Improving multifrontal methods by means of low-rank approximation techniques

Joint work with Patrick Amestoy , Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari and Jean-Yves L'Excellent, PhD started on October 1st, 2010 and financed by EDF.

Clément Weisbecker, ENSEEIHT-IRIT, University of Toulouse, France

SIAM LA12, Valencia, Spain, June 22, 2012

Introduction

Improving **MULTIFRONTAL METHODS** by means of
LOW-RANK APPROXIMATION TECHNIQUES

Multifrontal solver

- direct solver for large linear systems
- well known and studied

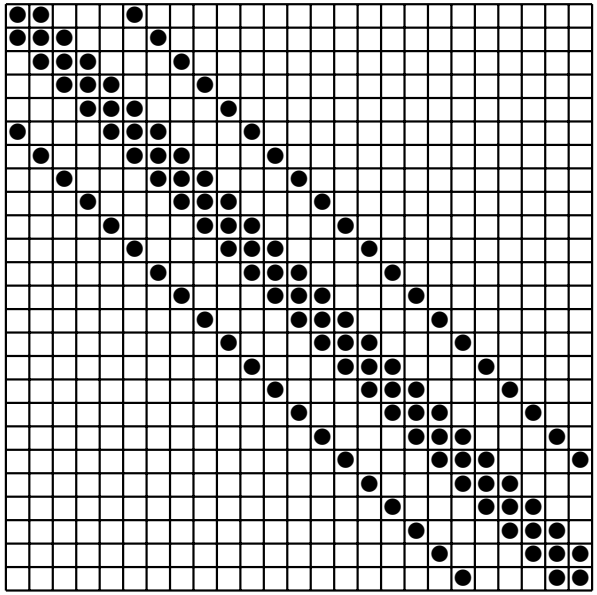
Low-rank approximations

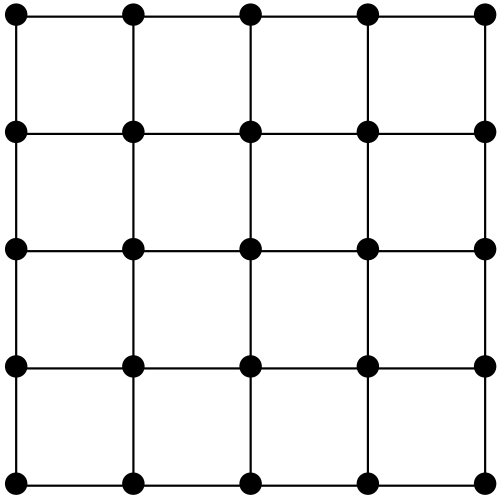
- compression and flop reduction
- accuracy controlled by a numerical parameter

⇒ Combine these two notions to improve multifrontal solvers (in the context of `MUMPS`)

The multifrontal method

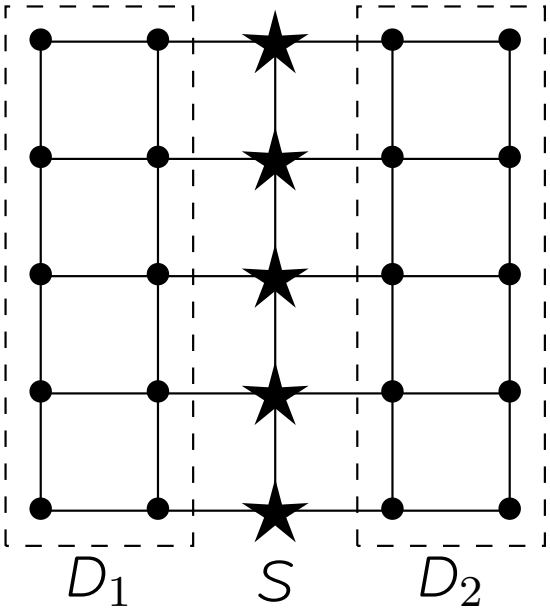
From matrices to graphs





- 1 variable of the matrix A = 1 node in the graph $\mathcal{G}$
- $A(i,j) \neq 0 \Leftrightarrow$ an edge between nodes i and j exists

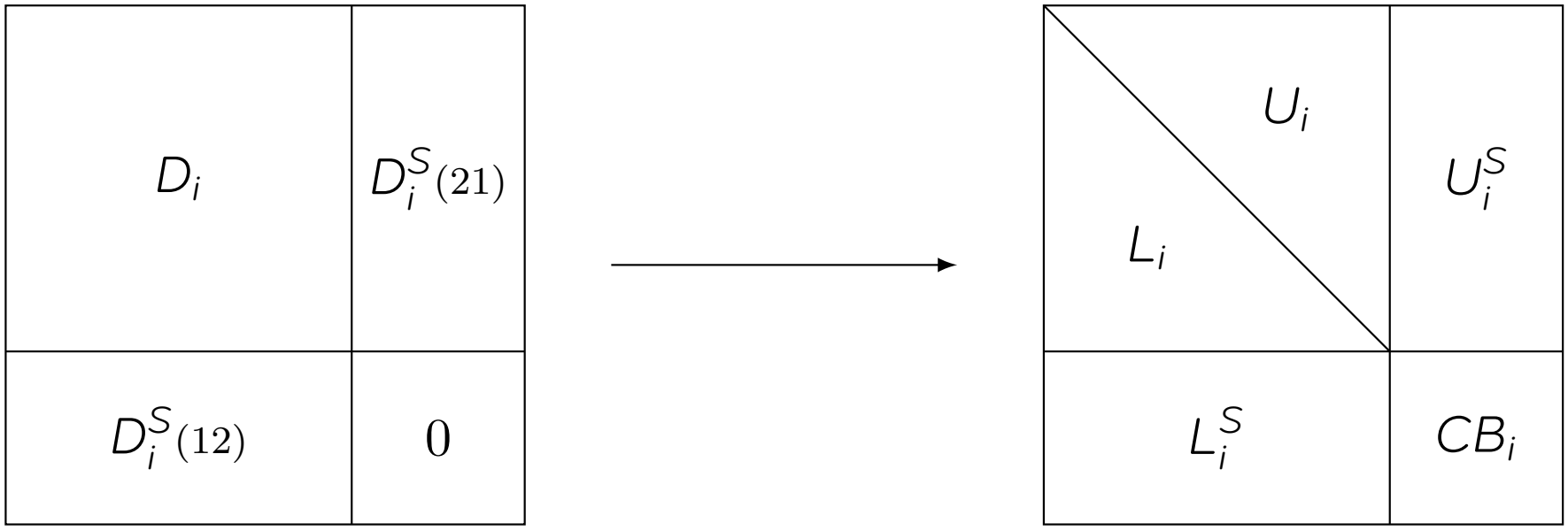
Nested dissection



D_1		
0	D_2	
D_1^S	D_2^S	S

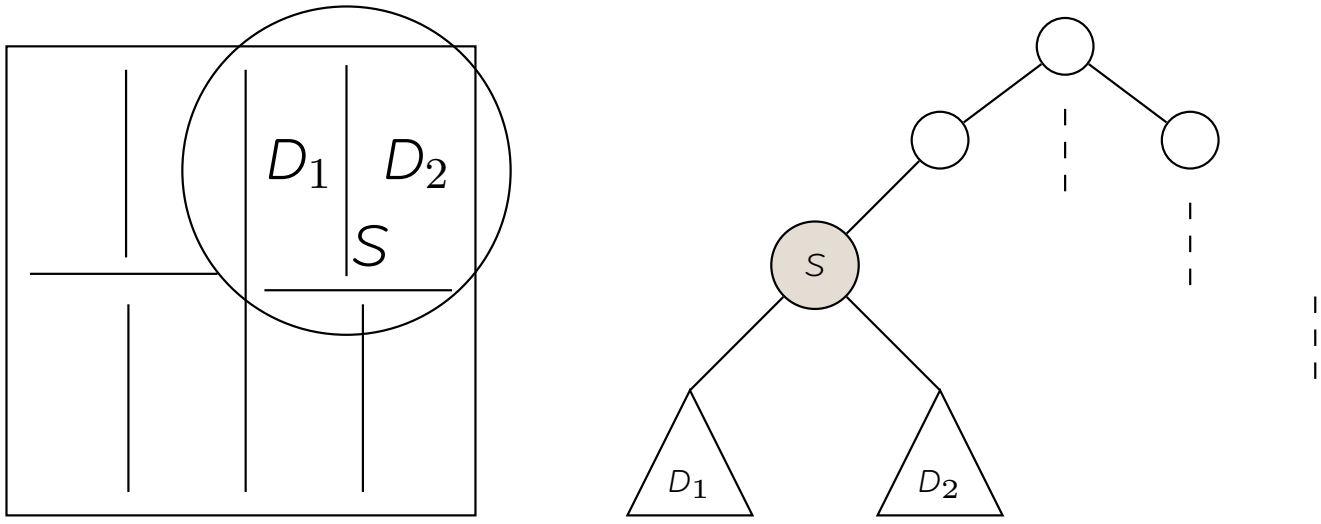
D_1 and D_2 are INDEPENDENT !

