



**HAL**  
open science

## Supernodes ordering to enhance Block Low-Rank compression in sparse direct solvers

Grégoire Pichon, Eric Darve, Mathieu Faverge, Pierre Ramet, Jean Roman

► **To cite this version:**

Grégoire Pichon, Eric Darve, Mathieu Faverge, Pierre Ramet, Jean Roman. Supernodes ordering to enhance Block Low-Rank compression in sparse direct solvers. PMAA 2018 - 10th International Workshop on Parallel Matrix Algorithms and Applications, Jun 2018, Zurich, Switzerland. hal-01956960

**HAL Id: hal-01956960**

**<https://inria.hal.science/hal-01956960v1>**

Submitted on 16 Dec 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



# Supernodes ordering to enhance Block Low-Rank compression in sparse direct solvers

June 28th, 2018 - PMAA

Grégoire Pichon<sup>a</sup>, Eric Darve<sup>b</sup>, Mathieu Faverge<sup>a</sup>, Pierre Ramet<sup>a</sup>, Jean Roman<sup>a</sup>

---

<sup>a</sup>Inria, Bordeaux INP, CNRS, University of Bordeaux

<sup>b</sup>Stanford University

## Context

### PaStiX library

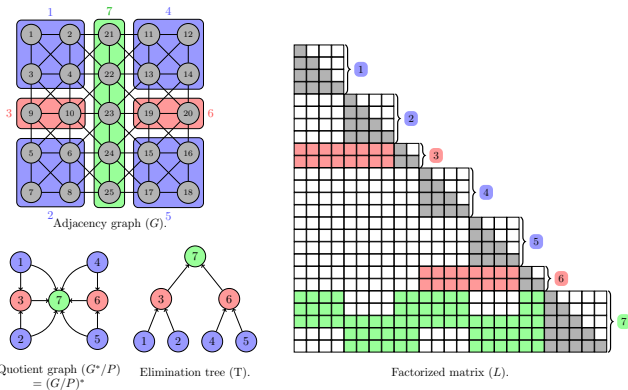
- Sparse direct solver with supernodal method
- Right-looking scheduling strategy
- Block low-rank support:
  - ▶ Large blocks are compressed into a low-rank form
  - ▶ Similar global behavior to the full-rank solver
  - ▶ *Minimal Memory* strategy to save memory
  - ▶ *Just-In-Time* strategy to reduce time-to-solution

Objective of this talk: study low-rank clustering strategies to enhance blocks compressibility

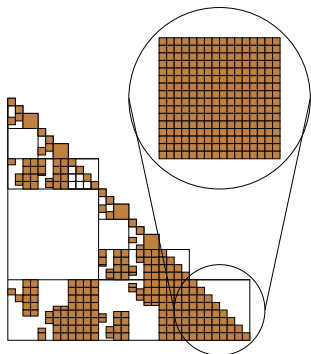
# Symbolic Factorization

## General approach

1. Build a partition with the nested dissection process
2. Compress information on data blocks
3. Compute the block elimination tree using the block quotient graph



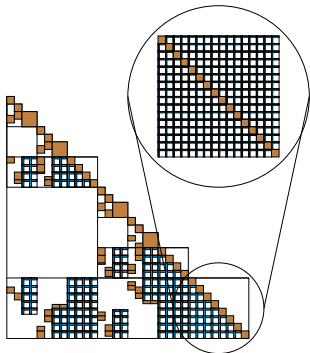
## From full-rank to low-rank



### Full-rank algorithm

- For each column-block / supernode:
  1. **Factorize** the diagonal block (POTRF/GETRF)
  2. **Solve** off-diagonal blocks in the current column (TRSM)
  3. **Update** the trailing matrix (GEMM)

