Hierarchical Lowrank Arithmetic with Binary Compression

Ronald Kriemann
Max Planck Institute MIS Leipzig

ICIAM 2023

MAX PLANCK INSTITUTE
FOR MATHEMATICS IN THE SCIENCES

Hierarchical Matrices

Hierarchical Matrices

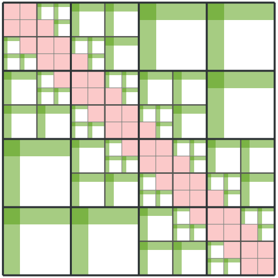
Approximate dense data $M_{\tau,\sigma} \in \mathbb{C}^{\#\tau \times \#\sigma}$ of $M \in \mathbb{C}^{n \times n}$ by

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

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

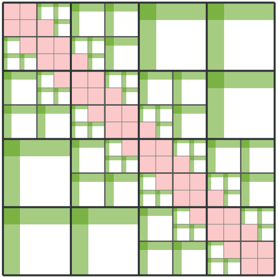
$$||M_{\tau,\sigma} - U_{\tau,\sigma} V_{\tau,\sigma}^H|| \leq \delta \qquad \text{or}$$

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



For M this yields an approximation $\widetilde{M}$ with $\mathcal{O}(n \log n)$ storage.

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

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

For M this yields an approximation $\widetilde{M}$ with $\mathcal{O}(n \log n)$ storage.

However, the datablocks in $\widetilde{M}$ (dense blocks and lowrank factors) are typically stored in FP64 (or FP32) even though for the *unit roundoff* u_{FP64} we normally have

$$u_{\text{FP64}} \ll \delta$$

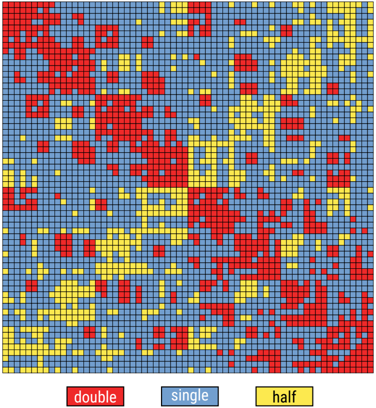
(or $u_{\text{FP64}} \ll |m_{ij} - \widetilde{m}_{ij}| \leq \delta / n$ for Frobenius norm and uniform error distribution).

Mixed Precision Storage

Mixed Precision in 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.



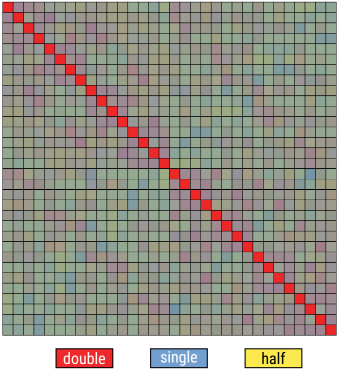
[1] Abdulah, Cao, Pei, Bosilca, Dongarra, Genton, Keyes, Ltaief, Sun: "Accelerating Geostatistical Modeling and Prediction With Mixed-Precision Computations: A High-Productivity Approach With PaRSEC", IEEE Trans. on Par. and Distr. Systems, 2022

Mixed Precision in 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.



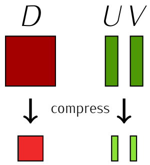
[1] Ooi, Iwashita, Fukaya, Ida, Yokota.: "Effect of Mixed Precision Computing on H-Matrix Vector Multiplication in BEM Analysis", Proceedings of HPCAsia2020, 2020

[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

Compression Libraries

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[1].



ZFP[2]

- bitplane truncation for 4^d blocks,
- fast,
- reliable error control only with fixed bitrate,
- limited compression rate.

SZ[3]/SZ3[4]

- uses curve fitting,
- good compression rates for general data,
- various error control options,
- issues with thread usage and compression rate.

MGARD[5]

- multi-grid technique plus lossless compression,
- various error control options,
- slow, issues with compression rate.

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

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

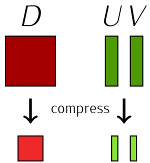
[3] Di, Cappello: "Fast Error-Bounded Lossy HPC Data Compression with SZ", IEEE IPDPS. pp. 730–739 (2016)

[4] Zhao, Di, Dmitriev, Tonellot, Chen, Cappello: "Optimizing Error-Bounded Lossy Compression for Scientific Data by Dynamic Spline Interpolation", IEEE 37th ICDE, 1643–1654 (2021)

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

Compression Libraries

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[1].



