

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

Journée des Doctorants IRIT, Toulouse, France, September 7, 2012

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

MULTIFRONTAL SOLVER

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

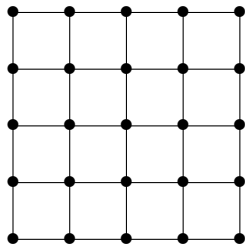
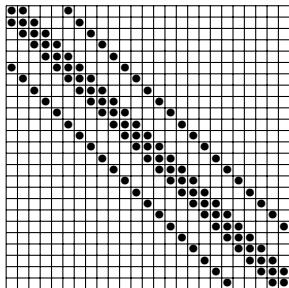
LOW-RANK APPROXIMATIONS

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

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

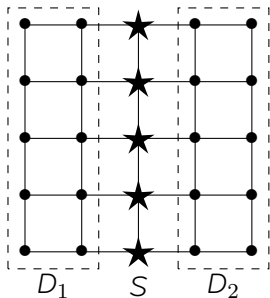
The multifrontal method

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

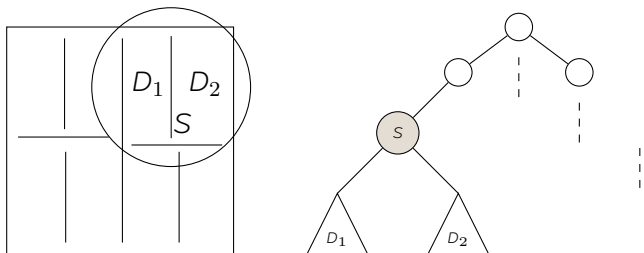
D_i	$D_i^S(21)$
$D_i^S(12)$	0



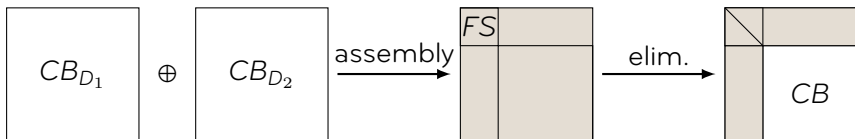
L_i	U_i	U_i^S
L_i^S		CB_i

- $S \leftarrow S + CB_1 + CB_2$
- variables of S ready to be eliminated

[assembly]
[elimination]



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



Reveal and exploit low-rank structures within the fronts

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 : $AB = U_A(V_A^T U_B)V_B^T$

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

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

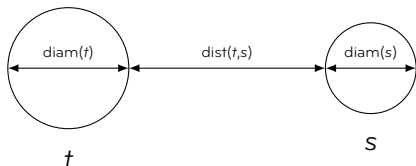
Is it *theoretically* worth demoting the block (t, s) ?

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

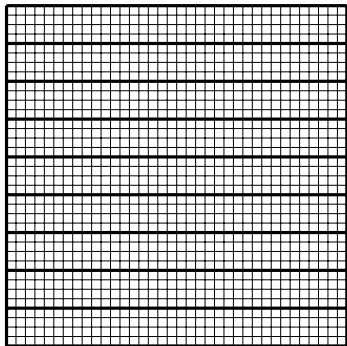
$$\text{diam}(t) + \text{diam}(s) \leq 2.\eta.\text{dist}(t, s)$$

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

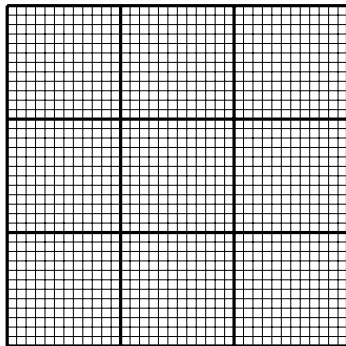


How to group the variables of a separator ?

