



HAL
open science

Utilisation de la compression low-rank pour réduire la complexité du solveur PaStiX

Grégoire Pichon, Mathieu Faverge, Pierre Ramet, Jean Roman

► To cite this version:

Grégoire Pichon, Mathieu Faverge, Pierre Ramet, Jean Roman. Utilisation de la compression low-rank pour réduire la complexité du solveur PaStiX. JCAD 2018 - Journées Calcul et Données, Oct 2018, Lyon, France. hal-01956928

HAL Id: hal-01956928

<https://inria.hal.science/hal-01956928v1>

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.



Utilisation de la compression low-rank pour réduire la complexité du solveur PaStiX

26 Septembre 2018 - JCAD 2018 - Lyon

Grégoire Pichon, *Mathieu Faverge*, Pierre Ramet, Jean Roman

Outline

1. Context
2. Block Low-Rank solver
 - From full-rank to low-rank
 - Impact of ordering techniques
3. Application: HORSE
4. Conclusion and future works

1

Context

The PaSTiX solver

Factorization

- sparse direct solver
- real and complex arithmetic
- LU , LL^t , LDL^t , LL^H and LDL^H factorizations
- Static pivoting
- Refinement with BICGSTAB, CG, GMRES

Parallelism

- Supernodal approach
- Native static scheduling (used in next experiments)
- Runtime systems: PARSEC and STARPU
- Heterogeneous architectures (GPUs, KNL)
- Distributed (Ongoing work)

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

Objectives

Current sparse direct solver costs for 3D problems

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

Current status

- Direct solver suffer from their high complexities
- Iterative or hybrid method are often problem dependent
- Machine precision is not required with inaccurate measures

Objectives: low-rank compression in a sparse direct solver

- Introduce compression of dense blocks
- Reduce time and/or memory complexity
- Keep the same level of parallelism

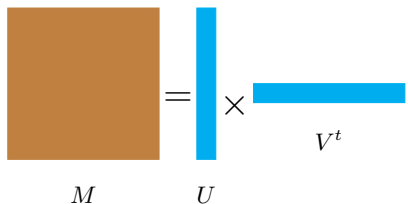
2

Block Low-Rank solver

2.1

From full-rank to low-rank

Low-rank compression


$$M = U \times V^t$$

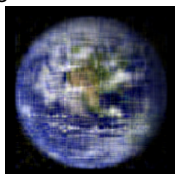
$U, V \in \mathbb{R}^{n \times r}$
 $M \in \mathbb{R}^{n \times n}$

M U

Storage in $2nr$ instead of n^2 .



Original picture.



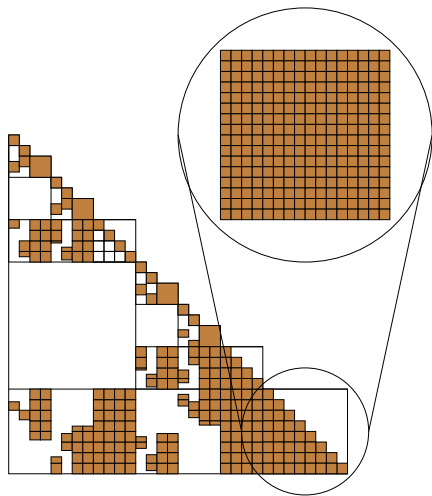
$r = 10$.



$r = 50$.

Compression of an image with $n = 500$.

BLR compression – Symbolic factorization



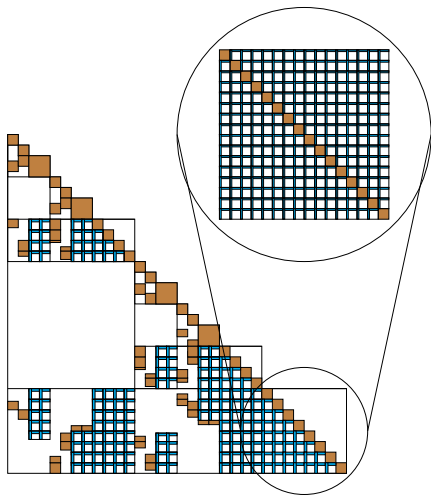
Approach

- 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

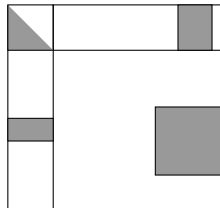
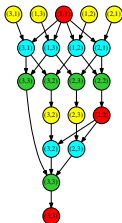
- Dense diagonal blocks
- TRSM are performed on **low-rank** off-diagonal blocks
- GEMM are performed between **low-rank** off-diagonal blocks