ZFP[2]

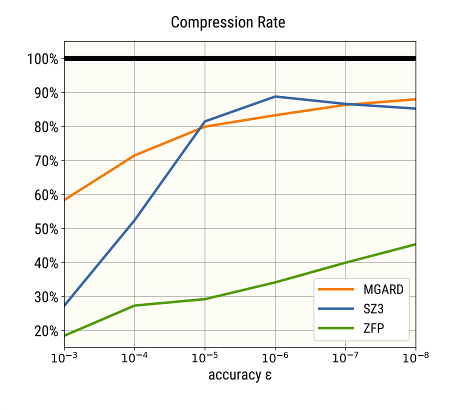
- bitplane truncation for 4^d blocks,
- fast,
- reliable error control only with fixed bitrate,
- limited compression rate.

SZ[3]/SZ3[4]

- uses curve fitting,
- good c
genera
- various
options
- issues
and co

MGARD[5]

- multi-grid technique plus
,



---

[1] K., Ltaief, Luong, Pérez, Im, Keyes: "*High-Performance Spatial Data*

[2] Lindstrom: "*Fixed-rate compressed floating-point arrays*", IEEE Tran

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

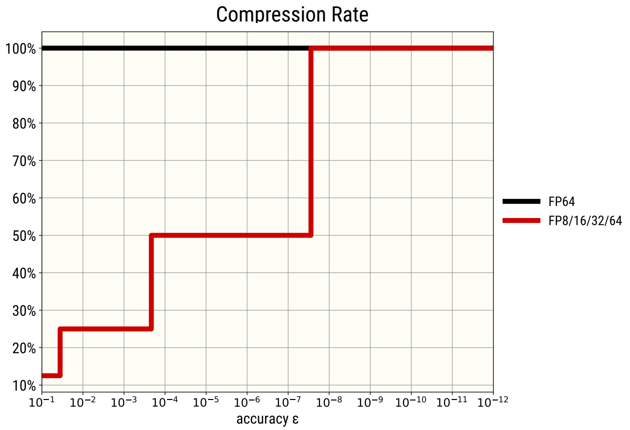
[4] Zhao, Di, Dmitriev, Tonellot, Chen, Cappello: "Optimizing Error-Bou olation", IEEE 37th ICDE, 1643–1654 (2021)

[5] Ainsworth, Tugluk, Whitney, Klasky: "Multilevel techniques for comp is.Sci. 19, 65–76 (2018)

IEEE 754 based

Formats with big jumps in bitsize/precision.

	S-E-M	u
FP64	1-11-52	$1 \cdot 10^{-16}$
FP32	1-8-23	$6 \cdot 10^{-8}$
TF32	1-8-10	$5 \cdot 10^{-4}$
BF16	1-8-7	$4 \cdot 10^{-3}$
FP16	1-5-10	$5 \cdot 10^{-4}$
FP8	1-4-3	$6 \cdot 10^{-2}$



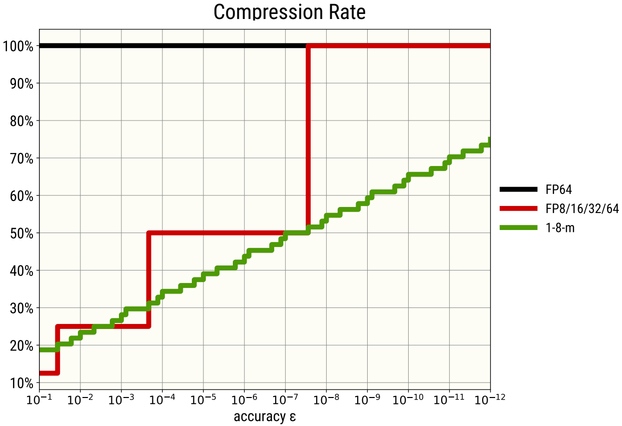
IEEE 754 based

Formats with big jumps in bitsize/precision.

Instead choose:

- mantissa bits *m* based on accuracy,

	S-E-M	u
FP64	1-11-52	$1 \cdot 10^{-16}$
FP32	1-8-23	$6 \cdot 10^{-8}$
TF32	1-8-10	$5 \cdot 10^{-4}$
BF16	1-8-7	$4 \cdot 10^{-3}$
FP16	1-5-10	$5 \cdot 10^{-4}$
FP8	1-4-3	$6 \cdot 10^{-2}$



IEEE 754 based

Formats with big jumps in bitsize/precision.

Instead choose:

- 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 $\log_{10} \frac{V_{\max}}{V_{\min}}$

IEEE 754 based

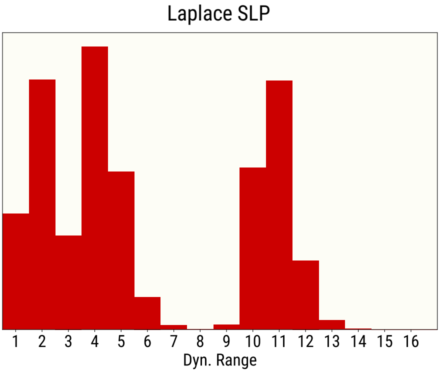
Formats with big jumps in bitsize/precision.

