



HAL
open science

Sparse supernodal solver using block low-rank compression: Design, performance and analysis

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

► **To cite this version:**

Grégoire Pichon, Eric Darve, Mathieu Faverge, Esragul Korkmaz, Pierre Ramet, et al.. Sparse supernodal solver using block low-rank compression: Design, performance and analysis. JOREK development meeting, Nov 2019, Cadarache, France. hal-02326407

HAL Id: hal-02326407

<https://inria.hal.science/hal-02326407>

Submitted on 22 Oct 2019

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.



université
de BORDEAUX



Sparse supernodal solver using block low-rank compression: Design, performance and analysis

November 21th, 2019 - JOREK development meeting - Cadarache

G. Pichon^a, E. Darve^b, M. Faverge^c, E. Korkmaz^c, P. Ramet^c, and J. Roman^c

^aUniversité Lyon 1

^bStanford University

^cInria, Bordeaux INP, CNRS, Université de Bordeaux

Context - PhD of Gregoire Pichon and Esragul Korkmaz

Current sparse direct solvers for 3D problems of size n

- $\Theta(n^2)$ time complexity
- $\Theta(n^{\frac{4}{3}})$ memory complexity
- BLAS Level 3 operations with computations on blocks

Low-rank compression in sparse direct solvers

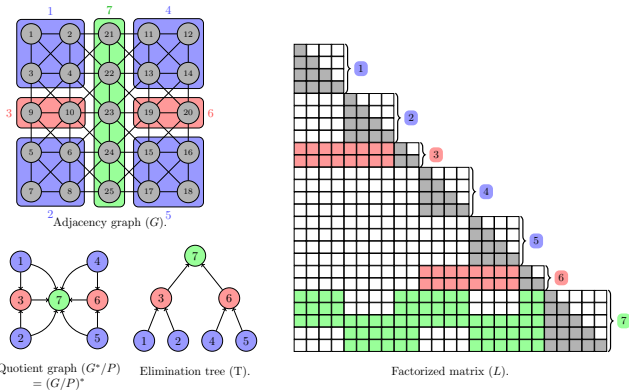
- Compress diagonal and off-diagonal blocks
- Use different compression formats: BLR, \mathcal{H} , HODLR, HSS, \mathcal{H}^2 ..
- Recently introduced in many solvers: MUMPS, PASTIX, STRUMPACK and many others

Objective of this talk: study low-rank clustering strategies to enhance blocks compressibility in the Block Low-Rank (BLR) case

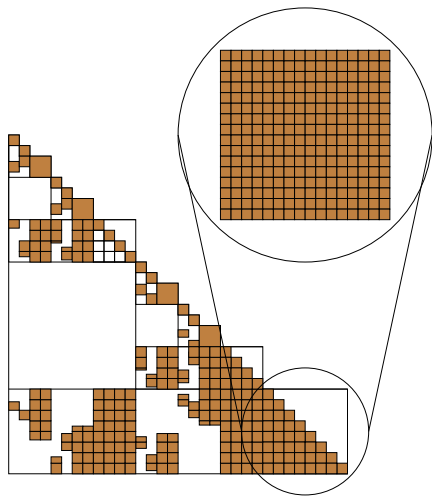
Symbolic Factorization

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



BLR compression – Symbolic factorization



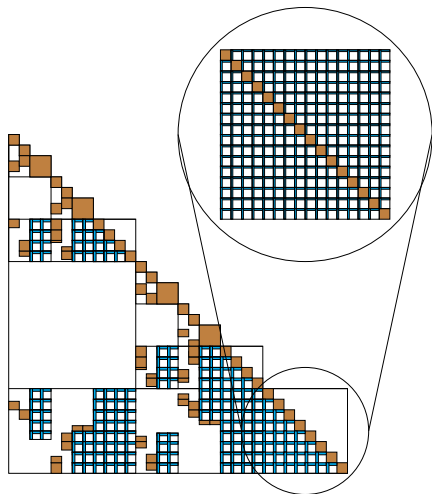
Approach

- Large supernodes are split
- It increases the level of parallelism

Operations

- Dense diagonal blocks
- TRSM are performed on dense off-diagonal blocks
- GEMM are performed between dense off-diagonal blocks

BLR compression – Symbolic factorization



Approach

- Large supernodes are split
- Large off-diagonal blocks are **low-rank**

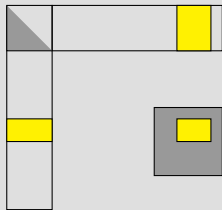
Operations

- Dense diagonal blocks
- TRSM are performed on **low-rank** off-diagonal blocks
- GEMM are performed between **low-rank** off-diagonal blocks

Summary of the possible updates

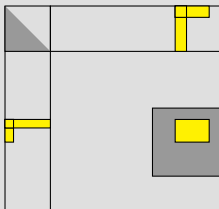
Cost of applying a $m \times n$ update to a $M \times N$ block.

Full rank



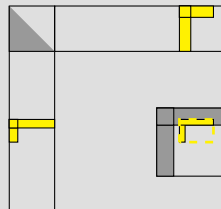
- A single GEMM kernel
- Cost depends on mn

Just-In-Time (non-fully structured)



- Serie of small GEMM kernels
- Cost depends on mn

Minimal Memory (fully structured)



- Complex extend-add operation
- Cost depends on MN

Can we reduce the number small updates, especially for fully structured case?

