Reducing the Memory Gap between Hierarchical Lowrank Formats

Ronald Kriemann
Max Planck Institute MiS Leipzig

Advanced numerical methods for non-local problems
Boğaziçi University, Istanbul

MAX PLANCK INSTITUTE
FOR MATHEMATICS IN THE SCIENCES

Hierarchical Matrices

Hierarchical Matrices

Approximate dense data $M_{\tau,\sigma} \in \mathbb{C}^{\#\tau \times \#\sigma}$ of $M \in \mathbb{C}^{n \times n}$ by

$$U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

with $U_{\tau,\sigma} \in \mathbb{C}^{\#\tau \times k}$, $V_{\tau,\sigma} \in \mathbb{C}^{\#\sigma \times k}$ and

$$k \ll \min(\#\tau, \#\sigma)$$

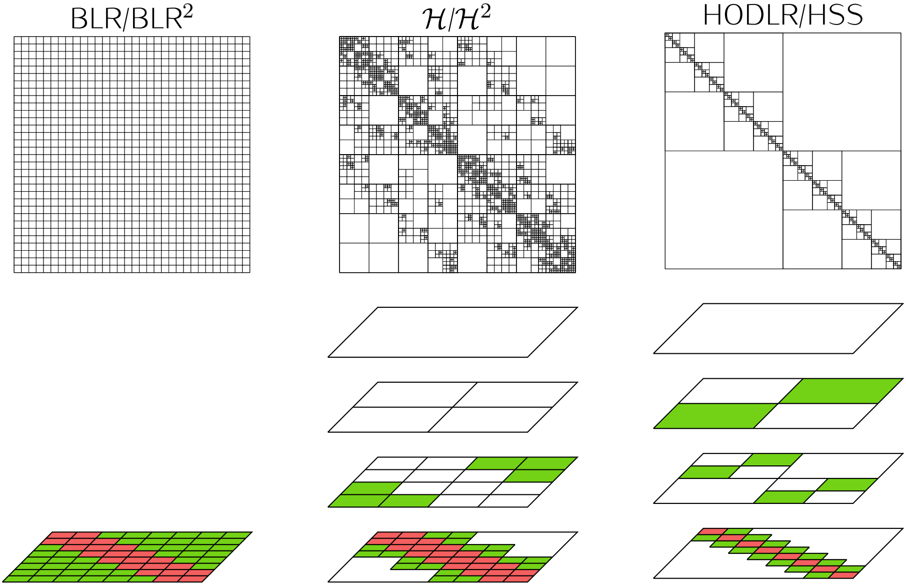
such that

$$||M_{\tau,\sigma} - U_{\tau,\sigma} V_{\tau,\sigma}^H|| \leq \delta \qquad \text{or}$$
$$||M_{\tau,\sigma} - U_{\tau,\sigma} V_{\tau,\sigma}^H|| \leq \varepsilon ||M_{\tau,\sigma}||$$

yielding an approximation $\widetilde{M}$ of M with $\mathcal{O}\left(n \log^\alpha n\right)$ storage.

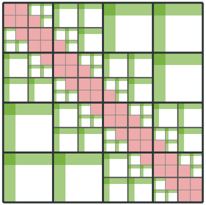
In the literature, many different formats of (hierarchical) lowrank matrices exist.

Block structure



Basis Representation

Separate Bases



$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

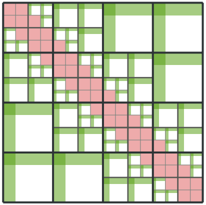
with

$$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$$

$$\mathcal{O}(n \log n)$$

Basis Representation

Separate Bases



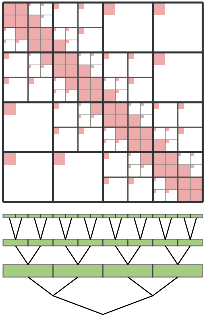
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

with

$$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, \; V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$$

$$\mathcal{O}(n \log n)$$

Shared Bases



$$M_{\tau,\sigma} = \mathcal{U}_\tau \cdot S_{\tau,\sigma} \cdot \mathcal{V}_\sigma{}^H$$

with

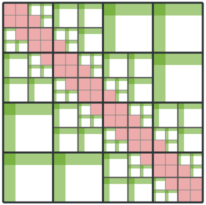
$$\mathcal{U}_\tau \in \mathbb{R}^{\#\tau \times k}, \mathcal{V}_\sigma \in \mathbb{R}^{\#\sigma \times k},$$

$$S_{\tau,\sigma} \in \mathbb{R}^{k \times k}$$

$$\underline{\mathcal{O}(n)} + \mathcal{O}(n \log n)$$

Basis Representation

Separate Bases



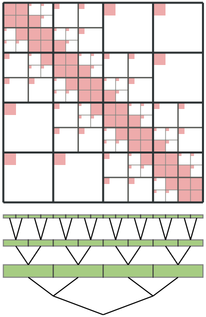
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

with

$$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$$

$$\mathcal{O}\left(n \log n\right)$$

Shared Bases



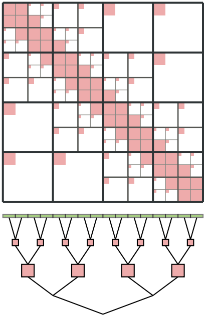
$$M_{\tau,\sigma} = \mathcal{U}_\tau \cdot S_{\tau,\sigma} \cdot \mathcal{V}_\sigma{}^H$$

with

$$\mathcal{U}_\tau \in \mathbb{R}^{\#\tau \times k}, \mathcal{V}_\sigma \in \mathbb{R}^{\#\sigma \times k},$$
$$S_{\tau,\sigma} \in \mathbb{R}^{k \times k}$$

$$\underline{\mathcal{O}\left(n\right)} + \mathcal{O}\left(n \log n\right)$$

Nested Bases



$$M_{\tau,\sigma} = \widetilde{\mathcal{U}}_\tau \cdot S_{\tau,\sigma} \cdot \widetilde{\mathcal{V}}_\sigma^H$$

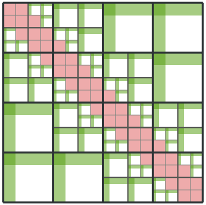
with

implicit $\widetilde{\mathcal{U}}_\tau, \widetilde{\mathcal{V}}_\sigma$

$$\mathcal{O}\left(n\right)$$

Basis Representation

$\mathcal{H}$



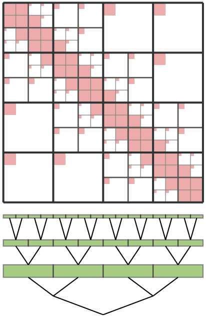
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

with

$$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$$

$$\mathcal{O}\left(n \log n\right)$$

Uniform-$\mathcal{H}$



$$M_{\tau,\sigma} = \mathcal{U}_\tau \cdot S_{\tau,\sigma} \cdot \mathcal{V}_\sigma{}^H$$

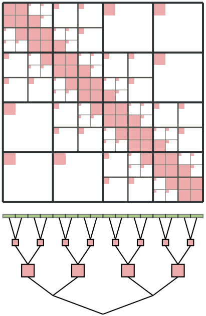
with

$$\mathcal{U}_\tau \in \mathbb{R}^{\#\tau \times k}, \mathcal{V}_\sigma \in \mathbb{R}^{\#\sigma \times k},$$

$$S_{\tau,\sigma} \in \mathbb{R}^{k \times k}$$

$$\underline{\mathcal{O}\left(n\right)} + \mathcal{O}\left(n \log n\right)$$

$\mathcal{H}^2$



$$M_{\tau,\sigma} = \widetilde{\mathcal{U}}_\tau \cdot S_{\tau,\sigma} \cdot \widetilde{\mathcal{V}}_\sigma^H$$

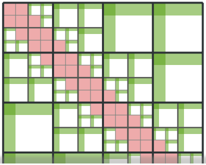
with

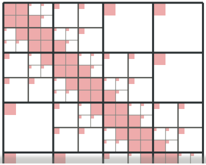
implicit $\widetilde{\mathcal{U}}_\tau, \widetilde{\mathcal{V}}_\sigma$

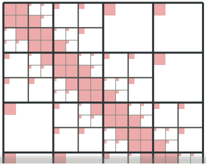
$$\mathcal{O}\left(n\right)$$

Basis Representation

$\mathcal{H}$	Uniform-$\mathcal{H}$	$\mathcal{H}^2$







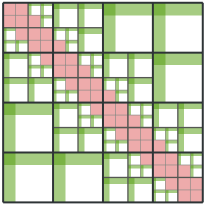
Definition 2.11. *Let $V_k = V_k(I \times I)$ as before and deduce from V_k the subspaces $V_k(b)$ for all blocks $b \in P_2$. An $\mathcal{H}$-matrix from $\mathcal{M}_{\mathcal{H},k}(I \times I, P_2)$ is a uniform $\mathcal{H}$-matrix, if all block matrices M^b, $b \in P_2$, appearing in (7) belong to $V_k(b)$. The set of uniform $\mathcal{H}$-matrices is denoted by $\mathcal{U}_{\mathcal{H},k}(I \times I, P_2, V_k)$.*

Hackbusch: "*A Sparse Matrix Arithmetic Based on $\mathcal{H}$-Matrices. Part I: Introduction to $\mathcal{H}$-Matrices*", Computing 62, 89–108, 1999.

with	with	with
$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$	$\mathcal{U}_\tau \in \mathbb{R}^{\#\tau \times k}, \mathcal{V}_\sigma \in \mathbb{R}^{\#\sigma \times k}$, $S_{\tau,\sigma} \in \mathbb{R}^{k \times k}$	*implicit* $\widetilde{\mathcal{U}}_\tau, \widetilde{\mathcal{V}}_\sigma$
$\mathcal{O}(n \log n)$	$\underline{\mathcal{O}(n)} + \mathcal{O}(n \log n)$	$\mathcal{O}(n)$

Basis Representation

$\mathcal{H}$



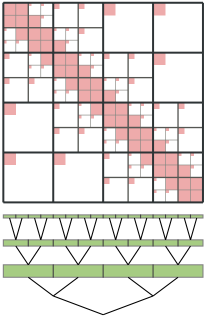
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

with

$$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$$

$$\mathcal{O}(n \log n)$$

Uniform-$\mathcal{H}$



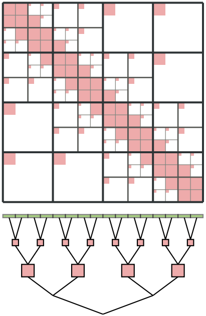
$$M_{\tau,\sigma} = \mathcal{U}_\tau \cdot S_{\tau,\sigma} \cdot \mathcal{V}_\sigma{}^H$$

with

$$\mathcal{U}_\tau \in \mathbb{R}^{\#\tau \times k}, \mathcal{V}_\sigma \in \mathbb{R}^{\#\sigma \times k},$$
$$S_{\tau,\sigma} \in \mathbb{R}^{k \times k}$$

$$\underline{\mathcal{O}(n)} + \mathcal{O}(n \log n)$$

$\mathcal{H}^2$



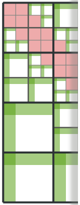
$$M_{\tau,\sigma} = \widetilde{\mathcal{U}}_\tau \cdot S_{\tau,\sigma} \cdot \widetilde{\mathcal{V}}_\sigma^H$$

