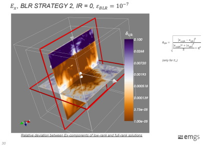
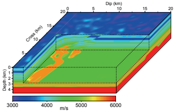
Direct solution of sparse systems of linear equations with sparse multiple right-hand sides

P. Amestoy[1], J.-Y. L'Excellent[2], G. Moreau[2], 1. Université de Toulouse INPT and IRIT , 2. Université de Lyon, Inria and LIP-ENS Lyon ,
gilles.moreau@ens-lyon.fr

Sparse Days, Cerfacs, September 6-8, 2017



Linear systems of equations :
$Ax = b$, A is sparse
Solve phase ($Ly = b$, $Ux = y$) may be critical.



Application coming from Helmholtz or Maxwell equations:

name	n (million)	$nrhs$	$nnz/nrhs$	T_{facto}	T_{solve}
sei70m	2.9	2302	587	1258	1267
sei50m	7.1	2302	486	6289	2985
E1	0.33	8000	9.8	55.2	291
E3	2.8	8000	7.5	1951	5610

Table : Characteristics of matrices and right-hand-sides.

Introduction

Objectives:

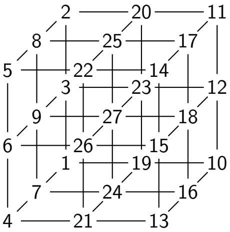
- focus on the forward solution phase $Ly = b$;
- exploit sparsity of right-hand-sides;
- limit the number of operations (Δ);

Overview

- Exploitation of sparse right-hand-sides
  - Context of study
  - Tree pruning
  - Exploitation of subintervals of columns at each node
- Minimizing the number of operations
  - Permutation of columns
  - Adapted blocking technique
- Conclusion

Ordering: reorder variables of the matrix A to reduce fill-in and build elimination tree:

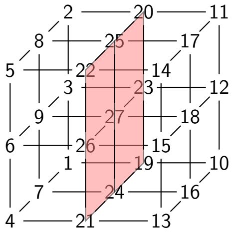
- Nested Dissection ⇒ build tree of separators.

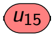


3D physical domain (cube)

Ordering: reorder variables of the matrix A to reduce fill-in and build elimination tree:

- Nested Dissection ⇒ build tree of separators.

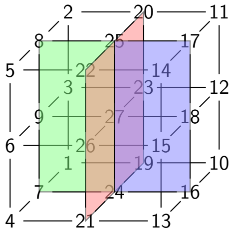


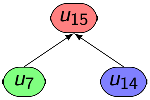


3D physical domain (cube)   separator tree

Ordering: reorder variables of the matrix A to reduce fill-in and build elimination tree:

- Nested Dissection ⇒ build tree of separators.





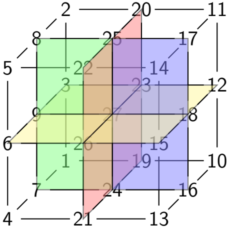
3D physical domain (cube)

separator tree

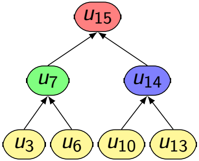
Context: analysis phase

Ordering: reorder variables of the matrix A to reduce fill-in and build elimination tree:

- Nested Dissection ⇒ build tree of separators.



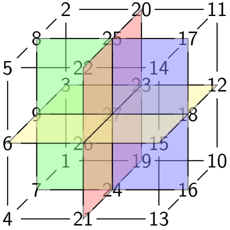
3D physical domain (cube)

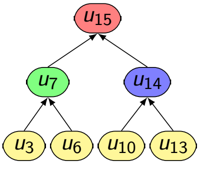


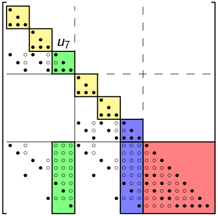
separator tree

Ordering: reorder variables of the matrix A to reduce fill-in and build elimination tree:

- Nested Dissection ⇒ build tree of separators.



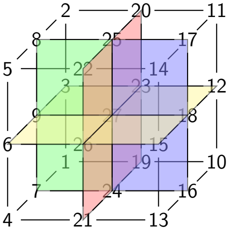




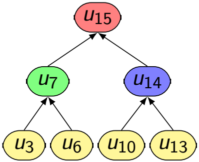
3D physical domain (cube) separator tree L factor

Ordering: reorder variables of the matrix A to reduce fill-in and build elimination tree:

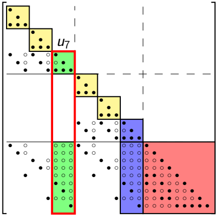
- Nested Dissection $\Rightarrow$ build tree of separators.



