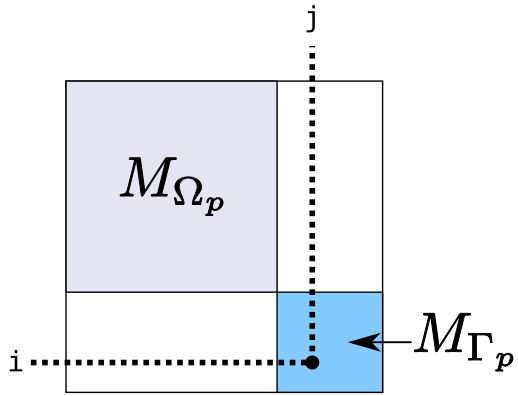
Special decomposition and ordering of the subset C :

Goal : Building a global Schur complement preconditioner (ILU) from the local domain matrices only.

ILU compatible with a domain decomposition

ILU follows a global ordering but fill-in is only allowed in local domain matrices





Incomplete factorization compatible with DD

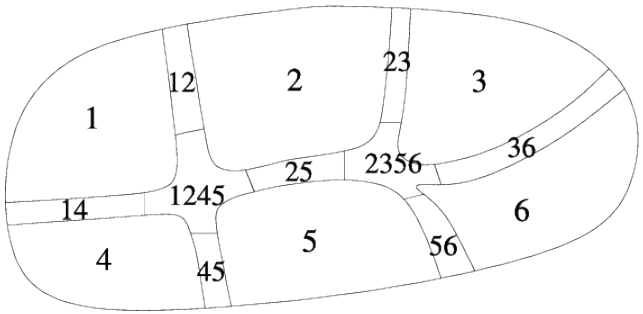
We want an incomplete factorization of the matrix without creating edge (fill-in) outside the local domain matrices (keep the parallelism).
Problem : in a domain, we need an ordering of the interface compatible with neighboring domains.

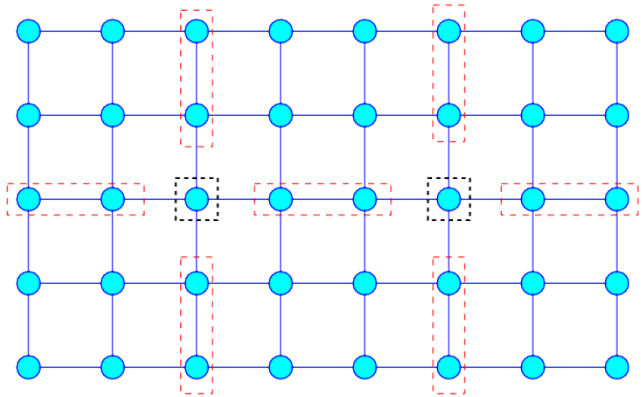
- Use a hierarchical interface decomposition [Hénon, Saad, 06] :
  - partition the interface nodes according to the domains they belong to.
  - respect some properties to ensure good parallelism and numerical behavior.

Hierarchical Interface Decomposition ordering

Special decomposition and ordering of the subset C :

Hierachical interface decomposition into connectors :





Rules :

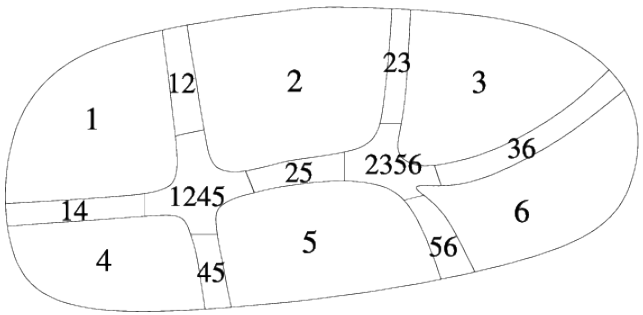
- No creation of edge (fill-in) outside the local domain matrices.
- Allow edges between connectors adjacent to the same subdomain.

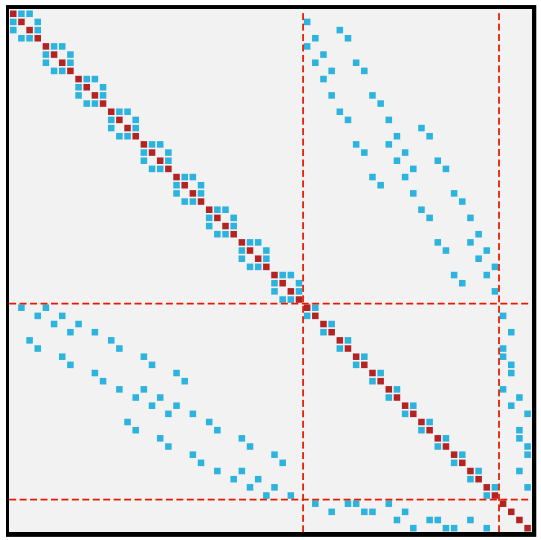
$\Rightarrow$ keep the parallelism (communication only between adjacent subdomains).

Hierarchical Interface Decomposition ordering

Special decomposition and ordering of the subset C :

Hierachical interface decomposition into connectors :





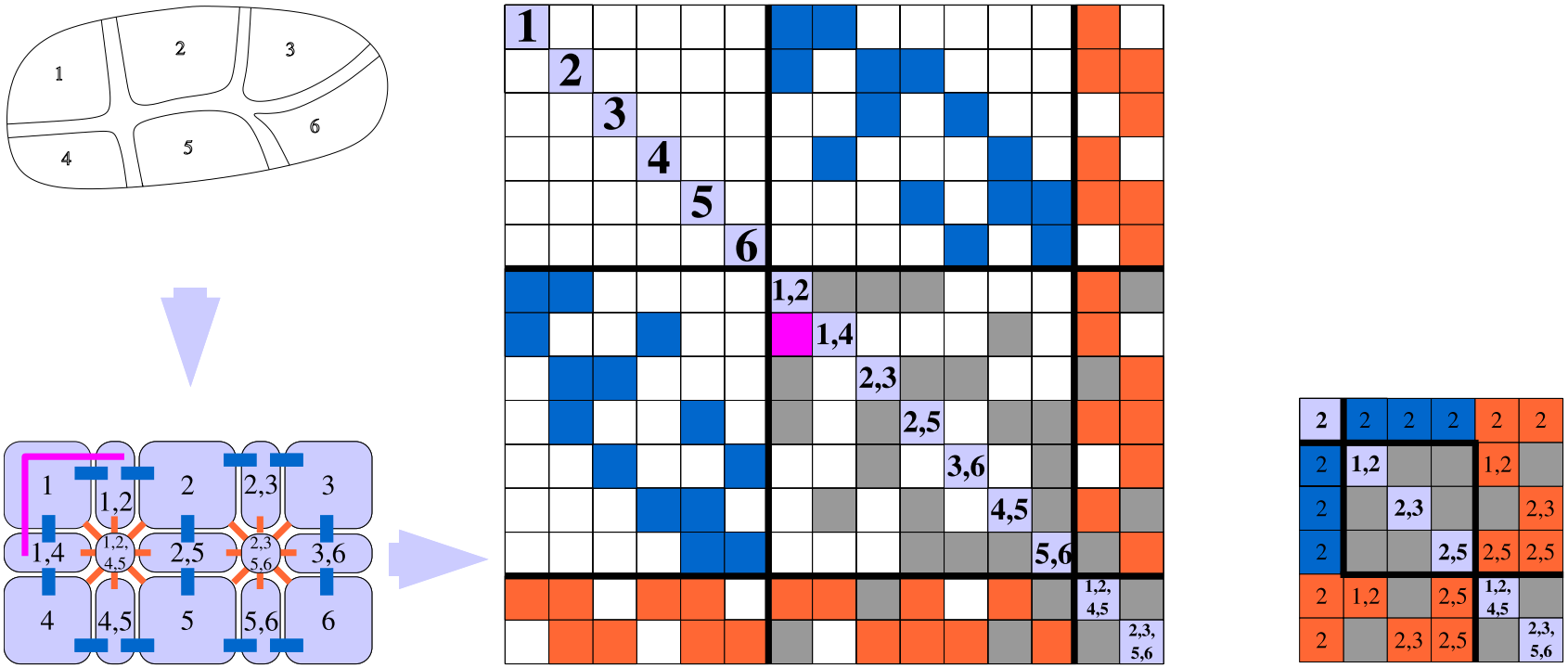
Rules :

- No creation of edge (fill-in) outside the local domain matrices.
- Allow edges between connectors adjacent to the same subdomain.