Compression kernels

Kernel	Complexity
Singular Value Decomposition (SVD)	$\Theta(mn^2)$
Rank-Revealing QR (RRQR)	$\Theta(mnr)$
RRQR with randomization	$\Theta(mnr)$
ACA, BDLR, CUR	$\Theta((m+n)r)$

Properties

- SVD provides the best ranks at a given accuracy with $\|\cdot\|_2$
- RRQR keeps a control of accuracy, but efficiency is poor due to pivoting
- Randomization techniques are suitable to perform a rank- r approximation but may be costly for computing an accurate representation
- The accuracy of ACA/BDLR/CUR is problem dependent

Experimental setup

Machine: 2 INTEL Xeon E5 – 2680v3 at 2.50 GHz

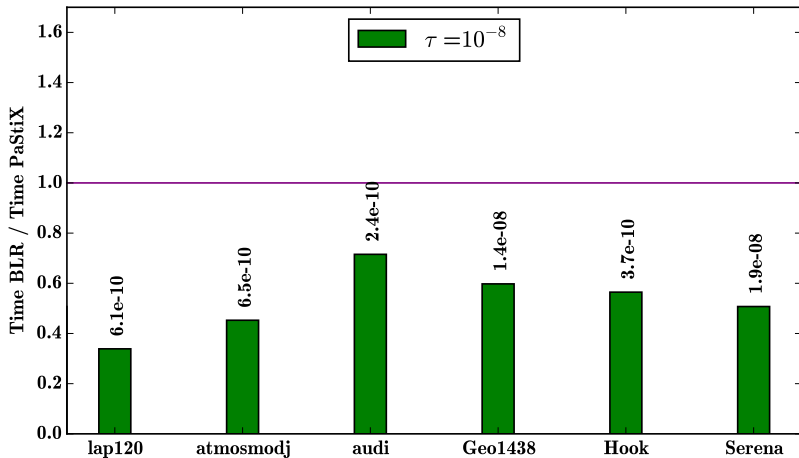
- 128 GB
- 24 threads
- Parallelism is obtained following PASTIX static scheduling for multi-threaded architectures (PARSEC version will be discussed later)

Entry parameters

- Tolerance τ : absolute parameter (normalized for each block)
- Compression method is RRQR
- Blocking sizes: between 128 and 256 in following experiments

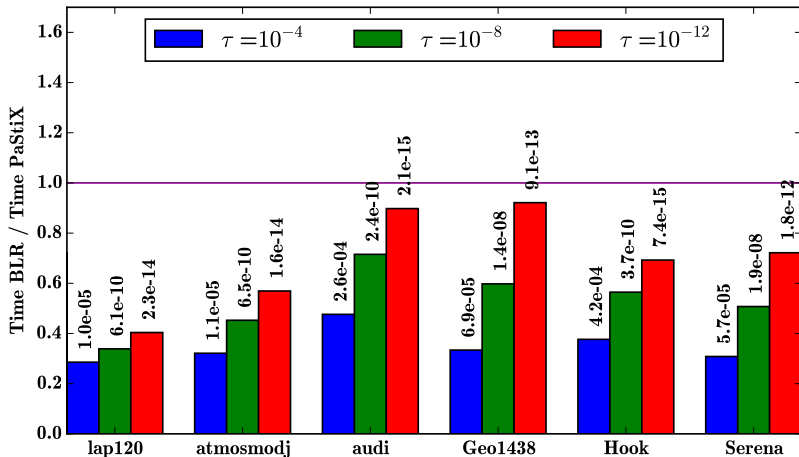
Matrix	Fact.	Size	Field
lap120	LDL^t	120 ³	Poisson problem
Atmosmodj	LU	1 270 432	atmospheric model
Audi	LL^t	943 695	structural problem
Geo1438	LL^t	1 437 960	geomechanical model of earth
Hook	LDL^t	1 498 023	model of a steel hook
Serena	LDL^t	1 391 349	gas reservoir simulation

Performance of RRQR/Just-In-Time wrt full-rank version



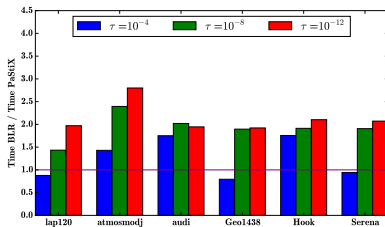
Factorization time reduced by a factor of 2 for $\tau = 10^{-8}$

Performance of RRQR/Just-In-Time wrt full-rank version



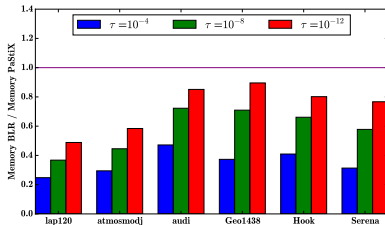
Factorization time reduced by a factor of 2 for $\tau = 10^{-8}$

Behavior of RRQR/Minimal Memory wrt full-rank version



Performance

- Increase by a factor of 1.9 for $\tau = 10^{-8}$
- Better for a lower accuracy



Memory peak

- Reduction by a factor of 1.7 for $\tau = 10^{-8}$
- Close to the results obtained using SVD

Related work

BLR

MUMPS BLR compression in a multifrontal solver with dense assembly, which can perform pivoting

Hierarchical

H-LIB Hierarchical matrices for sparse by Hackbusch et al. which does not exploit all structural zeroes

Hmat-OSS A. Falco's thesis that uses hierarchical matrices together with a symbolic factorization

CHOLMOD Supernodal solver using randomization with fixed rank

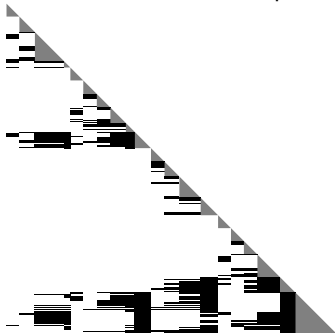
STRUMPACK HSS by Ghysels et al. with randomized sampling

Many other solvers

- Weak admissibility: HODLR, HSS
- Nested bases: HSS, \mathcal{H}^2
- Often used as preconditioners

Ordering for sparse matrices

One can reorder separators without modifying the fill-in



Symbolic factorization of a $8 \times 8 \times 8$ Laplacian

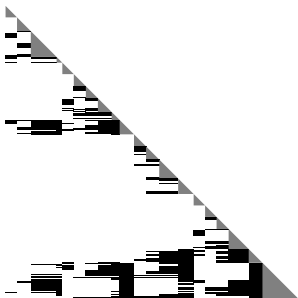
Enhance ordering

- Modify the partitioning process to enhance compressibility
- Reorder unknowns within a separator:
 - ▶ To enhance clustering within the separator
 - ▶ To enhance the coupling part

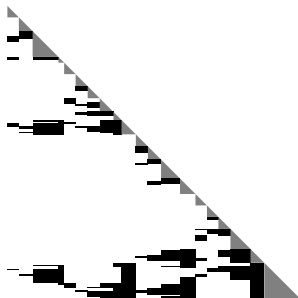
Reordering to minimize the number of off-diagonal blocks