Elimination of D_i

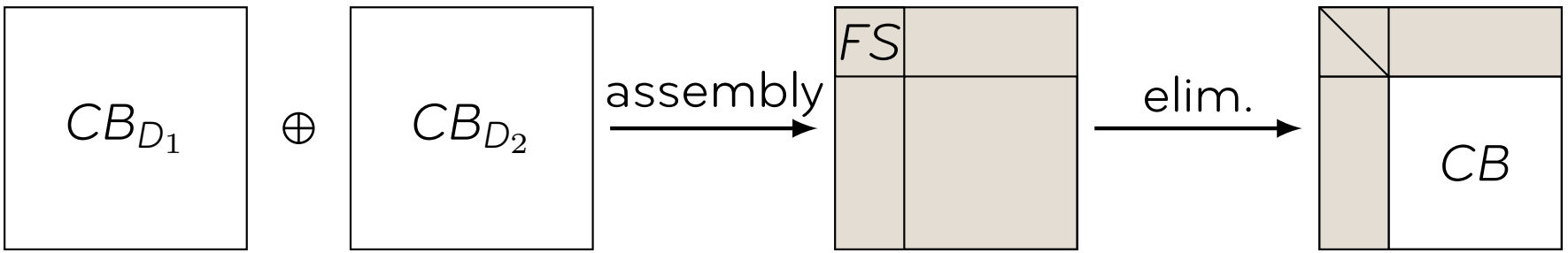


- $S \leftarrow S + CB_1 + CB_2$ *[assembly]*
- variables of S ready to be eliminated *[elimination]*

Generalization



At each node, a partial factorization of the frontal matrix is performed :



Goal of this study

Reveal and exploit low-rank structures within the fronts

Goal of this study

Reveal and exploit low-rank structures within the fronts

Why ?

- memory efficient : $A_{m,n} = U_{m,k} V_{n,k}^T + E_\varepsilon$ ("demote")
- computationally efficient : $AB = U_A (V_A^T U_B) V_B^T$

Goal of this study

Reveal and exploit low-rank structures within the fronts

Why ?

- memory efficient : $A_{m,n} = U_{m,k} V_{n,k}^T + E_\varepsilon$ ("demote")
- computationally efficient : $AB = U_A (V_A^T U_B) V_B^T$

Problem :

The fronts are FULL rank

Goal of this study

Reveal and exploit low-rank structures within the fronts

Why ?

- memory efficient : $A_{m,n} = U_{m,k} V_{n,k}^T + E_\varepsilon$ ("demote")
- computationally efficient : $AB = U_A (V_A^T U_B) V_B^T$

Problem :

The fronts are FULL rank

Solution :

Group variables to obtain low-rank *subblocks*

Grouping variables

Admissibility

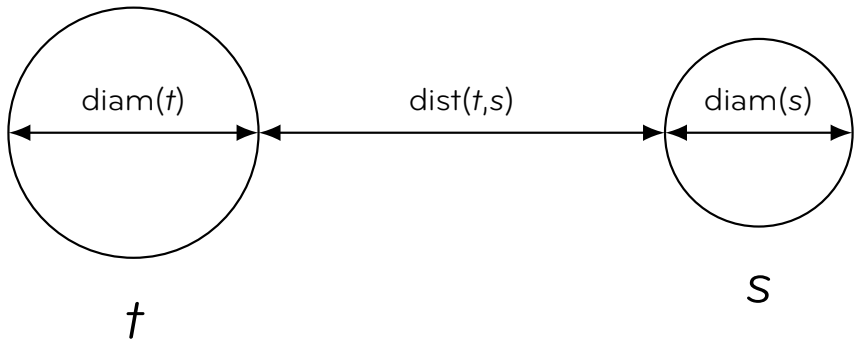
Is it *theoretically* worth demoting the block (t, s) ?

Admissibility condition for elliptic PDEs (Börm, Grasedyck, Hackbusch)

Let $t \subset \mathcal{I}$ and $s \subset \mathcal{J}$ be subsets of indices. The block (t, s) is said admissible if t and s satisfy

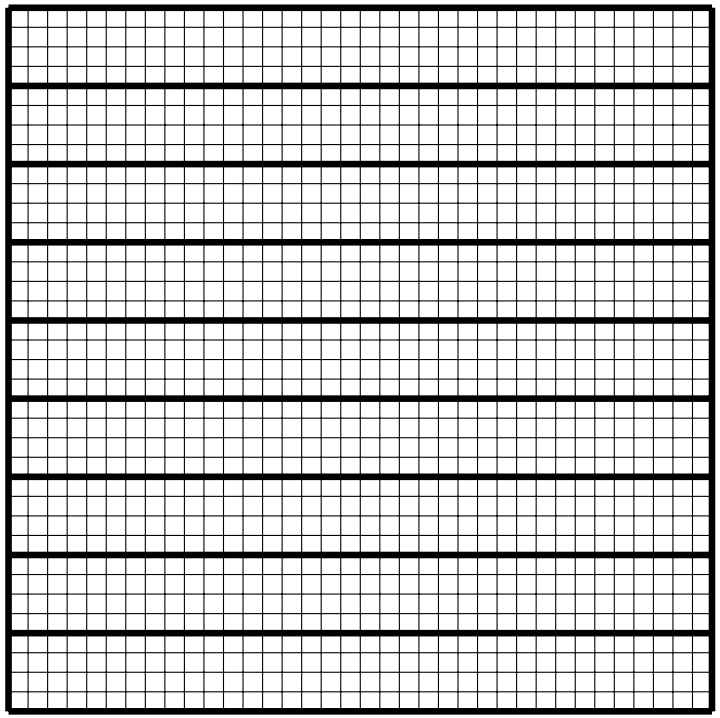
$$\text{diam}(t) + \text{diam}(s) \leq 2.\eta.\text{dist}(t, s)$$

where η is a problem dependant fixed parameter (usually 1 or 0.5)

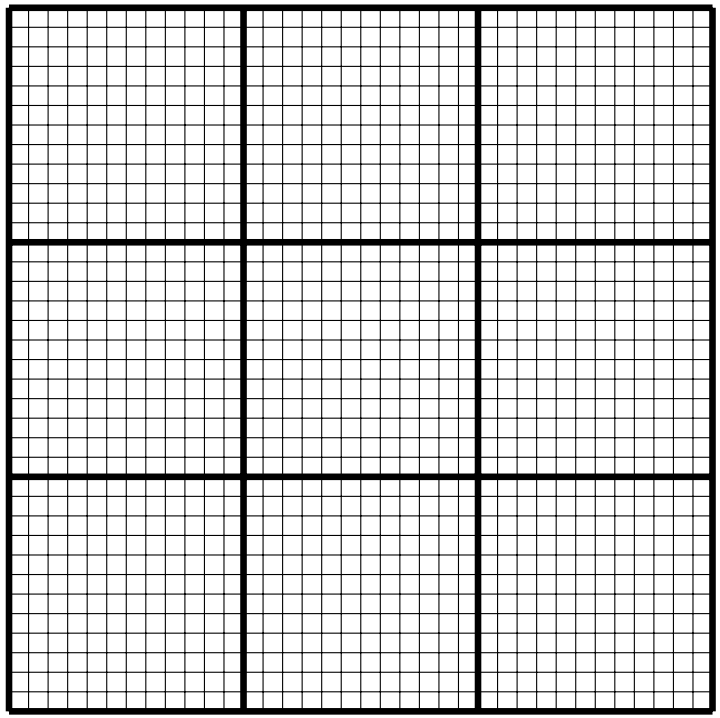


How to group the variables of a separator ?