$\Rightarrow$ keep the parallelism (communication only between adjacent subdomains).

Fill block pattern (1/2)

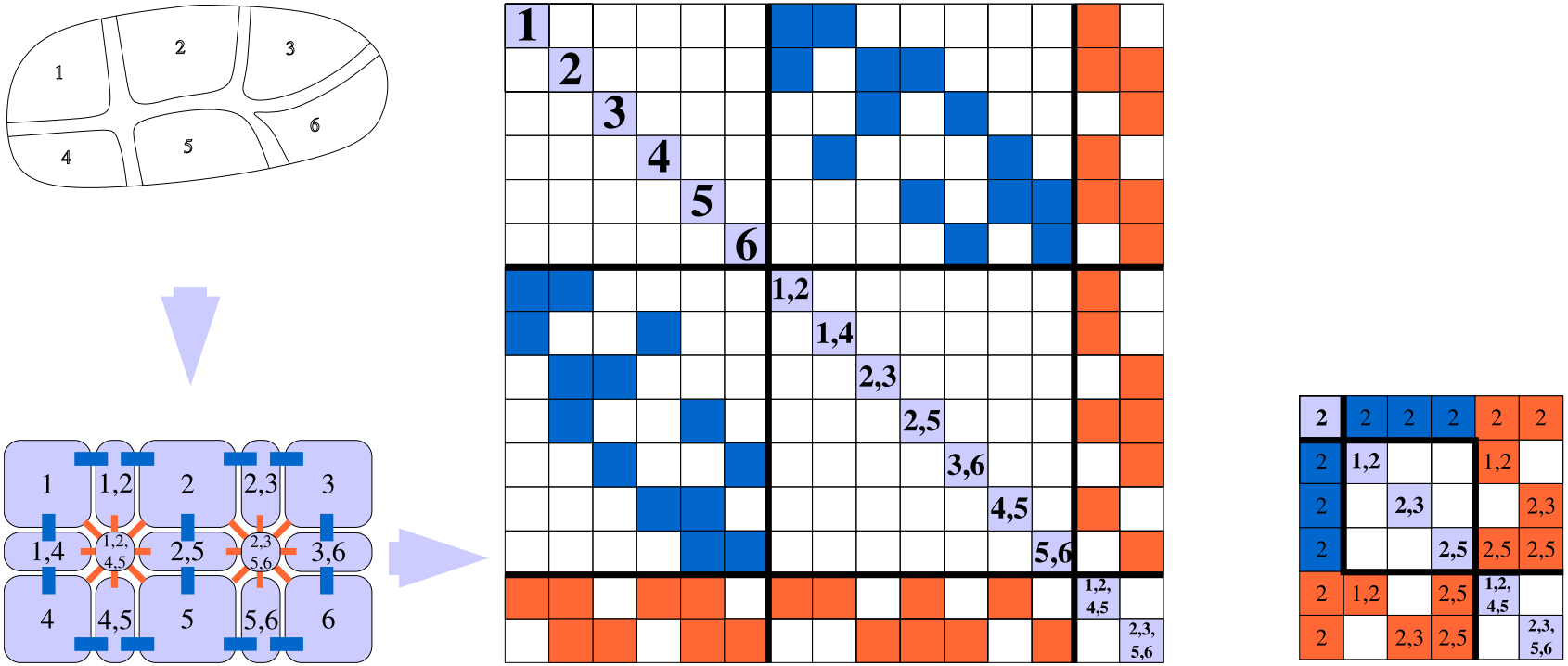
Block ILU factorization :



$\mathscr{R}_S$: Fill is allowed anywhere in the local domain matrices

Fill block pattern (2/2)

Block ILU factorization :

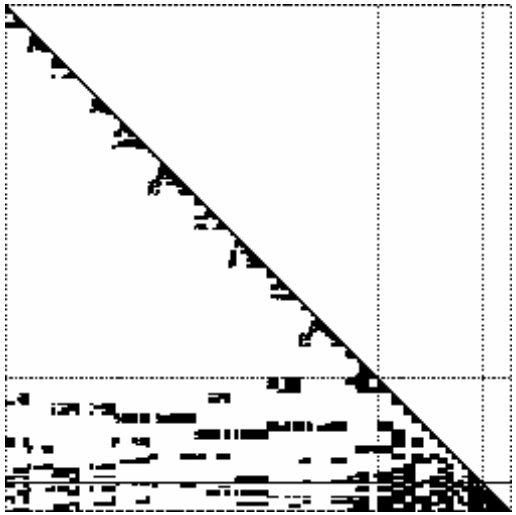


$\mathscr{R}_C$: Fill is not allowed outside the diagonal block structure of the initial matrix C (no edge between connectors of a same level).

Memory ?

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

Symbolic factorization of the matrix BCSSTK14 :



$\Rightarrow$ the most part of the fill-in appear on EU_B^{-1}, $L_B^{-1}F$ and S (3D)

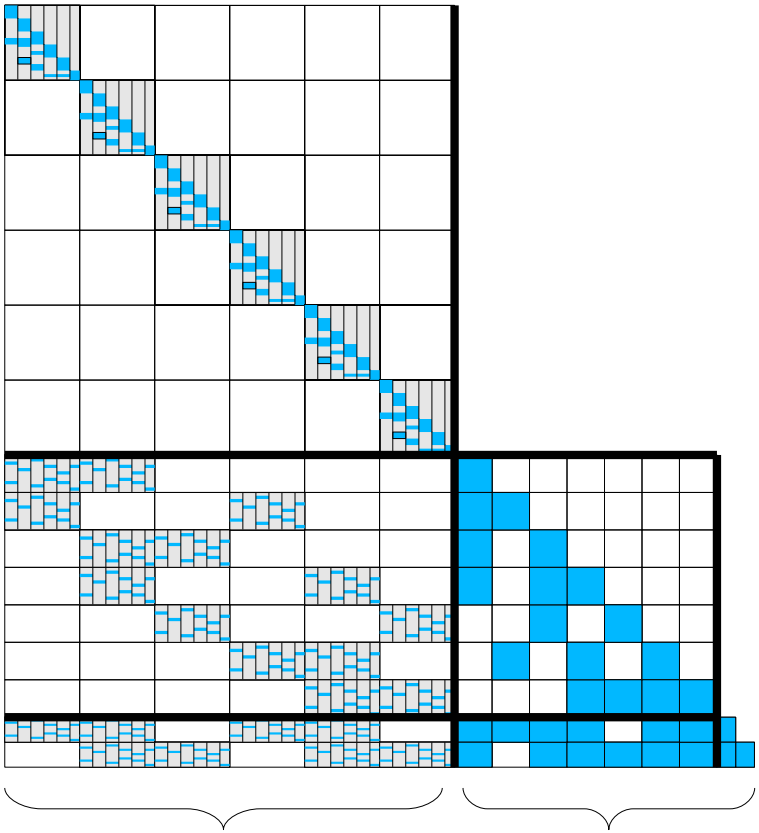
Problematic

Question : How to reduce the storage cost of EU_B^{-1}, $L_B^{-1}F$ and S ?