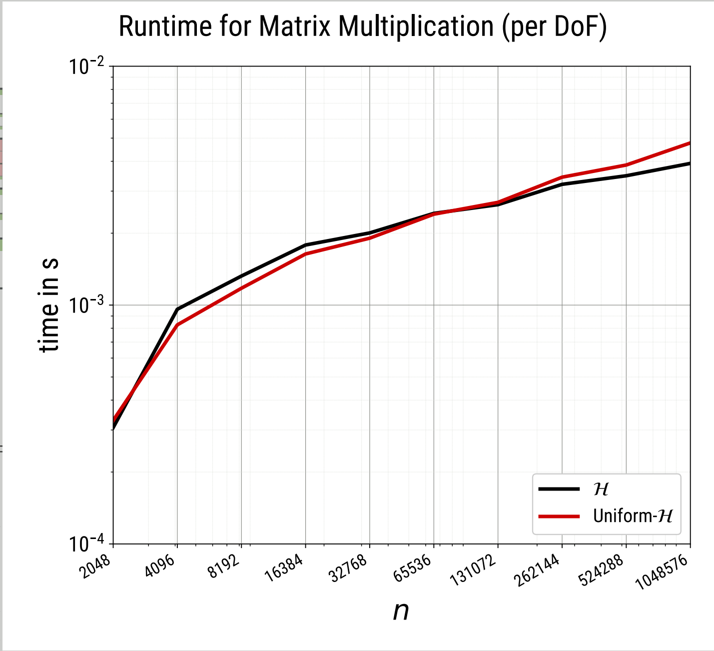
with

implicit $\widetilde{\mathcal{U}}_\tau, \widetilde{\mathcal{V}}_\sigma$

$$\mathcal{O}(n)$$

Basis Representation



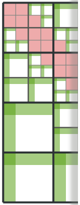


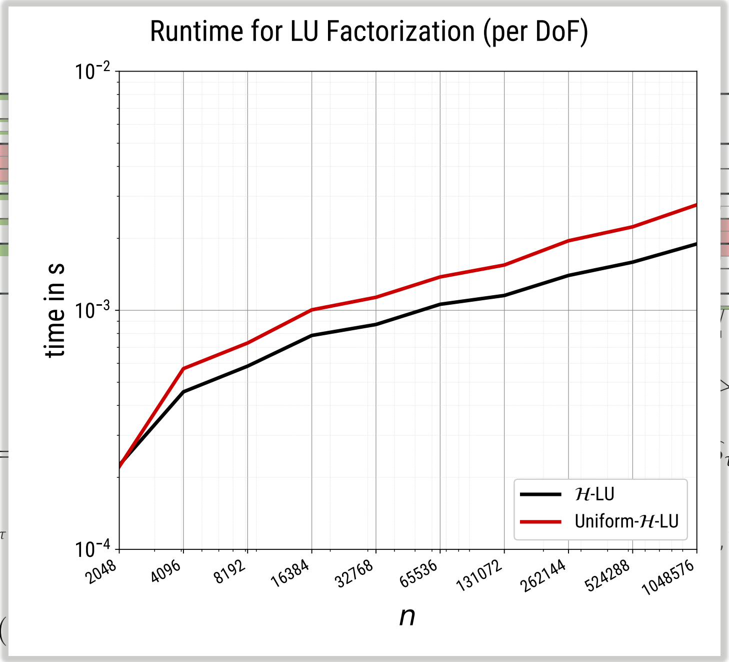
$M_{\tau,\sigma} =$ … $_{\tau,\sigma} \cdot \widetilde{\mathcal{V}}_\sigma^H$

$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau}$ … $, \widetilde{\mathcal{V}}_\sigma$

$\mathcal{O}($

Basis Representation





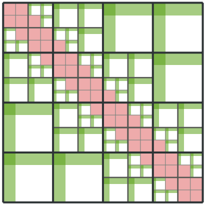
$M_{\tau,\sigma} =$ … $_{\tau,\sigma} \cdot \widetilde{\mathcal{V}}_\sigma^H$

$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau}$ … $, \widetilde{\mathcal{V}}_\sigma$

$\mathcal{O}($

Basis Representation

$\mathcal{H}$



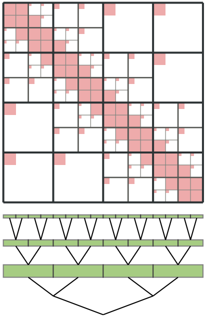
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$$

with

$$U_{\tau,\sigma} \in \mathbb{R}^{\#\tau \times k}, V_{\tau,\sigma} \in \mathbb{R}^{\#\sigma \times k}$$

$$\mathcal{O}(n \log n)$$

Uniform-$\mathcal{H}$



$$M_{\tau,\sigma} = \mathcal{U}_\tau \cdot S_{\tau,\sigma} \cdot \mathcal{V}_\sigma{}^H$$

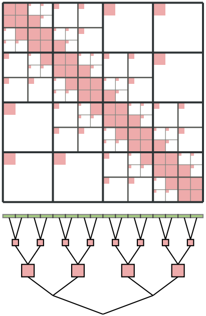
with

$$\mathcal{U}_\tau \in \mathbb{R}^{\#\tau \times k}, \mathcal{V}_\sigma \in \mathbb{R}^{\#\sigma \times k},$$

$$S_{\tau,\sigma} \in \mathbb{R}^{k \times k}$$

$$\underline{\mathcal{O}(n)} + \mathcal{O}(n \log n)$$

$\mathcal{H}^2$



$$M_{\tau,\sigma} = \widetilde{\mathcal{U}}_\tau \cdot S_{\tau,\sigma} \cdot \widetilde{\mathcal{V}}_\sigma^H$$

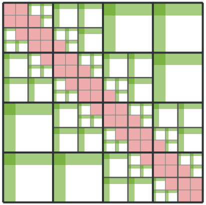
with

implicit $\widetilde{\mathcal{U}}_\tau, \widetilde{\mathcal{V}}_\sigma$

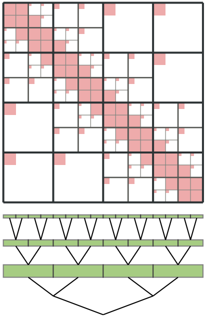
$$\mathcal{O}(n)$$

Basis Representation

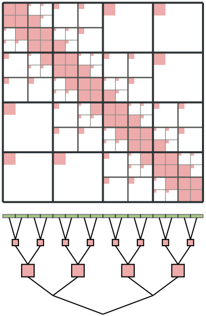
$\mathcal{H}$



Uniform-$\mathcal{H}$



$\mathcal{H}^2$



How do we *store* the data blocks (dense and lowrank)?

$\mathcal{O}(n \log n)$ $\underline{\mathcal{O}(n)} + \mathcal{O}(n \log n)$ $\mathcal{O}(n)$

Number Representation

Number Representation

For scientific computations almost always the IEEE-754 floating point standard is used.

- one sign bit
- e exponent bits and
- m mantissa bits

The mantissa bits define the floating point accuracy with *unit roundoff*

$$u = 2^{-(m+1)}$$

Most common formats:

	s-e-m	Bits	Unit Roundoff
FP80	1-15-64	80	2.7×10^{-20}
FP64	1-11-52	64	1.1×10^{-16}
FP32	1-8-23	32	6.0×10^{-8}

Number Representation

For scientific computations almost always the IEEE-754 floating point standard is used.

- one sign bit
- e exponent bits and
- m mantissa bits

The mantissa bits define the floating point accuracy with *unit roundoff*

$$u = 2^{-(m+1)}$$

Most common formats:

	s–e–m	Bits	Unit Roundoff
FP80	1-15-64	80	2.7×10^{-20}
FP64	1-11-52	64	1.1×10^{-16}
FP32	1-8-23	32	6.0×10^{-8}

However, often lowrank approximation is much coarser than the unit roundoff:

$$\varepsilon \gg u$$

Number Representation

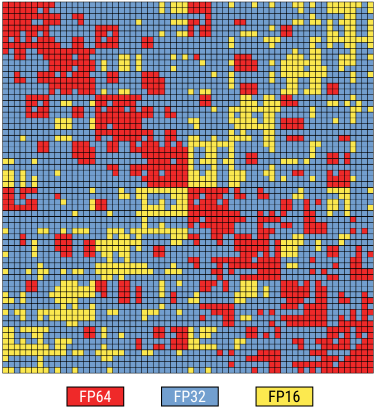
In recent years, more floating point formats were added to IEEE-754:

	s-e-m	Bits	Unit Roundoff
FP80	1-15-64	80	2.7×10^{-20}
FP64	1-11-52	64	1.1×10^{-16}
FP32	1-8-23	32	6.0×10^{-8}
TF32	1-8-10	19	4.9×10^{-4}
FP16	1-5-10	16	4.9×10^{-4}
BF16	1-8-7	16	3.9×10^{-3}
FP8	1-4-3	8	6.2×10^{-2}

How can we use these formats for storing matrix data?

Mixed Precision within Matrix[1]

Choose precision of lowrank block $U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$ based on $||M_{\tau,\sigma}||$.



Dense blocks always stored in FP64.

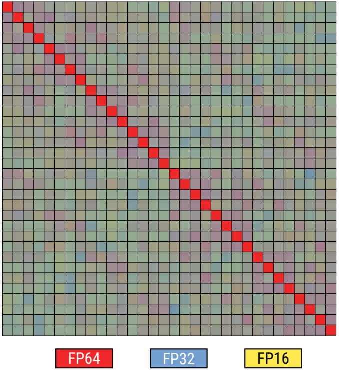
[1] Abdulah, Cao, Pei, Bosilca, Dongarra, Genton, Keyes, Ltaief, Sun: "*Accelerating Geostatistical Modeling and Prediction With Mixed-Precision Computations: A High-Productivity Approach With PaRSEC*", IEEE Trans. on Par. and Distr. Systems, 2022

Mixed Precision within Lowrank Block[1,2]

Represent $U_{\tau,\sigma} \cdot V_{\tau,\sigma}^H$ as

$$W \cdot \Sigma \cdot X^H = [W_1 W_2 W_3] \cdot \mathrm{diag}(\Sigma_1, \Sigma_2, \Sigma_3) \cdot [X_1 X_2 X_3]^H$$

with orthogonal W, X and splitting depending on the singular values σ_j in Σ_i.