Constraint : the admissibility condition should be satisfied

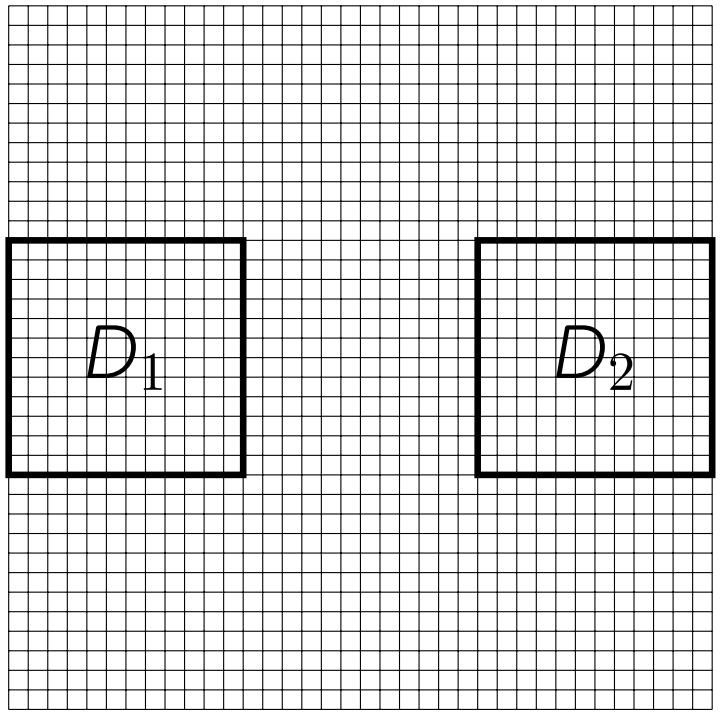


large diameters
fraction of memory used 83%

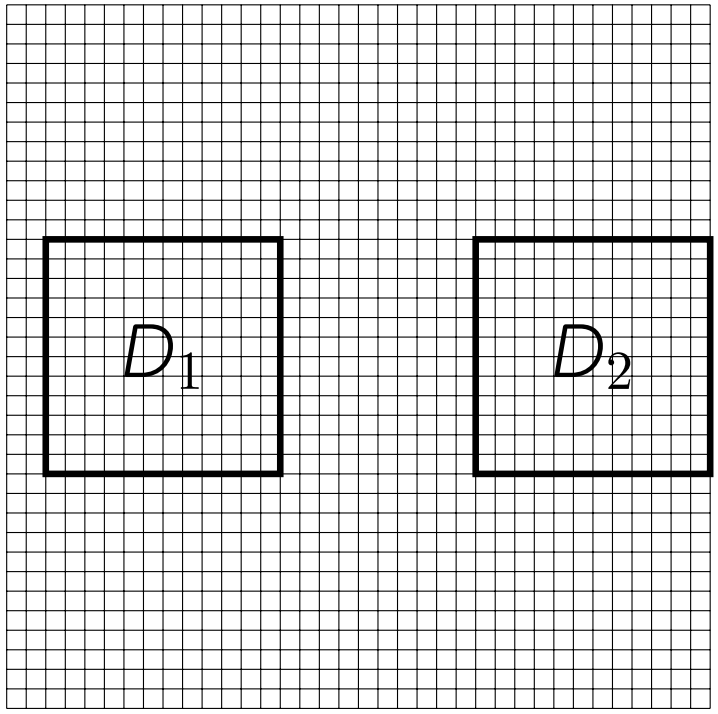


small diameters
fraction of memory used 57%

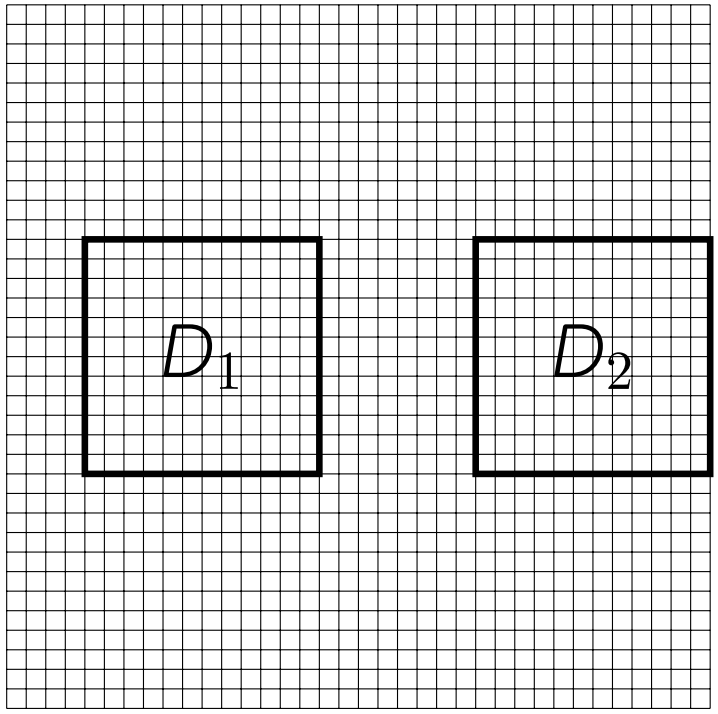
Rank vs distance (1)



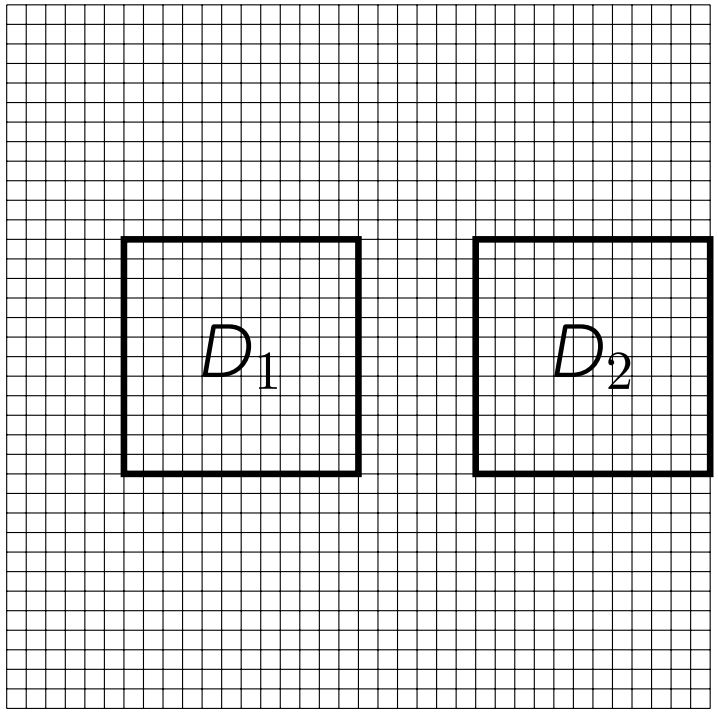
Rank vs distance (1)



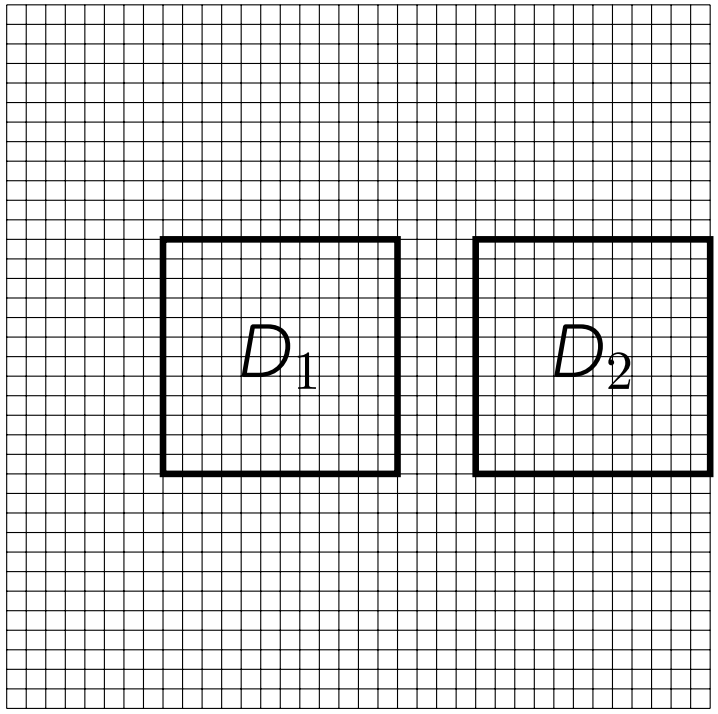
Rank vs distance (1)



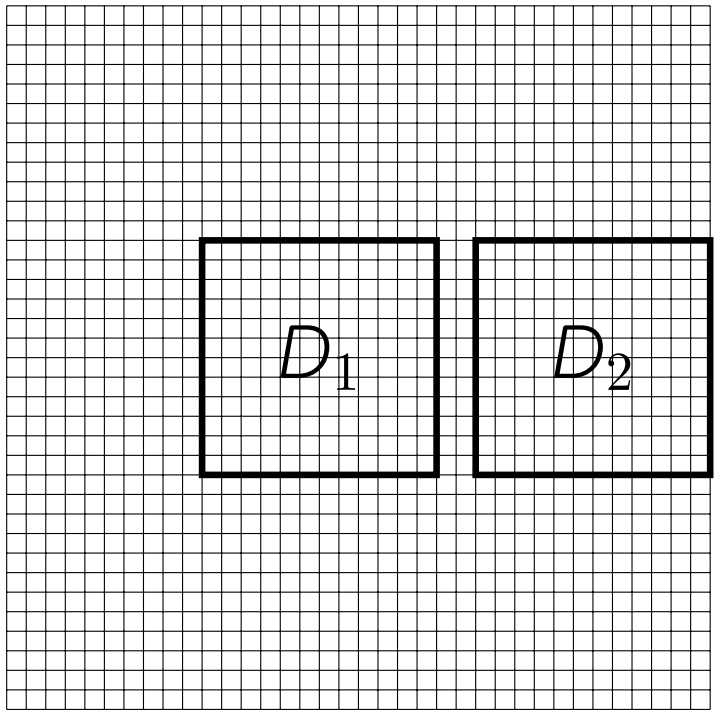
Rank vs distance (1)



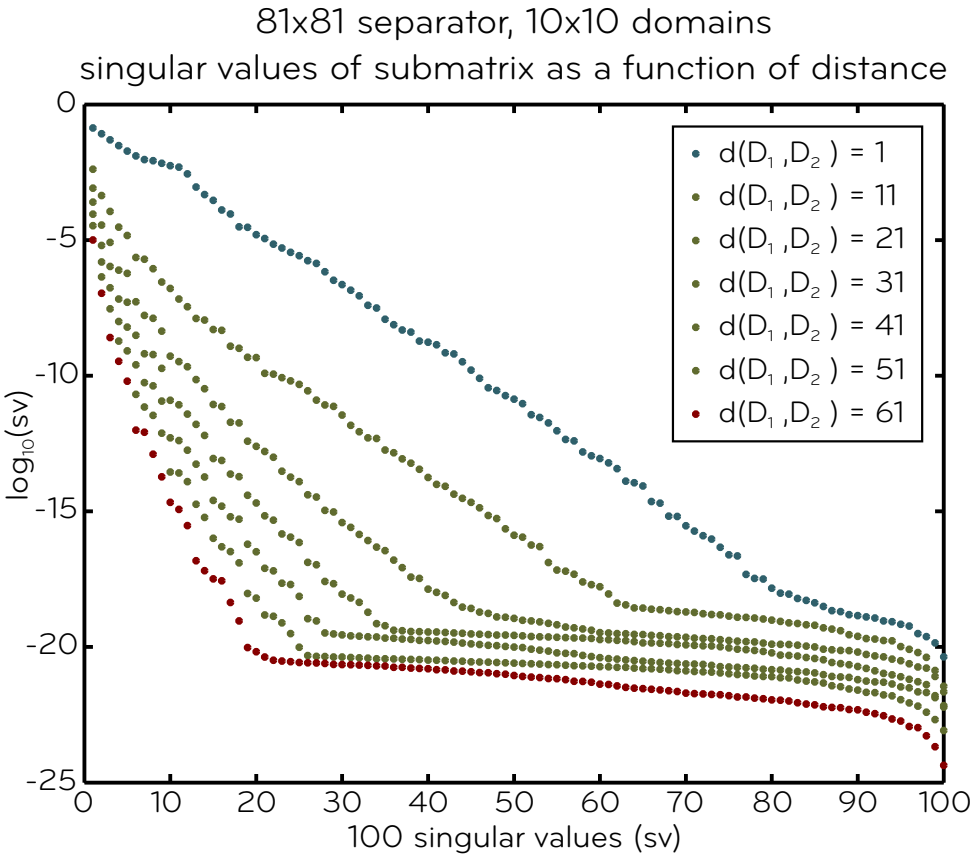
Rank vs distance (1)