Proposition

- Ordering of separators to minimize the number of off-diagonal blocks
- Does not impact memory consumption or the number of operations
- Increase granularity
 - ▶ Reduce the number of low-rank updates
 - ▶ Enhance the use of heterogeneous architectures (GPUs, Xeon Phi)



Without reordering (RCM)



With reordering

Reordering results

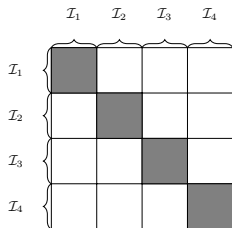
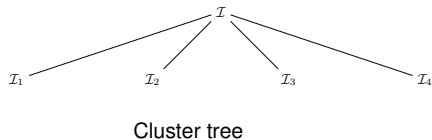
Summary

- Reduce the number of off-diagonal blocks and thus the overhead associated with low-rank updates
- Lead to larger data blocks suitable for modern architectures
- Always increase performance wrt SCOTCH

Architecture	Nb. units	Mean gain	Max gain
Westmere	12 cores	2%	6%
Xeon E5	24 cores	7%	13%
Fermi	12 cores + 1 to 3 M2070	10%	20%
Kepler	24 cores + 1 to 4 K40	15%	40%
Xeon Phi	64 cores	20%	40%

Performance gain for the full-rank factorization when using PARSEC runtime system with TSP instead of SCOTCH

BLR clustering for a dense matrix



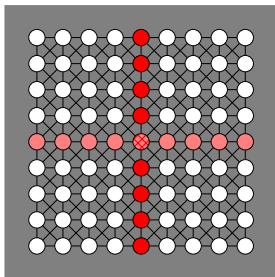
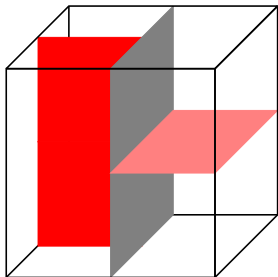
$$\sigma \times \tau \text{ is admissible} \iff \max(\text{diam}(\sigma), \text{diam}(\tau)) \leq \eta \text{ dist}(\sigma, \tau)$$

Can be computed with the knowledge of geometry; otherwise, k-way partitioning can be used for clustering separators

Towards sparse ordering dedicated to low-rank

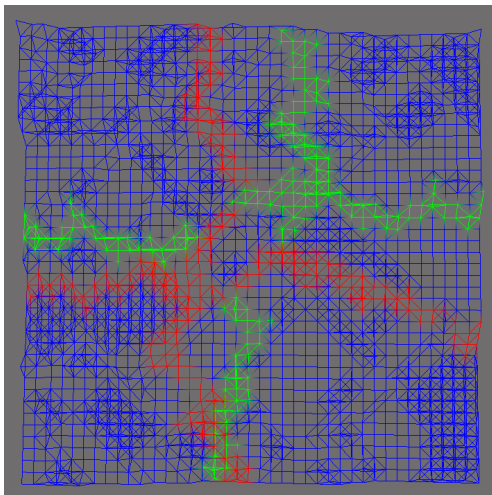
Nested dissection

1. Partition $V = A \cup B \cup C$
2. Order C with larger indices: $V_A < V_C$ and $V_B < V_C$
3. Apply the process recursively on A and B



As A and B are processed independently, their interaction with C is not the same, even for a perfect nested dissection of a $N \times N \times N$ cube

Example on a cube partitioned with SCOTCH



First separator of a $40 \times 40 \times 40$ Laplacian partitioned with SCOTCH

Suitable clustering strategy

Let us consider the last separator A_{22} and its interactions with previous supernodes A_{11} :

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

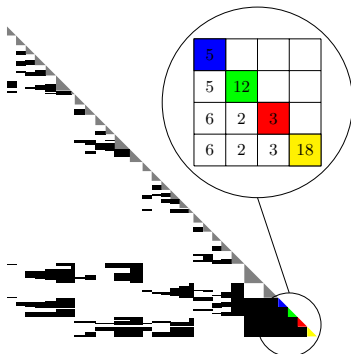
Existing approaches

- K-way partitioning orders correctly A_{22}
- The reordering strategy reduces the number of off-diagonal blocks in A_{12} and A_{21}

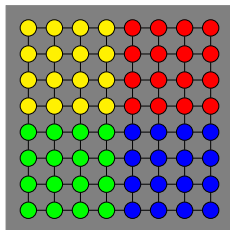
We introduce a new strategy to couple the advantages of both approaches, with suitable structure for both intra and inter separators blocks

Clustering techniques: existing solutions

- k-way partitioning (dense, multifrontal)



(a) Symbolic factorization

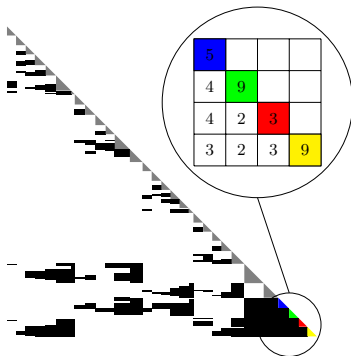


(b) First separator clustering

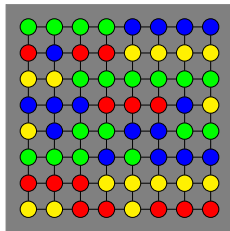
$8 \times 8 \times 8$ Laplacian partitioned using SCOTCH and k-way clustering on the first separator

Clustering techniques: existing solutions

- k-way partitioning (dense, multifrontal)
- 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

New heuristic based on projections

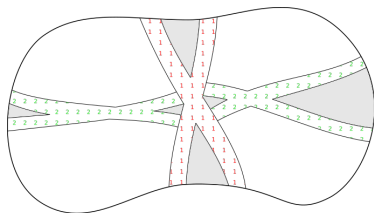
Pre-selected vertices of a separator are vertices connected to a close child in the elimination tree

For each separator we have:

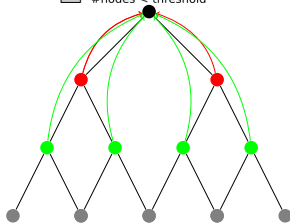
$$A_{22} = \begin{pmatrix} A_{kk} & A_{ks} \\ A_{sk} & A_{ss} \end{pmatrix}$$

where A_{ss} corresponds to pre-selected vertices.