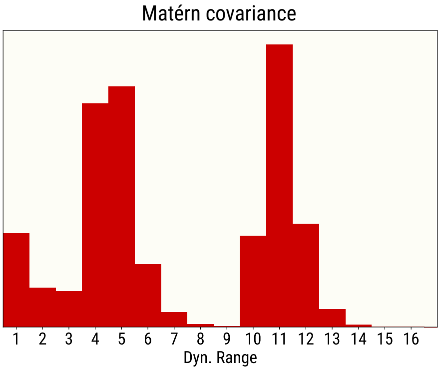
Instead choose:

- 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 $\log_{10} \frac{V_{\max}}{V_{\min}}$





IEEE 754 based

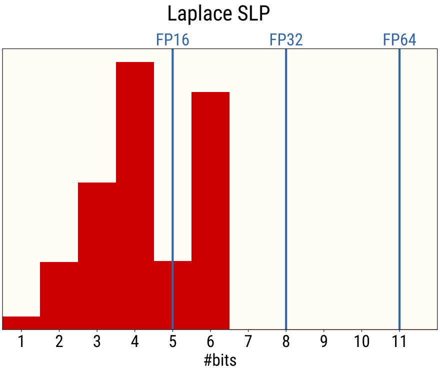
Formats with big jumps in bitsize/precision.

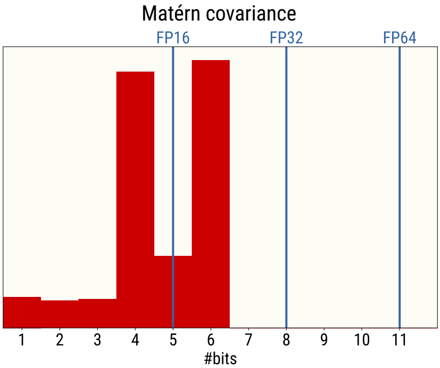
Instead choose:

- mantissa bits *m* based on accuracy,
- 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 $\log_{10} \frac{V_{\max}}{V_{\min}}$





IEEE 754 based

Formats with big jumps in bitsize/precision.

Instead choose:

- mantissa bits *m* based on accuracy,
- 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.

IEEE 754 based

Formats with big jumps in bitsize/precision.

Instead choose:

- mantissa bits *m* based on accuracy,
- 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

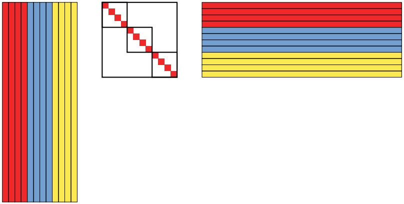
MP+Compression

MP+Compression

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

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

with unit roundoffs $u_1, \dots, u_p$ assuming $||\Sigma_i|| \leq \delta / u_i$.

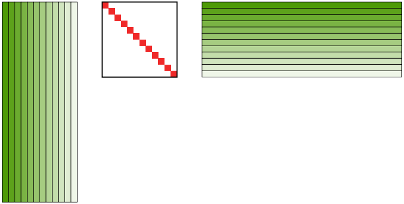


MP+Compression

Replace IEEE754 formats by compression scheme with adaptive error control:

Adaptive Precision compression for LowRank (APLR)

Choose precision $\widetilde{u}_i$ for columns (w_i, x_i) of W/X such that $\widetilde{u}_i \approx \delta/\sigma_i$.



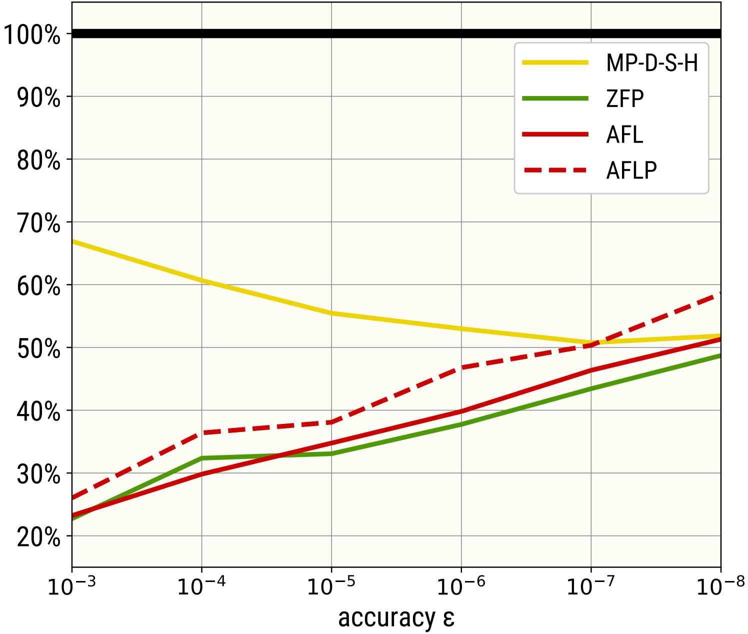
This yields *AFL-APLR*, *AFLP-APLR* and *ZFP-APLR*.