3D physical domain (cube)

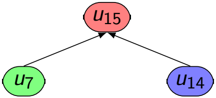


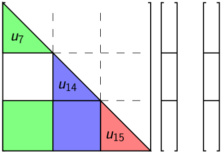
separator tree



L factor

Forward solution process





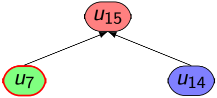
Block operations:

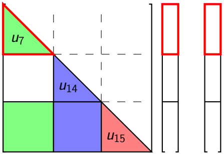
- $y_1 \leftarrow L_{11}^{-1} b_1$
- $b_2 \leftarrow b_2 - L_{21} y_1$

#flops for node u is given by:

$$\mathcal{F}_u = 2 * (\#\text{entries in } L_{11} + L_{21})$$

Forward solution process





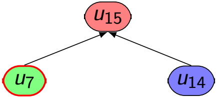
Block operations:

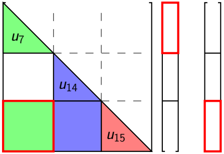
- $y_1 \leftarrow L_{11}^{-1} b_1$
- $b_2 \leftarrow b_2 - L_{21} y_1$

#flops for node u is given by:

$$\mathcal{F}_u = 2 * (\#\text{entries in } L_{11} + L_{21})$$

Forward solution process





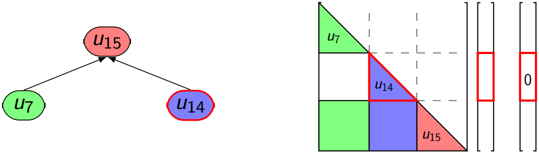
Block operations:

- $y_1 \leftarrow L_{11}^{-1} b_1$
- $b_2 \leftarrow b_2 - L_{21} y_1$

#flops for node u is given by:

$$\mathcal{F}_u = 2 * (\#\text{entries in } L_{11} + L_{21})$$

Forward solution process



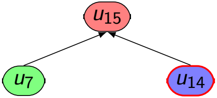
Block operations:

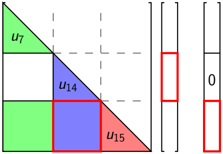
- $y_1 \leftarrow L_{11}^{-1} b_1$
- $b_2 \leftarrow b_2 - L_{21} y_1$

#flops for node u is given by:

$$\mathcal{F}_u = 2 * (\#\text{entries in } L_{11} + L_{21})$$

Forward solution process





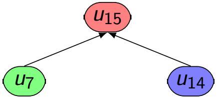
Block operations:

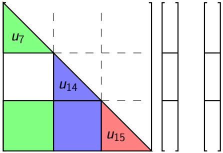
- $y_1 \leftarrow L_{11}^{-1} b_1$
- $b_2 \leftarrow b_2 - L_{21} y_1$

#flops for node u is given by:

$$\mathcal{F}_u = 2 * (\#\text{entries in } L_{11} + L_{21})$$

Forward solution process





Block operations:

- $y_1 \leftarrow L_{11}^{-1} b_1$
- $b_2 \leftarrow b_2 - L_{21} y_1$

#flops for node u is given by:

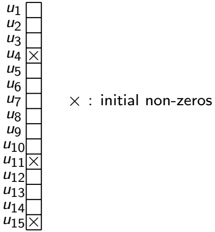
$$\mathcal{F}_u = 2 * (\#\text{entries in } L_{11} + L_{21})$$

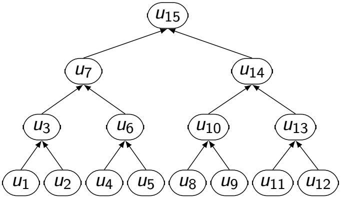
Total #flops:

$$\Delta = \sum_{u \in T} \mathcal{F}_u$$

Forward solution process

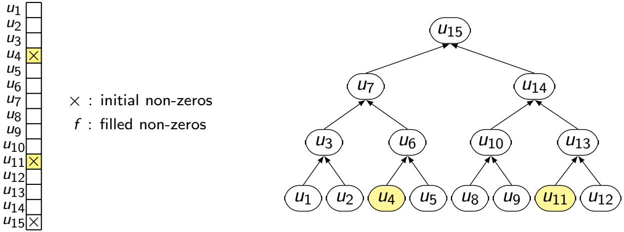
Forward solve phase processes the tree from bottom to top:





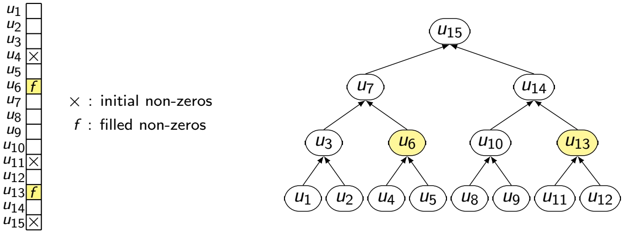
Forward solution process

Forward solve phase processes the tree from bottom to top:



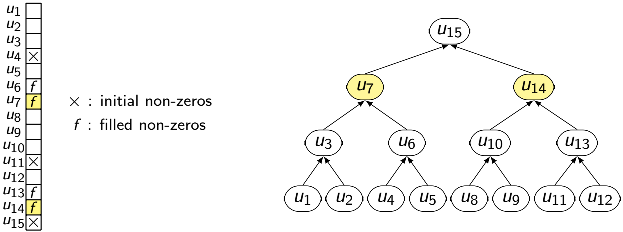
Forward solution process

Forward solve phase processes the tree from bottom to top:



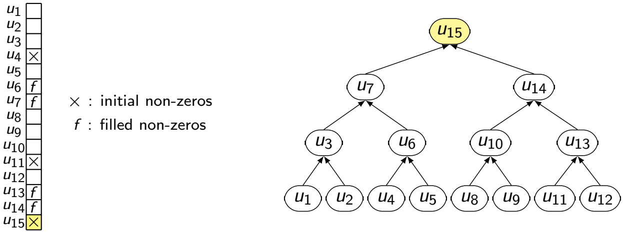
Forward solution process

Forward solve phase processes the tree from bottom to top:



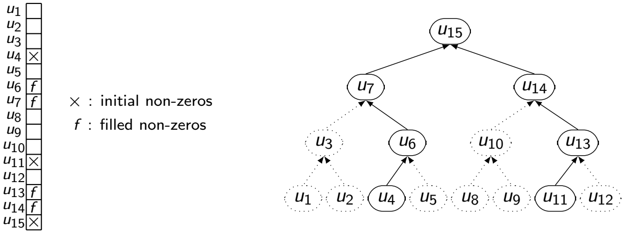
Forward solution process

Forward solve phase processes the tree from bottom to top:



Forward solution process

Forward solve phase processes the tree from bottom to top:



Computation follows paths in the tree T [Gilbert, 1994].

$\hookrightarrow$ **Tree pruning** ($T \rightarrow T_p(b)$) to reduce computation:

$$\Delta = \sum_{u \in T_p(b)} \mathcal{F}_u$$

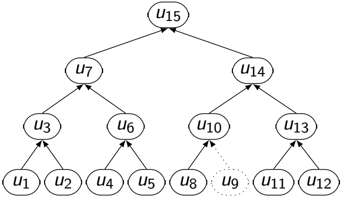
Exposition of padded zeros

When B is a matrix with multiple columns:

- use of BLAS 3 operations for efficiency;
- $T_p(B) = \bigcup T_p(B_i)$, where B_i is column i of B;

nrhs

	1	2	3	4	5	6
u_1	×					
u_2				×		
u_3	f			f		
u_4		×				
u_5					×	
u_6		f			f	
u_7	f	f		f	f	
u_8	×					×
u_9						
u_{10}	f					f
u_{11}			×			
u_{12}					×	
u_{13}			×		f	
u_{14}	f		f		f	f
u_{15}	×	f	f	f	×	f



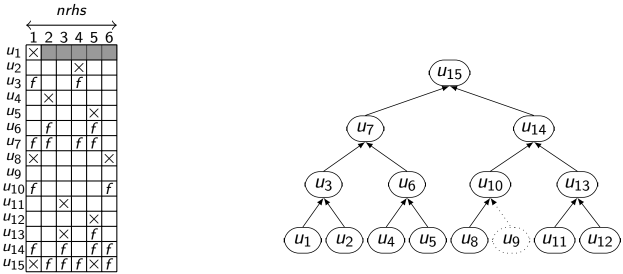
But still, extra computations are done ...

$$\Delta = nrhs \times \sum_{u \in T_p(B)} \mathcal{F}_u$$

Exposition of padded zeros

When B is a matrix with multiple columns:

- use of BLAS 3 operations for efficiency;
- $T_p(B) = \bigcup T_p(B_i)$, where B_i is column i of B;



But still, extra computations are done ...

$$\Delta = nrhs \times \sum_{u \in T_p(B)} \mathcal{F}_u$$

Solutions

What are the possible alternatives ?