2 important remarks :

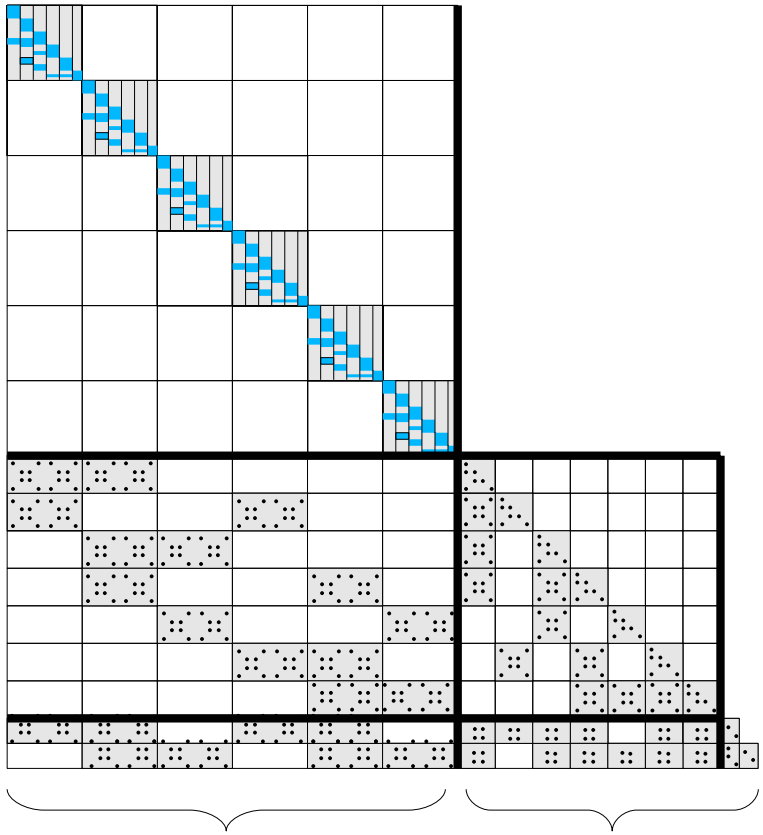
- The iterative resolution only needs the computation of $S.x$ (the Schur product). It can be computed using $(A_C - EU_B^{-1}.L_B^{-1}F).x$.
- EU_B^{-1}, $L_B^{-1}F$ and S are only temporary matrices to compute $\tilde{L_S}.\tilde{U_S}$.

Fill pattern inside the block pattern $\mathscr{R}_S$



Factorisation directe ILU par blocs

R_S



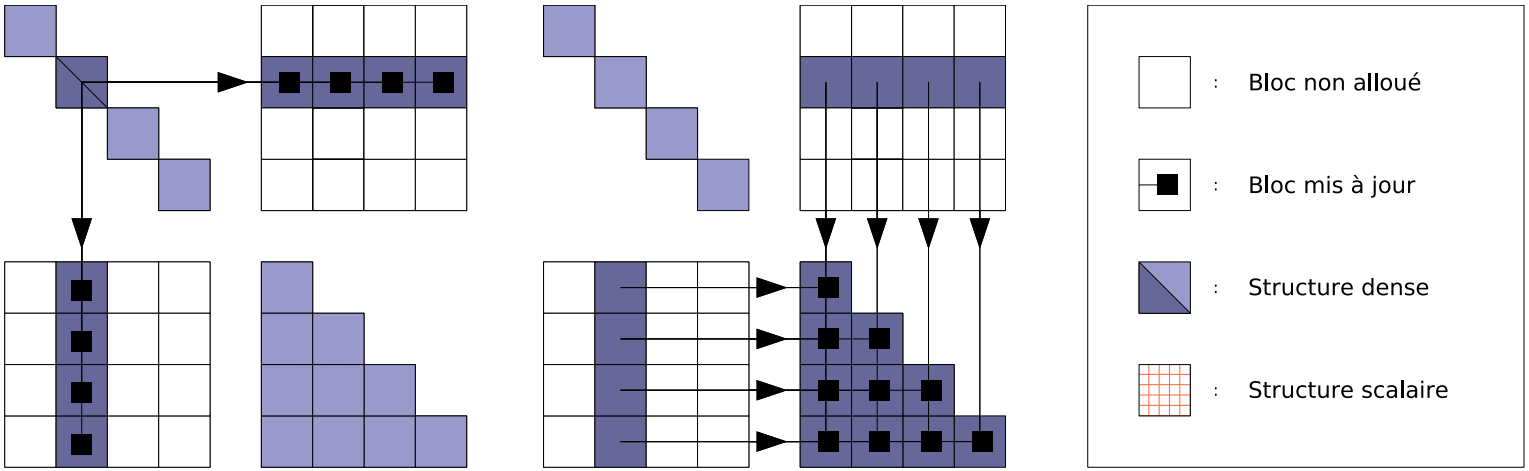
Factorisation directe ILUT scalaire

$R_S + \tau$

Algorithm I

Computation of $W = EU_B^{-1}$, $G = L_B^{-1}F$ and the exact Schur S (supernodal algorithm) :

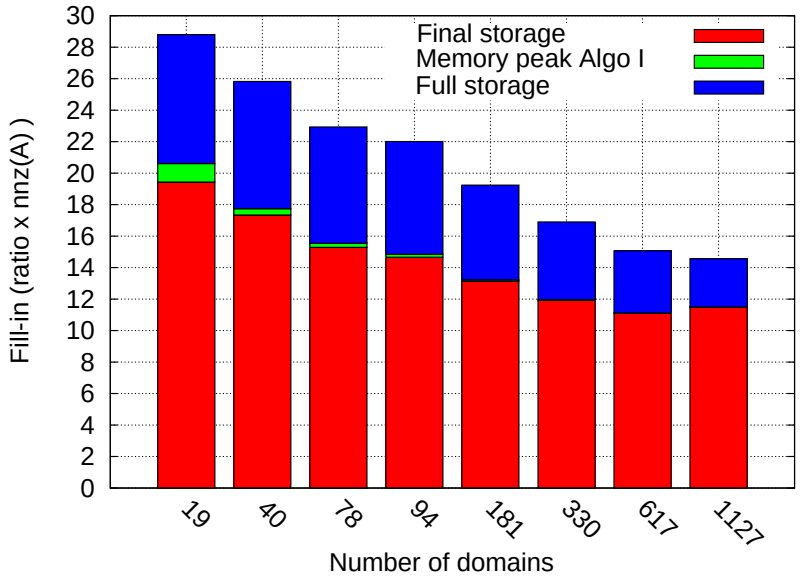
1. Preallocation of S (symbolic factorization).
2. Foreach subdomains, do :
   - Supernodal computation of the block-column of W (block-row of G) corresponding to the subdomains.
   - Compute the contribution of this block-column/row to S.
   - Free the block-column/row of W/G



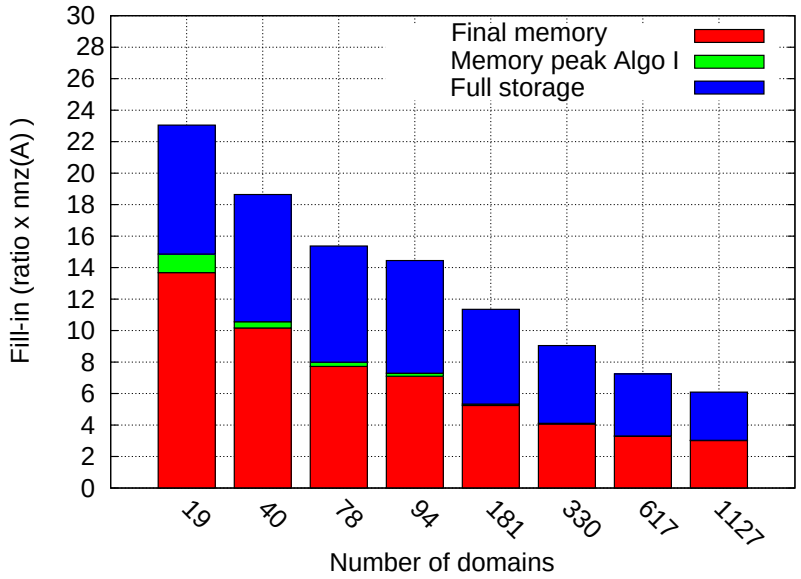
Right looking supernodal algorithm to compute S