Strategy *Just-In-Time*

Compress L

1. Eliminate each column block
 - 1.1 Factorize the dense diagonal block
Compress off-diagonal blocks belonging to the supernode
 - 1.2 Apply a TRSM on LR blocks (cheaper)
 - 1.3 LR update on dense matrices (*LR2GE* extend-add)
2. Solve triangular systems with low-rank blocks

	Compression
	GETRF (Facto)
	TRSM (Solve)
	LR2LR (Update)
	LR2GE (Update)

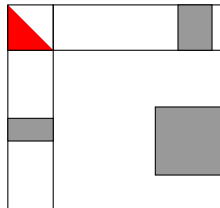
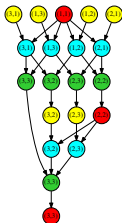


Strategy *Just-In-Time*

Compress L

1. Eliminate each column block
 - 1.1 Factorize the dense diagonal block
Compress off-diagonal blocks belonging to the supernode
 - 1.2 Apply a TRSM on LR blocks (cheaper)
 - 1.3 LR update on dense matrices (*LR2GE* extend-add)
2. Solve triangular systems with low-rank blocks






Yellow	Compression
Red	GETRF (Facto)
Cyan	TRSM (Solve)
Purple	LR2LR (Update)
Green	LR2GE (Update)

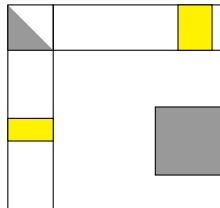
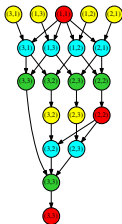


Strategy *Just-In-Time*

Compress L

1. Eliminate each column block
 - 1.1 Factorize the dense diagonal block
Compress off-diagonal blocks belonging to the supernode
 - 1.2 Apply a TRSM on LR blocks (cheaper)
 - 1.3 LR update on dense matrices (*LR2GE* extend-add)
2. Solve triangular systems with low-rank blocks






	Compression
	GETRF (Facto)
	TRSM (Solve)
	LR2LR (Update)
	LR2GE (Update)

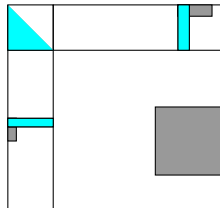
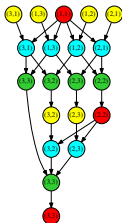


Strategy *Just-In-Time*

Compress L

1. Eliminate each column block
 - 1.1 Factorize the dense diagonal block
Compress off-diagonal blocks belonging to the supernode
 - 1.2 Apply a TRSM on LR blocks (cheaper)
 - 1.3 LR update on dense matrices (*LR2GE* extend-add)
2. Solve triangular systems with low-rank blocks






	Compression
	GETRF (Facto)
	TRSM (Solve)
	LR2LR (Update)
	LR2GE (Update)

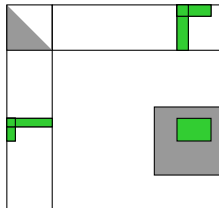
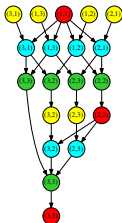


Strategy *Just-In-Time*

Compress L

1. Eliminate each column block
 - 1.1 Factorize the dense diagonal block
Compress off-diagonal blocks belonging to the supernode
 - 1.2 Apply a TRSM on LR blocks (cheaper)
 - 1.3 LR update on dense matrices (*LR2GE* extend-add)
2. Solve triangular systems with low-rank blocks

	Compression
	GETRF (Facto)
	TRSM (Solve)
	LR2LR (Update)
	LR2GE (Update)