- Indirections: rebuilding data structures;
- Sequential: solution phase on each column
  $\Rightarrow$ optimal ($\Delta = \Delta_{min}$) but not efficient;
- Regular blocking: how to build blocks ?
  - minimal access to factors (out of core) [Amestoy et al.,SISC,2012];
  - minimal number of operations (in core) [Yamazaki et al.,2013];
- Exploitation of subintervals of columns at each node [Amestoy et al.,SISC,2015].

Solutions

What are the possible alternatives ?

- Indirections: rebuilding data structures;
- Sequential: solution phase on each column
  $\Rightarrow$ optimal ($\Delta = \Delta_{min}$) but not efficient;
- Regular blocking: how to build blocks ?
  - minimal access to factors (out of core) [Amestoy et al.,SISC,2012];
  - minimal number of operations (in core) [Yamazaki et al.,2013];
- Exploitation of subintervals of columns at each node [Amestoy et al.,SISC,2015].

Let $u \in T$:

Active columns at node u

$$Z_u = \{i \in \{1, \ldots, m\} \mid u \in T_p(B_i)\}$$

Subinterval is given by:

$$\theta_u = \max(Z_u) - \min(Z_u) + 1$$

	1	2	3	4	5	6
u_1	×					
u_2				×		
u_3	f			f		
u_4		×				
u_5					×	
u_6		f			f	
u_7	f	f		f	f	
u_8	×					×
u_9						
u_{10}	f					f
u_{11}			×			
u_{12}					×	
u_{13}			×		f	
u_{14}	f		f		f	f
u_{15}	×	f	f	f	×	f

Let $u \in T$:

Active columns at node u

$$Z_u = \{i \in \{1, \dots, m\} \mid u \in T_p(B_i)\}$$

Subinterval is given by:

$$\theta_u = \max(Z_u) - \min(Z_u) + 1$$

Example: $\theta_{u_1} = 1$, $\theta_{u_{10}} = 6$

	1	2	3	4	5	6
u_1	×					
u_2				×		
u_3	f			f		
u_4		×				
u_5					×	
u_6		f			f	
u_7	f	f		f	f	
u_8	×					×
u_9						
u_{10}	f					f
u_{11}			×			
u_{12}					×	
u_{13}			×		f	
u_{14}	f		f		f	f
u_{15}	×	f	f	f	×	f

Work on node subintervals

Let $u \in T$:

Active columns at node u

$$Z_u = \{i \in \{1, \ldots, m\} \mid u \in T_p(B_i)\}$$

Subinterval is given by:

$$\theta_u = \max(Z_u) - \min(Z_u) + 1$$

Example: $\theta_{u_1} = 1$, $\theta_{u_{10}} = 6$

$$\Delta = \sum_{u \in T_p(B)} \mathcal{F}_u \times \theta_u$$

	1	2	3	4	5	6	
u_1	$\times$						$\theta_1 = 1$
u_2				$\times$			$\theta_2 = 1$
u_3	f			f			$\theta_3 = 4$
u_4		$\times$					$\theta_4 = 1$
u_5					$\times$		$\theta_5 = 1$
u_6		f			f		$\theta_6 = 4$
u_7	f	f		f	f		$\theta_7 = 5$
u_8	$\times$					$\times$	$\theta_8 = 6$
u_9							$\theta_9 = 0$
u_{10}	f					f	$\theta_{10} = 6$
u_{11}			$\times$				$\theta_{11} = 1$
u_{12}					$\times$		$\theta_{12} = 1$
u_{13}			$\times$		f		$\theta_{13} = 3$
u_{14}	f		f		f	f	$\theta_{14} = 6$
u_{15}	$\times$	f	f	f	$\times$	f	$\theta_{15} = 6$

Let $u \in T$:

Active columns at node u

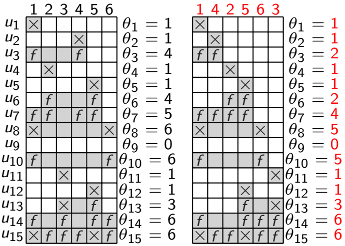
$$Z_u = \{i \in \{1, \dots, m\} \mid u \in T_p(B_i)\}$$

Subinterval is given by:

$$\theta_u = \max(Z_u) - \min(Z_u) + 1$$

Example: $\theta_{u_1} = 1$, $\theta_{u_{10}} = 6$

$$\Delta = \sum_{u \in T_p(B)} \mathcal{F}_u \times \theta_u$$



$\hookrightarrow$ Δ is extremely dependant on column permutation.