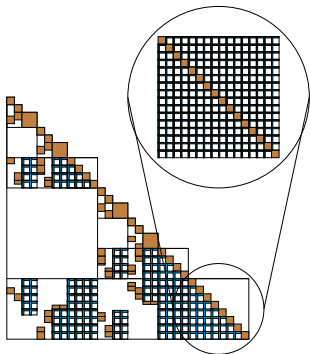
## From full-rank to low-rank



### *Just-In-Time* algorithm

- For each column-block / supernode:
  1. **Factorize** the diagonal block (POTRF/GETRF)
  2. **Compress** off-diagonal blocks
  3. **Solve low-rank** off-diagonal blocks in the current column (TRSM)
  4. **Update** the trailing matrix (LR2FR)
- Reduce the time to solution
- Memory peak is as large as full-rank

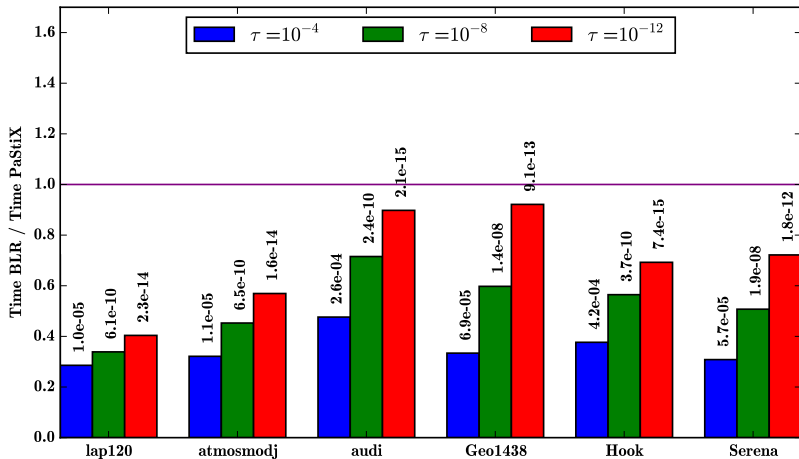
## From full-rank to low-rank



### Minimal Memory algorithm

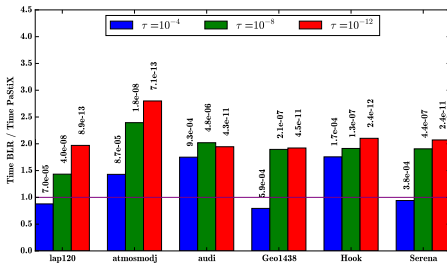
- Compress all off-diagonal blocks
- For each column-block / supernode:
  1. **Factorize** the diagonal block (POTRF/GETRF)
  2. **Solve** low-rank off-diagonal blocks in the current column (TRSM)
  3. **Update** the trailing matrix (LR2LR)
- Reduce the memory peak of the solver
- LR2LR updates may be too costly for *small* cases

## Performance of RRQR/*Just-In-Time*

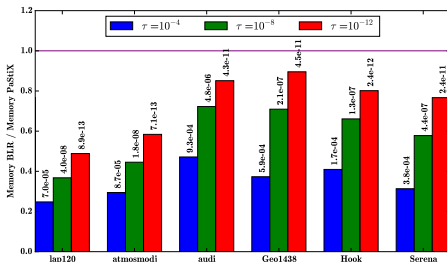




# Performance of RRQR/Minimal Memory

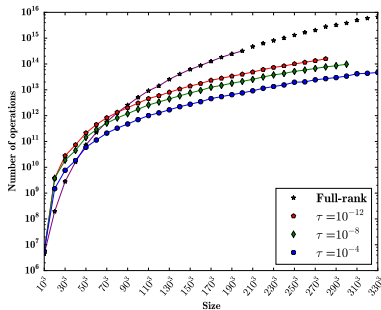
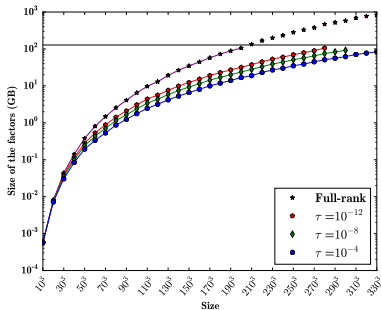