Algorithm I : Results

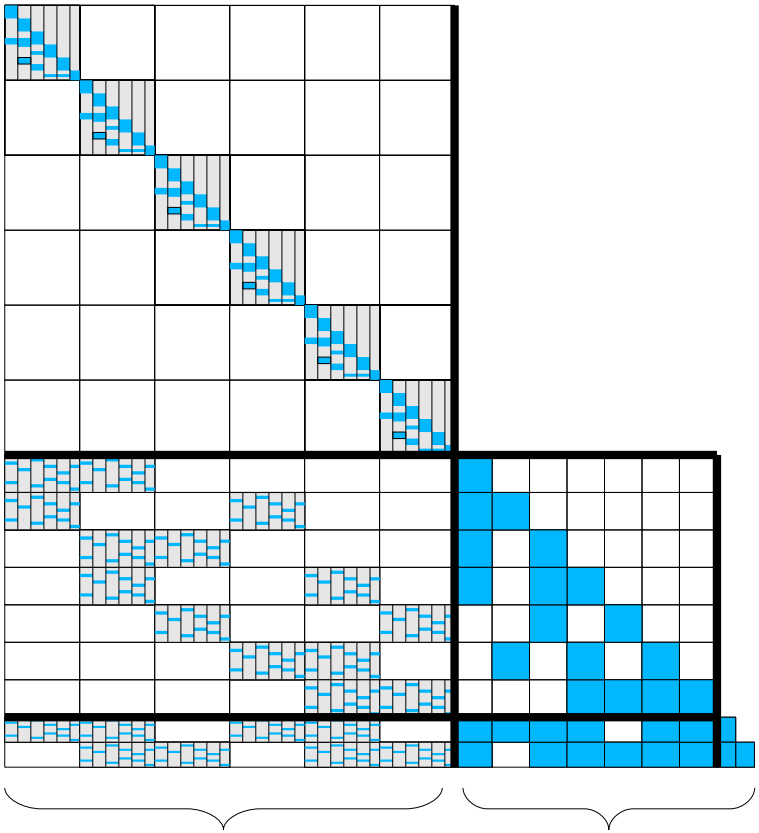
AUDI : $\mathscr{R}_S$ fill pattern



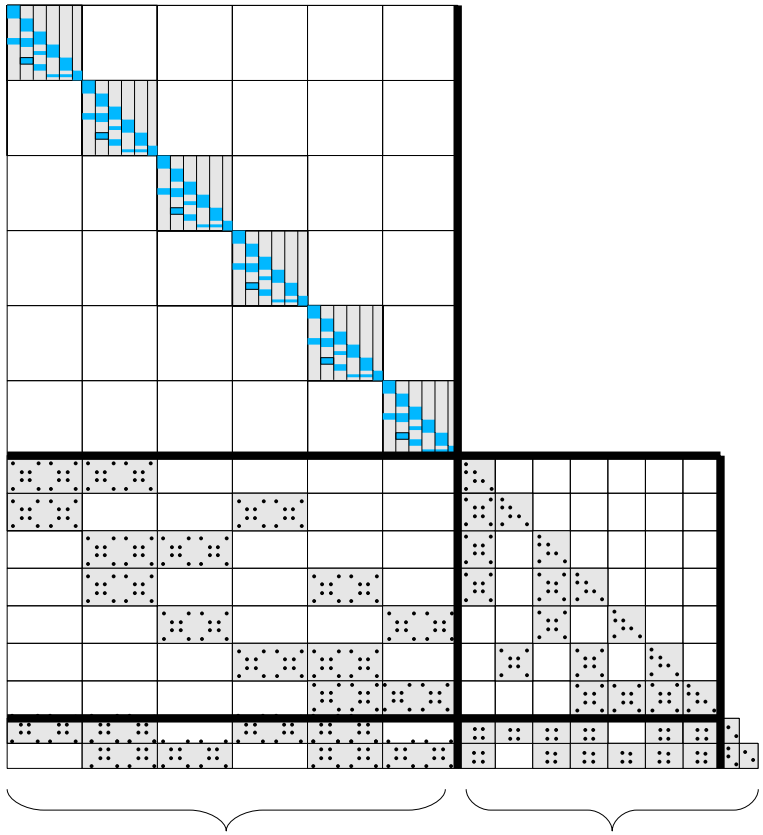
AUDI : $\mathscr{R}_C$ fill pattern



Fill pattern inside the block pattern $\mathscr{R}_S$



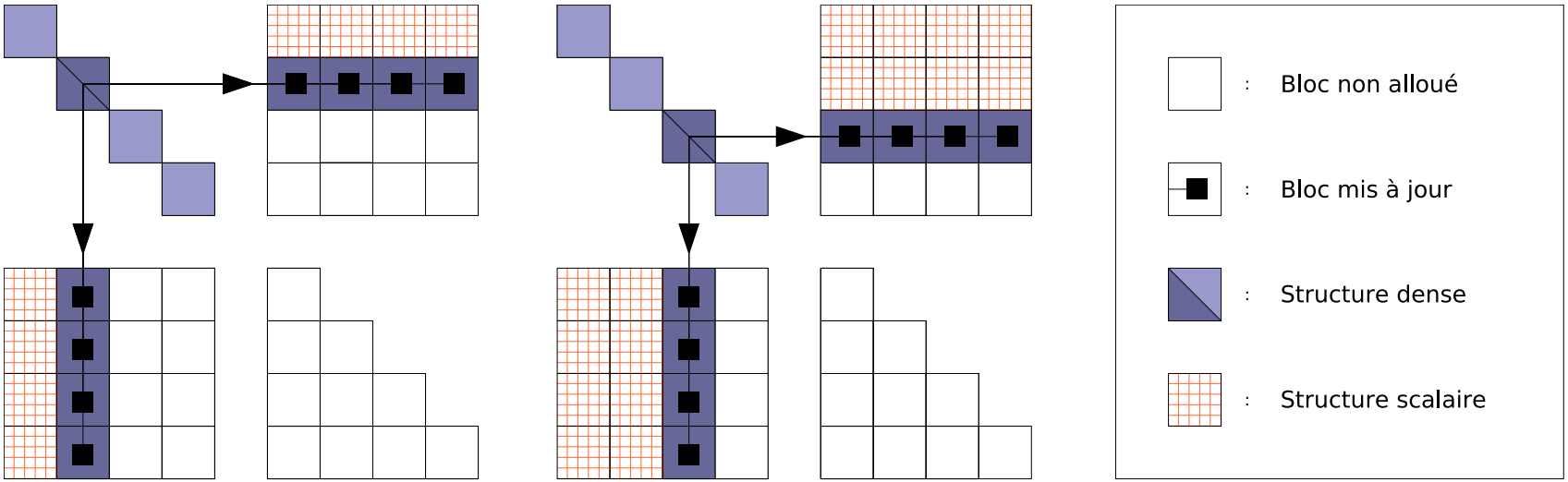
R_S



$R_S + \tau$

Algorithm II

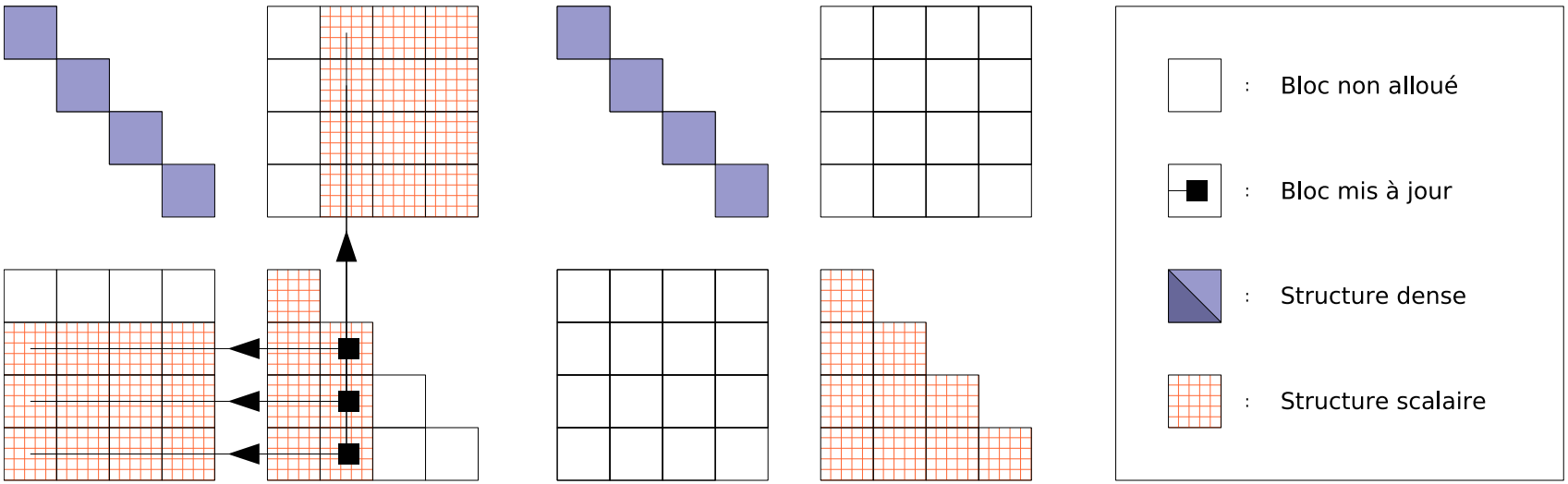
- Approximate computation of $W = EU_B^{-1}$, $G = L_B^{-1}F$: Matrices are sparsified during their computation according to a numerical threshold τ_W.