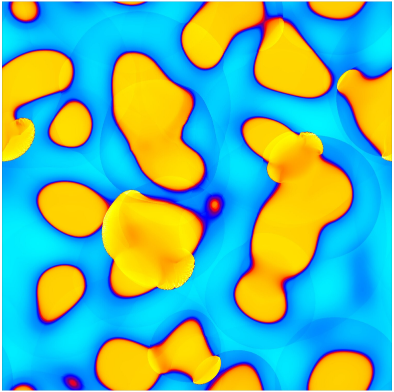


[1] Ooi, Iwashita, Fukaya, Ida, Yokota.: "*Effect of Mixed Precision Computing on H-Matrix Vector Multiplication in BEM Analysis*", Proceedings of HPCAsia2020, 2020

[2] Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, Mary: "*Mixed precision low-rank approximations and their application to block low-rank LU factorization*", IMA J. of Num. Analysis, 2022

Floating Point Compression

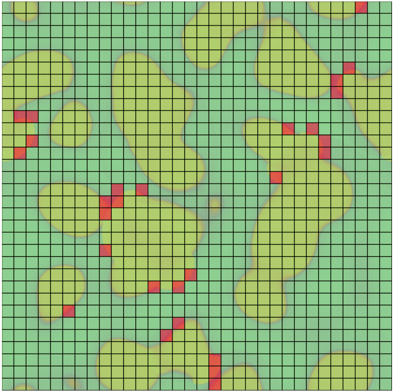
For a combustion application, lowrank approximation was combined with (lossy) floating point compression to minimize data storage[1]:



[1] K., Ltaief, Luong, Pérez, Im, Keyes: "*High-Performance Spatial Data Compression for Scientific Applications*", Euro-Par 2022

Floating Point Compression

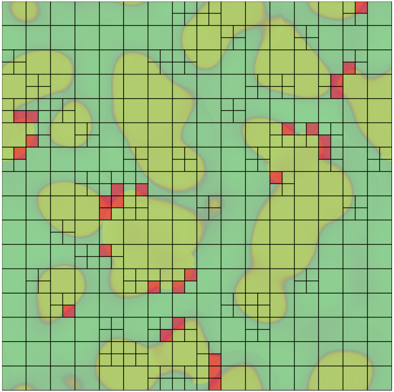
For a combustion application, lowrank approximation was combined with (lossy) floating point compression to minimize data storage[1]:



[1] K., Ltaief, Luong, Pérez, Im, Keyes: "*High-Performance Spatial Data Compression for Scientific Applications*", Euro-Par 2022

Floating Point Compression

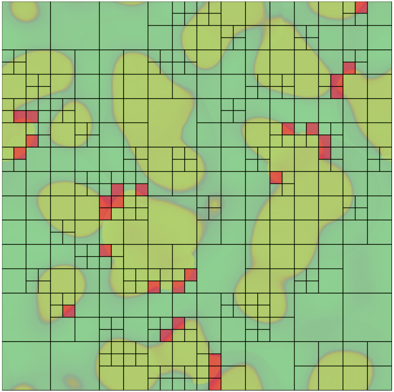
For a combustion application, lowrank approximation was combined with (lossy) floating point compression to minimize data storage[1]:



[1] K., Ltaief, Luong, Pérez, Im, Keyes: "*High-Performance Spatial Data Compression for Scientific Applications*", Euro-Par 2022

Floating Point Compression

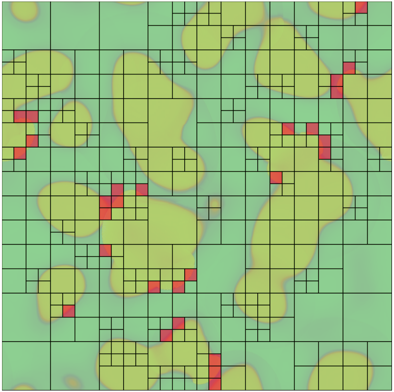
For a combustion application, lowrank approximation was combined with (lossy) floating point compression to minimize data storage[1]:

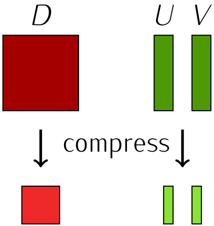


[1] K., Ltaief, Luong, Pérez, Im, Keyes: “*High-Performance Spatial Data Compression for Scientific Applications*”, Euro-Par 2022

Floating Point Compression

For a combustion application, lowrank approximation was combined with (lossy) floating point compression to minimize data storage[1]:

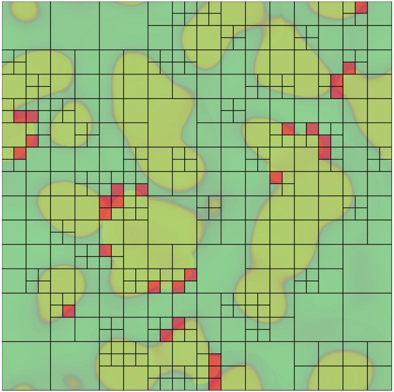


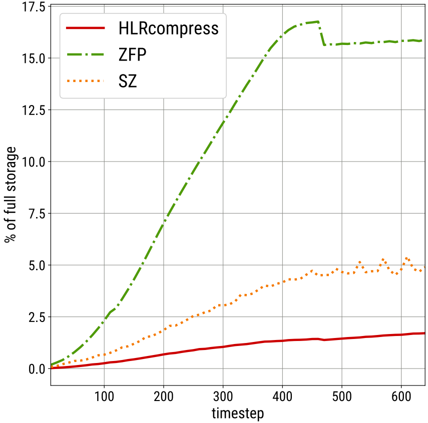


[1] K., Ltaief, Luong, Pérez, Im, Keyes: "*High-Performance Spatial Data Compression for Scientific Applications*", Euro-Par 2022

Floating Point Compression

For a combustion application, lowrank approximation was combined with (lossy) floating point compression to minimize data storage[1]:

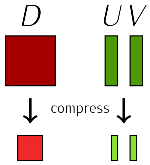




[1] K., Ltaief, Luong, Pérez, Im, Keyes: "*High-Performance Spatial Data Compression for Scientific Applications*", Euro-Par 2022

Floating Point Compression

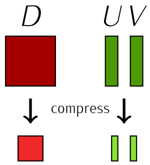
Directly compress data blocks $D_{\tau,\sigma}$ from dense blocks and $U_{\tau,\sigma}, V_{\tau,\sigma}$ from lowrank blocks using floating point compression schemes.



Assumption: compression scheme has *error control*.

Floating Point Compression

Directly compress data blocks $D_{\tau,\sigma}$ from dense blocks and $U_{\tau,\sigma}, V_{\tau,\sigma}$ from lowrank blocks using floating point compression schemes.



Assumption: compression scheme has *error control*.

ZFP[1]

- bitplane truncation for 4^d blocks,

SZ[2]/SZ3[3]

- uses curve fitting via splines,

BLOSC[4]

- bit shuffling plus lossless compression,
- optional mantissa truncation

MGARD[5]

- multi-grid technique plus lossless compression,

[1] Lindstrom: "*Fixed-rate compressed floating-point arrays*", IEEE Trans. on Vis. and Comp. Graphics 20(12), 2674–2683 (2014).

[2] Di, Cappello: "*Fast Error-Bounded Lossy HPC Data Compression with SZ*", IEEE IPDPS. pp. 730–739 (2016)

[3] Zhao,Di,Dmitriev,Tonellot,Chen,Cappello: "*Opt. Error-Bounded Lossy Comp. for Sci.Data by Dyn.Spline Interp.*", IEEE 37th ICDE, 1643–1654 (2021)

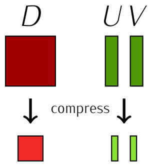
[4] `https://blosc.org`

[5] Ainsworth,Tugluk,Whitney,Klasky: "*Multilevel tech. for compression and reduction of sci.data – the univariate case*". Comp.Vis.Sci. 19, 65–76 (2018)

Floating Point Compression

Directly compress data blocks $D_{\tau,\sigma}$ from dense blocks and $U_{\tau,\sigma}$, $V_{\tau,\sigma}$ from lowrank blocks using floating point compression schemes.

Assumption: compression scheme has *error control*.



ZFP[1]

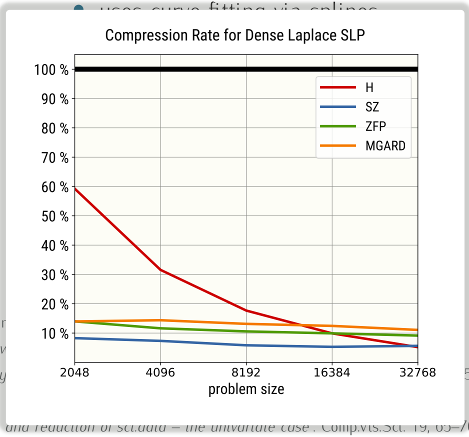
- bitplane truncation for 4^d blocks,

SZ[2]/SZ3[3]

- uses curve fitting via splines

BLOSC[4]

- bit shuffling plus lossless compression,
- optional mantissa truncation



[1] Lindstrom: "*Fixed-rate compressed floating-point arrays*", IEEE Tra

[2] Di, Cappello: "*Fast Error-Bounded Lossy HPC Data Compression w*

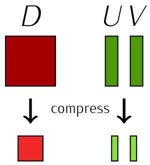
[3] Zhao,Di,Dmitriev,Tonellot,Chen,Cappello: "*Opt. Error-Bounded Lossy* … 54 (2021)

[4] `https://blosc.org`

[5] Ainsworth,Tugluk,Whitney,Klasky: "*Multilevel tech. for compression and reduction of sci.data – the univariate case*", Comp.Vis.Sci. 19, 65–76 (2018)

Floating Point Compression

Directly compress data blocks $D_{\tau,\sigma}$ from dense blocks and $U_{\tau,\sigma}, V_{\tau,\sigma}$ from lowrank blocks using floating point compression schemes.



Assumption: compression scheme has *error control*.

ZFP[1]

- bitplane truncation for 4^d blocks,

SZ[2]/SZ3[3]

- uses curve fitting via splines,

BLOSC[4]

- bit shuffling plus lossless compression,
- optional mantissa truncation

MGARD[5]

- multi-grid technique plus lossless compression,

But, SZ/SZ3/MGARD are *not* able to (further) compress lowrank data!

[1] Lindstrom: "*Fixed-rate compressed floating-point arrays*", IEEE Trans. on Vis. and Comp. Graphics 20(12), 2674–2683 (2014).

[2] Di, Cappello: "*Fast Error-Bounded Lossy HPC Data Compression with SZ*", IEEE IPDPS. pp. 730–739 (2016)

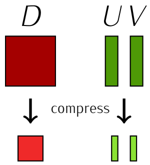
[3] Zhao,Di,Dmitriev,Tonellot,Chen,Cappello: "*Opt. Error-Bounded Lossy Comp. for Sci.Data by Dyn.Spline Interp.*", IEEE 37th ICDE, 1643–1654 (2021)

[4] `https://blosc.org`

[5] Ainsworth,Tugluk,Whitney,Klasky: "*Multilevel tech. for compression and reduction of sci.data – the univariate case*". Comp.Vis.Sci. 19, 65–76 (2018)

Floating Point Compression

Directly compress data blocks $D_{\tau,\sigma}$ from dense blocks and $U_{\tau,\sigma}$, $V_{\tau,\sigma}$ from lowrank blocks using floating point compression schemes.