Only interactions between non pre-selected vertices A_{kk} are compressed

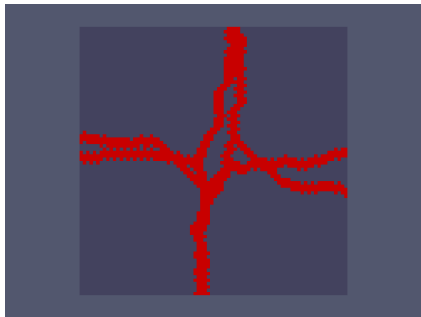


- Trace of the 1st separator
- Trace of the 2nd separator
- #nodes > threshold
- #nodes < threshold



Example - $80 \times 80 \times 80$ Laplacian matrix

Interactions with pre-selected vertices (in red) are not compressed

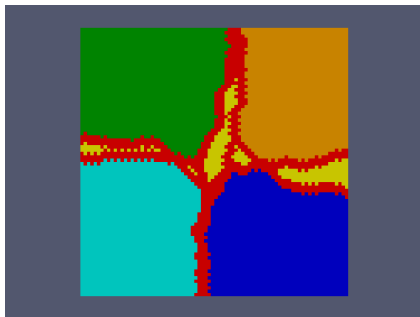


1. Compute pre-selected vertices

Projections heuristic on last separator of size 80×80

Example - $80 \times 80 \times 80$ Laplacian matrix

Interactions with pre-selected vertices (in red) are not compressed

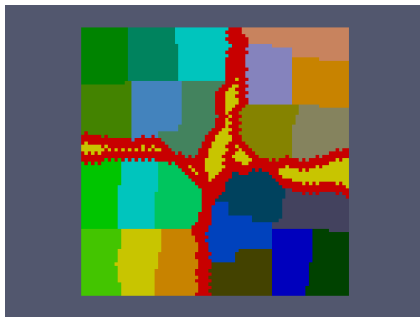


Projections heuristic on last separator of size
 80×80

1. Compute pre-selected vertices
2. Isolate connected components
3. Merge small components

Example - $80 \times 80 \times 80$ Laplacian matrix

Interactions with pre-selected vertices (in red) are not compressed



Projections heuristic on last separator of size 80×80

1. Compute pre-selected vertices
2. Isolate connected components
3. Merge small components
4. Apply k-way partitioning on large components
5. Perform TSP reordering on each cluster

Experimental setup

Architecture

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

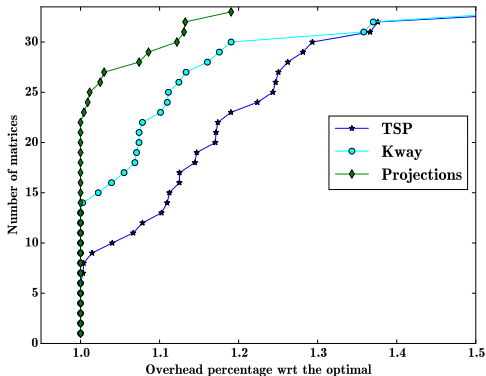
Set of matrices

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

Strategies studied for the PaStiX solver

- Reordering with TSP
- K-way partitioning using SCOTCH + TSP on each cluster
- The new Projections strategy + TSP on each cluster

Performance profile of the factorization time

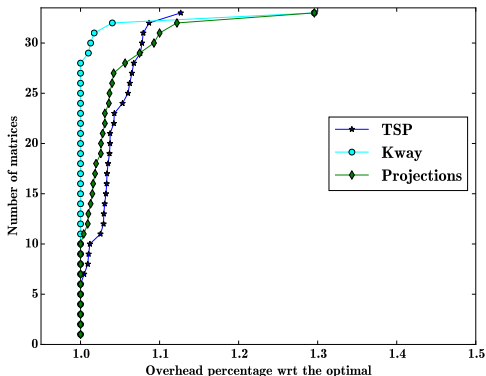


Method

- For each matrix, percentage wrt the best method
- Percentage are sorted increasingly
- A curve close to $x = 1$ is close to optimal

Minimal Memory (fully structured with low-rank updates), with $\tau = 10^{-8}$

Performance profile of the memory consumption



Method

- For each matrix, percentage wrt the best method
- Percentage are sorted increasingly
- A curve close to $x = 1$ is close to optimal

Minimal Memory (fully structured with low-rank updates), with $\tau = 10^{-8}$

Clustering summary

Ordering strategies

- Both reordering and k-way are limited to form suitable clusters
- Pre-selection of some unknowns allows to obtain well-separated clusters and to exhibit non-compressible operations

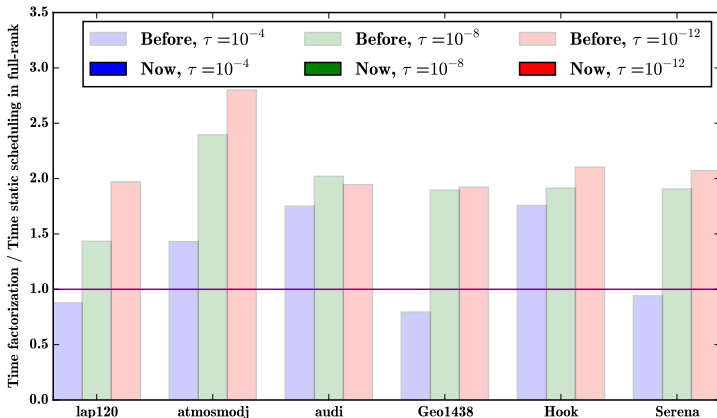
Impact on preprocessing stats

- Ordering time controlled: 70% overhead with respect to SCOTCH ordering
- Slight increase in the number of off-diagonal blocks: mean 1.05, max 1.35 for both **K-way** and **Projections** strategies