Algorithm II

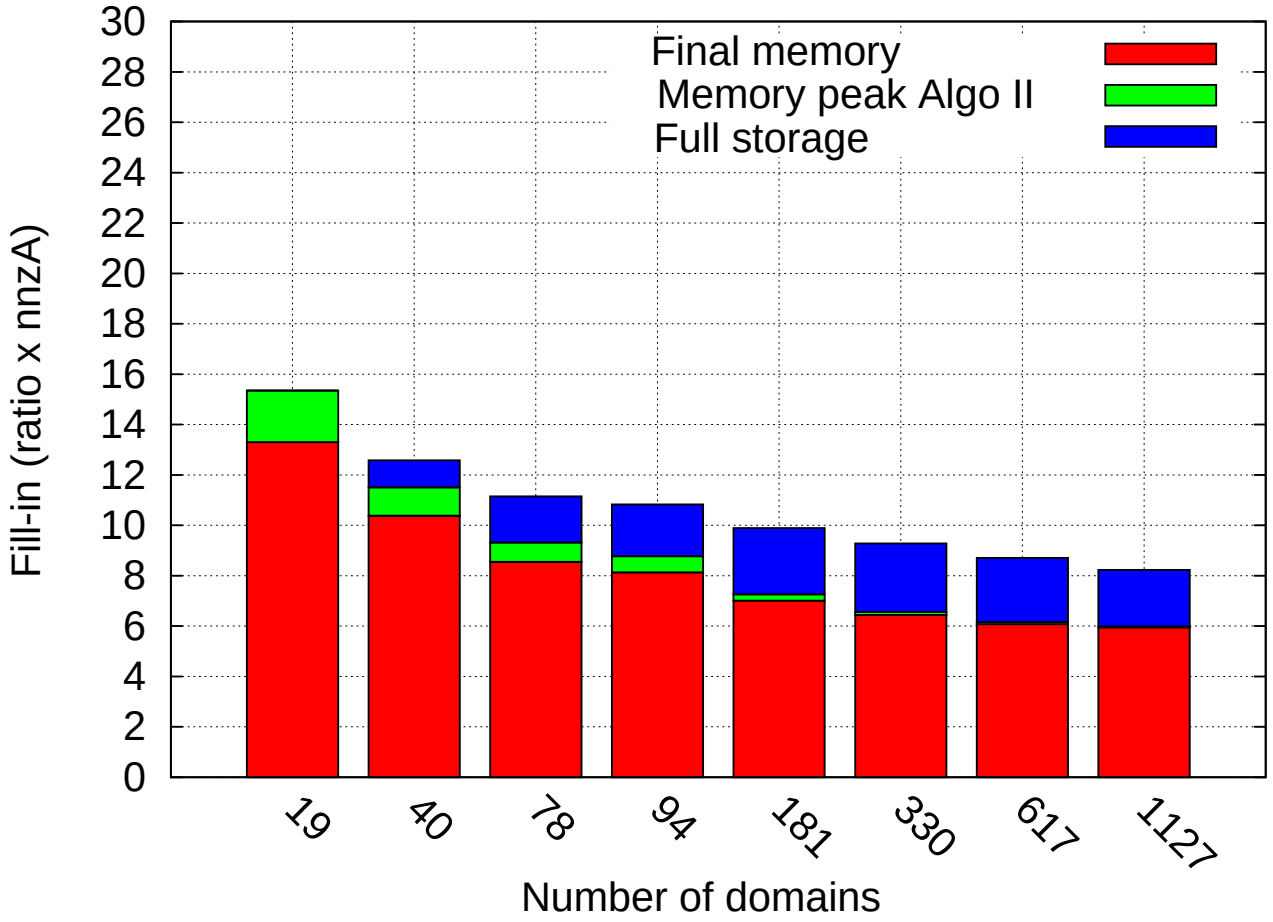
- Left-Looking incomplete factorization of $\tilde{S}$:
  Column algorithm according to the supernodal partition of A_B. The computation of $\tilde{S}$ is interlaced with the ILU(τ_S) factorization.

  Foreach subdomains, do :
  - Compute a column of $\tilde{S}$ from $\tilde{W}$ and $\tilde{G}$,
  - Factorize immediatly this column to limit the fill-in : ILU(τ_S).



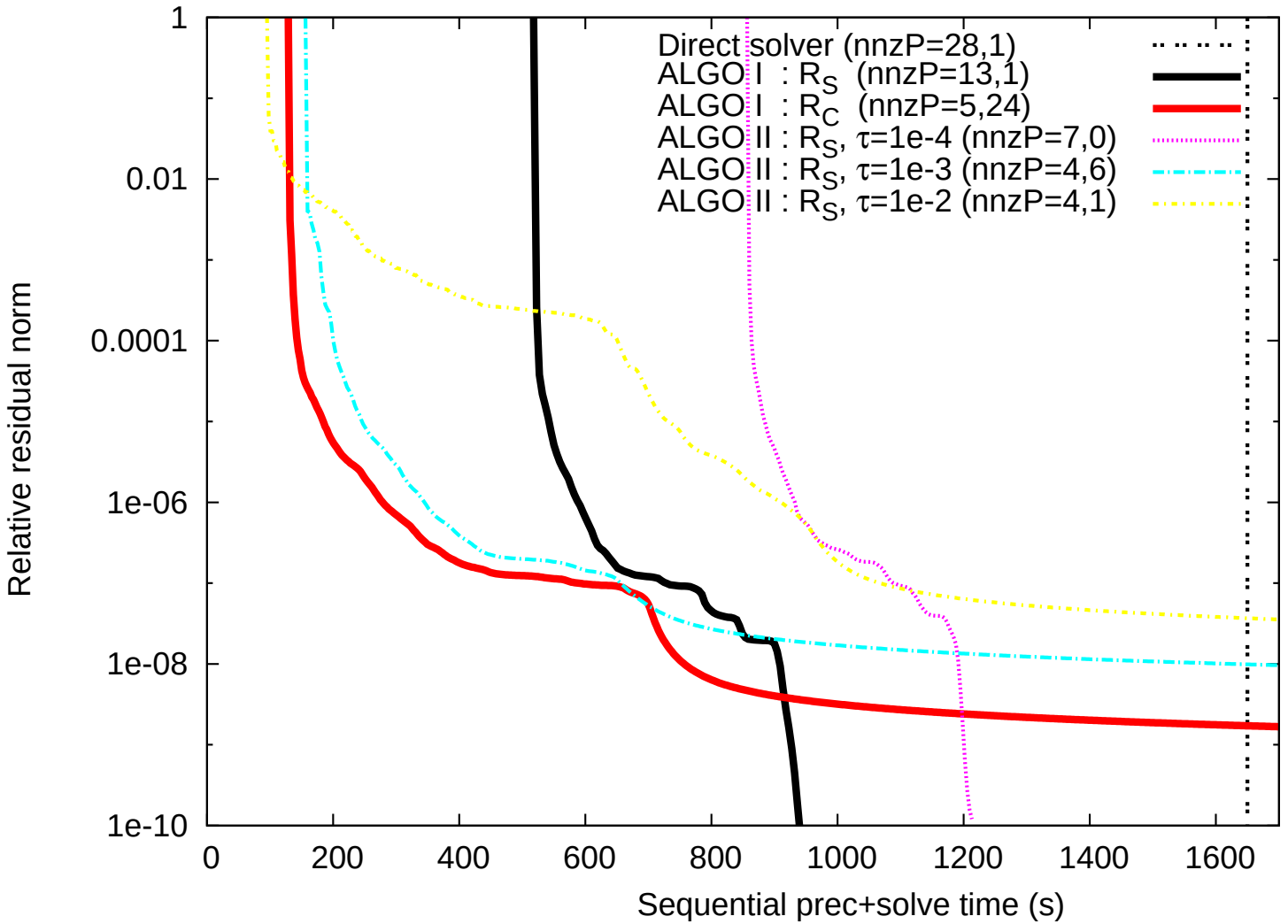
Algorithme II : Results

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



Convergence history (time)