Assumption: compression scheme has *error control*.

ZFP[1]

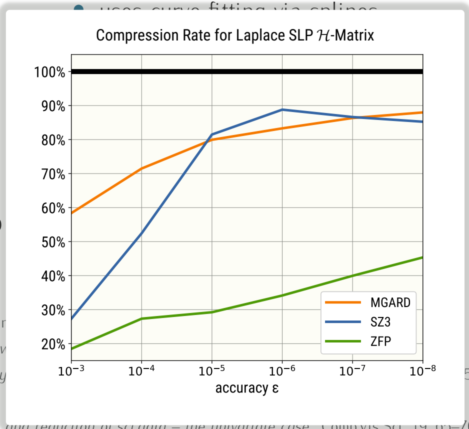
- bitplane truncation for 4^d blocks,

SZ[2]/SZ3[3]

BLOSC[4]

- bit shuffling plus lossless compression,
- optional mantissa truncation

But, SZ/SZ3/MGARD are *not* able to



[1] Lindstrom: “*Fixed-rate compressed floating-point arrays*”, IEEE Tra

[2] Di, Cappello: “*Fast Error-Bounded Lossy HPC Data Compression w*

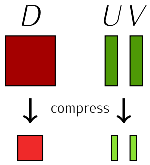
[3] Zhao,Di,Dmitriev,Tonellot,Chen,Cappello: “*Opt. Error-Bounded Lossy* ... 54 (2021)

[4] https://blosc.org

[5] Ainsworth,Tugluk,Whitney,Klasky: “*Multilevel tech. for compression and reduction of sci.data – the univariate case*”, Comp.Vis.Sci. 19, 65–76 (2018)

Floating Point Compression

Directly compress data blocks $D_{\tau,\sigma}$ from dense blocks and $U_{\tau,\sigma}$, $V_{\tau,\sigma}$ from lowrank blocks using floating point compression schemes.



Assumption: compression scheme has *error control*.

ZFP[1]

- bitplane truncation for 4^d blocks,

SZ[2]/SZ3[3]

- uses curve fitting via splines,

BLOSC[4]

- bit shuffling plus lossless compression,
- optional mantissa truncation

MGARD[5]

- multi-grid technique plus lossless compression,

But, SZ/SZ3/MGARD are *not* able to (further) compress lowrank data!

---

[1] Lindstrom: "*Fixed-rate compressed floating-point arrays*", IEEE Trans. on Vis. and Comp. Graphics 20(12), 2674–2683 (2014).

[2] Di, Cappello: "*Fast Error-Bounded Lossy HPC Data Compression with SZ*", IEEE IPDPS. pp. 730–739 (2016)

[3] Zhao,Di,Dmitriev,Tonellot,Chen,Cappello: "*Opt. Error-Bounded Lossy Comp. for Sci.Data by Dyn.Spline Interp.*", IEEE 37th ICDE, 1643–1654 (2021)

[4] `https://blosc.org`

[5] Ainsworth,Tugluk,Whitney,Klasky: "*Multilevel tech. for compression and reduction of sci.data – the univariate case*". Comp.Vis.Sci. 19, 65–76 (2018)

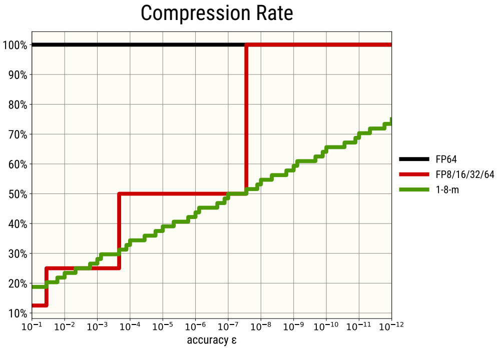
Floating Point Compression

Compression also possible within IEEE-754 scheme by choosing

- mantissa bits m based on accuracy

	s-e-m	u	Range[1]
FP64	1-11-52	$1 \cdot 10^{-16}$	631
FP32	1-8-23	$6 \cdot 10^{-8}$	83
TF32	1-8-10	$5 \cdot 10^{-4}$	79
BF16	1-8-7	$4 \cdot 10^{-3}$	78
FP16	1-5-10	$5 \cdot 10^{-4}$	12
FP8	1-4-3	$6 \cdot 10^{-2}$	5

[1] Dynamic range as $\log_{10} \frac{V_{\max}}{V_{\min}}$



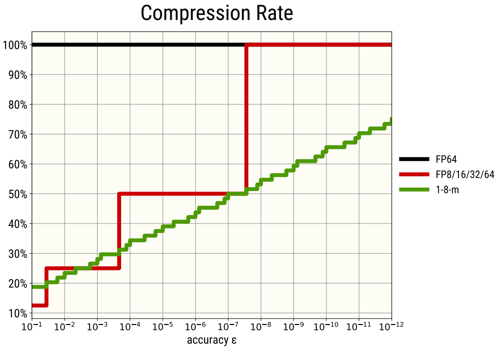
Floating Point Compression

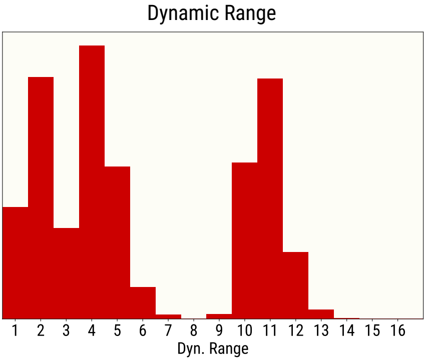
Compression also possible within IEEE-754 scheme by choosing

- mantissa bits m based on accuracy and
- exponent bits e based on dyn. range.

	s-e-m	u	Range[1]
FP64	1-11-52	$1 \cdot 10^{-16}$	631
FP32	1-8-23	$6 \cdot 10^{-8}$	83
TF32	1-8-10	$5 \cdot 10^{-4}$	79
BF16	1-8-7	$4 \cdot 10^{-3}$	78
FP16	1-5-10	$5 \cdot 10^{-4}$	12
FP8	1-4-3	$6 \cdot 10^{-2}$	5

[1] Dynamic range as $\log_{10} \frac{V_{\max}}{V_{\min}}$





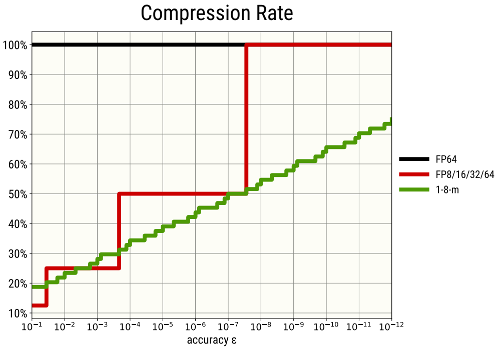
Floating Point Compression

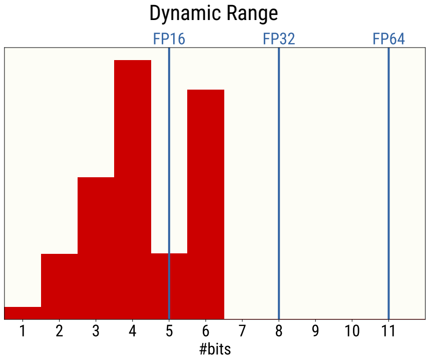
Compression also possible within IEEE-754 scheme by choosing

- mantissa bits m based on accuracy and
- exponent bits e based on dyn. range.

	s-e-m	u	Range[1]
FP64	1-11-52	$1 \cdot 10^{-16}$	631
FP32	1-8-23	$6 \cdot 10^{-8}$	83
TF32	1-8-10	$5 \cdot 10^{-4}$	79
BF16	1-8-7	$4 \cdot 10^{-3}$	78
FP16	1-5-10	$5 \cdot 10^{-4}$	12
FP8	1-4-3	$6 \cdot 10^{-2}$	5

[1] Dynamic range as $\log_{10} \frac{V_{\max}}{V_{\min}}$





Floating Point Compression

Compression also possible within IEEE-754 scheme by choosing

- mantissa bits *m* based on accuracy and
- exponent bits *e* based on dyn. range.

	s-e-m	u	Range[1]
FP64	1-11-52	$1 \cdot 10^{-16}$	631
FP32	1-8-23	$6 \cdot 10^{-8}$	83
TF32	1-8-10	$5 \cdot 10^{-4}$	79
BF16	1-8-7	$4 \cdot 10^{-3}$	78
FP16	1-5-10	$5 \cdot 10^{-4}$	12
FP8	1-4-3	$6 \cdot 10^{-2}$	5

[1] Dynamic range as $\log_{10} \frac{V_{\max}}{V_{\min}}$

AFL:

- fully adaptive choice of *m* and *e*,
- use 1-e-m to store data (with scaling and shifting),
- *slow* bit stream storage.

AFLP:

- choose *e* and *m* as in *AFL*,
- increase *m* such that $1 + e + m$ is multiple of 8

Adaptive Precision for Low-Rank Data

Adaptive Precision for Low-Rank Data

Given $||M_{\tau,\sigma} - U_{\tau,\sigma} V_{\tau,\sigma}^H|| \leq \delta$ and p floating point formats and

$$U_{\tau,\sigma} V_{\tau,\sigma}^H = W \Sigma X^H = \left(W_1 \dots W_p \right) \begin{pmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_p \end{pmatrix} \left(X_1 \dots X_p \right)^H$$

with unit roundoffs $u_1, \dots, u_p$ such that

$$||\Sigma_i|| \leq \frac{\delta}{u_i}$$

Let $\widetilde{M}_{\tau,\sigma}$ be the representation of $M_{\tau,\sigma}$ where W_i, X_i are stored in the i'th floating point format. Then the error $||M_{\tau,\sigma} - \widetilde{M}_{\tau,\sigma}||$ is bounded by[1]

$$||M_{\tau,\sigma} - \widetilde{M}_{\tau,\sigma}|| \leq \delta + \left(2(p-1) + \sum_{i=2}^{p} \sqrt{k_i} u_i \right) \delta$$

[1] Amestoy, Boiteau, Buttari, Gerest, Jézéquel, L'Excellent, Mary: "*Mixed precision low-rank approximations and their application to block low-rank LU factorization*", IMA J. of Num. Analysis, 2022

Adaptive Precision for Low-Rank Data

Replace the predefined IEEE-754 formats by a general compression scheme with adaptive error control.

Adaptive Precision for Low-Rank (APLR)

For all columns (w_i, x_i) of W/X *choose* precision $\widetilde{u}_i$ such that

$$\widetilde{u}_i = \frac{\delta}{\sigma_i}$$

The error $||M_{\tau,\sigma} - \widetilde{M}_{\tau,\sigma}||$ becomes

$$||M_{\tau,\sigma} - \widetilde{M}_{\tau,\sigma}|| \leq \delta + 2k\delta + \delta^2 \sum_{i=1}^{k} \frac{1}{\sigma_i}$$

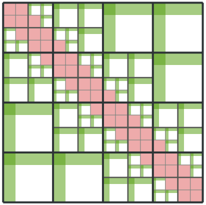
Remark

Dense matrix blocks are directly compressed.