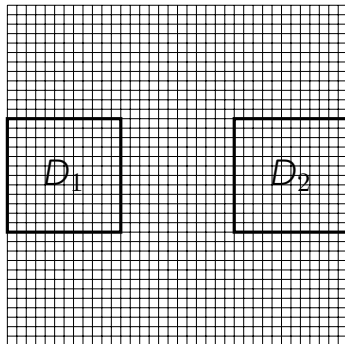
Constraint : the admissibility condition should be satisfied

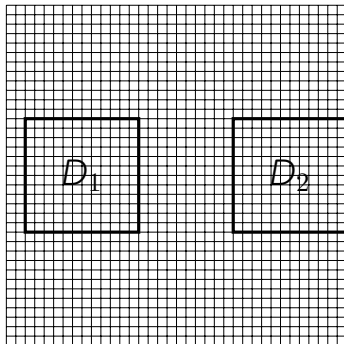


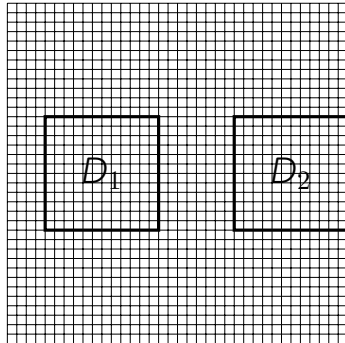
large diameters
fraction of memory used 83%

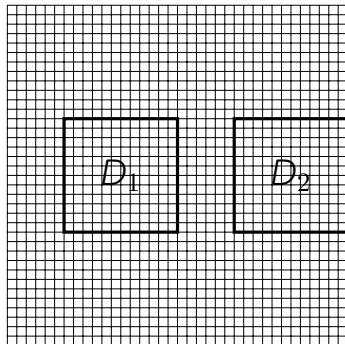


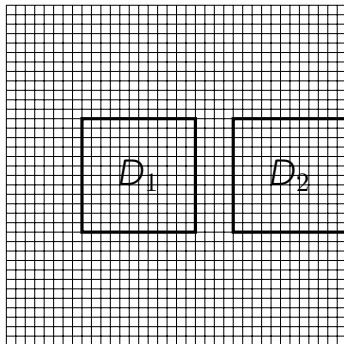
small diameters
fraction of memory used 57%

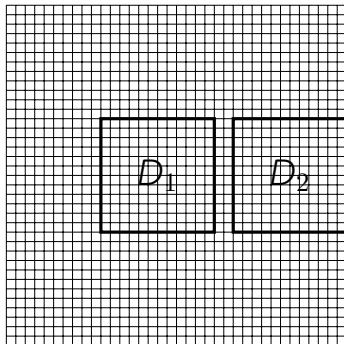




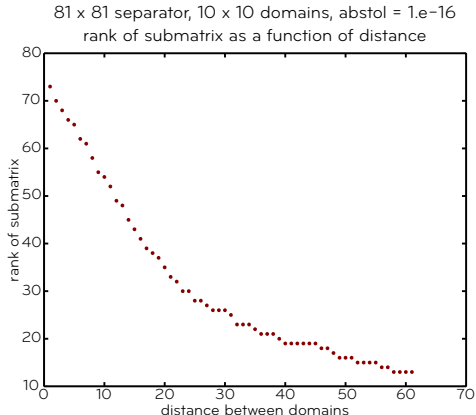
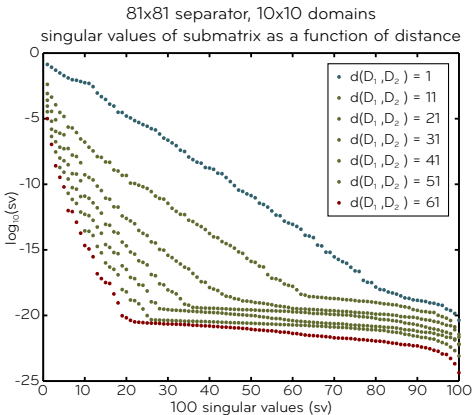








Rank vs distance (2)

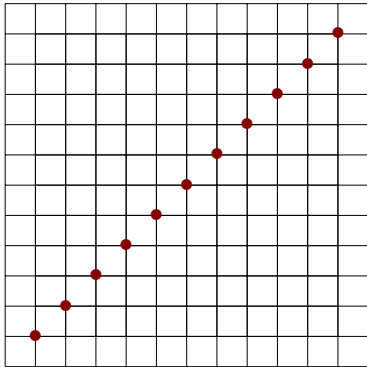


Halo algorithm to group a separator

- 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

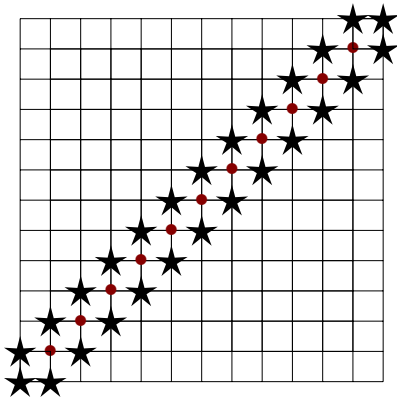
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator

Halo algorithm to group a separator

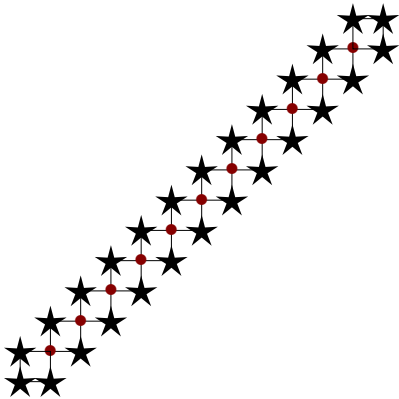
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



1. The separator
2. The halo

Halo algorithm to group a separator

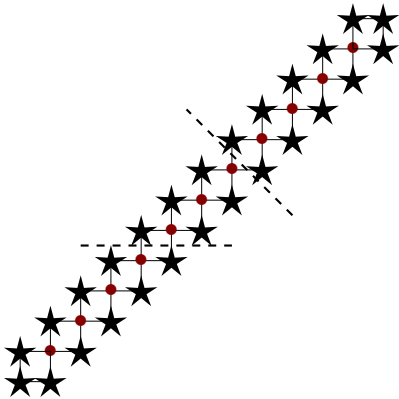
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



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

Halo algorithm to group a separator

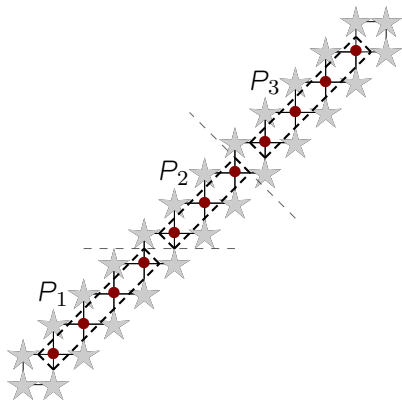
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



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

Halo algorithm to group a separator

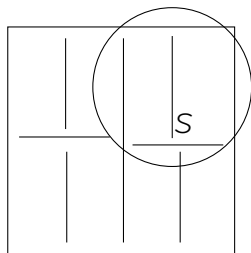
- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool



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 ?

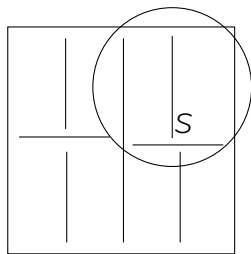
⇒ front = separator + border



How to group the variables of a front ?

⇒ front = separator + border

1- separator : halo

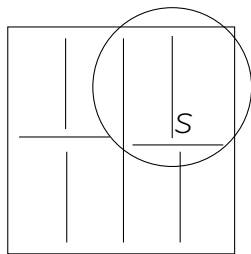


How to group the variables of a front ?

⇒ front = separator + border

1- separator : halo

2- border ? 2 choices :

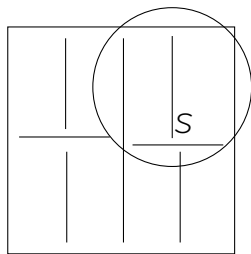


How to group the variables of a front ?

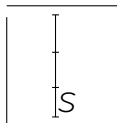
⇒ front = separator + border

1- separator : halo

2- border ? 2 choices :



EXPLICIT

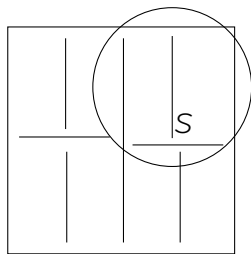


How to group the variables of a front ?