AUDI (943695) : 180 domains (average domain size : 2000 noeuds)



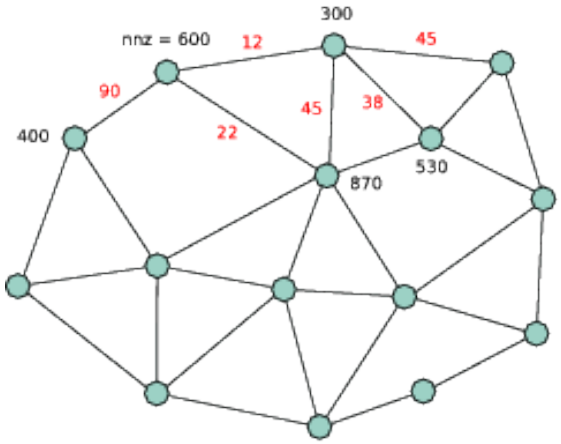
Overview on the parallelization

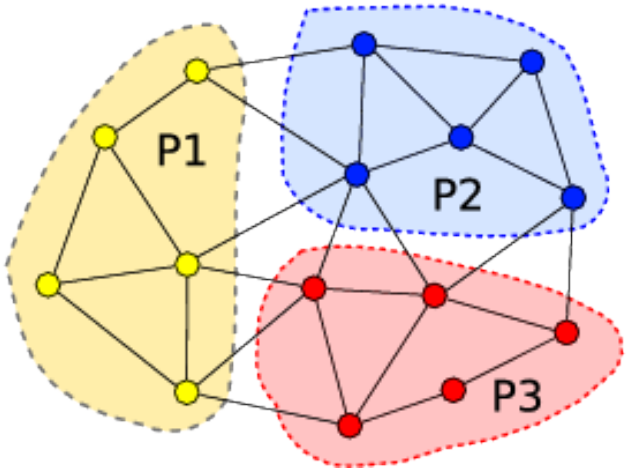
We use a multiple domains per processors parallelization scheme.

→ We choose a memory / convergence speed tradeoff independently from the number of processors.

Subdomains distribution on the processors :

- Obtained by partitioning the weighted "domain graph".
- Balance of $S.x$ computation (solving step) by using the symbolic factorization to compute the number of NNZ of the interiors of subdomains.





Results

Test cases :

- AUDI : structural mechanism (3D, PARASOL).
- MHD1 : magneto-hydrodynamic (3D, Univ. of Québec).
- HALTERE and AMANDE : electromagnetism (3D, CEA).

Matrices		Number of unknwons	Number of entries	Direct factorization	
				nnz(L,U)	OPC
AUDI	($\mathbb{R}$ sym.)	943695	39297771	$41,2$	$5,4\,.\,10^{12}$
MHD1	($\mathbb{R}$ non sym.)	485597	24233141	$52,4$	$9,0\,.\,10^{12}$
HALTERE	($\mathbb{C}$ sym.)	1288825	10476775	$38,7$	$7,5\,.\,10^{12}$
AMANDE	($\mathbb{C}$ sym.)	6994683	58477383	$53,9$	$1,5\,.\,10^{13}$

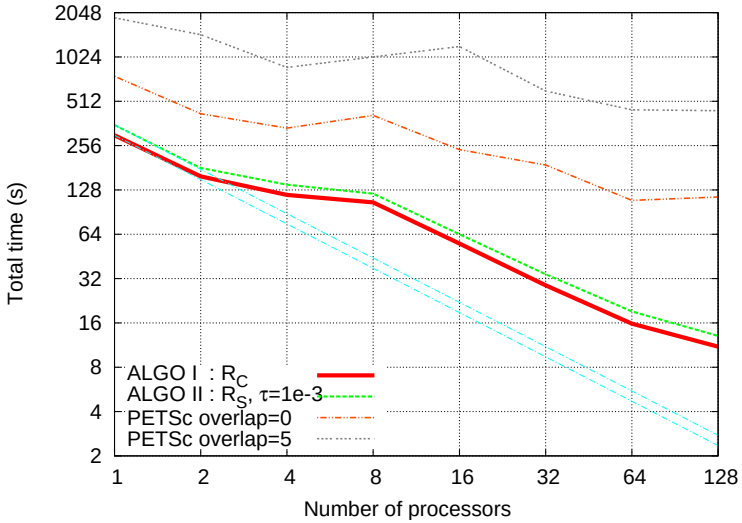
Experimental environment :

Supercomputer "Jade" from GENCI-CINES :

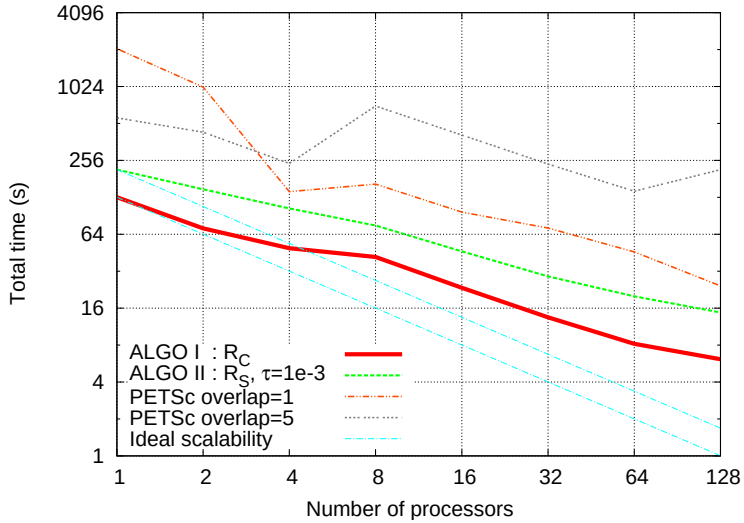
- Nodes : 2 Intel Xeon Quad-Core 3,0 GHz.
- 32 Go of memory per node.
- Infiniband network.

Time scalability (relative residual norm $= 10^{-7}$)

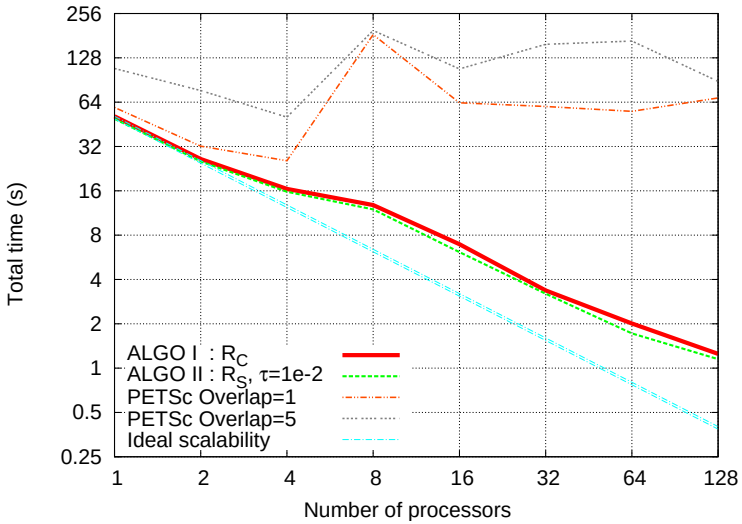
AUDI (181 domains)