Remark

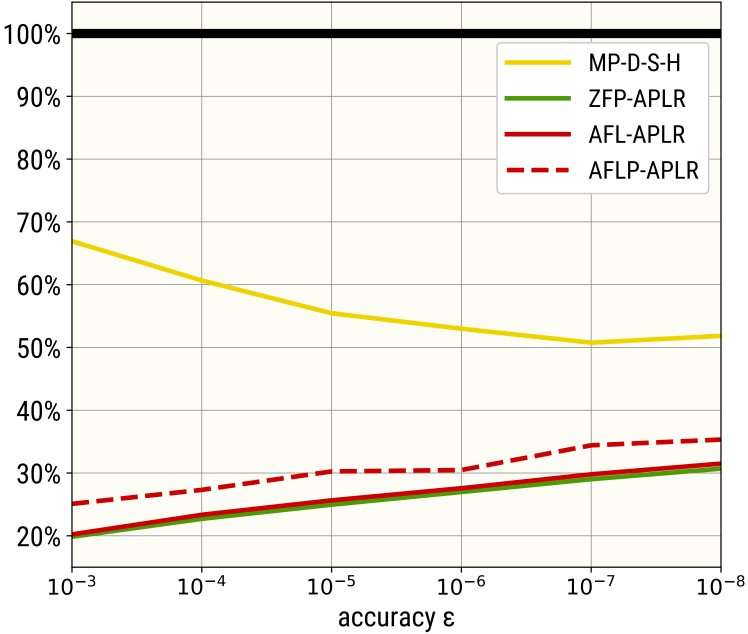
Dense matrix blocks are directly compressed via AFL/AFLP/ZFP.

Compression Rates

Laplace SLP, $n = 1.048.576$



Laplace SLP, $n = 1.048.576$



$\mathcal{H}$-Arithmetic

$\mathcal{H}$-Arithmetic

Decoupling of storage and compute precision[1], i.e., use compression scheme for storage only and do computations in FP64.

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

Aside from that, standard $\mathcal{H}$-arithmetic can be used.