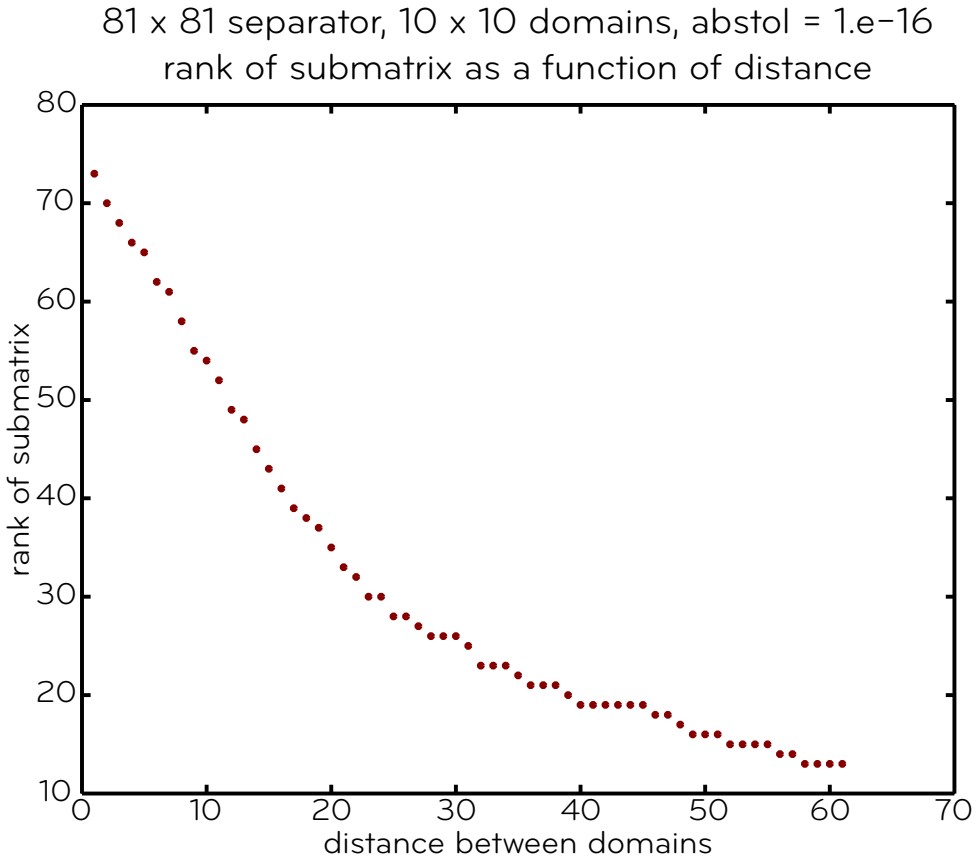


Rank vs distance (1)



Rank vs distance (2)



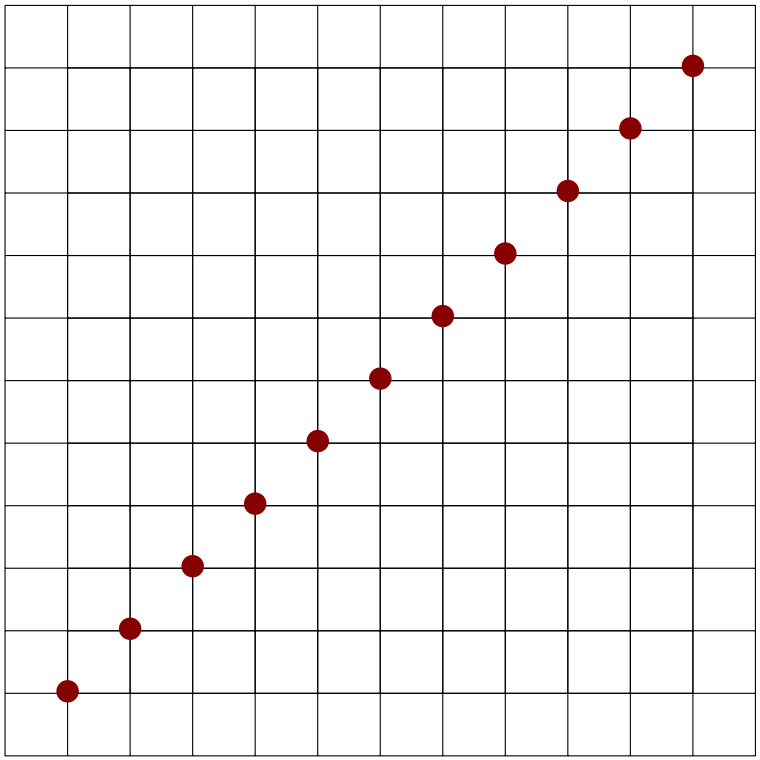


Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool

Halo algorithm to group a separator

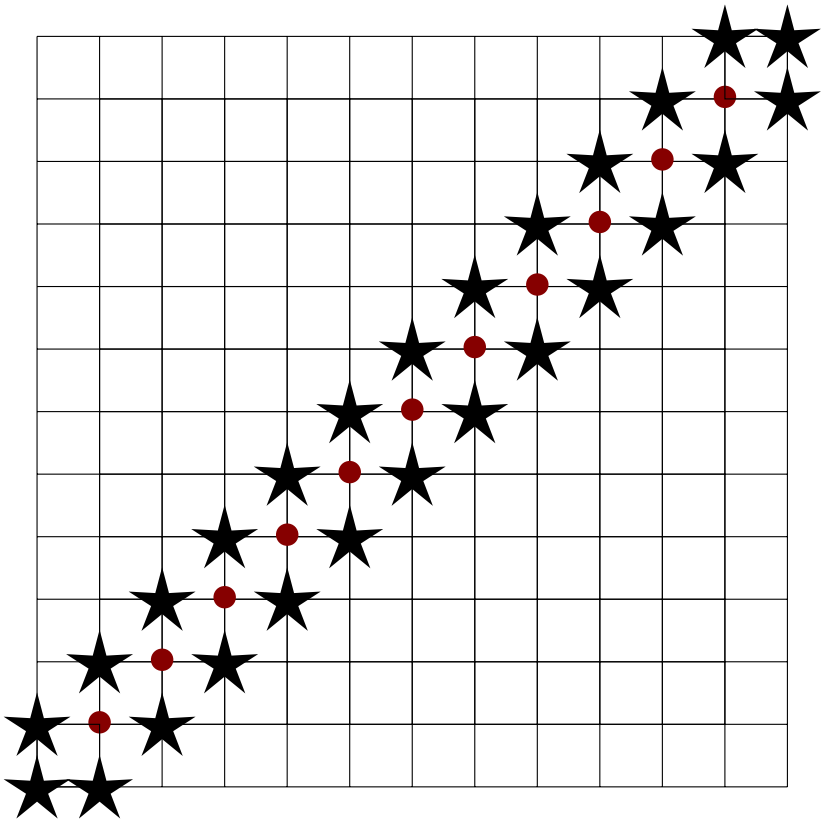
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator

Halo algorithm to group a separator

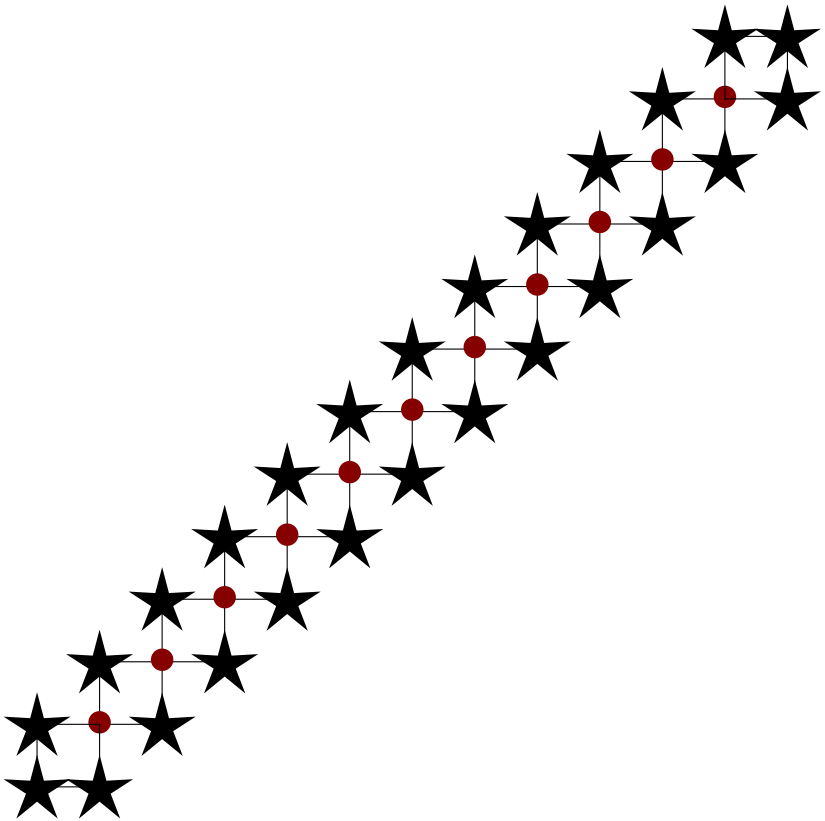
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator
2. The halo