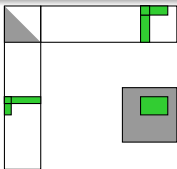
Summary of the *Just-In-Time* strategy

Assets

- The most expensive operation, the update, is less expensive with *LR2GE* kernel
- The formation of the dense update and its application is not expensive
- The size of the factors is reduced, as well as the solve cost

A limitation of this approach

- All blocks are allocated in full-rank before being compressed
- Limiting this constraint reduces the level of parallelism



Just-In-Time

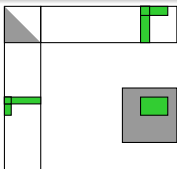
Summary of the *Just-In-Time* strategy

Assets

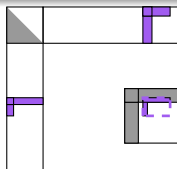
- The most expensive operation, the update, is less expensive with *LR2GE* kernel
- The formation of the dense update and its application is not expensive
- The size of the factors is reduced, as well as the solve cost

A limitation of this approach

- All blocks are allocated in full-rank before being compressed
- Limiting this constraint reduces the level of parallelism



Just-In-Time



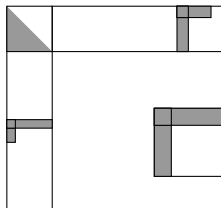
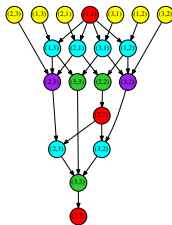
Minimal Memory

Strategy *Minimal Memory*

Compress A

1. Compress large off-diagonal blocks in A (exploiting sparsity)
2. Eliminate each column block
 - 2.1 Factorize the dense diagonal block
 - 2.2 Apply a TRSM on LR blocks (cheaper)
 - 2.3 LR update on LR matrices (*LR2LR* extend-add)
3. Solve triangular systems with LR blocks

Yellow	Compression
Red	GETRF (Facto)
Cyan	TRSM (Solve)
Purple	LR2LR (Update)
Green	LR2GE (Update)

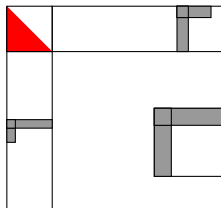
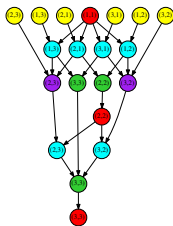


Strategy *Minimal Memory*

Compress A

1. Compress large off-diagonal blocks in A (exploiting sparsity)
2. Eliminate each column block
 - 2.1 Factorize the dense diagonal block
 - 2.2 Apply a TRSM on LR blocks (cheaper)
 - 2.3 LR update on LR matrices ($LR2LR$ extend-add)
3. Solve triangular systems with LR blocks

Yellow	Compression
Red	GETRF (Facto)
Cyan	TRSM (Solve)
Purple	LR2LR (Update)
Green	LR2GE (Update)

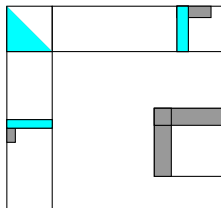
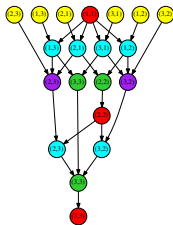


Strategy *Minimal Memory*

Compress A

1. Compress large off-diagonal blocks in A (exploiting sparsity)
2. Eliminate each column block
 - 2.1 Factorize the dense diagonal block
 - 2.2 Apply a TRSM on LR blocks (cheaper)
 - 2.3 LR update on LR matrices ($LR2LR$ extend-add)
3. Solve triangular systems with LR blocks

Yellow	Compression
Red	GETRF (Facto)
Cyan	TRSM (Solve)
Purple	LR2LR (Update)
Green	LR2GE (Update)

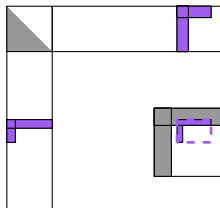
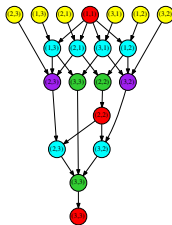