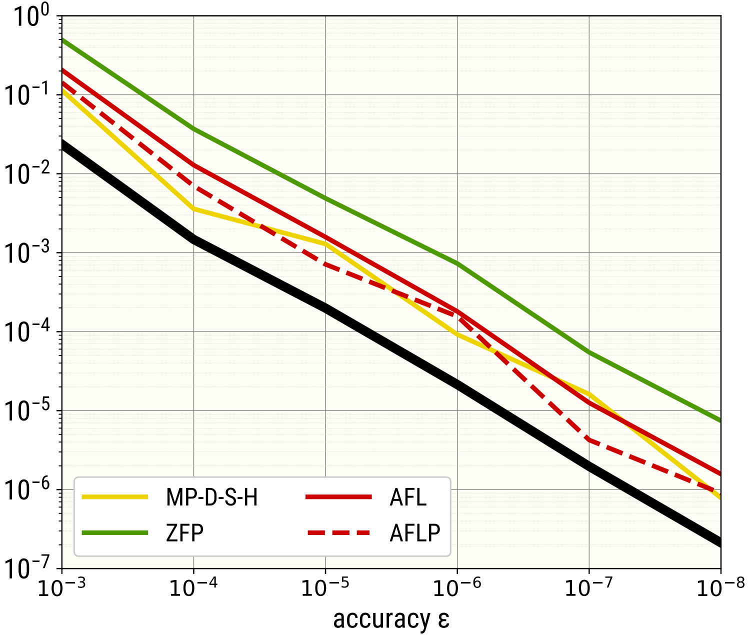
```
function TRUNCATION(in: U, V, ε, out: W, X)
    U^d := decompress(U);
    V^d := decompress(V);
    [Q_U, R_U] := qr( U^d );
    [Q_V, R_V] := qr( V^d );
    [U_s, S_s, V_s] := svd( R_U · R_V^H );
    k := rank(S_s, ε);
    W^d := Q_U · U_s(:, 1 : k) · S_s(1 : k, 1 : k);
    X^d := Q_V · V_s(:, 1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

APLR representation with special kernels and slightly more involved due to *reorthogonalization* of W and X factors.

[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.

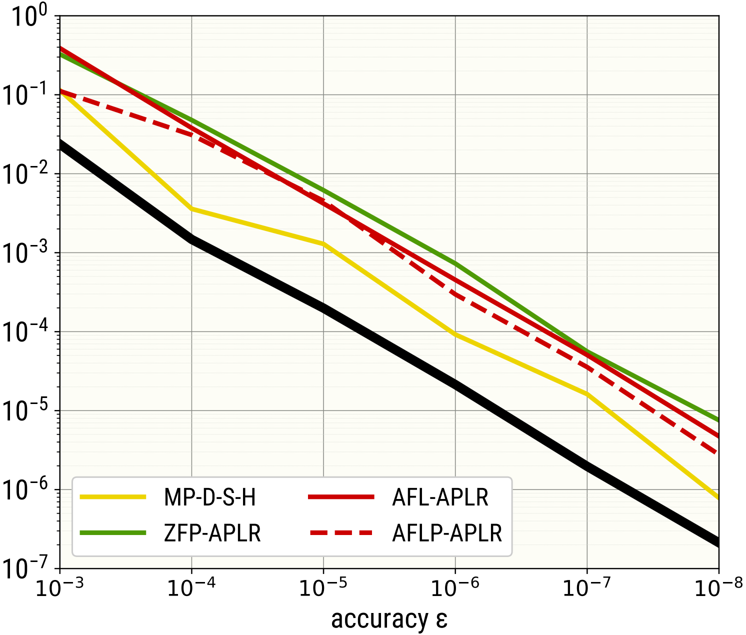
$\mathcal{H}$-LU Factorization

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



$\mathcal{H}$-LU Factorization

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



$\mathcal{H}$-LU Factorization

Problem

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

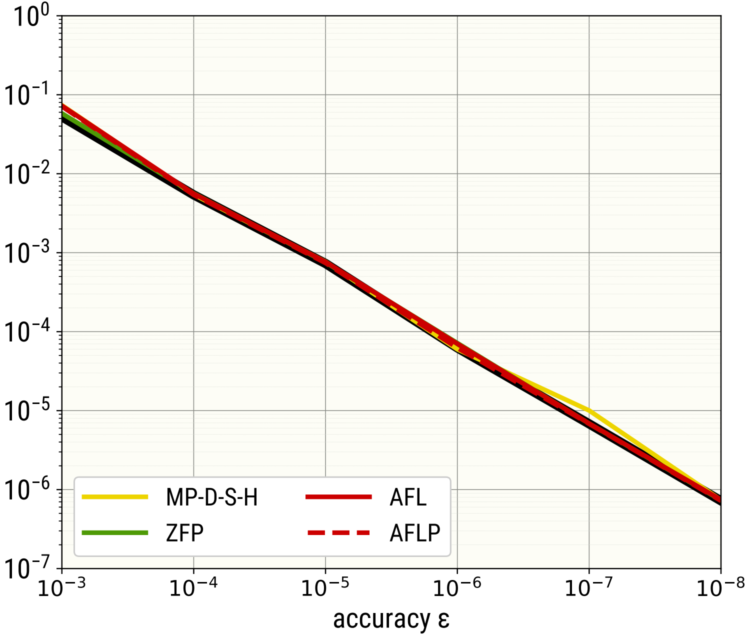
Options

1. tighter accuracy settings for compression during $\mathcal{H}$-arithmetic or
2. use *accumulator* based $\mathcal{H}$-arithmetic 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
    $C_{\tau,\sigma} := C_{\tau,\sigma} + \mathcal{A}_{\tau,\sigma}$;

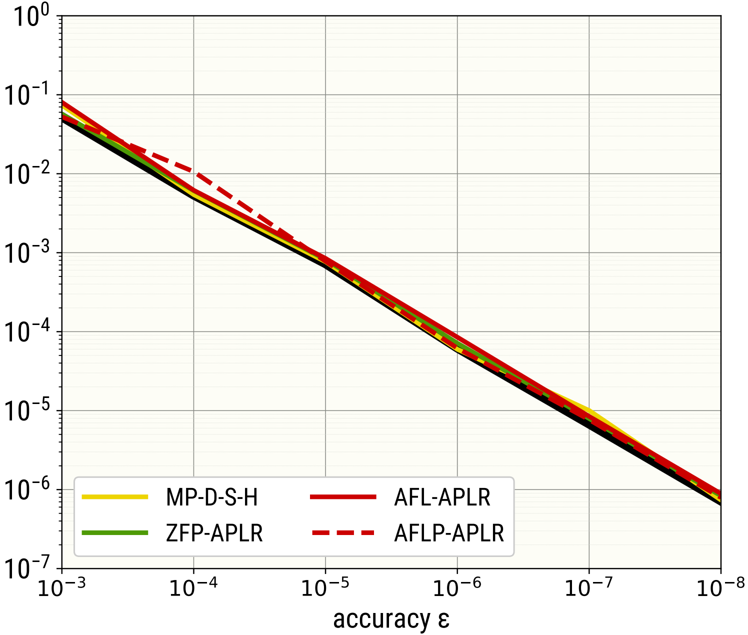
$\mathcal{H}$-LU Factorization

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



$\mathcal{H}$–LU Factorization

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