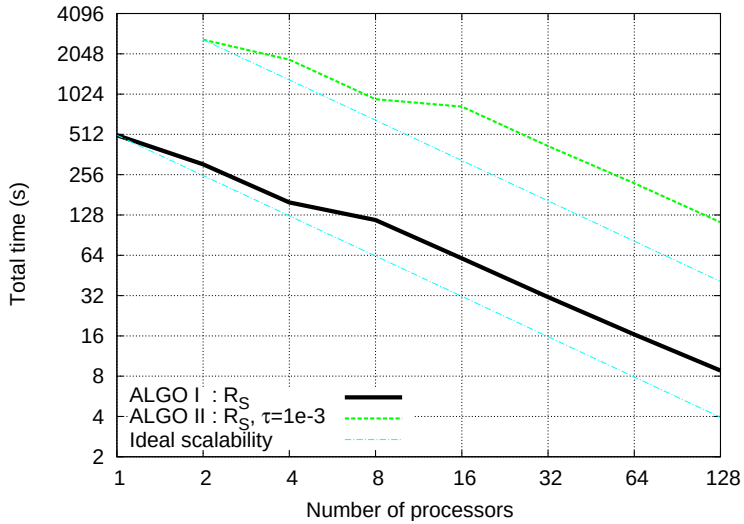
MHD1 (133 domains)



HALTERE (551 domains)



AMANDE (3034 domains)



10Millions : 3D mesh, electromagnetism problem (CEA)

Matrix	Number of unknowns	Number of entries	Direct factorization	
			nnz(L,U)	OPC
10Millions	10423737	89072871	75, 49	$4,4\,.\,10^{13}$

Direct solver : 277 seconds on 512 processors.

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

# proc	Precond. (s)	Sol. (s)	Total (s)	Memory	
				Prec.	Solv.
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

Conclusion

We have several algorithms to compute a Schur complement preconditioner :

- The scalability behaviour is roughly the same for the different version
- But different performance/memory trade-offs are obtained

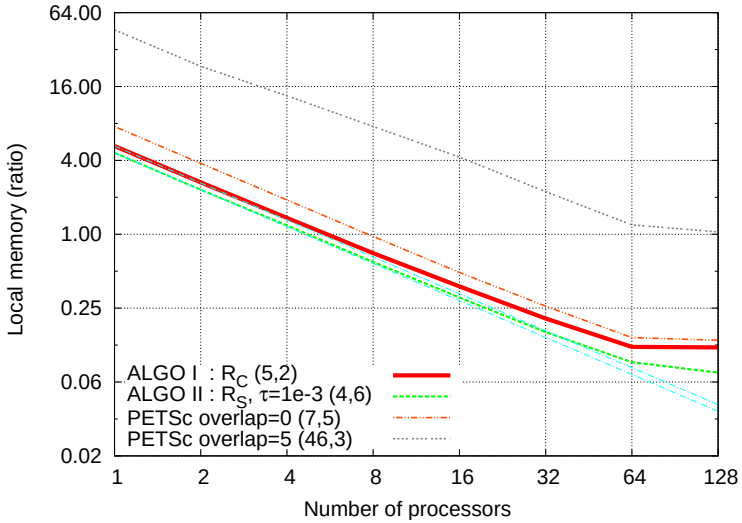
Prospects :

- Work on the ordering (with François Pellegrini) : better interface can be obtained.
- Parallelization of the graph ordering and partitioning step (HID algorithm and use of PT-Scotch).
- Propose some automatic tunings (reduce number of parameters)

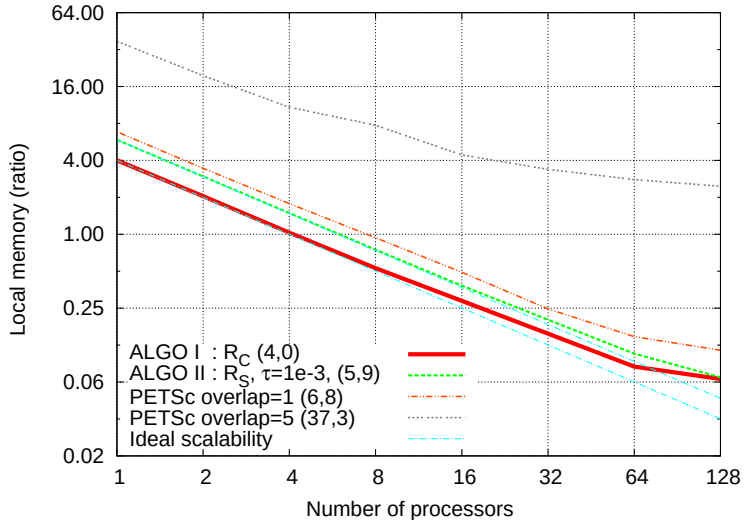
http://hips.gforge.inria.fr

Local fill-in

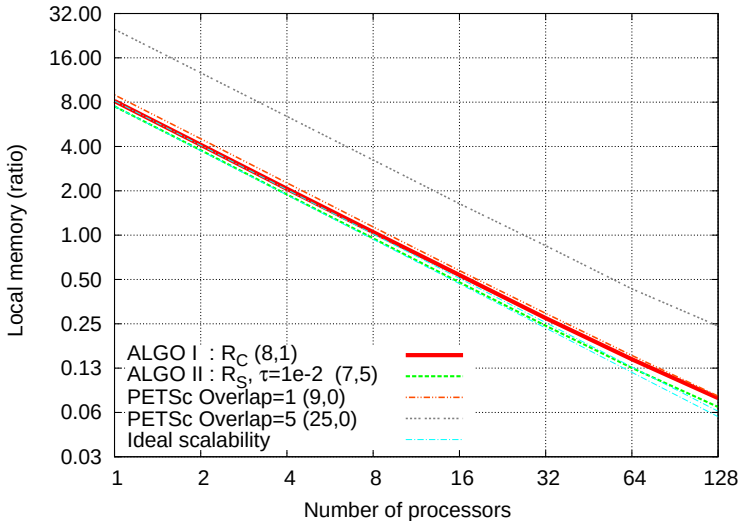
AUDI (181 domains)



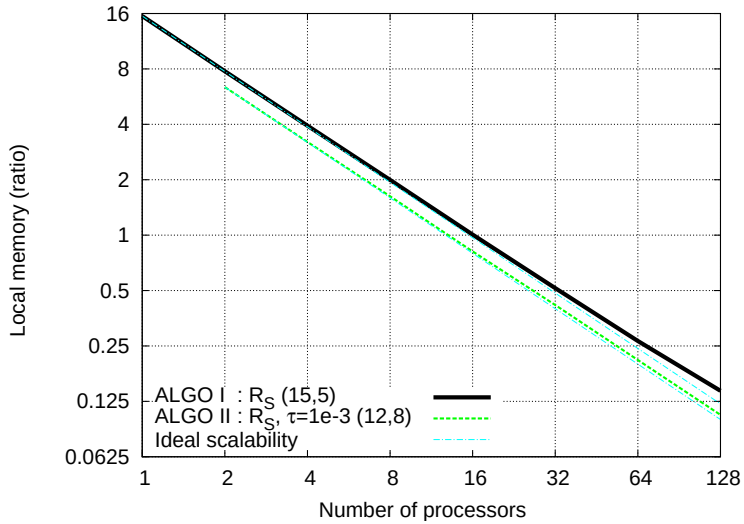
MHD1 (133 domains)



HALTERE (551 domains)

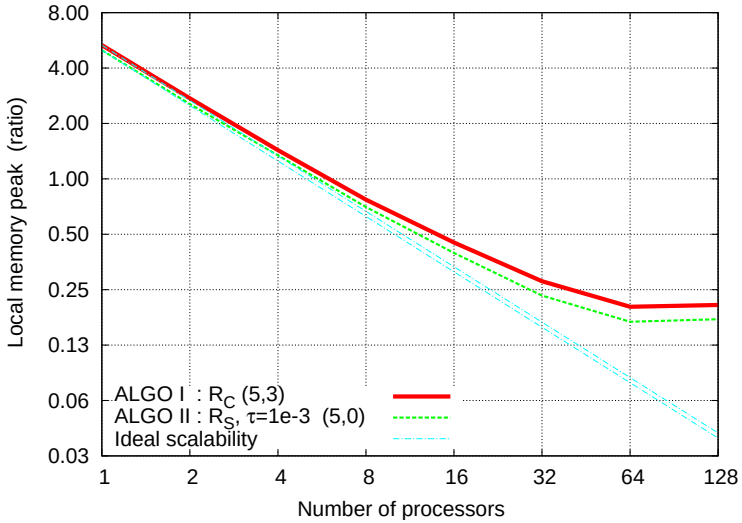


AMANDE (3034 domains)