Strategy *Minimal Memory*

Compress A

1. Compress large off-diagonal blocks in A (exploiting sparsity)
2. Eliminate each column block
 - 2.1 Factorize the dense diagonal block
 - 2.2 Apply a TRSM on LR blocks (cheaper)
 - 2.3 LR update on LR matrices (*LR2LR* extend-add)
3. Solve triangular systems with LR blocks

Yellow	Compression
Red	GETRF (Facto)
Cyan	TRSM (Solve)
Purple	LR2LR (Update)
Green	LR2GE (Update)



Experimental setup - architectures

GENCI - Occigen

- 2 INTEL Xeon *E5 – 2680v3* at 2.50 GHz
- 128 GB
- 2 × 12 cores

MCIA - Avakas

- 2 INTEL Xeon *x5675* at 3.06 GHz
- 48 GB
- 2 × 12 cores

PlaFRIM - Miriel

- 2 INTEL Xeon *E5 – 2680v3* at 2.50 GHz
- 128 GB
- 2 × 12 cores

Experimental setup - matrices

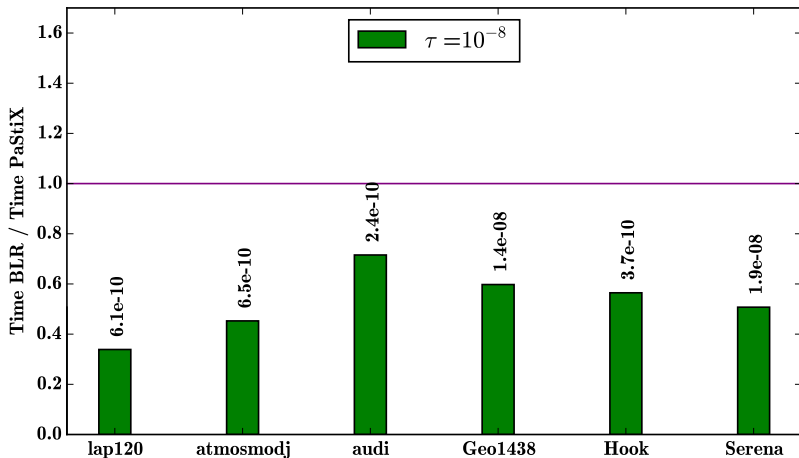
Large (33 matrices)

- Square matrices issued from the SuiteSparse Matrix Collection
- real or complex
- $50K \leq N \leq 5M$
- $nb_{ops} > 1$ TFlops
- Web and DNA matrices were removed

Small (6 matrices)

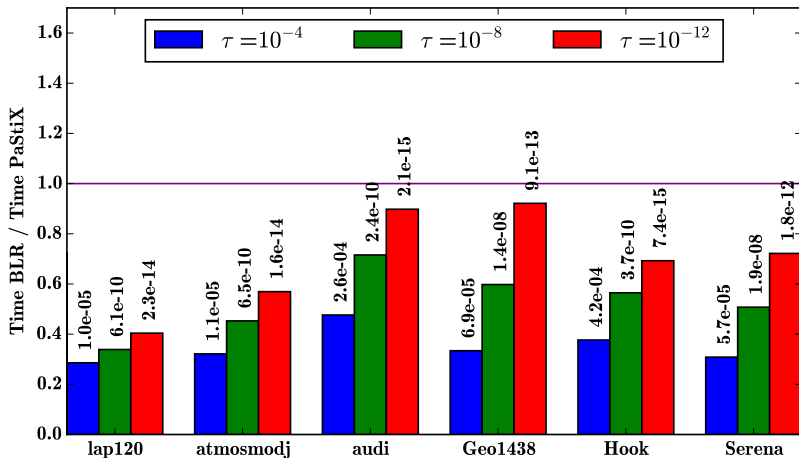
Matrix	Fact.	Size	Field
<i>Atmosmodj</i>	<i>LU</i>	1 270 432	atmospheric model
<i>Audi</i>	<i>LL^t</i>	943 695	structural problem
<i>Hook</i>	<i>LDL^t</i>	1 498 023	model of a steel hook
<i>Serena</i>	<i>LDL^t</i>	1 391 349	gas reservoir simulation
<i>Geo1438</i>	<i>LL^t</i>	1 437 960	geomechanical model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