Halo algorithm to group a separator

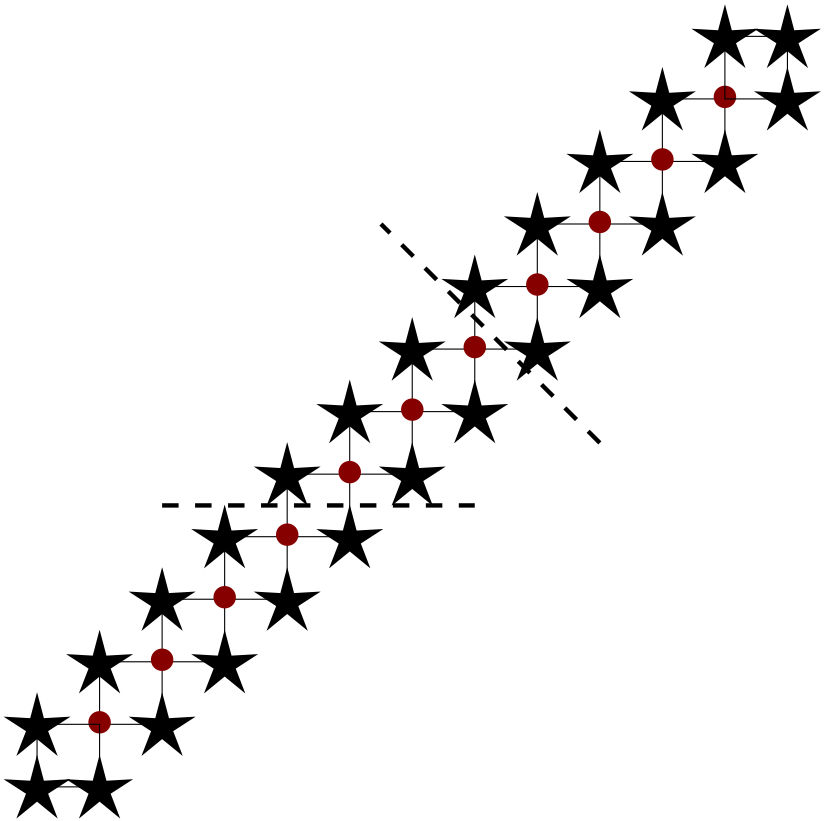
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator
2. The halo
3. Extraction of the halo

Halo algorithm to group a separator

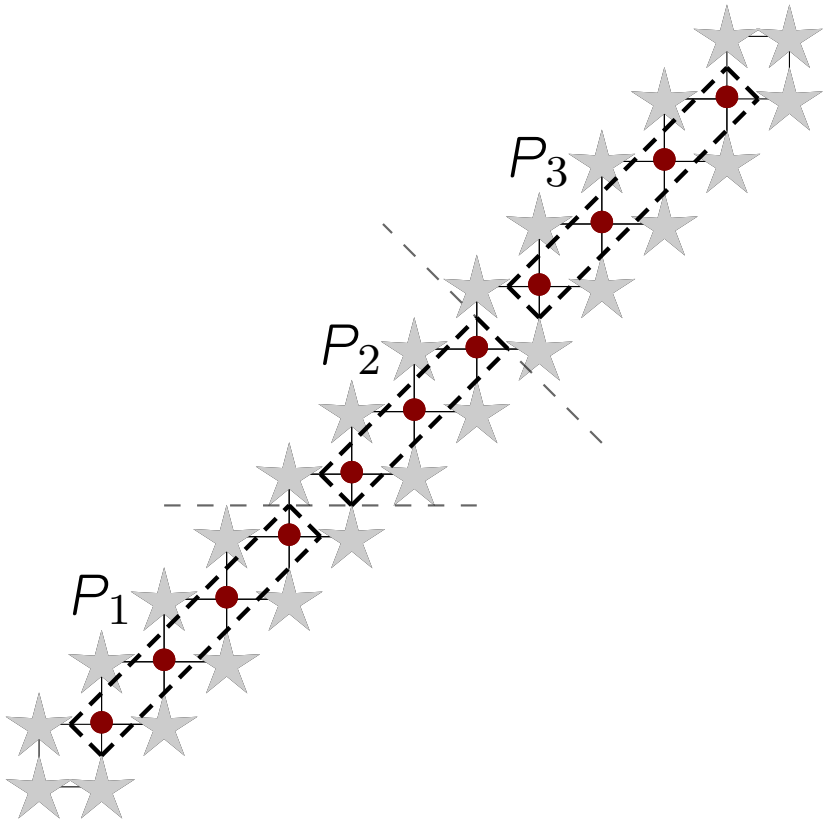
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator
2. The halo
3. Extraction of the halo
4. Partition of the halo

Halo algorithm to group a separator

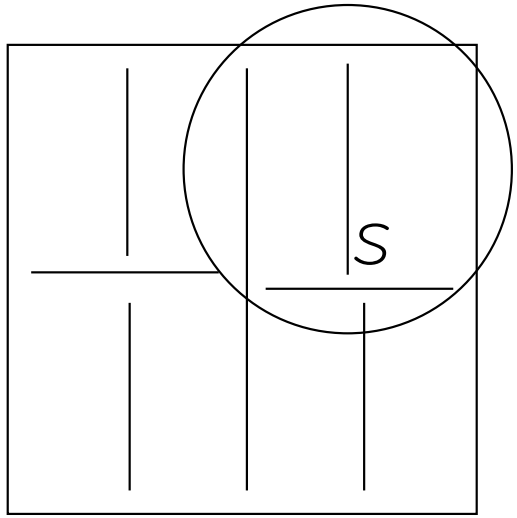
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator
2. The halo
3. Extraction of the halo
4. Partition of the halo
5. Partition of the separator (block size is fixed)

How to group the variables of a front ?

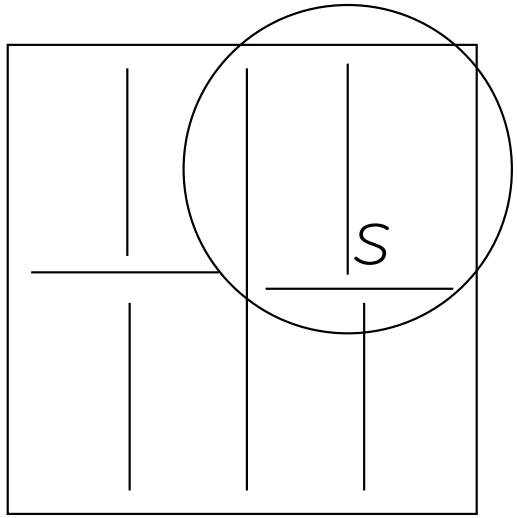
$\Rightarrow$ front = separator + border



How to group the variables of a front ?

$\Rightarrow$ front = separator + border

1- separator : halo

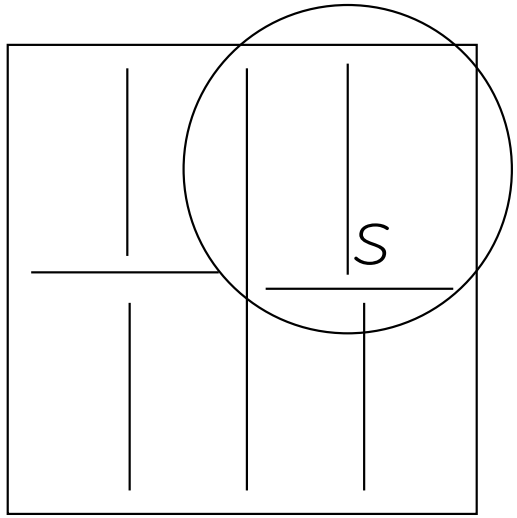


How to group the variables of a front ?

$\Rightarrow$ front = separator + border

1- separator : halo

2- border ? 2 choices :

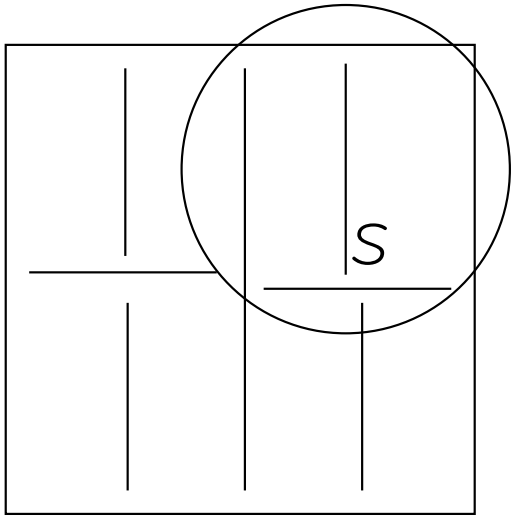


How to group the variables of a front ?