Factorization time



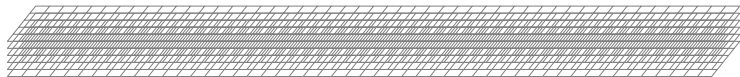
Memory footprint

# Scalability of *Minimal Memory* strategy on Laplacian matrices



- 128GB of memory
- Up to 8M unknowns in full-rank
- Up to 36M unknowns with  $\tau = 10^{-4}$

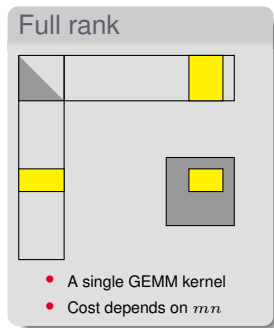
## Up to what limit?



- 96 cores machine with 3TB of memory
- $100 \times 100 \times 100K$  mesh
- Full-rank would have required 11TB of memory
- With  $\tau = 10^{-4}$ :
  - ▶ Only 2TB of memory for the coefficients
  - ▶ Less than 6 hours to factorize
  - ▶  $10^{-8}$  solution with 15 iterations of iterative refinement

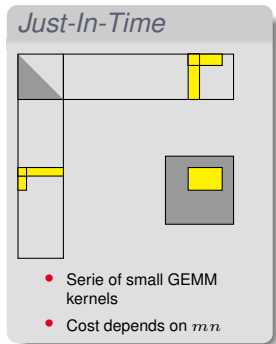
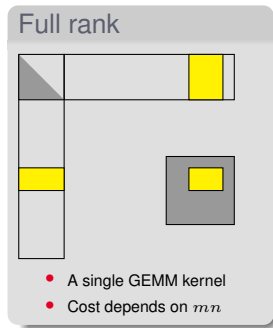
## Summary of the updates

Cost of applying a  $m \times n$  update to a  $M \times N$  block.



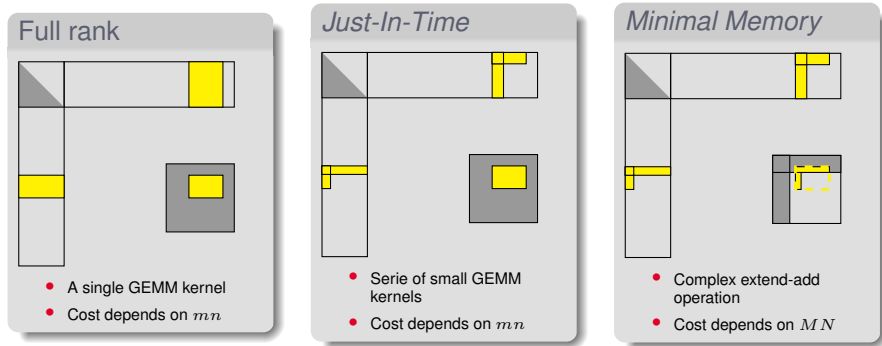
## Summary of the updates

Cost of applying a  $m \times n$  update to a  $M \times N$  block.



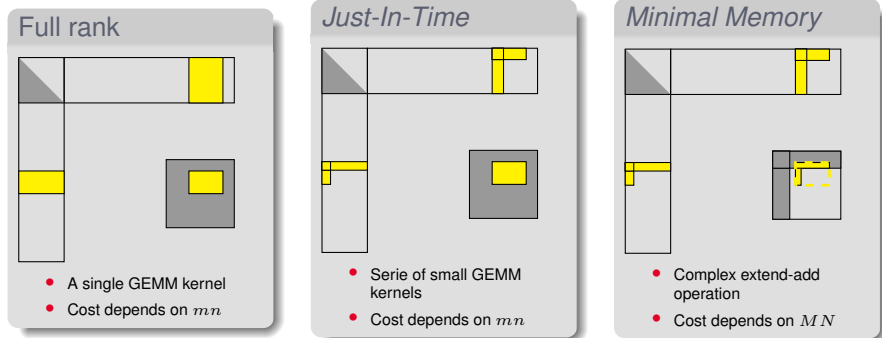
## Summary of the updates

Cost of applying a  $m \times n$  update to a  $M \times N$  block.