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



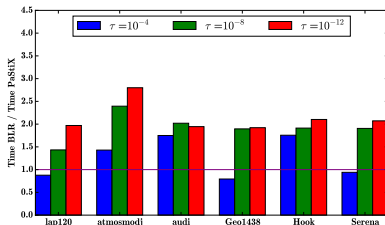
Factorization time reduced by a factor of 2 for $\tau = 10^{-8}$.

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



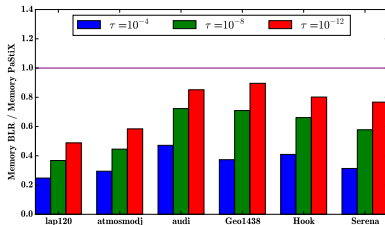
Factorization time reduced by a factor of 2 for $\tau = 10^{-8}$.

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 footprint

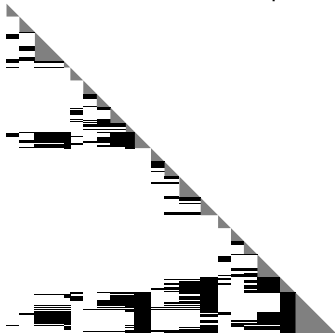
- Reduction by a factor of 1.7 for $\tau = 10^{-8}$
- Close to the results obtained using SVD

2.2

Impact of ordering techniques

Ordering for sparse matrices

One can reorder separators without modifying the fill-in.



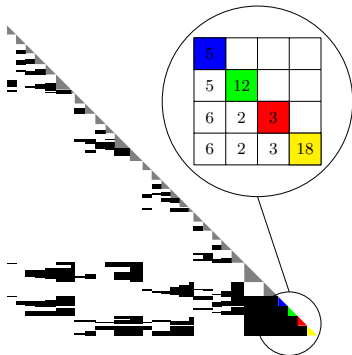
Symbolic factorization of a $8 \times 8 \times 8$ Laplacian.

Enhance ordering

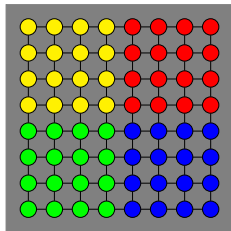
- Modify the partitioning process to enhance compressibility
- Reorder unknowns within a separator:
 - ▶ To enhance clustering within the separator
 - ▶ To enhance the coupling part

Clustering techniques: existing solutions

- k-way partitioning (dense, multifrontal)



(a) Symbolic factorization.

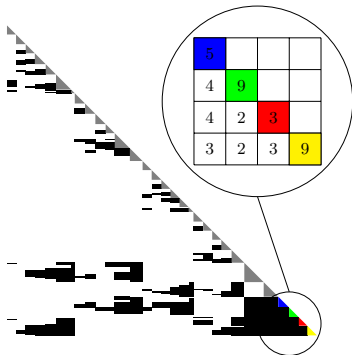


(b) First separator clustering.

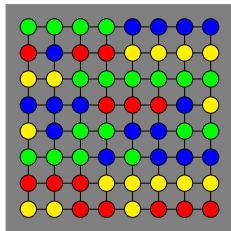
$8 \times 8 \times 8$ Laplacian partitioned using SCOTCH and k-way clustering on the first separator.

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.