Adaptive Precision for Low-Rank Data

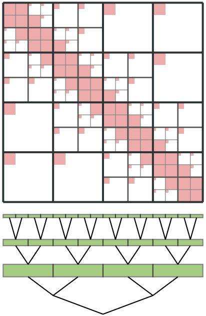
Depending on the $\mathcal{H}$–matrix format, APLR can be applied to different data.

$\mathcal{H}$



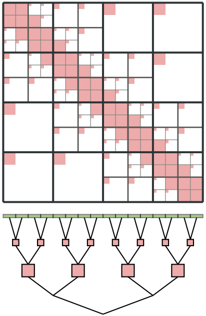
all *lowrank blocks*
$(\mathcal{O}(n \log n))$

Uniform-$\mathcal{H}$



all *cluster bases*
$(\mathcal{O}(n \log n))$

$\mathcal{H}^2$

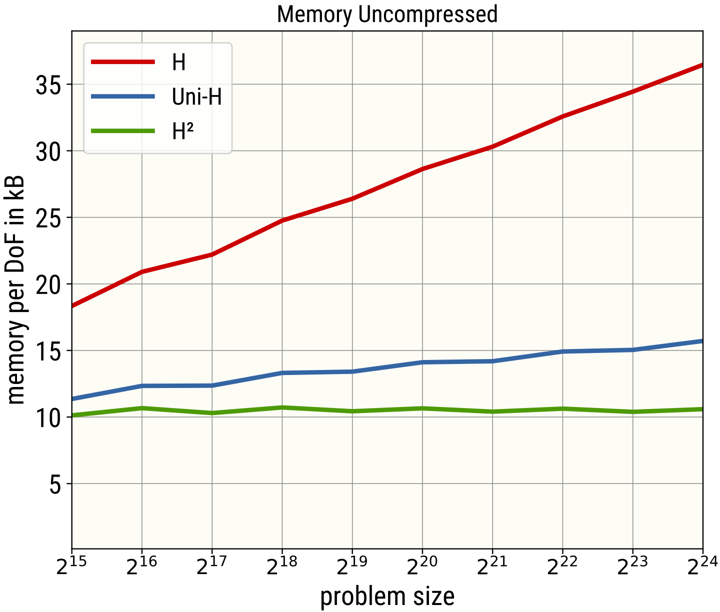


all *leaf* cluster bases
$(\mathcal{O}(n))$

For everything else, standard compression can be applied.

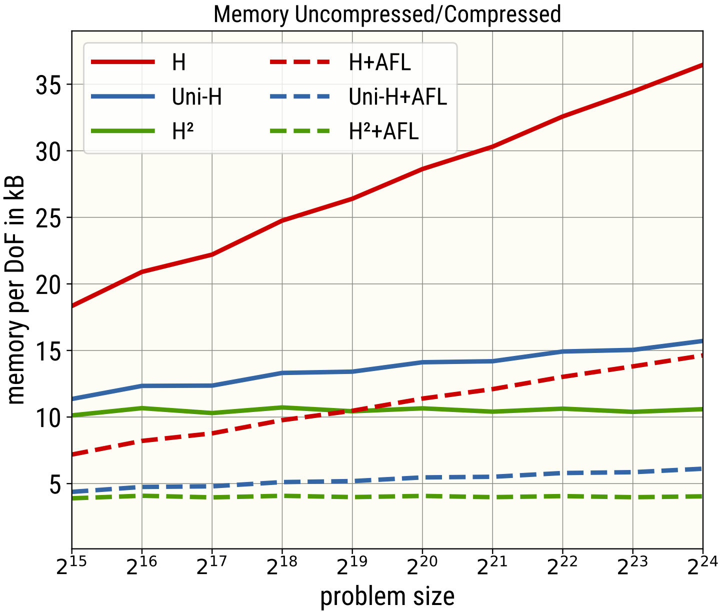
Adaptive Precision for Low-Rank Data

Laplace SLP ($\varepsilon = 10^{-6}$)



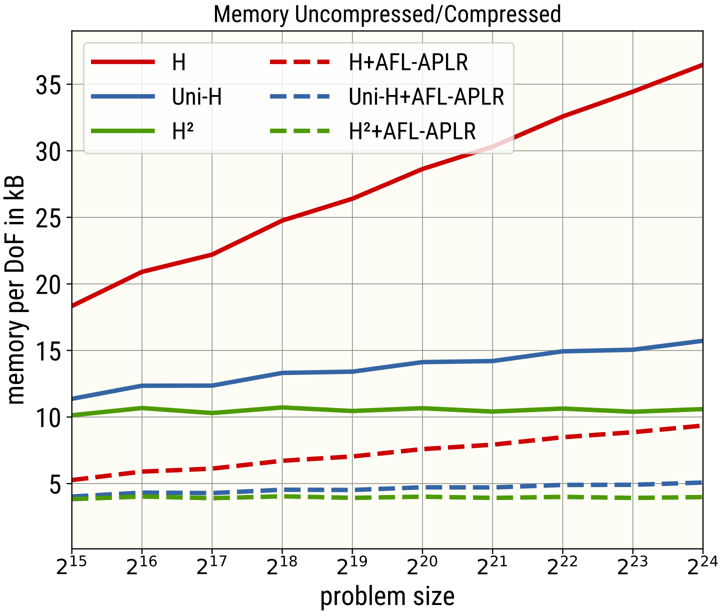
Adaptive Precision for Low-Rank Data

Laplace SLP ($\varepsilon = 10^{-6}$, *AFL*)



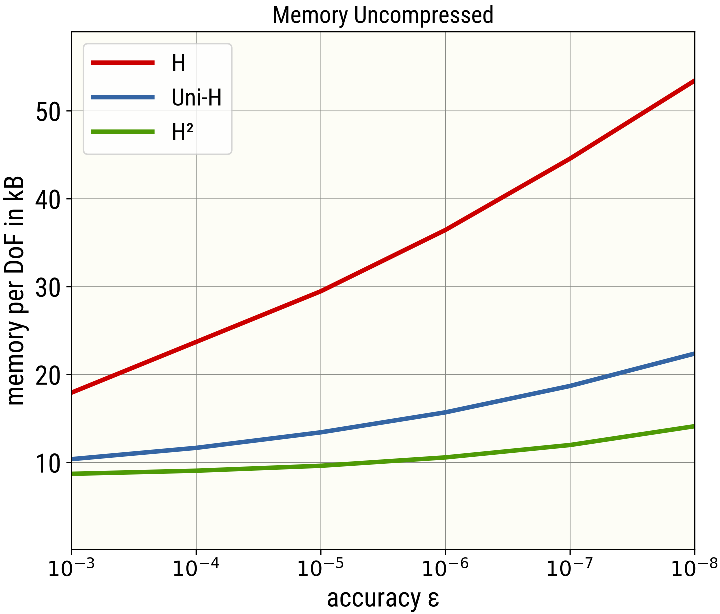
Adaptive Precision for Low-Rank Data

Laplace SLP ($\varepsilon = 10^{-6}$, AFL+*APLR*)



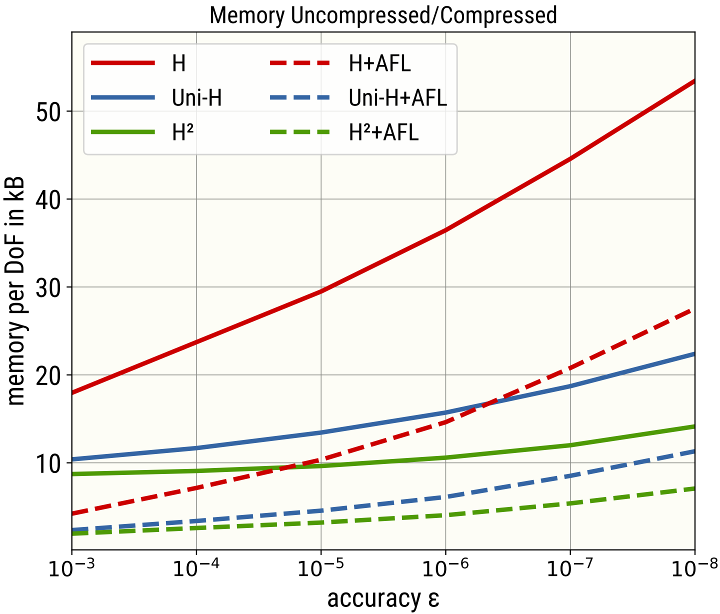
Adaptive Precision for Low-Rank Data

Laplace SLP ($n = 16.777.216$)



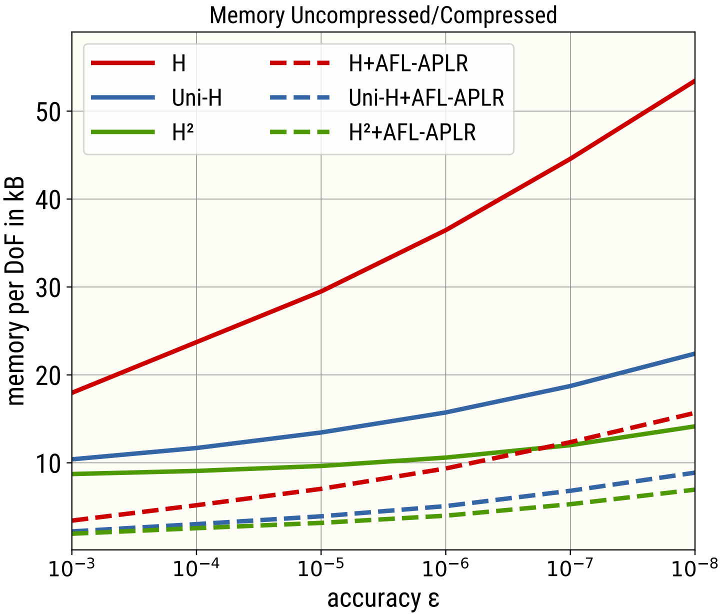
Adaptive Precision for Low-Rank Data

Laplace SLP ($n = 16.777.216$, *AFL*)