Conclusion

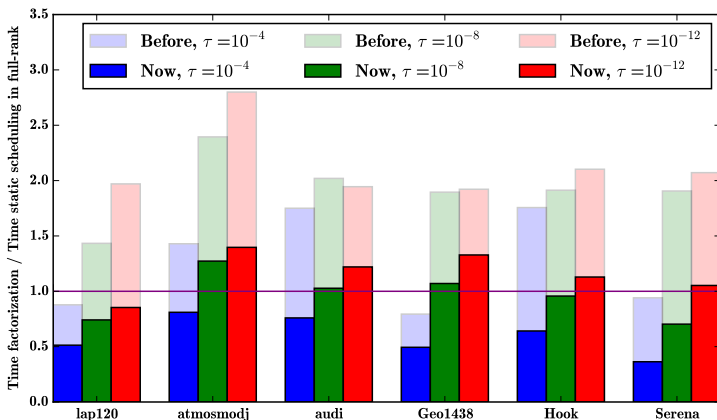
- Somewhat similar to ILU(k) to select more important data
- When moving to hierarchical matrices, the cluster trees may not be binary trees

Behavior of RRQR/Minimal Memory with PARSEC and Projections



Adding Projections and the use of PARSEC runtime system reduces factorization time

Behavior of RRQR/Minimal Memory with PARSEC and Projections



Adding Projections and the use of PARSEC runtime system reduces factorization time

HORSE: Practical application (S. Lanteri - ANR TECSER)

Method

- Introduce an hybrid variable to solve a system smaller than the one obtained through DG
- Perform Domain-Decomposition to solve this system
- Use a direct solver for subdomains

Solve 3D frequency-domain Maxwell's equations

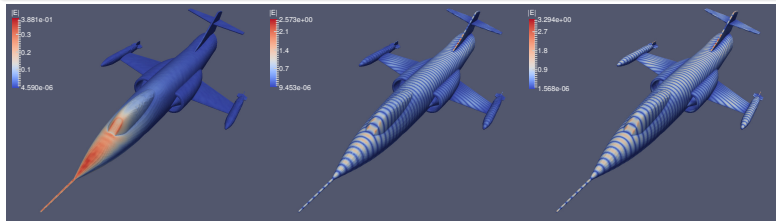
- $i\omega\epsilon_r\mathbf{E} - \mathbf{curl}\mathbf{H} = -\mathbf{J}$, in Ω
- $i\omega\mu_r\mathbf{H} + \mathbf{curl}\mathbf{E} = 0$, in Ω
- $\mathbf{n} \times \mathbf{E} = 0$, on Γ_m
- $\mathbf{n} \times \mathbf{E} + \mathbf{n} \times (\mathbf{n} \times \mathbf{H}) = \mathbf{n} \times \mathbf{E}^{inc} + \mathbf{n} \times (\mathbf{n} \times \mathbf{H}^{inc})$, on Γ_a

HORSE: Hybridizable DG method in 3D

Scattering of a plane wave by a jet

- Unstructured tetrahedral mesh: 1,645,874 elements / 3,521,251 faces
- Frequency: 600 MHz, Wavelength: $\lambda \simeq 0.5$ m, Penalty parameter: $\tau = 1$

HDG method	# DoF Λ field	# DoF EM field
HDG- \mathbb{P}_1	21,127,506	39,500,976
HDG- \mathbb{P}_2	42,255,012	98,752,440
HDG- \mathbb{P}_3	70,425,020	197,504,880



Contour line of $|\mathbf{E}|$ - HDG- \mathbb{P}_1 to HDG- \mathbb{P}_3

Experimental setup

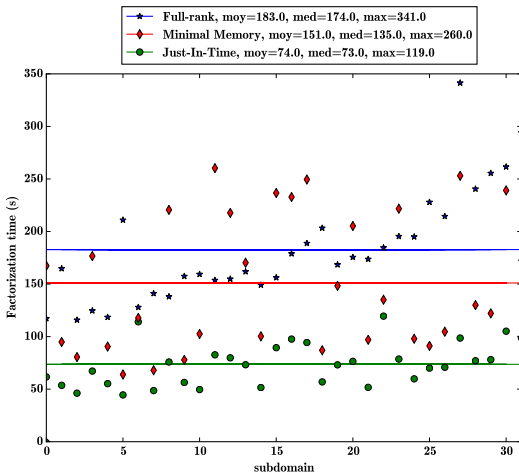
Study case

- Jet with \mathbb{P}_2 interpolation
- 42 255 012 unknowns for Λ
- A single factorization is performed on each subdomain
- There may be several solves for iterative refinement between subdomains

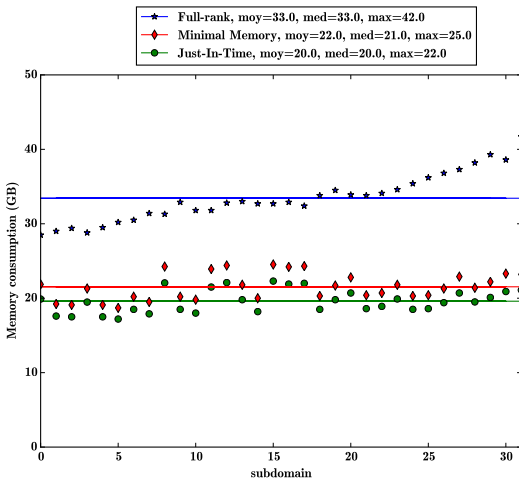
Machine

- Occigen: 24-cores nodes with 128 GB
- Each MPI process is hold on a socket with 12 cores
- In next experiments, we use 32, 48 or 64 subdomains
- Subdomains are ordered accordingly to the number of operations for the complete factorization