## Summary of the updates

Cost of applying a  $m \times n$  update to a  $M \times N$  block.



Can we reduce the number of those problematic updates?

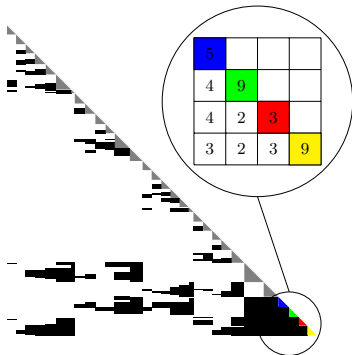
# 1

## Block Low-Rank clustering techniques

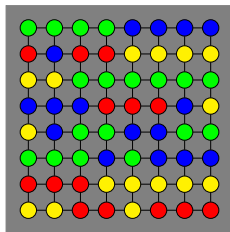


## Clustering techniques: existing solutions

- Supernode reordering techniques (supernodal)



(a) Symbolic factorization

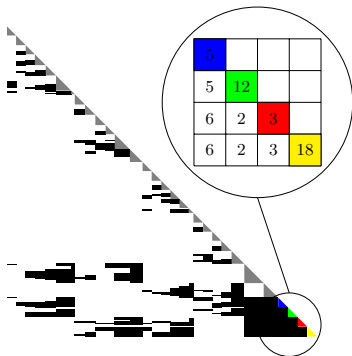


(b) First separator clustering

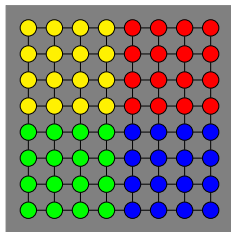
$8 \times 8 \times 8$  Laplacian partitioned using SCOTCH and Reordering clustering on the first separator

## Clustering techniques: existing solutions

- Supernode reordering techniques (supernodal)
- k-way partitioning (dense, multifrontal)



(a) Symbolic factorization



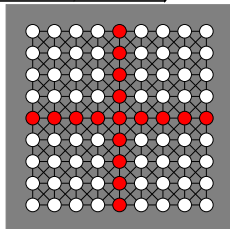
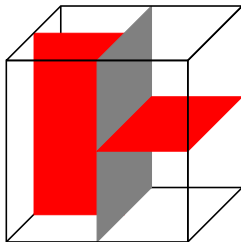
(b) First separator clustering

$8 \times 8 \times 8$  Laplacian partitioned using SCOTCH and k-way clustering on the first separator

## How to take benefit from both algorithms?

Pre-selection to detect uncompressible vertices

- Pre-select unknowns that represent close interactions and will not be compressible
- Idea of Boundary Distance Low-Rank (BDLR) introduced by Darve et al.
- For instance, pre-select interactions between a father and its direct children
- Hypothesis similar to  $ILU(k)$  factorizations



## How to take benefit from both algorithms?

*Hierarchical Interpolative Factorization for Elliptic Operators: Differential Equations*, Kenneth L. Ho and Lexing Ying, 2014.