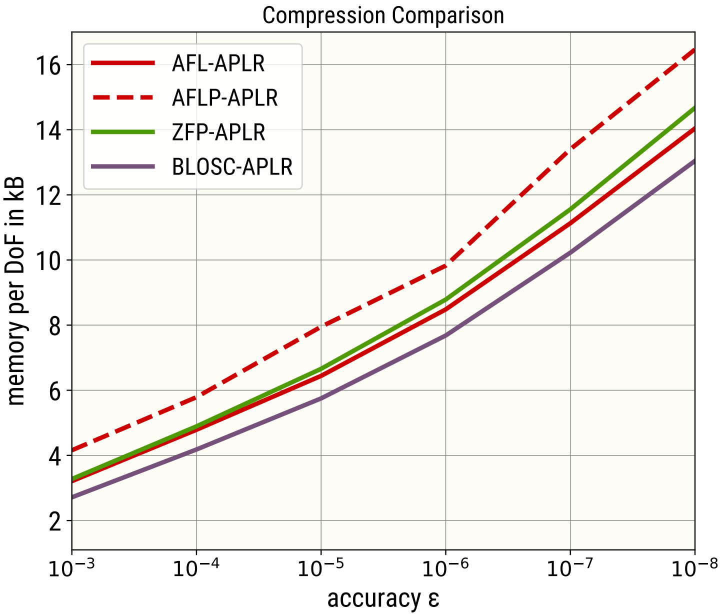
Adaptive Precision for Low-Rank Data

Laplace SLP ($n = 16.777.216$, AFL+*APLR*)



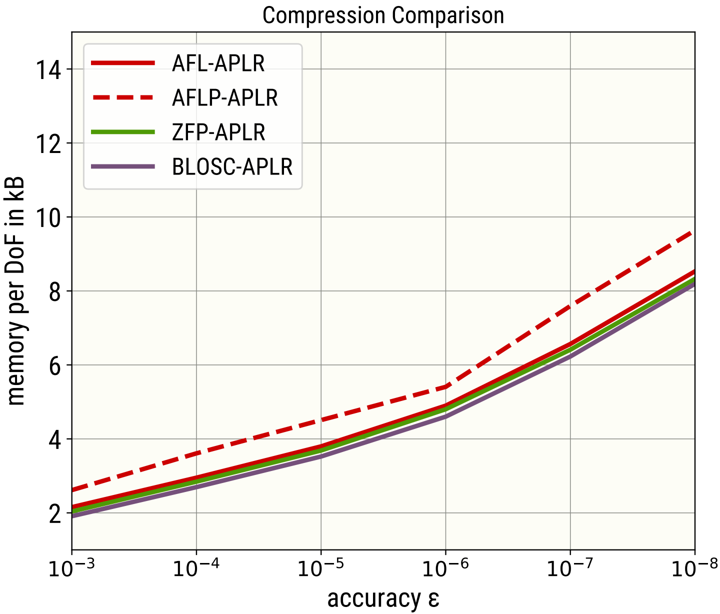
Adaptive Precision for Low-Rank Data

Laplace SLP ($n = 1.048.576$, X+APLR, $\mathcal{H}$)



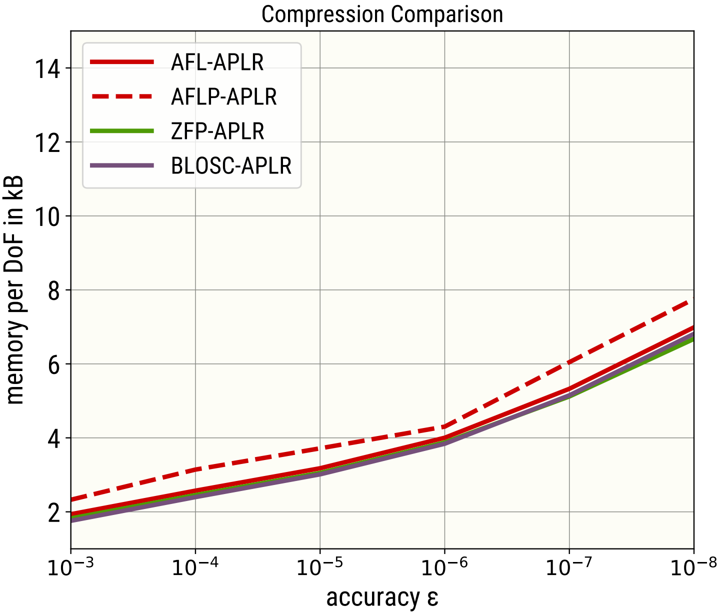
Adaptive Precision for Low-Rank Data

Laplace SLP ($n = 1.048.576$, X+APLR, Uniform-$\mathcal{H}$)



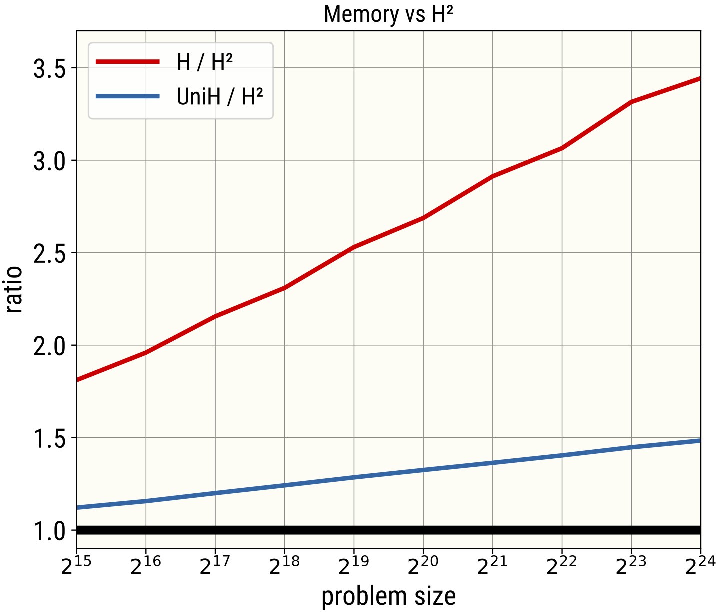
Adaptive Precision for Low-Rank Data

Laplace SLP ($n = 1.048.576$, X+APLR, $\mathcal{H}^2$)



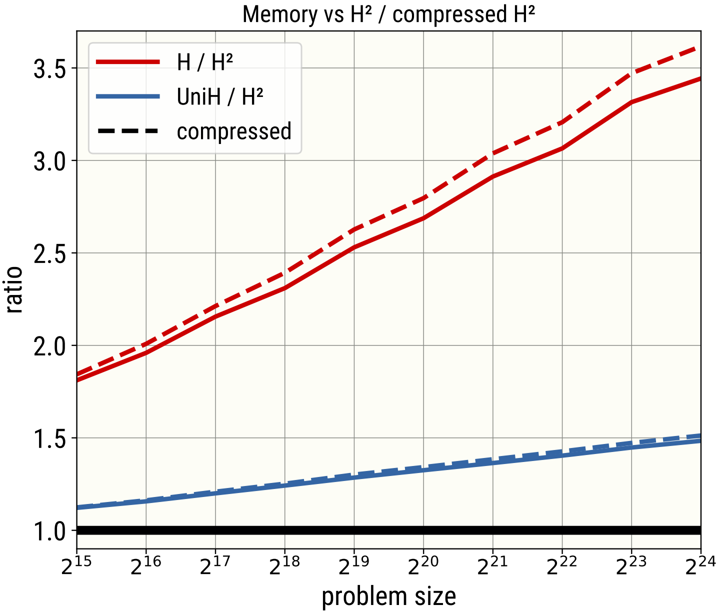
Adaptive Precision for Low-Rank Data

Laplace SLP ($\varepsilon = 10^{-6}$)



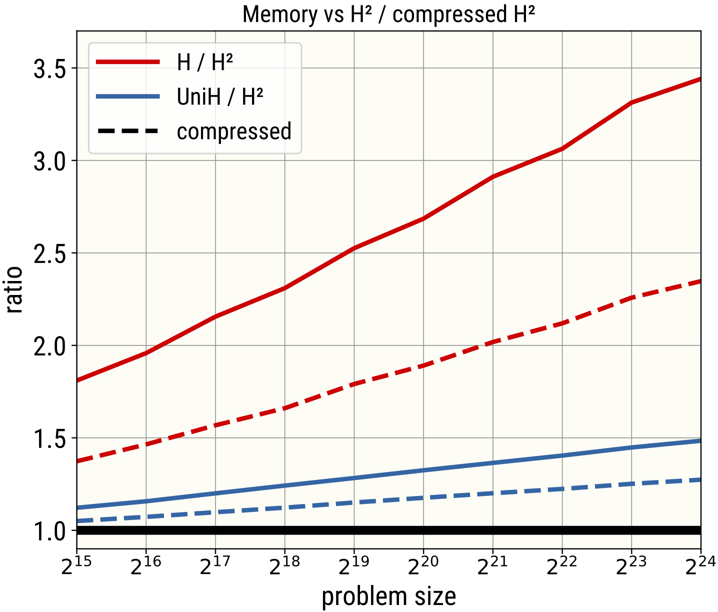
Adaptive Precision for Low-Rank Data

Laplace SLP ($\varepsilon = 10^{-6}$, *AFL*)



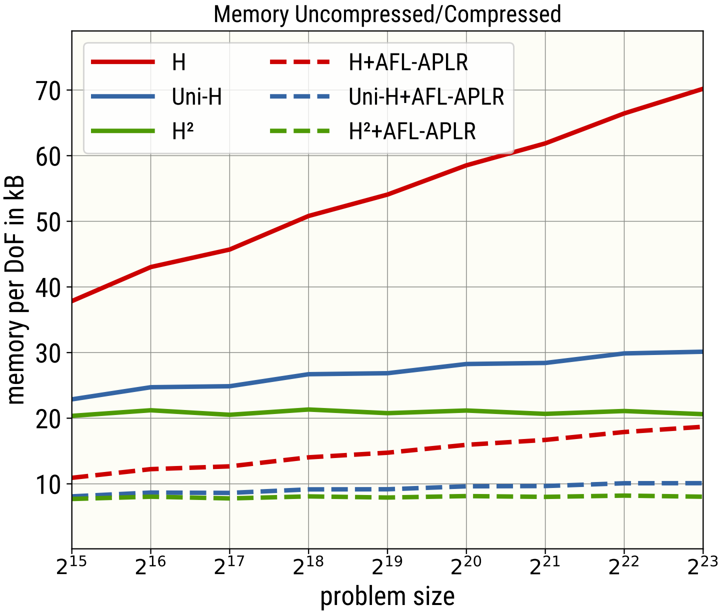
Adaptive Precision for Low-Rank Data

Laplace SLP ($\varepsilon = 10^{-6}$, AFL+*APLR*)



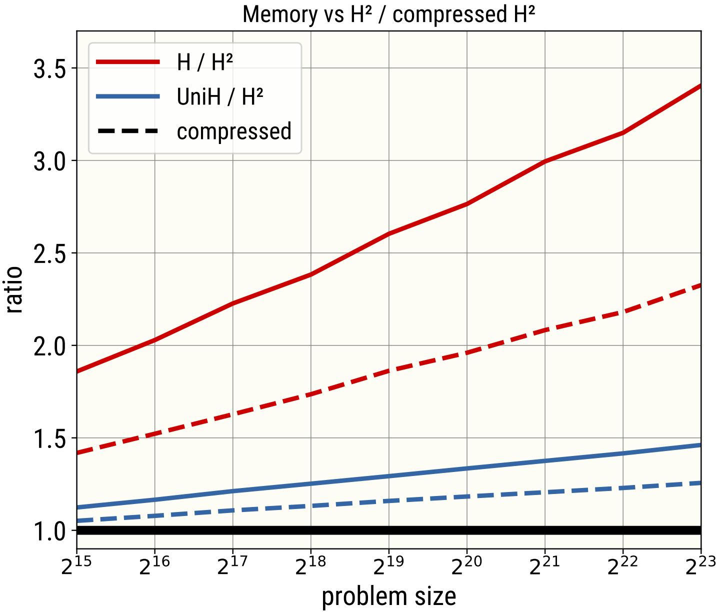
Adaptive Precision for Low-Rank Data

Helmholtz SLP ($\varepsilon = 10^{-6}, \kappa = 2$, AFL+APLR)