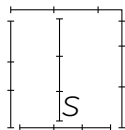
⇒ front = separator + border

1- separator : halo

2- border ? 2 choices :



EXPLICIT

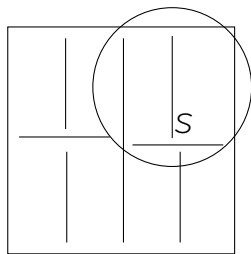


How to group the variables of a front ?

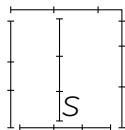
⇒ front = separator + border

1- separator : halo

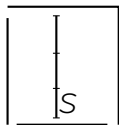
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

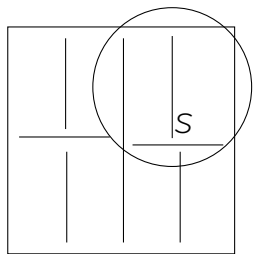


How to group the variables of a front ?

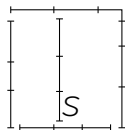
⇒ front = separator + border

1- separator : halo

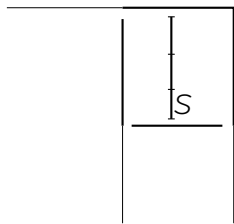
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

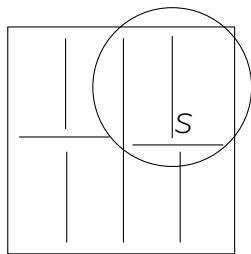


How to group the variables of a front ?

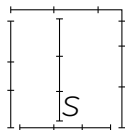
⇒ front = separator + border

1- separator : halo

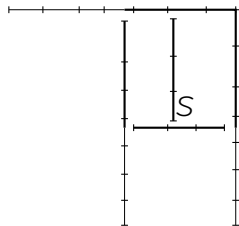
2- border ? 2 choices :



EXPLICIT



INHERITED (*top down*)

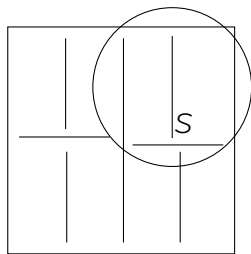


How to group the variables of a front ?

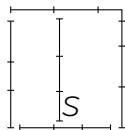
⇒ front = separator + border

1- separator : halo

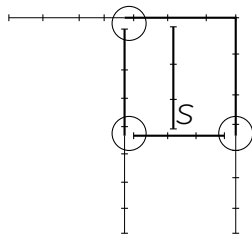
2- border ? 2 choices :



EXPLICIT



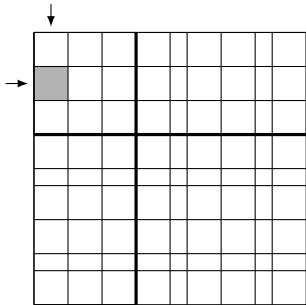
INHERITED (*top down*)



- “inherited” version is more than 2 times faster

Comparison

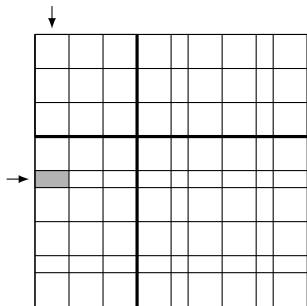
- "inherited" version is more than 2 times faster
- same results on L_{11}



- optimal \times optimal = optimal block

Comparison

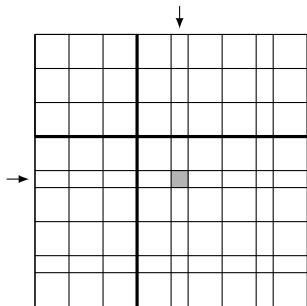
- "inherited" version is more than 2 times faster
- same results on L_{11}
- comparable results on L_{21} ($\sim 2 - 3\%$ lost)



- optimal \times optimal = optimal block
- small \times optimal = large enough block

Comparison

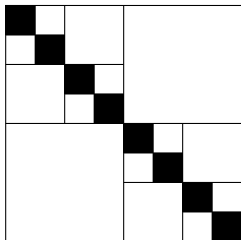
- "inherited" version is more than 2 times faster
- same results on L_{11}
- comparable results on L_{21} ($\sim 2 - 3\%$ lost)
- a little less good on CBs ($\sim 10\%$ lost)



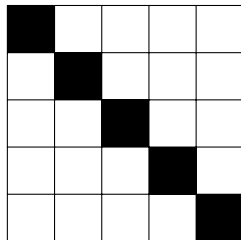
- optimal \times optimal = optimal block
- small \times optimal = large enough block
- small \times small = too small block

Exploiting low-rank blocks

\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) ...