Factorization on each subdomain – 32 processes with $\tau = 1e-4$



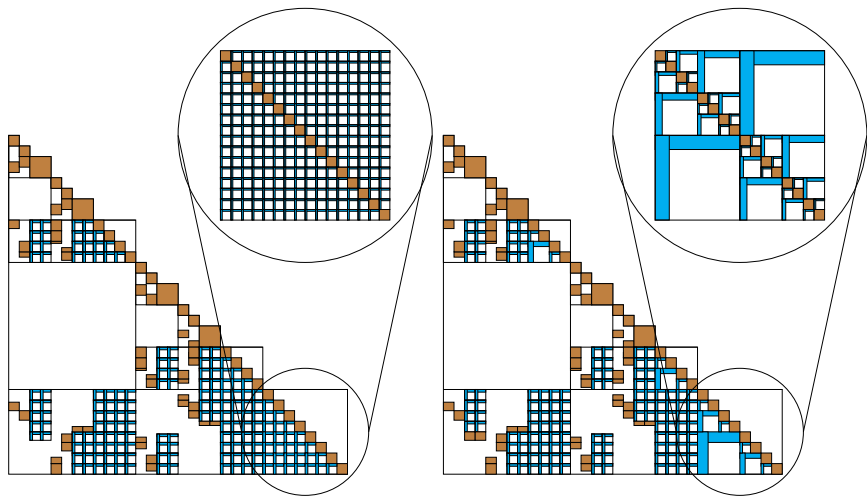
Memory Consumption on each subdomain – 32 processes with $\tau = 1e-4$



Impact on HORSE

Subs	τ	Method	Fact(s)	Nb. of iters	Refinement (s)	HDGM (s)	Memory (GB)
32	-	Full-rank	526.0	9	254.5	814.0	41.7
	1e-4	Just-In-Time	209.3	9	164.8	405.8	41.7 (22.3)
		Minimal-Memory	516.7	15	273.2	822.0	24.5
	1e-8	Just-In-Time	325.2	9	192.9	550.4	41.7 (29.4)
		Minimal-Memory	600.1	9	193.2	825.8	30.5
	48	-	Full-rank	237.3	8	118.6	374.6
1e-4		Just-In-Time	112.2	9	106.1	237.1	24.9 (14.1)
		Minimal-Memory	229.3	13	159.7	407.5	15.3
1e-8		Just-In-Time	171.5	8	105.8	296.1	24.9 (18.1)
		Minimal-Memory	319.4	8	109.8	447.9	18.8
64		-	Full-rank	179.8	9	104.7	298.3
	1e-4	Just-In-Time	79.7	10	91.1	184.1	17.4 (10.0)
		Minimal-Memory	138.1	13	120.7	272.2	11.0
	1e-8	Just-In-Time	124.6	9	90.0	228.2	17.4 (12.9)
		Minimal-Memory	239.2	9	91.5	344.5	13.7

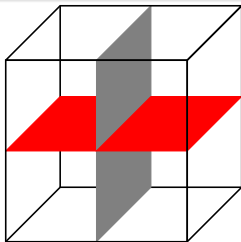
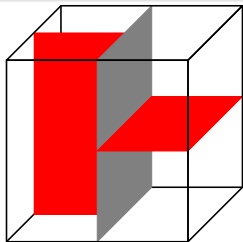
Future works: HODLR compression



Future works: align separators

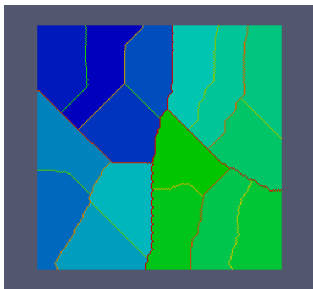
Contraint partitioning process

- Align separators to form clusters during the nested dissection process
- Preliminary work using fixed vertices
- When partitioning $G = A \cup B \cup C$, ensure that the children interaction with C is symmetric to have similar contribution pattern

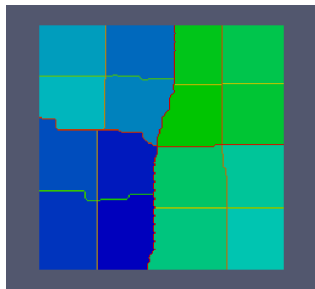


Result on a 200^2 Laplacian

With SCOTCH



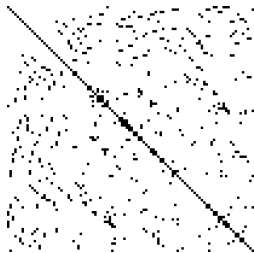
With ALIGNATOR



PASTIX 6.0.1

<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 integrated in master branch with clustering strategies
- Cholesky, *LDL* and *LU* factorizations
- GMRES, CG, BiCG iterative refinements



LR2LR kernel using SVD

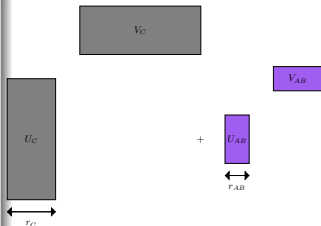
A low-rank structure $U_C V_C^t$ receives a low-rank contribution $U_{AB} V_{AB}^t$

Recompression algorithm

$$U_C V_C^t + U_{AB} V_{AB}^t = ([U_C, U_{AB}]) \times ([V_C, V_{AB}])^t$$

- QR: $[U_C, U_{AB}] = Q_1 R_1$
- QR: $[V_C, V_{AB}] = Q_2 R_2$
- SVD: $R_1 R_2^t = u \sigma v^t$

$$A = (Q_1 u \sigma) \times (Q_2 v)^t$$



The complexity of this operation depends on the dimensions of the target C
Slightly more complex algorithm for RRQR, that still requires zeroes padding

LR2LR kernel using SVD

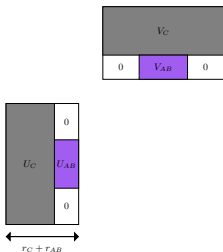
A low-rank structure $U_C V_C^t$ receives a low-rank contribution $U_{AB} V_{AB}^t$

Recompression algorithm

$$U_C V_C^t + U_{AB} V_{AB}^t = ([U_C, U_{AB}]) \times ([V_C, V_{AB}])^t$$

- QR: $[U_C, U_{AB}] = Q_1 R_1$
- QR: $[V_C, V_{AB}] = Q_2 R_2$
- SVD: $R_1 R_2^t = u \sigma v^t$

$$A = (Q_1 u \sigma) \times (Q_2 v)^t$$



The complexity of this operation depends on the dimensions of the target C
Slightly more complex algorithm for RRQR, that still requires zeroes padding