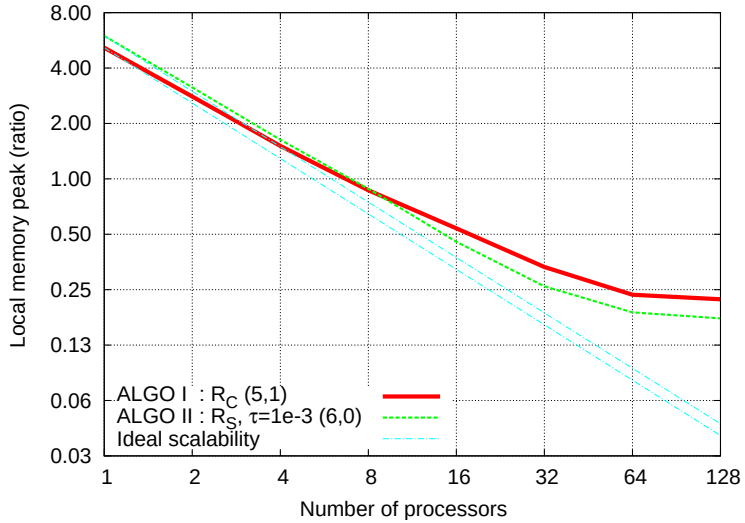


Local memory peak

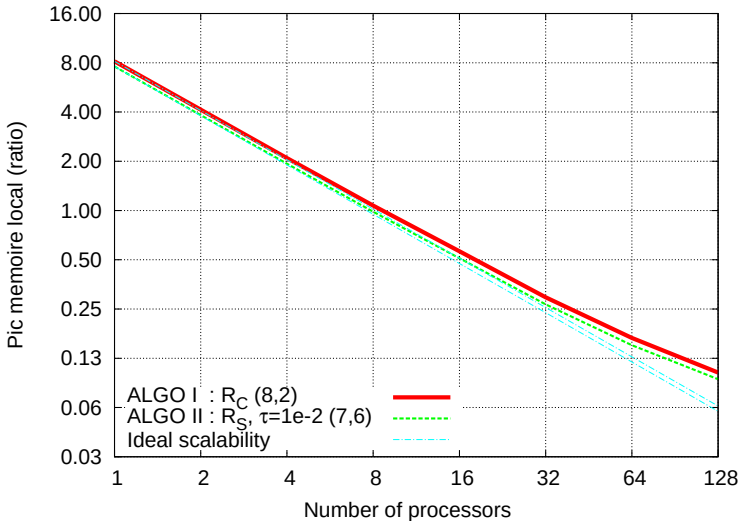
AUDI (181 domains)



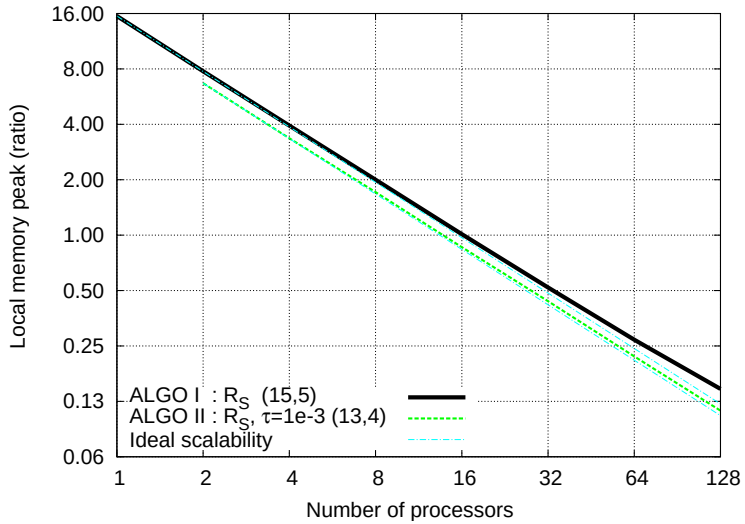
MHD1 (133 domains)



HALTERE (551 domains)

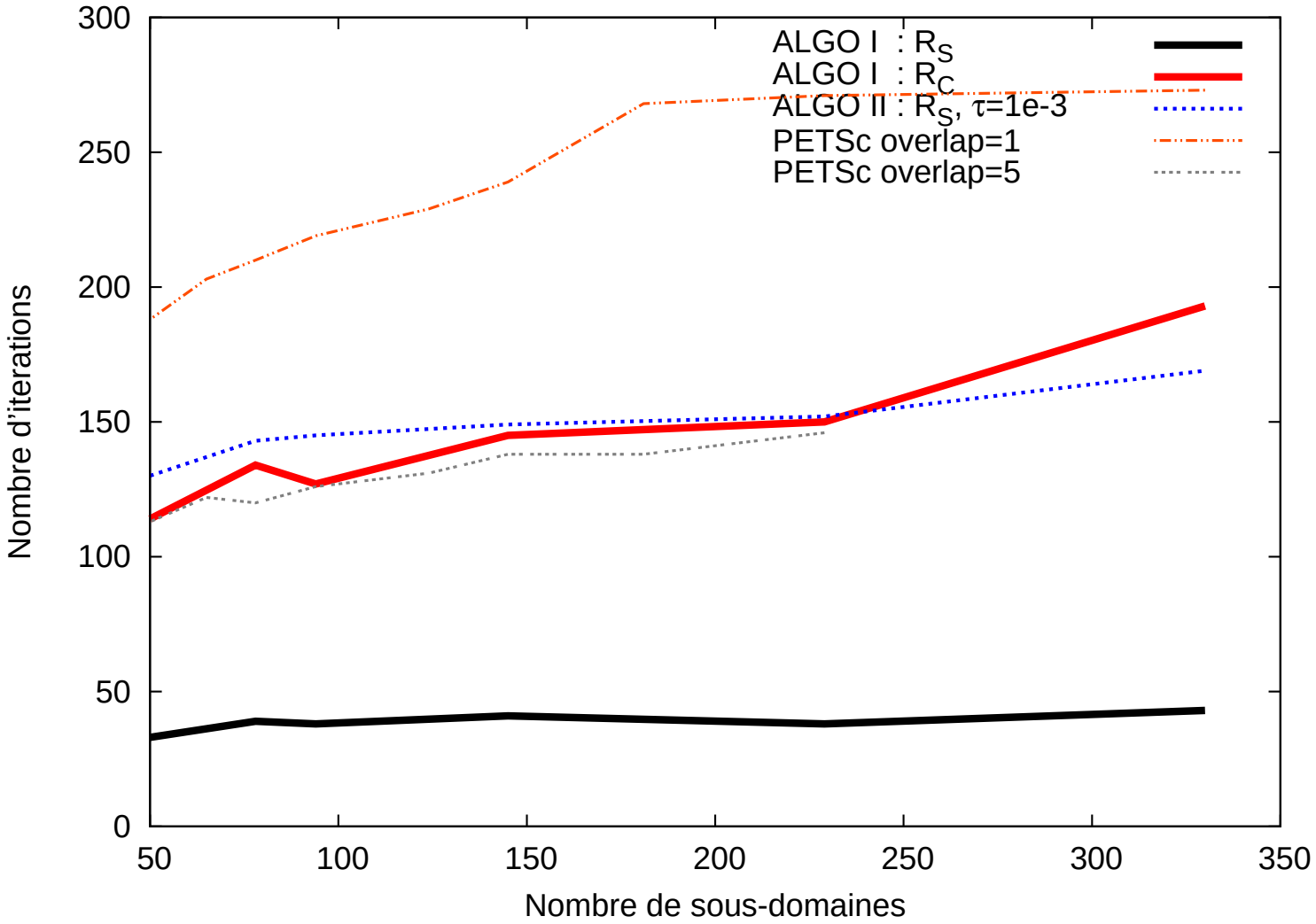


AMANDE (3034 domains)



Iterations/nb. of domains (accuracy $= 10^{-7}$)

AUDI (943695) :



Time/nb. of domains (accuracy = 10^{-7})

AUDI (943695) :