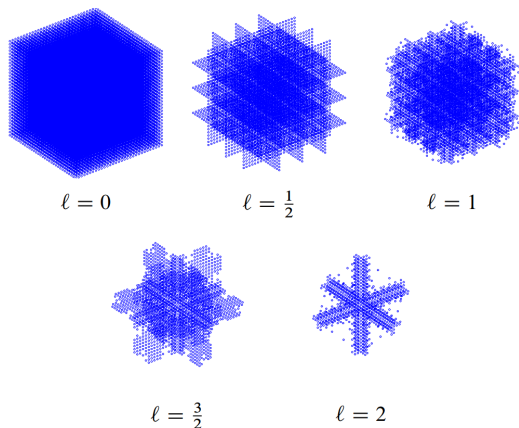


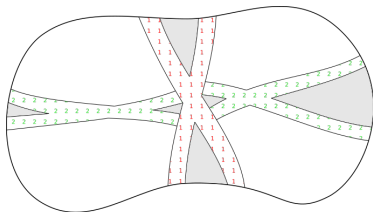
FIGURE 4.2. Active DOFs at each level  $\ell$  of HIF-DE in 3D.





## New heuristic based on projections

ComputeProjections(*nblvl*, *distp*, *width*)

Lets consider a supernode and the *nblvl* levels below in the elimination tree, with the constraint that nodes are issued from nested dissection, and not from minimum-fill as it happens at the bottom of the elimination tree.

- Each vertex of the separator being clustered is pre-selected iff it is a neighbor to one of the children at a distance *distp*
- The trace is then enlarged with neighbors within the separator at a distance *width*, similarly to BDLR.



-  Trace of the 1st separator
-  Trace of the 2nd separator
-  #nodes > threshold
-  #nodes < threshold

## New heuristic based on projections

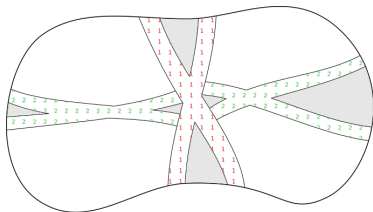
ComputeProjections(*nblvl*, *distp*, *width*)

Lets consider a supernode and the *nblvl* levels below in the elimination tree, with the constraint that nodes are issued from nested dissection, and not from minimum-fill as it happens at the bottom of the elimination tree.

- Each vertex of the separator being clustered is pre-selected iff it is a neighbor to one of the children at a distance *distp*
- The trace is then enlarged with neighbors within the separator at a distance *width*, similarly to BDLR.

How to control the number of pre-selected vertices?

- Ensure that this number is low wrt to the total number of vertices of a separator:  $\#selected < \alpha\sqrt{n}$
- Ensure that it creates at least two non-connected components



- Trace of the 1st separator
- Trace of the 2nd separator
- #nodes > threshold
- #nodes < threshold

## Proposed algorithm for the clustering of supernodes

---

### Algorithm 1 Cluster a supernode

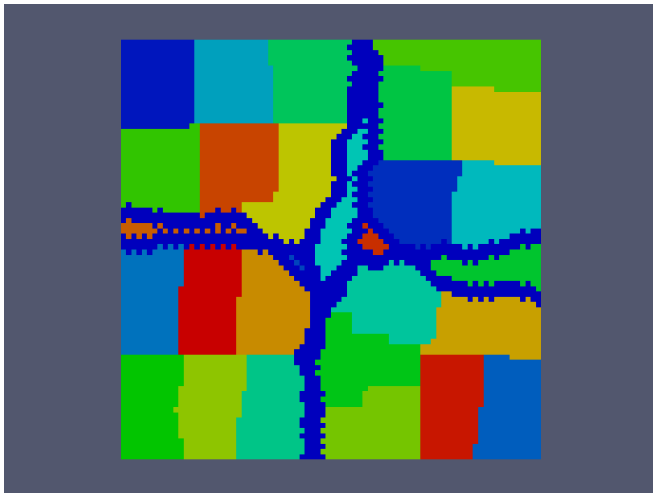
---

```
1: ConnectSupernode( dist )
2: ComputeProjections( levels, distp, width )
3: IsolateConnectedComponents()
4: For each connected component  $C$  Do
5:   If  $|C| < threshold$  Then
6:     Merge  $C$  into pre-selected vertices
7:   Else
8:     Kway()
9:     Reordering()
10:  End If
11: End For
12: Reordering() of pre-selected vertices
```

---

Projections are performed on separators of size larger than  $16 \times MAX\_BLOCKSIZE$ , to obtain at least four large connected subparts that will be split into four parts using kway.