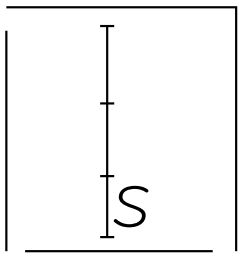
$\Rightarrow$ front = separator + border

1- separator : halo

2- border ? 2 choices :



EXPLICIT

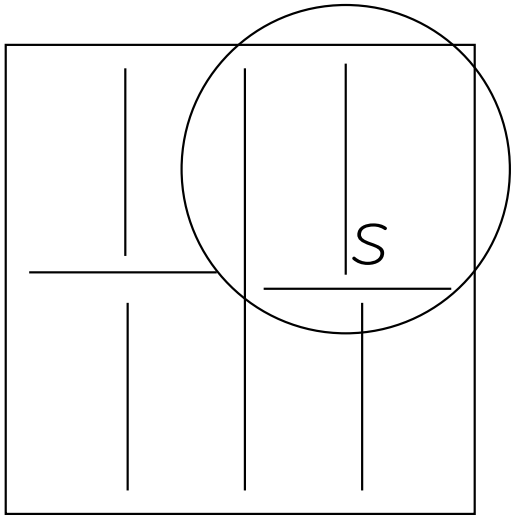


How to group the variables of a front ?

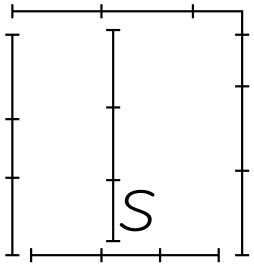
⇒ front = separator + border

1- separator : halo

2- border ? 2 choices :



EXPLICIT

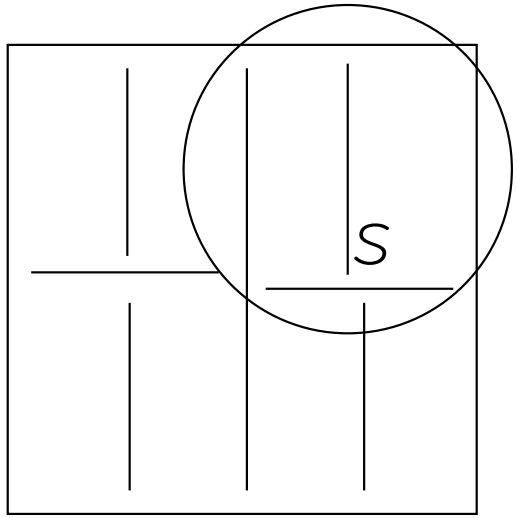


How to group the variables of a front ?

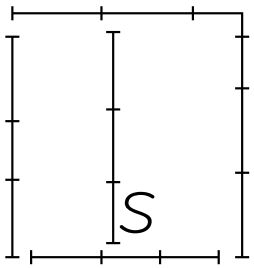
⇒ front = separator + border

1- separator : halo

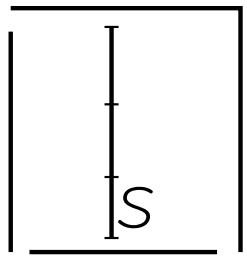
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

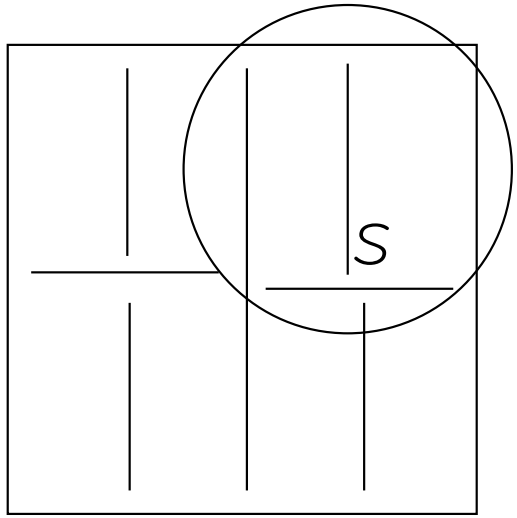


How to group the variables of a front ?

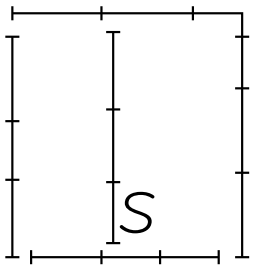
⇒ front = separator + border

1- separator : halo

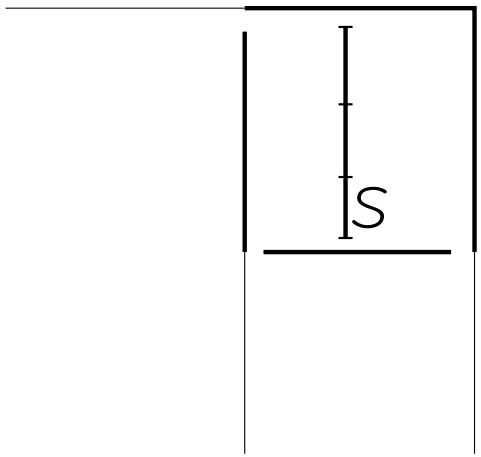
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

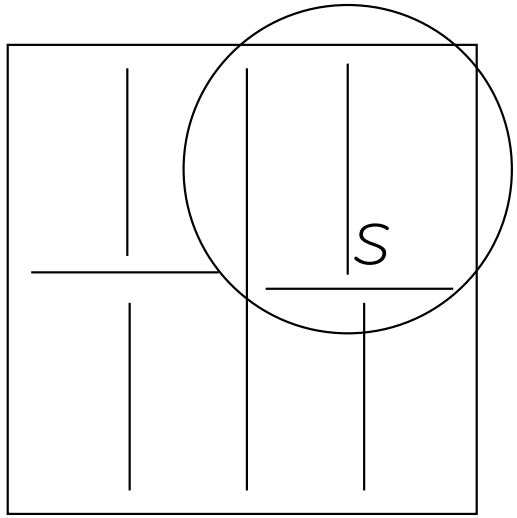


How to group the variables of a front ?

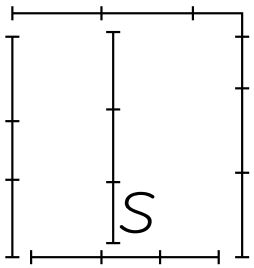
⇒ front = separator + border

1- separator : halo

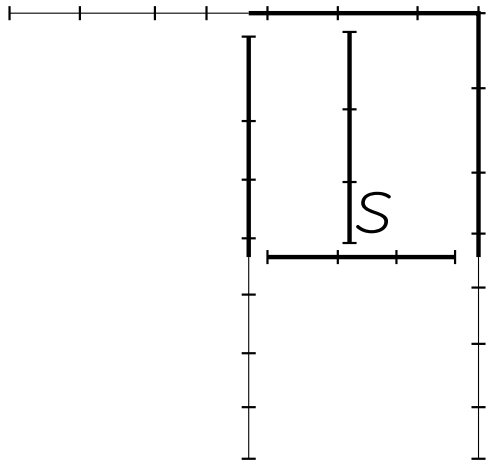
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

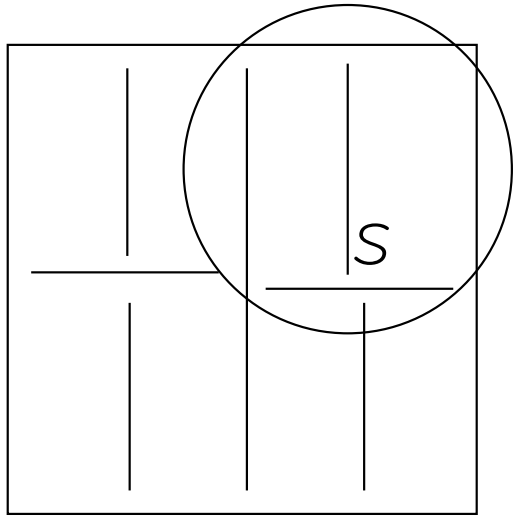


How to group the variables of a front ?