Problem statement & algorithms

Goal is to minimize or decrease $\Delta = \sum_{u \in T_p(B)} \mathcal{F}_u \times \theta_u$:

- find permutation σ of columns to decrease $\theta_u, \forall u \in T_p(B)$;
- in case of blocking, minimize the number of blocks.

Problem statement & algorithms

Goal is to minimize or decrease $\Delta = \sum_{u \in T_p(B)} \mathcal{F}_u \times \theta_u$:

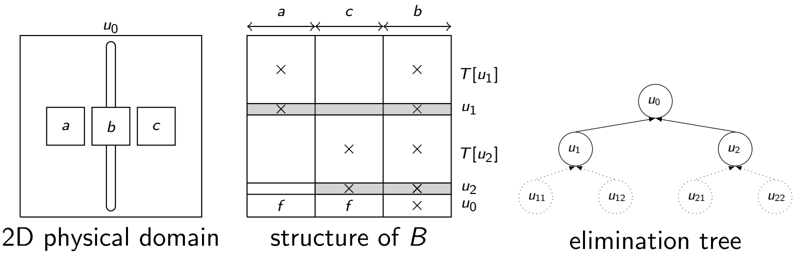
- find permutation σ of columns to decrease $\theta_u, \forall u \in T_p(B)$;
- in case of blocking, minimize the number of blocks.

Proposed heuristics:

- based on geometrical properties (Nested Dissection);
- generalization possible thanks to pruned tree $T_p(B)$.

Flat Tree Algorithm

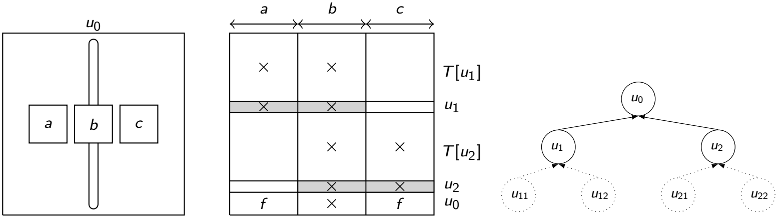
Intuition based on a simple 2D example:



- Nested Dissection $\Rightarrow$ partition right-hand-sides into 3 sets (a, b, c);
- $\theta_{u_1} = a + c + b$

Flat Tree Algorithm

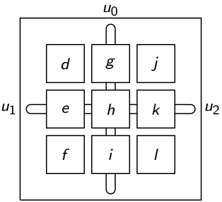
Intuition based on a simple 2D example:



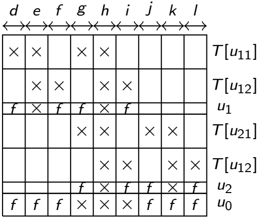
2D physical domain | structure of B | elimination tree

- Nested Dissection $\Rightarrow$ partition right-hand-sides into 3 sets (a, b, c);
- $\theta_{u_1} = a + c + b \Rightarrow \boldsymbol{\theta_{u_1} = a + b}$;

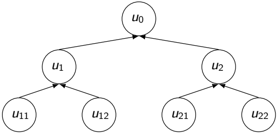
Intuition based on a simple 2D example:



2D physical domain



structure of B



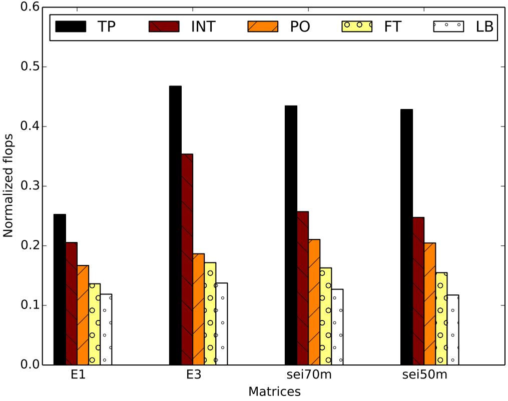
elimination tree

- Nested Dissection $\Rightarrow$ partition right-hand-sides into 3 sets (a, b, c);
- $\theta_{u_1} = a + c + b \Rightarrow \boldsymbol{\theta_{u_1} = a + b}$;

$\hookrightarrow$ **Top-down** approach + **local optimisation** for the nodes at the current layer in the tree.