Hardware/Software

Machine

- 2x64-core AMD Epyc 9554 (Genoa)
- 2x12 32GB DDR5-4800 DIMMs

Software

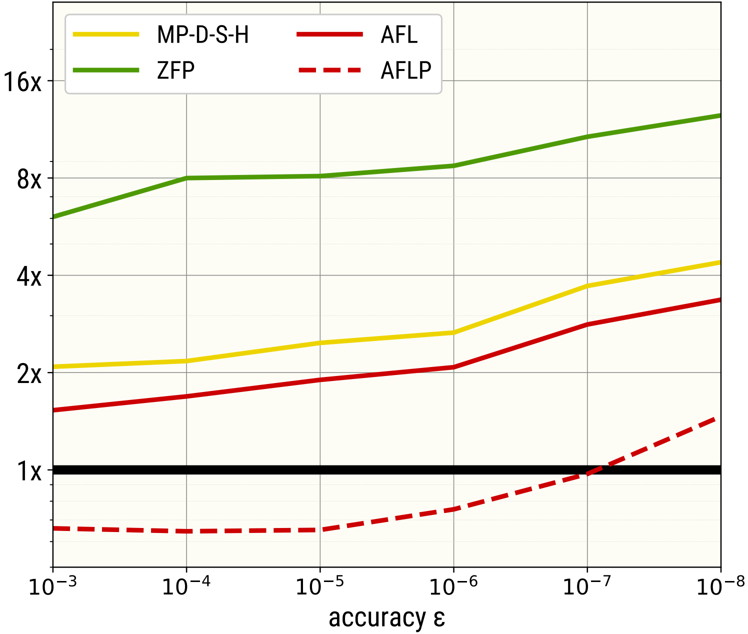
- HLR (`libhlr.org`)
- Intel TBB v2021.2
- Intel MKL v2022.0 (AVX512 code path)
- GCC 12

Benchmark

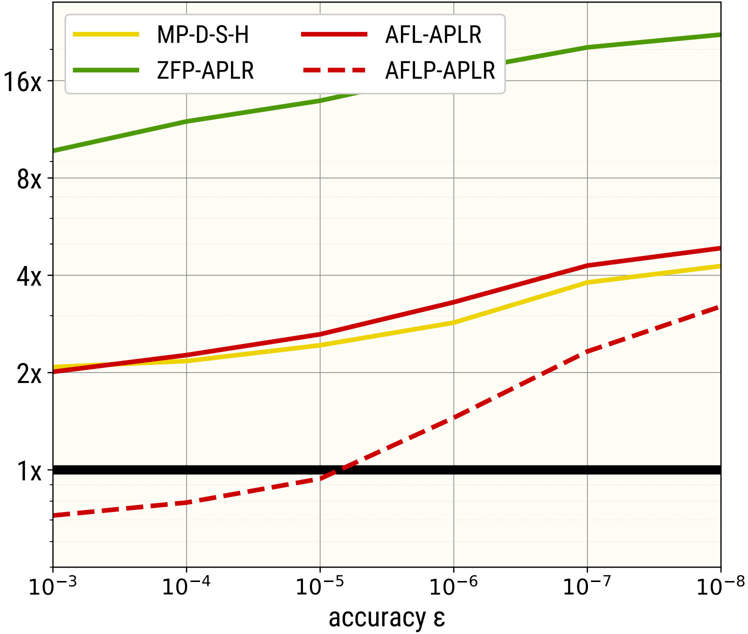
- Model problem: Laplace SLP on unit sphere, $n = 1.048.576$
- lowrank truncation via SVD
- runtime: median out of 10 runs

Matrix–Vector Multiplication

Matrix–Vector Multiplication

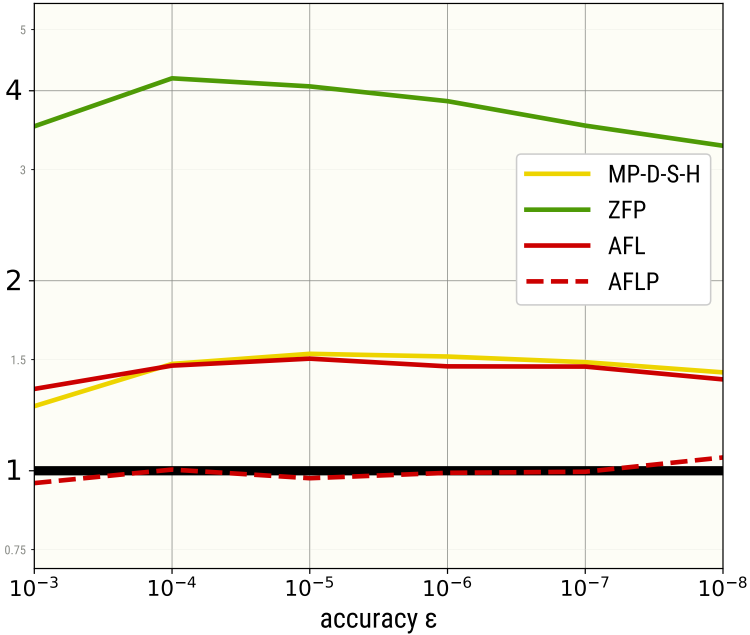


Matrix–Vector Multiplication

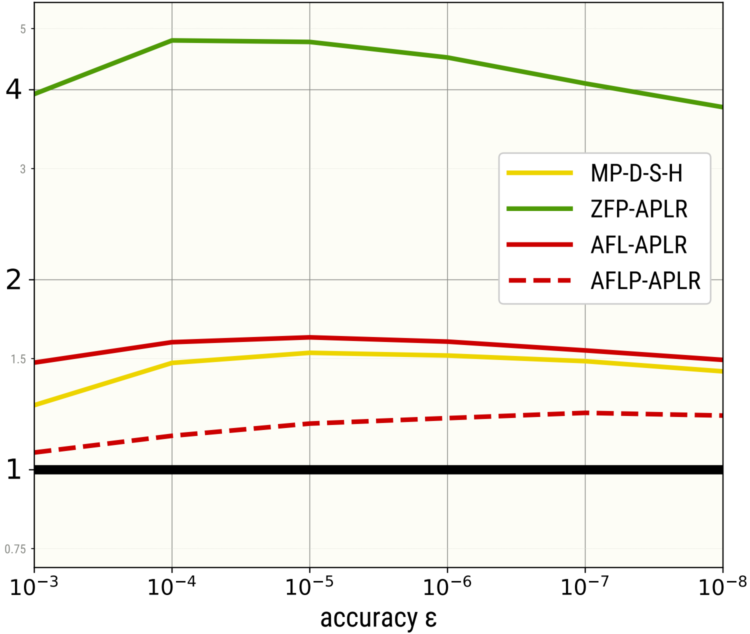


$\mathcal{H}$-LU Factorization

Rel. Runtime with Accumulator



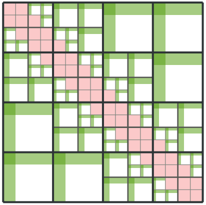
Rel. Runtime with Accumulator



What about other $\mathcal{H}$ formats?

What about other $\mathcal{H}$ formats?

$\mathcal{H}$



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

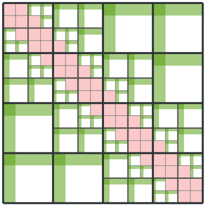
with

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

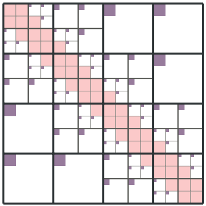
$$\mathcal{O}(n \log n)$$

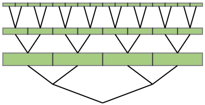
What about other $\mathcal{H}$ formats?

$\mathcal{H}$



Uniform-$\mathcal{H}$





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

with

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

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

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