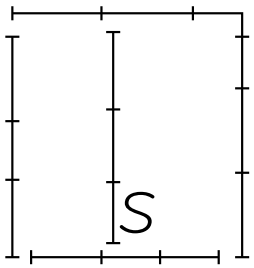
⇒ front = separator + border

1- separator : halo

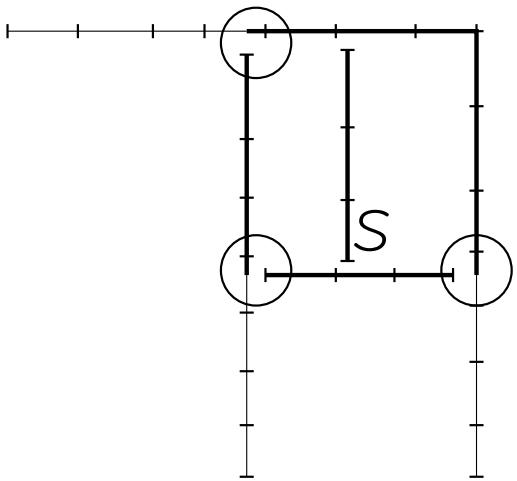
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

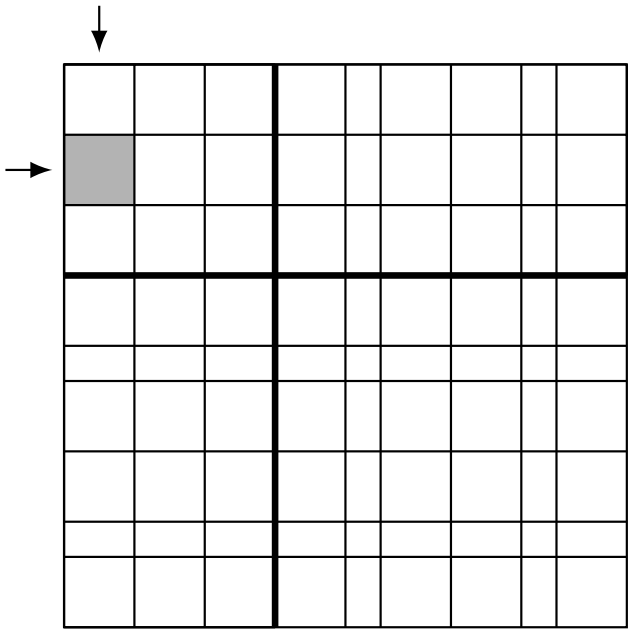


Comparison

- "inherited" version is more than 2 times faster

Comparison

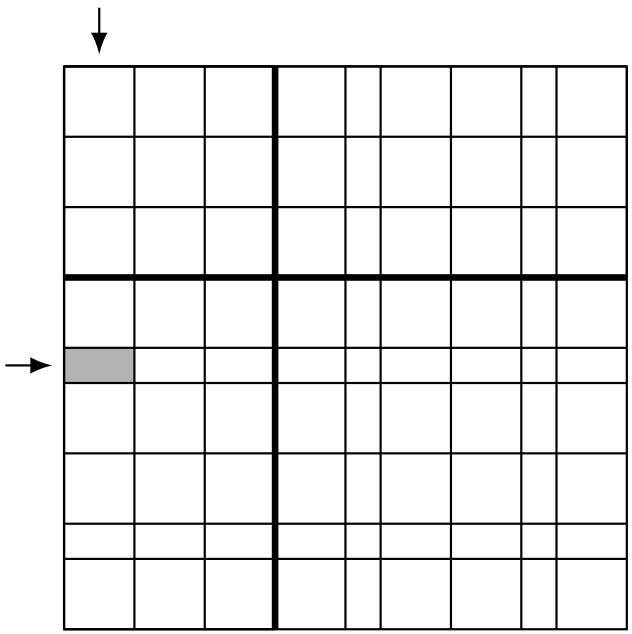
- "inherited" version is more than 2 times faster
- same results on L_{11}



- optimal × optimal = optimal block

Comparison

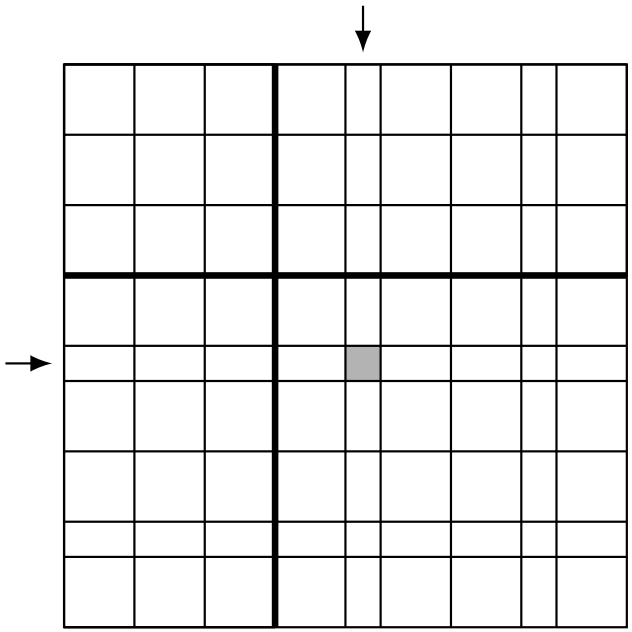
- "inherited" version is more than 2 times faster
- same results on L_{11}
- comparable results on L_{21} ($\sim 2-3\%$ lost)



- optimal × optimal = optimal block
- small × optimal = large enough block

Comparison

- "inherited" version is more than 2 times faster
- same results on L_{11}
- comparable results on L_{21} ($\sim 2-3\%$ lost)
- a little less good on CBs ($\sim 10\%$ lost)

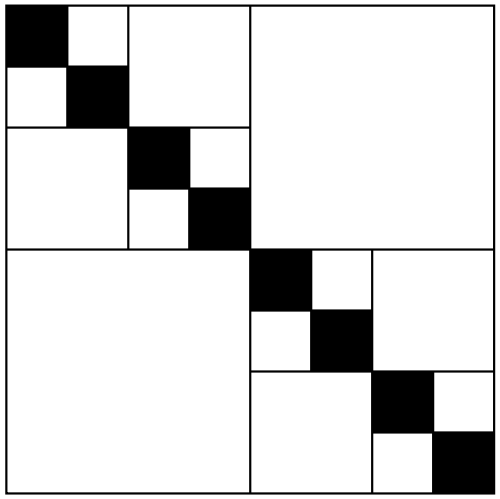


- optimal × optimal = optimal block
- small × optimal = large enough block
- small × small = too small block

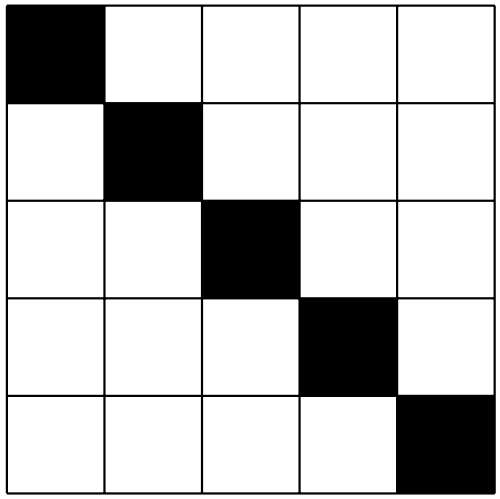
Exploiting low-rank blocks

Many matrix structures ...

$\mathcal{H}$ and $\mathcal{H}^2$ matrices (Hackbusch et al.), HSS matrices (Li, Napov, Xia et al.), HBS matrices (Gillman & Martinsson), BLR (us !) ...



HSS structure



BLR structure

Block Low-Rank (BLR) ...

- has less constraints for the grouping ;
- is more flexible (pivoting, ordering) ;
- seems more adapted to a multifrontal solver (small domains) ;