Performance gain of the TSP ordering in full-rank

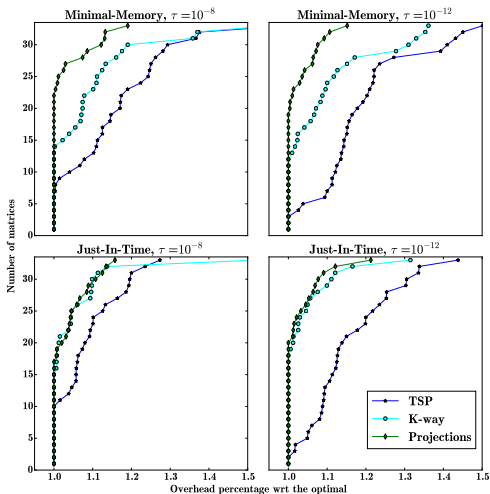
Results

- 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 when using PARSEC runtime system with TSP

Performance profile of the factorization time



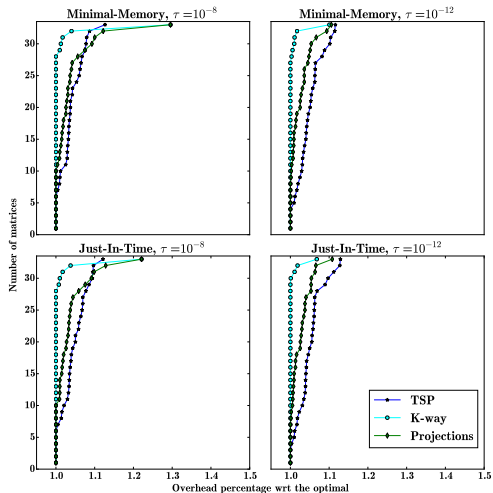
Method

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

Results

- K-way outperforms TSP by 6%
- Projections outperforms k-way by 6% for *Minimal Memory* strategy

Performance profile of the memory consumption



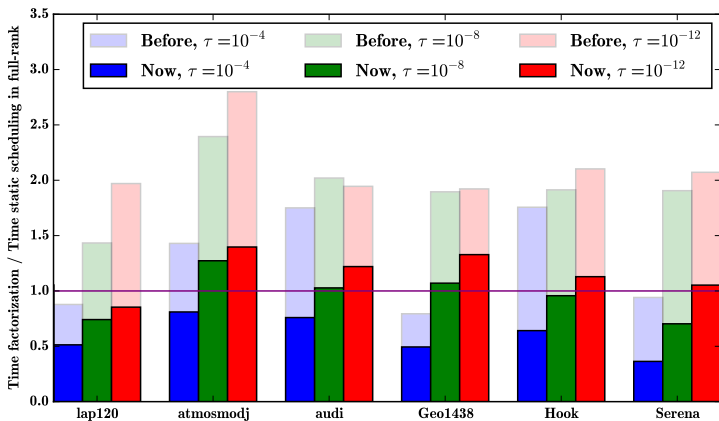
Method

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

Results

- K-way outperforms TSP by 4%
- K-way outperforms Projections by 2%

Behavior of RRQR/*Minimal Memory* with PARSEC and Projections



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

3

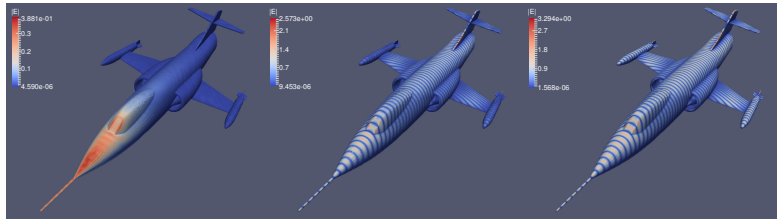
Application: HORSE

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 $|E|$ - HDG- \mathbb{P}_1 to HDG- \mathbb{P}_3

Impact on HORSE - Case HDG- \mathbb{P}_2

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

4

Conclusion and future works

Conclusion

- Block low-rank allows to reduce time and memory complexity of sparse direct solver
- *Possible to control* the final accuracy
- Exploit the same level of parallelism as the full-rank solver
- Benefit from the dynamic scheduling of the runtime systems implementation for free
- May accelerate hybrid solvers such as HORSE

Future work

Ordering strategies - E. Korkmaz (PhD)

- Target hierarchical compression formats
- Try to apply the strategy in the ordering library (Scotch, Metis)

Others

- Distributed version (load balancing)
Looking for an intern student/Engineer
- Low-rank operations on GPUs
- Use other low-rank compression kernels, such as randomization
- Integration in new applications

Thank you.

Questions?