LR2LR kernel using SVD

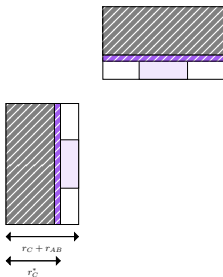
A low-rank structure $U_C V_C^t$ receives a low-rank contribution $U_{AB} V_{AB}^t$

Recompression algorithm

$$U_C V_C^t + U_{AB} V_{AB}^t = ([U_C, U_{AB}]) \times ([V_C, V_{AB}])^t$$

- QR: $[U_C, U_{AB}] = Q_1 R_1$
- QR: $[V_C, V_{AB}] = Q_2 R_2$
- SVD: $R_1 R_2^t = u \sigma v^t$

$$A = (Q_1 u \sigma) \times (Q_2 v)^t$$



The complexity of this operation depends on the dimensions of the target C
Slightly more complex algorithm for RRQR, that still requires zeroes padding

LR2LR kernel using RRQR

A low-rank structure $u_1 v_1^t$ receives a low-rank contribution $u_2 v_2^t$.
 u_1 and u_2 are orthogonal matrices

Algorithm

$$A = u_1 v_1^t + u_2 v_2^t = ([u_1, u_2]) \times ([v_1, v_2])^t$$

Orthogonalize u_2 with respect to u_1 :

$$u_2^* = u_2 - u_1(u_1^t u_2) \quad \Theta(mr_1 r_2)$$

Form new orthogonal basis, and normalize each column :

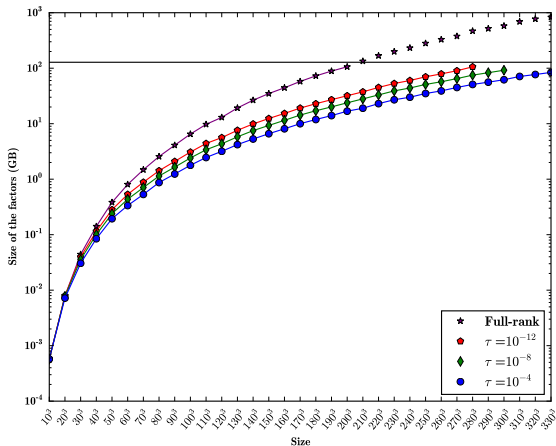
$$[u_1, u_2] = [u_1, u_2^*] \times \begin{pmatrix} I & u_1^t u_2 \\ 0 & I \end{pmatrix}$$

Apply a RRQR on :

$$\begin{pmatrix} I & u_1^t u_2 \\ 0 & I \end{pmatrix} \times ([v_1, v_2])^t$$

RRQR with truncation in $\Theta(n(r_1 + r_2)r_1^*)$. Less stable?

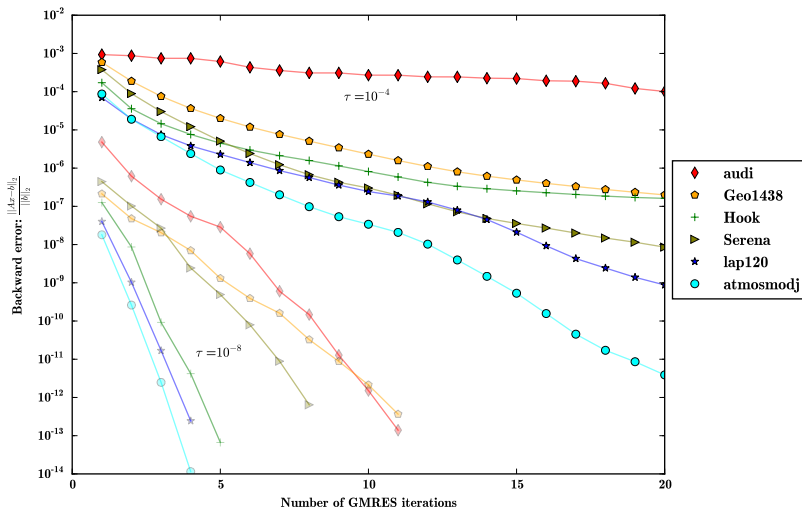
Scaling on Laplacians: memory consumption RRQR/Minimal Memory



Results

- Maximum consumption of 128 GB
- Problem of size $200^3 = 8M$ with full-rank
- Problem of size $330^3 = 36M$ with 10^{-4} tolerance

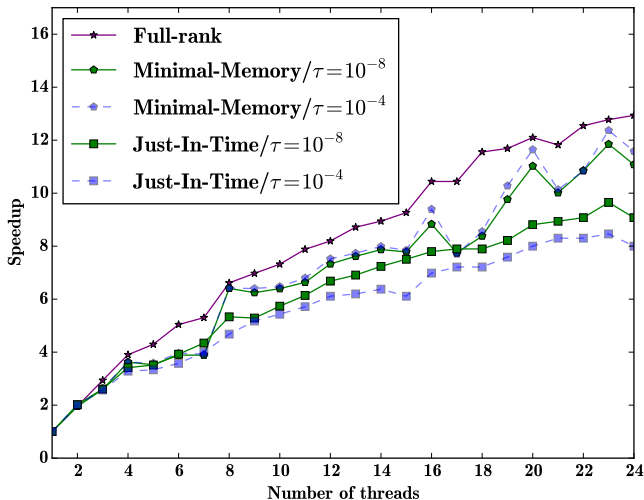
Convergence of RRQR/Minimal Memory



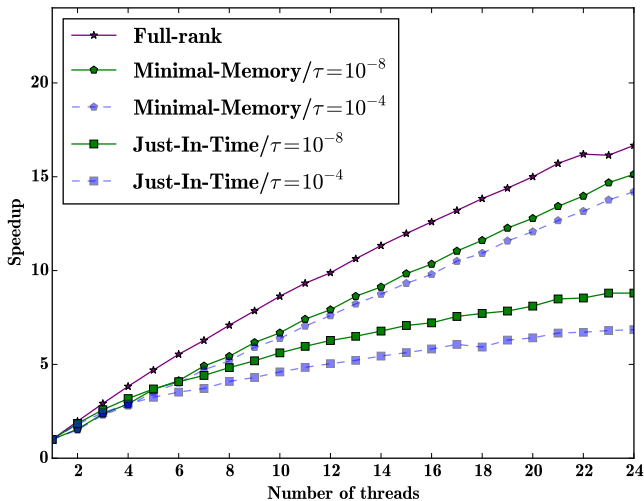
Costs distribution on the Atmosmodj matrix with $\tau = 10^{-8}$