Experiments

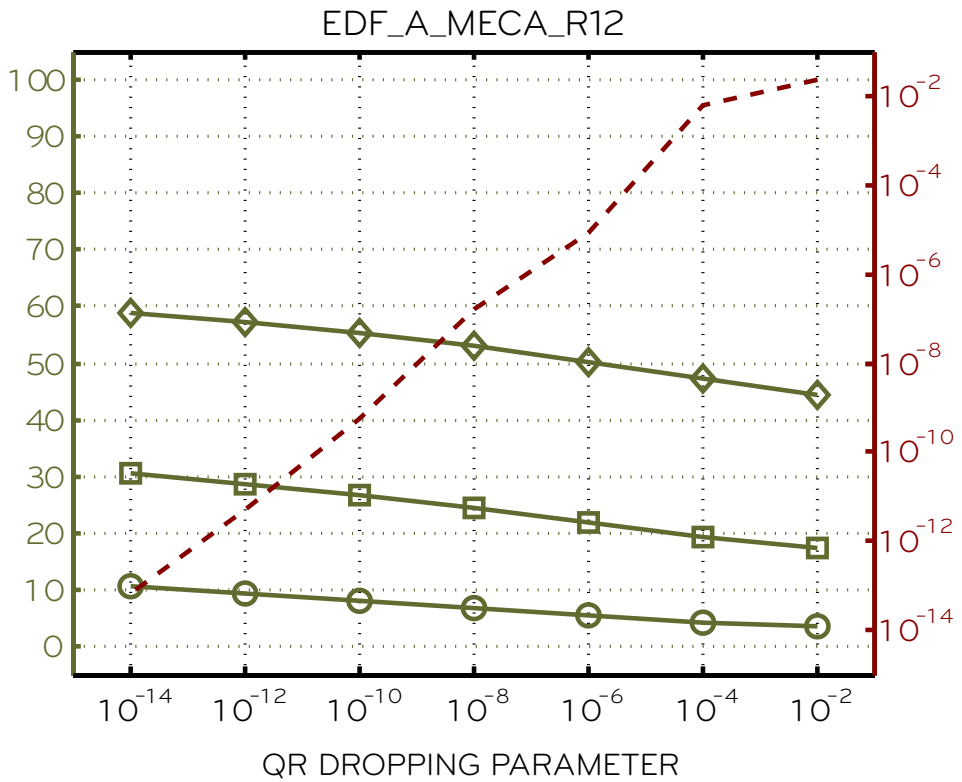
Set of problems

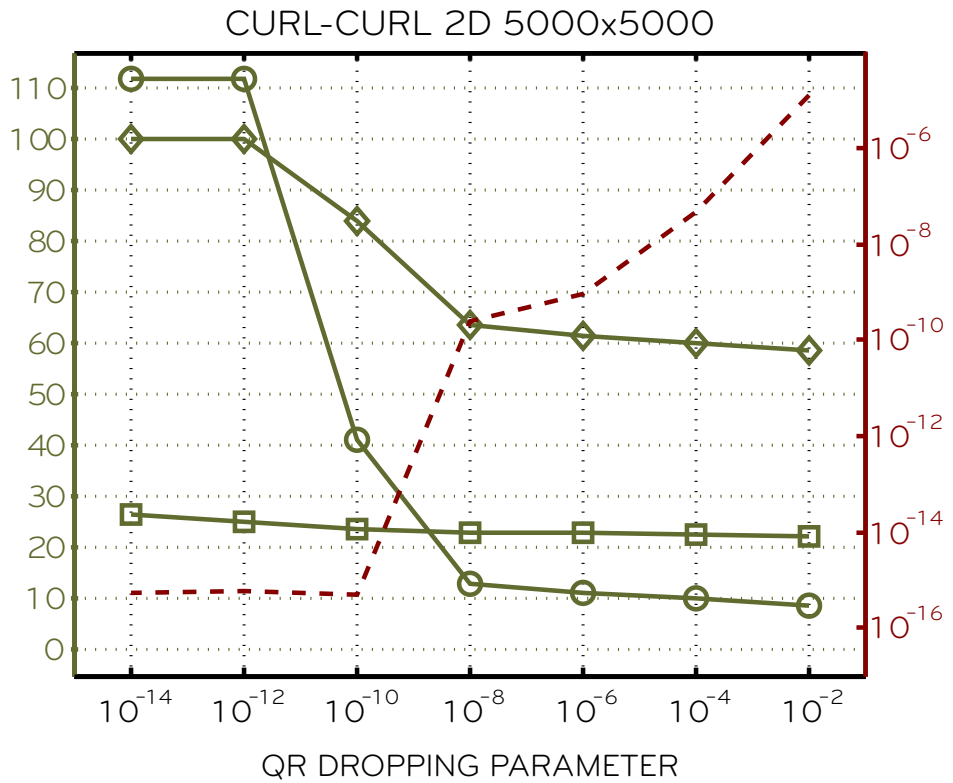
Name	Arith.	N	NZ	memory	flops	CSR$^{(1)}$	appli.
`Curl` 5000^2	D	50.10^6	2.10^8	29 GB	5.10^{12}	2.10^{-15}	∇
`Geoazur` 128^3	C	2.10^6	55.10^9	54 GB	6.10^{13}	3.10^{-4}	wave prop.
`EDF_A_MECA_R12`	D	134.10^6	1.10^9	151 GB	2.10^{14}	4.10^{-15}	mechanics
`EDF_D_THER_R7`	D	8.10^6	118.10^9	229 GB	1.10^{14}	8.10^{-15}	thermics

(1) CSR = Componentwise Scaled Residual = $\max_i \dfrac{|\mathbf{b} - \mathbf{A}\bar{\mathbf{x}}|_i}{(|\mathbf{b}| + |\mathbf{A}|\,|\bar{\mathbf{x}}|)_i}$

- 1 toy and 3 real industrial problems
- different applications
- target : harder and harder problems $\Rightarrow$ industrial partners

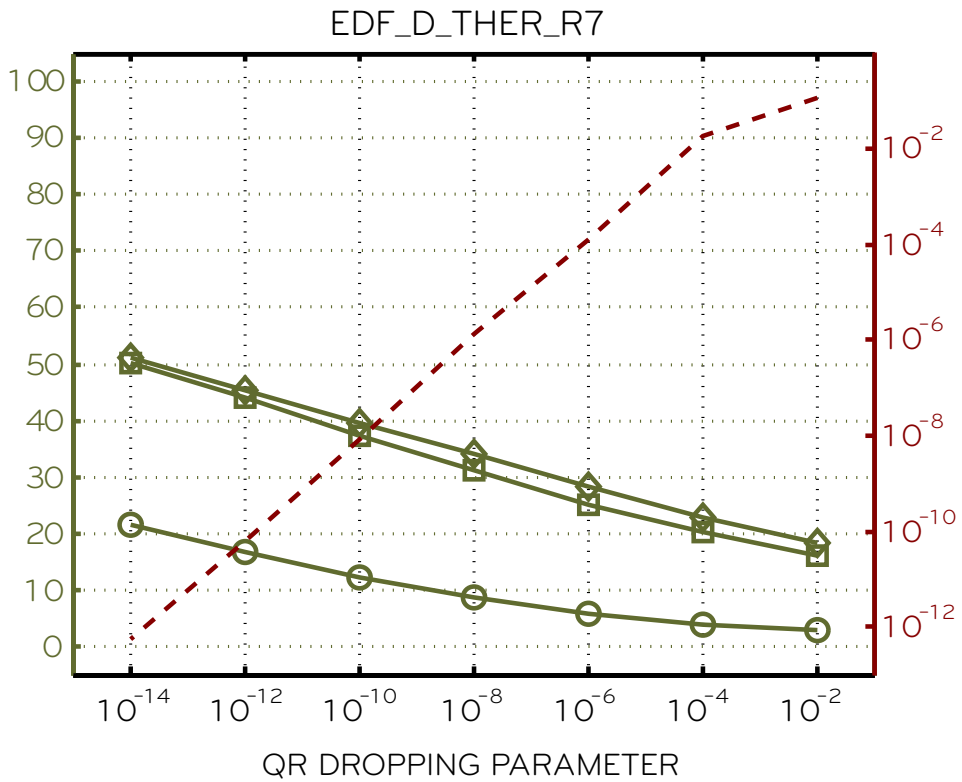
2D problems

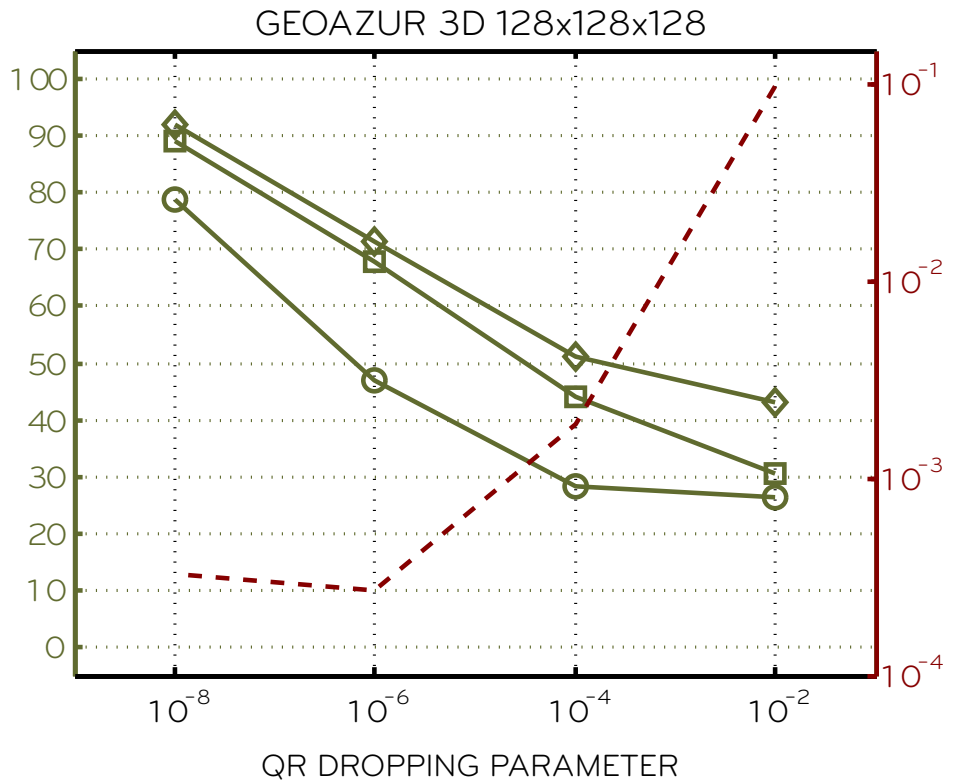




Name	N	memory	flops	CSR
EDF_A_MECA_R12	134.10^6	151 GB	2.10^{14}	4.10^{-15}
Curl-Curl 5000^2	50.10^6	29 GB	5.10^{12}	2.10^{-15}

3D problems





Name	N	memory	flops	CSR
EDF_D_THER_R7	8.10^6	229 GB	1.10^{14}	8.10^{-15}
Geoazur 128^3	2.10^6	54 GB	6.10^{13}	3.10^{-4}

Conclusion & perspectives

$\Rightarrow$ efficient method on applicative problems

- good memory reduction & large decrease in computations :
  150 GB $\rightarrow 60$ GB 7 H $\rightarrow 1$ H
- tree dependent method (any global ordering works)

$\Rightarrow$ can be used as a preconditioner or as an accurate solver

---

- in parallel, significant part of memory consumption is temporary data

$\Rightarrow$ rooms for improvements (CB compression)

- pivoting
- diversify tested problems
- error propagation study

Aknowledgements

- **Olivier Boiteau** (EDF R&D) for the matrices
- **Stéphane Operto** (SEISCOPE Project) for the 3D Geoazur generator
- the **Toulouse Computing Center** (CICT) and **Nicolas Renon**
- the **LBNL** (Berkeley)

¡Muchas gracias!

Any questions?