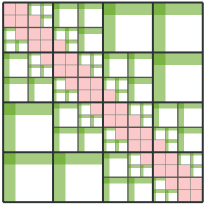
with

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

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

What about other $\mathcal{H}$ formats?

$\mathcal{H}$



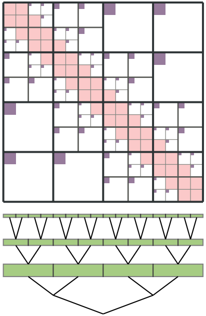
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^T$$

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-$\mathcal{H}$



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

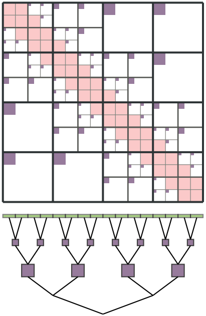
with

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

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

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

$\mathcal{H}^2$



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

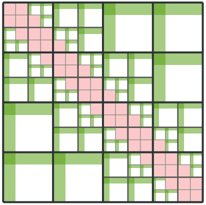
with

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

$\mathcal{O}(n)$

What about other $\mathcal{H}$ formats?

$\mathcal{H}$



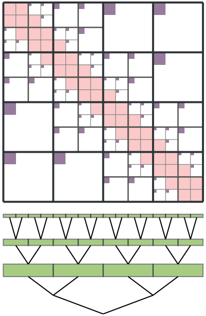
$$M_{\tau,\sigma} = U_{\tau,\sigma} \cdot V_{\tau,\sigma}^T$$

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-$\mathcal{H}$



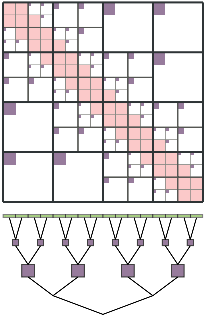
$$M_{\tau,\sigma} = \mathcal{U}_\tau \cdot S_{\tau,\sigma} \cdot \mathcal{V}_\sigma^T$$

with

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

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

$\mathcal{H}^2$



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