## Example - $80^3$ Laplacian matrix





# 2

## Numerical Experiments

## Experimental setup

### Architecture

- 2 INTEL Xeon E5 – 2680v3 at 2.50 GHz (24 threads)
- 128 GB

### Matrices

- 33 square matrices from the SuiteSparse Matrix Collection
- real or complex
- $50K \leq N \leq 5M$
- $nb_{ops} > 1TFlops$

### Solver

- PASTIX solver with SCOTCH ordering
- Static scheduling

## Parameters

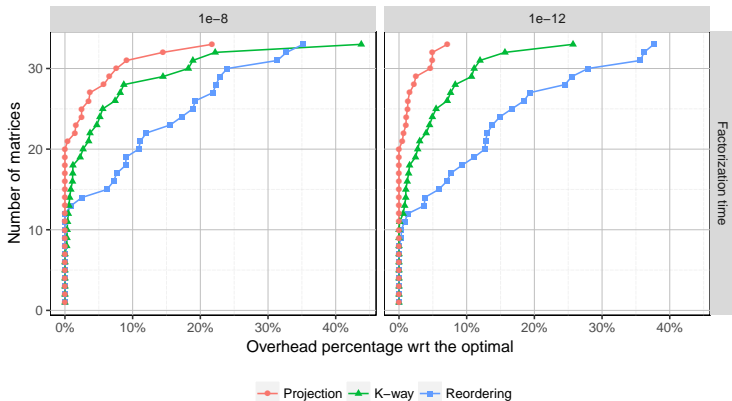
### BLR solver

- Tolerance  $\tau$ : absolute parameter (normalized for each block)
- Compression method is RRQR
- Compression strategy: *Minimal Memory* or *Just-In-Time*
- Blocking sizes: between 128 and 256 in following experiments

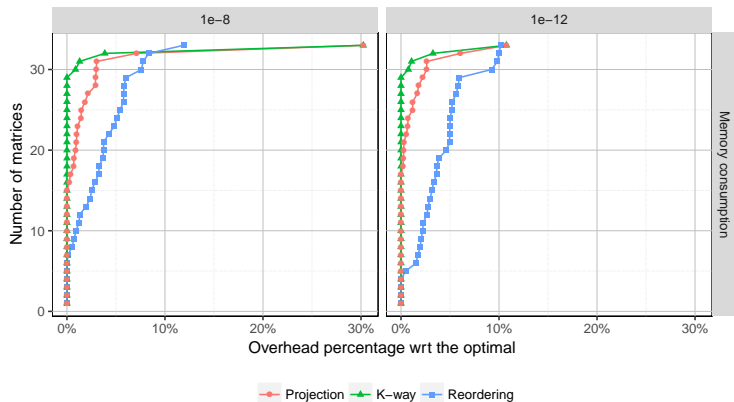
### Pre-selection

- Graphs are reconnected using distance 1 in the Halo
- Selected vertices are obtained with `ComputeProjections( 3, 1, 1 )`
- Projections are performed on supernodes of size larger than 16 times the blocking size parameters.
- K-way is configured to get partition fitting the blocking size parameters
- Connected components smaller than the minimal size for low-rank compression are merged into the pre-selected vertices set.

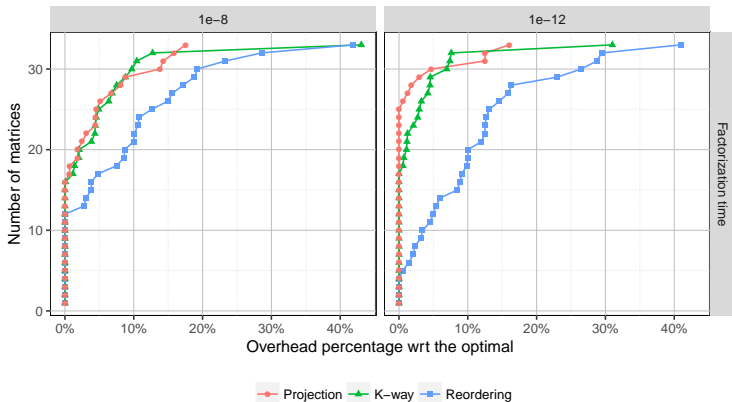
# Performance profile of the factorization time – *Minimal Memory*