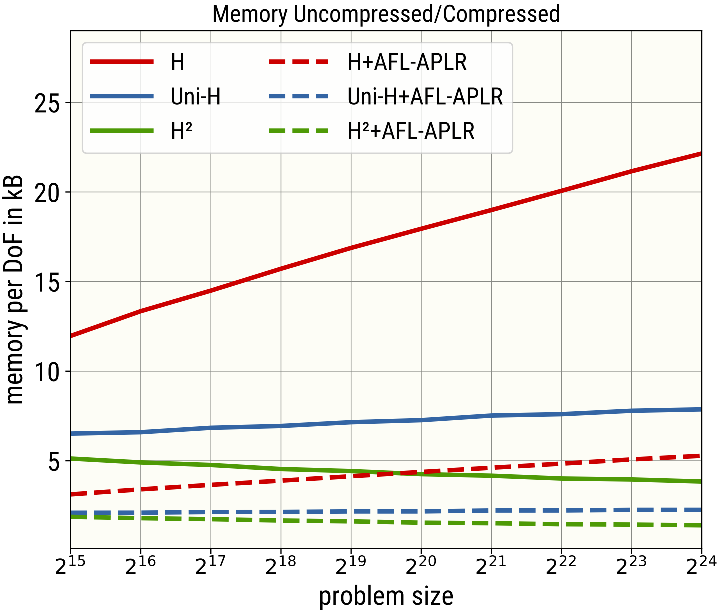


Adaptive Precision for Low-Rank Data

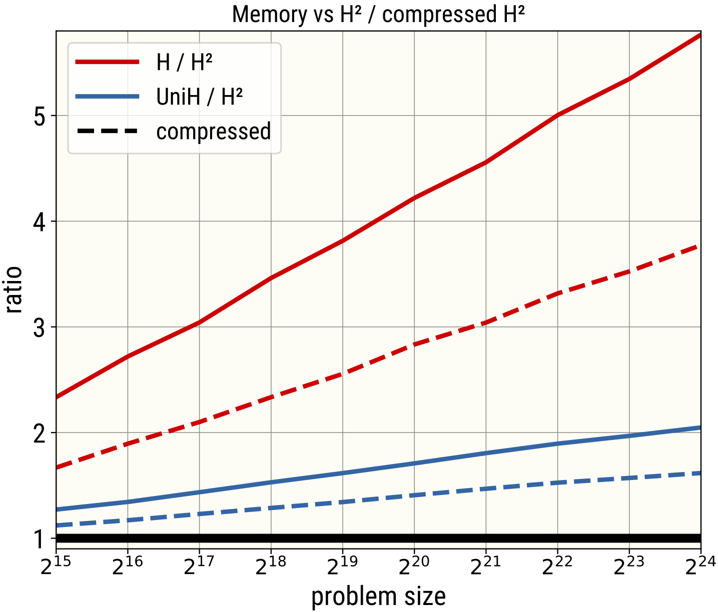
Helmholtz SLP ($\varepsilon = 10^{-6}, \kappa = 2$, AFL+APLR)



Matérn Covariance ($\varepsilon = 10^{-6}$, AFL+APLR)



Matérn Covariance ($\varepsilon = 10^{-6}$, AFL+APLR)



$\mathcal{H}$-Arithmetic

$\mathcal{H}$-Arithmetic

Decoupling of storage and compute precision[1]:

- compression only for storage,
- all computations in FP64.

Use *function-level* conversion due to BLAS/ LAPACK based arithmetic.

```
function MVM(in: U, V, x, inout: y)
    V_d := decompress(V);
    t := V_d^H x;
    U_d := decompress(U);
    y := y + U_d t;
```

```
function TRUNCATE(in: U, V, ε, out: W, X)
    U^d := decompress(U);
    V^d := decompress(V);
    [Q_U, R_U] := qr( U^d );
    [Q_V, R_V] := qr( V^d );
    [U_s, S_s, V_s] := svd( R_U · R_V^H );
    k := rank(S_s, ε);
    W^d := Q_U · U_s(:, 1 : k) · S_s(1 : k, 1 : k);
    X^d := Q_V · V_s(:, 1 : k);
    W := compress(W^d);
    X := compress(X^d);
```

[1] Anzt, Flegar, Grützmacher, Quintana-Ortí: "*Toward a modular precision ecosystem for high-performance computing*", Int. J. of HPC Applications, 33(6), 1069–1078, 2019.

$\mathcal{H}$-Arithmetic

Decoupling of storage and compute precision[1]:

- compression only for storage,
- all computations in FP64.

Use *function-level* conversion due to BLAS/LAPACK based arithmetic.

Alternative: decompress and compute *on-the-fly* for matrix-vector multiplication.

```
function MVM(in: U, V, x, inout: y)
    V_d := decompress(V);
    t := V_d^H x;
    U_d := decompress(U);
    y := y + U_d t;
```

```
function MVM_AFLP(in: U, V, x, inout: y)
    t := 0;
    for 0 ≤ ℓ < k do
        for 0 ≤ j < m do
            t_ℓ := t_ℓ + decompress(V_jℓ) x_j;
    for 0 ≤ ℓ < k do
        for 0 ≤ i < n do
            y_i := y_i + decompress(U_iℓ) t_ℓ;
```

[1] Anzt, Flegar, Grützmacher, Quintana-Ortí: "*Toward a modular precision ecosystem for high-performance computing*", Int. J. of HPC Applications, 33(6), 1069–1078, 2019.

Decoupling of storage and compute precision[1]:

- compression only for storage,
- all computations in FP64.

Use *function-level* conversion due to BLAS/LAPACK based arithmetic.

Alternative: decompress and compute *on-the-fly* for matrix-vector multiplication.

```
function MVM(in: U, V, x, inout: y)
    V_d := decompress(V);
    t := V_d^H x;
    U_d := decompress(U);
    y := y + U_d t;
```

```
function MVM_AFLP(in: U, V, x, inout: y)
    t := 0;
    for 0 ≤ ℓ < k do
        for 0 ≤ j < m do
            t_ℓ := t_ℓ + decompress(V_jℓ) x_j;
    for 0 ≤ ℓ < k do
        for 0 ≤ i < n do
            y_i := y_i + decompress(U_iℓ) t_ℓ;
```

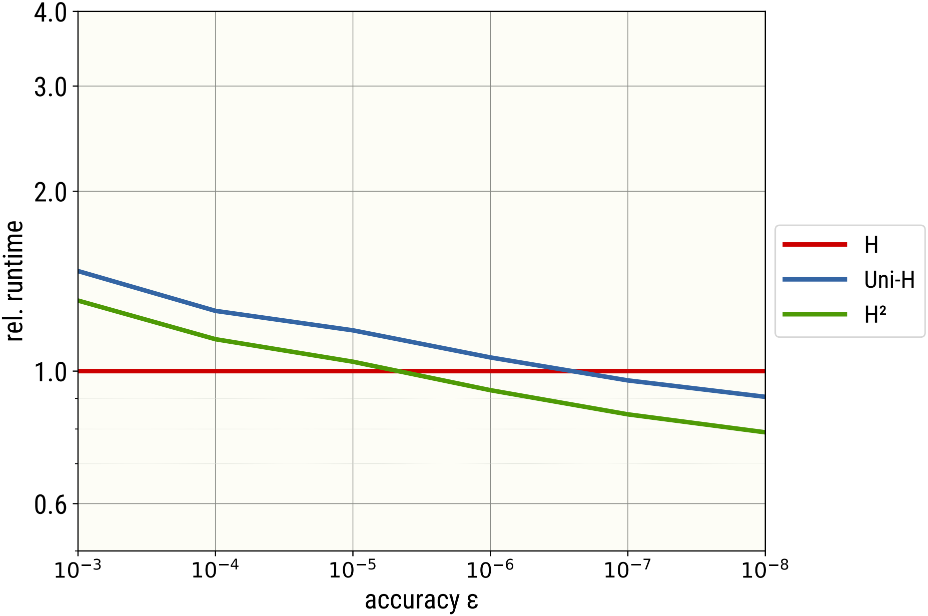
Hardware/Software

- *2x64*-core AMD Epyc 9554 with *2x12* 32GB DDR5-4800 DIMMs
- libHLR + oneTBB + oneMKL (AVX512)

[1] Anzt, Flegar, Grützmacher, Quintana-Ortí: "*Toward a modular precision ecosystem for high-performance computing*", Int. J. of HPC Applications, 33(6), 1069–1078, 2019.

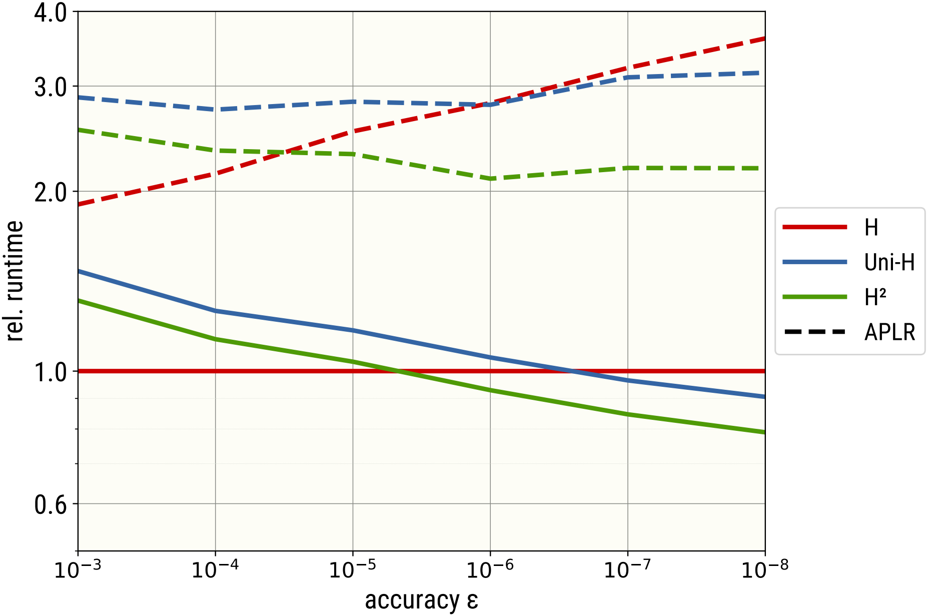
Matrix–Vector Multiplication

Laplace SLP ($n = 1.048.576$)