with

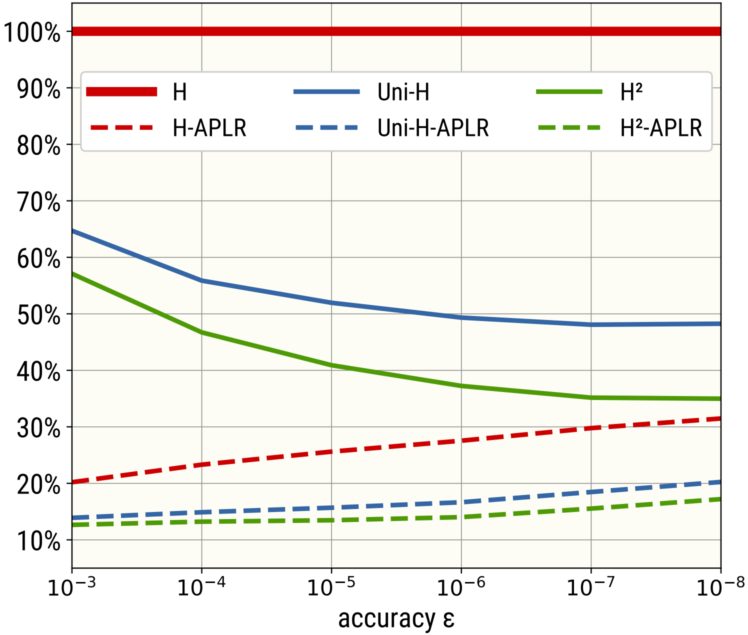
implicit $\widetilde{\mathcal{U}}_\tau, \widetilde{\mathcal{V}}_\sigma$

$\mathcal{O}(n)$

Apply compression schemes to data blocks in Uniform-$\mathcal{H}$ and $\mathcal{H}^2$.

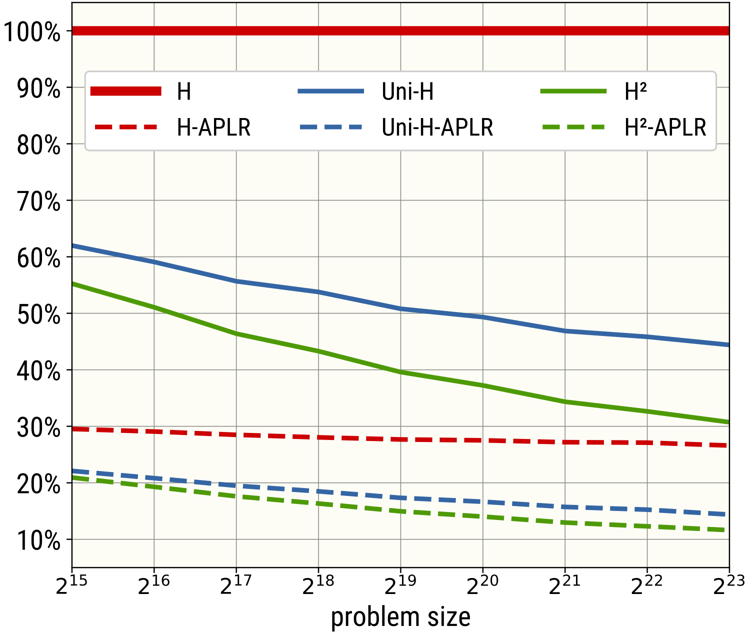
Compression Rates

Laplace SLP ($n = 1.048.576$, AFL)



Compression Rates

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



Conclusion

Conclusion

Storage for $\mathcal{H}$-matrices can be

- *significantly* optimized
- with *little impact* on (parallel) performance

by binary compression techniques.

Requirement: *fast* and *adaptive* compression!

libHLR.org

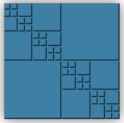
Conclusion

Storage for $\mathcal{H}$-matrices can be

- *significantly* optimized
- with *little impact* on (parallel) performance

by binary compression techniques.

Requirement: *fast* and *adaptive* compression!



libHLR.org

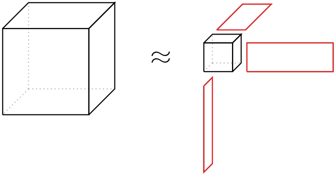
And Beyond?

For multi dimensional data $M \in \mathbb{C}^{n_1 \times \ldots \times n_d}$ the Tucker decomposition yields

$$M \approx G \times_1 U_1 \times_2 \ldots \times_d U_d$$

with orthogonal $U_i \in \mathbb{C}^{n_i \times k_i}$ and $G \in \mathbb{C}^{k_1 \times \ldots \times k_d}$.

Binary compression can be applied to G and *APLR to* U_i.



Thank You

MAX PLANCK INSTITUTE
FOR MATHEMATICS IN THE SCIENCES