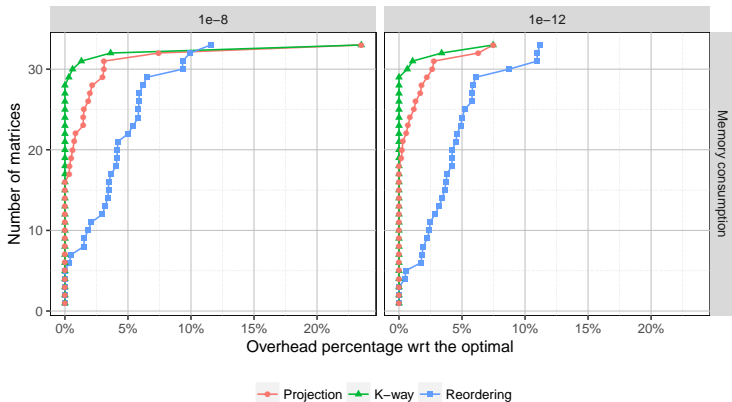
# Performance profile of the memory consumption – *Minimal Memory*



# Performance profile of the factorization time – *Just-In-Time*



# Performance profile of the memory consumption – *Just-In-Time*



# Conclusion

## Ordering strategies

- Both reordering and k-way are limited to form suitable clusters
  - ▶ reordering does not allow a good compression of separators
  - ▶ k-way compresses well the separators, but does not take into account updates
- Pre-selection of some unknowns allows to obtain well-separated clusters and to exhibit non-compressible operations
- We demonstrated it is suitable for reducing time-to-solution with only a small memory consumption increase



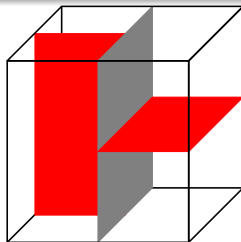
# Conclusion

## Ordering strategies

- Both reordering and k-way are limited to form suitable clusters
  - ▶ reordering does not allow a good compression of separators
  - ▶ k-way compresses well the separators, but does not take into account updates
- Pre-selection of some unknowns allows to obtain well-separated clusters and to exhibit non-compressible operations
- We demonstrated it is suitable for reducing time-to-solution with only a small memory consumption increase

## Future works

- Align separators to form clusters during the nested dissection process
- Preliminary work using fixed vertices



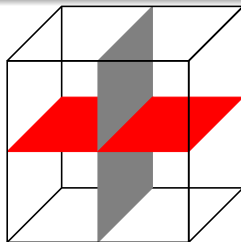
# Conclusion

## Ordering strategies

- Both reordering and k-way are limited to form suitable clusters
  - ▶ reordering does not allow a good compression of separators
  - ▶ k-way compresses well the separators, but does not take into account updates
- Pre-selection of some unknowns allows to obtain well-separated clusters and to exhibit non-compressible operations
- We demonstrated it is suitable for reducing time-to-solution with only a small memory consumption increase

## Future works

- Align separators to form clusters during the nested dissection process
- Preliminary work using fixed vertices



## PASTIX 6.1.0

<http://gitlab.inria.fr/solverstack/pastix>

- Support shared memory with different schedulers:
  - ▶ sequential
  - ▶ static scheduler
  - ▶ PARSEC/STARPU runtime systems with experimental GPU support
- Low-rank support
- Cholesky, *LDL* and *LU* factorizations
- GMRES, CG, BiCG iterative refinements

Thank you.