- less constraints for the grouping (any global ordering works) ;
- more efficient SVD (smaller blocks) ;
- L_{21} part ;

Experiments

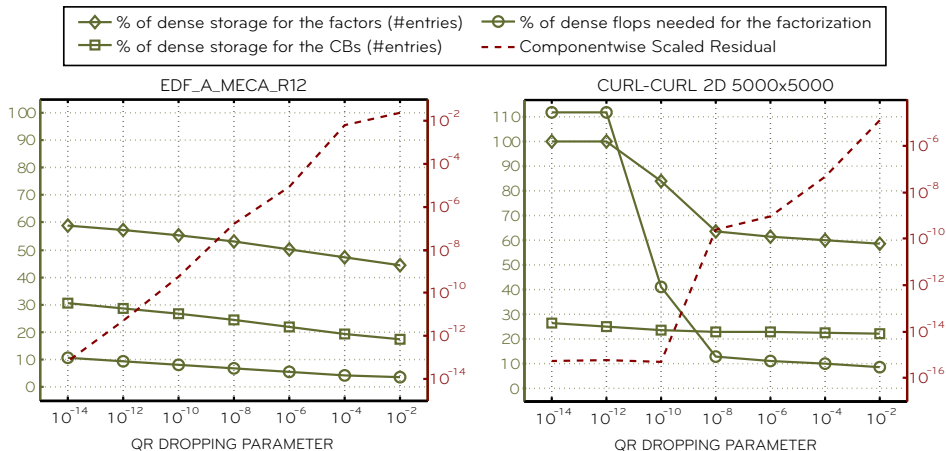
Set of problems

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 = $\max_i \frac{|\mathbf{b} - \mathbf{A}\bar{\mathbf{x}}|_i}{(|\mathbf{b}| + |\mathbf{A}| |\bar{\mathbf{x}}|)_i}$

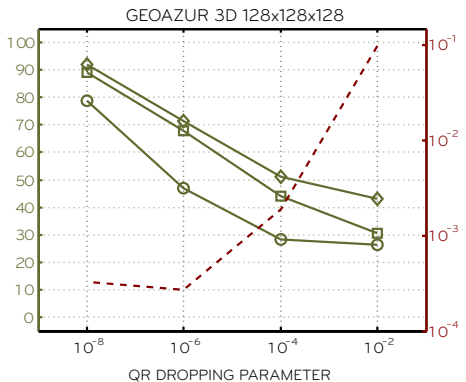
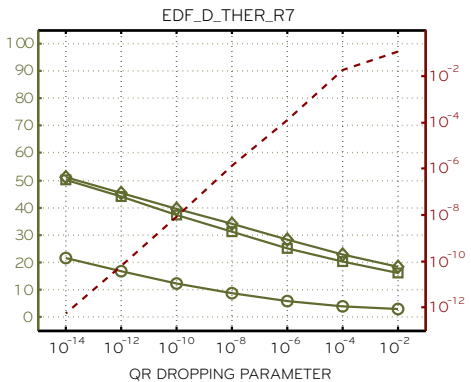
- 1 toy and 3 real industrial problems
- different applications
- target : harder and harder problems \Rightarrow industrial partners

2D problems



Name	N	memory	flops	CSR
EDF_A_MECA_R12	$134 \cdot 10^6$	151 GB	$2 \cdot 10^{14}$	$4 \cdot 10^{-15}$
Curl-Curl 5000 ²	$50 \cdot 10^6$	29 GB	$5 \cdot 10^{12}$	$2 \cdot 10^{-15}$

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}

⇒ efficient method on applicative problems

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

⇒ can be used as a preconditioner or as an accurate solver

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

⇒ rooms for improvements (CB compression)

- pivoting
- diversify tested problems
- error propagation study

- **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)



Thank you !

Any questions?