Results on the Flat Tree

flops: normalized with the dense case; Ordering: Nested Dissection;



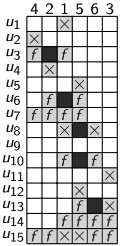
Strategies:

- TP = tree pruning only;
- INT = tree pruning + node interval+natural order;
- PO = tree pruning+node interval+Postorder;
- FT = tree pruning+node interval+Flat Tree;
- LB = Lower Bound (Δ_{min}).

$\hookrightarrow$ Still 28% above the lower bound on one case.

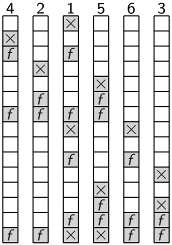
Adapted blocking technique

Objective: decrease Δ with the creation of a minimum number of groups.



Computations on explicit zeros still exist.

Objective: decrease Δ with the creation of a minimum number of groups.

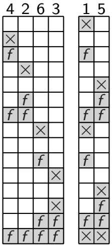


Δ_{min} may be obtain by creating *nrhs* groups:

- however, not performant (loss of BLAS 3 operations);

Adapted blocking technique

Objective: decrease Δ with the creation of a minimum number of groups.

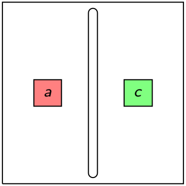


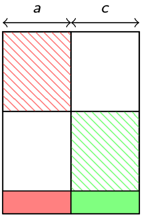
Δ_{min} may be obtain by creating *nrhs* groups:

- however, not performant (loss of BLAS 3 operations);
- need to find some property to group right hand sides together without introducing extra operations.

Adapted blocking technique

Principle (1): group sets of right hand sides that belong to different subdomains (starting with root separator).



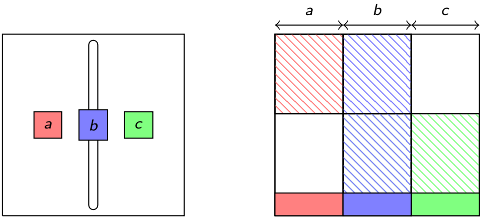


2D Domain structure of B

- non-zero structure of a and c are disjoint;

Adapted blocking technique

Principle (2): extract set of right hand sides that belong to both subdomains (starting with root separator).

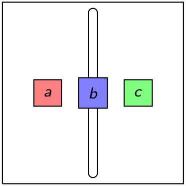


2D Domain   structure of B

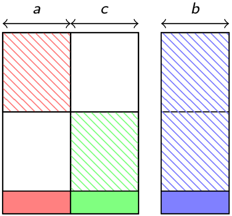
- non-zero structure of a and c are disjoint;
- whereas b may have the non-zero structure of both a and c;

Adapted blocking technique

Principle (2): extract set of right hand sides that belong to both subdomains (starting with root separator).



2D Domain



structure of B

- non-zero structure of a and c are disjoint;
- whereas b may have the non-zero structure of both a and c;
- thus, we extract them.

Comparison with a regular blocking strategy

Our Blocking algorithm (BLK):

- greedy algorithm to choose next group;
- stop condition: $\Delta < \Delta_{tol}$, where $\Delta_{tol} = 1.01\Delta_{min}$.

Regular blocking algorithm (REG):

- split in chunk of regular size;
- stop condition: $\Delta < \Delta_{tol}$, where $\Delta_{tol} = 1.01\Delta_{min}$.

nb groups	5Hz	7Hz	E1	E3
REG	328	255	363	258
BLK	3	3	4	4

Table : Number of groups created for each strategy with a tolerance such that $\Delta < 1.01 \times \Delta_{tol}$.

Conclusion

Achievements:

- implementation of two heuristics (permutation, blocking);
- 90% decrease in flops by exploiting sparsity;
- Up to 40% decrease in time for forward solve w.r.t. INT strategy and Nested Dissection ordering (sequential).

Perspectives:

- adapt the Flat Tree algorithm to unbalanced trees;
- parallelism and sparsity aspects of Flat Tree permutation;
- extend to more general test cases.

Acknowledgements

- LIP laboratory for access to the machines;
- EMGS et SEISCOPE for providing test cases;
- This work was performed within the frameworks of both the MUMPS consortium and the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

?

Thanks!
Questions?