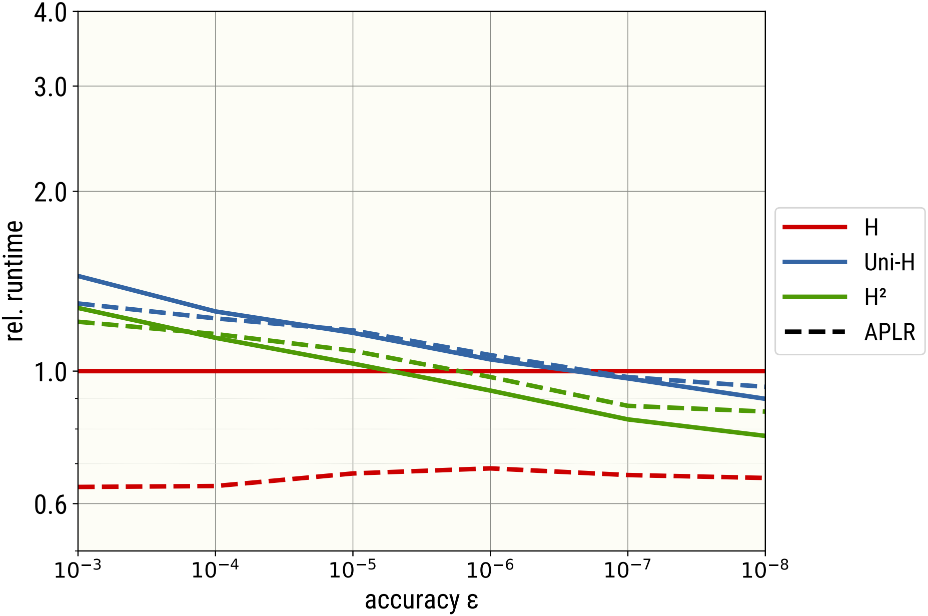
Matrix–Vector Multiplication

Laplace SLP ($n = 1.048.576$, *AFL+APLR*)



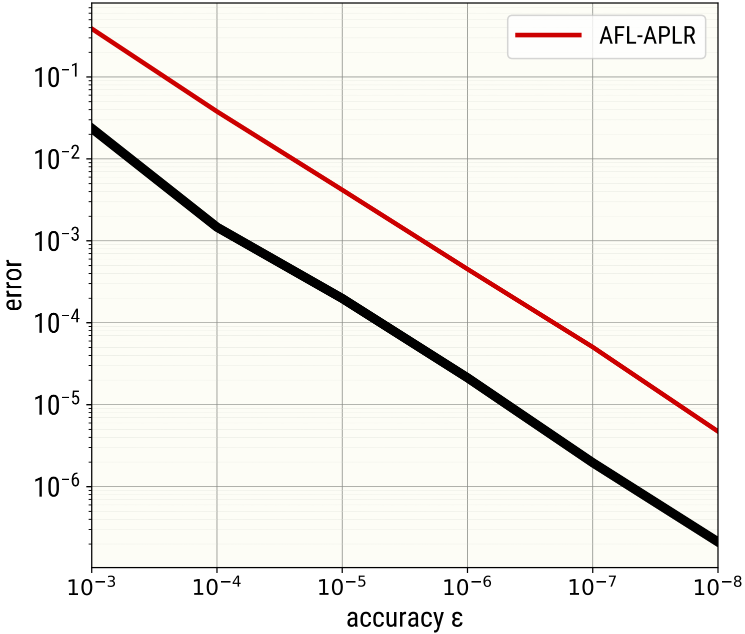
Matrix–Vector Multiplication

Laplace SLP ($n = 1.048.576$, *AFLP*+APLR, *on–the–fly*)



$\mathcal{H}$-LU Factorization

$\mathcal{H}$-LU Inversion Error $||I - M \cdot (LU)^{-1}||_2$



$\mathcal{H}$-LU Factorization

Problem

A significant error increase with standard $\mathcal{H}$-arithmetic.

Options

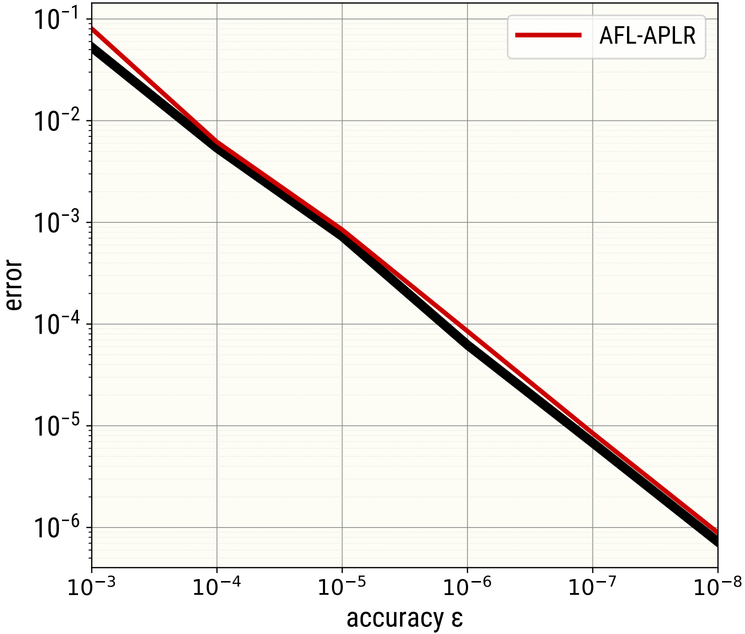
1. tighter accuracy settings for compression during $\mathcal{H}$-arithmetic or
2. use *accumulator based $\mathcal{H}$-arithmetic*[1] without compression of accumulator matrices.

function HMUL$(C_{\tau,\sigma}, \mathcal{A}_{\tau,\sigma}, \mathcal{P}_{\tau,\sigma})$
  for all updates $(A_{\tau,\rho}, B_{\rho,\sigma}) \in \mathcal{P}_{\tau,\sigma}$ **do**
    if $A_{\tau,\rho}/B_{\rho,\sigma}$ are dense/lowrank **then**
      $\mathcal{A}_{\tau,\sigma} := \mathcal{A}_{\tau,\sigma} + A_{\tau,\rho}B_{\rho,\sigma}$;
      $\mathcal{P}_{\tau,\sigma} := \mathcal{P}_{\tau,\sigma} \setminus \{(A_{\tau,\rho}, B_{\rho,\sigma})\}$;
  if $C_{\tau,\sigma}$ is structured **then**
    for all subblocks C_{τ_i,σ_j} **do**
      `hmul(`$C_{\tau_i,\sigma_j}, \mathcal{A}_{\tau,\sigma}|_{\tau_i,\sigma_j}, \mathcal{P}_{\tau,\sigma}|_{\tau_i,\sigma_j}$`);`
  else
    $C_{\tau,\sigma} := C_{\tau,\sigma} + \mathcal{A}_{\tau,\sigma}$;   *// Compression/Decompression*

[1] Börm: "*Hierarchical matrix arithmetic with accumulated updates*", CVS 20, 71–84, 2019

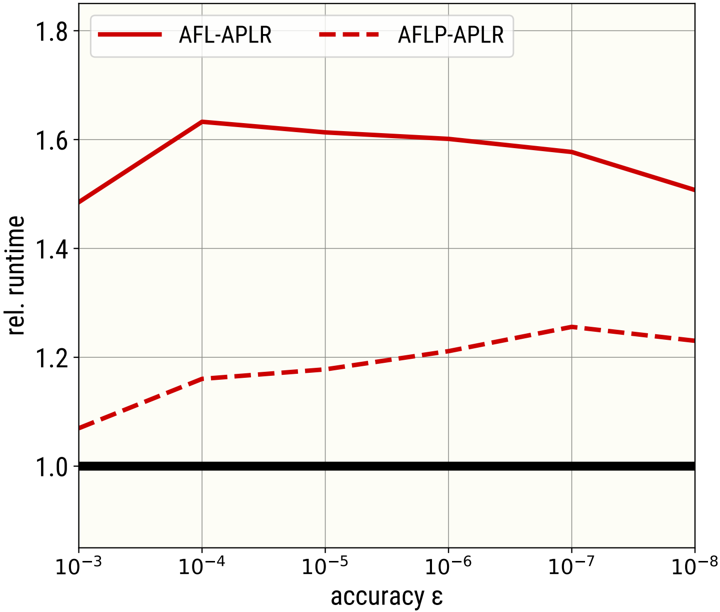
$\mathcal{H}$-LU Factorization

$\mathcal{H}$-LU Inversion Error $||I - A \cdot (LU)^{-1}||_2$ with Accumulator



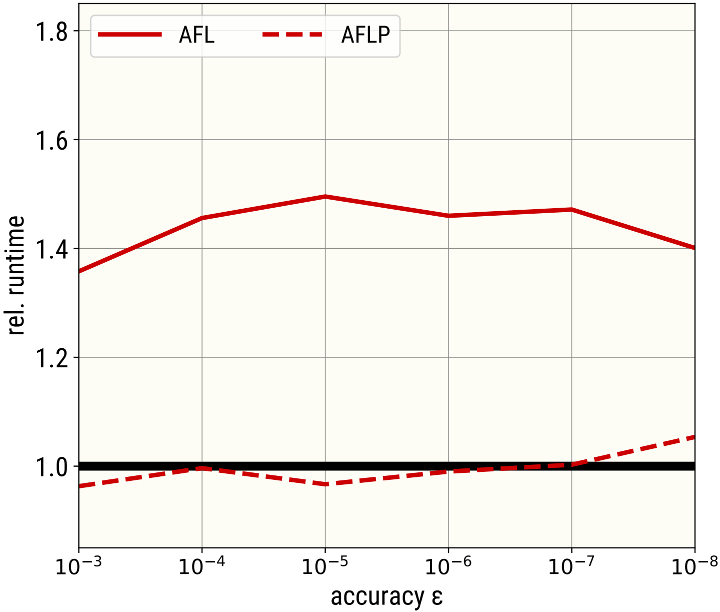
$\mathcal{H}$-LU Factorization

Laplace SLP ($n = 1.048.576$, AFL/AFLP, w/ APLR)



$\mathcal{H}$-LU Factorization

Laplace SLP ($n = 1.048.576$, AFL/AFLP, *w/o APLR*)



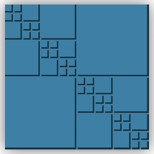
Conclusion

Conclusion

By using floating point compression storage for hierarchical lowrank matrices can be

- *significantly reduced*
- with *small impact* on (parallel) performance

The memory gap between the $\mathcal{H}$-matrices, Uniform-$\mathcal{H}$-matrices and $\mathcal{H}^2$-matrices can also be reduced by using *adaptive precision* compression for *lowrank* matrices.



libHLR.org

arxiv.org

Future Work

- adjustments to error control,
- more arithmetic with on-the-fly decompression

Thank You

MAX PLANCK INSTITUTE
FOR MATHEMATICS IN THE SCIENCES