	Full-rank	Just-In-Time		Minimal Memory	
		SVD	RRQR	SVD	RRQR
Factorization time (s)					
Compression	-	4.1e+02	3.4e+01	1.8e+02	5.6e+00
Block factorization (GETRF)	7.2e-01	7.4e-01	7.3e-01	7.8e-01	7.6e-01
Panel solve (TRSM)	1.7e+01	6.9e+00	7.4e+00	7.6e+00	7.9e+00
Update					
Formation of contribution	-	-	-	9.9e+01	4.2e+01
Addition of contribution	-	-	-	3.0e+03	7.3e+02
Dense update (GEMM)	4.6e+02	1.3e+02	9.7e+01	2.8e+01	2.4e+01
Total	4.7e+02	5.5e+02	1.4e+02	3.6e+03	8.1e+02
Solve time (s)	6.3e+00	1.9e+00	3.0e+00	1.5e+00	3.2e+00
Factor final size (GB)	16.3	6.95	7.49	6.85	7.31
Memory peak for the factors (GB)	16.3	16.3	16.3	6.85	7.31

Parallelism / static scheduling



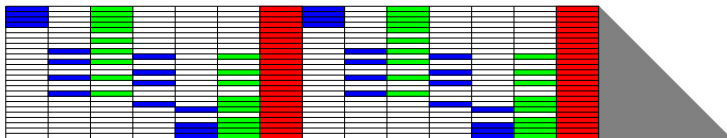
Parallelism / Parsec



Modeling of the problem

Proposition

- Define a distance between rows: the number of differences between off-diagonal blocks
- Express the problem as a Traveling Salesman Problem (TSP) to sort rows in order to minimize the overall distance
- Use heuristics to perform TSP with low complexity



Modeling of the problem

Proposition

- Define a distance between rows: the number of differences between off-diagonal blocks
- Express the problem as a Traveling Salesman Problem (TSP) to sort rows in order to minimize the overall distance
- Use heuristics to perform TSP with low complexity

1				
2				
3	↕ +1	↕ +1	↕ +0	↕ +0
4				

	1	2	3	4
1	0	-	-	-
2	3	0	-	-
3	3	2	0	-
4	1	4	2	0

Improvements with $\tau = 10^{-8}$ on a set of six matrices

Variant	Just-In-Time		Minimal Memory		
	Time PARSEC	Time static	Time PARSEC	Time static	Memory
strong admissibility	0.52	0.62	1.24	2.28	0.52
+ k-way	0.49	0.55	1.17	1.94	0.48
+ projections	0.50	0.55	1.19	1.90	0.50
+ ratio set to 0.5	0.51	0.56	0.96	1.43	0.54
+ ratio set to 0.25	0.50	0.55	1.19	1.89	0.50

Towards sparse ordering dedicated to low-rank

Nested dissection

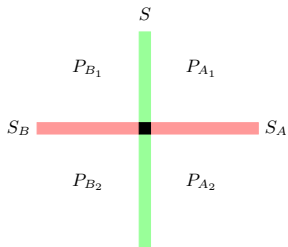
1. Partition $V = A \cup B \cup S$, such as any path from A to B goes through S
2. Order S with larger numbers: $V_A < V_B < V_S$
3. Apply the process recursively on A and B

As A and B are processed independently, their contribution on S is not symmetric

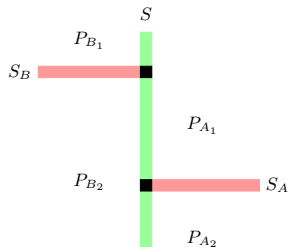
Current solutions in sparse low-rank solvers

- Find low-rank clustering by applying a k-way partitioning on each separator (Mumps, Strumpack)
- Reordering strategy to cluster close contributions (PaStiX, and cf. previous talk)

Simple representation of the problem on a 2D graph

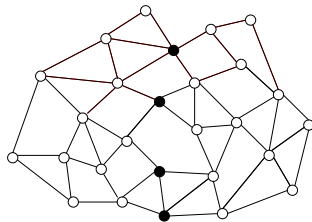


- Single interaction between S and $S_A \cup S_B$ in the original graph
- Two large interaction blocks $P_{A_1} \cup P_{B_1}$ and $P_{A_2} \cup P_{B_2}$ during the factorization



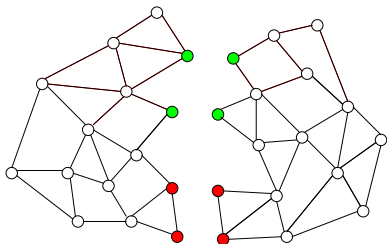
- Double interaction: between S and S_A and between S and S_B in the original graph
- Three “large” interaction blocks during the factorization

Fixed vertices



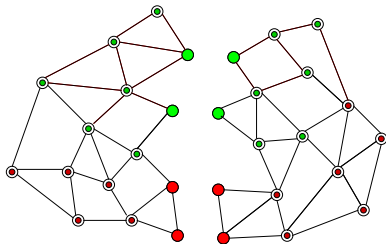
- Regular nested dissection on $G = A \cup B \cup C$.

Fixed vertices



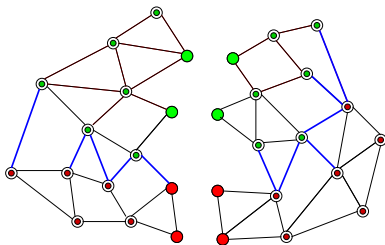
- Recursion with the halo on $A \cup C$ and $B \cup C$.
- C vertices are bi-partitioned into green and red vertices.

Fixed vertices



- The partition of C is extended to both $A \cup C$ and $B \cup C$.

Fixed vertices



- We got a new edge separator for both subgraphs.
- Turn those separators into vertex separators.